UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

JULIO CESAR DE PAIVA RIBEIRO

DESENVOLVIMENTO DE UMA BIBLIOTECA DIDÁTICA PARA ACESSO AOS COMPONENTES DA SPARTAN-3E STARTER KIT BOARD

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2017

JULIO CESAR DE PAIVA RIBEIRO

DESENVOLVIMENTO DE UMA BIBLIOTECA DIDÁTICA PARA ACESSO AOS COMPONENTES DA SPARTAN-3E STARTER KIT BOARD

Trabalho de Conclusão de Curso como requisito parcial à obtenção do título de Bacharel em Engenharia de Computação, do Departamento Acadêmico de Informática da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Giovanni Alfredo Guarneri

PATO BRANCO

2017

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Engenharia de Computação

TERMO DE APROVAÇÃO

Às 13h horas e 45 minutos do dia 26 de junho de 2017, na sala V105, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, reuniu-se a banca examinadora composta pelos professores Giovanni Alfredo Guarneri (orientador), Fabio Luiz Bertotti e Fernando Jose Avancini Schenatto para avaliar o trabalho de conclusão de curso com o título **Desenvolvimento de uma Biblioteca Didática para Acesso aos Componentes da *Spartan-3E Starter Kit Board***, do aluno **Julio Cesar de Paiva Ribeiro**, matrícula 01215515, do curso de Engenharia de Computação. Após a apresentação o candidato foi arguido pela banca examinadora. Em seguida foi realizada a deliberação pela banca examinadora que considerou o trabalho aprovado.

Prof. Giovanni Alfredo Guarneri
Orientador (UTFPR)

Prof. Fabio Luiz Bertotti
(UTFPR)

Prof. Fernando Jose Avancini Schenatto
(UTFPR)

Profa. Beatriz Terezinha Borsoi
Coordenador de TCC

Prof. Pablo Gauterio Cavalcanti
Coordenador do Curso de
Engenharia de Computação

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

AGRADECIMENTOS

Agradeço primeiramente a Deus, minha família pelo apoio integral, seja financeiramente ou emocionalmente, aos meus amigos, que nos momentos de dificuldade sempre estavam presentes quando precisei, minha namorada, pelo apoio e incentivo durante a graduação e desenvolvimento deste trabalho.

Também devo agradecer ao professor Giovanni pela oportunidade de orientação e desenvolvimento deste trabalho, sempre me mostrando que podemos melhorar.

UTFPR, universidade que abriu as portas para uma nova etapa em minha vida tornando possível a minha graduação como Engenheiro de Computação.

RESUMO

RIBEIRO, Julio Cesar. Desenvolvimento de uma biblioteca didática para acesso aos componentes da *Spartan-3E Starter Kit Board*. 2017. 97f. Trabalho de Conclusão de Curso de bacharelado em Engenharia de Computação - Universidade Tecnológica Federal do Paraná. Pato Branco, 2017.

A universidade possui inúmeros meios de dar suporte aos ensinamentos transmitidos aos alunos e isso faz toda a diferença na forma como eles irão tomar posse desse conhecimento. Livros, apostilas, simulações, *kits* de desenvolvimento tudo vem ao encontro de uma aprendizagem mais sólida, que garanta ao aluno reter melhor os conteúdos e aprimorá-los. Este trabalho apresenta o desenvolvimento de uma biblioteca utilizando a linguagem VHDL para acesso aos componentes da *Spartan-3E Starter Kit Board*. A metodologia de pesquisa adotada no desenvolvimento deste trabalho foi a de desenvolvimento experimental que é caracterizado pelo trabalho criativo levado a cabo de forma sistemática. A biblioteca desenvolvida contém os seguintes componentes: *encoder* rotativo, *displa*y LCD, porta PS/2, conversores A/D e D/A. Os componentes implementados foram modelados para serem utilizados na metodologia de projeto via diagrama esquemático, permitindo o usuário realizar aplicações como, conversão de sinal analógico para digital ou digital para analógico. Para a utilização dos componentes implementados se faz necessário somente o conhecimento sobre circuitos e sistemas digitais o que torna a *Spartan-3E Starter Kit Board* uma ferramenta mais didática. Além dos componentes implementados foi criada uma documentação que contém informações técnicas sobre os componentes desenvolvidos e um tutorial de utilização da biblioteca.

Palavras-chave: Biblioteca. FPGA. Spartan-3E. VHDL.

ABSTRACT

RIBEIRO, Julio Cesar. Development of an educational library for the access to components of the Spartan-3E Starter Kit Board. 2017. 97f. Trabalho de Conclusão de Curso de bacharelado em Engenharia de Computação - Universidade Tecnológica Federal do Paraná. Pato Branco, 2017.

The university has many ways of supporting the teachings passed on to the students and this makes all the difference in how they will take possession of this knowledge. Books, handouts, simulations, development kits all come to a more solid learning, which guarantees the student to retain the contents and improve them. This paper presents the development of a library using the VHDL language for access to components of the Spartan-3E Starter Kit Board. The research methodology adopted in the development of this work was that of experimental development that is characterized by the creative work carried out in a systematic way. The developed library contains the following components: rotary encoder, LCD display, PS/2 port, A/D and D/A converters. The implemented components were modeled to be used in the design methodology through a schematic diagram, allowing the user to perform applications such as analog to digital or digital to analog conversion. For the use of the implemented components, only the knowledge about circuits and digital systems is necessary, which makes the Spartan-3E Starter Kit Board a more didactic tool. In addition to the implemented components, a documentation was created that contains technical information about the components developed and a tutorial on using the library.

Keywords: Library. FPGA. Spartan-3E. VHDL.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

BIT	*Bitstream Format* (Formato de fluxo de *bits*)
CG RAM	*Character Generator Random Acess Memory* (Memória *RAM* para acesso aos caracteres)
CG ROM	*Character Generator Read-Only Memory* (Memória para leitura de caracteres)
CLBs	*Configurable Logic Blocks* (Blocos lógicos configuráveis)
CPLDs	*Complex Programmable Logic Devices* (Dispositivos complexos de lógica programável)
DAC	*Digital to analog converter* (Conversor digital para analógico)
DCM	*Digital Clock Manager* (Gerenciador de relógio digital)
DD RAM	*Display Data Random Acess Memory* (Memória *RAM* para exibir dados)
EDIF	*Eletronic Design Interchange Format* (Design eletrônico para troca de formato)
EEPROM	*Electrically Erasable Programmable Read Only Memory* (Memória somente de leitura apagável eletronicamente)
FPGAs	*Field Programmable Gate Array*s (Arranjo de Portas Programáveis em Campo)
GAL	*Generic Array Logic* (Arranjos lógicos genéricos)
IEEE	*Institute of Electrical and Eletronics Engineers* (Instituto de Engenheiros Eletricistas e Eletrônicos)
IOBs	*Input/Output Blocks* (Blocos de entrada/saída)
LCD	*Liquid Crystal Display* (*Display* de Cristal líquido)
LUTs	*Lookup Tables* (Tabelas de pesquisa)
PLDs	*Programmable Logic Devices* (Dispositivos lógicos programáveis)
PROM	*Programmable read-only memory* (Memória programável somente de leitura)
RAM	*Random Access Memory* (Acesso aleatório de memória)
RTL	*Register Transfer Level* (Nível de transferência de registro)
SDRAM DDR	*Synchronous dynamic random access memory* (Memória *RAM* dinâmica síncrona)
SPI	*Serial Peripheral Interface* (Interface periférica serial)

UCF User *Constraints File* (Arquivo de restrições do usuário)

VHDL *Very High Speed Integrated Circuit Hardware Description Language* (Linguagem de descrição de *hardware* para circuito de alta velocidade)

VHSIC *Very High Speed Integrated Circuit* (Circuito integrado de alta velocidade)

SUMÁRIO

1 INTRODUÇÃO

1.1 CONSIDERAÇÕES INICIAIS

Uma linguagem de descrição *hardware* é utilizada para desenvolver projetos de circuitos digitais em CPLDs (do original em inglês *Complex Programmable Logic Device*) ou FPGAs (do original em inglês *Field Programmable Gate Array*). Uma dessas linguagens mais utilizada atualmente é o VHDL (do original em inglês *Very High Speed Integrated Circuit Hardware Description Language*). A linguagem VHDL surgiu de uma iniciativa financiada pelo Departamento de Defesa dos Estados Unidos na década de 1980 e foi a primeira linguagem de descrição de *hardware* padronizada pelo IEEE (do original em inglês *Institute of Electrical and Eletronics Engineers*) (padrões 1076 e 1164). Tem como motivação fundamental a portabilidade, pois independe da tecnologia e do fornecedor em que está sendo implementada (PEDRONI, 2010).

Outra linguagem de descrição de *hardware* é o Verilog, que surgiu após a criação do VHDL e também é um padrão IEEE. Possui três revisões: Verilog-95 (padrão IEEE 1364-1995), Verilog 2001 (padrão IEEE 1364-2001) e Verilog 2005 (padrão IEEE 1364-2005). O Verilog permite ao projetista os meios para descrever um sistema digital em vários níveis de abstração, e também suporta ferramentas de projeto para síntese lógica, constituindo-se numa segunda opção de linguagem para desenvolvimento de projetos em CPLDs e FPGAs (IEEE, 2008).

CPLDs e FPGAs são dispositivos lógicos programáveis com construção e arquiteturas distintas. A construção de um CPLD consiste em vários PLDs (do original em inglês *Programmable Logic Devices*) fabricados no mesmo chip. Esses PLDs, organizados como uma pilha de blocos, se comunicam por meio de um arranjo de interconexões complexo e programável, contendo drivers de entrada/saída (PEDRONI, 2010). Já em uma FPGA, os diversos blocos de PLDs são organizados como uma matriz, contendo um número maior de elementos menores, porém mais sofisticados. As FPGAs diferem dos CPLDs não somente pela arquitetura, mais também pela tecnologia de fabricação, características embutidas, desempenho e custo (PEDRONI, 2010).

Dentre os fabricantes de CPLDs e FPGAs tem-se a Xilinx®, que conta com vários produtos, entre eles o CPLD XC2C64A e a FPGA XC3S500E. O CPLD XC2C64A é projetado para alto desempenho e aplicações de baixa potência. A FPGA XC3S500E faz parte

da família *Spartan-3E*®, reconhecida por proporcionar alto desempenho com um baixo custo (XILINX, 2013a).

A arquitetura apresentada na família *Spartan-3E*® é composta por cinco blocos de elementos programáveis fundamentais que são (XILINX, 2013a):

- Blocos lógicos configuráveis (CLBs do original em inglês *Configurable Logic Blocks*): implementam a lógica dos elementos presente na FPGA;
- Blocos de entrada/saída (IOBs do original em inglês *Input/Output Blocks*): controlam o fluxo de dados entre os pinos de entrada/saída e a lógica interna do dispositivo;
- Bloco de memória (*RAM* do original em inglês *Random Access Memory*): fornece o armazenamento de dados;
- Blocos de multiplicação: aceitam dois dados de 18 bits como entrada;
- Bloco gerenciador de *Clock* digital (DCM do original em inglês *Digital Clock Manager*): permite o gerenciamento do sinal de *Clock*.

A série XC2C64A de CPLD e a série XC3S500E de FPGA da Xilinx® fazem parte do *kit* de desenvolvimento *Spartan-3E Starter Kit Board* produzido pela empresa Digilent. O *Spartan-3E Starter Kit Board* é uma plataforma de desenvolvimento que visa criar projetos para FPGAs da série XC3S500E.

Como descrito, a configuração das FPGAs é feita por meio de uma linguagem de descrição de *hardware*, VHDL ou Verilog. Com a descrição feita, se torna possível o acesso aos componentes presentes no *kit* de desenvolvimento. Outra maneira de acesso aos componentes presentes no *kit* é por meio da metodologia de projeto via diagrama de componentes, onde toda a descrição é encapsulada e apresentada em um bloco com as portas de entrada e saída. A Xilinx® possui uma biblioteca padrão que contém os componentes encapsulados para a utilização em diagrama de componentes, porém essa biblioteca não contém todos os componentes que permitem o acesso a todos os recursos disponíveis no *kit*. Tendo em vista isso, a ideia de se desenvolver novos componentes amplia as possibilidades de utilização do *kit* via diagrama de componentes, tornando o uso da FPGA mais didático visto que o conhecimento da linguagem VHDL e Verilog não se faz necessário.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Implementar uma biblioteca em VHDL para configurar o uso dos seguintes componentes presentes no *Spartan-3E Starter Kit Board*: *encoder* rotativo, *display* LCD, porta PS/2, conversores D/A e A/D.

1.2.2 Objetivos Específicos

- Modelar os componentes a serem implementados na biblioteca de acesso ao *Spartan-3E Starter Kit Board*;
- Implementar os componentes modelados para o desenvolvimento de projetos via diagrama de componentes;
- Testar os componentes implementados via simulação e teste no kit de desenvolvimento;
- Encapsular os componentes implementados em uma biblioteca na linguagem VHDL;
- Criar a documentação da biblioteca de componentes implementados contendo informações técnicas, exemplos de funcionamento e uso.

1.3 JUSTIFICATIVA

A universidade possui inúmeros meios de dar suporte aos ensinamentos transmitidos aos alunos e isso faz toda a diferença na forma como eles tomarão posse desse conhecimento. Livros, apostilas, simulações, *kits* de desenvolvimento tudo vem ao encontro de uma aprendizagem mais sólida, que garanta ao aluno reter melhor os conteúdos e aprimorá-los. Além de garantir uma melhor forma de reter o ensinamento transmitido pelo professor o material didático pode servir também de suporte para outras pessoas.

O *Spartan-3E Starter Kit Board* é uma ferramenta de desenvolvimento de componentes acoplados no *kit* que permitem ao projetista realizar aplicações como a conversão de sinal analógico para digital ou digital para analógico, armazenamento de dados em memória *RAM* dinâmica e memória *Flash*. Para utilizar os componentes disponíveis, o projetista deve possuir conhecimentos sobre a linguagem VHDL.

O desenvolvimento de uma biblioteca para acesso aos elementos do *Spartan-3E Starter Kit Board* possibilitará que usuários desenvolvam projetos por diagrama de

componentes, permitindo a análise dos resultados via simulação ou carregando o projeto desenvolvido no próprio *kit*. A biblioteca desenvolvida facilitará também o desenvolvimento para usuários que necessitam criar projetos utilizando o *kit*, e não possuem o conhecimento e tempo para aprendizado sobre a linguagem VHDL, ou possuem o conhecimento mas desejam desenvolver o projeto por diagrama de componentes.

1.4 ESTRUTURA DO TRABALHO

Este documento encontra-se organizado em 5 capítulos, de tal maneira que:

O Capítulo 2 contém uma revisão bibliográfica dos conteúdos fundamentais para o desenvolvimento do trabalho. Neste capítulo, são abordados os conceitos sobre os dispositivos lógicos programáveis, linguagem de descrição de *hardware* e o *kit* de desenvolvimento.

No Capítulo 3, é apresentada a metodologia utilizada para o desenvolvimento do trabalho, apresentando os componentes a serem implementados e explicando o funcionamento destes.

No Capítulo 4, são apresentados os resultados obtidos e as discussões, o que inclui a biblioteca desenvolvida e os circuitos de testes dos componentes implementados.

O Capítulo 5 apresenta as conclusões do trabalho desenvolvido.

Apêndices, contém a documentação técnica dos componentes desenvolvidos, o tutorial de utilização da biblioteca criada e os arquivos de restrições de projeto dos componentes.

2 REFERENCIAL TEÓRICO

Este Capítulo apresenta os principais tópicos relevantes no contexto deste trabalho, em que é feita uma abordagem sobre os dispositivos lógicos programáveis e a linguagem de descrição de *hardware* VHDL. Ao final deste capítulo será possível entender as principais etapas do desenvolvimento de projetos utilizando a linguagem VHDL. Também será possível entender a arquitetura e conhecer os principais componentes presentes no *kit* de desenvolvimento utilizado nesse trabalho.

2.1 DISPOSITIVOS LÓGICOS PROGRAMÁVEIS

2.1.1 Histórico

Os dispositivos lógicos programáveis PLDs foram introduzidos na década de 1970 com o intuito de construir circuitos combinacionais lógicos que fossem programáveis. Os primeiros PLDs empregavam somente portas lógicas convencionais e visavam apenas a implementação de circuitos combinacionais. Mais tarde foram lançados PLDs com um *flip-flop* em cada saída do circuito, permitindo a implementação de funções sequenciais simples (PEDRONI, 2010).

No início da década de 1980, os PLDs tiveram uma nova modificação. Cada saída do PLD possuía circuitos lógicos que continham *flip-flop*, portas lógicas e multiplexadores permitindo vários modos de operação. Essa nova estrutura de PLD recebeu o nome de GAL (do original em inglês *Generic Array Logic*), que se diferenciava dos antigos PLDs tanto em arquitetura quanto na forma de fabricação, que permita a junção de vários dispositivos GAL no mesmo *chip*. Com a nova forma de fabricação, novas características foram criadas, como a adição de diversos pinos de entrada/saída, o que possibilitou novas funcionalidades aos dispositivos e deu origem aos CPLDs (PEDRONI, 2010).

Ainda na década de 1980, foram lançadas as FPGAs que eram compostas por diversos blocos de PLDs organizados como uma matriz, diferenciando-se dos CPLDs que eram construídos com diversos PLDs organizados como uma pilha de blocos. A nova arquitetura apresentada nas FPGAs permitia a junção de um número maior de elementos, possibilitando o desenvolvimento de projetos complexos e de elevado desempenho a um menor custo (PEDRONI, 2010).

2.1.2 Aspectos gerais FPGA

A arquitetura geral de uma FPGA trata-se de uma matriz de blocos, que é composta por três estruturas básicas: blocos lógicos, interconexões programáveis e blocos de entrada/saída. Os blocos de entrada/saída formam uma camada ao redor do dispositivo. Dentro da camada dos blocos de entrada/saída têm-se os blocos lógicos que implementam a lógica dos elementos e as interconexões programáveis que são as vias para interligar os blocos lógicos com os blocos de entrada/saída. A Figura 1 mostra a estrutura interna de uma FPGA.

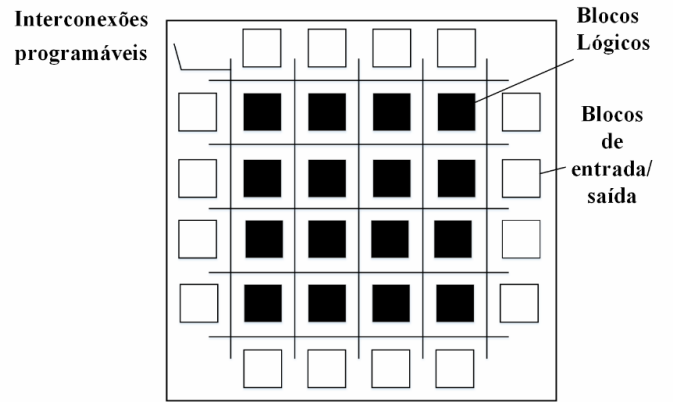

Figura 1 - Estrutura interna de uma FPGA
Fonte: adaptado de Bezerra (2010, p. 19).

Os blocos lógicos são construídos a partir de células lógicas, que são desenvolvidas a partir das LUTs (do original em inglês *Lookup Tables*). As LUTs presentes nos blocos lógicos são tabelas-verdade genéricas configuradas de acordo com a necessidade do projeto. A configuração das funções lógicas de cada bloco é feita nas células internas de memória, onde os valores armazenados nas células de memória determinam as funções a serem implementadas na FPGA (D'AMORE, 2005). A estrutura do bloco lógico de uma FPGA é representada na Figura 2.

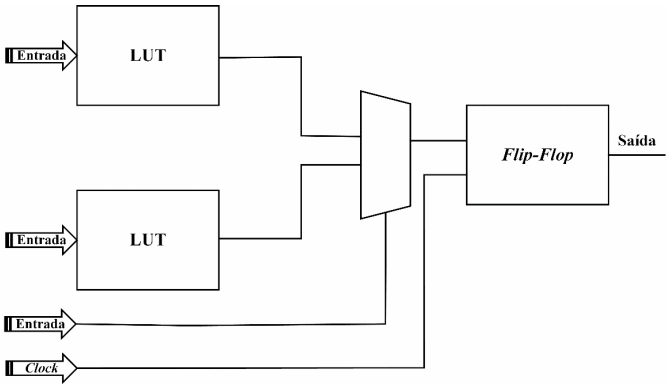

Figura 2 - Estrutura do bloco lógico de uma FPGA
Fonte: adaptado Xilinx (2013a, p. 4).

As interconexões programáveis são dispostas em formas de trilhas verticais e horizontais entre as linhas e as colunas dos blocos lógicos. São compostas por componentes condutores, que são usados para estabelecer a conexão entre as linhas (XILINX, 2013a). As interconexões programáveis são representadas na Figura 3.

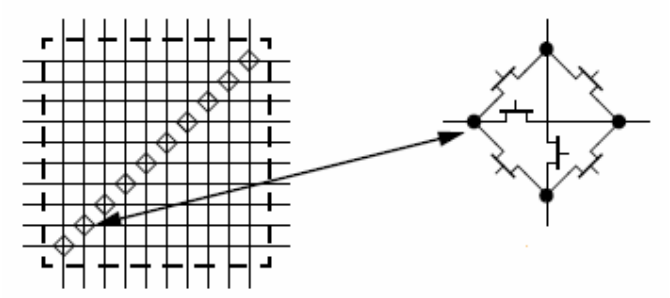

Figura 3 - Interconexões programáveis
Fonte: Xilinx (2013a, p. 12).

Os blocos de entrada/saída são responsáveis por controlar, entre a conexão interna das células lógicas com os pinos externos. Esses pinos podem operar como entrada, saída ou acesso bidirecional. Uma vez em que o bloco lógico envia um sinal para o bloco de entrada/saída, esse sinal passa por um multiplexador que faz a seleção do canal que receberá o sinal. Após a seleção do canal, o sinal é enviado ao pino externo. O bloco de entrada/saída de uma FPGA é representada na Figura 4.

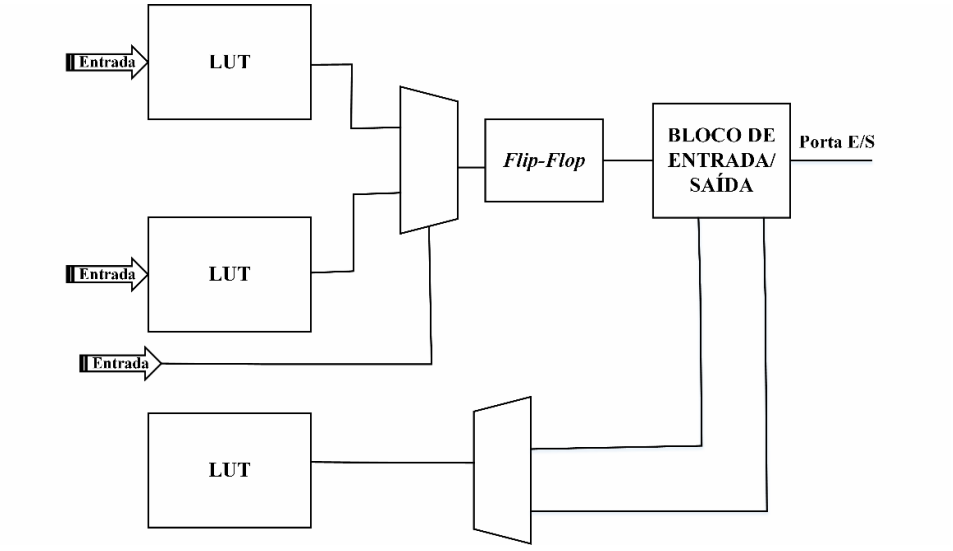

Figura 4 - Bloco de entrada/saída de uma FPGA
Fonte: adaptado Xilinx (2013a, p. 7).

2.1.3 *Spartan 3-E Starter Kit Board*

A *Spartan 3-E Starter Kit Board* é um *kit* de desenvolvimento projetado pela Xilinx®, produzido e comercializado pela Digilent. Dentre os principais recursos presentes estão a FPGA Xilinx XC3S500E *Spartan 3-E*, o CPLD Xilinx XC2C64A *CoolRunner*, uma memória *flash* PROM (do original em inglês *Programmable Read-Only Memory*) de configuração com 4 Mb, uma memória SDRAM DDR (do original em inglês *Synchronous Dynamic Random Access Memory*) de 64 MB e frequência de 100 MHz, um cristal oscilador de 50 MHz, conectores para expansão padrão FX2, um *display* de LCD (do original em inglês *Liquid Crystal Display*) composto por 2 linhas de 16 caracteres (XILINX, 2006). A *Spartan 3-E Starter Kit Board* é apresentada na Figura 5.

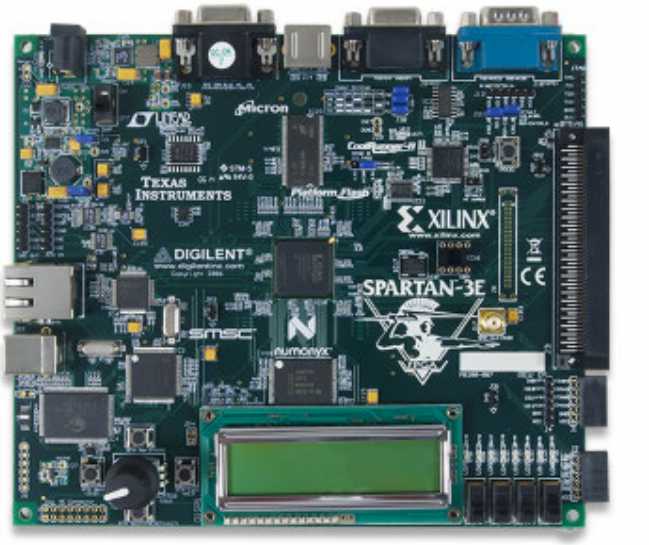

Figura 5 - Spartan 3-E Starter Kit Board
Fonte: Digilent (2016).

A FPGA XC3S500E presente no *kit* faz parte da família *Spartan-3E*®, conta com 46 linhas e 26 colunas de interligações programáveis totalizando 612 blocos lógicos. Possui ainda 232 pinos de entrada/saída e mais de 10,000 células lógicas, possibilitando soluções lógicas de alto desempenho. As aplicações típicas envolvendo o uso de FPGAs contam apenas com uma memória não-volátil. O *kit* de desenvolvimento conta com três fontes de memória de diferentes configurações o que garante maior capacidade de armazenamento para as configurações da FPGA (XILINX, 2013c).

Outros componentes presentes no *kit* que podem ser utilizados com a correta configuração da FPGA são: chaves deslizantes, *push-buttons*, *encoder* rotativo, *leds*, portas VGA, RS-232, PS/2, conversores D/A e A/D, memória *EEPROM* (do original em inglês

Electrically Erasable Programmable Read Only Memory) e controlador *Ethernet* (XILINX, 2006). Dentre esses componentes, os que irão compor a biblioteca a ser desenvolvida são: chaves deslizantes, *push-buttons*, *encoder* rotativo, *leds*, *display* LCD, porta PS/2, conversor D/A e A/D. O funcionamento e a implementação destes componentes são descritos no Capítulo 3.

2.2 LINGUAGEM DE DESCRIÇÃO DE *HARDWARE* VHDL

2.2.1 Histórico

A linguagem VHDL é uma linguagem de descrição de *hardware*. O seu desenvolvimento foi devido a necessidade de se ter uma ferramenta de projeto e documentação padrão para o VHSIC (do original em inglês *Very High Speed Integrated Circuit*) (D'AMORE, 2005). A linguagem VHDL descreve o comportamento de um circuito ou sistema elétrico, a partir da qual o mesmo possa ser implementado (PEDRONI, 2004).

Uma iniciativa criada pelo Departamento de Defesa dos Estado Unidos levou a definição de uma linguagem de descrição de *hardware* padrão, dando origem ao VHDL. Em 1987, o IEEE padronizou a primeira versão da linguagem VHDL. A primeira versão da linguagem VHDL recebeu o nome de VHDL-87. Esse padrão foi descrito no documento IEEE 1076-1987. Uma nova versão da linguagem foi criada no ano de 1993, dando origem ao padrão IEEE 1076-1993. As principais alterações dessa versão eram no tratamento de arquivos e não nas características para a síntese de circuitos, o que permitia que quase todas as descrições fossem compiladas em ambas as versões (D'AMORE, 2005).

Após a criação do padrão IEEE 1076-1993, outras três revisões foram feitas na linguagem, dando origem aos padrões IEEE 1076-2000, IEEE 1076-2002 e IEEE 1076-2008. O padrão IEEE 1076-2008 é a versão mais recente do VHDL, sendo uma revisão do padrão IEEE 1076-2002 que inclui novas características e melhorias a recursos previamente existentes na linguagem. Além das revisões, algumas ramificações do padrão IEEE 1076-1993 foram criadas a fim de facilitar e complementar a linguagem VHDL, o padrão IEEE 1164 e o padrão IEEE 1076.3 são ramificações do padrão IEEE 1076-1993. O primeiro define o pacote *Std_logic_1164* e o segundo os pacotes *Numeric_std* e *Numeric_bit.* Os pacotes definidos em VHDL são utilizados para o armazenamento de informações de uso comum, como tipos de dados e funções (D'AMORE, 2005).

O pacote padrão da linguagem de descrição de *hardware* define vários tipos de dados, sendo o tipo *bit* o que possui associação mais direta com um dado a ser manipulado por um circuito digital. O tipo de dado *bit* pode assumir apenas dois valores 0 ou 1, o que em muitos casos, limita a modelagem do circuito digital. Com a definição do padrão IEEE 1164 pode-se cobrir esta limitação encontrada na modelagem, definindo novos tipos de dados que permitem modelar condições de alta impedância e solucionar casos em mais de uma porta aciona o mesmo nó. Já no padrão IEEE 1076.3, são definidas funções para a realização de operações aritméticas, utilizando os dados no formato de vetores dando assim mais opções para o projetista (D'AMORE, 2005).

2.2.2 Aspectos gerais da linguagem

A linguagem VHDL suporta desenvolvimento de projetos com múltiplos níveis de hierarquia, podendo o comportamento de um circuito ser descrito pela interligação de outras descrições menores. Esses estilos são denominados estrutural e comportamental. O estilo estrutural está relacionado as interconexões das portas lógicas e o estilo comportamental está relacionado a abstração lógica do circuito. Eles podem ser mesclados em uma mesma descrição pela utilização da estrutura hierárquica, permitindo a descrição de projetos que partem de um nível mais elevado para um nível mais baixo de especificação. Essa abordagem é conhecida como *top-down design* (D'AMORE, 2005).

As linhas de comando das descrições de circuitos na linguagem VHDL são interpretadas pela ferramenta de síntese concorrentemente. Portanto, a ordem na apresentação dos comandos é irrelevante para o comportamento da descrição. A ocorrência de um evento em um sinal leva a ativação de todos os comandos sensíveis àquele sinal, da mesma forma que em um circuito, a mudança de um determinado valor em um nó afeta todas as entradas ligadas a esse ponto no circuito.

A linguagem permite também a síntese de códigos sequenciais, em que a execução feita pelo compilador segue a ordem de apresentação do código. No entanto, a descrição dos códigos sequenciais deve ser feita em regiões especiais do código VHDL e não podem ser misturados os com códigos concorrentes (D'AMORE, 2005).

A linguagem VHDL permite a definição de circuitos auxiliares na forma de procedimentos e funções. Os circuitos auxiliares definidos podem ser utilizados em rotinas de

conversão, definição de novos operadores lógicos e aritméticos “e outras operações não diretamente ligadas a um circuito passível de síntese” (D’AMORE, 2005, p. 2).

Outro recurso que pode ser utilizado na descrição de circuitos é a criação de bibliotecas e pacotes. Os pacotes são utilizados para o armazenamento de subprogramas, constantes ou definição de novos tipos, evitando a repetição de uma definição em todas as descrições. As bibliotecas são o local de armazenamento de informações compiladas, e são criadas com a utilização da ferramenta empregada para compilação e simulação. A linguagem permite ao projetista definir novas bibliotecas para o armazenamento de unidades de projeto num mesmo local, podendo ser utilizadas como uma área de uso comum, possibilitando o emprego de uma mesma descrição por diferentes programadores (LIPSETT; SCHAEFER; USSERY, 1993).

2.2.3 Metodologia de projeto dispositivos lógicos programáveis

O desenvolvimento de projetos utilizando a linguagem VHDL requer algumas etapas para que se possa realizar a síntese de um circuito. Essas etapas são apresentadas na Figura 6.

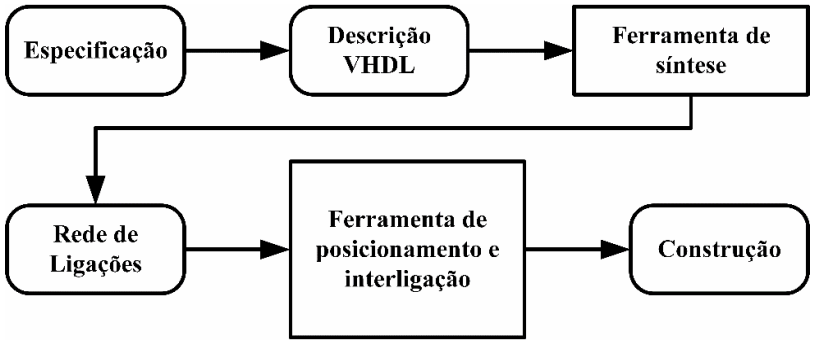

Figura 6- Etapas para o desenvolvimento de projeto
Fonte: adaptado de D’Amore (2005, p. 3).

As etapas de projeto apresentadas na Figura 6 serão explicadas nos tópicos subsequentes.

2.2.3.1 Descrição VHDL

O primeiro passo no desenvolvimento de circuitos é a especificação do projeto. Com a especificação do projeto definida, é gerada uma descrição VHDL do circuito que é

submetida a um compilador para a verificação da correspondência entre a especificação e o código.

A linguagem VHDL permite a descrição de um circuito utilizando diversos graus de abstração, o que gera estruturas abstratas que não permitem uma síntese direta. Portanto, para que se possa realizar a síntese de um processo abstrato, um processo de detalhamento dos elementos da estrutura é necessário. Após os detalhamentos, são realizadas simulações para assegurar a equivalência entre a especificação do projeto e a descrição proposta (D'AMORE, 2005). A elaboração da descrição VHDL é representa na Figura 7.

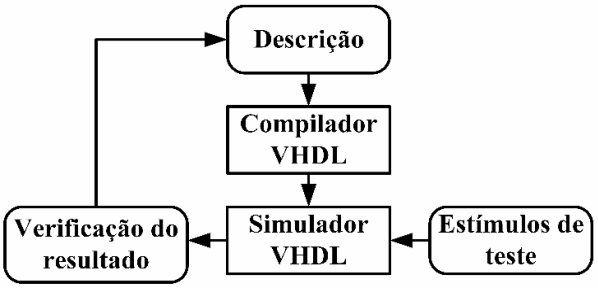

Figura 7- Simulação da descrição VHDL
Fonte: adaptado D'Amore (2005, p. 4).

Realizadas as simulações e assegurado a equivalência entre a especificação do projeto e a descrição proposta pode-se iniciar as etapas necessárias para que o projeto possa ser implementado no dispositivo lógico.

2.2.3.2 Síntese de circuito

Após a etapa de descrição, dá-se início ao processo de síntese que é responsável por definir as estruturas necessárias para implementação do circuito. A ferramenta de síntese executa o processo de inferência e interligação das estruturas necessárias para que o circuito seja gerado a partir da descrição. Primeiramente, é gerado um circuito no nível RTL (do original em inglês *Register Transfer Level*) empregando componentes genéricos disponíveis na ferramenta, tais como comparadores, somadores, registradores e portas lógicas. O circuito gerado no nível RTL é um circuito global, que não está associado a nenhuma tecnologia de fabricação específica. Com o término da síntese do circuito nível RTL, é iniciada a etapa de síntese para um novo circuito com base nas estruturas geradas no nível RTL, dando origem a um circuito que contém componentes específicos disponíveis na tecnologia empregada para

fabricação. Assim, para a realização dessa etapa, faz necessária a especificação do dispositivo que será utilizado (D'AMORE, 2005). O processo de síntese da descrição VHDL é representado na Figura 8.

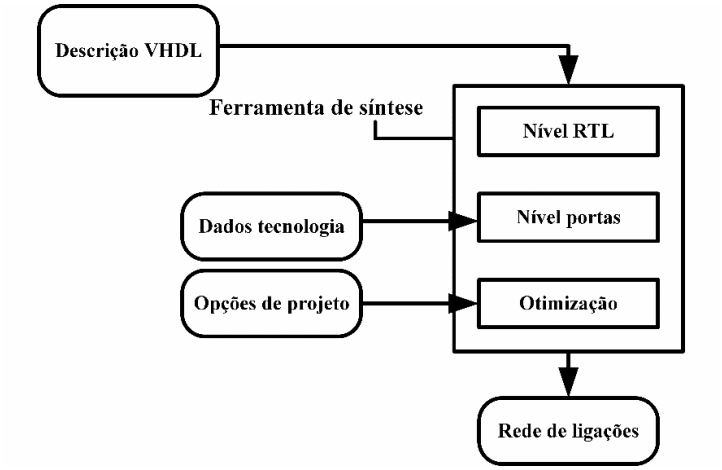

Figura 8 - Síntese da descrição VHDL
Fonte: adaptado D'Amore (2005, p. 4).

Concluída a síntese da descrição VHDL a rede ligações criada está pronta para ser utilizada pelo s*oftware* que irá fazer o processo de posicionamento e interligação.

2.2.3.3 Posicionamento e interligação

Como resultado obtido na etapa de síntese, tem-se um arquivo contendo uma rede de ligações entre os elementos disponíveis na tecnologia empregada. O formato de arquivo gerado nessa etapa depende da ferramenta de posicionamento e interligação, alguns exemplos de arquivos são o formato EDIF (do original em inglês *Eletronic Design Interchange Format*) e o formato BIT (do original em inglês *Bitstream Format*) (D'AMORE, 2005).

O arquivo gerado pela ferramenta de posicionamento e interligação é carregado no dispositivo definindo assim os caminhos de interligação. Os processos de posicionamento e interligação são dois processos mutuamente dependentes. O processo de posicionamento é quem realiza a atribuição de componentes da FPGA aos componentes lógicos do projeto. O processo de interligação é quem realiza a interconexão na FPGA para comunicação entre os componentes deste dispositivo. As ferramentas de posicionamento e interligação são, normalmente, fornecidas pelos próprios fabricantes devido às particularidades de cada tecnologia (D'AMORE, 2005). A Figura 9 representa o processo de posicionamento e interligação em que inicialmente é fornecido o arquivo de ligações e posteriormente utilizado

o *software* de roteamento responsável pelo posicionamento e interligação no dispositivo lógico de destino.

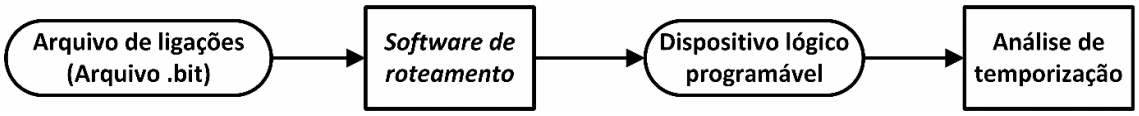

Figura 9 – Processo de posicionamento e interligação
Fonte: autoria própria.

Finalizada a etapa de posicionamento e interligação, o dispositivo lógico programável está pronto para a síntese da descrição criada pelo projetista, podendo-se iniciar a etapa de análise de temporização da rede de ligações. A análise da temporização é feita com a utilização de ferramentas especificas, que verificam se o circuito projetado apresenta o tempo de propagação esperado. "A temporização nesta etapa é precisa pois a ferramenta dispõe de informações sobre o caminho das interligações e consequentemente o atraso gerado por elas" (D'AMORE, 2005, p. 6).

2.3 *SOFTWARE* DE DESENVOLVIMENTO

2.3.1 ISE *Design Suite*

Para o desenvolvimento do trabalho será utilizado o *software* ISE *Design Suite* 14.7, desenvolvido pela Xilinx®. A versão utilizada será a 14.7, última liberada pelo fabricante, que encerrou o ciclo do produto em 23 de outubro de 2013. O ambiente permite o desenvolvimento de projetos de sistemas digitais para CPLDs e FPGAs fabricados pela Xilinx®, utilizando VHDL, Verilog e diagrama de componentes. Os CPLDs e FPGAs fabricados pela Xilinx® que possuem suporte para o ISE são: *CoolRunner* XPLA3, *CoolRunner-II*, XC500 CPLD, *Virtex*-4 FPGA, *Virtex*-5 FPGA, *Virtex*-6 FPGA, *Virtex*-7 FPGA, *Kintex*-7 FPGA, *Artix*-7 FPGA, *Spartan*-3 FPGA, *Spartan*-3A FPGA, *Spartan*-3AN FPGA, *Spartan*-3E FPGA, *Spartan*-3A DSP FPGA, *Spartan*-6 FPGA (XILINX, 2013b). A interface geral do *software* de desenvolvimento é apresentada na Figura 10.

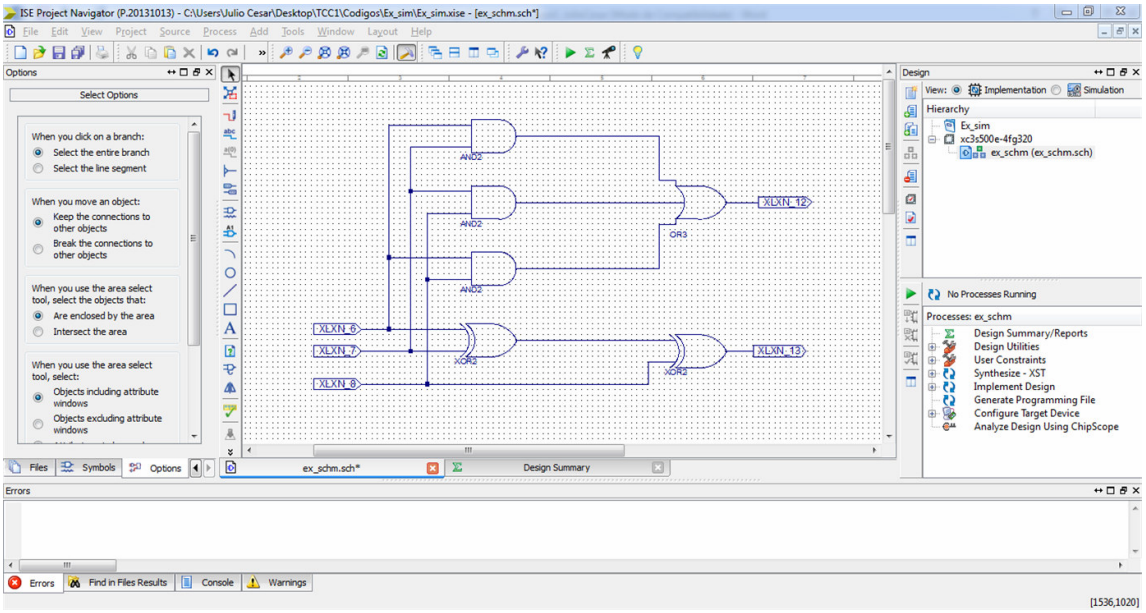

Figura 10 – Interface *software* de desenvolvimento
Fonte: autoria própria.

No campo *Hierarchy* pode-se verificar a hierarquia do projeto, onde o arquivo principal é mantido sempre no topo. Na aba *Design* têm-se as ferramentas disponíveis pelo ISE para o processamento dos arquivos do projeto criado, onde as ferramentas mostradas nesse campo dependem do arquivo selecionado. A aba *Design* é apresenta na Figura 11.

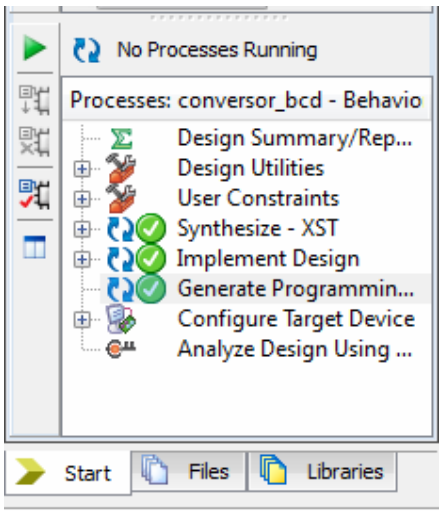

Figura 11 – Aba *Design* do *software ISE*
Fonte: autoria própria.

Os campos *Synthesize – XST* e *Implement Design* presentes na aba *Design* são responsáveis por sintetizar e implementar o projeto. Esses procedimentos criam um arquivo que contém as informações sobre as ligações que devem ser feitas dentro da FPGA garantindo

assim que seu funcionamento se comporte conforme foi projetada. O campo *Generate Programming File* é responsável por criar o arquivo de programação que será carregado dentro da FPGA.

Para carregar o arquivo de programação dentro da FPGA será utilizada a *software Impact*. Este *software* é uma ferramenta que se encontra dentro do *software* ISE e permite que o projetista detecte o dispositivo lógico conectado ao computador e carregue o arquivo de programação. Com o arquivo de programação carregado no *kit* de desenvolvimento a etapa de testes pode ser iniciada.

3 MATERIAIS E MÉTODOS

Neste Capítulo serão, inicialmente, apresentados os componentes a serem desenvolvidos, o funcionamento e a aplicabilidade de cada um. Em seguida, será descrita a metodologia adotada para a implementação de cada um dos componentes.

3.1 MATERIAIS

Nesta seção serão apresentados os componentes presentes no *kit*, o funcionamento e aplicabilidade de cada um.

3.1.1 Chaves deslizantes

O *kit* de desenvolvimento conta com quatro chaves deslizantes que podem ser configuradas como entrada de dados. As chaves deslizantes, quando colocadas na posição LIGADO, recebem uma tensão 3.3 V sinalizando nível lógico alto, e quando colocadas na posição DESLIGADO, recebem 0 V sinalizando nível lógico baixo (XILINX, 2006). As chaves deslizantes são apresentadas na Figura 12.

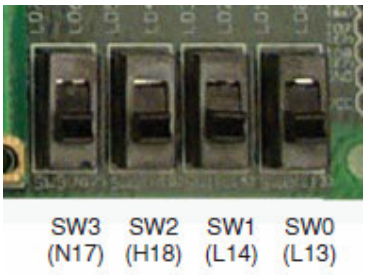

Figura 12 - Chaves deslizantes
Fonte: Xilinx (2006, p. 15).

3.1.2 *Push-buttons*

O *kit* dispõe de quatro *push-buttons*. Os *push-buttons* funcionam como sinais de entrada e são utilizados em projetos que necessitam de botões, tais como para *reset* de um sistema digital. O acionamento dos *push-buttons* é representado na Figura 13. Quando os *push-buttons* são pressionados, o sinal correspondente assume nível lógico 1. No caso contrário, o sinal permanece em nível lógico 0 (XILINX, 2006).

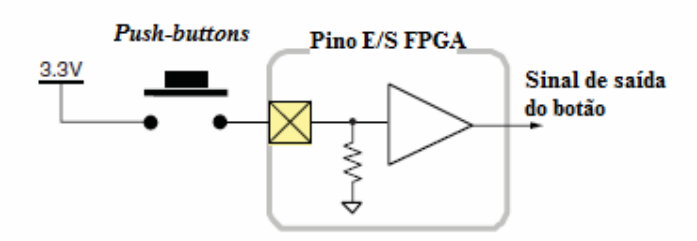

Figura 13 – Circuito dos *push-buttons* do *kit*
Fonte: adaptado Xilinx (2006, p. 17).

3.1.3 *Encoder* rotativo

O *encoder* rotativo presente no *kit* possui dois modos de funcionamento. No primeiro modo, o *encoder* se comporta como um *push-button* podendo operar como *reset* de um sistema digital, por exemplo. No segundo modo, a medida que o *encoder* é girado, os estados das saídas das A e B são modificados. As entradas, quando abertas, apresentam nível lógico alto na saída e quando fechadas apresentam nível lógico baixo (XILINX, 2006). O acionamento do *encoder* é apresentado na Figura 14.

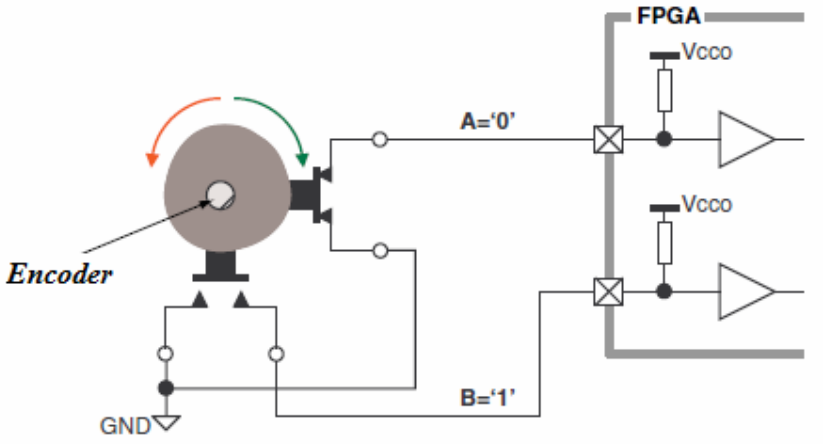

Figura 14 – Circuito ativação do *encoder* presente no *kit*
Fonte: adaptado Xilinx (2006, p. 18).

Inicialmente, quando o *encoder* não é girado, as chaves dos canais A e B encontram-se fechadas, apresentando 0 na saída. Quando o *encoder* é girado no sentido horário, a chave A que estava fechada é aberta apresentado 1 na saída, seguido pela chave B. Quando girado no sentido anti-horário, a chave B que estava fechada é aberta apresentando 1 na saída, seguido pela chave A (XILINX, 2006). A saída do sinal gerado pelas chaves é apresentada na Figura 15.

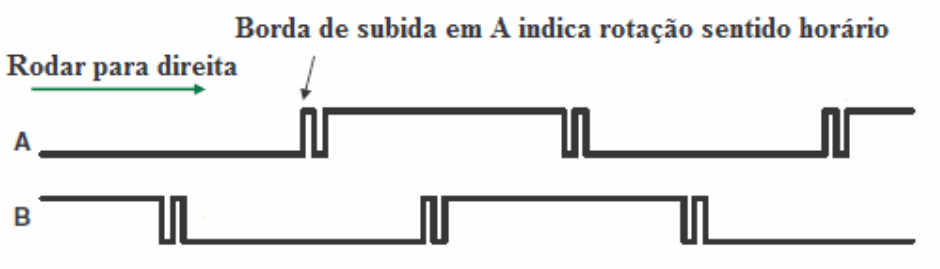

Figura 15 - Saída *encoder*
Fonte: adaptado Xilinx (2006, p. 19).

3.1.4 Conjunto de *Leds*

O *kit* dispõe de oito *leds*, que podem ser utilizados para representar as saídas de dados de um sistema digital. Cada *led* funciona individualmente, representando uma saída digital (XILINX, 2006). Os *leds* são apresentados na Figura 16.

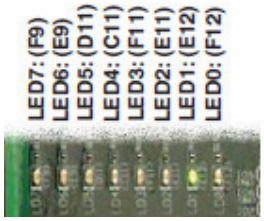

Figura 16 - *Leds*
Fonte: Xilinx (2006, p. 19).

3.1.5 *Display* de LCD

O *display* presente no *kit* possui um controlador gráfico interno fabricado pela Sitronix, modelo ST7066U. Esse controlador possui três memórias semicondutoras internas, cada uma possuindo um propósito específico. A primeira memória é a *DD RAM* (do original em inglês *Display Data Random Acess Memory*), que armazena os caracteres que são apresentados no *display*. A memória *DD RAM* possui capacidade de armazenar 40 caracteres por linha, mas como o *display* possui 16 caracteres por linha os demais endereços não são utilizados. A segunda memória é a *CG ROM* (do original em inglês *Character Generator Read-Only Memory*), que armazena o formato do *bitmap* de cada um dos caracteres predefinidos que a tela do *display* pode exibir. A terceira memória é a *CG RAM* (do original em inglês *Character Generator Random Acess Memory*), que fornece o espaço para criar *bitmaps* de caracteres personalizados (XILINX, 2006).

Além do controlador, o *display* possui alguns pinos de controle. O LCD_E é responsável por habilitar ou desabilitar as operações de leitura/escrita, LCD_RS é responsável por fazer a seleção de registrador interno e LCD_RW é responsável pelo controle de leitura/escrita do *display*. A interface do *display* com a FPGA é apresentada na Figura 17.

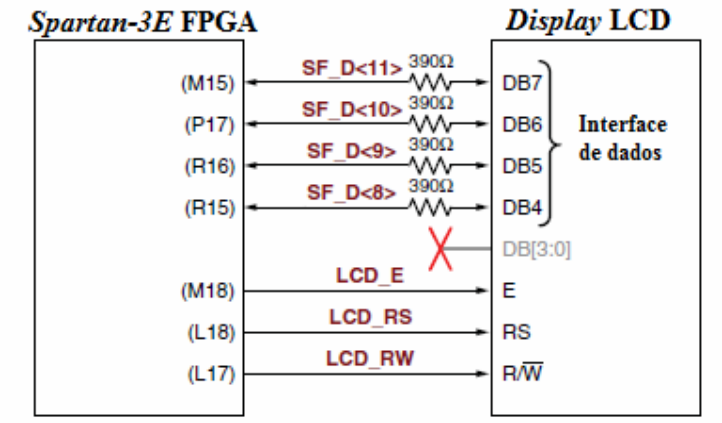

Figura 17 - Interface do *display* LCD
Fonte: adaptado Xilinx (2006, p. 41).

3.1.6 Porta PS/2

A porta PS/2 presente no *kit* inclui entrada para mouse e teclado. No entanto, para o desenvolvimento da biblioteca será considerado somente entrada para teclado. O teclado conectado na porta PS/2 do *kit* será utilizado como um extensor para entrada de dados, permitindo que o valor de cada tecla seja representado na saída por 8 *bits.* A entrada PS/2 possui seis pinos, dos quais somente quatro são utilizados, sendo um pino para alimentação, um para *GND*, um para dados e um para o *Clock* (XILINX, 2006). A porta PS/2 é representada na Figura 18.

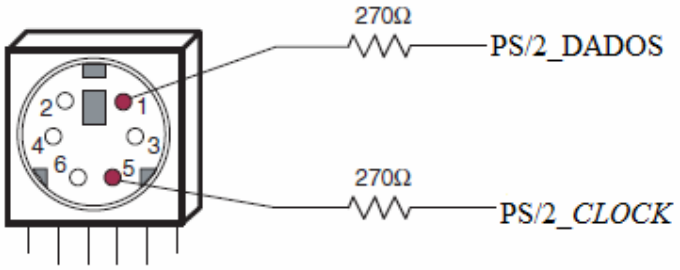

Figura 18 - Porta PS/2
Fonte: adaptado Xilinx (2006, p. 61).

3.1.7 Conversor Digital/Analógico

O conversor D/A presente no *kit* é o LTC2624 fabricado pela *Linear Technology*. Esse conversor conta com 12 *bits* de resolução e possui quatro conversores independentes (LINEAR TECHNOLOGY, 2004b). Os dados digitais de entrada desse conversor são enviados por meio de uma interface SPI (do original em inglês *Serial Peripheral Interface*). Os dados são por ela serialmente enviados, utilizando um único canal de comunicação, e a velocidade depende da frequência de transmissão do circuito (XILINX, 2006). O conversor D/A LTC2624 é representado na Figura 19.

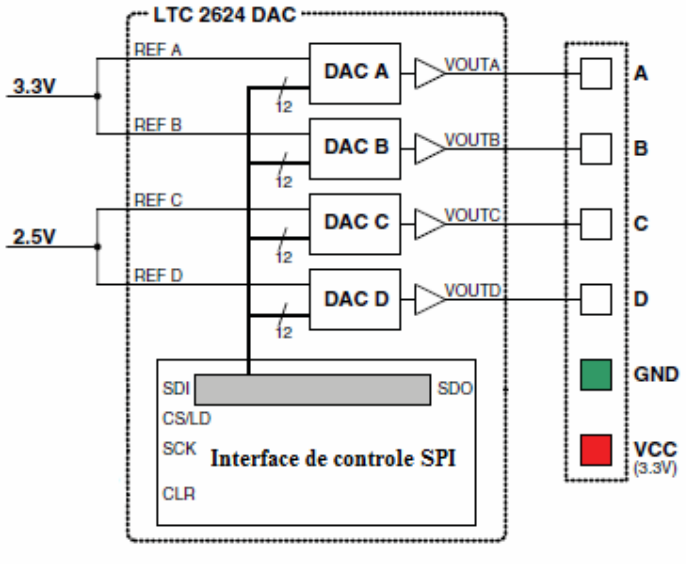

Figura 19 - Conversor D/A LTC2624
Fonte: adaptado Xilinx (2006, p. 68).

3.1.8 Conversor Analógico/Digital

O conversor A/D presente no *kit* é o LTC1407A-1 fabricado pela *Linear Technology*. Esse conversor conta com dois canais de entrada indepedentes, que são amostrados simultaneamente na borda de subida do sinal de início de conversão. A taxa máxima de conversão é de 1,5 Msps por canal. Cada saída possui 14-*bit* de resolução (LINEAR TECHNOLOGY, 2004a).

Para realizar o ajuste do sinal analógico que entra no conversor A/D é utilizado o dispositvo LTC6912-1 fabricado pela *Linear Technology*. O LTC6912-1 é um amplificador de ganho programável digitalmente. Esse amplificador possui canal duplo em que os ganho de

cada canal são programados de maneira independente utilizando a interface SPI (LINEAR TECHNOLOGY, 2004a). O conversor A/D e o amplificador são representados na Figura 20.

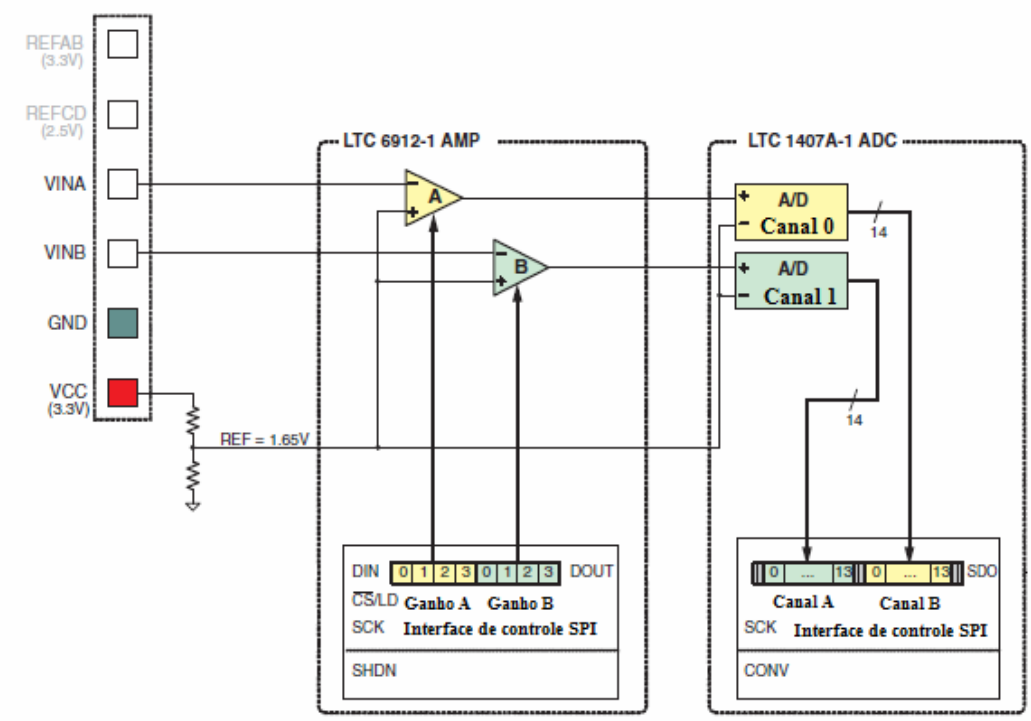

Figura 20 - Conversor A/D e amplificador
Fonte: adaptado Xilinx (2006, p. 74).

3.2 MÉTODOS

A abordagem metodológica adotada neste trabalho foi o desenvolvimento experimental que é caracterizado pelo trabalho criativo levado a cabo de forma sistemática para aumentar o campo dos conhecimentos (OCDE, 2002). Para o desenvolvimento dos componentes foi proposta uma metodologia de desenvolvimento que é apresentada de maneira geral na Figura 21.

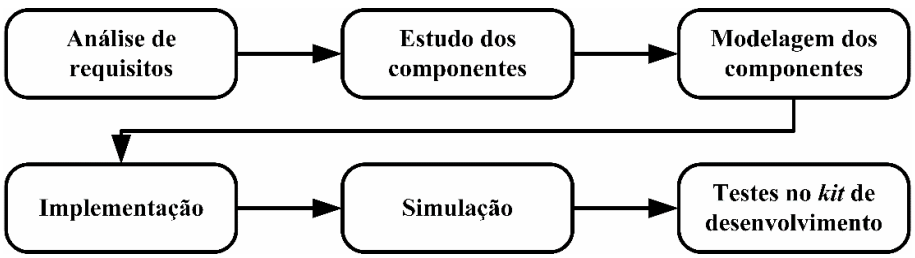

Figura 21 – Metodologia de desenvolvimento
Fonte: autoria própria.

3.2.1 Chaves deslizantes/*Push-buttons*

O acesso às chaves deslizantes e aos *push-buttons* se dá por meio da descrição no arquivo de restrições de projeto UCF (UCF do original em inglês *User Constraints File)*, em que o projetista declara a respectiva chave ou *push-button* a ser utilizado. O arquivo UCF é no formato texto, em que cada linha iniciada com a palavra "NET" representa um pino da FPGA. A palavra "NET" representa o nome do sinal dentro do circuito, o parâmetro "LOC" é a identificação do pino da FPGA, o parâmetro "IOSTANDARD" representa o tipo de tensão na entrada/saída (LVTTL representa sinal TTL de 3,3 V) e o último parâmetro representa o tipo de resistor a ser empregado na chave ou *push-button* (XILINX, 2008).

3.2.2 *Encoder* rotativo

O *encoder* será modelado nesse trabalho como um indicador de posição, em que cada posição absoluta do *encoder* será apresentada por um valor numérico. Sabe-se que quando detectada primeiramente uma borda de subida no canal A do *encoder*, o qual está sendo girado no sentido horário e quando detectada no canal B, o *encoder* está sendo girado no sentido anti-horário. Deste modo, quando o *encoder* for girado no sentido horário a posição é incrementada e quando for girado no sentido anti-horário a posição é decrementada. O *push-button* do *encoder* será modelado como botão de *reset*, em que quando pressionado o indicador volta para a posição inicial. Para acesso ao *push-button* do *encoder* será feita a descrição no arquivo de restrições de projeto UCF.

Para que o *encoder* apresente o funcionamento proposto foi necessário incialmente implementar um circuito *anti-bounce* para os canais A e B do *encoder*. O funcionamento do circuito *anti-bounce* consiste em atrasar os sinais dos canais A e B evitando que esses canais gerem "repiques" no sinal resultante. Com o circuito *anti-bounce* implementado, o segundo passo foi implementar um processo que verificasse se o *encoder* foi rotacionado.

O processo implementado realiza uma verificação a cada evento de *Clock*, a verificação feita por este processo consiste em detectar quando o *encoder* foi rotacionado podendo assim atualizar o indicador de posição. Após a atualização do indicador de posição uma decodificação é realizada transformando o valor do indicador em uma palavra de 8 *bits*. A palavra de 8 *bits* gerada é apresentada na saída do componente criado.

3.2.3 Conjunto de *Leds*

O acesso aos *leds* se dá por meio da descrição no arquivo de restrições de projeto UCF, em que o projetista declara o respectivo *led* a ser utilizado. O parâmetro "SLEW" presente no arquivo UCF representa a taxa de velocidade de cada saída individual do *leds* presente no dispositivo e o parâmetro "DRIVE" especifica a capacidade de corrente disponibilizada para os *buffers* de entrada/saída (XILINX, 2008).

3.2.4 *Display* de LCD

Para que o dado possa ser exibido no *display*, é necessário que ele seja gravado na memória *DD RAM*. O dado armazenado na *DD RAM* contém o endereço do caractere armazenado na memória *CG ROM*. O endereço de cada caractere é representado por 8 *bits*, onde os 4 primeiros são referentes ao *nibble* superior, e os 4 restantes são referentes ao *nibble* inferior (XILINX, 2006). O endereço dos caracteres da memória *CG ROM* são apresentados na Figura 22.

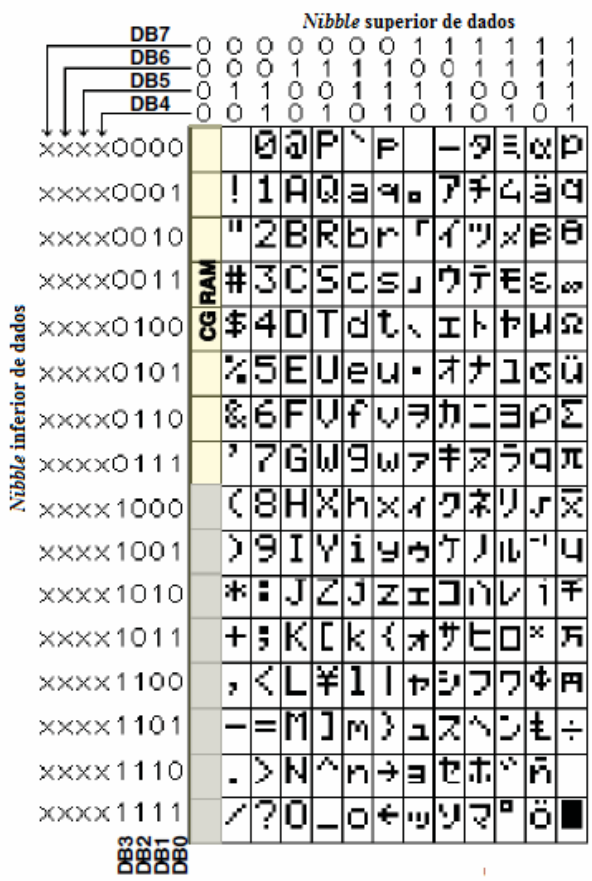

Figura 22 - Endereço dos caracteres da memória *CG ROM*
Fonte: Xilinx (2006, p. 45).

Para realizar a operação de escrita na memória *DD RAM* é necessário a inicialização dos pinos de controle. Primeiramente, é selecionada a fonte de *Clock*. Nesse caso será utilizado o oscilador presente no *kit* de frequência 50MHz. Após a seleção da fonte de *Clock* o pino LCD_RS é colocado em nível lógico alto, indicando que um dado será enviado, e o pino LCD_RW é colocado em nível lógico baixo, habilitando o *display* para a escrita de dados. Os pinos LCD_RS e LCD_RW devem permanecer no estado em que foram configurados por um tempo de 40 ns antes que o pino LCD_E seja habilitado para a escrita de dados. Após os 40 ns, o pino LCD_E é colocado em nível lógico alto, permanecendo nesse estado por 230 ns, o equivalente a 12 ciclos de *Clock.* Na operação de leitura os pinos LCD_RS e LCD_RW são colocados em nível lógico alto. O pino LCD_RS em nível lógico alto sinaliza que um dado será enviado, e o pino LCD_RW habilita o *display* para a leitura de dados. Para a operação de leitura, deve-se obedecer às mesmas especificações de tempo utilizado na operação de escrita (XILINX, 2006). Os tempos mínimos para a escrita/leitura de dados no *display* são apresentados na Figura 23.

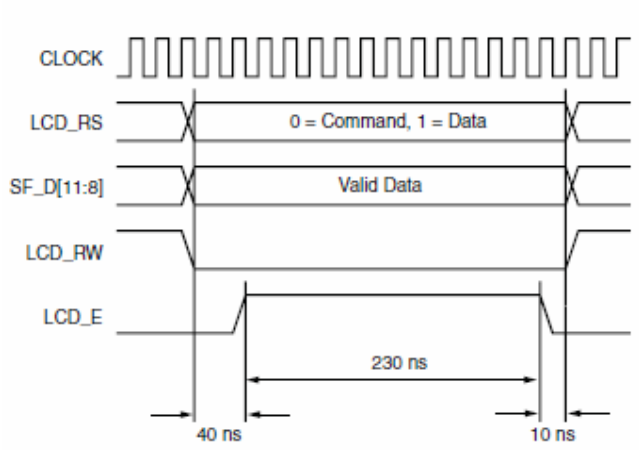

Figura 23 - Tempos mínimos para escrita/leitura de dados
Fonte: Xilinx (2006, p. 50).

Entendido os procedimentos necessários para a utilização do *display* a etapa de implementação da interface de inicialização e controle pode ser iniciada. A metodologia adotada para a implementação da interface foi desenvolvida utilizando uma máquina de estados. Uma máquina de estados pode ser implementada de duas maneiras. A primeira seria utilizando uma máquina de Moore, a segunda seria utilizando uma máquina de Mealey. O comportamento das máquinas de Moore e Mealey são idênticos, porém suas implementações são diferentes. A máquina de Moore possui uma função que gera uma palavra para cada estado da máquina e a saída só depende do estado atual da máquina. Já uma máquina Mealey gera uma palavra de saída para cada transição entre os estados, em que cada palavra de saída depende do estado atual e de seu valor de entrada (VIEIRA, 2006). A máquina de estados implementada para a interface do *display* foi baseada no funcionamento da máquina de Moore. A Figura 24 representa a máquina de estados implementada para inicialização e controle do *display*.

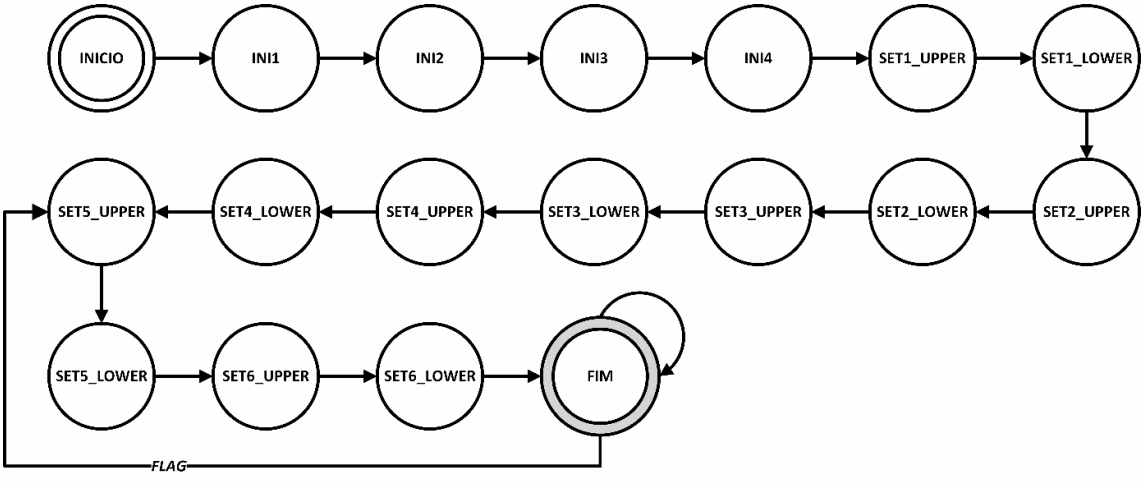

Figura 24 - Máquina de estados *display*
Fonte: autoria própria.

Os estados INI1, INI2, INI3 e INI4 são responsáveis por estabelecer a sequência de inicialização do *display*. O barramento SF_D nos estados INI1, INI2, INI3 recebe o valor 0x03 e no estado INI4 recebe o valor 0x02. Essa sequência de valores é responsável por inicializar o controlador do *display*. Após a inicialização do *display* é necessário realizar sua configuração. Inicialmente é configurado o tamanho da interface dos dados e o número de linhas que o *display* irá operar. Os estados responsáveis por essas configurações são os estados SET1_UPPER e SET1_LOWER nesses estados o barramento SF_D recebe o valor 0x28 que configura o *display* para a interface de dados de 4 *bits* e operação no modo de duas linhas.

Após a configuração do tamanho da interface de dados e o modo de operação das linhas é feita a configuração do cursor do *display*. O cursor é configurado com incremento automático de endereço, e após cada incremento ele é deslocado para a direita, ficando assim posicionado no próximo endereço da memória a ser escrito. Os estados responsáveis por essas configurações são SET2_UPPER e SET2_LOWER. Nesses estados o barramento SF_D recebe o valor 0x06.

Finalizada a configuração do cursor é feita a configuração da memória. Nesta aplicação os dados serão armazenados na *DD RAM*. Os estados que realizam a configuração da memória são SET3_UPPER e SET3_LOWER. Nesses estados o barramento SF_D recebe o valor 0x0C. Com a memória configurada é necessário limpar a tela do *display* para que a operação de escrita possa ser realizada. Os estados que realizam está operação são SET4_UPPER e SET4_LOWER nesses estados o barramento SF_D recebe o valor 0x01.

Para realizar a operação de escrita é necessário inicialmente especificar o endereço da *DD RAM* que o dado será armazenado e em seguida especificar o endereço do caractere armazenado na *CG ROM*. Os estados SET5_UPPER e SET5_LOWER são responsáveis por especificar o endereço da *DD RAM* e os estados SET6_UPPER e SET6_LOWER são responsáveis por especificar o endereço do caractere na *CG ROM*. Após a operação de escrita, a máquina de estados fica no estado FIM até receber o sinal de que um novo caractere será escrito no *display*. O sinal responsável por sinalizar que um novo dado será escrito no *display* é a entrada *FLAG*. Quando o sinal apresentar nível lógico alto a máquina de estados retorna ao estado SET5_UPPER, especificando o novo endereço a ser escrito e em seguida escreve o novo caractere no *display*.

3.2.5 Porta PS/2

Um teclado PS/2 utiliza códigos de varredura para comunicar o dado de uma tecla pressionada. Cada tecla possui um único código de verificação que é enviado sempre quando a tecla é pressionada. Quando uma tecla é pressionada e mantida, o teclado envia repetidamente o código de varredura a cada 100 ms. Quando a tecla é liberada, o teclado envia um código de chave F0 seguido da chave de varredura liberado (XILINX, 2006). Os códigos das teclas do teclado PS/2 são apresentados na Figura 25.

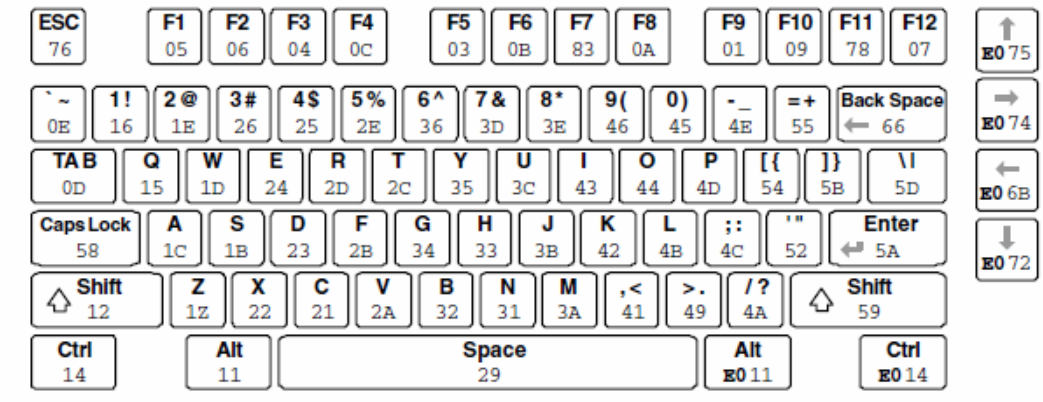

Figura 25 - Códigos teclas teclado PS/2
Fonte: Xilinx (2006, p. 63).

O teclado envia dados em palavras de 11 *bits*. A estrutura dos *bits* enviados é a seguinte: um *bit* de início, oito *bits* do código de varredura em que o *bit* menos significativo

vem primeiro, um *bit* de paridade e um *bit* de parada (XILINX, 2006). A estrutura dos *bits* é representada na Figura 26.

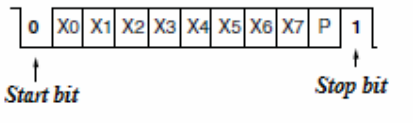

Figura 26 - Estrutura dos *bits*
Fonte: adaptado Xilinx (2006, p. 64).

Para realizar a captura da palavra de 11 *bits* enviada pelo teclado foi necessário incialmente implementar um divisor de frequência. A frequência fornecida pelo oscilador presente no *kit* é de 50MHz, porém o envio dos dados pelo teclado dever ser feito numa frequência de 30KHz. Com a frequência de operação ajustada, o próximo passo é implementar um *debounce* para garantir o recebimento dos dados referente a tecla pressionada. O *debounce* implementado gera um atraso no envio dos *bits* da palavra de 11 *bits*, este atraso garante que os possam ser capturados garantindo assim a consistência da palavra. Com o *debounce* implementado, o processo de captura pode ser implementado. A captura dos dados é realizada na borda de descida do sinal de *Clock*. Então, o processo implementado deve realizar uma verificação a cada evento de *Clock*. Capturada a palavra de 11 *bits*, o valor da tecla pressionada é apresentado na saída do componente criado por vetor um de 8 *bits*.

3.2.6 Conversor Digital/Analógico

Para a utilização do conversor se faz necessário a configuração de seus pinos de controle. Inicialmente, é feita a configuração do pino SPI_SCK responsável pela sincronização da transmissão e recepção dos dados. Após a configuração do pino SPI_SCK, o pino DAC_CS é colocado em nível lógico baixo, habilitando a transmissão de dados. Com a transmissão dos dados habilitada, a FPGA transmite os dados por meio do pino SPI_MOSI. Os dados transmitidos pelo pino SPI_MOSI são capturados pelo conversor LTC2624 na borda de subida do sinal de *Clock* transmitido pelo pino SPI_SCK (XILINX, 2006).

Os dados capturados pelo conversor LTC2624 são enviados para a FPGA na borda de descida do sinal de *Clock*, por meio do pino SPI_MISO, e a FPGA realiza a captura dos dados na próxima borda de subida do sinal de *Clock*. Está operação é realizada até que todos

os 32 *bits* de dados sejam transmitidos. Finalizada a transmissão, o pino DAC_CS é colocado em nível lógico alto desabilitando a transmissão de dados e habilitando o processo de conversão do LTC2624 (XILINX, 2006). A conexão da FPGA com o conversor D/A é apresentada na Figura 27.

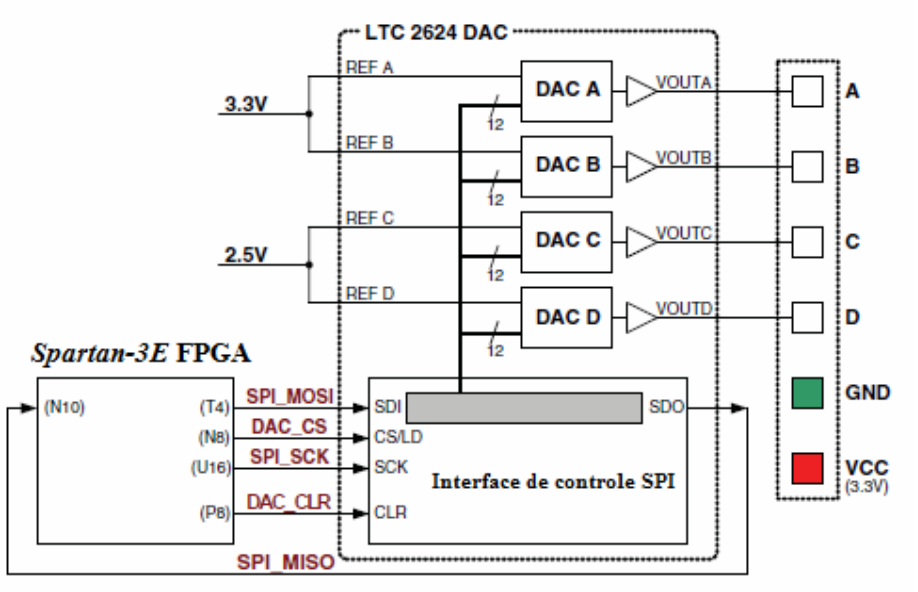

Figura 27 - Conexão da FPGA com o conversor D/A
Fonte: adaptado Xilinx (2006, p. 68).

A interface de controle SPI do conversor D/A apresentada na Figura 28 é formada por um registrador de deslocamento de 32 *bits*, onde cada palavra de 32 *bits* consiste em um comando, um endereço e um valor de dados. A medida que um novo comando entra no conversor D/A, a palavra de comando de 32 *bits* anterior é repetida para a FPGA afim de realizar a confirmação da comunicação. O protocolo da comunicação SPI é apresentado na Figura 28.

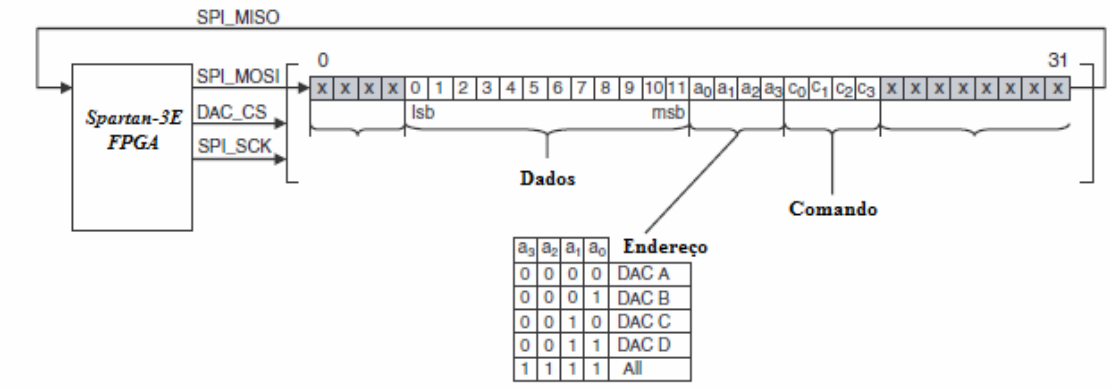

Figura 28 – Protocolo de comunicação SPI
Fonte: adaptado Xilinx (2006, p. 70).

Para implementar a interface SPI apresentada na Figura 28 foi criada uma máquina de estados baseada no funcionamento da máquina de Moore. A Figura 29 representa a máquina de estados implementada.

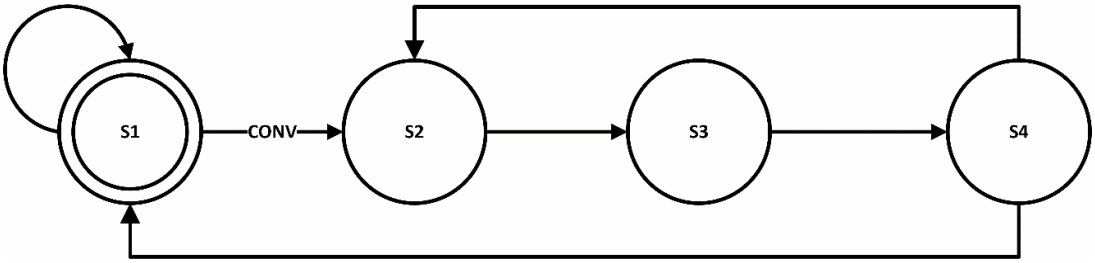

Figura 29 - Máquina de estados interface SPI conversor D/A
Fonte: autoria própria.

A máquina desenvolvida possui quatro estados S1, S2, S3 e S4. No estado S1 é realizado, inicialmente, uma verificação em que caso o sinal de entrada CONV apresente valor igual a um a máquina avança para o estado S2 caso contrário ela permanece no estado S1. A função da entrada CONV no estado S1 é de sinalizar o início da transmissão de um novo sinal digital pela interface SPI. Com a transmissão iniciada no estado S2, o *bit* mais significativo da palavra de 32 *bits* é recebido pelo vetor VET e, em seguida, a máquina avança para o estado S3. A função do vetor VET no estado S2 é receber toda a palavra de 32 *bits* para que a mesma possa ser processada no final da transmissão. No estado S3, o contador CONT é incrementado e máquina avança para o estado S4. O contador CONT em S3 tem a função de indicar a posição que o próximo *bit* a ser transmitido irá ocupar em VET. Em S4 é realizada uma verificação, caso CONT seja menor que 32, a máquina retorna para o estado S2 e VET receber o próximo *bit* a ser transmitido. Caso contrário a máquina retorna para o estado S1 e aguarda uma nova transmissão ser iniciada.

Após transmitir toda a palavra de 32 *bits*, o conversor apresenta na sua saída o respectivo valor analógico. O valor analógico convertido varia de acordo com o valor informado na entrada e com o canal utilizado. Os canais A e B utilizam tensão de referência de 3,3 V ± 5% e os canais C e D utilizam tensão de referência de 2,5 V ± 5%. Está tensão de referência é multiplicada pelo valor digital de 12 *bits* divididos por 2^{12}. As equações (1) e (2) representam as equações de conversão.

$$V_{saída} = \frac{V_D[11:0]}{4096} \mathbf{x}\ (3{,}3\ V) \tag{1}$$

$$V_{saída} = \frac{V_D[11:0]}{4096} \mathbf{x}\ (2{,}5\ V) \tag{2}$$

3.2.7 Conversor Analógico/Digital

Para que um sinal analógico possa ser convertido para digital pelo conversor A/D (LTC2624) é necessário a configuração dos pinos de controle. Inicialmente, é feito o ajuste do ganho do amplificador digital. Os valores de ganhos variam de acordo com a tensão de entrada (XILINX, 2006). A Tabela 1 representa os ganhos de acordo com as faixas de tensão na entrada.

Tabela 1 - Ganhos amplificador

Ganho	A3 B3	A2 B2	A1 B1	A0 B0	Faixa de tensão na entrada Mínimo	 Máximo
0	0	0	0	0		
-1	0	0	0	1	0,4 V	2,9 V
-2	0	0	1	0	1,025 V	2,275 V
-5	0	0	1	1	1,4 V	1,9 V
-10	0	1	0	0	1,525 V	1,775 V
-20	0	1	0	1	1,5875 V	1,7125 V
-50	0	1	1	0	1,625 V	1,675 V
-100	0	1	1	1	1,6375 V	1,6625 V

Fonte: adaptado Xilinx (2006, p. 75).

As faixas de tensão apresentadas na Tabela 1 representam a faixa de operação do conversor para o ganho escolhido. Os valores de tensão fora da faixa de operação não são capturados pelo conversor. Portanto a maior faixa de captura do conversor é de 0,4V até 2,9V.

Após seleção do ganho, de acordo com a tensão apresentada na entrada do conversor, a interface SPI envia o comando de 8 *bits*, consistindo em dois campos de 4 *bits* em que cada campo representa o ganho dos amplificadores A e B. O envio dos dados se inicia quando o pino AMP_CS é colocado em nível lógico baixo. O amplificador realiza a captura dos dados no pino SPI_MOSI na borda de subida do sinal de *Clock*, transmitido pelo pino SPI_SCK. Após a captura, os dados retornam para a FPGA por meio do pino AMP_DOUT na borda de descida do sinal de *Clock* transmitido pelo pino SPI_SCK (XILINX, 2006). O digrama de sinais da interface SPI é apresentado na Figura 30.

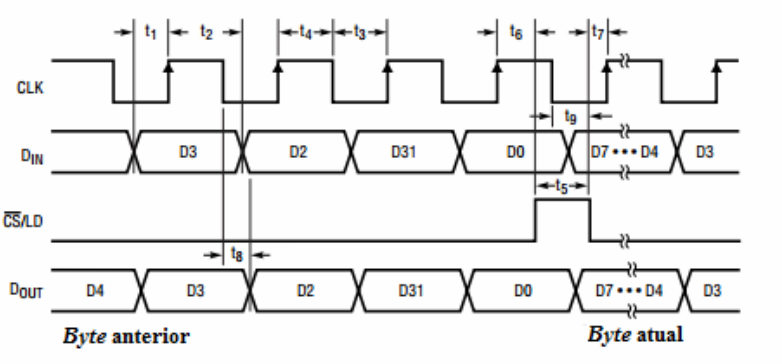

Figura 30 – Diagrama de sinais da interface SPI
Fonte: adaptado Linear Technology (2004a, p. 12).

Com os amplificadores configurados, o pino AD_CONV é colocado em nível lógico alto por 4 ns, habilitando a conversão dos dois canais simultaneamente. Os dados de cada canal são convertidos em dois campos de 14 *bits*, e são enviados para a FPGA por meio do pino SPI_MISO. A transmissão dos dados para a FPGA é iniciada quando o pino AD_CONV é colocado em nível lógico alto novamente por 4 ns, sinalizando o final da conversão (XILINX, 2006). A conexão da FPGA com o amplificador e o conversor A/D é apresentada na Figura 31.

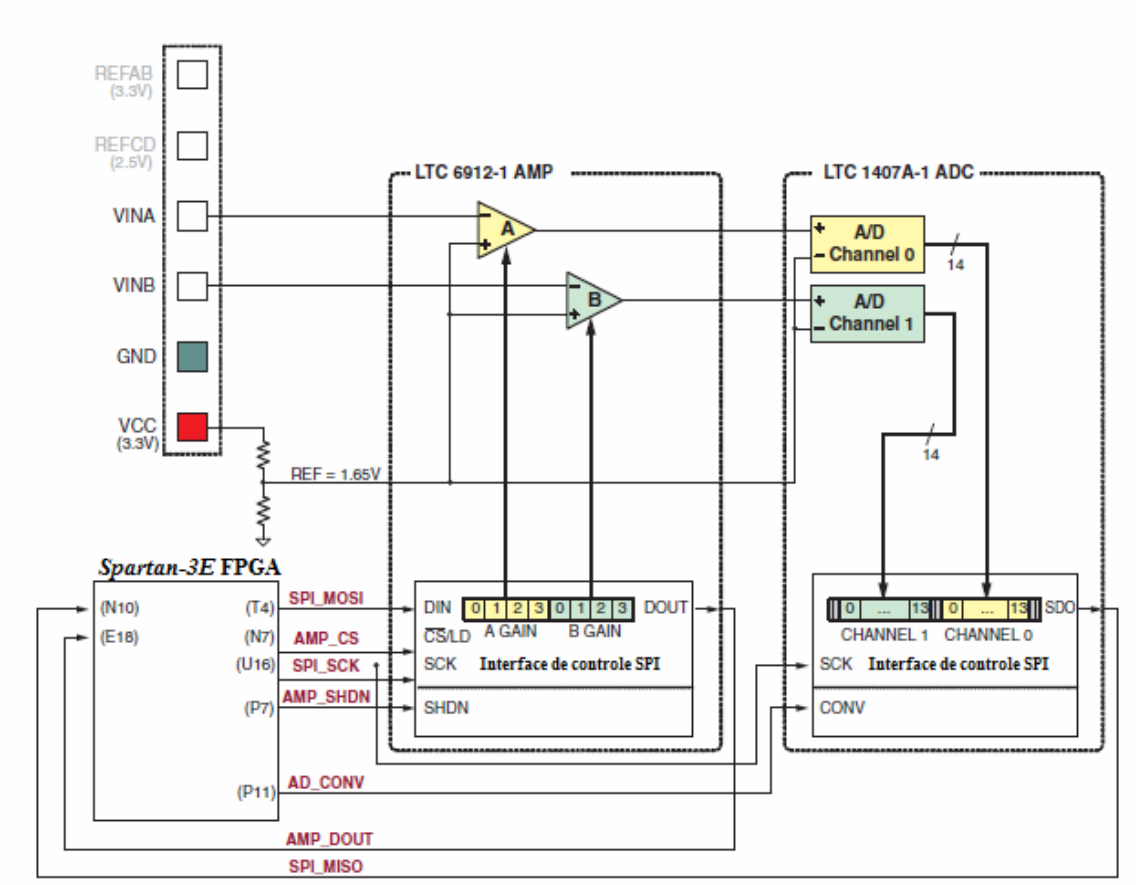

Figura 31 - Conexão da FPGA, amplificador e conversor A/D
Fonte: adaptado Xilinx (2006, p. 74).

A configuração dos amplificadores e o envio dos sinais convertidos são feitos por meio de uma interface de controle SPI. A implementação da interface SPI dos amplificadores e do conversor A/D foi implementada utilizando uma máquina de estados baseada no funcionamento da máquina de Moore. A Figura 32 representa máquina implementada.

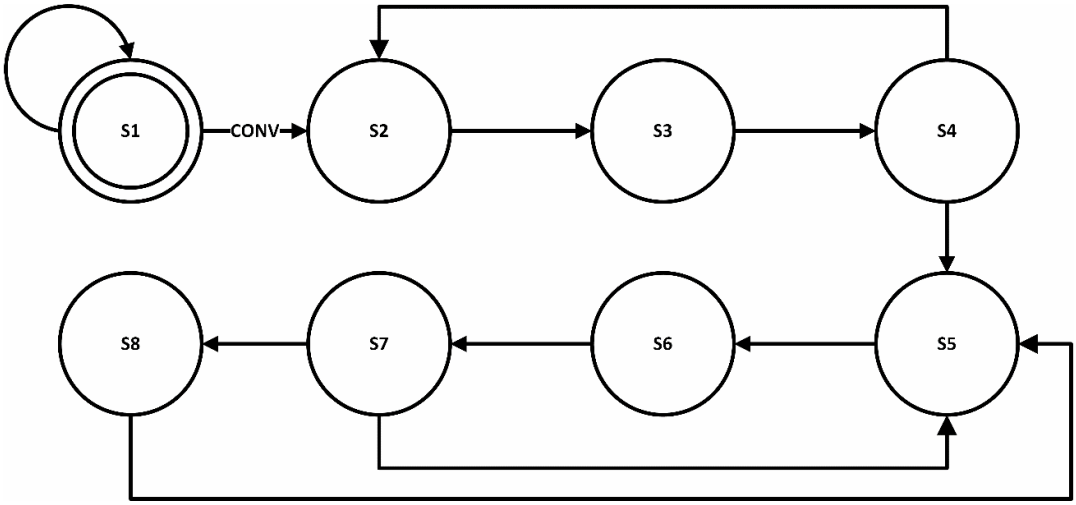

Figura 32 - Máquina de estados interface SPI conversor A/D
Fonte: autoria própria.

A máquina implementada possui oito estados S1, S2, S3, S4, S5, S6, S7 e S8. O funcionamento da máquina é divido em duas etapas: na primeira etapa é realizada a configuração dos amplificadores e na segunda etapa é realizada a captura dos valores analógicos convertidos. No estado S1, assim como na máquina implementada do conversor D/A, é realizada uma verificação por meio do pino de entrada CONV que consiste em sinalizar o início da transmissão de um novo sinal pela interface SPI. No estado S2, o vetor GANHO contém a palavra de 8 *bits* dos ganhos dos amplificadores, em que os 4 primeiros *bits* do vetor representam o ganho do amplificador A e os quatro últimos representam o ganho do amplificador B.

Na primeira execução da máquina no estado S2, o vetor GANHO envia o *bit* mais significativo da palavra de 8 *bits* e avança para o estado S3. Em S3 o contador CONT é incrementando e a máquina avança para S4. No estado S4, uma verificação é realizada. Caso CONT seja menor que 8, a máquina retorna para o estado S2 e o próximo *bit* do vetor GANHO é enviado. Caso contrário, a máquina avança para o estado S5. A segunda etapa da máquina é iniciada no estado S5. Neste estado o vetor ADC recebe o primeiro *bit* da conversão feita pelo canal A e avança para o estado S6. No estado S6, CONT é incrementado e a máquina avança para o estado S7. Em S7 uma nova verificação é feita, caso CONT seja menor que 34 a máquina retorna para o estado S5 e o vetor ADC recebe o próximo *bit* da palavra convertida caso contrário a máquina avança para o estado S8. Em S8 CONT é zerado e máquina retorna para o estado S5 para que os dados convertidos sejam novamente capturados pelo vetor ADC.

O valor digital convertido é apresentado no formato digital de complemento de dois. A representação de complemento de dois é amplamente utilizada nas arquiteturas dos dispositivos computacionais. Como o desenvolvimento deste trabalho visa uma abordagem mais didática, será adotada a representação do valor convertido no formato binário polarizada de 14 *bits*. Adotando esse formato de representação, o conversor irá variar seus valores convertidos de 0 a 16383. A equação de conversão é representada por:

$$D[13:0] = \left(\left(Ganho \times \left(\frac{V_{IN} - 1.65V}{1.25\,V}\right) \times 2^{13}\right) + 2^{13}\right) - 1 \quad . \tag{3}$$

O valor de ganho adotado para essa aplicação foi o valor de -1, pois com esse valor o conversor apresenta sua maior faixa de operação, garantindo assim uma maior aplicabilidade.

4 RESULTADOS

Neste Capítulo serão apresentados os resultados obtidos por meio da metodologia apresenta no Capítulo 3. Inicialmente, é apresentada a biblioteca desenvolvida e na sequência são apresentados os circuitos de teste implementados.

4.1 BIBLIOTECA DESENVOLVIDA

A biblioteca desenvolvida contém os seguintes componentes: *encoder* rotativo, *display* LCD, porta PS/2, conversor D/A e conversor A/D. Todos os componentes presentes na biblioteca foram modelados para o funcionamento em diagrama de componentes. Entretanto para a utilização dos componentes, alguns passos se fazem necessários. Na seção A.1 é apresentada a documentação técnica de cada componente, na seção A.2 é apresentado o tutorial de utilização da biblioteca, na seção A.3 é apresentado os arquivos UCF necessários para a utilização dos componentes implementados e na seção A.4 é apresentada as descrições em VHDL dos componentes implementados.

4.2 CIRCUITOS DE TESTES

Nesta seção são apresentados os circuitos de testes implementados para os componentes desenvolvidos.

4.2.1 *Encoder* rotativo

Como o *encoder* foi modelado para funcionar como um indicador de posição, em que cada posição absoluta do *encoder* apresenta um valor numérico. O exemplo desenvolvido apresenta nos *leds* do *kit* o respectivo valor indicado. A Figura 33 representa o digrama do componente *encoder* conectado aos pinos de entrada e saída.

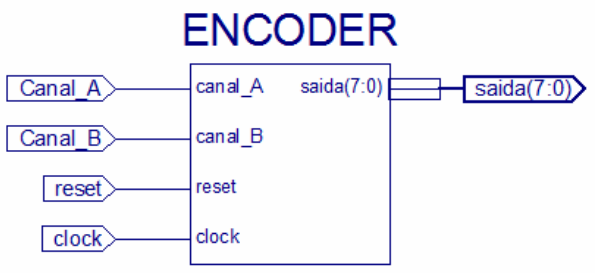

Figura 33 - Diagrama componente *encoder*
Fonte: autoria própria.

Com o circuito montado e o *kit* programado, os testes com o *encoder* são realizados. O valor do indicador de posição do *encoder* na Figura 34 apresenta o valor numérico cinco e a sua saída é apresentada nos *leds*. O valor apresentado no *leds* é 00000101_2, mostrando o funcionamento correto.

Figura 34 - Teste encoder *kit* desenvolvimento
Fonte: autoria própria.

4.2.2 *Display* de LCD

O *display* foi projetado para a escrita de dados, permitindo ao usuário especificar a posição e o caractere desejado. O exemplo de teste desenvolvido contém um decodificador de 8 *bits* para a escolha da posição em que o caractere será escrito e um contador de 8 *bits* que quando habilitado varia o caractere a ser escrito. A Figura 35 representa o circuito de teste do *display*.

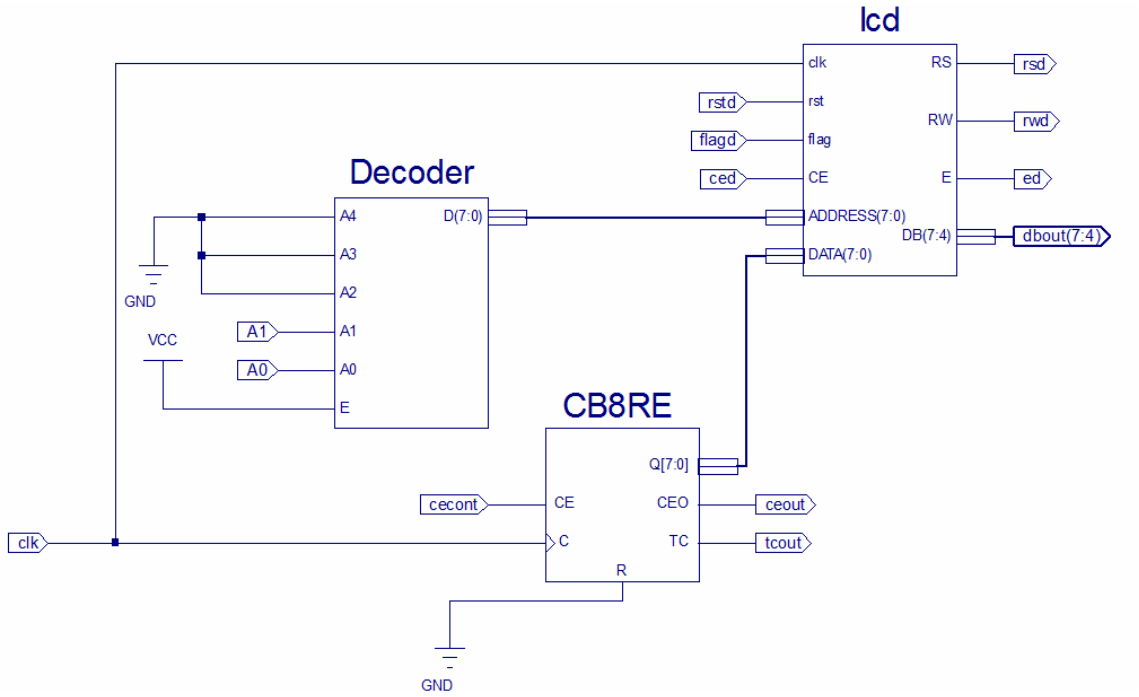

Figura 35 - Circuito esquemático implementado para testes do *display*
Fonte: autoria própria.

O decodificador de endereço do circuito de testes possui cinco entradas. As entradas A0 e A1 são acionadas pelas chaves deslizantes SW2 e SW3 presentes no *kit* e as entradas A2, A3 e A4 são conectadas ao pino terra. O pino CE do contador é responsável por habilitar a contagem. Este pino é acionado pela chave deslizante SW1. Para controle do *display*, o pino CE presente no componente LCD é acionado pela chave deslizante SW0 e os pinos *FLAG*, *RST*, são acionados pelos *push-buttons* H13 e K17. A Figura 36 representa o circuito de teste em funcionamento no *display*.

Figura 36 - Teste *display kit* desenvolvimento
Fonte: autoria própria.

4.2.3 Porta PS/2

O funcionamento do circuito de teste da porta PS/2 consiste em capturar o valor da tecla pressionada no teclado conectado à porta e mostrar seu valor nos *leds* presente no *kit.* A Figura 37 representa o circuito de teste da porta PS/2.

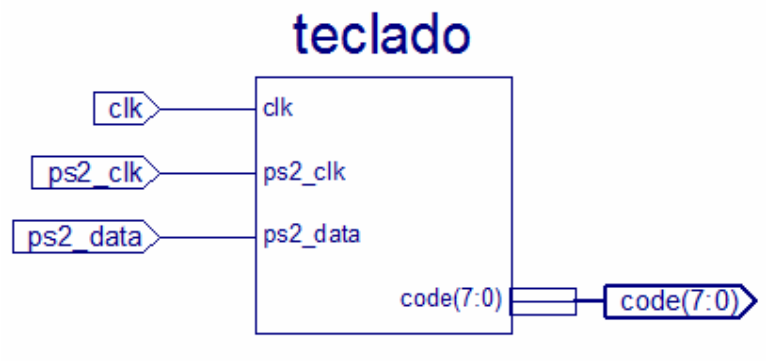

Figura 37 – Diagrama componente teclado
Fonte: autoria própria.

Com o circuito montado e o *kit* programado os testes com o teclado podem ser realizados. Pressionando a tecla A, o valor apresentado na saída apresentada dos *leds* é 00011100_2 que corresponde ao valor em hexadecimal 0x1C. Analisando a Figura 25, na Seção 3.2 é possível ver que o valor em hexadecimal apresentado corresponde a tecla A, demonstrando o funcionamento correto do componente. A Figura 38 representa o funcionamento da porta PS/2 no *kit* de desenvolvimento.

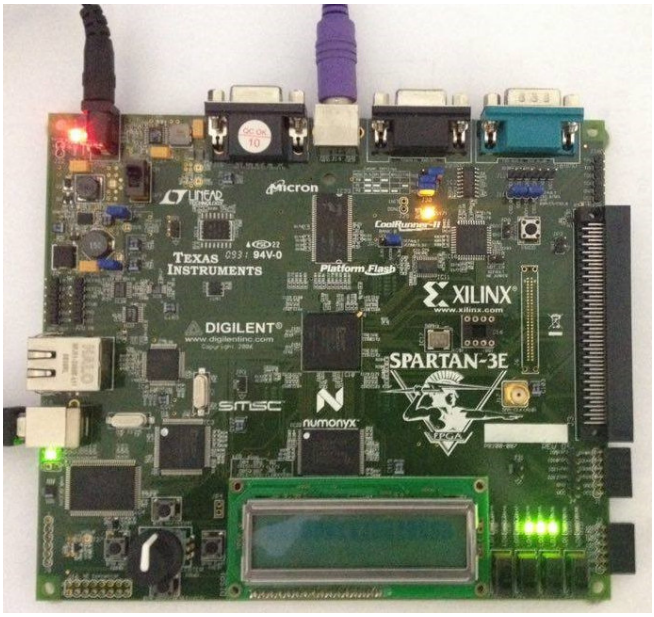

Figura 38 - Teste porta PS/2 *kit* desenvolvimento
Fonte: autoria própria.

4.2.4 Conversor Digital/Analógico

O circuito de teste do conversor D/A implementado possui dois decodificadores sendo um de 12 *bits* e outro de 4 *bits.* O decodificador de 12 *bits* possui três entradas. As entradas A0 e A1 são acionadas pelas chaves deslizantes SW0 e SW1 presentes no *kit* e a entrada A2 é conectada ao pino *VCC*. O funcionamento deste decodificador consiste em fornecer valores digitais para o pino de entrada DATA do conversor D/A. Já o decodificador de 4 *bits* possui duas entradas que são acionadas pelas chaves deslizantes SW2 e SW3. Seu funcionamento consiste em fornecer os valores digitais para o pino de entrada ADDRESS que permite a escolha do canal de conversão que será utilizado. Os valores analógicos convertidos são apresentados na saída de cada conversor e podem ser verificados com a utilização de um multímetro. A Figura 39 representa o circuito de teste do conversor D/A.

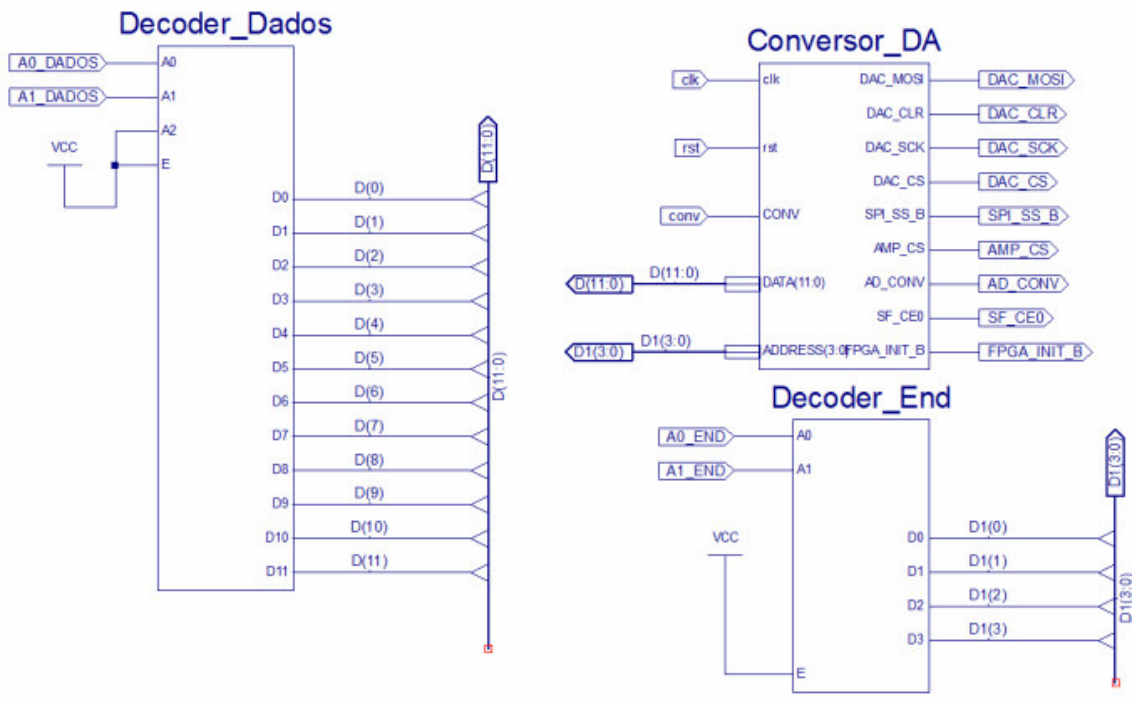

Figura 39 - Circuito esquemático implementado para teste do conversor D/A
Fonte: autoria própria.

Com o circuito montado e o *kit* programado foi possível realizar as medidas dos valores analógicos apresentados nas saídas dos conversores. A Tabela 2 representa os valores de tensão medidos.

Tabela 2 - Valores convertidos D/A

Conversor	Valor digital	Valor analógico medido	Valor analógico esperado	Erro
A	2048_{10}	1,62 V	1,65 V	-1,80%
A	1280_{10}	1,01 V	1,03 V	-1,94%
B	2048_{10}	1,62 V	1,65 V	-1,80%
B	1280_{10}	1,01 V	1,03 V	-1,94%
C	2048_{10}	1,26 V	1,25 V	+0,80%
C	1280_{10}	0,79 V	0,78 V	+1,28%
D	2048_{10}	1,26 V	1,25 V	+0,80%
D	1280_{10}	0,79 V	0,78 V	+1,28%

Fonte: autoria própria.

Os valores de tensão apresentados na Tabela 2 possuem o erro de tensão de referência dentro do especificado pelo fabricante de ±5%. Pode-se assim afirmar que o componente criado está funcionando de maneira correta.

4.2.5 Conversor Analógico/Digital

Para testar o componente A/D foi necessário implementar o circuito de teste da Figura 40.

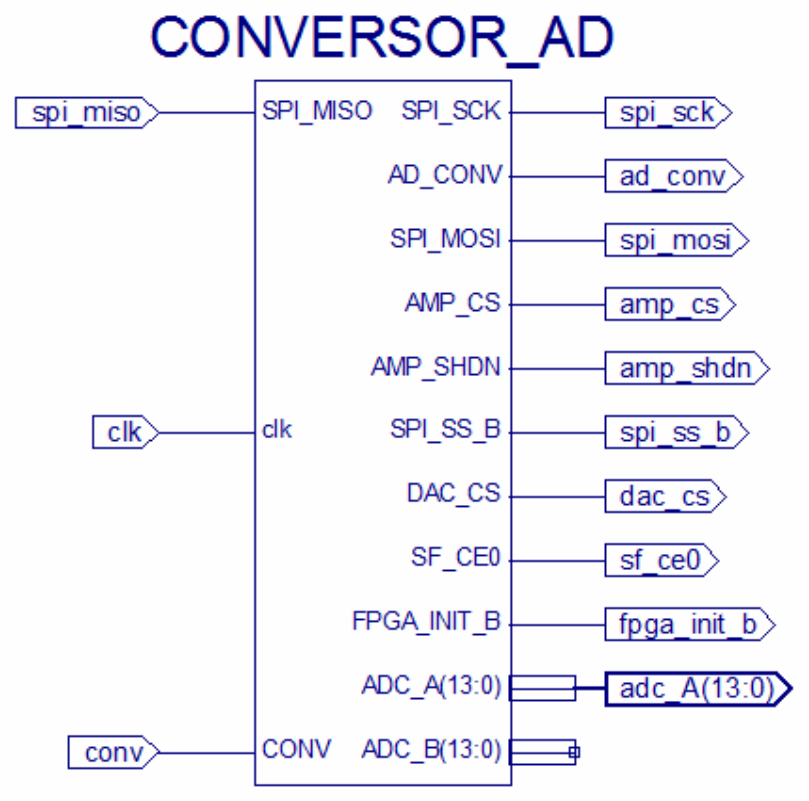

Figura 40 – Diagrama do conversor A/D
Fonte: autoria própria.

Com circuito da Figura 40 montado, foi necessário implementar um circuito externo que fornecesse um valor de tensão para que o conversor realizasse a captura do valor a ser convertido. Para realizar a montagem do circuito externo foi utilizado um potenciômetro com valor de resistência de 100kΩ. A Figura 41 represente a montagem do circuito externo.

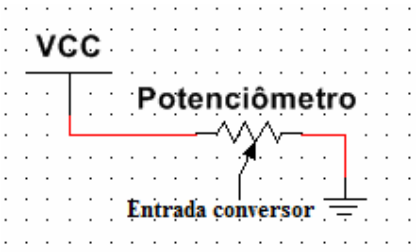

Figura 41 - Circuito externo
Fonte: autoria própria.

Conforme o terminal central do potenciômetro era rotacionado, a tensão na entrada conversor era alterada, fazendo com que o valor convertido fosse alterado. Para visualizar o funcionamento do conversor foram utilizados os *leds* presente no *kit*, em que os 8 *bits* mais significativos são representados. A Figura 42 representa o conversor A/D em funcionamento.

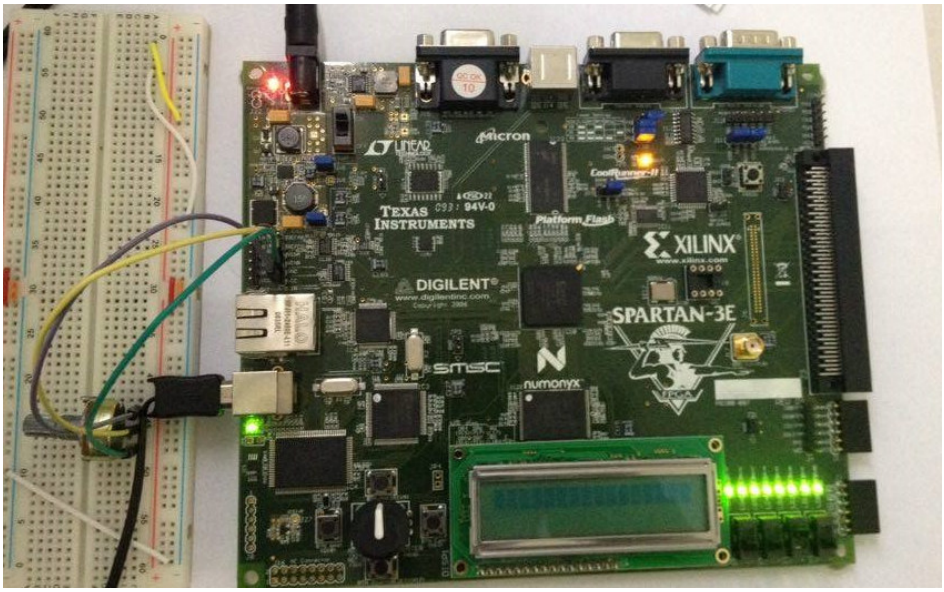

Figura 42 - Teste conversor A/D *kit* desenvolvimento
Fonte: autoria própria.

Após os testes no *kit* de desenvolvimento foram realizas algumas medidas. A Tabela 3 representa os valores digitais convertidos.

Tabela 3 - Valores convertidos A/D

Conversor	Valor analógico	Valor digital convertido	Valor digital calculado
A	0,4 V	16380_{10}	16383_{10}
A	1,0 V	12450_{10}	12450_{10}
A	2,0 V	5590_{10}	5597_{10}
A	2,5 V	2620_{10}	2620_{10}
B	0,4 V	16380_{10}	16383_{10}
B	1,0 V	12450_{10}	12450_{10}
B	2,0 V	5590_{10}	5597_{10}
B	2,5 V	2620_{10}	2620_{10}

Fonte: autoria própria.

Os valores digitais convertidos apresentados na Tabela 3 ficaram muito próximos dos valores digitais calculados, permitindo assim afirmar que o conversor está funcionando de maneira correta.

5 CONCLUSÃO

Neste trabalho foi apresentado o desenvolvimento de uma biblioteca utilizando a linguagem VHDL que permitisse o acesso aos componentes presentes na *Spartan-3E Starter Kit Board*. Dentre os componentes presentes no *kit* os escolhidos para compor a biblioteca desenvolvida foram: *encoder* rotativo, *display* LCD, porta PS/2 e conversores A/D e D/A. A escolha destes componentes é decorrente deles poderem ser utilizados em abordagens didáticas.

Os componentes desenvolvidos foram modelados para serem utilizados na metodologia de projeto de diagrama esquemático. Para realizar esta modelagem foi necessário o entendimento do funcionamento de cada componente. Entendido o funcionamento foi feita uma modelagem para que os componentes pudessem ser utilizados na metodologia de projeto escolhida. Algumas dificuldades foram encontradas na etapa de modelagem dos componentes pois em alguns casos a modelagem adota não proporcionava o funcionamento correto.

A escolha da metodologia de projeto de diagrama de componentes se deve ao fato da mesma permitir que a descrição em VHDL do componente seja encapsulada, e o componente seja representado por um bloco contendo as entradas e saídas, tornando assim possível o desenvolvimento de projetos por usuários sem o conhecimento sobre a linguagem VHDL.

Além de possibilitar o desenvolvimento de projetos por usuários sem conhecimento sobre a linguagem VHDL, a biblioteca desenvolvida proporciona também uma integração do usuário com o *kit* de desenvolvimento, permitindo a visualização da simulação na prática. Os componentes desenvolvidos podem, também, ser integrados com os componentes disponíveis da biblioteca padrão do *ISE*, aumentando assim o número de aplicações.

A biblioteca criada pode também ser utilizada em disciplinas de circuitos ou sistemas digitais como ferramenta auxiliar para aprendizado dos alunos, proporcionando assim um primeiro contato com dispositivos lógicos programáveis.

Para o desenvolvimento de futuros projetos, novos componentes podem ser criados e adicionados à biblioteca desenvolvida aumentando assim, o número, de componentes que irão complementar o acesso ao *kit* de desenvolvimento, ampliando o uso didático da ferramenta.

REFERÊNCIAS

BEZERRA, Matheus S. **Projeto, implementação e ensaios de um controlador PID utilizando FPGA.** 2010. 55 f. Monografia - Universidade Federal do Ceará, Fortaleza, 2010.

D'AMORE, Roberto. **VHDL: descrição e síntese de circuitos digitais**. Rio de Janeiro: LTC, 2005.

DIGILENT. Disponível em: <http://store.digilentinc.com/spartan-3e-starter-board-limited-time/>. Acesso em: 20 out. 2016.

IEEE. **IEEE P1364-2005**. 2008. Disponível em: <http://www.verilog.com/IEEEVerilog.html>. Acesso em: 29 maio. 2017.

LINEAR TECHNOLOGY. **LTC1407-1/LTC1407A-1**. 2004a. Disponível em: < http://cds.linear.com/docs/en/datasheet/14071fb.pdf>. Acesso em: 19 nov. 2016.

LINEAR TECHNOLOGY. **LTC2604/LTC2614/LTC2624**. 2004b. Disponível em: < http://cds.linear.com/docs/en/datasheet/2604fd.pdf>. Acesso em: 19 nov. 2016.

LIPSETT, Roger; SCHAEFER, Carl F.; USSERY, Cary. **VHDL Hardware Description and Design**. Massachusetts: Kluwer Academic Publishers Norwell, 1993.

OCDE. **Manual de Frascati**. 2002. Disponível em: <http://www.mct.gov.br/upd_blob/0225/225728.pdf>. Acesso em: 09 jul. 2017.

PEDRONI, Volnei A. **Circuit design with VHDL**. Cambridge: Morgan Kaufmann, 2004.

PEDRONI, Volnei A. **Eletrônica digital moderna e VHDL**. Rio de Janeiro: Elsevier, 2010.

VIEIRA, Newton J. **Introdução aos Fundamentos da Computação**. São Paulo: Thomson, 2006.

XILINX. **Spartan-3E Starter Kit Board user guide**. 2006. Disponível em: < https://reference.digilentinc.com/_media/s3e:s3estarter_ug.pdf >. Acesso em: 19 nov. 2016.

XILINX. **Constraints guide**. 2008. Disponível em: < https://www.xilinx.com/itp/xilinx10/books/docs/cgd/cgd.pdf >. Acesso em: 19 nov. 2016.

XILINX. **Spartan and Spartan-XL FPGA families data sheet**. 2013a. Disponível em: < https://www.xilinx.com/support/documentation/data_sheets/ds060.pdf >. Acesso em: 19 nov. 2016.

XILINX. **ISE Design Suite 14: Release Notes, Installation, and Licensing**. 2013b. Disponível em: <

https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/irn.pdf >. Acesso em: 19 nov. 2016.

XILINX. **Spartan-3E FPGA Family Data Sheet**. 2013c. Disponível em: < https://www.xilinx.com/support/documentation/data_sheets/ds312.pdf >. Acesso em: 19 nov. 2016.

APÊNDICES

A.1 COMPONENTES DA BIBLIOTECA *SPARTAN 3E*

Está seção contém a documentação técnica dos componentes presentes na biblioteca *Sparta 3E*. O conteúdo desta documentação técnica aborda o funcionamento geral de cada componente e a especificação dos pinos de entrada e saída.

A.1.1 *Encoder* rotativo

O diagrama do componente *encoder* possui a função de indicar a posição absoluta em que se encontra e apresentar em sua saída o valor numérico correspondente. A Figura 1 representa o diagrama do componente *encoder*.

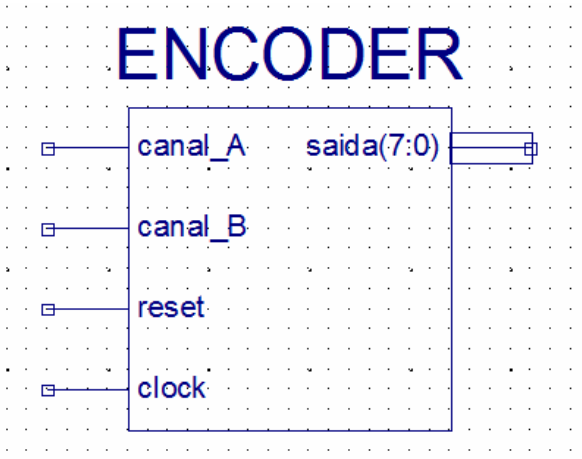

Figura 1 - Componente *encoder*
Fonte: autoria própria.

A Tabela 1 contém as especificações dos pinos de entrada do componente *encoder*.

Tabela 1 - Pinos de entrada *encoder*

Entrada	Funções
Canal_A	Fornece a conexão para o primeiro canal A do componente
Canal_B	Fornece a conexão para o segundo canal B do componente
clock	Fornece *Clock* para o componente
reset	Permite operação de *reset*

	do componente. 0: *reset* desabilitado 1: *reset habilitado*

Fonte: autoria própria.

O pino saída apresenta o valor numérico indicado pela posição que o encoder se encontra, codificado em um vetor de 8 *bits*.

A.1.2 *Display* LCD

O diagrama do componente LCD criado permite a operação de escrita no *display* LCD presente no *kit*. A Figura 2 representa o diagrama do componente LCD.

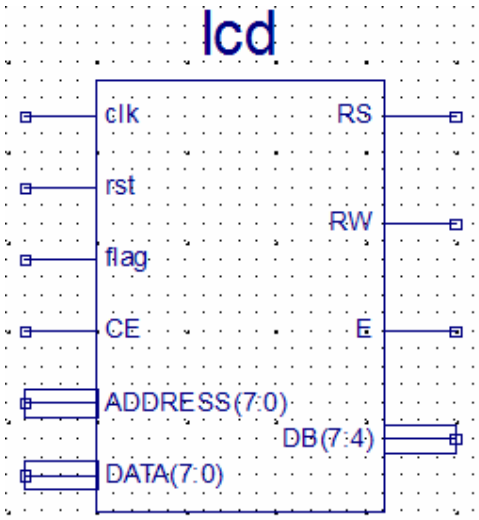

Figura 2 - Componente LCD
Fonte: autoria própria.

A Tabela 2 contém as especificações dos pinos de entrada do componente LCD.

Tabela 2 - Pinos de entrada *display* LCD

Entrada	Funções
clk	Fornece *Clock* para o componente
rst	Permite operação de *reset* do componente. 0: *rst* desabilitado 1: *rst habilitado*
flag	Sinaliza que um novo caractere será escrito no *display*. 0: *flag* desabilitada 1: *flag* habilitada
CE	Permite a escrita de dados no *display*. 0: CE desabilitado 1: CE habilitado

ADDRESS	Endereço em que o caractere será gravado
DATA	Caractere a ser gravado

Fonte: autoria própria.

Os pinos de saída RS, RW, E e DB estão presente no diagrama do componente pois devem ser conectados aos pinos específicos de saída da FPGA garantindo assim o funcionamento correto do componente LCD.

A.1.3 Porta PS/2

O componente teclado quando conectado à porta PS/2 permite realizar a captura do valor da tecla seleciona e apresentar em sua saída o respectivo valor. A Figura 3 representa o diagrama do componente teclado.

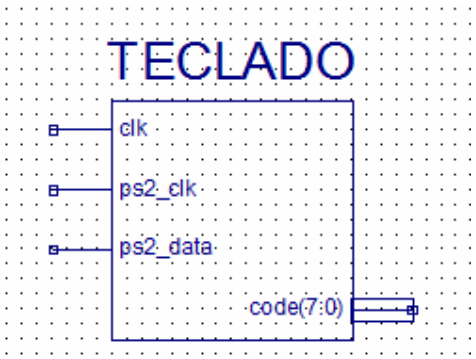

Figura 3 - Componente teclado
Fonte: autoria própria.

A Tabela 3 contém as especificações dos pinos de entrada do componente teclado.

Tabela 3 - Pinos entrada teclado

Entrada	Funções
clk	Fornece *Clock* para o componente
ps2_*clk*	Fornece *Clock* para o envio dos dados
ps2_data	Transmite os dados

Fonte: autoria própria.

A.1.4 Conversor Digital/Analógico

O diagrama do componente D/A permite a conversão de um sinal digital de 12 *bits* em um valor analógico, conta com quatro canais independentes para conversão do sinal onde os canais A e B trabalham com tensão de referência de 3,3 V e os canais C e D com tensão de referência de 2,5 V. A Figura 4 representa o diagrama do componente D/A.

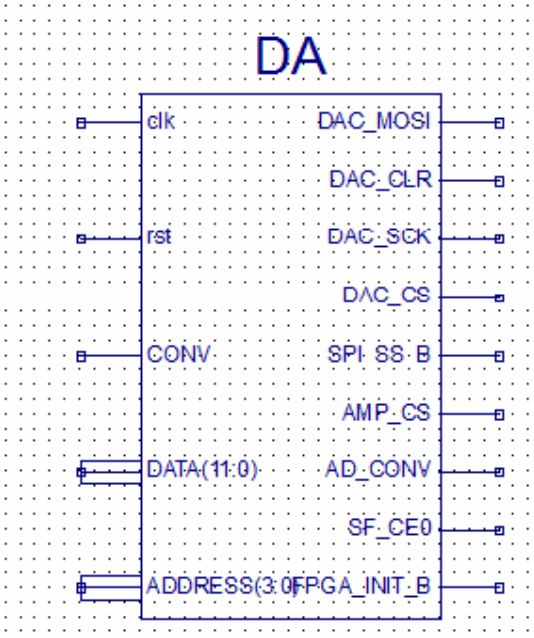

Figura 4 – Componente D/A
Fonte: autoria própria.

A Tabela 4 contém as especificações dos pinos de entrada do componente D/A.

Tabela 4 - Pinos de entrada conversor D/A

Pinos de entrada	Funções
clk	Fornece *Clock* para o componente
rst	Permite operação de *reset* do componente. 0: *rst* desabilitado 1: *rst habilitado*
CONV	Habilita a conversão do sinal. 0: CONV desabilitado 1: CONV habilitado
DATA	Valor de 12 *bits* a ser convertido
ADDRESS	Seleciona o canal do conversor que será utilizado

Fonte: autoria própria.

Os pinos de saída DAC_MOSI, DAC_CLR, DAC_SCK, DAC_CS fazem parte da interface interna de controle do componente D/A. Os pinos SPI_SS_B, AMP_CS,

AD_CONV, SF_CE0 e FPGA_INIT_B não fazem parte da interface interna de controle, porém necessitam ser desabilitados para que o componente funcione de maneira correta. Tanto os pinos da interface de controle quanto os pinos que não são da interface de controle devem ser conectados aos pinos específicos de saída da FPGA.

A.1.5 Conversor Analógico/Digital

O diagrama do componente A/D permite a conversão de um sinal analógico para um sinal digital de 14 *bits*. Conta com dois canais independes para a conversão do sinal analógico, em que cada canal do conversor opera numa faixa de tensão de 0,4 V a 2,9 V. O valor analógico de 0,4 V quando convertido é representado na saída por 16383 e o valor de 2,9 V é representado por 0. A Figura 5 representa o diagrama do componente A/D.

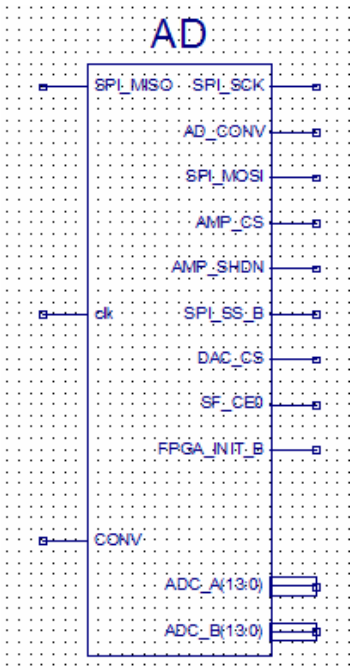

Figura 5 – Componente A/D
Fonte: autoria própria.

A Tabela 5 contém as especificações dos pinos de entrada do componente A/D.

Tabela 5 - Pinos de entrada conversor A/D

Pinos de entrada	Funções
SPI_MISO	Recebe serialmente os dados convertidos
clk	Fornece *Clock* para o componente
CONV	Habilita a conversão do

	sinal. 0: CONV desabilitado 1: CONV habilitado

Fonte: autoria própria.

No componente A/D assim como no componente D/A alguns dos pinos de saída fazem parte da interface interna de controle e outros não. Os pinos de saída que fazem parte da interface interna de controle são SPI_SCK, AD_CONV, SPI_MOSI, AMP_CS, AMP_SHDN e os que não fazem parte, porém necessitam ser desabilitados são SPI_SS_B, DAC_CS, SF_CE0 e FPGA_INIT_B. Para apresentar os sinais convertidos o componente A/D conta com duas saídas ADC_A e ADC_B que representa os sinais digitais convertidos pelos canais A e B do conversor A/D. Para que o componente funcione de maneira correta os pinos de saída devem ser conectados aos pinos específicos de saída da FPGA.

A.2 TUTORIAL UTILIZAÇÃO BIBLIOTECA

Está seção contém os passos necessários para utilização da biblioteca desenvolvida. O conteúdo desta seção é detalhado desde a criação de um projeto em diagrama esquemático até a utilização do *kit* de desenvolvimento.

A.2.1 Projeto em diagrama esquemático

A biblioteca desenvolvida contém os seguintes componentes *encoder*, *display* LCD, conversor D/A, conversor A/D e porta PS/2. Os componentes presentes na biblioteca estão prontos para a utilização em diagrama esquemático. Porém para que se possa utilizar se faz necessário criar um projeto em digrama esquemático.

Para criar um projeto em diagrama esquemático inicialmente o usuário irá abrir o *software ISE Design Suite*. A interface geral do *software ISE Design Suite* é representada pela Figura 10 da Seção 2.3.

Após abrir o *software ISE* será necessário criar um novo projeto, no menu principal clique na aba *File* e depois selecione a opção *New Project*, em seguida a janela *Create New Project* será aberta. A janela *Create New Project* é representada pela Figura 6.

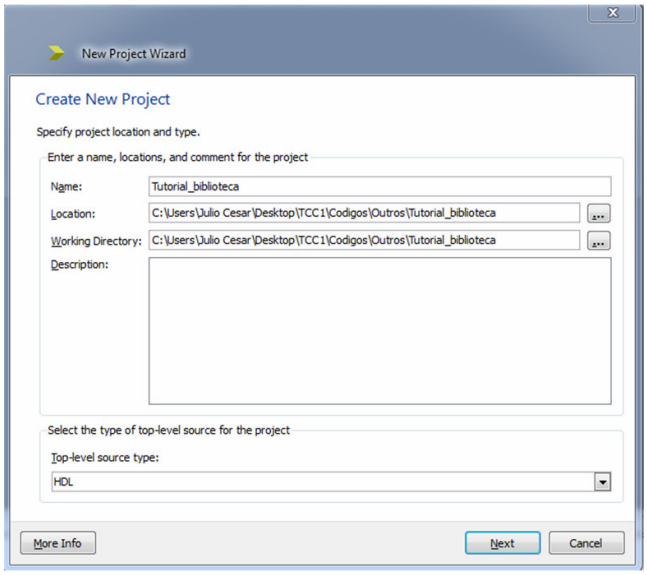

Figura 6 - Janela *Create New Project*
Fonte: autoria própria.

No campo *Name* selecione o nome desejado para o projeto e em seguida no campo *Location* informe o local no computador em que o projeto será salvo. Após especificar o nome e o local em que o arquivo será salvo clique no botão *Next*, em seguida será aberta a janela *Project Settings*. A janela *Project Settings* é representada na Figura 7.

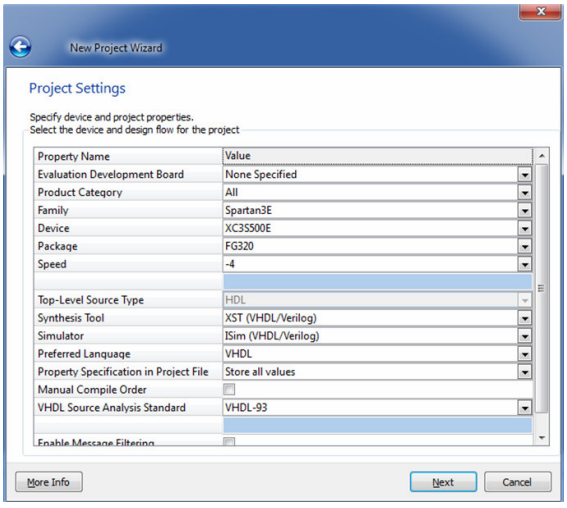

Figura 7 - Janela *Project Settings*
Fonte: autoria própria.

Na janela *Project Settings* alguns parâmetros referentes ao *kit Spartan 3E* devem ser ajustados. No campo *Family* escolher a opção *Spartan 3E*, no campo *Device* XC3S500E e no

campo *Package* FG320. Com os ajustes realizados clique no botão *Next* e em seguida no botão *Finish*.

Com o projeto criado o próximo passo é inserir um arquivo de descrição de circuito. Para inserir um novo arquivo no projeto selecione a aba *Project* e em seguida selecione a opção *New Source*. Selecionada a opção *New Source* a janela *Select Source Type* será aberta. A janela *Select Source Type* é representada pela Figura 8.

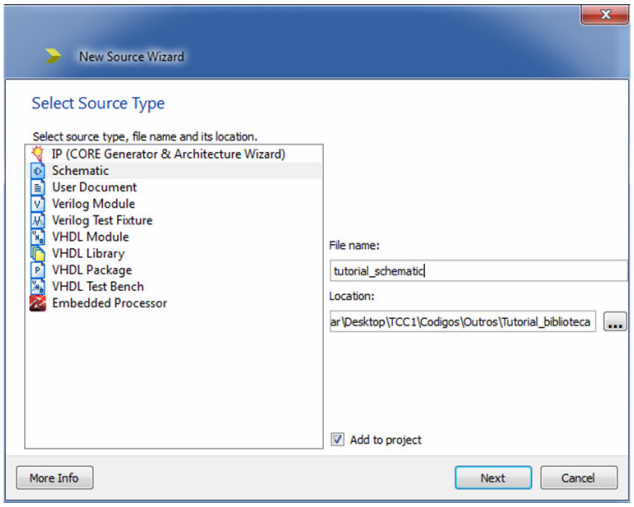

Figura 8 - Janela *Select Source Type*
Fonte: autoria própria.

Na janela *Select Source Type* selecione a opção *Schematic* e no campo *File name* informe o nome do arquivo a ser criado. Em seguida clique no botão Next e no botão *Finish*. Ao término desta etapa o ambiente de desenvolvimento em esquemático é criado. A Figura 9 representa o ambiente de desenvolvimento em esquemático.

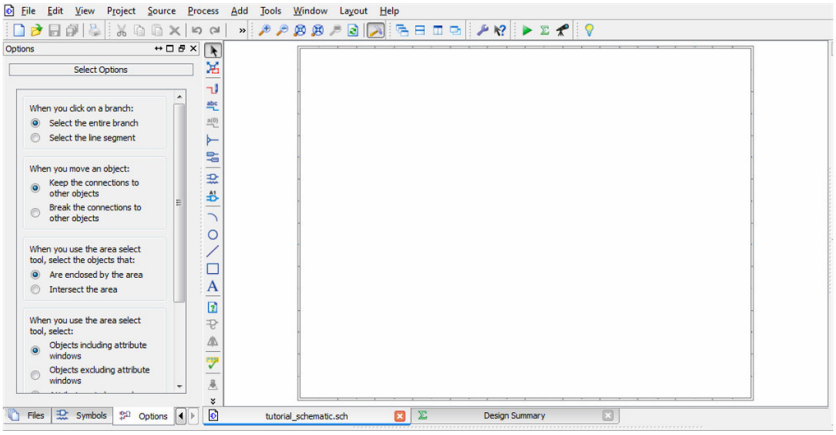

Figura 9 - Ambiente de desenvolvimento em esquemático
Fonte: autoria própria.

A.2.2 Importe a biblioteca *Spartan 3E*

Após criar o projeto em diagrama esquemático para utilizar a biblioteca criada será necessário realizar seu importe. Para importar a biblioteca, selecionar a aba *Libraries*, em seguida clicar com o botão direito do *mouse* e selecionar a opção *New VHDL Library*. A Figura 10 representa a aba *Libraries*.

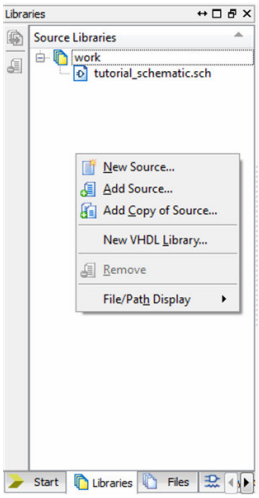

Figura 10 - Aba *Libraries*
Fonte: autoria própria.

Com a opção *New VHDL Library* selecionada será necessário especificar o diretório que se encontra a nova biblioteca a ser inserida. A Figura 11 representa a janela *New VHDL Library*.

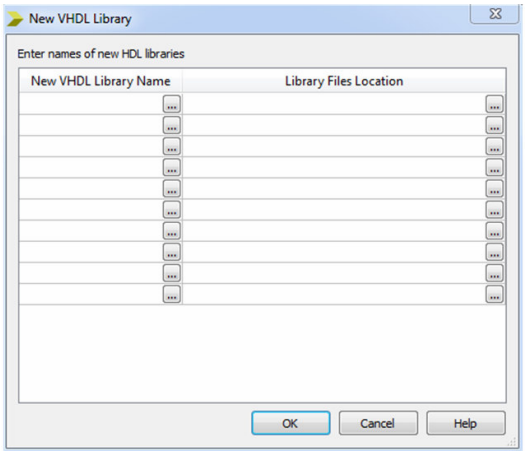

Figura 11 - Janela New VHDL Library
Fonte: autoria própria.

Após selecionar diretório em que se encontra a biblioteca, clique no botão *OK* e aguarde a janela *Adding Source Files* ser aberta. A Figura 12 representa a janela *Adding Source Files*.

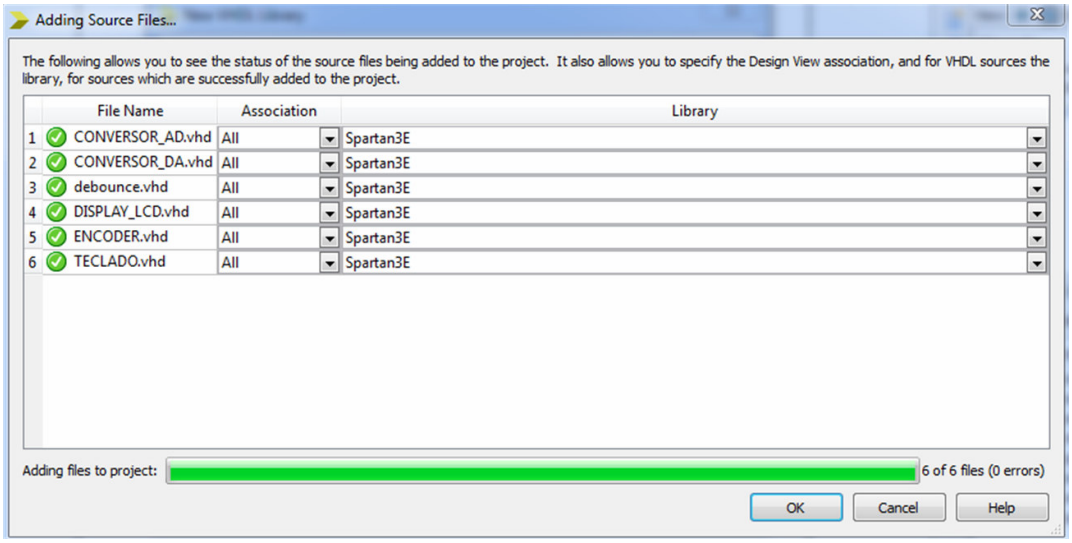

Figura 12 - Janela *Adding Source Files*
Fonte: autoria própria.

Observando a Figura 12 pode-se notar que os componentes que compõem a biblioteca foram adicionados no projeto. Retornando a aba *Libraries* observamos que a biblioteca *Spartan 3E* está incluída no projeto. A Figura 13 representa a aba *Libraries* atualizada.

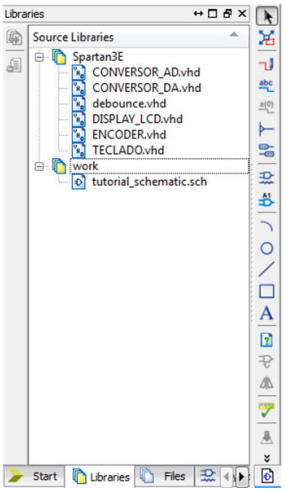

Figura 13 - Aba *Libraries* atualizada
Fonte: autoria própria.

A.2.3 Criando diagrama de componentes

Após importar a biblioteca *Spartan 3E* é necessário criar os diagramas dos componentes da biblioteca. O primeiro passo é selecionar um componente da biblioteca *Spartan 3E* e clicar com o botão direto do *mouse* em cima do componente selecionado. Em seguida, selecione a opção *Move to Library*, está opção irá mover o componente da biblioteca *Spartan 3E* para a biblioteca *work*. Estes passos devem ser feitos para todos os componentes da biblioteca *Spartan 3E*. A Figura 14 representa a operação de deslocamento dos componentes.

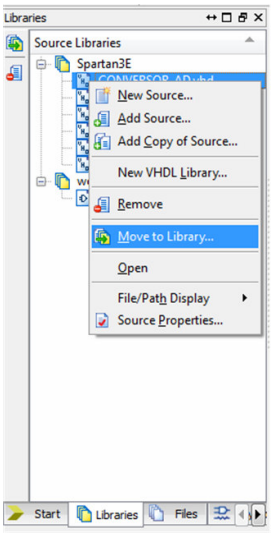

Figura 14 - Operação de deslocamento dos componentes
Fonte: autoria própria.

Após todos os componentes serem deslocados para a biblioteca *work*, selecione na aba *Hierarchy* um componente. Em seguida, na aba *Design Utilities*, dê um duplo clique na opção *Create Schematic Symbol*. A Figura 15 representa a aba *Design Utilities*.

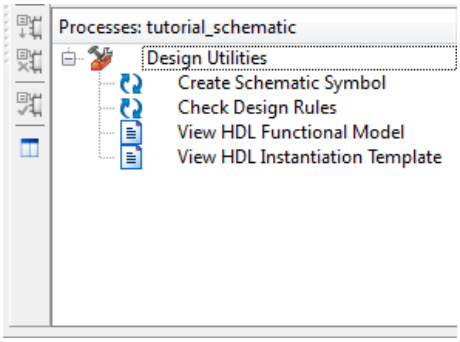

Figura 15 - Aba *Design Utilities*
Fonte: autoria própria.

Após o duplo clique na opção *Create Schematic Symbol* o diagrama do componente será criado. Esta operação deve ser feita para todos os componentes da biblioteca. Ao término desta operação selecionando a aba *Symbols* pode se observar os componentes criados na etapa anterior. A Figura 16 representa a aba *Symbols*.

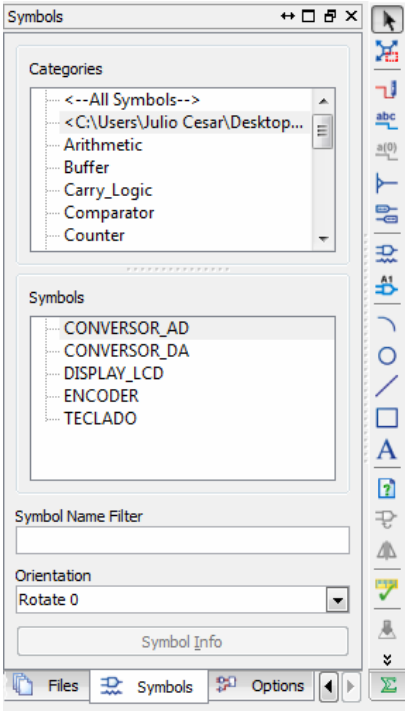

Figura 16 - Aba *Symbols*
Fonte: autoria própria.

A.2.4 Utilizando diagrama de componentes

Na aba *Symbols* selecione o componente desejado e coloque o mesmo na área de desenho. A Figura 17 representa a área de desenho.

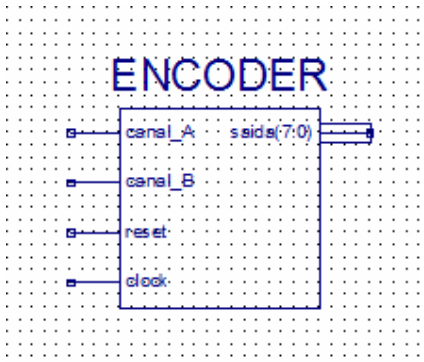

Figura 17 - Área de desenho
Fonte: autoria própria.

Com o componente posicionado na área de desenho, o próximo para a utilização é adicionar as conexões por fios entre as entradas e saída do componente selecionado. Para adicionar um fio selecione a aba *Add* e clique na opção *Wire*. Com a opção *Wire* selecionada mova o *mouse* até a marca existente nas pontas das entradas e saída do componente e clique com botão esquerdo do *mouse* para iniciar a colocação de uma conexão. Mova o *mouse* até o ponto de conexão desejado e clique novamente com o botão esquerdo do *mouse*. A Figura 18 representa o componente com os fios conectados.

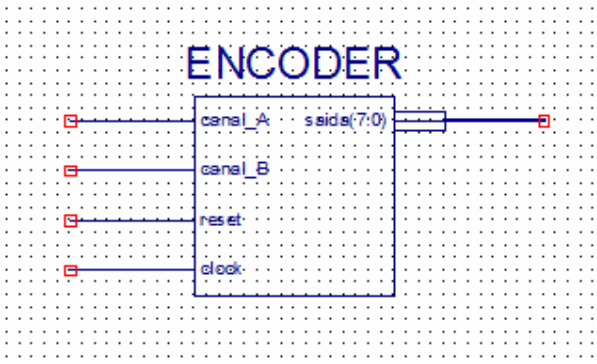

Figura 18 - Componente com os fios conectados
Fonte: autoria própria.

Após inserir os fios é necessário realizar a ligação dos pontos de entrada e saída do circuito. Os pontos de entrada e saída identificam os sinais necessários para que o circuito funcione corretamente. Para inserir esses pontos na aba *Add* selecione a opção *I/O Marker* e clique com o botão esquerdo *mouse* nas portas de entrada e saída do circuito. A Figura 19 representa o circuito com pinos de entrada e saída definidos.

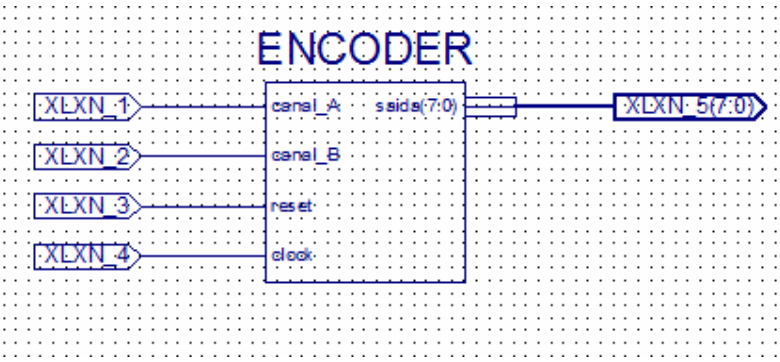

Figura 19 - Circuito com pinos de entrada e saída definidos
Fonte: autoria própria.

Os pontos de entrada e saída são retângulos pontudos identificados por códigos como XLXN_1. Esses códigos são atribuídos automaticamente pelo ISE e podem ser renomeados, facilitando assim a identificação da porta. Para renomear a porta clique com o botão direito do *mouse* na porta desejada e selecione a opção *Rename Port*. A Figura 20 representa a operação para renomear a porta.

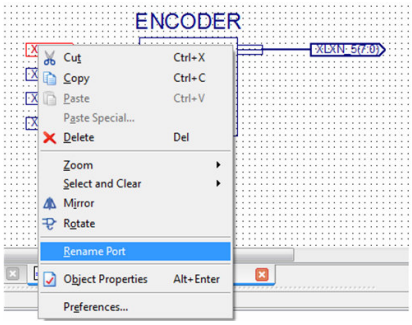

Figura 20 - Operação para renomear a porta
Fonte: autoria própria.

Após selecionar a opção *Rename Port* a janela *Rename Net* será aberta. No campo em que se encontra o nome atual do componente apague e insira o novo nome, e clique no botão *OK*. A Figura 21 representa a janela *Rename Net*.

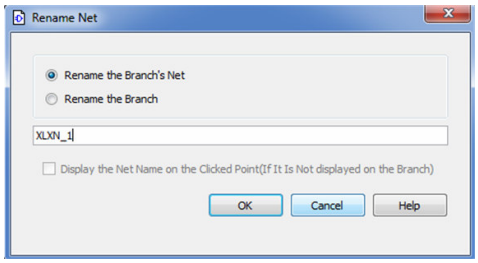

Figura 21 - Janela *Rename Net*
Fonte: autoria própria.

Terminada a etapa de renomeação das portas de entrada e saída, o componente apresenta a seguinte configuração. A Figura 22 representa o componente com as portas renomeadas.

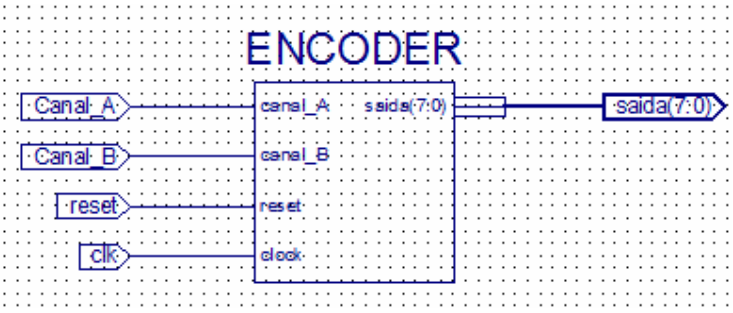

Figura 22 - Componente com as portas renomeadas
Fonte: autoria própria.

Com as portas renomeados o próximo passo é criar o arquivo o arquivo de restrições de projeto UCF. Na aba *Project* selecione a opção *New Source*, em seguida selecione o tipo de arquivo *Implementation Constraints File* na janela *Select Source Type*. A Figura 23 representa a seleção do arquivo *Implementation Constraints File*.

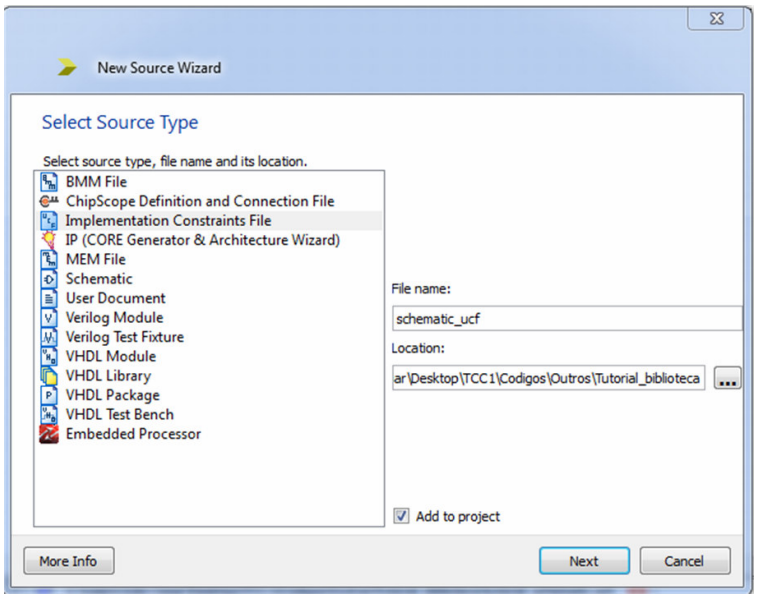

Figura 23 - Seleção do arquivo *Implementation Constraints File*
Fonte: autoria própria.

No campo *File name* informe o nome do arquivo de restrições de projeto e clique no botão *Next*. Após criar o arquivo UCF é necessário editar o mesmo para que os respectivos pinos de entrada e saída posam ser acessados. A Figura 24 representa o arquivo UCF editado.

```
1  NET "clk" LOC = "C9" | IOSTANDARD = LVCMOS33;
2  NET "CanalA" LOC = "K18" | IOSTANDARD = LVTTL | PULLUP;
3  NET "CanalB" LOC = "G18" | IOSTANDARD = LVTTL | PULLUP;
4  NET "rst" LOC = "V16" | IOSTANDARD = LVTTL | PULLDOWN;
5
6  NET "saida[7]" LOC = "F9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
7  NET "saida[6]" LOC = "E9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
8  NET "saida[5]" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
9  NET "saida[4]" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
10 NET "saida[3]" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
11 NET "saida[2]" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
12 NET "saida[1]" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
13 NET "saida[0]" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
```

Figura 24 - Arquivo UCF editado
Fonte: autoria própria.

Concluído a edição do arquivo UCF, o próximo passo a ser realizado é a síntese do projeto. Nesta etapa o ISE processa todos os arquivos do projeto e cria um arquivo com as informações sobre as ligações que devem ser feitas dentro da FPGA, garantindo assim que mesma se comporte conforme foi projetado. Para iniciar o processo de síntese selecione na aba *Design* o arquivo do módulo principal do projeto, neste exemplo o tutorial_schematic.sch. Com o arquivo principal selecionado, no campo *Process* de um duplo clique em *Generate Programming File*. Após algum tempo o ISE irá indicar que os processos *Synthesize – XST*, *Implement Design* e *Generate Programming File* foram completados com sucesso, mostrando três ícones verdes ao lado dos processos. A Figura 25 representa os processos completados com sucesso.

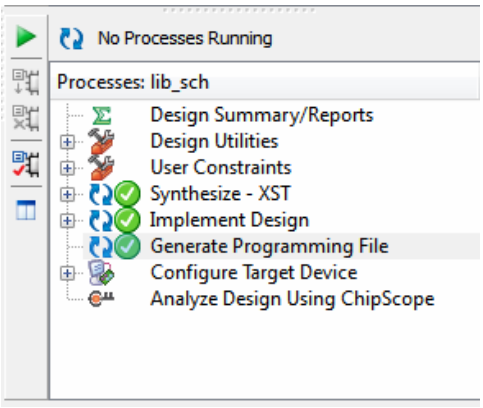

Figura 25 - Processos completados
Fonte: autoria própria.

Com os processos completados a próxima etapa é a de transferência do arquivo de restrições de projeto para a FPGA *Spartan 3E Starter Kit Board*. Na aba *Tools* selecione a

opção *Impact*, ao selecionar essa opção uma mensagem será exibida clique no botão *OK* e aguarde que o *software* seja aberto. A Figura 26 representa o *software Impact*.

Figura 26 - *Software Impact*
Fonte: autoria própria.

Com o *software Impact* iniciado de um duplo clique na opção *Boundary Scan* localizado na aba *iMPACT Flows*. Em seguida clique na aba *File* e selecione a opção *Initialize Chain*, está opção irá localizar e iniciar o *kit* conectado ao computador. Após localizar o *kit* a janela *Assign New Configuration File* será aberta. A Figura 27 representa a janela *Assign New Configuration File*.

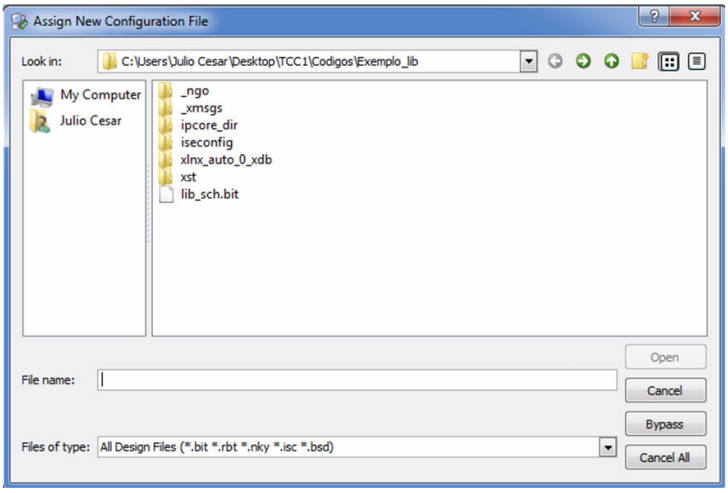

Figura 27 - Janela *Assign New Configuration File*
Fonte: autoria própria.

Na janela *Assign New Configuration File* localize a pasta do projeto e selecione o arquivo *.bit* associado ao projeto, no caso deste exemplo o *lib_sch.bit*. Com o arquivo selecionado clique no botão *Open* e aguarde o arquivo ser carregado. Após o arquivo ser carregado a janela *Device Programming Properties* será aberta, na aba *Bondary Scan* selecione a opção Device 1 (FPGA XC3S500E) e clique no botão *OK*. Ao clicar no botão *OK* estaremos selecionando o dispositivo que irá receber o arquivo de restrições do usuário. A Figura 28 representa a janela *Device Programming Properties*.

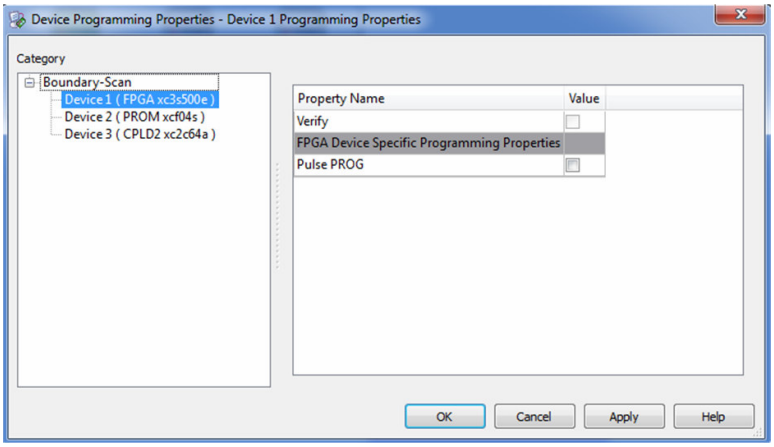

Figura 28 - Janela Device Programming Properties
Fonte: autoria própria.

Com o dispositivo XC3S500E selecionado vá até a aba *iMPACT Process* e de um duplo clique na opção *Program*, ao realizar está operação o dispositivo será programado. A Figura 29 representa o dispositivo programado.

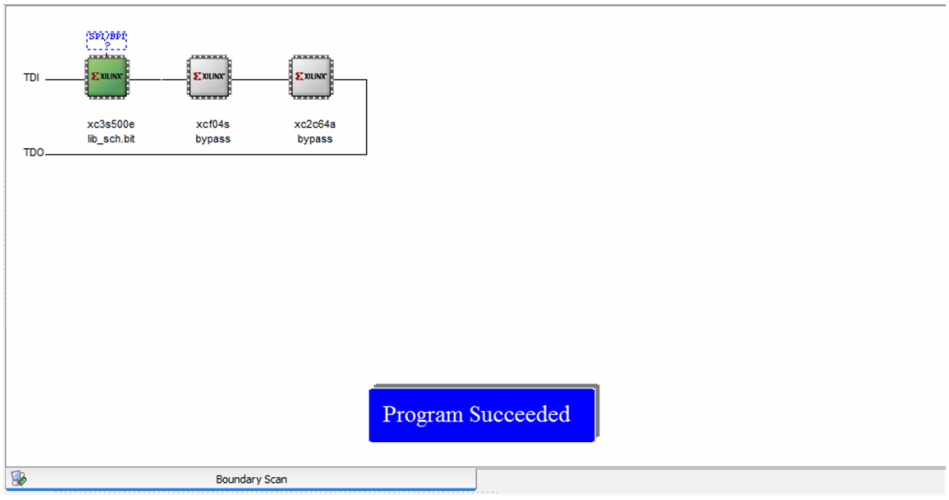

Figura 29 - Dispositivo programado
Fonte: autoria própria.

Com o *kit* programado os testes com o *encoder* podem ser realizados. Como descrito na Seção A.1 o *encoder* neste trabalho foi modelado para se comportar como um indicador de posição em que cada posição absoluta do *encoder* é representada por um valor numérico. Os valores numéricos neste exemplo serão representados nos *leds* presentes no *kit*. O *encoder* na Figura 30 se encontra na posição cinco e a saída é presente nos é *leds* 00000101_2. A Figura 30 representa o *encoder* em funcionamento no *kit*.

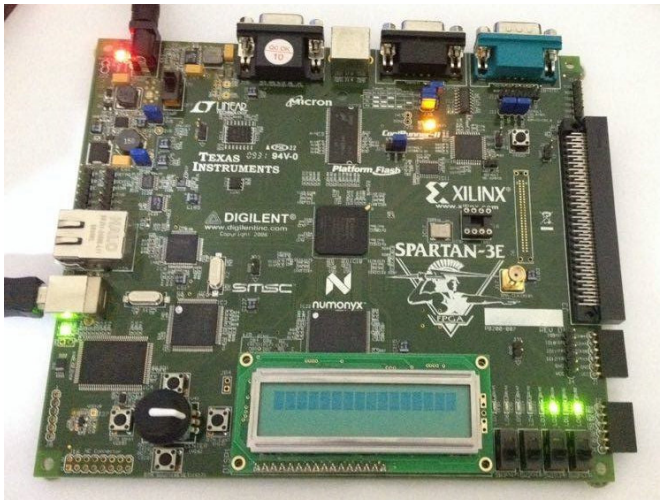

Figura 30 - Encoder em funcionamento
Fonte: autoria própria.

A.3 ARQUIVOS UCF

Nesta Seção estão contidos os arquivos UCF dos componentes que fazem parte da biblioteca desenvolvida. O arquivo UCF é utilizado para que se possa especificar quais pinos específicos da FPGA serão conectados com o componente que será utilizado. O arquivo UCF é no formato texto, em que cada linha iniciada com a palavra "NET" representa um pino da FPGA. Dos parâmetros contidos no arquivo UCF somente um necessita ser editado, a palavra que se encontra na frente da palavra "NET" representa o nome do pino do componente que está sendo utilizado, logo este nome deve ser o mesmo para que a associação entre o pino da FPGA e o pino componente possa ser feita.

Para a utilização das chaves deslizantes, *push-buttons* e *leds* deve-se utilizar o arquivo UCF representado na Figura 31.

```
#CHAVES DESLIZANTES
NET "SW<0>" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP ;
NET "SW<1>" LOC = "L14" | IOSTANDARD = LVTTL | PULLUP ;
NET "SW<2>" LOC = "H18" | IOSTANDARD = LVTTL | PULLUP ;
NET "SW<3>" LOC = "N17" | IOSTANDARD = LVTTL | PULLUP ;

#PUSH-BUTTONS
NET "BTN_EAST" LOC = "H13" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "BTN_NORTH" LOC = "V4" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "BTN_SOUTH" LOC = "K17" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "BTN_WEST" LOC = "D18" | IOSTANDARD = LVTTL | PULLDOWN ;

#LEDS
NET "LED<7>" LOC = "F9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<6>" LOC = "E9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<5>" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<4>" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
```

Figura 31 - Arquivo UCF chaves deslizantes, *push-buttons* e *leds*
Fonte: autoria própria.

Para a utilização dos componentes *encoder*, display LCD, porta PS/2, conversor D/A e conversor A/D deve-se utilizar o arquivo UCF representado na Figura 32.

```
#ENCODER
NET "ROT_A" LOC = "K18" | IOSTANDARD = LVTTL | PULLUP ;
NET "ROT_B" LOC = "G18" | IOSTANDARD = LVTTL | PULLUP ;
NET "ROT_CENTER" LOC = "V16" | IOSTANDARD = LVTTL | PULLDOWN ;

#DISPLAY LCD
NET "LCD_E" LOC = "M18" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "LCD_RS" LOC = "L18" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "LCD_RW" LOC = "L17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "SF_D<8>" LOC = "R15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "SF_D<9>" LOC = "R16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "SF_D<10>" LOC = "P17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "SF_D<11>" LOC = "M15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;

#PORTA PS/2
NET "PS2_CLK" LOC = "G14" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = SLOW ;
NET "PS2_DATA" LOC = "G13" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = SLOW ;

#CONVERSOR D/A
NET "SPI_MISO" LOC = "N10" | IOSTANDARD = LVCMOS33 ;
NET "SPI_MOSI" LOC = "T4" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
NET "SPI_SCK" LOC = "U16" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
NET "DAC_CS" LOC = "N8" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
NET "DAC_CLR" LOC = "P8" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;

#CONVERSOR A/D
NET "SPI_MOSI" LOC = "T4" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 6 ;
NET "AMP_CS" LOC = "N7" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 6 ;
NET "SPI_SCK" LOC = "U16" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
NET "AMP_SHDN" LOC = "P7" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 6 ;
NET "AMP_DOUT" LOC = "E18" | IOSTANDARD = LVCMOS33 ;
```

Figura 32 - Arquivo UCF componentes biblioteca
Fonte: autoria própria.

A.4 CÓDIGOS COMPONENTES DESENVOLVIDOS

A.4.1 *Encoder* rotativo

```
----------------------------------------------------------------------------------
--
-- Aluno: Julio Cesar de Paiva Ribeiro
-- Professor: Giovanni Alfredo Guarneri
-- Create Date: 17:48:20 06/05/2017
-- Module Name:    ENCODER - Behavioral
----------------------------------------------------------------------------------
--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ENCODER is
port( saida: out std_logic_vector(7 downto 0);
        canal_A: in std_logic;
        canal_B: in std_logic;
        reset: in std_logic;
        clock: in std_logic);
end ENCODER;

architecture Behavioral of ENCODER is
signal canal_A_in: std_logic;
signal canal_B_in: std_logic;
signal reset_in: std_logic;
signal vet_AB_in: std_logic_vector(1 downto 0);
signal rot_A: std_logic;
signal rot_B: std_logic;
signal delay_rot_A: std_logic;
signal rot_event: std_logic;
signal rot_esquerda: std_logic;

signal cont : integer range 0 to 19:= 0;

function decodificador(signal sel: integer range 0 to 19) return
std_logic_vector is

begin

    if(sel=0)then
        return "00000000";

    elsif(sel=1)then
        return "10000000";

    elsif(sel=2)then
        return "01000000";

    elsif(sel=3)then
        return "11000000";

    elsif(sel=4)then
        return "00100000";

    elsif(sel=5)then
        return "10100000";

    elsif(sel=6)then
        return "01100000";

    elsif(sel=7)then
        return "11100000";

    elsif(sel=8)then
        return "00010000";

    elsif(sel=9)then
        return "10010000";

    elsif(sel=10)then
        return "01010000";

    elsif(sel=11)then
```

```
69          return "11010000";
70
71      elsif(sel=12)then
72          return "00110000";
73
74      elsif(sel=13)then
75          return "10110000";
76
77      elsif(sel=14)then
78          return "01110000";
79
80      elsif(sel=15)then
81          return "11110000";
82
83      elsif(sel=16)then
84          return "00001000";
85
86      elsif(sel=17)then
87          return "10001000";
88
89      elsif(sel=18)then
90          return "01001000";
91
92      else
93          return "11001000";
94
95      end if;
96
97  end function;
98
99  begin
100
101 processo_1: process(clock)
102     begin
103         if(clock'event and clock='1')then
104             canal_A_in <= canal_A;
105             canal_B_in <= canal_B;
106             reset_in <= reset;
107             vet_AB_in <= canal_A_in & canal_B_in;
108
109             case vet_AB_in is
110
111                 when "00" => rot_A <= '0';
112                              rot_B <= rot_B;
113
114                 when "01" => rot_A <= rot_A;
115                              rot_B <= '0';
116
117                 when "10" => rot_A <= rot_A;
118                              rot_B <= '1';
119
120                 when "11" => rot_A <= '1';
121                              rot_B <= rot_B;
122
123                 when others => rot_A <= rot_A;
124                                rot_B <= rot_B;
125
126             end case;
127
128         end if;
129     end process processo_1;
130
131     processo_2: process(clock)
132     begin
133         if(clock'event and clock='1')then
134
135             delay_rot_A <= rot_A;
136
137             if(rot_A='1' and delay_rot_A = '0')then
138                 rot_event <= '1';
139                 rot_esquerda <= rot_B;
```

```
140
141             else
142                 rot_event <= '0';
143                 rot_esquerda <= rot_esquerda;
144
145             end if;
146
147         end if;
148
149     end process processo_2;
150
151     processo_3: process(clock)
152         variable cont_aux: integer range 0 to 19 := 0;
153     begin
154         if(clock'event and clock='1')then
155             if(reset_in='1')then
156                 cont_aux:=0;
157             elsif(rot_event='1')then
158                 if(rot_esquerda='1')then
159                     if(cont_aux = 0)then
160                         cont_aux:= 19;
161                     else
162                         cont_aux:= cont_aux-1;
163                     end if;
164                 else
165                     if(cont_aux=19)then
166                         cont_aux:= 0;
167                     else
168                         cont_aux:= cont_aux+1;
169                     end if;
170                 end if;
171             end if;
172
173         end if;
174
175         cont <= cont_aux;
176
177     end process processo_3;
178
179     saida <= decodificador(cont);
180 end Behavioral;
```

A.4.2 *Display* LCD

```
1  ----------------------------------------------------------------------------
   --
2  -- Aluno: Julio Cesar de Paiva Ribeiro
3  -- Professor: Giovanni Alfredo Guarneri
4  -- Create Date: 17:48:20 06/05/2017
5  -- Module Name: DISPLAY_LCD - Behavioral
6  ----------------------------------------------------------------------------
   --
7  library IEEE;
8  use IEEE.STD_LOGIC_1164.ALL;
9
10 entity DISPLAY_LCD is
11 port(clk:in std_logic;
12       rst:in std_logic;
13       flag: in std_logic;
14       CE: in std_logic;
15       ADDRESS: in std_logic_vector(7 downto 0);
16       DATA: in std_logic_vector(7 downto 0);
17       RS:out std_logic;
18       RW:out std_logic;
19       E:out std_logic;
20       DB:out std_logic_vector (7 downto 4));
21 end DISPLAY_LCD;
22
23 architecture Behavioral of DISPLAY_LCD is
24 type maquina is(ESPERA,SET1,SET2,SET3,SET4,SET5,SET6,SET7,SET8,FSET_UPPER,
   ESPERA1,FSET_LOWER,
25 ESPERA2,MODE_UPPER,ESPERA3,MODE_LOWER,ESPERA4,DISP_UPPER,ESPERA5,DISP_LOWER,
   ESPERA6,
26 CLEAR_UPPER,ESPERA7,CLEAR_LOWER,ESPERA8,SET_DDRAM_UPPER,ESPERA9,SET_DDRAM_LOWER
   ,ESPERA10,
27 WRT_UPPER,ESPERA11,WRT_LOWER,ESPERA12,FIM);
28
29 signal nx_state: maquina:=ESPERA;
30 begin
31
32     RW<='0';
33
34 process(clk,rst)
35     variable cont: integer range 0 to 50000000:=0;
36     variable cont2: integer range 0 to 50000000:=0;
37     variable cont3: integer range 0 to 50000000:=0;
38 begin
39     if(rst='1')then
40         nx_state<=ESPERA;
41         cont:=0;
42         cont2:=0;
43         cont3:=0;
44     elsif(clk'EVENT and clk='1')then
45         case nx_state is
46             when ESPERA =>
47                 if(cont = 2)then
48                     nx_state<=SET1;
49                     cont:=0;
50                 else
51                     RS<='0';
52                     E<='0';
53                     DB<="0000";
54                     nx_state<=ESPERA;
55                     cont:= cont + 1;
56                 end if;
57
58             when SET1 =>
59                 if(cont = 4)then
60                     if(cont2 = 13)then
61                         if(cont3 = 4)then
62                             nx_state<=SET2;
63                             cont:=0;
64                             cont2:=0;
65                             cont3:=0;
66                         else
```

```
                            E<='0';
                            nx_state<=SET1;
                            cont3:= cont3 + 1;
                        end if;
                    else
                        E<='1';
                        nx_state<=SET1;
                        cont2:= cont2 + 1;
                    end if;
                else
                    RS<='0';
                    E<='0';
                    DB<="0011";
                    nx_state<=SET1;
                    cont:= cont + 1;
                end if;

            when SET2 =>
                if(cont = 205000)then
                    nx_state<=SET3;
                    cont:=0;
                else
                    DB<="0000";
                    nx_state<=SET2;
                    cont:= cont + 1;
                end if;

            when SET3 =>
                if(cont = 4)then
                    if(cont2 = 13)then
                        if(cont3 = 4)then
                            nx_state<=SET4;
                            cont:=0;
                            cont2:=0;
                            cont3:=0;
                        else
                            E<='0';
                            nx_state<=SET3;
                            cont3:= cont3 + 1;
                        end if;
                    else
                        E<='1';
                        nx_state<=SET3;
                        cont2:= cont2 + 1;
                    end if;
                else
                    RS<='0';
                    E<='0';
                    DB<="0011";
                    nx_state<=SET3;
                    cont:= cont + 1;
                end if;

            when SET4 =>
                if(cont = 5000)then
                    nx_state<=SET5;
                    cont:=0;
                else
                    DB<="0000";
                    nx_state<=SET4;
                    cont:= cont + 1;
                end if;

            when SET5 =>
                if(cont = 4)then
                    if(cont2 = 13)then
                        if(cont3 = 4)then
                            nx_state<=SET6;
                            cont:=0;
                            cont2:=0;
                            cont3:=0;
```

```
                        else
                            E<='0';
                            nx_state<=SET5;
                            cont3:= cont3 + 1;
                        end if;
                    else
                        E<='1';
                        nx_state<=SET5;
                        cont2:= cont2 + 1;
                    end if;
                else
                    RS<='0';
                    E<='0';
                    DB<="0011";
                    nx_state<=SET5;
                    cont:= cont + 1;
                end if;

            when SET6 =>
                if(cont = 2000)then
                    nx_state<=SET7;
                    cont:=0;
                else
                    DB<="0000";
                    nx_state<=SET6;
                    cont:= cont + 1;
                end if;

            when SET7 =>
                if(cont = 4)then
                    if(cont2 = 13)then
                        if(cont3 = 4)then
                            nx_state<=SET8;
                            cont:=0;
                            cont2:=0;
                            cont3:=0;
                        else
                            E<='0';
                            nx_state<=SET7;
                            cont3:= cont3 + 1;
                        end if;
                    else
                        E<='1';
                        nx_state<=SET7;
                        cont2:= cont2 + 1;
                    end if;
                else
                    RS<='0';
                    E<='0';
                    DB<="0010";
                    nx_state<=SET7;
                    cont:= cont + 1;
                end if;

            when SET8 =>
                if(cont = 2000)then
                    nx_state<=FSET_UPPER;
                    cont:=0;
                else
                    DB<="0000";
                    nx_state<=SET8;
                    cont:= cont + 1;
                end if;

            when FSET_UPPER =>
                if(cont = 4)then
                    if(cont2 = 13)then
                        if(cont3 = 4)then
                            nx_state<=ESPERA1;
                            cont:=0;
                            cont2:=0;
```

```
209                             cont3:=0;
210                         else
211                             E<='0';
212                             nx_state<=FSET_UPPER;
213                             cont3:= cont3 + 1;
214                         end if;
215                     else
216                         E<='1';
217                         nx_state<=FSET_UPPER;
218                         cont2:= cont2 + 1;
219                     end if;
220                 else
221                     RS<='0';
222                     E<='0';
223                     DB<="0010";
224                     nx_state<=FSET_UPPER;
225                     cont:= cont + 1;
226                 end if;
227 
228             when ESPERA1 =>
229                 if(cont = 52)then
230                     nx_state<=FSET_LOWER;
231                     cont:=0;
232                 else
233                     DB<="0000";
234                     nx_state<=ESPERA1;
235                     cont:= cont + 1;
236                 end if;
237 
238             when FSET_LOWER =>
239                 if(cont = 4)then
240                     if(cont2 = 13)then
241                         if(cont3 = 4)then
242                             nx_state<=ESPERA2;
243                             cont:=0;
244                             cont2:=0;
245                             cont3:=0;
246                         else
247                             E<='0';
248                             nx_state<=FSET_LOWER;
249                             cont3:= cont3 + 1;
250                         end if;
251                     else
252                         E<='1';
253                         nx_state<=FSET_LOWER;
254                         cont2:= cont2 + 1;
255                     end if;
256                 else
257                     RS<='0';
258                     E<='0';
259                     DB<="1000";
260                     nx_state<=FSET_LOWER;
261                     cont:= cont + 1;
262                 end if;
263 
264             when ESPERA2 =>
265                 if(cont = 2000)then
266                     nx_state<=MODE_UPPER;
267                     cont:=0;
268                 else
269                     DB<="0000";
270                     nx_state<=ESPERA2;
271                     cont:= cont + 1;
272                 end if;
273 
274             when MODE_UPPER =>
275                 if(cont = 4)then
276                     if(cont2 = 13)then
277                         if(cont3 = 4)then
278                             nx_state<=ESPERA3;
279                             cont:=0;
```

```
                            cont2:=0;
                            cont3:=0;
                        else
                            E<='0';
                            nx_state<=MODE_UPPER;
                            cont3:= cont3 + 1;
                        end if;
                    else
                        E<='1';
                        nx_state<=MODE_UPPER;
                        cont2:= cont2 + 1;
                    end if;
                else
                    RS<='0';
                    E<='0';
                    DB<="0000";
                    nx_state<=MODE_UPPER;
                    cont:= cont + 1;
                end if;

            when ESPERA3 =>
                if(cont = 52)then
                    nx_state<=MODE_LOWER;
                    cont:=0;
                else
                    DB<="0000";
                    nx_state<=ESPERA3;
                    cont:= cont + 1;
                end if;

            when MODE_LOWER =>
                if(cont = 4)then
                    if(cont2 = 13)then
                        if(cont3 = 4)then
                            nx_state<=ESPERA4;
                            cont:=0;
                            cont2:=0;
                            cont3:=0;
                        else
                            E<='0';
                            nx_state<=MODE_LOWER;
                            cont3:= cont3 + 1;
                        end if;
                    else
                        E<='1';
                        nx_state<=MODE_LOWER;
                        cont2:= cont2 + 1;
                    end if;
                else
                    RS<='0';
                    E<='0';
                    DB<="0110";
                    nx_state<=MODE_LOWER;
                    cont:= cont + 1;
                end if;

            when ESPERA4 =>
                if(cont = 2000)then
                    nx_state<=DISP_UPPER;
                    cont:=0;
                else
                    DB<="0000";
                    nx_state<=ESPERA4;
                    cont:= cont + 1;
                end if;

            when DISP_UPPER =>
                if(cont = 4)then
                    if(cont2 = 13)then
                        if(cont3 = 4)then
                            nx_state<=ESPERA5;
```

```
                            cont:=0;
                            cont2:=0;
                            cont3:=0;
                        else
                            E<='0';
                            nx_state<=DISP_UPPER;
                            cont3:= cont3 + 1;
                        end if;
                    else
                        E<='1';
                        nx_state<=DISP_UPPER;
                        cont2:= cont2 + 1;
                    end if;
                else
                    RS<='0';
                    E<='0';
                    DB<="0000";
                    nx_state<=DISP_UPPER;
                    cont:= cont + 1;
                end if;

            when ESPERA5 =>
                if(cont = 52)then
                    nx_state<=DISP_LOWER;
                    cont:=0;
                else
                    DB<="0000";
                    nx_state<=ESPERA5;
                    cont:= cont + 1;
                end if;

            when DISP_LOWER =>
                if(cont = 4)then
                    if(cont2 = 13)then
                        if(cont3 = 4)then
                            nx_state<=ESPERA6;
                            cont:=0;
                            cont2:=0;
                            cont3:=0;
                        else
                            E<='0';
                            nx_state<=DISP_LOWER;
                            cont3:= cont3 + 1;
                        end if;
                    else
                        E<='1';
                        nx_state<=DISP_LOWER;
                        cont2:= cont2 + 1;
                    end if;
                else
                    RS<='0';
                    E<='0';
                    DB<="1100";
                    nx_state<=DISP_LOWER;
                    cont:= cont + 1;
                end if;

            when ESPERA6 =>
                if(cont = 2000)then
                    nx_state<=CLEAR_UPPER;
                    cont:=0;
                else
                    DB<="0000";
                    nx_state<=ESPERA6;
                    cont:= cont + 1;
                end if;

            when CLEAR_UPPER =>
                if(cont = 4)then
                    if(cont2 = 13)then
                        if(cont3 = 4)then
```

```
422                             nx_state<=ESPERA7;
423                             cont:=0;
424                             cont2:=0;
425                             cont3:=0;
426                         else
427                             E<='0';
428                             nx_state<=CLEAR_UPPER;
429                             cont3:= cont3 + 1;
430                         end if;
431                     else
432                         E<='1';
433                         nx_state<=CLEAR_UPPER;
434                         cont2:= cont2 + 1;
435                     end if;
436                 else
437                     RS<='0';
438                     E<='0';
439                     DB<="0000";
440                     nx_state<=CLEAR_UPPER;
441                     cont:= cont + 1;
442                 end if;
443 
444             when ESPERA7 =>
445                 if(cont = 52)then
446                     nx_state<=CLEAR_LOWER;
447                     cont:=0;
448                 else
449                     DB<="0000";
450                     nx_state<=ESPERA7;
451                     cont:= cont + 1;
452                 end if;
453 
454             when CLEAR_LOWER =>
455                 if(cont = 4)then
456                     if(cont2 = 13)then
457                         if(cont3 = 4)then
458                             nx_state<=ESPERA8;
459                             cont:=0;
460                             cont2:=0;
461                             cont3:=0;
462                         else
463                             E<='0';
464                             nx_state<=CLEAR_LOWER;
465                             cont3:= cont3 + 1;
466                         end if;
467                     else
468                         E<='1';
469                         nx_state<=CLEAR_LOWER;
470                         cont2:= cont2 + 1;
471                     end if;
472                 else
473                     RS<='0';
474                     E<='0';
475                     DB<="0001";
476                     nx_state<=CLEAR_LOWER;
477                     cont:= cont + 1;
478                 end if;
479 
480             when ESPERA8 =>
481                 if(cont = 82000)then
482                     nx_state<=SET_DDRAM_UPPER;
483                     cont:=0;
484                 else
485                     DB<="0000";
486                     nx_state<=ESPERA8;
487                     cont:= cont + 1;
488                 end if;
489 
490             when SET_DDRAM_UPPER =>
491                 if(cont = 4)then
492                     if(cont2 = 13)then
```

```
493                         if(cont3 = 4)then
494                             nx_state<=ESPERA9;
495                             cont:=0;
496                             cont2:=0;
497                             cont3:=0;
498                         else
499                             E<='0';
500                             nx_state<=SET_DDRAM_UPPER;
501                             cont3:= cont3 + 1;
502                         end if;
503                     else
504                         E<='1';
505                         nx_state<=SET_DDRAM_UPPER;
506                         cont2:= cont2 + 1;
507                     end if;
508                 elsif (CE = '1')then
509                     RS<='0';
510                     E<='0';
511                     DB<=ADDRESS(7 downto 4);
512                     --DB<="1000";
513                     nx_state<=SET_DDRAM_UPPER;
514                     cont:= cont + 1;
515                 else
516                     nx_state<=SET_DDRAM_UPPER;
517                 end if;
518 
519             when ESPERA9 =>
520                 if(cont = 52)then
521                     nx_state<=SET_DDRAM_LOWER;
522                     cont:=0;
523                 else
524                     DB<="0000";
525                     nx_state<=ESPERA9;
526                     cont:= cont + 1;
527                 end if;
528 
529             when SET_DDRAM_LOWER =>
530                 if(cont = 4)then
531                     if(cont2 = 13)then
532                         if(cont3 = 4)then
533                             nx_state<=ESPERA10;
534                             cont:=0;
535                             cont2:=0;
536                             cont3:=0;
537                         else
538                             E<='0';
539                             nx_state<=SET_DDRAM_LOWER;
540                             cont3:= cont3 + 1;
541                         end if;
542                     else
543                         E<='1';
544                         nx_state<=SET_DDRAM_LOWER;
545                         cont2:= cont2 + 1;
546                     end if;
547                 else
548                     RS<='0';
549                     E<='0';
550                     DB<=ADDRESS(3 downto 0);
551                     --DB<="0011";
552                     nx_state<=SET_DDRAM_LOWER;
553                     cont:= cont + 1;
554                 end if;
555 
556             when ESPERA10 =>
557                 if(cont = 2000)then
558                     nx_state<=WRT_UPPER;
559                     cont:=0;
560                 else
561                     DB<="0000";
562                     nx_state<=ESPERA10;
563                     cont:= cont + 1;
```

```
564                 end if;
565
566             when WRT_UPPER =>
567                 if(cont = 4)then
568                     if(cont2 = 13)then
569                         if(cont3 = 4)then
570                             nx_state<=ESPERA11;
571                             cont:=0;
572                             cont2:=0;
573                             cont3:=0;
574                         else
575                             E<='0';
576                             nx_state<=WRT_UPPER;
577                             cont3:= cont3 + 1;
578                         end if;
579                     else
580                         E<='1';
581                         nx_state<=WRT_UPPER;
582                         cont2:= cont2 + 1;
583                     end if;
584                 else
585                     RS<='1';
586                     E<='0';
587                     DB<=DATA(7 downto 4);
588                     --DB<="0101";
589                     nx_state<=WRT_UPPER;
590                     cont:= cont + 1;
591                 end if;
592
593             when ESPERA11 =>
594                 if(cont = 52)then
595                     nx_state<=WRT_LOWER;
596                     cont:=0;
597                 else
598                     RS<='0';
599                     DB<="0000";
600                     nx_state<=ESPERA11;
601                     cont:= cont + 1;
602                 end if;
603
604             when WRT_LOWER =>
605                 if(cont = 4)then
606                     if(cont2 = 13)then
607                         if(cont3 = 4)then
608                             nx_state<=ESPERA12;
609                             cont:=0;
610                             cont2:=0;
611                             cont3:=0;
612                         else
613                             E<='0';
614                             nx_state<=WRT_LOWER;
615                             cont3:= cont3 + 1;
616                         end if;
617                     else
618                         E<='1';
619                         nx_state<=WRT_LOWER;
620                         cont2:= cont2 + 1;
621                     end if;
622                 else
623                     RS<='1';
624                     E<='0';
625                     DB<=DATA(3 downto 0);
626                     --DB<="0000";
627                     nx_state<=WRT_LOWER;
628                     cont:= cont + 1;
629                 end if;
630
631             when ESPERA12 =>
632                 if(cont = 2000)then
633                     nx_state<=FIM;
634                     cont:=0;
```

```
635                     else
636                         RS<='0';
637                         DB<="0000";
638                         nx_state<=ESPERA12;
639                         cont:= cont + 1;
640                     end if;
641 
642                 when FIM =>
643                     if(flag = '0')then
644                         nx_state<=FIM;
645                     else
646                         nx_state<=SET_DDRAM_UPPER;
647                     end if;
648             end case;
649         end if;
650     end process;
651     end Behavioral;
```

A.4.3 Porta PS/2

```
1   ----------------------------------------------------------------------------------
    --
2   -- Aluno: Julio Cesar de Paiva Ribeiro
3   -- Professor: Giovanni Alfredo Guarneri
4   -- Create Date: 17:48:20 06/05/2017
5   -- Module Name: TECLADO - Behavioral
6   ----------------------------------------------------------------------------------
    --
7   library IEEE;
8   use IEEE.STD_LOGIC_1164.ALL;
9
10  entity TECLADO is
11      generic (clk_freq: integer:= 50_000_000;
12                debounce_counter_size : INTEGER := 8);
13      port(clk: in std_logic;
14           ps2_clk: in std_logic;
15           ps2_data: in std_logic;
16           --ps2_out: buffer std_logic:='0';
17           code: out std_logic_vector(7 downto 0));
18  end TECLADO;
19
20  architecture Behavioral of TECLADO is
21  signal sync: std_logic_vector(1 downto 0);
22  signal ps2_clk_db: std_logic;
23  signal ps2_data_db: std_logic;
24  signal ps2_word: std_logic_vector(10 downto 0);
25  signal error: std_logic;
26  signal cont_idle: integer range 0 to clk_freq/18_000;
27
28  component debouncer is
29          generic(counter_size : INTEGER);
30          port(clk: in std_logic;
31           tecla: in std_logic;
32           saida: out std_logic);
33  end component;
34
35  begin
36
37  process(clk)
38  begin
39      if(clk'EVENT and clk='1')then
40          sync(0)<= ps2_clk;
41          sync(1)<= ps2_data;
42      end if;
43  end process;
44
45  debouncer_ps2_clk: debouncer
46      GENERIC MAP(counter_size => debounce_counter_size)
47      PORT MAP(clk => clk, tecla=> sync(0), saida => ps2_clk_db);
48
49  debouncer_ps2_data: debouncer
50      GENERIC MAP(counter_size => debounce_counter_size)
51      PORT MAP(clk => clk, tecla=> sync(1), saida => ps2_data_db);
52
53  process(ps2_clk_db)
54  begin
55      if(ps2_clk_db'EVENT and  ps2_clk_db='0')then
56          ps2_word<= ps2_data_db & ps2_word(10 downto 1);
57      end if;
58  end process;
59
60   error <= NOT (NOT ps2_word(0) AND ps2_word(10) AND (ps2_word(9) XOR ps2_word(8
     ) XOR
61          ps2_word(7) XOR ps2_word(6) XOR ps2_word(5) XOR ps2_word(4) XOR
            ps2_word(3) XOR
62          ps2_word(2) XOR ps2_word(1)));
63
64  process(clk)
65  begin
66      if(clk'EVENT and clk='1')then
67          if(ps2_clk_db = '0')then
```

```
68              cont_idle<= 0;
69          elsif(cont_idle /= clk_freq/18_000)then
70              cont_idle<= cont_idle + 1;
71          end if;
72
73          if(cont_idle = clk_freq/18_000 and error='0')then
74              code(7 downto 0)<= ps2_word(8 downto 1);
75          end if;
76      end if;
77  end process;
78  end Behavioral;
79
80
```

A.4.4 Conversor Digital/Analógico

```
----------------------------------------------------------------------------------
--
-- Aluno: Julio Cesar de Paiva Ribeiro
-- Professor: Giovanni Alfredo Guarneri
-- Create Date: 17:48:20 06/05/2017
-- Module Name: CONVERSOR_DA - Behavioral
----------------------------------------------------------------------------------
--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity CONVERSOR_DA is
port(clk: in std_logic;
      rst: in std_logic;
      DATA: in std_logic_vector(11 downto 0);
      ADDRESS: in std_logic_vector(3 downto 0);
      CONV: in std_logic;
      DAC_MOSI: out std_logic;
      DAC_CLR: out std_logic;
      DAC_SCK: out std_logic;
      DAC_CS: out std_logic;
      SPI_SS_B : out  STD_LOGIC;        -- Serial Flash
     AMP_CS : out  STD_LOGIC;           -- Amplifier for ADC
     AD_CONV : out  STD_LOGIC;      -- ADC Conversion start
     SF_CE0 : out  STD_LOGIC;           -- StrataFlash
     FPGA_INIT_B : out  STD_LOGIC);
end CONVERSOR_DA;

architecture Behavioral of CONVERSOR_DA is
type maquina is(S1,S2,S3,S4,S5);
signal nx_state: maquina:= S1;

signal flag: std_logic:='0';
signal DATA_AUX: std_logic_vector(31 downto 0);
signal DATA_1: std_logic_vector(31 downto 0);
begin

    SPI_SS_B <= '1';
    AMP_CS <= '1';
    AD_CONV <= '0';
    SF_CE0 <= '1';
    FPGA_INIT_B <= '0';

process(clk)
begin
    if(clk'EVENT and clk='1')then
        if(flag = '1')then
            DATA_AUX(31 downto 24)<= (others => '0');
            DATA_AUX(23 downto 20)<= "0011";
            --DATA_AUX(19 downto 16)<= "1111";
            --DATA_AUX(15 downto 4)<= "100000000000";
            DATA_AUX(19 downto 16)<= ADDRESS(3 downto 0);
            DATA_AUX(15 downto 4)<= DATA(11 downto 0);
            DATA_AUX(3 downto 0)<= (others => '0');
        end if;
    end if;
end process;

process(DATA_AUX)
begin
    for i in 31 downto 0 loop
        DATA_1(i)<= DATA_AUX(31 - i);
    end loop;
end process;

process(clk,rst)
    variable cont: integer range 0 to 32:=0;
begin
    if(rst = '1')then
        DAC_CS<='1';
        DAC_CLR<='0';
```

```
70          DAC_SCK<='0';
71          DAC_MOSI<='0';
72
73      elsif(clk'EVENT and clk = '1')then
74          case nx_state is
75              when S1 =>
76                  if(CONV = '1')then
77                      DAC_SCK<='0';
78                      DAC_CS<= '1';
79                      DAC_CLR<='1';
80                      DAC_MOSI<='0';
81                      flag<='1';
82                      nx_state<= S2;
83                  else
84                      nx_state<=S1;
85                  end if;
86
87              when S2 =>
88                  flag<='0';
89                  DAC_SCK<='0';
90                  DAC_CS<='0';
91                  DAC_CLR<='1';
92                  DAC_MOSI<=DATA_1(cont);
93                  nx_state<= S3;
94
95              when S3 =>
96                  nx_state<= S4;
97
98              when S4 =>
99                  DAC_SCK<='1';
100                 DAC_CS<='0';
101                 DAC_CLR<='1';
102                 nx_state<= S5;
103                 cont:= cont + 1;
104
105             when S5 =>
106                 DAC_SCK<='1';
107                 DAC_CS<='0';
108                 DAC_CLR<='1';
109                 if(cont = 32)then
110                     cont:=0;
111                     nx_state<= S1;
112                 else
113                     nx_state<= S2;
114                 end if;
115         end case;
116     end if;
117 end process;
118 end Behavioral;
```

A.4.5 Conversor Analógico/ Digital

```
1   ----------------------------------------------------------------------------------
    --
2   -- Aluno: Julio Cesar de Paiva Ribeiro
3   -- Professor: Giovanni Alfredo Guarneri
4   -- Create Date: 17:48:20 06/05/2017
5   -- Module Name: CONVERSOR_AD - Behavioral
6   ----------------------------------------------------------------------------------
    --
7   library IEEE;
8   use IEEE.STD_LOGIC_1164.ALL;
9   use IEEE.STD_LOGIC_ARITH.ALL;
10  use IEEE.STD_LOGIC_SIGNED.ALL;
11  use IEEE.STD_LOGIC_UNSIGNED.ALL;
12
13  entity CONVERSOR_AD is
14  port(  SPI_SCK : out  std_logic;
15         AD_CONV : out  std_logic;
16         SPI_MISO : in  std_logic;
17         SPI_MOSI : out  std_logic;
18         AMP_CS : out  std_logic;
19         AMP_SHDN : out  std_logic;
20           clk: in std_logic;
21           ADC_A: out std_logic_vector(13 downto 0);
22           ADC_B: out std_logic_vector(13 downto 0);
23           SPI_SS_B: out std_logic;
24           DAC_CS: out std_logic;
25          SF_CE0: out std_logic;
26          FPGA_INIT_B: out std_logic;
27           CONV: in std_logic);
28  end CONVERSOR_AD;
29
30  architecture Behavioral of CONVERSOR_AD is
31  type state_type is (S0,S1,S2,A1);
32  signal nx_state : state_type := S0;
33
34  type state_type_clock is (clock_on, clock_off);
35  signal state_clock: state_type_clock := clock_off;
36
37  signal cont: integer range 0 to 34:=0;
38  signal indice: integer range 0 to 40 := 0;
39
40  signal clk_sample : std_logic:= '0';
41  signal risingedge : std_logic:= '1';
42
43  signal ADC_1: signed (13 downto 0);
44  signal ADC_2: signed (13 downto 0);
45  signal out_1: std_logic_vector(13 downto 0);
46  signal out_2: std_logic_vector (13 downto 0);
47  signal ref_1: signed (13 downto 0):="10000000000000";
48  signal ref_2: signed (13 downto 0):="10000000000000";
49  signal ganho_A: std_logic_vector(3 downto 0);
50  signal ganho_B: std_logic_vector(3 downto 0);
51
52  begin
53
54      SPI_SS_B<='1';
55      DAC_CS<='1';
56      SF_CE0<='1';
57      FPGA_INIT_B<='0';
58
59  --divide o clock para 1.5MHz
60  process(clk)
61  begin
62      if(clk'EVENT and clk='1')then
63          if(cont = 33)then
64              risingedge <= risingedge xor '1';
65              clk_sample <= clk_sample xor '1';
66              cont<=0;
67          else
68              cont<= cont + 1;
69          end if;
```

```
70      end if;
71  end process;
72
73  --processo que vai variar SPI_SCK
74  process(clk)
75  begin
76      if(clk'EVENT and clk='1')then
77          case state_clock is
78              when clock_on =>
79                  SPI_SCK<= clk_sample;
80
81              when clock_off =>
82                  SPI_SCK<='0';
83          end case;
84      end if;
85  end process;
86
87  --processo principal
88  process(clk)
89  begin
90      if(clk'EVENT and clk='1')then
91          if((cont = 33) and (risingedge = '1'))then
92              case nx_state is
93                  when S0 =>
94                      if(CONV = '1')then
95                          ganho_A(3 downto 0)<="0001";
96                          ganho_B(3 downto 0)<="0001";
97                          nx_state<=S1;
98                      else
99                          nx_state<=S0;
100                     end if;
101
102                 when S1 =>
103                     AMP_CS<='0';
104                     AMP_SHDN<='0';
105                     if(indice < 4)then
106                         SPI_MOSI<= ganho_B(3 - indice);
107                         indice<= indice + 1;
108                         state_clock<= clock_on;
109                         nx_state<= S1;
110
111                     elsif((indice > 3) and (indice < 8))then
112                         SPI_MOSI<= ganho_A(7 - indice);
113                         indice<= indice + 1;
114                         nx_state<= S1;
115
116                     elsif(indice = 8)then
117                         indice<=0;
118                         AMP_CS<='1';
119                         nx_state<= S2;
120                     end if;
121
122                 when S2 =>
123                     if(indice < 2)then
124                         AD_CONV<='1';
125                         indice<= indice + 1;
126                         nx_state<= S2;
127
128                     elsif(indice = 2)then
129                         AD_CONV<='0';
130                         indice<=0;
131                         nx_state<= A1;
132                     end if;
133
134                 when A1 =>
135                     state_clock<=clock_on;
136                     if(indice = 34)then
137                         indice<=0;
138                         state_clock<=clock_off;
139
140                         out_2(13 downto 0)<= ADC_2(13 downto 0) + ref_2(13
```