UNIVERSIDADE FEDERAL DO OESTE DO PARÁ
INSTITUTO DE ENGENHARIA E GEOCIÊNCIAS
PROGRAMA DE CIÊNCIA E TECNOLOGIA
BACHARELADO INTERDISCIPLINAR EM CIÊNCIA E TECNOLOGIA

DINY SILVANE TEIXEIRA E SILVA
RANIERY DA SILVA RÊGO

FREQUENCÍMETRO UTILIZANDO FPGA

SANTARÉM - PA
2018

DINY SILVANE TEIXEIRA E SILVA
RANIERY DA SILVA RÊGO

FREQUENCÍMETRO UTILIZANDO FPGA

Trabalho de Conclusão de Curso – TCC apresentado ao Curso de Bacharelado Interdisciplinar em Ciência e Tecnologia da Universidade Federal do Oeste do Pará, como requisito parcial para obtenção do título de Bacharel Interdisciplinar em Ciência e Tecnologia.

Orientador: Prof. Me. Marcel Antonionni Andrade Romano

SANTARÉM - PA
2018

Dados Internacionais de Catalogação-na-Publicação (CIP)
Sistema Integrado de Bibliotecas – SIBI/UFOPA

R343f Rêgo, Raniery da Silva
Frequencímetro utilizando FPGA. / Raniery da Silva Rêgo; Diny Silvane Teixeira e Silva.- Santarém- Pa, 2018.
60fls.: il.
Inclui bibliografias.

Orientador:Marcel Antonioni Andrade Romano
Trabalho de Conclusão de Curso (Graduação) – Universidade Federal do Oeste do Pará, Instituto de Engenharia e Geociências, Curso Bacharelado Interdisciplinar em Ciência e Tecnologia.

1. Lógica programável. 2. FPGA. 3. VHDL. I. Romano, Marcel Antonioni Andrade, *orient*. II. Título.

CDD: 23 ed. 500

Bibliotecário - Documentalista: Selma Souza – CRB/2 1.096

DINY SILVANE TEIXEIRA E SILVA
RANIERY DA SILVA RÊGO

FREQUENCÍMETRO UTILIZANDO FPGA

Trabalho de Conclusão de Curso – TCC apresentado ao Curso de Bacharelado Interdisciplinar em Ciência e Tecnologia da Universidade Federal do Oeste do Pará, como requisito parcial para obtenção do título de Bacharel em Ciência e Tecnologia.

Aprovado em: 28 de setembro de 2018.

BANCA EXAMINADORA

Prof. Me. Marcel Antonioni Andrade Romano - Orientador

Prof. Me. Gilson Fernandes Braga Junior - Ufopa

Prof. Dr. Nelson de Souza Amorim - Ufopa

AGRADECIMENTOS

A Deus.

À família.

Ao Prof. Me. Marcel Antonionni Andrade Romano.

Ao técnico do Laboratório de Sinais e Sistemas, Alírio Tenório Furtado Neto.

LISTA DE FIGURAS

Figura 1 – A configuração interna de um FPGA. 13
Figura 2 – Mercado Americano de FPGA por aplicação, 2014 - 2024 (USD Bilhões) 16
Figura 3 – Padrões para descrição de circuitos ou sistemas digitais. 17
Figura 4 – A placa DE2. 20
Figura 5 – Interface do programa Quartus II, versão 13, Web Edition. 21
Figura 6 – Interface do *software* ModelSim Altera *Starter Edition* 10.1d. 22
Figura 7. Gerador de funções e formas de ondas arbitrárias de 30MHz Agilent. 23
Figura 8 – Diagrama do fluxo de projeto. 23
Figura 9 – Criação de novo projeto. 25
Figura 10 – Nomeação de diretório, projeto e entidade. 26
Figura 11 – Seleção do FPGA. 26
Figura 12 – Criação de novo arquivo de texto. 27
Figura 13 – Seleção da linguagem na qual o projeto será descrito. 27
Figura 14 – Rótulo dos sete segmentos do *display*. 28
Figura 15 – Entidade do frequencímetro. 29
Figura 16 – Processo para obtenção de *clock* de 1 Hz. 30
Figura 17 – Processo para contagem da frequência e decomposição dos algarismos. 30
Figura 18 – Componente para exibição do caractere no *display* de sete segmentos. 31
Figura 19 – Pinos padrão do LCD e rótulo correspondente na placa DE2 32
Figura 20 – Entidade do projeto. 33
Figura 21 – Processo para contagem da frequência e separação dos algarismos do projeto para apresentação da frequência lida no display LCD. 34
Figura 22 – Laço para envio dos caracteres para o barramento de dados. 36
Figura 23 – Seleção da ferramenta de simulação do software Quartus II 37
Figura 24 – Seleção do projeto a ser simulado no ModelSim. 37
Figura 25 – Configuração das entradas do projeto. 38
Figura 26 – Entrada do período de simulação 38
Figura 27 – Formas de onda da simulação funcional para o *display* de 7 segmentos. 39
Figura 28 – Relatório de síntese no Quartus II do projeto para apresentação da frequência lida no *display* de 7 segmentos. 40
Figura 29 – Designação das portas declaradas no código para os pinos do FPGA. 40
Figura 30 – Formas de onda da simulação temporal para sinal de entrada de 1000 Hz. 41
Figura 31 – Teste com gerador de sinais. 42
Figura 32 – Formas de onda da simulação funcional da rotina de inicialização do projeto para apresentação da frequência lida no *display LCD*. 42
Figura 33 – Formas de onda da simulação funcional após 500 milissegundos do projeto para apresentação da frequência lida no *display LCD*. 43
Figura 34 – Relatório de síntese do projeto para apresentação da frequência lida no *display LCD*. 43
Figura 35 – Simulação temporal do projeto para apresentação da frequência lida no *display LCD*. 44
Figura 36 –Teste com gerador de sinais para sinal de entrada igual a 1000 Hz. 45

SUMÁRIO

1 INTRODUÇÃO 9
2. OBJETIVOS 11
2.1 Geral 11
2.2 Específicos 11
3 DISPOSITIVOS LÓGICO PROGRAMÁVEIS 12
3.1 SPLD 12
3.2 CPLD 13
3.3 FPGA 13
3.3.1 Tecnologias 14
3.3.2 Aplicações 15
4 LINGUAGENS DE DESCRIÇÃO DE *HARDWARE* 17
4.1 VHDL 17
5 MATERIAIS 19
5.1 Altera DE2 19
5.2 Quartus II 21
5. 3 ModelSim 21
5. 4 Gerador de funções e formas de ondas arbitrárias 22
6 MÉTODOS 23
6.1 Entrada do projeto 25
6.1.1 Entrada do projeto para exibição da frequência no *display* de sete segmentos 28
6.1.2 Entrada do projeto para exibição da frequência no *display* LCD 32
6.2 Passo-a-passo para execução de simulações 36
7 RESULTADOS E DISCUSSÃO 39
7.1 Exibição da frequência no *display* de sete segmentos 39
7.1.1 Simulação Funcional 39
7.1.2 Síntese 39
7.1.3 Implementação 40
7.1.4 Simulação temporal 41
7.1.5 Download 41
7.2 Exibição da frequência no *display* LCD 42
7.2.2 Simulação Funcional 42
7.2.3 Síntese 43

7.2.4 Implementação 43
7.2.5 Simulação temporal 44
7.2.6 Download 44
8 CONSIDERAÇÕES FINAIS 46
REFERÊNCIAS 47
APÊNDICE A – Código do projeto para exibição da frequência no *display* de 7 segmentos 49
APÊNDICE B – Atribuição de pinos do projeto para exibição da frequência no *display* de 7 segmentos 52
APÊNDICE C – Código do projeto para exibição da frequência no *display* LCD 53
APÊNDICE D – Atribuição de pinos do projeto para exibição da frequência no *display* LCD 60
ANEXO A – Instruções para o LCD 61

RESUMO

O FPGA é um circuito lógico programável constituído de blocos lógicos organizados em padrão matricial altamente reconfiguráveis, permitindo ao usuário o controle sobre as funções a serem executadas de acordo com a necessidade. Este pode ser associado com uma linguagem de descrição de *hardware* para circuitos integrados de altíssima velocidade, que permite que os processos sejam executados concomitantemente, o que desobriga o programador de se preocupar com a ordem dos processos ao longo do código. Neste trabalho, utilizou-se o kit educacional DE2 da Altera©, pertencente à família de placas programáveis em FPGA, fazendo um registro do fluxo de projeto para implementação de um instrumento de monitoramento de frequência.

Palavras-chaves: lógica programável, FPGA, VHDL.

ABSTRACT

The FPGA is a programmable logic circuit consisting of logic blocks organized in highly reconfigurable matrix patterns, allowing the user to control the functions to be executed according to the need. This can be associated with a hardware description language for very high-speed integrated circuits, which allows the processes to be executed concomitantly, which frees the programmer from worrying about the order of processes throughout the code. In this work the educational kit DE2 of Altera © was used, belonging to the family of programmable boards in FPGA, making a project flow record for implementation of a frequency monitoring instrument.

Keywords: Programmable logic, FPGA, VHDL.

1 INTRODUÇÃO

Após a criação do ENIAC, em meados da segunda guerra mundial, visando cálculos mais precisos de balística, o desenvolvimento dos computadores e suas seguintes gerações propiciou o surgimento de novos dispositivos eletrônicos, atendendo às necessidades de uma tecnologia em rápido crescimento. Nesse ínterim, na década de 1980, os diagramas em desenho já não eram suficientes para abarcar a quantidade de circuitos digitais existentes até mesmo em um único chip, e assim as linguagens de descrição de *hardware*, que foram padronizadas para evitar conflitos entre os diversos circuitos integrados produzidos na época, ganharam impulso notoriamente.

A lógica de programação necessita de *hardware* e *software*. Dispositivos de lógica programável podem ser programados para realizar funções lógicas especificadas pelo fabricante ou pelo usuário. Uma vantagem da lógica programável sobre a lógica de funções fixas é que os dispositivos programáveis ocupam bem menos espaço na placa para uma mesma quantidade de lógica. Outra vantagem é que, com a lógica programável, podem-se alterar os projetos com facilidade sem alterações físicas no *hardware* ou substituição de componentes. Além disso, um projeto lógico geralmente pode ser implementado mais rápido e com um menor custo com a lógica programável do que com circuito integrado (CI) de função fixa. Dentre os dispositivos de lógica programável, o surgimento do FPGA possibilitou ao programador reconfigurar o dispositivo dependendo da função requerida.

Assim, foi possível criar tecnologias capazes de conectar *hardwares* externos a uma placa de desenvolvimento, conectar os pinos necessários, e fazer com que a placa interagisse (leitura/escrita) com sinais do *hardware* a ele conectado. Nesse aspecto de interfaces entre *hardwares* externos entre si, as áreas comuns de aplicação de FPGA incluem os setores automotivo, militar e aeroespacial, além de soluções para equipamentos médicos, automação industrial, sistemas de armazenamento em larga escala, computação de alta performance, *broadcasting*, instrumentação e outras [1] [2].

Com tal crescimento de aplicações e usos de lógica programável, é coerente que haja uma disseminação dos conhecimentos acerca do assunto, facilitando o acesso a esta tecnologia. Nesse sentido, o desenvolvimento de um dispositivo prático que propicie a consolidação de conhecimentos associados a Sistemas Digitais em Circuitos Programáveis (FPGA) proporciona espaço para o desenvolvimento de protótipos e projetos para pesquisas em inteligência computacional.

Tendo em vista a percepção do mercado de sistemas embarcados no cenário brasileiro, em 2015, o Portal Embarcados realizou uma pesquisa aberta, destinada aos profissionais atuantes no mercado nacional do segmento de sistemas embarcados. O objetivo era caracterizar o perfil do profissional no desenvolvimento de sistemas embarcados [3].

Uma das tecnologias em debate de aplicabilidade na comunidade de sistemas embarcados é o FPGA. Seu principal uso ainda se deve à tecnologia de processamento paralelo, diferente da linguagem de programação sequencial, usada em microcontroladores.

Além disso, o FPGA permite a possibilidade de teste e validações de códigos que demandam pouco período de tempo, antes da sua implementação, ainda na fase de protótipo. Porém, apenas 63% dos profissionais tem experiência com FPGA [3].

Do ponto de vista nacional, de acordo com a pesquisa, 61% dos profissionais são da Região Sudeste, 28% da Região Sul, e as regiões Norte e Centro Oeste, juntas, somam apenas 11% do mercado [3]. Mostrando que a região setentrional do país possui potencial para expandir o estudo desta tecnologia, ainda pouco difundida no mercado de sistemas embarcados.

Portanto, o presente trabalho tem relevância na divulgação da tecnologia FPGA na região norte do país.

2. OBJETIVOS

2.1 Geral

O presente trabalho busca utilizar o FPGA empregando a linguagem VHDL em aplicações de engenharia, para que de modo tátil consiga-se interfacear uma placa de desenvolvimento com *hardware* externo.

2.2 Específicos

- Implementar um sistema de monitoramento de frequência utilizando FPGA.
- Ampliar o entendimento sobre Sistemas Digitais em Circuitos Programáveis (FPGA).

3 DISPOSITIVOS LÓGICO PROGRAMÁVEIS

Existe uma larga faixa de circuitos integrados (CI), padronizados com funções variadas e circuitos lógicos em chips, que podem ser usados para implementar circuitos ou sistemas [4]. Esses circuitos integrados estão presentes no mercado com custo razoavelmente baixo, sendo amplamente utilizados por projetistas.

A maioria dos dispositivos eletrônicos utilizados atualmente, porém, possuem esses circuitos integrados programados no ato da fabricação para aplicações específicas e não permitem reconfiguração. Em geral, essa rigidez torna o projeto do circuito pouco tolerante a erros e é conveniente apenas em dispositivos projetados como matriz para produção em massa, em virtude do alto custo da fase de construção da matriz [5].

Além disso, para projetos que demandam centenas ou milhares de CI há dificuldades associadas ao espaço necessário na placa de circuito impresso e ao tempo gasto soldando e testando estes componentes [4]. A fim de reduzir o número de CIs, há soluções LSI (*large scale integration*) e VLSI (*very large scale integration*) para funções padronizadas (memórias, microprocessadores, sintetizadores de voz, chips de calculadoras e outras). Para casos em que não há soluções LSI e VLSI é conveniente o uso de dispositivos lógicos programáveis, ou de lógica programável, PLD (*Programmable Logic Device*), como "alternativa para substituir um grande número de CIs padronizados por um único CI" [4].

Um dispositivo lógico programável, desse modo, é um CI com vasta quantidade de portas, registradores e *flip-flops* (FF) conectadas no chip. Sendo que grande parte são fusíveis disponíveis para "queima". Isso o torna programável porque sua função final dependerá da escolha das conexões que ficam abertas ou que permaneçam inalteradas [4]. Os fusíveis que são queimados são determinados pelo fabricante de acordo com as instruções do cliente, ou mesmo pelo próprio cliente [4]. Este processo é denominado de programação porque resulta em um padrão de interconexões de portas, registradores e FFs [4].

Os tipos principais de dispositivos lógicos programáveis pelo usuário são PLD e FPGA (*Field Programmable Gate Arrays*) [6]. As PLDs são subdivididas em PLDs simples (SPLD) e complexas (CLPDs) [6].

3.1 SPLD

Os SPLDs são subdivididos em PAL (*programmable array logic*) e GAL (*generic array logic*). Os dispositivos PAL são compostos por um arranjo programável de portas AND e um arranjo fixo de portas OR e geralmente são programados apenas uma vez (OTP – *one-*

time programmable), enquanto que um dispositivo GAL pode ser reprogramado; porém, há uma linha tênue entre PAL e GAL, pois alguns dispositivos programáveis ainda são chamados de PAL, e um GAL, por sua vez, essencialmente é um PAL reprogramável.

A estrutura de PAL e GAL é um arranjo OR fixo e um arranjo AND programável, sendo basicamente uma arquitetura de soma de produtos.

3.2 CPLD

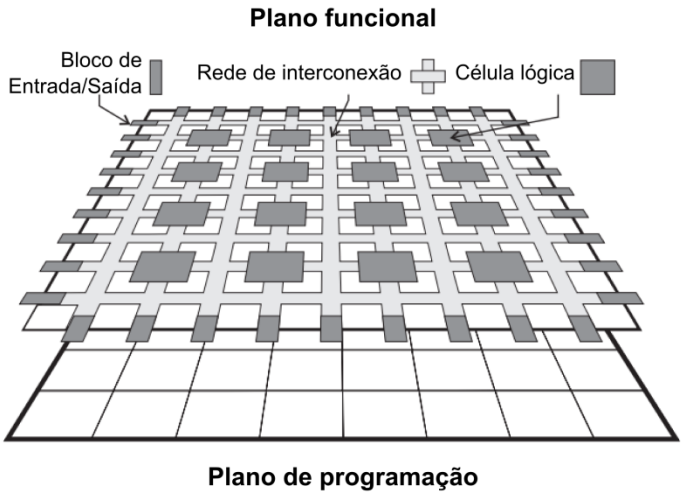
Um CPLD é um dispositivo lógico programável complexo por ser apenas um dispositivo com múltiplos SPLDs. Ele pode substituir diversos CIs de funções fixas e proporciona maior capacidade de projetos com circuitos lógicos maiores.

3.3 FPGA

Em 1984 a empresa Xilinx apresentou a tecnologia de portas programáveis [7] presente no FPGA. Os FPGAs "são circuitos programáveis compostos por um conjunto de células lógicas ou blocos lógicos alocados em forma de matriz" [8]. A inovação introduzida pelos FPGAs foi o número indefinido de ciclos de reconfiguração (cada um com duração de poucos milissegundos) e a alta complexidade [7]. Este circuito permite que o usuário (programador) reconfigure o dispositivo de acordo com suas necessidades.

A arquitetura de um FPGA é composta por dois elementos fundamentais: o plano funcional e o plano de programação [7] (Figura 1).

Figura 1 – A configuração interna de um FPGA.



Fonte: [7], com adaptações.

O usuário não vê o plano de programação que implementa fisicamente a configuração do FPGA. Há muitos elementos de armazenamento simples neste plano de programação, e cada um desses elementos é responsável por controlar uma função específica de determinada parte do plano funcional.

No plano funcional, há três elementos fundamentais:

a) células ou blocos lógicos: permitem que funções booleanas arbitrárias sejam implementadas, com cerca de 4 a 6 entradas. FPGAs modernas têm alguns milhares de células lógicas [7] [8];
b) blocos de entrada/saída: localizam-se na borda do FPGA e ligando-o ao ambiente exterior. As células E/S são programadas como entrada, saída, ou pino bidirecional [7] [8];
c) interconexões programáveis: conectam blocos lógicos e de entrada e saída por meio de canais de roteamento [6] [7] [8].

Dentre as vantagens do FPGA, há a velocidade, baixo consumo de energia, redução do tamanho e quantidade de componentes (e do custo), maior flexibilidade e possibilidade de atualização [9].

3.3.1 Tecnologias

Há cinco tecnologias de processos usadas para implementar conexões programáveis em dispositivos lógico programáveis:

a) Tecnologia de fusível: Dispositivos dessa categoria utilizam fusíveis para ligar linhas e colunas na matriz de interconexão. Desse modo, todas as conexões estão inicialmente em curto-circuito. Ao programar o dispositivo, selecionam-se fusíveis para queima e fusíveis para permanecer intactos. Esses dispositivos são programáveis apenas uma vez (OTP – *one-time programmable*) [6].
b) Tecnologia antifusível: Nessa categoria, os dispositivos empregam antifusíveis para ligar linhas e colunas na matriz de interconexão. Os antifusíveis são compostos por dois condutores separados por um isolante, ou seja, as conexões iniciam em circuito aberto. O equipamento empregado na programação desses dispositivos aplica tensão suficiente para romper o isolamento das conexões desejadas. Esses dispositivos também são programáveis apenas uma vez (OTP) [6].

c) Tecnologia EPROM (*electrically programmable read-only memory*): Nessa tecnologia a conexão programável é um transistor de porta flutuante (tipo especial de transistor MOS) [6]. Em geral, os dispositivos baseados em EPROM são programáveis uma vez. Mas, aqueles que possuem encapsulamento com "janela" podem ser apagados com luz ultravioleta (UV) e reprogramados [6].

d) Tecnologia EEPROM/*Flash*: A tecnologia baseada em EEPROM (*electrically-erasable programmable read-only memory*) utiliza transistores de porta flutuante que podem ser apagadas e reprogramadas eletricamente sem a necessidade de luz UV ou equipamentos especiais [6]. Dispositivos com essa tecnologia podem ser programados após serem instalados na placa de circuito impresso, e muitos deles podem ser reprogramados enquanto operam num sistema [6]. Um arranjo *flash* é um tipo de arranjo EEPROM apagável de forma mais rápida que a tecnologia EEPROM padrão [6].

e) Tecnologia SRAM (*static random-access memory*): A tecnologia SRAM é volátil, isto é, ela não retém os dados quando a alimentação é desligada. Quando a alimentação é ligada, os dados da programação têm que ser carregados na memória fim de reprogramar o dispositivo [6].

As tecnologias baseadas em SRAM e Flash dominaram a maior fatia da indústria em 2015 e espera-se que elas mantenham seu domínio até 2023 [10].

3.3.2 Aplicações

Os FPGAs são plataformas flexíveis para desenvolvimento de soluções e têm ampla gama de aplicações nas seguintes áreas:

a) Telecomunicação: Interfaces SONET, interfaces de fibras ópticas, ATM, interfaces ISDN, controlador de *voice-mail*, multiplexadores T1 e compressão de dados [8].

b) Indústria: Equipamentos de teste e medidas, equipamentos médicos, controle remoto, robótica, emulador ASIC e sistemas de visão [8].

c) Automotiva: sistema *infotainment*, processamento de imagens e conectividade [2];

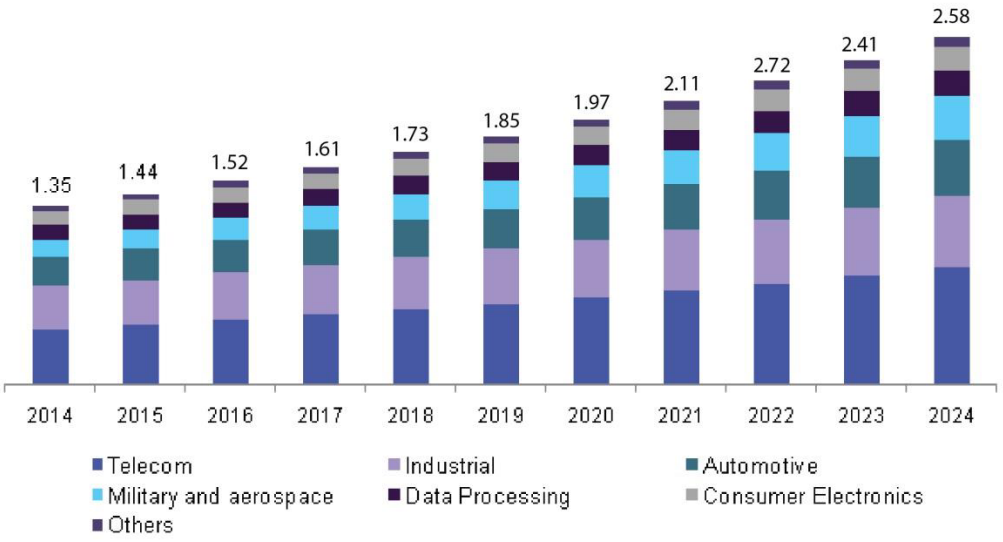
d) Militar e aeroespacial: sistemas de computadores, comunicação e controle de fogo [8];

e) Processamento de dados: processamento digital de sinais, Multiplexadores, roteadores, vídeo conferência, criptografia, modems, compressão de dados e Wireless [2] [8];
f) Eletrônicos de consumo: *displays*, câmeras digitais, impressoras multifuncionais, eletrônicos portáteis [2];
g) Computação de alta performance: servidores, super computadores e mineração de dados [1];
h) Dispositivos médicos: ultrassom e sistemas cirúrgicos [1].

A Altera Corp. (agora parte da Intel) e a Xilinx Inc. estão na indústria há mais de 30 anos e juntas detêm quase 90% da participação no mercado global de FPGAs [10]. Essas duas empresas são altamente integradas em toda a cadeia de valor e estão intimamente envolvidas nos processos de projeto e teste [10]. A competição dentro da indústria é muito alta, com participantes-chave competindo constantemente uns com os outros para criar tecnologias inovadoras [10]. Eles têm ofertas similares de produtos e seguem preços competitivos para ganhar uma fatia maior do mercado [10].

A Figura 2 exibe a distribuição de aplicações ao longo dos anos no mercado americano e a projeção até 2024.

Figura 2 – Mercado Americano de FPGA por aplicação, 2014 - 2024 (USD Bilhões)



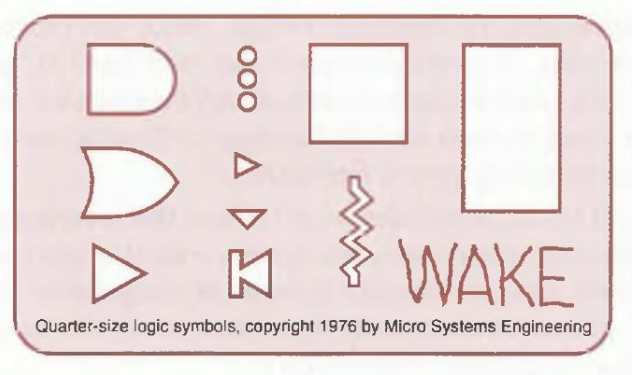
Fonte: [10], com adaptações.

O bom desempenho no mercado dos FPGAs é atribuído ao curto prazo de comercialização de produtos e serviços, alto desempenho, alto grau de confiabilidade, bem como soluções rentáveis, entre outros [10].

4 LINGUAGENS DE DESCRIÇÃO DE *HARDWARE*

Até a década de 1980, o desenho de diagramas era o principal meio para descrever circuitos e sistemas digitais [11]. As principais ferramentas utilizadas para descrever projetos digitais incluíam régua, lápis e os padrões da Figura 3.

Figura 3 – Padrões para descrição de circuitos ou sistemas digitais.



Fonte: [11].

Com o passar dos anos, o uso de diagramas tornou-se inviável em virtude do crescimento acelerado do número de portas disponíveis no mesmo *chip*. Com a crescente complexidade dos circuitos e sistemas digitais, o uso de linguagens de descrição de *hardware* (HDL – *hardware description languages*), cuja descrição de projeto do dispositivo lógico se dá via texto, popularizou-se naturalmente a fim de garantir portabilidade aos projetos.

Algumas das HDL incluem ABEL, VHDL, Verilog e AHDL. A maioria dos fabricantes de lógica programável oferece pacotes de software que suportam as linguagens VHDL (VHSIC *Hardware Description Language*) e Verilog porque elas são HDLs padrão [6].

Neste trabalho, o projeto foi implementado em VHDL. Dentre as vantagens da utilização VHDL, há a redução do tempo/custo de desenvolvimento, o maior nível de abstração, a independência entre projetos e tecnologia, a facilidade para atualização de projetos e o grande número de usuários no mundo [8].

4.1 VHDL

VHDL (VHSIC *Hardware Description Language*) é uma linguagem de descrição de *hardware* para circuitos integrados de altíssima velocidade (VHSIC – *Very High Speed Integrated Circuit*) [12]. O sistema ou circuito pode ser implementado a partir da descrição comportamental (funcionamento dos componentes do circuito) por meio de um código VHDL.

O desenvolvimento da linguagem foi financiado pelo Departamento de Defesa dos Estados Unidos em meados da década de 1980, já que se fez preciso uma padronização para o projeto VHSIC, na ferramenta de projeto e documentação [12] [13]. A padronização era necessária em virtude da incompatibilidade gerada pelas diferentes linguagens de descrição de *hardware* utilizadas pelos fabricantes de circuitos integrados. Isso se tornou uma vantagem no desenvolvimento de circuitos em HDL em vista da portabilidade adquirida entre sistemas de desenvolvimentos, mesmo de fabricantes diferentes [14]. O Instituto de Engenheiros Eletrônicos e Eletricistas (IEEE – *Institute of Electrical and Electronics Engineers*) adotou a VHDL como linguagem padrão e esta é designada como o padrão 1076-1993 do IEEE [6].

Uma das características fundamentais da linguagem VHDL é o paralelismo na execução dos comandos. Enquanto a maioria dos programas de computador é implementada de maneira sequencial, excetuando regiões específicas do código (processos, funções e procedimentos, por exemplo), em VHDL todos os comandos são executados concorrentemente. Isto torna a ordem de distribuição das declarações em VHDL irrelevante para o comportamento da descrição [12].

5 MATERIAIS

A programação de um dispositivo lógico programável depende de quatro itens: um computador, um software de desenvolvimento, o dispositivo lógico programável e um sistema para conexão do dispositivo ao computador [6].

O circuito descrito foi sintetizado no FPGA Cyclone II, modelo EP2C35F672C6 (Altera DE2). Os *softwares* Quartus II , versão 13, *Web Edition* e ModelSim Altera Starter Edition 10.1d foram utilizados para descrição e simulação, respectivamente. Empregou-se o gerador de funções e formas de ondas arbitrárias de 30MHz Agilent 33521A para testes.

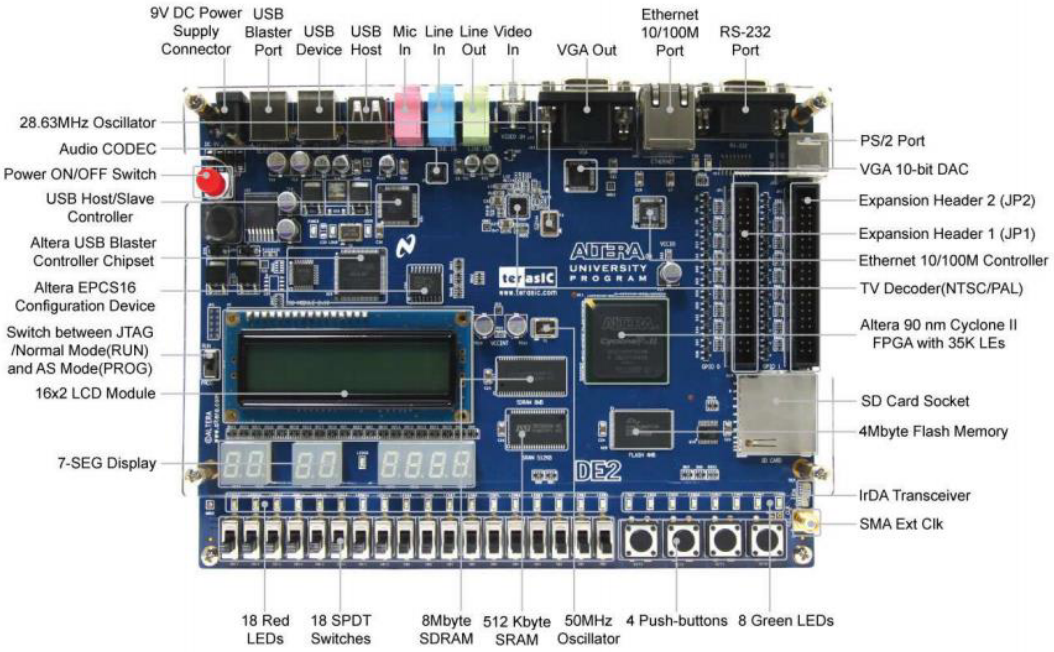
5.1 Altera DE2

A placa Altera DE2 é um *kit* de desenvolvimento da família de placas programáveis em FPGA, neste caso, a Cyclone II. Trata-se de uma ferramenta de aprendizagem de sistemas digitais e permite que sejam desenvolvidas aplicações simples de lógica combinacional ou aplicações mais elaboradas em lógica sequencial, pois permite que sejam processados sinais de áudio e vídeo, por exemplo.

A placa também não precisa de dispositivo externo para que seja programada, pois, para isso, basta conectá-la a um computador por meio de uma entrada USB; se os *drivers* da placa estiverem corretamente instalados o computador alimentará e reconhecerá o *hardware* da placa.

A Figura 4 mostra a placa DE2 com seus componentes indicados.

Figura 4 – A placa DE2.



Fonte: [15].

O FPGA do kit possui 35 mil elementos lógicos e outros recursos descritos de maneira sucinta na Tabela 1.

Tabela 1 – Informações da placa DE2.

Recurso	Descrição
FPGA	Cyclone II EP2C35F672C6 com dispositivo de configuração serial EPCS16 16-Mbit
Processador	Nios II
Interfaces de Entrada/Saída	USB-Blaster integrada para configuração da FPGA Line In/Out, Entrada para microfone (CODEC de áudio de 24-bit) Saída de vídeo Out (VGA 10-bit DAC) Entrada de vídeo (NTSC/PAL/Multi-formato) RS232 Porta para infravermelho Porta para mouse PS/2 ou teclado Ethernet 10/100 USB 2.0 (tipos A e B) *Expansion headers* (duas com 40-pinos)
Memória	SDRAM de 8 MB, SRAM de 512 KB, Flash de 4 MB Entrada para cartão de memória SD
Displays	Oito *displays* de 7- segmentos *Display* LCD 16 x 2
Chaves e LEDs	18 chaves 18 LEDs vermelhos 9 LEDs verdes 4 botões
Clocks	50 MHz clock 27 MHz clock Entrada para clock externo SMA

Fonte: [15].

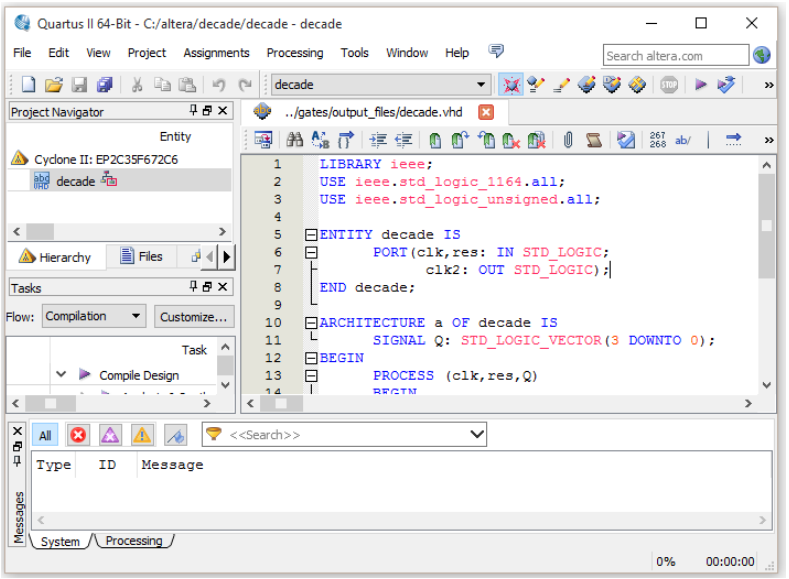
5.2 Quartus II

Os fabricantes de SPLD, CPLD e FPGA fornecem para cada dispositivo um pacote de suporte de *software*, ou projeto auxiliado por computador (CAD – *computer aided design*). A combinação do *software* com o *hardware* como unidade funcional influencia o resultado mostrado na lógica programável [6].

Esse ambiente de desenvolvimento agrupa um conjunto de ferramentas necessárias para o projeto de circuitos digitais. As ferramentas incluem editor de texto para escrever, editar e salvar os programas em HDL, além de compilador, sintetizador, simulador e outras.

Neste trabalho foi utilizado o *software* Quartus II fornecido atualmente pela Intel. A versão gratuita desse programa é a *Web Edition* e na Figura 5 é possível visualizar a interface da versão que possui suporte para o kit DE2.

Figura 5 – Interface do programa Quartus II, versão 13, Web Edition.



Fonte: Captura de tela do *software* Quartus II.

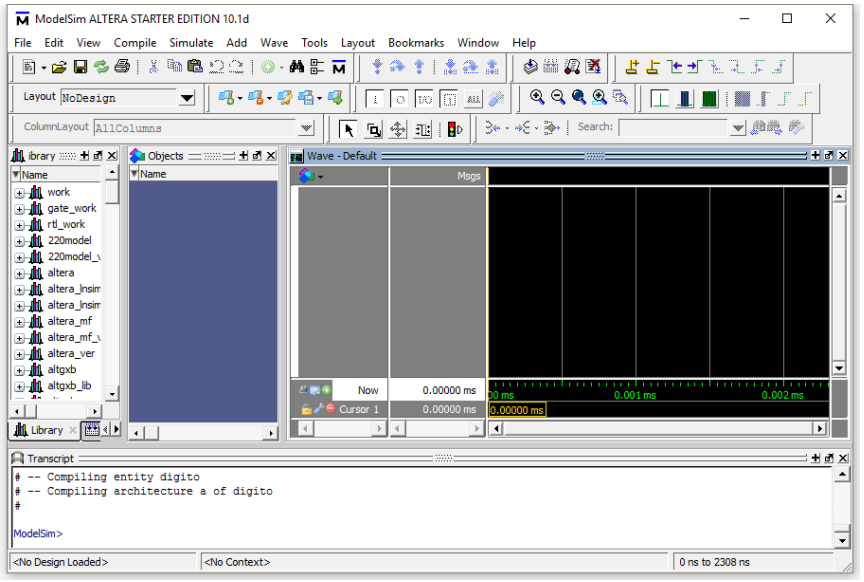
5. 3 ModelSim

Antes de o projeto ser transferido para o *hardware*, é possível realizar simulações, a fim de garantir a funcionalidade e evitar problemas de temporização que afetem a operação global do dispositivo.

Neste trabalho foi utilizado o *software* ModelSim, que é uma ferramenta para simulação do ambiente VHDL com objetivo de fazer verificação do código-fonte e averiguar se há erros na simulação.

A Figura 6 exibe a interface do programa. Na janela *Wave* é possível visualizar os resultados da simulação em formas de ondas HDL, e seus respectivos valores; permitindo análise detalhada após o fim da simulação [16].

Figura 6 – Interface do *software* ModelSim Altera *Starter Edition* 10.1d.



Fonte: Captura de tela do *software* ModelSim.

5. 4 Gerador de funções e formas de ondas arbitrárias

Os geradores de forma de onda produzem formas de onda analógicas que servem como fonte de sinal para teste de diversos aparelhos e equipamentos eletrônicos.

O Agilent 33521A é um gerador funções e formas de onda arbitrárias de 30 MHz e um gerador de pulsos em um único instrumento (Figura 7). Para geração de formas de onda, este dispositivo possui as seguintes especificações [17]:

- Largura de banda de 30 MHz para pulsos, ondas senoidais e quadradas;
- Jitter < 40 ps e distorção harmônica total menor que 0,04% para sinais precisos;
- Taxa de amostragem de 250 Msa/s, 16 bits para for formas de onda arbitrárias com maior resolução de tempo;
- Formas de onda arbitrárias ponto a ponto para representações mais acuradas dos sinais determinados pelo usuário.

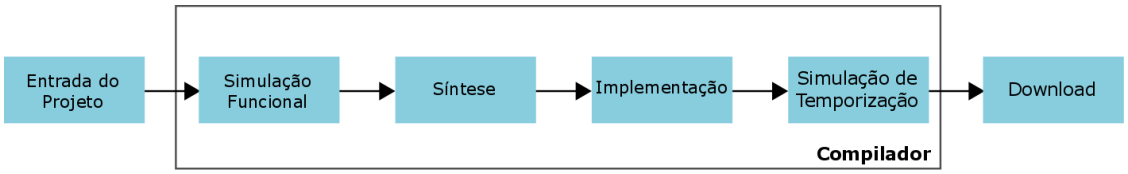
Figura 7 – Gerador de funções e formas de ondas arbitrárias de 30MHz Agilent.

Fonte: [17].

6 MÉTODOS

A metodologia adotada seguiu o fluxo de projeto, ou seja, o processo de implementação de um projeto lógico em um FPGA ou outro dispositivo de lógica programável. Isso envolve os seguintes passos: (1) entrada do projeto, (2) simulação funcional, (3) síntese, (4) implementação, (5) simulação de temporização e (6) *download* [6]. A Figura 8 ilustra o fluxo de processo em diagrama de blocos. Cada passo será explanado a seguir conforme Floyd [6].

Figura 8 – Diagrama do fluxo de projeto.



Fonte: [4], com adaptações.

A especificação de um projeto é inserida no *software* de desenvolvimento de projeto usando entrada baseada em texto (HDL), entrada gráfica ou descrição em diagrama de estados.

Após a inserção do projeto, o código é compilado. O compilador traduz o código-fonte ou entrada análoga para o código-objeto a ser simulado ou transferido para o dispositivo programável. O código, nesta etapa, é analisado pelo compilador na busca por erros de sintaxe [11].

Em seguida, executa-se a simulação funcional. Nesta fase, o projeto é simulado em uma ferramenta de *software* independente do *hardware*. O simulador verifica se são alcançadas as saídas esperadas para entradas específicas antes de sintetizar o projeto para o *hardware*. Caso sejam detectadas falhas, são realizadas as correções necessárias no código-fonte ou outra entrada escolhida na primeira etapa.

O passo seguinte é a síntese. A ferramenta de síntese seleciona as estruturas necessárias para implementar um circuito que corresponda à descrição [13]. Isto é, o projeto é otimizado reduzindo número de portas, eliminando lógica redundante e substituindo elementos lógicos por outros mais eficientes. Assim é gerada uma lista (*netlist*) de forma padronizada e independente do dispositivo.

A *netlist* é a base de dados para implementação. Ela é basicamente uma lista de conectividade que descreve os componentes e como serão interconectados [6]. Nesta etapa de implementação, a estrutura lógica descrita pela lista é mapeada no *hardware* a ser programado. A saída desta etapa depende do *hardware* utilizado e é um arquivo denominado sequência de bits.

A etapa seguinte é a simulação de temporização. Ela garante que não haja falhas no projeto ou problemas de temporização em função dos atrasos na propagação.

Sanadas as falhas da etapa anterior, realiza-se o *download* da sequência de bits para o dispositivo escolhido.

Os projetos têm como entradas o *clock* de 50 MHz da placa e o pino 0 da *General Port Input/Output-0* (rotulado GPIO-0 no *kit*) no qual é ligado o sinal a ser lido. Foram implementados dois códigos: um no qual a frequência lida é apresentada no *display* de sete segmentos e outro no qual esta é apresentada no *display* LCD. Há 8 *displays* de 7 segmentos no kit, rotulados como HEX7, HEX6, HEX5, HEX4, HEX3, HEX2, HEX1, HEX0. No projeto utilizaremos os *displays* HEX3, HEX2, HEX1, HEX0.

No projeto no qual a frequência é apresentada no *display* LCD, é possível alterar a unidade do dado lido com as chaves da placa. No kit, há 18 chaves, das quais foram utilizadas: SW3, SW2, SW1, SW0.

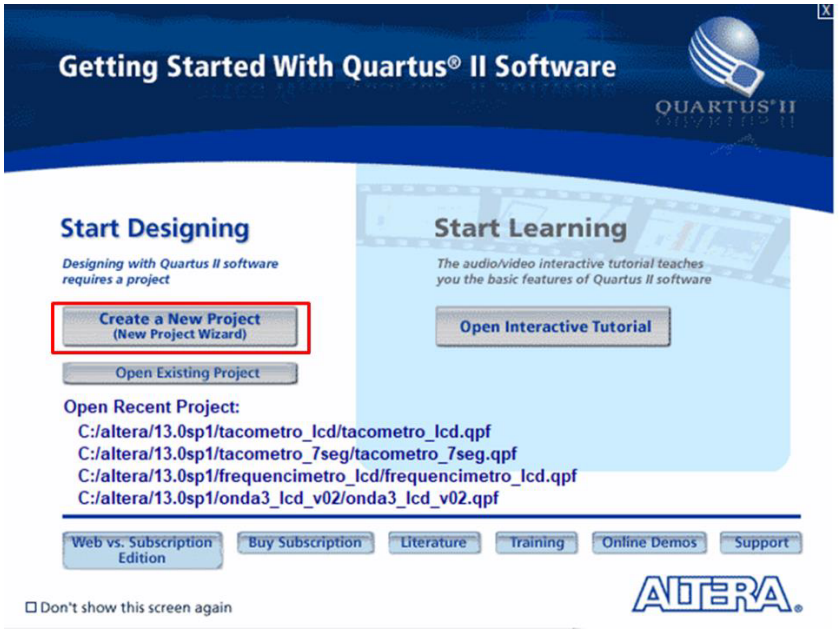
No código, a biblioteca empregada em ambos os projetos foi a “IEEE”. Os pacotes usados dessa biblioteca e suas respectivas funções são:

a) STD_LOGIC_1164: define tipos, subtipos e funções caracterizados por valores lógicos. Exemplos: “std_logic”, “std_logic_vector” e “rising_edge” [18];
b) STD_LOGIC_ARITH: combinado com o pacote “STD_LOGIC_1164” define alguns tipos para representação numérica, operações aritméticas básicas, funções para comparação e conversão como “+”, “-”,“<”, “conv_integer” [18];
c) STD_LOGIC_UNSIGNED: contém operações aritméticas básicas e funções de comparação para “std_logic_vector”’.

6.1 Entrada do projeto

O primeiro passo para criação de um novo projeto no programa Quartus II, versão 13, Web Edition é selecionar “Create New Project” na janela de boas vindas Figura 9.

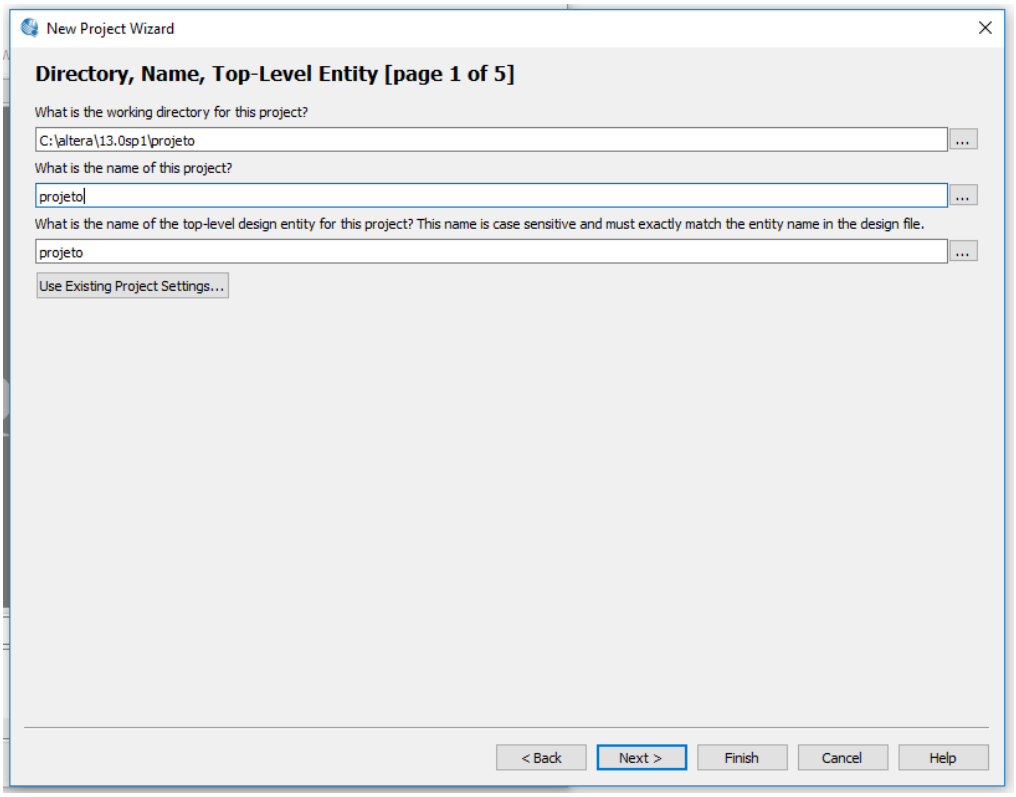
Figura 9 – Criação de novo projeto.



Fonte: Captura de tela do *software* Quartus II.

O passo seguinte é nomear o projeto (Figura 10). É necessário que o projeto, a pasta que agrupará os arquivos do projeto e a entidade tenham o mesmo nome.

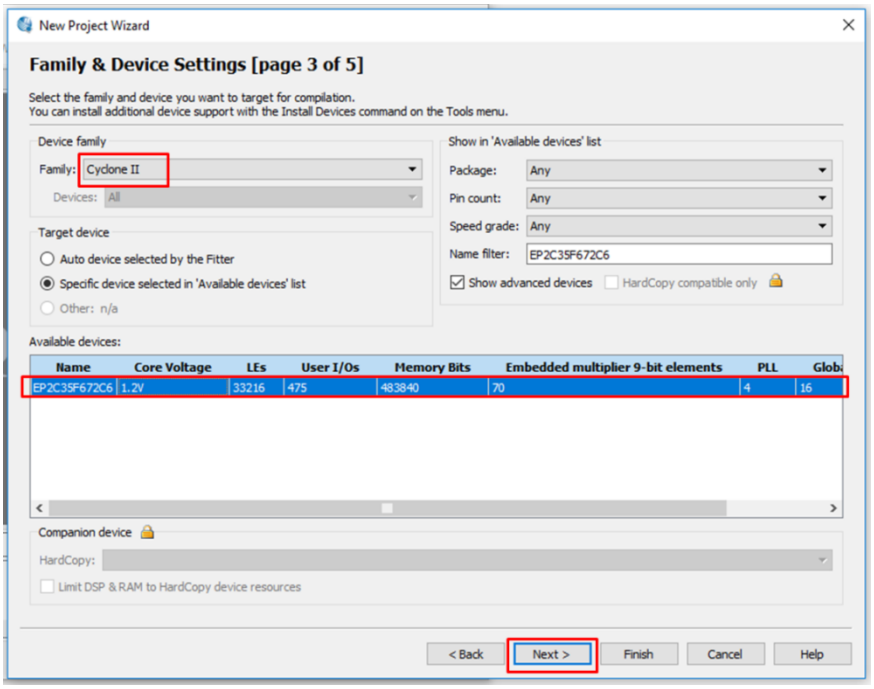
Figura 10 – Nomeação de diretório, projeto e entidade.



Fonte: Captura de tela do *software* Quartus II.

O terceiro passo permite selecionar o dispositivo alvo do projeto (Figura 11).

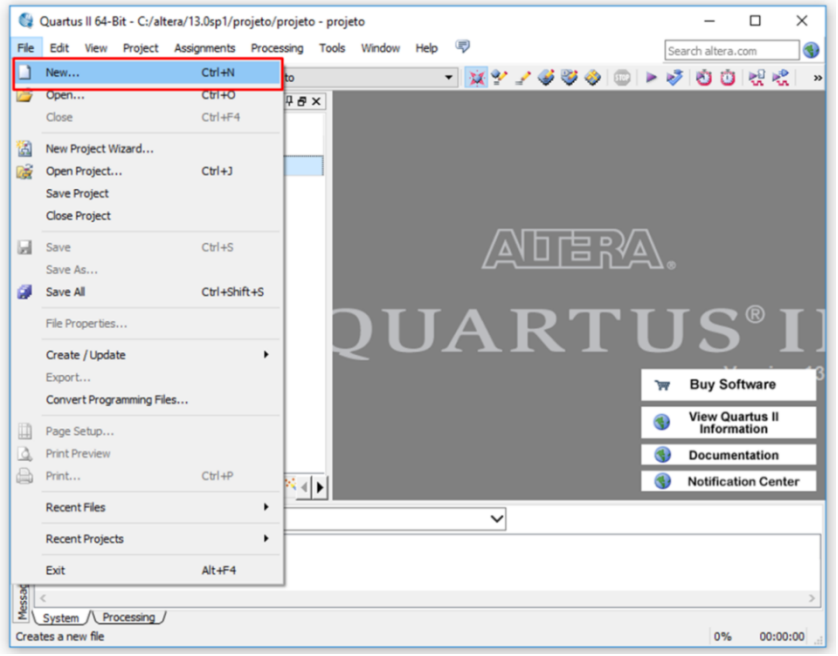
Figura 11 – Seleção do FPGA.



Fonte: Captura de tela do *software* Quartus II.

Após a finalização, cria-se um novo arquivo de texto para que o projeto seja descrito (Figura 12).

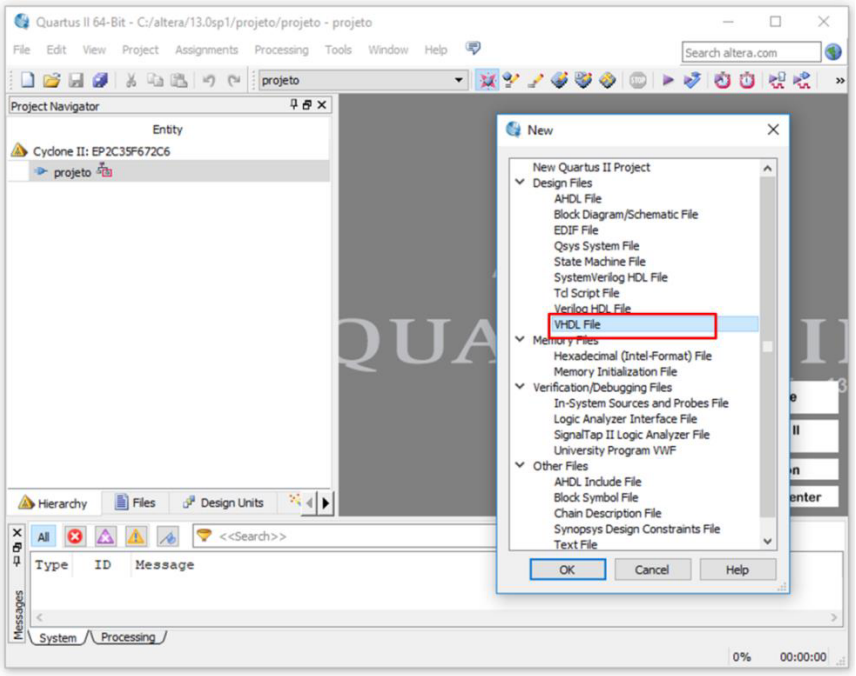
Figura 12 – Criação de novo arquivo de texto.



Fonte: Captura de tela do *software* Quartus II.

Por fim, seleciona-se a opção de arquivo correspondente à linguagem na qual o projeto será descrito (Figura 13).

Figura 13 – Seleção da linguagem na qual o projeto será descrito.



Fonte: Captura de tela do *software* Quartus II.

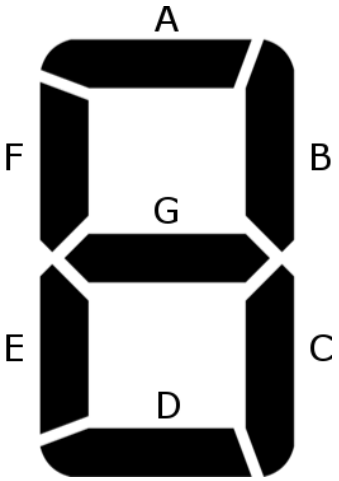
6.1.1 Entrada do projeto para exibição da frequência no *display* de sete segmentos

*Display*s de sete segmentos são comumente utilizados em diversos tipos de produtos. Tais *displays* são usados com circuitos lógicos decodificadores de números binários ativando os dígitos necessários no *display*. Cada segmento, se energizado de forma padronizada, pode em conjunto mostrar cada um dos dez dígitos no *display*. Um tipo de *display* de 7 segmentos comum usa diodos emissores de luz (LED - *light emitting diodes*); e cada segmento é um LED que, ao passar uma corrente através dele, é ativado emitindo luz.

Há dois tipos de *display*s de 7 segmentos: catodo comum e anodo comum. O arranjo anodo comum exige que o circuito acionado dê uma tensão de nível baixo para ativar certo segmento. Quando esse segmento tem um nível baixo nele aplicado, o LED é ligado. O arranjo catodo comum exige que o segmento seja acionado com uma tensão de nível alto. Quando no segmento é aplicado um nível alto de tensão o LED é ligado [6]. No *kit* Altera DE2 há 8 *display*s anodo comum.

Por padrão, os segmentos do *display* de sete segmentos (A, B, C, D, E, F, G) são rotulados conforme o diagrama na Figura 14.

Figura 14 – Rótulo dos sete segmentos do *display.*



Fonte: Elaborada pelos autores.

A Tabela 2 indica o nível lógico de cada segmento do display para exibição dos algarismos de 0 a 9 no arranjo anodo comum.

Tabela 2 – Pinagem do *Display* de 7 segmentos.

Caractere	A	B	C	D	E	F	G
0	0	0	0	0	0	0	1
1	1	0	0	1	1	1	1
2	1	0	0	1	1	1	1

Caractere	A	B	C	D	E	F	G
3	0	0	0	0	1	1	0
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	0
6	0	1	0	0	0	0	0
7	0	0	0	1	1	1	1
8	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0

Fonte: Elaborada pelos autores.

A Figura 15 exibe a entidade descrita para o projeto no *software* Quartus II. No comando "entity" são descritas as entradas e saídas do projeto. A porta "clock50MHz" representa a entrada do *clock* de 50 MHz do *kit*. A porta "sinal" representa a entrada do sinal cuja frequência será lida (pino 0 da GPIO-0). As portas "HEX3", "HEX2", "HEX1" e "HEX0" representam a saída para apresentação do dado no *display* de 7 segmentos. Os sinais da saída foram nomeados com o nome do rótulo de cada *display* da placa. A unidade da frequência lida é hertz.

Figura 15 – Entidade do frequencímetro.

```
ENTITY tacometro_7seg IS
      PORT
      (
            clock50Mhz: IN STD_LOGIC;
            sinal: IN STD_LOGIC;
            HEX3,HEX2,HEX1,HEX0 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0)
       );
END tacometro_7seg;
```

Fonte: Elaborada pelos autores.

Após a declaração dos sinais a serem utilizados no código, na arquitetura (comando "architecture"), o circuito é descrito. Esta reúne o conjunto de processos a fim de executar a tarefa desejada.

Foi implementado um processo para criar um *clock* de 1 Hz a partir do *clock* de entrada de 50 MHz (Figura 16). O sinal de *clock* comuta entre os estados ALTO e BAIXO em intervalos fixos.

A variável "prescaler" foi iniciada com 25.000.000 em binário.

A variável "contador" foi iniciada com zero e incrementada em uma unidade a cada transição de nível baixo para alto do *clock* de 50 MHz até que fosse maior que "prescaler". Ao

atingir 25.000.001, o sinal “Contador” recebe zero e o sinal “clock1Hz” alterna entre os estados alto e baixo a cada 500 milissegundos, criando assim o *clock* de 1 Hz.

Figura 16 – Processo para obtenção de *clock* de 1 Hz.

```
PROCESS(clock50Mhz, clock1Hz)
BEGIN
      IF RISING_EDGE(clock50Mhz) THEN
            contador <= contador + 1;
            IF(contador > prescaler) THEN
                  clock1Hz <= NOT clock1Hz;
                  contador <= (OTHERS => '0');
            END IF;
      END IF;
END PROCESS;
```

Fonte: Elaborada pelos autores.

No processo posterior (Figura 17), realiza-se a contagem da frequência e decomposição dos algarismos correspondentes a unidade de milhar, centena, dezena e unidade a cada mudança de estado do *clock* de 1 Hz. A variável “Freq1” é incrementada a cada transição de nível alto para baixo do sinal de entrada e a variável “Freq2” a cada transição de nível baixo para alto quando o sinal “clock1Hz” é zero. Ambas, “Freq1” e “Freq2”, são somadas e armazenadas na variável “FreqSoma”.

Quando “clock1Hz” igual a um, a variável “frequencia” recebe “FreqSoma” e “Freq1” e “Freq2” recebem zero.

Figura 17 – Processo para contagem da frequência e decomposição dos algarismos.

```
PROCESS (clock1Hz, sinal)
      BEGIN
            IF clock1Hz = '0' THEN
                  IF (sinal' EVENT AND sinal = '0') THEN
                        Freq1 <= Freq1 + 1;
                  ELSIF (sinal' EVENT AND sinal = '1') THEN
                        Freq2 <= Freq2 + 1;
                  END IF;

                  FreqSoma <= Freq1 + Freq2;

                  ELSIF clock1Hz = '1' THEN
                        Freq1 <= 0;
                        Freq2 <= 0;
                        frequencia <= FreqSoma;
            END IF;
            restoM  <= frequencia mod 1000;
            M  <= (frequencia - restoM)/1000;
            restoC <= restoM mod 100;
            C <= (restoM - restoC)/100;
            restoD <= restoC mod 10;
            D <= (restoC - restoD)/10;
            U <= restoD;
```

```
END PROCESS;
```

Fonte: Elaborada pelos autores.

Ainda no processo da Figura 17, para determinar o algarismo correspondente à unidade de milhar (“M”), utiliza-se o operador “mod” para encontrar o resto da divisão da frequência lida por 1000 (armazenado em “restoM”). Este resto é subtraído da frequência, dividido por mil e o valor armazenado na variável “M”.

Para o algarismo da centena (“C”), encontra-se o resto da divisão de “restoM” por 100 (armazenado em “restoC”), subtrai-se o resto de “restoM”, divide-se o resultado por 100, e armazena-se na variável “C”.

Para o algarismo da dezena (“D”), encontra-se o resto da divisão de “restoC” por 10 (armazenado em “restoD”), subtrai-se o resto de “restoC”, divide-se o resultado por 10, e armazena-se na variável “D”.

O resto da dezena (“restoD”) corresponde à unidade “U”.

Com os algarismos é possível acionar o *display* de sete segmentos de anodo comum disponível no *kit*.

No código principal, o componente da Figura 18 é acionado para a unidade de milhar, centena, dezena e unidade conforme a Tabela 2.

Figura 18 – Componente para exibição do caractere no *display* de sete segmentos.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY digito IS
      PORT
      (
      INT         : IN NATURAL RANGE 0 TO 9;
      HEX  : OUT STD_LOGIC_VECTOR (6 DOWNTO 0)
      );
END digito;

ARCHITECTURE a OF digito IS
BEGIN
      WITH INT SELECT

             HEX <= "0000001" WHEN 0,
                          "1001111" WHEN 1,
                          "0010010" WHEN 2,
                          "0000110" WHEN 3,
                          "1001100" WHEN 4,
                          "0100100" WHEN 5,
                          "0100000" WHEN 6,
                          "0001111" WHEN 7,
                          "0000000" WHEN 8,
                          "0000100" WHEN 9;
END a;
```

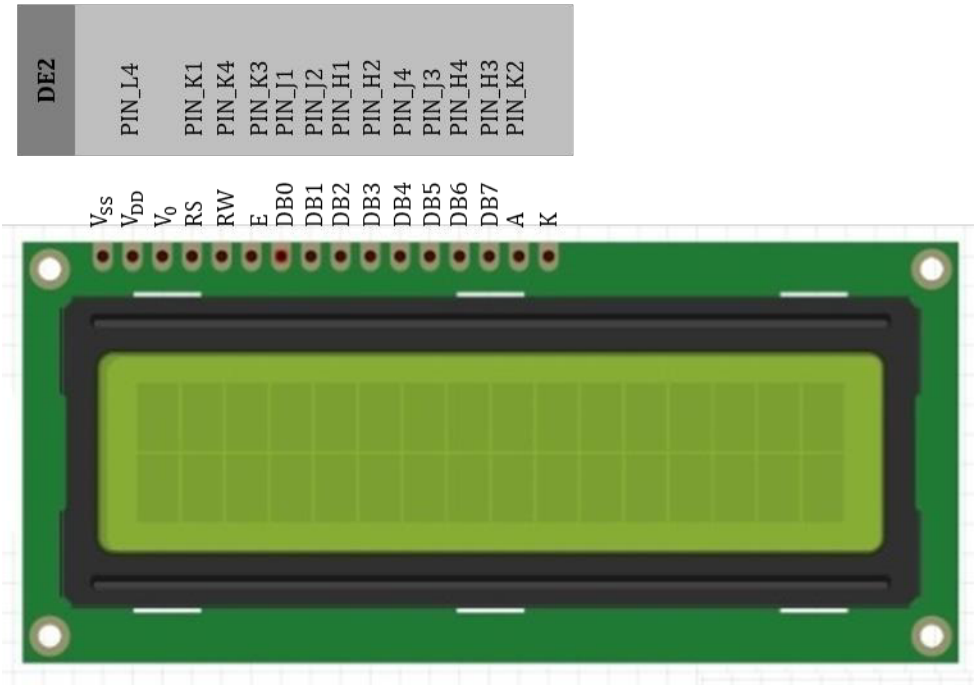
Fonte: Elaborada pelos autores

6.1.2 Entrada do projeto para exibição da frequência no *display* LCD

O módulo LCD modelo CFAH1602B-TMC-JP disponível no *kit* DE2 possui 640 *pixels* (duas linhas com 16 caracteres por linha). Esse dispositivo contêm um conjunto de caracteres ASCII *standard* e ainda caracteres japoneses, gregos e símbolos matemáticos. Também é possível ligar cada *pixel* individualmente.

Os 16 pinos do módulo devem fazer interface com o FPGA e são configurados no código em VHDL para exibição dos dados lidos.

Figura 19 – Pinos padrão do LCD e rótulo correspondente na placa DE2



Fonte: Captura de tela do *software* Fritzing com adaptações.

A descrição de cada pino está disposta na tabela Tabela 3.

Tabela 3 – Pinagem do módulo LCD.

Pino	Símbolo	Nível	Descrição
1	V_{SS}	0V	Terra
2	V_{DD}	5,0V	Alimentação
3	Vo	Variável	Tensão para ajuste de contraste
4	RS	H/L	H –Dado, L – Instrução
5	RW	H/L	1 – Leitura, 0 – Escrita
6	E	H,H→L	Sinal de habilitação do chip.
7	$DB0$	H/L	Dado 0

Pino	Símbolo	Nível	Descrição
8	$DB1$	H/L	Dado 1
9	$DB2$	H/L	Dado 2
10	$DB3$	H/L	Dado 3
11	$DB4$	H/L	Dado 4
12	$DB5$	H/L	Dado 5
13	$DB6$	H/L	Dado 6
14	$DB7$	H/L	Dado 7
15	A	-	Anodo para LED *backlight*
16	K	-	Catodo para LED *backlight*

Fonte: [19].

O pino RS (4) seleciona registros a fim de interpretar o tipo de dados no barramento. Se o pino estiver no nível lógico baixo (0 ou "L"), os dados enviados ao dispositivo serão lidos como instrução. Em nível alto (1 ou "H"), os dados lidos/escritos serão tratados como caracteres.

O pino R/W (5) indica o sentido dos dados entre a placa e o LCD. Em nível baixo, habilita-se a escrita de dados no LCD. Em nível alto, habilita-se a leitura de dados do LCD.

O pino E (6) ativa leitura/escrita de dados. Quando este pino está em nível baixo, o LCD é desabilitado. Em nível alto, o dispositivo verifica o estado dos pinos RS e R/W e executa as instruções.

Os pinos DB0 a DB7 (7 a 14) compõem o barramento de dados. Estes oito sinais trabalham em paralelo e podem ser transferidos ou recebidos entre a placa e o LCD.

Na entidade descrita para o projeto (Figura 20), o barramento de dados foi representado por "lcd_data_0" a "lcd_data_7". Os pinos RS (lcd_rs), E (lcd_e), R/W (lcd_rw), V_{DD} (lcd_on), e V_0 (lcd_blon) foram mapeados como saídas. Como entrada, além das utilizadas do projeto anterior ("sinal" representa o sinal cuja frequência será lida do pino 0 da GPIO-0 e "clock50MHz" corresponde ao *clock* de 50 MHz disponível no *kit*), foram incluídas as chaves SW0, SW1, SW2, SW3, para seleção da unidade a ser exibida no *display*.

Figura 20 – Entidade do projeto.

```
ENTITY tacometro_lcd IS

      PORT(
      reset              : IN     std_logic;
      clock50MHz         : IN     std_logic;
      sinal              : IN     std_logic;
```

```
      lcd_rs              : OUT    std_logic;
      lcd_e               : OUT    std_logic;
      lcd_rw              : OUT    std_logic;
      lcd_on              : OUT    std_logic;
      lcd_blon            : OUT    std_logic;

      lcd_data_0          : INOUT  STD_LOGIC;
      lcd_data_1          : INOUT  STD_LOGIC;
      lcd_data_2          : INOUT  STD_LOGIC;
      lcd_data_3          : INOUT  STD_LOGIC;
      lcd_data_4          : INOUT  STD_LOGIC;
      lcd_data_5          : INOUT  STD_LOGIC;
      lcd_data_6          : INOUT  STD_LOGIC;
      lcd_data_7          : INOUT  STD_LOGIC;

      SW0    : IN    STD_LOGIC;
      SW1    : IN    STD_LOGIC;
      SW2    : IN    STD_LOGIC;
      SW3    : IN    STD_LOGIC

   );
END tacometro_lcd ;
```

Fonte: Elaborada pelos autores.

O processo para contagem da frequência e decomposição dos algarismos é semelhante ao do código para exibição da frequência no *display* de sete segmentos. Mas, neste projeto, incluiu-se a dezena e centena de milhar e a unidade de milhão (Figura 21).

Figura 21 – Processo para contagem da frequência e separação dos algarismos do projeto para apresentação da frequência lida no display LCD.

```
PROCESS (clock1Hz, sinal)
      BEGIN

            IF clock1Hz = '0' THEN
                  IF (sinal' EVENT AND sinal = '0') THEN
                        Freq1 <= Freq1 + 1;

                  ELSIF (sinal' EVENT AND sinal = '1') THEN
                        Freq2 <= Freq2 + 1;
                  END IF;
                        FreqSoma <= Freq1 + Freq2;
                  ELSIF clock1Hz = '1' THEN
                        IF (SW2 = '1') OR (SW3 = '1') THEN
                        frequencia <= FreqSoma*60;
                  ELSE frequencia <= FreqSoma;
                  END IF;
                  Freq1 <= 0;
                  Freq2 <= 0;
            END IF;

            restoM3  <= frequencia mod 1000000;
            M3  <= (frequencia - restoM3)/1000000; -- Unidade de milhão
            restoM2  <= restoM3 mod 100000;
            M2  <= (restoM3 - restoM2)/100000; -- Centena de milhar
            restoM1  <= restoM2 mod 10000;
```

```
            M1  <= (restoM2 - restoM1)/10000; -- Dezena de milhar
            restoM  <= restoM1 mod 1000;
            M  <= (restoM1 - restoM)/1000;
            restoC <= restoM mod 100;
            C <= (restoM - restoC)/100;
            restoD <= restoC mod 10;
            D <= (restoC - restoD)/10;
            U <= restoD;

END PROCESS;
```

Fonte: Elaborada pelos autores.

Para deixar o LCD operacional, é necessário implementar uma rotina de inicialização. A folha de dados do dispositivo contém a lista de comandos para inicialização e configuração. Essa lista é exibida no ANEXO A.

Para inicialização do módulo foi implementada uma máquina de estados [20] seguindo os passos determinados na folha de dados do dispositivo [19]:

a) *function set*: o barramento de dados recebe o valor 38 em hexadecimal. Este comando corresponde à soma das instruções para utilizar 8 bits no barramento de dados (DL=1), 2 linhas do display (N=1) e fonte 5x7 pontos (F=0);
b) *display off control*: o *display* é desligado (D=0), o cursor desativado (C=0), desativa o piscar do cursor (B=0). Assim, no código, o barramento de dados recebe o valor 8 em hexadecimal para executar a instrução supracitada;
c) *clear display*: limpa o *display* e retorna o cursor para a primeira posição da primeira linha. Nesta etapa, o barramento de dados recebe o valor 1 em hexadecimal;
d) *display on control*: o *display* é ligado (D=1), o cursor desativado (C=0), desativa o piscar do cursor (B=0). Para essa instrução o barramento de dados recebe o valor 0C em hexadecimal;
e) *entry mode set*: define que o cursor se movimentará para a direita (I/D=1) e desativa o deslocamento automático (S=0). Para isso, o barramento de dados recebe o valor 6 em hexadecimal.

O primeiro estado após a rotina de inicialização escreve os caracteres no *display*. Cada caractere é predefinido como um número hexadecimal. Dependendo da chave ativada é selecionada uma cadeia de caracteres específica: cada um dos 16 caracteres da primeira linha é formado por um conjunto de 8 *bits* enviados ao barramento, um *byte* de cada vez; a exibição final dos 16 caracteres tem a aparência de serem simultâneos, apesar de serem exibidos um de cada vez, permanecendo na tela por aproximadamente 2,5 ms.

Além disso, há um laço no qual, dependendo da chave ativada, é possível alterar a unidade da frequência exibida. Ou seja, a escolha da unidade determina a cadeia de caracteres a ser exibida no *display*.

Foi criado um vetor com o valor hexadecimal correspondente aos caracteres 0 a 9. O número a ser exibido é igual a sua posição neste vetor.

Por padrão, a frequência é exibida em hertz (Hz). Ativando apenas a chave 1 (SW1), a frequência é exibida em KHz, ativando a chave 2 (SW2), em rotações por minuto (rpm) e ligando a chave 3 (SW3), em min^{-1}. A Figura 22 exibe o laço contido no estado que escreve o caractere no *display*, quando a chave desejada é ativada, a variável "next_char" recebe um caractere por vez da cadeia correspondente a chave escolhida. A variável "char_count" realiza a contagem do caractere a ser enviado ao barramento de dados (a variável "next_char" é atribuída a "lcd_data_valor").

Figura 22 – Laço para envio dos caracteres para o barramento de dados.

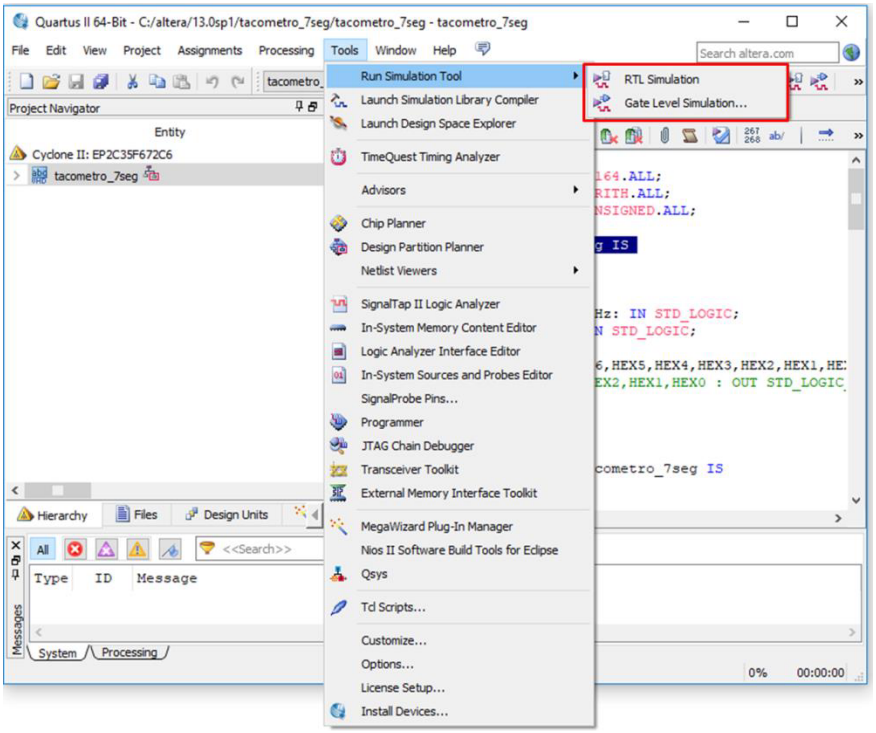
```
PROCESS (SW)
   BEGIN
   CASE (SW) IS
      WHEN "0010" =>
      -- SW1 = 1, frequencia em KHz
      next_char <= string1 (CONV_INTEGER(char_count));
      WHEN "0100" =>
      -- SW2 = 1, frequencia em rpm
      next_char <= string2 (CONV_INTEGER(char_count));
      WHEN "1000" =>
      -- SW3 = 1, frequencia em min^-1
      next_char <= string3(CONV_INTEGER(char_count));
      WHEN OTHERS =>
      -- Outro caso, frequencia em Hz
      next_char <= string0(CONV_INTEGER(char_count));
   END CASE;
END PROCESS;
```

Fonte: Elaborada pelos autores.

6.2 Passo-a-passo para execução de simulações

O primeiro passo para execução da simulação é selecionar qual a simulação desejada no programa Quartus II (Figura 23). No menu "Tools", item "Run Simulation Tool" há duas opções de simulação: "RTL Simulation" (funcional) e "Gate Level Simulation" (temporal). O procedimento descrito a seguir é análogo para ambas.

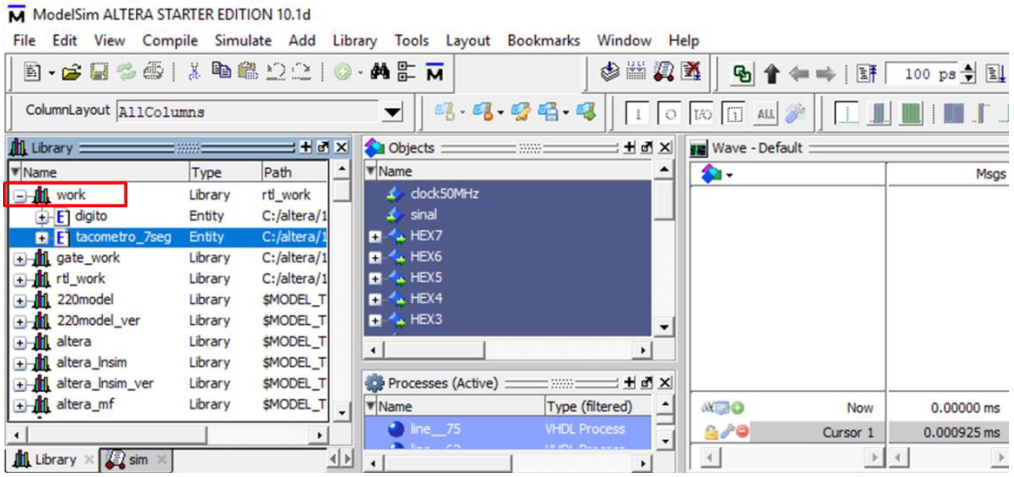
Figura 23 – Seleção da ferramenta de simulação do software Quartus II



Fonte: Captura de tela do *software* Quartus II.

Após a seleção da simulação desejada, será aberto automaticamente o ModelSim e o segundo passo é selecionar o projeto a ser simulado na pasta “work” (Figura 24).

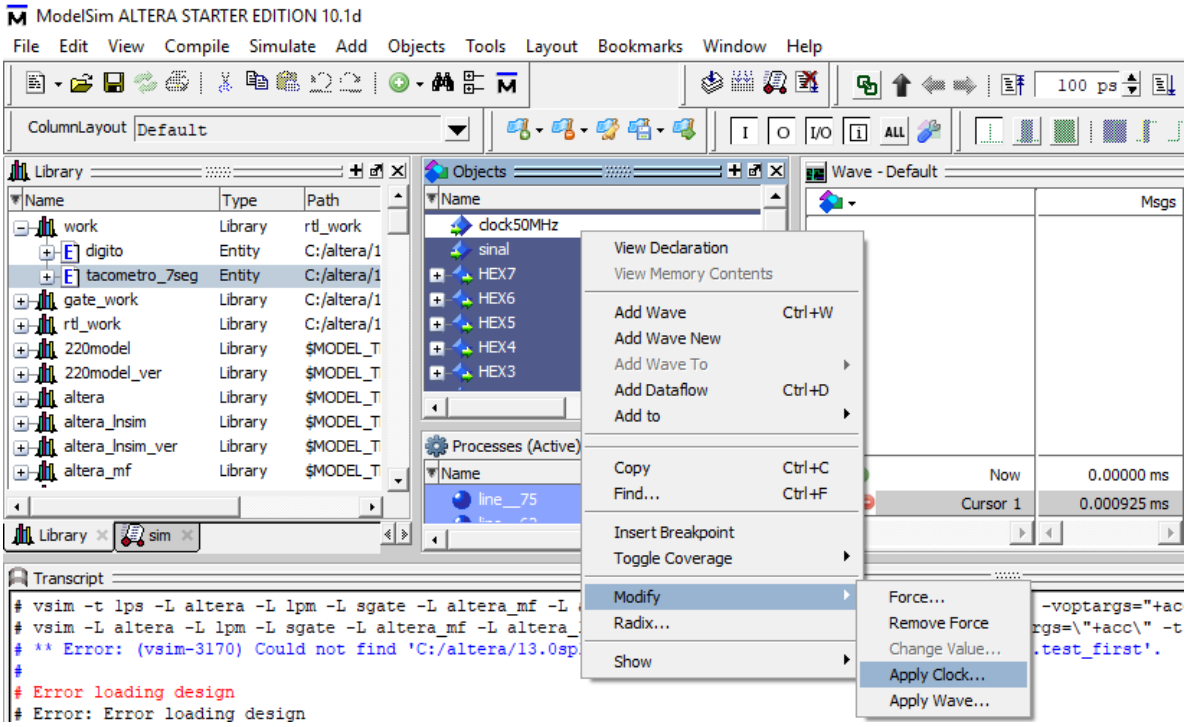
Figura 24 – Seleção do projeto a ser simulado no ModelSim.



Fonte: Captura de tela do *software* ModelSim.

No terceiro passo, configuram-se os sinais de entrada clicando com o botão direito em cada variável na janela “Objects” (Figura 25).

Figura 25 – Configuração das entradas do projeto.

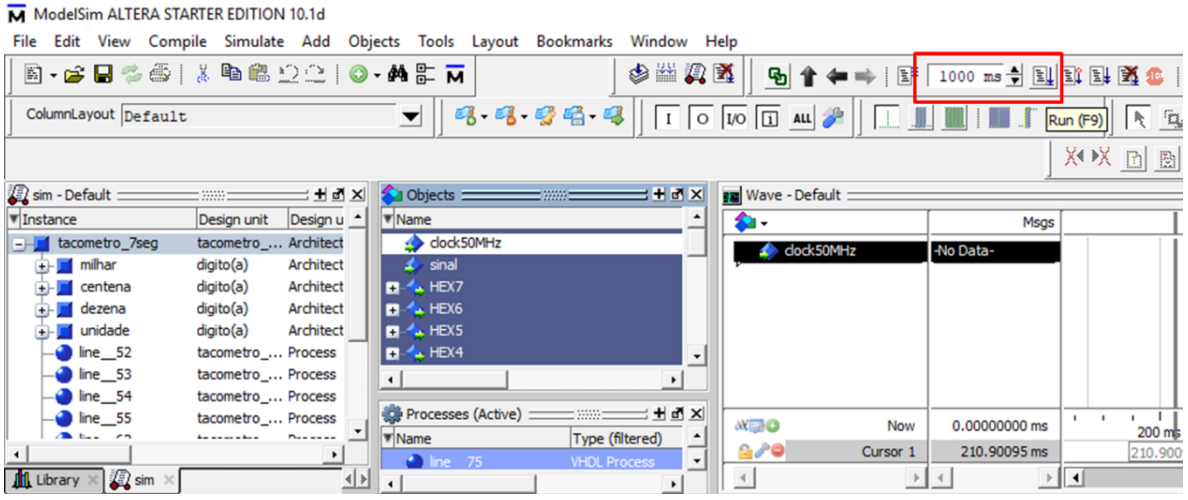


Fonte: Captura de tela do *software* ModelSim.

No quarto passo, incluem-se os sinais cujo comportamento o usuário deseja visualizar na janela “Wave”. Esse procedimento pode ser executado simplesmente selecionando e arrastado os sinais desejados para o primeiro campo da janela “Wave” ou selecionando a apção “Add Wave” após clicar com o botão direito na variável de interesse (Figura 25).

O penúltimo passo é a escolha do período a ser simulado. Na barra de menus, o usuário pode determinar o intervalo a ser simulado e por fim executar a simulação no ícone ao lado do campo de entrada do período (Figura 26).

Figura 26 – Entrada do período de simulação .



Fonte: Captura de tela do *software* ModelSim.

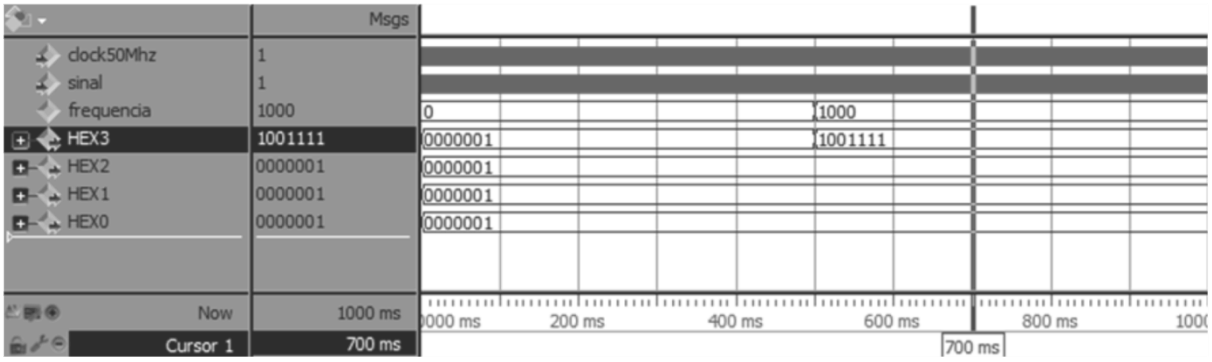
7 RESULTADOS E DISCUSSÃO

7.1 Exibição da frequência no *display* de sete segmentos

7.1.1 Simulação Funcional

O passo seguinte foi a execução da simulação funcional. O sinal de entrada tem frequência igual a 1000 Hz. A Figura 27 mostra as forma de onda de simulação para o projeto.

Figura 27 – Formas de onda da simulação funcional para o *display* de 7 segmentos.



Fonte: Captura de tela do *software* Modelsim.

Na Figura 27, observa-se que o valor da frequência lida ("frequencia") e os caracteres exibidos no *display* ("HEX3", "HEX2", "HEX1" e "HEX0") correspondem ao esperado. Antes de 500 ms, a frequência é nula e todos os *displays* recebem o vetor de bits equivalente ao caractere "0". Após 500 milissegundos, o *display* "HEX3" recebe o vetor correspondente ao caractere "1" e os demais permaneceram com o caractere "0".

7.1.2 Síntese

A etapa seguinte foi a execução da síntese. A Figura 28 exibe o relatório de análise e síntese do projeto.

Figura 28 – Relatório de síntese no Quartus II do projeto para apresentação da frequência lida no *display* de 7 segmentos.

Analysis & Synthesis Summary	
Analysis & Synthesis Status	Successful - Mon Jun 05 16:19:37 2017
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	tacometro_7seg
Top-level Entity Name	tacometro_7seg
Family	Cyclone II
Total logic elements	7,230
Total combinational functions	7,230
Dedicated logic registers	90
Total registers	90
Total pins	30
Total virtual pins	0
Total memory bits	0
Embedded Multiplier 9-bit elements	0
Total PLLs	0

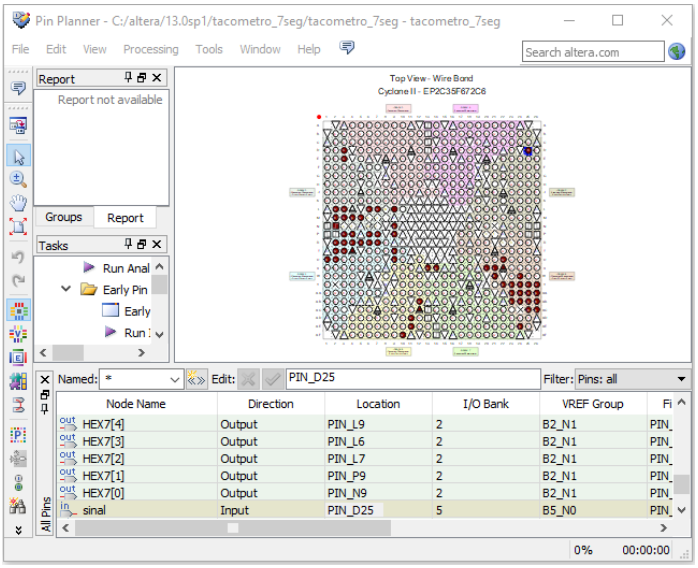
Fonte: Captura de tela do *software* Quartus.

O relatório de síntese indica a utilização de 7230 elementos lógicos, o equivalente a 22% dos elementos da placa. Além disso, 30 pinos são necessários para implementar o circuito, que correspondem a 6% dos pinos disponíveis no *kit*.

7.1.3 Implementação

O fabricante disponibiliza o quadro com a localização de todos os pinos da placa, além de especificar o nome do pino/sinal e sua função. Com auxílio desse documento, atribui-se a localização desejada para as portas declaradas no código na janela *Pin Planner* (Figura 29).

Figura 29 – Designação das portas declaradas no código para os pinos do FPGA.



Fonte: Captura de tela da janela "*Pin Planner*" do *software* Quartus II.

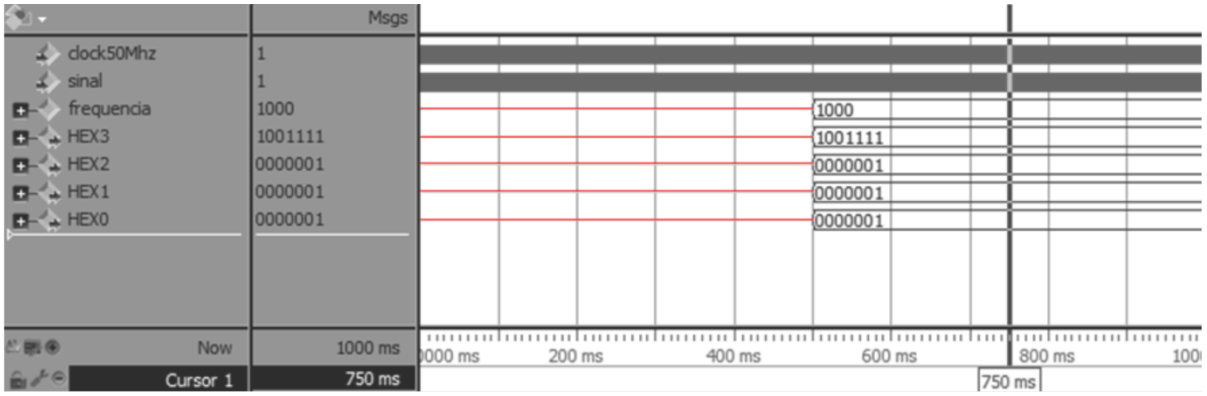
Os pinos foram configurados conforme APÊNDICE B. O referido apêndice exibe o quadro com atribuição de pinos do projeto para exibição da frequência no *display* de 7 segmentos.

Após o mapeamento dos pinos, realiza-se a compilação do código novamente.

7.1.4 Simulação temporal

No passo seguinte, executou-se a simulação temporal para o sinal de entrada de 1000 Hz a fim de verificar o comportamento do circuito projetado no dispositivo. O mesmo resultado da simulação funcional é obtido após 500 ms, que corresponde ao período de contagem da frequência (Figura 30). As linhas em vermelho, no período ente 0 e 500 ms, representam a indefinição do valor dos sinais “frequência”, “HEX3”, “HEX2”, “HEX1” e “HEX0” neste intervalo.

Figura 30 – Formas de onda da simulação temporal para sinal de entrada de 1000 Hz.

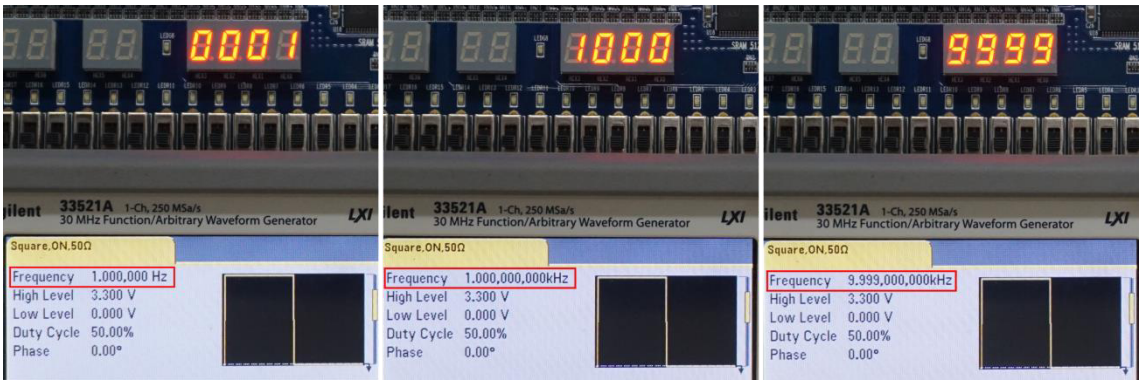


Fonte: Captura de tela do *software* Quartus.

7.1.5 Download

Efetua-se a transferência do projeto à placa por meio do cabo USB. Após o *download* para a placa do projeto para a exibição da frequência no *display* de sete segmentos, o gerador de sinais foi utilizado para testes. A Figura 31exibe os testes para o sinal de entrada com frequência de 1 Hz, 1000 Hz e 9999 Hz.

Figura 31 – Teste com gerador de sinais.



Fonte: Elaborada pelos autores.

Conforme observado na Figura 31, os resultados obtidos após 500 ms no *hardware* foram tão satisfatórios quanto nas simulações. O valor de frequência selecionado no gerador de frequência é o mesmo apresentado no *kit*.

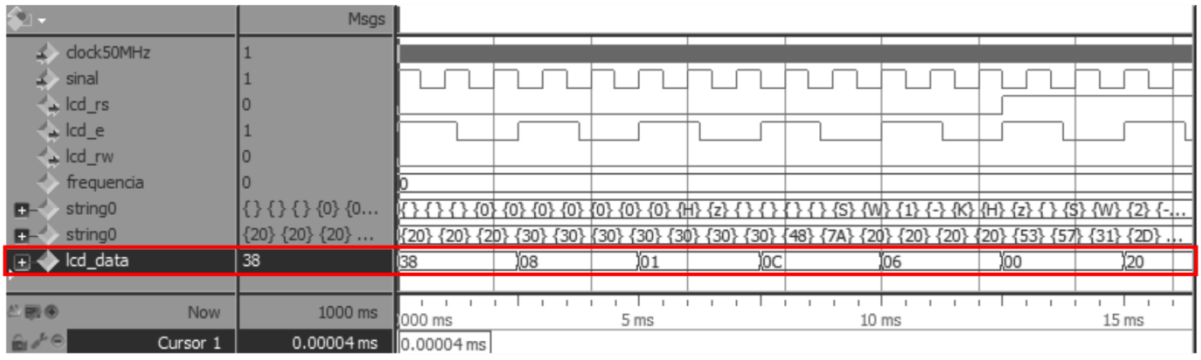
Nos primeiros 500 ms, os *displays* exibem números aleatórios, como previsto na simulação temporal (Figura 30) em vez de zeros como exposto na simulação funcional (Figura 27).

7.2 Exibição da frequência no *display* LCD

7.2.2 Simulação Funcional

No trecho ampliado da simulação funcional apresentado na Figura 32, é possível observar, na região destacada, a rotina de inicialização no barramento de dados (“lcd_data”). Neste período, o sinal “lcd_rs” está em nível baixo, visto que os dados enviados foram lidos como instrução.

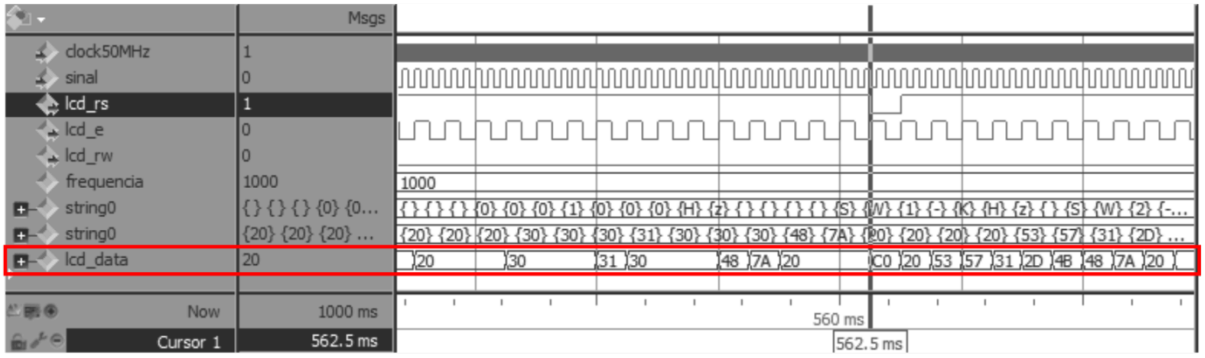
Figura 32 – Formas de onda da simulação funcional da rotina de inicialização do projeto para apresentação da frequência lida no *display LCD*.



Fonte: Captura de tela do *software* Modelsim.

Na Figura 33, é possível perceber que a cadeia de caracteres (“string0”) com a frequência lida é enviada ao barramento de dados (lcd_data).

Figura 33 – Formas de onda da simulação funcional após 500 milissegundos do projeto para apresentação da frequência lida no *display LCD*.



Fonte: Captura de tela do *software* Modelsim.

7.2.3 Síntese

Após a simulação funcional foi executada a síntese. A Figura 34 exibe o relatório de análise e síntese do projeto.

Figura 34 – Relatório de síntese do projeto para apresentação da frequência lida no *display LCD*.

Analysis & Synthesis Summary	
Analysis & Synthesis Status	Successful - Thu Jun 08 00:15:30 2017
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	tacometro_lcd
Top-level Entity Name	tacometro_lcd
Family	Cyclone II
Total logic elements	14,368
Total combinational functions	14,366
Dedicated logic registers	159
Total registers	159
Total pins	20
Total virtual pins	0
Total memory bits	0
Embedded Multiplier 9-bit elements	0
Total PLLs	0

Fonte: Captura de tela do *software* Quartus.

O relatório de síntese indica que foram utilizados 14369 elementos lógicos, o equivalente a 43% dos elementos da placa, e que 20 pinos são necessários para implementar o circuito, que correspondem a 4% dos pinos disponíveis no *kit*.

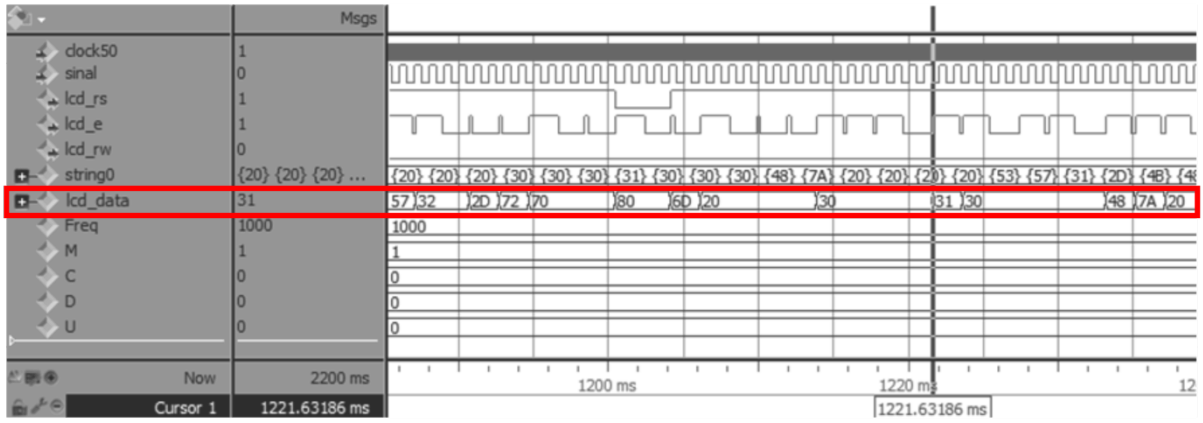
7.2.4 Implementação

Após a síntese, foi possível implementar o projeto, ou seja, os pinos foram configurados conforme APÊNDICE D – Atribuição de pinos do projeto para exibição da frequência no *display* LCD. O referido apêndice exibe o quadro com atribuição de pinos do projeto para exibição da frequência no *display* LCD.

7.2.5 Simulação temporal

Após a implementação, executou-se a simulação temporal (Figura 35) para verificar o comportamento do circuito projetado no dispositivo. No caso ilustrado, foi atribuída à entrada "sinal", cuja frequência será lida, um sinal de 1000 Hz e a "clock50" um sinal de 50MHz.

Figura 35 – Simulação temporal do projeto para apresentação da frequência lida no *display LCD*.



Fonte: Captura de tela do *software* Modelsim.

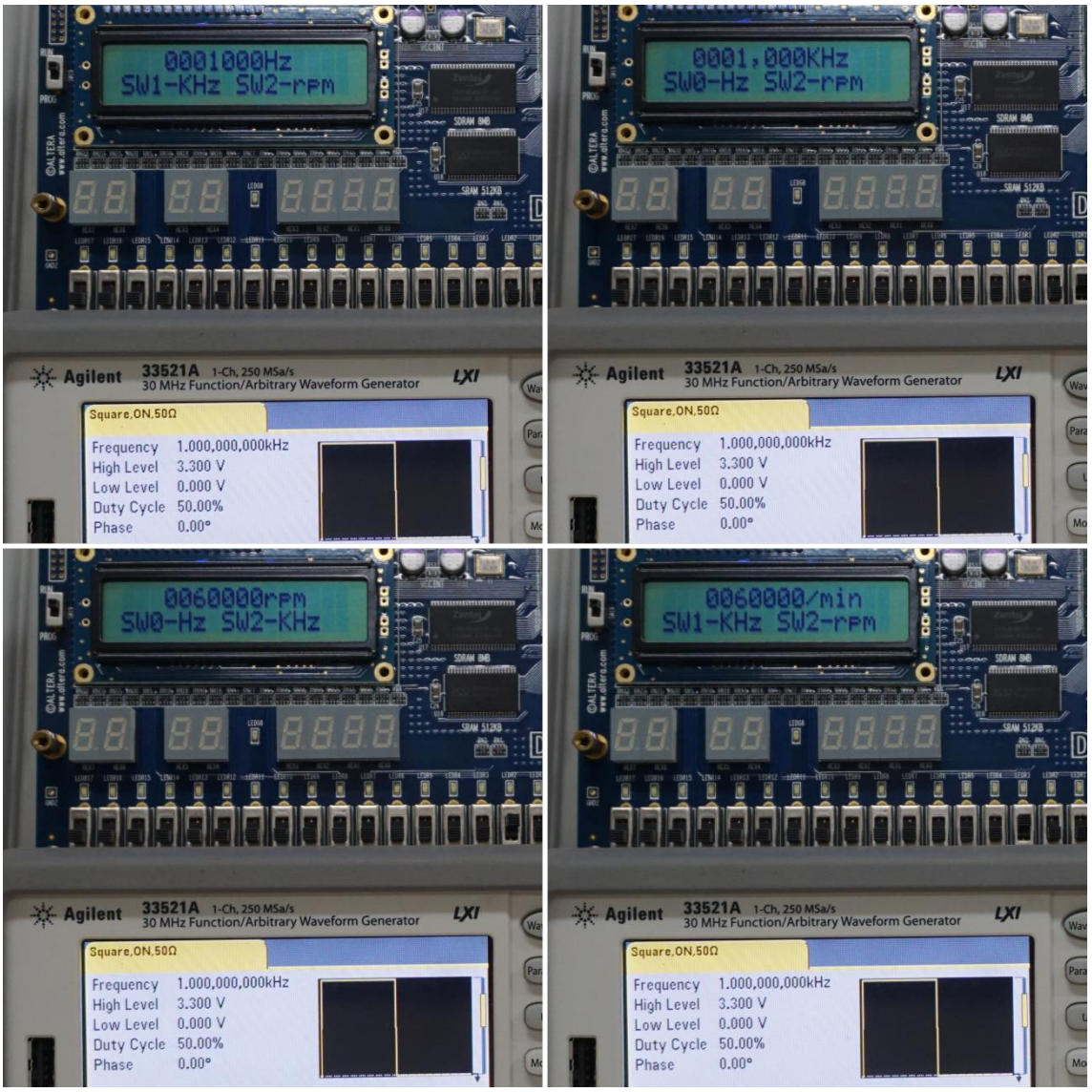
Nas formas de onda da simulação de temporização da Figura 35, a frequência lida ("Freq") foi 1000 Hz, conforme o esperado. O sinal "string0" exibe a cadeia de caracteres a ser enviada ao barramento de dados. É possível visualizar o envio sequencial dos caracteres ao barramento de dados (lcd_data).

Na "string0" e "lcd_data", os dados são apresentados em hexadecimal. No display LCD, há um número em hexadecimal correspondente a cada caractere. No caso da Figura 35, por exemplo, o número 20 em hexadecimal corresponde a um espaço em branco, 30 ao caractere "0", 31 ao caractere "1", 48 ao caractere "H" e 7A ao "z". Assim, na tela será exibida a cadeia de caracteres "1000 Hz" na primeira linha.

7.2.6 Download

A Figura 36 apresenta os testes para o sinal de entrada com frequência de 1000 Hz do projeto para a exibição da frequência no *display* LCD. O valor esperado, caso ativada apenas a chave SW1 é 1,000 KHz, para SW2 60000 rpm, para SW3 60000 min^{-1}. Se as chaves estiverem desativadas ou ativadas simultaneamente, será exibida a frequência em hertz.

Figura 36 –Teste com gerador de sinais para sinal de entrada igual a 1000 Hz.



Fonte: Elaborada pelos autores.

Conforme observado na Figura 31e Figura 36, os resultados obtidos no *hardware* após 500 ms foram tão satisfatórios quanto nas simulações funcional e temporal.

8 CONSIDERAÇÕES FINAIS

Esta aplicação simples possibilitou aprendizagem da linguagem VHDL.

Usando a linguagem VHDL para interfacear a placa de desenvolvimento do *kit* DE2 da Altera© (onde um dos componentes da mesma é um circuito integrado programável FPGA) foi acoplado o circuito implementado com *hardware* externo de emissão de sinais de ondas, sendo possível registrar, de modo satisfatório, o fluxo de projeto da implementação de um instrumento para monitoramento de frequência.

A resposta do sistema real (mostrado no *display* de sete segmentos para valores de frequência entre 0 e 9999 Hz ou no visor LCD do *kit*) é suficientemente rápida para leitura da frequência do sinal, necessitando de meio segundo para ser exibida. O tempo de 0,5 s equivale ao período de contagem da frequência pelo *kit* DE2.

Assim, os projetos em VHDL alcançaram os resultados propostos. Lembrando que a análise de frequência é apenas um dos usos suportados pelo *kit* da Altera DE2. Outras aplicabilidades em FPGA podem ser conseguidas acessando os outros barramentos de entrada e saída disponíveis na placa.

E tendo em vista o mercado nacional, o estudo de FPGA em sistemas embarcados apresenta um cenário propício para o desenvolvimento desta tecnologia e divulgação da mesma na região norte do país. Mostrando a relevância do presente trabalho para linguagem de descrição de hardware VHDL em conjunto com a tecnologia FPGA.

REFERÊNCIAS

1. INTEL CORPORATION. Intel FPGA Solutions, 2017. Disponivel em: <https://www.altera.com/solutions/industry.html>. Acesso em: 20 Janeiro 2017.

2. XILINX INCORPORATION. Applications, 2017. Disponivel em: <https://www.xilinx.com/applications.html>. Acesso em: 20 Janeiro 2017.

3. EMBARCADOS. Análise do Mercado de Trabalho de Sistemas Embarcados no Brasil 2015. **Embarcados**, 2015. Disponivel em: <https://www.embarcados.com.br/editorial-analise-do-mercado-de-trabalho-de-sistemas-embarcados-2015/>. Acesso em: 30 Setembro 2018.

4. TOCCI, R. J.; WIDMER, N. S. **Sistemas digitais:** princípios e aplicações. Rio de Janeiro: LTC, 2000.

5. HÄRMMELI, F. D. **Desenvolvimento de um analisador de elementos de rede baseado no padrão gigabit ethernet**. Pontifícia Universidade Católica do Rio de Janeiro. Rio de Janeiro. 2008.

6. FLOYD, T. L. **Sistemas Digitais:** Fundamentos e Aplicações. 9. ed. Porto Alegre: Bookman, 2007.

7. JÄHNE, B.; HAUSSECKER, H.; GEISSLER, P. **Handbook of Computer Vision and Applications:** Systems and Applications. San Diego: Academic Press, v. 3, 1999.

8. ORDONEZ, E. D. M. et al. **Projeto, desempenho e aplicações de Sistemas Digitais em Circuitos Programáveis (FPGAs)**. Pompéia: Bless, 2003.

9. TODDMAN, T. J. et al. Reconfigurable Computing: Architectures, Design Methods, and Applications. **IEE Proceedings: Computer & Digital Techniques**, v. 152, n. 2, p. 193-208, Março 2005.

10. GRAND VIEW RESEARCH. Field Programmable Gate Array (FPGA) Market Analysis By Technology (SRAM, EEPROM, Antifuse, Flash), By Application (Consumer Electronics, Automotive, Industrial, Data Processing, Military & Aerospace, Telecom), And Segment Forecasts, 2018 - 2024. **Grand View Research**, 2018. Disponivel em: <https://www.grandviewresearch.com/industry-analysis/fpga-market>. Acesso em: Março 2018.

11. WAKERLY, J. F. **Digital design:** principles and practices. 4. ed. Upper Saddle River: Pearson Prentice Hall, 2006.

12. PEDRONI, V. A. **Circuit Design with VHDL**. Cambridge: MIT Press, 2004.

13. D'AMORE, R. **VHDL:** Descição e Síntese de Circuitos Digitais. 1. ed. Rio de Janeiro: LTC, 2005.

14. MIDORIKAWA, E. T. Uma Introdução às Linguagens de Descrição de Hardware. **Site da Escola Politécnica da Universidade de São Paulo**, 2007. Disponivel em: <http://www.pcs.usp.br/~edson/intro-hdl.pdf>. Acesso em: 18 Janeiro 2017.

15. ALTERA CORPORATION. **DE2 Development and Education Board:** User Manual. 1.6. ed. [S.l.]: Altera Corporation, 2012.

16. MENTOR GRAPHICS. **ModelSim® Tutorial**. Wilsonville, Oregon: Mentor Graphics Corporation, 2012.

17. AGILENT TECHNOLOGIES. **Agilent 30 MHz Function/Arbitrary Waveform Generators**. USA: [s.n.], 2010.

18. RUSHTON, A. **VHDL for logic synthesis**. Chichester: Wiley, 2011.

19. CRYSTALFONTZ AMERICA. **CFAH1602B-TMC-JP**. Valleyford: [s.n.].

20. GERRY, O. HD44780 LCD Display Tutorial using VHDL and the ALTERA DE2 Board, 2017. Disponivel em: <http://www.digital-circuitry.com/Projects_LCD_DISPLAYS.htm>. Acesso em: 2 Janeiro 2017.

21. HITACHI. **HD44780U (LCD-II):** Dot Matrix Liquid Crystal Display Controller/Driver. Tóquio: [s.n.], 1998.

22. BARBACENA, I. L.; FLEURY, C. A. Display LCD, 1996. Disponivel em: <ftp://ftp.dca.fee.unicamp.br/pub/docs/ea079/complementos/Lcd.pdf>. Acesso em: 06 Abril 2017.

APÊNDICE A – Código do projeto para exibição da frequência no *display* de 7 segmentos

Código “tacometro_7seg.vhd”

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY tacometro_7seg IS

      PORT
      (
            clock50Mhz: IN STD_LOGIC;
            sinal: IN STD_LOGIC;
            HEX3,HEX2,HEX1,HEX0 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0)

       );

END tacometro_7seg;

ARCHITECTURE a OF tacometro_7seg IS

COMPONENT digito IS

      PORT (INT         : IN NATURAL RANGE 0 TO 9;
                  HEX   : OUT STD_LOGIC_VECTOR(6 DOWNTO 0));

END COMPONENT;

            signal  prescaler:  STD_LOGIC_VECTOR(24  downto  0)  :=
"10111110101111000001000000"; -- 25.000.000 em binário
            signal contador: STD_LOGIC_VECTOR(24 downto 0) := (others
=> '0');
            signal clock1Hz : std_logic := '0';

            signal freq1 : integer :=0;
            signal freq2 : integer :=0;
            signal FreqSoma : integer :=0;
            signal frequencia : integer :=0;

            signal restoM : integer :=0;
            signal restoC : integer :=0;
            signal restoD : integer :=0;
            signal U: integer :=0;
            signal D: integer :=0;
            signal C: integer :=0;
            signal M: integer :=0;

            signal apaga: STD_LOGIC_VECTOR (6 DOWNTO 0) := "1111111";

BEGIN

      HEX7 <= apaga;
      HEX6 <= apaga;
      HEX5 <= apaga;
      HEX4 <= apaga;

      milhar  : digito PORT MAP (M,HEX3);
```

```
      centena : digito PORT MAP (C,HEX2);
      dezena  : digito PORT MAP (D,HEX1);
      unidade : digito PORT MAP (U,HEX0);

      PROCESS(clock50Mhz, clock1Hz)
      BEGIN

            IF RISING_EDGE(clock50Mhz) THEN
                  contador <= contador + 1;
                  IF(contador > prescaler) THEN
                        clock1Hz <= NOT clock1Hz;
                        contador <= (OTHERS => '0');
         END IF;
            END IF;

            END PROCESS;

            PROCESS (clock1Hz, sinal)
            BEGIN

                  IF clock1Hz = '0' THEN

                        IF (sinal' EVENT AND sinal = '0') THEN
                              Freq1 <= Freq1 + 1;

                        ELSIF (sinal' EVENT AND sinal = '1') THEN
                              Freq2 <= Freq2 + 1;
                        END IF;

                        FreqSoma <= Freq1 + Freq2;

                  ELSIF clock1Hz = '1' THEN
                        Freq1 <= 0;
                        Freq2 <= 0;
                        frequencia <= FreqSoma;
                  END IF;

                  restoM  <= frequencia mod 1000;
                  M  <= (frequencia - restoM)/1000;
                  restoC <= restoM mod 100;
                  C <= (restoM - restoC)/100;
                  restoD <= restoC mod 10;
                  D <= (restoC - restoD)/10;
                  U <= restoD;

            END PROCESS;


END a;
```

Componente “digito.vhd”

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY digito IS
      PORT
      (
      INT         : IN NATURAL RANGE 0 TO 9;
      HEX  : OUT STD_LOGIC_VECTOR (6 DOWNTO 0)
      );
```

```
END digito;

ARCHITECTURE a OF digito IS

BEGIN

      WITH INT SELECT

             HEX <= "0000001" WHEN 0,
                          "1001111" WHEN 1,
                          "0010010" WHEN 2,
                          "0000110" WHEN 3,
                          "1001100" WHEN 4,
                          "0100100" WHEN 5,
                          "0100000" WHEN 6,
                          "0001111" WHEN 7,
                          "0000000" WHEN 8,
                          "0000100" WHEN 9;

END a;
```

APÊNDICE B – Atribuição de pinos do projeto para exibição da frequência no *display* de 7 segmentos

Porta	Direção	Pino
clock50Mhz	Entrada	PIN_N2
HEX0[6]	Saída	PIN_AF10
HEX0[5]	Saída	PIN_AB12
HEX0[4]	Saída	PIN_AC12
HEX0[3]	Saída	PIN_AD11
HEX0[2]	Saída	PIN_AE11
HEX0[1]	Saída	PIN_V14
HEX0[0]	Saída	PIN_V13
HEX1[6]	Saída	PIN_V20
HEX1[5]	Saída	PIN_V21
HEX1[4]	Saída	PIN_W21
HEX1[3]	Saída	PIN_Y22
HEX1[2]	Saída	PIN_AA24
HEX1[1]	Saída	PIN_AA23
HEX1[0]	Saída	PIN_AB24
HEX2[6]	Saída	PIN_AB23
HEX2[5]	Saída	PIN_V22
HEX2[4]	Saída	PIN_AC25
HEX2[3]	Saída	PIN_AC26
HEX2[2]	Saída	PIN_AB26
HEX2[1]	Saída	PIN_AB25
HEX2[0]	Saída	PIN_Y24
HEX3[6]	Saída	PIN_Y23
HEX3[5]	Saída	PIN_AA25
HEX3[4]	Saída	PIN_AA26
HEX3[3]	Saída	PIN_Y26
HEX3[2]	Saída	PIN_Y25
HEX3[1]	Saída	PIN_U22
HEX3[0]	Saída	PIN_W24
Sinal	Entrada	PIN_D25

APÊNDICE C – Código do projeto para exibição da frequência no *display* LCD

```
LIBRARY IEEE;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_ARITH.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY tacometro_lcd IS


      PORT(
      reset              : IN     std_logic;
      clock50MHz              : IN     std_logic;
      sinal                        : IN     std_logic;

      lcd_rs             : OUT    std_logic;
      lcd_e              : OUT    std_logic;
      lcd_rw             : OUT    std_logic;
      lcd_on             : OUT    std_logic;
      lcd_blon           : OUT    std_logic;

      lcd_data_0         : INOUT  STD_LOGIC;
      lcd_data_1         : INOUT  STD_LOGIC;
      lcd_data_2         : INOUT  STD_LOGIC;
      lcd_data_3         : INOUT  STD_LOGIC;
      lcd_data_4         : INOUT  STD_LOGIC;
      lcd_data_5         : INOUT  STD_LOGIC;
      lcd_data_6         : INOUT  STD_LOGIC;
      lcd_data_7         : INOUT  STD_LOGIC;

      SW0    : IN    STD_LOGIC;
      SW1    : IN    STD_LOGIC;
      SW2    : IN    STD_LOGIC;
      SW3    : IN    STD_LOGIC

   );
END tacometro_lcd ;


--
ARCHITECTURE a OF tacometro_lcd IS

      type string32 is array ( 0 to 31 ) of STD_LOGIC_VECTOR( 7 downto 0 );
      type string10 is array ( 0 to 9 ) of STD_LOGIC_VECTOR( 7 downto 0 );
      type estado_type is (func_set, display_on, mode_set, print_string,
                                    Linha2, return_home, drop_lcd_e,
reset1, reset2,
                                     reset3, display_off, display_clear);

      signal estado, prox         : estado_type;

      signal string0       : string32;--:=x"00";
      signal string1       : string32;--:=x"00";
      signal string2       : string32;--:=x"00";
      signal string3       : string32;--:=x"00";

      signal N                     : string10;--:=x"00";


      signal lcd_data_valor: STD_LOGIC_VECTOR(7 downto 0):=x"00";
      signal next_char   : STD_LOGIC_VECTOR(7 downto 0):=x"00";
```

```
      signal clk_count_400hz             : STD_LOGIC_VECTOR(23 downto 0);
      signal char_count                  : STD_LOGIC_VECTOR(4 downto 0) :=
"00000";
      signal clk_400hz_enable,lcd_rw_int : std_logic;

      signal lcd_data        : STD_LOGIC_VECTOR(7 downto 0) ;

      signal SW              : STD_LOGIC_VECTOR(3 DOWNTO 0);

      signal prescaler: STD_LOGIC_VECTOR(24 downto 0) :=
"1011111010111100001000000"; -- 12.500.000 em binário
      signal contador: STD_LOGIC_VECTOR(24 downto 0) := (OTHERS => '0');
      signal clock1Hz : std_logic := '0';

               signal prescaler400Hz: STD_LOGIC_VECTOR(11 downto 0) := "
111101000010010 0";  -- 12.500.000 em binário
      signal contador400Hz: STD_LOGIC_VECTOR(11 downto 0) := (OTHERS =>
'0');
      signal clock400Hz : std_logic := '0';


      signal Freq1 : integer :=0;
      signal Freq2 : integer :=0;
      signal FreqSoma : integer :=0;
      signal frequencia : integer :=0;

      signal restoM3 : integer :=0;
      signal restoM2 : integer :=0;
      signal restoM1 : integer :=0;
      signal restoM : integer :=0;
      signal restoC : integer :=0;
      signal restoD : integer :=0;

      signal M3: integer :=0;
      signal M2: integer :=0;
      signal M1: integer :=0;
      signal M: integer :=0;
      signal C: integer :=0;
      signal D: integer :=0;
      signal U: integer :=0;


BEGIN


      lcd_data_0 <= lcd_data(0);
      lcd_data_1 <= lcd_data(1);
      lcd_data_2 <= lcd_data(2);
      lcd_data_3 <= lcd_data(3);
      lcd_data_4 <= lcd_data(4);
      lcd_data_5 <= lcd_data(5);
      lcd_data_6 <= lcd_data(6);
      lcd_data_7 <= lcd_data(7);

      SW(0) <= SW0;
      SW(1) <= SW1;
      SW(2) <= SW2;
      SW(3) <= SW3;

      N <= (x"30",x"31",x"32",x"33",x"34",x"35",x"36",x"37",x"38",x"39");
```

```
      PROCESS(clock50MHz, clock1Hz)
      BEGIN

            IF RISING_EDGE(clock50MHz) THEN
                  contador <= contador + 1;
                  IF(contador > prescaler) THEN
                        clock1Hz <= NOT clock1Hz;
                        contador <= (OTHERS => '0');
         END IF;
            END IF;

            END PROCESS;


            PROCESS (clock1Hz, sinal)
            BEGIN

                  IF clock1Hz = '0' THEN
                        IF (sinal' EVENT AND sinal = '0') THEN
                              Freq1 <= Freq1 + 1;

                        ELSIF (sinal' EVENT AND sinal = '1') THEN
                              Freq2 <= Freq2 + 1;
                        END IF;
                        FreqSoma <= Freq1 + Freq2;
                  ELSIF clock1Hz = '1' THEN
                        IF (SW2 = '1') OR (SW3 = '1') THEN
                              frequencia <= FreqSoma*60;
                        ELSE frequencia <= FreqSoma;
                        END IF;
                        Freq1 <= 0;
                        Freq2 <= 0;
                  END IF;

                  restoM3  <= frequencia mod 1000000;
                  M3  <= (frequencia - restoM3)/1000000;
                  restoM2  <= restoM3 mod 100000;
                  M2  <= (restoM3 - restoM2)/100000;
                  restoM1  <= restoM2 mod 10000;
                  M1  <= (restoM2 - restoM1)/10000;
                  restoM  <= restoM1 mod 1000;
                  M  <= (restoM1 - restoM)/1000;
                  restoC <= restoM mod 100;
                  C <= (restoM - restoC)/100;
                  restoD <= restoC mod 10;
                  D <= (restoC - restoD)/10;
                  U <= restoD;

            END PROCESS;



lcd_data <= lcd_data_valor when lcd_rw_int = '0' else "ZZZZZZZZ";
lcd_rw <= lcd_rw_int;

process(clock50MHz)

begin

string0  <=
```

```
  (
----                                                    NUM   NUM    NUM
NUM     NUM   NUM   NUM    H        z

x"20",x"20",x"20",N(M3),N(M2),N(M1),N(M),N(C),N(D),N(U),x"48",x"7A",
x"20",x"20",x"20",x"20",

x"53",x"57",x"31",x"2D",x"4B",x"48",x"7A",x"20",x"53",x"57",x"32",x"2D",x"7
2",x"70",x"6D",x"20"
-- Linha 2   S    W      1    -      K     H     z            S    W      2
-     r           p           m

   );


  string1<=
  (
--                                        NUM   NUM    NUM    NUM    ,   NUM
        NUM   NUM    K        H           z

x"20",x"20",N(M3),N(M2),N(M1),N(M),x"2C",N(C),N(D),N(U),x"4B",x"48",x"7A",
x"20",x"20",x"20",



x"53",x"57",x"30",x"2D",x"48",x"7A",x"20",x"53",x"57",x"32",x"2D",x"72",x"7
0",x"6D",x"20",x"20"
-- Linha 2   S    W      0    -      H     z           S    W      2    -
r           p           m
   );


  string2 <=
    (
--                                                    NUM   NUM    NUM
NUM     NUM   NUM   NUM    r        p           m

x"20",x"20",x"20",N(M3),N(M2),N(M1),N(M),N(C),N(D),N(U),x"72",x"70",
x"6D",x"20",x"20",x"20",

x"53",x"57",x"30",x"2D",x"48",x"7A",x"20",x"53",x"57",x"31",x"2D",x"4B",x"4
8",x"7A",x"20",x"20"
-- Linha 2   S    W      0    -      H     z           S    W      1    -
K           H           z
   );


      string3 <=
  (
--                                              NUM    NUM   NUM  NUM   NUM
      NUM  NUM    /           m            i           n

x"20",x"20",x"20",N(M3),N(M2),N(M1),N(M),N(C),N(D),N(U),x"2F",x"6D",
x"69",x"6E",x"20",x"20",

x"53",x"57",x"31",x"2D",x"4B",x"48",x"7A",x"20",x"53",x"57",x"32",x"2D",x"7
2",x"70",x"6D",x"20"
-- Linha 2   S    W      1    -      K     H     z            S    W      2
-     r           p           m

   );
```

```
      PROCESS(clock50MHz)
      BEGIN

            IF RISING_EDGE(clock50MHz) THEN
                  contador_400Hz <= contador_400Hz + 1;
                  IF(contador > prescaler) THEN
                        clock400Hz <= NOT clock400Hz;
                        contador <= (OTHERS => '0');
         END IF;
            END IF;

            END PROCESS;


PROCESS (SW)
   BEGIN
   CASE (SW) IS


      WHEN "0010" =>
      -- SW1 = 1, frequencia em KHz
      next_char <= string1 (CONV_INTEGER(char_count));

      WHEN "0100" =>
      -- SW2 = 1, frequencia em rpm
      next_char <= string2 (CONV_INTEGER(char_count));

      WHEN "1000" =>
      -- SW3 = 1, frequencia em min^-1
      next_char <= string3(CONV_INTEGER(char_count));
      WHEN OTHERS =>
      -- Outro caso, frequencia em Hz
      next_char <= string0(CONV_INTEGER(char_count));
   END CASE;

END PROCESS;


process (clock50MHz, reset)
begin
        if reset = '0' then
           estado <= reset1;
           lcd_data_valor <= x"38"; -- RESET
           prox <= reset2;
           lcd_e <= '1';
           lcd_rs <= '0';
           lcd_rw_int <= '0';


             elsif rising_edge(clock50MHz) then
            if clock400Hz = '1' then

                case estado is

                      when reset1 =>
                           lcd_e <= '1';
                           lcd_rs <= '0';
                           lcd_rw_int <= '0';
                           lcd_data_valor <= x"38";
```

```
                    estado <= drop_lcd_e;
                    prox <= reset2;
                    char_count <= "00000";

               when reset2 =>
                    lcd_e <= '1';
                    lcd_rs <= '0';
                    lcd_rw_int <= '0';
                    lcd_data_valor <= x"38";
                    estado <= drop_lcd_e;
                    prox <= reset3;

               when reset3 =>
                    lcd_e <= '1';
                    lcd_rs <= '0';
                    lcd_rw_int <= '0';
                    lcd_data_valor <= x"38";
                    estado <= drop_lcd_e;
                    prox <= func_set;

               when func_set =>
                    lcd_e <= '1';
                    lcd_rs <= '0';
                    lcd_rw_int <= '0';
                    lcd_data_valor <= x"38";
                    estado <= drop_lcd_e;
                    prox <= display_off;


               when display_off =>
                    lcd_e <= '1';
                    lcd_rs <= '0';
                    lcd_rw_int <= '0';
                    lcd_data_valor <= x"08";
                    estado <= drop_lcd_e;
                    prox <= display_clear;

               when display_clear =>
                    lcd_e <= '1';
                    lcd_rs <= '0';
                    lcd_rw_int <= '0';
                    lcd_data_valor <= x"01";
                    estado <= drop_lcd_e;
                    prox <= display_on;

               when display_on =>
                    lcd_e <= '1';
                    lcd_rs <= '0';
                    lcd_rw_int <= '0';
                    lcd_data_valor <= x"0C";
                    estado <= drop_lcd_e;
                    prox <= mode_set;

               when mode_set =>
                    lcd_e <= '1';
                    lcd_rs <= '0';
                    lcd_rw_int <= '0';
                    lcd_data_valor <= x"06";
                    estado <= drop_lcd_e;
                    prox <= print_string;
```

```
                    when print_string =>

                     estado <= drop_lcd_e;
                          lcd_e <= '1';
                          lcd_rs <= '1';
                          lcd_rw_int <= '0';
          lcd_data_valor <= next_char;

          estado <= drop_lcd_e;

          IF (char_count < 31) THEN
                               char_count <= char_count +1;
                                       ELSE
                               char_count <= "00000";
          END IF;

                          if char_count = 15 then
          prox <= Linha2;
                             elsif (char_count = 31) then
                                   prox <= return_home;
                             else
                                   prox <= print_string;
                             end if;


                    when Linha2 =>
                          lcd_e <= '1';
                          lcd_rs <= '0';
                          lcd_rw_int <= '0';
                          lcd_data_valor <= x"c0";
                          estado <= drop_lcd_e;
                          prox <= print_string;


                    when return_home =>
                          lcd_e <= '1';
                          lcd_rs <= '0';
                          lcd_rw_int <= '0';
                          lcd_data_valor <= x"80";
                          estado <= drop_lcd_e;
                          prox <= print_string;


                    when drop_lcd_e =>
                          estado <= prox;
                          lcd_e <= '0';
                          lcd_blon <= '1';
                          lcd_on   <= '1';
                     end case;

            end if;
      end if;

end process;

END ARCHITECTURE a;
```

APÊNDICE D – Atribuição de pinos do projeto para exibição da frequência no *display* LCD

Porta	Direção	Pino
Sinal	Entrada	PIN_D25
clock50MHz	Entrada	PIN_N2
SW0	Entrada	PIN_N25
SW1	Entrada	PIN_N26
SW2	Entrada	PIN_P25
SW3	Entrada	PIN_AE14
lcd_blon	Saída	PIN_K2
lcd_data_0	Saída	PIN_J1
lcd_data_1	Saída	PIN_J2
lcd_data_2	Saída	PIN_H1
lcd_data_3	Saída	PIN_H2
lcd_data_4	Saída	PIN_J4
lcd_data_5	Saída	PIN_J3
lcd_data_6	Saída	PIN_H4
lcd_data_7	Saída	PIN_H3
lcd_e	Saída	PIN_K3
lcd_on	Saída	PIN_L4
lcd_rs	Saída	PIN_K1
lcd_rw	Saída	PIN_K4

ANEXO A – Instruções para o LCD

	Código										
Instrução	**RS**	**R/W**	**DB7**	**DB6**	**DB5**	**DB4**	**DB3**	**DB2**	**DB1**	**DB0**	**Descrição**
Clear Display	0	0	0	0	0	0	0	0	0	1	Limpa o *display* e retorna o cursor para a primeira posição da primeira linha.
Return home	0	0	0	0	0	0	0	0	1	-	Retorna o cursor para a primeira linha da primeira coluna. Retorna a mensagem previamente deslocada para sua posição original
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Define a direção na qual o cursor se movimentará (I/D) e especifica deslocamento automático (S). Estas ações são executadas durante leitura ou gravação de dados.
Display on/off control	0	0	0	0	0	0	1	D	C	B	Liga/desliga o *display* (D), liga/desliga o cursor (C), ativa/desativa o piscar do cursor (B).
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	-	-	Move o cursor e desloca o *display* (S/C) para a direita/esquerda (R/L) sem alterar o conteúdo da DDRAM.
Function set	0	0	0	0	1	DL	N	F	-	-	Define o número de bits para comunicação (DL), número de linhas do *display* (N) e o tamanho da fonte (F).
Set CGRAM address	0	0	0	1	ACG	ACG	ACG	ACG	ACG	ACG	Define o endereço da CGRAM. O dado da CGRAM é enviado e recebido após esta configuração.
Set DDRAM address	0	0	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	Define o endereço da DDRAM. O dado da DDRAM é enviado e recebido após esta configuração.

	Código										
Instrução	**RS**	**R/W**	**DB7**	**DB6**	**DB5**	**DB4**	**DB3**	**DB2**	**DB1**	**DB0**	**Descrição**
Read busy flag & address	0	1	BF	AC	AC	AC	AC	AC	AC	AC	Flag do Read Busy (BF) e contador de endereços
Write data to CG or DDRAM	1	0	Write data								Escreve dados na CGRAM ou DDRAM.
Read data to CG or DDRAM	1	1	Read data								Lê dados da CGRAM ou DDRAM

I/D=1:	Incrementa
I/D=0:	Decrementa
S=1:	*Display* desloca automaticamnte
S/C=1:	Deslocamento do *display*
S/C=0:	Movimento do cursor
R/L=1:	Desloca para a direita
R/L=0:	Desloca para a esquerda
DL=1:	Interface de 8 bits
DL=0:	Interface de 4 bits
N=1:	2 linhas
N=0:	1 linha
F=1:	Caractere 5x10 pontos
F=0:	Caractere 5x7 pontos
BF=1:	Operando internamente
BF=0:	Aceitando instruções

DDRAM:	*Display* data RAM
CGRAM	Character generator RAM
ACG:	Endereço CGRAM
ADD:	Endereço DDRAM
AC:	Contador de endereços para CGRAM e DDRAM