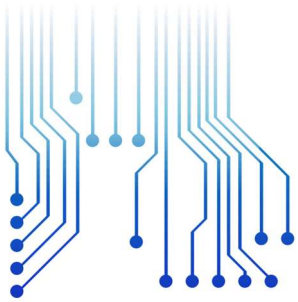
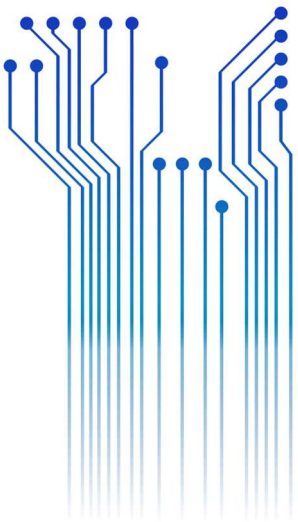


CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

JOSÉ SAMUEL MENDES

TRABALHO DE CONCLUSÃO DE CURSO

IMPLEMENTAÇÃO DE UMA INTERFACE GRÁFICA PARA A SIMULAÇÃO DE LINGUAGENS DE DESCRIÇÃO DE HARDWARE.



Campina Grande
2018

JOSÉ SAMUEL MENDES

IMPLEMENTAÇÃO DE UMA INTERFACE GRÁFICA PARA A SIMULAÇÃO DE LINGUAGENS DE DESCRIÇÃO DE HARDWARE.

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

Área de Concentração: Microeletrônica

Orientador:
Gutemberg Gonçalves dos Santos Júnior, D. Sc.

Campina Grande
2018

JOSÉ SAMUEL MENDES

IMPLEMENTAÇÃO DE UMA INTERFACE GRÁFICA PARA A SIMULAÇÃO DE LINGUAGENS DE DESCRIÇÃO DE HARDWARE.

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

Área de Concentração: Microeletrônica

Aprovado em ____ de dezembro de 2018

_____

Professor Gutemberg Gonçalves dos Santos Júnior, D.Sc.
Universidade Federal de Campina Grande
Orientador

_____

Professor Marcos Ricardo Alcântara Morais, D.Sc.
Universidade Federal de Campina Grand
Avaliador

Dedico este trabalho aos meus avós, pela educação e toda experiência de vida compartilhada.

AGRADECIMENTOS

Aos meus pais, pelo apoio incondicional e por fazerem todos os esforços possíveis para realização deste sonho.

A toda a minha família, pelo suporte, incentivo e por se alegrarem comigo a cada vitória, em especial aos meus irmãos pelos anos de convivência em Campina Grande, durante a graduação.

A minha namorada Mariane, pelo companheirismo, afeto, paciência e ajuda durante a elaboração deste trabalho.

Aos Pseudomitos, grupo de estudo no qual participei durante toda a graduação, realizando várias reuniões para discutir resoluções de questões, tirar dúvidas e conversas sobre os diversos assuntos da universidade, principalmente nas vésperas de avaliações.

Aos amigos de outrora, aos que vieram só agora, mas principalmente aos que ficarão para sempre.

A todos que fazem parte do X-MEN Lab (PEM) que durante o fim da graduação se tornou minha segunda casa.

Aos professores Marcos Morais e Gutemberg Júnior, por todos os ensinamentos, orientação e oportunidades concedidas, permitindo que eu descobrisse áreas de interesse da minha formação e por acreditarem na microeletrônica no Brasil.

“Don’t be sorry, be better”

Kratos-*God of War*

RESUMO

Este trabalho propõe uma interface gráfica dos principais periféricos de um dispositivo FPGA, tais como: chaves, LEDs, botões e display de sete segmentos. Com o intuito de aplicação nas principais disciplinas práticas relacionadas a circuitos digitais na UFCG, visando facilitar o aprendizado do aluno, bem como direcionar o estudo em casa do mesmo. Além de enriquecer o conhecimento do aluno, propondo novos conhecimentos sobre simulação e testes de circuitos digitais.

Palavras-chave: FPGA, TCL/TK, *Testbench,* Simulação.

ABSTRACT

This work propose an interface of the major peripherals of a FPGA, like: switches, keys, LEDs, seven segments displays. With the intention of applicate on the major laboratory classes of digital circuits at UFCG. Looking for a better learning of the student, as well as, orientate the learning at home. Besides improve the knowledge about the simulation and tests of digital circuits.

Keywords: FPGA, TCL/TK, *Testbench,* Simalation.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

CAD	*Computer Aided Design*
DUT	*Design Under Test*
FPGA	*Field Programmable Gate Array*
GUI	*Graphical User Interface*
HDL	*Hardware Discritive Language*
IEEE	*Instute of Electrical and Eletronics Engineers*
LUT	*Look-Up Table*
nMOS	*N-type Metal-Oxide-Semiconductor*
RTL	*Register Transfer Language*
SRAM	*Static Random Access Memory*
TCL	*Tool Command Language*
TK	*Tool Kit*

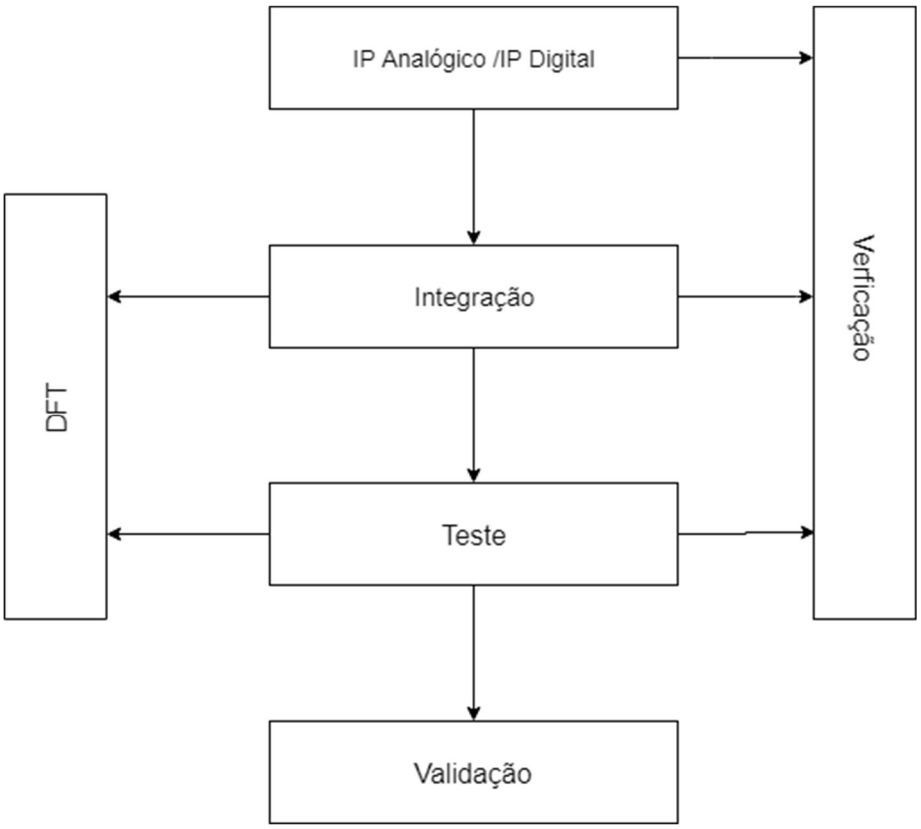
SUMÁRIO

1 INTRODUÇÃO

Com o desenvolvimento tecnológico humano, foi-se cada vez mais aumentando a quantidade de informação. E era necessário utilizar dispositivos que fossem capazes de lidar com toda essa quantidade de informação gerada de uma forma simples e precisa. A partir desta perspectiva deu-se a ideia dos sistemas digitais.

Devido a esta demanda o desenvolvimento de circuitos digitais integrados é uma das que mais cresce no mundo inteiro. Para suprir toda esta demanda que ocorre se é preciso criar métodos que visam automatizar e agilizar todo esse processo de desenvolvimento de circuitos digitais. Na Figura 1 abaixo pode-se observar o fluxo do processo de desenvolvimento de um SoC (*System-on-a-Chip*).

Figura 1 – Fluxo Do Processo De Desenvolvimento De Um SoC



Fonte: Autor (2018)

Pode-se dizer que toda essa automatização e processos de criação destes circuitos se dá devido a existência das linguagens descritivas de hardwares. Graças a estas linguagens é possível descrever o funcionamento do circuito, a sua concepção e organização, e ainda capaz de testá-lo para verificar e validar o seu funcionamento por meio de simulação de uma formar rápida e segura.

Além disto, existe dispositivos compostos de diversas funções lógicas, memórias, DSP, blocos de entradas e saídas, flexíveis e programáveis. Estes dispositivos são conhecidos como FPGAs (Field Progamable Gate Array), eles permitem fazer testes com alguns circuitos de forma rápida e prática, e sendo assim, possível obter resultados confiáveis. A Figura 2 abaixo ilustra um modelo de um FPGA.

Figura 2 - Modelo de um Dispositivo FPGA

Fonte: https://www.terasic.com.tw

Devido a esta flexibilidade, estes dispositivos são amplamente utilizados no aprendizado de disciplinas que envolvem circuitos digitais, principalmente em disciplinas que englobam um conteúdo mais prático. Porém em alguns casos estes dispositivos podem limitar o aprendizado destes alunos, devido ao seu alto custo não sendo possível que todos alunos sejam capazes de possuir um dispositivo deste em sua residência.

Uma alternativa prática capaz de resolver esta dificuldade, seria a utilização de ferramenta capazes de realizar simulações destes circuitos digitais, que no ponto de vista das disciplinas, o aprendizado e resultados obtidos utilizando estes dispositivos ou através destas simulações serão praticamente o mesmo.

Visando este contexto é possível criar uma interface gráfica, através de uma linguagem de scripts conhecida como TCL/TK. No qual seja capaz de abstrair os principais periféricos de um dispositivo de FPGA, e através de simulações e *testbenchs* simular o funcionamento de um FPGA.

A partir disto é possível dirigir o aluno cada vez ao âmbito de simulação de circuitos digitais, fazendo com que o mesmo seja capaz de verificar e simular o funcionamento de qualquer circuito digital implementado pelo mesmo.

1.1 OBJETIVOS

O objetivo geral deste trabalho é propor a implementação de uma interface gráfica, que a partir de simulações, seja capaz de abstrair a utilização de FPGAs utilizadas amplamente em disciplinas que envolvam circuitos digitais.

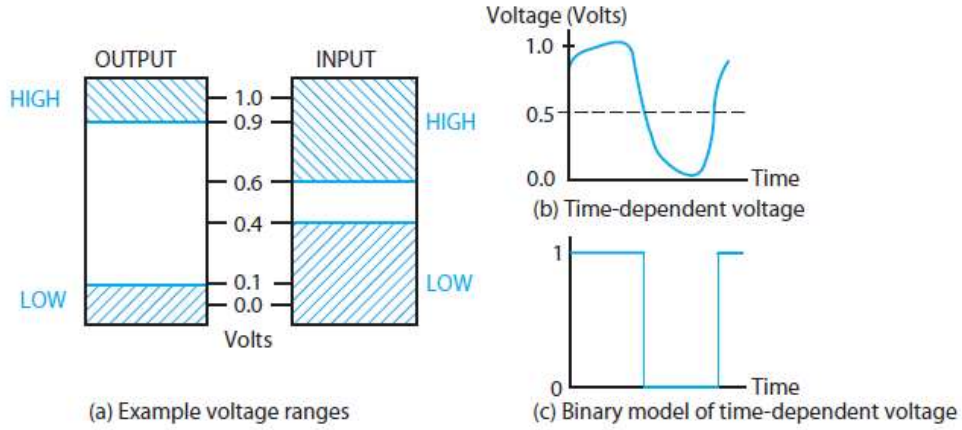
- Revisão bibliográfica sobre circuitos digitais e linguagens descritivas de hardwares;
- Implementar através da linguagem TCL/TK a interface que vise abstrair os periféricos mais utilizados nas disciplinas, tais como chaves, botões, LEDs entre outros;
- Desenvolver *testbenchs* que sejam capazes de se comunicar com esta interface;
- Desenvolver um ambiente que seja capaz de guiar o aluno cada vez mais para o ambiente de simulações.

2 CIRCUITOS DIGITAIS

É possível dizer que atualmente, os sistemas digitais são capazes de armazenar, mover e processar informações. Para entender o princípio básico dos sistemas digitais, vamos supor que deseja-se medir a temperatura, na qual é de natureza contínua, com um sensor e depois converter em tensão elétrica, também de natureza contínua. Essa tensão de natureza contínua é conhecida como um sinal analógico, no qual é uma das maneiras de representar a temperatura. Outra possibilidade é representar esta temperatura por meio de valores discretos de tensão. Na qual os valores medidos são amostrados dentro de um conjunto finito de números. Já essa tensão de natureza discreta é conhecida como sinal digital. De forma alternativa podemos representar estes valores discretos por múltiplos sinais de tensão, no qual cada combinação destes sinais representam um único valor discreto. Por exemplo, digamos que os valores de temperatura foram amostrados em um conjunto entre -143 e 113 graus, poderíamos representar os 256 valores de temperaturas com uma combinação de 8 sinais, no qual cada um destes 8 sinais assumem 2 valores. Ou seja $2^8 = 256$. Este modelo de sinal representa a base do funcionamento dos circuitos da eletrônica digital. Conhecido como sinal binário ele varia entre 0 e 1, dígitos do sistema numérico binário.

Estes valores quando amostrados para sinais de tensão são tipicamente chamados de **nível alto** (*high*) e **nível baixo** (*low*). Na Figura 3 abaixo podemos ver o conjunto de valores de tensão que representa o nível alto e nível baixo, para entradas (*input*) e para saídas (*output)* de um circuito digital.

Figura 3 – Conjunto de valores de tensão de um circuito digital



Fonte: MANO, M.M. Computer System Architecture, Prentice-Hall International, Inc., 1993.

Como podemos ver na Figura 3 o valor de tensão para saídas em nível alto varia entre 0,9 e 1,1 volts e para saídas em nível baixo é entre -0,1 e 0,1 volts. Já para entradas temos que em nível alto a tensão varia entre 0,6 e 1,1 volts e para nível baixo a tensão varia entre -0,1 e 0,4 volts. O fato de o alcance da entrada ser maior ocorre para garantir o funcionamento correto dos circuitos digitais, já que devido ao seu comportamento e ruídos indesejáveis tensões podem ser adicionadas ou subtraídas na saída.

Sendo assim, pode-se perceber que o sistema numérico binário é a base do funcionamento de qualquer circuito digital. Portanto, devido os algarismos 0 e 1 estarem associados ao sistema numérico binário, esta é a principal forma de representar o alcance do sinal digital. Estes dígitos binários são comumente conhecidos como *bits*. A informação que estes circuitos recebem são codificadas em combinações de bits e assim processadas. Quando processados essa informação é capaz de representar uma grande quantidade de outros conjuntos discretos como por exemplo um teclado alfanumérico ou até mesmo especificar a um computador um conjunto de instruções a ser realizado.

2.1 CIRCUITOS COMBINACIONAIS E SEQUENCIAIS:

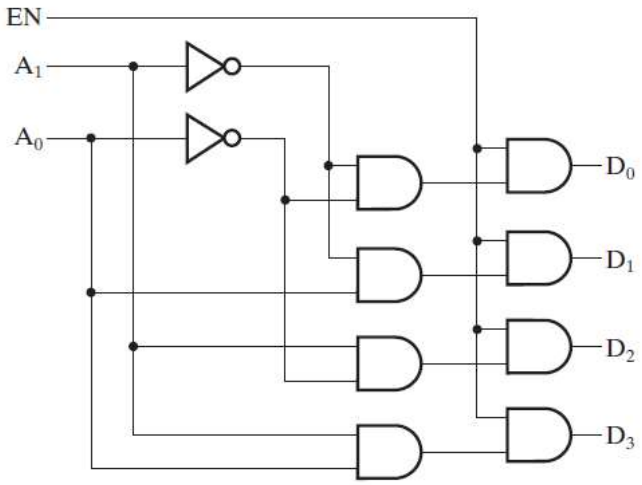
Para ajudar na criação e desenvolvimento de circuitos digitais é preciso abstrair um pouco mais além do sistema binário. Esta camada de abstração é conhecida por portas lógicas. Onde utilizando a combinação de várias dessas portas lógicas somos capazes de

criar e desenvolver os circuitos digitais. A nível de portas lógicas pode-se dizer que existe basicamente dois tipos de circuitos: Circuitos Combinacionais e Circuitos Sequencias.

Os circuitos combinacionais são definidos pelo fato de a saída depender unicamente dos seus valores de entradas atuais. São nesses circuitos onde se é conhecido as principais portas lógicas, equações Booleanas, além de codificadores, decodificadores, somadores, comparadores e entre outros. A Figura 4 ilustra um exemplo de circuito combinacional.

Figura 4 – Modelo de um Circuito Combinacional

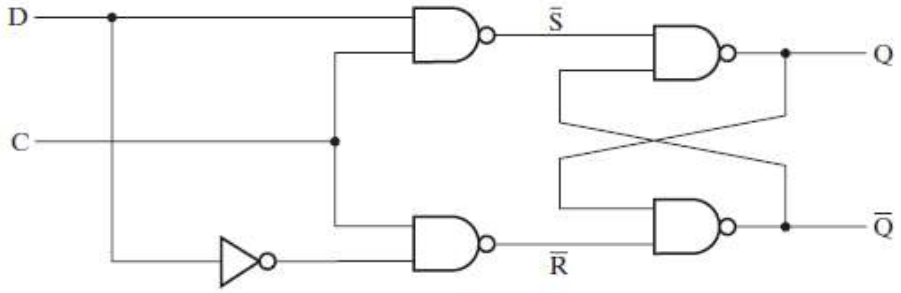
EN	A_1	A_0	D_0	D_1	D_2	D_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1



Fonte: MANO, M.M. Computer System Architecture, Prentice-Hall International, Inc., 1993

Já em circuitos sequenciais a saída não depende unicamente da entrada atual, mas da sequência de valores anteriores desta entrada. Grande parte destes circuitos são utilizados para armazenar resultados obtidos através de circuitos combinacionais. E assim podendo diminuir o custo e a complexidade dos circuitos digitais. Exemplo de circuitos sequenciais são os flip-flops e lacthes e um exemplo de um circuito que é a combinação entre circuitos combinacionais e sequencias são as máquinas de estado, na qual, se faz presente em grande parte dos projetos de circuitos digitais, devido a sua flexibilidade e facilidade de implementação. Abaixo na Figura 5 ilustra-se um exemplo de um circuito sequencial.

Figura 5 – Exemplo de um circuito sequencial



(a) Logic diagram

C	D	Next state of Q
0	X	No change
1	0	Q = 0; Reset state
1	1	Q = 1; Set state

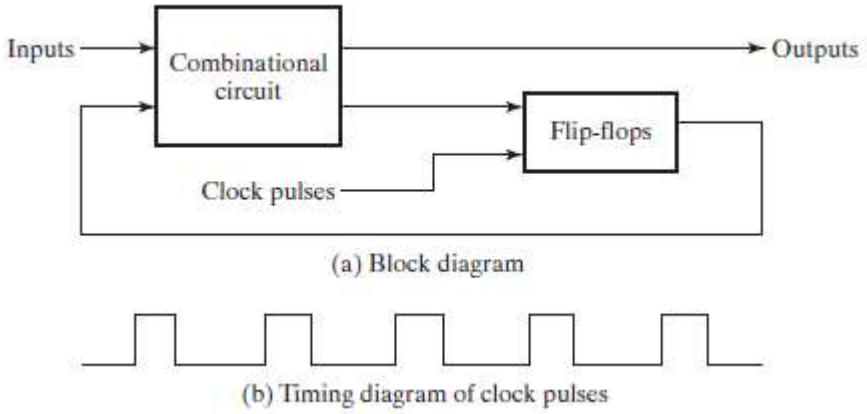
(b) Function table

Fonte: MANO, M.M. Computer System Architecture, Prentice-Hall International, Inc., 1993

Os circuitos sequencias podem ser classificados em dois modelos: Circuitos síncronos e circuitos assíncronos. O comportamento de um circuito síncrono se baseia no fato de que é possível conhecer todos os seus estados em um determinado instante de tempo discreto. Já nos circuitos assíncronos os seus valores são alterados continuamente no tempo.

A partir destes circuitos, pode-se obter algumas conclusões: os circuitos síncronos são mais lentos que os circuitos assíncronos, pois para que mudanças de estados ocorram ele precisa esperar o sinal discreto no tempo. Porém apesar deles serem mais lento que os circuitos assíncronos o modelo síncrono é mais utilizado em circuitos digitais, devido a seu comportamento previsível, acarretando em uma maior facilidade na possibilidade de encontrar possíveis defeitos. Os circuitos assíncronos são utilizados em aplicações bem mais específicas. Abaixo na Figura 6 ilustra-se o modelo de um circuito síncrono.

Figura 6 – Modelo de um Circuito Síncrono



Fonte: MANO, M.M. Computer System Architecture, Prentice-Hall International, Inc., 1993

2.2 LINGUAGENS DESCRITIVAS DE HARDWARE:

Utilizar-se de circuitos combinacionais e sequenciais para abstrair os circuitos digitais em alguns casos pode se mostrar bastante útil. Porém quanto mais complexos e maiores os circuitos digitais, mais complicado se torna a implementação dos mesmos através deste nível de abstração, por isso a utilização das Linguagens Descritivas de Hardware ou **HDL** (*hardware discritive language*). Sendo elas cruciais para o crescimento e desenvolvimento de circuitos digital.

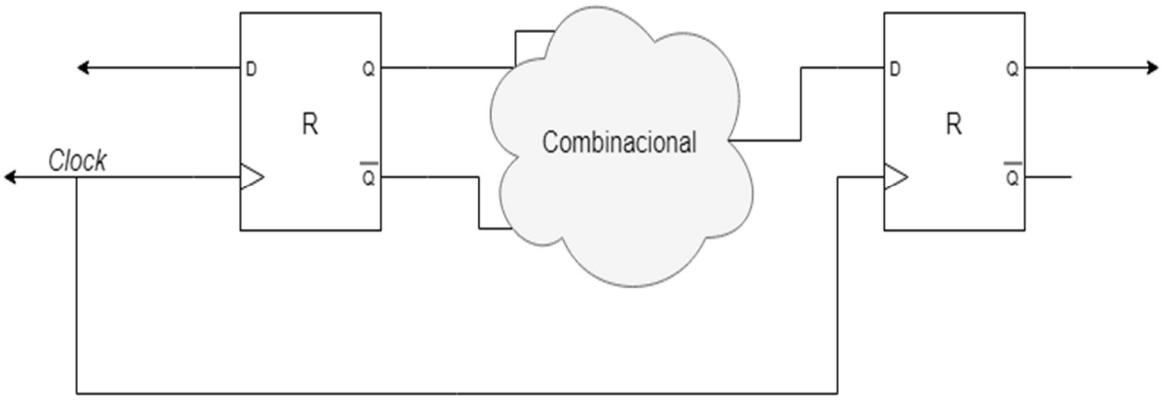
As HDLs são bem semelhantes às linguagens de software convencionais, porém elas têm suas particularidades, em prol de descrever as estruturas e comportamentos dos hardwares nos quais deseja-se implementar. Como sua principal característica temos que as HDLs realizam operações paralelas visando o funcionamento de hardware, diferentemente das linguagens de software, que em sua maioria, representa operações seriais.

Estas linguagens são capazes de representar equações booleanas, tabelas verdades, lógica aritmética, codificadores, operações aritméticas, máquinas de estado e entre outras aplicações, tudo isso em um alto nível de abstração. Graças a esse alto nível de abstração é possível repartir grande circuitos em circuitos menores, agilizando assim o processo de desenvolvimento de hardwares. Note que todas essas descrições podem ser simuladas, desde que elas representem o mesmo sistema em termos de funcionamento, ou seja a saída

esperada deve ser a mesma para um estímulo aplicado na entrada. Essa característica é vital para equipes de verificação.

Apesar de todas as estas características serem mais do que suficiente para utilização de HDLs a sua principal razão delas serem tão importantes no desenvolvimento de hardwares é sua síntese lógica. A descrição de um sistema através de uma HDL pode ser escrita em nível intermediário, conhecido como RTL (*Register Transfer Language*). Em um RTL, o comportamento do circuito é descrito em termos do fluxo de sinais (ou transferência de dados entre os registradores presentes no hardware e as operações lógicas conduzidas com estes sinais. A síntese lógica através de uma biblioteca de componentes converte esta descrição em uma conexão de porta lógicas primitivas que implementa o circuito descrito em RTL. Tornando assim a implementação de lógicas complexas bem mais eficiente. O exemplo de uma estrutura de RTL é exposto na Figura 7 abaixo.

Figura 7 – Modelo estrutural de um RTL



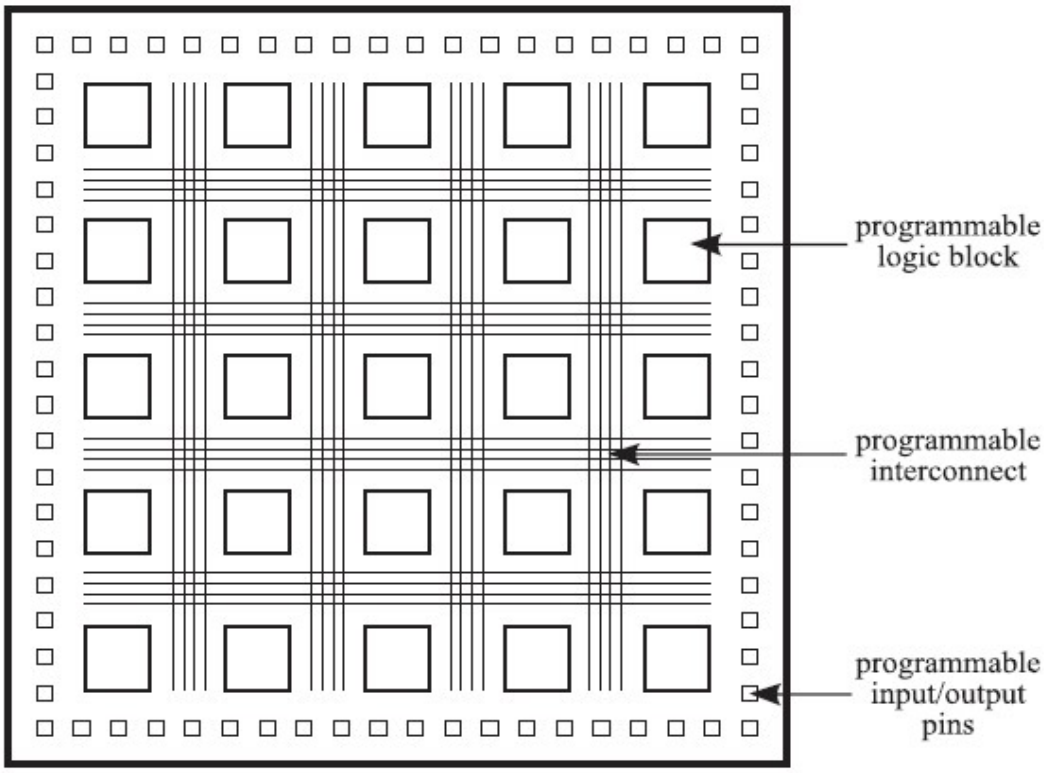
Fonte: Autor (2018)

Atualmente VHDL, Verilog e SystemVeriliog são as linguagens mais usadas para o desenvolvimento de circuitos digitais. Estas linguagens são padronizadas pelo o **IEEE** (*Instute of Electrical and Eletronics Engineers*)**.** Toda implementação destas linguagens deve respeitar este padrão, fazendo assim, com que as implementações desenvolvidas possam ser reutilizadas ou reaproveitadas em outras aplicações independentemente da ferramenta utilizada para compilar, elaborar e sintetizar esta implementação.

3 FPGA

FPGAs (*Field Programmable Gate Array)* são dispositivos de lógica programável, no geral, os FPGAs são conhecidos por possuir três elementos programáveis: blocos de lógicas programáveis (*logic programmable blocks)*, interconexões programáveis (*programmable interconnect*) e pinos de entrada e saída programáveis, como é possível ver na Figura 8 abaixo.

Figura 8 – Diagrama Estrutural de um FPGA



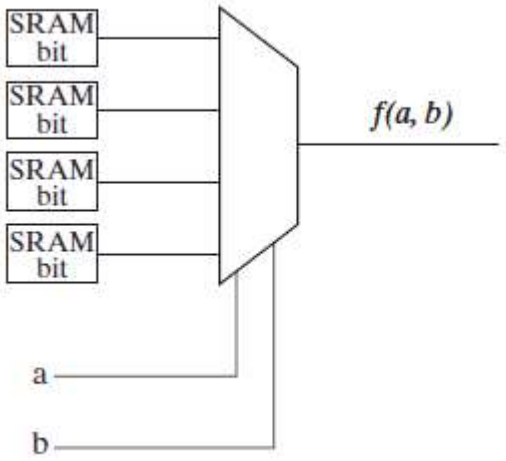
:Fonte: MANO, M.M. Computer System Architecture, Prentice-Hall International, Inc., 1993

Além destas características citadas vários FPGAs possuem blocos especiais de lógica dedicada, como memórias, componentes aritméticos e até mesmo microprocessadores.

3.1 LÓGICA DE BLOCOS PROGRAMÁVEIS

Uma lógica de blocos programáveis contém lógicas combinacionais e sequenciais que podem ser configuradas para serem implementadas de diferentes formas. Grande parte dos FPGAs implementam estas funções combinacionais baseando-se em uma *Look-Up Table* (LUT). Uma LUT é uma memória 2^k x 1 que implementa uma tabela verdade para uma função de *k* variáveis, conhecida como *k*-LUT. A Figura 9 abaixo ilustra uma LUT de duas entradas. Qualquer uma das dezesseis possíveis funções de duas variáveis pode ser implementada ao configurar previamente os bits das SRAMs.

Figura 9 - LUT de Duas Entradas.



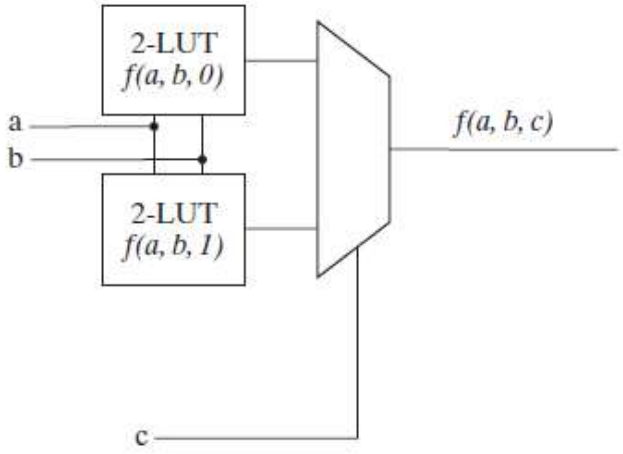
Fonte: MANO, M.M. Computer System Architecture, Prentice-Hall International, Inc., 1993

Caso queira-se implementar funções com mais do que *k*-variáveis, vários *k*-LUT podem ser conectados com multiplexadores. Ao utilizar multiplexadores para combinar LUTs, obedece-se ao teorema da expansão de Shannon, diz que estados de qualquer função Booleana $f(x_1, x_2, x_3, \dots, x_k)$ pode ser expresso como:

$$f(x_1, x_2, x_3, \dots, x_k) = x_k \cdot f(x_1, x_2, x_3, \dots 1) + \overline{x_k} \cdot f(x_1, x_2, x_3, \dots 0)$$

A Figura 10 abaixo ilustra a implementação de três funções combinando dois 2-LUT, com um multiplexador.

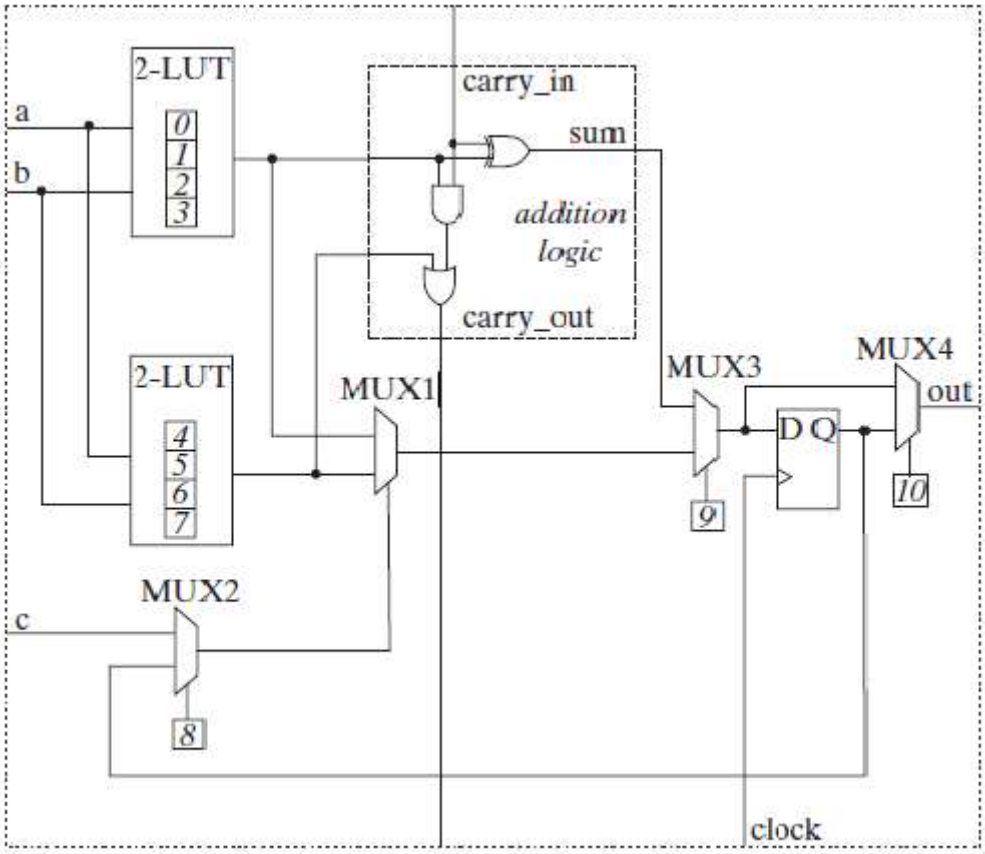
Figura 10 – Implementação de três funções combinando dois 2-LUTs com um multiplexador



Fonte: MANO, M.M. Computer System Architecture, Prentice-Hall International, Inc., 1993

Além de LUTs blocos de lógica programável, também podem conter: multiplexadores, *flip-flops* entre outras lógicas para possibilitar a configuração dos blocos, e assim, prover uma ampla variedade de funções. A Figura 11 abaixo ilustra uma implementação de um bloco lógico programável.

Figura 11- Implementação de um Bloco Lógico Programável



Fonte: MANO, M.M. Computer System Architecture, Prentice-Hall International, Inc., 1993

Neste bloco temos cinco principais características:

1. Um par de 2-LUTs implementando funções combinacionais;
2. Um flip-flop tipo D implementando funções sequenciais;
3. Um somador completo de 1-bit;
4. Conjunto de multiplexadores que selecionam qual funcionalidade o bloco irá exercer;
5. Conjunto de memórias SRAM de configurações que controlam os LUTs e os multiplexadores, representadas pelos números de 0 a 10 dento dos quadrados.

3.2 LÓGICA DE INTERCONEXÕES PROGRAMÁVEIS

Uma rede de interconexões programáveis seleciona as conexões entre blocos lógicos de forma a criar circuitos, que seriam grandes, caso fosse implementado apenas em um bloco lógico. Esta rede é composta de um conjunto de fios e chaves programáveis.

Geralmente estas chaves consiste de um único transistor nMOS e assim como nos blocos de lógica programável, o *gate* do transistor é controlado usando um bit de configuração. Estas conexões devem percorrer os FPGAs afim de ser capaz de implementar uma ampla quantidade de tipos de circuitos.

Com o intuito de minimizar problemas de sincronismo devido ao atraso de propagação e *skew*, implementa-se conexões dedicadas para sinais de *clock* e *reset* compartilhados globalmente.

3.3 LÓGICA DE ENTRADAS E SAÍDAS (*I/O*) PROGRAMÁVEL

Os FPGAs precisam ser conectados com o mudo de fora, particularmente um FPGA precisa ser capaz de prover uma ampla gama de entrada e saídas dependendo de qual circuito irá ser implementado e então o FPGA precisa ser compatível com os requisitos de tensão e velocidade dos componentes elétricos que irão ser conectados. Devido a esses dois parâmetros os FPGAs precisam possuir um largo número de pinos que sejam configuráveis afim de suportar um abrangente número de padrões de interfaces elétricas.

Estas interfaces elétricas são caracterizadas pela tensão de nível lógico, corrente fornecida ou absorvida, velocidade em que sinal pode comutar e entre várias outras características que um pino de entrada e saída pode possuir. Estas configurações de FPGAs são feitas de acordo com o fornecedor e quais são as aplicações para qual deseja-se utilizar o dispositivo.

4 LINGUAGEM TCL

A linguagem de programação TCL (*Tool Command Language*) é uma linguagem *open-source* interpretativa, capaz de prover facilidades, tais como variáveis, procedimentos, estruturas de controles bem como outras inúmeras características não encontradas em outras linguagens. Se é possível rodar TCL em praticamente todos os sistemas operacionais conhecidos atualmente, tais como Unix, Macintosh e Windows.

Esta linguagem é bastante flexível e é utilizada em uma grande gama de aplicações, tais como: interação automatizada com programas externos, embarcar bibliotecas em programas de aplicação, linguagens descritivas de hardware e *scripts* em geral.

TCL foi originalmente desenvolvida como um comando de linguagem reutilizável para ferramentas **CAD** *(Computer Aided Design)*, na qual até hoje é bastante utilizada na criação de scripts e automatização no processo e projeto de elaboração de circuitos digitais integrados. O interpretador é implementado como uma biblioteca em C que pode ser anexada em qualquer aplicação, é bastante simples adicionar novas funções ao interpretador. Assim pode-se considerar o TCL como uma "linguagem macro" que pode ser utilizada em várias aplicações.

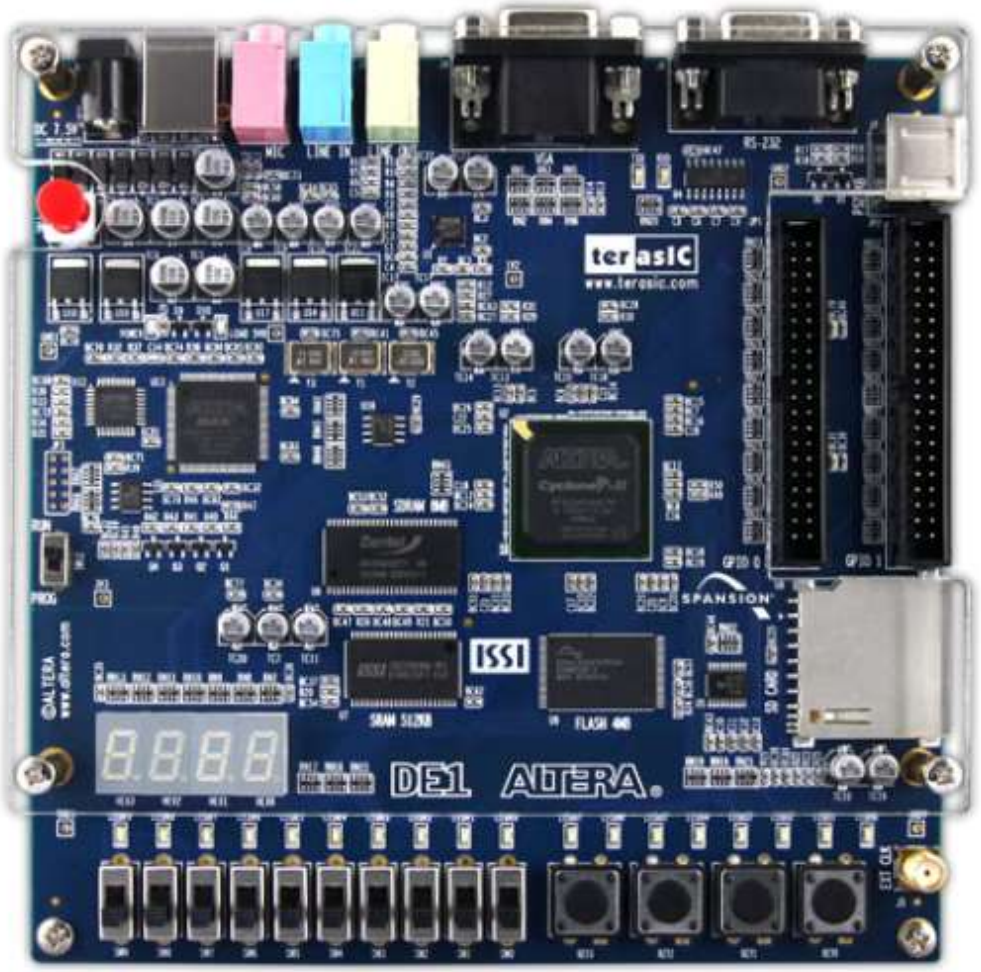
4.1 TCL/TK

TK (*Tool Kit)* é a mais popular extensão da linguagem TCL, voltada para interfaces gráficas, na qual é desenvolvida para se obter um alto nível de abstração entre o usuário e determinada aplicação. TK é bastante conhecida por ser uma GUI (*Graphical User Interface)* não apenas para TCL, mas para outras inúmeras linguagens, capaz de produzir uma grande variedade de aplicações sendo possível rodar através de quase todas s sistemas operacionais sem precisar realizar alterações.

Algumas de suas principais funções são criar janelas, na quais são constituídas de *widgets* que são controlados por funções geométricas (capazes de desenhar formas geométricas para interagirem com estes *widgets*) e até iterações com periféricos (como mouse e teclado) capazes de acionar eventos desejados.

Neste projeto foi utilizado TCL/TK afim de abstrair os periféricos, mais utilizados nas disciplinas de Laboratório de Arquitetura e Circuitos Digitais, nos quais sãos as chaves, botões, LEDs e o display de sete segmentos. O FPGA utilizado nos laboratórios é a DE1 Cyclone® como mostra a Figura 12 abaixo:

Figura 12 - DE1 Cyclone®



Fonte: < http://www.anita-simulators.org.uk>

4.1.1 BOTÃO *PUSH*

Para implementar estes botões utilizou-se o comando o ***button*** (Figura 13):

Figura 13 - Comando ***Button***

```
button .fpga.sw.b1  -text "KEY1" -command Reset_proc1
pack   .fpga.sw.b1  -side "left"   -pady 1 -padx 1 -fill x
```

Fonte: Autor (2018)

Assim como várias outras linguagens o comando ***button*** funciona como uma função onde o usuário passa os parâmetros desejados, neste caso os parâmetros utilizados foram o *–text* e o *–command* onde um define uma caixa de texto no botão e outra define a função que irá ser realizada ao apertar o respectivo botão, estas funções serão explicadas mais à frente. Assim foram implementados os quatros botões como mostra a Figura 14 abaixo:

Figura 14 – Implementação dos Botões

Fonte: Autor (2018)

4.1.2 CHAVES

A implementação da chave é bem análoga ao botão, para se implementar as chaves existentes no DE01, utilizou-se o comando ***checkbutton*** (Figura 15):

Figura 15 - Comando ***Checkbutton***

```
145 checkbutton .fpga.sw.c0 -text "SW0" -command look_valueSW0 -offvalue false -onvalue true -variable "SW0"
146 pack        .fpga.sw.c0 -side "left" -pady 1 -padx 1 -fill x
147
```

Fonte: Autor (2018)

Os parâmetros *–text* e o *-command*, tem a mesma funcionalidade do comando anterior, o parâmetro *–variable* define o nome da variável que irá armazenar o valor atual do estado da chave, e os parâmetros *-offvalue* e *-onvalue* definem os valores que irão

representar os estados das variáveis. Segue abaixo, na Figura 16, a implementação das dez chaves existentes na DE01.

Figura 16 – Implementação de chaves

Fonte: Autor (2018)

No exemplo da Figura 16 quando a chave não está selecionada o valor da variável SW0 será *false* enquanto que quando ela está selecionada, o valor da variável SW1 será *true* devido aos comandos *–offvalue* e *-onvalue.*

4.1.3 LEDs

Para implementar os *LED*s foi-se necessário usar o comando *canvas* (Figura 17), este comando se usa para desenhar formas geométricas, que para o *LED* será a forma oval.

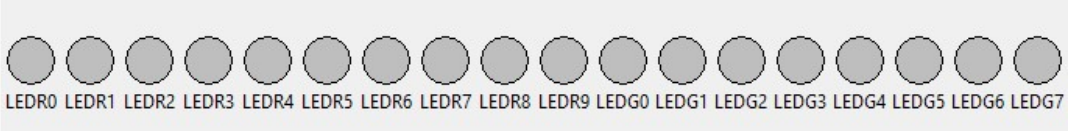
Figura 17 – Comando *canvas*

```
29
30  canvas .fpga.leds.lr0  -width 1c   -height 2c
31  pack .fpga.leds.lr0    -fill both  -side "left"
32  .fpga.leds.lr0  create oval 0.1c 0.1c 0.9c 0.9c -fill gray -tag lr0
33  .fpga.leds.lr0 create text 0.5c 1.2c -text "LEDR0"
34
```

Fonte: Autor (2018)

No comando *canvas* se cria o *frame* onde irá realizar o desenho e em seguida é utilizado o comando *create oval* onde se é desenhado o *LED* o comando parâmetro *fill* escolhe a cor que preenche o *LED* e o *tag* é a variável que representa essa forma geométrica. Na Figura 18 abaixo ilustra a interface representativa dos *LED*s:

Figura 18 – Interface representativa dos LEDs



Fonte: Autor (2018)

4.1.4 DISPLAY DE SETE SEGMENTOS

Da mesma forma que se usou a função *canvas* para o *LED* foi-se utilizada para desenhar o display de sete segmentos, como é mostrado na Figura 19 abaixo:

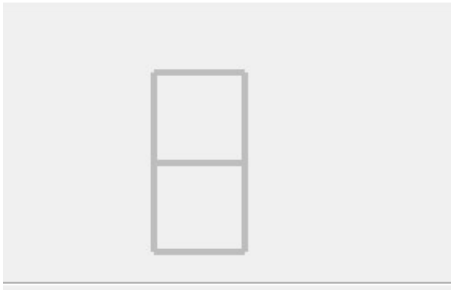
Figura 19 – Função canvas para desenho do display de sete segmentos

```
328  canvas .fpga.ss.ss_cv1 -height 4c -width 8c
329  pack .fpga.ss.ss_cv1  -fill both -side left
330  .fpga.ss.ss_cv1 create line 2.5c 0.5c 4c 0.5c -fill gray -width 4 -tag ss1_a
331  .fpga.ss.ss_cv1 create line 4c 0.5c 4c 2c  -fill gray -width 4 -tag ss1_b
332  .fpga.ss.ss_cv1 create line 4c 2c 4c 3.5c -fill gray -width 4 -tag ss1_c
333  .fpga.ss.ss_cv1 create line 2.5c 3.5c 4c 3.5c -fill gray -width 4 -tag ss1_d
334  .fpga.ss.ss_cv1 create line 2.5c 3.5c 2.5c 2c   -fill gray -width 4 -tag ss1_e
335  .fpga.ss.ss_cv1 create line 2.5c 0.5c 2.5c 2c -fill gray -width 4 -tag ss1_f
336  .fpga.ss.ss_cv1 create line 2.5c 2c 4c 2c -fill gray -width 4 -tag ss1_g
337
```

Fonte: Autor (2018)

Onde para desenhar as linhas **a, b, c, d, e, f** e **g** do display aplica-se o *create line* para cada um dos mesmos. Assim foi possível desenhar os displays de sete segmentos. Como é possível ver na Figura 20 abaixo.

Figura 20 – Display de Sete Segmentos

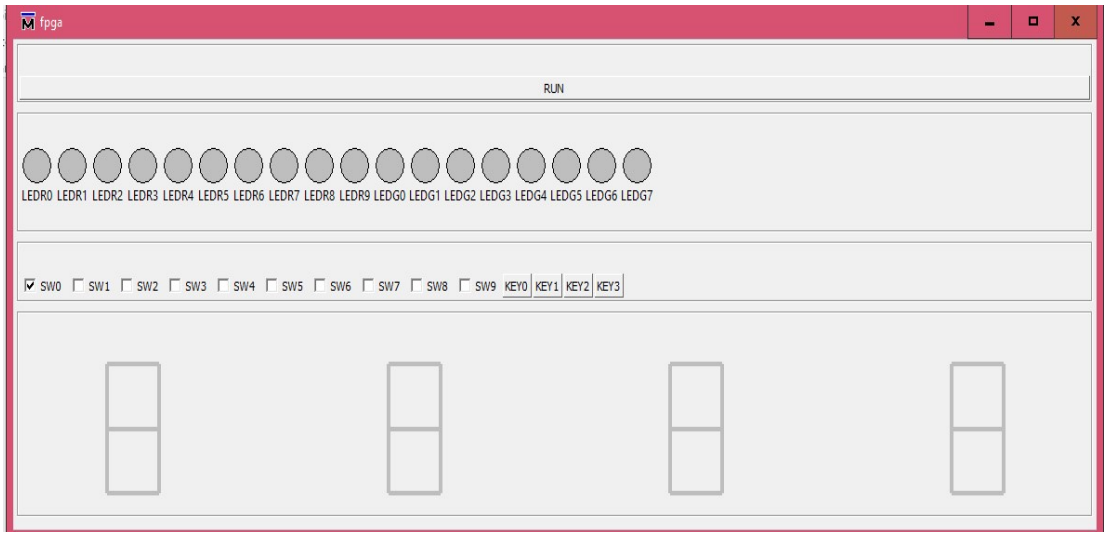


Fonte: Autor (2018)

4.2 INTERFACE FINAL

E a partir das combinações destes componentes foi-se possível abstrair um modelo (Figura 21) de um FPGA DE01 e suas principais funções utilizadas nas disciplinas de Laboratório de Arquitetura de Sistemas Digitais e Laboratório de Circuitos Lógicos.

Figura 21 - Modelo Final da Interface



Fonte: Autor (2018)

5 SIMULAÇÃO E TESTBENCHS

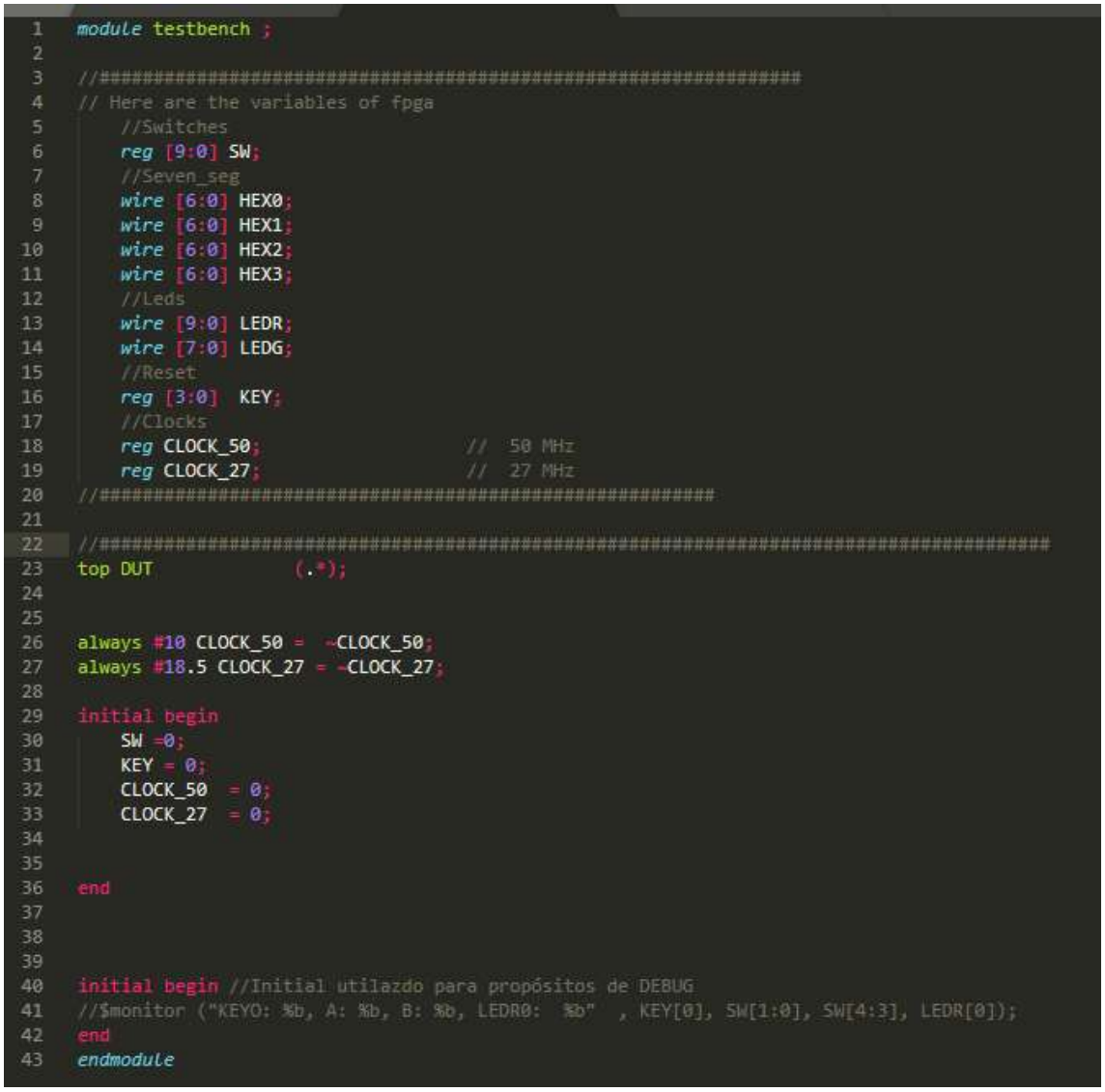
Devido à grande complexidade dos hardwares descritos de formas eficientes em HDL, se faz necessário o uso de estrutura especial elaborada em HDL conhecida como ***testbench***. O *testbench* é uma descrição que inclui o projeto a ser testado tipicamente chamado de DUT (*Design Under Test)*. A principal diferença de um *testbench* para uma descrição em HDL convencional, é que ele apresenta funções que descrevem hardwares bem como funções que descrevem softwares. Onde são aplicados valores nas entradas do DUT e a partir dessas funções obtém-se as saídas. A construção destes *testbenchs* provê

maior flexibilidade na hora de verificar e validar funcionalmente de grandes projetos de circuitos integrados.

Para este projeto foi-se elaborado um *testbench* no qual suas varáveis se comunicam diretamente com o *script* em TCL/TK e os comandos de simulação do ModelSim®.

Como é possível ver na Figura 22 abaixo as variáveis foram criadas como o mesmo nome do Mod_Teste existente no laboratório afim de facilitar o uso para os alunos, com isto, o aluno irá instanciar o seu topo ao DUT e as variáveis na qual deseja usar, caso ele queira associar uma chave ao *LED*, por exemplo, ele deve usar um *assign*, assim como é feito no laboratório. No *initial* a inicialização das entradas se faz necessário, pois como é uma simulação, caso não se faça esta inicialização pode acarretar em erros devido a propagação de “X”s.

Figura 22 – *Testebench* utilizado para se comunicar com a interface



```
module testbench ;

//=================================================================
// Here are the variables of fpga
    //Switches
    reg [9:0] SW;
    //Seven_seg
    wire [6:0] HEX0;
    wire [6:0] HEX1;
    wire [6:0] HEX2;
    wire [6:0] HEX3;
    //Leds
    wire [9:0] LEDR;
    wire [7:0] LEDG;
    //Reset
    reg [3:0]  KEY;
    //Clocks
    reg CLOCK_50;                  //  50 MHz
    reg CLOCK_27;                  //  27 MHz
//=========================================================

//=============================================================================================
top DUT             (.*);


always #10 CLOCK_50 =  ~CLOCK_50;
always #18.5 CLOCK_27 = ~CLOCK_27;

initial begin
    SW =0;
    KEY = 0;
    CLOCK_50  = 0;
    CLOCK_27  = 0;


end



initial begin //Initial utilazdo para propósitos de DEBUG
//$monitor ("KEYO: %b, A: %b, B: %b, LEDR0:  %b"  , KEY[0], SW[1:0], SW[4:3], LEDR[0]);
end
endmodule
```

Fonte: Autor (2018)

6 MODO DE FUNCIONAMENTO

A partir do *testbench* conecta-se através do ModelSim® a interface em TCL/TK com o mesmo. E então ao aluno interagir com os periféricos da interface, no quais estão associados ao *testbench,* o *script* irá identificar estas mudanças e assim utilizar comandos de simulação do ModelSim® para alterar os valores das variáveis do *testbench* e assim interagir novamente com o *script*.

6.1 TCL E MODELSIM®

Como dito anteriormente a comunicação entra as variáveis do *testbench* e ModelSim® se dá através do TCL, e irá ser explanado a seguir como que é realizada esta interação.

6.1.1 COMANDO *PROC*

Como visto anteriormente os botões e chaves tinha como parâmetro o *–command* neste parâmetro foi passado alguns *proc*s, que são nada mais do que funções que realizam procedimentos quando instanciadas.

Adicionou-se *procs* para os comandos da chave e dos botões, na Figura 23 abaixo pode-se ver a implementação do *–command* para chave.

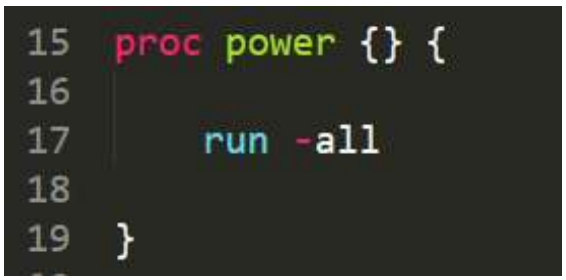
Figura 23 – Implementação do comando para as chaves

```
129 proc look_valueSW0 {} {
130     global SW0
131         if { $SW0 == true } {
132 
133             force  -freeze /SW\[0\] 1
134 
135 
136 
137         } else {
138 
139             force  -freeze /SW\[0\] 0
140 
141 
142 
143         }
144 }
```

Fonte: Autor (2018)

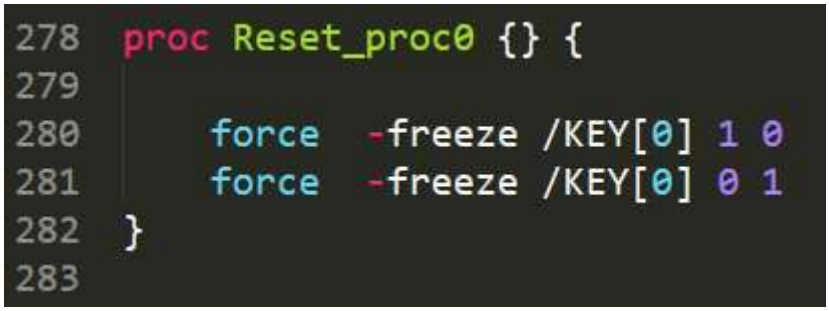
Neste caso verifica-se o status da chave na interface, e de acordo com este, altera-se o valor da variável que representa a chave no *testbench* através do comando de simulação do ModelSim® *force*. Implementou-se também para os botões como pode-se ver nas Figura 24 e Figura 25 abaixo:

Figura 24 – Implementação do Botão *run*



Fonte: Autor (2018)

Figura 25- Implementaçâo dos botões *KEY*



Fonte: Autor (2018)

Para os botões foram criados dois procs um para iniciar a simulação através do comando de simulação *run* do ModelSim®, e outro para os botões do FPGA, da mesma forma das chaves, utilizou-se o comando de simulação force para alterar os valores no *testbench*.

6.1.2 COMANDO *WHEN*

Para os LEDs e displays, primeiro verifica-se o valor da variável no *testbench* e depois a interface responde, seja acendendo um LED ou uma linha do display. Para verificar o valor das variáveis utiliza-se o comando do ModelSim® *when* no qual tem um funcionamento bem próximo ao um *if* de uma linguagem qualquer de programação.

Como é possível observar nas Figura 26 e Figura 27 abaixo de acordo com os valores das variáveis no *testbench* os periféricos vão reagir diferentemente. Para alterar a saída na interface utilizou-se o comando *itemconfigure* do TCL/TK.

Figura 26 – Implementação do Comando *when* para os LEDs

```
377  when {/LEDR[0] = 1 } {
378      .fpga.leds.lr0 itemconfigure lr0 -fill red
379  }
380  when {/LEDR[0] = 0 } {
381      .fpga.leds.lr0 itemconfigure lr0 -fill gray
382  }
383
```

Fonte: Autor (2018)

Figura 27 – Implementação do Comando *when* para o Display de Sete Segementos

```
524  when {/HEX0[1] = 1} {
525
526      .fpga.ss.ss_cv1 itemconfigure ss1_b -fill red -width 4
527  }
528
529  when {/HEX0[1] = 0} {
530
531      .fpga.ss.ss_cv1 itemconfigure ss1_b -fill gray -width 0
532  }
```

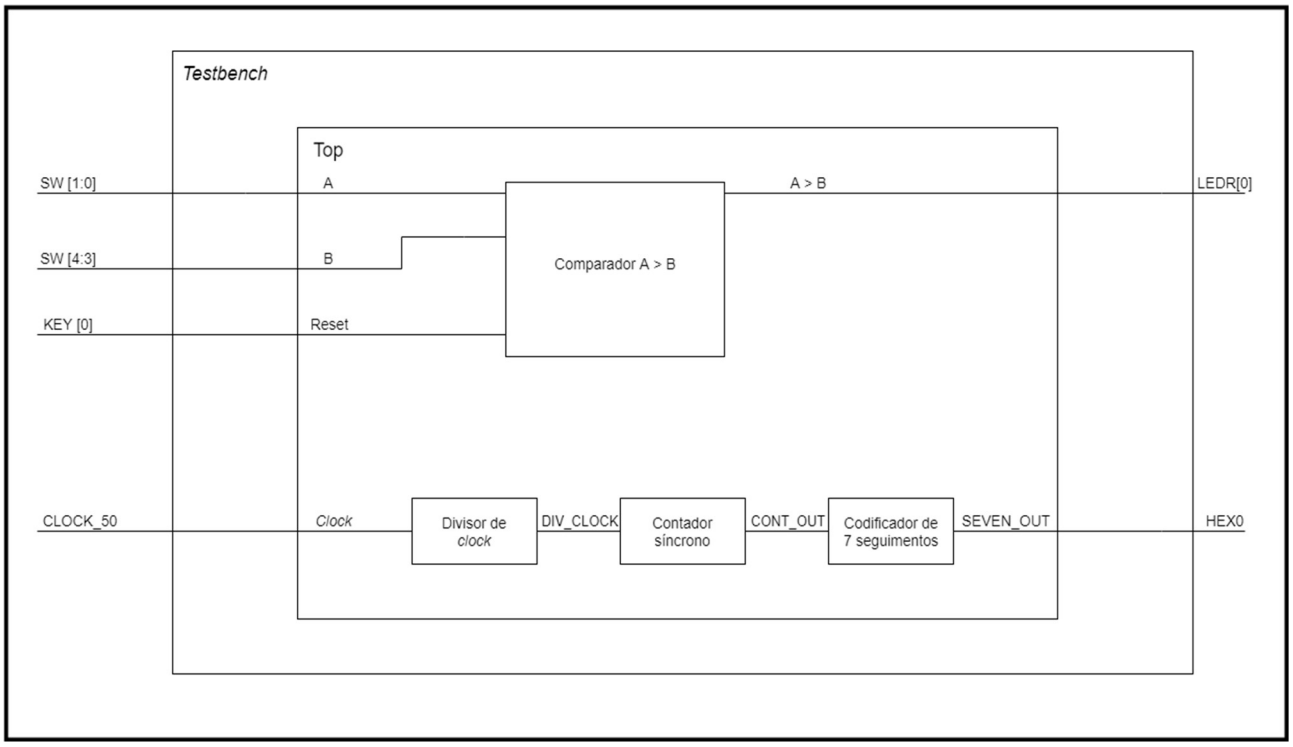
Fonte: Autor (2018)

7 RESULTADOS OBTIDOS

Afim de se testar a interface e verificar seu devido funcionamento, implementou-se alguns circuitos simples e instanciou-se o mesmo no *testbench.* Os circuitos implementados foram: um circuito combinacional comparador entre dois números de dois bits onde a saída é 1 se o número “A” for maior que o “B” e 0, caso contrário, o codificador para o display de sete segmentos onde a entrada é de 4 bits e a saída é 7 bits, um contador síncrono e um divisor de *clock* tendo em vista que os relógios do FPGA são de 50 e 27 MHz.

Todos estes circuitos foram instanciados em um topo e em seguida este topo foi instanciando no *testbench* na Figura 28 abaixo temos o modelo de como realizou-se as conexões:

Figura 28 – Modelo de Conexões



Fonte: Autor (2018)

Para representar “A” um número de dois bits utiliza-se as chaves 0 e 1, para representar o número “B”, também de dois bits, utilizou-se as chaves 3 e 4, para o reset utilzou-se o botão KEY0 e o *clock* de 50 MHz. Na saída tem-se o display “HEX0” conectado a saída de um codificador de sete segmentos, no qual a entrada é a saída do contador síncrono.

Afim de verificar os valores e o funcionamento de outros periféricos ligou-se as chaves 0 e 1 aos LEDs verde 0 e 1 e as chaves 3 e 4 foram ligadas aos LEDs verdes 3 e 4 alémd de ligar o HEX0 aos HEX1, HEX2 e HEX3. Para isto foi utilizado o comando *assign* possível ver na Figura 29 abaixo:

Figura 29 - Comando *assign*

```
30   assign LEDG[1:0] = SW[1:0] ;
31   assign LEDG[4:3] = SW[4:3] ;
32   assign HEX1      = HEX0    ;
33   assign HEX2      = HEX0    ;
34   assign HEX3      = HEX0    ;
35
```

Fonte: Autor (2018)

Afim de comparar os resultados também utilizou-se o comando ***monitor*** como pode-se ver na Figura 30 abaixo:

Figura 30 - Comando ***monitor***

```
$monitor ("Time: %d, Reset: %b, A: %b, B: %b, LEDR0:  %b, HEX0: %b,"  , $time, KEY[0], SW[1:0], SW[4:3], LEDR[0], HEX0);
```

Fonte: Autor (2018)

A partir desta implementação foi possível verificar o funcionmento da interface e obter os seguintes resultados abaixo.

Figura 31- Resultados Obtidos

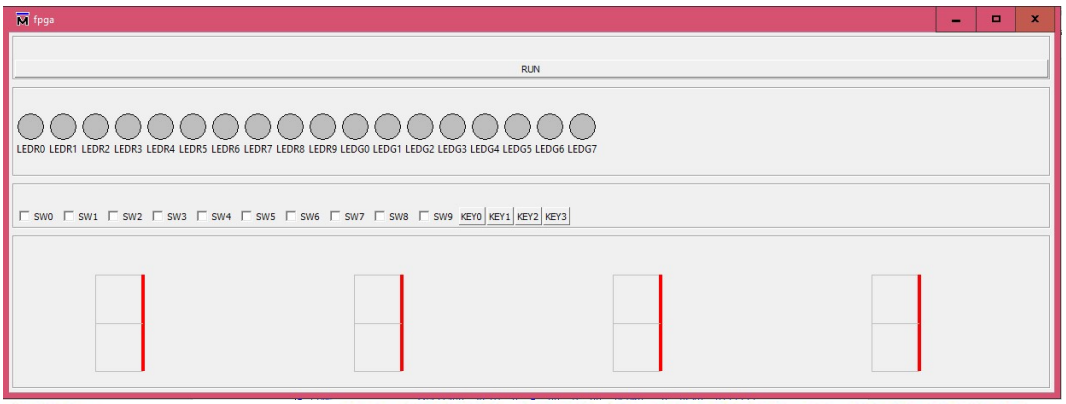
```
# Time:                    0, KEY0: 0, A: 00, B: 00, LEDR0:  x, HEX0: xxxxxxx,
# Time:             26957979, KEY0: 1, A: 00, B: 00, LEDR0:  0, HEX0: 0111111,
# Time:             26957980, KEY0: 0, A: 00, B: 00, LEDR0:  0, HEX0: 0111111,
```

Fonte: Autor (2018)

Na Figura 31 acima tem-se os valores logo após apertar o botão de KEY0 os valores são resetados e inicia-se o contador síncrono, para este caso o divisor de *clock* foi

de um período de 20 nano segundos para um período de um segundo, as figuras abaixo ilustram o primeiro pulso de *clock*.

Figura 32- Resultado obtido para o primeiro puslo de *clock*




```
# Time:                0ns, KEY0: 0, A: 00, B: 00, LEDR0:  x, HEX0: xxxxxxx,
# Time:         14547768ns, KEY0: 1, A: 00, B: 00, LEDR0:  0, HEX0: 0111111,
# Time:         14547769ns, KEY0: 0, A: 00, B: 00, LEDR0:  0, HEX0: 0111111,
# Time:        514547750ns, KEY0: 0, A: 00, B: 00, LEDR0:  0, HEX0: 0000110,
```

Fonte: Autor (2018)

Na Figura 32 acima é possível visualizar o primeiro pulso de *clock* e verificar o valor no display mudando, nota-se que neste caso o primeiro pulso de *clock* foi em 0,5 segundos isto ocorre porque o valor inical do *clock* no *reset* é 0, portanto meio segundo depois o valor do *clock* é igual à 1. Porém a partir do segundo pulso o valor do próximo puslo de *clock* na borda de subida irá ocorrer após um segundo como é possível ver na

```
# Time:                0ns, KEY0: 0, A: 00, B: 00, LEDR0:  x, HEX0: xxxxxxx,
# Time:         14547768ns, KEY0: 1, A: 00, B: 00, LEDR0:  0, HEX0: 0111111,
# Time:         14547769ns, KEY0: 0, A: 00, B: 00, LEDR0:  0, HEX0: 0111111,
# Time:        514547750ns, KEY0: 0, A: 00, B: 00, LEDR0:  0, HEX0: 0000110,
# Time:       1514547750ns, KEY0: 0, A: 00, B: 00, LEDR0:  0, HEX0: 1011011,
# Time:       2514547750ns, KEY0: 0, A: 00, B: 00, LEDR0:  0, HEX0: 1001111,
```

abaixo.

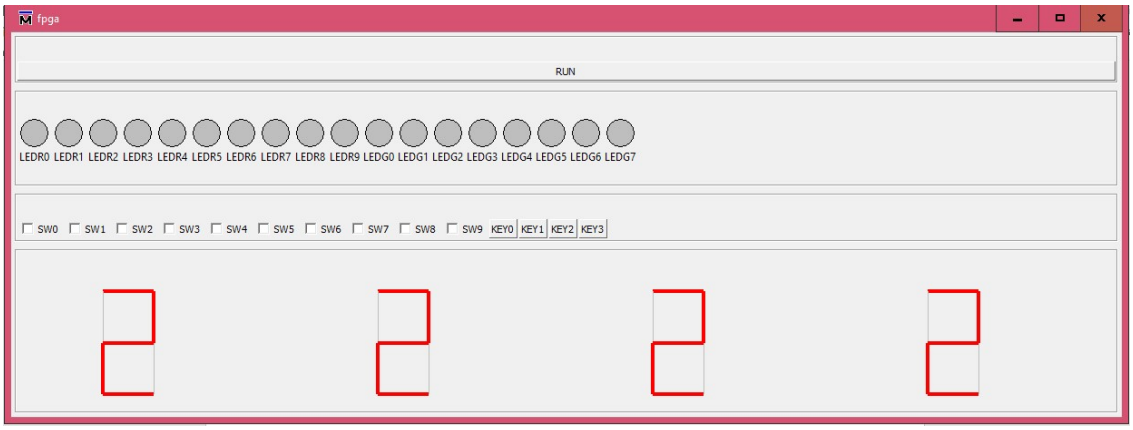


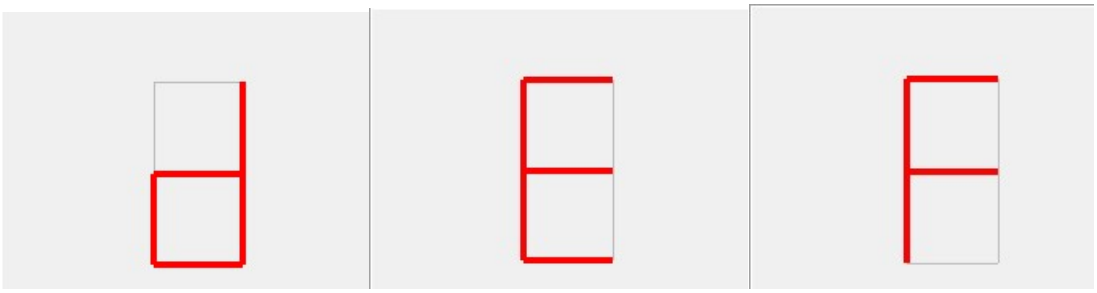
Figura 33 - Resultado Obtido para o Número 2

```
# Time:                    0ns, KEY0: 0, A: 00, B: 00, LEDR0:  x, HEX0: xxxxxxx,
# Time:             14547768ns, KEY0: 1, A: 00, B: 00, LEDR0:  0, HEX0: 0111111,
# Time:             14547769ns, KEY0: 0, A: 00, B: 00, LEDR0:  0, HEX0: 0111111,
# Time:            514547750ns, KEY0: 0, A: 00, B: 00, LEDR0:  0, HEX0: 0000110,
# Time:           1514547750ns, KEY0: 0, A: 00, B: 00, LEDR0:  0, HEX0: 1011011,
# Time:           2514547750ns, KEY0: 0, A: 00, B: 00, LEDR0:  0, HEX0: 1001111,
```

Fonte: Autor (2018)

E seguiu-se assim até completar o número “F” do sistema hexadecimal, como mostram as figuras abaixo.

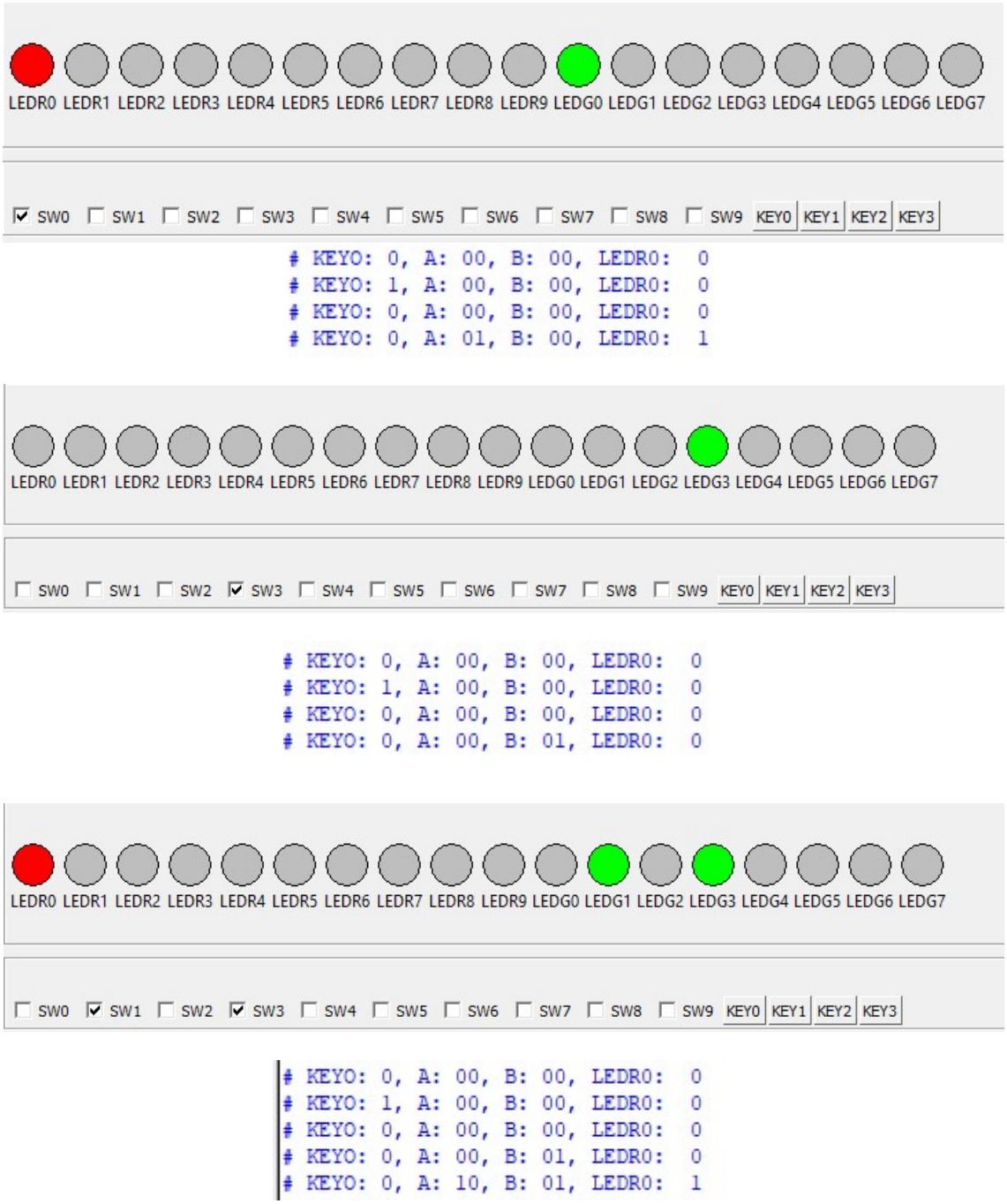
Figura 34 - Resultado obtido para os números **d, e** e f.



Fonte: Autor (2018)

Em seguida, afim de testar o funcionamento do circuito comparador A maior que B e comparou-se os resultados obtidos através do *monitor* com os resultados obtidos na interface. É possível verificar os resultados nas figuras abaixo:

Figura 35 - Resultados obtidos para o circuito comparador



Fonte: Autor (2018)

A partir da Figura 35 acima é possível comparar os resultados da interface com os resultados obtidos no monitor e assim, garantir o funcionamento da mesma

8 CONCLUSÃO E TRABALHOS FUTUROS

A partir dos resultados obtidos pode-se concluir que a implementação de uma interface gráfica que vise simular os periféricos de um dispositivo FPGA, foi-se realizada com sucesso e com isso pode-se dizer que já é possível implementar a mesma nos laboratórios das disciplinas nas quais a interface será utilizada.

Desta forma aluno será capaz de realizar grande parte dos experimentos propostos nos laboratórios das disciplinas em sua residência ou qualquer outro lugar. Além de que isto ainda pode instigar o mesmo com relação à utilização de *testebenchs* e simulação tornando-o capaz de realizar simulações e testes sem a necessidade de um dispositivo FPGA.

Deve-se levar em conta que a apesar de a ferramenta de simulação utilizada ter sido o ModelSim®, como a interface foi criada através da linguagem TCL/TK, é possível então, utilizar qualquer outra ferramenta de simulação mudando apenas os comandos de simulação que mudam de acordo com a ferramenta.

É importante salientar que apesar de se estar tentando simular a interface de um FPGA, que ainda se trata de uma simulação de circuitos digitais, portanto deve-se levar em conta alguns detalhes como o fato de o tempo de simulação ser diferente de o tempo "real" isso quer dizer que o tempo de um *clock* de 50MHz é diferente de um tempo de *clock* de 50MHz simulado. Outra coisa que se faz bastante necessário é inicialização de variáveis (criação de *resets*), pois caso não implementado pode levar a propagação de **X**s acarretando em um mal funcionamento do circuito digital elaborado.

Como trabalhos futuros, pode-se recomendar a implementação de uma interface melhorada com adição de imagens e *frames* fazendo com que a mesma se aproxime cada vez mais da aparência de um FPGA mesmo. Outra proposta seria implementar outros modelos de FPGA com mais periféricos, um exemplo disso seria a adição de um monitor de LCD, existente em alguns modelos de FPGAs.

9 REFERÊNCIAS BIBLIOGRÁFICA

- MANO, M.M. Computer System Architecture, Prentice-Hall International, Inc., 1993

- IDOETA, I.V e CAPUANO, F.G - Elementos de eletrônica digital, Ed.

- Art of Writing TestBenches (2014). Disponível em <www.asic-world.com/verilog/art_testbench_writing.html>

- MentorGraphics (2003). ModelSim® *SE Tutorial*.

- MentorGraphics (2005). ModelSim® Command Reference Manual

- About Tcl/TK. Disponível em <www.tcl.tk/about>

- What is Tcl Tk. Disponível em < http://www.tcltk.com>

- Tcl/Tk Buttons. Disponível em <www.doulos.com/knowhow/tcltk/examples/buttons>