

**UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA**



**INTERFACEAMENTO DE ENTRADA E SAÍDA EM
APLICAÇÕES COM USO DE FPGA**

Tiago Samir de Sousa Freire

Fortaleza
Dezembro de 2010

TIAGO SAMIR DE SOUSA FREIRE

**INTERFACEAMENTO DE ENTRADA E SAÍDA EM
APLICAÇÕES COM USO DE FPGA**

Monografia submetida à Universidade Federal do Ceará como parte dos requisitos necessários para a obtenção do grau de Engenheiro Eletricista.

Orientador: Prof. Msc. Paulo Peixoto Praça.

Fortaleza
Dezembro de 2010

TIAGO SAMIR DE SOUSA FREIRE

**INTERFACEAMENTO DE ENTRADA E SAÍDA EM
APLICAÇÕES COM USO DE FPGA**

Esta monografia foi julgada adequada para obtenção do título de Engenheiro Eletricista e aprovada em sua forma final pela Coordenação de Graduação em Engenharia Elétrica na Universidade Federal do Ceará.

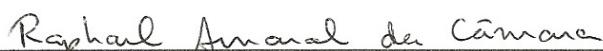


Tiago Samir de Sousa Freire

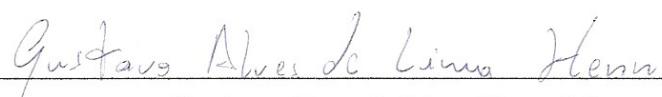
Banca Examinadora:



Prof. Paulo Peixoto Praça, Msc.
Presidente



Raphael Amaral da Câmara, Msc.



Gustavo Alves de Lima Henn, Msc.

Fortaleza, Dezembro de 2010

AGRADECIMENTOS

Primeiramente a Deus, pelo dom da vida e pela chance que me foi concedida.

Ao professor Msc. Paulo Peixoto Praça, pela sua orientação, paciência, amizade e disponibilidade durante todo este tempo. Agradeço pela oportunidade de trabalhar e aprender mais nessa área, de minha grande admiração, da eletrônica digital.

Ao técnico Sr. Jorge, que me deu força, incentivo e sempre me recebeu muito bem, com todo o auxilio necessário, nas várias e várias horas de montagens e testes no laboratório.

A todos os outros professores do Departamento de Engenharia Elétrica da UFC e aos demais funcionários do departamento responsáveis diretamente ou indiretamente pela minha formação acadêmica.

Aos meus colegas de curso, remanescentes ou não, que sempre estiveram ao meu lado e me deram apoio durante toda a graduação.

À minha família, a todos os meus amigos e a minha namorada Ana Paula, por todo suporte e pela ajuda em todos os momentos da minha vida acadêmica, bons ou ruins e que compreenderam minha ausência durante os períodos de dedicação aos estudos. Todos são muito importantes para mim.

Freire, T. S. S. “INTERFACEAMENTO DE ENTRADA E SAÍDA EM APLICAÇÕES COM USO DE FPGA”, Universidade Federal do Ceará – UFC, 2010, 100p.

Este trabalho apresenta algumas das principais características dos FPGAs (*Field Programmable Gate Array*), mais especificamente do kit de desenvolvimento da família Cyclone II do fabricante Altera, objetivando mostrar a grande aplicabilidade, diversidade e eficiência no uso desses dispositivos de baixo consumo de energia. Foi realizado um estudo da interface homem-máquina (IHM) no FPGA, a partir de uma análise dos protocolos e metodologia de reconhecimento da entrada de dados, por teclado, e também da comunicação externa do mesmo, através de uma saída de vídeo VGA (*Video Graphics Array*), do kit de desenvolvimento, para um monitor comum de tubos de raios catódicos ou CRT (*Cathode Ray Tubes*). Como resultados experimentais, foram criados blocos de programa, em linguagem VHDL, a serem gravados no FPGA para simular um jogo muito popular intitulado TETRIS®, tendo como entrada dos comandos o teclado PS/2 e, o monitor como interface de saída, em cores RGB (*Red-Green-Blue*), para o usuário. O software QUARTUS II® é o ambiente de desenvolvimento usado na criação dos códigos, dos blocos e na programação do dispositivo lógico estudado.

Palavras-Chave: Interface, FPGA, teclado PS/2, vídeo, VHDL.

Freire, T. S. S. "INTERFACEAMENTO DE ENTRADA E SAÍDA EM APLICAÇÕES COM USO DE FPGA", Universidade Federal do Ceará – UFC, 2010, 100p.

This work presents some of the main characteristics of the FPGAs (*Field Programmable Gate Array*), more specifically of the Cyclone II family's development kit from the manufacturer Altera, with the objective of showing the large applicability, diversity and efficiency in using these low power consumption devices. A study about the man-machine interface (MMI) in the FPGA was carried out from the analysis of the protocols and the recognizing methodology of entering data, by keyboard, and also from its external communication, by a VGA (*Video Graphics Array*) output, in the development kit, to a common CRT (*Cathode Ray Tubes*). As experimental results, it was created blocks of programs, in VHDL language, to be recorded in the FPGA to simulate a popular jogo, called TETRIS®, which uses the PS/2 keyboard as the commands input and the screen as output interface, in RGB (*Red-Green-Blue*) colors, to the user. The *software* QUARTUS II® is the developing environment used in the creation of codes, blocks and programming of the logic device studied.

Keywords: Interface, FPGA, PS/2 keyboard, video, VHDL.

SUMÁRIO

Lista de Figuras	ix
Lista de Tabelas	xi
Lista de Abreviaturas.....	xii

INTRODUÇÃO	1
-------------------------	----------

CAPÍTULO 1

OS DISPOSITIVOS LÓGICOS PROGRAMÁVEIS

1.1 Introdução	3
1.2 Histórico e Evolução	3
1.2.1 Arquitetura e Características dos FPGAs	6
1.3 Família Cyclone II da Altera	8
1.4 QUARTUS II® – Ambiente de Desenvolvimento.....	12
1.5 Linguagem de Programação	15
1.5.1 A Linguagem VHDL.....	16
1.6 Conclusões.....	19

CAPÍTULO 2

INTERFACES DE ENTRADA E SAÍDA (PS/2 E VGA)

2.1 Introdução	20
2.2 Interface de Entrada – Teclado.....	20
2.2.1 O <i>Scancode</i> e os Tipos de Conectores Padrão.....	22
2.2.2 A Comunicação do Teclado no Cyclone II	23
2.3 Interface de Saída – Vídeo.....	25
2.3.1 Os Padrões de Vídeo e sua Resolução.....	25
2.3.2 Monitores.....	27
2.3.3 Sistema de Cores RGB	29
2.3.4 A Configuração do modo VGA no Cyclone II.....	31
2.4 Conclusões.....	34

CAPÍTULO 3

BLOCOS DA APLICAÇÃO TETRIS®

3.1 Introdução	36
--------------------------	----

3.2	O Jogo TETRIS®	36
3.3	Bloco de Leitura do Teclado	37
3.4	Bloco de Conversão.....	40
3.5	Blocos Divisores de Freqüência	41
3.6	Bloco Peça	41
3.7	Bloco TETRIS®	42
3.8	Bloco de Sincronismo.....	45
3.9	Bloco Saída de Vídeo	47
3.10	Conclusões.....	48
 CAPÍTULO 4		
RESULTADOS EXPERIMENTAIS		
4.1	Introdução	50
4.2	Montagem e Configuração dos Blocos.....	48
4.3	Testes do Programa e Jogabilidade	52
4.4	Conclusões.....	53
 CONCLUSÃO.....		55
 Referências Bibliográficas		57
 APÊNDICE A – CÓDIGO DO PROGRAMA EM VHDL		59
A.1	VHDL do Bloco Teclado.....	60
A.2	VHDL do Bloco Converter_Código.....	61
A.3	VHDL do Bloco Div_Freq_25	63
A.4	VHDL do Bloco Div_Freq_mov	63
A.5	VHDL do Bloco Sincronismo	64
A.6	VHDL do Bloco Peça	65
A.7	VHDL do Bloco TETRIS®	66
A.8	VHDL do Bloco Video_out.....	76
 ANEXO – TABELA ASCII / SCancode		82

LISTA DE FIGURAS

Figura 1.1 – CIIs disponíveis com destaque à lógica programável	3
Figura 1.2 – Ramificações da lógica programável	4
Figura 1.3 – Esquema de um PLD na alusão à “caixa preta” de Vranesic [13]	5
Figura 1.4 – Matriz de blocos lógicos configuráveis (ou programáveis)	5
Figura 1.5 – Esquemático de blocos internos do FPGA	6
Figura 1.6 – Arquitetura geral de roteamento de um FPGA	7
Figura 1.7 – Placa de desenvolvimento (vista superior)	10
Figura 1.8 – Diagrama de blocos geral do dispositivo	11
Figura 1.9 – Esquemático para a aplicação proposta. Uso da entrada PS/2 (1), <i>display</i> (2) e saída de vídeo (3)	11
Figura 1.10 – Tela inicial do ambiente de desenvolvimento QUARTUS II®	13
Figura 1.11 – Esquemas principais de metodologia para um projeto digital	13
Figura 1.12 – Configuração do dispositivo para início do projeto	14
Figura 1.13 – Processo utilizado na programação de um PLD	15
Figura 1.14 – Etapas da descrição de hardware em um projeto [13]	17
Figura 2.1 – Teclado de 83 teclas (a) e de 93 teclas (b), antecessores do PS/2	21
Figura 2.2 – Disposição das teclas do teclado US internacional (a) e do teclado ABNT brasileiro (b) com o “Ç”	21
Figura 2.3 – Exemplo de um teclado padrão internacional e seu <i>scancode</i> correspondente....	22
Figura 2.4 – Tipos de conectores com destaque para o PS/2	23
Figura 2.5 – Esquemático do circuito do conector PS/2 no FPGA (Cyclone II)	24
Figura 2.6 – Diagrama de tempo da transmissão de um dado	24
Figura 2.7 – Monitor CRT e representação interna do tubo de raios catódicos	28
Figura 2.8 – Vista em corte de um tubo de monitor, com suas placas de deflexão	28
Figura 2.9 – Cubo das oito cores básicas geradas no sistema RGB	29
Figura 2.10 – A tríade de pixels em um monitor colorido (a) e o fluxo de elétrons das cores no tubo (b)	30
Figura 2.11 – Esquemático do circuito VGA no FPGA (Cyclone II)	31
Figura 2.12 – Varredura horizontal e vertical na formação das cores RGB	32
Figura 2.13 – Diagrama de tempo do sincronismo horizontal	33
Figura 2.14 – Diagrama de tempo do sincronismo vertical.....	33
Figura 3.1 – Exemplo de formatação do jogo TETRIS®	36

Figura 3.2 – Diagrama de blocos final do projeto e seus respectivos pinos.....	38
Figura 3.3 – <i>Displays</i> de sete segmentos utilizados na aplicação	40
Figura 3.4 – Representação da área do jogo (15 linhas) e da peça (3 linhas) para o algoritmo de “descida”	43
Figura 3.5 – Formação da área visível com uma menor resolução (64 x 48)	46
Figura 4.1 – Criação de blocos, no software QUARTUS II®	50
Figura 4.2 – Pasta principal do projeto contendo todos os 8 blocos da aplicação.....	51
Figura 4.3 – Imagem do jogo TETRIS® em execução na tela.....	53
Figura 4.4 – Imagem de teste das características do jogo, limpar linha e pontuação.....	54

LISTA DE TABELAS

Tabela 1.1 – Comparativos de chips (CIs) em custo benéfico	5
Tabela 1.2 – Ranking mundial dos maiores fabricantes de PLD/FPGA com suas receitas e vendas no mercado em 2007-2008, com Altera em destaque	9
Tabela 1.3 – Relação dos principais operadores classificados por tipo.....	18
Tabela 2.1 – Pinos de conexão para o circuito PS/2 no FPGA	23
Tabela 2.2 – Resoluções para recepção de alta definição utilizada nas TVs.....	27
Tabela 2.3 – Relação entre as cores disponíveis e a quantidade de bits por pixel na memória de vídeo	30
Tabela 2.4 – Especificações de tempo para sincronismo no Cyclone II	34
Tabela 2.5 – Pinos para configuração de vídeo no FPGA	35
Tabela 3.1 – Algumas operações lógicas básicas entre dois bits.....	44
Tabela 4.1 – Configuração dos pinos para gravação no FPGA	52

LISTA DE ABREVIATURAS

- ABEL – Advanced Boolean Equation Language
ABNT – Associação Brasileira de Normas Técnicas
AHDL – Altera Hardware Description Language
API – Application Programming Interface
ASIC – Application Specific Integrated Circuit
AT – Advanced Technology
ATX – Advanced Technology Extended
CGA – Color Graphics Adapter
CI – Circuito Integrado
CLB – Configuration Logic Block
CMYK – Cyan, Magenta, Yellow, Key
CPLD – Complex Programmable Logic Device
CRT – Cathode Ray Tube
DOS – Disk Operating System
EGA – Enhanced Graphics Adapter
EPROM – Erasable Programmable Read Only Memory
EEPROM – Electrical Erasable Programmable Read Only Memory
FIFO – First-In First-Out
FPGA – Field Programmable Gate Array
HDL – Hardware Descript Language
IEEE – Institute of Electrical and Electronic Engineers
IHM – Interface Homem Máquina
IOB – Input Output Block
JTAG – Joint Test Action Group
LCD – Liquid Crystal Display
LSB – Least Significant Bit
MDA – Monochrome Display Adapter
MSB – Most Significant Bit
PC – Personal Computer
PLB – Programmable Logic Block
PLD – Programmable Logic Device
RAM – Random Access Memory

RGB – Red, Green, Blue

SD – Secure Digital

SDRAM – Syncronous Dinamic Random Access Memory

SRAM – Static Random Access Memory

SVGA – Super Video Graphics Array

USB – Universal Serial Bus

VESA – Video Eletronics Standards Association

VGA – Video Graphics Array

VHDL – VHSIC Hardware Description Language

VHSIC – Very High Speed Integrated Circuit

INTRODUÇÃO

Antigamente, os engenheiros dispunham apenas de circuitos integrados discretos para uso em seus projetos, os circuitos digitais eram, em sua maioria, grandes. Todos os componentes de um projeto estavam limitados às proporções de seu tamanho físico e dos tamanhos da sua linha de produção, dos seus custos finais no orçamento da empresa, da dimensão do seu consumo de energia, do peso, entre outros fatores quantitativos [1]. Conseqüentemente, esses componentes mais antigos foram perdendo espaço e sendo substituídos por outros mais novos e cada vez mais “enxutos”, uma vez que a qualidade final do produto acabava sempre limitada pelo impacto causado por esses fatores de dimensionamento na produção. Dessa forma, com a baixa rotatividade, quebra de produção por parte do fabricante e desinteresse dos importadores por esses componentes mais antigos, os engenheiros se depararam então com várias dificuldades de produção no país, devido, principalmente, à grande escassez desses componentes no mercado e o crescente enfraquecimento de fornecimento dos mesmos.

Com a necessidade do desenvolvimento de CIs (Circuitos Integrados) que permitam uma maior compactação dos circuitos, mas com a possibilidade de uso por todos os tipos de usuários, sejam estes pequenas, médias ou até mesmo grandes empresas, surgiram os dispositivos lógicos programáveis ou PLDs (*Programmable Logic Devices*) que, entre outras características, possuem seus circuitos configuráveis de diferentes modos e em aplicações distintas, dependendo apenas da necessidade de uso desejada pelos seus usuários. Junto a uma portabilidade de uso criada, o custo é barateado, pois torna dispensável a produção de centenas de milhares daqueles componentes, além de uma máscara de desenvolvimento, geralmente caríssima por parte do fabricante, que passa a ser desnecessária [1].

No atual crescimento tecnológico, onde a facilidade de uso (ou acessibilidade) de um determinado equipamento, o seu tamanho físico, ou ainda sua portabilidade e seu custo-benefício final, são critérios sempre impactantes no sucesso frente à grande concorrência, esses dispositivos programáveis são bem vistos no atendimento a muitos desses requisitos. O FPGA é um bom exemplo, pois além de adequar-se aos preços do segmento de mercado, pode ser programado inúmeras vezes pelo usuário e ainda possui muitos recursos, entre estes a interface homem-máquina, que traduz a acessibilidade e facilidade de uso objetivada por todos.

Essa interação homem-máquina pode ser definida como o esforço que o usuário terá para prover uma entrada de dados, ou um comando, e o esforço para interpretar o resultado de sua saída em consequência da entrada dada [2]. O teclado e o monitor são exemplos de dispositivos de entrada e saída, respectivamente, que serão configurados aos protocolos de comunicação existentes no FPGA da família Cyclone II, a partir da criação de um jogo simples, em linguagem de programação VHDL, a ser compilado e aplicado ao dispositivo com o uso do ambiente de desenvolvimento QUARTUS II® da empresa Altera.

A utilização dessas ferramentas de software com o aperfeiçoamento do *hardware* reconfigurável dos FPGAs [3], permite ao usuário uma enorme simplicidade e velocidade no desenvolvimento de seus projetos.

O capítulo 1 apresenta uma fundamentação teórica sobre os dispositivos lógicos programáveis e a placa de desenvolvimento Cyclone II, bem como a linguagem de programação e as outras ferramentas utilizadas para o desenvolvimento desse projeto.

No capítulo 2 é mostrado o protocolo de comunicação dos dispositivos de entrada e saída de dados do trabalho, teclado e monitor de vídeo, respectivamente. São mostrados os tipos de teclado e ainda o sistema de cores de vídeo RGB com sua formação, detalhamento de pixels e sinais de sincronismo.

O capítulo 3 apresenta a metodologia e o detalhamento das etapas da criação do jogo TETRIS® e dos seus blocos de código.

No capítulo 4 são apresentados os testes e resultados experimentais na composição dessa interface.

Por fim, apresenta-se conclusões do presente trabalho, bem como a discussão de alguns problemas e dificuldades encontradas na implementação do projeto.

CAPÍTULO 1

OS DISPOSITIVOS LÓGICOS PROGRAMÁVEIS

1.1 – INTRODUÇÃO

Este capítulo apresenta um breve histórico do crescimento dos dispositivos lógicos até o FPGA. As principais características, aplicações e a arquitetura interna do FPGA no kit de desenvolvimento proposto para a realização do trabalho, bem como a configuração, funcionamento e *layout* das interfaces de entrada e saída da placa Cyclone II [4] também são apresentadas.

Também é mostrado o *software* QUARTUS II®, responsável pela programação e toda a configuração do dispositivo lógico programável, além da linguagem de descrição de *hardware* VHDL que compõe o código fonte do projeto implementado.

1.2 – HISTÓRICO E EVOLUÇÃO

Nas últimas décadas, a maneira como os sistemas digitais são elaborados tem evoluído drasticamente. Na década de 60, eram construídos a partir de componentes discretos, como transistores e resistores, por exemplo. Com o surgimento de circuitos integrados (CIs), foi possível unir vários transistores dentro de uma única pastilha de silício. Atualmente, com a evolução dessa tecnologia, são integrados cerca de 14 milhões de transistores por cm³, com previsão de atingir 100 milhões nos próximos anos [5]. A figura 1.1 apresenta um diagrama com a variedade de CIs, para a implementação de circuitos lógicos, construídos nessas pastilhas de silício.

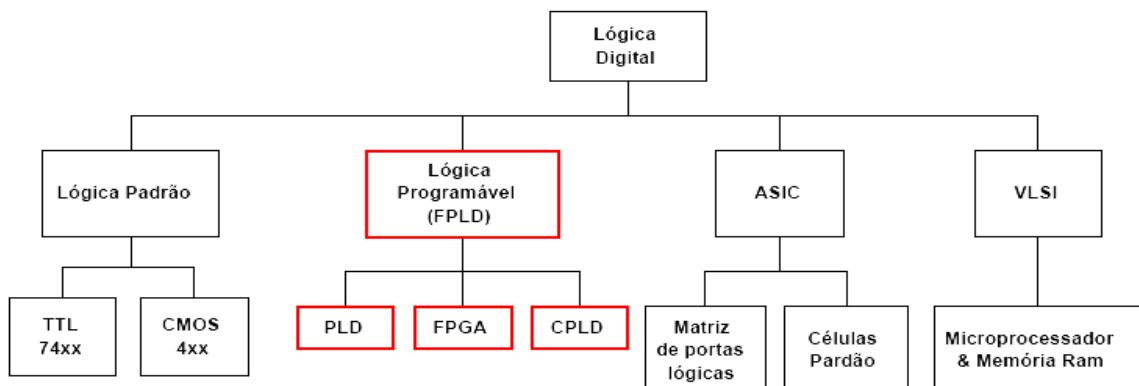


Figura 1.1 – CIs disponíveis com destaque à Lógica Programável.

Para a elaboração de um circuito com lógica definida pelo usuário, eram sempre necessários diferentes tipos de CIs, pois eles realizam operações pré-definidas pelo fabricante. Dessa forma, o tamanho físico do circuito aumentava na área necessária para a conexão, havia perda de desempenho devido ao atraso nas portas de entrada e saída e um aumento no consumo de energia. Após a década de 70 [5] surgiram então, para suprir esses problemas, outros tipos de circuitos integrados, agora com configuração e funcionalidades definidas pelo usuário e não mais pelo fabricante. Entre esses tipos, o ASIC (*Application Specific Integrated Circuit*), PLD (*Programmable Logic Device*), CPLD (*Complex Programmable Logic Device*) e o FPGA (*Field Programmable Gate Array*) se destacavam. As principais ramificações desses dispositivos são apresentadas na figura 1.2.

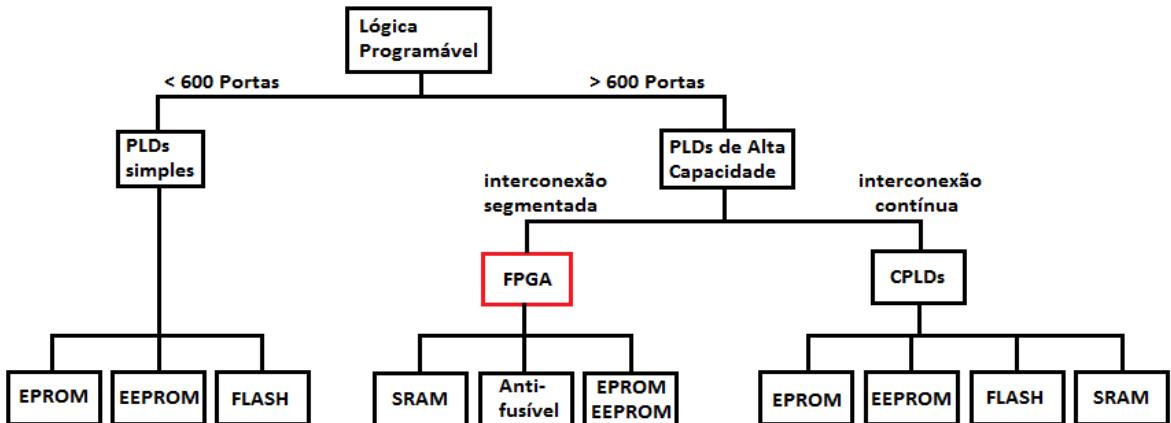


Figura 1.2 – Ramificações da lógica programável.

No ASIC as funções eram definidas pelo usuário, mas implementadas pelo fabricante e de custo elevado, devido à tecnologia especial utilizada no processo de fabricação, na integração de uma máscara única requerida em cada projeto específico [3]. Assim, os ASICs não podiam ser atualizados e tinham uma funcionalidade fixa após serem fabricados. Contudo, os custos eram minimizados com a produção em larga escala e ainda possuíam baixo consumo e alto desempenho.

Já o PLD, o CPLD e o FPGA permitem tanto a implementação quanto a sua funcionalidade definidas pelo usuário, com um desempenho um pouco menor que os ASICs, porém com grande facilidade de desenvolvimento e consequentemente um menor tempo final de projeto [6]. A tabela 1.1 mostra um comparativo de desempenho entre chips, como o ASIC, o FPGA e um microprocessador.

Então a maioria desses dispositivos (exceto aqueles com arquitetura anti-fusível) podem ser reprogramáveis, garantindo-se assim o uso de atualizações disponibilizadas pela empresa

produtora de maneira simples, fácil e rápida, sem a necessidade de um novo desenvolvimento do produto para versões atuais [1].

Tabela 1.1 – Comparativos de chips (CIs) em custo-benefício.

Desempenho	NREs	Custo Unit.	TTM
ASIC	ASIC	FPGA	ASIC
FPGA	FPGA	MICRO	FPGA
MICRO	MICRO	ASIC	MICRO

(*MICRO = Microprocessador)

Na figura 1.3, é esquematizada a arquitetura do PLD visto por Vranesic [5], representado como uma caixa preta contendo blocos e chaves programáveis. Um PLD consiste de um grande arranjo de portas AND e OR, que podem ser programadas para alcançar funções lógicas específicas [7]. É um CI de propósito geral que contém vários elementos lógicos que podem ser configurados de diversas maneiras, para implementar qualquer circuito lógico necessário pelo usuário.

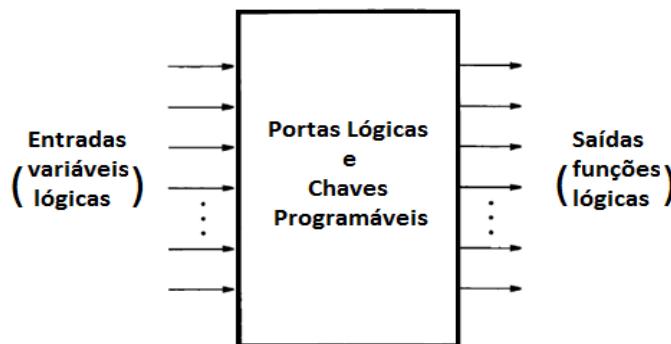


Figura 1.3 – Esquema de um PLD na alusão à “caixa preta” de Vranesic [5].

Nos PLDs mais comuns, cada bloco lógico é normalmente formado por dois níveis de portas AND-OR e portas AND de várias entradas. São blocos ligados eletricamente para que assim, seja possível a formação dos mais diversos circuitos de lógica combinacional e seqüencial [8]. Na figura 1.4 é mostrada uma matriz desses blocos lógicos programáveis.

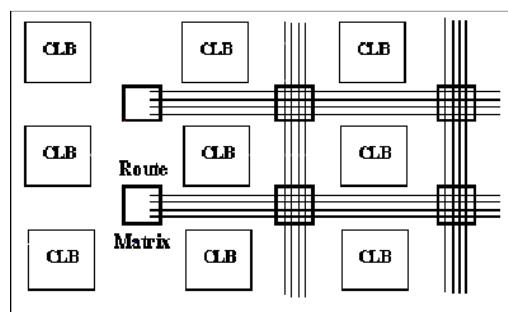


Figura 1.4 – Matriz de Blocos Lógicos Configuráveis (ou Programáveis).

Para a realização deste trabalho, um tipo especial de PLD e um pouco diferenciado, em relação ao nível de estrutura das células lógicas, foi escolhido: o FPGA. Ele surgiu em 1985, com a empresa americana Xilinx Inc. [9], e possui uma grande versatilidade de implementação, pois é feito de múltiplos níveis, com um número maior de portas lógicas [3] e uma maior integração entre os diversos blocos lógicos configuráveis (CLBs), ou PLBs (*Programmable Logic Block*), e facilidade no uso para propósitos diversos.

1.2.1 – ARQUITETURA E CARACTERÍSTICAS DO FPGA

Resumidamente, o FPGA pode ser descrito como um arranjo fortemente condensado de blocos de pequenos circuitos, composto de algumas portas lógicas e flip-flops, com alguns sinais de interface. É um circuito integrado que pode ser configurado por software e que implementa circuitos digitais, como processadores, interfaces, controladores e decodificadores [10]. Possui programação através de protocolo simples e de fácil implementação.

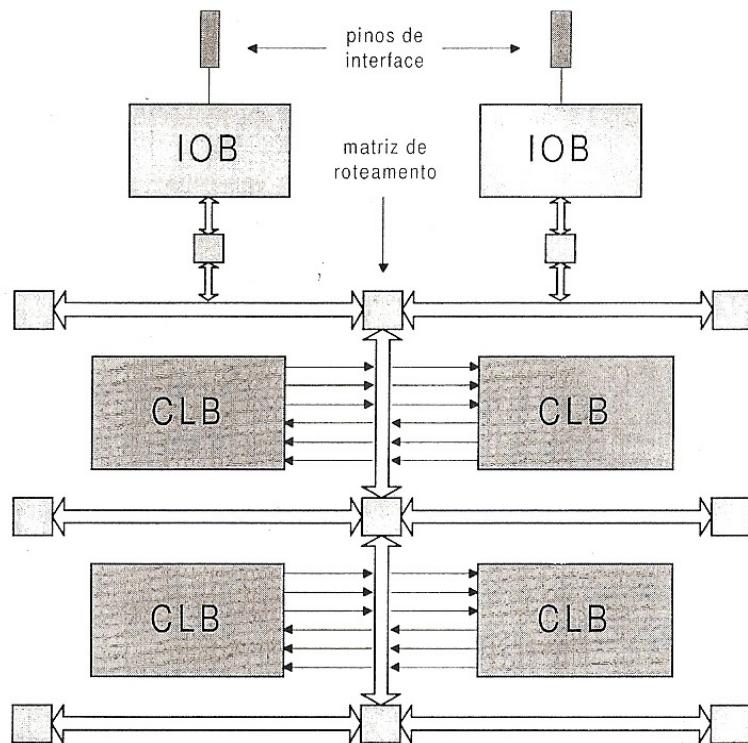


Figura 1.5 – Esquemático de blocos internos do FPGA.

A figura 1.5 mostra uma estrutura conceitual de um FPGA, onde: CLB (*Configuration Logical Blocks*) são os blocos lógicos configuráveis ou programáveis, e IOB (*Input Output Blocks*) são blocos de entrada e saída para o dispositivo.

Uma célula lógica pode ser configurada para desempenhar funções simples, e uma matriz de roteamento (ou chaves programáveis) pode ser personalizada para fornecer interconexões entre as células lógicas. Uma arquitetura personalizada pode ser implementada pela especificação das funções de cada célula lógica e seletivamente configurar a conexão de cada chave programável [11]. Uma vez que o *design* e a síntese estejam completos, pode-se então usar um simples cabo adaptador, no caso do kit Cyclone II utilizado no projeto, um cabo USB (*Universal Serial Bus*), para fazer o download da célula lógica desejada e sua configuração no FPGA, e assim obter o circuito desejado pelo usuário.

A maioria dos dispositivos FPGA incorpora macro-blocos, ou macro-células. Essas, por sua vez, são projetadas e fabricadas ao nível de transistor, e suas funcionalidades complementam as células lógicas gerais [11]. São utilizadas em geral, e nesse trabalho, macro-células que incluem blocos de memória, multiplicadores combinacionais, gestão de circuitos de *clocks*, e circuitos de interfaces de I/O. Os dispositivos FPGA avançados podem até conter um ou mais núcleos de processador pré-fabricados.

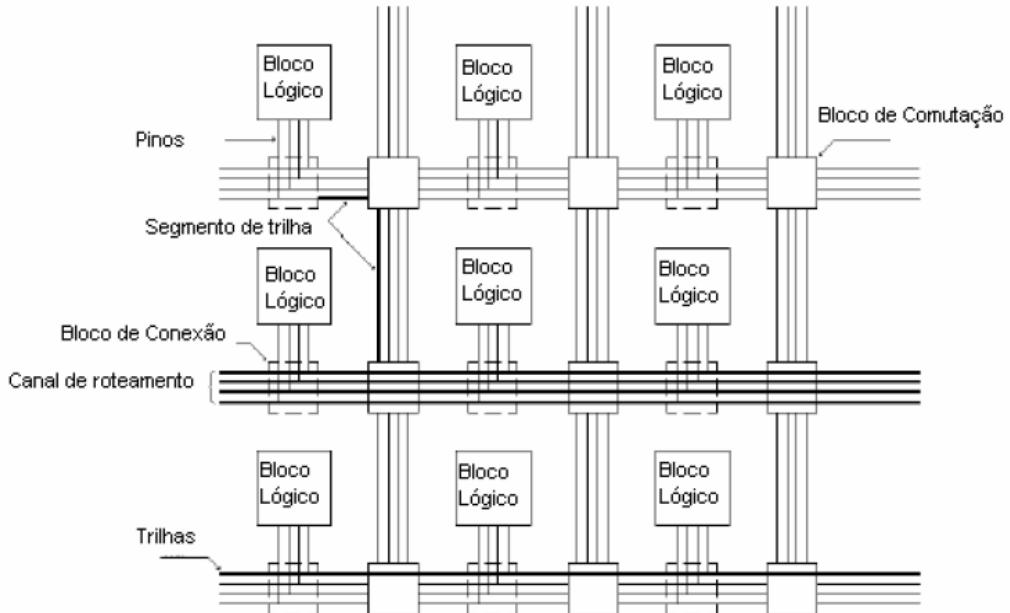


Figura 1.6 – Arquitetura geral de roteamento de um FPGA.

Os blocos de memória embutidos constituem no principal elemento da arquitetura básica, por possuírem os elementos funcionais para a construção de lógicas pelo usuário [9]. Cada um desses blocos pode conter, por exemplo, 2048 bits programáveis que podem ser configurados como RAM FIFO (*First-In First-Out*) e ainda para atuar como um bloco de memória RAM de tamanhos variáveis [3].

Existem três tecnologias principais de FPGA:

- SRAM (*Static Random Access Memory*): Utiliza portas de transmissão (transistor de passagem) ou multiplexador controlado por memória SRAM. Assim, essa tecnologia permite uma rápida reprogramação do circuito, mas com a necessidade de componentes de hardware auxiliar, como uma memória externa tipo FLASH, devido à volatilidade da SRAM [9];
- *Antifuse*: Exige grande concentração de transistores que podem ser configurados em modo corte ou condução (dispositivo de dois terminais). No estado programado (em corte) apresenta alta impedância. Em condução, implementa conexão entre os terminais. É uma tecnologia mais barata que a SRAM;
- *Gate Flutuante*: É semelhante à tecnologia utilizada em memórias EPROM/EEPROM (*Erasable Programmable Read Only Memory / Eletrical EPROM*). Tem a vantagem de permitir a reprogramação e a retenção de dados.

Todas as conexões internas são compostas por segmentos metálicos com pontos de chaveamento programável, para que se implementem as trilhas desejadas. Assim, os FPGAs podem ser utilizados para a implementação de praticamente qualquer *design* de *hardware*. Por conta da maior quantidade de *flip-flops*, sua arquitetura é muito mais flexível que a de um CPLD [9]. As aplicações baseadas registradores e aplicações seqüenciais são melhores e possuem um custo menor se comparado aos CPLDs. Isso torna o FPGA uma excelente opção na construção de grandes projetos lógicos [3].

Entre as diversas aplicações do FPGA, foi criado um programa que simula um jogo muito conhecido, chamado TETRIS®, como um teste para a verificação da interface de interação homem-máquina no kit de desenvolvimento.

1.3 - FAMÍLIA CYCLONE II DA ALTERA

Altera Corporation é uma empresa líder em soluções inovadoras de lógica personalizada. Empresa fabricante, reconhecida mundialmente, de dispositivos lógicos programáveis, hoje conta com aproximadamente 3000 funcionários em 19 países, responsáveis por novas idéias e preocupações com consumo de energia, desempenho e custo final de seus produtos. A tabela 1.2 apresenta um comparativo dos maiores fabricantes de dispositivos lógicos programáveis em 2008, com destaque a Altera que representa 35,5% do

mercado mundial. Junto ao fabricante Xilinx, com 51,2%, são somados quase 90% das receitas totais em todo o mundo.

Desse modo, são criadas soluções para uma grande variedade de indústrias de várias áreas, como automotiva, comunicações, área militar, processamento de vídeo (computacional), etc.

Tabela 1.2 – Ranking mundial dos maiores fabricantes de PLD/FPGA com suas receitas e vendas no mercado em 2007-2008, com Altera em destaque.

Posição em 2007	em 2008	Companhia	Receita (\$M) 2007	Receita (\$M) 2008	mudança de receita 2007 - 2008	Quota de mercado em 2008
1	1	Xilinx	1,809	1.906	5.4%	51.2%
2	2	Altera	1,216	1,323	8.8%	35.5%
3	3	Lattice Semiconductor	229	222	-3.1%	6.0
4	4	Actel	196	218	11.2%	5.9%
6	5	QuickLogic	28	23	-17.9%	0.6%
5	6	Cypress Semiconductor	32	21	-34.4%	0.6%
7	7	Atmel	14	9	-35.7%	0.2%
8	8	Chengdu Sino Microelectronics System	4	3	-25.0%	0.1%
		Outros	0	0	NM	0.0%
		Mercado Total	3,528	3,725	5.6%	100.0%

Fonte: Gartner

Além dos dispositivos programáveis, a Altera trabalha no desenvolvimento de ferramentas de software integrado, processadores embutidos e núcleos otimizados, projetos de referência, e uma grande variedade de kits de desenvolvimento como o Cyclone II, que foi utilizado nesse projeto.

O kit Cyclone II FPGA *Starter Development Board* possui vários periféricos integrados os quais possibilitam o usuário a testar vários tipos de circuitos, desde os mais simples até projetos mais avançados que utilizam multimídia, por exemplo. Todos sem a necessidade de implementações complexas de *hardware* como: interfaces de aplicativos (API's), controle de software ou controladores de memórias [4]. A figura 1.7 ilustra a visão geral da placa dessa placa de desenvolvimento.

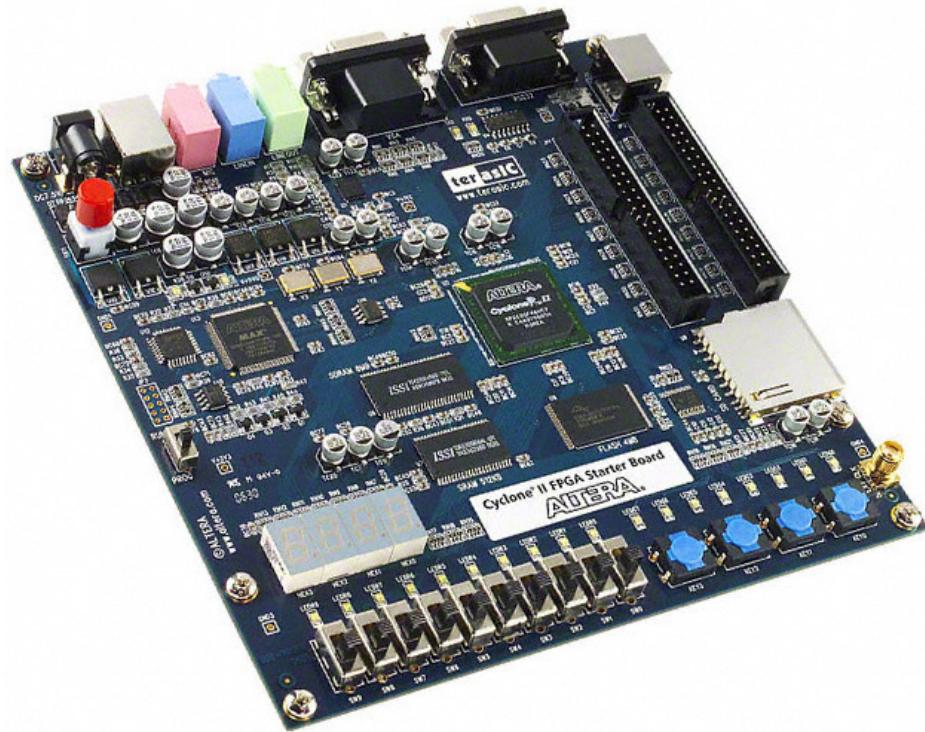


Figura 1.7 – Placa de desenvolvimento (vista superior).

As principais características de *hardware* da placa de desenvolvimento são:

- Possui dispositivo de controle serial Altera EPCS4;
- Controlador USB-Blaster para programação e controle dos APIs, com suporte para modos de programação JTAG e *Active Serial*;
- 512 Kbyte SRAM;
- 8 Mbyte SDRAM;
- 4 Mbyte de memória *flash*;
- Conector para cartão de memória SD;
- 4 botões;
- 10 chaves seletoras;
- 10 leds vermelhos;
- 8 leds verdes;
- Clocks internos de 50 MHz, 27 MHz e 24 MHz;
- CODEC de áudio de 24 bits com qualidade de CD, saída e entrada de linha e entrada de microfone;
- Conector de saída VGA DAC (malha de resistores 4-bit);
- Conector serial RS232 de 9 pinos;
- Conector PS/2 para *mouse/teclado*;

- Dois conectores de expansão de 40 pinos com resistores de proteção;
- Alimentação com adaptador de 7,5 V ou pelo cabo USB;

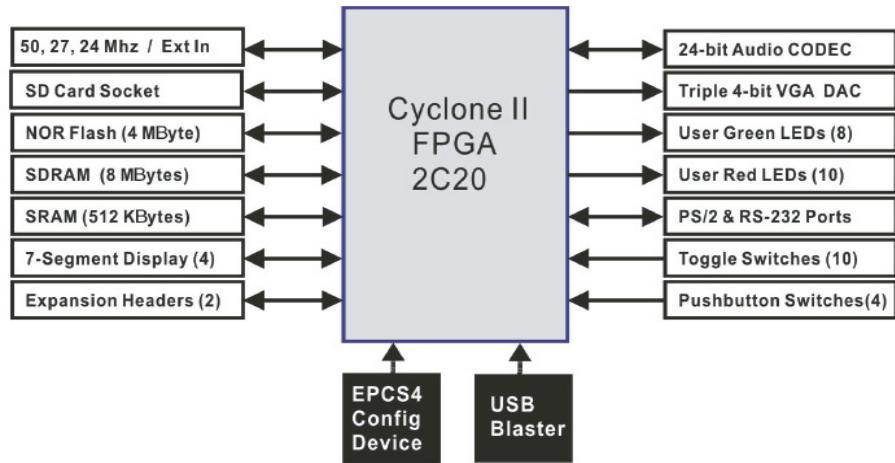


Figura 1.8 – Diagrama de blocos geral do dispositivo.

A figura 1.8 apresenta um diagrama com os principais componentes de entrada e/ou saída presentes nas placas da família 2C20.

O kit proporciona ao usuário uma máxima flexibilidade onde todos os blocos se conectam através do dispositivo FPGA no Cyclone II. Sua arquitetura de duas dimensões de linhas e colunas permitem a implementação de lógicas personalizadas. Coluna e linha propiciam velocidades diferentes para fornecer sinal de interconexões entre blocos de matriz lógica, blocos de memória e multiplicadores embutidos. Assim, o usuário pode executar qualquer projeto de sistema configurando esse dispositivo.

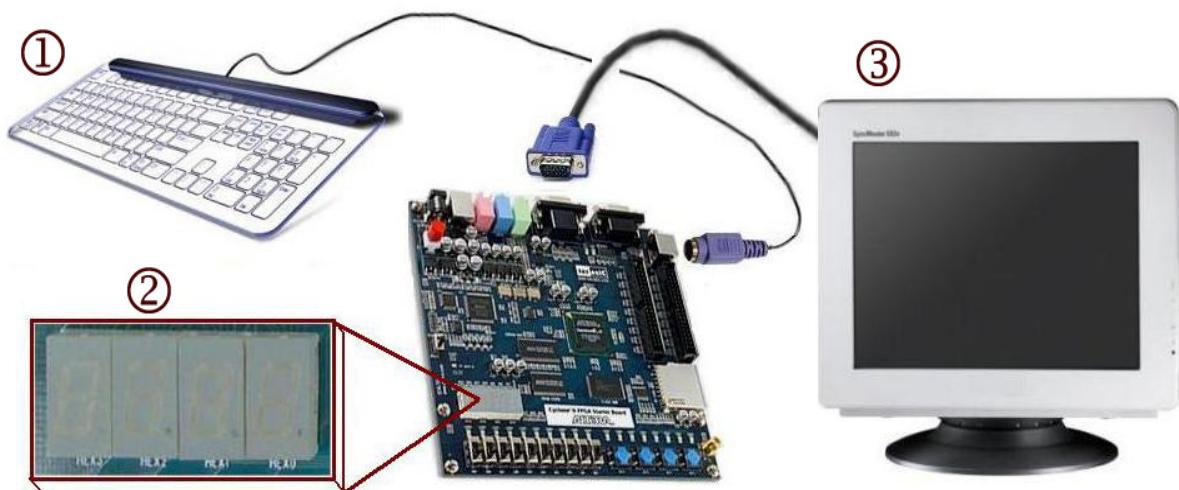


Figura 1.9 – Esquemático para a aplicação proposta. Uso da entrada PS/2 (1), display (2) e saída de vídeo (3).

A figura 1.9 apresenta o esquema de utilização das interfaces de hardware no projeto proposto nesse trabalho. Primeiramente é usada a entrada de dados pelo teclado PS/2, a seguir é mostrado, em hexadecimal, o valor da entrada dada no display de sete segmentos e é comandada a aplicação a ser mostrada no monitor de vídeo VGA. Essas interfaces são tratadas com maiores detalhes nos capítulos posteriores.

1.4 – QUARTUS II® – O AMBIENTE DE DESENVOLVIMENTO

O QUARTUS II® é uma importante ferramenta computacional de análise e síntese de projetos com o objetivo principal de reduzir o tempo de desenvolvimento de circuitos digitais. É um *software* integrado, fornecido gratuitamente pela Altera, que é usado na construção, configuração e implementação geral do projeto [3].

Algumas características das especificações de entrada no *software*:

- Possui um editor gráfico, com elementos primitivos, portas lógicas básicas e outros elementos guardados (macrofunções) em uma biblioteca, para uso na área do projeto interligando suas portas;
- Permite descrição do circuito lógico em códigos escritos, a partir de um editor de texto apto a receber linguagem de programação como AHDL, VHDL e Verilog;
- É dotado ainda de um editor de formas-de-onda (*waveforms*), onde são definidas variadas formas de onda a partir de níveis lógicos definidos pelo usuário. São usadas geralmente para simulações finais das aplicações criadas.

A partir de um código escrito no editor de texto do QUARTUS II®, em linguagem de programação VHDL, são gerados os blocos da aplicação desejada, são realizados testes gerais (etapa de compilação), é criado um arquivo com extensão *.sof*, ou *.pof*, contendo a configuração final a ser programada no FPGA, e ainda é feita essa gravação física (programação) no dispositivo lógico desejado. A figura 1.10 mostra a tela inicial desse *software* fornecido pela Altera.

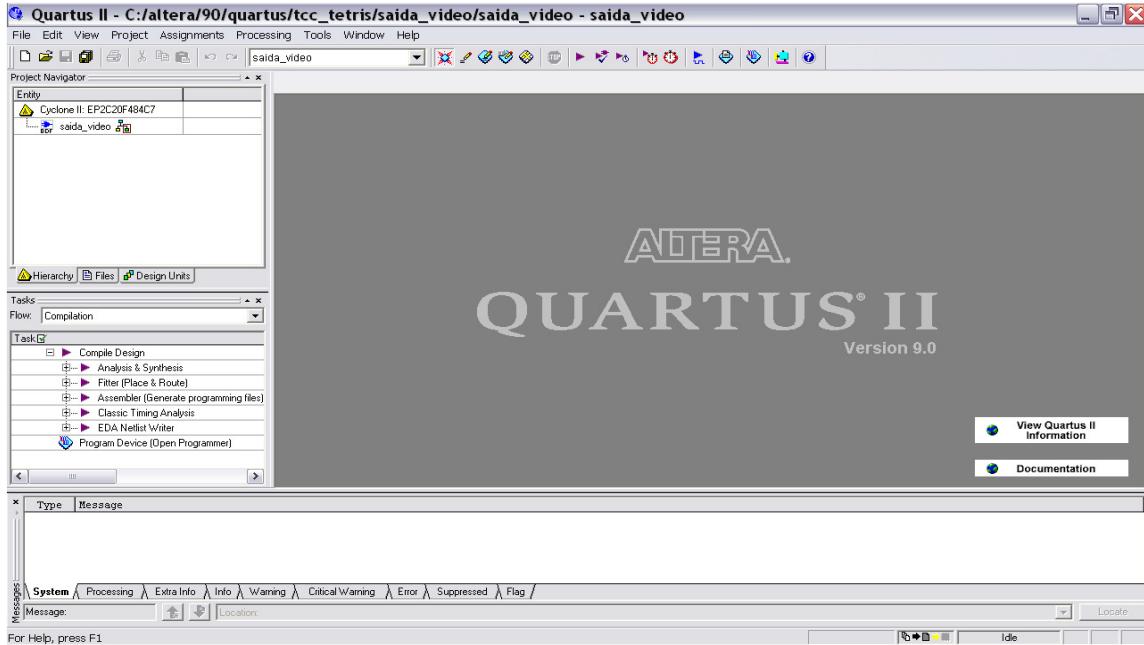


Figura 1.10 – Tela inicial do ambiente de desenvolvimento QUARTUS II®.

Essa ferramenta apresenta muitos recursos, por meio de suas bibliotecas internas, onde estão disponíveis diversos componentes lógicos, que podem auxiliar muito o projetista, principalmente no modo editor gráfico.

Tanto no QUARTUS II® como em diversos aplicativos de desenvolvimento de circuitos digitais configuráveis, todos os dados referentes ao circuito devem estar contidos em um projeto. Um projeto é um ambiente em que é possível desenhar um circuito lógico digital, executar compilações, simulações, analisar sinais em tempo de execução, etc. Um projeto inclui um nome, uma pasta e o dispositivo onde o projeto será carregado [12]. Pode ser esquematizado como dois níveis principais, como mostrado na figura 1.11.

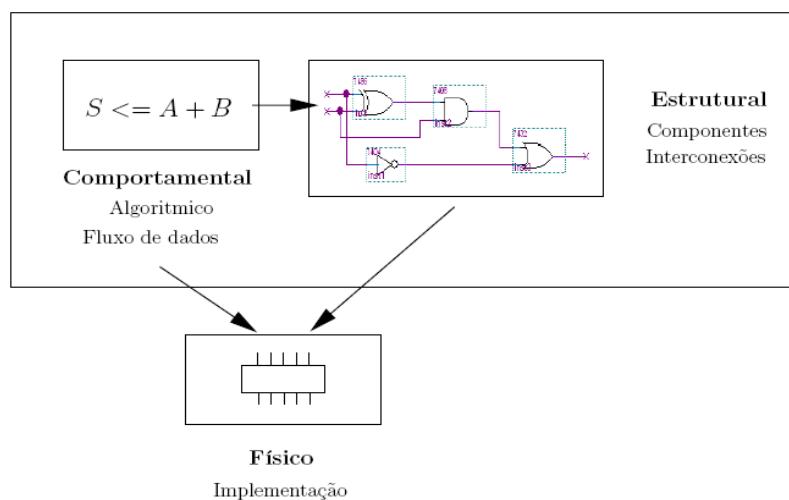


Figura 1.11 – Esquemas principais de metodologia para um projeto digital.

Assim, independente da complexidade do projeto, a descrição e desenvolvimento são facilitados. Ao nível comportamental, é feita uma descrição detalhada do circuito mostrando a relação entre os sinais de entrada e saída, como diagrama de estados, por exemplo. A nível estrutural é realizada uma descrição em termos das conexões de portas lógicas (ou estruturas lógicas do circuito proposto). Por fim, o nível físico comporta a programação (implementação) propriamente dita do dispositivo utilizado (FPGA).

O ambiente de desenvolvimento QUARTUS II® é dotado de ambos os níveis, além de outras características, como a descrição comportamental do projeto a partir de linguagem de programação simplificada e capacidade de simulações funcionais. Ao iniciar um novo projeto no software, após definidos o local (pasta) e nome do mesmo, é feita a escolha da família da placa de desenvolvimento que será usada, também da empresa Altera (nesse caso o Cyclone II da família EP2C20F484C7), como visto na figura 1.12.

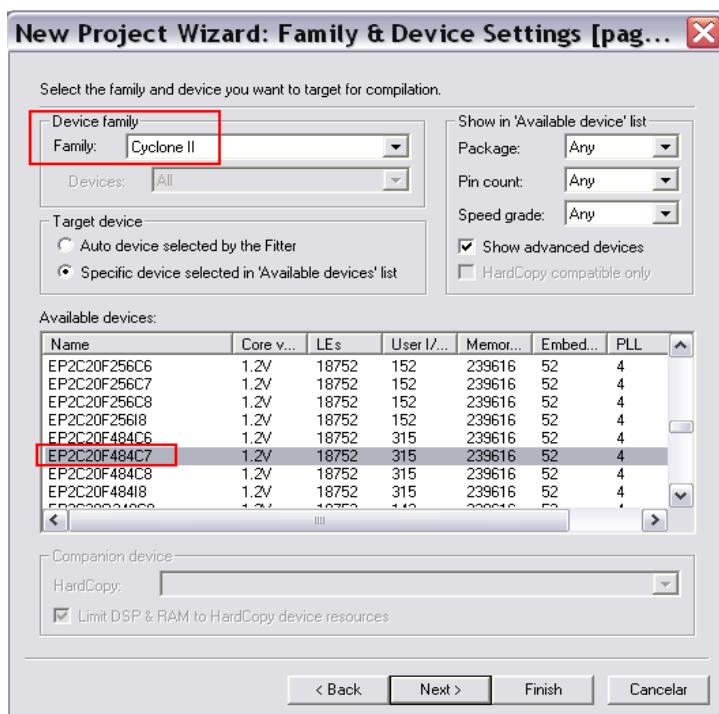


Figura 1.12 – Configuração do dispositivo para inicio do projeto.

Com o novo projeto criado, é escolhido um novo arquivo para edição de acordo com o tipo de linguagem desejada, ou ainda um novo diagrama de blocos para desenho do circuito lógico proposto pelo usuário.

Por fim, terminada a edição e compilação do código completo (ou montagem do diagrama esquemático do circuito), é feita, através da opção *Assignment Editor*, a

configuração dos pinos referentes às portas reais de entrada e saída do Cyclone II, e o chip do FPGA é fisicamente programado (gravado) a partir da opção *Programmer* do software.

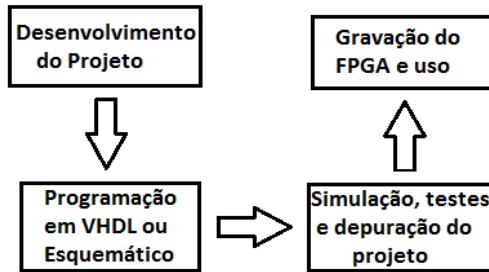


Figura 1.13 – Processo utilizado na programação de um PLD.

A figura 1.13 sintetiza as etapas necessárias para o desenvolvimento de uma aplicação em um dispositivo lógico programável, as quais podem ser implementadas de uma forma razoavelmente simples e rápida com a utilização da ferramenta de software da Altera.

1.5 – LINGUAGEM DE PROGRAMAÇÃO

No meio industrial e acadêmico estão disponíveis diversas linguagens de descrição de hardware. São linguagens que descrevem o comportamento de um controlador digital de diversas formas, como equações lógicas, funcional, temporal, diagrama de formas de onda, tabelas verdades, etc. Entre as mais comuns estão: ABEL, AHDL, VHDL e Verilog.

Resumidamente, são descritas como sendo:

- **ABEL (Advanced Boolean Equation Language):** É uma linguagem padrão da indústria usada para programar CPLDs. É uma linguagem de descrição de hardware para componentes não muito complexos.
- **AHDL (Altera Hardware Description Language):** É disponibilizada no editor de texto dos softwares MAX+PLUS II® e QUARTUS II®, ambos desenvolvidos pela Altera para auxiliar os projetistas de circuitos eletrônicos a documentar e simular projetos, principalmente se baseados em CPLDs e FPGAs;
- **VHDL (VHSIC Hardware Description Language):** VHSIC (*Very High Speed Integrated Circuit*) é um programa organizado pelos EUA, no inicio dos anos 80, que necessitava de uma linguagem normalizada para descrever a estrutura e funcionalidade de circuitos lógicos. Essa linguagem (VHDL) foi padronizada

pelo IEEE em 1987, e permite simular, projetar e implementar funções lógicas em dispositivos PLDs;

- Verilog: É uma linguagem de descrição de hardware desenvolvida em 1983, pela *Gateway Design Automation*, adquirida em 1989 pela empresa Cadence. Foi padronizada pelo IEEE em 1995, e com características similares à linguagem VHDL, também permite simular, projetar e implementar funções lógicas em dispositivos lógicos programáveis, como CPLDs e FPGAs.

1.5.1 – A LINGUAGEM VHDL

A utilização de linguagens gráficas como esquemático, por exemplo, para descrever circuitos lógicos, vem tornando-se cada vez menos popular se comparado a linguagens textuais, sem muita complexidade e menos trabalhosas para a construção de uma aplicação pelo usuário.

Os projetos desenvolvidos por esquemático permitem o uso de portas lógicas, flip-flops e sub-circuitos que, por sua vez, podem ser compostos de mais portas lógicas e outros sub-circuitos, formando vários níveis hierárquicos. Muitos projetistas preferem essa técnica, pois ela é visualmente mais atrativa e se tem uma visão mais clara do relacionamento entre os vários blocos.

No esquemático o sistema é normalmente especificado como uma interconexão de elementos, e não na forma de comportamento do sistema. Contudo, geralmente o projetista recebe as especificações de sua proposta de trabalho na forma do comportamento desejado, ou seja, o que o sistema deve fazer sob certas condições. Além disso, mesmo usando-se esquemáticos, o projeto começa a ficar inacessível, ou ainda menos atraente, para circuitos com muitas portas lógicas.

Com as linguagens HDLs (*Hardware Description Languages*), foi permitida, aos desenvolvedores, uma modelagem de processos concorrentes que são comumente encontrados em elementos de hardware. Com isso essas linguagens tornaram-se cada vez mais populares e rapidamente aceitas [13]. Elas eliminam a dificuldade de conversão manual da descrição do projeto em um conjunto de equações booleanas. A figura 1.14 representa um diagrama de blocos básico para a criação de um projeto, a partir de um algoritmo desejado.

Para a descrição dos circuitos lógicos propostos nesse projeto, foi utilizada a linguagem VHDL, presente no compilador do *software* da Altera. Essa linguagem foi originalmente

destinada a dois propósitos principais [5]. Primeiro, ela foi usada como documentação para descrever a estrutura dos circuitos digitais complexos. Com padrão oficial IEEE, a VHDL tornou-se uma maneira comum de documentar circuitos lógicos implementados por vários projetistas. Em segundo lugar, ela fornece recursos para modelar o comportamento de um circuito lógico, o que permite sua utilização como entrada para os programas que são usados para simular a operação do circuito, como é o caso do QUARTUS II®, por exemplo.

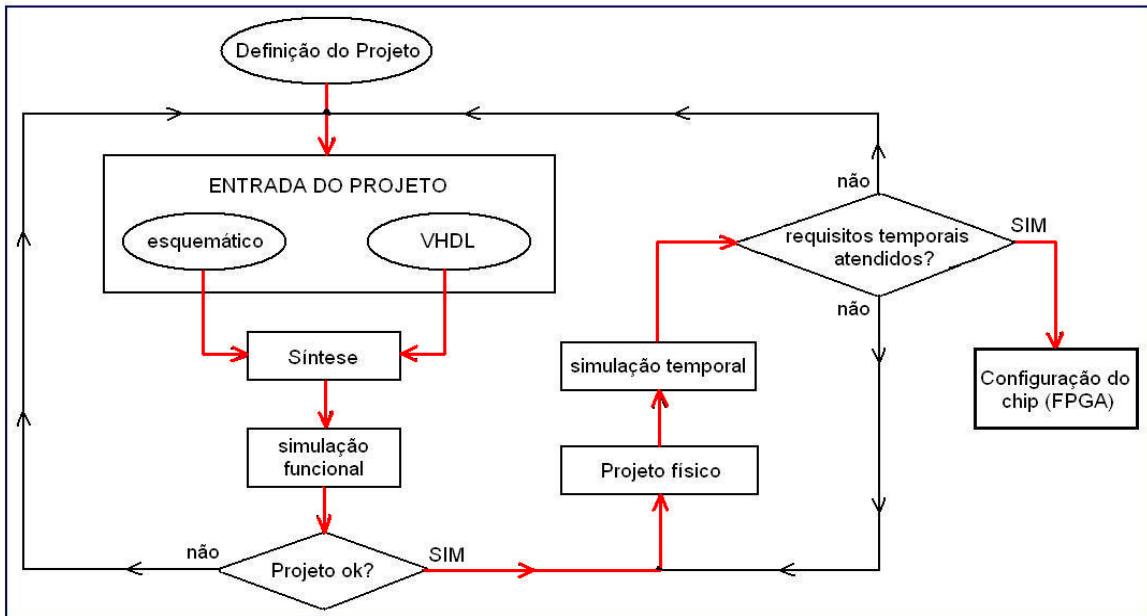


Figura 1.14 – Etapas da descrição de hardware em um projeto [13].

Um sistema digital é geralmente descrito como uma coleção hierárquica de módulos. Cada módulo tem um conjunto de portas que constituem a sua interface com o mundo exterior [14]. Em VHDL, uma entidade (*Entity*) é como um módulo que pode ser usado como um componente em um projeto, ou que pode ser o módulo de mais alto nível do projeto. As portas lógicas de entrada e saída do sistema são declaradas na entidade. Além disso, características opcionais na declaração da entidade podem ser usadas para definir um comportamento especial e monitorar o seu funcionamento.

Uma vez que uma entidade foi especificada, uma ou mais implementações dessa entidade serão descritos em um bloco de arquitetura (*Architecture*). Cada um desses pode descrever uma visão diferente da entidade. As declarações no corpo da arquitetura definem os itens que serão utilizados para construir a descrição do projeto [14]. Em particular, os sinais, as constantes e os componentes podem ser declarados neste trecho e utilizados para criar uma descrição estrutural em termos de instâncias dos componentes.

Podem existir ainda, na estrutura da linguagem, os pacotes (*Packages*). Eles provêm um local para o armazenamento de constantes, subprogramas, bem como declarações de tipos, sinais e componentes [15]. Em outras palavras, o objetivo principal de um *Package* é encapsular os elementos que podem ser compartilhados (globalmente), entre duas ou mais unidades do projeto. É uma área de armazenamento comum usada para armazenar dados que sejam compartilhados entre várias entidades [16]. Declarar dados dentro de um pacote permite que os dados sejam referenciados por outras entidades, ou seja, o dado é compartilhado.

A tabela 1.3 apresenta alguns dos principais operadores utilizados na linguagem. As operações, os tipos de variáveis, constantes e sinais, bem como o uso dos comandos seqüenciais básicos são mostrados ao longo do trabalho, em comentários e na descrição dos códigos implementados para o projeto.

Tabela 1.3 – Relação dos principais operadores classificados por tipo [16].

classe	operadores
lógicos	and or nand nor xor xnor
relacionais	= /= < <= > >=
deslocamento	sll srl sla sra rol ror
adição	+ - &
sinal	+ -
multiplicação	* / mod rem
diversos	** abs not

Em resumo, a linguagem VHDL permanece até hoje com algumas características principais [6], como:

- Os projetos podem ser decompostos hierarquicamente.
- Cada elemento do projeto tem uma interface bem definida (para interconexões com outros elementos) e uma especificação precisa do comportamento.
- As especificações do comportamento podem usar um algoritmo ou uma estrutura de hardware real para definir a operação de um elemento. Por exemplo, um elemento pode ser definido inicialmente por um algoritmo, que permita a verificação do projeto para uso de elementos de alto nível; e depois, o algoritmo pode ser substituído por uma estrutura de hardware.
- Tudo pode ser modelado. A linguagem trata da estrutura de circuitos seqüenciais síncronos, bem como assíncronos.

- A linguagem é concorrente, ou seja, a ordem dos comandos não importa. Comandos seqüenciais são possíveis apenas em rotinas e tratamentos específicos.
- A operação lógica e o comportamento no tempo de um projeto em VHDL podem ser simulados.

1.6 – CONCLUSÕES

Os dispositivos lógicos programáveis têm se tornado escolhas essenciais para projetos onde o tempo e o custo final são fatores críticos. Dentre esses dispositivos o FPGA foi escolhido como destaque devido ao seu baixo consumo de energia e sua facilidade de uso em implementações variadas. O kit Cyclone II possibilita ao projetista a criação de sistemas simples ou mais complexos, isso tudo com a vantagem de ter em mãos uma grande quantidade de periféricos para uso.

O ambiente de desenvolvimento oferecido pela Altera, é uma importante ferramenta na manipulação desses dispositivos permitindo, através de uma linguagem muito intuitiva e de fácil compreensão (VHDL), a configuração de diversas interfaces de hardware, com destaque especial para a saída de vídeo e entrada de dados pelo teclado, que serão tratadas ao longo desse trabalho.

CAPÍTULO 2

INTERFACES DE ENTRADA E SAÍDA (PS/2 E VGA)

2.1 - INTRODUÇÃO

Este capítulo tem a finalidade de mostrar como é o protocolo de comunicação na entrada de dados a partir de um teclado, bem como na saída de vídeo em um monitor CRT, objetivando a estruturação dos blocos principais da aplicação e a configuração dos sinais de sincronismo necessários à correta comunicação com o FPGA.

São apresentados os tipos de teclados, seus pinos de configuração, o código e os tipos de caracteres gerados ao pressionarmos uma tecla e como ocorre a transferência completa de uma palavra de dados. Também são mostradas as principais características do vídeo, a resolução, o mapeamento de *pixels*, a freqüência de transmissão e o sincronismo para todas as linhas horizontais e verticais das imagens geradas no sistema de cores RGB.

2.2 – INTERFACE DE ENTRADA – TECLADO

Antigamente, quando os primeiros computadores pessoais foram lançados, só existia um tipo de teclado, com 83 teclas. Logo depois veio o teclado com 84 teclas, apenas com a inclusão da tecla *System Request* (Sys Rq) e um aumento de tamanho da tecla espaço [17]. Mas os usuários eram sempre tendenciosos a novos modelos de teclado, com teclas mais personalizadas (mais firmes, mais macias, etc.) e com caracteres estrangeiros, padronizados especialmente para cada região do planeta.

Com o lançamento do AT (*Advanced Technogy*), padrão de melhorias gerais de *hardware* criado pela Intel, foi lançado um novo teclado, de 93 teclas [18]. E nos anos 80, com o surgimento do conector PS/2, foram lançados os teclados de 101 teclas ou *Enhanced Keyboard* (teclado aprimorado) com 12 teclas de função na fileira superior (ao contrário das 10 do lado esquerdo no teclado de 84 teclas), área separada para as teclas de setas, etc. [17]. Na figura 2.1 são mostrados teclados de 84 e 93 teclas.

Os teclados mais novos, utilizados nos PCs até os dias atuais, seguem o padrão aprimorado e são compostos por 102 teclas, podendo variar até 105, com teclas produzidas especialmente para o windows (atalhos), como o menu iniciar, por exemplo [18]. Existem

ainda outras variações, podendo alcançar um número de até 130 teclas, com conjunto de comandos próprios, teclas de funções alinhadas, teclas de multimídia, de controle do navegador da internet e muitas teclas programáveis [19].

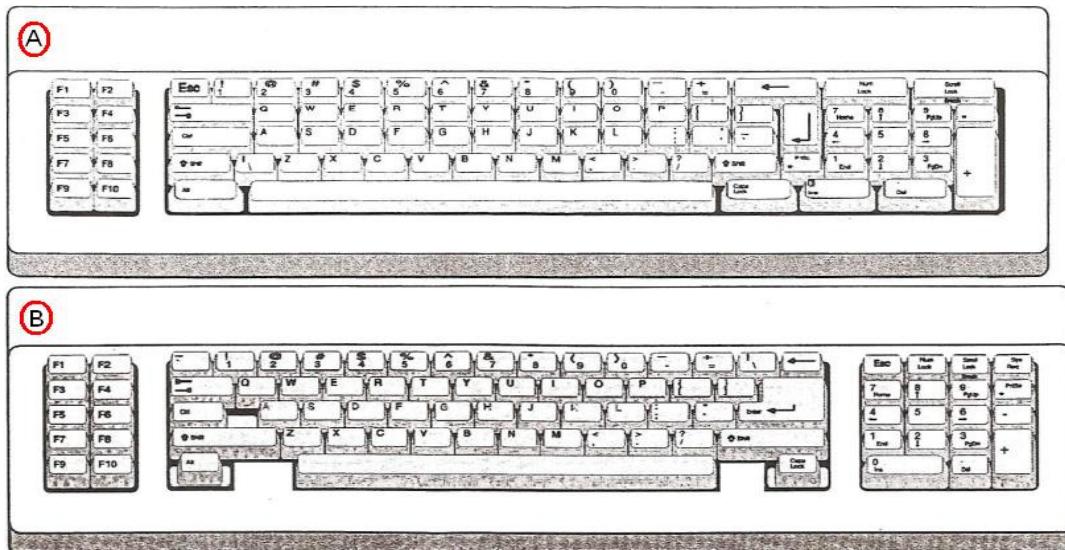


Figura 2.1 – Teclado de 84 teclas (A) e de 93 teclas (B), antecessores do PS/2.

Quanto ao tipo, a maioria dos computadores utiliza teclados do tipo US Internacional. Entretanto, em diversos outros países do mundo, utilizam-se outros padrões, como o teclado brasileiro ABNT (Associação Brasileira de Normas Técnicas) e ABNT2, por exemplo [18]. Entre suas características eles têm algumas teclas em posições diferentes e o “Ç” incluso, que não é encontrado no teclado internacional. Essas características são apresentadas na figura 2.2.

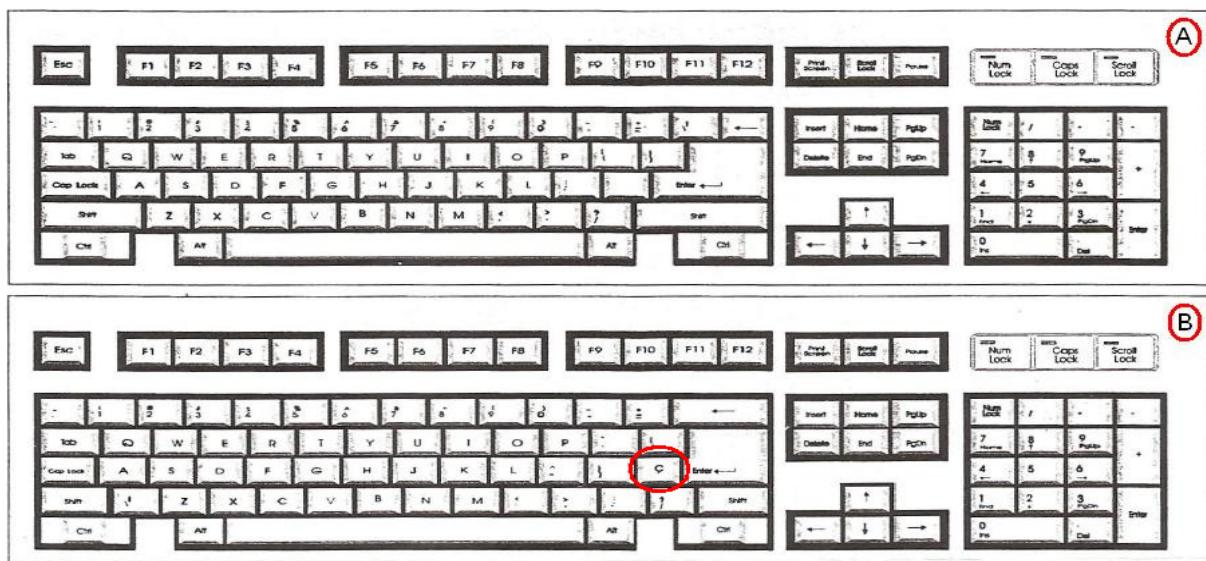


Figura 2.2 – Disposição das teclas do teclado US Internacional (A) e do teclado ABNT brasileiro (B) com o “Ç”.

2.2.1 – O SCancode E OS TIPOS DE CONECTORES PADRÃO

O teclado IBM, padrão mais utilizado atualmente, gera um código especial chamado *scancode* (diferente do código ASCII), formado por um byte representado geralmente em hexadecimal, ao hospedeiro a cada vez que uma tecla é pressionada. Para se detectar quando mais de uma tecla for pressionada simultaneamente, ao soltar uma tecla um novo dado é transmitido e é gerado um *break-code*, geralmente diferente do *scancode* inicial da ação de pressionar (*make*) no bit mais significativo [10]. Na figura 2.3 é apresentada uma relação das principais teclas com seus respectivos códigos. Em anexo, ao final deste trabalho, é apresentada uma tabela detalhada de referencia dos scancodes e ASCII.

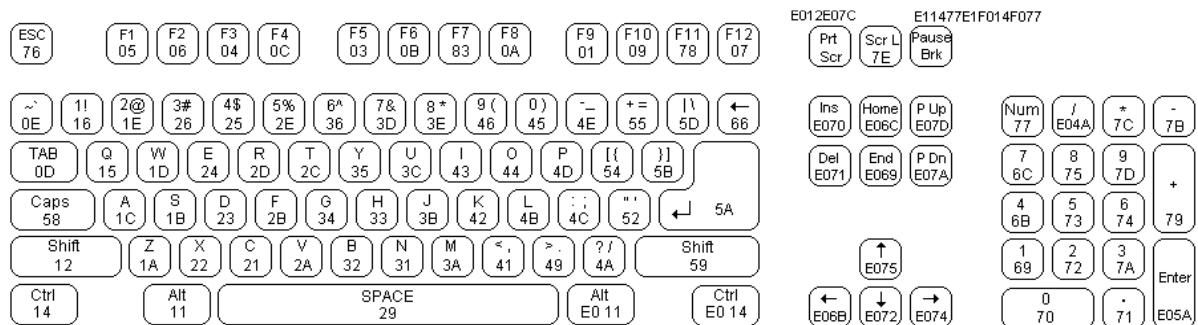


Figura 2.3 – Exemplo de um teclado padrão internacional e seu *scancode* correspondente.

Nos anos 80, e até aproximadamente 1998, praticamente todos os computadores pessoais usavam em seus teclados, um conector DIN de 5 pinos. Esses conectores eram utilizados em aparelhos de som e, por serem muito baratos e comuns, foram aproveitados para a conexão dos teclados nos PCs [19]. Com o passar do tempo e o lançamento das placas com padrão ATX (*Advanced Technology Extended*) ao longo dos anos 90, surgiram os conectores PS/2, padrão utilizado na construção desse projeto.

Então, o teclado comunica-se através desse conector DIN ou através de um conector do tipo PS/2 (mini-DIN de 6 pinos), que é o padrão de interface usado para o teclado no kit de desenvolvimento Cyclone II. Em ambos os casos, os teclados são compatíveis entre si com uma linha que transmite o sinal de clock, outra dos dados e as demais a alimentação (+5V) e o terra (GND), diferenciando-se somente pelo tipo de conector apresentado [18]. A figura 2.4 mostra esses conectores com seus pinos referenciados.

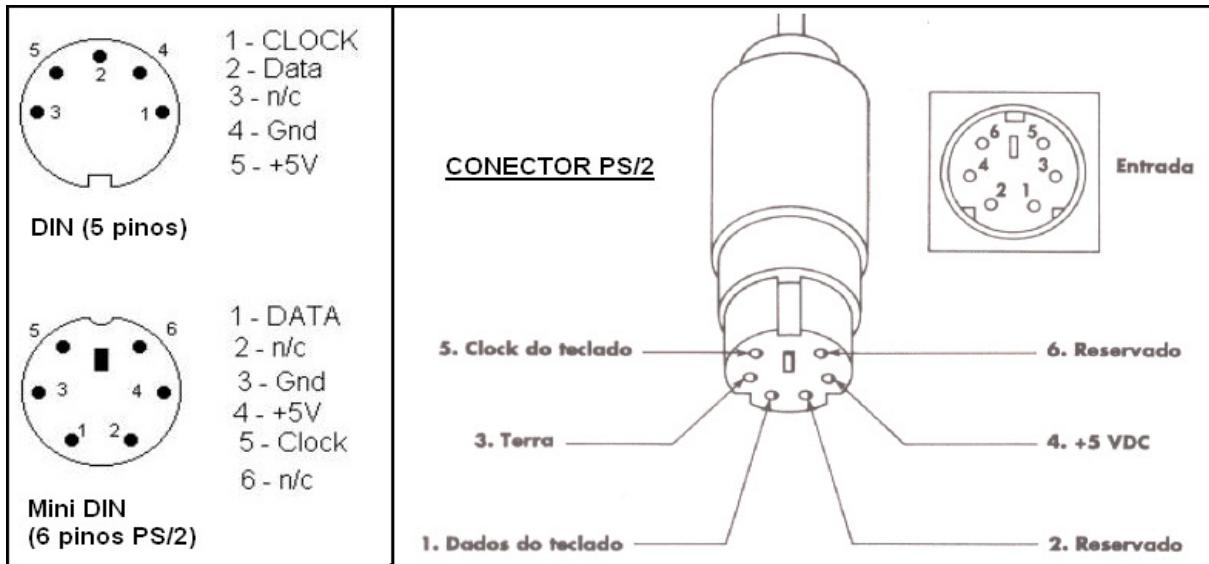


Figura 2.4 – Tipos de conectores com destaque para o PS/2.

Existe também o padrão USB (*Universal Serial Bus*), muito utilizado atualmente para a comunicação de teclados, mouses e outros periféricos a vários sistemas. E ainda é possível a comunicação do teclado por dispositivos sem fio como Bluetooth, por exemplo, mas estes não serão abordados no trabalho, por não fazerem parte da interface presente no kit de desenvolvimento Cyclone II da Altera.

2.2.2 – A COMUNICAÇÃO DO TECLADO NO CYCLONE II

A porta PS/2 é uma interface bastante utilizada por teclado e mouse para se comunicar através da entrada de dados. O pino de dados faz uma transmissão em fluxo serial e o pino de clock especifica quando um dado é valido e pode ser recuperado [11]. No FPGA configuramos esses sinais do conector (PS2_DAT e PS2_CLK) nos pinos PIN_J14 e PIN_H15, respectivamente, conforme a tabela 2.1 [4]. O circuito do conector PS/2 no kit de desenvolvimento é mostrado na figura 2.5.

Tabela 2.1 – Pinos de conexão para o circuito PS/2 no FPGA.

Signal Name	FPGA Pin	Description
PS2_CLK	PIN_H15	PS/2 Clock
PS2_DAT	PIN_J14	PS/2 Data

A comunicação no conector PS/2 é síncrona, funciona de 10 kHz a 16 kHz no nível TTL (*Transistor–Transistor Logic*) e ocorre sempre iniciando do bit menos significativo

(LSB) do dado a ser transmitido [20]. O elemento que gera o sincronismo na rede é sempre o teclado, ou seja, ele tem seu próprio clock interno.

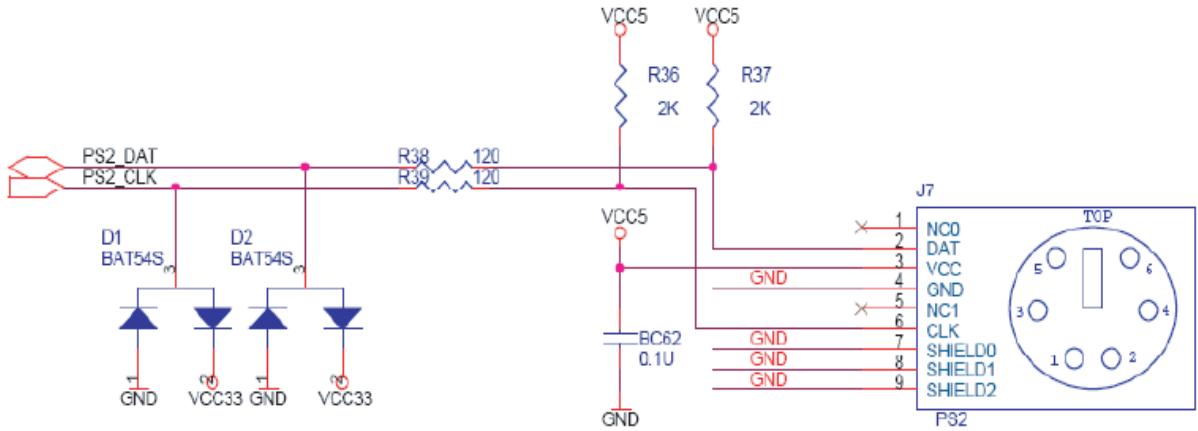


Figura 2.5 – Esquemático do circuito do conector PS/2 no FPGA (Cyclone II).

A transmissão completa de um dado (byte) é feita em um pacote de 11 bits, sendo estes passados bit a bit (serialmente) e representados por um *start bit*, oito bits de dados, um bit de paridade (ímpar) e um *stop bit*, respectivamente, conforme indicado na figura 2.6. Enquanto não há atividade na comunicação serial do teclado, o sinal de clock (PS/2_CLK) e de dados (PS/2_DAT) permanecem em nível alto. Os dados, por sua vez, são lidos na borda de descida do *clock*, a partir do *Start Bit* que deve ser obrigatoriamente zero (nível lógico baixo).

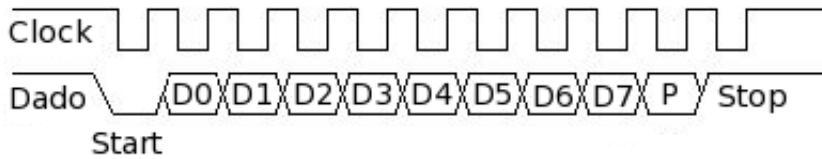


Figura 2.6 – Diagrama de tempo da transmissão de um dado.

Após a contagem dos 11 bits iniciais é necessário um intervalo de mais 11 bits em nível lógico zero, para identificar o início de um novo código que será gerado (dessa vez o break-code) na ação de soltar aquela mesma tecla. Por fim, são contabilizados mais esses 11 bits do break-code, totalizando 33 bits para completar a varredura final da ação de pressionar e soltar uma tecla. O tratamento dessa lógica de transmissão do teclado para a proposta de projeto no Cyclone II é mostrado no capítulo 3 desse trabalho (seções 3.3 e 3.4).

2.3 – INTERFACE DE SAÍDA – VÍDEO

A impressão visual causada por um programa sempre é importante, e expressões como interface amigável e facilidade de uso, costumam ser enfatizadas na utilização de um *software*. A qualidade das imagens que se vê na tela pode influenciar totalmente na avaliação sobre essas facilidades e praticidades oferecidas por um programa.

Até o início dos anos 90, o uso dos computadores era baseado em caracteres. O sistema operacional MS-DOS não apresentava gráficos e suas telas de comando eram baseadas em texto [19]. Assim, há apenas alguns anos atrás, era comum os computadores serem equipados com placas de vídeo e monitores CGA (*Color Graphics Adapter*), que além de gerarem uma imagem de baixíssima qualidade, mal permitia trabalhos com uma interface gráfica.

A popularidade do Windows fez com que a demanda por velocidade gráfica, antes restrita a aplicações específicas, fosse incorporada por uma faixa mais ampla de usuários. Embora as resoluções mais altas ainda não sejam essenciais para os usuários comerciais mais comuns, quase todos os aplicativos produzidos atualmente funcionam em modo gráfico e não em modo de texto. Em particular, os aplicativos para Windows, que funcionam em modo gráfico, parecem lentos quando comparados aos aplicativos para DOS, por exemplo, que operam em modo de texto. Isso acontece, em parte, porque o microprocessador consome muito tempo calculando e exibindo as imagens da tela.

2.3.1 – OS PADRÕES DE VÍDEO E SUA RESOLUÇÃO

Como acontece com a maioria dos componentes e periféricos de computador, cada novo padrão de vídeo para aplicativos comerciais comuns é sempre uma evolução do padrão anterior, que melhoravam cada vez mais a qualidade das imagens exibidas no monitor pelo aumento da resolução e/ou do número de cores [17].

A resolução corresponde ao número de pontos, conhecidos como pixels, que podem ser exibidos no monitor, por exemplo, o modo VGA (*Video Graphics Array*) com 640 pontos horizontais x 480 verticais. Em resumo, refere-se ao número de pixels horizontais e verticais que podem ser exibidos na tela.

Os padrões de vídeo evoluíram muito desde que os primeiros PCs foram lançados no mercado, principalmente com o surgimento do Windows, que teve como grande sucesso,

entre outras coisas, o uso de telas totalmente gráficas, com ícones, figuras e interface visual mais atraente para o usuário [19].

Entre os padrões de vídeo mais comuns, com suas respectivas resoluções, pode-se citar:

- MDA (*Monochrome Display Adapter*) e CGA: São padrões primitivos, não mais utilizados devido à grande limitação de cores e resolução. O MDA limitava a exibição de textos em 25 linhas por 80 colunas de resolução, com no máximo 2000 caracteres por tela, não suportando mais que duas cores. Já o CGA podia exibir gráficos numa resolução de 320 x 200, com suporte de até 4 cores simultaneamente. Em resumo, eram padrões restritos a interfaces de texto, mas que atendiam aos requisitos exigidos pelos primeiros PCs lançados;
- EGA (*Enhanced Graphics Adapter*): Padrão de vídeo compatível com o CGA desenvolvido pela IBM entre os anos de 1984 e 1985, especialmente para os novos computadores PC AT, com suporte a resoluções de até 640 x 350 e 16 cores simultâneas de um total de 64 cores disponíveis;
- VGA: Com uma palheta de 262.000 cores, suporta exibição de 256 cores simultaneamente, com resolução de 640 x 480. Foi uma grande revolução sobre os padrões mais antigos e se tornou requisito mínimo para rodar o Windows 95/98. Foi ainda aperfeiçoadado para exibição de 16 cores com resolução de 800 x 600 e representou um marco tão importante que todas as alterações posteriores basearam-se nesse padrão. A proposta de aplicação desse trabalho utiliza esse padrão clássico de vídeo com uma resolução de 640 x 480;
- SVGA (*Super VGA*): É um padrão mais atual, capaz de exibir 24 bits (mais de 16 milhões) de cores, trazendo uma maior realidade às imagens produzidas na tela, sendo assim também chamado de *true-color* (cores reais). O padrão VESA-1 (*Video Electronics Standards Association*) suportava resoluções de 320 x 200 até 1280 x 1024. Com o tempo foram lançados VESA-2 e o VESA-3 com resoluções variadas chegando até a 1600 x 1200.

A resolução e a quantidade de cores simultâneas que uma interface de vídeo é capaz de exibir estão relacionadas à quantidade de memória de vídeo que a interface possui [18]. Assim, é possível definir, para determinada resolução, quanto de memória se faz necessária aproximadamente, conforme a equação 2.1 (o resultado é dado em bytes).

$$\boxed{\text{resol. horiz} \times \text{resol. vert} \times \text{bits por pontos} \div 8 = \text{quant. min. de memoria de video}} \quad (\text{Eq. 2.1})$$

Atualmente, com o surgimento da tecnologia de alta definição nas TVs, por exemplo, varias outras faixas de resolução são criadas, e novos aperfeiçoamentos são lançados constantemente, inclusive para monitores de computadores pessoais – monitores LCD. Algumas dessas principais resoluções podem ser vistas na tabela 2.2.

Tabela 2.2 – Resoluções para recepção de alta definição utilizadas nas TVs.

HDTV	Painel (tipo de resolução)	Linhas Verticais (largura da tela)	Linhas Horizontais (Altura da tela)	Relação Visual (formato da tela)
Sim	1080 i (Full HD)	1920	1080	16:9
Sim	WXGA	1366	768	16:9
Sim	WXGA	1280	768	15:9
Sim	720 p	1280	720	16:9
Sim	XGA	1024	768	4:3
Não	SVGA	800	600	4:3
Não	WVGA / EDTV	852	480	16:9
Não	VGASDTV NTSC	640	480	4:3

2.3.2 – MONITORES

O monitor é um dispositivo de saída conectado através de um cabo (VGA) à placa principal, que traduz os sinais recebidos para os tipos de sinais empregados na exibição de imagens. Inicialmente eram usados para exibir apenas caracteres em apenas uma cor, como mostrado nos primeiros padrões de vídeo lançados (seção 2.3.1). Posteriormente, com o avanço das interfaces gráficas foram lançados novos modelos em cores e para diversas aplicações gráficas.

Monitores mais antigos contêm um grande tubo de vácuo semelhante aos utilizados em aparelhos de TV de alguns anos atrás [17]. Atualmente, também são utilizados monitores com tela de cristal líquido para PCs mais modernos, inclusive para laptops e notebooks diversos.

Entretanto, nesse trabalho será tratado o monitor CRT (*Cathode Ray Tube*) comum, como hardware principal de vídeo, devido ao acesso mais fácil em laboratórios para testes e estudo, o que não impossibilita o funcionamento do projeto em monitores LCD ou de diferentes tamanhos (polegadas).

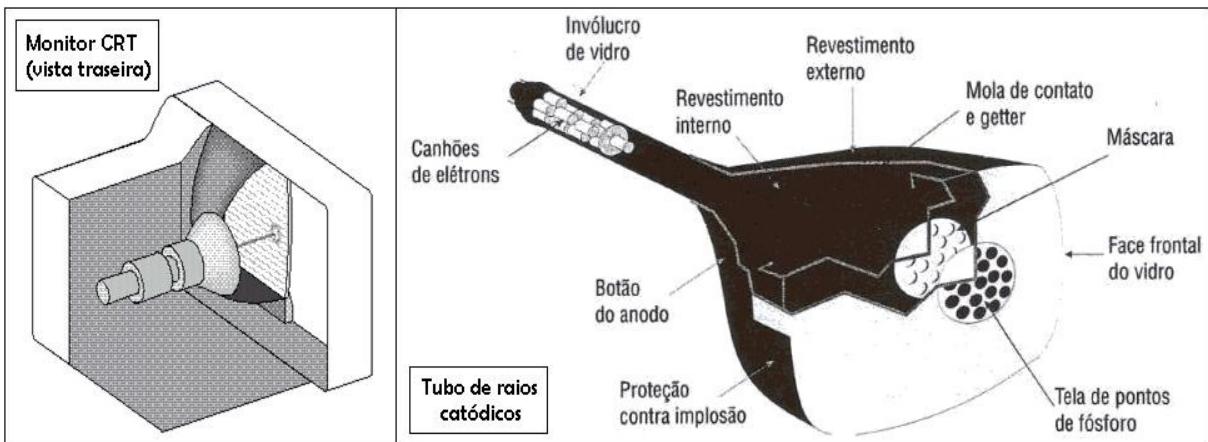


Figura 2.7 – Monitor CRT e representação interna do tubo de raios catódicos.

O tubo de imagem, ou tubo de raios catódicos (CRT), é o coração desse monitor de vídeo. Em seu interior não existe ar e é criada uma diferença de potencial entre duas placas situadas dentro de um “pescoço alongado” [18]. Essa ddp é alta o suficiente para que os elétrons saltem de uma placa para outra. A imagem então é formada pela emissão desses elétrons através de um chamado canhão de elétrons sob uma superfície coberta por uma camada de material fosforescente, gerando um ponto na tela (pixel).

Em cada etapa é desenhado um novo ponto na tela até que se realize uma varredura completa na tela. A intensidade do brilho do ponto é controlada pelo nível de tensão introduzido pelo sinal de vídeo e o controle do caminho percorrido pelo feixe de elétrons é feito através de placas de deflexão horizontais e verticais, que produzem um campo magnético determinando o caminho a ser percorrido pelo elétron [17]. Essas placas são responsáveis então pelos sinais de sincronismo a serem gerados para a correta varredura da tela, tratados na seção 2.3.4.

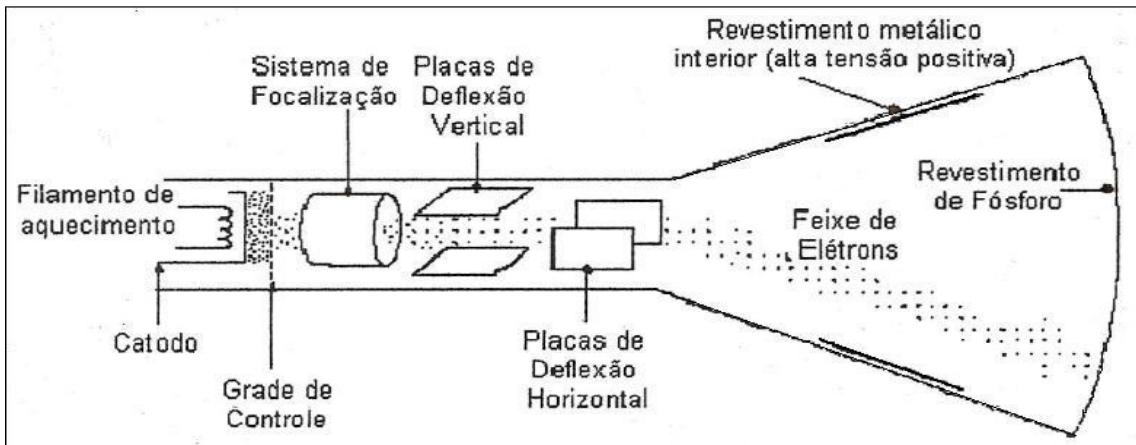


Figura 2.8 – Vista em corte de um tubo de monitor, com suas placas de deflexão.

Portanto, a formação da imagem dar-se-á quando o fluxo de elétrons atingir a máscara metálica revestida de fósforo, localizada logo atrás da tela de vidro, fazendo-a brilhar de acordo com o ponto bombardeado [18]. Em monitores monocromáticos há somente uma máscara de fósforo, e a cor da tela será dada justamente pela cor do fósforo (em geral, verde, laranja ou branco). Em um monitor colorido, existem três máscaras, cada uma contendo uma camada de fósforo de acordo com as três cores primárias de vídeo (vermelho, verde, azul) – conhecido como sistema RGB (*Red – Green – Blue*).

2.3.3 – SISTEMA DE CORES RGB

É um sistema que permite a construção, através de combinações simples, de todas as cores perceptíveis ao olho humano [18]. Esse sistema emite informações de luz em três níveis, formando o preto (nenhuma luz) ou branco (máxima iluminação) ou tonalidades de cor, como representado no cubo de cores da figura 2.9, ao se misturar três cores básicas – o vermelho, o verde e o azul. Uma aplicação comum desse modelo é a tela ou *display* de um monitor com tubo de raios catódicos.

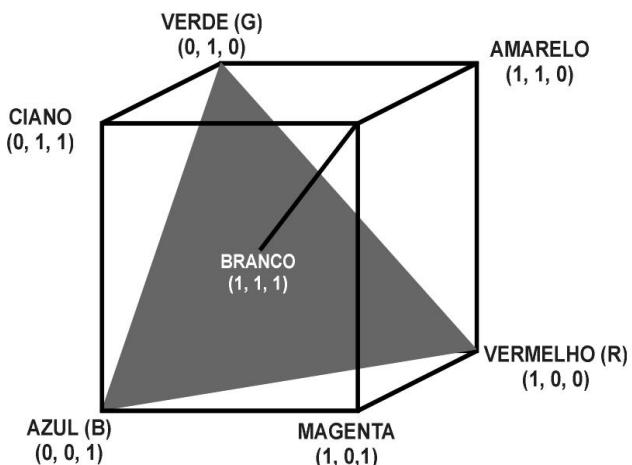


Figura 2.9 – Cubo das oito cores básicas geradas no sistema RGB.

A capacidade de manipular *pixels* (pontos na tela) de modo a mostrar uma combinação de pontos acessos e apagados pode ser interpretada como a apresentação de gráficos na tela [18]. Por exemplo, pode-se definir o bit ‘1’ para um ponto acesso e bit ‘0’ para um ponto apagado, de acordo com a posição desse ponto dentro da memória de vídeo. Com a

combinação das tonalidades de cada cor em cada *pixel* gerado pode ser obtida uma grande quantidade de cores, como apresentado na tabela 2.3.

Tabela 2.3 – Relação entre as cores disponíveis e a quantidade de bits por pixel na memória de vídeo.

QUANTIDADE DE BITS POR PONTO	CORES SIMULTÂNEAS DISPONÍVEIS
2	4
4	16
8	256
15	32.768 (32K) (Hi Color)
16	65.536 (64K) (Hi Color)
24	16.777.216 (16M) (RGB True Color)
32	4.294.967.296 (4G) (CMYK True Color)

O funcionamento do sistema RGB ocorre com três canhões de elétrons presentes no tubo, sendo um para cada cor. Assim, cada canhão mira na máscara de fósforo de modo que o fluxo de elétrons passe por um furo específico ativando, ao final do processo, três pontos (um vermelho, um verde e um azul) [17], ou seja, os elétrons disparados por um dos canhões só atingem os pontos de fósforo verde, os elétrons do segundo atingem apenas os pontos de fósforo vermelho, e o terceiro no azul. Esses pontos, por sua vez, são combinados de modo a compor um único *pixel* na tela, conforme a figura 2.10.

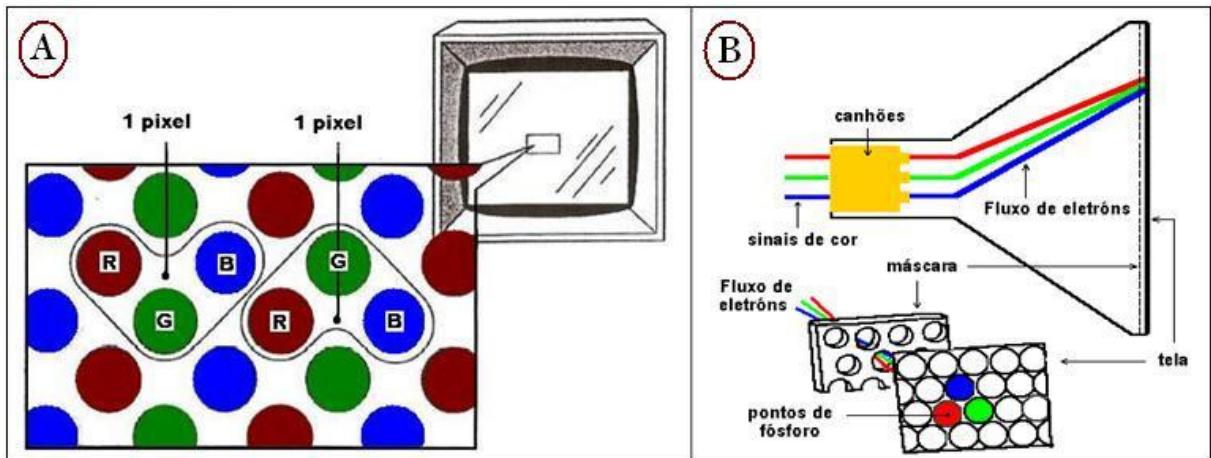


Figura 2.10 – A tríade de pixels em um monitor colorido (a) e o fluxo de elétrons das cores no tubo (b).

Os sinais das cores R, G e B são analógicos e devem ficar entre zero e 0,7 V [11], sendo este o valor máximo equivalente à saturação da cor. Portanto, o sistema RGB é baseado em luz, que ao ser projetada pelo tubo gera as cores perceptíveis ao olho humano. Como essas cores utilizam a luz para serem projetadas, então elas não podem ser impressas, apenas

visualizadas no monitor. Outro sistema (CMYK), onde a cor é formada por pigmentos, é responsável pela formação da impressão.

2.3.4 – A CONFIGURAÇÃO DO MODO VGA NO CYCLONE II

O kit de desenvolvimento possui um conversor digital analógico VGA de 4 bits produzindo uma saída com resolução de 640 x 480 a uma freqüência de 25 MHz, suportando até 100 MHz de *refresh* (atualização da tela) [4]. Em nível de hardware, o FPGA fornece a sincronização por um conector D-SUB de 16 pinos localizado na placa, com uma rede de resistores para produzir os sinais de dados analógicos. O sistema RGB no cyclone II é obtido a partir de 4 bits para cada sinal cor (vermelho, verde e azul), permitindo assim o alcance de diversas tonalidades – 4 bits equivalem a 16 possíveis tonalidades em cada cor. A figura 2.11 mostra o circuito VGA do kit Cyclone II.

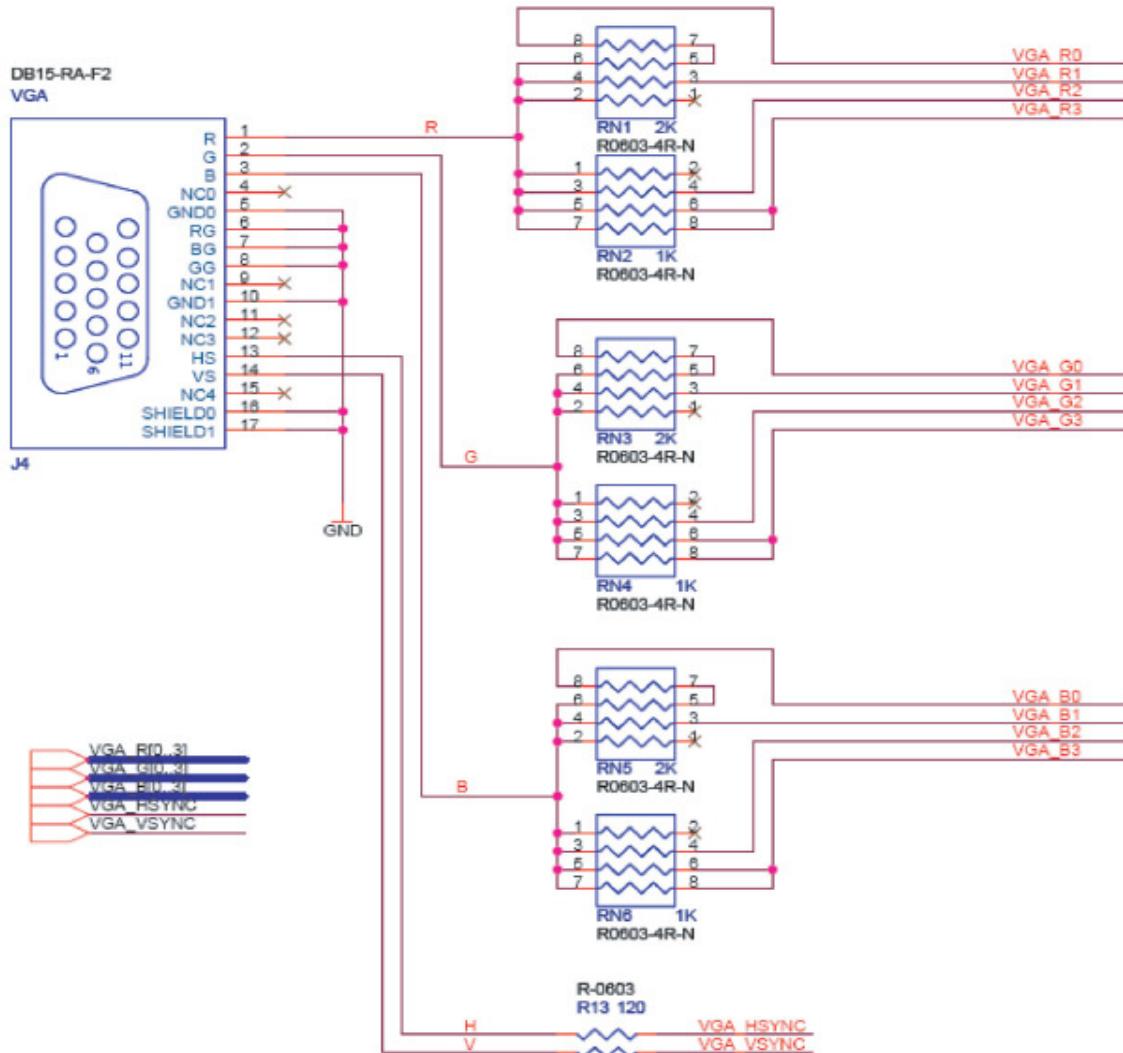


Figura 2.11 – Esquemático do circuito VGA no FPGA (Cyclone II).

Para a construção de imagens no monitor, deve haver um deslocamento organizado do fluxo de elétrons na máscara da tela [18]. Esse deslocamento é chamado de varredura e ocorre de forma seqüencial, iniciando-se na primeira linha e na primeira coluna do canto superior esquerdo da tela, varrendo todos os pontos até completar o final da linha e assim, partir para a segunda linha, conforme esquematizado na figura 2.12. Essa verificação se completa ao serem percorridas todas as linhas da tela, ou seja, é finalizado no canto inferior direito da tela do monitor.

Um controlador de vídeo é responsável por gerar sinais de sincronismo e as saídas de dados para cada pixel de forma serial. O sincronismo horizontal (*hsync*) especifica o tempo requerido para percorrer uma linha, o vertical (*vsync*) controla o tempo necessário para percorrer toda a tela [11]. Dessa forma, um circuito digital de controle dos sinais de sincronismos requer que seja definida, de forma objetiva, a disposição da parte visível na tela do monitor.

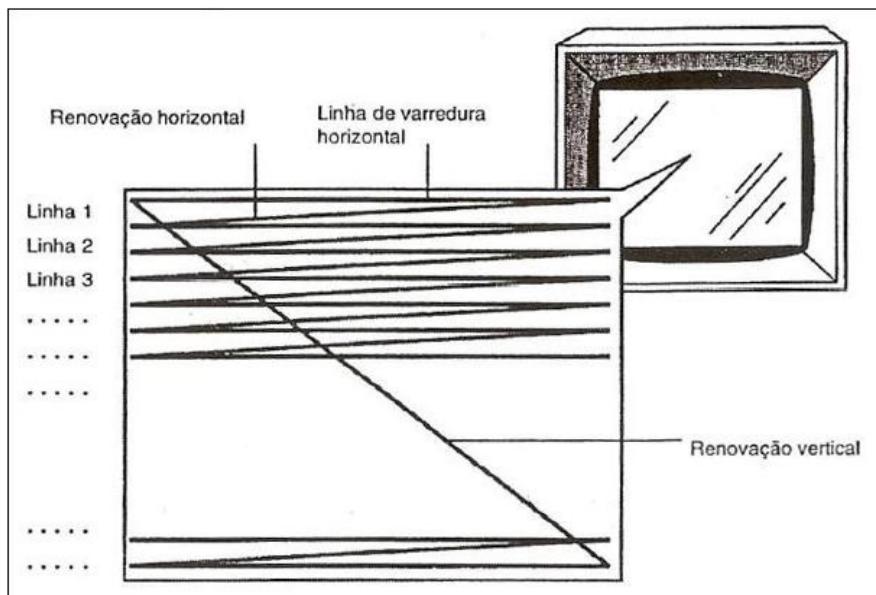


Figura 2.12 – Varredura horizontal e vertical na formação das cores RGB.

Os monitores CRT, durante o sincronismo horizontal, apresentam uma pequena borda preta ao redor da imagem, definidas como borda da esquerda (*back porch*) e borda da direita (*front porch*). Também no sincronismo vertical há uma borda inferior e uma borda superior – *bottom border* e *top border* [11]. Para uma tela com padrão VGA e resolução de 640 x 480 pixels, usada nesse projeto, os valores dessas bordas, da área visível (*display*) e do pulso em nível lógico baixo (*retrace*) para gerar o sincronismo horizontal e vertical, são mostrados nas figuras 2.13 e 2.14, respectivamente.

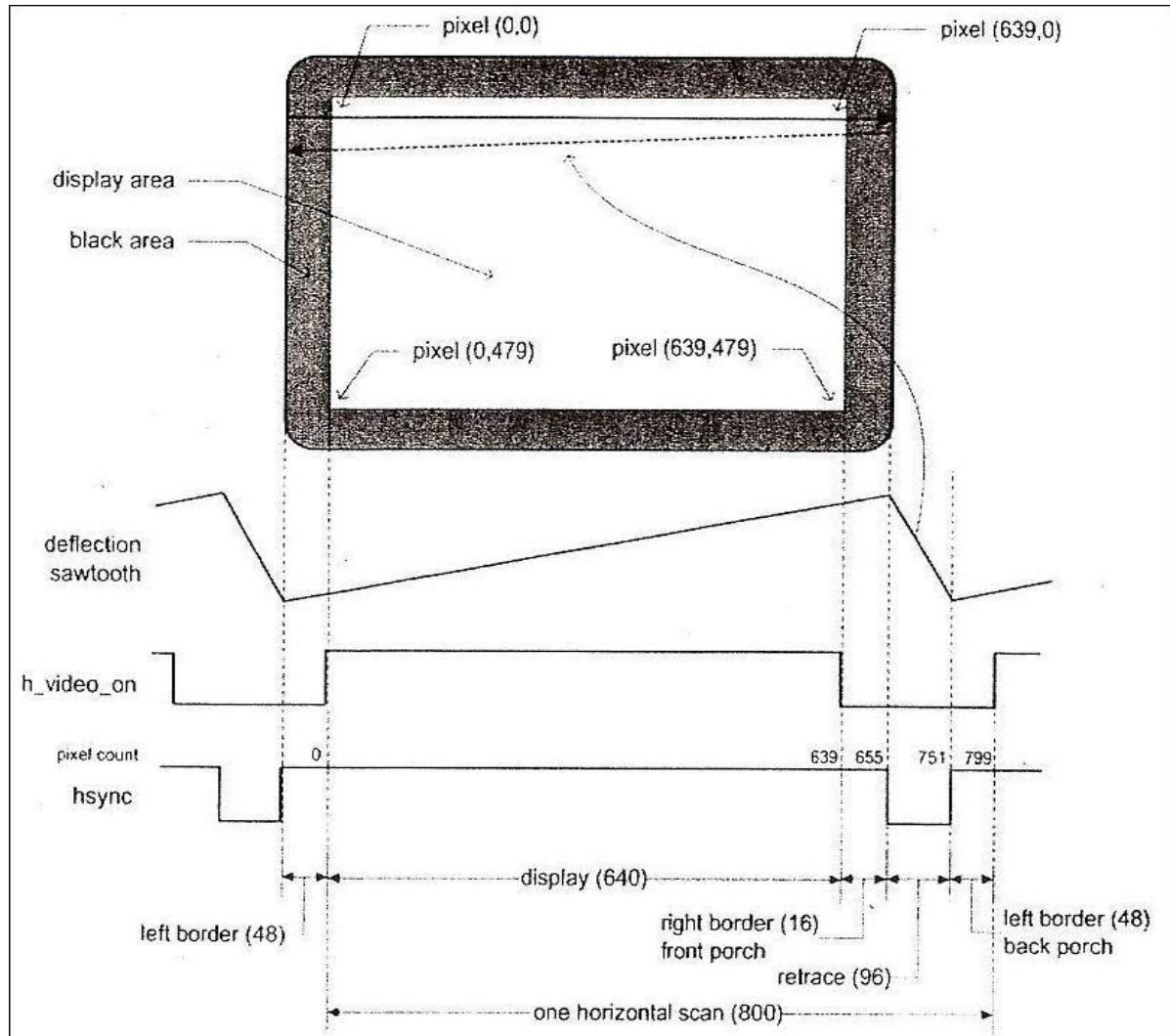


Figura 2.13 – Diagrama de tempo do sincronismo horizontal.

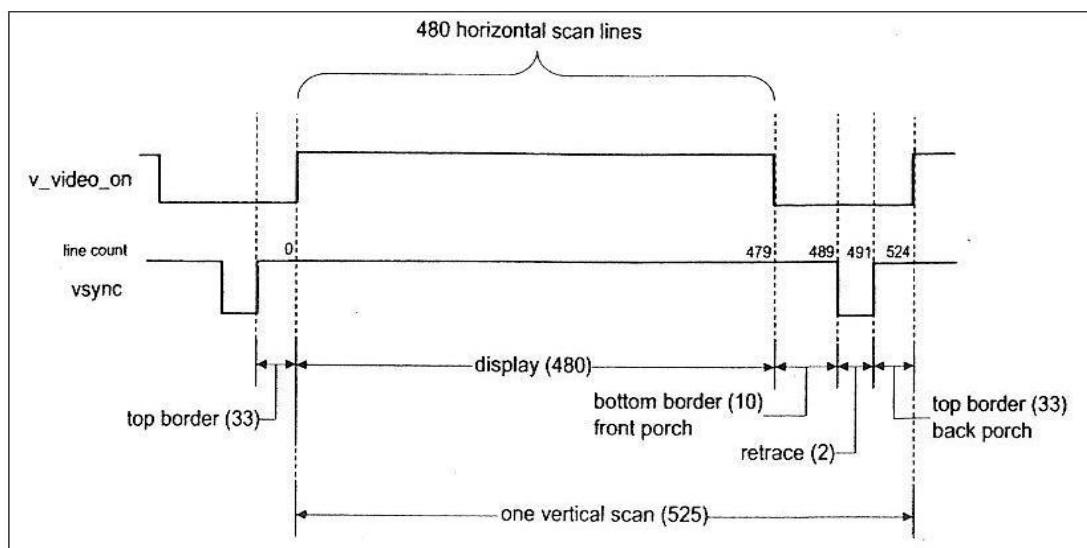


Figura 2.14 – Diagrama de tempo do sincronismo vertical.

Tabela 2.4 – Especificações de tempo para sincronismo no Cyclone II.

<i>Especificações Horizontais</i>						
Configuração	Resolução (HxV)	a (μ s)	b (μ s)	c (μ s)	d (μ s)	Pixel clock (MHz)
VGA (60 Hz)	640 x 480	3.8	1.9	25.4	0.6	25 (640/c)
<i>Especificações Verticais</i>						
Configuração	Resolução (HxV)	a (linhas)	b (linhas)	c (linhas)	d (linhas)	
VGA (60 Hz)	640 x 480	2	33	480	10	

A tabela 2.4 apresenta os valores especificados [4] para o sincronismo horizontal e o sincronismo vertical do modo de vídeo no FPGA, onde:

- “a” é o pulso de sincronismo (*retrace*);
- “b” é a borda anterior da tela (*back porch*);
- “c” é a área visível de display do vídeo;
- “d” é a borda posterior da tela (*front porch*).

Essa varredura deve se repetir constantemente, pois os pontos de luz emitidos pela máscara de fósforo têm uma duração curta e apaga-se rapidamente. Um determinado ponto fica aceso o tempo suficiente para a varredura do próximo quadro [21]. Esse processo é conhecido como *refresh* (ou atualização) da tela.

Para um monitor com uma freqüência de 60 Hz (60 quadros por segundo) é fácil deduzir que ele gastará 1/60 s na varredura de cada quadro. Assim, quanto maior a freqüência vertical de um monitor de vídeo, mais quadros por segundo ele apresentará e, consequentemente, melhor será a qualidade da imagem e menor será o prejuízo à visão do usuário, pois a atualização ocorrerá de forma imperceptível ao olho humano [11]. Valores abaixo de 60 Hz podem ser prejudiciais, pois é fácil notar a tela piscando ao fazer o *refresh* quadro a quadro da imagem.

2.4 – CONCLUSÕES

O protocolo de comunicação de um teclado PS/2 é, de certa forma, bem simplificado, pois ele possui dois sinais internos próprios – um de *clock* e um de dados – que ficam ativos ou prontos a partir do momento que o teclado é alimentado (conectado na placa). Assim, ao

pressionar uma tecla imediatamente é gerado um código que precisa apenas ser tratado, ou convertido, para o uso ou comandos desejados no projeto.

Um cuidado importante deve ser sempre lembrado no tratamento dessa comunicação de dados pelo teclado. Uma vez pressionada um tecla, haverá a formação de três palavras “distintas” de dados – o primeiro *scancode*, com os bits da ação de pressionar tecla; o segundo, com o mesmo tamanho e bits em zero (intervalo de *Acknowledgement*); e o *scancode* número três, referente à ação soltar tecla.

No dispositivo de saída de vídeo VGA, os sinais de sincronismo horizontal e vertical são a chave principal para o correto funcionamento do protocolo. A partir deles é definida a área disponível para as imagens e animações do projeto em qualquer local da tela desejada, facilmente escolhida pelo usuário – dentro da resolução de 640 *pixels* por linha em um total de 480 linhas por tela.

Quanto aos sinais de cores R, G e B, apenas são definidos pelo projetista quando, onde e em que momentos da aplicação eles devem estar acessos (ativos), sendo necessária apenas sua configuração de acordo com os pinos no FPGA, como mostra a tabela 2.5 [4].

Tabela 2.5 – Pinos para configuração de vídeo no FPGA.

Sinal	Pinos do FPGA	Descrição
VGA_R[0]	PIN_D9	VGA Red[0]
VGA_R[1]	PIN_C9	VGA Red[1]
VGA_R[2]	PIN_A7	VGA Red[2]
VGA_R[3]	PIN_B7	VGA Red[3]
VGA_G[0]	PIN_B8	VGA Green[0]
VGA_G[1]	PIN_C10	VGA Green[1]
VGA_G[2]	PIN_B9	VGA Green[2]
VGA_G[3]	PIN_A8	VGA Green[3]
VGA_B[0]	PIN_A9	VGA Blue[0]
VGA_B[1]	PIN_D11	VGA Blue[1]
VGA_B[2]	PIN_A10	VGA Blue[2]
VGA_B[3]	PIN_B10	VGA Blue[3]
VGA_HS	PIN_A11	VGA H_SYNC
VGA_VS	PIN_B11	VGA V_SYNC

CAPÍTULO 3

BLOCOS DA APLICAÇÃO TETRIS®

3.1 – INTRODUÇÃO

Este capítulo apresenta uma breve explanação a respeito do funcionamento do jogo TETRIS®, qual seu objetivo e o que é necessário para sua composição. Foi ainda escolhido o seu visual e características gerais na implementação desse trabalho em especial.

São criados blocos, a partir de códigos em linguagem VHDL, para melhor representar a arquitetura e formatação da aplicação, interligando logicamente as interfaces de E/S utilizadas. Os comentários e explicações dos códigos e da construção de cada bloco são mostrados ao longo do capítulo.

3.2 – O JOGO TETRIS®

TETRIS® é um jogo eletrônico muito popular, desenvolvido em 1984 por engenheiros russos especialistas em ciência da computação. A figura 3.1 mostra um exemplo clássico da interface do jogo. O objetivo é encaixar peças de diversos formatos que descem do topo de uma tela, de modo que, quando uma linha é completada, ela desaparece e a pontuação do jogador aumenta. O jogo acaba quando as linhas horizontais incompletas se empilham até o topo da área das peças na tela.

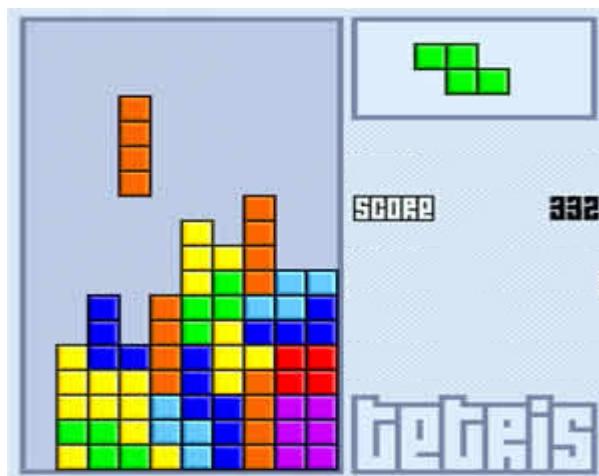


Figura 3.1 – Exemplo de formatação do jogo tetris.

O sucesso desse simples jogo é tão grande, que até hoje são criadas novas variações do jogo, com novos visuais gráficos e novas peças com características especiais, como transparência ou peças “explosivas” que apagam outras peças, por exemplo.

Para a aplicação desejada nesse trabalho, algumas características da jogabilidade (movimentos) do jogo foram tomadas como críticas (ou essenciais):

- As peças devem rotacionar em sentido horário. Também é permitido o movimento lateral (esquerda ou direita) a qualquer momento, durante a descida, a partir de comandos dados no teclado.
- Essas peças também devem descer de modo constante, com um tempo pré-estabelecido de aproximadamente uma posição a cada segundo.
- Ao chegar ao fim da tela ou ao encontrar outra peça na descida, a peça atual deve parar imediatamente e acionar sua rotina de verificação (analisar se há linha cheia, incrementar pontuação, ou finalizar o jogo, por exemplo).
- As peças devem ser de formas variadas e serem escolhidas sempre de modo aleatório, de modo a garantir uma boa jogabilidade ao jogo.
- Deve existir uma tecla ou botão de *reset* que zera as variáveis do jogo e reinicia com a tela limpa e uma nova peça.

3.3 – BLOCO DE LEITURA DO TECLADO

A formação do projeto TETRIS® é composta por oito blocos, que representam a divisão do código em trechos menores a fim de proporcionar um melhor entendimento, uma maior organização do projeto e facilitar a busca por erros de compilação. Além disso, a criação dos blocos objetiva também modificações futuras, como adição de novas peças ou o uso de sons durante o jogo, por exemplo.

A figura 3.2 apresenta o diagrama de blocos completo do projeto no modo esquemático do QUARTUS II®, com a união de todas as oito partes e a configuração aos pinos de entrada/saída do kit de desenvolvimento.

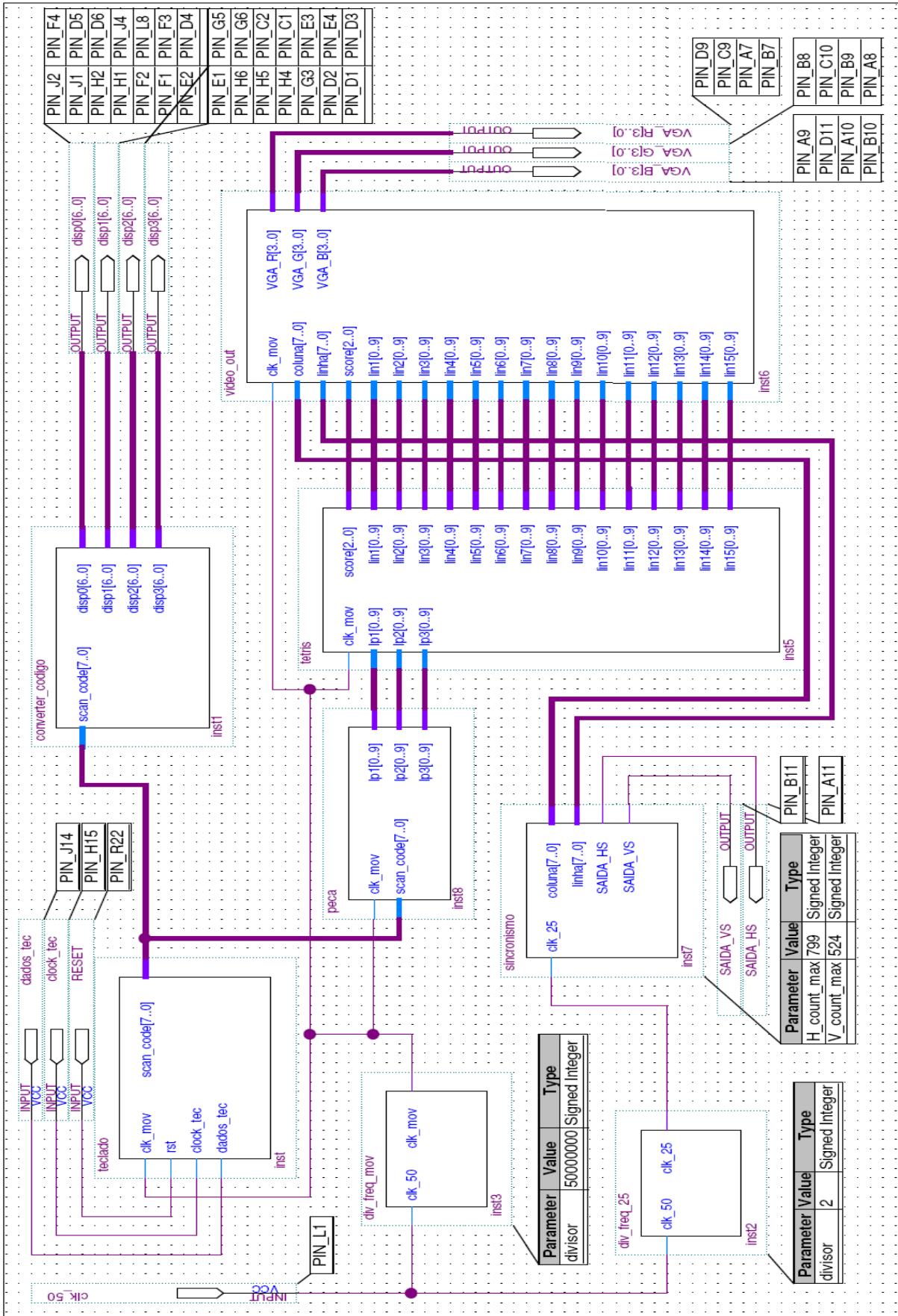


Figura 3.2 – Diagrama de blocos final do projeto e seus respectivos pinos.

O bloco de leitura do teclado tem como portas de entrada um sinal de *reset* (para limpar as variáveis), um sinal de *clock* do teclado e um sinal de dados do teclado; e como porta de saída o *scancode* lido com tamanho 8 bits, que servirá para exibição nos *displays* (bloco *converter_codigo*) e, principalmente, como comandos do jogo na geração das peças e de seus movimentos (bloco *peça*). Na arquitetura do código são definidos, inicialmente, duas variáveis do tipo sinal. O contador, que é um inteiro iniciado em 0 e podendo variar de 0 a 32, ou seja, é um contador que realiza uma contagem máxima de 33 valores. E o sinal chamado *byte*, que é um vetor de 8 *bits* onde será guardado a palavra completa de dados gerados em *scancode*.

O processo principal (PROCESS: *ler_scancodes*) se inicia com um IF de configuração do *reset* (“rst”), que retorna um valor zero para o contador e para o *scancode*, se pressionado. Caso o reset não seja pressionado, então é aguardada uma borda de descida no *clock* do teclado (leitura de dados do teclado), e o contador é incrementado em uma unidade.

A partir desse ponto é iniciada uma rotina de tratamento, do tipo CASE IS, para cada incremento do contador. É importante lembrar que a ocorrência da transmissão completa de um dado se dá em 11 bits – um *start* bit, oito bits de dados, um bit de paridade e um *stop* bit, como exposto no capítulo 2, seção 2.2.

Desse modo, no código do bloco teclado, quando o contador for 0, então nada é feito, pois significa apenas que esse é o bit de partida. Quando o contador tiver o valor de 1 a 8, ele vai sendo incrementado e o sinal *byte* vai recebendo os dados do teclado (bit a bit). Nos próximos dois valores (9 e 10), representados pelos bits de paridade e o *stop* bit (sendo este último, em nível lógico 1), o contador é novamente incrementado e a saída *scancode* recebe o valor final do sinal *byte*. Por fim, quando o contador for 11 é gerado um *acknowledgement* (*ACK*), em nível lógico 0, que representa o início de um intervalo de onze pulsos em 0 – palavra de dados vazia. Os últimos 11 bits da contagem estão relacionados à ação de soltar a tecla, que gera um outro *scancode* diferente, mas que não é de interesse para a aplicação tratada nesse projeto e, assim, não é posto como saída no bloco gerado.

Em resumo, são 33 bits gerados na ação completa de entrada de dados, sendo 11 bits ao pressionar a tecla, 11 bits de espaço vazio (nível lógico 1) de reconhecimento ou *acknowledgement* e, finalmente, os últimos 11 bits ao soltar a tecla.

3.4 – BLOCO DE CONVERSÃO

Essa parte do código tem declarado em sua entidade uma entrada principal (e única), que recebe o byte de dados (*scancode*), e quatro saídas representando os quatro displays de sete segmentos do kit de desenvolvimento. A função desse bloco é apenas a conversão da palavra gerada, em *scancode*, para um valor ASCII equivalente (conforme a tabela em anexo).

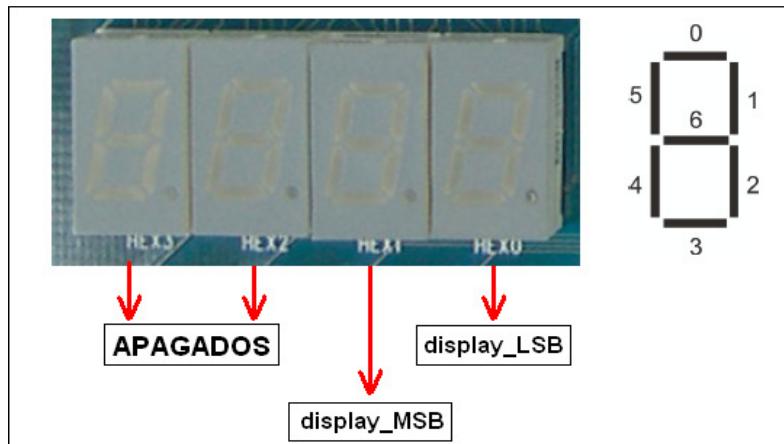


Figura 3.3 – Displays de sete segmentos utilizados na aplicação.

Após a declaração das portas na entidade, a arquitetura inicia-se com a definição de uma tabela de conversão para o display de sete segmentos (figura 3.3). Sabendo-se que, em nível lógico ‘0’ o *display* permanece apagado e em nível lógico ‘1’ fica aceso, então são definidas constantes de 7 bits, representando os valores em hexadecimal de 0 a F, incluindo um valor apagado, para os dois primeiros *displays*, que não serão utilizados.

O primeiro processo (*conv_scan_code*) toma como referência principal a tabela de *scancodes* em anexo, dos valores em ASCII para cada palavra gerada no teclado. Um sinal chamado *ascii* recebe os números e caracteres principais, todos em valores hexadecimais, como x“30”, por exemplo.

Por fim, dois trechos de saída definem o display menos significativo (*display_LSB*) e o mais significativo (*display_MSB*) para a exibição desejada. Os outros dois displays restantes recebem o valor apagado especificado no parágrafo anterior (valor “1111111”).

3.5 – BLOCOS DIVISORES DE FREQUÊNCIA

A princípio são necessários dois blocos divisores de freqüência principais: o “*div_freq_25*” e o “*div_freq_mov*”. Ambos têm como entrada o *clock* principal de 50 MHz do FPGA (pino L1) e uma estrutura similar, com a diferença do *clock* gerado em suas saídas:

- DIV_FREQ_25: tem como finalidade a geração de um *clock* de 25 MHz, que será utilizado para o funcionamento da interface de vídeo VGA. Em sua entidade, além da entrada *clk_50* e da saída *clk_25*, já citadas acima, é declarada uma constante genérica chamada de divisor, com valor igual 2. A arquitetura desse bloco é bem simples e formada por um único processo, que faz uso de uma variável auxiliar para fazer a comparação com o *clock* de 50 MHz, de modo a dividir essa freqüência pela metade, formando a saída de 25 MHz desejada;
- DIV_FREQ_MOV: Esse bloco, por sua vez, utiliza o mesmo modelo simplificado do bloco anterior, também com a mesma entrada do *clock* interno de 50 MHz do FPGA, mas com uma saída de *clock* de 1 Hz (*clk_mov*), que é obtida com um divisor agora igual a 50000000. Essa freqüência será utilizada, no bloco de vídeo, para representar o movimento de descida constante das peças do jogo, equivalente ao tempo de 1 segundo. Esse *clock* é muito importante no sincronismo dos movimentos e da animação geral do jogo.

3.6 – BLOCO PEÇA

Este bloco tem como objetivo principal a configuração, formatação e movimentação da peça em geral. O “*clk_mov*”, gerado no divisor de freqüência anterior, é uma entrada responsável pelo sincronismo de cada movimento – esquerda, direita e giro – da peça. A outra entrada do bloco é o *scancode*, que dita a escolha de cada tecla responsável por cada movimento desejado. Como saída, são dadas três linhas referentes à peça criada (*lp1*, *lp2* e *lp3*), sendo estas atualizadas a cada ciclo de *clk_mov*, de acordo com sua nova posição.

A arquitetura é bem simples e pode ser resumida em um único processo, que ocorre a cada ciclo do “*clk_mov*”, e que, por sua vez, é composto de três rotinas IF/ELSE da seguinte forma:

- Primeiro IF: Se o “scan_code” for igual a 16 (em hexadecimal) – equivalente à tecla número 1 – então as três linhas da peça rotacionam uma posição, um bit, para direita. Isso equivale a um movimento para esquerda da peça;
- Segundo IF (ELSIF): Caso não seja a tecla 1, se o “scan_code” for igual a 1E (em hexadecimal) – referente à tecla 2 do teclado – então as linhas da peça agora rotacionam um bit para o outro lado. Nesse caso a peça é movimentada para a direita;
- Terceiro IF (ELSIF): Aqui ocorre uma sub-rotina de giro da peça. Ela ocorre quando for pressionada a tecla SPACE do teclado, ou seja, quando o “scan_code” for igual a 29, em hexadecimal. Se for pressionada essa tecla, um contador com valor máximo igual a 4 é incrementado, e para cada incremento desse contador são atualizadas as três linhas da peça, de modo a formar as 4 possíveis posições de giro da mesma. Ao final da contagem é reiniciado o contador e a peça gira novamente, voltando à sua posição normal.

Por fim, caso não ocorra nenhuma dessas possibilidades, ou seja, se não for pressionada nenhuma dessas teclas definidas, as linhas de saída apenas recebem os valores pré-definidos inicialmente como sinais auxiliares.

Dois pontos são importantes de serem observados na construção desse bloco peça. Primeiro que a atualização desses movimentos na peça ocorre a cada 1 segundo aproximadamente, uma vez que o processo principal depende diretamente do “clk_mov” definido para esse fim. E o segundo ponto é que o movimento de descida não ocorre neste bloco. Este movimento será especialmente tratado no “bloco tetris” seguinte.

3.7 – BLOCO TETRIS

Apesar de um pouco extenso – com mais de 700 linhas de código VHDL – a formação desse bloco é de fácil compreensão e basicamente é apenas bem repetitivo. Isso se deve ao fato da quantidade de linhas definidas como área principal do jogo – 15 linhas – que devem ser comparadas, uma a uma, com cada uma das linhas da peça (lp1, lp2 e lp3). Cada linha é formada por 10 bits, que permite assim espaço suficiente para movimentos laterais. A definição da posição de cada linha bem como seu tamanho, é abordada nas próximas seções.

Para o propósito do algoritmo de formação do “bloco tetris”, apenas deve-se tomar uma área (15 x 10 pontos) em lugar qualquer do espaço, como esquematizado na figura 3.4. O

objetivo é comparar a cada segundo (a cada ciclo de “clk_mov”) as linhas lp3, lp2 e lp1 com as linhas imediatamente abaixo delas (lin1, lin2, lin3, etc.) de modo que se o resultado for positivo (se a peça puder descer) então a linha seguinte recebe o valor da anterior, e assim sucessivamente, como por exemplo: *lin1* recebe *lp3*. Dessa forma a peça vai descendo linha a linha, a cada segundo até ocorrer uma exceção (fim da tela ou encontrar um obstáculo no caminho de descida).

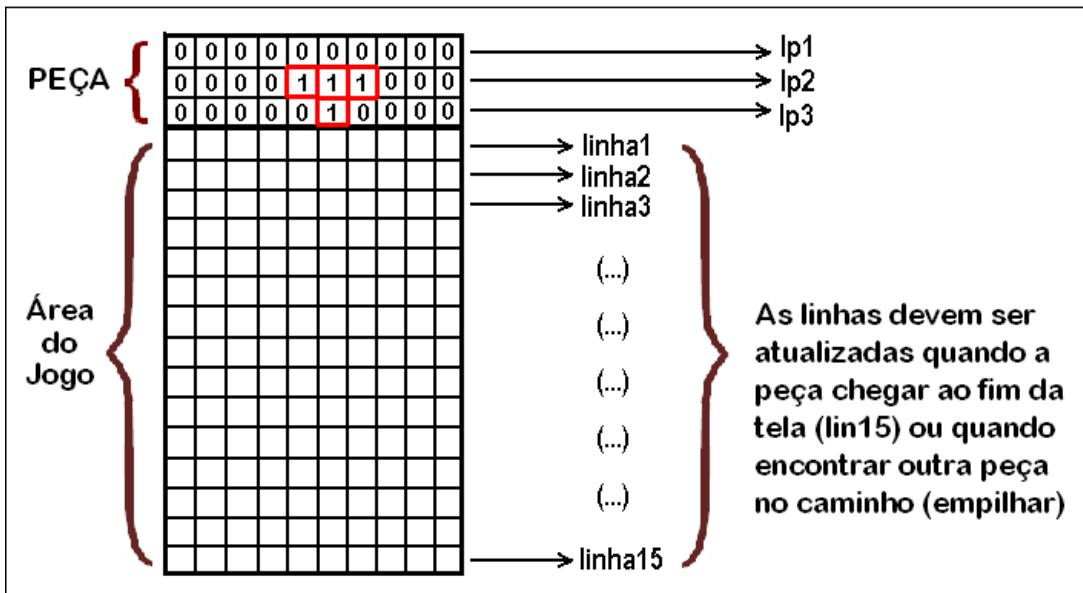


Figura 3.4 – Representação da área do jogo (15 linhas) e da peça (3 linhas) para o algoritmo de “descida”.

Em termos de código, é criado um contador “tempo” de valor máximo 16, que representa a rotina de comparação para cada uma das 15 linhas, com as linhas da peça. Dentro do processo principal, que ocorre a cada “clk_mov”, se o “tempo” for igual a 1, então a comparação ocorrerá na linha 1, e serão atualizadas todas as linhas (*lin1_next*, *lin2_next*, *lin3_next*, etc.) e os registradores de cada linha (*lin1_reg*, *lin2_reg*, etc.).

Contudo, se esta linha for agora uma linha cheia (“1111111111”), ocorre então outra sub-rotina que limpará a linha e somará uma pontuação (nesse jogo a pontuação máxima foi definida em 5 pontos). O processo de limpar linha é bem intuitivo e é mais bem entendido com um exemplo: SE a linha 5, por exemplo, for uma linha cheia, ENTÃO “*lin5*” recebe “*lin4*”, a linha 4 receberá a linha 3, “*lin3*” recebe “*lin2*”, “*lin2*” recebe “*lin1*”, e a linha 1 recebe zero. Obviamente as outras linhas abaixo da linha 5, não mudam e devem receber seus valores anteriores (“*lin6*” recebe “*lin6*”, “*lin7*” recebe “*lin7*”, etc.).

Após essa rotina completa na linha 1, o contador “tempo” é incrementado e o código passa (no próximo segundo) a verificar toda a comparação novamente agora para a linha 2.

Dessa forma é verificado, linha a linha, se a peça pode descer, se a linha atual é uma linha cheia e, caso seja linha cheia, somar pontuação, atualizar tela e reiniciar o contador “tempo” para a próxima peça.

Quanto à operação realizada nas comparações, é importante lembrar como funcionam os operadores lógicos NAND e OR, que são utilizados no desenvolvimento desse bloco. Essas operações comparam, bit a bit, uma linha com outra de mesmo tamanho (bit_vector (0 TO 9), ou seja, 10 bits).

Tabela 3.1 – Algumas operações lógicas básicas entre dois bits.

AND			NAND			OR			NOR		
A	B	S	A	B	S	A	B	S	A	B	S
0	0	0	0	0	1	0	0	0	0	0	1
0	1	0	0	1	1	0	1	1	0	1	0
1	0	0	1	0	1	1	0	1	1	0	0
1	1	1	1	1	0	1	1	1	1	1	0

Observando então a tabela 3.1, a compreensão dessas operações em cada rotina do código fica simples. Tomando como exemplo a rotina de descida na linha 1: A primeira comparação, no primeiro IF, ocorre entre a linha “lp3” da peça (linha mais inferior) com a “lin1_reg” (registrador da posição atual da linha 1). Para saber se a peça poderá desce uma linha, é realizada a operação “lp3 NAND lin1_reg”, que retorna “0” se, e somente se, dois bits consecutivos sejam “1” (ainda conforme a tabela 4.1). Assim, se esta operação retornar uma linha inteira de 1’s (“1111111111”), então significa que ela poderá descer, caso contrário a peça não pode mais descer e devem ser salvas as posições e reiniciado o contador para uma outra peça. Finalmente, ainda se a peça puder descer, deve existir outra comparação para saber se esta linha estará cheia ou não. Então ocorre a operação “lp3 OR lin1_reg”, que retorna um valor “0” se, e somente se, no mínimo dois bits consecutivos sejam “0”. Ou seja, dois bits consecutivos em zero representam uma espaço vazio na linha, logo se o resultado da operação for “1111111111”, essa linha atual está cheia e em consequência será executada a sub-rotina de apagar linha e somar pontuação.

3.8 – BLOCO DE SINCRONISMO

O ponto mais importante para exibição de uma imagem qualquer na tela do monitor é o bloco de sincronismo, onde são gerados os sinais *hsync* e *vsync*, de sincronismo horizontal e vertical, respectivamente.

A entrada desse bloco é composta apenas do clock de 25 MHz, freqüência necessária para o vídeo no FPGA, gerada no bloco “*div_freq_25*” da seção anterior. Lembrando que será utilizada uma resolução de 640 x 480 pixels, são então definidos dois contadores genéricos (*H_count_max* e *V_count_max*) que representam os valores máximos na varredura horizontal (800) e na varredura vertical (525) da tela. Esses valores são obtidos somando-se a área visível do display, as bordas e o pulso de sincronismo, como mostrado abaixo:

- $H_count_max = 800 [640 \text{ (area do display)} + 16 \text{ (front porch)} + 48 \text{ (back porch)} + 96 \text{ (pulso de sincronismo horizontal)}];$
- $V_count_max = 525 [480 \text{ (area do display)} + 10 \text{ (front porch)} + 33 \text{ (back porch)} + 2 \text{ (pulso de sincronismo vertical)}];$

As figuras 3.14 e 3.15, do capítulo anterior, apresentam esses diagramas de tempo da varredura horizontal e vertical, respectivamente [23].

Para representar esses sinais de sincronismo, foram definidas as saídas do tipo bit “*SAIDA_HS*” e “*SAIDA_VS*”. Completando a declaração de portas na entidade do bloco, são dadas ainda outras duas saídas, do tipo inteiro, “*coluna*” e “*linha*”, que representarão os eixos horizontais e verticais da área disponível para manipulação das imagens na tela. A geração desses eixos será melhor comentada ao fim dessa seção.

É importante observar que o sincronismo vertical se dá somente a partir da geração do sinal de pulso do sincronismo horizontal. Fisicamente, pode-se dizer que são percorridos primeiramente todos os pontos horizontais da linha número 1, e só então será iniciada a contagem da segunda linha (varredura vertical). Logo após, serão percorridos todos os pontos da linha número 2, para em seguida passar para a próxima varredura vertical (linha 3); e assim sucessivamente, até serem percorridas todas as 480 linhas da tela. Isso mostra porque a varredura horizontal é medida em *pixels* (pontos) e a varredura vertical é medida em “linhas”. Com isso, agora se observa facilmente que a leitura da próxima linha só inicia após a varredura do ultimo pixel da linha anterior.

A arquitetura do bloco é formada basicamente por três processos distintos: *sinal_Horiz* (sincronismo horizontal), *sinal_Vert* (sincronismo vertical), e *superpixel* (divisão da área visível (640 x 480) por 10 – para facilitar o mapeamento de bits em (64 x 48) pontos).

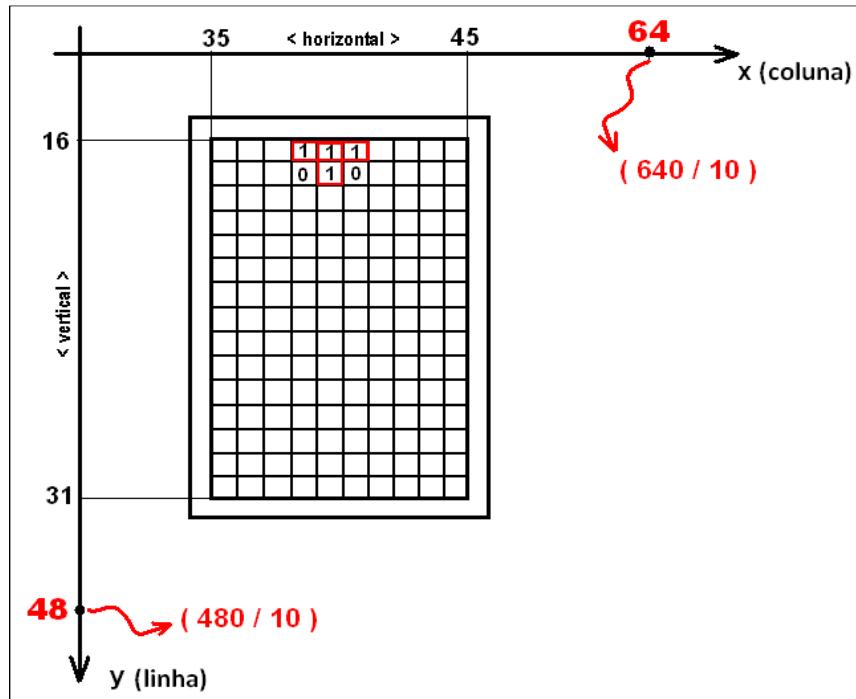


Figura 3.5 – Formação da área visível com uma menor resolução (64 x 48).

Dois contadores, *count_x* e *count_y*, são declarados para representar os valores máximos de contagem no funcionamento dos processos. Um sinal (*clk_vs*) também é criado para representar o pulso de saída horizontal, que será a referência de entrada no sincronismo vertical. E são ainda definidos outros sinais para contagem e endereço das novas linhas e colunas, com menor resolução. A figura 3.5 apresenta a criação da nova área, com resolução 64 x 48.

Assim, o primeiro processo (PROCESS: *sinal_Horiz*), iniciado a partir do clock principal de 25 MHz, realiza uma rotina IF de modo que, se a contagem horizontal for igual ao valor máximo (se tiver no último ponto da linha), então *SAIDA_HS*, *count_x* e *clk_vs* recebem zero; caso não seja o final de uma linha, então, para cada borda de subida do *clk_25*, o contador do eixo-x é incrementado em 1 e se esse contador for igual a 96 é gerado o pulso de saída, para o sincronismo horizontal (*SAIDA_HS*) e para o início da contagem vertical (*clk_vs*).

No segundo processo (PROCESS: *sinal_Vert*), iniciado a cada mudança em *clk_vs*, a rotina anterior se repete de forma bem parecida, dessa vez efetuando incrementos na

contagem vertical, ou seja, no eixo-y (*count_y*), e lembrando que o valor do pulso de sincronismo vertical não é mais 96 e sim 2, correspondendo à *SAIDA_VS* que será configurada no FPGA.

Por fim, o terceiro processo (PROCESS: *superpixel*), faz uma divisão da resolução a fim de criar uma condição de equivalência para 1 pixel na tela, antes com resolução 1x1, em uma nova resolução de 10x10 para cada pixel (ponto) na tela. Essa rotina é feita com o objetivo único de facilitar o mapeamento futuro de uma grande área de pontos da tela, que será utilizada na formatação do jogo, para guardar posições finais das peças mais adiante. Isso pode ser feito levando-se em conta que a resolução não é um ponto crítico para a aplicação desejada.

Outro detalhe importante a ser notado é que esse processo de *compactação* da resolução dos pixels na tela pode ser desconsiderado inteiramente sem prejuízo algum na formação de imagens diversas no monitor. Ao contrário, como já foi exposto antes, quanto mais pontos temos disponíveis na tela (quanto maior a resolução), maior é a qualidade da imagem gerada. Porém mais complicada poderá ser a rotina de controle (mapeamento de bits) de todos esses pontos no vídeo.

3.9 – BLOCO SAÍDA DE VÍDEO

Com pouco mais de 300 linhas de código, esse bloco é um dos mais importantes na elaboração do projeto, pois tem como saída três vetores de bits (*VGA_R*, *VGA_G* e *VGA_B*) referentes aos três sinais de cores do sistema RGB. Sua entrada é composta por todos os itens necessários para exibição do jogo na tela: as 15 linhas da área do jogo, o *score* (pontuação), o *clock* do movimento principal (*clk_mov*), e os inteiros “linha” e “coluna” referentes ao mapeamento do superpixel gerado (64x48).

Cada objeto desenhado na tela deve ter inicialmente sua posição definida dentro dos limites exibíveis no monitor, conforme a figura 3.4 da seção anterior. Assim, as quatro barras, duas horizontais e duas verticais, da grade do jogo são definidas em constantes que ditam seus limites nos eixos x e y. O mesmo deve ser feito para o *score* e para a área do jogo, que nada mais é do que outro objeto (um grande retângulo de 15 linhas x 10 colunas) dentro a grade (entre as barras constantes).

Para ser possível a comparação linha a linha, de modo a descer ou não a peça, foi realizado um mapeamento de todos os objetos em bits (zeros e uns) – por esse motivo fez-se

necessária a diminuição da resolução (em um superpixel), uma vez que cada ponto exibido no jogo precisa ser mapeado. Assim sendo, foram criados vários arrays de vetores de bits para cada objeto em um novo tipo de função definida como ROM, fazendo referência à posição de memória desses objetos, de tal forma que o bit em nível lógico “1” representa um ponto com alguma cor, definida posteriormente, ativa na tela e o nível lógico “0” é simplesmente um ponto apagado (preto).

Particularmente, o objeto “AREA1”, tem seus arrays compostos diretamente pelo valor atualizado das linhas (lin1, lin2, lin3, lin4,..., lin15) geradas no “bloco tetris” anterior, a fim de exibir o movimento de descida da peça a cada segundo.

A partir desses conceitos é possível desenhar qualquer objeto na tela, com um array formado por varias linhas, sendo cada linha composta de um vetor de bits em zeros e uns, de acordo com a imagem desejada. Sinais ao longo do corpo do código habilitam, ou não, a exibição de cores (RGB) de cada objeto e um processo final exibe o resultado atualizado dessas imagens geradas. Caso determinada região da tela não tenham seus pontos coloridos na aplicação do jogo, então “VGA_R”, “VGA_G” e “VGA_B” recebem o valor “0000”, ou seja, recebem a cor preta, a fim de poupar energia consumida pelo monitor.

3.10 – CONCLUSÕES

O protótipo do jogo TETRIS® desenvolvido nesse projeto, é capaz de trabalhar apenas com um único tipo de peça, definida (ou desenhada) no “bloco peça”. Como o propósito principal do trabalho é mostrar, no FPGA, o uso da interface de vídeo a partir de comandos em um dispositivo de entrada (teclado), essa peça única apresentou-se satisfatória no desenvolvimento da aplicação.

Contudo, a divisão do código inteiro por blocos, objetiva não somente uma melhor organização do projeto, mas também, e principalmente, a modificação futura do mesmo ao se acrescentar algum bloco adicional ou na fácil localização e correção de algum problema. Assim sendo, a geração de n peças no projeto requer apenas a inclusão de outros n blocos-peça e, obviamente, uma pequena rotina de alternância na entrada do “bloco tetris”, para receber aleatoriamente uma nova peça a cada final da contagem tempo.

Outra vantagem da divisão do código em blocos é evidenciada em códigos muito extensos, onde o erro humano ocorre mais facilmente e pode passar completamente despercebido, inclusive durante a compilação, como quando duas variáveis são trocadas, por

exemplo, em algum trecho do longo código e não ocasiona nenhum erro na análise geral do projeto, mas realiza um procedimento diferente do que é desejado ao se executar a aplicação. Então códigos com 500, 1000 ou mais linhas, que são divididos em blocos (ou trechos) de códigos menores são sempre menos suscetíveis a esse tipo de falha.

Os códigos comentados relacionados à formação de cada um desses blocos e na constituição de todo o projeto encontram-se no apêndice desse trabalho.

CAPÍTULO 4

RESULTADOS EXPERIMENTAIS

4.1 – INTRODUÇÃO

Nesse capítulo são apresentados todos os resultados experimentais da aplicação TETRIS® proposta como projeto, é feita a configuração geral dos dispositivos de *hardware* utilizados, bem como as características e dificuldades de compilação, simulação e gravação no FPGA do kit de desenvolvimento.

Por fim, são realizados testes na relação jogo/jogador, que inclui todas as experiências do jogador durante a sua interação com os sistemas gerados no jogo e que descreve a facilidade na qual o jogo pode ser manuseado, ou seja, sua jogabilidade.

4.2 – MONTAGEM E CONFIGURAÇÃO DOS BLOCOS

O uso do *software* da Altera, permite ao projetista escrever todos os códigos desejados em um editor de texto próprio do programa. Para criar um bloco a partir de um arquivo “.vhdl”, o Quatus possui uma opção chamada “Create Symbol Files for Current File” (criar arquivos de símbolo para o arquivo atual), como mostra a figura 4.1, que efetua uma rápida análise e síntese ao nível de portas lógicas do VHDL atual e gera um novo bloco na pasta principal do projeto em desenvolvimento (figura 4.2).

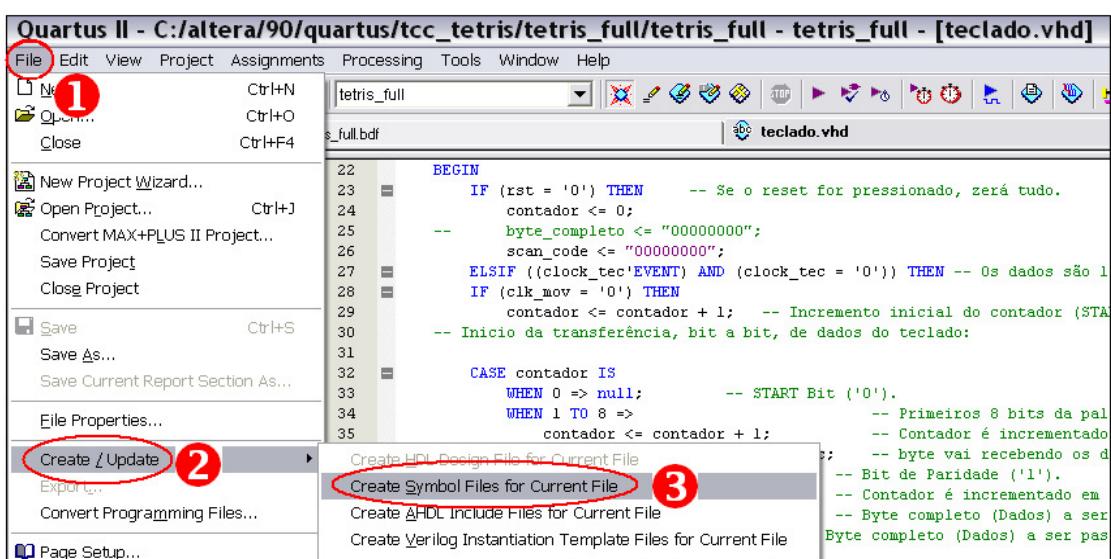


Figura 4.1 – Criação de blocos, no software QUARTUS II®.

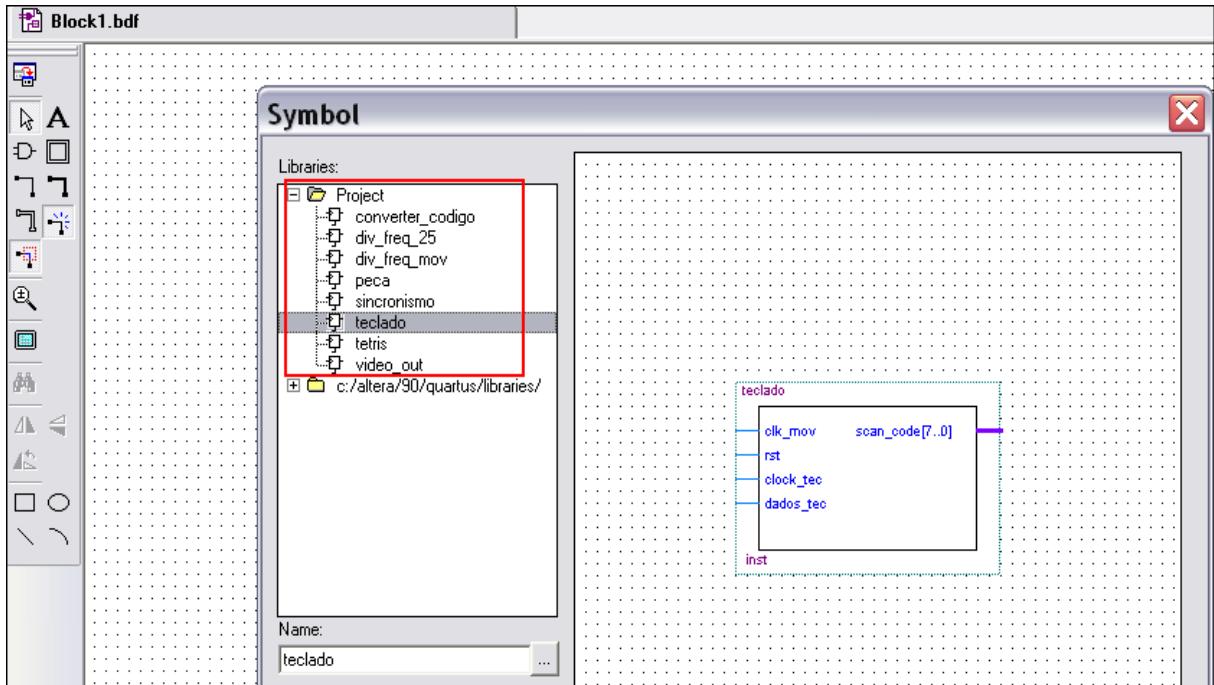


Figura 4.2 – Pasta principal do projeto contendo todos os 8 blocos da aplicação.

Conforme explanação do capítulo anterior, esses blocos criados possuem pinos de entrada e de saída que serão interligados a outros blocos e finalmente formarão o diagrama de blocos final da aplicação – conforme figura 3.2 mostrada no capítulo anterior. Após a montagem desse diagrama completo é realizada novamente a compilação (síntese geral) do projeto. A ausência de erros nessa etapa indica ao projetista que o tanto os códigos escritos em VHDL, como a montagem dos blocos ocorreu de forma satisfatória e o dispositivo agora está pronto para ser programado.

Entretanto, para se efetuar a programação física do FPGA, ainda no QUARTUS II®, é necessário identificar para o Cyclone II todos os pinos que serão utilizados na aplicação desejada: o clock de 50 MHz, os traços de cada displays de 7 segmentos, os pinos da saída VGA de vídeo e os pinos da entrada PS/2 do teclado. O restante dos pinos disponíveis no kit, como chaves, botões, e outras interfaces I/O que não serão usadas já estão por *default* desabilitados e apenas precisam ser desconsiderados nessa etapa. A tabela 4.1 abaixo apresenta a configuração de todos os pinos necessários ao projeto do jogo TETRIS®.

Finalmente, uma última compilação é responsável por gerar um arquivo chamado “tetris_full.pof” que contém uma compactação de todo o projeto funcional. Então o Cyclone II pode agora ser conectado ao computador, através do cabo USB e, a partir da opção *programmer*, é realizada a gravação do FPGA.

Tabela 4.1 – Configuração dos pinos para gravação no FPGA.

To	Location	I/O Bank	I/O Standard	General Function	Special Function
clk_50	PIN_L1	2	3.3-V LVTTL	Dedicated Clock	CLK0, LVDSCLK0p, Input
clock_tec	PIN_H15	4	3.3-V LVTTL	Column I/O	LVDS57p
dados_tec	PIN_J14	4	3.3-V LVTTL	Column I/O	LVDS55p
disp0[0]	PIN_J2	2	3.3-V LVTTL	Row I/O	LVDS16n
disp0[1]	PIN_J1	2	3.3-V LVTTL	Row I/O	LVDS16p, DPCLK0/DQS0L/CQ1L
disp0[2]	PIN_H2	2	3.3-V LVTTL	Row I/O	LVDS17n
disp0[3]	PIN_H1	2	3.3-V LVTTL	Row I/O	LVDS17p
disp0[4]	PIN_F2	2	3.3-V LVTTL	Row I/O	LVDS18n
disp0[5]	PIN_F1	2	3.3-V LVTTL	Row I/O	LVDS18p
disp0[6]	PIN_E2	2	3.3-V LVTTL	Row I/O	LVDS19n
disp1[0]	PIN_E1	2	3.3-V LVTTL	Row I/O	LVDS19p
disp1[1]	PIN_H6	2	3.3-V LVTTL	Row I/O	LVDS20n
disp1[2]	PIN_H5	2	3.3-V LVTTL	Row I/O	LVDS20p, CDPCLK0/DQS2L/CQ3L
disp1[3]	PIN_H4	2	3.3-V LVTTL	Row I/O	LVDS21n
disp1[4]	PIN_G3	2	3.3-V LVTTL	Row I/O	LVDS21p
disp1[5]	PIN_D2	2	3.3-V LVTTL	Row I/O	LVDS22n
disp1[6]	PIN_D1	2	3.3-V LVTTL	Row I/O	LVDS22p
disp2[0]	PIN_G5	2	3.3-V LVTTL	Row I/O	LVDS23n
disp2[1]	PIN_G6	2	3.3-V LVTTL	Row I/O	LVDS23p
disp2[2]	PIN_C2	2	3.3-V LVTTL	Row I/O	LVDS24n
disp2[3]	PIN_C1	2	3.3-V LVTTL	Row I/O	LVDS24p
disp2[4]	PIN_E3	2	3.3-V LVTTL	Row I/O	LVDS25p
disp2[5]	PIN_E4	2	3.3-V LVTTL	Row I/O	LVDS25n
disp2[6]	PIN_D3	2	3.3-V LVTTL	Row I/O	LVDS26p/CRC_ERROR
disp3[0]	PIN_F4	2	3.3-V LVTTL	Row I/O	VREFB2N0
disp3[1]	PIN_D5	2	3.3-V LVTTL	Row I/O	PLL3_OUTp
disp3[2]	PIN_D6	2	3.3-V LVTTL	Row I/O	PLL3_OUTn
disp3[3]	PIN_J4	2	3.3-V LVTTL	Row I/O	
disp3[4]	PIN_L8	2	3.3-V LVTTL	Row I/O	
disp3[5]	PIN_F3	2	3.3-V LVTTL	Row I/O	
disp3[6]	PIN_D4	2	3.3-V LVTTL	Row I/O	LVDS26n/CLKUSR
RESET	PIN_R22	6	3.3-V LVTTL	Row I/O	LVDS81p
SAIDA_HS	PIN_A11	3	3.3-V LVTTL	Column I/O	LVDS44p, DPCLK10/DQS5T/CQ5T#
SAIDA_VS	PIN_B11	3	3.3-V LVTTL	Column I/O	LVDS44n
VGA_B[0]	PIN_A9	3	3.3-V LVTTL	Column I/O	LVDS39p
VGA_B[1]	PIN_D11	3	3.3-V LVTTL	Column I/O	LVDS43p
VGA_B[2]	PIN_A10	3	3.3-V LVTTL	Column I/O	LVDS41p
VGA_B[3]	PIN_B10	3	3.3-V LVTTL	Column I/O	LVDS41n
VGA_G[0]	PIN_B8	3	3.3-V LVTTL	Column I/O	LVDS36n
VGA_G[1]	PIN_C10	3	3.3-V LVTTL	Column I/O	VREFB3N0
VGA_G[2]	PIN_B9	3	3.3-V LVTTL	Column I/O	LVDS39n
VGA_G[3]	PIN_A8	3	3.3-V LVTTL	Column I/O	LVDS36p, DPCLK11/DQS3T/CQ3T#
VGA_R[0]	PIN_D9	3	3.3-V LVTTL	Column I/O	LVDS37p
VGA_R[1]	PIN_C9	3	3.3-V LVTTL	Column I/O	LVDS33p
VGA_R[2]	PIN_A7	3	3.3-V LVTTL	Column I/O	LVDS35p
VGA_R[3]	PIN_B7	3	3.3-V LVTTL	Column I/O	LVDS35n

4.3 – TESTES DO PROGRAMA E JOGABILIDADE

A geração de imagens na tela do monitor foi tratada como um dos principais propósitos para o desenvolvimento desse projeto TETRIS®. Juntamente ao vídeo também se fez necessário a presença de comandos que possibilitem ao usuário, ou jogador, interagir de maneira satisfatória com o jogo, de modo a obter sempre uma resposta positiva na tela às suas entradas dadas através do teclado. Dessa forma os recursos visuais, ainda que limitados a imagens com uma qualidade não muito aprimorada, são essenciais nessa relação jogo-jogador.

No protótipo criado do jogo as peças podem ser empilhadas facilmente e ao se formar uma linha completa, esta é apagada a fim de somar uma pontuação para o jogador, conforme ditam as regras do jogo TETRIS® original. O projeto criado permite ainda ao usuário controlar movimentos em único, e principal, objeto: a peça. Ao pressionar determinadas teclas no teclado PS/2, é possível movê-la para a esquerda (tecla 1), para a direita (tecla 2) e ainda rotacioná-la (tecla espaço), de acordo com a vontade do jogador. A figura 4.3 mostra alguns testes no projeto, exibidos na tela do monitor.

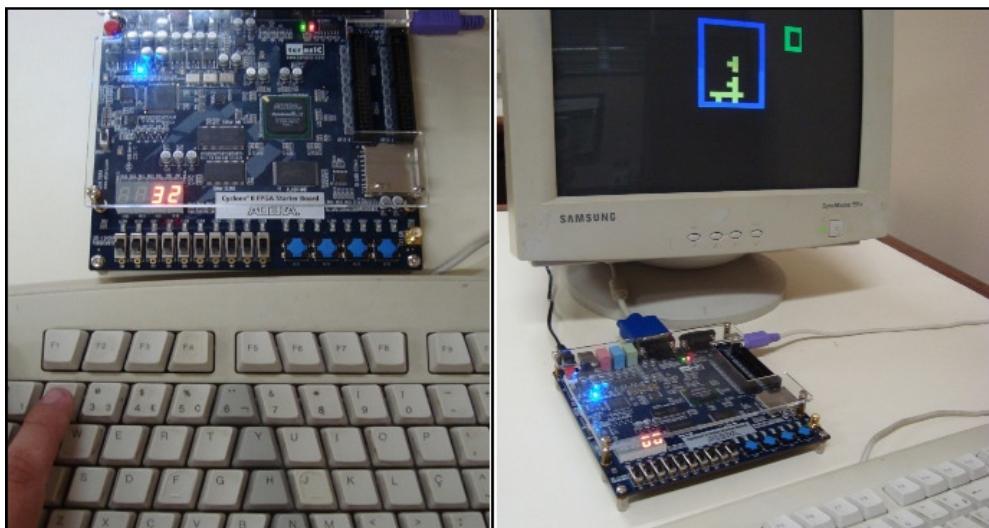


Figura 4.3 – Imagem do jogo TETRIS® exibido na tela no monitor.

Entretanto, os movimentos na tela apresentaram-se com um pequeno atraso, de aproximadamente 1 segundo, após a tecla ser pressionada. Isso ocorre devido ao tempo de atualização da peça que acontece meio ciclo antes (na borda de subida do “clk_mov”) em comparação ao avanço de uma posição, ou uma linha, da peça (na borda de descida do “clk_mov”). A atualização da nova posição da peça não pode ocorrer exatamente no mesmo momento de descida da mesma, pois é preciso primeiro exibi-la na tela, para depois modificá-la. Se o “bloco-peça” não ocorrer um instante antes do “bloco-tetris”, este irá varrer todas as

suas rotinas antes mesmo de existir uma peça no jogo. Em outras palavras, na prática, é exibida dentro da área do jogo (grade azul) uma tela em vazia (preta) durante 15 segundos (rotina das 15 linhas iniciais) antes de aparecer a primeira peça.

Por fim, quanto à pontuação ao se obter uma linha horizontal cheia, foi definido um valor de *score* máximo igual a 5, pois este é apenas demonstrativo de forma a exemplificar o funcionamento do jogo. Quando uma peça é empilhada em uma posição de modo a completar uma linha horizontal, imediatamente a pontuação é incrementada e esta linha é apagada, consequentemente fazendo descer todas as linhas logo acima. Esse procedimento é testado na figura 4.4.

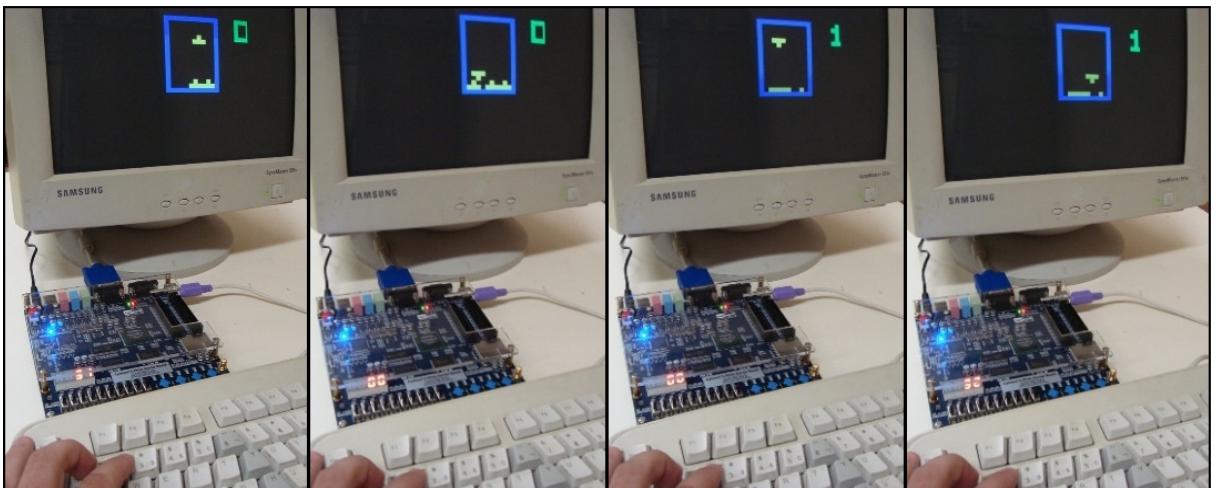


Figura 4.4 – Imagem de teste das características do jogo, limpar linha e pontuação.

4.4 – CONCLUSÕES

A jogabilidade do jogo, no geral, apresentou-se um pouco prejudicada devido ao pequeno atraso de resposta da peça à tecla pressionada, conforme mencionado anteriormente. A aparência do jogo, às vistas de um usuário comum aparenta-se bem limitada. Entretanto o objetivo principal da proposta no jogo é alcançado: mover peça para esquerda e direita, rotacionar e por fim empilhar as peças de modo a somar pontos na formação de uma linha completa.

Mensagens de vitória, ou outras mensagens como *game over* (fim de jogo), bem como outros recursos visuais adicionais (sejam estes desenhos ou textos) por todo o espaço da tela, podem ser inclusos sem maiores dificuldades, bastando apenas mais algumas linhas de código VHDL ou outros blocos integrados ao sistema. Dessa forma, o projetista tem todo o suporte para usar sua criatividade e gerar os mais diversos efeitos gráficos desejados para seu jogo.

CONCLUSÃO

Os protocolos da entrada de dados pelo teclado e da comunicação na saída de vídeo, podem ser facilmente configurados no Cyclone II. Para o teclado, que já tem seu próprio clock interno, apenas é preciso tratar os três códigos (33 bits) gerados no toque de uma tecla conforme desejado. Na geração da imagem no monitor, um cuidado especial faz-se necessário apenas na criação dos sinais de sincronismo horizontais e verticais da tela. Para “pintar” a tela resta então somente definir as posições de cada objeto que se quer desenhar com sua respectiva cor de escolha.

É possível também o uso de outros protocolos no Cyclone II, como a comunicação serial, saída e entrada de conectores de áudio, *mouse*, conectores de cartão de memória e USB, etc. Assim sendo o kit de desenvolvimento possibilita ao usuário uma grande facilidade de uso em aplicações diversas, com uma grande variedade de periféricos.

Quanto à proposta do jogo TETRIS® utilizada como exemplo para demonstrar a manipulação de dados de entrada em um monitor de vídeo, a grande dificuldade inicial foi o mapeamento de memória dos *pixels* da tela. Esse mapeamento é essencial para o funcionamento do jogo, uma vez é preciso comparar bit a bit a peça com as linhas e colunas da área do jogo, a fim de permitir movê-la, ou não, para os lados e para baixo.

De modo a diminuir, consideravelmente, o tamanho do código, foi utilizada uma escala de 10x10 para um único ponto, o que ocasionou uma perda considerável na qualidade das imagens geradas. Entretanto, como a resolução não representa um problema crítico para esta aplicação simplificada, ela pôde então ser fixada em 64 x 48, sem maiores problemas para projeto.

Os resultados experimentais forma obtidos em testes de gravação diretos no FPGA, uma vez que as simulações no QUARTUS II®, em formas-de-onda, não eram satisfatórias para o propósito desejado. Isso porque a resposta principal do projeto é dada diretamente na tela do monitor ao formar os objetos, as imagens requeridas e as combinações necessárias ao empilhamento das peças e de toda a rotina do TETRIS®. Assim, a característica da extrema facilidade de regravação do dispositivo, a partir de uma simples compilação e síntese do código, possibilitou todos os testes necessários ao êxito da aplicação.

A partir de toda a base dos blocos e dos códigos montada para o tetris nesse projeto, é possível implementar outros aperfeiçoamentos, como sons ao cair uma peça ou ao vencer o jogo, outras peças de formatos variados e de cores diversas, a mudança de fases (aumentando

a velocidade de queda das peças) e diversas outras animações à critério da imaginação do projetista. Do mesmo modo, inúmeros outros jogos podem ser desenvolvidos e implementados fazendo-se uso dessa ferramenta de desenvolvimento apresentada.

Dante do exposto, pode-se afirmar que o objetivo do trabalho foi alcançado, ou seja, é possível, a partir do uso puramente de linguagem de programação, gerar interfaces de interação homem-máquina no dispositivo lógico programável estudado – o FPGA do kit de desenvolvimento Cyclone II. Fica comprovado, assim, o poder e a versatilidade de uso que essa ferramenta pode proporcionar ao projetista, baseado em uma descrição de *hardware* rápida e bem intuitiva.

REFERÊNCIAS BIBLIOGRÁFICAS

- [1] Soares, M. J., “*Dispositivos Lógicos Programáveis*”. Saber Eletrônica [Tecnologia - Informática - Automação], Vol.40, Nº 375, Abr. 2004, pp. 34-38.
- [2] *Interface do Utilizador*, na URL: http://pt.wikipedia.org/wiki/Interface_do_utilizador, acesso em setembro de 2010.
- [3] Costa, C., “*Projetando Controladores Digitais com FPGA*”, 1ª Ed. São Paulo: Novatec Editora, 2006.
- [4] Altera, 2006. *Cyclone II FPGA Starter Development Board – Reference Manual*, na URL: http://www.altera.com/literature/ug/ug_cii_starter_board.pdf, acesso em agosto de 2010.
- [5] Vranesic, Z., Brown, S., “*Fundamentals of Digital Logic with VHDL Design*”, 2ª Ed. United States, New York: McGraw-Hill, 2005.
- [6] Wakerly, John F., “*Digital Design: Principles & Practices*”, 3ª Ed. New Jersey: Prentice Hall, Inc. 2000.
- [7] Floyd, Thomaz L., “*Digital Fundamentals*”, 7ª Ed. New Jersey: Prentice Hall, 2000.
- [8] Soares, M. J., “*Dispositivos Lógicos Programáveis – parte 2*”. Saber Eletrônica [Tecnologia - Informática - Automação], Vol.40, Nº 376, Mai. 2004, pp. 22-25.
- [9] Zelenovsky, R., Mendonça, A., “*Eletrônica Digital: Curso prático e exercícios*”, 1ª Ed. Rio de Janeiro: Mz Editora Ltda, 2004.
- [10] Zelenovsky, R., Mendonça, A., “*PC: um Guia Prático de Hardware e Interfaceamento*”, 3ª Ed. Rio de Janeiro: Mz Editora Ltda, 2002.
- [11] Chu, Pong P., “*FPGA Prototyping by VHDL examples: Xilinx Spartan-3 Version*”, 3ª Ed. New Jersey: John Wiley & Sons, Inc – Wiley Interscience, 2008.

- [12] Quartus II Subscription Edition Software. *Descrição geral do software*, na URL:
<http://www.altera.com/products/software/quartus-ii/subscription-edition/qts-se-index.html>,
acesso em agosto de 2010.
- [13] Santos, J. P. S. “*Sistema Embocado para Rastreamento de Imagens*”, T.C.C. da
Universidade de Brasília – UnB, 2009, 63p.
- [14] Ashenden, Peter J., “*The VHDL Cookbook*”, 1^a Ed. Austrália. University of Adelaide –
Dept. Computer Science, 1990.
- [15] D’Amore, R., “*VHDL: Descrição e Síntese de Circuitos Digitais*”, 3^a Ed. Rio de Janeiro:
Mz Editora Ltda, 2002.
- [16] Perry, Douglas L., “*VHDL: Computer hardware description language*”, 3^a Ed,
Singapore: McGraw-Hill, 1999.
- [17] Halliday, Caroline M., “*Segredos do PC*”, 1^a Ed. São Paulo: Barkeley Editora, 1997.
- [18] Torres, G., “*Hardware: Curso Completo*”, 3^a Ed. Rio de Janeiro: Axcel Books, 1999.
- [19] Vasconcelos, L., “*Como montar, configurar e expander seu PC*”, 7^a Ed. São Paulo:
Makron Books Ltda, 2001.
- [20] Souza, Vitor A., “*Comunicação com teclado PS2*”. Saber Eletrônica [Tecnologia -
Informática - Automação], Vol.43, Nº 416, Set 2007, pp. 62-65.
- [21] Einsfeldt, A., “*Uso de lógica programável num gerador de padrões para vídeo VGA*”.
Saber Eletrônica [Tecnologia - Informática - Automação], Vol.41, Nº 397, Fev. 2006, pp.
16-22.

APÊNDICE A

(CÓDIGO DO PROGRAMA EM VHDL)

A.1 – BLOCO TECLADO

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.STD.TEXTIO.ALL;

ENTITY teclado IS
    PORT( clk_mov : IN BIT;          -- Clock referente a duração de 1 segundo no jogo
          rst : IN STD_LOGIC;        -- Sinal de Reset
          clock_tec : IN STD_LOGIC;   -- Sinal de Clock do Teclado
          dados_tec : IN STD_LOGIC;   -- Sinal de Dados do Teclado
          scan_code: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)); -- Valor Lido
END teclado;

ARCHITECTURE leitura OF teclado IS

SIGNAL contador : NATURAL RANGE 0 TO 32 := 0;      -- Numero de bits lidos na AÇÃO COMPLETA
SIGNAL byte : STD_LOGIC_VECTOR (7 DOWNTO 0);       -- ScanCode enviado pelo teclado bit a bit

BEGIN
    ler_scancodes: PROCESS (clock_tec, rst) -- Processo que lê scancode do teclado pressionado
    BEGIN
        IF (rst = '0') THEN      -- Se o reset for pressionado, zerá tudo.
            contador <= 0;
            scan_code <= "00000000";
        ELSIF ((clock_tec'EVENT) AND (clock_tec = '0')) THEN -- Dados são lidos na borda de descida.
            IF (clk_mov = '0') THEN
                contador <= contador + 1; -- Incremento inicial do contador (START Bit)
-- Início da transferência, bit a bit, de dados do teclado:
                CASE contador IS
                    WHEN 0 => null;           -- START Bit ('0').
                    WHEN 1 TO 8 =>           -- Primeiros 8 bits da palavra de dados
                        contador <= contador + 1;      -- Contador é incrementado em 1, e
                        byte(contador - 1) <= dados_tec; -- byte vai recebendo os dados do teclado.
                    WHEN 9 =>               -- Bit de Paridade ('1').
                        contador <= contador + 1;      -- Contador é incrementado em 1, e
                        scan_code <= byte;           -- Byte completo (Dados) a ser passado.
                    WHEN 10 =>              -- Se STOP Bit é igual a '1'
                        IF (dados_tec = '1') THEN
                            contador <= contador + 1;      -- Contador é incrementado em 1.
                        ELSE null;                  -- Caso contrário não é o bit de parada
                        END IF;
                    WHEN 11 =>              -- Fim da transmissão da ação de PRESSIONAR TECLA
                        IF (dados_tec = '0') THEN
                            contador <= contador + 1;      -- Teclado deve gerar um pulso igual a '0'(ACK)
                        ELSE null;                  -- Contador incrementa 1.
                        END IF;
                    WHEN 12 TO 20 =>         -- Intervalo de pulsos para uma nova palavra.
                        contador <= contador + 1;      -- Apenas é incrementado o contador.
                    WHEN 21 =>              -- Fim do intervalo de 10 bits.
                        IF (dados_tec = '1') THEN
                            contador <= contador + 1;      -- START Bit, da ação de SOLTAR TECLA
                        ELSE null;                  -- bits gerados na ação de SOLTAR TECLA (não interessa)
                        END IF;
                    WHEN 22 =>              -- STOP Bit, da ação de SOLTAR TECLA
                        IF (dados_tec = '0') THEN
                            contador <= 0;           -- Reinicia o contador.
                        ELSE null;
                        END IF;
                    END CASE;
                END IF;
            END IF;
        END PROCESS ler_scancodes;
    END leitura;

```

A.2 – BLOCO CONVERTER_CÓDIGO

```

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

ENTITY converter_codigo IS
    PORT( scan_code : IN STD_LOGIC_VECTOR (7 DOWNTO 0); -- ScanCode gerado pelo teclado
          disp0 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);           -- Display MENOS significativo
          disp1 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);           -- Display MAIS significativo
          disp2, disp3 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0));  -- Displays não usados (apagados)
END converter_codigo;

ARCHITECTURE conversao OF converter_codigo IS

    -- TABELA DE CONVERSÃO (PARA O DISPLAY DE 7 SEGMENTOS)
    -- ('0'= acesso, e '1'= apagado)           gfedcba
    CONSTANT zero   : STD_LOGIC_VECTOR(6 DOWNTO 0) := "1000000";      -- Numero 0 no display
    CONSTANT un     : STD_LOGIC_VECTOR(6 DOWNTO 0) := "1111001";      -- Numero 1 no display
    CONSTANT dois   : STD_LOGIC_VECTOR(6 DOWNTO 0) := "0100100";      -- Numero 2 no display
    CONSTANT tres   : STD_LOGIC_VECTOR(6 DOWNTO 0) := "0110000";      -- Numero 3 no display
    CONSTANT quatro : STD_LOGIC_VECTOR(6 DOWNTO 0) := "0011001";      -- Numero 4 no display
    CONSTANT cinco  : STD_LOGIC_VECTOR(6 DOWNTO 0) := "0010010";      -- Numero 5 no display
    CONSTANT seis   : STD_LOGIC_VECTOR(6 DOWNTO 0) := "0000010";      -- Numero 6 no display
    CONSTANT sete   : STD_LOGIC_VECTOR(6 DOWNTO 0) := "1111000";      -- Numero 7 no display
    CONSTANT oito   : STD_LOGIC_VECTOR(6 DOWNTO 0) := "0000000";      -- Numero 8 no display
    CONSTANT nove   : STD_LOGIC_VECTOR(6 DOWNTO 0) := "0010000";      -- Numero 9 no display
    CONSTANT A       : STD_LOGIC_VECTOR(6 DOWNTO 0) := "0001000";      -- Letra A no display
    CONSTANT B       : STD_LOGIC_VECTOR(6 DOWNTO 0) := "0000011";      -- Letra B no display
    CONSTANT C       : STD_LOGIC_VECTOR(6 DOWNTO 0) := "1000110";      -- Letra C no display
    CONSTANT D       : STD_LOGIC_VECTOR(6 DOWNTO 0) := "0100001";      -- Letra D no display
    CONSTANT E       : STD_LOGIC_VECTOR(6 DOWNTO 0) := "0000110";      -- Letra E no display
    CONSTANT F       : STD_LOGIC_VECTOR(6 DOWNTO 0) := "0000110";      -- Letra F no display
    CONSTANT apagado: STD_LOGIC_VECTOR(6 DOWNTO 0) := "1111111";      -- DISPLAY APAGADO
    -- CONSTANT vazio  : STD_LOGIC_VECTOR(6 DOWNTO 0) := "1111110";      -- DISPLAY COM TRAÇO(vazio)

    SIGNAL ascii : STD_LOGIC_VECTOR (7 DOWNTO 0); -- Código convertido em ASCII

BEGIN
    conv_scan_code: PROCESS (scan_code)           -- Esse processo apenas converte SCAN_CODE em ASCII
    BEGIN
        CASE scan_code IS                      -- para simples verificação e comparação de tabelas.
            -- Dígitos Numéricos:
            WHEN x"45" => ascii <= x"30";      -- número 0
            WHEN x"16" => ascii <= x"31";      -- número 1
            WHEN x"1E" => ascii <= x"32";      -- número 2
            WHEN x"26" => ascii <= x"33";      -- número 3
            WHEN x"25" => ascii <= x"34";      -- número 4
            WHEN x"2E" => ascii <= x"35";      -- número 5
            WHEN x"36" => ascii <= x"36";      -- número 6
            WHEN x"3D" => ascii <= x"37";      -- número 7
            WHEN x"3E" => ascii <= x"38";      -- número 8
            WHEN x"46" => ascii <= x"39";      -- número 9
            -- Caracteres:
            WHEN x"1C" => ascii <= x"41";      -- Letra A
            WHEN x"32" => ascii <= x"42";      -- Letra B
            WHEN x"21" => ascii <= x"43";      -- Letra C
            WHEN x"23" => ascii <= x"44";      -- Letra D
            WHEN x"24" => ascii <= x"45";      -- Letra E
            WHEN x"2B" => ascii <= x"46";      -- Letra F
            WHEN x"34" => ascii <= x"47";      -- Letra G
            WHEN x"33" => ascii <= x"48";      -- Letra H
            WHEN x"43" => ascii <= x"49";      -- Letra I
            WHEN x"3B" => ascii <= x"4A";      -- Letra J
            WHEN x"42" => ascii <= x"4B";      -- Letra K
            WHEN x"4B" => ascii <= x"4C";      -- Letra L
            WHEN x"3A" => ascii <= x"4D";      -- Letra M
            WHEN x"31" => ascii <= x"4E";      -- Letra N
            WHEN x"44" => ascii <= x"4F";      -- Letra O
            WHEN x"4D" => ascii <= x"50";      -- Letra P
            WHEN x"15" => ascii <= x"51";      -- Letra Q
            WHEN x"2D" => ascii <= x"52";      -- Letra R
            WHEN x"1B" => ascii <= x"53";      -- Letra S
        END CASE;
    END;

```

```

WHEN x"2C" => ascii <= x"54";    -- Letra T
WHEN x"3C" => ascii <= x"55";    -- Letra U
WHEN x"2A" => ascii <= x"56";    -- Letra V
WHEN x"1D" => ascii <= x"57";    -- Letra W
WHEN x"22" => ascii <= x"58";    -- Letra X
WHEN x"35" => ascii <= x"59";    -- Letra Y
WHEN x"1A" => ascii <= x"5A";    -- Letra Z
WHEN x"29" => ascii <= x"20";    -- Tecla SPACE
WHEN x"5A" => ascii <= x"0D";    -- Tecla ENTER
WHEN x"75" => ascii <= x"A1";    -- Seta para CIMA (ASCII adotado)
WHEN x"72" => ascii <= x"A2";    -- Seta para BAIXO (ASCII adotado)
WHEN x"6B" => ascii <= x"A3";    -- Seta para ESQUERDA (ASCII adotado)
WHEN x"74" => ascii <= x"A4";    -- Seta para DIREITA (ASCII adotado)
WHEN x"00" => ascii <= x"00";    -- VALOR INICIAL EM ZERO (ASCII adotado)
WHEN OTHERS => null;
END CASE;
END PROCESS conv_scan_code;

display_LSB : BLOCK -- Bloco da saida, em hexadecimal, no display MENOS significativo (LSB)
BEGIN
  disp0 <= zero WHEN (ascii(3 DOWNTO 0) = x"0") ELSE
    um WHEN (ascii(3 DOWNTO 0) = x"1") ELSE
    dois WHEN (ascii(3 DOWNTO 0) = x"2") ELSE
    tres WHEN (ascii(3 DOWNTO 0) = x"3") ELSE
    quatro WHEN (ascii(3 DOWNTO 0) = x"4") ELSE
    cinco WHEN (ascii(3 DOWNTO 0) = x"5") ELSE
    seis WHEN (ascii(3 DOWNTO 0) = x"6") ELSE
    sete WHEN (ascii(3 DOWNTO 0) = x"7") ELSE
    oito WHEN (ascii(3 DOWNTO 0) = x"8") ELSE
    nove WHEN (ascii(3 DOWNTO 0) = x"9") ELSE
    A WHEN (ascii(3 DOWNTO 0) = x"A") ELSE
    B WHEN (ascii(3 DOWNTO 0) = x"B") ELSE
    C WHEN (ascii(3 DOWNTO 0) = x"C") ELSE
    D WHEN (ascii(3 DOWNTO 0) = x"D") ELSE
    E WHEN (ascii(3 DOWNTO 0) = x"E") ELSE
    F;
END BLOCK display_LSB;

display_MSB : BLOCK -- Bloco da saida, em hexadecimal, no display MAIS significativo (MSB)
BEGIN
  displ <= zero WHEN (ascii(7 DOWNTO 4) = x"0") ELSE
    um WHEN (ascii(7 DOWNTO 4) = x"1") ELSE
    dois WHEN (ascii(7 DOWNTO 4) = x"2") ELSE
    tres WHEN (ascii(7 DOWNTO 4) = x"3") ELSE
    quatro WHEN (ascii(7 DOWNTO 4) = x"4") ELSE
    cinco WHEN (ascii(7 DOWNTO 4) = x"5") ELSE
    seis WHEN (ascii(7 DOWNTO 4) = x"6") ELSE
    sete WHEN (ascii(7 DOWNTO 4) = x"7") ELSE
    oito WHEN (ascii(7 DOWNTO 4) = x"8") ELSE
    nove WHEN (ascii(7 DOWNTO 4) = x"9") ELSE
    A WHEN (ascii(7 DOWNTO 4) = x"A") ELSE
    B WHEN (ascii(7 DOWNTO 4) = x"B") ELSE
    C WHEN (ascii(7 DOWNTO 4) = x"C") ELSE
    D WHEN (ascii(7 DOWNTO 4) = x"D") ELSE
    E WHEN (ascii(7 DOWNTO 4) = x"E") ELSE
    F;
END BLOCK display_MSB;

disp2 <= apagado;   -- Display não utilizado (apagado)
disp3 <= apagado;   -- Display não utilizado (apagado)

END conversao;

```

A.3 – BLOCO DIV_FREQ_25

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY div_freq_25 IS
    GENERIC (divisor : INTEGER := 2);    -- Divisor é uma inteiro igual a 2
    PORT (clk_50 : IN STD_LOGIC;
          clk_25 : OUT STD_LOGIC);
END div_freq_25;

ARCHITECTURE divisao OF div_freq_25 IS
BEGIN
    dividir : PROCESS (clk_50)
    VARIABLE aux : INTEGER RANGE 0 TO divisor := 0;
    BEGIN
        IF(aux = divisor) THEN    -- Se aux for igual a 2, então
            aux := 0;             -- zera aux;
            clk_25 <= '1';        -- e é gerado um pulso em clk_25
        ELSIF( (clk_50'EVENT) AND (clk_50 = '1') ) THEN    -- Caso contrário, para cada pulso
            aux := aux + 1;       -- de clk_50, aux é incrementado
        IF (aux = divisor/2) THEN    -- Se aux = 1 (a metade)
            clk_25 <= '0';        -- clk_25 recebe zero.
        END IF;
        END IF;
    END PROCESS dividir;
END divisao;

```

A.4 – BLOCO DIV_FREQ_MOV

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY div_freq_mov IS
    GENERIC (divisor : INTEGER := 50000000);
    PORT (clk_50 : IN STD_LOGIC;
          clk_mov : OUT STD_LOGIC);
END div_freq_mov;

ARCHITECTURE divisao OF div_freq_mov IS
BEGIN
    dividir : PROCESS (clk_50)
    VARIABLE aux : INTEGER RANGE 0 TO divisor := 0;
    BEGIN
        IF (aux = divisor) THEN
            aux := 0;
            clk_mov <= '1';
        ELSIF ((clk_50'EVENT) AND (clk_50 = '1')) THEN
            aux := aux + 1;
            clk_mov <= '0';
        END IF;
    END PROCESS dividir;
END divisao;

```

A.5 – BLOCO SÍNCRONISMO

```

ENTITY sincronismo IS
  GENERIC ( H_count_max : INTEGER := 799;      -- São 800 PIXELS no total por cada LINHA
            V_count_max : INTEGER := 524);
  PORT   ( clk_25 : IN BIT;
            coluna, linha : OUT INTEGER RANGE 0 TO 128;
            SAIDA_HS, SAIDA_VS : OUT BIT);
END sincronismo;

ARCHITECTURE calculo OF sincronismo IS
  SIGNAL clk_vs : BIT;    -- Sinal de saída gerado no fim de uma linha horizontal
  SIGNAL count_x : INTEGER RANGE H_count_max DOWNTO 0 := 0;
  SIGNAL count_y : INTEGER RANGE V_count_max DOWNTO 0 := 0;
  SIGNAL COL_count, LIN_count : INTEGER;
  SIGNAL COL_addr, LIN_addr : INTEGER;
BEGIN
----- SÍNCRONISMO HORIZONTAL -----
  sinal_Horiz : PROCESS (clk_25)
  BEGIN
    IF (count_x = H_count_max - 1) THEN
      count_x <= 0;
      SAIDA_HS <= '0';
      clk_vs <= '0';
    ELSIF ((clk_25'EVENT) AND (clk_25 = '1')) THEN
      count_x <= count_x + 1;
      IF (count_x = 96) THEN    -- Pulso de sincronismo Horizontal (96)
        SAIDA_HS <= '1';
        clk_vs <= '1';
      END IF;
    END IF;
  END PROCESS sinal_Horiz;
----- SÍNCRONISMO VERTICAL -----
  sinal_Vert : PROCESS (clk_vs)
  BEGIN
    IF (count_y = V_count_max) THEN
      count_y <= 0;
      SAIDA_VS <= '0';
    ELSIF ((clk_vs'EVENT) AND (clk_vs = '0')) THEN    -- Os pulsos de sincronismos ocorrem
      count_y <= count_y + 1;                          -- em nível lógico baixo.
      IF (count_y = 2) THEN    -- Pulso de sincronismo Vertical (2)
        SAIDA_VS <= '1';
      END IF;
    END IF;
  END PROCESS sinal_Vert;
-->     DEFININDO SUPERPIXEL 10x10 DE 640x480 PARA MAPEAR MEMÓRIA EM 64x48
----- superpixel : PROCESS (clk_25)
  BEGIN
    IF ((clk_25'EVENT) AND (clk_25 = '1')) THEN
      IF (count_x <= 639) THEN      -- colunas (pixels) ativas no display em uma linha (até 640)
        IF (COL_count <= 10) THEN
          COL_count <= COL_count + 1;
        ELSE
          COL_count <= 0;
          COL_addr <= COL_addr + 1;
        END IF;
      ELSE                      -- Fim da contagem de coluna
        COL_count <= 0;
        COL_addr <= 0;
      END IF;
      IF (count_x = 641) THEN    -- Início da próxima linha
        LIN_count <= LIN_count + 1;
        IF (LIN_count = 10) THEN
          LIN_count <= 0;
          LIN_addr <= LIN_addr + 1;
        END IF;
      END IF;
    END IF;
  END PROCESS superpixel;

```

```

END IF; -- Fim da contagem de linha
IF (count_y >= 479) THEN -- o valor máx de linhas por tela é 480
    LIN_count <= 0;
    LIN_addr <= 0;
END IF;
END IF;
END PROCESS superpixel;

-----
-- ENTÃO É PASSADO A CONTAGEM DOS PIXELS HORIZ E VERT,
-- COMO SAÍDA PARA O BLOCO DE CONFIGURAÇÃO DE CORES DO VÍDEO
-----

coluna <= COL_addr;
linha <= LIN_addr;
END calculo;

```

A.6 – BLOCO PEÇA

```

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

ENTITY peca IS
    PORT ( clk_mov : IN BIT;
           scan_code : IN STD_LOGIC_VECTOR (7 DOWNTO 0); -- ScanCode gerado pelo teclado
           lp1, lp2, lp3 : OUT BIT_VECTOR(0 TO 9)); -- Linhas de formato da peça
END peca;

ARCHITECTURE formato OF peca IS
    SIGNAL aux_lp1 : BIT_VECTOR(0 TO 9) := "0000000000";
    SIGNAL aux_lp2 : BIT_VECTOR(0 TO 9) := "0001110000";
    SIGNAL aux_lp3 : BIT_VECTOR(0 TO 9) := "0000100000";
    SIGNAL vetor_lp1, vetor_lp2, vetor_lp3 : BIT_VECTOR(0 TO 9) := "0000000000";
    SIGNAL cont_giro : NATURAL RANGE 0 TO 4 := 0; -- Contador de giro da peça
BEGIN
    deslocamento: PROCESS (clk_mov)
    BEGIN
        IF ((clk_mov'EVENT) AND (clk_mov = '1')) THEN
            IF (scan_code = x"16") THEN -- Se for pressionada a tecla 1
                aux_lp1 <= (aux_lp1 ROR 1); -- Rotina que move peça para a esquerda
                aux_lp2 <= (aux_lp2 ROR 1);
                aux_lp3 <= (aux_lp3 ROR 1);
            ELSIF (scan_code = x"1E") THEN -- Se for pressionada a tecla 2
                aux_lp1 <= (aux_lp1 ROL 1); -- Rotina que move peça para a direita
                aux_lp2 <= (aux_lp2 ROL 1);
                aux_lp3 <= (aux_lp3 ROL 1);
            ELSIF (scan_code = x"29") THEN -- Se for pressionada a tecla SPACE
                cont_giro <= cont_giro + 1; -- Rotina de giro da peça (4 posições)
                IF (cont_giro = 1) THEN
                    aux_lp1 <= "0000100000";
                    aux_lp2 <= "0000110000";
                    aux_lp3 <= "0000100000";
                ELSIF (cont_giro = 2) THEN
                    aux_lp1 <= "0000000000";
                    aux_lp2 <= "0000100000";
                    aux_lp3 <= "0001110000";
                ELSIF (cont_giro = 3) THEN
                    aux_lp1 <= "0000010000";
                    aux_lp2 <= "0000110000";
                    aux_lp3 <= "0000010000";
                ELSIF (cont_giro = 4) THEN
                    aux_lp1 <= "0000000000";
                    aux_lp2 <= "0001110000";
                    aux_lp3 <= "0000100000";
                    cont_giro <= 0;
                END IF;
            END IF;
        END IF;
    END PROCESS;

```

```

    ELSE
        null;
    END IF;
    vetor_lp1 <= aux_lp1;
    vetor_lp2 <= aux_lp2;
    vetor_lp3 <= aux_lp3;
END IF;
END PROCESS deslocamento;

lp1 <= vetor_lp1;
lp2 <= vetor_lp2;
lp3 <= vetor_lp3;

END formato;

```

A.7 – BLOCO TETRIS

```

ENTITY tetris IS
    PORT  ( clk_mov : IN BIT;
            lp1, lp2, lp3 : IN BIT_VECTOR(0 TO 9);
            score : OUT NATURAL RANGE 0 TO 5;
            lin1, lin2, lin3, lin4, lin5, lin6, lin7, lin8, lin9, lin10,
            lin11, lin12, lin13, lin14, lin15 : OUT BIT_VECTOR(0 TO 9));
END tetris;

ARCHITECTURE jogo OF tetris IS

    SIGNAL lin1_reg, lin2_reg, lin3_reg, lin4_reg, lin5_reg, lin6_reg, lin7_reg,
           lin8_reg, lin9_reg, lin10_reg, lin11_reg, lin12_reg, lin13_reg,
           lin14_reg, lin15_reg : BIT_VECTOR(0 TO 9):= "0000000000";
    SIGNAL lin1_next, lin2_next, lin3_next, lin4_next, lin5_next, lin6_next, lin7_next,
           lin8_next, lin9_next, lin10_next, lin11_next, lin12_next, lin13_next,
           lin14_next, lin15_next : BIT_VECTOR(0 TO 9):= "0000000000";
    SIGNAL tempo : NATURAL RANGE 1 TO 16 := 1;
    SIGNAL pontos: NATURAL RANGE 0 TO 5 := 0;

BEGIN

descer_linha: PROCESS (clk_mov)
BEGIN
    IF ((clk_mov'EVENT) AND (clk_mov = '0')) THEN      -- Na borda de subida é atualizada a peça.
        IF (tempo = 1) THEN
            IF ((lp3 NAND lin1_reg) = "1111111111") THEN      -- Se a peça puder descer, então...
                lin1_next <= (lp3 OR lin1_reg);
                IF ((lp3 OR lin1_reg) = "1111111111") THEN      -- Se a LINHA 1 estiver cheia...
                    lin1_next <= "0000000000";
                    lin2_next <= lin2_reg;
                    lin3_next <= lin3_reg;
                    lin4_next <= lin4_reg;
                    lin5_next <= lin5_reg;
                    lin6_next <= lin6_reg;
                    lin7_next <= lin7_reg;
                    lin8_next <= lin8_reg;
                    lin9_next <= lin9_reg;
                    lin10_next <= lin10_reg;
                    lin11_next <= lin11_reg;
                    lin12_next <= lin12_reg;
                    lin13_next <= lin13_reg;
                    lin14_next <= lin14_reg;
                    lin15_next <= lin15_reg;
                    IF (Pontos < 5) THEN      -- Se a pontuação for menor que 5 então
                        Pontos <= Pontos + 1;      -- incrementa 1 ponto.
                    ELSIF (Pontos = 5) THEN      -- Se a pontuação chegar ao valor final (= 5) então
                        Pontos <= 0;              -- zera a pontuação e exibe mensagem de vitória.
                    END IF;
                    tempo <= 1;
                ELSE      -- Caso esta linha não esteja cheia...
                    tempo <= tempo + 1;      -- Passa para a próxima linha.
                END IF;
            END IF;
        END IF;
    END PROCESS;

```

```

ELSE    -- Caso a peça não possa mais descer...
-----
-- Atualiza registradores e reinicia contagem --
-----
lin1_reg <= lin1_next; lin2_reg <= lin2_next; lin3_reg <= lin3_next;
lin4_reg <= lin4_next; lin5_reg <= lin5_next; lin6_reg <= lin6_next;
lin7_reg <= lin7_next; lin8_reg <= lin8_next; lin9_reg <= lin9_next;
lin10_reg <= lin10_next; lin11_reg <= lin11_next; lin12_reg <= lin12_next;
lin13_reg <= lin13_next; lin14_reg <= lin14_next; lin15_reg <= lin15_next;
tempo <= 1;
END IF;

ELSIF (tempo = 2) THEN
  IF (((lp3 NAND lin2_reg) = "1111111111") AND ((lp2 NAND lin1_reg) = "1111111111")) THEN
    -- Se a peça puder descer, então...
    lin2_next <= (lp3 OR lin2_reg);
    lin1_next <= (lp2 OR lin1_reg);
    IF ((lp3 OR lin2_reg) = "1111111111") THEN    -- Se a LINHA 2 estiver cheia...
      lin1_next <= "0000000000";
      lin2_next <= (lp2 OR lin1_reg);
      lin3_next <= lin3_reg;
      lin4_next <= lin4_reg;
      lin5_next <= lin5_reg;
      lin6_next <= lin6_reg;
      lin7_next <= lin7_reg;
      lin8_next <= lin8_reg;
      lin9_next <= lin9_reg;
      lin10_next <= lin10_reg;
      lin11_next <= lin11_reg;
      lin12_next <= lin12_reg;
      lin13_next <= lin13_reg;
      lin14_next <= lin14_reg;
      lin15_next <= lin15_reg;
      IF (Pontos < 5) THEN    -- Se a pontuação for menor que 5 então
        Pontos <= Pontos + 1; -- incrementa 1 ponto.
      ELSIF (Pontos = 5) THEN    -- Se a pontuação chegar ao valor final (= 5) então
        Pontos <= 0;           -- zera a pontuação e exibe mensagem de vitória.
      END IF;
      tempo <= 1;
    ELSE    -- Caso esta linha não esteja cheia...
      tempo <= tempo + 1; -- Passa para a próxima linha.
    END IF;
  ELSE    -- Caso a peça não possa mais descer...
  -----
  -- Atualiza registradores e reinicia contagem --
  -----
  lin1_reg <= lin1_next; lin2_reg <= lin2_next; lin3_reg <= lin3_next;
  lin4_reg <= lin4_next; lin5_reg <= lin5_next; lin6_reg <= lin6_next;
  lin7_reg <= lin7_next; lin8_reg <= lin8_next; lin9_reg <= lin9_next;
  lin10_reg <= lin10_next; lin11_reg <= lin11_next; lin12_reg <= lin12_next;
  lin13_reg <= lin13_next; lin14_reg <= lin14_next; lin15_reg <= lin15_next;
  tempo <= 1;
END IF;

ELSIF (tempo = 3) THEN
  IF (((lp3 NAND lin3_reg) = "1111111111")AND ((lp2 NAND lin2_reg) = "1111111111")AND
    ((lp1 NAND lin1_reg) = "1111111111")) THEN    -- Se a peça puder descer, então...
    lin3_next <= (lp3 OR lin3_reg);
    lin2_next <= (lp2 OR lin2_reg);
    lin1_next <= (lp1 OR lin1_reg);
    IF ((lp3 OR lin3_reg) = "1111111111") THEN    -- Se a LINHA 3 estiver cheia...
      lin1_next <= "0000000000";
      lin2_next <= (lp1 OR lin1_reg);
      lin3_next <= (lp2 OR lin2_reg);
      lin4_next <= lin4_reg;
      lin5_next <= lin5_reg;
      lin6_next <= lin6_reg;
      lin7_next <= lin7_reg;
      lin8_next <= lin8_reg;
      lin9_next <= lin9_reg;
      lin10_next <= lin10_reg;
      lin11_next <= lin11_reg;
      lin12_next <= lin12_reg;
      lin13_next <= lin13_reg;
      lin14_next <= lin14_reg;
      lin15_next <= lin15_reg;
    ELSE    -- Caso esta linha não esteja cheia...
      tempo <= tempo + 1; -- Passa para a próxima linha.
    END IF;
  ELSE    -- Caso a peça não possa mais descer...
  -----
  -- Atualiza registradores e reinicia contagem --
  -----
  lin1_reg <= lin1_next; lin2_reg <= lin2_next; lin3_reg <= lin3_next;
  lin4_reg <= lin4_next; lin5_reg <= lin5_next; lin6_reg <= lin6_next;
  lin7_reg <= lin7_next; lin8_reg <= lin8_next; lin9_reg <= lin9_next;
  lin10_reg <= lin10_next; lin11_reg <= lin11_next; lin12_reg <= lin12_next;
  lin13_reg <= lin13_next; lin14_reg <= lin14_next; lin15_reg <= lin15_next;
  tempo <= 1;
END IF;

```

```

        IF (Pontos < 5) THEN      -- Se a pontuação for menor que 5 então
            Pontos <= Pontos + 1; -- incrementa 1 ponto.
        ELSIF (Pontos = 5) THEN   -- Se a pontuação chegar ao valor final (= 5) então
            Pontos <= 0;          -- zera a pontuação e exibe mensagem de vitória.
        END IF;
        Tempo <= 1;
    ELSE      -- Caso esta linha não esteja cheia...
        Tempo <= Tempo + 1; -- Passa para a próxima linha.
    END IF;
ELSE      -- Caso a peça não possa mais descer...
-----
-- Atualiza registradores e reinicia contagem --
-----
lin1_Reg <= lin1_Next; lin2_Reg <= lin2_Next; lin3_Reg <= lin3_Next;
lin4_Reg <= lin4_Next; lin5_Reg <= lin5_Next; lin6_Reg <= lin6_Next;
lin7_Reg <= lin7_Next; lin8_Reg <= lin8_Next; lin9_Reg <= lin9_Next;
lin10_Reg <= lin10_Next; lin11_Reg <= lin11_Next; lin12_Reg <= lin12_Next;
lin13_Reg <= lin13_Next; lin14_Reg <= lin14_Next; lin15_Reg <= lin15_Next;
Tempo <= 1;
END IF;

ELSIF (Tempo = 4) THEN
    IF (((lp3 NAND lin4_Reg) = "1111111111")AND ((lp2 NAND lin3_Reg) = "1111111111")AND
        ((lp1 NAND lin2_Reg) = "1111111111")) THEN      -- Se a peça puder descer, então...
        lin4_Next <= (lp3 OR lin4_Reg);
        lin3_Next <= (lp2 OR lin3_Reg);
        lin2_Next <= (lp1 OR lin2_Reg);
        lin1_Next <= lin1_Reg;
        IF ((lp3 OR lin4_Reg) = "1111111111") THEN      -- Se a LINHA 4 estiver cheia...
            lin1_Next <= "0000000000";
            lin2_Next <= lin1_Reg;
            lin3_Next <= (lp1 OR lin2_Reg);
            lin4_Next <= (lp2 OR lin3_Reg);
            lin5_Next <= lin5_Reg;
            lin6_Next <= lin6_Reg;
            lin7_Next <= lin7_Reg;
            lin8_Next <= lin8_Reg;
            lin9_Next <= lin9_Reg;
            lin10_Next <= lin10_Reg;
            lin11_Next <= lin11_Reg;
            lin12_Next <= lin12_Reg;
            lin13_Next <= lin13_Reg;
            lin14_Next <= lin14_Reg;
            lin15_Next <= lin15_Reg;
            IF (Pontos < 5) THEN      -- Se a pontuação for menor que 5 então
                Pontos <= Pontos + 1; -- incrementa 1 ponto.
            ELSIF (Pontos = 5) THEN   -- Se a pontuação chegar ao valor final (= 5) então
                Pontos <= 0;          -- zera a pontuação e exibe mensagem de vitória.
            END IF;
            Tempo <= 1;
        ELSE      -- Caso esta linha não esteja cheia...
            Tempo <= Tempo + 1; -- Passa para a próxima linha.
        END IF;
    ELSE      -- Caso a peça não possa mais descer...
    -----
    -- Atualiza registradores e reinicia contagem --
    -----
    lin1_Reg <= lin1_Next; lin2_Reg <= lin2_Next; lin3_Reg <= lin3_Next;
    lin4_Reg <= lin4_Next; lin5_Reg <= lin5_Next; lin6_Reg <= lin6_Next;
    lin7_Reg <= lin7_Next; lin8_Reg <= lin8_Next; lin9_Reg <= lin9_Next;
    lin10_Reg <= lin10_Next; lin11_Reg <= lin11_Next; lin12_Reg <= lin12_Next;
    lin13_Reg <= lin13_Next; lin14_Reg <= lin14_Next; lin15_Reg <= lin15_Next;
    Tempo <= 1;
END IF;

ELSIF (Tempo = 5) THEN
    IF (((lp3 NAND lin5_Reg) = "1111111111")AND ((lp2 NAND lin4_Reg) = "1111111111")AND
        ((lp1 NAND lin3_Reg) = "1111111111")) THEN      -- Se a peça puder descer, então...
        lin5_Next <= (lp3 OR lin5_Reg);
        lin4_Next <= (lp2 OR lin4_Reg);
        lin3_Next <= (lp1 OR lin3_Reg);

```

```

lin2_next <= lin2_reg;
lin1_next <= lin1_reg;
IF ((lp3 OR lin5_reg) = "1111111111") THEN -- Se a LINHA 5 estiver cheia...
    lin1_next <= "0000000000";
    lin2_next <= lin1_reg;
    lin3_next <= lin2_reg;
    lin4_next <= (lp1 OR lin3_reg);
    lin5_next <= (lp2 OR lin4_reg);
    lin6_next <= lin6_reg;
    lin7_next <= lin7_reg;
    lin8_next <= lin8_reg;
    lin9_next <= lin9_reg;
    lin10_next <= lin10_reg;
    lin11_next <= lin11_reg;
    lin12_next <= lin12_reg;
    lin13_next <= lin13_reg;
    lin14_next <= lin14_reg;
    lin15_next <= lin15_reg;
    IF (Pontos < 5) THEN -- Se a pontuação for menor que 5 então
        Pontos <= Pontos + 1; -- incrementa 1 ponto.
    ELSIF (Pontos = 5) THEN -- Se a pontuação chegar ao valor final (= 5) então
        Pontos <= 0; -- zera a pontuação e exibe mensagem de vitória.
    END IF;
    tempo <= 1;
ELSE -- Caso esta linha não esteja cheia...
    tempo <= tempo + 1; -- Passa para a próxima linha.
END IF;
ELSE -- Caso a peça não possa mais descer...
-----
-- Atualiza registradores e reinicia contagem --
-----
lin1_reg <= lin1_next; lin2_reg <= lin2_next; lin3_reg <= lin3_next;
lin4_reg <= lin4_next; lin5_reg <= lin5_next; lin6_reg <= lin6_next;
lin7_reg <= lin7_next; lin8_reg <= lin8_next; lin9_reg <= lin9_next;
lin10_reg <= lin10_next; lin11_reg <= lin11_next; lin12_reg <= lin12_next;
lin13_reg <= lin13_next; lin14_reg <= lin14_next; lin15_reg <= lin15_next;
tempo <= 1;
END IF;

ELSIF (tempo = 6) THEN
    IF (((lp3 NAND lin6_reg) = "1111111111")AND ((lp2 NAND lin5_reg) = "1111111111")AND
        ((lp1 NAND lin4_reg) = "1111111111")) THEN -- Se a peça puder descer, então...
        lin6_next <= (lp3 OR lin6_reg);
        lin5_next <= (lp2 OR lin5_reg);
        lin4_next <= (lp1 OR lin4_reg);
        lin3_next <= lin3_reg;
        lin2_next <= lin2_reg;
        lin1_next <= lin1_reg;
        IF ((lp3 OR lin6_reg) = "1111111111") THEN -- Se a LINHA 6 estiver cheia...
            lin1_next <= "0000000000";
            lin2_next <= lin1_reg;
            lin3_next <= lin2_reg;
            lin4_next <= lin3_reg;
            lin5_next <= (lp1 OR lin4_reg);
            lin6_next <= (lp2 OR lin5_reg);
            lin7_next <= lin7_reg;
            lin8_next <= lin8_reg;
            lin9_next <= lin9_reg;
            lin10_next <= lin10_reg;
            lin11_next <= lin11_reg;
            lin12_next <= lin12_reg;
            lin13_next <= lin13_reg;
            lin14_next <= lin14_reg;
            lin15_next <= lin15_reg;
            IF (Pontos < 5) THEN -- Se a pontuação for menor que 5 então
                Pontos <= Pontos + 1; -- incrementa 1 ponto.
            ELSIF (Pontos = 5) THEN -- Se a pontuação chegar ao valor final (= 5) então
                Pontos <= 0; -- zera a pontuação e exibe mensagem de vitória.
            END IF;
            tempo <= 1;
        ELSE -- Caso esta linha não esteja cheia...
            tempo <= tempo + 1; -- Passa para a próxima linha.
        END IF;
    ELSE -- Caso a peça não possa mais descer...
    -----
    -- Atualiza registradores e reinicia contagem --
    -----
    lin1_reg <= lin1_next; lin2_reg <= lin2_next; lin3_reg <= lin3_next;

```

```

lin4_reg <= lin4_next; lin5_reg <= lin5_next; lin6_reg <= lin6_next;
lin7_reg <= lin7_next; lin8_reg <= lin8_next; lin9_reg <= lin9_next;
lin10_reg <= lin10_next; lin11_reg <= lin11_next; lin12_reg <= lin12_next;
lin13_reg <= lin13_next; lin14_reg <= lin14_next; lin15_reg <= lin15_next;
tempo <= 1;
END IF;

ELSIF (tempo = 7) THEN
  IF (((lp3 NAND lin7_reg) = "1111111111")AND ((lp2 NAND lin6_reg) = "1111111111")AND
    ((lp1 NAND lin5_reg) = "1111111111")) THEN      -- Se a peça puder descer, então...
    lin7_next <= (lp3 OR lin7_reg);
    lin6_next <= (lp2 OR lin6_reg);
    lin5_next <= (lp1 OR lin5_reg);
    lin4_next <= lin4_reg;
    lin3_next <= lin3_reg;
    lin2_next <= lin2_reg;
    lin1_next <= lin1_reg;
    IF ((lp3 OR lin7_reg) = "1111111111") THEN      -- Se a LINHA 7 estiver cheia...
      lin1_next <= "0000000000";
      lin2_next <= lin1_reg;
      lin3_next <= lin2_reg;
      lin4_next <= lin3_reg;
      lin5_next <= lin4_reg;
      lin6_next <= (lp1 OR lin5_reg);
      lin7_next <= (lp2 OR lin6_reg);
      lin8_next <= lin8_reg;
      lin9_next <= lin9_reg;
      lin10_next <= lin10_reg;
      lin11_next <= lin11_reg;
      lin12_next <= lin12_reg;
      lin13_next <= lin13_reg;
      lin14_next <= lin14_reg;
      lin15_next <= lin15_reg;
      IF (Pontos < 5) THEN      -- Se a pontuação for menor que 5 então
        Pontos <= Pontos + 1;   -- incrementa 1 ponto.
      ELSIF (Pontos = 5) THEN    -- Se a pontuação chegar ao valor final (= 5) então
        Pontos <= 0;           -- zera a pontuação e exibe mensagem de vitória.
      END IF;
      tempo <= 1;
    ELSE      -- Caso esta linha não esteja cheia...
      tempo <= tempo + 1; -- Passa para a próxima linha.
    END IF;
  ELSE      -- Caso a peça não possa mais descer...
    -----
    -- Atualiza registradores e reinicia contagem --
    -----
    lin1_reg <= lin1_next; lin2_reg <= lin2_next; lin3_reg <= lin3_next;
    lin4_reg <= lin4_next; lin5_reg <= lin5_next; lin6_reg <= lin6_next;
    lin7_reg <= lin7_next; lin8_reg <= lin8_next; lin9_reg <= lin9_next;
    lin10_reg <= lin10_next; lin11_reg <= lin11_next; lin12_reg <= lin12_next;
    lin13_reg <= lin13_next; lin14_reg <= lin14_next; lin15_reg <= lin15_next;
    tempo <= 1;
  END IF;

ELSIF (tempo = 8) THEN
  IF (((lp3 NAND lin8_reg) = "1111111111")AND ((lp2 NAND lin7_reg) = "1111111111")AND
    ((lp1 NAND lin6_reg) = "1111111111")) THEN      -- Se a peça puder descer, então...
    lin8_next <= (lp3 OR lin8_reg);
    lin7_next <= (lp2 OR lin7_reg);
    lin6_next <= (lp1 OR lin6_reg);
    lin5_next <= lin5_reg;
    lin4_next <= lin4_reg;
    lin3_next <= lin3_reg;
    lin2_next <= lin2_reg;
    lin1_next <= lin1_reg;
    IF ((lp3 OR lin8_reg) = "1111111111") THEN      -- Se a LINHA 8 estiver cheia...
      lin1_next <= "0000000000";
      lin2_next <= lin1_reg;
      lin3_next <= lin2_reg;
      lin4_next <= lin3_reg;
      lin5_next <= lin4_reg;
      lin6_next <= lin5_reg;
      lin7_next <= (lp1 OR lin6_reg);
      lin8_next <= (lp2 OR lin7_reg);
      lin9_next <= lin9_reg;
      lin10_next <= lin10_reg;

```

```

lin11_next <= lin11_reg;
lin12_next <= lin12_reg;
lin13_next <= lin13_reg;
lin14_next <= lin14_reg;
lin15_next <= lin15_reg;
IF (Pontos < 5) THEN      -- Se a pontuação for menor que 5 então
    Pontos <= Pontos + 1; -- incrementa 1 ponto.
ELSIF (Pontos = 5) THEN   -- Se a pontuação chegar ao valor final (= 5) então
    Pontos <= 0;           -- zera a pontuação e exibe mensagem de vitória.
END IF;
tempo <= 1;
ELSE      -- Caso esta linha não esteja cheia...
    tempo <= tempo + 1;    -- Passa para a próxima linha
END IF;
ELSE      -- Caso a peça não possa mais descer...
-----
-- Atualiza registradores e reinicia contagem --
-----
lin1_reg <= lin1_next; lin2_reg <= lin2_next; lin3_reg <= lin3_next;
lin4_reg <= lin4_next; lin5_reg <= lin5_next; lin6_reg <= lin6_next;
lin7_reg <= lin7_next; lin8_reg <= lin8_next; lin9_reg <= lin9_next;
lin10_reg <= lin10_next; lin11_reg <= lin11_next; lin12_reg <= lin12_next;
lin13_reg <= lin13_next; lin14_reg <= lin14_next; lin15_reg <= lin15_next;
tempo <= 1;
END IF;

ELSIF (tempo = 9) THEN
    IF (((lp3 NAND lin9_reg) = "1111111111")AND ((lp2 NAND lin8_reg) = "1111111111")AND
        ((lp1 NAND lin7_reg) = "1111111111")) THEN      -- Se a peça puder descer, então...
        lin9_next <= (lp3 OR lin9_reg);
        lin8_next <= (lp2 OR lin8_reg);
        lin7_next <= (lp1 OR lin7_reg);
        lin6_next <= lin6_reg;
        lin5_next <= lin5_reg;
        lin4_next <= lin4_reg;
        lin3_next <= lin3_reg;
        lin2_next <= lin2_reg;
        lin1_next <= lin1_reg;
        IF ((lp3 OR lin9_reg) = "1111111111") THEN      -- Se a LINHA 9 estiver cheia...
            lin1_next <= "0000000000";
            lin2_next <= lin1_reg;
            lin3_next <= lin2_reg;
            lin4_next <= lin3_reg;
            lin5_next <= lin4_reg;
            lin6_next <= lin5_reg;
            lin7_next <= lin6_reg;
            lin8_next <= (lp1 OR lin7_reg);
            lin9_next <= (lp2 OR lin8_reg);
            lin10_next <= lin10_reg;
            lin11_next <= lin11_reg;
            lin12_next <= lin12_reg;
            lin13_next <= lin13_reg;
            lin14_next <= lin14_reg;
            lin15_next <= lin15_reg;
            IF (Pontos < 5) THEN      -- Se a pontuação for menor que 5 então
                Pontos <= Pontos + 1; -- incrementa 1 ponto.
            ELSIF (Pontos = 5) THEN   -- Se a pontuação chegar ao valor final (= 5) então
                Pontos <= 0;           -- zera a pontuação e exibe mensagem de vitória.
            END IF;
            tempo <= 1;
        ELSE      -- Caso esta linha não esteja cheia...
            tempo <= tempo + 1;    -- Passa para a próxima linha
        END IF;
    ELSE      -- Caso a peça não possa mais descer...
    -----
    -- Atualiza registradores e reinicia contagem --
    -----
    lin1_reg <= lin1_next; lin2_reg <= lin2_next; lin3_reg <= lin3_next;
    lin4_reg <= lin4_next; lin5_reg <= lin5_next; lin6_reg <= lin6_next;
    lin7_reg <= lin7_next; lin8_reg <= lin8_next; lin9_reg <= lin9_next;
    lin10_reg <= lin10_next; lin11_reg <= lin11_next; lin12_reg <= lin12_next;
    lin13_reg <= lin13_next; lin14_reg <= lin14_next; lin15_reg <= lin15_next;
    tempo <= 1;
END IF;

ELSIF (tempo = 10) THEN

```

```

IF (((lp3 NAND lin10_reg) = "1111111111")AND ((lp2 NAND lin9_reg) = "1111111111")AND
((lp1 NAND lin8_reg) = "1111111111")) THEN      -- Se a peça puder descer, então...
lin10_next <= (lp3 OR lin10_reg);
lin9_next <= (lp2 OR lin9_reg);
lin8_next <= (lp1 OR lin8_reg);
lin7_next <= lin7_reg;
lin6_next <= lin6_reg;
lin5_next <= lin5_reg;
lin4_next <= lin4_reg;
lin3_next <= lin3_reg;
lin2_next <= lin2_reg;
lin1_next <= lin1_reg;
IF ((lp3 OR lin10_reg) = "1111111111") THEN -- Se a LINHA 10 estiver cheia...
    lin1_next <= "0000000000";
    lin2_next <= lin1_reg;
    lin3_next <= lin2_reg;
    lin4_next <= lin3_reg;
    lin5_next <= lin4_reg;
    lin6_next <= lin5_reg;
    lin7_next <= lin6_reg;
    lin8_next <= lin7_reg;
    lin9_next <= (lp1 OR lin8_reg);
    lin10_next <= (lp2 OR lin9_reg);
    lin11_next <= lin11_reg;
    lin12_next <= lin12_reg;
    lin13_next <= lin13_reg;
    lin14_next <= lin14_reg;
    lin15_next <= lin15_reg;
    IF (Pontos < 5) THEN      -- Se a pontuação for menor que 5 então
        Pontos <= Pontos + 1;  -- incrementa 1 ponto.
    ELSIF (Pontos = 5) THEN  -- Se a pontuação chegar ao valor final (= 5) então
        Pontos <= 0;          -- zera a pontuação e exibe mensagem de vitória.
    END IF;
    tempo <= 1;
ELSE      -- Caso esta linha não esteja cheia...
    tempo <= tempo + 1;      -- Passa para a próxima linha
END IF;
ELSE      -- Caso a peça não possa mais descer...
-----
-- Atualiza registradores e reinicia contagem --
-----
lin1_reg <= lin1_next; lin2_reg <= lin2_next; lin3_reg <= lin3_next;
lin4_reg <= lin4_next; lin5_reg <= lin5_next; lin6_reg <= lin6_next;
lin7_reg <= lin7_next; lin8_reg <= lin8_next; lin9_reg <= lin9_next;
lin10_reg <= lin10_next; lin11_reg <= lin11_next; lin12_reg <= lin12_next;
lin13_reg <= lin13_next; lin14_reg <= lin14_next; lin15_reg <= lin15_next;
tempo <= 1;
END IF;

ELSIF (tempo = 11) THEN
    IF (((lp3 NAND lin11_reg) = "1111111111")AND ((lp2 NAND lin10_reg) = "1111111111")AND
((lp1 NAND lin9_reg) = "1111111111")) THEN      -- Se a peça puder descer, então...
    lin11_next <= (lp3 OR lin11_reg);
    lin10_next <= (lp2 OR lin10_reg);
    lin9_next <= (lp1 OR lin9_reg);
    lin8_next <= lin8_reg;
    lin7_next <= lin7_reg;
    lin6_next <= lin6_reg;
    lin5_next <= lin5_reg;
    lin4_next <= lin4_reg;
    lin3_next <= lin3_reg;
    lin2_next <= lin2_reg;
    lin1_next <= lin1_reg;
    IF ((lp3 OR lin11_reg) = "1111111111") THEN -- Se a LINHA 11 estiver cheia...
        lin1_next <= "0000000000";
        lin2_next <= lin1_reg;
        lin3_next <= lin2_reg;
        lin4_next <= lin3_reg;
        lin5_next <= lin4_reg;
        lin6_next <= lin5_reg;
        lin7_next <= lin6_reg;
        lin8_next <= lin7_reg;
        lin9_next <= lin8_reg;
        lin10_next <= (lp1 OR lin9_reg);
        lin11_next <= (lp2 OR lin10_reg);
        lin12_next <= lin12_reg;
        lin13_next <= lin13_reg;

```

```

lin14_next <= lin14_reg;
lin15_next <= lin15_reg;
IF (Pontos < 5) THEN          -- Se a pontuação for menor que 5 então
    Pontos <= Pontos + 1;    -- incrementa 1 ponto.
ELSIF (Pontos = 5) THEN      -- Se a pontuação chegar ao valor final (= 5) então
    Pontos <= 0;             -- zera a pontuação e exibe mensagem de vitória.
END IF;
Tempo <= 1;
ELSE   -- Caso esta linha não esteja cheia...
    Tempo <= Tempo + 1;     -- Passa para a próxima linha
END IF;
ELSE   -- Caso a peça não possa mais descer...
-----
-- Atualiza registradores e reinicia contagem --
-----
lin1_reg <= lin1_next; lin2_reg <= lin2_next; lin3_reg <= lin3_next;
lin4_reg <= lin4_next; lin5_reg <= lin5_next; lin6_reg <= lin6_next;
lin7_reg <= lin7_next; lin8_reg <= lin8_next; lin9_reg <= lin9_next;
lin10_reg <= lin10_next; lin11_reg <= lin11_next; lin12_reg <= lin12_next;
lin13_reg <= lin13_next; lin14_reg <= lin14_next; lin15_reg <= lin15_next;
Tempo <= 1;
END IF;

ELSIF (tempo = 12) THEN
    IF (((lp3 NAND lin12_reg) = "1111111111") AND ((lp2 NAND lin11_reg) = "1111111111") AND
        ((lp1 NAND lin10_reg) = "1111111111")) THEN  -- Se a peça puder descer, então...
        lin12_next <= (lp3 OR lin12_reg);
        lin11_next <= (lp2 OR lin11_reg);
        lin10_next <= (lp1 OR lin10_reg);
        lin9_next <= lin9_reg;
        lin8_next <= lin8_reg;
        lin7_next <= lin7_reg;
        lin6_next <= lin6_reg;
        lin5_next <= lin5_reg;
        lin4_next <= lin4_reg;
        lin3_next <= lin3_reg;
        lin2_next <= lin2_reg;
        lin1_next <= lin1_reg;
        IF ((lp3 OR lin12_reg) = "1111111111") THEN  -- Se a LINHA 12 estiver cheia...
            lin1_next <= "0000000000";
            lin2_next <= lin1_reg;
            lin3_next <= lin2_reg;
            lin4_next <= lin3_reg;
            lin5_next <= lin4_reg;
            lin6_next <= lin5_reg;
            lin7_next <= lin6_reg;
            lin8_next <= lin7_reg;
            lin9_next <= lin8_reg;
            lin10_next <= lin9_reg;
            lin11_next <= (lp1 OR lin10_reg);
            lin12_next <= (lp2 OR lin11_reg);
            lin13_next <= lin13_reg;
            lin14_next <= lin14_reg;
            lin15_next <= lin15_reg;
            IF (Pontos < 5) THEN          -- Se a pontuação for menor que 5 então
                Pontos <= Pontos + 1;    -- incrementa 1 ponto.
            ELSIF (Pontos = 5) THEN      -- Se a pontuação chegar ao valor final (= 5) então
                Pontos <= 0;             -- zera a pontuação e exibe mensagem de vitória.
            END IF;
            Tempo <= 1;
        ELSE   -- Caso esta linha não esteja cheia...
            Tempo <= Tempo + 1;     -- Passa para a próxima linha
        END IF;
    ELSE   -- Caso a peça não possa mais descer...
        -----
        -- Atualiza registradores e reinicia contagem --
        -----
        lin1_reg <= lin1_next; lin2_reg <= lin2_next; lin3_reg <= lin3_next;
        lin4_reg <= lin4_next; lin5_reg <= lin5_next; lin6_reg <= lin6_next;
        lin7_reg <= lin7_next; lin8_reg <= lin8_next; lin9_reg <= lin9_next;
        lin10_reg <= lin10_next; lin11_reg <= lin11_next; lin12_reg <= lin12_next;
        lin13_reg <= lin13_next; lin14_reg <= lin14_next; lin15_reg <= lin15_next;
        Tempo <= 1;
    END IF;

```

```

ELSIF (tempo = 13) THEN
  IF (((lp3 NAND lin13_reg) = "1111111111")AND ((lp2 NAND lin12_reg) = "1111111111")AND
    ((lp1 NAND lin11_reg) = "1111111111")) THEN -- Se a peça puder descer, então...
    lin13_next <= (lp3 OR lin13_reg);
    lin12_next <= (lp2 OR lin12_reg);
    lin11_next <= (lp1 OR lin11_reg);
    lin10_next <= lin10_reg;
    lin9_next <= lin9_reg;
    lin8_next <= lin8_reg;
    lin7_next <= lin7_reg;
    lin6_next <= lin6_reg;
    lin5_next <= lin5_reg;
    lin4_next <= lin4_reg;
    lin3_next <= lin3_reg;
    lin2_next <= lin2_reg;
    lin1_next <= lin1_reg;
    IF ((lp3 OR lin13_reg) = "1111111111") THEN -- Se a LINHA 13 estiver cheia...
      lin1_next <= "0000000000";
      lin2_next <= lin1_reg;
      lin3_next <= lin2_reg;
      lin4_next <= lin3_reg;
      lin5_next <= lin4_reg;
      lin6_next <= lin5_reg;
      lin7_next <= lin6_reg;
      lin8_next <= lin7_reg;
      lin9_next <= lin8_reg;
      lin10_next <= lin9_reg;
      lin11_next <= lin10_reg;
      lin12_next <= (lp1 OR lin11_reg);
      lin13_next <= (lp2 OR lin12_reg);
      lin14_next <= lin14_reg;
      lin15_next <= lin15_reg;
      IF (Pontos < 5) THEN -- Se a pontuação for menor que 5 então
        Pontos <= Pontos + 1; -- incrementa 1 ponto.
      ELSIF (Pontos = 5) THEN -- Se a pontuação chegar ao valor final (= 5) então
        Pontos <= 0; -- zera a pontuação e exibe mensagem de vitória.
      END IF;
      tempo <= 1;
    ELSE -- Caso esta linha não esteja cheia...
      tempo <= tempo + 1; -- Passa para o próxima linha
    END IF;
  ELSE -- Caso a peça não possa mais descer...
  -----
  -- Atualiza registradores e reinicia contagem --
  -----
  lin1_reg <= lin1_next; lin2_reg <= lin2_next; lin3_reg <= lin3_next;
  lin4_reg <= lin4_next; lin5_reg <= lin5_next; lin6_reg <= lin6_next;
  lin7_reg <= lin7_next; lin8_reg <= lin8_next; lin9_reg <= lin9_next;
  lin10_reg <= lin10_next; lin11_reg <= lin11_next; lin12_reg <= lin12_next;
  lin13_reg <= lin13_next; lin14_reg <= lin14_next; lin15_reg <= lin15_next;
  tempo <= 1;
END IF;

ELSIF (tempo = 14) THEN
  IF (((lp3 NAND lin14_reg) = "1111111111")AND ((lp2 NAND lin13_reg) = "1111111111")AND
    ((lp1 NAND lin12_reg) = "1111111111")) THEN -- Se a peça puder descer, então...
    lin14_next <= (lp3 OR lin14_reg);
    lin13_next <= (lp2 OR lin13_reg);
    lin12_next <= (lp1 OR lin12_reg);
    lin11_next <= lin11_reg;
    lin10_next <= lin10_reg;
    lin9_next <= lin9_reg;
    lin8_next <= lin8_reg;
    lin7_next <= lin7_reg;
    lin6_next <= lin6_reg;
    lin5_next <= lin5_reg;
    lin4_next <= lin4_reg;
    lin3_next <= lin3_reg;
    lin2_next <= lin2_reg;
    lin1_next <= lin1_reg;
    IF ((lp3 OR lin14_reg) = "1111111111") THEN -- Se a LINHA 14 estiver cheia...
      lin1_next <= "0000000000";
      lin2_next <= lin1_reg;
      lin3_next <= lin2_reg;

```

```

lin4_next <= lin3_reg;
lin5_next <= lin4_reg;
lin6_next <= lin5_reg;
lin7_next <= lin6_reg;
lin8_next <= lin7_reg;
lin9_next <= lin8_reg;
lin10_next <= lin9_reg;
lin11_next <= lin10_reg;
lin12_next <= lin11_reg;
lin13_next <= (lp1 OR lin12_reg);
lin14_next <= (lp2 OR lin13_reg);
lin15_next <= lin15_reg;
IF (Pontos < 5) THEN      -- Se a pontuação for menor que 5 então
    Pontos <= Pontos + 1; -- incrementa 1 ponto.
ELSIF (Pontos = 5) THEN   -- Se a pontuação chegar ao valor final (= 5) então
    Pontos <= 0;           -- zera a pontuação e exibe mensagem de vitória.
END IF;
Tempo <= 1;
ELSE      -- Caso esta linha não esteja cheia...
    Tempo <= Tempo + 1;    -- Passa para a próxima linha.
END IF;
ELSE      -- Caso a peça não possa mais descer...
-----
-- Atualiza registradores e reinicia contagem --
-----
lin1_reg <= lin1_next; lin2_reg <= lin2_next; lin3_reg <= lin3_next;
lin4_reg <= lin4_next; lin5_reg <= lin5_next; lin6_reg <= lin6_next;
lin7_reg <= lin7_next; lin8_reg <= lin8_next; lin9_reg <= lin9_next;
lin10_reg <= lin10_next; lin11_reg <= lin11_next; lin12_reg <= lin12_next;
lin13_reg <= lin13_next; lin14_reg <= lin14_next; lin15_reg <= lin15_next;
tempo <= 1;
END IF;

ELSIF (tempo = 15) THEN
    IF (((lp3 NAND lin15_reg) = "1111111111")AND ((lp2 NAND lin14_reg) = "1111111111")AND
        ((lp1 NAND lin13_reg) = "1111111111")) THEN -- Se a peça puder descer, então...
        lin15_next <= (lp3 OR lin15_reg);
        lin14_next <= (lp2 OR lin14_reg);
        lin13_next <= (lp1 OR lin13_reg);
        lin12_next <= lin12_reg;
        lin11_next <= lin11_reg;
        lin10_next <= lin10_reg;
        lin9_next <= lin9_reg;
        lin8_next <= lin8_reg;
        lin7_next <= lin7_reg;
        lin6_next <= lin6_reg;
        lin5_next <= lin5_reg;
        lin4_next <= lin4_reg;
        lin3_next <= lin3_reg;
        lin2_next <= lin2_reg;
        lin1_next <= lin1_reg;
    IF ((lp3 OR lin15_reg) = "1111111111") THEN -- Se a LINHA 15 estiver cheia...
        lin1_next <= "0000000000";
        lin2_next <= lin1_reg;
        lin3_next <= lin2_reg;
        lin4_next <= lin3_reg;
        lin5_next <= lin4_reg;
        lin6_next <= lin5_reg;
        lin7_next <= lin6_reg;
        lin8_next <= lin7_reg;
        lin9_next <= lin8_reg;
        lin10_next <= lin9_reg;
        lin11_next <= lin10_reg;
        lin12_next <= lin11_reg;
        lin13_next <= lin12_reg;
        lin14_next <= (lp1 OR lin13_reg);
        lin15_next <= (lp2 OR lin14_reg);
    IF (Pontos < 5) THEN      -- Se a pontuação for menor que 5 então
        Pontos <= Pontos + 1; -- incrementa 1 ponto.
    ELSIF (Pontos = 5) THEN   -- Se a pontuação chegar ao valor final (= 5) então
        Pontos <= 0;           -- zera a pontuação e exibe mensagem de vitória.
    END IF;
    Tempo <= 1;
ELSE      -- Caso esta linha não esteja cheia...

```

```

        tempo <= tempo + 1;      -- Passa para a proxima linha
    END IF;
ELSE    -- Caso a peça não possa mais descer...
-----  

    -- Atualiza registradores e reinicia contagem --
-----  

lin1_reg <= lin1_next; lin2_reg <= lin2_next; lin3_reg <= lin3_next;
lin4_reg <= lin4_next; lin5_reg <= lin5_next; lin6_reg <= lin6_next;
lin7_reg <= lin7_next; lin8_reg <= lin8_next; lin9_reg <= lin9_next;
lin10_reg <= lin10_next; lin11_reg <= lin11_next; lin12_reg <= lin12_next;
lin13_reg <= lin13_next; lin14_reg <= lin14_next; lin15_reg <= lin15_next;
tempo <= 1;
END IF;

ELSIF (tempo = 16) THEN
-----  

-- Atualiza registradores e reinicia contagem --
-----  

lin1_reg <= lin1_next; lin2_reg <= lin2_next; lin3_reg <= lin3_next;
lin4_reg <= lin4_next; lin5_reg <= lin5_next; lin6_reg <= lin6_next;
lin7_reg <= lin7_next; lin8_reg <= lin8_next; lin9_reg <= lin9_next;
lin10_reg <= lin10_next; lin11_reg <= lin11_next; lin12_reg <= lin12_next;
lin13_reg <= lin13_next; lin14_reg <= lin14_next; lin15_reg <= lin15_next;
tempo <= 1;
END IF;
END IF;
END PROCESS descer_linha;

lin1 <= lin1_next; lin2 <= lin2_next; lin3 <= lin3_next; lin4 <= lin4_next;
lin5 <= lin5_next; lin6 <= lin6_next; lin7 <= lin7_next; lin8 <= lin8_next;
lin9 <= lin9_next; lin10 <= lin10_next; lin11 <= lin11_next; lin12 <= lin12_next;
lin13 <= lin13_next; lin14 <= lin14_next; lin15 <= lin15_next;

score <= pontos;

END jogo;

```

A.8 – BLOCO VIDEO_OUT

```

ENTITY video_out IS
PORT  ( clk_mov : IN BIT;
        coluna, linha: IN INTEGER RANGE 0 TO 128;
        score : IN NATURAL RANGE 0 TO 5;
        lin1, lin2, lin3, lin4, lin5, lin6, lin7, lin8, lin9, lin10,
        lin11, lin12, lin13, lin14, lin15 : IN BIT_VECTOR(0 TO 9);
        VGA_R: OUT BIT_VECTOR (3 DOWNTO 0);
        VGA_G: OUT BIT_VECTOR (3 DOWNTO 0);
        VGA_B: OUT BIT_VECTOR (3 DOWNTO 0));
END video_out;

ARCHITECTURE rgb OF video_out IS

SIGNAL pix_x, pix_y : INTEGER;
-----  

-- Coordenadas x e y das 4 paredes da grade do jogo e dos demais componentes
-----  

CONSTANT vert1_x_esq : INTEGER := 33;
CONSTANT vert1_x_dir : INTEGER := 35;
CONSTANT vert1_y_cima : INTEGER := 15;
CONSTANT vert1_y_baixo : INTEGER := 32;
CONSTANT vert2_x_esq : INTEGER := 45;
CONSTANT vert2_x_dir : INTEGER := 47;
CONSTANT vert2_y_cima : INTEGER := 15;
CONSTANT vert2_y_baixo : INTEGER := 32;
CONSTANT horiz1_x_esq : INTEGER := 34;
CONSTANT horiz1_x_dir : INTEGER := 46;
CONSTANT horiz1_y_cima : INTEGER := 15;
CONSTANT horiz1_y_baixo : INTEGER := 16;
CONSTANT horiz2_x_esq : INTEGER := 34;

```

```

CONSTANT horiz2_x_dir : INTEGER := 46;
CONSTANT horiz2_y_cima : INTEGER := 31;
CONSTANT horiz2_y_baixo : INTEGER := 33;

SIGNAL areal_x_esq : INTEGER := 35;
SIGNAL areal_x_dir : INTEGER := 45;
SIGNAL areal_y_cima : INTEGER := 16;
SIGNAL areal_y_baixo : INTEGER := 31;

SIGNAL score_x_esq : INTEGER := 50;
SIGNAL score_x_dir : INTEGER := 55;
SIGNAL score_y_cima : INTEGER := 15;
SIGNAL score_y_baixo : INTEGER := 20;

SIGNAL lin1x, lin2x, lin3x, lin4x, lin5x, lin6x, lin7x, lin8x, lin9x, lin10x,
lin11x, lin12x, lin13x, lin14x, lin15x : BIT_VECTOR(0 TO 9) := "0000000000";
-----
```

```

TYPE rom_area IS ARRAY(0 TO 14) OF BIT_VECTOR(0 TO 9); -- Área (ou tela) do jogo
TYPE rom_grade_vert IS ARRAY(0 TO 16) OF BIT_VECTOR(0 TO 1); -- Barras verticais da grade
TYPE rom_grade_horiz IS ARRAY(0 TO 1) OF BIT_VECTOR (0 TO 11); -- Barras horizontais da grade
TYPE rom_score IS ARRAY(0 TO 4) OF BIT_VECTOR (0 TO 3); -- Pontuação
CONSTANT BARRA_V1 : rom_grade_vert :=
(
  "10","10","10","10","10","10","10","10","10","10","10","10","10","10" --
17
);
CONSTANT BARRA_V2 : rom_grade_vert :=
(
  "01","01","01","01","01","01","01","01","01","01","01","01","01","01" --
17
);
CONSTANT BARRA_H2 : rom_grade_horiz :=
(
  "111111111111",
  "000000000000"
);
SIGNAL AREA1 : rom_area :=
(
  lin1x, lin2x, lin3x, lin4x, lin5x, lin6x, lin7x, lin8x, lin9x,
  lin10x, lin11x, lin12x, lin13x, lin14x, lin15x
);
CONSTANT SCORE0 : rom_score :=
(
  "1111",
  "1001",
  "1001",
  "1001",
  "1111"
);
CONSTANT SCORE1 : rom_score :=
(
  "0100",
  "0110",
  "0100",
  "0100",
  "1110"
);
CONSTANT SCORE2 : rom_score :=
(
  "1111",
  "1000",
  "1111",
  "0001",
  "1111"
);
CONSTANT SCORE3 : rom_score :=
(
  "1111",
  "1000",
  "1111",
  "1000",
  "1111"
);

```

```

CONSTANT SCORE4 : rom_score :=
(
  "1001",
  "1001",
  "1111",
  "1000",
  "1000"
);
-- CONSTANT SCORE5 : rom_score :=
-- (
--   "1111",
--   "0001",
--   "1111",
--   "1000",
--   "1111"
-- );
-----  

-- SINAIS REFERENTES AO MAPEAMENTO DA MEMÓRIA (ROM)
-----  

SIGNAL rom_addr_areal, rom_addr_H1, rom_addr_H2, rom_addr_V1, rom_addr_V2 : INTEGER;
SIGNAL rom_addr_score0, rom_addr_score1, rom_addr_score2, rom_addr_score3, rom_addr_score4 :  

INTEGER;  

-----  

SIGNAL rom_col_areal, rom_col_H1, rom_col_H2, rom_col_V1, rom_col_V2 : INTEGER;
SIGNAL rom_col_score0, rom_col_score1, rom_col_score2, rom_col_score3, rom_col_score4 : INTEGER;  

-----  

SIGNAL rom_dados_V1, rom_dados_V2 : BIT_VECTOR(1 DOWNTO 0);
SIGNAL rom_dados_H1, rom_dados_H2 : BIT_VECTOR(11 DOWNTO 0);
SIGNAL rom_dados_areal : BIT_VECTOR(9 DOWNTO 0);
SIGNAL rom_dados_score0, rom_dados_score1, rom_dados_score2, rom_dados_score3,  

rom_dados_score4 : BIT_VECTOR(3 DOWNTO 0);  

-----  

SIGNAL rom_bit_areal, rom_bit_H1, rom_bit_H2, rom_bit_V1, rom_bit_V2 : BIT;
SIGNAL rom_bit_score0, rom_bit_score1, rom_bit_score2, rom_bit_score3, rom_bit_score4 : BIT;  

-----  

SIGNAL H1_on, H1_liga_on, H2_on, H2_liga_on, V1_on, V1_liga_on, V2_on, V2_liga_on : BIT;
SIGNAL areal_on, areal_liga_on : BIT;
SIGNAL score_on : BIT;
SIGNAL score0_liga_on, score1_liga_on, score2_liga_on, score3_liga_on, score4_liga_on : BIT;  

-----  

SIGNAL H1_VGA_R, H1_VGA_G, H1_VGA_B, H2_VGA_R, H2_VGA_G, H2_VGA_B : BIT_VECTOR(3 DOWNTO 0);
SIGNAL V1_VGA_R, V1_VGA_G, V1_VGA_B, V2_VGA_R, V2_VGA_G, V2_VGA_B : BIT_VECTOR(3 DOWNTO 0);
SIGNAL areal_VGA_R, areal_VGA_G, areal_VGA_B : BIT_VECTOR(3 DOWNTO 0);
SIGNAL score_VGA_R, score_VGA_G, score_VGA_B : BIT_VECTOR(3 DOWNTO 0);  

-----  

BEGIN  

  PROCESS (clk_mov)
  BEGIN
    IF ((clk_mov'EVENT) AND (clk_mov ='0')) THEN
      AREAL <=
      (
        lin1, lin2, lin3, lin4, lin5, lin6, lin7, lin8, lin9,
        lin10, lin11, lin12, lin13, lin14, lin15
      );
    END IF;
  END PROCESS;  

  pix_x <= coluna;
  pix_y <= linha;  

-----  

--> DEFININDO PIXELS DAS BARRAS DA GRADE, E A SAÍDA RGB CORRESPONDENTE
-----  

V1_on <= '1' WHEN ((pix_x >= vert1_x_esq) AND (pix_x < vert1_x_dir)
  AND (pix_y >= vert1_y_cima) AND (pix_y < vert1_y_baixo)) ELSE
  '0';
  rom_addr_V1 <= pix_y - vert1_y_cima;
  rom_col_V1 <= pix_x - vert1_x_esq;
  rom_dados_V1 <= BARRA_V1(rom_addr_V1);
  rom_bit_V1 <= rom_dados_V1(rom_col_V1);
  V1_liga_on <= '1' WHEN ((V1_on = '1') AND (rom_bit_V1 = '1')) ELSE
  '0';
  V1_VGA_R <= "0000";

```

```

        V1_VGA_G <= "0000";
        V1_VGA_B <= "1111"; -- Azul
V2_on <= '1' WHEN ((pix_x >= vert2_x_esq) AND (pix_x < vert2_x_dir)
                     AND (pix_y >= vert2_y_cima) AND (pix_y < vert2_y_baixo)) ELSE
    '0';
    rom_addr_V2 <= pix_y - vert2_y_cima;
    rom_col_V2 <= pix_x - vert2_x_esq;
    rom_dados_V2 <= BARRA_V2(rom_addr_V2);
    rom_bit_V2 <= rom_dados_V2(rom_col_V2);
    V2_liga_on <= '1' WHEN ((V2_on = '1') AND (rom_bit_V2 = '1')) ELSE
        '0';
    V2_VGA_R <= "0000";
    V2_VGA_G <= "0000";
    V2_VGA_B <= "1111"; -- Azul
H1_liga_on <= '1' WHEN((pix_x >= horiz1_x_esq) AND (pix_x < horiz1_x_dir)
                     AND (pix_y >= horiz1_y_cima) AND (pix_y < horiz1_y_baixo)) ELSE
    '0';
    H1_VGA_R <= "0000";
    H1_VGA_G <= "0000";
    H1_VGA_B <= "1111"; -- Azul
H2_on <= '1' WHEN((pix_x >= horiz2_x_esq) AND (pix_x < horiz2_x_dir)
                     AND (pix_y >= horiz2_y_cima) AND (pix_y < horiz2_y_baixo)) ELSE
    '0';
    rom_addr_H2 <= pix_y - horiz2_y_cima;
    rom_col_H2 <= pix_x - horiz2_x_esq;
    rom_dados_H2 <= BARRA_H2(rom_addr_H2);
    rom_bit_H2 <= rom_dados_H2(rom_col_H2);
    H2_liga_on <= '1' WHEN ((H2_on = '1') AND (rom_bit_H2 = '1')) ELSE
        '0';
    H2_VGA_R <= "0000";
    H2_VGA_G <= "0000";
    H2_VGA_B <= "1111"; -- Azul
-----
--> CONFIGURANDO A EXIBIÇÃO DA ÁREA DE MOVIMENTO DAS PEÇAS
-----
areal_on <= '1' WHEN((pix_x >= areal_x_esq) AND (pix_x < areal_x_dir)
                     AND (pix_y >= areal_y_cima) AND (pix_y < areal_y_baixo)) ELSE
    '0';
    rom_addr_areal <= pix_y - areal_y_cima;
    rom_col_areal <= pix_x - areal_x_esq;
    rom_dados_areal <= AREAL(rom_addr_areal);
    rom_bit_areal <= rom_dados_areal(rom_col_areal);
    areal_liga_on <= '1' WHEN ((areal_on = '1') AND (rom_bit_areal = '1')) ELSE
        '0';
    areal_VGA_R <= "1111";
    areal_VGA_G <= "1111";
    areal_VGA_B <= "0000"; -- AMARELO
-----
--> CONFIGURANDO A EXIBIÇÃO DA PONTUAÇÃO
-----
score_on <= '1' WHEN((pix_x >= score_x_esq) AND (pix_x <= score_x_dir)
                     AND (pix_y >= score_y_cima) AND (pix_y < score_y_baixo)) ELSE
    '0';
-- SCORE 0:
    rom_addr_score0 <= pix_y - score_y_cima;
    rom_col_score0 <= pix_x - score_x_esq;
    rom_dados_score0 <= SCORE0(rom_addr_score0);
    rom_bit_score0 <= rom_dados_score0(rom_col_score0);
    score0_liga_on <= '1' WHEN ((score_on = '1') AND (rom_bit_score0 = '1') AND (score = 0)) ELSE
        '0';
-- SCORE 1:
    rom_addr_score1 <= pix_y - score_y_cima;
    rom_col_score1 <= pix_x - score_x_esq;
    rom_dados_score1 <= SCORE1(rom_addr_score1);
    rom_bit_score1 <= rom_dados_score1(rom_col_score1);
    score1_liga_on <= '1' WHEN ((score_on = '1') AND (rom_bit_score1 = '1') AND (score = 1)) ELSE
        '0';
-- SCORE 2:
    rom_addr_score2 <= pix_y - score_y_cima;
    rom_col_score2 <= pix_x - score_x_esq;
    rom_dados_score2 <= SCORE2(rom_addr_score2);
    rom_bit_score2 <= rom_dados_score2(rom_col_score2);
    score2_liga_on <= '1' WHEN ((score_on = '1') AND (rom_bit_score2 = '1') AND (score = 2)) ELSE
        '0';

```

```

-- SCORE 3:
rom_addr_score3 <= pix_y - score_y_cima;
rom_col_score3 <= pix_x - score_x_esq;
rom_dados_score3 <= SCORE3(rom_addr_score3);
rom_bit_score3 <= rom_dados_score3(rom_col_score3);
score3_liga_on <= '1' WHEN ((score_on = '1') AND (rom_bit_score3 = '1') AND (score = 3)) ELSE
    '0';
-- SCORE 4:
rom_addr_score4 <= pix_y - score_y_cima;
rom_col_score4 <= pix_x - score_x_esq;
rom_dados_score4 <= SCORE4(rom_addr_score4);
rom_bit_score4 <= rom_dados_score4(rom_col_score4);
score4_liga_on <= '1' WHEN ((score_on = '1') AND (rom_bit_score4 = '1') AND (score = 4)) ELSE
    '0';
-- -- SCORE 5:
-- rom_addr_score5 <= pix_y - score_y_cima;
-- rom_col_score5 <= pix_x - score_x_esq;
-- rom_dados_score5 <= SCORE5(rom_addr_score5);
-- rom_bit_score5 <= rom_dados_score5(rom_col_score5);
-- score5_liga_on <= '1' WHEN ((score_on = '1') AND (rom_bit_score5 = '1') AND (score = 5)) ELSE
    '0';

score_VGA_R <= "0000";
score_VGA_G <= "1111";
score_VGA_B <= "0000"; -- VERDE

-----
--> HABILITAÇÃO/CONFIGURAÇÃO DA SAÍDA DE VÍDEO VGA
-----

cores_RGB: PROCESS (H1_liga_on, H1_VGA_R, H1_VGA_G, H1_VGA_B, areal_liga_on,
                    H2_liga_on, H2_VGA_R, H2_VGA_G, H2_VGA_B, areal_VGA_R,
                    V1_liga_on, V1_VGA_R, V1_VGA_G, V1_VGA_B, areal_VGA_G,
                    V2_liga_on, V2_VGA_R, V2_VGA_G, V2_VGA_B, areal_VGA_B,
                    score_VGA_R, score_VGA_G, score_VGA_B, score0_liga_on, score1_liga_on,
                    score2_liga_on, score3_liga_on, score4_liga_on, --, score5_liga_on)
BEGIN
    IF (H1_liga_on = '1') THEN
        VGA_R <= H1_VGA_R;
        VGA_G <= H1_VGA_G;
        VGA_B <= H1_VGA_B;
    ELSIF (H2_liga_on = '1') THEN
        VGA_R <= H2_VGA_R;
        VGA_G <= H2_VGA_G;
        VGA_B <= H2_VGA_B;
    ELSIF (V1_liga_on = '1') THEN
        VGA_R <= V1_VGA_R;
        VGA_G <= V1_VGA_G;
        VGA_B <= V1_VGA_B;
    ELSIF (V2_liga_on = '1') THEN
        VGA_R <= V2_VGA_R;
        VGA_G <= V2_VGA_G;
        VGA_B <= V2_VGA_B;
    ELSIF (areal_liga_on = '1') THEN
        VGA_R <= areal_VGA_R;
        VGA_G <= areal_VGA_G;
        VGA_B <= areal_VGA_B;
    ELSIF (score0_liga_on = '1') THEN
        VGA_R <= score_VGA_R;
        VGA_G <= score_VGA_G;
        VGA_B <= score_VGA_B;
    ELSIF (score1_liga_on = '1') THEN
        VGA_R <= score_VGA_R;
        VGA_G <= score_VGA_G;
        VGA_B <= score_VGA_B;
    ELSIF (score2_liga_on = '1') THEN
        VGA_R <= score_VGA_R;
        VGA_G <= score_VGA_G;
        VGA_B <= score_VGA_B;
    ELSIF (score3_liga_on = '1') THEN
        VGA_R <= score_VGA_R;
        VGA_G <= score_VGA_G;
        VGA_B <= score_VGA_B;
    ELSIF (score4_liga_on = '1') THEN
        VGA_R <= score_VGA_R;

```

```
        VGA_G <= score_VGA_G;
        VGA_B <= score_VGA_B;
--      ELSIF (score5_liga_on = '1') THEN
--          VGA_R <= score_VGA_R;
--          VGA_G <= score_VGA_G;
--          VGA_B <= score_VGA_B;
      ELSE
          VGA_R <= "0000";
          VGA_G <= "0000";
          VGA_B <= "0000";
      END IF;
END PROCESS cores_RGB;
END rgb;
```

ANEXO**TABELA ASCII / SCancode**

Tabela de Scancodes gerados nas ações de pressionar tecla (make) e soltar tecla (break):

key location	101/102 Enhanced Keyboard	scan 1 make	scan 1 break	scan 2 make	scan 2 break
	DO NOT USE	00	80	00	F0 00
	DO NOT USE	E0_00	E0_80	E0_00	E0_F0 00
1	~ `	29	A9	0E	F0 0E
		E0_29	E0_A9	E0_0E	E0_F0 0E
2	! 1	02	82	16	F0 16
		E0_02	E0_82	E0_16	E0_F0 16
3	@ 2	03	83	1E	F0 1E
		E0_03	E0_83	E0_1E	E0_F0 1E
4	# 3	04	84	26	F0 26
		E0_04	E0_84	E0_26	E0_F0 26
5	\$ 4	05	85	25	F0 25
		E0_05	E0_85	E0_25	E0_F0 25
6	% 5	06	86	2E	F0 2E
		E0_06	E0_86	E0_2E	E0_F0 2E
7	^ 6	07	87	36	F0 36
		E0_07	E0_87	E0_36	E0_F0 36
8	& 7	08	88	3D	F0 3D
		E0_08	E0_88	E0_3D	E0_F0 3D
9	* 8	09	89	3E	F0 3E
		E0_09	E0_89	E0_3E	E0_F0 3E
10	(9	0A	8A	46	F0 46
		E0_0A	E0_8A	E0_46	E0_F0 46
11) 0	0B	8B	45	F0 45
		E0_0B	E0_8B	E0_45	E0_F0 45
12	-	0C	8C	4E	F0 4E
		E0_0C	E0_8C	E0_4E	E0_F0 4E
13	+ =	0D	8D	55	F0 55
		E0_0D	E0_8D	E0_55	E0_F0 55
15	Backspace	0E	8E	66	F0 66
		E0_0E	E0_8E	E0_66	E0_F0 66
16	Tab	0F	8F	0D	F0 0D
		E0_0F	E0_8F	E0_0D	E0_F0 0D
17	Q	10	90	15	F0 15
		E0_10	E0_90	E0_15	E0_F0 15
18	W	11	91	1D	F0 1D
		E0_11	E0_91	E0_1D	E0_F0 1D
19	E	12	92	24	F0 24
		E0_12	E0_92	E0_24	E0_F0 24
20	R	13	93	2D	F0 2D
		E0_13	E0_93	E0_2D	E0_F0 2D
21	T	14	94	2C	F0 2C
		E0_14	E0_94	E0_2C	E0_F0 2C
22	Y	15	95	35	F0 35
		E0_15	E0_95	E0_35	E0_F0 35
23	U	16	96	3C	F0 3C
		E0_16	E0_96	E0_3C	E0_F0 3C
24	I	17	97	43	F0 43
		E0_17	E0_97	E0_43	E0_F0 43
25	O	18	98	44	F0 44
		E0_18	E0_98	E0_44	E0_F0 44
26	P	19	99	4D	F0 4D
		E0_19	E0_99	E0_4D	E0_F0 4D
27	{ [1A	9A	54	F0 54
		E0_1A	E0_9A	E0_54	E0_F0 54
28	}]	1B	9B	5B	F0 5B
		E0_1B	E0_9B	E0_5B	E0_F0 5B
29*	\	2B	AB	5D	F0 5D

key location	101/102 Enhanced Keyboard	scan 1 make	scan 1 break	scan 2 make	scan 2 brake
		E0_2B	E0_AB	E0_5D	E0_F0 5D
30	Caps Lock	3A	BA	58	F0 58
		E0_3A	E0_BA	E0_58	E0_F0 58
31	A	1E	9E	1C	F0 1C
		E0_1E	E0_9E	E0_1C	E0_F0 1C
32	S	1F	9F	1B	F0 1B
		E0_1F	E0_9F	E0_1B	E0_F0 1B
33	D	20	A0	23	F0 23
		E0_20	E0_A0	E0_23	E0_F0 23
34	F	21	A1	2B	F0 2B
		E0_21	E0_A1	E0_2B	E0_F0 2B
35	G	22	A2	34	F0 34
		E0_22	E0_A2	E0_34	E0_F0 34
36	H	23	A3	33	F0 33
		E0_23	E0_A3	E0_33	E0_F0 33
37	J	24	A4	3B	F0 3B
		E0_24	E0_A4	E0_3B	E0_F0 3B
38	K	25	A5	42	F0 42
		E0_25	E0_A5	E0_42	E0_F0 42
39	L	26	A6	4B	F0 4B
		E0_26	E0_A6	E0_4B	E0_F0 4B
40	:	27	A7	4C	F0 4C
		E0_27	E0_A7	E0_4C	E0_F0 4C
41	" "	28	A8	52	F0 52
		E0_28	E0_A8	E0_52	E0_F0 52
42**		2B	AB	5D	F0 5D
		E0_2B	E0_AB	E0_5D	E0_F0 5D
43	Enter	1C	9C	5A	F0 5A
44	L SHIFT	2A	AA	12	F0 12
		E0_2A	E0_AA	E0_12	E0_F0 12
45**		56	D6	61	F0 61
		E0_56	E0_D6	E0_61	E0_F0 61
46	Z	2C	AC	1A	F0 1A
		E0_2C	E0_AC	E0_1A	E0_F0 1A
47	X	2D	AD	22	F0 22
		E0_2D	E0_AD	E0_22	E0_F0 22
48	C	2E	AE	21	F0 21
		E0_2E	E0_AE	E0_21	E0_F0 21
49	V	2F	AF	2A	F0 2A
		E0_2F	E0_AF	E0_2A	E0_F0 2A
50	B	30	B0	32	F0 32
		E0_30	E0_B0	E0_32	E0_F0 32
51	N	31	B1	31	F0 31
		E0_31	E0_B1	E0_31	E0_F0 31
52	M	32	B2	3A	F0 3A
		E0_32	E0_B2	E0_3A	E0_F0 3A
53	< ,	33	B3	41	F0 41
		E0_33	E0_B3	E0_41	E0_F0 41
54	> .	34	B4	49	F0 49
		E0_34	E0_B4	E0_49	E0_F0 49
55	? /	35	B5	4A	F0 4A
		E0_35	E0_B5	E0_4A	E0_F0 4A
56***		73	F3	51	F0 51
		E0_73	E0_F3	E0_51	E0_F0 51
57	R SHIFT	36	B6	59	F0 59
		E0_36	E0_B6	E0_59	E0_F0 59
58	L CTRL	1D	9D	14	F0 14
60	L ALT	38	B8	11	F0 11
		E0_38	E0_B8	E0_11	E0_F0 11
61	Space Bar	39	B9	29	F0 29
		E0_39	E0_B9	E0_29	E0_F0 29
62	R ALT	E0 38	E0 B8	E0 11	E0 F0 11

key location	101/102 Enhanced Keyboard	scan 1 make	scan 1 break	scan 2 make	scan 2 brake
64	R CTRL	E0 1D	E0 9D	E0 14	E0 F0 14
75	Insert	Note 1	Note 1	Note 2	Note 2
76	Delete	Note 1	Note 1	Note 2	Note 2
79	L Arrow	Note 1	Note 1	Note 2	Note 2
80	Home	Note 1	Note 1	Note 2	Note 2
81	End	Note 1	Note 1	Note 2	Note 2
83	Up Arrow	Note 1	Note 1	Note 2	Note 2
84	Dn Arrow	Note 1	Note 1	Note 2	Note 2
85	Page Up	Note 1	Note 1	Note 2	Note 2
86	Page Down	Note 1	Note 1	Note 2	Note 2
89	R Arrow	Note 1	Note 1	Note 2	Note 2
90	Num Lock	45	C5	77	F0 77
		E0 45	E0 C5	E0 77	E0 F0 77
91	Numeric 7	47	C7	6C	F0 6C
92	Numeric 4	4B	CB	6B	F0 6B
93	Numeric 1	4F	CF	69	F0 69
95	Numeric /	Note 3	Note 3	Note 3	Note 3
96	Numeric 8	48	C8	75	F0 75
97	Numeric 5	4C	CC	73	F0 73
98	Numeric 2	50	D0	72	F0 72
99	Numeric 0	52	D2	70	F0 70
100	Numeric *	37	B7	7C	F0 7C
		E0 37	E0 B7	E0 7C	E0 F0 7C
101	Numeric 9	49	C9	7D	F0 7D
102	Numeric 6	4D	CD	74	F0 74
103	Numeric 3	51	D1	7A	F0 7A
104	Numeric .	53	D3	71	F0 71
105	Numeric -	4A	CA	7B	F0 7B
106	Numeric +	4E	CE	79	F0 79
107***		7E	FE	6D	F0 6D
	DO NOT USE	E0 7E	E0 FE	E0 6D	E0 F0 6D
108	Numeric Enter	E0 1C	E0 9C	E0 5A	E0 F0 5A
110	Esc	01	81	76	F0 76
		E0 01	E0 81	E0 76	E0 F0 76
112	F1	3B	BB	05	F0 05
		E0 3B	E0 BB	E0 05	E0 F0 05
113	F2	3C	BC	06	F0 06
		E0 3C	E0 BC	E0 06	E0 F0 06
114	F3	3D	BD	04	F0 05
		E0 3D	E0 BD	E0 04	E0 F0 05
115	F4	3E	BE	0C	F0 0C
		E0 3E	E0 BE	E0 0C	E0 F0 0C
116	F5	3F	BF	03	F0 03
		E0 3F	E0 BF	E0 03	E0 F0 03
117	F6	40	C0	0B	F0 0B
		E0 40	E0 C0	E0 0B	E0 F0 0B
118	F7	41	C1	83	F0 83
		E0 41	E0 C1	E0 83	E0 F0 83
119	F8	42	C2	0A	F0 0A
		E0 42	E0 C2	E0 0A	E0 F0 0A
120	F9	43	C3	01	F0 01
		E0 43	E0 C3	E0 01	E0 F0 01
121	F10	44	C4	09	F0 09
		E0 44	E0 C4	E0 09	E0 F0 09
122	F11	57	D7	78	F0 78
123	F12	58	D8	07	F0 07
124	Print Screen	Note 4	Note 4	Note 4	Note 4
125	Scroll Lock	46	C6	7E	F0 7E
		E0 46	E0 C6	E0 7E	E0 F0 7E
126	Pause	Note 5	Note 5	Note 5	Note 5
		59	D9	0F	F0 0F

key location	101/102 Enhanced Keyboard	scan 1 make	scan 1 break	scan 2 make	scan 2 brake
		E0_59	E0_D9	E0_0F	E0_F0_0F
		5B	DB	1F	F0_1F
Left Win		E0_5B	E0_DB	E0_1F	E0_F0_1F
		5C	DC	27	F0_27
Right Win		E0_5C	E0_DC	E0_27	E0_F0_27
		5D	DD	2F	F0_2F
Application		E0_5D	E0_DD	E0_2F	E0_F0_2F
		5E	DE	37	F0_37
ACPI Power		E0_5E	E0_DE	E0_37	E0_F0_37
		5F	DF	3F	F0_3F
ACPI Sleep		E0_5F	E0_DF	E0_3F	E0_F0_3F
DO NOT USE		60	E0	47	F0_47
DO NOT USE		E0_60	E0_E0	E0_47	E0_F0_47
DO NOT USE		61	E1	4F	F0_4F
DO NOT USE		E0_61	E0_E1	E0_4F	E0_F0_4F
		62	E2	56	F0_56
		E0_62	E0_E2	E0_56	E0_F0_56
		63	E3	5E	F0_5E
ACPI Wake		E0_63	E0_E3	E0_5E	E0_F0_5E
		64	E4	08	F0_08
		E0_64	E0_E4	E0_08	E0_F0_08
		65	E5	10	F0_10
		E0_65	E0_E5	E0_10	E0_F0_10
		66	E6	18	F0_18
		E0_66	E0_E6	E0_18	E0_F0_18
		67	E7	20	F0_20
		E0_67	E0_E7	E0_20	E0_F0_20
		68	E8	28	F0_28
		E0_68	E0_E8	E0_28	E0_F0_28
		69	E9	30	F0_30
		E0_69	E0_E9	E0_30	E0_F0_30
		6A	EA	38	F0_38
		E0_6A	E0_EA	E0_38	E0_F0_38
		6B	EB	40	F0_40
		E0_6B	E0_EB	E0_40	E0_F0_40
		6C	EC	48	F0_48
		E0_6C	E0_EC	E0_48	E0_F0_48
		6D	ED	50	F0_50
		E0_6D	E0_ED	E0_50	E0_F0_50
		6E	EE	57	F0_57
		E0_6E	E0_EE	E0_57	E0_F0_57
		6F	EF	6F	F0_6F
		E0_6F	E0_EF	E0_6F	E0_F0_6F
DBE_KATAKANA‡		70	F0	13	F0_13
		E0_70	E0_F0	E0_13	E0_F0_13
		71	F1	19	F0_19
		E0_71	E0_F1	E0_19	E0_F0_19
		72	F2	39	F0_39
		E0_72	E0_F2	E0_39	E0_F0_39
		74	F4	53	F0_53
		E0_74	E0_F4	E0_53	E0_F0_53
		75	F5	5C	F0_5C
		E0_75	E0_F5	E0_5C	E0_F0_5C
		76	F6	5F	F0_5F
		E0_76	E0_F6	E0_5F	E0_F0_5F
DBE_SBCSCHAR‡		77	F7	62	F0_62
		E0_77	E0_F7	E0_62	E0_F0_62
		78	F8	63	F0_63
		E0_78	E0_F8	E0_63	E0_F0_63
CONVERT‡		79	F9	64	F0_64
		E0_79	E0_F9	E0_64	E0_F0_64

key location	101/102 Enhanced Keyboard	scan 1 make	scan 1 break	scan 2 make	scan 2 brake
	DO NOT USE	7A	FA	65	F0 65
	DO NOT USE	E0 7A	E0 FA	E0 65	E0 F0 65
	NONCONVERT‡	7B	FB	67	F0 67
	DO NOT USE	E0 7B	E0 FB	E0 67	E0 F0 67
	DO NOT USE	7C	FC	68	F0 68
	DO NOT USE	E0 7C	E0 FC	E0 68	E0 F0 68
	DO NOT USE	7D	FD	6A	F0 6A
	DO NOT USE	E0 7D	E0 FD	E0 6A	E0 F0 6A
	DO NOT USE	7F	FF	6E	F0 6E
	DO NOT USE	E0 7F	E0 FF	E0 6E	E0 F0 6E

*Nota 1:

Key Location	US key assignment	Base Make	Base Break
75	Insert	E0 52	E0 D2
76	Delete	E0 53	E0 D3
79	Left Arrow	E0 4B	E0 CB
80	Home	E0 47	E0 C7
81	End	E0 4F	E0 CF
83	Up Arrow	E0 48	E0 C8
84	Dn Arrow	E0 50	E0 D0
85	Page Up	E0 49	E0 C9
86	Page Down	E0 51	E0 D1
89	Right Arrow	E0 4D	E0 CD

*Nota 2:

Key Location	US key assignment	Base Make	Base Break
75	Insert	E0 52	E0 D2
76	Delete	E0 53	E0 D3
79	Left Arrow	E0 4B	E0 CB
80	Home	E0 47	E0 C7
81	End	E0 4F	E0 CF
83	Up Arrow	E0 48	E0 C8
84	Dn Arrow	E0 50	E0 D0
85	Page Up	E0 49	E0 C9
86	Page Down	E0 51	E0 D1
89	Right Arrow	E0 4D	E0 CD