UNIVERSIDADE FEEVALE

MARCELO FABIANO VIER

PROTÓTIPO DE UM SISTEMA DE CAPTURA DE IMAGENS DE CÂMERA CCD MONOCROMÁTICA E TRANSFERÊNCIA PARA UM MONITOR DE VÍDEO, IMPLEMENTADO EM ARQUITETURA RECONFIGURÁVEL

Novo Hamburgo, julho de 2010.

MARCELO FABIANO VIER

PROTÓTIPO DE UM SISTEMA DE CAPTURA DE IMAGENS DE CÂMERA CCD MONOCROMÁTICA E TRANFERÊNCIA PARA UM MONITOR DE VÍDEO, IMPLEMENTADO EM ARQUITETURA RECONFIGURÁVEL

Universidade FEEVALE
Instituto de Ciências Exatas e Tecnológicas-ICET
Engenharia Eletrônica
Trabalho de Conclusão de Curso

Professor Orientador: Delfim Luiz Torok

Novo Hamburgo, julho de 2010.

FOLHA DE APROVAÇÃO

Dedico este trabalho aos meus pais, Normélio e Leatrice que sempre me incentivaram.

AGRADECIMENTO

À minha família, pelo acompanhamento e apoio, pela compreensão e os estímulos.

À minha querida noiva Daiana Caroline Britz, por estar do meu lado o tempo inteiro.

À Delfim Torok, paciente orientador e organizador dos meus pensamentos desconexos.

À BommTempo informática, pela compreensão aos dias ausentes da empresa e dedicados a este trabalho.

A todos meus amigos e colegas de faculdade que de uma forma ou de outra contribuíram para concluir esta importante etapa da minha vida.

RESUMO

O Trabalho de Conclusão de Curso descrito nesta monografia trata do estudo, desenvolvimento e implementação de um protótipo de sistema de captura de imagens de câmera CCD monocromática e transferência para um monitor de vídeo padrão implementado em arquitetura reconfigurável. A digitalização do sinal de vídeo composto NTSC, é realizado por um circuito decodificador de vídeo (ADV7183B) da Analog Divices[1] . A implementação do controle de captura das imagens é realizado pela Plataforma de Prototipação Spartan 3E Starter Kit da XILINX[2] . Esta plataforma é equipada com uma matriz de portas programáveis no campo, do inglês Field Programmable Gate Array (FPGA) XC3S500 da XILINX. Para configurar a placa de prototipação é utilizada uma linguagem de descrição de hardware, em inglês Hardware Description Language (HDL). As imagens são exibidas em um monitor de vídeo padrão.

Palavras chave: Vídeo; FPGA; VHDL.

---

[1] Analog Divices- É uma empresa de semicondutores multinacional americana especializada na conversão de dados e tecnologia de condicionamento de sinal.
[2] XILINX- Empresa de semicondutores a se especializar na fabricação de hardware reconfigurável, sendo a primeira inventora do (FPGA).

ABSTRACT

The graduation monograph work described here, is about the studies, development and implementation of a prototype capturing images system from monochrome CCd camera and transferred to a standard video monitor, implemented on reconfigurable architecture. The digitization of the composite signal NTSC is performed by a video decoder circuit (ADV7183B) from Analog Divices. The implementation of the image capture control is performed by XILINX Spartan 3E Starter Kit prototyping platform. This platform is equipped with XILINX Field Programmable Gate Array (FPGA) XC3S500. A hardware description language (HDL) is used to configure prototyping board. The images are displayed on a video display standard.

Word Key: Video; FPGA; VHDL.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AD	Analog to Digital
ASCII	American Standard Code for Information Interchange
BNC	Bayonet Neill-Concelman
CAD	Computer Aided Desing
CCD	Charge Coupled Display
CLB	Configurable Logic Block
CPLD	Complex Programable Logic Device
CVBS	Composite Video Baseband Signal
DA	Digital to Analog
DCE	Data Communications Equipment
DDR	Double Data Rate
CRT	Cathode Ray Tube
DTE	Data Terminal Equipment
EEPROM	Electrically Erasable PROM
FPGA	Field Port Gate Array
GIF	Graphics Interchange Format
GND	Ground
HDL	Hardware Description Language
HCPLD	High Capacity Programable Logic Device
HSYNC	Horizontal Synchronization
Hz	Hertz
I2C	Inter-Intergrated Circuit
IEEE	Institute of Electrical and Electronics Engineers
IOB	In/Out Block
IRE	Institute of Radio Engineers
ISE	Integrated Software Environment
ISP	In System Program
JPEG	Joint Photographic Experts Group
LCD	Liquid Crystal Display
LUT	Look Up Table
LZW	Lempel Ziv Welch
NTSC	National Television Standards Commitee
PAL	Phase Altenate Lines
Pixel	Picture Element
PLD	Programable Logic Device

PROM	Programmable ROM
RAM	Random Acess Memory
RCA	Radio Corporation of America
ROM	Read Only Memory
SB	Switch Box
S-Video	Separated Video
SECAM	Systeme Electronique Couleur Avec Memoire
SDRAM	Synchronous Dynamic RAM
SORC	System on a reconfigurable chip
SRAM	Static RAM
TV	Televisão
USB	Universal Standard Bus
V	Volt
VGA	Video Graphics Array
VHDL	VHSIC Hardware Description Language
VHF	Very High Frequency
VHSIC	Very High Speed Integrated Circuits
VSYNC	Vertical Synchronization

SUMÁRIO

INTRODUÇÃO

Para ser possível exibir uma imagem no monitor de vídeo padrão através da Placa de Prototipação Spartan 3E Starter Kit da XILINX, é necessário estudar os padrões e formatos dos sinais de vídeo, para assim ser possível desenvolver e implementar um módulo capaz de capturar e exibir a imagem no monitor. A captura de imagens é realizada por uma câmera CCD monocromática, e para digitalizar e decodificar o sinal analógico é utilizado a Placa VDEC1 capacitada com um circuito decodificador de vídeo.

Para uma melhor compreensão do assunto abordado, é realizado inicialmente um estudo sobre a imagem digital, sinais de vídeo analógico, lentes de câmeras, formatos de vídeo. Após são estudadas as placas Decodificadora de Vídeo VDEC1 e de Prototipação Spartan 3E Starter Kit da XILINX. Na seqüência é desenvolvido o Sistema de Captura de imagens, contendo os módulos de Configuração da Placa Decodificadora, Captura e Armazenamento, bem como o Módulo de Controle do Monitor VGA. Todo o Sistema de Captura é desenvolvido em Linguagem de Descrição de Hardware.

O Sistema de Captura permite através de seus módulos configurar a Placa Decodificadora de Vídeo, acoplada a Plataforma de Prototipação, para vídeo Composto NTSC, bem como capturar os sinais de dados do vídeo Composto e armazena-los em uma

memória de dupla porta, permitindo ao módulo de Controle ao monitor VGA transferir a imagem armazenada na memória para o monitor.

O volume deste Trabalho está dividido em sete capítulos. No Primeiro Capítulo é apresentado um estudo sobre imagem digital, amostragem, quantização e dispositivos para aquisição de imagens. No Segundo Capítulo são estudados os sinais de vídeo, bem como a forma e métodos para capturar os dados referentes aos pontos de imagem (pixels). No Terceiro Capítulo é realizado um breve estudo da Linguagem de Descrição de Hardware, métodos para simulação e testes dos módulos que compõem o Sistema de Captura, bem como as técnicas para desenvolvimento e implementação dos mesmos. O Quarto Capítulo trata-se de um estudo sobre Hardware Reconfigurável, e da Placa de Prototipação. No Quinto Capítulo são estudados os dispositivos de Hardwares Específicos para interconecta-los a Plataforma de Prototipação, de forma a constituírem um Sistema de Captura de Imagem, descritos em Linguagem de Descrição de Hardware, configurados na Plataforma de Prototipação. Já no Sexto Capítulo é apresentado o desenvolvimento dos módulos de, Configuração, Captura e Armazenamento dos pontos de imagem, bem como o Controle do Monitor VGA. No Último Capítulo são realizados os testes e validação do Sistema como um todo apresentando imagens padrão no Monitor de Vídeo. Por fim são apresentadas as considerações finais do trabalho, e propostas para trabalhos futuros.

1 IMAGEM DIGITAL

Máquinas e computadores não podem manipular imagens contínuas, para isso estes sistemas utilizam-se de imagens digitais. Uma imagem digital pode ser definida como uma função bidimensional *f(x, y),* onde *x* e *y* são coordenadas num plano bidimensional, e o valor de f em qualquer par de coordenadas *(x, y)* é chamada de intensidade ou escala de cinza da imagem em cada ponto. Quando, em uma imagem *x, y* e amplitude *f* são todos discretos e quantizados, chama-se a imagem de digital (GONZALEZ, 1992). Uma imagem digital é composta de um número finito de elementos, onde cada elemento possui uma localização e um valor. Cada elemento da matriz é denominado de *pixel*[3] ou *pel*, ambos abreviaturas da expressão em inglês *picture element*. Em uma imagem digital cada *pixel* corresponde a uma parte física de um objeto tridimensional no mundo "real". Este objeto é iluminado por algum tipo de luz, onde parte e absorvida e parte e refletida. A parte da luz refletida é então captada por um sensor de imagem que captura a imagem do objeto, este sensor é quem determina o valor para cada *pixel* na imagem adquirida (PETROU, 1999).

Cada *pixel* é armazenado e juntos formam um mapa de *bits "bit-map*", esse mapeamento de *bit* serve para reproduzir a imagem digitalmente. Uma imagem

---

[3] Pixel - É o menor componente de uma imagem digital, quanto maior o número de pixels da imagem, maior é a capacidade de representar detalhes.

monocromática é uma função de intensidade de luz bidimensional *f(x,y)*, onde *x,y* denotam coordenadas espaciais. O valor de *f* no ponto *x,y* é proporcional ao brilho ou nível de cinza da imagem (ROSA, 2009). Na Figura 1.1 observa-se a distribuição dos *pixels* na imagem digital.

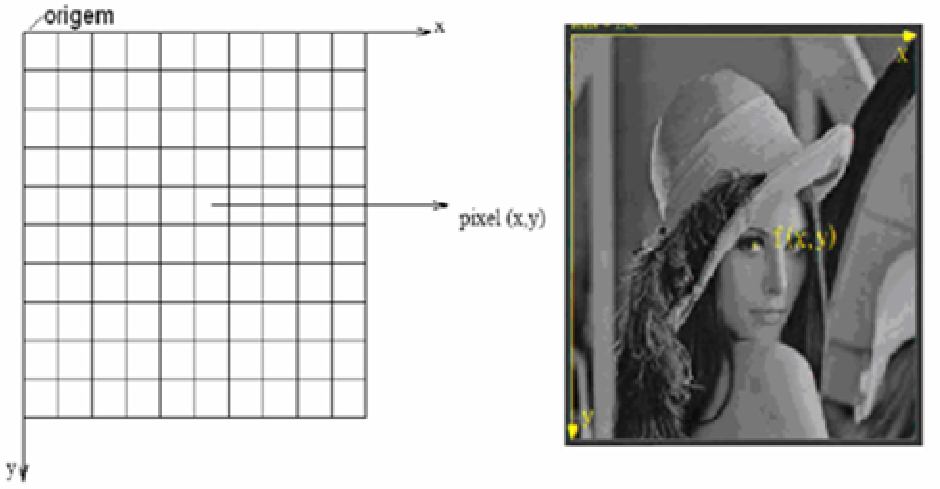


Figura 1.1 – Distribuição dos *pixels* em uma imagem.
Fonte – Adaptado de ROSA, 2009.

Uma função *f(x,y)* precisa ser digitalizada tanto espacialmente quanto em amplitude, para ser adequada para processamento computacional. A digitalização das coordenadas espaciais *(x,y)* é denominado amostragem e a digitalização da amplitude é chamada quantização em níveis de cinza (GONZALEZ, 2007).

1.1 Amostragem

O elemento básico de uma imagem de computador é o *pixel*, uma imagem é composta de *pixels* distribuídos em uma matriz retangular, onde cada *pixel* representa não apenas um ponto na imagem, mas uma região retangular, o valor associado com o *pixel* deve representar a intensidade média na célula correspondente para aquela região. Amostragem é o processo pelo qual uma imagem contínua se torna discreta, ou seja, é divida em diversas

amostras de tamanho definido, como uma cena captada por uma câmera, é decomposto em um conjunto de *pixels* (MYLER, 1993).

O valor associado com o *pixel* deve representar a intensidade média na célula correspondente para aquela região. O *pixel* também não tem um tamanho definido, já que por definição o tamanho do *pixel* se dá pelo número de linhas *M* e colunas *N* que a imagem é dividida (GONZALEZ, 2007).

A Figura 1.2 mostra uma mesma imagem representada com um número diferente de *pixels*. A Figura 1.2a e a Figura 1.2b representam a mesma imagem com o tamanho do *pixel* diferentes, a primeira em maior número que a segunda e de forma subseqüente comparados as dimensões da Figura 1.2c e Figura 1.2d.

Figura 1.2 – Imagem digital (a, b, c e d) com diferentes números de *pixels*.
Fonte – RUSS, 1999.

Na Figura 1.2c e Figura 1.2d as imagens são pobres na resolução espacial e as descontinuidades dos valores de cinza nas bordas entre os *pixels* aparecem prejudicando a visualização da imagem (RUSS, 1999).

Diminuindo o tamanho dos *pixels*, o efeito torna-se menos pronunciado como na Figura 1.2b, até o ponto onde se começa a ter a impressão de uma imagem espacialmente contínua como mostrado na Figura 1.2a. Isto acontece quando os *pixels* tornam-se menores do que a resolução espacial do sistema visual humano. Portanto o tamanho do *pixel* deve ser menor do que a resolução espacial do sistema visual de uma distância nominal do observador (RUSS, 1999). Isto significa que para uma boa imagem digital o tamanho do *pixel* deve ser menor do que as escalas mais finas dos objetos que se deseja estudar.

1.2 Quantização

Na quantização de uma imagem monocromática, podemos atribuir um código de comprimento uniforme para cada amostra da imagem. Esse valor *M* é a intensidade ou a quantidade de níveis de cinza diferentes que o *pixel* pode assumir na imagem digital monocromática. Se *m* é o número de bits de código atribuído a cada amostra, a variável *M* é igual a 2^m, mais conhecida como resolução de quantização. O número de níveis *M* é escolhido de modo a que a qualidade da imagem resultante é aceitável para os olhos humanos (ACHARYA, 2005).

A Figura 1.3 mostra imagens quantizadas com diferentes valores da escala de cinza. A Figura 1.3a com m = 16 e a Figura 1.3b com m = 8, apresentam uma resolução que é o suficiente para dar-se a ilusão de uma mudança contínua nos valores de cinza.

Figura 1.3 – Imagem (a, b, c e d) quantizada com diferentes níveis de cinza.
Fonte – RUSS, 1999.

Pode-se ver que um número baixo de valores na escala de cinza, se deve ao menor número de *bits*, como na Figura 1.3c (m = 4) e na Figura 1.3d (m = 2), gera bordas falsas e dificultam o reconhecimento de objetos que mostram pouca variação nos valores de cinza (JÄHNE, 2002). Quando m = 1 existem apenas 2 níveis de quantização e a imagem é chamada de binária (ACHARYA, 2005).

Geralmente, os dados da imagem são quantizados em 8 bits ou 256 valores de cinza, onde cada *pixel* é quantizado em um *byte*. Este tamanho de um *byte* é mais adequado processamento em computadores pessoais (ACHARYA, 2005).

1.3 Tamanho da Imagem Digital

Para determinar o tamanho do arquivo de uma imagem é necessário definir o número de *pixels* e a resolução de quantização que a imagem possui. Para determinar o tamanho da

imagem digital (T) deve-se multiplicar o número de *pixels* que formam a imagem pela quantidade de bits da resolução de quantização, como pode ser observado na Equação 1.1, onde *L* é o número de linhas e *C* é o número de colunas, que determinam a quantidade de *pixels*, e *m* é o número de bits da resolução de quantização.

$$T = L * C * G \tag{1.1}$$

Para determinar quantidade de níveis discretos de cinza G deve-se utilizar a Equação 1.2.

$$G = 2^m \tag{1.2}$$

Desta forma, uma imagem de 128x128 *pixels* com 64 níveis (6 *bits*) de cinza requer 98.304 *bits* para armazenamento (GONZALEZ, 2007).

1.4 Aquisição de Imagens

Uma imagem para ser processada pelos sistemas eletrônicos, precisa ser capturada através de sensores para converter os dados em sinais elétricos, para posteriormente serem digitalizados e processados pelos sistemas computacionais.

Dois dispositivos que tem um maior destaque nesta área são, o sensor tipo CCD (*Charge Coupled Devices*) e o sensor tipo CMOS (*Complementary Metal Oxide Semiconductor*), na câmera de vídeo o papel de ambos é o mesmo. Os *pixels* são compostos por milhares de células minúsculas, e estas células transformam-se ao receberem mais ou menos luz, através de um efeito descoberto por Einstein (que lhe valeu o prêmio Nobel de 1905, o efeito fotoelétrico), as células geram energia elétrica ao receberem energia luminosa. Esta energia é proporcional à intensidade com que são iluminados os elementos fotossensíveis e é esse fato que propicia seu emprego na geração de imagens, pois como se sabe as imagens só são percebidas por nós porque possuem contraste, ou seja, partes claras misturadas com partes menos claras ou mais escuras que se misturam para formar os contornos e nuances dos objetos e pessoas (BAPTISTA, 2009).

As câmeras CCD empregam uma matriz densa de elementos fotossensíveis, difundidas em uma pastilha de silício retangular. A imagem a ser digitalizada está focada no

wafer[4] usando um sistema de lentes, os dados referentes a intensidade e localização dos elementos são retirados e decodificados por circuitos eletrônicos especiais (MYLER, 1993).

O CCD utiliza um chip para o registro de imagens, *pixel a pixel*, de um painel paralelo contendo minúsculos capacitores, a seguir deixa-se o painel com as células fotoelétricas exposto à luz capturando uma dada imagem nele projetada pelas lentes da câmera. Partes mais claras provocam a geração de mais energia elétrica, partes mais escuras, de menos, essas cargas acumuladas são denominada sinal de vídeo, representando a imagem registrada pela câmera (BAPTISTA, 2009). Uma visão esquemática de um sistema de câmera é representado na Figura 1.4.

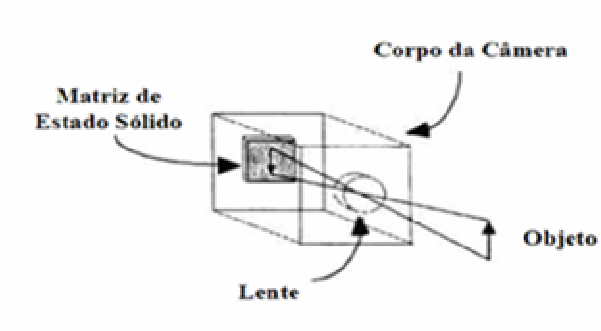


Figura 1.4 – Representação de uma câmera CCD.
Fonte – MYLLER, 1993.

Na Figura 1.5 é observa-se o formato de um *chip* CCD, este tipo de tecnologia permite colocar toda uma câmera dentro de um único *chip* (JÄHNE, 2002).

[4] Wafe- Wafer é uma fina fatia de material semicondutor na qual microcircuitos são construídos pela dopagem (por, exemplo difusão ou implementação de íons), separação química com ácidos, e deposição de vários materiais.

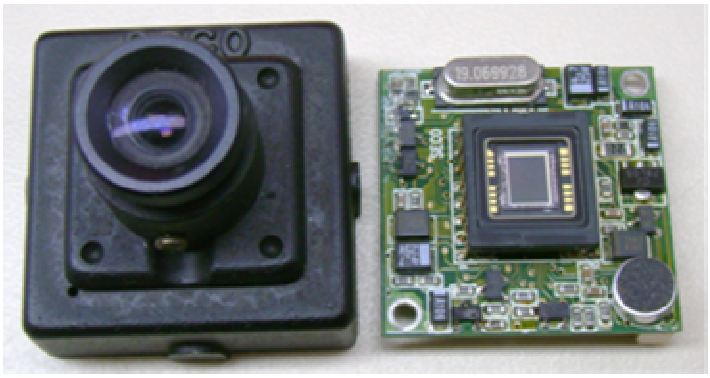


Figura 1.5 – Foto da câmera apresentando o sensor CCD.
Fonte – Autor, 2010.

Sensores CCD proporcionam uma melhor imagem em baixa luminosidade, já os sensores CMOS consomem menos energia do que os sensores CCD, com isso dispositivos portáteis que utilizam essa tecnologia tem uma grande vantagem em relação duração das pilhas e baterias que alimentam estes dispositivos. Uma vez que os sensores CMOS usam a mesma plataforma de produção que a maioria dos microprocessadores e *chips* de memória, eles são de mais fácil fabricação e mais rentáveis do que os sensores CCD. Sensores de imagem CMOS requerem uma única potência para a operação de somente 2-50 miliwatts por saída de *pixel* (XILINX, 2005a).

1.5 Formatos de Armazenamento de Imagens Digitais

As imagens para serem armazenadas, são convertidas em arquivos binários com formatos específicos. Geralmente, os arquivos de imagens contem um cabeçalho (*header*) ou campos informativos, que descrevem as características da imagem digital e os dados da imagem. Nestes cabeçalhos estão informações referentes ao número de linhas, número de colunas, número de bits usados, resolução horizontal, resolução vertical, tipo de compressão utilizada para armazenar a imagem, data e hora de aquisição, tipo de sensor que capturou a imagem, e outras informações relevantes (FELGUEIRAS, 2008).

Entre os formatos de armazenamento de imagem digitais mais usados, destacam-se os seguintes:

- BMP (*Windows bitmap*) - Formato gráfico que permite o armazenamento de cores em 24 bits, desenvolvido pela Microsoft (FELGUEIRAS, 2008);

- TIFF (*Tagget Image Fili Format*) - Foi desenvolvido em 1986 pela *Aldus e Microsoft* com o objetivo de padronizar imagens geradas por equipamentos digitais. Armazena imagens *true color* (24 ou 32 bits). É amplamente utilizado por programas de processamento de imagens, e é o formato bastante utilizado para saídas de *scanner*[5] . Utiliza o método de compressão LZW (FELGUEIRAS, 2008);

- GIF (*Grafics Interchange Format*) - Foi criado pela *Compuserve* para transmissão de imagens do tipo *bitmap* para a internet. As imagens deste formato são comprimidas e codificadas pela especificação LZW (sem perda de informação), suporta 8 bits por *pixel* (FELGUEIRAS, 2008);

- PCX (*Zoft file Format*) - Foi criado para o *Paintbrush* da *Microsoft* (na época do DOS[6]). A versão atual suporta cores em 24 bits(FELGUEIRAS, 2008);

- RAW (Bruto ou rústico) - É um formato onde os *pixels* são armazenados em formato binário(com 8, 16 ou 24 bits). Neste formato somente os dados são escritos no arquivo, sendo necessário que o usuário informe parâmetros relativos a imagem, tais como

---

[5] Scanner- É um periférico de entrada responsável por digitalizar imagens, fotos e textos impressos para o computador.
[6] DOS -Sigla para *Disk Operating System* ou sistema operacional em disco).

número de linhas, número de colunas, e número de bits usados na codificação da imagem (FELGUEIRAS, 2008);

- JPEG(*Joint Photographic Experts Group*) - JPEG é um método comumente usado para comprimir imagens fotográficas. O grau de redução pode ser ajustado, o que permite a você escolher o tamanho de armazenamento e seu compromisso com a qualidade da imagem. Tipicamente, o JPEG atinge uma compressão de 10 para 1 (FELGUEIRAS, 2008).

1.6 Imagem Colorida

O uso de cores em processamento de imagens tem dois fatores que se destacam. Primeiramente a cor é um poderoso descritor que simplifica a identificação do objeto e a extração da cena. Em segundo lugar o olho humano é capaz de discernir milhares de tons e intensidades de cores (GONZALEZ, 2007).

Em 1666, Isaac Newton descobriu que um feixe de luz solar ao passar por um prisma de vidro, o feixe de luz emergente consiste de um espectro contínuo de cores variando do violeta ao vermelho, como pode ser visto na Figura 1.6. O espectro de cores pode ser dividido em seis amplas regiões: violeta; azul; verde; amarelo; laranja e vermelho. Nenhuma cor no espectro termina abruptamente, mas cada cor mistura-se suavemente com a próxima, como pode ser observado na representação da Figura 1.6 (GONZALEZ, 2007).

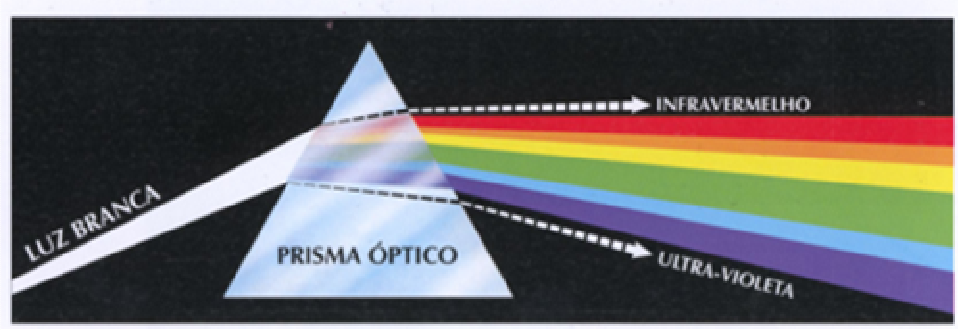


Figura 1.6 – Representação do feixe de luz solar ao atravessar um prisma.
Fonte – GONZALEZ, 2007.

As cores que o ser humano observa em um objeto são determinadas pela natureza da luz refletida neste objeto, portanto todas as cores são vistas como combinações variáveis das três cores primárias: vermelho (R, do inglês *red*); verde (G, do inglês *green*) e azul (B, do inglês *blue*). As três cores primárias são misturadas para produzir diferentes matizes[7] como pode ser visto na Figura 1.7 (GONZALEZ, 2007).

Figura 1.7 – Representação da mistura de cores.
Fonte – Adaptado de GONZALEZ, 2007.

No modelo RGB, cada cor aparece nos seus componentes espectrais de vermelho, verde e azul. Este modelo baseia-se num sistema de coordenadas cartesianas, na Figura 1.8 observamos a representação do cubo de cores RGB nos três cantos: cianto; magenta e

[7] Matizes - É é uma das três propriedades da cor que nos permite classificar e distinguir uma cor de outra através de termos como vermelho, verde, azul.

amarelo. Se encontram nos outros dois cantos: preto, na origem e o branco está no canto mais distante da origem. O modelo de cores RGB consiste em três planos de imagens independentes, um para cada cor primária. Os pontos ao longo da diagonal principal têm valores de cinza desde preto na origem até branco no ponto (1,1,1). Quando alimentadas num monitor RGB, estas três cores combinam-se sobre a tela de fósforo para produzir uma imagem de cores composta (GONZALEZ, 2007).

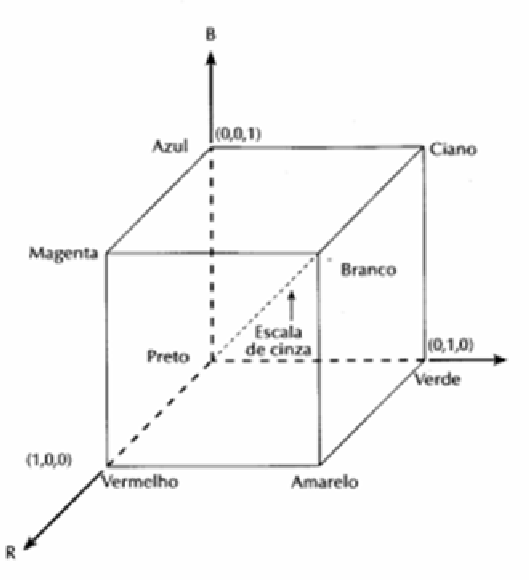


Figura 1.8 – Representação do cubo de cores RGB.
Fonte – GONZALEZ, 2007.

2 IMAGEM DE VÍDEO

A imagem é desenhada na tela de uma televisão ou monitor de computador da esquerda para a direita e de cima para baixo. Horizontalmente a imagem é desenhada no visor uma linha de cada vez controlada pelo pulso de sincronização horizontal (H-Sync), que no final de cada linha informa ao circuito de varredura para repassar para a borda esquerda da tela e em seguida iniciar a digitalização da linha seguinte.Começando no topo, todas as linhas na tela são digitalizados desta forma. Um conjunto completo de linhas formam um quadro.

O pulso que informa ao circuito de varredura para repassar para o topo da tela e começar a digitalizar o próximo quadro, ou imagem, é chamado de pulso de sincronização vertical (V-Sync). Esta seqüência é repetida em um ritmo rápido o suficiente para que as imagens dêem a impressão de movimento contínuo (MAXIM, 2001).

Cada pulso de sincronismo horizontal informa o final de uma linha, existe uma porção da forma de onda do sinal de vídeo responsável por fazer o retraço horizontal, esse sinal faz o apagamento do traço e o retorno da borda direita para borda esquerda da tela, permitindo com que uma nova linha possa ser desenhada na tela, este intervalo é denominado de apagamento horizontal, do inglês *horizontal blanking interval* (MAXIM, 2001).

Ao final de cada quadro um circuito de retraço vertical se encarrega de acionar o apagamento do traço até o topo da tela para iniciar um novo quadro. Este retraço vertical é chamado de apagamento vertical, do inglês *vertical blanking interval* (MAXIM, 2001).

2.1 Varredura de Vídeo

Existem dois modos diferentes de desenhar a imagem na tela da TV, o entrelaçado e o progressivo.Os sinais de televisão normalmente são entrelaçados, e os sinais de monitor de computador são tipicamente progressivo (não entrelaçado). Estes dois formatos são incompatíveis entre si, um teria de ser convertido para o outro antes de qualquer tratamento comum pudesse ser feito. A técnica de varredura entrelaçada divide um quadro completo em duas metades, onde cada imagem, referida como frame, e dividida em duas sub-imagens separadas, referidas como campos, ou *fields* em inglês. Dois *fields* constroem um frame. Um campo, chamado de ímpar rastreia as linhas ímpares da imagem de vídeo. O outro campo verifica as linhas pares da imagem de vídeo (NATIONAL, 2006). A Figura 2.1 representa linhas de varredura entrelaçada na tela.

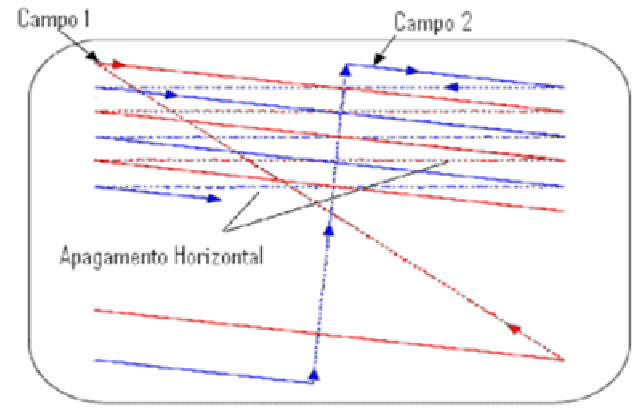


Figura 2.1 – Representação da varredura entrelaçada.
Fonte – Adapatdo de NATIONAL, 2006.

É possível também observar que existem as linhas de apagamento horizontal que não são visíveis e que servem para posicionar o sinal no começo da próxima linha do mesmo campo para a varredura. Esse método de gerar um quadro a partir da soma das linhas dos dois campos originou o nome entrelaçado.

No sistema progressivo o frame é desenhado por inteiro pela varredura seqüencial das linhas. A Figura 2.2 mostra o método de varredura progressiva. A varredura percorre todas as linhas de cima para baixo e da esquerda para a direita para gerar o frame. Aqui também se encontram as linhas de Apagamento Horizontal que posicionam a varredura do final de uma linha sempre para o início da próxima linha (MAXIM, 2001).

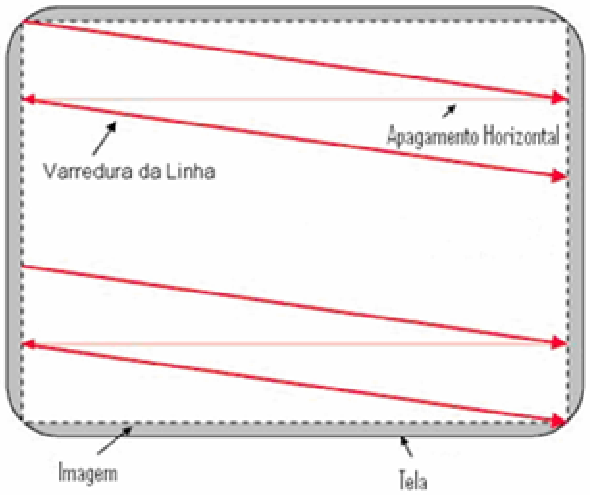


Figura 2.2 – Método de varredura progressiva.
Fonte – MAXIM, 2001.

O sistema de varredura progressivo é encontrado em monitores de computador sistemas digitais de TV, como por exemplo, NTSC progressivo, monitores de plasma e LCD.

Esse sistema tem como vantagem uma maior definição vertical e não sofrer efeitos de aliasing[8] (MAXIM, 2001).

2.2 Formatos de Vídeo

Na época em que a segunda guerra mundial chegava ao fim, haviam pesquisas mostrando que a transmissão de imagens a longa distância era viável, então formou-se um comitê para propor um padrão para essas transmissões. As imagens seriam propagadas através de ondas de rádio, tendo a *US Federal Communications Commission* dividido já em 1945 um trecho do espectro de ondas VHF (*Very High Frequency*) em 13 canais, determinando com isso um tamanho máximo de faixa de transmissão (banda) para cada um desses canais (BAPTISTA, 2009).

Os formatos de vídeo para televisão variam de país para país. No início dos anos 50 nos EUA foi desenvolvido o Comitê Nacional de Sistemas de Televisão, do inglês *National Television Standards Commitee* (NTSC). Foi estabelecido que a freqüência de troca dos quadros nas imagens seria de 60 quadros por segundo, igual aos 60 Hz utilizados na corrente elétrica nos EUA, porque isso facilitava o controle do sincronismo dessas imagens no aparelho receptor, portanto a eletricidade que alimentava o televisor ajudava a realizar assim essa tarefa. Mas com a implementação da televisão a cores, o formato NTSC teve a sua freqüência reduzida de 60 Hz para 59.94 Hz, em virtude do acréscimo da sub-portadora que carrega as informações de cor gerar flutuação de áudio durante a transmissão (BAPTISTA, 2009).

---

[8] Aliasing – Ruído causado pela distorção do espectro do sinal de tempo contínuo original.

A imagem seria desenhada através de linhas, uma abaixo da outra em um total de 525 por quadro. Destas, 480 conteriam a porção efetivamente visível e as demais conteriam códigos para orientar o tubo de imagem em sua tarefa de desenhar uma linha, voltar e desenhar a linha de baixo, e após desenhar todo o quadro de retornar ao início para o desenho do próximo quadro. Como a largura de banda disponível não era suficiente para transmitir uma imagem completa, com todas as linhas, 60 vezes por segundo, optou-se por dividi-la em 2 partes, uma com as linhas pares e outra com as ímpares, mostradas alternadamente, a cada 29,97Hz utilizando o sistema de varredura entrelaçada. Neste sistema, cada "metade" da imagem (linhas pares ou linhas ímpares) recebe o nome de campo, e a freqüência com que são desenhados na tela não é perceptível para o espectador (BABTISTA, 2009).

No final dos anos 60 surgia na Alemanha o padrão PAL, Linhas Alternadas em Fase, do inglês *Phase Alternate Lines*, propondo-se a corrigir vários problemas existentes no NTSC referentes à reprodução de cor. A técnica usada era inverter a fase do sinal de cor para linhas alternadas na tela, daí o nome dado ao padrão. A reprodução de cores resultou mais precisa do que no padrão NTSC. Neste país a corrente elétrica alternada era gerada em 50 Hz, por isso a freqüência de mudança dos campos foi especificada como 50 Hz e não 60 Hz, sendo as imagens transmitidas a 25 Hz ao invés de 30 Hz. Esta redução na cadência de mudança das imagens fez com que essas mudanças ficassem um pouco mais "visíveis" do que no padrão NTSC, para compensar a perda na qualidade visual, a quantidade de linhas na tela foi ampliada, mostrando 625 linhas (BAPTISTA, 2009).

O padrão PAL-M, utiliza 60 Hz, portanto as imagens são transmitidas com freqüência de 30 Hz, assim a imagem aparenta ser mais nítida e definida (BAPTISTA, 2009).

Na França foi desenvolvido o padrão SECAM, Cores Sequênciais com Memória , em francês *Systeme Electronique Couleur Avec Memoire* semelhante em alguns aspectos ao PAL e também se propondo a suprir as deficiências do NTSC, utiliza também 25Hz para exibir as imagens entrelaçadas. As diferenças entre PAL e SECAM são tão pequenas que a conversão entre os mesmos pode ser feita por um simples decodificador e a maioria dos receptores PAL é capaz de exibir imagens (porém em preto e branco) transmitidas em SECAM (BAPTISTA, 2009).

O padrão NSTC apresenta 525 linhas por quadro, porém o número de linhas por quadros visíveis é de 480 linhas, porque as restantes contêm sinais de sincronismo, retraço e outros dados (NATIONAL, 2006). A tabela 1 representa características dos padrões de sinal de vídeo.

Tabela 1 – Características dos padrões de sinal de vídeo.

Características	NTSC	PAL	SECAM	PAL M
Linhas /quadros	525/60	625/50	625/50	525/60
Freqüência Horizontal (KHz)	15.734	15.625	15.625	15.750
Freqüência Vertical (Hz)	59,94	50	50	60
Sub-portadora de Cor (MHz)	3.579545	4.433618	-	3.575611
Freqüência da Portadora de áudio (MHz)	4.5	5.5	5.5	4.5

Fonte – NATIONAL ,2006.

2.3 Interfaces de Vídeo

As interfaces de vídeo podem ser separadas e classificadas de acordo com a codificação que utilizam para combinar informações de vídeo. Cada interface de vídeo possui conectores e características específicos, e a qualidade do sinal das imagens está relacionada a complexidade da codificação utilizada. Dentre as interfaces de vídeo que se destacam estão: Vídeo Composto; Vídeo Separado e Vídeo Componente.

2.3.1 Vídeo Composto

O sinal de vídeo composto é o mais utilizado em interface de vídeo analógico. Também pode ser denominado "Sinal de Vídeo Composto em Banda Base", do inglês *Composite Video Baseband Signal* (CVBS). Ele combina as informações de brilho (*luma*), de cor (*chroma*), referência de cor (*Color Burst*) e sincronismos (HSYNC e VSYNC) em apenas um cabo. O conector usado é geralmente um conector RCA com impedância de 75Ω, geralmente de cor amarela (MAXIM, 2001).

Uma forma típica de onda de sinal NTSC de vídeo composto é representado na Figura 2.3.

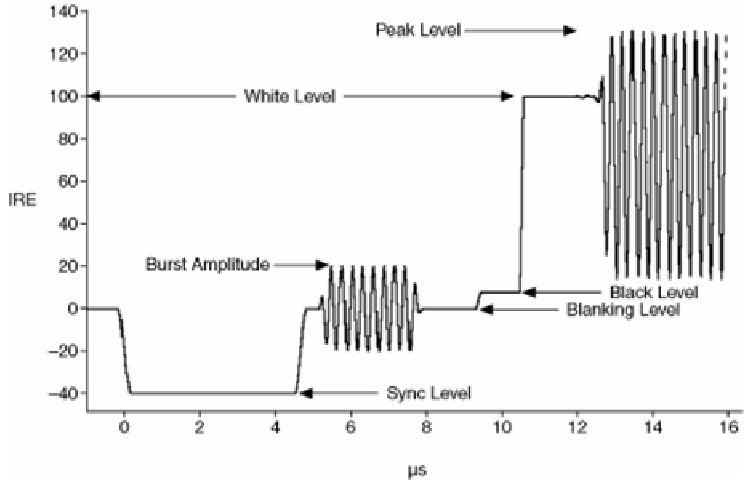


Figura 2.3 – Representação do forma de onda do sinal de vídeo composto.
Fonte – NATIONAL, 2006.

A unidade de medição de amplitude e dada em IRE, sigla do Instituto de Engenheiros de Rádio, do inglês *Institute of Radio Engenieers* (IRE) é uma unidade arbitrária em 140 IRE = 1Vpp. A região indicada por *Sync Level* recebe os pulsos de HSYNC que devem ter uma amplitude de -40 IRE. O intervalo de apagamento horizontal deve estar em 0 IRE, como indica a representação da Figura 2.3, denominado nível de apagamento (*Blanking Level*). Os sinais de referência de cor (*Color Burst*) devem ter uma amplitude de 40 IRE de pico a pico. Já a região ativa de *pixels*, é gerada de acordo com a informação de brilho que cada *pixel* na linha, o sinal pode variar do *Black Level* que é 7,5 IRE até *White Level* que é de 100 IRE. O *Peak Level* indica 120 IRE, que é tolerância máxima no NTSC dos sinais das componentes de cor que são moduladas sobre as informações de vídeo (MAXIM, 2001).

2.3.2 Vídeo Separado

A denominação vídeo separado vem do inglês *Separate Video* (S-Video), também conhecida como Y/C, é um sinal de vídeo que transmite brilho (*Luma*) que é o sinal Y, e a cor (*chroma*), que é o sinal C, em dois conjuntos separados de fios. O conector é um Mini-

DIN que se assemelha a uma pequena versão de um conector de teclado do tipo PS2 (MAXIM, 2001).

O conector Mini-DIN de 4 pinos, como é ilustrado na foto da Figura 2.4, é o mais comum de vários tipos de conectores S-Video, onde o número 1 indica o sinal de GND de brilho, o número 2 indica o sinal de GND de cor, o número 3 indica o sinal de brilho e o número 4 é o sinal de cor. Seu terminal também possui uma impedância de 75Ω.

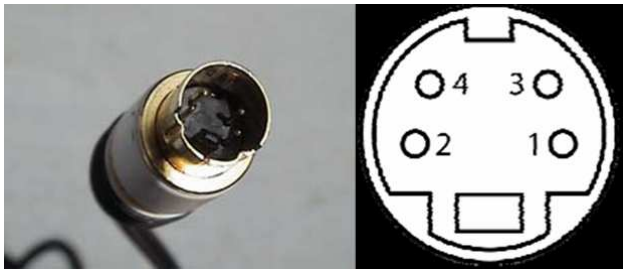


Figura 2.4 – Foto e pinagem do conector fêmea Mini-dim.
Fonte – HUYGHE, 2007.

2.3.3 Vídeo Componente

A interface de vídeo componente possui um mínimo de codificação, o sinal de vídeo está quase no seu formato nativo, se obtém desta forma uma melhor qualidade na imagem, porém necessita-se de uma maior largura de banda ou um número maior de fios para a comunicação. De acordo com a sua aplicação podem apresentar diferentes tipos de configurações.

A interface de vídeo componente utilizam três fios de cores padrão:vermelho; verde e azul (RGB). O formato RGB (*Red, Green e Blue*) normalmente utilizadas em aplicações

para computador, enquanto que diferença de cores dos sinais são geralmente utilizados em aplicações para televisão.

A teoria por trás dessa combinação é que cada uma das base de R, G, B e componentes podem ser derivados a partir destes sinais de diferença (MAXIM, 2001). Algumas variações dos sinais de vídeo componente são descritas a seguir:

- Y, B–Y, R–Y representa o brilho e sinais de diferença de cor;

- Y, Pr, Pb são versões de Y, B–Y, R–Y. Normalmente encontradas em equipamentos para o consumidor final;

- Y, Cr, Cb são sinais digitais equivalentes a Y, Pr, Pb. Algumas vezes utilizados incorretamente no lugar de Y, Pr, Pb;

- Y, U, V não e uma interface padrão. São sinais quadráticos intermediários, utilizados na formação de sinais Y/C e composto. Algumas vezes utilizado incorretamente como interface vídeo componente.

As interfaces de vídeo componente são usadas em aplicações para a televisão e identificados por três terminais do tipo RCA ou BNC nas cores verde (Y), azul (PB) e vermelho (PR), conforme ilustra a foto da Figura 2.5.

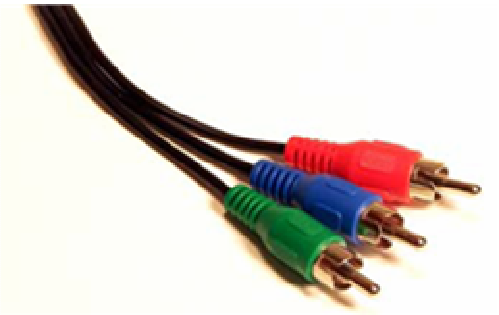


Figura 2.5 – Foto dos terminais de vídeo componente.
Fonte – HUYGHE, 2007.

2.3.3.1 Interface de Sinais Computacionais

O tipo de formato utilizado em sistemas computacionais é formado pelo R (vermelho), G (verde), B (azul) e S (sincronismo) ou R, G, B, H (sincronismo horizontal) e V (sincronismo vertical), onde os sinais de sincronismos são totalmente separados, e cada cor carrega sua informação de brilho. Os cinco sinais, R, G, B, H e V, são enviados por meio de um cabo blindado. O conector quase sempre e de 15 pinos do tipo D-Sub. Em algumas vezes os sinais de sincronismo são misturados aos sinais RGB, tipicamente ao sinal de componente verde. Em alguns casos raros os sinais de sincronismo podem ser transmitidos em conjuntos como os componentes de sinal vermelho ou azul (MAXIM, 2001).

2.4 Monitores de Vídeo

Para ser possível interagir com computadores e dispositivos eletrônicos, se faz necessário o uso cada vez mais de monitores e displays. Atualmente, temos em uso basicamente quatro tecnologias de monitores, CRT, LCD, PLASMA e OLED, como o Trabalho descrito nesta Monografia utiliza apenas monitores do tipo CRT e LCD, os monitores de PLASMA e OLED estão fora do escopo de estudo.

2.4.1 CRT

Em monitores do tipo tubo de raios catódicos (CRT), do inglês *Cathode Ray Tube* a imagem é formada pela emissão de elétrons através de um "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 exibido é controlada pelo nível de voltagem introduzido pelo sinal de vídeo. O controle do caminho percorrido pelo feixe de elétrons é feito através de placas defletoras, horizontal e vertical, que produzem um campo magnético determinando o caminho que o elétron deve percorrer. Estas placas defletoras são responsáveis pela identificação dos sinais de sincronismo.

Um controlador de vídeo é responsável por gerar os 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 as linhas percorrer toda a tela (CHU, 2008).

Sendo assim, um circuito digital de controle dos sinais de sincronismos requer que sejam definidas de forma objetiva, a disposição da parte visível da tela no monitor. Monitores CRT, durante o sincronismo horizontal, apresentam uma pequena borda preta ao redor da imagem, definidas a partir dos valores de borda da esquerda (*back porch*) e de borda da direita (*front porch*). Durante o sincronismo vertical também são verificadas regiões onde o sinal de vídeo deve ser desativado, regiões chamadas *de bottom border* e *top border* (CHU, 2008).

Estes dados, além de todos os elementos que compõem uma varredura horizontal e vertical, são representados na Figura 2.6. Nela é ilustrado um diagrama dos ciclos de

sincronismo horizontal e vertical para dispositivos CRT com resolução de 640x480 pixels, baseado no padrão VGA.

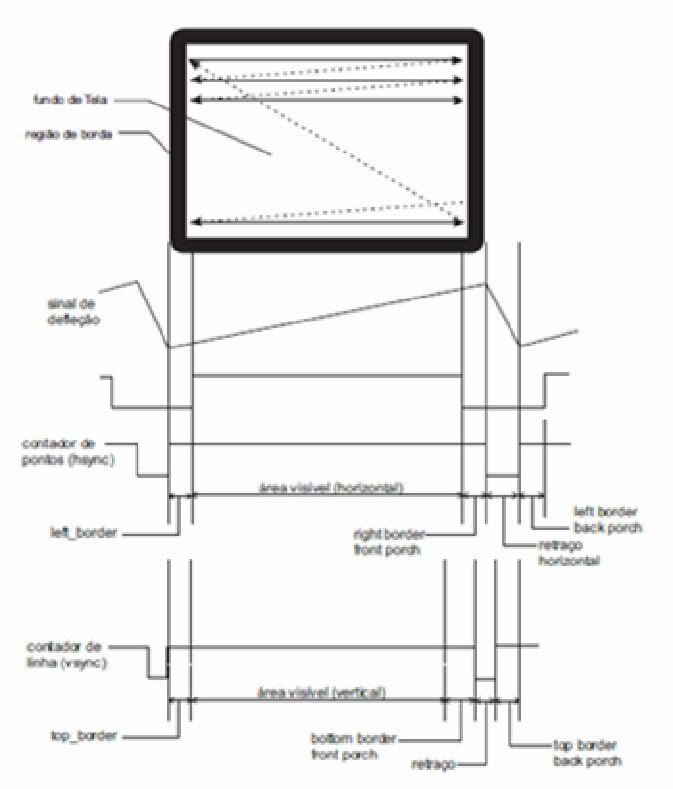


Figura 2.6 – Representação das características da varredura horizontal e vertical.
Fonte – CHU, 2008.

2.4.2 LCD

Telas de cristal líquido, do inglês *Liquid Crystal Display* (LCD) são fabricadas sobre uma placa de vidro, utilizando camadas de silício amorfo depositadas sobre ela, a tela de LCD de matriz ativa possui um transistor para cada ponto da tela e um pequeno sulco onde é depositado o cristal líquido, este cristal líquido quando recebe corrente elétrica tem sua estrutura molecular alterada, se opondo a passagem da luz. Em seu estado normal, o cristal líquido é transparente, mas ao receber carga elétrica torna-se opaco, impedindo a passagem de luz. Cada *pixel* é formado por três pontos, e cada ponto é controlado por um transistor

responsável por aplicar a correta tensão para cada cor primária e gerar a correta tonalidade (MORIMOTO, 2007).

Existem atualmente duas tecnologias de fabricação de telas de LCD, conhecidas como matriz passiva, *Dual-Scan Supertwist Nematic* (DSTN) e matriz ativa, *Thin-Film Transistor* (TFT). As telas de matriz passiva apresentam um ângulo de visão mais restrito, e um tempo maior para a imagem ser atualizada, do que as telas com matriz ativa. Atualmente todos os monitores de LCD são de matriz ativa, a ponto de serem genericamente chamadas de "telas TFT". O TFT é uma técnica de fabricação utilizada para construir os transistores sobre o vidro do monitor, isso é possível através de um processo de deposição onde é criada uma fina camada de silício amorfo sobre o substrato de vidro, contudo esta camada de silício não é muito transparente, desta forma é utilizado um processo de litografia para criar a estrutura do transistor, para deixar apenas a parte ocupada pelo transistor sem excessos é usado um processo de banho químico. Da mesma forma dos processadores, o processo é repetido várias vezes (pelo menos 5), utilizando máscaras de litografia diferentes, de forma a criar as diversas camadas que formam os transistores, a Figura 2.7 ilustra um transistor após o processo de litografia aplicado sobre uma das camadas de silício amorfo e um transistor pronto. Uma tela de matiz ativa com resolução de 1024x768 possui mais de 2 milhões de transistores (MORIMOTO, 2007).

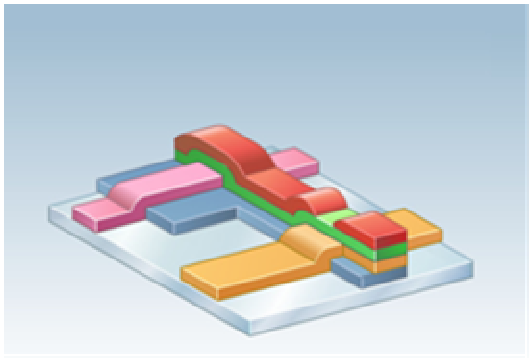


Figura 2.7 – Transistor após processo de litografia e banho químico.
Fonte – AU OPTRONICS CORP. , 2007.

As telas LCDs usados em monitores utilizam uma fonte de luz localizada na parte de trás do painel, que pertence a uma unidade de *backlight*. Figura 2.8 ilustra as partes que compõem uma tela de LCD, onde se observa o sistema de iluminação composto por lâmpadas de catodo frio que utiliza um inversor para gerar a energia suficiente à lâmpada. Atualmente os monitores utilizam diodo emissor de luz (LED), do inglês *Light Emitting Diode*, para iluminar a tela (MORIMOTO, 2007).

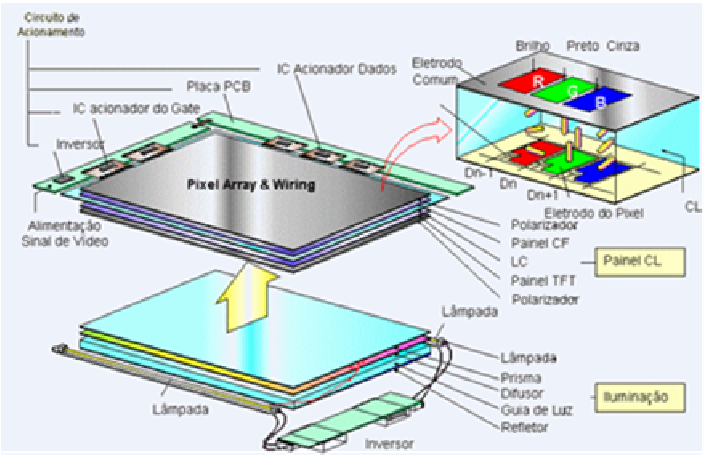


Figura 2.8 – Esquema de iluminação de telas LCD.
Fonte – WIKIPEDIA.

2.5 Resolução de Vídeo

O padrão desenvolvido pela IBM em 1987, conhecido como Matriz Gráfica de Vídeo, do inglês *Video Graphics Array* (VGA) é o termo usado freqüentemente para se referir a um resolução de 640 x 480, independentemente do hardware utilizado para produzir a imagem. Também pode se referir ao conector de 15 pinos tipo D-sub, a Figura 2.9 mostra a pinagem deste conector, que ainda é muito freqüente para sinais de vídeo analógico (HUYGHE, 2007).

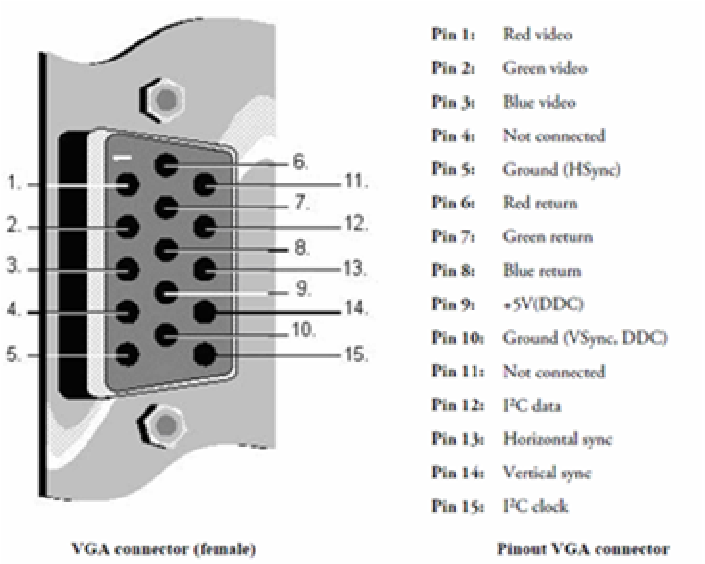


Figura 2.9 – Esquema do conector 15 pinos tipo D-sub e respectiva pinagem.
Fonte – HUYGHE, 2007.

A resolução é expressa em número de *pixels* horizontais por *pixels* verticais. Ela pode ser expressa por uma lista de padrões e respectivas resoluções conforme a Tabela 2.

Tabela 2 – Tabela de padrões de resoluções de vídeo.

Padrões de Resoluções de Vídeo		
Padrão	Pixels	Linhas
QVGA	320	240
VGA	640	480
SVGA	800	600
XGA	1024	768
WXGA	1280	800
WXGA+	1440	900
SXGA	1280	1024
SXGA+	1400	1050
UXGA	1600	1200
WSXGA	1680	1050
WUXGA	1920	1200
WQXGA	2560	1600

Fonte – XILINX,2005a.

2.5.1 Sinais para Resolução VGA

Para a exibição da imagem em um monitor de vídeo são necessários 5 sinais ativos:

- Horizontal Sync- sinal digital, para sincronismo horizontal do vídeo;
- Vertical Sync- sinal digital, para sincronismo vertical do vídeo;
- R- sinal analógico (0-0,7v), para controle da cor;
- G; sinal analógico (0-0,7v), para controle da cor;
- B; sinal analógico (0-0,7v), para controle da cor.

O Sinal de Cor é formado por três sinais analógicos que variam de 0 V à 0,7 V. Cada sinal analógico corresponde a uma cor primária (Azul, Verde, Vermelho). Os níveis

analógicos destes sinais especificam a intensidade de cada uma destas core, variando entre ausência da cor (0 V) à brilho máximo da cor (0,7 V).

Analisando os sinais de um controlador VGA, existem além dos sinais de sincronismo hsync e vsync, existe o apagamento para o retraço horizontal e vertical. Assim nas bordas, durante o retraço o sinal de vídeo deve ser desativado. No retraço horizontal, o sinal de vídeo deve ser desativado na região em que o feixe de elétrons retorna para a borda esquerda, da mesma forma no retraço vertical, o sinal deve ser desligado na região em que o feixe retorna para o topo da tela (CHU, 2008).

Alterando os níveis analógicos do sinal RGB, todas as outras cores são geradas. O processo de exibição da tela começa no canto superior esquerdo e mostra um *pixel* de cada vez, da esquerda para a direita. Ao final de uma linha, é incrementado uma linha, para endereçar a próxima coluna, depois que todas as linhas da tela for varrida, o processo de atualização ocorre novamente. O sinal de vídeo deve varrer toda a tela pelo menos 60 vezes por segundo para garantir o movimento da imagem e reduzir o flicker[9] , este período de varredura é chamado de taxa de atualização. Na resolução VGA, 640 por 480, usando um clock de 25MHz, e com uma taxa de atualização 60 Hz, isto corresponde a 40 ns por *pixel*. O sinal de sincronização horizontal marca o início e fim de cada linha com pulsos curtos em nível lógico '0' e garante que o monitor visualiza os *pixels* entre as extremidades esquerda e direita da área visível da tela. O sinal de sincronização vertical marca o início e o fim de cada imagem (composta por m linhas). Na Figura 2.10 é possível observar que para cada pulso de

[9] Flicker - É uma impressão de instabilidade da sensação visual, ocorre quando um monitor trabalha com uma freqüência vertical menor que 60 Hz, onde uma sombra parece percorrer constantemente a tela, fazendo com que esta pareça estar piscando.

sincronismo vertical é gerado um retraço vertical, da mesma forma a cada pulso de sincronismo horizontal é gerado um retraço de horizontal (SANCHEZ, 2007).

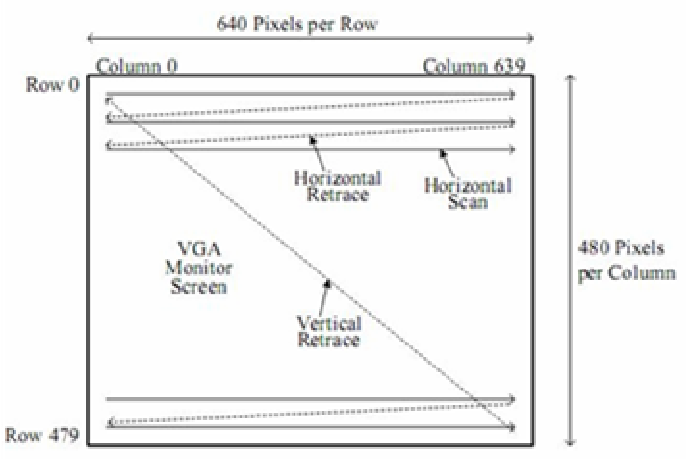


Figura 2.10 – Representação dos sinais de retraço horizontal e vertical.
Fonte – HWANG, 2008.

Durante o período de retraço os dados do *pixel* não são exibidos, os sinais RGB devem ser fixados à cor preta (todos zero) (SANCHEZ, 2007).

A Figura 2.11 ilustra o diagrama de tempos da resolução 640x480. Um ciclo de varredura vertical, dura 16,784 ms, considerando já o período de retraço, que equivale a freqüência de 59,58 Hz. Essa é quantidade de vezes que a imagem é re-desenhada na tela. Cada ciclo de varredura de vertical é composto por 480 ciclos de varredura de horizontal, mais o período de retraço. Ao todo o ciclo de varredura horizontal dura 31,77µS. O que equivale a freqüência de 31,476 Khz. Dentro do período de 31,77µS que forma o ciclo de varredura horizontal, 25,42 µS formam o período de informação de cor. Considerando que cada linha é composta por 640 *pixels*, cada *pixel* terá a duração de 25,42 µS/640. Isso equivale a freqüência de 25,175 MHz (HAWANG, 2008).

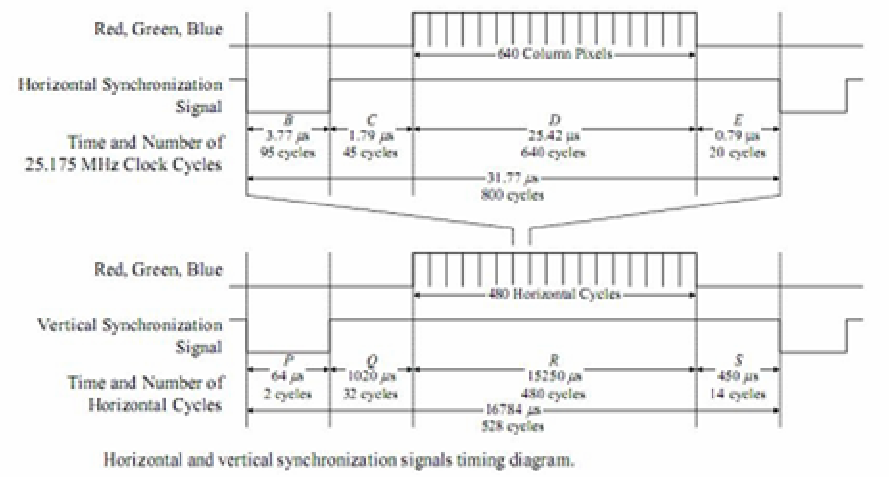


Figura 2.11 – Diagrama de temporização do sincronismo horizontal e vertical, da VGA. Fonte – HWANG, 2008.

3 VHDL

Uma Linguagem de Descrição de Circuitos (HDL), do inglês *Hardware Description Language*, possibilita o intercâmbio de informações referentes ao comportamento de um circuito entre, fabricantes e fornecedores de sistemas de projetos. Outra aplicação de uma linguagem para descrição de circuito pode ser encontrada no teste de circuitos e na síntese do circuito descrito.

Por necessidades do Departamento de Defesa dos Estados Unidos da América (DARPA) foi preciso criar uma ferramenta de projeto e documentação padrão para o projeto VHSIC- *Very High Speed Integrated Circuit*, e desta forma surgiu a Linguagem VHDL. Em 1981 este departamento realizou um encontro de especialistas para discutir os métodos para descrição de circuitos, em 1983 o Departamento de Defesa definiu os requisitos para padronizar uma Linguagem de Descrição de Circuitos, as empresas "IBM", "Texas", "Intermetrics", firmaram um contrato para o desenvolvimento da linguagem e ferramentas. A padronização pelo IEEE- *Institute of Electrical and Electronic Engineer*, teve como base a versão 7.2, cujo processo foi auxiliado pela empresa "CLSI" em 1986. No ano seguinte após revisões propostas por acadêmicos e representantes de industrias e do governo dos Estados Unidos, surgiu o padrão IEEE 1076-1987. Seis anos depois, em 1993 o IEEE aprovou a versão IEEE 1076-1993, cujas alterações não representaram características significativas para síntese de circuitos. Para facilitar o uso da linguagem, foram propostos dois novos padrões, o IEEE 1164 e o IEEE 1076.3, o primeiro define o pacote "Std_logic_1164", e segundo os

pacotes "*Numeric_std*" e "*Numeric_bit*". Estes pacotes em VHDL ou "*package*", é um local para armazenamento de informações de uso comum, tais como tipos de dados, funções etc. O pacote padrão da linguagem define vários tipos de dados, o tipo "*bit*" possui correlação mais direta com o dado a ser manipulado pelo circuito digital, contudo o tipo "*bit*" pode assumir apenas "0" ou "1", o que pode limitar o modelamento. O padrão IEEE 1164 cobre esta lacuna permitindo modelar condições de alta impedância, e casos em que mais de uma porta acionar o mesmo nó. O padrão IEEE 1076.3 facilita a síntese de circuitos, propondo novos tipos de dados e funções, como operações aritméticas de soma e multiplicação em formatos de vetores, possibilitando mais opções para o projetista, já que qualquer dado é convertido em um conjunto de bits no circuito implementado. O padrão IEEE 1076.4 estabelecido em 1995, é um pacote que contém procedimentos e funções com o objetivo de modelar atrasos com precisão, e é este modelo de temporização que os fabricantes de FPGA e ASICs podem seguir (AMORE, 2005).

A Linguagem VHDL possibilita múltiplos níveis de hierarquia para um projeto, pois a descrição pode consistir na interligação de outras descrições menores. Estes estilos são o comportamental e o estrutural, e podem ser mesclados em uma mesma descrição (AMORE 2005).

A Linguagem VHDL permite descrever um mesmo circuito com diversos graus de abstração, um processo iterativo de simulações e detalhamento dos elementos da estrutura é executado até atingir uma descrição que permite a síntese, e até que as simulações assegurem a equivalência entre a especificação do projeto e a descrição proposta. Os três passos em um projeto empregando uma Linguagem de Descrição de Circuitos está representado na Figura 3.1, onde a partir das especificações do projeto, é gerada uma descrição VHDL, que é

submetida a um simulador para a verificação de coerência entre a especificação e o código, após esta mesma descrição é analisada por uma ferramenta de síntese para alcançar as estruturas necessárias para um circuito que corresponda à descrição (AMORE, 2005).

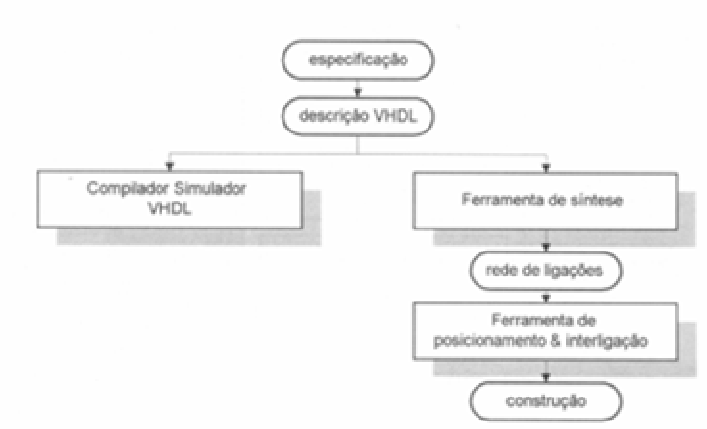


Figura 3.1 – Bloco diagrama das etapas de um projeto.
Fonte – AMORE, 2005.

Após esta etapa é gerado um arquivo contendo uma rede de ligações de elementos básicos disponíveis na tecnologia do dispositivo empregado. A Linguagem VHDL possibilita descrever um mesmo circuito com diversos graus de abstração, a Figura 3.2 ilustra a etapa de elaboração da descrição, onde um processo repetitivo de simulações é executado até atingir uma descrição que permita a síntese.

Após esta etapa é gerado um arquivo contendo uma rede de ligações de elementos básicos disponíveis na tecnologia do dispositivo empregado. A linguagem VHDL possibilita descrever um mesmo circuito com diversos graus de abstração, a Figura 3.2 ilustra a etapa de elaboração da descrição, onde um processo repetitivo de simulações é executado até atingir uma descrição que permita a síntese.

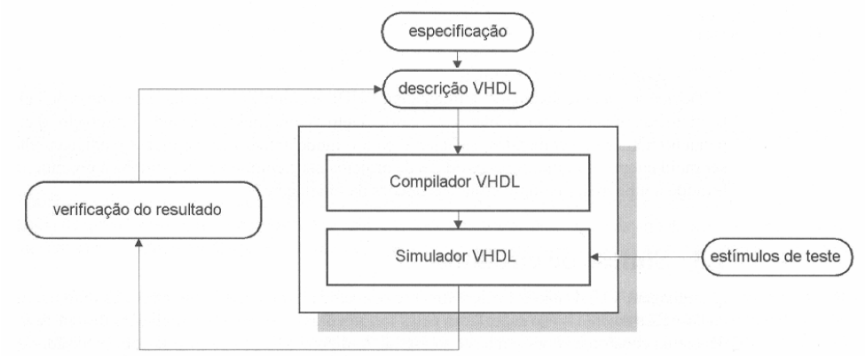


Figura 3.2 – Bloco diagrama da elaboração da descrição VHDL.
Fonte – AMORE, 2005.

O processo de síntese ilustrado na Figura 3.3, é iniciado assim que se concluir o processo de descrição, onde a ferramenta de síntese após a verificação de erros de sintaxe, executa o processo de conclusão e interligação das estruturas necessárias para o circuito a ser gerado. Neste momento, é gerado um circuito no nível RTL- Register *Transfer Level*, usando comparadores, somadores, registradores e portas lógicas, porém o circuito gerado não está vinculado a nenhuma tecnologia de fabricação em específico e tão pouco está otimizado.

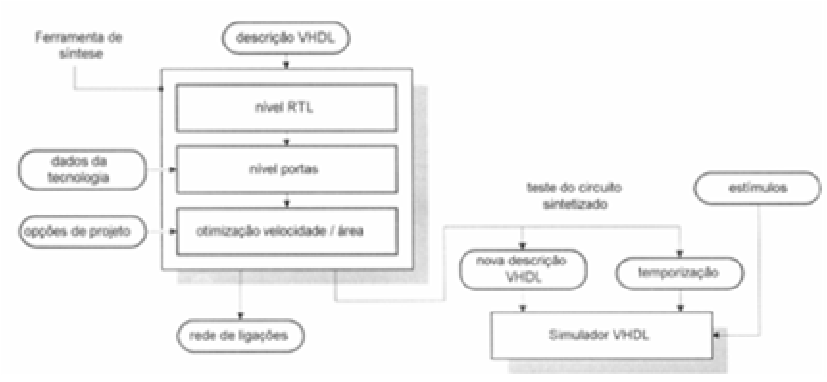


Figura 3.3 – Bloco Diagrama da síntese da descrição VHDL.
Fonte – AMORE, 2005.

O próximo passo é gerar um novo circuito a partir da estrutura RTL, pois esta apenas emprega primitivas genéricas da ferramenta, já que o circuito contém apenas elementos

disponíveis na tecnologia empregada na fabricação, por isso se faz necessário especificar o dispositivo. Neste momento é importante ressaltar que dois parâmetros conflitantes se fazem presentes, otimização de custos e otimização para velocidade, como um dos resultados desta etapa, tem-se um arquivo contendo uma rede de ligações entre os elementos disponíveis na tecnologia empregada. A temporização estimada nessa etapa pode conter diferença de até 20% em relação ao circuito real, isso porque os atrasos referentes as interligações não podem ser determinados com precisão (AMORE, 2005).

Se baseando nas ligações geradas pela ferramenta de síntese, a ferramenta que realiza o posicionamento e interligação dos componentes, *place and route*, assenta cada primitiva em um local do dispositivo empregado e define um caminho para interligação com as demais primitivas. Normalmente por se tratar de dispositivos lógicos programáveis, os próprios fabricantes fornecem as ferramentas par realizar o posicionamento e as interligações, devido a particularidade de cada tecnologia. As informações de temporização e atrasos tem uma estimativa mais precisa nesta etapa (AMORE, 2005).

3.1 Testbench

O uso da Linguagem VHDL possui uma estratégia de validação funcional, trata-se de um conjunto de instruções específicas para geração de estímulos e captura de resultados. As especificações do circuito descrito são simulados com a finalidade de verificar a coerência dos sinais gerados com os projetados, essa estratégia de validação funcional do módulo é baseada no uso de *Testbench* (TOROK, 2009).

Um *Testbench* é uma bancada de testes virtual, implementada como uma descrição também em VHDL, que por sua vez contém uma instância VHDL do dispositivo a testar. O *Testbench* é um sistema autônomo, que descreve o comportamento do ambiente externo, instanciando o módulo sob teste e interagindo com este, através de estímulos produzidos a partir da especificação de uma sucessão de tarefas previamente preparadas para o módulo em teste funcional. É importante ressaltar que o comportamento de um modelo implementado para validação funcional com o uso de *Testbench* é diferente de sua realização física, e deve ser utilizado somente para testes (TOROK, 2009).

3.2 Máquina de Estados

Uma Máquina de Estados é um circuito seqüencial que transita numa seqüência predefinida de estados. A transição entre os estados é comandada por sinais de controle. O estado atual é definido por um elemento de memória, e o estado futuro é determinado com base no estado atual e na condição das entradas.

Existem dois estilos de Máquina de Estados, *Mearly* e *Moore*. Na máquina de *Moore*, o valor da saída depende exclusivamente do estado atual, enquanto que na máquina de *Mearly*, o valor de saída é função do estado atual e da condição das entradas (AMORE, 2005).

A transição entre os estados é controlada por um sinal de clock, e pode conter condições de inicialização síncrona ou assíncrona. A Figura 3.4, mostra uma máquina de estados finitos, esta máquina é apenas ilustrativa e tem apenas três estados, ela possui um sinal de clock *clk* e um pino de entrada *a*.

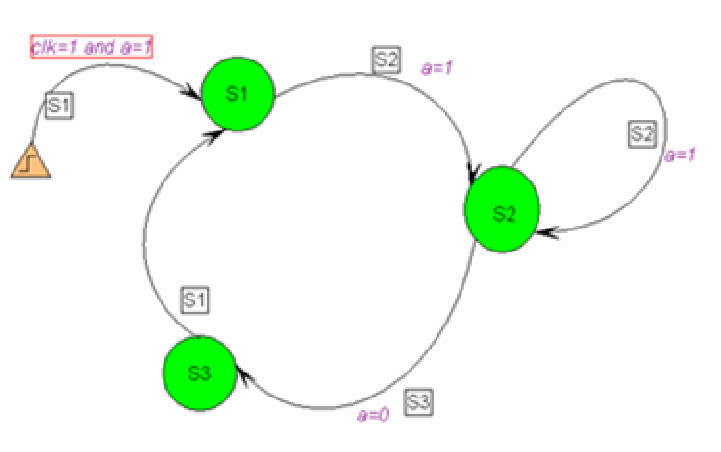


Figura 3.4 – Diagrama de uma máquina de estados.
Fonte – Autor, 2010.

Esta máquina de estados vai para o estado S1 quando ocorrer uma borda de subida de *clk* e quando *a* for igual a '1', A máquina transita para S2 quando *a* for '1' novamente, e fica presa em S2 enquanto *a*='1', quando *a*='0' ela vai para S3 e em seguida para S1, onde o ciclo se repete. Cada mudança de estado ocorre na borda de subida de *clk*. Abaixo está o código VHDL da máquina de estados da Figura 3.4.

```
Sreg0_machine: process (clk)

begin

if clk'event and clk = '1' then
     if clk=1 and a=1 then
            Sreg0 <= S1;
            S1;
     else
     case Sreg0 is
            when S1 =>
                  if a=1 then
                        Sreg0 <= S2;
                        S2;
                  end if;
            when S2 =>
                  if a=0 then
                        Sreg0 <= S3;
                        S3;
```

```
                elsif a=1 then
                      Sreg0 <= S2;
                      S2;
                end if;
          when S3 =>
                Sreg0 <= S1;
                S1;
          when others =>
                null;
     end case;
     end if;
end if;
end process;
```

3.3 Contadores

Um contador pode ser descrito como uma máquina de estados, onde o código de cada estado corresponde ao valor da contagem. Outra possibilidade para descrição de contadores é o emprego de operações aritméticas de soma ou subtração, controladas por um sinal de clock. Desta forma se cria variáveis numéricas de acordo com a necessidade, e se emprega estas variáveis dentro dos estados de modo que uma condição seja imposta, e enquanto ela não for satisfeita a máquina fica neste estado, abaixo é mostrado o código VHDL de um estado de uma máquina que utiliza um contador.

```
when S1 =>
if hss = '1' then
     if c1 < 16 then
       c1 := c1 + 1;
     Sreg1 <= S3;
else
     Sreg1 <= S2;
end if;
```

Neste código se observa que a máquina de estados apenas irá para o estado S2 se a variável C1 for maior que 15, caso contrário ela é incrementada a cada pulso de clock até atingir o valor 15.

4 HARDWARE RECONFIGURÀVEL

Muitas aplicações computacionais necessitam alterações freqüentes de suas funcionalidades, o rápido desenvolvimento dos recursos de concepção de circuitos integrados, tanto na área de processos quanto na área de CAD, tornou possível o surgimento de dispositivos com Lógica Programável. Estes dispositivos permitem aos usuários implementar circuitos complexos, e fazer alterações sem a necessidade do uso de onerosos recursos de fundição em silício, para a fabricação dos mesmos. Tendo como grande atrativo o fato de tais circuitos poderem ser reprogramados, isso possibilitou a prototipação de circuitos eletrônicos, gerando assim economia de tempo e dinheiro para a concepção de projetos eletrônicos (TERROSO , 1998).

No caso de implementação em hardware, as aplicações flexíveis são obtidas principalmente através de uso de dispositivos tais como o FPGA (*Field Programmable Gate Array*). Os FPGAs modificaram a tradicional distinção entre hardware e software, visto que sua funcionalidade em hardware pode ser alterada de forma total ou parcial ou até mesmo dinâmica. (TOROK, 2001).

Os FPGAs são circuitos programáveis compostos por um conjunto de células lógicas ou blocos lógicos alocados em forma de uma matriz. Geralmente, a funcionalidade destes blocos assim como o seu roteamento, são configuráveis por software. A palavra *Field* indica que a configuração do circuito pode ser feita pelo usuário final (TERROSO, 1998).

A arquitetura do FPGAs, ilustrada pela Figura 4.1, consiste em uma forma genérica de representar o FPGA, esta forma é composta por uma matriz de elementos agrupados em blocos lógicos configuráveis, que através de barramentos de interconexão configuráveis, podem ser interconectados. Semelhante a uma PAL(*Programmable Array Logic*),que através de blocos de chaves configuráveis pelo usuário podem realizar interconexões entre os elementos. E por meio de blocos de entrada/saída configuráveis é realizado o interfaceamento com o mundo externo (TOROK, 2001).

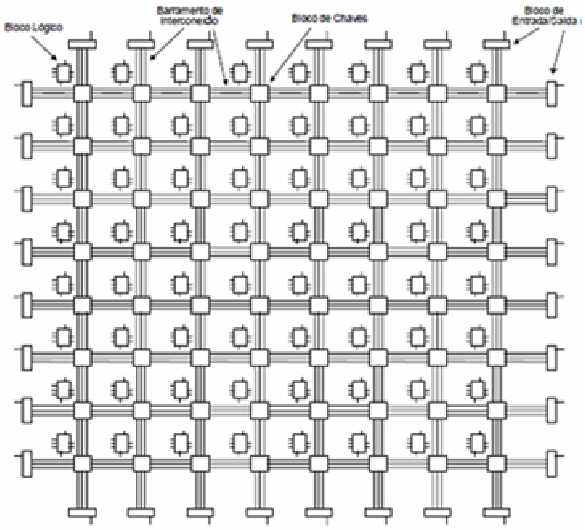


Figura 4.1 – Arquitetura genérica de um FPGA.
Fonte – TOROK, 2001.

4.1 Plataforma de Prototipação

Sistemas dotados com dispositivos digitais reconfiguráveis de forma a implementar um hardware recebem o nome de Plataformas de Prototipação, estas disponibilizam ainda circuitos externos como: memórias; conversores e interfaces. Estas interfaces podem ser usados em conjunto com o dispositivo de lógica programável, ampliando ainda mais as possibilidades do projetista facilitando as alterações do sistema digital e reduzindo o tempo de projeto dos produtos.

Algumas plataformas de prototipação mais recentes são dotadas da tecnologia de *In System Program* (ISP). A ISP possibilita a configuração, reconfiguração e testes do dispositivo reconfigurável sem ter que retirá-lo do circuito onde se encontra. Utilizando-se para isso de programas específicos disponibilizados pelo fabricante da plataforma, é possível comunicar a plataforma a um computador por meio das portas serial, paralela ou USB (XILINX, 2009).

As vantagens de se utilizar plataformas de prototipação com FPGAs modernos diferem dos discutidos até aqui sobretudo em quatro aspectos. O primeiro é a forma do bloco lógico básico, que mudou de um bloco monolítico para um conjunto de blocos com roteamento local de alto desempenho, introduzindo o conceito denominado em inglês de *cluster*[10]. O segundo aspecto é a adição de módulos com funcionalidade específica aos dispositivos, visando suprir a deficiência de implementar tal funcionalidade de maneira eficiente usando blocos lógicos convencionais. Entre os blocos mais difundidos encontram se memórias de acesso simples ou duplo, controles de clock tais como DLLs ou PLLs e

[10] Cluster- É o nome dado a um sistema montado com mais de um computador, cujo objetivo é fazer com que o processamento da aplicação seja distribuído entre os computadores.

multiplicadores. O terceiro aspecto é a adição de capacidades especiais ao dispositivo, tal como a reconfigurabilidade parcial, visando aumentar ainda mais as vantagens mais marcantes de FPGAs em relação a outros dispositivos de hardware. O último aspecto é a agregação *de IP hard/Soft Cores* de processadores, proprietários ou de outros fabricantes estabelecidos diretamente no mesmo substrato de silício que um FPGA. Isto produz o conceito de sistemas em um CI reconfigurável, ou SORC (do inglês, *System-on-a-Reconfigurable Chip*) (TOROK, 2001).

Existe um grande número de plataformas na área comercial e de pesquisa, possibilitando várias opções de escolha. A plataforma de prototipação comercial que oferece os recursos necessários para o projeto deste trabalho é revisada a seguir.

4.1.1 Plataforma Spartan-3E Starter KIT

A Plataforma de Prototipação Spartan-3E Starter Kit, da empresa XILINX, será usada para realização deste trabalho. Esta Plataforma de Prototipação dá suporte ao FPGA XC3S500 da família Spartan-3E. A Figura 4.2 mostra a Plataforma de Prototipação da XILINX.

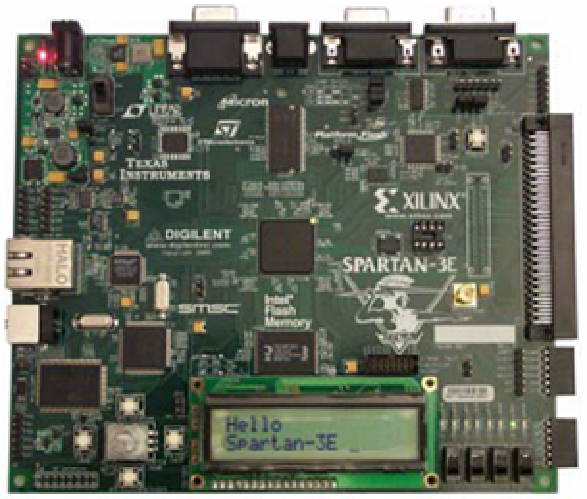


Figura 4.2 – Foto da Plataforma de Prototipação Spartan-3E Starter Kit.
Fonte – XILINX, 2006a.

Na Figura 4.2 é possível observar o FPGA no centro da plataforma. Os botões tipo *Push Botton* ao lado do dispositivo LCD. Outros circuitos que merecem destaque na plataforma são:

- CPLD XC2C64A CoolRunner II;

- Conversores DA e AD;

- Memória Strata FLASH PROM;

- Memória DDR-SDRAM;

- Oscilador;

- Dispositivo LCD;

- Interfaces Ethernet, USB, Serial, VGA e PS/2;

- Conector de expansão FX2.

4.1.2 FPGA_XC3S500E

O XC3S500E é um FPGA da família Spartan 3E da XILINX, com tecnologia de fabricação de 90nm, possibilita uma densidade de 500.000 portas lógicas equivalentes. Possui 1.164 CLBs, com 360 Kbits de memória RAM distribuídas em 20 blocos. O XC3S500 pode operar numa freqüência de até 300MHz. O FPGA possui 320 pinos dos quais 232 pinos são utilizados para entradas e saídas (I/O), divididos em 4 bancos (0, 1, 2 e 3) onde cada bloco possui sua própria alimentação(XILINX, 2009). Uma descrição detalhada dos pinos pode ser encontrada na Figura 4.3.

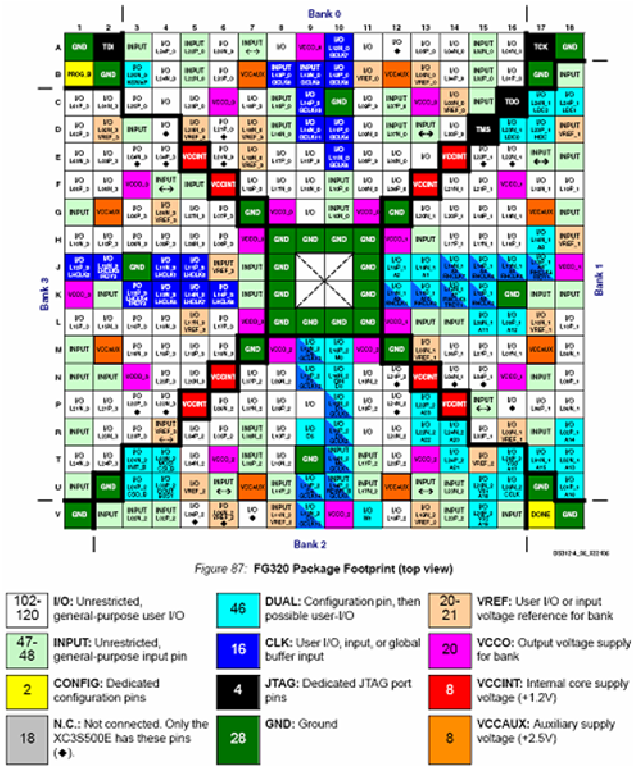


Figura 4.3 – Disposição dos pinos do FPGA XCS500E
Fonte – XILINX, 2009.

4.1.3 CPLD

O CPLD XC2C64A *CoolRunner-II*, da empresa XILINX, também é um dispositivo reconfigurável pelo usuário. O CPLD é reservado para o gerenciamento das memórias externas do FPGA (Flash PROM da XILINX e StrataFlash PROM da Intel). A Figura 4.4 ilustra as conexões entre o FPGA e o CPLD.

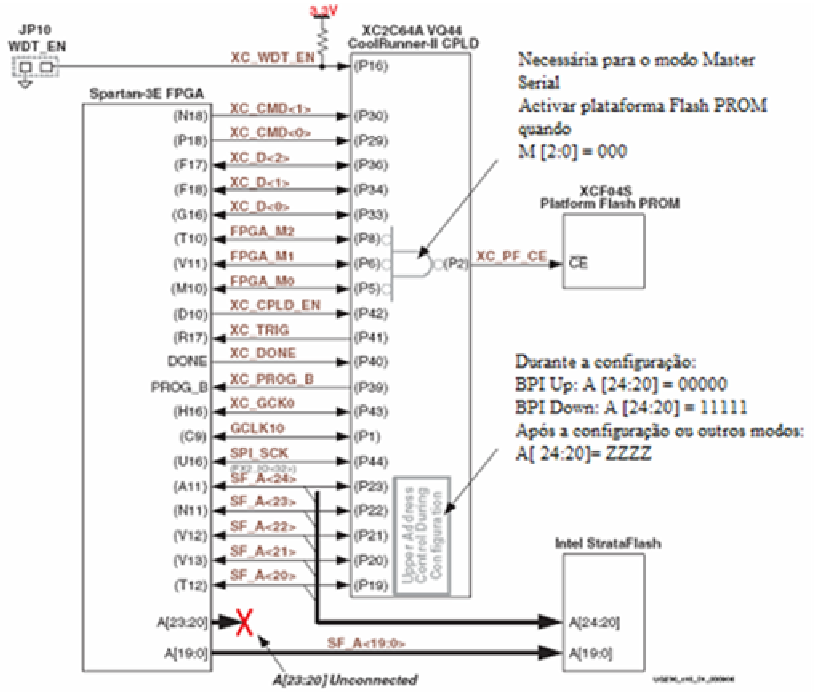


Figura 4.4 – Esquema de ligação do FPGA e CPLD.
Fonte – XILINX, 2006a.

Na Figura 4.4 é possível verificar que os pinos P8, P6 e P5 do CPLD recebem do FPGA o sinal que controla o acesso a memória Flash PROM. O CPLD também pode ser usado para facilitar as interconexões entre outros componentes da plataforma de prototipação.

4.1.4 Conversores DA e AD

A Plataforma de Prototipação possui o Conversor Digital para Analógico (DA) LTC2624, de quatro canais, com 12 bits de resolução. Suas 4 saídas podem ser verificadas nos pinos de J5 conforme mostra a Figura 4.5.

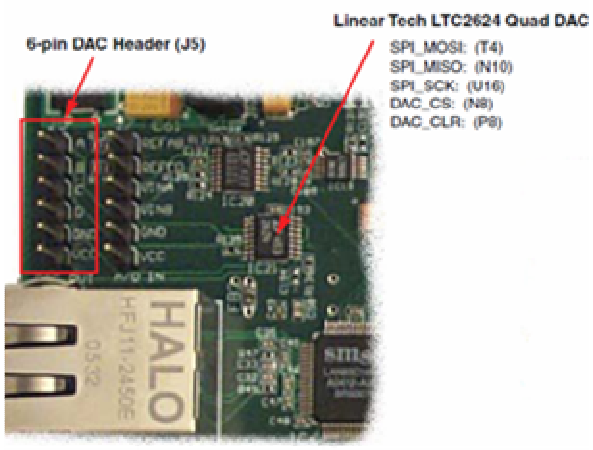


Figura 4.5 – Detalhe do Conversor Analógico/Digital.
Fonte – XILINX, 2006a.

O FPGA usa um periférico de interface serial para enviar dados para cada um dos quatro canais do conversor DA. A Figura 4.6 representa a ligação do FPGA com o Conversor DA.

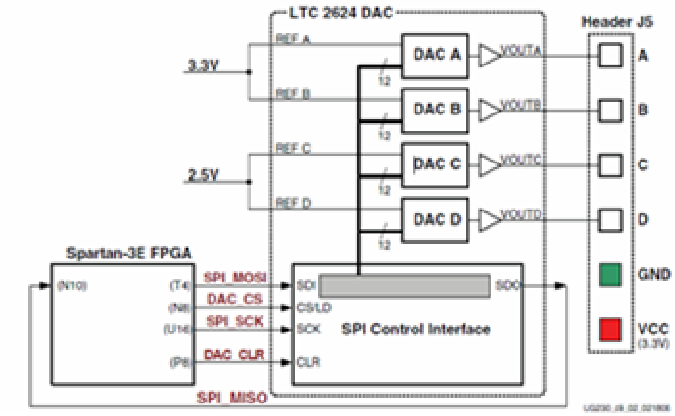


Figura 4.6 – Esquema de ligação do conversor DA com o FPGA.
Fonte – XILINX, 2006a.

A tensão de saída de cada um dos canais é descrita na Equação 4.1. Deve-se observar que o valor de dados[11:0] pode variar de 0 a 4096 (os 12 bits da resolução), as tensões de referência dos canais A e B é de 3,3V e a tensão de referência dos canais C e D é de 2,5V.

$$V_{out} = \frac{dados[11:0]}{4096} \bullet V_{REFERÊNCIA} \qquad (4.1)$$

A Plataforma de Prototipação possui também um circuito composto pelo LTC6912-1 e LTC 1407A-1, que é um conversor de sinais analógicos para digitais. O LTC6912-1 é um pré-amplificador que serve para ajustar os valores de amplitude dos sinais de entrada para o conversor. O LTC 1407A-1 é um conversor AD de 14 bits de resolução. O circuito pode trabalhar com dois canais de entrada VINA e VINB, que são encontrados nos pinos de J7. Tanto o pré-amplificador quanto o conversor AD são configurados e controlados serialmente pelo FPGA, utilizando a interface SPI. A Figura 4.7 mostra a disposição destes componentes.

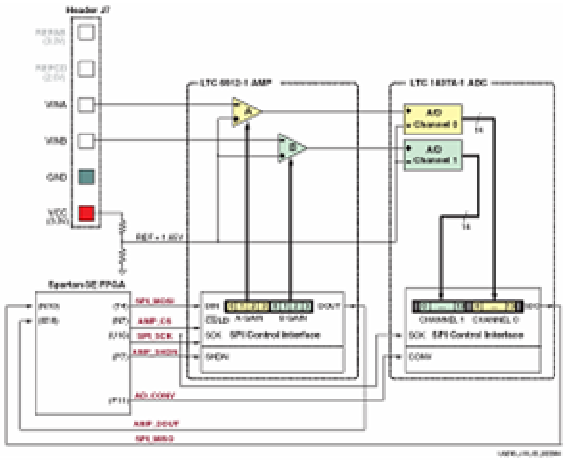


Figura 4.7 – Esquema de ligação do conversor AD com o FPGA.
Fonte – XILINX, 2006a.

4.1.5 Memória DDR-SDRAM

O Kit Spartan-3E disponibiliza uma memória dinâmica de acesso aleatório de 512Mbits(32M X 16) produzido pela *Micron*, para o armazenamento de dados voláteis. A

tecnologia DDR (*Double-Data-Rate*) SDRAM (MT46V32M16) possibilita gerar comandos de acesso e receber os dados referentes às leituras duas vezes por ciclo de clock, executando uma operação no início do ciclo e outra no final, com uma interface de 16-bits de dados (XILINX, 2006a). A Figura 4.8 mostra como a interface de dados de 16 bits da memória é ligada ao Banco3 de I/O do FPGA. A memória é alimentada por uma fonte de 2,5V.

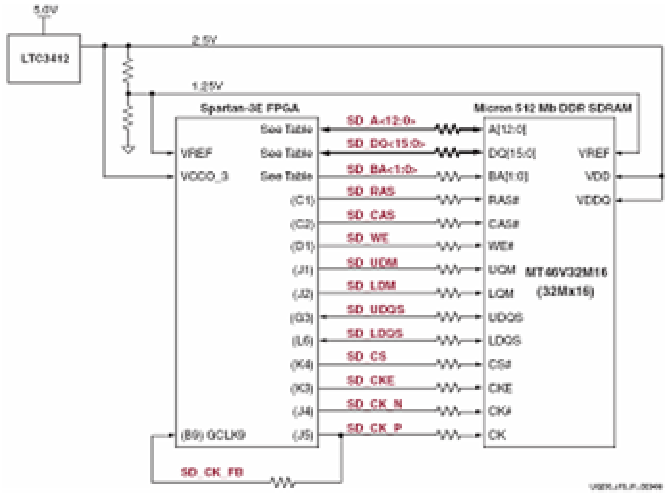


Figura 4.8 – Esquema de ligação entre DDR-SRAM e o FPGA.
Fonte – XILINX, 2006a.

4.1.6 Memória Flash ROM

O kit Spartan-3E inclui uma memória do tipo StrataFlash ROM fabricada pela Intel, com capacidade de armazenamento de 128Mbits (16Mbytes) para o armazenar dados não voláteis. É conectada ao FPGA e ao CPLD, conforme ilustra a Figura 4.9.

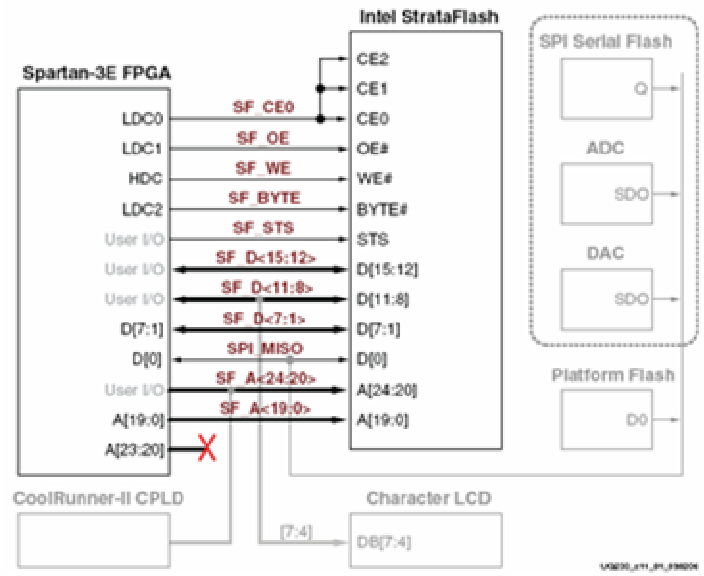


Figura 4.9 – Esquema de ligação da memória Flash RAM com o FPGA.
Fonte – XILINX, 2006a.

Na Figura 4.9 é possível observar que uma parte do barramento de dados entre a StrataFlash ROM e o FPGA é comum com o dispositivo LCD, assim como a linha 0 de dados é comum para os dados dos conversores.

4.1.7 Interfaces

A plataforma possui duas portas seriais RS-232. Uma porta com conector fêmea DB9 (DCE) para a interconexão com um computador e outra porta com conector macho DB9 (DTE) para a interconexão com dispositivos de comunicação, como pode ser observado na Figura 4.10. No DCE o pino 2 de transmissão está ligado ao pino M13 do FPGA e o pino 3 de recepção está ligado R7. No DTE o pino 2 de transmissão está ligado ao pino M14 e o pino 3 de recepção está ligado U8 do FPGA.

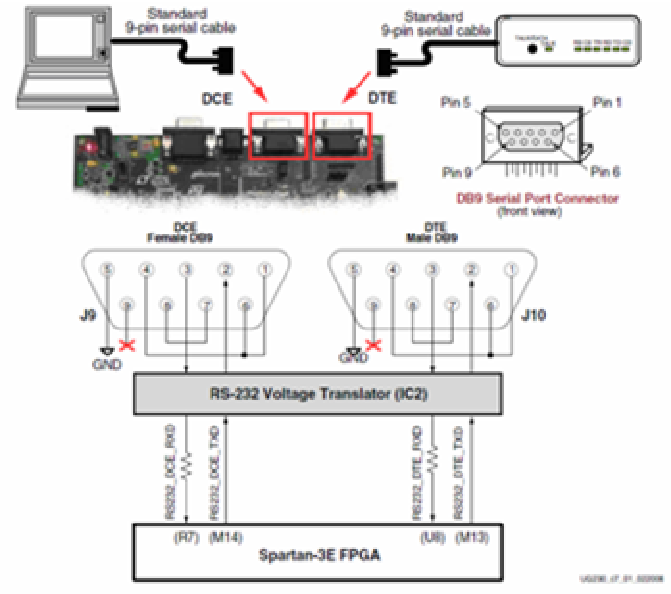


Figura 4.10 – Conectores macho e fêmea DB9.
Fonte – XILINX, 2006a.

A placa possui uma porta PS/2 para mouse e teclado. Essa porta possui um conector Mini-DIN 6 pinos onde o pino 1 de dados é ligado ao G13 e o pino 5 de clock é ligado ao pino G14 do FPGA, como mostra a Figura 4.11.

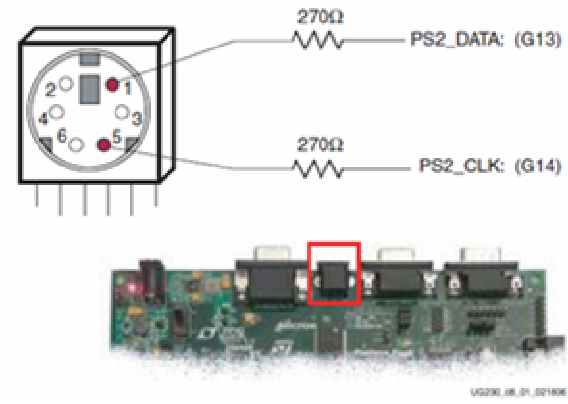


Figura 4.11 – Detalhe do conector PS2, e respectivo diagrama dos pinos.
Fonte – XILINX, 2006a.

A Plataforma de Prototipação conta com uma interface *Ethernet* 10/100 formada pelo chip LAN83C185 com um conector RJ-45. Todo o sincronismo é controlado através de um oscilador *on-board* de 25MHz. Implementando um *Media Access Controller* (MAC) no

FPGA é possível criar aplicações para a rede. Na figura 4.12 pode-se observar a interface Ethernet 10/100.

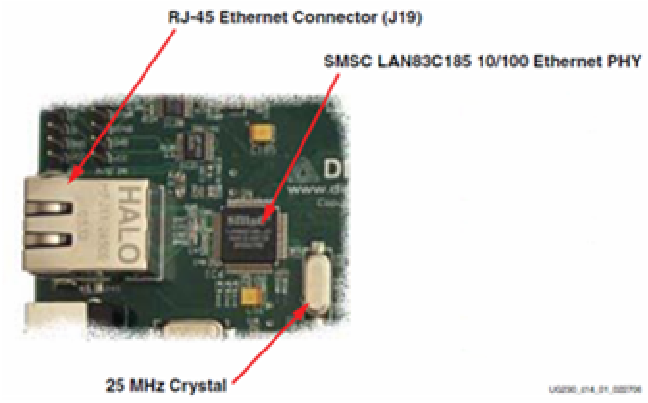


Figura 4.12 – Detalhe do conector da interface Ethernet 10/100.
Fonte – XILINX, 2006a.

A interface USB é reservada para realizar a comunicação do FPGA com a ferramenta de modelagem do computador (XILINX, 2006a).

4.1.8 Conector VGA

A Placa de Prototipação conta com uma porta VGA, permitindo que se ligue diretamente a maioria dos monitores de computadores ou painéis de LCD. A Figura 4.13 mostra a ligação do conector DB15 da porta com o FPGA.

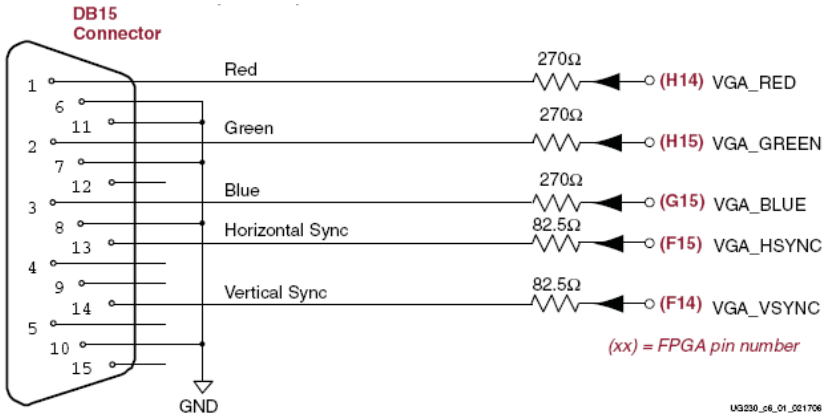


Figura 4.13 – Esquema de ligação da porta VGA com o FPGA.
Fonte – XILINX, 2006a.

O kit Spartan-3E aciona diretamente os cinco sinais VGA via resistores. Cada linha tem um resistor em série para cada uma das três cores, VGA_RED, VGA_GREEN e VGA_BLUE. O resistor em série, em combinação com a impedância de 75Ω incorporado no cabo VGA, garante que os sinais de cor permanecer no intervalo de 0V a 0,7V, especificado para o conector VGA. Os sinais VGA_HSYNC e VGA_VSYNC, respectivamente sinais de sincronismo horizontal e vertical usam o nível de unidade padrão LVTTL ou LVCMOS33. Os sinais de cor VGA_RED, VGA_GREEN e VGA_BLUE gerar as oito cores apresentadas na Tabela 3.

Tabela 3 – Sinais de cor RGB, para formar as oito cores primárias.

VGA_RED	VGA_GREEN	VGA_BLUE	Resulting Color
0	0	0	Black
0	0	1	Blue
0	1	0	Green
0	1	1	Cyan
1	0	0	Red
1	0	1	Magenta
1	1	0	Yellow
1	1	1	White

Fonte – XILINX, 2006a.

Monitores CRT utilizam amplitude modulada, movendo feixes de elétrons para exibir informações sobre uma tela revestida de fósforo. Os monitores LCDs usam uma pequena quantidade de tensão através de uma de cristal líquido, modificando assim a luz através do cristal com *pixel* a *pixel*. A seguinte descrição é válida a monitores CRT e LCDs , pois eles usam o mesmo sinal . Dentro de um monitor a varredura da tela segue um padrão de varredura, horizontalmente da esquerda para a direita e verticalmente de cima para baixo. Conforme mostrado na Figura 4.14, a informação é exibido apenas quando o feixe está se movendo da esquerda para a direita e de cima para baixo, e não durante o tempo que o feixe retorna para a esquerda ou para a borda superior da imagem . Parte do tempo da exposição potencial é, portanto, perdido em períodos em que o feixe do obturador é redefinido e estabilizado para começar um novo quadro.

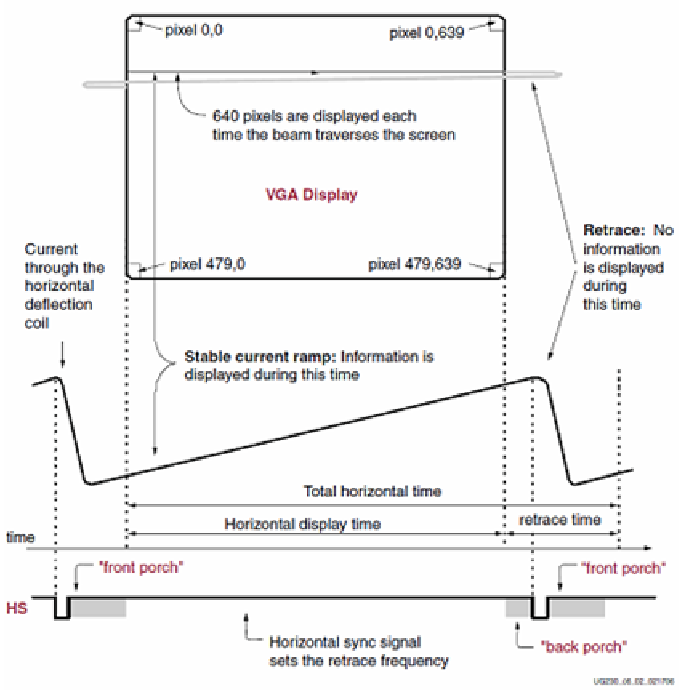


Figura 4.14 – Padrão de sincronismo para varredura de monitores.
Fonte – XILINX, 2006a.

Os tempos de sinal na Figura 4.14 são derivados da resolução de 640 pixels por 480 linhas, usando uma freqüência de 25 MHz de clock de *pixel* e 60 Hz para atualização de cada quadro.

4.1.9 Display LCD

A Plataforma de Prototipação conta com um visor de cristal líquido (LCD) de 2 linhas por 16 colunas. O FPGA se comunica com o LCD por uma interface de dados de 4 bits, tendo ainda alguns outros sinais para controle, conforme mostrado na Figura 4.15. O LCD pode exibir uma série de informações utilizando caracteres padrão ASCII, caracteres japoneses e alguns outros caracteres personalizados. Contudo estas exibições não são rápidas, cerca de meio segundo. Pode se dizer que as exibições são lentas comparadas com o oscilador de 50MHz da plataforma. O LCD utiliza o processador *PicoBlaze* para o controle dos tempos de exibição e o conteúdo do visor (XILINX, 2006a).

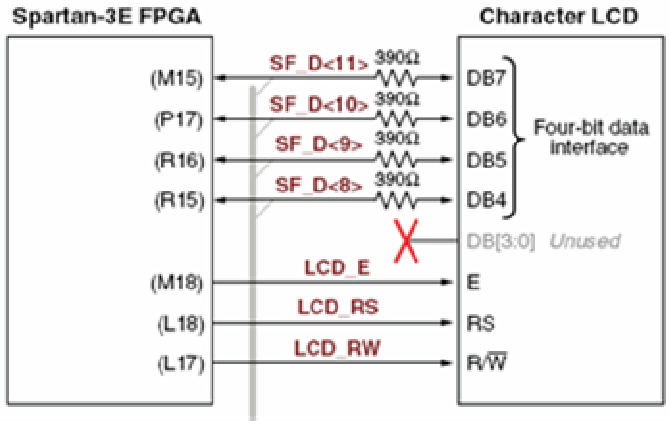


Figura 4.15 – Esquema de ligação do display LCD com o FPGA.
Fonte – XILINX, 2006a.

4.1.10 Fontes de Clock

A Plataforma fornece ao FPGA três fontes de entrada de clock, como pode ser observado na Figura 4.16. Uma das fontes de clock é um oscilador de 50MHz encontrado na própria placa que é ligado ao pino C9 do FPGA. Este oscilador deve ser alimentado por 3,3V, para que tenha um bom desempenho, o *jumper*[11] 9 controla a fonte de alimentação deste circuito. Pode-se também usar um oscilador acoplado no soquete DIP de 8 pinos da plataforma e é ligado ao pino B8 do FPGA. Existe ainda a possibilidade de obter o clock de uma fonte externa através do conector de entrada SMA que é ligado ao pino A10 do FPGA. O conector SMA também pode ser usado como uma saída de sinal pelo FPGA (XILINX, 2006a).

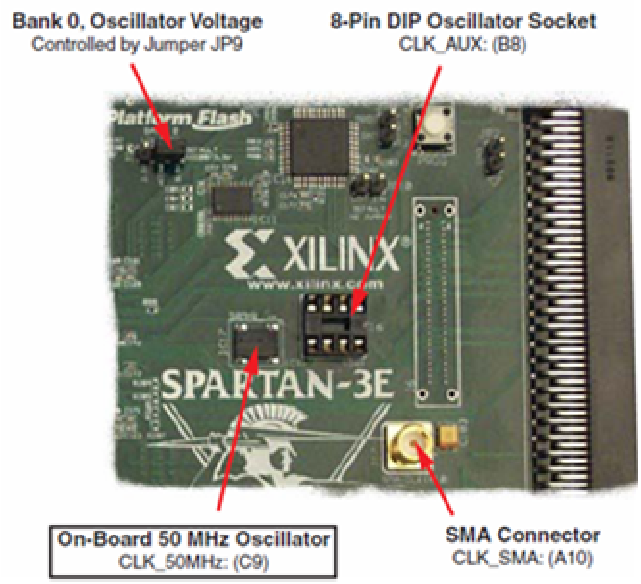


Figura 4.16 – Detalhes das fontes de clock disponíveis.
Fonte – XILINX, 2006a.

4.1.11 Conector de expansão FX2

[11] Jumper- É uma ligação móvel entre dois pontos de um circuito eletrônico.

A Plataforma de Prototipação também conta com um conector de expansão conhecido por Hiroshi FX2, que pode ser visualizado na figura 4.17.

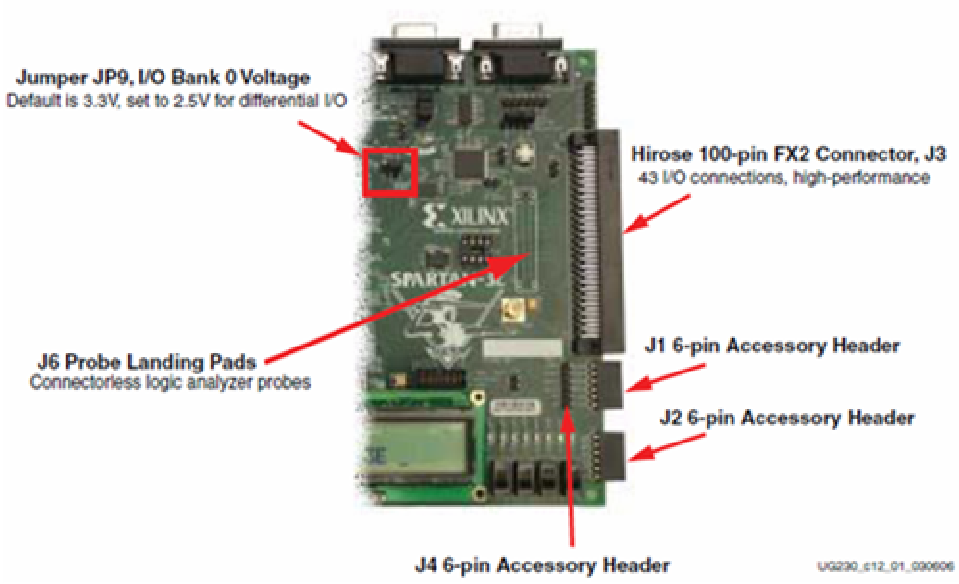


Figura 4.17 – Detalhe do conector de ezpansão FX2.
Fonte – XILINX, 2006a.

Este conector visa facilitar a conexão da plataforma com outros dispositivos, como é o caso da VDE1 usada neste trabalho. Esse conector de 100 pinos é associado diretamente a 43 pinos do FPGA e permite o acoplamento de outras placas de expansão (XILINX, 2006a). Os pinos deste conector e sua associação com o FPGA podem ser encontrados na Tabela 4.

Tabela 4 – Relação dos pinos FX2 com o FPGA.

Signal Name	FPGA Pin	Shared Header Connections					FX2 Connector		FPGA Pin	Signal Name
		LED	J1	J2	JP4	J6	A (top)	B (bottom)		
	VCCO_0						1	1		SHIELD
	VCCO_0						2	2	GND	GND
TMS_B							3	3		TDO_XC2C
JTSEL							4	4		TCK_B
TDO_FX2							5	5	GND	GND
FX2_IO1	B4		•			•	6	6	GND	GND
FX2_IO2	A4		•			•	7	7	GND	GND
FX2_IO3	D5		•			•	8	8	GND	GND
FX2_IO4	C5		•			•	9	9	GND	GND
FX2_IO5	A6			•		•	10	10	GND	GND
FX2_IO6	B6			•		•	11	11	GND	GND
FX2_IO7	E7			•		•	12	12	GND	GND
FX2_IO8	F7			•		•	13	13	GND	GND
FX2_IO9	D7				•	•	14	14	GND	GND
FX2_IO10	C7				•	•	15	15	GND	GND
FX2_IO11	F8				•	•	16	16	GND	GND
FX2_IO12	E8				•	•	17	17	GND	GND
FX2_IO13	F9	LD7				•	18	18	GND	GND
FX2_IO14	E9	LD6				•	19	19	GND	GND
FX2_IO15	D11	LD5				•	20	20	GND	GND
FX2_IO16	C11	LD4				•	21	21	GND	GND
FX2_IO17	F11	LD3				•	22	22	GND	GND
FX2_IO18	E11	LD2				•	23	23	GND	GND
FX2_IO19	E12	LD1					24	24	GND	GND
FX2_IO20	F12	LD0					25	25	GND	GND
FX2_IO21	A13						26	26	GND	GND
FX2_IO22	B13						27	27	GND	GND
FX2_IO23	A14						28	28	GND	GND
FX2_IO24	B14						29	29	GND	GND
FX2_IO25	C14						30	30	GND	GND
FX2_IO26	D14						31	31	GND	GND
FX2_IO27	A16						32	32	GND	GND
FX2_IO28	B16						33	33	GND	GND
FX2_IO29	E13						34	34	GND	GND
FX2_IO30	C4						35	35	GND	GND
FX2_IO31	B11						36	36	GND	GND
FX2_IO32	A11						37	37	GND	GND
FX2_IO33	A8						38	38	GND	GND
FX2_IO34	G9						39	39	GND	GND
FX2_IP35	D12						40	40	GND	GND
FX2_IP36	C12						41	41	GND	GND
FX2_IP37	A15						42	42	GND	GND
FX2_IP38	B15						43	43	GND	GND
FX2_IO39	C3						44	44	GND	GND
FX2_IP40	C15						45	45	GND	GND
GND	GND						46	46	E10	FX2_CLKIN
FX2_CLKOUT	D10						47	47	GND	GND
GND	GND						48	48	D9	FX2_CLKIO
5.0V							49	49		5.0V
5.0V							50	50		SHIELD

Fonte – XILINX, 2006a.

4.2 Ferramentas para síntese e Configuração do Hardware

Os fabricantes das placas de prototipação disponibilizam ferramentas que permitem ao projetista desenvolver, sintetizar e simular os circuitos lógicos dentro do hardware

reconfigurável. Estas ferramentas utilizam computadores para se comunicar com o hardware reconfigurável, do inglês *Computer Aided Design* (CAD) . Dessa forma, para a configuração do circuito no FPGA XC3S500E da XILINX, utiliza-se a ferramenta *Integrated Software Environment* (ISE), produzido pela XILINX. O software ISE é disponibilizado em duas versões o ISE *Fundation WebPACK* que é uma versão limitada mas gratuita e o ISE *Foundation* que é o programa na sua versão completa. A versão utilizada para o desenvolvimento do protótipo foi a *ISE Fundation WebPACK 10.1*, que pode ser adquirida no site da XILINX.

O ambiente ISE possibilita a modelagem do FPGA XC3S500E, de duas formas diferentes a partir de um esquemático que posteriormente é convertido em linguagem de descrição de hardware ou programando diretamente com uma linguagem de descrição de hardware. O software ISE oferece suporte a todas as etapas de um projeto, especificação, modelagem, síntese, mapeamento e implementação, é possível ainda realizar simulações e testes, utilizando-se de *Testbenchs*. A interface com o usuário do software ISE é o *Project Navigator*, como pode ser visualizado na Figura 4.18.

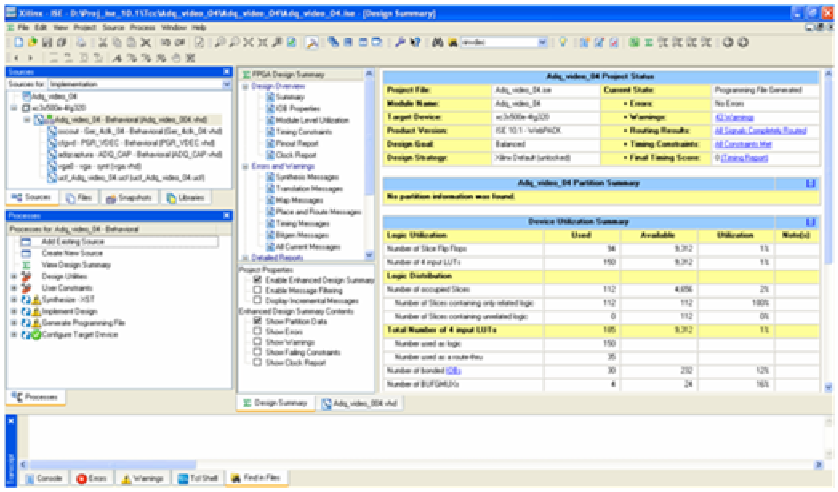


Figura 4.18 – Tela principal do ambiente do Project Navigator.
Fonte – Autor, 2010.

Após os processos de síntese e implementação, a ferramenta ISE gera um arquivo de extensão .BIT. A ferramenta ISE usa o programa *iMPACT* para configurar o FPGA, que realiza a comunicação entre o FPGA e o computador. A Figura 4.19 exibe a tela do programa *iMPACT*, após todos as etapas de síntese e programação do FPGA na plataforma de prototipação.

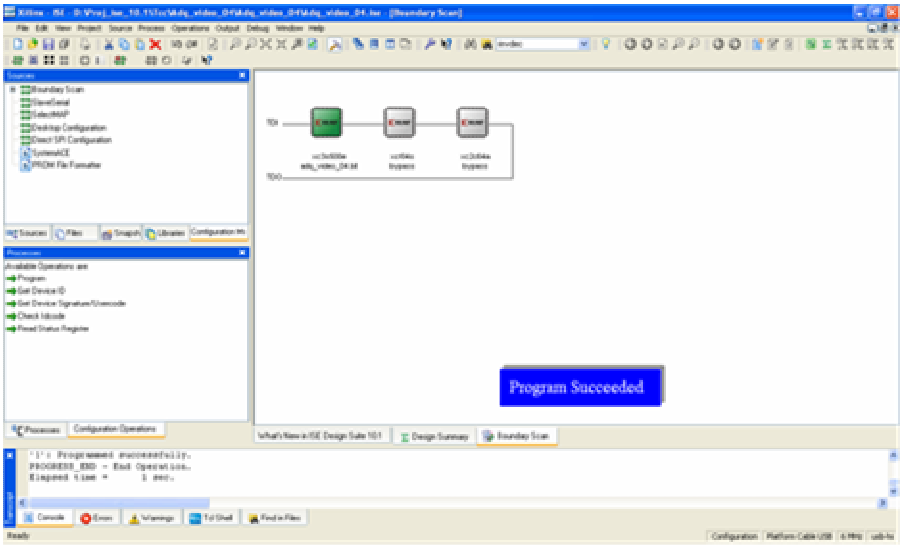


Figura 4.19 – Tela do ambiente do Software iMPACT.
Fonte – Autor, 2010.

Quando o Software *iMPACT* é executado, ele verifica o tipo de conexão existente entre o computador e a plataforma de prototipação onde se encontra o FPGA. Neste projeto a comunicação é fisicamente realizada através da porta USB. Após o *iMPACT* estabelecer a comunicação, ele configura o FPGA carregando o arquivo .BIT. O *iMPACT* pode configurar outros hardwares como o CPLD e a memória PROM da Plataforma de Prototipação Starter Kit Spartan-3E, contudo neste projeto estes dispositivos não foram programados.

5 HARDWARE ESPECÍFICO

A fim de realizar todas as propostas do trabalho, será necessário além da Plataforma de Prototipação o uso de alguns dispositivos adicionais específicos, tais como representados na Figura 5.1. Dividindo assim o Sistema de Captura em quatro módulos que serão capazes de realizar todas as etapas do projeto.

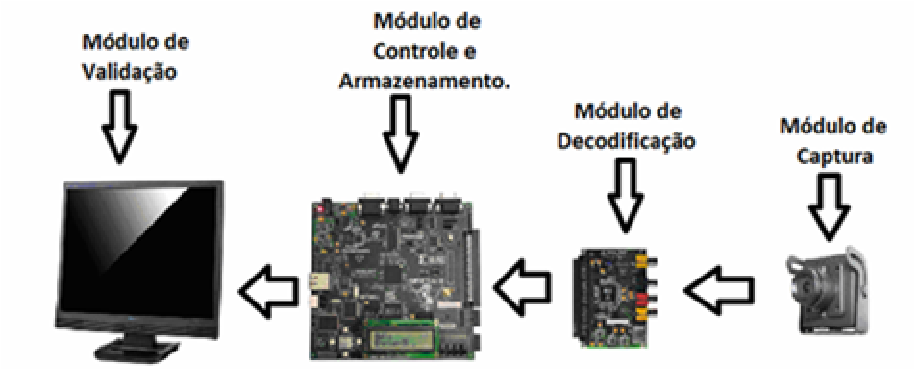


Figura 5.1 – Representação dos módulos necessários para validação do projeto.

Fonte – Autor, 2010.

5.1 Módulo de Captura

O Módulo de Captura é realizado por uma câmera de vídeo monocromática μCAM (EIA) produzida pela Kodo. A tecnologia utilizada para a captura da imagem e a conversão em sinais elétricos é a *Charge Coupled Display* (CCD).

Os CCDs são detectores de silício que convertem fótons em elétrons. Cada elemento do detector é chamado de um *píxel* , como estudado no Capítulo 1. O sinal de saída fornecido pela câmera é o padrão NTSC, com varredura entrelaçada, com interface de vídeo composto, ou seja, carrega todos os dados de imagem em um único sinal. Ela possui lentes de 3,6mm com uma abertura angular de 92,6°. A iluminação mínima necessária para operação é de 0,4 *lux*[12]. Sua saída de vídeo é de 1Vpp com uma impedância de 75 ohms. A imagem fornecida pela câmera tem uma resolução de 510 *pixels* na horizontal por 492 *pixels* na vertical, para maiores detalhes está disponível no anexo A o manual completo da câmera.

A tensão VCC é fornecida por uma fonte Agilent mod. E3631A. A câmera monocromática μCAM possui quatro fios de comunicação. Os fios vermelho e azul são os fios para alimentação da câmera. Respectivamente as saídas RED e BLUE da fonte, como mostrado na Figura 5.2, são ligadas a eles. Os fios preto e amarelo são para o sinal de saída. O fio preto é o terra e o fio amarelo contém o sinal de vídeo, esses fios são ligados a um conector RCA.

Este módulo necessita de um sistema de alimentação de 8,5 a 14,5VDC. O circuito montado para a alimentação da câmera fornece 9V DC cujo esquema é mostrado na Figura 5.2. O manual completo da μCAM pode ser visto no Anexo A.

---

[12] Lux- É a unidade de iluminação de um lúmen por metro quadrado, uma vela acesa a 30cm de distância equivale a 10 lux.

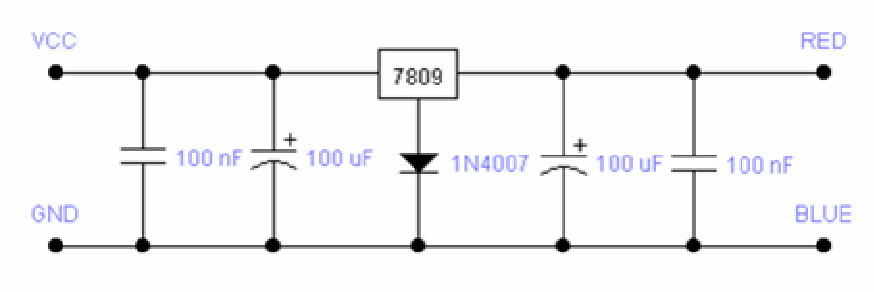


Figura 5.2 – Esquema da fonte de alimentação da câmera KODO.
Fonte – THIELE, 2007.

5.2 Módulo de Decodificação

O Módulo de Decodificação é responsável por extrair informações de vídeo provenientes do sinal fornecido pelo módulo de captura e disponibiliza-las ao módulo de controle e armazenamento. Para realizar essa decodificação de dados é utilizada a placa decodificadora de vídeo VDEC1, produzida pela DIGILENT, Figura 5.3.

Figura 5.3 – Foto da Placa Decodificadora de Vídeo (VDEC1).
Fonte – DIGILENT, 2005a.

A Placa Decodificadora de sinais de vídeo VDEC1, é equipada com o *chip* ADV7183B, que detecta o padrão de sinais analógicos de banda base de televisão, NTSC,

PAL ou SECAM, e digita-os através de três conversores analógicos digitais de 10 bits de resolução, operando em uma freqüência de 54MHz. O ADV7183B é fabricado com tecnologia CMOS de 3,3 V, o que assegura uma maior funcionalidade com menor dissipação de energia. Nele encontra-se também o barramento de controle compatível com o protocolo I2C (SCLK e SDA), onde é possível configurar controles pré-programáveis de cores, brilho, saturação e contraste (ANALOG DEVICES, 2005).

A VDEC1 contém fontes de potência filtradas e estabilizadas, uma fonte de clock estável, terminais de entradas BNC com 75 ohms para as interfaces de vídeo componente e vídeo composto e um conector fêmea Mini-DIN para interface S-vídeo .A VDEC1 possui um conector Hiroshi FX2 que permite a interconexão com a Plataforma de Prototipação Spartan 3E Starter Kit da XILINX. Neste conector são disponibilizados os dados de saída que podem ser emitidos em 8 ou 16 bits representados por P0 até P15. O Anexo B possui a relação dos pinos do conector Hiroschi FX2 com o FPGA.

Os sinais de sincronismos de vídeo horizontal (HS), vertical (VS) e de identificação de campo (FIELD) e clock de 27MHz(LLC1) (DIGILENT, 2005a).A Figura 5.4 demonstra o esquema funcional da VDEC1. No Anexo C está o bloco diagrama funcional do chip ADV7183B.

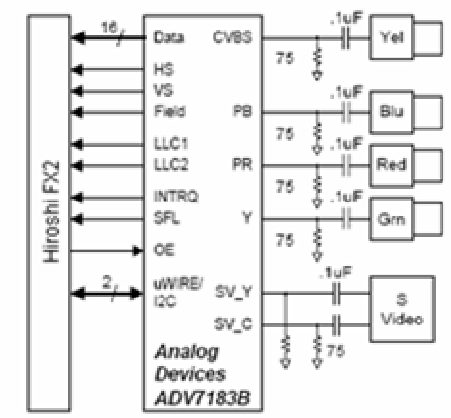


Figura 5.4 – Esquema simplificado da VDEC1.
Fonte – DIGILENT, 2005a.

A Placa Decodificadora VDEC1 precisa ser programada primeiramente, para decodificar o sinal de vídeo. O endereço de programação do dispositivo ADV7183B é "0100000". Para configurar um registrador do chip ADV7183B da VDEC1, são necessários 3 *bytes*. Os primeiros 8 *bits* são de endereçamento, os próximos *8 bits* apontam para um sub-endereço de um registrador interno. Os últimos 8 *bits* são os dados que são carregados neste registrador. O *chip* decodificador incrementa automaticamente o valor do sub-endereço em 1, mudando para o próximo registrador e para receber mais um pacote de 8 *bits* de dados para serem armazenados. Para configurar outro registrador que não esteja na seqüência é preciso iniciar uma nova comunicação (ANALOG DEVICES, 2005a).

5.2.1 Protocolo I2C

O Protocolo I2C, pode ser definido como "Sistema de Comunicação Digital executado por somente duas linhas". A aproximadamente 23 anos atrás a Philips, voltada a simplificar a comunicação digital entre dois dispositivos, desenvolveu um sistema bastante

simples designado por I2C . Ele utiliza somente duas linhas de comunicação denominadas por, *serial data* SDA e *serial clock* SCL. As linhas SDA e SCL carregam somente informações digitais, e portanto operam dentro dos limites de 0 a 5 volts. Para que este sistema funcione corretamente, alguns protocolos bem definidos devem ser obedecidos, como por exemplo, o sinal de início da transmissão(*start*), e os sinal de final da transmissão (*stop*). As duas vias de comunicação, um sinal de clock (SCL) e um sinal de dados (SDA), são ligadas a uma fonte de alimentação por resistores pull-up. Dessa forma enquanto o barramento estiver livre, ambas ficam em nível lógico '1'. Os níveis lógicos baixo e alto não têm valores pré-estabelecidos devido à variedade das tecnologias empregadas na fabricação dos dispositivos intercomunicados, dependendo exclusivamente dos valores da fonte de alimentação (PHILIPS, 2000).

Cada dispositivo possui um endereço único no barramento, desta forma qualquer dispositivo pode atuar como transmissor ou receptor. O transmissor será o dispositivo que envia dados pelo barramento, e receptor é aquele que recebe dados pelo barramento. O dispositivo que inicia a comunicação é definido como *Master*, ele gera o clock e encerra a comunicação. O dispositivo receptor é endereçado, e recebe o nome de *Slave*. No protocolo I2C pode existir mais de um dispositivo *Master* (PHILIPS, 2000).

Observando a Figura 5.5, é possível verificar que na comunicação do protocolo I2C existe uma condição de partida, essa condição ocorre quando há uma mudança de nível lógico de '1' para '0' em SDA enquanto o SCL está em '1'. A condição de parada, ocorre quando há uma mudança de nível lógico de '0 'para '1' de SDA enquanto o SCL está em nível lógico '1.'

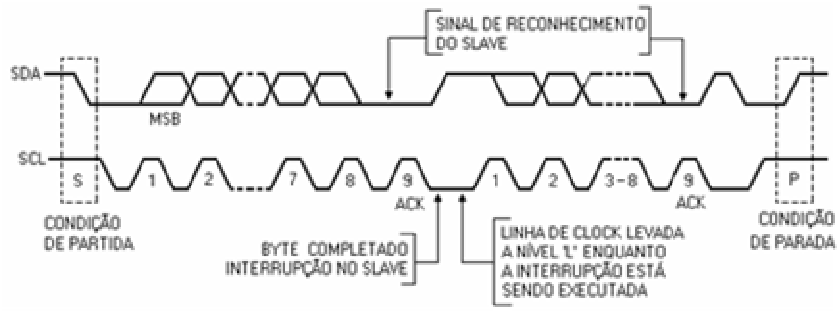


Figura 5.5 – Diagrama das condições do protocolo I2C.
Fonte – Adaptado de PHILIPS, 2000.

Quando ocorrer a condição de *START* o dispositivo *master* inicia a comunicação, ele envia um *byte*, dos quais 7 *bits* são de endereçamento e o primeiro bit indica se é leitura (*Read*) com lógica '1', ou quando é escrita (*Whrite*) de dados este bit está em lógica '0'. O dispositivo *slave* daquele endereço então informa através de um *bit* com lógica '0' se o *byte* foi recebido, e com lógica '1'caso não receba o *byte*, esse bit de reconhecimento recebe o nome de ACK (*Acknowledge*). Após o dispositivo *master* ou *slave* envia 8 *bits* de dados , ele aguarda por um ACK do outro dispositivo. Os *bits* de: endereço; R/W; DADOS e ACK são lidos quando SCL está em nível lógico '1', por isso, as mudanças em SDA devem ocorrer quando SCL está em '0'. Não há limite de quantidade de dados enviada para um endereço, apenas deve-se respeitar a seqüência de 8 bits de dados e o bit de ACK. Para demonstrar que a transmissão chegou ao fim o *maste*r gera a condição de STOP. A taxa de transferência é de até 100KHz no modo padrão ou até 400KHz no modo rápido(PHILIPS, 2000).

5.3 Módulo de Validação

O Módulo de Validação é composto pelo monitor de vídeo, no qual devem ser mostradas as imagens capturadas pelo Módulo de Captura. Para isso foi implementado na

Plataforma de Prototipação o componente VGA, responsável por gerar os sinais de sincronismo horizontal, vertical e de cor RGB para o monitor.

6 PROJETO EM VHDL

Para a descrição do projeto em VHDL dos módulos de Configuração, Captura e Armazenamento e Controle VGA, utiliza-se a ferramenta ISE. As descrições de componentes são desenvolvidas, baseadas nos sinal padrão de cada um dos dispositivos envolvidos no Sistema de Captura, para implementação no FPGA.

O desenvolvimento do Módulo de Configuração, do dispositivo VDEC1, é baseado na descrição utilizada em um trabalho de conclusão anterior (Thiele, 2007). Este Módulo é responsável por enviar sinais de configuração, por meio do conector hiroshi FX2, para o Módulo de Decodificação, e realizar a configuração do chip ADV7183B. Os dados vindos do Módulo de Captura são carregados nas Ram blocks, também implementada na Placa de Prototipação e responsáveis pelo armazenamento dos dados. Outra função importante do Módulo de Controle é o de realizar o sincronismo entre os componentes, para exibir no monitor os dados armazenados na memória, e iniciar todos os componentes do projeto, através de botões existentes na plataforma. A seguir será descritas de forma detalhada o Sistema de Captura e os componentes que compõem este Projeto.

6.1 Componente Proj_CapImg

O Sistema Protótipo de Captura de Imagens, denominado Proj_CapImg, é o *Top Level* (do Inglês, Nível mais Alto) do projeto em VHDL, é nele que todos os componentes do projeto estão instanciados. Neste bloco de nível mais elevado são feitas todas as conexões entre os componentes, bem como deste com os pinos de entrada e saída do Dispositivo FPGA. A Figura 6.1 ilustra o Bloco Diagrama, reduzido, do Proj_CapImg.

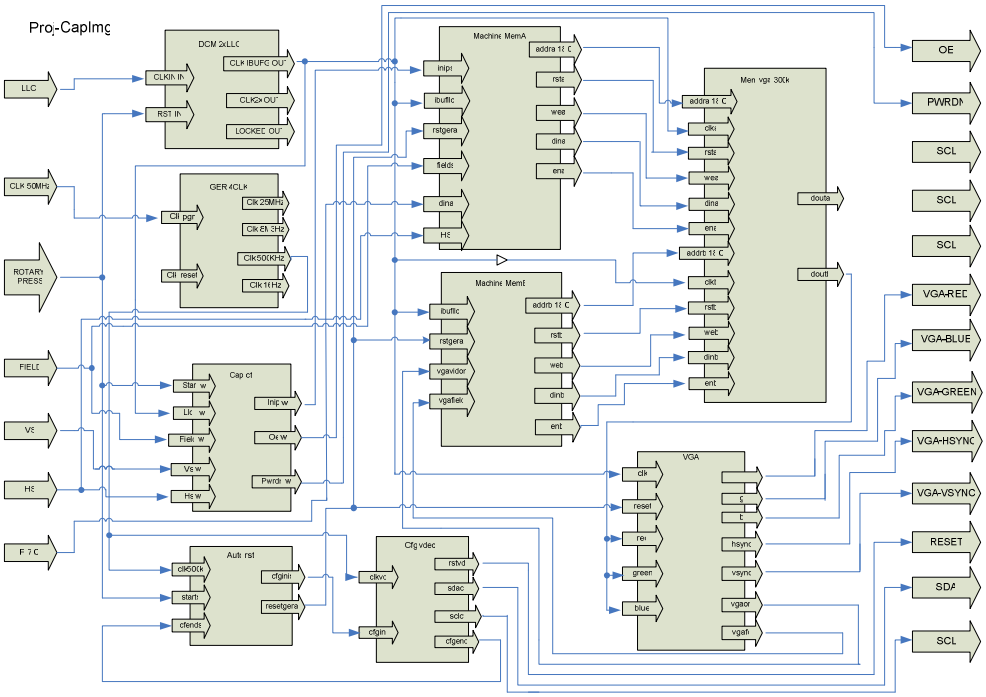


Figura 6.1 – Diagrama em blocos do Sistema Proj_CapImg.
Fonte – Autor, 2010.

O CLK_50MHz é o clock da Placa de Prototipação, este clock é ligado ao componente GER_4CLK.

ROTARY_PRESS é o botão rotatório central da placa, o pino de entrada do FPGA para este botão é configurado para que fiquem ligado a um resistor pull-down internamente, este botão inicia (Reset) o componente DCM_2xLLC1, gera um pulso de 'start' para o componente Cap_Ctl e para a máquina Auto_rst. A fim de facilitar a compreensão, o reset é

um pulso para limpar qualquer erro ou pendência de eventos e trazer um sistema à condição normal ou estado inicial

O barramento de entrada de 8 *bits* P recebe os sinais de dados de P8 a P15 que são as informações de imagem dos *pixels* quantizados em 8 *bits*, neste projeto são designados como P0 a P7.

LLC1 é a entrada de clock de 27MHz fornecidos pela VDEC1, que é ligada ao componente DCM_2xLLC1, como esta freqüência é ligada a diversos componentes, ela foi ligada ao um *Buffer*[13] para melhorar o desempenho.

FIELD, VS, HS recebem os sinais de controle da VDEC1, informação de campo, sincronismo vertical e horizontal.

A saída PWRDN envia um sinal que deixa o chip em um modo de espera (*Stand by*). A saída OE (*Output Enable*) envia o sinal para habilitar os sinais de saída do *chip* ADV7183B.

A saída SCL e a porta bidirecional SDA possuem os sinais da comunicação do protocolo I2C para a configuração do chip ADV7183B. A saída RESET é o 'reset' gerado pela VDEC1.

Os pinos de saída VGA_RED, VGA_GREEN, VGA_BLUE, VGA_HSYNC, VGA_VSYNC, possuem os sinais para o Conector VGA da Placa de Prototipação.

---

[13] Buffer- É uma região de memória temporária utilizada para escrita e leitura de dados.

Está implementada no projeto a máquina de estados finitos Auto_rst, o bloco desta máquina está representado na Figura 6.2. Esta máquina tem por finalidade gerar o 'reset' do componente VGA, e das máquinas de controle da memória, Machine_MemA e Machine_MemB, e gerar o pulso de configuração da VDEC1 .

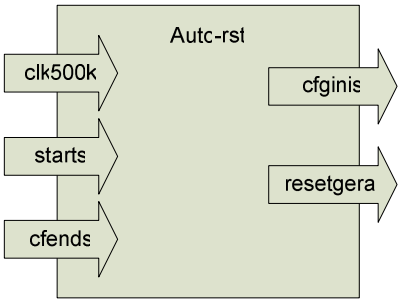


Figura 6.2 – Bloco diagrama da máquina Auto_rst.
Fonte – Autor, 2010.

A entrada starts receber um pulso do ROTARY_PRESS, para configura e resetar o componente CFG_VDEC1, através do sinal 'cfginis', e aguarda o sinal 'cfgend' que indica o fim da configuração da VDEC1, para iniciar os componetes VGA e as máquinas de controle da memória. Esta máquina opera com um clock de 500KHz fornecido pelo componente Ger_4clk. O código VHDL desta máquina pode ser visto abaixo.

```
auto_rst:  process (clk500k, starts, cfgends)
begin
if (clk500k'event and clk500k = '0') then
     if starts = '1' then
            cfginis <='1';
            rstgeral <='1';
            Sreg2 <= S1;
     else
     case Sreg2 is
            when S1 =>
              cfginis  <='0';
              sreg2 <=S2;
            when S2 =>
              Sreg2 <= S3;
```

```
            when S3 =>
              if cfgends = '1' then
               rstgeral   <='0';
               sreg2 <=S4;
            else
              sreg2 <=S2;
              end if;
            when S4 =>
         Sreg2 <= S4;
       when others =>null;
      end case;
      end if;
end if;
end process;
```

Quando o pino starts estiver em '1' na borda de subida do clock, então a VDEC1 é configurada e todos os componentes ficam resetados. No estado S1 a VDEC1 deixa de ser configurada e a máquina transita para S2 onde apenas aguarda o próximo pulso de clock para ir ao estado S3, onde a máquina aguarda o sinal de 'cfgends' que informa o fim da configuração da VDEC1 e deixa de resetar os componentes. Esta máquina fica presa no estado S4 até o próximo pulso de starts.

Para controlar a escrita e leitura da memória são implementadas duas máquinas de estados finitos, Machine_MemA e Machine_MemB.

Machine_MemA controla a escrita dos dados no componente Mem_Vga_300K. Na Figura 6.3 está o bloco desta máquina.

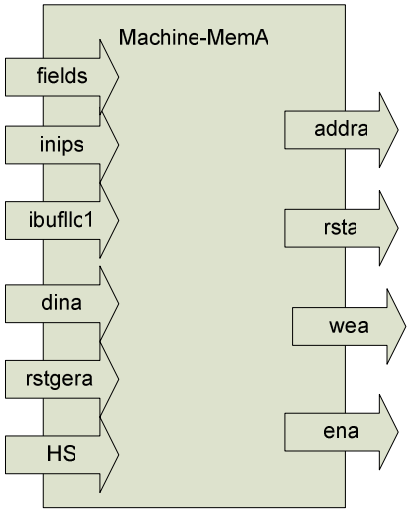


Figura 6.3 – Bloco diagrama da máquina Machine_MemA.
Fonte – Autor, 2010.

A entrada 'inips' recebe o sinal gerado pelo componente Cap_ctl para indicar o primeiro *pixel* válido, a ser armazenado na memória. Esta máquina opera com 27MHz do componente DCM_2xLLC1, o sinal de 'resetgeral' é gerado pela máquina Auto_rst, e o pino dina possui o dado P(7) (oitavo *bit* do *pixel* quantizado pela VDEC1) a serem gravado na memória. As saídas desta máquina endereçam o bloco A do componente Mem_Vga_300K, a máquina também controla os sinais de reset do bloco A (rsta), habilitar escrita (wea), habilitar o bloco A (ena) e disponibilizar os dados (dina). O código VHDL da máquina pode ser visto abaixo.

```
Machine_MemA: process (clk2xllc1, inips, fields, HS, rstgeral)
variable cntPix    : integer range 0 to 1024;
variable cntLin    : integer range 0 to 1024;
variable vdecendLin : integer range 0 to 1;
begin
if ibufllc1s'event and ibufllc1s = '1' then
     if rstgeral='1' then
           rsta <='1';
      dina <='0';
        wea <='1';
           ena <= '1';
           cntPix := 0;
```

```
            cntLin := 0;
            vdecendLin := 0;
            addra <= "0000000000000000000";
            Sreg0  <= S1;
       else
      case Sreg0 is
            when S1 =>
               rsta <='0';
                   Sreg0 <= S2;
            when S2 =>
           if fields ='0' and  inips = '1' then
                        cntPix := 0;
                        cntLin := 0;
                        vdecendLin := 0;
                     addra <= "0000000000000000000";
              elsif fields ='0' then
                  dina <= '0';
                        if Hs = '0' then
                              vdecendLin := 0;
                        end if;
                        if  vdecendLin = 0 then
                           if cntPix < 640 Then
                                dina <= p(7);
                                cntPix := cntPix + 1;
                             addra <= addra + "0000000000000000001";
                           else
                                cntPix := 0;
                                cntLin := cntLin + 1;
                                vdecendLin := 1;
                                if cntLin < 247 Then
                                  addra <= addra + "1010000000";--64
                                else
                    cntLin := 0;
                                  addra <= "0000000000000000000";
                                end if;
                           end if;

                     end if;
              end if;
          sreg0 <=S2;
              when others =>null;
      end case;
      end if;
end if;
end process;
```

Esta máquina utiliza três variáveis para auxiliar no controle: cntpix (contador de pixels); cntlin (contador de linhas) e vdecendLin (fim de linha da VDEC1). O estado S1 da máquina libera o bloco A da memória e salta para S2, onde aguarda o campo do vídeo 'fields' for par, e o sinal de 'inips' for '1'. Para zerar as varáveis cntpix, cntlin e vdecendLin, e aguarda o sincronismo horizontal da VDEC1 'HS' ser '0' para endereçar a memória. Enquanto o número de *pixels* for menor que 640 os dado de P(7) são enviados para dina. Toda vez que o contador de *pixels* cntpix for 639 o endereço addra é incrementado 640 posições, para iniciar uma nova linha. Este ciclo se repete por 246 vezes, pois apenas o campo par é gravado com dados de P(7), enquanto que o campo ímpar é gravado com '0', exibindo uma linha preta na tela. O número de linhas pares é definido pelo número de linhas que a câmera captura, que são 495 linhas, para controlar quando a máquina deve colocar os dados ou o valor '0', é utilizado a variável vdecendLin.

Para controlar o bloco B da memória é usado a máquina de estados Machine_MemB, esta máquina tem por finalidade controlar a leitura do componente Mem_Vga_300K, usando sinais gerados pelo componente VGA. A Figura 6.4 mostra o bloco desta máquina.

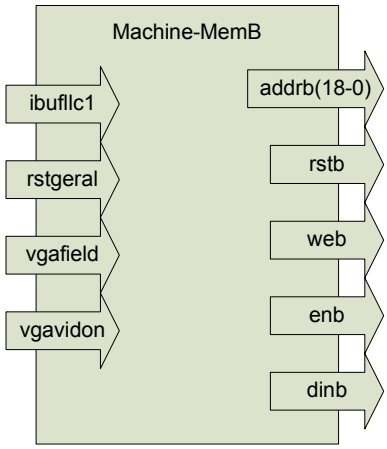


Figura 6.4 – Bloco diagrama da máquina de estados Machine_MemB.
Fonte – Autor, 2010.

A máquina Machine_MemB, recebe o clock de 27MHz do componente DCM_2xLLc1, e o reset da máquina Auto_rst. Os sinais 'vgafield' e 'vgavidon' são respectivamente, sinal de vídeo ativo vertical e sinal de vídeo ativo do componente VGA. As saídas desta máquina são responsáveis por endereçar e controlar o bloco B da memória, para leitura dos dados, pelo próprio componente VGA. O código VHDL da máquina pode ser analisado abaixo.

```
Machine_MemB: process (ibufllc1s, vgafield, rstgeral)
begin
if ibufllc1s'event and ibufllc1s = '1' then
     if rstgeral='1' then
           rstb <='1';
      dinb <='0';
         web <='0';
           enb <= '1';
           addrb <= "0000000000000000000";
           Sreg1  <= S1;
     else
     case Sreg1 is
           when S1 =>
              rstb <='0';
                 Sreg1 <= S2;
           when S2 =>
               if vgafield = '0' then
                     addrb <= "0000000000000000000";
               else
                    if vgavidon='1' then
                              addrb <= addrb + "0000000000000000001";
                    end if;
                end if;
          sreg1 <=S2;
            when others =>
                 null;
     end case;
     end if;
end if;
end process;
```

No estado S1 o rstb é colocado em '0', liberando o bloco B da memória, e a máquina salta para o estado S2 onde o sinal 'vgafield' é analizado. Se 'vgafield' for '0', trata-se de uma nova linha, então o endereço addrb é zerado e somente quando o sinal 'vgavidon' for '1' a máquina incrementar a leitura *pixel* a *pixel*. Os dados relacionados ao *pixel* são enviados as entradas RGB do componente VGA, através da saída doutb da memória. O código VHDL completo do projeto pode ser visto no apêndice A.

Cada pino do projeto foi ligado ao FPGA através de um arquivo com extensão UCF, User Constraints. É um arquivo de restrições onde são determinadas as ligações dos pinos de estrada e saída do projeto com o FPGA e demais dispositivos da placa de prototipação. Por padrão, cada projeto ISE tem um arquivo UCF com o mesmo nome do projeto Top Level. O arquivo de restrições completo pode ser visto no apêndice A.

6.2 Osciladores

A Placa de Prototipação tem diversas possibilidades de fontes de clock disponíveis, dentre elas, se utiliza apenas a de 50MHz. A VDEC1 possui um oscilador LLC1 de 27MHz e LLC2 de 13,5MHz, contudo se utiliza apenas a freqüência LLC1.

Os blocos que compõem o Sistema de Captura não operam apenas nesta duas freqüência , em virtude disto se faz necessário a implementação de componentes capazes de gera as freqüências necessárias a cada componente. Para tanto é possível por meio de contadores dividir a freqüência de entrada em freqüências menores, ou ainda multiplicar fontes de clock com o uso de DCM (Digital Clock Managers).

6.2.1 DCM

Digital Clock Managers (DCM's) fornecem capacidades avançadas de geração de freqüência para FPGA's da família Spartan 3. Com o uso do DCM é possível multiplicar ou dividir a freqüência de clock de entrada e sintetizar uma nova freqüência de clock, melhorando assim o desempenho do sistema.

Os blocos DCM tem conexões dedicadas às entradas de *global buffer*, que são parte integrante da infra-estrutura do FPGA. O bloco da primitiva DCM está representada na Figura 6.5 (XILINX, 2006b).

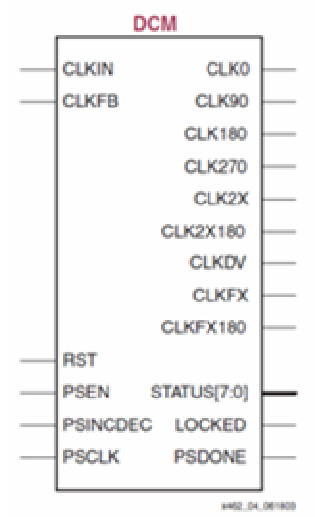


Figura 6.5 – Bloco diagrama da primitiva DCM.
Fonte – XILINX, 2006b.

As portas do DCM de: conexão; atributos; propriedades e restrições utilizadas neste trabalho estão resumidas abaixo:

-CLKIN. Pino de entrada do clock para o DCM. A freqüência CLKIN deve estar dentro dos limites especificados pelo *DataSheet* [14]do FPGA do kit Spartan-3;

-RST. Pino de entrada, reset assíncrono do bloco DCM;

-CLK2X. Pino de saída, possui o dobro da freqüência de entrada sem deslocamento de fase, seu ciclo é 50% do ciclo do clock de entrada;

-LOCKED. Pino de saída, sinal utilizado para verificar quando o clock de saída do DCM tem valores válidos, para ter valores válidos, deve ter lógica '1' e respeitar a faixa de freqüência de entrada que deve estar no mínimo em 1MHz e no máximo 280MHz (XILINX, 2006b).

6.2.2 Componente DCM_2xLLC1

O componente DCM_2XLLC1 é responsável por gerar duas freqüências de clock para diversos componentes do Sistema de Captura. Ele tem como entrada o pino CLKIN_IN que recebe a freqüência de 27MHz fornecido pela VDEC1, e possui três saídas, CLK_IBUFG_OUT que é a freqüência de entrada reforçada através de um buffer, CLK2x_OUT com o dobro da freqüência de entrada, e LOCKED_OUT que sinaliza quando o clock de saída do DCM tem valores válidos. A Figura 6.6 mostra o bloco diagrama deste componente.

---

[14] DataSheet- É um termo técnico usado para identificar um documento relativo a um determinado produto.

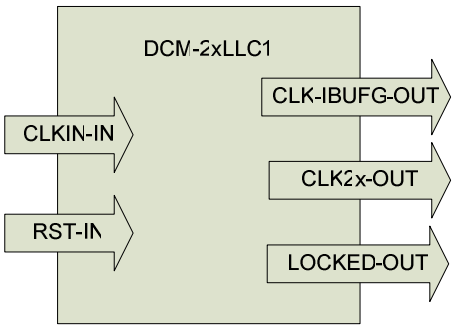


Figura 6.6 – Bloco diagrama do Componente DCM_2xLLC1.
Fonte – Autor, 2010.

O código VHDL do componente DCM_2XLLC1 pode ser visto abaixo.

```
begin
   GND_BIT <= '0';
   CLKIN_IBUFG_OUT <= CLKIN_IBUFG;
     CLK2X_OUT <= CLK2X_OBUF;
   CLK2X_BUFG_INST : BUFG
      port map (I=>CLK2X_BUF,
                O=> CLK2X_OBUF);
   CLKIN_IN_INST : BUF
      port map (I=>CLKIN_IN,
                O=> CLKIN_IBUFG);
```

O código VHDL completo do componente DCM_2xLLC1 pode ser visto no Apêndice A.

6.2.3 Componente Ger_4clk

O componente Ger_4clk uma das fontes de clocks para o Componentes Cfg_Vdec1, e para a máquina de Auto_rst do projeto. Ele tem como entrada o Clk_pgr que recebe o sinal de 50MHz fornecido pela Placa de Prototipação. Utilizando contadores internos, este componente divide o clock de entrada gerando quatro diferentes fontes de clock de menor

freqüência. As freqüências produzidas são de 25MHz, 8.3 MHz, 500 KHz e 16.6 Hz. A Figura 6.7 mostra o bloco diagrama deste componente.

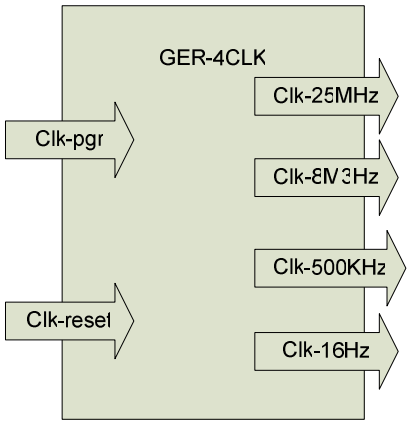


Figura 6.7 – Bloco diagrama do componente Ger_4clk.
Fonte – Autor, 2010.

A freqüência de 25MHz foi utilizada para testar e validar o funcionamento do componente VGA, já a freqüência de 500KHz é utilizada para o funcionamento do componente Cfg_Vdec1 e para gerar o reset automático do Proj_CapImg, enquanto que a freqüência de 16,6Hz é utilizada para sinalizar através do LED_0 o funcionamento do componente Ger_4clk. O código VHDL do componente Ger_4clk pode ser visto abaixo.

```
process(clk_pgr, clK_reset)
begin
   If clK_reset  = '1' then
    div8m3 <= "00";
    ckl8m3Hz <= '0';
   elsif clk_pgr'event and clk_pgr = '1' then
            div8m3 <= div8m3 + "01";
            if div8m3 = "10" then
               div8m3 <= "00";
       ckl8m3Hz <= not ckl8m3Hz;
            end if;
    end if;
end process;

process (clk_pgr, clk_reset)
```

```
begin
    If clk_reset ='1' then
       clk25M   <='0';
      elsif clk_pgr'event and clk_pgr = '1' then
            clk25M <= not clk25M;

     end if;
end process;
```

O código VHDL completo do componente Ger_4clk pode ser visto não apêndice A.

6.3 Componente Cfg_Vdec1

A Placa VDEC1 é responsável por decodificar os sinais de vídeo vindos do Módulo de Captura, para tanto é necessário configurar o *chip* ADV7183B. O código em VHDL do componente CFG_VDEC1, é responsável por gerar os sinais do protocolo de comunicação I2C e configurar o *chip* ADV7183B. A Figura 6.8 mostra o bloco diagrama deste componente.

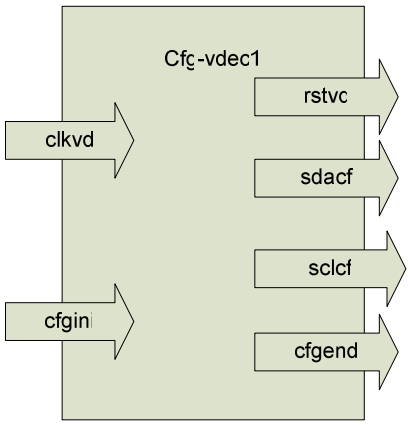


Figura 6.8 – Bloco diagrama do Componente Cfg_Vdec1.
Fonte – Autor, 2010.

Neste componente a entrada CLKVD recebe o sinal de 500KHz de clock gerado pelo componente Ger_4clk para ser utilizado como referência para a máquina de estados. A entrada CFGINI recebe um pulso oriundo da saída CFGINIS da máquina de reset automático, gerado pelo ROTARY_PRESS (botão rotatório central da Placa de Prototipação) que determina o reset automático e o início da configuração do chip ADV7183B. A saída RESET envia o sinal de reset para a saída RESET do Módulo Proj_CapImg. A entrada-saída SDACF é ligada a saída SDA do projeto *Top Level* e a saída SCLCF é ligada a saída SCL do Módulo Proj_CapImg, que são os pinos reservados para a interface I2C com a VDEC1, a saída CFGENDS sinaliza com lógica '1' quando a VDEC1 termina de ser programada.

Conforme manual, o endereço para configurar um registrador do ADV7183B é "0100000". Para tanto é necessário enviar uma seqüência de 3 *bytes*, primeiramente se envia 8 *bits* de endereçamento, o *chip* decodificador reconhece os próximos 8 *bits* como um sub-endereço que aponta para um registrador interno, e os últimos 8 *bits* de dados são carregados neste registrador. O *chip* decodificador incrementa o valor do sub-endereço em 1, mudando para o próximo registrador e possibilitando receber mais um pacote de 8 *bits* de dados para serem carregados neste. Para configurar outro registrador que não esteja na seqüência é necessário iniciar uma nova comunicação. A Figura 6.9 ilustra a seqüência do protocolo para a escrita de dados.

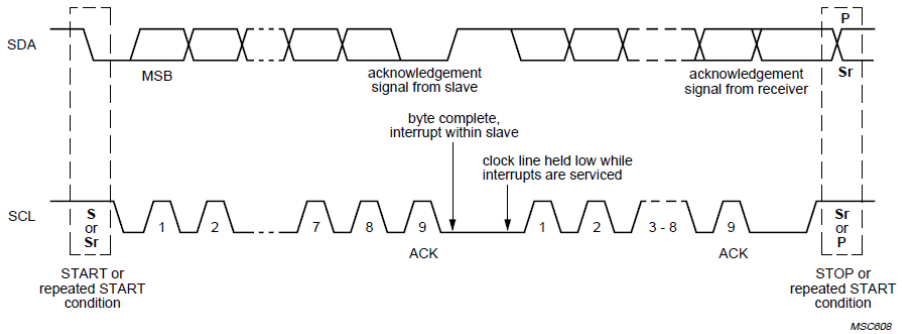


Figura 6.9 – Representação do Protocolo I2C para escrita de dados.
Fonte – PHILIPS, 2000.

Para implementar o componente Cfg_Vdec1, se usa uma máquina de estados finitos. A Figura 6.10 mostra o fluxograma do funcionamento desta máquina de estado, que gera os sinais do Protocolo I2C para o processo de escrita de dados.

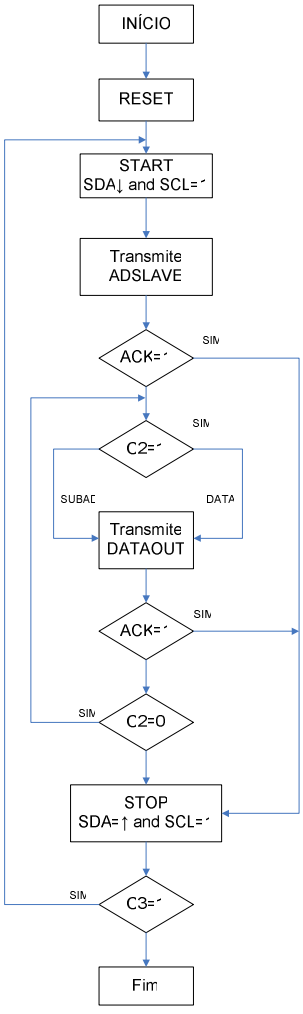


Figura 6.10 – Fluxograma da máquina de estados do Componente Cfg_Vdec1.
Fonte – Autor, 2010.

Esta máquina possui três variáveis de sinalização que auxiliam no controle. C1 controla a ordem de envio dos 8 *bits* do endereço e R/W, sub-endereço e dados. C2 determina o envio do sub-endereço ou dado. C3 controla a linha da matriz que está sendo enviada.

Os primeiros estados Ssi e Ss0 são responsáveis pelo reset do ADV7183B gerando um pulso de lógica '0' na saída RESET. Os dois próximos estados Ss1 e Ss2 geram a condição de START do protocolo I2C.

Os estados Ss3, Ss4, Ss5 e Ss6 carregam o SDA com o vetor ADSLAVE, bit por bit. O vetor de 8 bits ADSLAVE contém o valor '01000000', onde os 7 primeiros *bits* são o endereço do *chip* ADV7183B e o último é o *bit* de escrita.

Os estados Ss7, Ss8, Ss9, Ss10 Ss11 realizam o teste de ACK. Inicialmente o SDA é deixado em alta impedância para receber o sinal de ACK do decodificador. Se os dados carregados são válidos o ADV7183D sinaliza com lógica '0' o SDA, durante o próximo período de clock de SCL.

O estado Ss12, Ss13,Ss14 e Ss15 controla que tipo de informação é carregada no vetor DATAOUT, através da variável de sinalização C2. Se a variável C2 = 0 o vetor DATAOUT é carregado com um sub-endereço, se C2 = 1, DATAOUT é carregado com um dado. Os estados Ss13, Ss14 e Ss15 carregam o SDA com os 8 *bits* do vetor DATAOUT, bit a bit.

O estado Ss16, Ss17, Ss18 e Ss19, são responsáveis por esperar o *bit* de reconhecimento, o estado Ss20 verifica a variável C2, se o valor é sub-endereço, a máquina volta para o Ss12, se não trata-se de um dado. Os estados Ss21 e Ss22, são responsáveis pela condição de Stop do protocolo I2c.

O estado Ss23, utilizando a variável C3, determina qual linha das matrizes está sendo enviada pelo protocolo. Duas matrizes foram criadas dentro do CFG_VDEC1, uma onde são armazenados os sub-endereços e uma onde são guardados os dados de configuração. O registrador com o sub-endereço 00H determina qual o tipo de entrada que é usada e os formatos suportados por essa. Foi carregado nesse registrador o valor 04H, que habilita a entrada de Vídeo Composto e o formato NTSC. E os demais registradores foram deixados com o seu valor de configuração inicial.

O estado Ss24 é responsável por indicar o final da programação da VDEC1, levando a lógica '1' a saída CFGEND.

A descrição completa do componente CFG_VDEC1 em VHDL pode ser encontrada no APÊNDICE A.

6.4 Componente VGA

O Componente VGA é responsável por exibir imagem no monitor de vídeo, são necessários os sinais de sincronismo horizontal, vertical e os sinais de RGB. Na Figura 6.11 pode-se observar o bloco diagrama do Componente VGA, e os pinos de entrada e saída deste componente.

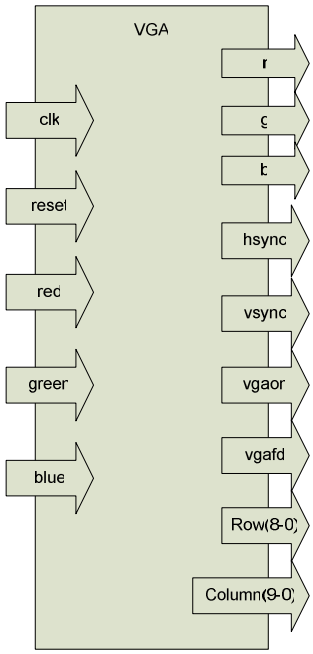


Figura 6.11 – Bloco diagrama do Componente VGA.
Fonte – Autor, 2010.

Para gerar os sinais de temporização da resolução VGA, foram sintetizados dois contadores, um para o sincronismo horizontal e outro para o sincronismo vertical. O contador do sincronismo horizontal (hcount) é incrementado a cada pulso de clock, enquanto que o sincronismo vertical (vcount) é incrementado a cada estouro do contador hcount. Os contadores foram implementados de forma a implementar os ciclos da Figura 6.12.

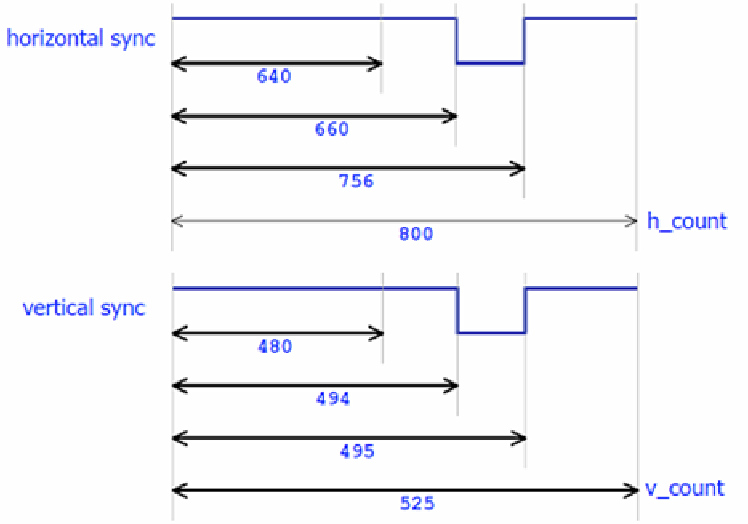


Figura 6.12 – Diagrama de sincronismo para resolução VGA.
Fonte – Autor, 2010.

Para gerar os pulsos de sincronismo é necessário observar os tempos em que os sinais do VGA devem ficar em lógica '1' ou '0'. O código VHDL do contador horizontal e vertical podem ser visto a seguir.

```
hcounter: process (clk, reset)
                   begin
                   if reset='1'
                      then hcount <= (others => '0');
                      else if (clk'event and clk='1')
                              then if hcount=799
                                    then hcount <= (others => '0');
                                    else hcount <= hcount + 1;
                                end if;
                      end if;
                    end if;
   end process;
vcounter: process (clk, reset)
    begin
    if reset='1'
        then vcount <= (others => '0');
        else if (clk'event and clk='1')
             then if hcount=699
                    then if vcount=524
                        then vcount <= (others => '0');
                        else vcount <= vcount + 1;
                     end if;
              end if;
         end if;
       end if;
end process;
```

O contador horizontal conta ao total 800 ciclos de clock , dos quais 640 são a região ativa do pixel. O contador vertical conta 525 ciclos de sincronismo horizontal para gerar os pulsos de vertical, dos quais 480 representam a região ativa das linhas. Para gerar o sincronismo horizontal e vertical é usado um contador de 700 ciclos de clock, após a

contagem destes 700 ciclos inicia a contagem do sincronismo vertical, isso possibilita o correto sincronismo entre eles. O processo de sincronismo horizontal mantém o hsync em nível lógico '1' do estado de contagem 0 até 659, e durante o tempo de contagem 659 à 755 o hsync é colocado em nível lógico '0', o que corresponde a 3,77µs. Para o sincronismo vertical o contador mantém o vsync em nível lógico '0' apenas da contagem de 493 à 494, o que corresponde a 64µs, no restante da contagem o nível lógico de vsync é '1'. A seguir está o código para gerar os sinais de sincronismo horizontal e vertical.

```
sync: process (clk, reset)
       begin
       if reset='1'
           then hsync <= '0';
                vsync <= '0';
           else if (clk'event and clk='1')
                    then if (hcount<=755 and hcount>=659)
                         then hsync <= '0';
                         else hsync <= '1';
                    end if;
                    if (vcount<=494 and vcount>=493)
                        then vsync <= '0';
                        else vsync <= '1';
                     end if;
             end if;
         end if;
         end process;
```

Além do sincronismo horizontal e vertical também é necessário gerar a região ativa do vídeo, ou seja, apenas 640 dos *pixels* horizontais e 480 dos *pixels* verticais serão exibidos, para tanto é necessário que se obedeça aos tempos da resolução VGA. A seguir está representado o código VHDL responsável por gerar a região ativa do *pixel*.

```
process (hcount)
   begin
   videoh <= '1';
```

```
   column <= hcount;
   if hcount>639
      then videoh <= '0';
           column <= (others => '0');
   end if;
   end process;
process (vcount)
    begin
    videov <= '1';
    row <= vcount(8 downto 0);
    if vcount>479
       then videov <= '0';
            row <= (others => '0');
     end if;
     end process;
videoon <= videoh and videov;
vgaon      <= videoon;
  vgafd  <= videov;
    colors: process (clk, reset)
            begin
            if reset='1'
               then r <= '0';
                    g <= '0';
                    b <= '0';
               elsif (clk'event and clk='1')
                  then r <= red and videoon;
                       g <= green and videoon;
                       b <= blue and videoon;
                end if;
                end process;
  end synt;
```

Os pinos de saída VGAON e VGAFD, contém respectivamente o sinal de videoon e videov, estes sinais são usados para controle do endereçamento de leitura da memória.

O código VHDL complete do componente VGA pode-se visto no Apêndice A.

6.5 Componente Mem_Vga_300K

O Componente Mem_Vga_300K tem como função armazenar os dados digitais provenientes do Módulo de Decodificação e disponibiliza-los ao VGA. A Figura 6.13 mostra o bloco diagrama da Mem_Vga_300K.

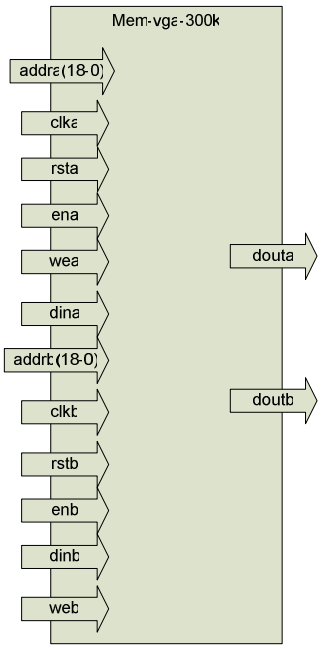


Figura 6.13 – Bloco diagrama do Componente Mem_Vga_300K.
Fonte – Autor, 2010.

Para a construção deste componente foram usados os blocos dedicados para memória disponíveis no FPGA. A Família Spartan 3E apresentam diferentes quantidades de blocos de memória, que variam de acordo com o tamanho do FPGA. Para o FPGA XC3S500E estão disponíveis 20 blocos de memória cada um com 2304 bytes. A Tabela 5, mostra a distribuição dos blocos e a quantidade de memória disponível para os diferentes FPGAs da família Spartan-3E (XILINX, 2006a).

Tabela 5 – Quantidade de RAM Blocks para diferentes FPGAs da Família Spartan-3E.

Device	RAM Columns	RAM Blocks per Column	Total RAM Blocks	Total RAM Bits	Total RAM Kbits
XC3S100E	1	4	4	73728	72
XC3S250E	2	6	12	221184	216
XC3S500E	2	10	20	368640	360
XC3S1200E	2	14	28	516096	504
XC3S1600E	2	18	36	663552	648

Fonte – XILINX, 2005b.

Estes blocos de memória podem ser agrupados e configurados para obter diferentes larguras e profundidades para formar a memória desejada para o projeto. Para poder usar estes RAM Blocks no projeto, é necessário utilizar as bibliotecas primitivas de memória. Dessa forma ao descrever a memória instanciam-se primeiro o tamanho da memória em Kbits e se é porta única ou dupla. A Tabela 6 mostra as características específicas de um grupo das bibliotecas primitivas de RAM Blocks de 16Kbits.

Tabela 6 – Primitivas de RAM Blocks de 16Kbits.

Organization	Memory Depth	Data Width	Parity Width	DI/DO	DIP/DOP	ADDR	Single-Port Primitive	Total RAM Kbits
512x36	512	32	4	(31:0)	(3:0)	(8:0)	RAMB16_S36	18K
1Kx18	1024	16	2	(15:0)	(1:0)	(9:0)	RAMB16_S18	18K
2Kx9	2048	8	1	(7:0)	(0:0)	(10:0)	RAMB16_S9	18K
4Kx4	4096	4	-	(3:0)	-	(11:0)	RAMB16_S4	16K
8Kx2	8192	2	-	(1:0)	-	(12:0)	RAMB16_S2	16K
16Kx1	16384	1	-	(0:0)	-	(13:0)	RAMB16_S1	16K

Fonte – XILINX, 2005b.

Uma memória com porta dupla (*dual port*) permite o acesso simultâneo para leitura e escrita, ou de duas leituras por dois dispositivos diferentes. A Figura 6.14 demonstra uma biblioteca primitiva genérica de uma RAM Block 16Kbits de porta dupla, onde podem ser visualizados os barramentos de endereçamento e dados, e os pinos de controle da memória.

Figura 6.14 – Diagrama gebérico de uma RAM Block de 16K bits primitiva.
Fonte – XILINX, 2002.

O Diagrama de tempo do funcionamento destes blocos de memória para os processos de escrita ou de leitura de dados pode ser observado na Figura 6.15.

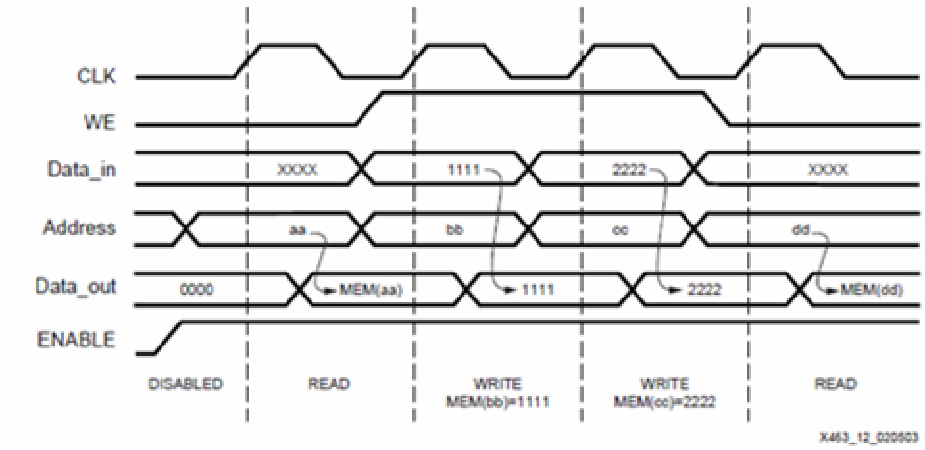


Figura 6.15 – Diagrama de tempo do processo de escrita e leitura na RAM Block.
Fonte – XILINX, 2005b.

O funcionamento da memória tem como referência os pulsos de clock da memória. Para o processo de escrita, os sinais de habilitação de escrita e de habilitação do *chip* devem estar devidamente acionados. O processo de escrita exige que o endereçamento da memória e

os dados na entrada já estejam definidos quando o clock ocorrer (XILINX, 2005b). Todos os limites de tempos de escrita e endereçamento podem ser verificados na Tabela 7.

Tabela 7 – Tempos de escrita e endereçamento das RAM Blocks da Família Spartan 3E.

Symbol	Description	Speed Grade				Units
		-5		-4		
		Min	Max	Min	Max	
Clock-to-Output Times						
T_{BCKO}	When reading from block RAM, the delay from the active transition at the CLK input to data appearing at the DOUT output	-	2.45	-	2.82	ns
Setup Times						
T_{BACK}	Setup time for the ADDR inputs before the active transition at the CLK input of the block RAM	0.33	-	0.38	-	ns
T_{BDCK}	Setup time for data at the DIN inputs before the active transition at the CLK input of the block RAM	0.23	-	0.23	-	ns
T_{BECK}	Setup time for the EN input before the active transition at the CLK input of the block RAM	0.67	-	0.77	-	ns
T_{BWCK}	Setup time for the WE input before the active transition at the CLK input of the block RAM	1.09	-	1.26	-	ns
Hold Times						
T_{BCKA}	Hold time on the ADDR inputs after the active transition at the CLK input	0.12	-	0.14	-	ns
T_{BCKD}	Hold time on the DIN inputs after the active transition at the CLK input	0.12	-	0.13	-	ns
T_{BCKE}	Hold time on the EN input after the active transition at the CLK input	0	-	0	-	ns
T_{BCKW}	Hold time on the WE input after the active transition at the CLK input	0	-	0	-	ns
Clock Timing						
T_{BPWH}	High pulse width of the CLK signal	1.39	-	1.59	-	ns
T_{BPWL}	Low pulse width of the CLK signal	1.39	-	1.59	-	ns
Clock Frequency						
F_{BRAM}	Block RAM clock frequency. RAM read output value written back into RAM, for shift-registers and circular buffers. Write-only or read-only performance is faster.	0	270	0	230	MHz

Fonte – XILINX, 2008.

De acordo com a necessidade do projeto, analisando o manual do FPGA necessitou-se criar uma memória com tamanho de 16 x 1. Observando a Tabela 5, escolheu-se para a construção do componente Mem_Vga_300K, o RAMB16_S1_S1 que possui porta dupla, e cada bloco possui 16384 bits de memória.

Para capturar um quadro inteiro da imagem fornecida pela câmera monocromática com uma resolução de 640 *pixels* horizontais por 480 linhas verticais do Módulo de Captura,

considerando ainda que cada *pixel* é quantizado em 1 *bit* no Módulo de Decodificação, seriam necessários 307200 *bits* de memória para uma captura sem perda de informação. A memória total disponível no FPGA XC3S500E é de 360 Kbits. Dessa maneira optou-se por utilizar 19 blocos de 16384 *bits*, totalizando 310612 *bits*. O código em VHDL do componente RAMB16_S1_S1 pode ser visto a seguir:

```
component RAMB16_S1_S1
   port (DOA0 : out STD_LOGIC;
         DOB0 : out STD_LOGIC;
         ADDRA : in STD_LOGIC_VECTOR (13 downto 0);
         ADDRB : in STD_LOGIC_VECTOR (13 downto 0);
         CLKA : in STD_LOGIC;
         CLKB : in STD_LOGIC;
         DIA0 : in STD_LOGIC;
         DIB0 : in STD_LOGIC;
         ENA : in STD_LOGIC;
         ENB : in STD_LOGIC;
         SSRA : in STD_LOGIC;
         SSRB : in STD_LOGIC;
         WEA : in STD_LOGIC;
         WEB : in STD_LOGIC);
end component;
```

Para obter-se a memória de 307200 *bits*, foram necessários 19 destes RAM blocks. O primeiro processo descrito no componente Mem_Vga_300K tem a função de unir estes 19 blocos de memória para que os demais componentes do projeto reconheçam somente como uma única memória. Para isso foram acrescidos 5 *bits* ao endereçamento de memória. Os 5 *bits* mais significativos do endereçamento do componente Mem_Vga_300K servem como um seletor para os 19 RAM Blocks instanciados. O código VHDL para a escrita dos RAM Blocks é mostrado a seguir:

```
enan_0 <= '1' when (addra (18 downto 14) = "00000") and (ena = '1') else '0';
```

```
       enan_1 <= '1' when (addra (18 downto 14) = "00001")  and (ena = '1')
else '0';
       enan_2 <= '1' when (addra (18 downto 14) = "00010") and (ena = '1')
else '0';
       enan_3 <= '1' when (addra (18 downto 14) = "00011") and (ena = '1')
else '0';
       enan_4 <= '1' when (addra (18 downto 14) = "00100") and (ena = '1')
else '0';
       enan_5 <= '1' when (addra (18 downto 14) = "00101") and (ena = '1')
else '0';
       enan_6 <= '1' when (addra (18 downto 14) = "00110") and (ena = '1')
else '0';
       enan_7 <= '1' when (addra (18 downto 14) = "00111") and (ena = '1')
else '0';
       enan_8 <= '1' when (addra (18 downto 14) = "01000") and (ena = '1')
else '0';
       enan_9 <= '1' when (addra (18 downto 14) = "01001") and (ena = '1')
else '0';
       enan_10 <= '1' when (addra (18 downto 14) = "01010") and (ena = '1')
else '0';
       enan_11 <= '1' when (addra (18 downto 14) = "01011") and (ena = '1')
else '0';
       enan_12 <= '1' when (addra (18 downto 14) = "01100") and (ena = '1')
else '0';
       enan_13 <= '1' when (addra (18 downto 14) = "01101") and (ena = '1')
else '0';
       enan_14 <= '1' when (addra (18 downto 14) = "01110") and (ena = '1')
else '0';
       enan_15 <= '1' when (addra (18 downto 14) = "01111") and (ena = '1')
else '0';
       enan_16 <= '1' when (addra (18 downto 14) = "10000") and (ena = '1')
else '0';
       enan_17 <= '1' when (addra (18 downto 14) = "10001") and (ena = '1')
else '0';
       enan_18 <= '1' when (addra (18 downto 14) = "10010") and (ena = '1')
else '0';
```

Quando estes 5 *bits* do endereçamento selecionam um RAM Block no processo de leitura, este bloco de memória liga o seu barramento de saída ao barramento de saída do

componente Mem_Vga_300K. Do mesmo modo, no processo de escrita, o bloco de memória liga o seu barramento de entrada ao barramento de entrada do componente. O código VHDL para a leitura dos RAM Blocks é mostrado a seguir:

```
process(addrb)
begin
    if (addrb (18 downto 14) =    "00000") then
            doutb <= dobn_0;
    elsif (addrb (18 downto 14) = "00001") then
            doutb <= dobn_1;
    elsif (addrb (18 downto 14) = "00010") then
            doutb <= dobn_2;
    elsif (addrb (18 downto 14) = "00011") then
            doutb <= dobn_3;
     elsif (addrb (18 downto 14) = "00100") then
            doutb <=dobn_4;
     elsif (addrb (18 downto 14) = "00101") then
            doutb <=dobn_5;
     elsif (addrb (18 downto 14) = "00110") then
            doutb <= dobn_6;
     elsif (addrb (18 downto 14) = "00111") then
            doutb <= dobn_7;
     elsif (addrb (18 downto 14) = "01000") then
            doutb <= dobn_8;
     elsif (addrb (18 downto 14) = "01001") then
            doutb <= dobn_9;
     elsif (addrb (18 downto 14) = "01010") then
            doutb <= dobn_10;
     elsif (addrb (18 downto 14) = "01011") then
            doutb <= dobn_11;
     elsif (addrb (18 downto 14) = "01100") then
            doutb <= dobn_12;
     elsif (addrb (18 downto 14) = "01101") then
            doutb <=dobn_13;
     elsif (addrb (18 downto 14) = "01110") then
            doutb <=dobn_14;
     elsif (addrb (18 downto 14) = "01111") then
            doutb <=dobn_15;
     elsif (addrb (18 downto 14) = "10000") then
            doutb <=dobn_16;
     elsif (addrb (18 downto 14) = "10001") then
            doutb <=dobn_17;
```

```
     elsif (addrb (18 downto 14) = "10010") then
            doutb <=dobn_18;
      end if;
end process;
```

A descrição completa do componente pode ser encontrada no APÊNDICE A.

6.6 Componente Cap_Ctl

O componente Cap_Ctl é responsável pelo controle da captura das imagens, ele utiliza os sinais vindos do Módulo de Decodificação, a partir deste sinais controla o inicio da gravação do componente Mem_Vga_300K. A Figura 6.16 mostra o diagrama em bloco do componente Cap_Ctl.

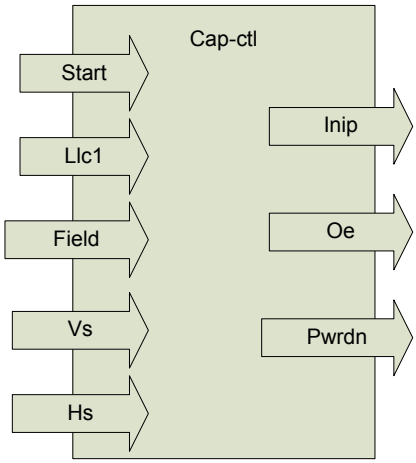


Figura 6.16 – Bloco diagrama do Componente Cap_Ctl.
Fonte – Autor, 2010.

A entrada `Start` recebe um pulso de sinal gerado pelo ROTARY_PRESS da Plataforma de Prototipação, para iniciar a máquina de estados. As entradas Fiels, Vs e Hs recebem os sinais do Módulo de Decodificação. `Llc1` recebe um sinal de clock de 27MHz vindo do componente DCM_2xLLC1 . Field recebe o sinal que diferencia os 2 campos da

imagem. Quando '0' trata-se do primeiro campo e quando Field recebe '1' trata-se do segundo campo. A entrada Vs recebe o sinal de VSYNC e a entrada Hs recebe o sinal de HSYNC.

A saída Oe envia o sinal de *Output Enable* e a saída Pwrdn envia o sinal de *Stand by* para o ADV7183B. Os sinais de Oe e Pwrdn tem valores fixos, sempre em lógica '0' e '1' respectivamente, para que a placa decodificadora esteja sempre ativa.

Este componente foi implementado por uma máquina de estados finitos, a Figura 6.17 mostra o fluxograma desta máquina de estados.

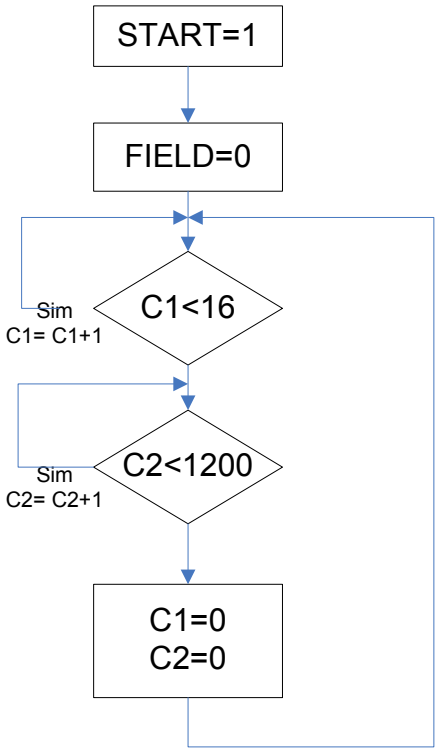


Figura 6.17 – Fluxograma da máquina de estados Cap_Ctl.
Fonte – Autor, 2010.

A máquina permanece no estado S0 até que receba um pulso de Start, então zerar as variáveis C1 e C2 e salta para o estado S1. Estas variáveis auxiliam no controle da máquina, C1 é uma variável que conta o número de pulsos de HSYNC até a primeira linha do campo,

Estes números de HSYNC variam dependendo do campo. Por isso o primeiro campo inicia com C1 = 0 e o segundo campo começa com C1 = 1. C2 é uma variável que conta o número de clocks até a região ativa de pixels. Segundo o manual do chip ADV7183B, após 276 pulsos de clock está a região ativa de pixels. No entanto C2 deve contar 1200 ciclos de clock, pois é endereçado apenas o campo Field par. C2 é uma variável que conta o número de clocks para gerar a saída inip, que informar a região ativa do pixel, este pulso é utilizado pela máquina de estados Machine_MemA, para controlar a escrita dos dados na memória. Os estados S1 e S2 detectam a borda de descida do sinal de Field para iniciar a captura no campo par. Os estados S3 e S4 contam os pulsos de clock até o início da primeira linha, a variável C1 é incrementada uma vez por pulso, e quando C1 = 15 então a máquina salta para o estado S5. No estado S5, enquanto a variável C2 < 1200, a máquina incrementa C2 e repete-se o estado S5 a cada pulso de clock de LLC1. Quando C2 = 1199, a máquina e coloca a saída Inip em '0'.Informando que a região ativa do pixel. Então a máquina salta para S6. O estado S6 tem a função colocar o Inip em '1', deixando de escrever dados na memória. S7 é o estado onde as variáveis são zeradas e a máquina salta para S1.

A descrição completa em VHDL do componente Cap_Ctl se encontra no APÊNDICE A.

6.7 Componente VDECTOVGA

O componente VDECTOVGA, foi a primeira tentativa de sincronismo entre o Módulo de Decodificação e o monitor VGA. Contudo não se obteve os resultados esperados, em virtude das freqüências horizontal e vertical serem incompatíveis com o padrão VGA.

O componente VGA foi desenvolvido para operar com uma freqüência de 25MHz, contudo testes realizados comprovaram que este componente também funciona com a freqüência de 27MHz gerada pela VDEC1, com base nestes experimentos realizados na Plataforma de Prototipação, e sabendo que para o correto funcionamento dos componentes VGA e VDEC1, estes deveriam estar sincronizados, desenvolveu-se então o componente VDECTOVGA, este componente foi desenvolvido utilizando quatro máquinas de estado, com o objetivo de gerar os sinais de sincronismo horizontal e vertical e a região ativa vertical e horizontal do vídeo, tomando como referência os sinais FIELD, HSYNC, VSYNC e freqüência LLC1, gerados pela VDEC1. A Figura 6.18 mostra o diagrama em blocos deste componente.

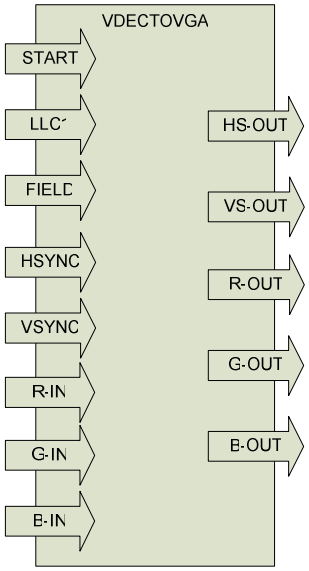


Figura 6.18 – Diagrama em bloco do Componente VDECTOVGA.
Fonte – Autor, 2010.

Quatro máquinas de estado, Sreg1, Sreg2, Sreg3 e Sreg4, foram implementadas de forma a gerar os sinais de sincronismo da resolução VGA (640x480). Para gerar os tempos de sincronismo da VGA se criou onze variáveis para auxiliar na geração dos sinais. C1, C2 e C3 são as variáveis para o sincronismo horizontal, usadas para gerar os tempos de 3,77µs, 28µs e

3,77µs respectivamente. As variáveis C4, C5, C6 e C7 são responsáveis por gerar o sincronismo vertical, C4 e C6 para indicar o início do sincronismo e C5 e C7 para gerar o tempo de 64µs. O sinal ativo do vídeo horizontal é controlado através das variáveis C8 e C9, onde C8 gera o tempo de 1,79µs para iniciar o sinal do vídeo e C9 controla o tempo de vídeo ativo de 25,42µs. As variáveis C10 e C11 são usadas para gerar a região ativa vertical do vídeo, C10 para gerar 1020µs para o início da região e C11 para determinar 15250µs referente ao tempo de vídeo ativo vertical.

A primeira máquina de estados Sreg1 é responsável por gerar o sincronismo horizontal, ela utiliza o próprio sincronismo horizontal da VDEC1 para gerar o sincronismo horizontal do VGA, também e preciso gerar um pulso intermediário aos pulsos de sincronismo horizontal da VDEC1, como mostra a Figura 6.19, 0nde HS_OUT é o sinal gerado pela máquina e HSYNC é o sincronismo da VDEC1.

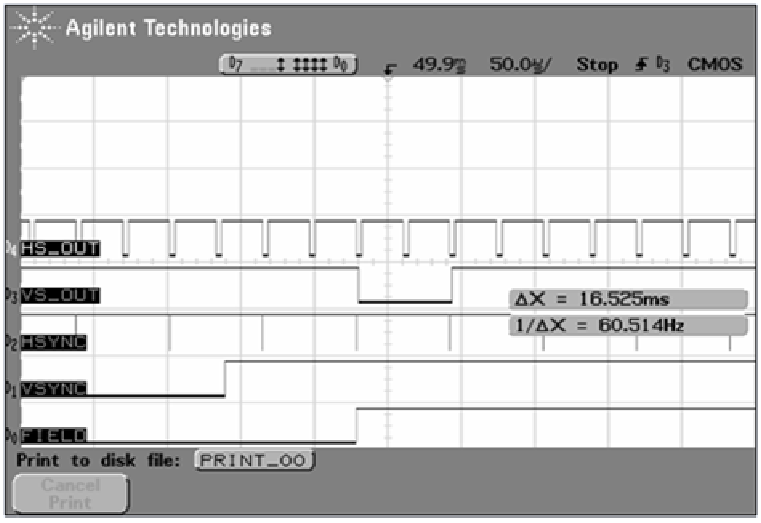


Figura 6.19 – Sinal de sincronismo horizontal e vertical do Componente VDECTOVGA.
Fonte – Autor, 2010.

A máquina de estado Sreg2 tem por finalidade gerar o sincronismo vertical, se baseando nos sinais de FIELD e VSYNC gerados pela VDEC1. É necessário analisar os sinais

de FIELD e VSYNC e gerar um atraso no sinal do sincronismo, pois o início e final do sinal VSYNC não coincide com as bordas do sinal FIELD.

Sreg3 é a máquina de estados desenvolvida para gerar sinal da região ativa horizontal do vídeo. Esta máquina utilizava duas variáveis, C8 para contar o início da região ativa , e C9 para manter a região ativa. Esta máquina utiliza o sinal HS_OUT para gerar a região ativa horizontal do vídeo. Na Figura 6.20 é possível analisar a região ativa do vídeo horizontal.

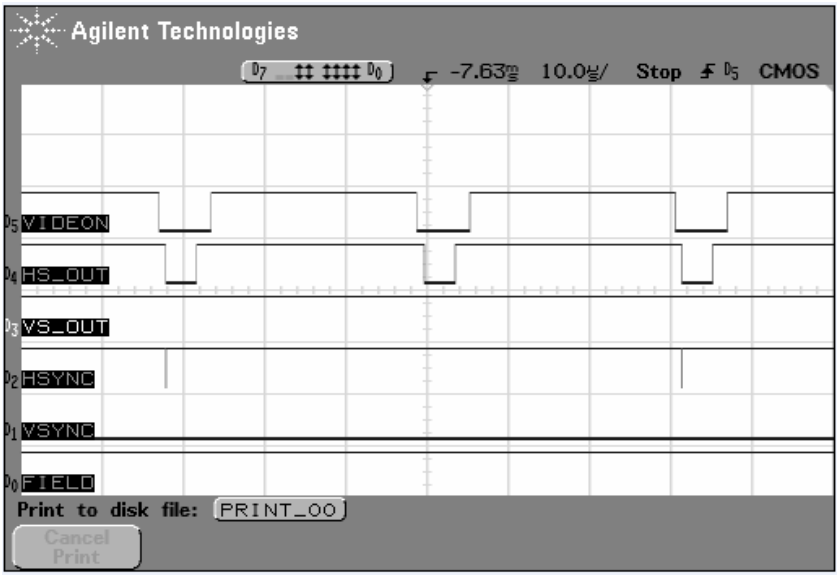


Figura 6.20 – Imagem da captura do sinal ativo horizontal do vídeo.
Fonte – Autor, 2010.

Sreg4 foi a máquina desenvolvida para controlar a região ativa vertical do vídeo. Da mesma forma que Sreg3 utiliza duas variáveis, C10 para iniciar a região ativa e C11 para manter o sinal ativo, utilizando o sinal VS_OUT.

Na Figura 6.19 é possível observar que o sinal de sincronismo vertical não segue o padrão VGA, pois a freqüência vertical é de 60,514Hz, sendo que a freqüência correta seria de 59,58Hz. A freqüência que o monitor de vídeo suporta deve seguir os tempos das

resoluções padrões, o fato de se conseguir gerar a freqüência vertical de 60,514Hz por si só não impossibilitaria a visualização de imagens no monitor, contudo como pode ser analisado na Figura 6.21 a freqüência horizontal é 31,60KHz, que segue o padrão VGA, desta forma não existe a possibilidade das duas freqüências serem utilizadas na mesma resolução. Este motivo impossibilitou a continuação do projeto baseado no princípio de gerar os sinais VGA a partir dos sinais gerados pela VDEC1, já que o sincronismo horizontal ficou correto e o vertical não permitiu ser ajustado.

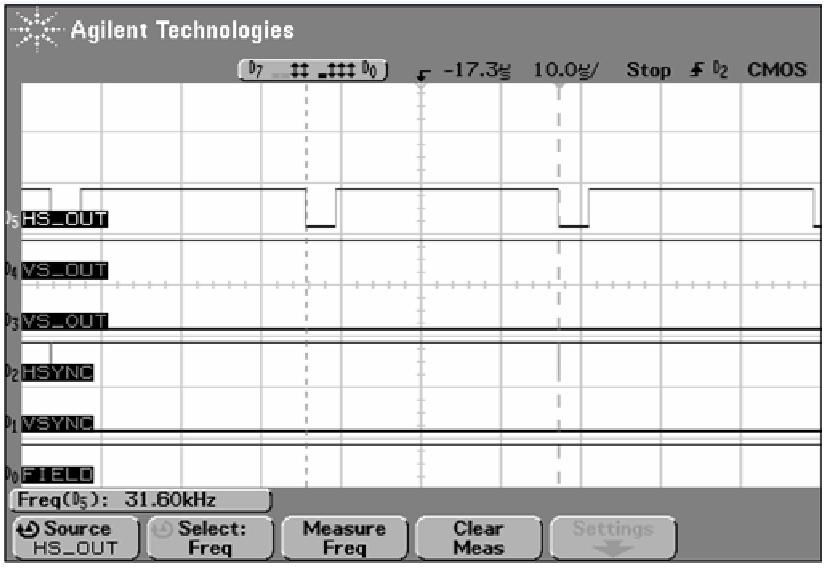


Figura 6.21 – Imagem da freqüência horizontal gerada pelo componente VDECTOVGA.
Fonte – Autor, 2010.

No Apêndice B está o código VHDL do componente VDECTOVGA. Colocar imagem do HS_OUT

7 TESTE de VALIDAÇÃO DO SISTEMA

Para verificar o funcionamento do protótipo, se optou por dividir os teste em módulos.Desta forma se pode verificar o funcionamento individual dos componentes do projeto, facilitando o trabalho e verificar eventuais anomalias.

Sendo assim se dividiu os testes em três grupos, um valida todo o hardware referente ao Módulo de Captura e Decodificação, o outro valida o Módulo de Controle e Armazenamento, e o terceiro valida o Módulo de Exibição das imagens.

7.1 Teste do Módulo de Captura

O módulo de captura é composto pela câmera Kodo e a fonte que fornece energia à ela. Na Figura 7.1 verifica-se o teste do funcionamento da fonte da câmera. Com o uso de uma fonte da Agilent modelo E3631A, regulada para fornecer 12VDC para alimentar a fonte da câmera, é possível com o auxílio de um multímetro da Minipa modelo ET-1600 verificar 9,58VDC na saída da fonte que alimenta a câmera, o que determina o perfeito funcionamento da fonte, já que a câmera de acordo com o manual opera na faixa de 7,5VDC à 14,5VDC, também é possível verificar o consumo de 119mA de todo o conjunto do módulo de captura.

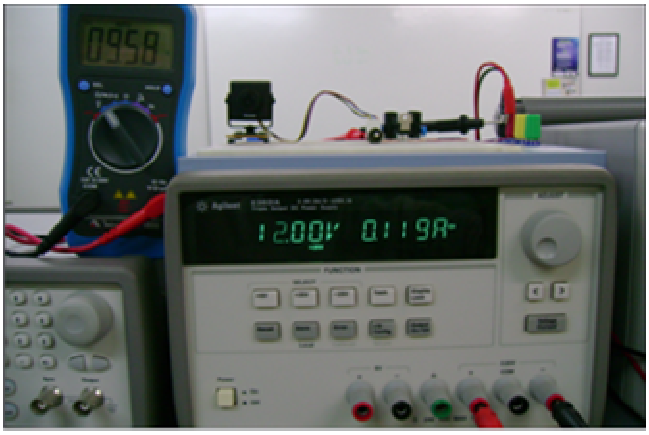


Figura 7.1 – Teste da fonte do Módulo de Captura.
Fonte – Autor, 2010.

Para testar a câmera utilizou-se uma folha com barras brancas e pretas disposta em frente a câmera, a imagem visualizada pode ser vista na Figura 7.2.

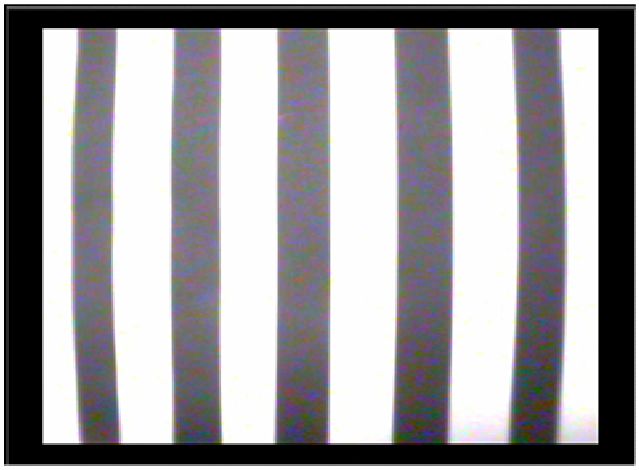


Figura 7.2 – Imagem de barras capturadas pela câmera.
Fonte – Autor, 2010.

Com o auxílio de um osciloscópio da Agilent mod. 54622D se avaliou a característica do sinal da câmera. Na figura 7.3 pode se observar que a imagem capturada pela câmera corresponde as barras colocadas diante dela.

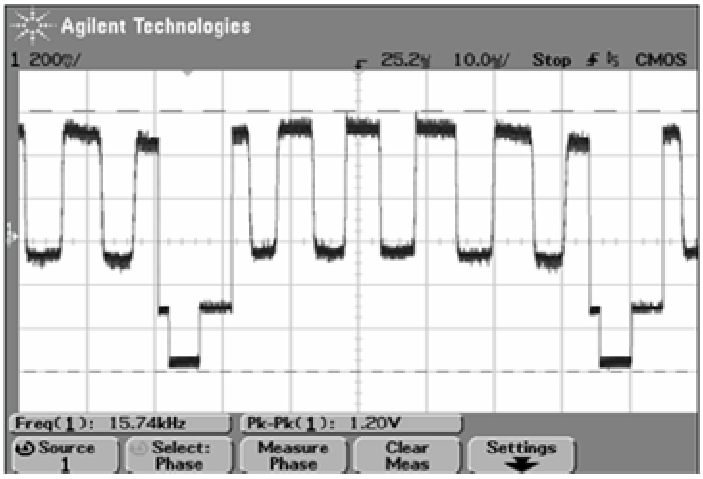


Figura 7.3 – Imagem de uma linha de vídeo capturadas pelo osciloscópio.
Fonte – Autor, 2010.

É possível observar na forma de onda do osciloscópio da Figura 7.3, seis formas de ondas altas, que correspondem as barras brancas da imagem capturada e cinco formas de ondas baixas que correspondem às cinco barras pretas. A freqüência de uma linha medida no osciloscópio foi de 15,74 KHz. No estudo realizado este valor é de aproximadamente 15,734 KHz. Com estes testes se pôde confirmar o formato de vídeo NTSC fornecido pela câmera como também o funcionamento do Módulo de Captura para fornecer o sinal adequado ao Módulo de Decodificação.

7.2 Teste do Módulo de Decodificação

Para testar o módulo de decodificação que é composto pela placa decodificadora de vídeo VDEC1, usa-se o osciloscópio para verificar os sinais de sincronismos fornecidos pela placa decodificadora em relação à entrada do sinal de vídeo. Na Figura 7.4 se observa o sinal de FIELD que diferencia os campos par e ímpar, o sinal V_SYNC, referente ao sincronismo vertical, e o sinal de VIDEO fornecido pela câmera.

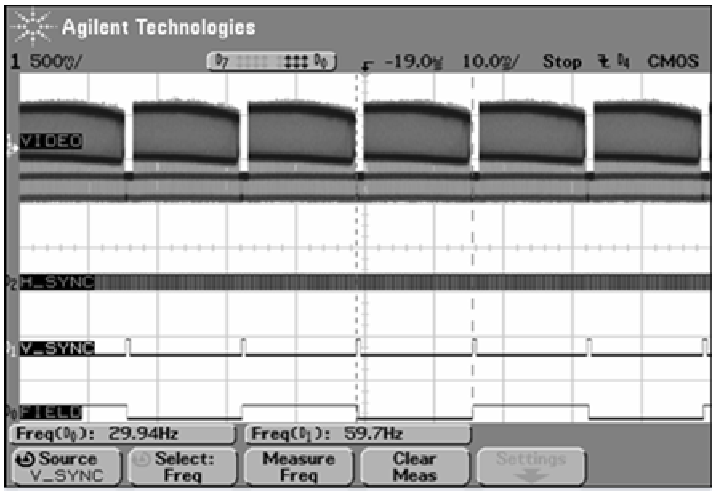


Figura 7.4 – Imagem do sincronismo da VDEC1 em relação ao sinal de vídeo da câmera.
Fonte – Autor, 2010.

Observa-se que para cada campo par e ímpar ocorre também o sinal de pulso de sincronismo vertical V_SYNC, que ocorre a cada mudança de campo. É possível também ver o pulso de H_SYNC de sincronismo horizontal que delimita cada linha dos campos. A freqüência do FIELD é de 29,94Hz e V_SYNC é de 59,7Hz, o que caracteriza o formato de vídeo NTSC. FIELD possui uma freqüência de 29,94 campos por segundo, o que resultaria numa freqüência de 59,70Hz por quadro.

Com o auxílio do osciloscópio verificou-se um atraso da saída de VDEC1 em relação ao sinal de entrada de vídeo, isso pode se visto na Figura 7.5. Diante da câmera é colocado um papel com barras brancas e pretas, e desta forma se analisa o sinal de entrada de vídeo colocando a ponteira analógica do osciloscópio na saída da câmera e visualizando o sinal VIDEO da Figura 7.5. O sinal P_7 representa o bit mais significativo do *pixel* gerado pela VDEC1, desta forma se mede com o auxílio de cursores um atraso de 170µs, o que equivale a aproximadamente 2,5 linhas da imagem de atraso em relação ao sinal de entrada.

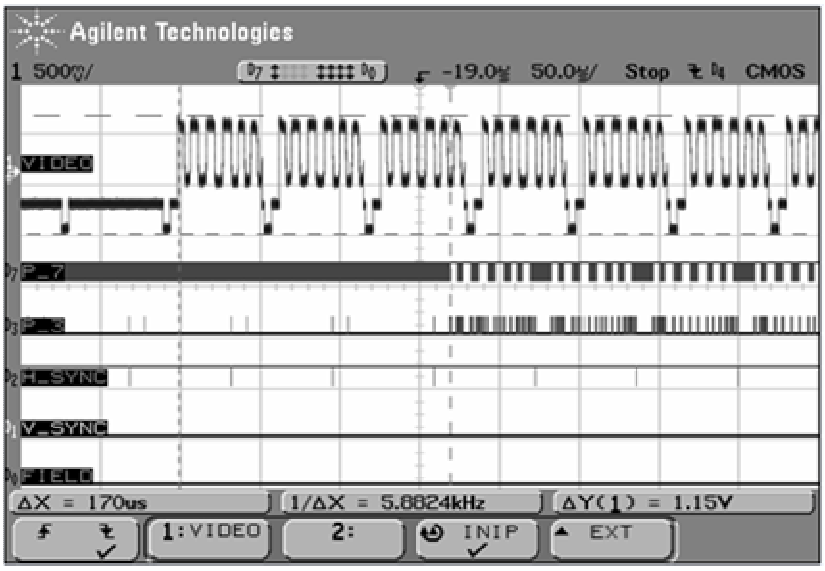


Figura 7.5 – Imagem do atraso do sinal de vídeo da VDEC1 em relação a entrada de vídeo
Fonte – Autor, 2010.

Este atraso é gerado porque a VDEC1 processar os sinais de vídeo recebidos, para posteriormente disponibiliza-los para o módulo de armazenamento.

7.3 Testes Módulo de Controle

Para decodificar os sinais de vídeo vindos do Módulo de Captura o Módulo de Controle deve ser programado através do protocolo I2C para configurar o chip ADV7183B da VDEC1. Na Figura 7.6 é possível verificar que ocorre uma mudança de nível lógico de '1' para '0' em SDA quando o SCL está em '1', o que corresponde ao sinal de START do protocolo I2C.

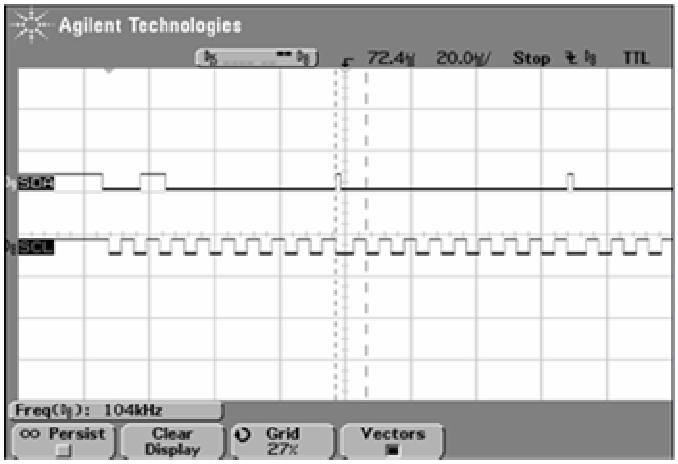


Figura 7.6 – Imagem do sinal de START do protocolo I2C.
Fonte – Autor, 2010.

De acordo com o manual a taxa de transferência do protocolo I2C é de até 100KHz. A Figura 7.7 mostra a taxa de transferência de 104KHz, e o sinal correspondente a condição de STOP, que ocorre quando há uma mudança de nível lógico de '0' para '1' de SDA quando o SCL está em nível lógico '1'.

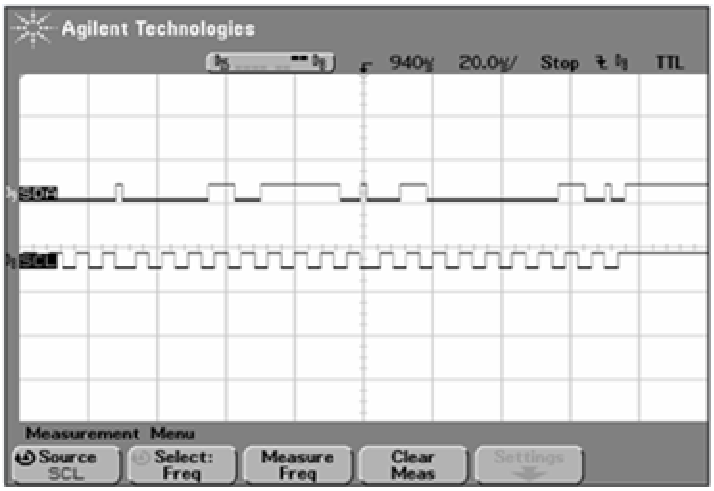


Figura 7.7 – Imagem da condição de STOP do protocolo I2C.
Fonte – Autor, 2010.

7.4 Testes Módulo de Armazenamento

Para validar o funcionamento do módulo de armazenamento, foi implementada uma máquina de estados que coloca no dado de entrada dina da memória o próprio endereço addra(6), desta forma durante 64 pulsos de clock a memória armazena '0', e durante 64 pulsos de clock ela armazena '1', mostrando 10 barras no monitor de vídeo, 5 brancas e 5 pretas. A Figura 7.8 mostra a imagem capturada, e validando o funcionamento do componente Mem_Vga_300K.

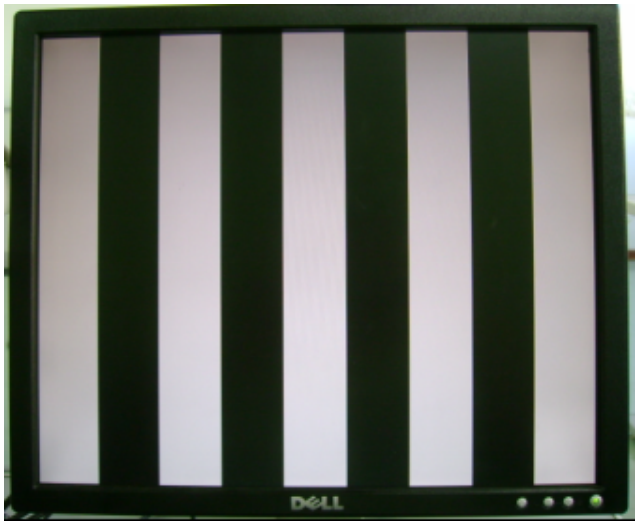


Figura 7.8 – Foto das barras armazenadas na memória.
Fonte – Autor, 2010.

Na figura 7.8 se observa as cinco barras brancas, porém apenas quatro barras pretas, mostrando que 64 *pixels* não estão sendo exibidos no monitor, isso ocorre porque o componente VGA está funcionando a freqüência de 27MHz fornecida pelo Componente DCM_2xLLC1, ao se utilizar a freqüência de 25MHz as dez barras são mostradas.

7.5 Testes Módulo de Exibição

Para verificar o funcionamento do componente VGA, foi criado um *Testbench* que gera os estímulos dos sinais de entrada clk, reset, red, green, e blue, para verificar os sinais de

saída r, g, b, hsync, vsync, row e column. Estes dois últimos sinais são os contadores responsáveis por gerar os sinais de sincronismo horizontal e vertical. A seguir está o código VHDL do *Testbench* do componente VGA.

```
vga0: vga PORT MAP (clk => tb_clk,
 reset_vga => tb_reset_vga,
 hsync => tb_hsync,
 vsync => tb_vsync,
 red => '1',
 green => '1',
 blue => '1' );
estimulo: PROCESS
   begin
      tb_reset_vga <='1';
      tb_clk <='0';
      wait for 10 ns;
      tb_reset_vga <='0';
      loop
         wait for 19 ns;
            tb_clk <= not tb_clk;
        end loop;
end PROCESS estimulo;
```

Na Figura 7.9 é possível observar que para cada incremento do contador de linha (row) é necessário ocorrer o estouro do contador de coluna (column).

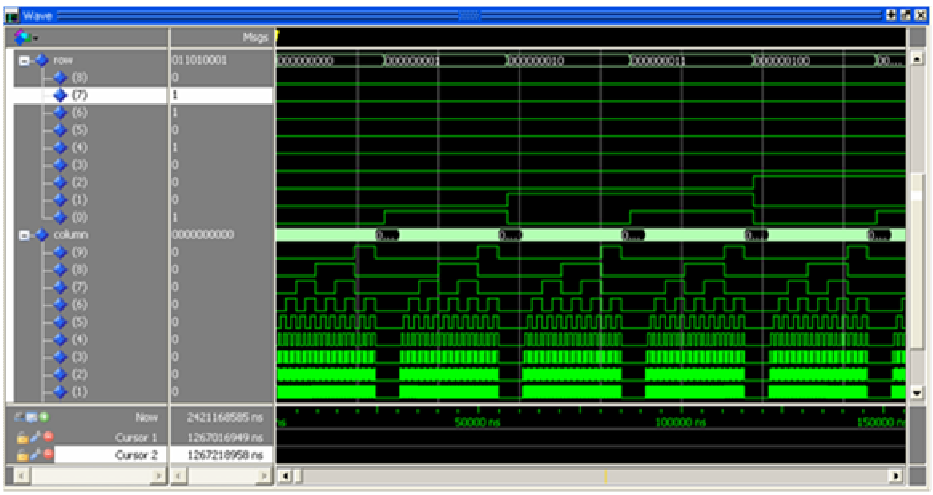


Figura 7.9 – Imagem do contador de linhas e colunas.
Fonte – Autor 2010.

Para testar o código VHDL na placa de prototipação, fixou-se a cor *red* em '0' e as cores *green* e *blue* em '1', o resultado é a cor *cyan*, na Figura 7.10 se observa o monitor de vídeo mostrando a imagem.

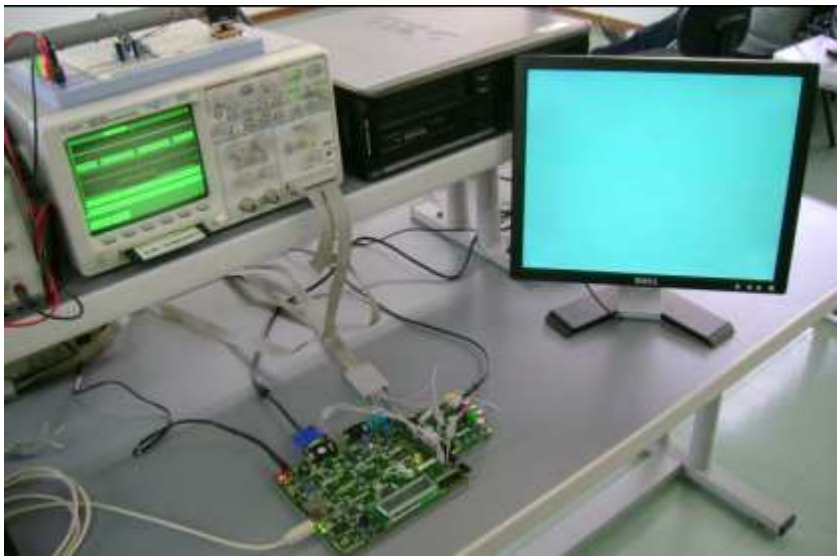


Figura 7.10 – Foto do monitor exibindo a cor *cyan*.
Fonte – Autor, 2010.

7.6 Testes Módulo de Validação

O teste do Módulo de Validação verifica o funcionamento do programa Proj_CapImg, para a leitura de dados armazenados na memória, e exibição no monitor de vídeo.

Neste teste de validação da funcionalidade do Sistema de Captura de Imagem, inicialmente é ligado o Módulo de Captura, e logo após pressionado o botão para o início da configuração do Módulo de Decodificação pelo Componente Cfg_Vdec1. A máquina de

estados Auto_rst recebe o pulso do botão ROTARY_PRESS e inicializa o Componente Cfg_Vdec1, quando este componente termina a configuração envia o sinal de final de configuração, cfgend, para a máquina Auto_rst e esta reseta as máquinas de controle (A e B) da Mem_Vga_300K, e o Componente VGA.

Desta forma inicia-se o processo de leitura da varredura da imagem capturada pela Câmera, sendo armazenado endereço após endereço, o bit mais significativo do pixel decodificado (256 níveis de cinza) pela VDEC1 e endereçado pela Machine_MemA, para o armazenamento bit a bit na memória.

Simultâneamente (paralelo) a Machine_MemB inicia a varredura de leitura da memória enderessando a primeira posição e enviando pixel a pixel, para cada um dos endereços, ao componente VGA, que por sua vez gera os sinais necessários para o Monitor de Vídeo bem como a Machine_MemB para controlar e sincronizar os bits transferidos da memória para as entradas R,G e B do Componente VGA.

Neste teste é utilizado uma imagem padrão, inicialmente barras verticais pretas e brancas para delimitar o campo de visualização da câmera. Após realizar o teste com a imagem de uma pessoa posicionada neste campo, observa-se, que o monitor apresenta a imagem da pessoa com as áreas mais escuras em preto e as mais claras em branco, comprovando que o Sistema de Captura executa a captura e exibição de forma satisfatória, como um todo, conforme ilustra a Figura 7.11.

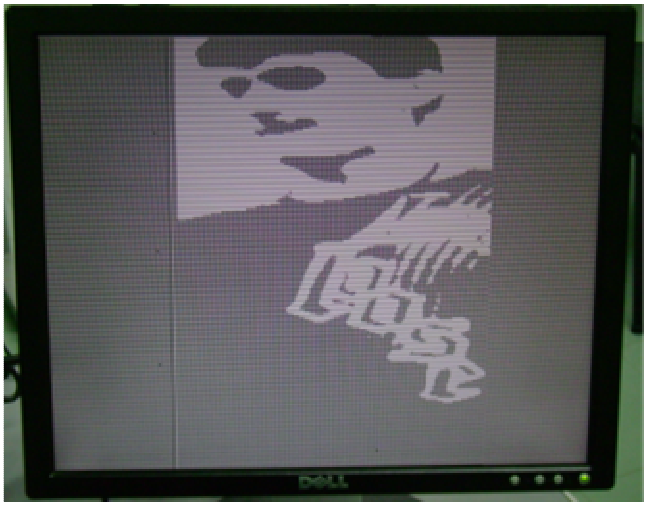


Figura 7.11 – Foto da imagem exibida no monitor de vídeo.
Fonte – Autor, 2010.

Apesar da imagem capturada apresentar-se conforme o esperado, sendo esta constituída somente pelas linhas pares da imagem capturada, observa-se que existem duas barras verticais, uma em cada lado, não esperadas.Esta discrepância está em estudos até o momento porém não havendo tempo hábil para maiores teste e investigações desta não conformidade com a imagem esperada inicialmente.

CONSIDERAÇÕES FINAIS

As pesquisas realizadas no decorrer deste Trabalho são fundamentais e servem como embasamento teórico para o desenvolvimento do Sistema de Captura de Imagens. Os estudos sobre a imagem de vídeo e respectivo formato, interfaces e sinais do Módulo de Decodificação são importantes para o desenvolvimento do Módulo de Captura .

Durante o desenvolvimento do projeto é utilizada com mais intensidade a Linguagem de Descrição de Hardware para desenvolver os componentes implementados em Hardware Reconfiguráveis.

O objetivo principal deste Trabalho é o desenvolvimento e implementação de um Sistema de Captura de imagens utilizando Hardware Reconfigurável e transferência para um monitor de vídeo padão. Considera-se que o objetivo geral do Trabalho foi satisfatoriamente atendido, a tecnologia de arquiteturas reconfiguráveis, especificamente FPGAs, foi dominada, havendo assim a capacitação para futuros trabalhos adicionais nesta área de conhecimento.

No objetivo de testar e validar o sistema de captura de imagens como um todo, houve o atendimento satisfatório, visto que puderam ser visualizadas as imagens de objetos dispostas diante da câmera, bem como imagens do ambiente onde a câmera se encontra.

Em relação a proposta de trabalhos futuros suger-se a implementação de um Sistema de Detecção de Bordas que delimitem o tamanho das superfícies dos objetos e fornecem informações importantes na extração das características da imagem. Aperfeiçoar o Componente VGA de forma a ser capaz de operar na freqüência de 27MHz sem a geração de barras laterais e/ou perda de campo de imagem. Pesquisar novas plataformas que possuam mais memórias para o armazenamento de mais de um bit por pixel da imagem.

REFERÊNCIAS BIBLIOGRÁFICAS

ACHARYA, Tinku; RAY, Ajoy K.. Image Processing Principles and Applications. USA: John Wiley & Sons, 2005. 451p.

ANALOG DEVICES. Multiformat SDTV Video Decoder: DataSheet. 2005. 100p. Disponível em: <http:// www.analog.com/static/imported-files/data_sheets/ADV7183B.pdf >. Acesso em: 30 abril 2010.

BAPTISTA, Eduardo. *Exposição / Lentes*. Artigo. Revista Zoom Magazine. Disponível em: < http://www.fazendovideo.com.br/vtart_096.asp>. Acesso em: 15 novembro 2009.

BAPTISTA, Eduardo. Sinal de Vídeo NTSC, PAL e SECAM. Artigo. Revista Zoom Magazine. Disponível em: < http://www.fazendovideo.com.br/vtart_058.asp>. Acesso em: 15 novembro 2009.

CHU, Pong P.. FPGA PROTOTYPING BY VERILOG EXAMPLES: XILINX SPARTAN . 3.ed. EUA: Wiley, 2008. 508p.

D'AMORE, Roberto. VHDL: Descrição e Síntese de Circuitos Digitais. 1.ed. Rio de Janeiro: LTC, 2005. 259p.

DIGILENT, Corporation. Video Decoder Board 1: Reference Manual. Revisão 1.0, 2005a. 2p. Disponível em: <http://www.DIGILENTinc.com>. Acesso em: 22 agosto 2009.

DIGILENT, Corporation. Video Decoder Board 1: Schematics. Revisão c, 2005b. 3p. Disponível em: <http://www.DIGILENTinc.com>. Acesso em: 22 agosto 2009.

FELGUEIRAS, Carlos; GARROTT, João. Introdução ao Processamento Digital de Imagem: Implementação em Java. 1.ed. Lisboa: FCA- Editora de Informática, Lda, 2008. 160p.

GONZALEZ, Rafael C.; WOODS, Richard E.. Digital Image Processing. 1ªed. EUA: Addison Wesley, 1992. 716p.

GONZALEZ, Rafael C.; WOODS, Richard E.. PROCESSAMENTO DE IMAGENS DIGITAIS. 3.ed. São Paulo: EDGARD BLÜCHER Ltda, 2007. 509p.

HUYGHE, Pieter, CALLANT, Jelmer. Realtime Video Processing. Industriele Wetenschappen em Technologie. Nederland: 2007. 111p.

HWANG, Enoch. Build a VGA Minitor Controller, EUA, p. 1 - 6, nov. 2008. Disponível em: <http://www.circuitcellar.com>. Acesso em: 8 mai. 2010.

JÄHNE, Bernd. Digital Image Processing. 5ªed. Germany: 2002. 598p.

MAXIM, Dallas. Video Basics. Aplication Note 734. 2001. p12. Disponível em: <http://www.maxim-ic.com/an734>. Acesso em: 3 de abril 2010.

MORGAN, Jolvani. TÉCNICAS DE SEGMENTAÇÃO DE IMAGENS NA GERAÇÃO DE PROGRAMAS PARA MÁQUINAS DE COMANDO NUMÉRICO. Santa Maria: 2008. 100p. Tese (Mestrado em Tecnologia da Informação) - Centro de Tecnologia, Universidade Federal de Santa Maria, 2008.

MORIMOTO, Carlos C. .*Monitores*. Tutorial. 2007. Disponível em: <http://www.guiadohardware.net/tutoriais/monitores-1/>. Acesso em: 30 maio 2010.

MYLER, Harley R.; WEEKS, Arthur R.. Computer Imaging Recipes in C. EUA: Prentice Hall , 1993. 284p.

NATIONAL, Instruments. Video Signal Measurement and Generation Fundamentals: Tutorial. 2006. Disponível em: < http://zone.ni.com/devzone/cda/tut/p/id/4750>. Acesso em: 25 de outubro 2009.

PHILIPS, Semiconductors. The I2C Bus Specification.Versão 2.1 EUA,2000.46p.Disponível em:<http://www.nxp.com/acrobat_download2/literature/9398/39340011.pdf>.Acesso em: 02 maio 2010.

ROSA, Rubens. Imagem na Forma Digital. Apostila 2009 43p. Disponível em: http:<http://www.ele.ita.br/~rubens/InfoTRM/imagem-p1.pdf>. Acesso em: 04 novembro 2009.

RUSS, John C.. The Image Processing Handbook. 3.ed. EUA: CRC Press LLC, 1999. 767p.

SANCHEZ, Eduardo. A VGA Display Controller, Lausanne, p. 1 - 8, mar. 2007. Disponível em: <http://lslwww.epfl.ch>. Acesso em: 5 mai. 2010.

SCHMITZ, Jerônimo C.. Captura de Imagem e Reconhecimento de Bordas utlizando Linguagem de Descrição de Hardware VHDL. Novo Hamburgo: 2007. 171p. Trabalho de Conclusão – Faculdade de Engenharia Eletrônica, Feevale, 2007.

TERROSO, Anderson Royes. DISPOSITIVO LÓGICOS PROGRAMÁVEL (FPGA) E LINGUAGEM DE DESCRIÇÃO DE HARDWARE (VHDL). Porto Alegre: 2008. 43p. MINICURSO, PUCRS, SETEMBRO DE 1998.

THIELE, Cristiano Carafini. Protótipo de um Sistema Baseado em ARQUITETURA RECONFIGURÁVEL para Aquisição e Armazenamento de Imagem. Novo Hamburgo: 2007. 131p. Trabalho de Conclusão de Curso- Faculdade de Engenharia Eletrônica, Feevale, 2007.

TOROK, Delfim Luiz. Projeto Visando a Prototipação do Protocolo de Acesso ao Meio de Redes Ethernet. Porto Alegre: 2001. 118p. Tese (Mestrado em Ciência da Computação) - Faculdade de Informática, Pontifícia Universidade Católica do Rio Grande do Sul, 2001.

TOROK, Delfim Luiz. Linguagem de Descrição de Hardware. Novo Hamburgo: 2009. 46p.Apostila de Sala de Aula. Instituto de Ciências Exatas e Tecnológicas- Feevale, 2009.

WIKIPEDIA, Free Encyclopedia. LCD: Article. Disponível em: < http://en.wikipedia.org/wiki/LCD>. Acesso em: 20 abril 2010.

WIKIPEDIA, Free Encyclopedia. Tubo de Raios Catódicos: Artigo. Disponível em:<http://pt.wikipedia.org/wiki/Tubo_de_raios_cat%C3%B3dicos>. Data de acesso, 04 junho 2010.

XILINX, Corporation. Design of a Digital Camera with CoolRunner-II CPLDs: Application Note.XAPP390, Versão 1.1, 2005a 20p. Disponível em: <http://www.XILINX.com>. Acesso em: 20 abril 2010.

XILINX, Corporation. **Spartan-3 Generation FPGA**: User Guide. UG331, Versão 1.4, 2008. 520p. Disponível em: <http://www.XILINX.com>. Acesso em: 3 junho 2007

XILINX, Corporation. **Spartan-3E FPGA**: Data Sheet. DS312, Versão 3.8, 2009. 233p. Disponível em: <http://www.XILINX.com>. Acesso em: 30 outubro 2009.

XILINX, Corporation. Spartan-**3E Libraries Guide for HDL Designs**: User Guide. ISE 10.1, 2002. 517p. Disponível em: <http://www.XILINX.com>. Acesso em: 15 maio 2010.

XILINX, Corporation. **Spartan-3E Starter Kit Board**: User Guide. UG230, Versão 1.0a, 2006a. 174p. Disponível em: <http://www.XILINX.com>. Acesso em: 30 outubro 2009.

XILINX, Corporation. **Using Block RAM in Spartan-3 Generation FPGAs:** Application Note.XAPP463, Versão 2.0, 2005b. 40p. Disponível em: <http://www.XILINX.com>. Acesso em: 05 maio 2010.

XILINX, Corporation. **Using Digital Clock Managers (DCMs) in Spartan-3 FPGAs:** Application Note.XAPP462, Versão 1.1, 2006b. 68p. Disponível em: <http://www.XILINX.com>. Acesso em: 18 maio 2010.

APÊNDICE A – CODIGO FONTE VHDL

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

entity Proj_CapImg_05 is
    Port (
                    CLK_50MHZ         : in std_logic;
                    BTN_NORTH  : in std_logic;
                    BTN_SOUTH  : in std_logic;
                    BTN_WEST   : in std_logic;
                    ROTARY_PRESS      : in std_logic;

                    RESET     : out std_logic;
                    SCL        : out std_logic;
                    SDA        : inout std_logic;
                    OE         : out std_logic;
                    PWRDN              : out std_logic;
                    LLC1       : in std_logic;
                    FIELD              : in std_logic;
                    VS                 : in std_logic;
                    HS                 : in std_logic;
                    P                  : in std_logic_vector(7 downto 0);
                    PS2_CLK   : out std_logic;

                    PS2_DATA  : out std_logic;

                    CLK_SMA    : out std_logic;
```

```
                          VGA_RED, VGA_GREEN, VGA_BLUE,VGA_HSYNC ,VGA_VSYNC : out
std_logic
                         );
        end Proj_CapImg_05;

        architecture Behavioral of Proj_CapImg_05 is

        signal clk2xllc1, ibufllc1s    :std_logic;

        COMPONENT DCM_2XLLC1
             PORT(
                    CLKIN_IN : IN std_logic;
                    RST_IN : IN std_logic;
                    CLKIN_IBUFG_OUT : OUT std_logic;
                    CLK2X_OUT : OUT std_logic;
                    LOCKED_OUT : OUT std_logic
                    );
        END COMPONENT;

        -- Ger_4clk --
        signal clkpgr,clkreset, clk25M, clk8m3, clk500k, clk16    : std_logic;

        Component Ger_4clk
             Port(
                           clk_pgr    : in   STD_LOGIC;
                         clk_reset    : in   STD_LOGIC;
                           clk_25MHz  : out  std_logic;
                    clk_8M3Hz  : out  STD_LOGIC;
                    clk_500KHz : out  STD_LOGIC;
                    clk_16Hz   : out  STD_LOGIC
                       );
        end Component;

        -- config_VDEC1 --
        signal sdacfs,sclcfs                                : std_logic;
        signal cfginis, rstvds, cfgends : std_logic;

        Component cfg_vdec1
             Port(
                         clkvd   :in std_logic;
                         cfgini  :in std_logic;
                         rstvd     :out std_logic;
                         sdacf         :inout std_logic;
                         sclcf         :out std_logic;
                         cfgend  :out std_logic
                       );
```

```
        end Component;

        -- Cap_ctl --
        signal sinaiss,dpix                                              :
STD_LOGIC_VECTOR(7 downto 0);
        signal inips,oes,pwrdns,starts,stops  : std_logic;
        signal llc1s                                                     :
std_logic;
        signal fields,hss,vss,smas, selects         : std_logic;

        Component Cap_ctl
             Port(
                          start_w     : in std_logic;
                          llc1_w    : in std_logic;
                          field_w             : in std_logic;
                          vs_w         : in std_logic;
                          hs_w         : in std_logic;
                          inip_w      : out std_logic;
                          oe_w        : out std_logic;
                          pwrdn_w     : out std_logic
                     );
        end Component;

        --Inputs -----Mem_Vga_300K -----------------------------------
           signal addra : std_logic_vector(18 downto 0) := (others => '0');
           signal addrb : std_logic_vector(18 downto 0) := (others => '0');
           signal clka  : std_logic := '0';
           signal clkb  : std_logic := '0';
           signal rsta  : std_logic :='0';
           signal rstb  : std_logic :='0';
           signal ena   : std_logic := '0';
           signal enb   : std_logic := '0';
           signal dina  : std_logic := '0';
           signal dinb  : std_logic := '0';
           signal wea   : std_logic := '0';
           signal web   : std_logic := '0';
           --Outputs -----Mem_Vga_300K -----
           signal douta : std_logic;
           signal doutb : std_logic;

        COMPONENT Mem_Vga_300K
            PORT(
                 addra : IN  std_logic_vector(18 downto 0);
                 addrb : IN  std_logic_vector(18 downto 0);
                 clka : IN  std_logic;
                 clkb : IN  std_logic;
```

```
          rsta  : IN std_logic;
            rstb  : in std_logic;
          ena : IN  std_logic;
          enb : IN  std_logic;
          dina : IN  std_logic;
          dinb : IN  std_logic;
          douta : OUT  std_logic;
          doutb : OUT  std_logic;
          wea : IN  std_logic;
          web : IN  std_logic
         );
    END COMPONENT;

    ------Component Vga--------------
    signal clkvga, rstvga, redvga, greenvga, bluevga :std_logic;
    signal vgavidon, vgafield :std_logic;
    signal rowvga : std_logic_vector(8 downto 0);
    signal columnvga : std_logic_vector(9 downto 0);

    Component Vga
        port( clk, reset                  : in std_logic;
              red, green, blue            : in std_logic;
              r, g, b, hsync, vsync       : out std_logic;
                vgaon                     : out std_logic;
                vgafd                     : out std_logic;
              row                         : out std_logic_vector(8 downto 0);
              column                      : out std_logic_vector(9 downto 0));
        end component;

    ------ Main Control----------------------------------------
    signal clksma, clkaux, vgared, vgagreen,vgablue, vgahsync,vgavsync     :
std_logic;
    signal ps2clk, ps2data   : std_logic;

    -- SYMBOLIC ENCODED clkmeme machine: Sreg0
    type estado_type is (S1, S2, S3, S4, S5, S6, S7);
    signal Sreg0: estado_type;
    signal Sreg1: estado_type;
    signal Sreg2: estado_type;
    signal Sreg3: estado_type;

    ------- sinais auxiliares -------------------------
       signal clkmema, clkmemb, rstgeral  : std_logic;
         signal row0 : std_logic;
    begin
```

```
Inst_DCM_2XLLC1: DCM_2XLLC1 PORT MAP(  CLKIN_IN => llc1s,

     RST_IN => starts,

     CLKIN_IBUFG_OUT => ibufllc1s,

     CLK2X_OUT => clk2xllc1,

     LOCKED_OUT => open

     );

oscout: Ger_4clk  port map (clk_pgr   => clkpgr,
                        clk_reset => clkreset,
                        clk_25MHz => clk25M,
                         clk_8M3Hz => clk8m3,
                         clk_500KHz=> clk500k,
                        clk_16Hz  => clk16
                  );

cfgvd: cfg_vdec1 port map (clkvd=>clk500k,
                        rstvd=>rstvds,
                        cfgini=>cfginis,
                        sdacf=>sdacfs,
                        sclcf=>sclcfs,
                        cfgend=>cfgends
                        );

adqctl: Cap_ctl port map (llc1_w=>ibufllc1s,
                         start_w=>starts,
                        field_w=>fields,
                        vs_w=>vss,
                        hs_w=>hss,
                        inip_w=> inips,
                        oe_w=>oes,
                        pwrdn_w=>pwrdns
                        );

mem: Mem_Vga_300K PORT MAP (    addra => addra,
                                addrb => addrb,
                                clka => clkmema,
                                clkb => not clkmemb,
                                rsta => rsta,
                                rstb => rstb,
                                ena => ena,
                                enb => enb,
```

```
                                dina => dina,
                                dinb => dinb,
                                douta => douta,
                                doutb => doutb,
                                wea => wea,
                                web => web
                                 );


        vgasignal: Vga port map (       clk=> clkmemb,
                                        reset=>  rstgeral,
                                        red   => redvga,
                                        green => greenvga,
                                         blue  => bluevga,
                                        r     =>vgared,
                                        g     =>vgagreen,
                                         b     =>vgablue,
                                        hsync =>vgahsync,
                                        vsync => vgavsync,
                                        vgaon => vgavidon,
                                        vgafd => vgafield,
                                        row   =>rowvga,
                                         column =>columnvga
                           );
        --- External Test Points J14 ----------------------------------------------
-
        ps2clk            <=     vgavidon;
        ps2data           <=  rstvga;
        PS2_CLK           <=  ps2clk;
        PS2_DATA          <=     ps2data;
        -----External Test Points: J17 <= CLK_SMA   IC16(B8) <= CLK_AUX ------------
---
        clksma         <=  smas;
        CLK_SMA     <=  clksma;
        --Vdec1: sinais para controle-------------------------------------------------
----
        SCL<=sclcfs;
        SDA<=sdacfs;
        RESET<=rstvds;
        OE<=oes;
        PWRDN<=pwrdns;
        llc1s <= LLC1;
        fields<=FIELD;
        vss<=not VS;
        hss<=not HS;
        -----Sinais para conect VGA J15 ----------------------------------------
```

```
        ------In--------
        redvga   <= doutb;
        greenvga <= doutb;
        bluevga  <= doutb;
        -----out--------
        VGA_RED     <=    vgared;
        VGA_GREEN   <=    vgagreen;
        VGA_BLUE    <=    vgablue;
        VGA_HSYNC   <=    vgahsync;
        VGA_VSYNC   <=    vgavsync;
        ---Sinais Auxiliares-------------------------------------------------------
---
        smas        <=    inips;
        clkpgr      <= CLK_50MHZ;
        ------------------------------------------------------------------------
---------------
        clkmemb <= ibufllc1s;

        Machine_MemA: process (clk2xllc1, inips, fields, HS, rstgeral)

        variable cntPix   : integer range 0 to 1024;
        variable cntLin   : integer range 0 to 1024;
        variable vdecendLin : integer range 0 to 1;

        begin

        if ibufllc1s'event and ibufllc1s = '1' then
             if rstgeral='1' then
                   rsta <='1';
              dina <='0';
                wea <='1';
                   ena <= '1';
                   cntPix := 0;
                   cntLin := 0;
                   vdecendLin := 0;
                   addra <= "0000000000000000000";
                   Sreg0  <= S1;
              else
              ---------------------------------------
             case Sreg0 is
                   when S1 =>
                      rsta <='0';
                          Sreg0 <= S2;
                   when S2 =>

                  if fields ='0' and  inips = '1' then
```

```
                        cntPix := 0;
                        cntLin := 0;
                        vdecendLin := 0;
                    addra <= "0000000000000000000";

            elsif fields ='0' then
                dina <= '0';
                        if Hs = '0' then
                              vdecendLin := 0;
                        end if;
                        if  vdecendLin = 0 then
                           if cntPix < 640 Then
                                dina <= p(7);
                                cntPix := cntPix + 1;

                                 addra <= addra + "0000000000000000001";
                           else
                                cntPix := 0;
                                cntLin := cntLin + 1;
                                vdecendLin := 1;
                                if cntLin < 247 Then
                                  addra <= addra + "1010000000";--640


                                else
                     cntLin := 0;
                                  addra <= "0000000000000000000";
                                end if;
                           end if;

                      end if;
             end if;
        sreg0 <=S2;
            when others =>
                  null;
     end case;
     end if;
end if;
end process;

-------------------------------------------------------------------------
row0 <= '1' when rowvga = "000000000" else '0';

clkmema <= ibufllc1s;
Machine_MemB: process (ibufllc1s, vgafield, rstgeral)
```

```
begin

if ibufllc1s'event and ibufllc1s = '1' then
     if rstgeral='1' then
           rstb <='1';
      dinb <='0';
        web <='0';
           enb <= '1';
           addrb <= "0000000000000000000";

           Sreg1  <= S1;
     else
     ---------------------------------------
     case Sreg1 is
           when S1 =>
              rstb <='0';
                 Sreg1 <= S2;
           when S2 =>

               if vgafield = '0' then
                      addrb <= "0000000000000000000";
               else
                     if vgavidon='1' then
                               addrb <= addrb + "0000000000000000001";
                     end if;
                end if;
          sreg1 <=S2;
            when others =>
                 null;
     end case;
     end if;
end if;
end process;

starts      <=     ROTARY_PRESS;
clkreset <= '0';

auto_rst:  process (clk500k, starts, cfgends)

begin

if (clk500k'event and clk500k = '0') then
     if starts = '1' then
           cfginis             <='1';
           rstgeral     <='1';
           Sreg2        <= S1;
```

```
     else
     ---------------------------------------
     case Sreg2 is
           when S1 =>
             cfginis <='0';
             sreg2      <=S2;
           when S2 =>
             Sreg2 <= S3;
           when S3 =>
             if cfgends = '1' then
              rstgeral   <='0';
              sreg2 <=S4;
           else
             sreg2      <=S2;
             end if;
           when S4 =>
        Sreg2 <= S4;
      when others =>
                  null;
     end case;
     end if;
end if;
end process;
end Behavioral;
-----------------------------------------------
--------------Componente Ger_4clk-------------
-----------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity Ger_4clk is
    Port ( clk_pgr  : in   STD_LOGIC;
             clk_reset  : in   STD_LOGIC;
                    clk_25MHz  : out  std_logic;
           clk_8M3Hz  : out  STD_LOGIC;
           clk_500KHz : out  STD_LOGIC;
           clk_16Hz   : out  STD_LOGIC);
end Ger_4clk;

architecture behavioral of Ger_4clk is

--signal declaration
signal ckl8m3Hz, clk16Hz :std_logic;
signal clk25M  :STD_LOGIC;
```

```
signal div8m3  :STD_LOGIC_VECTOR (1 downto 0);
signal div500  :STD_LOGIC_VECTOR (3 downto 0);
signal div16  :STD_LOGIC_VECTOR (17 downto 0);
--component declarations


begin

--component instantiations

process(clk_pgr, clK_reset)
begin
   If clK_reset  = '1' then
    div8m3 <= "00";
    ckl8m3Hz <= '0';
   elsif clk_pgr'event and clk_pgr = '1' then
            div8m3 <= div8m3 + "01";
            if div8m3 = "10" then
               div8m3 <= "00";
       ckl8m3Hz <= not ckl8m3Hz;
            end if;
    end if;
end process;

process (clk_pgr, clk_reset)
begin
    If clk_reset ='1' then
       clk25M   <='0';
      elsif clk_pgr'event and clk_pgr = '1' then
            clk25M <= not clk25M;

     end if;
end process;

process(ckl8m3Hz, clK_reset)
begin
    If clK_reset  = '1' then
        div500 <= "0000";
        div16 <= "000000000000000000";
        clk16Hz <= '0';
    elsif ckl8m3Hz'event and ckl8m3Hz = '1' then
               div500 <= div500 + "0001";
                   div16 <= div16 + "000000000000001";
                   if div16 = "111111100101000000" then
                     div16 <= "000000000000000000";
                     clk16Hz <= not clk16Hz;
```

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

clk_25MHz <= clk25M;
clk_8M3Hz <= ckl8m3Hz;
clk_500KHz <= div500(3);
clk_16Hz <= clk16Hz;
-------------------------
-- Copyright (c) 1995-2009 XILINX, Inc.  All rights reserved.
------------------------

-- Module DCM_2XLLC1
-- Generated by XILINX Architecture Wizard
-- Written for synthesis tool: XST

library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;
library UNISIM;
use UNISIM.Vcomponents.ALL;

entity DCM_2XLLC1 is
   port ( CLKIN_IN        : in    std_logic;
          RST_IN          : in    std_logic;
          CLKIN_IBUFG_OUT : out   std_logic;
          CLK2X_OUT       : out   std_logic;
          LOCKED_OUT      : out   std_logic);
end DCM_2XLLC1;

architecture BEHAVIORAL of DCM_2XLLC1 is
   signal CLK2X_OBUF                                    : std_logic;
   signal CLK2X_BUF             : std_logic;
   Signal CLKIN_IBUFG           : std_logic;
     signal GND_BIT             : std_logic;
begin
   GND_BIT <= '0';
   CLKIN_IBUFG_OUT <= CLKIN_IBUFG;
     CLK2X_OUT <= CLK2X_OBUF;

   CLK2X_BUFG_INST : BUFG
      port map (I=>CLK2X_BUF,
                O=> CLK2X_OBUF);
   CLKIN_IN_INST : BUF
      port map (I=>CLKIN_IN,
                O=> CLKIN_IBUFG);
```

```
   DCM_SP_INST : DCM_SP
   generic map( CLK_FEEDBACK => "1X",
            CLKDV_DIVIDE => 2.0,
            CLKFX_DIVIDE => 1,
            CLKFX_MULTIPLY => 4,
            CLKIN_DIVIDE_BY_2 => FALSE,
            CLKIN_PERIOD => 37.037,
            CLKOUT_PHASE_SHIFT => "NONE",
            DESKEW_ADJUST => "SYSTEM_SYNCHRONOUS",
            DFS_FREQUENCY_MODE => "LOW",
            DLL_FREQUENCY_MODE => "LOW",
            DUTY_CYCLE_CORRECTION => TRUE,
            FACTORY_JF => x"C080",
            PHASE_SHIFT => 0,
            STARTUP_WAIT => FALSE)
      port map (CLKFB=>CLK2X_OBUF,
                CLKIN=>CLKIN_IN,
                DSSEN=>GND_BIT,
                PSCLK=>GND_BIT,
                PSEN=>GND_BIT,
                PSINCDEC=>GND_BIT,
                RST=>RST_IN,
                CLKDV=>open,
                CLKFX=>open,
                CLKFX180=>open,
                CLK0=>open,
                CLK2X=>CLK2X_BUF,
                CLK2X180=>open,
                CLK90=>open,
                CLK180=>open,
                CLK270=>open,
                LOCKED=>LOCKED_OUT,
                PSDONE=>open,
                STATUS=>open);

end BEHAVIORAL;
----------------------------------------------
-------------Cfg_Vdec1------------------------
----------------------------------------------

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

```
        entity cfg_vdec1 is
             Port (
                          clkvd   :in std_logic;
                          cfgini  :in std_logic;
                          rstvd     :out std_logic;
                          sdacf          :inout std_logic;
                          sclcf          :out std_logic;
                          cfgend  :out std_logic
                    );
        end cfg_vdec1;

        architecture Behavioral of cfg_vdec1 is

        type Sreg1_type is (Ssi, Ss0, Ss1, Ss2, Ss3, Ss4, Ss5, Ss6, Ss7, Ss8, Ss9,
Ss10,Ss11, Ss12,
                                                    Ss13,  Ss14,  Ss15,  Ss16,  Ss17,
Ss18, Ss19, Ss20, Ss21, Ss22, Ss23, Ss24);

        type datav is array(0 to 18) of std_logic_vector( 7 downto 0);

        signal Sreg1                                        : Sreg1_type;
        signal data,subad                             : datav;
        signal sda,scl,rstvds,cfgends         : std_logic;
        signal dataout                                      :   std_logic_vector(   7
downto 0);
        constant adslave                              :  std_logic_vector(  7  downto
0):= x"40";

        begin

        rstvd<=rstvds;
        sdacf<=sda;
        sclcf<=scl;
        cfgend <= cfgends;

        Sreg1_machine: process (clkvd)

        variable c1 : integer range 0 to 7;
        variable c2 : integer range 0 to 2;
        variable c3 : integer range 0 to 18;

        begin
        if clkvd'event and clkvd = '1' then
             if cfgini = '1' then
                sda             <= '1';
```

```
            scl          <= '1';
            rstvds <= '1';
            cfgends      <= '0';
            c1       := 7;
            c2       := 0;
            c3              := 0;
            Sreg1  <= Ssi;
     else
      case Sreg1 is
-- Reset do VDEC1
                  when Ssi =>
                 rstvds  <= '0';
                    Sreg1      <= Ss0;
                  when Ss0 =>
                 rstvds  <= '1';
                    Sreg1      <= Ss1;
-- Condiçao de START
                  when Ss1 =>
                 sda            <= '0';
                    scl        <= '1';
                    Sreg1      <= Ss2;
                  when Ss2 =>
                    sda    <= '0';
                    scl    <= '0';
                    Sreg1  <= Ss3;
-- Transmissao do endereço
               when Ss3 =>
                 sda    <= adslave(c1);
                 Sreg1  <= Ss4;
                  when Ss4 =>
                    scl    <= '1';
                    Sreg1  <= Ss5;
                  when Ss5 =>
                 scl        <= '1';
                    Sreg1  <= Ss6;
         when Ss6 =>
                    scl    <= '0';
                    if c1 > 0  then
                       c1  := c1 - 1;
                     Sreg1 <= Ss3;
                    else
                       c1   := 7;
                         Sreg1 <= Ss7;
                    end if;
                  when Ss7 =>
                    sda    <= 'Z';
```

```
                    Sreg1  <= Ss8;
                  when Ss8 =>
                    scl    <= '1';
                    Sreg1  <= Ss9;
                  when Ss9 =>
                    Sreg1  <= Ss10;
               when Ss10 =>
                    scl    <= '0';
                    Sreg1  <= Ss11;
                  when Ss11 =>
                    scl    <= '0';
                    sda    <= '0';

                    Sreg1 <= Ss12;
-- Transmissao do sub-endereço e dado
                  when Ss12 =>    -- SDA = subad(c1) e SCL=0
                 if c2 < 1 then
                     dataout <= subad(c3);
                 else
                     dataout <= data(c3);
                 end if;
                 sda    <= dataout(c1);
                 Sreg1  <= Ss13;
                  when Ss13 =>          -- SCL=1
                    scl    <= '1';
                    Sreg1  <= Ss14;
                  when Ss14 =>              -- SCL=1
                 scl         <= '1';
                    Sreg1   <= Ss15;
         when Ss15 =>                 -- SCL=0
                    scl     <= '0';
                    if c1 > 0  then
                       c1   := c1 - 1;
                    Sreg1   <= Ss12;
                    else
                       c1   := 7;
                      Sreg1 <= Ss16;
                    end if;
-- Espera do bit de reconhecimento
                  when Ss16 =>
                    sda     <= 'Z';
                    Sreg1   <= Ss17;
                  when Ss17 =>
                    scl     <= '1';
                    Sreg1   <= Ss18;
                  when Ss18 =>
```

```
                    Sreg1   <= Ss19;
               when Ss19 =>
                    scl     <= '0';
                    Sreg1   <= Ss20;
-- Verificaçao de contadores p/ subadress/data
                  when Ss20 =>
                    scl     <= '0';
                    sda     <= '0';
                    if c2 < 1 then
                      c2    := c2 + 1;
                      Sreg1 <= Ss12;
                    else
                      Sreg1 <= Ss21;
                    end if;
-- Condiçao de STOP
                   when Ss21 =>
                    scl     <= '1';
                    Sreg1   <= Ss22;
         when Ss22 =>
                    sda     <= '1';
                    Sreg1   <= Ss23;
-- Verificaçao de contador da linha da matriz de config
         when Ss23 =>
           if c3 < 2 then
                       c3 := c3 + 1;
                       c2 := 0;
                     Sreg1 <= Ss1;
                    else
                     Sreg1 <= Ss24;
                    end if;
                  when Ss24 =>
                   cfgends <= '1';
                   Sreg1   <= Ss24;
         when others =>
                               null;
     end case;
  end if;
 end if;
end process;

---- Matriz de dados para configuração
subad(0)<=x"00";  data(0)<=x"04";          -- dado
subad(1)<=x"15";  data(1)<=x"00";          -- dado
subad(2)<=x"17";  data(2)<=x"41";          -- dado
subad(3)<=x"3A";  data(3)<=x"16";          -- dado
subad(4)<=x"50";  data(4)<=x"04";          -- dado
```

```
subad(5)<=x"0E";  data(5)<=x"80";           -- dado
subad(6)<=x"50";  data(6)<=x"20";           -- dado
subad(7)<=x"52";  data(7)<=x"18";           -- dado
subad(8)<=x"58";  data(8)<=x"ED";           -- dado
subad(9)<=x"77";  data(9)<=x"C5";           -- dado
subad(10)<=x"7C"; data(10)<=x"93";          -- dado
subad(11)<=x"7D"; data(11)<=x"00";          -- dado
subad(12)<=x"D0"; data(12)<=x"48";          -- dado
subad(13)<=x"D5"; data(13)<=x"A0";          -- dado
subad(14)<=x"D7"; data(14)<=x"EA";          -- dado
subad(15)<=x"E4"; data(15)<=x"3E";          -- dado
subad(16)<=x"E9"; data(16)<=x"3E";          -- dado
subad(17)<=x"EA"; data(17)<=x"0F";          -- dado
subad(18)<=x"0E"; data(18)<=x"00";          -- dado

end Behavioral;

--0x00 0x04 --CVBS input on AIN5.
--0x15 0x00 --Slow down digital clamps.
--0x17 0x41 --Set CSFM to SH1.
--0x3A 0x16 --Power down ADC 1 and ADC 2.
--0x50 0x04 --Set DNR threshold to 4 for flat response.
----------------------------------------------------
-------------Mem_Vga_300K-------------------------
----------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
--Library UNISIM;
--use UNISIM.vcomponents.all;

ENTITY Mem_Vga_300K IS
     port (
     addra: IN std_logic_VECTOR(18 downto 0);
     addrb: IN std_logic_VECTOR(18 downto 0);
     clka: IN std_logic;
     clkb: IN std_logic;
     ena : IN std_logic;
     rsta: IN std_logic;
     rstb: IN std_logic;
     enb : In std_logic;
     dina: IN std_logic;
     dinb: IN std_logic;
     douta: OUT std_logic;
     doutb: OUT std_logic;
     wea: IN std_logic;
     web: IN std_logic);
```

```
END Mem_Vga_300K;

ARCHITECTURE behavioral OF Mem_Vga_300K IS

component RAMB16_S1_S1

   port (DOA0 : out STD_LOGIC;
         DOB0 : out STD_LOGIC;
         ADDRA : in STD_LOGIC_VECTOR (13 downto 0);
         ADDRB : in STD_LOGIC_VECTOR (13 downto 0);
         CLKA : in STD_LOGIC;
         CLKB : in STD_LOGIC;
         DIA0 : in STD_LOGIC;
         DIB0 : in STD_LOGIC;
         ENA : in STD_LOGIC;
         ENB : in STD_LOGIC;
         SSRA : in STD_LOGIC;
         SSRB : in STD_LOGIC;
         WEA : in STD_LOGIC;
         WEB : in STD_LOGIC);
end component;

type barout is array (0 to 1) of STD_LOGIC;

signal  doan_0,dobn_0          : std_logic;
signal  doan_1,dobn_1          : std_logic;
signal  doan_2,dobn_2          : std_logic;
signal  doan_3,dobn_3          : std_logic;
signal  doan_4,dobn_4          : std_logic;
signal  doan_5,dobn_5          : std_logic;
signal  doan_6,dobn_6          : std_logic;
signal  doan_7,dobn_7          : std_logic;
signal  doan_8,dobn_8          : std_logic;
signal  doan_9,dobn_9          : std_logic;
signal  doan_10,dobn_10        : std_logic;
signal  doan_11,dobn_11        : std_logic;
signal  doan_12,dobn_12        : std_logic;
signal  doan_13,dobn_13        : std_logic;
signal  doan_14,dobn_14        : std_logic;
signal  doan_15,dobn_15        : std_logic;
signal  doan_16,dobn_16  : std_logic;
signal  doan_17,dobn_17        : std_logic;
signal  doan_18,dobn_18        : std_logic;

signal  dian,dibn              : STD_LOGIC;
signal  enan_0,enbn_0       : STD_LOGIC;
```

```
signal  enan_1,enbn_1       : STD_LOGIC;
signal  enan_2,enbn_2       : STD_LOGIC;
signal  enan_3,enbn_3       : STD_LOGIC;
signal  enan_4,enbn_4       : STD_LOGIC;
signal  enan_5,enbn_5       : STD_LOGIC;
signal  enan_6,enbn_6       : STD_LOGIC;
signal  enan_7,enbn_7       : STD_LOGIC;
signal  enan_8,enbn_8       : STD_LOGIC;
signal  enan_9,enbn_9       : STD_LOGIC;
signal  enan_10,enbn_10  : STD_LOGIC;
signal  enan_11,enbn_11  : STD_LOGIC;
signal  enan_12,enbn_12  : STD_LOGIC;
signal  enan_13,enbn_13     : STD_LOGIC;
signal  enan_14,enbn_14  : STD_LOGIC;
signal  enan_15,enbn_15  : STD_LOGIC;
signal  enan_16,enbn_16  : STD_LOGIC;
signal  enan_17,enbn_17  : STD_LOGIC;
signal  enan_18,enbn_18  : STD_LOGIC;

Begin

blockram_0: RAMB16_S1_S1 port map ( ADDRA=>addra(13 downto 0),
                                    ADDRB=>addrb(13 downto 0),
                                    CLKA=>clka,
                                    CLKB=>clkb,
                                    ENA=>enan_0,
                                    SSRA=>rsta,
                                    SSRB=>rstb,
                                    ENB=>enbn_0,
                                    DIA0=>dina,
                                    DIB0=>dinb,
                                   DOA0=>doan_0,
                                   DOB0=>dobn_0,
                                  WEA=> wea,
                                 WEB=>  web );


blockram_1: RAMB16_S1_S1 port map ( ADDRA=>addra(13 downto 0),
                                    ADDRB=>addrb(13 downto 0),
                                    CLKA=>clka,
                                    CLKB=>clkb,
                                    ENA=>enan_1,
                                    SSRA=>rsta,
                                    SSRB=>rstb,
                                    ENB=>enbn_1,
                                    DIA0=>dina,
```

```
                                    DIB0=>dinb,
                                   DOA0=>doan_1,
                                   DOB0=>dobn_1,
                                  WEA=> wea,
                                 WEB=>  web );

blockram_2: RAMB16_S1_S1 port map ( ADDRA=>addra(13 downto 0),
                                    ADDRB=>addrb(13 downto 0),
                                    CLKA=>clka,
                                    CLKB=>clkb,
                                    ENA=>enan_2,
                                    SSRA=>rsta,
                                    SSRB=>rstb,
                                    ENB=>enbn_2,
                                    DIA0=>dina,
                                    DIB0=>dinb,
                                   DOA0=>doan_2,
                                   DOB0=>dobn_2,
                                  WEA=> wea,
                                 WEB=>  web );

blockram_3: RAMB16_S1_S1 port map ( ADDRA=>addra(13 downto 0),
                                    ADDRB=>addrb(13 downto 0),
                                    CLKA=>clka,
                                    CLKB=>clkb,
                                    ENA=>enan_3,
                                    SSRA=>rsta,
                                    SSRB=>rstb,
                                    ENB=>enbn_3,
                                    DIA0=>dina,
                                    DIB0=>dinb,
                                   DOA0=>doan_3,
                                   DOB0=>dobn_3,
                                  WEA=> wea,
                                 WEB=>  web );

blockram_4: RAMB16_S1_S1 port map ( ADDRA=>addra(13 downto 0),
                                    ADDRB=>addrb(13 downto 0),
                                    CLKA=>clka,
                                    CLKB=>clkb,
                                    ENA=>enan_4,
                                    SSRA=>rsta,
                                    SSRB=>rstb,
                                    ENB=>enbn_4,
                                    DIA0=>dina,
                                    DIB0=>dinb,
```

```
                                    DOA0=>doan_4,
                                    DOB0=>dobn_4,
                                   WEA=> wea,
                                  WEB=>  web );

blockram_5: RAMB16_S1_S1 port map ( ADDRA=>addra(13 downto 0),
                                     ADDRB=>addrb(13 downto 0),
                                     CLKA=>clka,
                                     CLKB=>clkb,
                                     ENA=>enan_5,
                                     SSRA=>rsta,
                                     SSRB=>rstb,
                                     ENB=>enbn_5,
                                     DIA0=>dina,
                                     DIB0=>dinb,
                                    DOA0=>doan_5,
                                    DOB0=>dobn_5,
                                   WEA=> wea,
                                  WEB=>  web );

blockram_6: RAMB16_S1_S1 port map ( ADDRA=>addra(13 downto 0),
                                     ADDRB=>addrb(13 downto 0),
                                     CLKA=>clka,
                                     CLKB=>clkb,
                                     ENA=>enan_6,
                                     SSRA=>rsta,
                                     SSRB=>rstb,
                                     ENB=>enbn_6,
                                     DIA0=>dina,
                                     DIB0=>dinb,
                                    DOA0=>doan_6,
                                    DOB0=>dobn_6,
                                   WEA=> wea,
                                  WEB=>  web );

blockram_7: RAMB16_S1_S1 port map ( ADDRA=>addra(13 downto 0),
                                     ADDRB=>addrb(13 downto 0),
                                     CLKA=>clka,
                                     CLKB=>clkb,
                                     ENA=>enan_7,
                                     SSRA=>rsta,
                                     SSRB=>rstb,
                                     ENB=>enbn_7,
                                     DIA0=>dina,
                                     DIB0=>dinb,
                                    DOA0=>doan_7,
```

```
                                    DOB0=>dobn_7,
                                   WEA=> wea,
                                  WEB=>  web );

blockram_8: RAMB16_S1_S1 port map ( ADDRA=>addra(13 downto 0),
                                    ADDRB=>addrb(13 downto 0),
                                    CLKA=>clka,
                                    CLKB=>clkb,
                                    ENA=>enan_8,
                                    SSRA=>rsta,
                                    SSRB=>rstb,
                                    ENB=>enbn_8,
                                    DIA0=>dina,
                                    DIB0=>dinb,
                                   DOA0=>doan_8,
                                    DOB0=>dobn_8,
                                   WEA=> wea,
                                  WEB=>  web );

blockram_9: RAMB16_S1_S1 port map ( ADDRA=>addra(13 downto 0),
                                    ADDRB=>addrb(13 downto 0),
                                    CLKA=>clka,
                                    CLKB=>clkb,
                                    ENA=>enan_9,
                                    SSRA=>rsta,
                                    SSRB=>rstb,
                                    ENB=>enbn_9,
                                    DIA0=>dina,
                                    DIB0=>dinb,
                                   DOA0=>doan_9,
                                    DOB0=>dobn_9,
                                   WEA=> wea,
                                  WEB=>  web );

blockram_10: RAMB16_S1_S1 port map ( ADDRA=>addra(13 downto 0),
                                    ADDRB=>addrb(13 downto 0),
                                    CLKA=>clka,
                                    CLKB=>clkb,
                                    ENA=>enan_10,
                                    SSRA=>rsta,
                                    SSRB=>rstb,
                                    ENB=>enbn_10,
                                    DIA0=>dina,
                                    DIB0=>dinb,
                                   DOA0=>doan_10,
                                    DOB0=>dobn_10,
```

```
                                    WEA=> wea,
                                   WEB=>  web );

blockram_11: RAMB16_S1_S1 port map ( ADDRA=>addra(13 downto 0),
                                     ADDRB=>addrb(13 downto 0),
                                     CLKA=>clka,
                                     CLKB=>clkb,
                                     ENA=>enan_11,
                                     SSRA=>rsta,
                                     SSRB=>rstb,
                                     ENB=>enbn_11,
                                     DIA0=>dina,
                                     DIB0=>dinb,
                                    DOA0=>doan_11,
                                    DOB0=>dobn_11,
                                   WEA=> wea,
                                  WEB=>  web );

blockram_12: RAMB16_S1_S1 port map ( ADDRA=>addra(13 downto 0),
                                     ADDRB=>addrb(13 downto 0),
                                     CLKA=>clka,
                                     CLKB=>clkb,
                                     ENA=>enan_12,
                                     SSRA=>rsta,
                                     SSRB=>rstb,
                                     ENB=>enbn_12,
                                     DIA0=>dina,
                                     DIB0=>dinb,
                                    DOA0=>doan_12,
                                    DOB0=>dobn_12,
                                   WEA=> wea,
                                  WEB=>  web );

blockram_13: RAMB16_S1_S1 port map ( ADDRA=>addra(13 downto 0),
                                     ADDRB=>addrb(13 downto 0),
                                     CLKA=>clka,
                                     CLKB=>clkb,
                                     ENA=>enan_13,
                                     SSRA=>rsta,
                                     SSRB=>rstb,
                                     ENB=>enbn_13,
                                     DIA0=>dina,
                                     DIB0=>dinb,
                                    DOA0=>doan_13,
                                    DOB0=>dobn_13,
                                   WEA=> wea,
```

```
                                    WEB=>  web );

blockram_14: RAMB16_S1_S1 port map ( ADDRA=>addra(13 downto 0),
                                     ADDRB=>addrb(13 downto 0),
                                     CLKA=>clka,
                                     CLKB=>clkb,
                                     ENA=>enan_14,
                                     SSRA=>rsta,
                                     SSRB=>rstb,
                                     ENB=>enbn_14,
                                     DIA0=>dina,
                                     DIB0=>dinb,
                                    DOA0=>doan_14,
                                    DOB0=>dobn_14,
                                   WEA=> wea,
                                  WEB=>  web );

blockram_15: RAMB16_S1_S1 port map ( ADDRA=>addra(13 downto 0),
                                     ADDRB=>addrb(13 downto 0),
                                     CLKA=>clka,
                                     CLKB=>clkb,
                                     ENA=>enan_15,
                                     SSRA=>rsta,
                                     SSRB=>rstb,
                                     ENB=>enbn_15,
                                     DIA0=>dina,
                                     DIB0=>dinb,
                                    DOA0=>doan_15,
                                    DOB0=>dobn_15,
                                   WEA=> wea,
                                  WEB=>  web );

blockram_16: RAMB16_S1_S1 port map ( ADDRA=>addra(13 downto 0),
                                     ADDRB=>addrb(13 downto 0),
                                     CLKA=>clka,
                                     CLKB=>clkb,
                                     ENA=>enan_16,
                                     SSRA=>rsta,
                                     SSRB=>rstb,
                                     ENB=>enbn_16,
                                     DIA0=>dina,
                                     DIB0=>dinb,
                                    DOA0=>doan_16,
                                    DOB0=>dobn_16,
                                   WEA=> wea,
                                  WEB=>  web );
```

```
blockram_17: RAMB16_S1_S1 port map ( ADDRA=>addra(13 downto 0),
                                     ADDRB=>addrb(13 downto 0),
                                     CLKA=>clka,
                                     CLKB=>clkb,
                                     ENA=>enan_17,
                                     SSRA=>rsta,
                                     SSRB=>rstb,
                                     ENB=>enbn_17,
                                     DIA0=>dina,
                                     DIB0=>dinb,
                                    DOA0=>doan_17,
                                    DOB0=>dobn_17,
                                   WEA=> wea,
                                  WEB=>  web );

blockram_18: RAMB16_S1_S1 port map ( ADDRA=>addra(13 downto 0),
                                     ADDRB=>addrb(13 downto 0),
                                     CLKA=>clka,
                                     CLKB=>clkb,
                                     ENA=>enan_18,
                                     SSRA=>rsta,
                                     SSRB=>rstb,
                                     ENB=>enbn_18,
                                     DIA0=>dina,
                                     DIB0=>dinb,
                                    DOA0=>doan_18,
                                    DOB0=>dobn_18,
                                   WEA=> wea,
                                  WEB=>  web );




------ Selecao da block ram para saida da memoria RAM dautb-------

process(addra)
begin
     if (addra (18 downto 14) =    "00000") then
            douta <= doan_0;
       elsif (addra (18 downto 14) = "00001") then
            douta <= doan_1;
```

```
       elsif (addra (18 downto 14) = "00010") then
     douta <= doan_2;
       elsif (addra (18 downto 14) = "00011") then
            douta <= doan_3;
     elsif (addra (18 downto 14) = "00100") then
            douta <= doan_4;
     elsif (addra (18 downto 14) = "00101") then
            douta <= doan_5;
     elsif (addra (18 downto 14) = "00110") then
            douta <= doan_6;
     elsif (addra (18 downto 14) = "00111") then
            douta <= doan_7;
     elsif (addra (18 downto 14) = "01000") then
            douta <=doan_8;
     elsif (addra (18 downto 14) = "01001") then
            douta <= doan_9;
     elsif (addra (18 downto 14) = "01010") then
            douta <= doan_10;
     elsif (addra (18 downto 14) = "01011") then
            douta <= doan_11;
     elsif (addra (18 downto 14) = "01100") then
            douta <= doan_12;
     elsif (addra (18 downto 14) = "01101") then
            douta <= doan_13;
     elsif (addra (18 downto 14) = "01110") then
            douta <=doan_14;
     elsif (addra (18 downto 14) = "01111") then
            douta <=doan_15;
     elsif (addra (18 downto 14) = "10000") then
            douta <=doan_16;
     elsif (addra (18 downto 14) = "10001") then
            douta <= doan_17;
     elsif (addra (18 downto 14) = "10010") then
            douta <=doan_18;
       end if;
end process;

------ Selecao da block ram para saida da memoria RAM dautb------

process(addrb)
begin
     if (addrb (18 downto 14) =    "00000") then
            doutb <= dobn_0;
       elsif (addrb (18 downto 14) = "00001") then
            doutb <= dobn_1;
       elsif (addrb (18 downto 14) = "00010") then
```

```
                doutb <= dobn_2;
              elsif (addrb (18 downto 14) = "00011") then
                doutb <= dobn_3;
            elsif (addrb (18 downto 14) = "00100") then
                doutb <=dobn_4;
            elsif (addrb (18 downto 14) = "00101") then
                doutb <=dobn_5;
            elsif (addrb (18 downto 14) = "00110") then
                doutb <= dobn_6;
            elsif (addrb (18 downto 14) = "00111") then
                doutb <= dobn_7;
            elsif (addrb (18 downto 14) = "01000") then
                doutb <= dobn_8;
            elsif (addrb (18 downto 14) = "01001") then
                doutb <= dobn_9;
            elsif (addrb (18 downto 14) = "01010") then
                doutb <= dobn_10;
            elsif (addrb (18 downto 14) = "01011") then
                doutb <= dobn_11;
            elsif (addrb (18 downto 14) = "01100") then
                doutb <= dobn_12;
            elsif (addrb (18 downto 14) = "01101") then
                doutb <=dobn_13;
            elsif (addrb (18 downto 14) = "01110") then
                doutb <=dobn_14;
            elsif (addrb (18 downto 14) = "01111") then
                doutb <=dobn_15;
            elsif (addrb (18 downto 14) = "10000") then
                doutb <=dobn_16;
            elsif (addrb (18 downto 14) = "10001") then
                doutb <=dobn_17;
            elsif (addrb (18 downto 14) = "10010") then
                doutb <=dobn_18;
             end if;
        end process;

        --------Habilita endereços do bloco A-------

        enan_0 <= '1' when (addra (18 downto 14) = "00000") and (ena = '1') else
'0';
        enan_1 <= '1' when (addra (18 downto 14) = "00001")  and (ena = '1') else
'0';
        enan_2 <= '1' when (addra (18 downto 14) = "00010") and (ena = '1') else
'0';
        enan_3 <= '1' when (addra (18 downto 14) = "00011") and (ena = '1') else
'0';
```

```
        enan_4 <= '1' when (addra (18 downto 14) = "00100") and (ena = '1') else
'0';
        enan_5 <= '1' when (addra (18 downto 14) = "00101") and (ena = '1') else
'0';
        enan_6 <= '1' when (addra (18 downto 14) = "00110") and (ena = '1') else
'0';
        enan_7 <= '1' when (addra (18 downto 14) = "00111") and (ena = '1') else
'0';
        enan_8 <= '1' when (addra (18 downto 14) = "01000") and (ena = '1') else
'0';
        enan_9 <= '1' when (addra (18 downto 14) = "01001") and (ena = '1') else
'0';
        enan_10 <= '1' when (addra (18 downto 14) = "01010") and (ena = '1') else
'0';
        enan_11 <= '1' when (addra (18 downto 14) = "01011") and (ena = '1') else
'0';
        enan_12 <= '1' when (addra (18 downto 14) = "01100") and (ena = '1') else
'0';
        enan_13 <= '1' when (addra (18 downto 14) = "01101") and (ena = '1') else
'0';
        enan_14 <= '1' when (addra (18 downto 14) = "01110") and (ena = '1') else
'0';
        enan_15 <= '1' when (addra (18 downto 14) = "01111") and (ena = '1') else
'0';
        enan_16 <= '1' when (addra (18 downto 14) = "10000") and (ena = '1') else
'0';
        enan_17 <= '1' when (addra (18 downto 14) = "10001") and (ena = '1') else
'0';
        enan_18 <= '1' when (addra (18 downto 14) = "10010") and (ena = '1') else
'0';

        ----- Habilita endereços do bloco B-----

        enbn_0 <= '1' when (addrb (18 downto 14) = "00000") and (enb = '1') else
'0';
        enbn_1 <= '1' when (addrb (18 downto 14) = "00001")  and (enb = '1') else
'0';
        enbn_2 <= '1' when (addrb (18 downto 14) = "00010") and (enb = '1') else
'0';
        enbn_3 <= '1' when (addrb (18 downto 14) = "00011") and (enb = '1') else
'0';
        enbn_4 <= '1' when (addrb (18 downto 14) = "00100") and (enb = '1') else
'0';
        enbn_5 <= '1' when (addrb (18 downto 14) = "00101") and (enb = '1') else
'0';
```

```
        enbn_6 <= '1' when (addrb (18 downto 14) = "00110") and (enb = '1') else
'0';
        enbn_7 <= '1' when (addrb (18 downto 14) = "00111") and (enb = '1') else
'0';
        enbn_8 <= '1' when (addrb (18 downto 14) = "01000") and (enb = '1') else
'0';
        enbn_9 <= '1' when (addrb (18 downto 14) = "01001") and (enb = '1') else
'0';
        enbn_10 <= '1' when (addrb (18 downto 14) = "01010") and (enb = '1') else
'0';
        enbn_11 <= '1' when (addrb (18 downto 14) = "01011") and (enb = '1') else
'0';
        enbn_12 <= '1' when (addrb (18 downto 14) = "01100") and (enb = '1') else
'0';
        enbn_13 <= '1' when (addrb (18 downto 14) = "01101") and (enb = '1') else
'0';
        enbn_14 <= '1' when (addrb (18 downto 14) = "01110") and (enb = '1') else
'0';
        enbn_15 <= '1' when (addrb (18 downto 14) = "01111") and (enb = '1') else
'0';
        enbn_16 <= '1' when (addrb (18 downto 14) = "10000") and (enb = '1') else
'0';
        enbn_17 <= '1' when (addrb (18 downto 14) = "10001") and (enb = '1') else
'0';
        enbn_18 <= '1' when (addrb (18 downto 14) = "10010") and (enb = '1') else
'0';

        end behavioral;

        ----------------------------------------------
        ----------------- Cap_ctl--------------------
        ----------------------------------------------
        library IEEE;
        use IEEE.STD_LOGIC_1164.ALL;
        use IEEE.STD_LOGIC_ARITH.ALL;
        use IEEE.STD_LOGIC_UNSIGNED.ALL;

        entity Cap_ctl is
            Port (
                          start_w      : in std_logic;
                          llc1_w               : in std_logic;
                          inip_w        : out std_logic;
                          field_w           : in std_logic;
                          vs_w                    : in std_logic;
                          hs_w                    : in std_logic;
                          oe_w         : out std_logic;
```

```
                    pwrdn_w      : out std_logic );

end Cap_ctl;

architecture Behavioral of Cap_ctl is

type estado_type is (S0, S1 ,S2, S3, S4, S5, S6, S7, s8, s9, s11);
signal Sreg1                          : estado_type;
signal starts,oes,pwrdns              : std_logic;
signal fields, vss, hss, llc1, inip   : std_logic;

begin
--sinais de entrada
starts<=start_w;
fields<=field_w;
vss<=vs_w;
hss<=hs_w;
oes<='0';
pwrdns<='1';
llc1<= llc1_w;
--sinais de saida
oe_w<=oes;
pwrdn_w<=pwrdns;
inip_w<=inip;

Sreg1_machine: process (llc1, starts)

variable c1 : integer range 0 to 30;
variable c2 : integer range 0 to 128;
variable c3 : integer range 0 to 2023;

begin

if llc1'event and llc1 = '1' then
     if starts = '1' then
           inip   <= '0';
           c1     := 0;
           c2     := 0;
           c3           := 0;
           Sreg1  <= S1;
     else
      case Sreg1 is
-- Condiçao de START para captura de campo impar
          when S1 =>
   if fields = '1' then
                    Sreg1      <= S2;
```

```
                end if;
              when S2 =>
        if fields = '0' then
                      inip <= '1';
                   Sreg1      <= S3;
                end if;
              when S3 =>
                   if hss = '0' then
                              Sreg1 <= S4;
                   end if;
                when S4 =>
           if hss = '1' then
                              if c1 < 16 then
                                     c1 := c1 + 1;
                                     Sreg1 <= S3;
                              else
                                     Sreg1 <= S5;
                              end if;
                         end if;
                when S5 =>

                              c2 := 0;
                              Sreg1 <= S6;
          when S6 =>
                  if c3 < 1200 then
                              c3 := c3+ 1;
                              Sreg1 <= S6;
                   else
                              inip <= '0';
                              Sreg1 <= S7;
            end if;
                  when S7 =>
                              c1 := 0;
                              c2 := 0;
                              c3 := 0;
                              Sreg1 <= S1;
                when others =>
                              null;
     end case;
  end if;
 end if;
end process;
end Behavioral;
----------------------------------------------------
--------------------VGA-----------------------------
----------------------------------------------------
```

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

        entity Vga is
            port( clk, reset                 : in std_logic;
                  red, green, blue           : in std_logic;
                  r, g, b, hsync, vsync      : out std_logic;
                           vgaon                                  : out std_logic;
                           vgafd                                  : out std_logic;
                  row                                             :              out
std_logic_vector(8 downto 0);
                  column                                    :  out  std_logic_vector(9
downto 0));
             end Vga;

           architecture Behavioral of Vga is

                signal videoon, videov, videoh : std_logic;
                signal hcount, vcount : std_logic_vector(9 downto 0);


                 begin

                hcounter: process (clk, reset)
                           begin
                           if reset='1'                          -- reseta em 1
                              then hcount <= (others => '0');
                              else if (clk'event and clk='1')
                                      then  if  hcount=799          --recebe  800  com
todos os sinais de retraço.
                                            then hcount <= (others => '0');
                                            else hcount <= hcount + 1;
                                       end if;
                              end if;
                            end if;
           end process;


           process (hcount)
           begin
           videoh <= '1';
           column <= hcount;
           if hcount>639
              then videoh <= '0';
```

```
           column <= (others => '0');
   end if;
   end process;
vcounter: process (clk, reset)
    begin
    if reset='1'
        then vcount <= (others => '0');
        else if (clk'event and clk='1')
             then if hcount=699
                    then if vcount=524
                        then vcount <= (others => '0');
                        else vcount <= vcount + 1;
                     end if;
              end if;
         end if;
       end if;
end process;

    process (vcount)
    begin
    videov <= '1';
    row <= vcount(8 downto 0);
    if vcount>479
       then videov <= '0';
            row <= (others => '0');
     end if;
     end process;


 sync: process (clk, reset)
       begin
       if reset='1'
           then hsync <= '0';
                vsync <= '0';
           else if (clk'event and clk='1')
                    then if (hcount<=755 and hcount>=659)
                         then hsync <= '0';
                         else hsync <= '1';
                    end if;
                    if (vcount<=494 and vcount>=493)
                        then vsync <= '0';
                        else vsync <= '1';
                     end if;
              end if;
         end if;
         end process;
```

```
  videoon <= videoh and videov;
  vgaon     <= videoon;
  vgafd  <= videov;

  colors: process (clk, reset)
            begin
            if reset='1'
               then r <= '0';
                    g <= '0';
                    b <= '0';
               elsif (clk'event and clk='1')
                  then r <= red and videoon;
                       g <= green and videoon;
                       b <= blue and videoon;
                end if;
                end process;
  end Behavioral;
-------------------------------------------------------------
-----------Arquivo de Restrições-----------------------------
-------------------------------------------------------------
NET "rotary_press" LOC = "V16" | IOSTANDARD = LVTTL | PULLDOWN;
NET "BTN_NORTH"  LOC = "V4" |IOSTANDARD = LVTTL  |PULLDOWN ;
NET "BTN_SOUTH"  LOC = "K17" |IOSTANDARD = LVTTL  |PULLDOWN ;
NET "BTN_WEST"  LOC = "D18" |IOSTANDARD = LVTTL  |PULLDOWN ;
NET "CLK_50MHZ"  LOC = "C9" |IOSTANDARD = LVTTL ;
NET "FIELD"  LOC = "D7" |IOSTANDARD = LVCMOS33 ;
NET "HS"  LOC = "F8" |IOSTANDARD = LVCMOS33 ;
NET "LLC1"  LOC = "E10" |IOSTANDARD = LVCMOS33  |PULLDOWN ;
NET "OE"  LOC = "F7" |IOSTANDARD = LVCMOS33  |SLEW = FAST  |DRIVE = 8 ;
NET "P<0>"  LOC = "D11" |IOSTANDARD = LVCMOS33  |PULLDOWN ;
NET "P<1>"  LOC = "E9" |IOSTANDARD = LVCMOS33  |PULLDOWN ;
NET "P<2>"  LOC = "F9" |IOSTANDARD = LVCMOS33  |PULLDOWN ;
NET "P<3>"  LOC = "E8" |IOSTANDARD = LVCMOS33  |PULLDOWN ;
NET "P<4>"  LOC = "E7" |IOSTANDARD = LVCMOS33  |PULLDOWN ;
NET "P<5>"  LOC = "B6" |IOSTANDARD = LVCMOS33  |PULLDOWN ;
NET "P<6>"  LOC = "A6" |IOSTANDARD = LVCMOS33  |PULLDOWN ;
NET "P<7>"  LOC = "C5" |IOSTANDARD = LVCMOS33  |PULLDOWN ;
NET "PS2_CLK"  LOC = "G14" |IOSTANDARD = LVCMOS33  |DRIVE = 8  |SLEW = SLOW
;
NET "PS2_DATA"  LOC = "G13" |IOSTANDARD = LVCMOS33  |DRIVE = 8  |SLEW = SLOW
;
NET "PWRDN"  LOC = "A16" |IOSTANDARD = LVCMOS33  |SLEW = FAST  |DRIVE = 8 ;
NET "RESET"  LOC = "B4" |IOSTANDARD = LVCMOS33  |SLEW = FAST  |DRIVE = 8 ;
NET "SCL"  LOC = "D5" |IOSTANDARD = LVCMOS33  |SLEW = FAST  |DRIVE = 8 ;
NET "SDA"  LOC = "A4" |IOSTANDARD = LVCMOS33  |SLEW = FAST  |DRIVE = 8 ;
```

```
NET "VS"  LOC = "C7" |IOSTANDARD = LVCMOS33 ;
NET "CLK_SMA"  LOC = A10 |IOSTANDARD = LVCMOS33;
NET "VGA_RED" LOC = "H14" | IOSTANDARD = LVTTL | DRIVE = 8 | SLEW = FAST ;
NET "VGA_GREEN" LOC = "H15" | IOSTANDARD = LVTTL | DRIVE = 8 | SLEW = FAST ;
NET "VGA_BLUE" LOC = "G15" | IOSTANDARD = LVTTL | DRIVE = 8 | SLEW = FAST ;
NET "VGA_HSYNC" LOC = "F15" | IOSTANDARD = LVTTL | DRIVE = 8 | SLEW = FAST ;
NET "VGA_VSYNC" LOC = "F14" | IOSTANDARD = LVTTL | DRIVE = 8 | SLEW = FAST ;
```

APÊNDICE B – CÓDIGO FONTE VHDL VDECTOVGA

```
----------------------------------------------------
-----------Componente VDECTOVGA-----------------------
----------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity vdectovga is
    Port ( start_vdec : in   STD_LOGIC;
               llc1_vdec  : in   STD_LOGIC;
               field_vdec : in   STD_LOGIC;
               hsync_vdec : in   STD_LOGIC;
               vsync_vdec : in   STD_LOGIC;
               r_in_vdec  : in   STD_LOGIC;
               g_in_vdec  : in   STD_LOGIC;
               b_in_vdec  : in   STD_LOGIC;
               vsout_vtv  : out  STD_LOGIC;
               hsout_vtv  : out  STD_LOGIC;
       r_out_vtv  : out  STD_LOGIC;
             g_out_vtv  : out  STD_LOGIC;
             b_out_vtv  : out  STD_LOGIC);
  end vdectovga;

architecture Behavioral of vdectovga is

type estado_type is (S0, S1 ,S2, S3, S4, S5,S6, S7, S8, S9,S10);
signal Sreg1 : estado_type;
signal Sreg2 : estado_type;
signal Sreg3 : estado_type;
signal Sreg4 : estado_type;

signal vsoutvtv, hsoutvtv          : std_logic;
signal videoon,videoh,videov   : std_logic;

Begin
```

```
        --sinais de saida
        vsout_vtv <= vsoutvtv;
        hsout_vtv <= hsoutvtv;

        --------------------------------------------------------------------------
        ---------------Gera os sinais de sincronismo Horizontal-------------------
        --------------------------------------------------------------------------

        Sreg1_machine: process (llc1_vdec , field_vdec, hsync_vdec )

        variable c1 : integer range 0 to 200;
        variable c2        : integer range 0 to 1000;
        variable c3        : integer range 0 to 200;
        begin

        if llc1_vdec'event and llc1_vdec = '1' then
             if  start_vdec = '0' then        -- condicao inicial
                        hsoutvtv <= '0';

                      c1 := 0;            -- conta ciclos de clock para sincronismo
horizontal ( 3,77us, 100 ciclos)
                      c2 := 0;            -- conta ciclos de clock para sincronismo
horizontal ( 28us, 755 ciclos)
                      c3:= 0;             -- conta ciclos de clock para sincronismo
horizontal ( 3,77us, 100 ciclos)

                      Sreg1       <= S1;
                             else
                        case Sreg1 is
         when S1 =>        -- espera hs
                 if hsync_vdec = '0' then
                           hsoutvtv <= '0';

                      Sreg1 <= S2;
                 end if;

        when S2 =>  -- conta hs
                        if c1 < 101 then
                 c1 :=c1+1;
                 sreg1 <=S2;
                 else
                 hsoutvtv <='1';
                      Sreg1 <=S3;
                        end if;
         when S3 =>                      --zera c1 e espera próximo pulso horizontal
                        c1:=0;
```

```
                Sreg1<=S4;
         when S4 =>
              if hsync_vdec = '1' then
                        hsoutvtv <= '1';
                   Sreg1 <=S5;
                                end if;
                                when S5 =>
                    -- conta hs
                  if c2 < 755 then
                            c2 :=c2+1;
                            sreg1 <=S5;
                            else
                           hsoutvtv <='0';
                     sreg1 <=S6;
                     end if;
                   when S6 =>
                       if c3 < 101 then
                            c3 :=c3+1;
                            sreg1 <=S6;
                            else
                           hsoutvtv <='1';
                     sreg1 <=S7;
                     end if;
                   when S7 =>                      --zera c2, c3 e vai para o início da
fsm
                   c2:=0;
                        c3:=0;

                   sreg1<=S1;



                           when others =>
                                       null;
              end case;
           end if;
         end if;
        end process;



        ---------------------------------------------------------------
        ---------------Gera os sinais de sincronismo Vertical---------
        ---------------------------------------------------------------

        Sreg2_machine: process (llc1_vdec , field_vdec,vsync_vdec, hsync_vdec )
```

```
        variable c4 : integer range 0 to 10000;       -- conta ciclos de clock para
sincronismo vertical para ajustar início do sinc do vga ( 193us, 5221 ciclos)
        variable c5 : integer range 0 to 2000;        -- conta ciclos de clock para
sincronismo vertical ( 64us, 1727 ciclos)
        variable c6 : integer range 0 to 20000;       -- conta ciclos de clock para
sincronismo vertical para ajustar início do sinc do vga ( 92us, 2473 ciclos)
        variable c7 : integer range 0 to 2000;        -- conta ciclos de clock para
sincronismo vertical ( 64us, 1727 ciclos)
        begin

        if llc1_vdec'event and llc1_vdec = '1' then
             if  start_vdec = '0' then        -- condição inicial
                                  vsoutvtv <= '1';

                    c4      := 0;
                    c5      := 0;                              --  conta  ciclos  de
clock para sincronismo vertical(64us, 2 ciclos)
                    c6      := 0;
                    c7      := 0;
                   Sreg2  <= S1;          -- conta ciclos de clock para sincronismo
vertical(64us, 2 ciclos)
             else
               case Sreg2 is

                      when S1 =>
--verifica o field, se field=1 vai para s2, senão para s5
                           if field_vdec = '1' and
                             vsync_vdec = '1' then
                                       Sreg2 <= S2; else
                                       Sreg2 <= S5;
                             end if;
                             When S2 =>
                             if c4<6904 then
                             c4:=c4+1;
                             sreg2 <=S2;
                             else vsoutvtv <= '0';
                             Sreg2  <=S3;
                             end if;
                           when S3 =>
                  if c5 < 1715 then
                             c5 :=c5+1;
                             sreg2 <=S3;
                             else vsoutvtv <='1';
                   sreg2 <=S4;
                   end if;
```

```
        when S4=>
if  vsync_vdec = '1' then
            c4:=0;
        c5:=0;
        else
        sreg2<=S1;
            end if;
             ----Sinais quando Field for '0'
             When S5 =>
             if field_vdec = '0' and
                   vsync_vdec = '1' then
             Sreg2 <=S6;
             else Sreg2<=S1;
             end if;
             When S6 =>
             if c6 < 2449 then
                   c6 :=c6+1;
                   sreg2 <=S6;
                   else vsoutvtv <='0';
              Sreg2 <= S7;
                  end if;
                   When S7 =>
                   if c7 < 1715 then
                    c7 :=c7+1;
                    sreg2 <=S7;
                    else vsoutvtv <='1';
           sreg2 <=S8;

                   end if;
                   When S8 =>


       if  vsync_vdec = '1' then
            c7:=0;
        c6:=0;
        else
        sreg2<=S1;
             end if;
                  when others =>
                              null;
     end case;
  end if;
 end if;
end process;
```

```
        ----------------------------------------------------------------
        --------Sinais para gerar regiao ativa do video-----------------
        ----------------------------------------------------------------
        Sreg3_machine: process (llc1_vdec, start_vdec, hsoutvtv )

        variable c10      : integer range 0 to 10000;      -- conta ciclos de clock
para iniciar regiao ativa do video horizontal( 1,79us, 48 ciclos)
        variable c11      : integer range 0 to 10000;      -- conta ciclos de clock
para contar regiao ativa do video horizontal( 25,42us, 686 ciclos)
        begin

        if llc1_vdec'event and llc1_vdec = '1' then
             if  start_vdec = '0' then        -- condição inicial
                                 videoh  <= '0';


                      c10     := 0;
                      c11         := 0;

                    Sreg3  <= S1;
             else
               case Sreg3 is

        when S1 =>
                           if  hsoutvtv= '1'  then
                                       Sreg3 <= S2;
                             end if;
                             When S2 =>
                             if c10<46 then
                             c10:=c10+1;
                             sreg3 <=S2;
                             else videoh <= '1';
                             Sreg3  <=S3;
                             end if;
                           when S3 =>
                    if c11 < 682 then
                             c11 :=c11+1;
                             sreg3 <=S3;
                             else videoh <='0';
                       sreg3 <=S4;
                       end if;
                   when S4=>
                         if hsoutvtv = '0' then
                         c10:=0;
                   c11:=0;
```

```
                sreg3<=S1;
                    end if;

        when others =>
                                        null;
             end case;
          end if;
         end if;
        end process;
       Sreg4_machine: process (llc1_vdec, start_vdec, vsoutvtv)
        variable c12      : integer range 0 to 100000;        -- conta ciclos de
clock para iniciar regiao ativa do video vertical( 1020us, 25540 ciclos)
        variable c13      : integer range 0 to 1000000;       -- conta ciclos de
clock para contar regiao ativa do video vertical( 15250us, 411750 ciclos)
        begin
        if llc1_vdec'event and llc1_vdec = '1' then
             if  start_vdec = '0' then        -- condicao inicial
                                videov  <= '0';

                    c12     := 0;
                    c13          := 0;

                   Sreg4  <= S1;
             else
               case Sreg4 is

        when S1 =>
                           if  vsoutvtv= '1'  then
                                       Sreg4 <= S2;
                             end if;
                             When S2 =>
                             if c12< 25743 then
                             c12:=c12+1;
                             sreg4 <=S2;
                             else videov <= '1';
                             Sreg4  <=S3;
                             end if;
                           when S3 =>
                    if c13 < 410048 then
                             c13 :=c13+1;
                             sreg4 <=S3;
                             else videov <='0';
                     sreg4 <=S4;
                     end if;
                    when S4=>
                          if vsoutvtv = '0' then
```

```
              c12:=0;
        c13:=0;

        sreg4<=S1;
            end if;
when others =>
                                null;
     end case;
  end if;
 end if;
end process;

--------Atribuir apenas região ativa do vídeo ao r g b-----------------
videoon <= videoh and videov;
    colors: process (llc1_vdec, start_vdec)
            begin
            if start_vdec='0'
               then r_out_vtv <= '0';       --Tela na cor preta
                    g_out_vtv <= '0';       --'0' porque ausência de cores
                    b_out_vtv <= '0';
               elsif (llc1_vdec'event and llc1_vdec='1')
                  then r_out_vtv <= r_in_vdec and videoon;
                       g_out_vtv <= g_in_vdec and videoon;
                       b_out_vtv <= b_in_vdec and videoon;

                end if;
                end process;
endBehavioral;
```

ANEXO A- MANUAL DA CÂMERA KODO

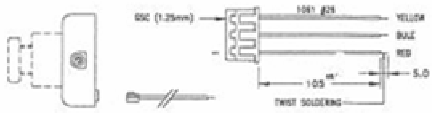
INSTALLATION & OPERATION INSTRUCTION

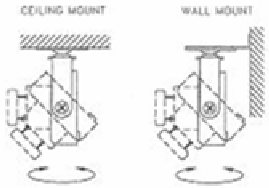
MODEL : µCAM (Black and White CCD CAMERA)

Please read this instruction manual carefully before use.

GENERAL:

* Supply +9V DC to CAMERA and you can have video signal on Video out.
* Electronic iris can be used in applications with a wide spectrum of lighting.
* Angle of view is determined by lens type in use (standard : 3.6mm lens)

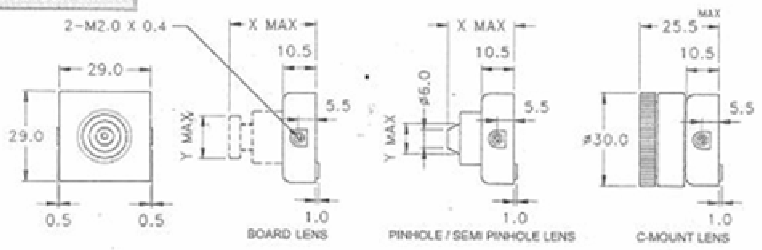




INSTALLATION:

* According to the application front surface mounting can be done with 2mm screws.
* Solder the connector according to the drawing as below:

DIMENSION:

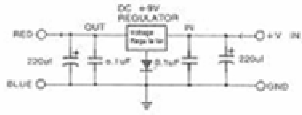


[µCAM : B/W CCD CMAERA WITHOUT SIDE GRIP.]

* X & Y sizes refer to the lens and may differ the lens type.
* If cable extension is required on video out (YELLOW), be sure to use 75 ohms coax cable. If cable extension is required on ground(BLUE), be sure to use sealed brass cable.
* Installation to be made by M2mm screws. Using front hole or side grip.
* Use only from 7.5V DC to 14.5V DC powers supply to operate B/W CCD CAMERA.

RECOMENDABLE CIRCUIT DRAWING OF REGULATED POWER SUPPLY:

* Select the board according to the polarity of the three pin regulator.
* If ripples occur on the power source and cause distortion on the monitor picture, add inductor on circuit of power supply.



PRECAUTION:

* Be sure to use from 7.5V DC to 14.5V DC power source with the correct polarity.
* Do not get wet as it is intended for indoor use.
* Do not striking of hard objects and do not drop it.
* It would be automatically void any warranties that repairing or attempting change circuit and/or applying incorrect power to circuit.

Aiming the B/W CCD CAMERA to spotlight of high contrast and/or intense light may cause blooming to the picture on monitor.

* Camera is equipped with electronic iris shutter circuit, therefore use under fluorescent lamp and/or mercury(vapor) lamp may cause flickering on picture on monitor.
* Avoid using camera near electric appliances that cause various forms of damage.
* Operational instructions should be read in its entirety before connection of power source.
* If camera does not operate properly after it has been connected to the monitor, review the connector soldering instructions to insure proper operation.
* If you have any question or require further information, please contact the local dealer who sold the goods.
* In case any question or further information is required regarding the goods please contact the local distributor.

TECHNICAL SPECIFICATION:

MODEL	μ CAM (EIA)	μ CAM (CCIR)
PICK-UP ELEMENT	1/4 INCH INTERLINE TRANSFER CCD IMAGE SENSOR	
NUMBER OF PIXELS (Useful)	510(H) * 492(V)	500(H) * 582(V)
UNIT SIZE	7.15 μ m(H) * 5.55 μ m(v)	7.30 μ m(H) * 4.70 μ m(v)
SYNC SYSTEM	INTERNAL	
SCANNING SYSTEM	2:1 INTERLACE	
VIDEO OUTPUT	1 Vp_p 75 OHM (UNBALANCED)	
STANDARD BUIL-IN LENS	G-3.6	
S/N RATIO	MORE THAN 45dB (AGC OFF)	
ELECTRONIC IRIS	SHUTTER SPEED 1/60~1/100000sec	SHUTTER SPEED 1/50~1/100000sec
GAMMA CHARACTERISTIC	GAMMA = 0.45	
AGC	MORE THAN 32dB	
POWER SUPPLY	DC 9V [DC +7.5V ~ DC +14.5V]	
CURRENT	MAX 105mA [at 9V]	
STORAGE TEMPERATURE	-20° C ~ 70° C	
OPERATING TEMPERATURE	-10° C ~ 55° C	
WEIGHT	20g	

	LENS					
SPECIFICATION OF LENS	G-2.5	G-3.6	G-6.0	G-12.0	G-3.2	G-3.7
TYPE	GLASS	HYBRID	GLASS	GLASS	PINHOLE	SEMI PIN
FOCAL LENGTH	2.5mm	3.6mm	6.0mm	12.0mm	3.2mm	3.7mm
F POINT	F2.5	F2.5	F2.0	F2.5	F4.5	F2.0
ANGULAR VIEW	130°	92.6°	54°	25.7°	105°	90°
RESOLUTION(HORIZONTAL)	OVER 380 TV LINES (CENTER)				OVER 350 TV LINES (CENTER)	
MINIMUM ILLUMINATION	0.4Lx		0.6Lx	0.6Lx	3Lx	
DIMENSION (X,Y)	32.5Ø17.0	27.5Ø14.0	28.5Ø15.0	26.5Ø14.0	14.5,Ø10.0	18.0,Ø10.0
LENS VARIATION	OPTION	STANDARD	OPTION	OPTION	OPTION	OPTION
	We have so wide lens variation from 2.1 to 16mm board lens. We also provide from 3.2 to 5.5mm pin hole or semi pin hole lens and C-mount lens could be prepared.					

※ We reserve the right to change the design and specification without notice.

KODO Corporation

801, CENTER PLAZA B.D, 1307-7, SEOCHO-DONG, SEOCHO-GU,
SEOUL, KOREA
Tel : 82-2-3481-2248-9 Fax : 82-2-3482-8993

ANEXO B –RELAÇÃO PINAGEM CONECTOR FX2 COM O FPGA

Hiroschi FX2					
Pinos A	VDEC1	FPGA	Pinos B		
1	VCC33	VCCO_0	1	Shield	
2	VCC33	VCCO_0	2	GND	GND
3	NC		3	NC	
4	NC		4	NC	
5	NC		5	GND	GND
6	RESET	B4	6	GND	GND
7	SDA	A4	7	GND	GND
8	SCLK	D5	8	GND	GND
9	P15	C5	9	GND	GND
10	P14	A6	10	GND	GND
11	P13	B6	11	GND	GND
12	P12	E7	12	GND	GND
13	OE	F7	13	GND	GND
14	FIELD	D7	14	GND	GND
15	VS	C7	15	GND	GND
16	HS	F8	16	GND	GND
17	P11	E8	17	GND	GND
18	P10	F9	18	GND	GND
19	P9	E9	19	GND	GND
20	P8	D11	20	GND	GND
21	INTRQ	C11	21	GND	GND
22	SFL	F11	22	GND	GND
23	P7	E11	23	GND	GND
24	P6	E12	24	GND	GND
25	P5	F12	25	GND	GND
26	P4	A13	26	GND	GND
27	P3	B13	27	GND	GND
28	P2	A14	28	GND	GND
29	LLC2	B14	29	GND	GND
30	P1	C14	30	GND	GND
31	P0	D14	31	GND	GND
32	PWRDN	A16	32	GND	GND
33	NC	B16	33	GND	GND
34	NC	E13	34	GND	GND
35	NC	C4	35	GND	GND
36	NC	B11	36	GND	GND
37	NC	A11	37	GND	GND
38	NC	A8	38	GND	GND
39	NC	G9	39	GND	GND
40	NC	D12	40	GND	GND
41	NC	C12	41	GND	GND
42	NC	A15	42	GND	GND
43	NC	B15	43	GND	GND
44	NC	C3	44	GND	GND
45	NC	C15	45	GND	GND
46	GND	GND	46	GND	GND
47	NC	D10	47	LLC1	E10
48	GND	GND	48	NC	D9
49	VCC5		49	VCC5	
50	VCC5		50	Shield	

ANEXO C- BLOCO DIAGRAMA FUNCIONAL DO CHIP ADV7183B.

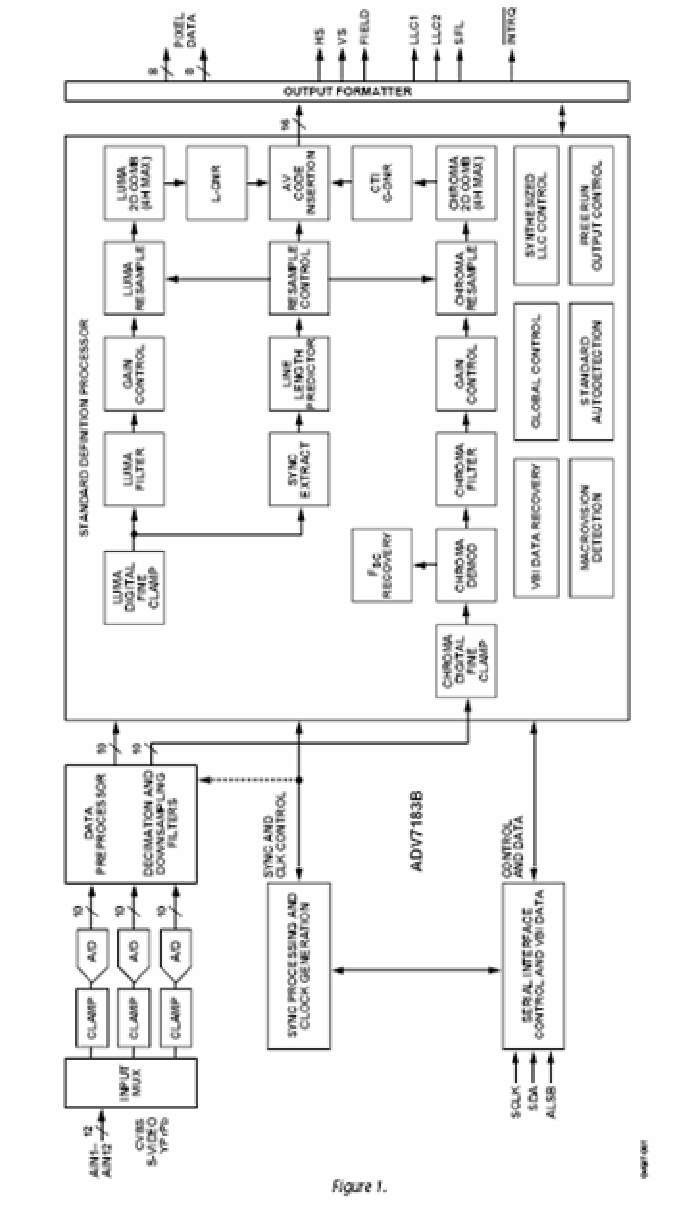


Figure 1.