UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

SISTEMA EMBARCADO PARA RASTREAMENTO DE IMAGENS

João Paulo Silveira dos Santos

ORIENTADOR: RICARDO LOPES DE QUEIROZ

PROJETO FINAL DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

BRASÍLIA/DF: JUNHO/2009

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

SISTEMA EMBARCADO PARA RASTREAMENTO DE IMAGENS

João Paulo Silveira dos Santos

MONOGRAFIA DE PROJETO FINAL DE GRADUAÇÃO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO.

APROVADA POR:

_____

RICARDO LOPES DE QUEIROZ, Ph.D, ENE/UnB
(ORIENTADOR)

_____

RICARDO PEZZUOL JACOBI, Dr, CIC/UnB
(EXAMINADOR)

_____

RICARDO ZELENOVSKY, Dr, ENE/UnB
(EXAMINADOR)

BRASÍLIA, 19 DE JUNHO DE 2009.

RESUMO

O presente trabalho apresenta um sistema embarcado em FPGA para processamento de imagens. O principal objetivo é a identificação de uma luva e o rastreamento de sua posição em uma seqüência de imagens. Este projeto consistiu no desenvolvimento dos controladores para aquisição das imagens pelo FPGA, processamento por intermédio do método de rastreamento baseado na cor do objeto e desenvolvimento dos controladores para exibição do resultado em um monitor VGA.

ABSTRACT

This work presents a FPGA embedded system for image processing. The main objective is the identification of a glove and the tracking of its position in a sequence of images. This project developed controllers for the image acquisition to the FPGA system, processing this data by the tracking method, based on the object color, and development of the controllers for the exhibition of the result of the processing in a monitor VGA.

Agradecimentos

Agradeço a Deus pela vocação recebida;

Agradeço à minha Mãe, meu Pai, meu irmão e minha família, que sempre me apoiaram e me incentivaram nos estudos e trabalhos necessários à minha formação;

Agradeço a Kelma Jaqueline, que com sua amizade sincera e companhia, sempre me apoiou e esteve ao meu lado durante a realização deste trabalho;

Aos amigos Marcos Emmanuel, Alfredo e Fernando pelas sugestões e apoio a realização deste trabalho;

Aos professores Ricardo Jacobi e Ricardo de Queiroz pela orientação acadêmica e pelo compartilhamento de seus conhecimentos;

Ao departamento de Ciência da Computação pelo espaço cedido no laboratório e a liberação para uso da placa DE-2 durante a realização deste trabalho;

Ao departamento de Engenharia Elétrica e a Universidade de Brasília, em especial aos professores do departamento de Engenharia Elétrica que serviram de inspiração.

Enfim, a todos que contribuíram para a realização deste trabalho os meus sinceros agradecimentos.

SUMÁRIO

LISTA DE TABELAS.

LISTA DE FIGURAS

LISTA DE SIGLAS

AS: *Active Serial*.

CLB: *Configurable Logic Block.*

CMOS: C*omplementary Metal Oxide Semiconductor.*

CMYK: Ciano, Magenta, Yellow and Black

CODEC: *CODer- DECoder.*

CPU: *central processing unit*.

DAC: *digital-to-analog converter*.

DARPA: *Defense Advanced Research Projects Agency.*

DDR: *Double Data Rating*.

DVD: *Digital Video Disc.*

EAV: *End of Active Video*.

EEPROM: *Electrical Erasable Programmable Read Only Memory*.

EPROM: *Erasable Programmable Read Only Memory*.

FPGA: *Field Programmable Gate Array.*

HDL: *Hardware description language*

HDTV: *High Definition Television.*

HSI: *Hue, Saturation and Intensity.*

HSV: *Hue, Saturation and Value.*

I/O: *Input/ Output*

IOB: *Input Output Block*

IrDA: *Infrared Data Association*.

ISP: *In System Programmability.*

ITU – R: *International Telecommunication Union – Radiocommunication.*

LAB: *Logic Array Block.*

LE: *Logic Element*.

LED: *Light emission Diode.*

LER: Lesões de Esforço Repetitivo.

LSB: *Least Significant Bit.*

LUT: *Look-up Table.*

MOS: *Metal Oxide Semiconductor*.

MSB: *Most Significant Bit.*

NTSC: *National Television System Committee.*

PAL: *Phase Alternation Line.*

PCI: *Personal Computer Interface.*

PLL: *Phase Locked Loop.*

RGB: *Red, Green and Blue.*

RTL: *Register Transfer Level.*

SAV: *Start of Active Video*.

SCL: *Serial Clock Line.*

SD: *Secure Digital.*

SDA: *Serial Data Line.*

SDRAM: *Synchronous Dynamic Random Access Memory*.

SECAM: *Sequetiel Couler Avec Mémoire*.

SOPC: *System-On-Programmable-Chip*.

SRAM: *Static Random Access Memory*.

TRC: Tubo de Raios Catódicos.

USB: *Universal Serial Bus*.

VGA: *Video Graphics Array.*

VHDL: VHSIC *Hardware Description Language.*

VHF: Very High frequency.

VHSIC: Very High Speed Integrated Circuit.

XGA: Extended *Graphics Array.*

YCbCr: *luminance, blue chrominance, red chrominance.*

YIQ: Luminance, In-phase, Quadrature.

YUV: *Luminance/chrominance.*

1. INTRODUÇÃO

1.1. *Interação Homem Máquina*

Atualmente, os principais dispositivos de interação homem máquina são teclado e mouse, que fazem com que o homem se adapte a requisitos da máquina de uma forma não natural. O uso intensivo dessas interfaces pode inclusive ocasionar problemas de saúde, como lesões de esforço repetitivo (LER).

Há alguns anos, pesquisadores têm dedicado esforços para o desenvolvimento de formas mais naturais para interação homem máquina. A forma mais natural de comunicação humana é através de imagens ou sons, assim uma interface homem máquina capaz de identificar comandos de voz ou de gestos seria mais intuitiva.

Sistemas capazes de identificar comando de voz têm sido desenvolvidos há décadas. Alguns produtos foram desenvolvidos e são comercializados como, por exemplo, celulares que tem capacidade de executarem tarefas a partir de comandos de voz.

Por outro lado, sistemas baseados em visão computacional não tiveram desenvolvimento similar, pois exigem grande esforço computacional e complexidade dos algoritmos. Entretanto, nos últimos anos o aumento da capacidade computacional tem viabilizado o desenvolvimento do processamento de vídeo em tempo real e as pesquisas sobre visão computacional têm evoluído significativamente. Algumas aplicações como realidade virtual, operações remotas e tradução de linguagem de sinais tem sido beneficiadas com o desenvolvimento das técnicas de reconhecimento de gestos.

Um sistema de reconhecimento de imagens pode ser utilizado nas seguintes áreas:

- Interface homem máquina: O mouse ou teclado pode ser controlado através de gestos da mão.
- Jogos de computador: usar a mão para interagir com o jogo pode ser uma forma mais natural em diversas aplicações.
- Controle de sistemas mecânicos: usando a mão para controlar remotamente um dispositivo manipulador.

- Vigilância: Para observação do deslocamento de pessoas ou objetos no ambiente.

1.2. *Motivação e Objetivos*

A principal motivação desse trabalho foi de desenvolver um sistema embarcado para trabalhar com processamento de imagens em tempo real. A arquitetura desse sistema embarcado baseia-se em dispositivos reconfiguráveis, o que torna o desenvolvimento do sistema mais flexível e adaptável a outros sistemas. O dispositivo reconfigurável empregado é o FPGA (*Field Programmable Gate Array*) capaz de executar algoritmos em *hardware*.

Este trabalho de conclusão de curso tem o objetivo de desenvolver um sistema embarcado em FPGA para processamento de imagens capaz de identificar e acompanhar o movimento de um objeto de cor pré-definida em uma sequência de imagens.

Utilizando os recursos da placa DE-2 da Altera esse sistema é capaz de processar imagens oriundas de uma câmera fotográfica digital convencional e exibir o resultado de seu processamento em um monitor VGA comum.

1.3. *Organização do trabalho*

O capítulo 2 apresenta a fundamentação teórica e as ferramentas utilizadas para o desenvolvimento desse projeto. O capítulo fala sobre a placa utilizada e os recursos de programa e *hardware* utilizados para execução desse trabalho.

No capítulo 3 é apresentado um pouco da teoria de espaço de cores e dos principais sistemas de vídeo utilizados para o desenvolvimento do sistema. O capítulo 4 descreve as etapas de desenvolvimento do sistema discutindo alguns problemas que surgiram durante o desenvolvimento deste trabalho.

O capítulo 5 apresenta os teste e resultados do trabalho. E por fim, o capítulo 6 apresenta a conclusão do trabalho.

2. PLATAFORMA DE DESENVOLVIMENTO

2.1. *Introdução.*

Este capítulo tem o objetivo de apresentar as tecnologias utilizadas para o desenvolvimento deste trabalho de conclusão de curso. Inicia com uma breve explicação da placa utilizada para o do sistema e do seu principal componente, o FPGA (*Field Programmable Gate Array*). Aborda a seguir os recursos utilizados para o desenvolvimento do trabalho, incluindo os programas e a linguagem de descrição de hardware utilizada.

2.2. *Placa Altera DE2.*

Para o desenvolvimento deste trabalho de conclusão de curso utilizamos a placa de desenvolvimento e ensino Altera DE2. Trata-se de uma ferramenta útil para a aprendizagem sobre sistemas digitais reconfiguráveis. Ao contrário das principais placas de desenvolvimento que tem finalidade educacional e são baseadas em FPGAs, a placa Altera DE2 possui uma plataforma completa pronta para atividades de ensino, podendo ser usada em atividades que vão desde simples exercícios que ilustram conceitos fundamentais a projetos que exigem um conhecimento mais sofisticado.

A placa Altera DE2 tem como elemento principal o FPGA Cyclone II 2C35 que possui encapsulamento de 672 pinos. Os componentes mais importantes da placa estão conectados a esse FPGA, o que garante ao usuário o controle de todos os aspectos operacionais da placa.

2.2.1. *Componentes e Layout da placa Altera DE2*

A placa ALTERA DE2 possui diversas características que permitem ao usuário implementar uma grande diversidade de circuitos desde simples circuitos até projetos multimídias. A figura 2.1 é uma fotografia da placa DE2 com a identificação de cada componente que compõe a placa.

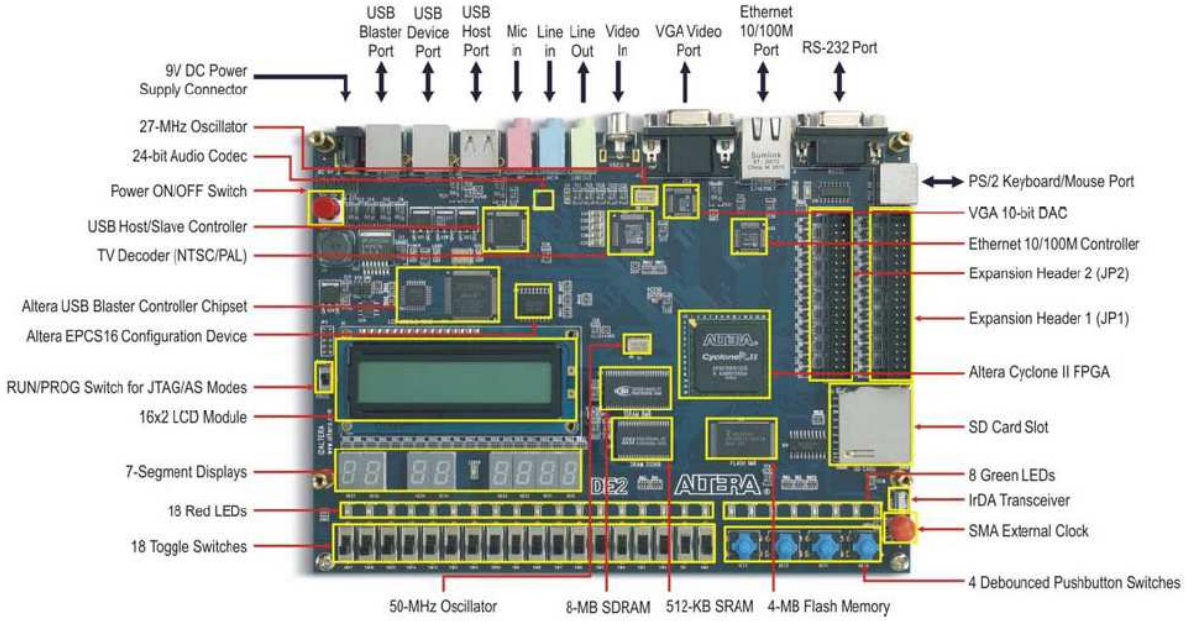


Figura 2-1 Placa Altera DE2 [1].

Conforme ilustra a figura 2.1 a DE2 possui os seguintes componentes.

- FPGA Altera *Cyclone*® II 2C35;
- Dispositivo de configuração Serial - EPCS16;
- USB Blaster (*on board*) para programação e controle API para usuário; ambos JTAG e Active Serial. (AS) são modos de programação suportados;
- 512-Kbyte SRAM;
- 8-Mbyte SDRAM;
- 4-Mbyte memória Flash;
- SD Card socket (leitor de cartões SD);
- 4 chaves do tipo *pushbutton*;
- 18 Interruptores ;
- 18 LEDs vermelhos;
- 9 LEDs verdes;
- Osciladores de 50-MHz e 27-MHz para fornecimento de relógios;
- 24-bit audio CODEC com conectores para *line-in*, *line-out* e microfone;
- VGA DAC (três *10-bit high-speed* conversores analógico digital) com conector do tipo VGA-out;
- TV *Decoder* (NTSC/PAL) e entrada para conector de TV;
- Controlador *Ethernet* 10/100 com um conector;

- Controlador USB *Host/Slave* com conectores USB do tipo A e B;
- Transceptor RS-232 com conector de 9 pinos;
- Conector PS/2 para mouse/teclado;
- Transceptor IrDA;
- Duas 40-pinos *Expansion Headers* com proteção do tipo diodo.

2.3. *FPGA (Field Programmable Gate Array)*

O FPGA é um circuito integrado que pode ser configurado por *software* e serve para implementar circuitos digitais, como por exemplo, processadores, interfaces, controladores e decodificadores. Basicamente, um FPGA consiste em uma matriz de módulos que podem ser de três tipos [2].

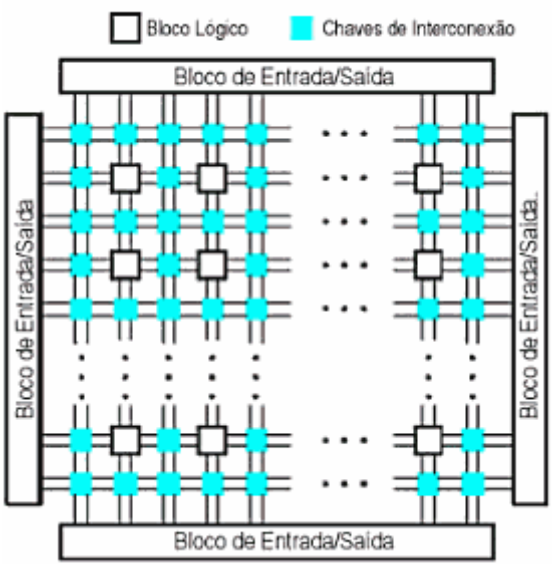


Figura 2-2 - Estrutura simplificada de um FPGA [3].

O principal elemento do FPGA é o bloco lógico reconfigurável (CLB- Configurable Logic Block) estruturado na forma de um arranjo bi-dimensional. Os CLBs possuem elementos que permitem a construção da lógica do usuário. O principal componente de um CLB é LUT (Look-up Table), uma memória onde são escritas as tabelas verdade das funções lógicas a serem implementadas. Além da LUT, o CLB também possui um flip-flop do tipo D que permite o CLB trabalhar de forma combinacional (sem relógio) ou síncrono (com relógio), também existe um sinal de habilitação [4].

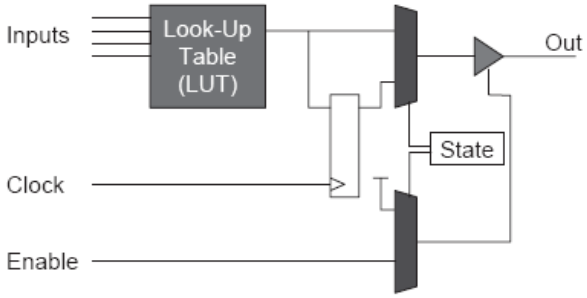


Figura 2-3 - Diagrama de blocos do CLB de um FPGA. [4]

Além do CLB o FPGA possui circuitos de interface entre as saídas dos CLB's e o exterior do FPGA, chamados de IOBs (Input Output Blocks). Estes circuitos são constituídos por *Buffers* bidirecionais com saída em alta-impedância. É pela programação de um IOB que um pino é configurado como entrada, saída, bidirecional ou coletor aberto. Por fim, o terceiro tipo de bloco é composto pelas matrizes de interconexões que são recursos responsáveis por interligar as entradas e saídas dos CLBs e IOBs. Os módulos de interconexão estão situados nas interseções entre os canais de roteamento verticais e horizontais, interligando os segmentos de fiação que permitem rotear os sinais entre os CLBs e os IOBs. As chaves programáveis de roteamento apresentam algumas características, que definem propriedades do FPGA tais como a velocidade e o tempo de propagação dos sinais e características tais como volatilidade. Basicamente existem hoje três tipos de tecnologia de programação das chaves de roteamento, sendo elas:

a)SRAM (*Static Random Access Memory*): nessa tecnologia, a chave de roteamento ou comutador é um transistor de passagem ou um multiplexador controlado por uma memória estática de acesso aleatório SRAM. Por causa da volatilidade dessas memórias, os FPGAs baseados nessa tecnologia precisam de uma memória externa tipo FLASH EEPROM. A desvantagem dessa tecnologia é ocupar muito espaço no circuito integrado, porém é rapidamente reprogramável.

b)Antifuse: essa tecnologia baseia-se num dispositivo de dois terminais, que no estado não programado apresenta uma alta impedância (circuito aberto). Aplicando-se uma tensão, por exemplo, entre 11 e 20 Vdc, o dispositivo forma um caminho de baixa impedância entre seus terminais.

c) *Gate* flutuante: a tecnologia *Gate* flutuante baseia-se em transistores MOS (*Metal Oxide Semiconductor*), especialmente construído com dois gates flutuantes semelhantes aos usados nas memórias EPROM (*Erasable Programmable Read Only Memory*) e EEPROM (*Electrical* EPROM). A maior vantagem dessa tecnologia é a sua capacidade de programação e a retenção dos dados. Além disso, da mesma forma que uma memória EEPROM, os dados podem ser programados com o circuito integrado instalado na placa, característica denominada ISP (*In System Programmability*).

2.4. *Arquitetura do FPGA da família Cyclone II.*

A Família Cyclone™ II é uma família de baixo custo e alta densidade desenvolvida pela Altera. Essa família destaca-se entre as outras famílias de FPGAs de alta densidade por possuir um custo mais baixo que as demais. A placa Altera DE2 possui um FPGA dessa família por isso será feita uma rápida descrição da mesma.

A arquitetura de um FPGA Cyclone II contém um arranjo bidimensional simétrico de linhas e colunas de elementos lógicos baseados em células de armazenamento do tipo SRAM. A arquitetura é baseada em LABs, cada LAB (*Logic Array Block)* é composto por uma série de LE (*Logic Element).* As linhas e as colunas são interconectadas com velocidades variadas capazes de fornecer interconexões eficientes entre LABs, blocos de memória e multiplicadores internos [5].

Cada LABs contêm 16 LEs (*logic elements*). Um LE é uma pequena unidade lógica que permite uma implementação eficiente das funções lógicas do usuário. LABs são organizados em linhas e colunas no dispositivo. A família CycloneII tem uma densidade entre 4.068 a 68.416 LEs. A figura 2.4 apresenta o diagrama de blocos de um LE da arquitetura Cyclone II.

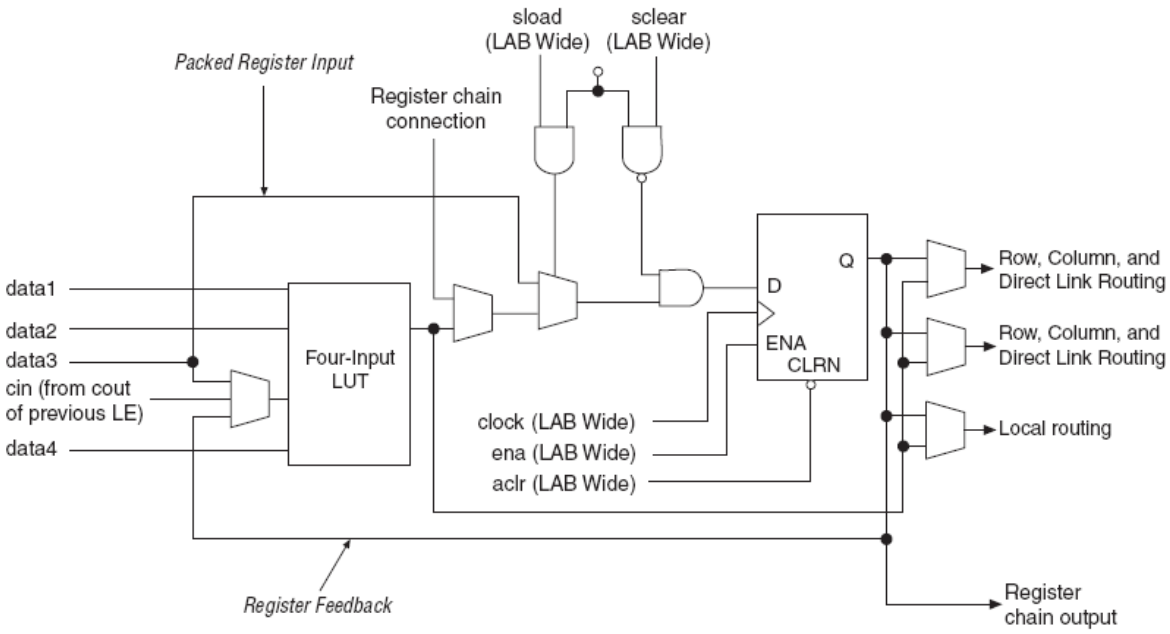


Figura 2-4 - LE (Logic Element) da arquitetura Cyclone II. [5]

Os dispositivos Cyclone II possuem uma rede global de relógio que permite até 4 PLLs (*phase Locked Loops*). Essa rede consiste de 16 linhas de relógio que percorrem todo o dispositivo, podendo alimentar todos os elementos, tais como IOBs (*Input or Output blocks*), LES, multiplicadores e blocos de memória internos. Além disso, os PLLs Cyclone II podem gerar relógios com diversas freqüências distintas.

Os blocos de memória M4K são blocos de memória bidirecionais com 4k bits mais bits de paridade. Esses blocos são organizados em colunas pelo dispositivo entre alguns LAB. Os blocos multiplicadores internos são capazes de implementar até duas multiplicações de 9x 9 bits ou uma multiplicação de 18x18 bits a uma velocidade de até 250 MHz.

Cada pino de um dispositivo da família Cyclone II é alimentado por um IOB localizado no final de linhas e colunas de LABs, que é localizada na periferia do dispositivo. Os pinos de I/O suportam diversos padrões, tais como o padrão PCI (*Personal Computer Interface*) e o padrão para interface de memória externa do tipo DDR (*Double Data Rating*).

A figura 2.5 mostra um diagrama de blocos do dispositivo EP2C20 da família Cyclone™ II:

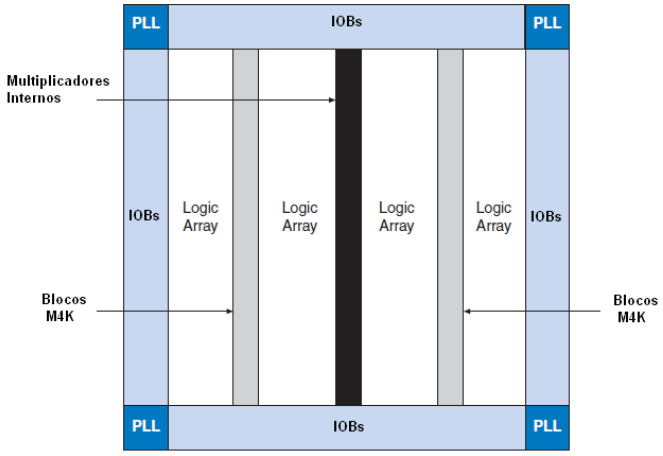


Figura 2-5 - Diagrama de blocos do dispositivo EP2C20 da família Cyclone™ II [5].

O número de blocos de memórias M4K, PLLs, blocos multiplicadores internos, linhas e colunas de LABs dependem de cada dispositivo da família Cyclone ™ II. A tabela 2-1 mostra os recursos dos dispositivos dessa família, o dispositivo presente na placa Altera® DE2 é do tipo EP2C35

Tabela 2-1 - Característica dos dispositivos da família Cyclone™ II.[5]

Dispositivo	Colunas LAB	Linhas LAB	LEs	PLLs	Blocos de Memória M4K	Blocos multiplicadores Internos
EP2C5	24	13	4,608	2	26	13
EP2C8	30	18	8,256	2	36	18
EP2C20	46	26	18,752	4	52	26
EP2C35	60	35	33,216	4	105	35
EP2C50	74	43	50,528	4	129	86
EP2C70	86	50	68,416	4	250	150

2.5. *O programa Quartus II*

O programa Quartus 2 é um ambiente de desenvolvimento de projetos fornecido pela empresa Altera. Trata-se de uma ferramenta computacional de análise e síntese de sistemas digitais, capaz de reduzir o tempo de desenvolvimento do projeto. Esse programa possui duas versões e ambas estão disponíveis para download na internet ou em DVD. A versão completa é disponibilizada por meio de uma assinatura de um ano de validade e tem suporte para todos os dispositivos da Altera. A versão gratuita desse programa é a *Web-edition* que tem suporte a maioria dos dispositivos da Altera.

O programa Quartus II pode ser descrito como uma ferramenta completa para o desenvolvimento de projetos baseados em FPGA, pois este programa possui todas as ferramentas de criação de um projeto sem a necessidade de nenhuma ferramenta externa. Este programa permite o desenvolvimento de um projeto seguindo diferentes metodologias. A síntese do projeto é baseada nos recurso do FPGA alvo. Além disso, o Quartus II provê ferramentas para simulação lógica, analise de tempo, edição de planta baixa e geração e carga de configurações nos FPGAs.

A especificação do circuito pode ser feita por editor de esquemático, que gera o diagrama lógico do circuito, ou por editor de texto através da descrição do circuito lógico utilizando linguagens de descrição de *hardware*. Além disso, o uso do editor de formas de onda auxilia na simulação do circuito.

Existem ainda outras ferramentas do programa Quartus II que facilitam o desenvolvimento de um projeto como as ferramentas *MegaWizard Manager* e SOPC *Builder* que facilitam a adição de lógica e códigos de propriedade intelectual ao projeto.

Para a correção e avaliação do projeto, o programa Quartus II possui diversas ferramentas de depuração que ajudam a analisar o projeto em cada estado, que vai desde o seu estado inicial até ao programa final carregado no dispositivo, o que é feito por meio de simulações, análise de dados internamente ao dispositivo, entre outras.

2.6. *Linguagens de descrição de Hardware*

Há vários anos, linguagens de programação tais como Pascal e C são utilizadas para descrever programas que são executados em computadores de forma convencional. De forma similar, no campo do projeto digital, os projetistas sentiram a necessidade de uma linguagem para descrever os circuitos digitais. Dessa forma, as linguagens de descrição de Hardware (HDL - *hardware description languages*) foram criadas. As linguagens HDLs permitem aos projetistas modelarem processos concorrentes que são comumente encontrados em elementos de hardware, e com isso as linguagens de descrição de hardware tais como Verilog HDL e VHDL tornaram-se populares. A linguagem Verilog HDL foi lançada em 1983 pela empresa Gateway Design Automation. Algum tempo depois a linguagem VHDL foi desenvolvida através de um contrato da DARPA. Os simuladores de

Verilog e VHDL que permitem a simulação de circuitos digitais de grande proporção, foram rapidamente aceitos pelos projetistas.

Mesmo com a popularidade das linguagens HDLs para verificação de lógica, os projetistas tinham que manualmente traduzir o projeto feito em linguagens HDL para o esquema do circuito com as interconexões entre as portas lógicas. Foi no final dos anos 1980 que a metodologia de descrição de hardware mudou radicalmente. A partir de então, o projetista especificava como era o tráfego de dados pelos registradores e como eles seriam processados em nível de registradores (RTL - *register transfer level*) e ao utilizar uma linguagem de descrição de hardware, os detalhes das interconexões para implementar o circuito eram automaticamente extraídos por meio de ferramentas de síntese.

A grande aceitação do projeto utilizando ferramentas de síntese lógica fez com que as linguagens HDLs se transformassem na principal ferramenta para projetos digitais. Os projetistas não precisavam mais localizar cada transistor para construir um circuito digital, eles podiam descrever circuitos complexos em um nível abstrato em termos de sua funcionalidade e fluxo de dados, utilizando construções similares às providas por linguagem de programação. As ferramentas de síntese lógica são responsáveis por implementar a funcionalidade especificada em termos de elementos lógicos de suas interconexões.

Além disso, as linguagens HDLs começaram a ser usadas para o desenvolvimento em nível de sistemas, ou seja, utilizadas para simulação em placas de sistemas interconexão de barramentos, FPGAs e PAL. Com isso, o desenvolvimento de cada circuito integrado, usando linguagens de descrição de Hardware, poderia ser verificado diretamente numa placa.

2.7. *Verilog*

A linguagem de descrição de hardware Verilog permite o desenvolvimento de sistemas digitais através de descrição em alto nível de abstração, ao mesmo tempo permite o uso de ferramentas computacionais para auxílio ao projeto.

Mesmo que a sintaxe da linguagem Verilog seja parecida com a linguagem C, sua semântica é baseada em operações concorrentes de hardware, o que é totalmente diferente da linguagem C.

A linguagem Verilog descreve um sistema digital como um conjunto de módulos (modules). Cada um destes módulos tem interface com outros módulos assim como uma descrição de como estão ligados. Um módulo representa uma unidade lógica que pode ser especificada por comportamento ou estruturalmente (ou a combinação dos dois). Uma especificação por comportamento define o comportamento de um sistema digital (módulo) que usa as construções tradicionais de uma linguagem de programação, como por exemplo, ifs, instruções de atribuição, entre outras. Uma especificação estrutural expressa o comportamento de um sistema digital (módulo) como uma interconexão de módulos secundários. Os módulos secundários devem ser primitivos ou especificados por comportamento. A primitiva de Verilog inclui portas lógicas, por exemplo, uma porta NAND. [6].

A estrutura básica de um módulo é:

```
module  <nome_do_módulo> ( < lista_de_portas > );
        < declarações >
        < elementos_do_módulo>
endmodule
```

<nome_do_módulo> é o nome que identifica o módulo. <lista_de_portas> são as portas de comunicação do módulo com outros elementos dos sistema que podem ser de entrada (*input),* de saída (*output)* ou portas bidirecionais (*inout)* que são usados na ligação a outros módulos. Em <declarações> temos a especificação das estruturas de dados como registros ou memórias bem outras construções executáveis como funções.

Os <elementos_do_módulo> descrevem o comportamento, ou a estruturação do módulo ou ambos e pode ter um bloco *initial*, blocos *always*, atribuições contínuas ou instâncias de módulos, etc. Apresenta-se abaixo, o exemplo de um modulo com o comportamento de uma porta NAND.

```
//Modelo do comportamento de uma porta NAND
module NAND(in1, in2, out);
        input in1, in2;
        output out;
        //atribuição contínua
        assign out = ~(in1 & in2);
endmodule
```

As portas in1, in2 e out são os nomes dos fios de entrada e saída do módulo. A instrução *assign* é uma forma utilizada para modelar circuitos combinacionais, onde as saídas são resultados diretos das entradas. Ou seja, as variáveis da direita na expressão são

constantemente monitoradas e quando ocorrem alterações, a expressão é reavaliada e o resultado é propagado para o lado esquerdo(*out*).

Um módulo Verilog é instanciado no início do programa e dura até o fim do programa, pois como se trata de modelos de hardware não é possível retirar ou adicionar elementos dos módulos depois que estão funcionando. Cada vez que um módulo é instanciado, ele é nomeado. Como podemos ver no exemplo abaixo, onde temos os módulos NAND1 e NAND2 que são instancias da porta NAND do exemplo anterior. O módulo abaixo é a especificação de um módulo AND a partir de dois módulos NAND. A saída de um dos módulos é ligada as duas entradas do outro por meio de um fio interno (*wire*) w1:

```
module AND(in1, in2, out);
        //Modelo estrutural de uma porta AND a partir de duas NANDs
        input in1, in2;
        output out;
        wire w1;
        //duas instâncias de NAND
        NAND NAND1 (in1, in2, w1);
        NAND NAND2 (w1, w1, out);
endmodule
```

Para instanciar um módulo devemos usar a seguinte sintaxe.

<nome_do_módulo> <nome_instância> (<lista_portas>);

Os principais tipos de dados em Verilog são registradores (*reg*) e fios (*wire*). As variáveis *reg* armazenam o último valor que lhes foi atribuído, ao contrário das linhas que representam ligações físicas entre entidades estruturais tais como portas. Uma variável *wire* representa apenas o nome do fio e não armazena valores.

As variáveis *reg* e *wire* podem ter quatro valores, que são: 0 zero lógico ou falso, 1 um lógico ou verdadeiro, x valor lógico indeterminado e z alta impedância em portas *tri-state*. As variáveis *reg* são inicializadas com x e qualquer fio (*wire*) não conectado também tem valor x.

O tamanho deste tipo de variáveis pode ser especificado na declaração. Por exemplo, as declarações seguintes:

```
reg [0:7] A, B;
```

```
wire [3:0] s_dados;
```

especificam os registradores A e B com 8 bits cada e o bit mais significativo é o bit 0, o fio s_dados possui 4 bits onde o bit 3 é o mais significativo.

3. PROCESSAMENTO DE IMAGENS E SISTEMAS DE VÍDEO

3.1. *Introdução*

Este capítulo tem o objetivo de apresentar uma breve noção da teoria de espaço de cores. Apresenta também uma descrição do funcionamento de um monitor e descreve os principais sistemas de vídeo utilizados durante o desenvolvimento do projeto.

3.2. *Introdução*

Este capítulo tem o objetivo de apresentar uma breve noção da teoria de espaço de cores. Apresenta também uma descrição do funcionamento de um monitor e descreve os principais sistemas de vídeo utilizados durante o desenvolvimento do projeto.

3.3. *Espaço de cores*

Um espaço de cores é uma representação matemática de um conjunto de cores. Existem diversos espaços de cores, dentre eles podemos citar o modelo RGB (*red*, *green* e *blue* ou vermelho, verde e azul) que é utilizado em computadores, o modelo YIQ, YUV ou YCbCr usado em sistemas de vídeo, e CMYK, utilizado em impressoras coloridas [7]. Nenhum desses modelos de cores está diretamente relacionado com a noção mais intuitiva de tom, saturação e brilho. Para a programação, o processamento e a manipulação desses sistemas de cores foram criados modelos tais como HSI (tom, saturação e intensidade) e HSV (tom, saturação e valor).

3.3.1. Sistema de cores RGB

O sistema RGB é formado pelas cores vermelhas, verde e azul. Ele é usado pelos computadores em seu sistema de exibição de informações. Vermelho, verde e azul são as três cores primarias aditivas que ao se somarem, produzem a cor desejada. Esse sistema é representado por um sistema de coordenadas cartesianas tridimensional conforme a figura 3.1. A diagonal da figura representa intensidades iguais nas três cores, o que resulta no conjunto de tons de cinza.

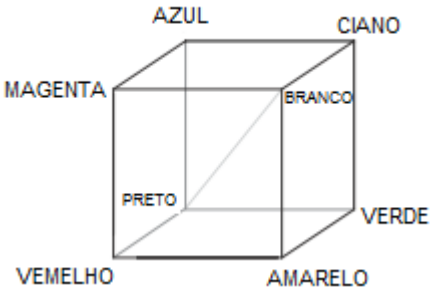


Figura 3-1- Cubo do sistema RGB [7]

O sistema de cores RGB é o mais usado no sistema gráfico de computadores. Por essa razão, a escolha do espaço de cor RGB no desenvolvimento de um projeto em computação simplifica sua arquitetura pela disponibilidade de diversas bibliotecas e API's gráficas baseadas nesse sistema de cor.

Entretanto, o sistema RGB não é o mais eficiente para tratar as imagens do mundo real. As três componentes de cor precisam ter uma mesma largura de banda para gerar qualquer cor do cubo de cores do sistema RGB. Como conseqüência, um sistema de bufferização precisa, da mesma forma, ter a mesma resolução para as três cores. Além disso, o sistema RGB não é o mais eficiente para fazer o processamento de imagens. Por exemplo, para modificar a intensidade ou a cor de um determinado pixel é necessário ler as três cores do *buffer* da imagem, fazer as modificações necessárias e gravar novamente as três cores no *buffer*. Se o sistema tiver gravado no buffer de imagem valores de intensidade e cor, alguns processamentos podem ser mais rápidos. Por essas e outras razões, muitos sistemas de vídeo usam luminosidade e duas diferenças entre cores. Os sistemas mais comuns que utilizam esse padrão são YUV, YIQ e YCbCr que são sistemas similares mas com algumas diferenças entre si [7].

3.3.2. Sistema de cores YUV

O espaço de cores YUV é usado pelos padrões de sistema televisão em cores PAL (P*hase Alternation Line*), NTSC (*National Television System Committee*) e SECAM (*Sequetiel Couler Avec Mémoire*). O sistema de cores Preto e Branco utiliza apenas a informação de luminância. Para introduzir informação de cor no sistema televisivo, mantendo a compatibilidade com o sistema Preto e Branco, foram introduzidas as componentes de cor U e V. Desta forma, os receptores de televisão em preto e branco

poderiam ser utilizados normalmente, capturando apenas a informação de luminância, enquanto que os aparelhos coloridos decodificam a informação adicional de cor para exibir a imagem colorida [7].

3.3.3. Sistema de cores YIQ

O espaço de cores YIQ é uma forma derivada do espaço de cores YUV, utilizada opcionalmente pelo padrão NTSC. A letra I significa “em fase” (*in-phase*) e a letra Q significa “em quadradatura” (in quadrature), representando o método de modulação usado para transmitir a informação de cor [7].

3.3.4. Sistema de cores YCbCr

O espaço de cores YCbCr foi desenvolvido como parte da resolução ITU-R BT.601, que visa criar um padrão mundial para uma componente de vídeo digital. O padrão YCbCr é uma codificação digital para o espaço de cores YUV. As três componentes tem por definição 8 bits, sendo que a componente de luminância varia entre a faixa de 16-235 e as componentes Cb e Cr variam na faixa de 16-240 [7]. Existem diversos formatos de YCbCr tais como 4:4:4, 4:2:2, 4:1:1 e 4:2:0 [7]. Nesses formatos, a primeira componente refere-se ao número de amostras de luminância, enquanto que a segunda representa a crominância azul e a terceira a crominância vermelha.

3.4. *Tecnologias de exibição de Imagens*

Atualmente, existem diversas tecnologias de exibição de imagens, as mais comuns são NTSC/PAL, SECAM E VGA/XGA. Além de existirem variações dessas tecnologias existem também tecnologias mais novas como HDTV. Basicamente, essas tecnologias funcionam da mesma forma, ou seja, para desenhar uma figura elas desenham um pixel por vez.

A tecnologia usada para formar uma imagem em uma tela é que rege as características do sinal de vídeo. Há muito tempo, os monitores com tubo de raios catódicos (TRC) são usados para exibir imagens. O TRC é o maior componente interno de um monitor baseado nessa tecnologia. A imagem mostrada pelo TRC é formada por um feixe de elétrons que percorre toda a tela, esse feixe é responsável por excitar o fósforo presente na parte frontal do tubo. O feixe deve percorrer toda a tela iniciando pelo canto

superior esquerdo e percorrer da esquerda para a direita e de cima para baixo em uma sequência de linhas horizontais. Cada linha é composta por um conjunto de *pixels*. Um conjunto de *pixels* e linhas formam uma imagem e para o deslocamento desse feixe existe um conjunto de bobinas que utilizam campo magnético ou elétrico para desviar o feixe para a posição apropriada na tela do TRC [8].

Grande parte do desenvolvimento da tecnologia usada nos TRCs foi baseada nas propriedades elétricas e dos recursos disponíveis no tempo de sua criação. Como a freqüência utilizada na rede elétrica americana é de 60Hz decidiu-se, então, usar a freqüência da rede elétrica como base de freqüência, por isso a taxa de campos por segundo é de 60. No início das transmissões de televisão, as imagens dos TRCs eram em preto e branco. O primeiro padrão de transmissão de imagens era monocromático e foi definido por uma associação entre um comitê, emissoras de televisão e fabricantes de receptores, no início dos anos 1950. Essas imagens eram formadas em níveis de cinza e deveriam ser transmitidas analogicamente de um ponto a outro.

Em 1945, o espectro de altas frequências VHF foi dividido em 13 canais pelo US Federal Communication Commision, Dessa forma, a largura de banda máxima de cada canal estava determinada, e caberia às especificações técnicas acrescentar, posteriormente, a informação dentro desses limites de banda. Além disso, com o desenvolvimento do sistema colorido, o novo padrão deveria permitir que os aparelhos em preto e branco continuassem a exibir as imagens extraídas da transmissão para sistemas coloridos, por isso a informação de cor teve de ser adicionada ao sinal de uma forma que não alterasse muito a forma nem a largura de banda do sinal transmitido.

O processo de varredura para controlar o movimento do feixe de elétrons de um monitor TRC envolve uma série de passos. A imagem é formada por um conjunto de linhas horizontais e ao alcançar o fim da tela, o feixe de elétrons deve retornar ao ponto de partida, ou seja, ao canto superior esquerdo para recomeçar a desenhar um novo quadro. Para coordenar esse processo devem existir sinais que contenham pulsos de sincronismo para informar os eventos do eixo horizontal e vertical. Cada linha deve ser iniciada com um pulso de sincronismo horizontal seguido pelos valores dos pixels que compõem a linha e ao final um novo pulso de sincronismo deve informar ao monitor para voltar ao canto esquerdo para iniciar uma nova linha. Então, após desenhar todas as linhas do quadro, um sinal de sincronismo vertical deve informar ao monitor para iniciar um novo quadro. No

topo e na base da tela existe uma região chamada de *blanking*, onde não ocorre a exibição de imagem [8].

No sistema NTSC cada quadro é composto por metade das linhas da imagem, ou seja, $525/2 = 262{,}5\ linhas$. O Feixe de elétrons faz a varredura em linhas alternadas escrevendo o primeiro campo. O próximo campo é escrito nas linhas restantes e assim, cria-se um quadro completo. A figura 3.2 abaixo apresenta um padrão onde às formas de onda dente de serra horizontal e vertical ilustram a varredura entrelaçada de linhas ímpares. Para simplificar no exemplo são utilizadas apenas 21 linhas que são entrelaçadas com dois campos por quadro. Cada campo contém metade das linhas 10,5 linhas, em cada campo considera-se que uma linha será utilizada durante o retraço vertical [13].

Começando no canto superior esquerdo, o ponto A na figura 3.2, o feixe varre a primeira linha da esquerda para a direita e retorna para a esquerda começando a varrer a terceira linha do quadro dessa forma todas as linhas ímpares são varridas até chegar a parte inferior do quadro. Após varrer 9,5 linhas o feixe chega ao ponto B, nesse ponto inicia o retraço vertical, durante esse retraço uma linha é varrida. O ponto C é o ponto final do retraço e a partir desse ponto o feixe varre as linhas pares até chegar ao ponto D onde o retraço vertical começa levando o feixe do ponto D para o ponto A [13].

As formas de onda dente de serra apresentadas na figura 3.2, são responsáveis pelo movimento do feixe pela tela, a onda de maior freqüência é responsável pelo traço e retraço horizontal, o aumento linear da corrente nas bobinas de deflexão horizontal deflete o feixe através da tela num movimento uniforme e continuo do traço da esquerda para a direita da tela. No pico da subida a forma de onda dente de serra inverte sua direção e diminui rapidamente para seu valor inicial essa reversão é responsável por trazer o feixe para o ponto inicial. Já a onda de menor freqüência é responsável pelo movimento vertical do feixe na tela, de forma parecida com a deflexão horizontal a onda faz com que o movimento de traço na tela seja para baixo com velocidade uniforme e o retraço da onda ocorre rapidamente de forma que o feixe retorne ao topo da tela. Esse processo cria um quadro completo na tela e se repete 30 vezes por segundo.

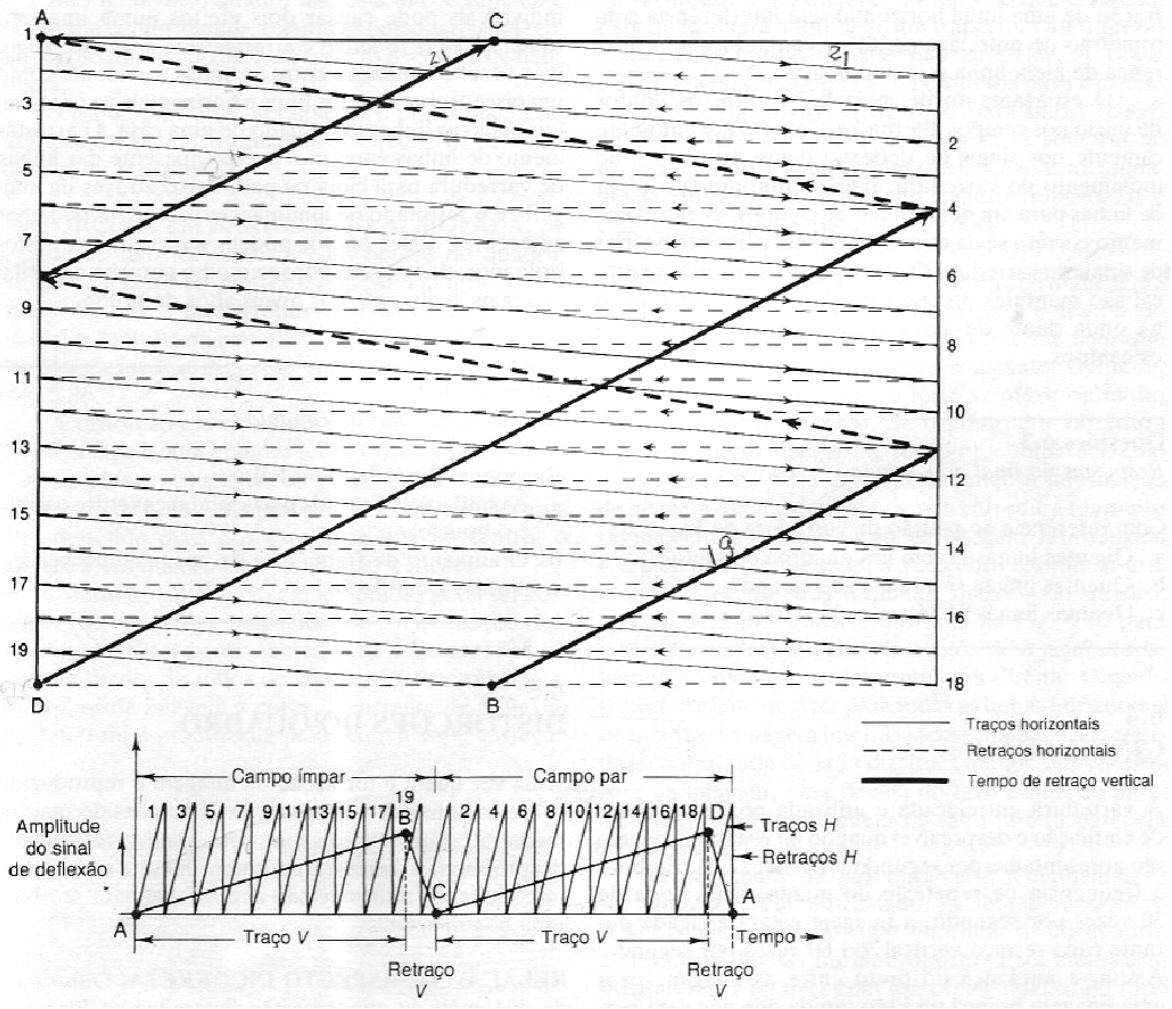


Figura 3-2 Amostra do padrão de varredura para um quadro com 21 linhas de entrelaçamento, 10,5 linhas por campo. A forma de ibda debtter de serra correspondendo a deflexão H e V são mostradas embaixo [13].

3.5. *Sistemas de vídeo*

Essa sessão tem o objetivo de apresentar as características dos sinais de vídeos utilizados no trabalho. O primeiro sinal é o baseado no padrão VGA 640X480 usado para exibir as imagens recebidas e tratadas pelo FPGA. O segundo padrão é o baseado na resolução BT 656 Interface de vídeo para circuitos integrados, que dita o formato de vídeo entregue ao FPGA pelo circuito integrado, após ter convertido o vídeo enviado pela câmera no formato NTSC.

3.5.1. Padrão VGA

A idéia básica de um sinal VGA é a de transmitir os sinais de vermelho, verde e azul de cada pixel sem nenhuma codificação em linhas analógicas separadas. Adicionadas

a essas linhas, existem também outras duas linhas, uma de sincronismo vertical e outra de sincronismo horizontal, que são digitais [9].

O processo de exibição de uma imagem é exemplificado na figura 3.2. Esse processo é iniciado no canto superior esquerdo com o pixel 0,0 e ocorre de forma progressiva, no final de cada linha, a linha é incrementada e a coluna é zerada

Os sinais de sincronismo são responsáveis por pintar ou reiniciar a tela. O sinal de sincronismo horizontal diz ao monitor quando exibir a próxima linha e o sinal de sincronismo vertical diz ao monitor a hora de iniciar a exibir um novo campo. Na figura 3.3 temos os detalhes do sinal de sincronismo vertical e na figura 3.4 o sinal de sincronismo horizontal. Esses sinais devem ser exatos, pois todos os monitores testados durante o trabalhão desligaram quando dois sinais de sincronismo não tinham os valores corretos.

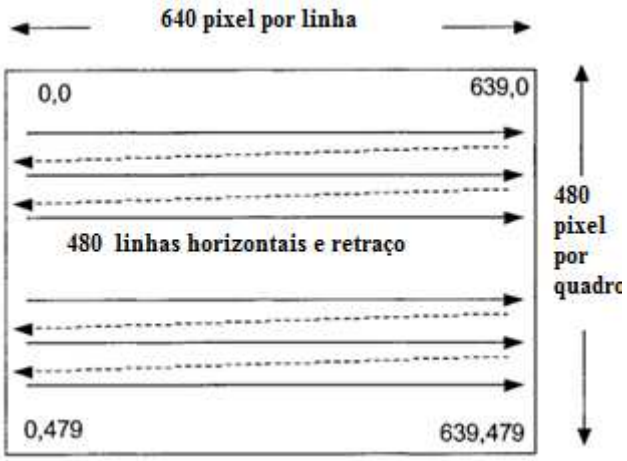


Figura 3-3 - Imagem VGA 640x 480 pixels [9]

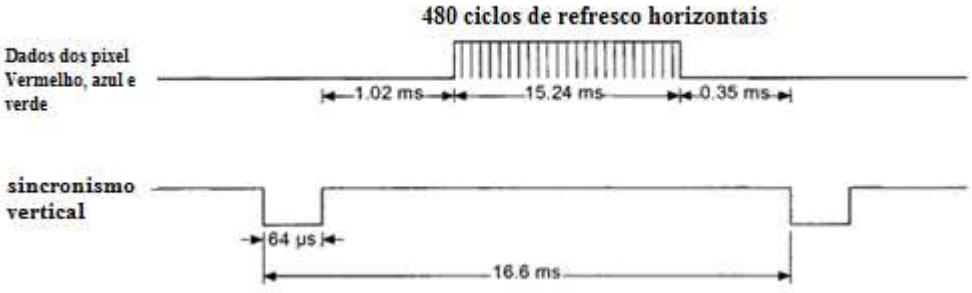


Figura 3-4 - Temporização do sinal de sincronismo vertical para 640x 480, 60Hz [9].

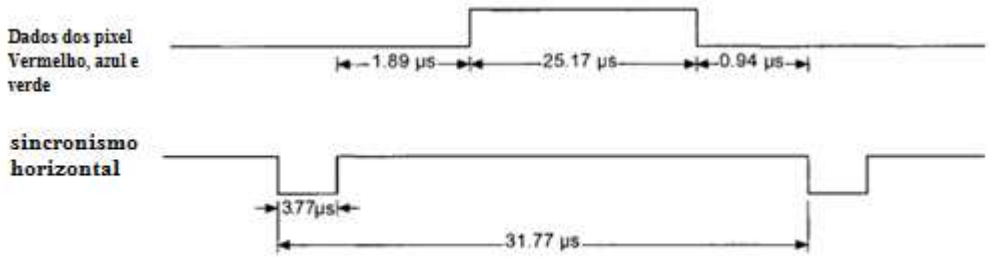


Figura 3-5 - Temporização do sinal de sincronismo horizontal para 640x 480, 60Hz [9].

3.5.2. ITU-R BT 656 Interface de vídeo para circuitos integrados.

A resolução ITU-R BT.656 define as interfaces paralela e serial para transmitir vídeos digitais no espaço de cores YCbCr 4:2:2 entre equipamentos de estúdio e aplicações de vídeo profissionais. Esse padrão utiliza 8 ou 10 bits com dados de YCbCr multiplexados e um relógio de 27 MHz. Ao invés de utilizar os sinais convencionais de sincronismo horizontal e vertical em linhas separadas, essas informações são transmitidas por meio de códigos adicionados a informação de vídeo, reduzindo o número de fios (e pinos do Circuito integrado) necessários para uma interface BT.656 [10].

Dados auxiliares (tais como áudio, *closed caption*) podem ser adicionados a informação de vídeo durante o intervalo de *blanking*, onde nenhuma informação de imagem é transmitida. Eliminando a necessidade de uma interface de áudio separada e sinais de controle adicionais.

Os dados no formato YCbCr 4:2:2 são multiplexados em palavras de 8 ou 10 bits e são transmitidos da seguinte forma: $Cb_0Y_0Cr_0Y_1Cb_2Y_2Cr_2$, ou seja, 2 *bytes* de Y para cada *byte* de Cb ou Cr e assim por diante. Adicionados ao sinal de vídeo, existem os sinais de controle que são chamados de SAV (do inglês *start of active video* - início do vídeo ativo) e EAV (do inglês *End of active video* – Fim do vídeo ativo). Esses sinais eliminam a necessidade de linhas separadas para controle de sincronismo vertical e horizontal. Após um código de SAV, a transmissão do vídeo é iniciada com uma amostra de Cb na sequência de CbYCr até um código de EAV ser transmitido. A figura 3.5 ilustra o formato de uma linha transmitida.

Cada linha do vídeo é amostrada a 13,5 MHz, gerando 720 amostras de YCbCR 4:4:4 de 24 bits. Essas amostras são, então, convertidas para YCbCr 4:2:2 de 16 bits,

resultando em 720 amostras de Y e 360 amostras de Cb e de 360 Cr a cada linha. As informações de Y, de Cb e de Cr são multiplexadas e o clock é dobrado para 27MHz.

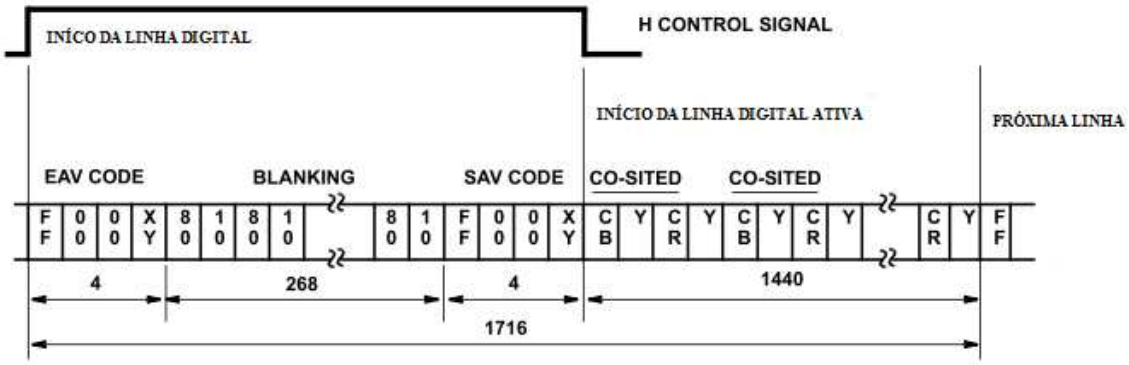


Figura 3-6 - Interface paralela BT.656 para dados no formato de sistemas de vídeo de 525/60.

Os dados de SAV e EAV são formados por um sequência de 4 bytes compostos por FF, 00, 00 e XY onde XY é um byte que contém informações sobre o vídeo, essas informações estão listadas na tabela abaixo:

Tabela 3-1- BT.656 sequência dos bits de SAV e EAV.

	dado 8 bits								**dado 10 bits**		**Valor em hexadecimal(8bits)**
	D9	**D8**	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**	
Início de um SAV ou EAV	1	1	1	1	1	1	1	1	1	1	FF
	0	0	0	0	0	0	0	0	0	0	00
	0	0	0	0	0	0	0	0	0	0	00
Palavra de status	1	F	V	H	P3	P2	P1	P0	0	0	--

A palavra XY que indica se a sequência representa um EAV ou SAV é compostas pelos bits na ordem detalhada na tabela 3.1 acima e cada bit tem o seguinte significado:

- F = 0 para campo 1; F = 1 para campo 2;
- V = 1 durante o blanking vertical;
- H = 0 representa SAV e H = 1 EAV;
- Os bits P3 até P0 são os bits de proteção.

- $P3 = V \oplus H$;
- $P2 = F \oplus H$;
- $P1 = F \oplus V$;
- $P0 = F \oplus V \oplus H$;

onde $\oplus$ representa a função ou-exclusivo. Esses bits de proteção são utilizados para a detecção e correção de erros na recepção da palavra XY.

Em cada imagem gerada existem intervalos de *blanking* horizontal entre as linhas e de *blanking* vertical durante o intervalo de retorno ao início do quadro. Cada linha terá o formato descrito pela figura 3.5 e o formato do quadro é descrito pela figura 3.6.

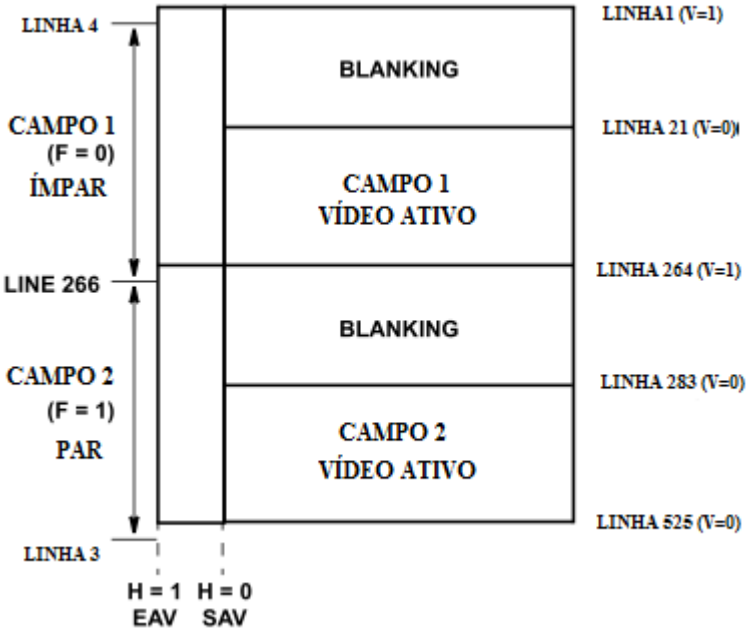


Figura 3-7 - Intervalos típicos para blanking vertical para sistemas de vídeo 525/60.

4. SISTEMA DESENVOLVIDO E RESULTADOS

4.1. *Introdução*

Este capítulo tem o objetivo de apresentar o sistema desenvolvido. Esse sistema é dividido em módulos, cada bloco foi criado e testado antes de ser adicionado aos demais. Alguns problemas encontrados durante o desenvolvimento do projeto serão listados com a solução adotada. O sistema desenvolvido é capaz de rastrear o deslocamento de um objeto seguindo o seu centróide, sendo a distinção entre os objetos das imagens é feita pela cor do objeto previamente definida.

4.2. *Visão global do sistema*

Para o desenvolvimento do sistema foi necessário um estudo sobre os recursos disponíveis na Placa ALTERA DE-2. A captura de vídeo poderia ser feita de duas formas, utilizando uma webcam ligada a entrada USB da placa ou utilizando uma câmera fotográfica digital ligada a entrada de vídeo composto da placa. Por facilidade, foi escolhida a entrada de vídeo composto.

Após a captura das imagens, a próxima etapa é o seu armazenamento. A placa ALTERA DE-2 disponibiliza como recursos de memória 60 Kbytes de memória interna ao FPGA, 8 Mbytes de memória SDRAM, 512 Kbytes de memória SRAM, 4 Mbytes de memória FLASH e um leitor de cartões do tipo SD. Durante o projeto, controladores de memória SRAM e SDRAM foram desenvolvidos, porém o método de captura de imagem utilizado não necessitou de mais que 3 kbytes de memória e para isso pequenas memórias foram criadas internamente ao FPGA.

Após a captura e armazenamento da imagem, a próxima etapa é a localização e rastreamento do objeto. A técnica utilizada é a busca pelo centróide do objeto que se deseja localizar. Sendo o centróide o ponto central de uma entidade, decidiu-se, então, rastrear objetos baseado em sua cor. A cor escolhida foi a verde, pelo fato de ser uma das cores primárias do sistema RGB. Para o cálculo do centróide, deve-se calcular a média das coordenadas dos pixels que compõem o objeto.

Por fim, a imagem resultante deveria ser exibida em um monitor VGA, utilizando o conector VGA presente na placa ALTERA DE-2. A informação exibida no monitor deve

ser controlada pelas chaves presentes na placa. Basicamente as possíveis imagens exibidas são: Imagem capturada pela câmera, Imagem capturada pela câmera mais o centróide representado por um quadrado na cor azul, e a imagem filtrada utilizada para o cálculo do centróide.

4.3. *Módulo Controlador VGA*

A placa ALTERA DE-2 possui um conector de saída VGA. Os sinais de sincronismo para o monitor VGA são gerados diretamente pelo FPGA. Para gerar os sinais analógicos, contendo a informação de cor (vermelha, verde e azul), a placa possui o dispositivo ADV7123, que é um conversor digital-analógico de alta velocidade com palavras de 10 bits. Com esse dispositivo pode-se gerar sinais com uma resolução de 1600 X 1200, com freqüência de 100MHz. A figura 4.1 abaixo representa o circuito esquemático deste controlador VGA.

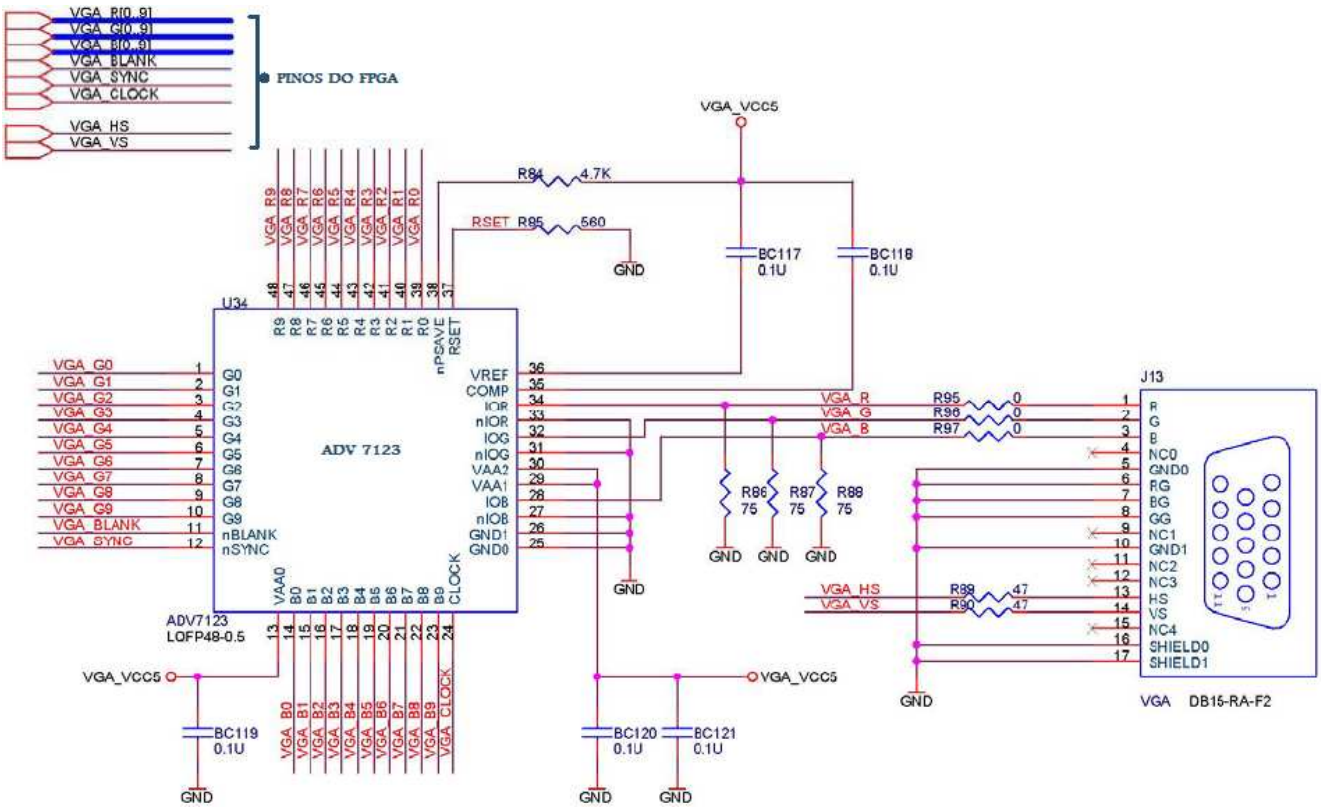


Figura 4-1- Circuito esquemático do controlador de VGA [1].

O formato de saída de vídeo adotado foi o VGA standard, no qual a resolução da tela é de 640x480 pixels. Logo, o sinal de vídeo deve desenhar cada quadro numa freqüência fixa de 60 Hz a fim de se obter o efeito de movimento da imagem. Com isso, a

taxa de atualização é de 40ns para cada pixel do vídeo. Ou seja, a taxa com que os pixels deverão estar disponíveis é de 25 MHz.

Inicialmente, o controlador VGA desenvolvido para esse trabalho de conclusão de curso, possuía apenas os sinais de controle de sincronismo vertical e horizontal do monitor e os sinais de controle do conversor ADV7123, que geravam os sinais de controle explicados no capitulo 3 seção 3.4.1. Para testar esse módulo foi criado um pequeno módulo de controle para desenhar algumas barras coloridas nas cores primárias e nas cores secundárias. O resultado está ilustrado na figura 4.2, a qual mostra o correto funcionamento deste módulo.

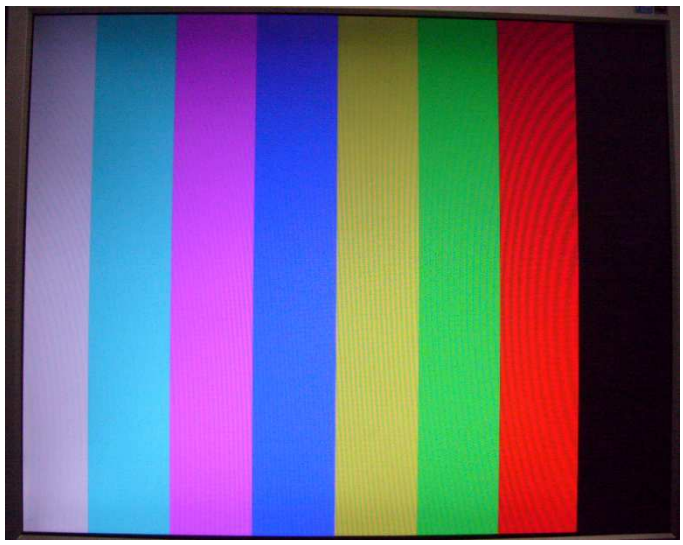


Figura 4-2 - fotografia do resultado do teste do controlador de vídeo.

Até este ponto, o módulo controlador VGA estava funcionando corretamente e não era necessária a adição de nenhum novo sinal de controle. Com esse resultado, encerramos a criação do módulo VGA e partimos para a próxima etapa. A recepção do vídeo.

4.4. *Módulo de Aquisição de Vídeo*

A placa DE2 dispõe de um conversor analógico-digital ADV7181 da Analog Devices [2]. O ADV7181 é um decodificador de vídeo integrado que detecta e converte sinais de banda-base de televisão nos padrões utilizados mundialmente NTSC, PAL e SECAM em sinais de vídeo digital 4:2:2 de 8 ou 16 bits. A placa DE2 está configurada para receber esses sinais digitais apenas em 8 bits.

Para a operação adequada do ADV7181, é necessária a sua configuração, sendo necessário configurar a temporização dos sinais de sincronismo vertical e horizontal,

quantidade de bits das amostras, canal de entrada etc. A configuração é realizada através do barramento I2C da placa DE2.

4.4.1. O barramento I2C

O barramento I2C foi desenvolvido originalmente pela Philips **[3].** O propósito era interconectar a CPU a *chips* controladores de televisores. O barramento I2C é designado para comunicação entre diversos dispositivos relativamente lentos que são acessados intermitentemente. O protocolo adotado é um protocolo simples de pequena largura de banda, chegando a operar a uma taxa de 400 kbps.

O barramento I2C consiste fisicamente de 2 fios e uma conexão com o terra. Os dois fios são bi-direcionais, SDA (*Serial Data Line*) e SCL (*Serial Clock Line*).

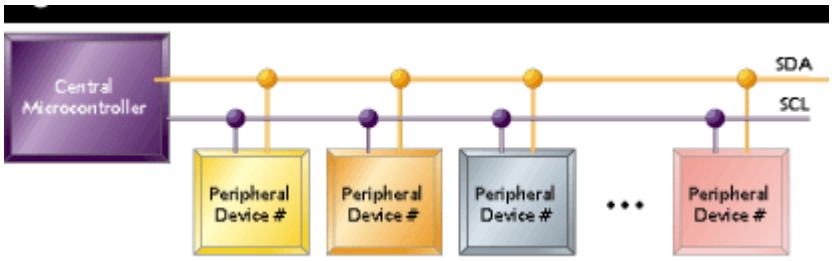


Figura 4-3 - Barramento I2C

A Figura 4.3 apresenta o esquema típico do barramento. Os sinais SCL e SDA em conjunto permitem que a transmissão seja serial. Os dados transmitidos são compostos por 8 bits, transmitidos do mais significativo (MSB) para o menos significativo (LSB). A comunicação inicia-se com o Sinal de Start Bit (S), na qual se mantém a linha SCL em nível lógico alto e a linha de dados muda para o nível baixo, conforme indicado pela Figura 4.4 (a). Em seguida, o endereço do dispositivo é enviado serialmente. O dispositivo correspondente ao endereço enviado responderá com um *Ackwnoledge bit* (a linha de dados é colocada em nível lógico baixo pelo dispositivo). Em seguida, os dados são enviados serialmente e, após a recepção dos 8 bits, o dispositivo envia novamente um *Ackwnoledge bit*. Ao término da comunicação, o dispositivo mestre deverá enviar um sinal de *Stop Bit* (F), conforme a Figura 4.4(b). A Figura 4.5 apresenta um exemplo de uma seqüência completa da comunicação através do barramento I2C.

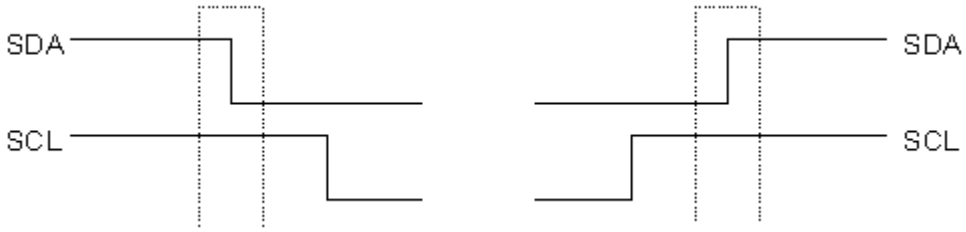


Figura 4-4 - (a) sinalização do Start Bit, (b) sinalização do Stop Bit.

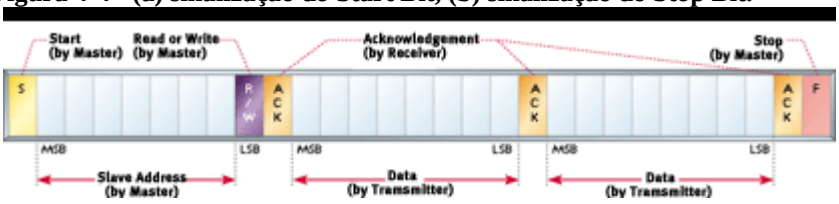


Figura 4-5 - Sequência de comunicação via Barramento I2C [14].

No barramento I2C da placa DE2, o endereço de escrita nos registradores do decodificador de ADV7181 é o 0X40. A escrita em cada registrador ocorre enviando primeiramente o endereço 0x40, em seguida o endereço do registrador interno e depois o valor a ser carregado no mesmo.

O módulo de configuração do ADV7181 tem seu funcionamento caracterizado por uma máquina de estados que lê valores gravados em uma memória ROM e os transfere serialmente de acordo com as regras do barramento I2C.

4.4.2. A Interface de entrada de Vídeo

Após realizar a configuração do decodificador de vídeo, a interface de vídeo está apta a receber os *pixels* do vídeo digitalizado. O ADV7181 disponibiliza em sua porta de saída o sinal de vídeo de acordo com a norma ITU-R BT.656 [5], detalhes dessa resolução foram apresentados no capítulo 3 seção 3.4.2. Os valores disponibilizados na porta de saída estão no formato YCrCb.

Um aspecto relevante a ser notado é o fato de o vídeo decodificado ser disponibilizado na saída do decodificador no formato entrelaçado. Ou seja, primeiramente as linhas ímpares são disponibilizadas e, em seguidas, as pares. Cada linha de vídeo é enviada pelo decodificador segundo uma codificação padronizada por norma [5]. O início de uma linha é sinalizado quando o sinal de SAV (*Start of Active Video*) é enviado e o término é sinalizado por um sinal de EAV (*End of Active Vídeo*).

Durante o intervalo de *Blanking horizontal*, o decodificador envia os valores 0x10 e 0x80, seguidamente.

Seguindo este padrão de envio do decodificador, o módulo de entrada de vídeo, inicialmente para fins de teste, filtra e trabalha somente com os valores de sinal Y por meio de uma máquina de estados. Os valores filtrados são disponibilizados ao módulo de *buffer* do sistema. A freqüência com que estes sinais são disponibilizados na saída do módulo controlador de VGA é de 13,5 MHz.

4.5. *Módulo de Buffer do sistema*

O *Buffer* está posicionado entre o módulo controlador VGA e o módulo de entrada de vídeo. Sua presença se faz necessária devido ao fato de os módulos de entrada e saída de vídeo operarem em freqüências diferentes. Os valores disponibilizados pelo módulo de entrada de vídeo são atualizados na freqüência de 13,5 MHz e os valores requeridos na entrada do módulo controlador devem ser atualizados na freqüência de 25 MHz. Sendo assim, a função do buffer é atender os requisitos de cada módulo e permitir que o fluxo de dados entre eles não seja interrompido.

Inicialmente, a estratégia adotada para a bufferização consiste em enviar ao módulo de saída linhas duplicadas, tendo em vista que a freqüência com que este requer os valores dos pixels é quase o dobro da freqüência com que o buffer é alimentado. O tamanho do buffer implementado é de 3 linhas, ou seja, 1440 bytes.

O resultado da aplicação deste procedimento é que a imagem exibida apresenta um aspecto de esticamento vertical e um rolamento em direção a região superior do monitor. Como podemos ver na figura 4.6 abaixo.

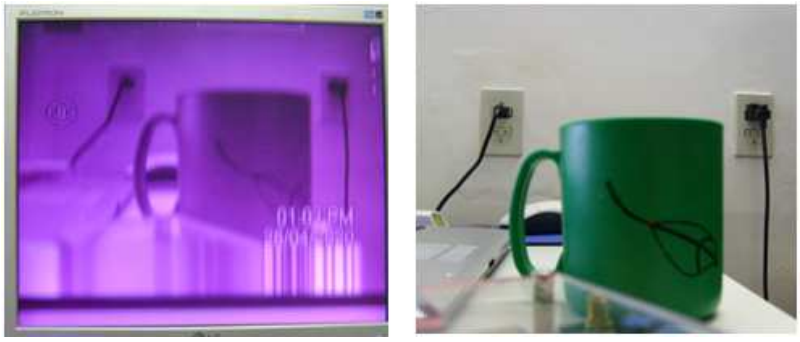


Figura 4-6 - (a) Imagem exibida no monitor VGA com um rolamento vertical. (b) imagem captura pela câmera.

Para solucionar esse problema, algumas soluções foram levantadas a primeira era de guardar imagens completas em uma das memórias. No instante em que o primeiro quadro já estivesse na memória a exibição era iniciada enquanto os próximos quadros continuariam a serem gravados na memória. Para isso seria necessário construir o controlador para as memórias externas, foram construídos um controlador para a memória SDRAM e outro para a memória SRAM que serão explicados na próxima seção. Por fim a solução adotada foi a de sincronizar o sinal de saída com o sinal de entrada a partir dos sinais de sincronismo vertical e horizontal com algumas modificações. Essa solução será explicada na seção 4.7.

4.6. *Controladores de memória SRAM e SDRAM*

A Placa ALTERA DE-2 possui o chip de memória IS61LV256 da ISSI. Essa memória é uma memória CMOS de alta velocidade organizada em 262.144 palavras de 16 bits. Com tempo de acesso típico entre 10 e 15ns. Além disso, essa memória funciona de forma assíncrona, sem a necessidade de sinais de relógio ou de refresco [11].

O módulo de controle da SRAM foi construído de forma a receber os pedidos de leitura ou escrita de módulos externos, seu principal desafio era de ser rápido o bastante para realizar operações de leitura e escrita em intervalos de tempo menores que 40ns, já que a saída de vídeo escrevia os dados a uma freqüência de 25MHz. O módulo de controle foi construído baseado em uma máquina de estados. Em teste, a maior velocidade que esse módulo atingiu foi, com um relógio de 100MHz, 4 estados para terminar uma operação de escrita e de 6 estados par realizar uma operação de leitura, o que fazia com que o controlador não fosse rápido o suficiente para trabalho com o sistema de vídeo desenvolvido.

Dessa forma, o módulo de controle da SRAM desenvolvido não atingiu seu objetivo, foi então levantada a hipótese de que a memória SDRAM poderia solucionar esse problema.

A placa ALTERA DE-2 possui uma memória SDRAM de 8-Mbytes, organizada em 4 bancos de 1.048.576 palavras de 16 bits. O controle de uma memória SDRAM é mais complicado que o da memória SRAM. A construção desse módulo levou muito tempo e não atingiu resultados satisfatórios. Pois uma memória dinâmica possui muitos parâmetros

de temporização que são complexos de serem executados, de tal forma que não foi possível desenvolver um controlador capaz de trabalhar corretamente com os sinais dessa memória.

Em pesquisas pela internet, foi encontrado um módulo de controle da memória SDRAM que fazia escritas e leituras de 4 bytes em sequência a uma velocidade de 66Mhz. Com a adaptação desse módulo de controle da SDRAM e associação ao módulo de buffer, diversas tentativas de escrita de um quadro na memória SDRAM e leitura desse quadro para exibição no monitor VGA foram feitas. Porém o resultado não foi atingido. Ao utilizar uma ferramenta do programa Quartus II, capaz de fazer a verificação das formas de onda de sinais internos ao FPGA, constatou-se que os dados lidos da memória, eram disponibilizados com atraso, depois que o módulo de saída de vídeo já tinha iniciado a conversão digital-analógica para exibição no monitor.

Nesse período uma nova estratégia foi traçada, a idéia era buscar controlar os sinais de sincronismo vertical e horizontal do monitor VGA baseado nos sinais de sincronismo fornecidos pelo circuito integrado ADV7181 utilizado pelo módulo de entrada de vídeo.

4.7. *Módulo de exibição de imagens*

O circuito integrado decodificador ADV7181, além de fornecer os dados do sinal de vídeo no padrão ITU-R BT.656 possui ainda pinos com sinais de controle, que são os sinais de sincronismo vertical e horizontal, e sinal de relógio na freqüência de envio de bits que compõem o quadro.

O bloco decodificador de entrada de vídeo extrai o sinal YCbCr (4:4:4) a partir do sinal YCbCr (4:2:2), além disso ele fornece sinal de relógio de 13,5MHz com sinais de *blanking* indicando o período de dados válidos. Como o sinal de vídeo é entrelaçado as linhas exibidas são repetidas, por isso é necessário que a freqüência dos pixels seja dobrada para 27MHz e o sinal de sincronismo horizontal alterado para 31,4kHz a partir do sinal de 15,7kHz. Os dados são armazenados em memórias internas ao FPGA com capacidade para 1kbyte de modo que cada componente de cor é armazenada em uma memória diferente.

O módulo decodificador VGA foi alterado de tal forma que o sinal de sincronismo vertical é iniciado com o flanco de subida do sinal de sincronismo vertical do módulo de entrada e o sinal de sincronismo horizontal do módulo VGA é sincronizado com o flanco de descida do sinal de sincronismo horizontal que foi dobrado.

Com essas modificações a imagem exibida estava corretamente alinhada com o monitor e não apresentava nenhum rolamento na tela, como ilustra a figura 4.7.

Figura 4-7 - Imagem exibida pelo monitor VGA

Com a exibição correta das imagens, a próxima etapa é implementar o algoritmo de reconhecimento de imagens, esse algoritmo faz a busca do objeto a partir de sua cor, dessa forma, surgiu a necessidade de fazermos uma mudança de espaço de cor, do YCbCr para o RGB, com os dados nesse espaço também é possível exibir a imagem colorida no monitor, já que o mesmo também trabalha com sinais no espaço RGb.

Em [12], podemos encontrar as equações que fazem a mudança entre os espaço de cores YCbCr para o RGB. E são elas:

$$R = 1{,}164(Y - 16) + 1{,}596(Cr - 128); \quad (4.1)$$

$$G = 1{,}164(Y - 16) - 0{,}813(Cr - 128) - 0{,}391(Cb - 128); \quad (4.2)$$

$$B = 1{,}164(Y - 16) + 2{,}018(Cb - 128). \quad (4.3)$$

Com a imagem já no espaço de cor RGB é feita a localização do objeto baseado em sua cor. O algoritmo que calcula o centróide está constantemente observando os valores das componentes RGB. A posição do centróide é calculada pelas equações 4.4 e 4.5.

$$X_{méd} = \frac{1}{n}\sum_{i=0}^{n} Xi; \quad (4.4)$$

$$Y_{méd} = \frac{1}{n}\sum_{i=0}^{n} Yn. \quad (4.5)$$

Quando o pixel é considerado como pixel válido ele é adicionado as equações 4.4 e 4.5, onde *X* é a componente horizontal da posição do pixel e *Y* é a componente vertical da posição do pixel e *n* é a quantidade de pixel válidos.

Com a estimativa da posição do centróide que desejamos localizar a próxima etapa foi a calibragem do sistema, com a definição do intervalo de pixels válidos para o cálculo do centróide. Essa etapa será explicada no capitulo 5.

4.8. *Diagrama de blocos do sistema desenvolvido*

Na figura 4.8 abaixo, temos o diagrama de blocos do sistema desenvolvido, os blocos internos ao bloco FPGA, representam os blocos que foram desenvolvidos durante o projeto, e os blocos externos são os circuitos integrados que estão conectados diretamente ao FPGA.

O bloco decodificador ITU é responsável por receber os dados do conversor ADV 7181 e interpretá-los, armazenar as informações de cada componente de cor que está no formato YCbCr e entregar ao módulo YCbCR/RGB para que a conversão entre os formatos seja feita. O Bloco Filtro/Centróide é responsável por fazer a filtragem das informações do quadro e o cálculo do centróide. Por fim o bloco Controlador VGA é responsável por fazer o sincronismo dos sinais de sincronismo vertical e horizontal de entrada com os sinais de sincronismo de saída entregues ao monitor.

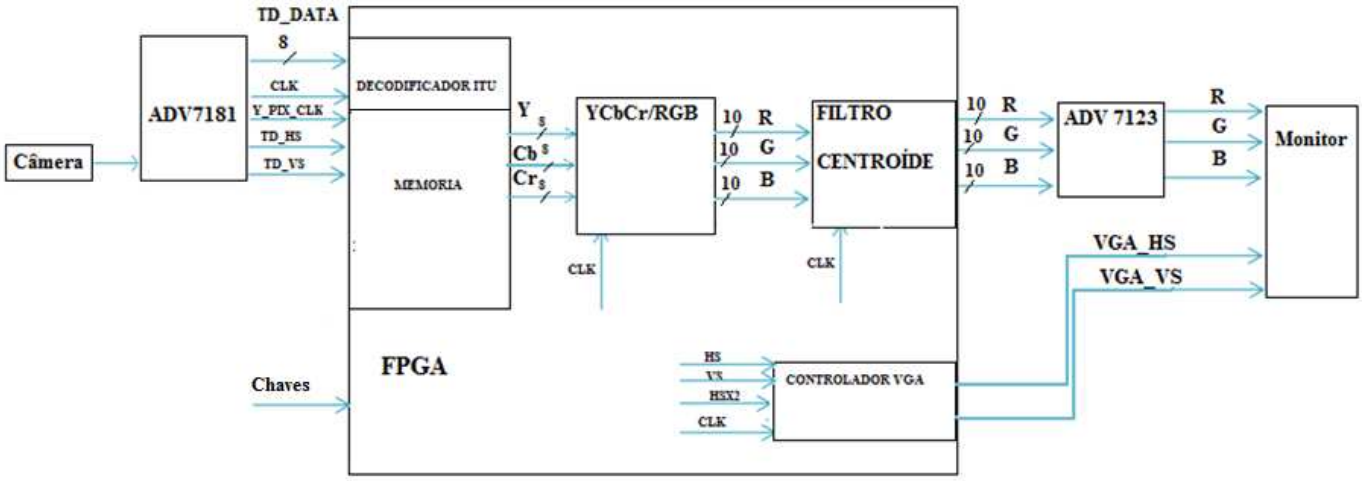


Figura 4-8 - Diagrama de Blocos do sistema desenvolvido.

5. Resultados

5.1. *Introdução*

O objetivo desse capitulo é apresentar as etapas realizadas para a calibragem do sistema e alguns testes realizados para a validação do funcionamento do sistema.

5.2. *Calibragem*

Um sistema de detecção de imagens deve permitir que o usuário utilize o sistema em diferentes condições de iluminação, por exemplo, em um ambiente onde a iluminação natural do sol modifica as características do local e da fonte de iluminação da câmera [1]. Por isso uma recalibração do sistema deve ser feita, ou no sistema deverá existir diferentes calibrações que podem ser acionadas pelo usuário para que o sistema melhore seus resultados.

A primeira etapa para fazer a calibragem do sistema é a escolha do objeto que deverá ser localizado e identificar o intervalo de cores válido para a diferenciação do objeto dos demais na cena.

Esse trabalho de conclusão de curso tem por objetivo à localização e o rastreamento de objetos na cor verde. Para fazer a calibragem de dois tons de verde utilizamos uma luva na cor verde claro e uma caneca na cor verde escuro, dessa forma o sistema pode trabalha em dois tons de verde, sendo possível o ajuste da tom de verde com o acionamento de uma chave.

Um objeto para ser considerado da cor verde tem a componente verde de seus *pixels* com um valor alto. Além disso, as outras componentes de cor devem tem um valor baixo. Cada cor no sistema de cores RGB exibido tem 10 bits, dessa forma o intervalo de intensidade válido para cada cor é entre 0 e 1023. A calibragem foi feita em um ambiente com iluminação fluorescente branca. Após alguns testes escolhemos o intervalo de cor válido para a localização de cada objeto, os valores estão apresentados a tabela 5.1 abaixo:

Tabela 5-1 - Intervalo de cor válida para a localização do objeto

Objeto	Luva	Caneca
R (vermelho)	R < 511	R < 511
G (verde)	G > 511	G > 511
B (azul)	B < 511	B > 511

5.3. *Testes*

Após a etapa de calibragem do sistema alguns testes utilizando a luva e a caneca foram feitos. Os testes foram feitos apenas em locais com iluminação fluorescente branca.

Todas as figuras abaixo são fotografias das imagens exibidas pelo monitor VGA, retiradas por uma câmera fotográfica digital convencional. As figuras 5.1(a) e 5.1(b) exibem a imagem exibida pelo monitor VGA na figura 5.1(a) é exibida a caneca e na figura 5.1(b) a luva.

Figura 5-1 - (a) Imagem da caneca exibida pelo monitor VGA. (b) Imagem da luva exibida pelo monitor VGA.

O próximo teste foi acionar a chave que muda o modo de exibição para exibir apenas os pixels considerados válidos pelo sistema, nessa etapa, é possível verificar o efeito da calibração do objeto. As figuras 5.2(a) e 5.2(b) mostram a identificação da caneca utilizando a calibragem da caneca, figura 5.2(a), e a calibragem da luva figura 5.2(b), nelas podemos verificar a diferença entre as calibragens.

Figura 5-2 - (a) Fotografia do monitor VGA exibindo a imagem segmentada da caneca com calibração para a caneca. (b) Fotografia do monitor VGA exibindo a imagem segmentada da caneca com calibração para a luva.

Da mesma forma as figuras 5.3(a) e 5.3(b) mostram o resultado da identificação da luva utilizando as duas calibragens previamente definidas no sistema.

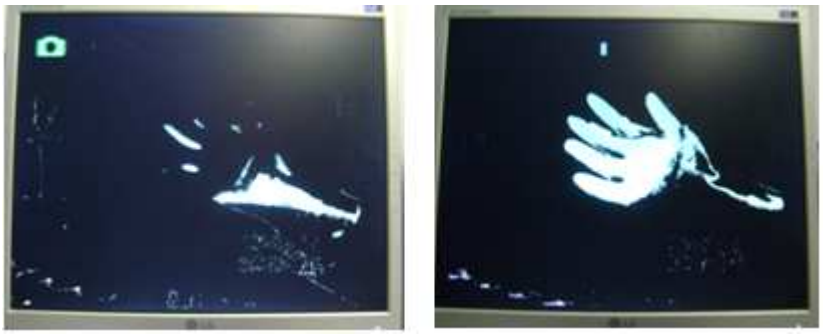


Figura 5-3 - (a) Fotografia do monitor VGA exibindo a imagem segmentada da luva com calibração para a caneca. (b) Fotografia do monitor VGA exibindo a imagem segmentada da luva com calibração para a luva.

A próxima etapa foi acionar a chave que mostra o centróide identificado, esse centróide é representado por um quadrado de tamanho 10x10 pixels na cor azul, o centróide é o ponto central desse quadrado. O resultado dessa etapa está ilustrados nas figuras 5.4 (a) e 5.4 (b)

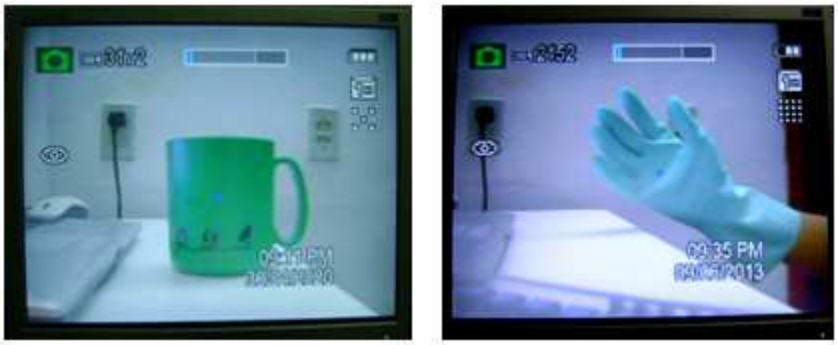


Figura 5-4 - Fotografia do monitor VGA exibindo a imagem com a estimativa da localização do centróide em azul, sendo o objeto (a) caneca, (b) luva.

A última etapa de testes foi verificar se o sistema estava seguindo o deslocamento do objeto em uma sequência de imagens. A figura sequência de imagens da figura 5.5 mostra que o centróide acompanha corretamente o deslocamento da luva, e a sequência de imagens da figura 5.6 apresenta o do centróide seguindo o movimento da caneca na cena.

Figura 5-5 - Sequência de imagens mostrando o deslocamento do centróide acompanhando o movimento da luva.

Figura 5-6 - Sequência de imagens mostrando o deslocamento do centróide acompanhando o movimento da caneca.

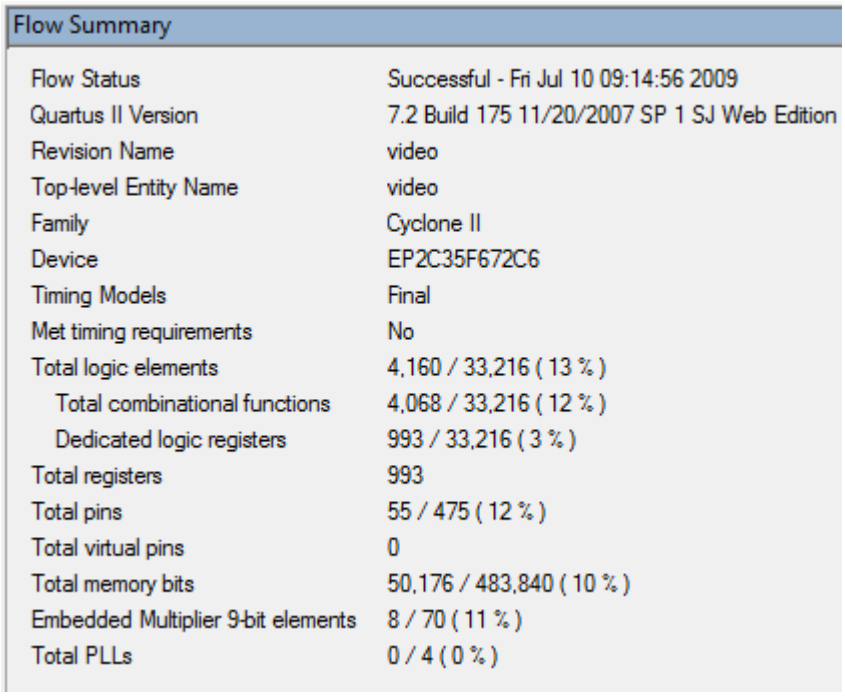
6. CONCLUSÃO.

Por meio deste trabalho foi possível aprender sobre implementação de sistemas digitais em FPGA, controle de memória e técnicas iniciais sobre processamento de imagens.

Uma das etapas mais importantes do trabalho foi à construção de um controlador capaz de trabalhar corretamente com os dados enviados pelo circuito integrado ADV7181, responsável pela entrada dos dados, e a construção do controlador VGA para a exibição dos resultados do processamento da imagem.

Ainda na parte da construção dos circuitos digitais responsáveis pelo controle dos circuitos integrados que estão conectados ao FPGA, uma dificuldade encontrada, que não foi solucionada, foi a construção de controladores de memória SRAM e SDRAM que fossem suficientemente rápidos para trabalhar com o vídeo em tempo real.

A figura 6.1 abaixo apresenta a utilização do FPGA pelo sistema desenvolvido, nela podemos ver que no FPGA ainda existe muitos recurso disponíveis. Foram utilizados apenas 10% da capacidade de memória interna e 13% de elementos lógicos. Assim, é possível observar existem muitos recursos disponíveis para a melhoria do sistema.



Flow Summary	
Flow Status	Successful - Fri Jul 10 09:14:56 2009
Quartus II Version	7.2 Build 175 11/20/2007 SP 1 SJ Web Edition
Revision Name	video
Top-level Entity Name	video
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Met timing requirements	No
Total logic elements	4,160 / 33,216 (13 %)
Total combinational functions	4,068 / 33,216 (12 %)
Dedicated logic registers	993 / 33,216 (3 %)
Total registers	993
Total pins	55 / 475 (12 %)
Total virtual pins	0
Total memory bits	50,176 / 483,840 (10 %)
Embedded Multiplier 9-bit elements	8 / 70 (11 %)
Total PLLs	0 / 4 (0 %)

Figura 6-1 - Relatório da utilização dos recursos do FPGA gerado pelo programa Quartus II após compilação do projeto.

Com os testes, percebe-se que em um sistema de localização de objetos em imagens baseado em sua cor é muito dependente da iluminação do ambiente, fazendo com que o método utilizado no trabalho não seja tão eficiente, de tal forma que o sistema tem a garantia de funcionar apenas nos ambientes para qual foi feita a calibragem em situações de iluminação constante.

Deve-se levar em conta que o método de localização escolhido foi o que melhor se adaptou as condições de memória do sistema desenvolvido.

Para trabalhos futuros o método de localização pode ser melhorado para diminuir a dependência da iluminação do ambiente, ou uma forma de calibragem da cor do objeto pode ser adicionada, para que o usuário possa calibrar o sistema sempre que desejar.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] ALTERA, Corp (2006). **DE2 Development and Education Board, User Manual**. Disponível em: <ftp://ftp.altera.com/up/pub/Webdocs/DE2_UserManual.pdf>. Acesso em janeiro de 2009.

[2] ZELENOVSKY, Ricardo; MENDONÇA, Alexandre. **PC: um guia prático de hardware e interfaceamento.** 3. ed. Rio de Janeiro: MZ Editora, 2002. 1031 p.

[3] COSTA, César; **Projetando controladores digitais com FPGA**. São Paulo: Novatec editora, 2006. 160p.

[4] WILSON, Peter R; **Design recipes for FPGAs.** Grã-Bretanha: British Library Cataloguing in Publication Data. 2006. 289 p.

[5] ALTERA, Corp. (2008). **Cyclone II Device Handbook, Volume 1.** Disponível em: <http://www.altera.com/literature/lit-cyc2.jsp>. Acesso em janeiro de 2009.

[6] THOMAS, Donald E; MOORBY, Philip R. **Verilog hardware description language.** 2. ed. Boston: Kluwer Academic, 1995. 275 p.

[7] JACK, Keith. **Video demystified: a handbook for the digital engineer**. 4th ed. Amsterdam: Elsevier, c2005. 927 p.

[8] LaMothe, André. **The Black Art of Video Game Console Design.** E.U.A.: Sams Publishing, 2005. 956p.

[9] Hamblen, James O; Hall, Tyson S.; Furman, Michael D. **Rapid prototyping of digital systems - Quartus II edition.** E.U.A.: Springer, 2006. 370p.

[10] INTERSIL , Corp (2002). **BT.656 Video Interface for ICs. Application Note.** Disponível em: < http://www.intersil.com/data/an/an9728.pdf>. Acesso em dezembro de 2008.

[11] ISSI, Corp. **256K x 16 HIGH SPEED ASYNCHRONOUS CMOS STATIC RAM** Disponível em: < http://www1.cs.columbia.edu/~sedwards/classes/2007/4840/ISSI-IS61LV25616-SRAM.pdf> acesso em junho 2009.

[12] INTERSIL, Corp (2002). **YCbCr to RGB Considerations. Application Note.** Disponível em: < http://www.intersil.com/data/an/an9728.pdf>. Acesso em dezembro de 2008.

[13] GROB, Bernard. **Televisao e sistemas de video.** 5. ed. Rio de janeiro: Guanabara, 1989. 385.

[14] Kalinsky, David. Kalinsky, Roee **Embedded.com, Introduction to I2C** http://embedded.com/story/OEG20010718S0073 Acesso em julho de 2008.

Anexos

Código VHDL e Verilog desenvolvidos para o projeto.

```
module Video (
                        OSC_27,
                        RESET,
                        VGA_BLANK,
                        VGA_SYNC,
                        VGA_CLOCK,
                        VGA_HS,
                        VGA_VS,
                        VGA_R,
                        VGA_G,
                        VGA_B,
                        TD_D,
                        TD_HS,
                        TD_VS,
                        SW,
                        TD_RESET
);
input               OSC_27;
input                   RESET;
input                   [2:0]SW;
output                  VGA_BLANK;
output                  VGA_SYNC;
output                  VGA_CLOCK;
output                  VGA_HS;
output                  VGA_VS;
output                  [9:0]VGA_R;
output                  [9:0]VGA_G;
output                  [9:0]VGA_B;
input                   [7:0]TD_D;
input                   TD_HS;
input                   TD_VS;
output                  TD_RESET;

wire [7:0]      Y;       //4:4:4 Y
wire [7:0]      Cb;      //4:4:4 Cb
wire [7:0]      Cr;      //4:4:4 Cr
wire mTD_HSx2;


decodificador_itu u1
(
        .CLOCK(OSC_27),         //relogio do sistema
        .TD_D(TD_D[7:0]),       //4:2:2 video data
        .TD_HS(TD_HS),          //Decoder_hs
        .TD_VS(TD_VS),          //Decoder_vs
        .Y(Y[7:0]),             //4:4:4 Y
        .Cb(Cb[7:0]),           //4:4:4 Cb
        .Cr(Cr[7:0]),           //4:4:4 Cr
    .HSx2(mTD_HSx2),
        .blank(VGA_BLANK)
);

reg     [10:0]L_COUNTER;
reg     [10:0]RL_COUNTER;
wire    sync_reset=(RL_COUNTER==9)?1:0;
reg     sync_en;
reg     [7:0]delay;
reg             [27:0]VGA_R1;
reg             [27:0]VGA_G1;
reg             [27:0]VGA_B1;
reg     [9:0] r_out, g_out, b_out;
reg     [11:0] r, g, b;
reg     [2:0] cont_quadros;
reg     blue;


assign  TD_RESET = 1'b1;
assign VGA_R = r_out;
assign VGA_G = g_out;
assign VGA_B = b_out;
reg     [31:0] linha, coluna, n_pixels, cex1, cey1;
reg     [31:0]  ce_x[7:0];
reg     [31:0]  ce_y[7:0];
```

```
reg             [9:0]   H_Cont;
reg             [9:0]   V_Cont;
reg                             oVGA_H_SYNC;
reg                             oVGA_V_SYNC;
reg                             Pre_HS;
reg                             Pre_VS;
reg                             mACT_HS;
reg                             mACT_VS;


always@(posedge OSC_27) begin

        // conversor YCbCr para RGB
        VGA_R1 = 551*Y + 756*Cr;
        VGA_G1 = 551*Y -186*Cb - 385*Cr;
        VGA_B1 = 551*Y +955*Cb;

        r <=( ( VGA_R1 - 105555 ) >>7 ) + 1;
        g <=( ( VGA_G1 + 64218  ) >>7 ) + 1;
        b <=( ( VGA_B1 - 131072 ) >>7 ) + 1;
        blue = SW[2]?(b>511):(b<511);
        // Desenho do centroide 10x10 pixels. Imagem Real ou Imagem com filtro para
cores verdes.
        if ((H_Cont > cex1 - 5) && (H_Cont < cex1 + 5) && ( V_Cont > cey1 - 5) &&
(V_Cont < cey1 + 5) && (SW[0] == 1)) begin
                r_out = 0;
                g_out = 0;
                b_out = 1023;
        end else begin
                if (SW[1] == 0) begin
                        if (r[11])
                                r_out <=0;
                        else if (r> 1023)
                                r_out <= 1023;
                        else
                                r_out <= r;
                        if (g[11])
                                g_out <=0;
                        else if (g > 1023)
                                g_out <= 1023;
                        else
                                g_out <= g;
                        if (b[11])
                                b_out <=0;
                        else if (b > 1023)
                                b_out<= 1023;
                        else
                                b_out <= b;
                end else begin

                        if ((r < 511) && (g > 511) && blue ) begin
                                if (r> 1023)
                                        r_out <= 1023;
                                else
                                        r_out <= r;
                                if (g > 1023)
                                        g_out <= 1023;
                                else
                                        g_out <= g;
                                if (b > 1023)
                                        b_out<= 1023;
                                else
                                        b_out <= b;
                        end else begin
                                r_out = 0;
                                g_out = 0;
                                b_out = 0;
                        end
                end
        end

        if ((r < 511) && (g > 511) && (blue) ) begin // busca por pixels validos.

                coluna = coluna + H_Cont;
```

```
                linha = linha + V_Cont;
                n_pixels = n_pixels + 1;
        end
        // calculo do centroide
        if (V_Cont === 517) begin
                cont_quadros = cont_quadros + 1;
                ce_x[cont_quadros] = (coluna / n_pixels);
                ce_y[cont_quadros] = (linha / n_pixels);
        end
        // calculo da media movel dos 8 ultimos centroides.
        if (cont_quadros === 7) begin
                cex1 = (ce_x[0] + ce_x[1] + ce_x[2] + ce_x[3] +ce_x[4] +ce_x[
ce_x[6] + ce_x[7] )>>3;
                cey1 = (ce_y[0] + ce_y[1] + ce_y[2] + ce_y[3] +ce_y[4] +ce_y[
ce_y[6] + ce_y[7] )>>3;
        end

        if (V_Cont > 517) begin
                coluna <= 0;
                linha <= 0;
                n_pixels <= 0;
        end
end

// Sinais de controle do Monitor RGB.
always@(posedge OSC_27 or negedge sync_en)
begin
        if(!sync_en)
        begin
                Pre_HS          <=      0;
                mACT_HS         <=      0;
                H_Cont          <=      0;
                oVGA_H_SYNC     <=      0;
        end
        else
        begin
                Pre_HS  <=      mTD_HSx2;
                if({Pre_HS,mTD_HSx2}==2'b10)
                mACT_HS <=      1;
                if(mACT_HS)
                begin
                        //      H_Sync Contador
                        if( H_Cont < 852 )
                        H_Cont  <=      H_Cont+1;
                        else
                        begin
                                H_Cont  <=      0;
                                mACT_HS <=      0;
                        end
                        //      H_Sync Gerador
                        if( H_Cont < 96 )
                        oVGA_H_SYNC     <=      0;
                        else
                        oVGA_H_SYNC     <=      1;
                end
                else
                begin
                        oVGA_H_SYNC     <=      0;
                        H_Cont          <=      0;

                end
        end
end

always@(posedge OSC_27 or negedge sync_en)//<<
begin
        if(!sync_en)//<<
        begin
                Pre_VS          <=      1;
                mACT_VS         <=      0;
                V_Cont          <=      0;
                oVGA_V_SYNC     <=      0;
        end
        else
```

```
        begin
                Pre_VS  <=      TD_VS;
                if({Pre_VS,TD_VS}==2'b01)
                mACT_VS <=      1;
                if( (H_Cont==1) && mACT_VS)
                begin
                        //      V_Sync Contador
                        if( V_Cont < 524 )
                        V_Cont  <=      V_Cont+1;
                        else begin
                                V_Cont  <=      0;

                        end
                        //      V_Sync Gerador
                        if(     V_Cont < 2 )
                        oVGA_V_SYNC     <=      0;
                        else
                        oVGA_V_SYNC     <=      1;
                end
        end
end

assign  VGA_HS          =       oVGA_H_SYNC;
assign  VGA_VS          =       oVGA_V_SYNC;
assign  VGA_SYNC        =       1'b0;
assign  VGA_CLOCK       =       OSC_27;


always @(posedge TD_HS) begin
if (TD_VS) L_COUNTER=0;
                        else L_COUNTER=L_COUNTER+1;
end

always @(posedge TD_VS) begin
                RL_COUNTER=L_COUNTER;//1714
end
always@(negedge sync_reset or posedge TD_VS) begin
if (!sync_reset)
        delay=0;
    else if (delay < 250)
         delay=delay+1;
end

always@(negedge sync_reset or negedge TD_VS) begin
if (!sync_reset)
                sync_en=0;
else if (delay < 100)
    sync_en=0;
    else
        sync_en=1;
end


endmodule
```

```
module decodificador_itu (
        input CLOCK,
        input [7:0]TD_D,
        input TD_HS,
        input TD_VS,
    input SW0,
    input SW1,
        output [7:0]Y,
        output [7:0]Cb,
        output [7:0]Cr,
    output reg HSx2,
        output VSx1,
        output reg Ypix_clock,
    output blank
);

        assign  VSx1=TD_VS;

/////// FF 00 00 SAV ou(EAV) //////
        reg[7:0]R1,R2,R3;
        reg[7:0]RR1,RR2,RR3;
        wire  Y_check=( (R3==8'hff) && (R2==8'h00) && (R1==8'h00) )?1:0;
        always @(posedge CLOCK) begin
           RR1=TD_D;
           RR2=R1;
           RR3=R2;
        end
        always @(negedge CLOCK) begin
            R1=RR1;
            R2=RR2;
            R3=RR3;
        end

/////SAV ou(EAV)?? ////
        reg START,Field;
        always @(posedge CLOCK) begin
        if (Y_check==1)
        begin
            START=~TD_D[4];
                Field= TD_D[6];
        end
        end

//////YUV4:2:2 para YUV4:4:4/////
        reg [7:0]YY;
        reg [7:0]CCb,Cbb;
        reg [7:0]CCr,Crr;
        reg [1:0]COUNTER;
        always @(posedge CLOCK) begin
                if (!START)
                   COUNTER=0;
                        else COUNTER=COUNTER+1;
    end
        always @(posedge CLOCK) begin
                 case (COUNTER)
                        0:begin
                                Cbb = TD_D;
                                Ypix_clock =0;
                        end
                        1:begin
                                YY  = TD_D;
                                CCr=Crr;
                                CCb=Cbb;
                                Ypix_clock =1;
                        end
                        2:begin
                                Crr = TD_D;
                                Ypix_clock =0;
                        end
                        3:begin
                                YY  = TD_D;
                                CCr=Crr;
                                CCb=Cbb;
                                Ypix_clock =1;
```

```
                        end
        endcase
    end

///// contador do tamanho de H  ////
        reg [10:0]H_COUNTER;
        reg [10:0]RH_COUNTER;
        always @(posedge CLOCK) begin
                if (TD_HS) H_COUNTER=0;
                        else H_COUNTER=H_COUNTER+1;
    end
        always @(posedge TD_HS) begin
                RH_COUNTER=H_COUNTER;
    end

/////gerador HSx2////
    always @(posedge CLOCK) begin
                if (
                        ((H_COUNTER >= 0) && (H_COUNTER < `sync)) ||
                    ((H_COUNTER >= RH_COUNTER[10:1]) && (H_COUNTER <
(RH_COUNTER[10:1]+`sync+1)))
                        )
                HSx2=0;
                else
                HSx2=1;
    end

/////H blank para HSx2/////
        reg [10:0]h;
        reg h_tr;
        reg h_tr_h;
        always @(posedge CLOCK) begin
    if(!HSx2) h=0;
        else
                h=h+1;
    end
        always @(posedge CLOCK) begin
     if ((h< 51) || (h > 771))
                h_tr=0;
                else
                h_tr=1;
        end
        always @(posedge CLOCK) begin
     if ((h< 41) || (h > 781))
                h_tr_h=0;
                else
                h_tr_h=1;
        end

/////V blank para HSx2//////
        reg[10:0]Vde_counter;
        always@(posedge  HSx2)begin
        if (TD_VS==0)
                Vde_counter=0;
        else
                Vde_counter=Vde_counter+1;
        end

///v-h blank output//
        wire vde=((Vde_counter > 31) && (Vde_counter < 511)) ? 1:0;//480
        wire blank_h  = h_tr   & vde;

///vga blank output//
        assign  blank  =   h_tr_h & vde;

/////dual port RAM////
        wire  [7:0]Yw;
        wire  [7:0]Cwr;
        wire  [7:0]Cwb;
        dul_port_c1024 YYYR(
                .iDATA(YY[7:0]),
                .iHSYNC(TD_HS),
                .iHSYNCx2(blank),
                .Y_CLOCK(Ypix_clock),
```

```
                .Y_CLOCKx2(CLOCK),
                .oDATA(Yw[7:0]),
                .field(Field),
                .VS(TD_VS)
        );
        dul_port_c1024 CBB(
                .iDATA(CCb[7:0]),
                .iHSYNC(TD_HS),
                .iHSYNCx2(blank),
                .Y_CLOCK(Ypix_clock),
                .Y_CLOCKx2(CLOCK),
                .oDATA(Cwb[7:0]),
                .field(Field),
                .VS(TD_VS)
        );
        dul_port_c1024 CRR(
                .iDATA(CCr[7:0]),
                .iHSYNC(TD_HS),
                .iHSYNCx2(blank),
                .Y_CLOCK(Ypix_clock),
                .Y_CLOCKx2(CLOCK),
                .oDATA(Cwr[7:0]),
                .field(Field),
                .VS(TD_VS)
        );

/////YUV 4:4:4 output////
        assign Y  = (blank_h)?Yw   :8'h10;
        assign Cr = (blank_h)?Cwr  :8'h80;
        assign Cb = (blank_h)?Cwb  :8'h80;

endmodule
```

```
module dul_port_c1024(
        iDATA,
        iHSYNC,
        iHSYNCx2,
        Y_CLOCK,
        Y_CLOCKx2,
        VS,
        oDATA,
        field

);

input  [7:0]iDATA;
input  iHSYNC;
input  iHSYNCx2;
input  Y_CLOCK;
input  Y_CLOCKx2;
input  field;
input  VS;
output [7:0]oDATA;

reg I;
always@(negedge iHSYNC)begin
        if (VS)
                I=field;
        else
                I=~I;
end

reg  [9:0]counter;
always@(posedge iHSYNC or posedge Y_CLOCK)begin
if (iHSYNC)
         counter=0;
else counter=counter+1;
end

reg  [9:0]counterx2;
always@(negedge iHSYNCx2 or posedge Y_CLOCKx2)begin
if (!iHSYNCx2)
         counterx2=0;
else counterx2=counterx2+1;
end

wire [7:0]DATA_a,DATA_b;
wire I_a= I;
wire I_b=~I;
wire [9:0]COUNTER_a=(I==1)?counter:counterx2;
wire [9:0]COUNTER_b=(I==0)?counter:counterx2;
wire CLOCK_a=(I==1)?~Y_CLOCK:~Y_CLOCKx2;
wire CLOCK_b=(I==0)?~Y_CLOCK:~Y_CLOCKx2;
wire [7:0]oDATA=(I==0)?DATA_a:DATA_b;


RAM2 u(
        .data_a(iDATA[7:0]),
        .wren_a(I_a),
        .address_a(COUNTER_a[9:0]),
        .clock_a(CLOCK_a),
        .q_a(DATA_a[7:0]),

        .data_b(iDATA[7:0]),
        .wren_b(I_b),
        .address_b(COUNTER_b[9:0]),
        .clock_b(CLOCK_b),
        .q_b(DATA_b[7:0])
        );

endmodule
```