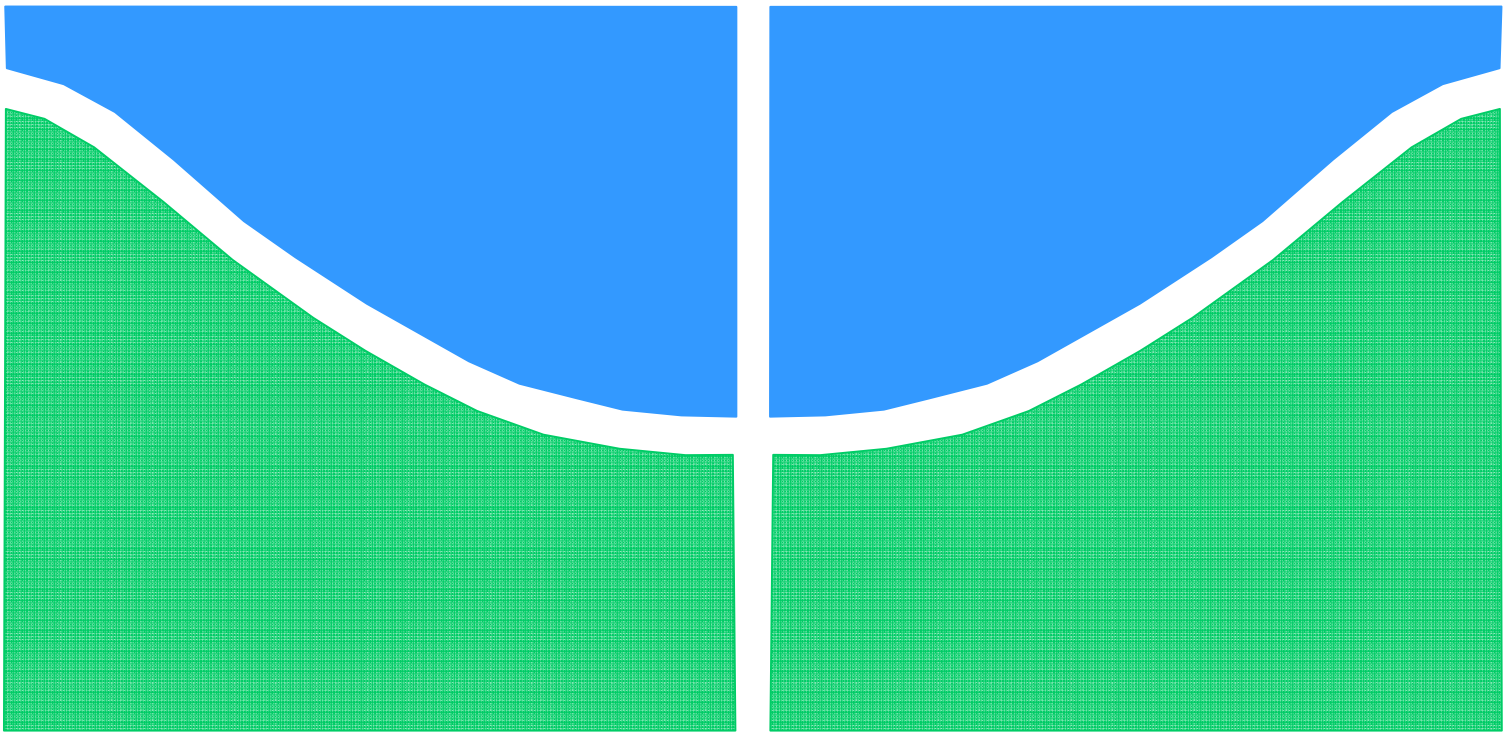

TRABALHO DE GRADUAÇÃO

CÂMERA DIGITAL CMOS PARA PROJETOS EMBARCADOS

Andrey Dutra Prado

Brasília, 4 de agosto de 2006

UNIVERSIDADE DE BRASILIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASILIA

Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

CÂMERA DIGITAL CMOS PARA PROJETOS EMBARCADOS

Andrey Dutra Prado

Relatório submetido como requisito parcial para obtenção
do grau de Engenheiro Eletricista

Banca Examinadora

Prof. Alexandre Ricardo Soares Romariz, UnB/ ENE
(Orientador) _____

Prof. Geovany Araújo Borges, UnB/ ENE
(Co-orientador) _____

Prof. Adson Ferreira da Rocha, UnB/ ENE _____

Dedicatória(s)

Dedico este trabalho a minha família e amigos que acreditaram que o mesmo poderia ser executado por mim, e esperaram todo esse tempo para ver atendidas suas expectativas.

Andrey Dutra Prado

Agradecimentos

Agradeço primeiramente a meus pais que me deram apoio durante todo o tempo em que estive comprometido com este trabalho.
Agradeço também as minhas irmãs que souberam compreender e apoiar todas as atitudes que tomei com o intuito de alcançar este objetivo.
Meus professores orientadores Alexandre Romariz e Geovany Borges, merecem um agradecimento especial pela paciência que tiveram ao longo do processo e pela oportunidade que me deram de apresentar os resultados que obtive.
Finalmente, agradeço aos meus amigos e meu cunhado pela valiosa ajuda e pela confiança que depositaram em mim.

Andrey Dutra Prado

RESUMO

Este manuscrito trata da construção de uma câmera CMOS com interface digital, propondo um sistema de aquisição e pré-processamento de imagem aplicado a projetos embarcados. Sugere-se, ainda, a implementação em hardware configurável, permitindo maior adequação das funções específicas às necessidades das aplicações.

ABSTRACT

This manuscript deals with the construction of a CMOS camera with digital interface, proposing an acquisition system and an image pre-processing system, applied to on board projects and implemented with configurable hardware, allowing better adaptation to specific application-dependent requirements.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Símbolos Latinos

T	Período	[s]
f	Frequência	[Hz]

Siglas

CMOS	Complementary Metal Oxide Semiconductor
CCD	Charged Couple Device
RAM	Random Acess Memory
FPGA	Field Programmable Gate Array
CPLD	Complex Programmable Logic Device
VHDL	Very High Speed Integrated Circuit Hardware Description Language
CIF	Common Intermediate Format
I2C	Inter-IC
PAL	Programmable Array Logic
SRAM	Static Random Acess Memory

1 INTRODUÇÃO

1.1 PROPOSTA DE TRABALHO.

Tendo em vista, fundamentalmente, projetos de pesquisa, neste trabalho é proposta a construção de uma câmera CMOS com interface digital, que deve permitir uma fácil inserção em projetos embarcados, mantendo programação aberta, qualidade e baixo custo.

A primeira característica marcante do projeto é apresentar uma câmera que possa facilmente ser acoplada a um barramento local como se fosse um periférico. A interface digital simples da câmera, constituída por 8 bits de dados bidirecionais, 2 bits de endereço para acessar os registros de controle da câmera, 1 bit de controle de leitura (RD), 1 bit de controle de escrita (WR), e 1 bit de controle de habilitação, além dos bits de alimentação 5 volts, deve permitir a programação do sensor da câmera assim como o controle da aquisição das imagens.

A aquisição da imagem será feita de maneira indireta, uma vez que o projeto da câmera prevê a utilização de uma memória RAM para armazenamento da última imagem adquirida. Sendo assim, a transferência da imagem deve ser feita depois de finalizado o ciclo de aquisição, que compreende a transferência da imagem do sensor para a RAM.

A vantagem vislumbrada com esse processo de aquisição indireta consiste principalmente em adaptar a velocidade de aquisição das imagens às possíveis limitações dos elementos responsáveis pelo atual processamento das mesmas. Será possível, então, instalar a câmera diretamente a barramentos microprocessadores de baixa velocidade, assim como utilizar placas de interface específicas, seriais ou paralelas, com *drivers* de alta velocidade.

A arquitetura proposta para a câmera digital CMOS pode ser vista na figura abaixo.

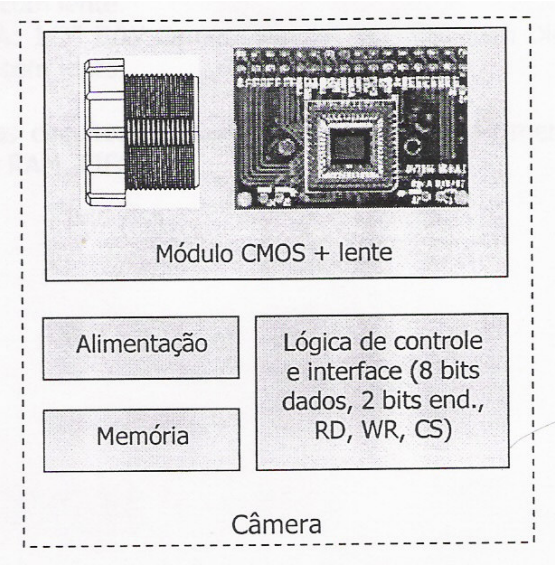

Figura 1.1. Arquitetura proposta para a câmera digital CMOS.

Será utilizado, como pode ser visto na figura 1.1, um módulo CMOS acompanhado da lente.

Finalmente, para implementação da câmera será necessário o projeto de um *hardware* para fazer a conexão entre os diversos dispositivos, como o sensor, a interface digital e a memória. Para tanto foi

proposta a utilização de *hardware* configurável, que significa o emprego de um FPGA (*Field Programmable Gate Array*), ou CPLD (*Complex Programmable Logic Device*).

A utilização do FPGA permite a definição por parte do usuário da funcionalidade do chip, ou, em outras palavras, do *hardware* por ele implementado. Dessa forma, o custo de fabricação e o tempo necessário para obtenção do produto e adaptação às necessidades de novas aplicações são mais baixos quando comparados aos de outras soluções.

A figura abaixo apresenta um diagrama com uma seqüência de funções que devem ser implementadas no *hardware* configurável para o funcionamento da câmera sugerida.

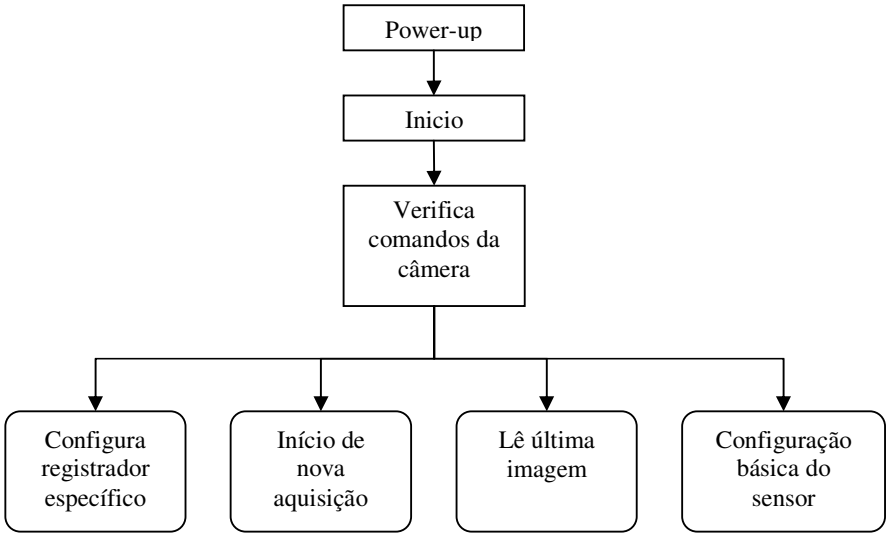

Figura 1.2. Fluxograma do comportamento do *hardware* da câmera.

Pela figura 1.2 é possível identificar que a primeira função do *hardware* é trazer tanto o sensor da câmera, quanto os elementos de controle da mesma para um estado conhecido após a energização do circuito.

Estando os elementos de controle da câmera em estados conhecidos, será feita uma leitura dos comandos enviados pelo processador da imagem. Estes comandos são enviados a partir do barramento de dados bidirecional, e dos dois bits de endereçamento citados na estrutura da interface da câmera.

De acordo com a entrada lida, o *hardware* deve ser capaz de implementar as seguintes funções detalhadas abaixo.

- Configuração básica do sensor. Existe uma configuração mínima necessária do sensor da câmera para que possam ser adquiridas imagens a partir da mesma. Sendo assim, essa configuração não obrigatoriamente deve ser solicitada pelo processador, porém deve ser feita ao menos uma vez depois da energização do circuito para que sinais de vídeo válidos possam ser disponibilizados.
- Início de nova aquisição. As aquisições de imagens serão demandadas pelo elemento processador da imagem. Portanto, o *hardware* deve estar preparado para atender as solicitações de aquisições de novas imagens quando for necessário.
- Leitura da última imagem. Uma das características da câmera, citada anteriormente, é o fato de que as imagens são adquiridas de forma indireta. Isso significa que primeiramente os *pixels* vindos do sensor são armazenados na memória e depois lidos a partir da mesma pelo processador da imagem. Sendo assim, uma das possibilidades vislumbradas é a leitura da última imagem adquirida, a qual se encontra armazenada na memória, ainda que novas aquisições não tenham sido solicitadas.
- Configuração de registrador específico. O sensor da câmera apresenta diversos registradores, a partir dos quais é possível configurar diferentes formas de aquisição de imagens. Sendo assim,

o processador, através do *hardware* configurável, deve ser capaz de acessar os registradores citados de forma a adequar a aquisição de imagens às suas necessidades da melhor maneira possível.

1.2 CONTEXTUALIZAÇÃO

O elemento básico que determina as características de uma câmera digital é o sensor, dispositivo eletrônico-digital responsável pela aquisição da imagem. Os sensores podem estar baseados em dois tipos de tecnologias: CCD (*Charged Couple Device*), e o CMOS (*Complementary Metal Oxide Semiconductor*).

As principais características dos sensores CCD são maior sensibilidade à luz, mais qualidade e também preço mais alto, enquanto os do tipo CMOS possuem menos sensibilidade, menor qualidade, porém são de mais fácil fabricação e apresentam menores preços.

Tradicionalmente, os sensores CMOS são aplicados em sistemas portáteis, onde não se requer alta definição, enquanto que os CCD são aplicados em produtos tais como filmadoras e câmeras digitais de alta qualidade.

O interesse na tecnologia CMOS vem crescendo gradativamente. A diferença de fabricação de CCDs, que deve ser feita em plantas especializadas a partir de matérias-primas não muito comuns, tem influenciado bastante nesta tendência, uma vez que os sensores CMOS, assim como microprocessadores e outros dispositivos da mesma tecnologia, podem ser fabricados em linhas de produção normais de semicondutores e utilizando matéria-prima barata e de uso generalizado.

Outra característica que vem chamando a atenção de fabricantes é a grande capacidade de integração de outros circuitos ao chip contendo o sensor da tecnologia CMOS. Isso permite grande flexibilidade para novos projetos, assim como contribui para que, na medida em que a tecnologia se adapta de maneira mais eficiente às câmeras digitais, haja uma baixa considerável nos preços, no consumo de energia, e no tamanho.

Atualmente, portanto, o nível oferecido pelas câmeras com CCD ainda é melhor que o das equipadas com CMOS. No entanto, a qualidade destas últimas vem aumentando progressivamente. É possível citar funções surpreendentes observadas atualmente nas webcams, que ainda assim são ofertadas por preços reduzidos. Sendo assim, é razoável dizer que a tecnologia CMOS apresenta as melhores perspectivas para o futuro.

Apesar das boas perspectivas com relação à disponibilidade e acesso aos sensores CMOS para o desenvolvimento de sistemas de visão, o que se observa no meio comercial é a manutenção de arquiteturas caras, complexas e protegidas por propriedade intelectual, o que caracteriza uma problemática em outros ambientes, como no acadêmico, onde são realizadas pesquisas, e se requer controle e dinamismo para possíveis alterações.

Para finalizar a contextualização a respeito do assunto tratado neste manuscrito, vale ressaltar que a importância dos sistemas de visão nos dias de hoje pode ser atribuída à existência de diversas aplicações no meio comercial, que atuam como facilitadores e otimizadores de processos produtivos das empresas.

Um bom exemplo a ser colocado é o sistema de visão industrial utilizado para inspeção em linhas de fabricação e embalagem. As imagens da linha de produção são processadas e fornecem informações para que o sistema tome decisões e controle processos de fabricação ou outros equipamentos, como robôs e mecanismos de separação de produtos com defeitos. Dessa forma, para o exemplo, nota-se que é possível realizar a inspeção de 100% da produção, preservando a atividade humana, exposta à fadiga e a maior probabilidade de erros neste tipo de operação.

1.3 APRESENTAÇÃO DO MANUSCRITO

Neste relatório, o Capitulo 2 é dedicado à apresentação dos componentes utilizados no projeto da câmera. É feita, inicialmente, uma introdução ao sensor CMOS aqui empregado, assim como uma descrição do protocolo serial I2C utilizado para configuração do mesmo. Em seguida, encontra se uma abordagem teórica sobre CPLDs e FPGAs, e uma apresentação do dispositivo utilizado neste projeto. Uma breve caracterização da linguagem VHDL é disponibilizada. Finalmente, é citada a memória RAM escolhida para este trabalho e suas principais características.

No capítulo 3 são mostradas as implementações práticas, como a placa de circuitos impressos que contem o módulo CMOS acompanhado da lente, a placa de circuitos impressos desenvolvida para o protótipo da câmera, e a idéia das máquinas de estado implementadas em VHDL e responsáveis pelo controle de aquisição de imagens, interface serial I2C com o sensor, e controle de gravação e leitura na memória.

O capítulo 4 é reservado para apresentação e discussão a cerca dos resultados obtidos, e, por fim, no capitulo 5 são apresentadas as conclusões gerais e recomendações para trabalhos futuros.

2 REVISAO BIBLIOGRÁFICA

2.1 SENSOR DE IMAGEM MONOCROMÁTICO CMOS VV5404

O sensor de imagem utilizado neste projeto é o VV5404 produzido pela VISION Limited. O padrão da imagem disponibilizada por este dispositivo é CIF (*Common Intermediate Format*). Este é um padrão de formato de vídeo para videoconferências. O formato CIF é parte da determinação ITU (*International Telecommunication Union*) H.261 a respeito de videoconferência. Outros padrões com resoluções acima ou abaixo do original foram estabelecidos. A tabela abaixo apresenta o padrão CIF, e outros derivados do original.

Tabela 2.1. Padrões de vídeo para videoconferência e suas principais características.

CIF Format	Resolution	Bit Rate at 30 fps (Mbps)
CIF (full)	352x288	36.5
QCIF (Quarter)	176x144	9.1
SQCIF(SubQuarter)	128x96	4.4

A figura abaixo apresenta um diagrama de blocos do sensor VV5404. Como é possível perceber, esse dispositivo incorpora um razoável número de controles embutidos, eliminando a necessidade de componentes adicionais para aquisição das imagens no formato digital.

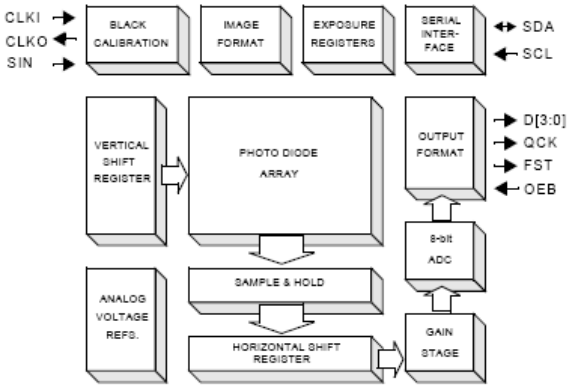

Figura 2.1. Diagrama de blocos do sensor de imagens VV5404 (*Mono and Colour Digital Vídeo CMOS Image Sensor VV5404 & VV6404, Datasheet, Figure 1).*

2.1.1 INTERFACE DO SENSOR VV5404

A interface digital do sensor compreende os componentes listados abaixo.

- Um barramento de dados de 4 bits para envio das informações de vídeo e controle.
- Um clock de qualificação de dados, QCK, o qual possui algumas formas de operação programáveis por uma interface serial.
- Um sinal de início de *frame*, FST.
- Um barramento de 2 bits de interface serial, SCL e SDA. Este barramento é utilizado para configurar e controlar o dispositivo, e utiliza o protocolo I2C.

- Um bit de sincronismo, SIN. Este bit de entrada é utilizado para o sincronismo na operação de várias câmeras.
- Um bit de controle da habilitação, OEB. Este sinal é responsável por habilitar o barramento de dados e os sinais de qualificação de dados QCK, FST.

Os *pixels*, compostos por 8 bits, são disponibilizados pelo barramento como pares de *nibbles*, sendo o mais significativo primeiro.

Juntamente com os bytes de dados, são enviados também códigos representando o início e o fim de quadros, assim como o início e fim de linhas. Esses códigos, ou seqüências de controle embutidas, podem ser utilizados para que o elemento responsável pelo processamento possa fazer o sincronismo da imagem recebida.

Existem três formas básicas de aquisição de imagens a partir do sensor VV5404:

- O processador da imagem fornece o clock do sensor, CLKI, e utiliza as seqüências de controle embutidas para sincronizar os quadros e linhas recebidas.
- Um cristal é utilizado para gerar o clock do sensor, e os dados recebidos são amostrados a partir de um dos modos de operação do sinal QCK. Neste caso, novamente as seqüências de controle embutidas são utilizadas para sincronizar os quadros e linhas recebidas.
- O processador da imagem utiliza os sinais FST, e QCK para sincronizar os dados de vídeo recebidos. Este modo de operação é destinado principalmente às aplicações que adquirem as imagens quadro a quadro.

O barramento de interface serial permite o controle do funcionamento e da configuração do sensor. Vale ressaltar que tanto o controle de exposição quanto o de ganho também são configurados via interface serial.

Um diagrama dos modos de interface do sensor com o processador pode ser visto na figura abaixo.

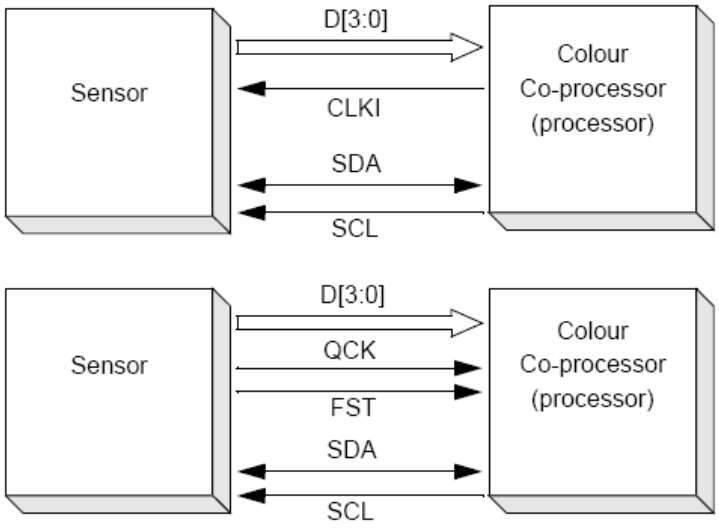

Figura 2.2. Opções de Interface com o sensor VV5404 (*Mono and Colour Digital Vídeo CMOS Image Sensor VV5404 & VV6404, Datasheet, Figure 2).*

2.1.2 CLOCK DE QUALIFICAÇÃO DE DADOS, QCK.

Como foi citado no subitem anterior, a aquisição de dados a partir do sensor depende do sinal QCK para ser implementada. Este sinal tem por função a qualificação do barramento de dados.

Existem duas freqüências de operação para o clock de qualificação de dados.

- A primeira, chamada de *fast* QCK, opera na freqüência de envio de *nibbles*. Um *nibble* é enviado para cada borda de descida deste sinal, independente se corresponde ao mais significativo ou menos significativo.
- *Slow* QCK, a segunda freqüência de operação, opera na taxa de recebimento de *pixels*. Na borda de subida do sinal é recebido o *nibble* mais significativo, enquanto que na borda de descida é recebido o menos significativo.

A tabela abaixo apresenta a freqüência de envio de *pixels* de acordo com o clock de entrada do sensor e sua divisão, a qual pode ser configurada a partir da interface serial.

Tabela 2.2. Divisor do clock para um cristal de 14,318 MHz (*Mono and Colour Digital Vídeo CMOS Image Sensor VV5404 & VV6404, Datasheet, Table 30).*

CLKI (MHz)	Clock Div Reg		Divisor	Pixel Frequency (MHz)	Frame rate (fps)		Comments
	bit 1	bit 0			30 fps	25 fps	
14.31818	0	0	2	7.15909			Not Valid
14.31818	0	1	4	3.57954	30.0	25.0	
14.31818	1	0	8	1.78977	15.0	12.5	
14.31818	1	1	16	0.89489	7.50	6.25	

A freqüência de *nibbles* é o dobro da freqüência de *pixels*. Isso se deve ao fato de um *pixel* ser composto por pares de *nibbles*.

Além das duas freqüências existem também diferentes modos de operação para o sinal QCK, os quais podem ser configurados a partir do barramento de interface serial. Estes possíveis modos de operação são apresentados na tabela abaixo.

Tabela 2.3. Modos de operação do QCK (*Mono and Colour Digital Vídeo CMOS Image Sensor VV5404 & VV6404, Datasheet, Table 19).*

QCK mode[1:0]		QCK state
0	0	Off
0	1	Free Running
1	0	Valid during data and control period of line *(see note)*
1	1	Valid only during data period of line

Na figura abaixo é apresentado o diagrama de tempo para cada um dos modos de operação do sinal QCK.

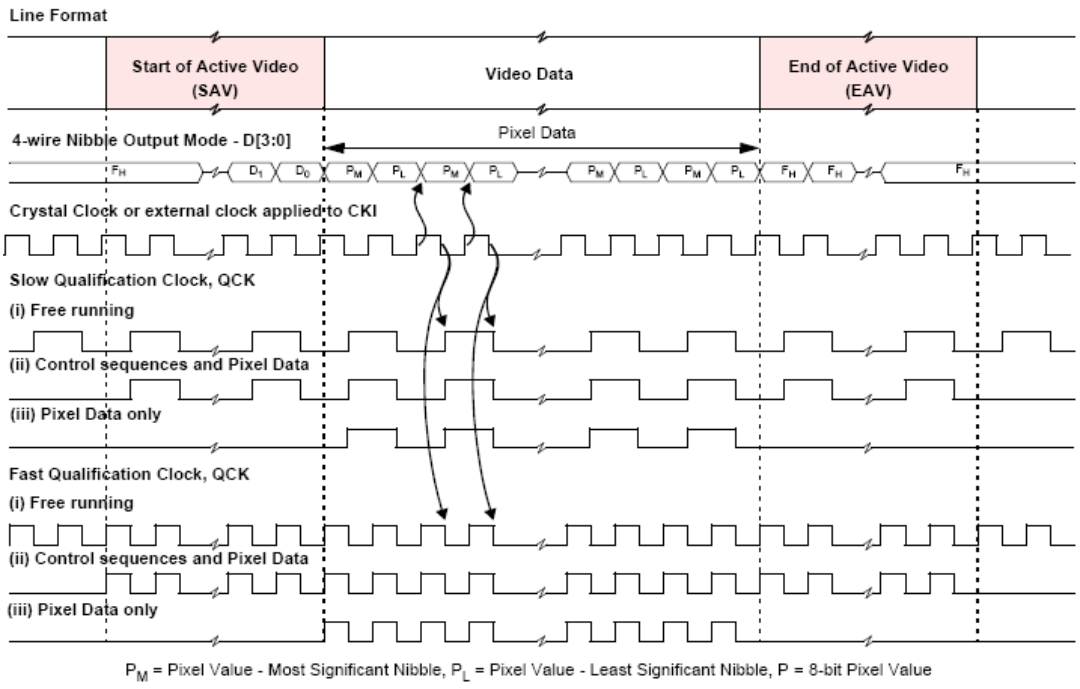

Figura 2.3. Diagrama de tempo para a qualificação dos dados de saída a partir de QCK (*Mono and Colour Digital Vídeo CMOS Image Sensor VV5404 & VV6404, Datasheet, Figure 13).*

2.1.3 SINAL DE INÍCIO DE *FRAME*, FST.

Assim como o QCK, o FST também é utilizado para qualificação dos dados recebidos pelo sensor. Existem três modos de operação possíveis para esse sinal, programáveis a partir do barramento de interface serial.

- O primeiro modo de operação é desabilitado. Quando este modo de operação é configurado o pino correspondente ao FST apresenta na sua saída nível lógico baixo.
- O segundo modo de operação é como sinal de início de quadro. Neste caso o sinal FST ocorre uma vez a cada quadro, e fica em nível lógico alto durante o período de uma linha, correspondente a linha de início de *frame*. Juntamente com as seqüências de controle embutido que sinalizam a ocorrência da mesma, essa linha apresenta os valores configurados em todos os registradores do sensor.
- O terceiro modo de operação é o desalocado. Para este modo de operação o pino correspondente ao FST disponibiliza para o sensor a outra freqüência não configurada para o sinal QCK, ou seja, se *slow* QCK estiver configurado no sensor, por exemplo, um sinal correspondente ao *fast* QCK será disponibilizado no pino destinado ao FST.

O diagrama de tempo para o modo de operação início de quadro é apresentado na figura abaixo.

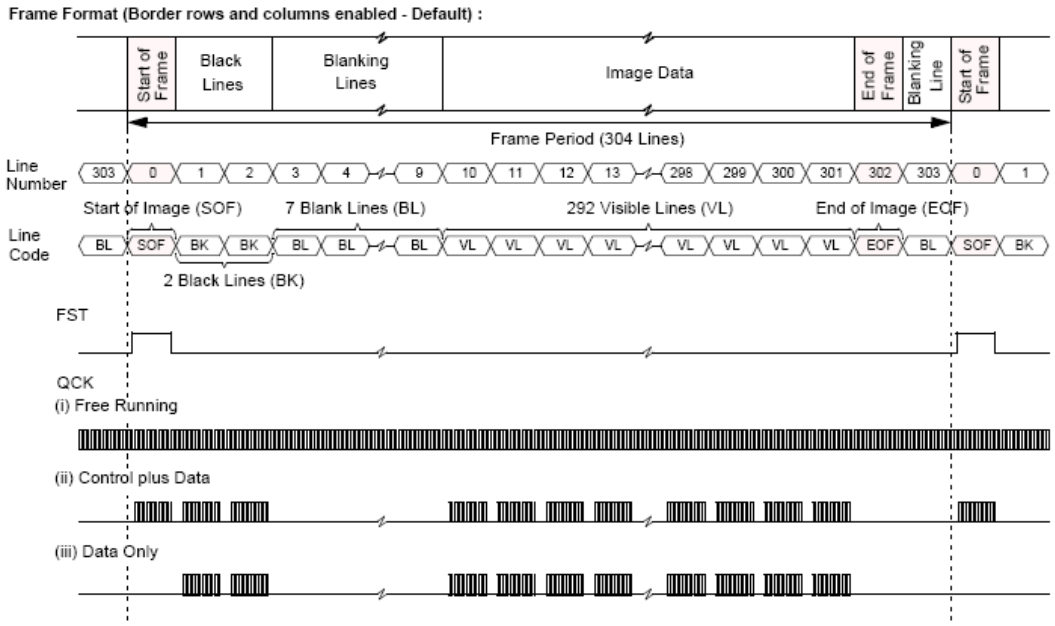

Figura 2.4. Diagrama de tempo para a qualificação de início de *frame* a partir de FST (*Mono and Colour Digital Vídeo CMOS Image Sensor VV5404 & VV6404, Datasheet, Figure 14)*.

2.1.4 BARRAMENTO DE CONTROLE SERIAL, I2C.

A escrita da informação de configuração para o sensor, e a leitura das informações de diagnóstico e configurações são feitas a partir de uma interface serial de 2 bits.

Essa interface serial de 2 bits implementa uma versão simplificada do protocolo I2C (*Inter-IC*). Este protocolo foi inicialmente apresentado pela Philips para dispositivos de produção em massa, como televisões e equipamentos de áudio, e era responsável pela comunicação entre circuitos integrados. Atualmente é utilizado também para realizar a comunicação entre dispositivo de controle inteligente, como microcontroladores, circuitos de propósito geral, como memórias, e aplicações orientadas a circuitos, como circuitos de processamento de sinal para sistemas de vídeo.

Para o caso específico do VV5404, as principais características da interface serial são apresentadas abaixo.

- Tamanho variável das mensagens de escrita e leitura;
- Informação com origem ou destino no sensor com endereçamento indexado;
- Atualização automática do índice após mensagem de escrita ou leitura;
- Mensagem de aborto com sinalização negativa a partir do processador da imagem;
- Mensagens orientadas a byte.

O processador de vídeo atua como um servidor da comunicação e a câmera como um cliente receptor e transmissor. A comunicação do servidor para a câmera utiliza a forma de 8 bits de dados com um clock de freqüência máxima de 100 kHz, sendo o clock gerado pelo próprio processador. Vale ressaltar que o clock determina a taxa de transferência de dados verificada. O protocolo de transferência de dados é mostrado, de forma abreviada, na figura abaixo.

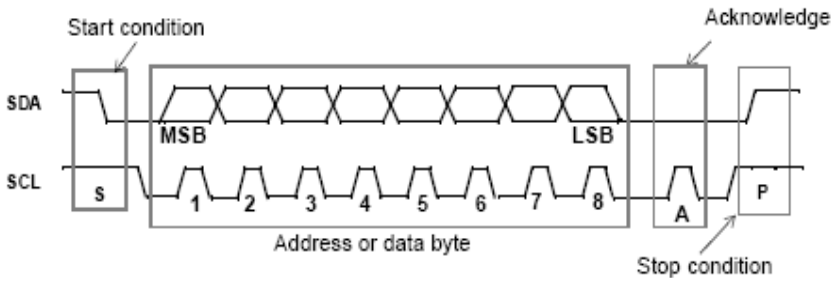

Figura 2.5. Protocolo de transferência de dados a partir da interface serial (*Mono and Colour Digital Vídeo CMOS Image Sensor VV5404 & VV6404, Datasheet, Figure 17).*

A informação é disponibilizada a partir de pacotes de 8 bits sempre seguidos de um bit de confirmação. As informações internas são produzidas pela amostragem do sinal SDA na borda de subida do sinal SCL. Os dados externos devem estar estáveis durante o período de nível alto do SCL. As exceções a esta regra são as condições de parada e início, onde o sinal SDA apresenta uma borda de subida e uma borda de descida respectivamente enquanto o sinal SCL se mantém em nível alto.

Toda comunicação da interface serial com o sensor deve começar com uma condição de início. Se a condição de inicio for seguida por um endereço válido, então o restante da comunicação pode continuar. O sensor dá o sinal de recebimento de um endereço válido ajustando o sinal SDA para um nível lógico baixo. O estado do bit de leitura ou escrita, o qual corresponde ao bit menos significativo do byte de endereço, é armazenado e o próximo byte de dados amostrado a partir do SDA pode ser interpretado.

Durante a seqüência de escrita, o segundo byte recebido é um índice utilizado para apontar para um registrador interno. O bit mais significativo deste byte é um sinal para auto-incremento do índice. Se esse sinal é habilitado, então a interface serial automaticamente incrementa o índice por uma unidade após cada sinal de confirmação do cliente. O servidor, então, pode enviar bytes de dados continuamente para o cliente até que o mesmo deixe de enviar o sinal de confirmação ou o servidor termine a comunicação de escrita com uma condição de parada ou uma condição de início repetida. Se a condição de auto-incremento do índice está habilitada, o servidor não precisa enviar os índices para acompanhar os bytes de dados enviados.

Assim que os dados são recebidos pelo cliente, são escritos bit por bit em um registrador serial/paralelo. Depois que cada byte de dados é recebido, um sinal de confirmação é gerado, e os dados são armazenados no registrador interno endereçado pelo índice corrente.

Durante uma mensagem de leitura, o índice corrente é lido pelo byte subseqüente ao byte de endereço do dispositivo. O próximo byte lido a partir do dispositivo do cliente é o conteúdo do registrador endereçado pelo índice corrente. O conteúdo do registrador é paralelamente carregado no registrador serial/paralelo e enviado de maneira síncrona com SCL.

Ao final de cada byte, tanto na escrita quanto na leitura, um sinal de conhecimento é esperado pelo dispositivo receptor. Apesar de o sensor ser sempre considerado um cliente, ele atua como transmissor quando o servidor solicita uma leitura do sensor.

No final de cada seqüência de leitura ou escrita, o valor final do índice no registrador será uma unidade maior que a última localidade lida ou escrita quando o bit de auto-incremento estiver habilitado. Uma leitura subseqüente utilizará este índice para retornar dados a partir dos registradores internos.

Uma mensagem somente pode ser terminada pelo barramento servidor enviando uma condição de parada, um início repetido, ou um sinal de confirmação negativo após a leitura completa do byte durante a operação de leitura.

A figura abaixo apresenta o formato dos dados enviados ou recebidos através da comunicação serial

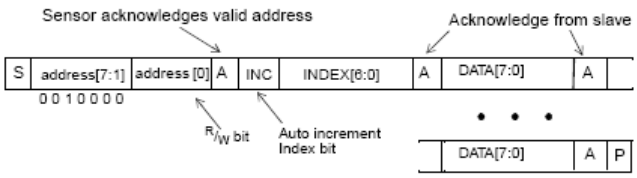

Figura 2.6. Formato dos dados para a comunicação serial (*Mono and Colour Digital Vídeo CMOS Image Sensor VV5404 & VV6404, Datasheet, Figure 18).*

2.2 FPGA E CPLD

O FPGA e o CPLD são dois tipos de *hardware* configurável. Nas próximas subseções será feita uma breve descrição de cada um deles, além da apresentação do dispositivo escolhido para ser utilizado no atual projeto.

2.2.1 CPLD

Os CPLDs (*Complex Programmable Logic Device*) são dispositivos lógicos programáveis que se encontram em um nível de complexidade entre os FPGAs e os PALs (*Programmable Array Logic*) apresentando características de ambos.

PALs também são dispositivos lógicos programáveis utilizados para implementar circuitos lógicos combinacionais baseados na idéia de que funções lógicas podem ser implementadas na forma de soma e produto booleana (portas OR e AND).

O CPLD de fato é constituído por múltiplos blocos, similares aos PALs, interconectados através de chaves programáveis, os quais, por sua vez, se encontram conectados a sub-circuitos chamados blocos I/O. Estes últimos são ligados aos pinos de entrada e saída do dispositivo. A figura abaixo apresenta uma arquitetura comum de um CPLD.

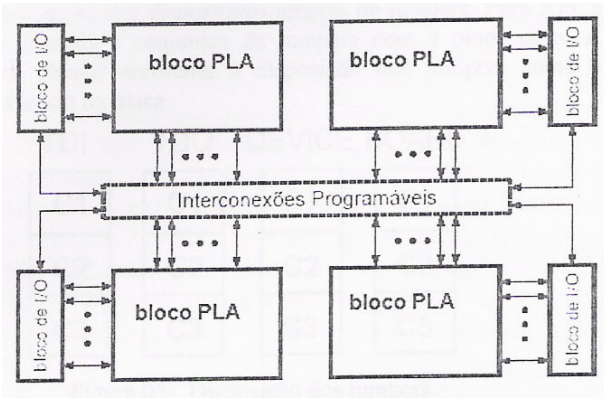

Figura 2.7. Arquitetura típica de um CPLD.

Uma das características marcantes deste tipo de dispositivo é a memória de configuração não volátil. Diferentemente de muitos FPGAs uma memória externa para armazenamento da configuração não é requerida e o dispositivo pode funcionar assim que energizado.

É possível destacar, também, o grande número de portas lógicas disponíveis nos CPLDs. Estes geralmente apresentam o equivalente a centenas ou dezenas de centenas de portas lógicas, permitindo a implementação de dispositivos com processamento de dados de moderada complexidade, e máquinas de estado complexas.

Em geral, CPLDs são escolhidos em vez de FPGAs quando lógicas de alta performance são desejadas. Devido a sua arquitetura interna menos flexível, os atrasos através do CPLD são de mais fácil previsão e tipicamente menores.

2.2.2 FPGA

Um FPGA (*Field Programmable Gate Array*) é um semicondutor que contem componentes lógicos programáveis e conexões também programáveis. As funções lógicas neste tipo de dispositivo não são implementadas de maneira tradicional. Neste, cada elemento lógico é na verdade uma pequena matriz de memória, localizada nos blocos lógicos do FPGA, que é programada diretamente com a função desejada, a partir da tabela verdade.

A estrutura do FPGA se apresenta com blocos I/O para conectar pinos de entrada e saída do dispositivo, blocos lógicos dispostos em um arranjo bidimensional e um conjunto de chaves de interconexões organizadas como canais de roteamento entre as linhas e colunas dos blocos lógicos. A figura abaixo apresenta a estrutura descrita.

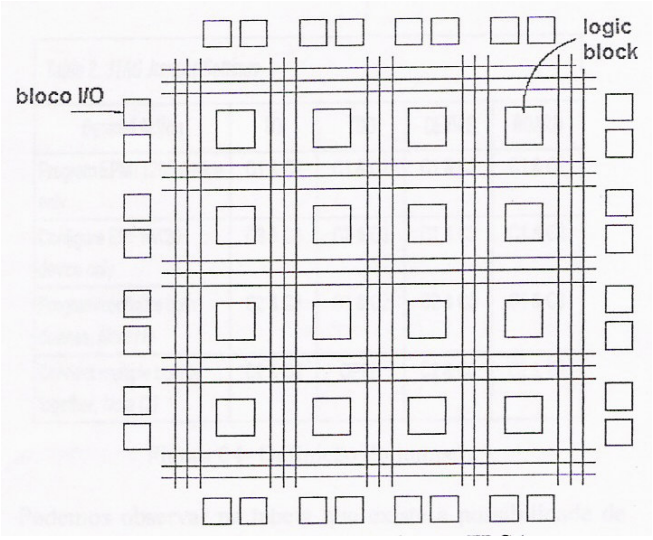

Figura 2.8. Arquitetura do FPGA (Brown & Rose, *FPGA and CPLD Architectures: A Tutorial,1996, Figure2).*

FPGAs podem ser utilizados para implementar diversos tipo de *hardware*. Uma utilização comum é para gerar protótipos que eventualmente serão produzidos em ASICs (*Application Specific Integrated Circuit*). De fato, FPGAs são, em geral, mais lentos que as suas respectivas replicas ASIC, não permitem projetos tão complexos, e consomem mais potência. No entanto, possuem enumeras vantagens como menor tempo de comercialização, possibilidade de reprogramação para o ajuste de falhas, e menores custos relacionados a erros de projeto.

Quando comparados aos CPLDs, os FPGAs apresentam se mais flexíveis com relação a sua arquitetura interna. Isso permite que sejam utilizados em aplicações com elevados números de registradores e aplicações com pilhas. Podem ainda ser utilizados em soluções de processador mais software, particularmente quando o processamento dos dados de entrada deve ser feito em altas freqüências. Finalmente, por possuírem maiores densidades de portas lógica e custarem menos que os CPLDs correspondentes, são, de fato, a melhor escolha para a implementação de projetos maiores.

Para implementar a interface entre os diversos componentes da câmera foi utilizado o chip EPM7128SL84-10 da Altera.

Os principais motivos que justificaram a utilização deste componente foram o fato de que, por ser um CPLD, não apresenta memória de configuração volátil, o número de pinos para I/O é suficiente ao desenvolvimento do projeto, e existe a possibilidade de utilização de um soquete adaptador de PLCC para DIP em uma placa de circuitos impressos, de forma que não há a necessidade de nenhum procedimento especial de soldagem.

O chip utilizado pertence à família MAX7000S de dispositivos lógicos programáveis produzidos pela Altera.

As principais características deste dispositivo são listadas abaixo.

- Possui o encapsulamento PLCC de 84 pinos;
- Disponibiliza ao usuário um número máximo de 64 pinos de I/O;
- Apresenta 2500 portas lógicas utilizáveis;
- Possui 128 macrocélulas disponíveis;
- Apresenta circuito de teste e configuração JTAG embutido.

A linguagem utilizada para programação no CPLD foi o VHDL (*Very High Speed Integrated Circuit Hardware Description Language*).

O VHDL é uma linguagem para descrição de sistemas eletrônicos digitais que apresenta as características listadas abaixo.

- Possibilita a descrição da estrutura do projeto, de forma que este último seja decomposto em sub-projetos, e que os sub-projetos sejam então interconectados;
- Permite a especificação da função implementada utilizando formas comuns de linguagens de programação;
- Torna possível a simulação do projeto antes de sua produção, de forma que os desenvolvedores podem rapidamente comparar alternativas e testes para correções sem o tempo e o custo geralmente verificados em implementações de *hardware*.

2.3 MEMÓRIA RAM

O chip de memória utilizado no projeto foi o BS62LV4001, o qual corresponde a uma memória SRAM (*Static Random Acess Memory*) CMOS com a capacidade de armazenamento de 524288 palavras de 8 bits, e que opera com a tensão de 5.5V. Apresenta encapsulamento PDIP de 32 pinos e velocidade de acesso de 70 ns.

A velocidade de acesso e capacidade da memória constituíram as principais preocupações na escolha do componente.

Com relação à capacidade de armazenamento, havia a necessidade de que a memória tivesse a capacidade de armazenar no mínimo 103952 palavras de 8 bits. Este valor corresponde ao número de *pixels* necessários para uma imagem com resolução 356x292.

Para o parâmetro do tempo de acesso, era desejado um componente que permitisse a gravação e a leitura de dados num período máximo de 280 ns. Este período corresponde ao envio de *pixels* numa freqüência de 3,57954 MHZ, freqüência máxima de *pixels* conforme a Tab. (2.2).

Considerando os dois parâmetros descritos acima, o encapsulamento e a disponibilidade no mercado, foi escolhida a memória em questão, a qual apresenta capacidade de armazenamento 4 vezes maior e tempo de acesso 4 vezes menor que os valores desejados.

A tabela abaixo apresenta os sinais da memória e suas respectivas funções.

Tabela 2.4. Sinais da memória e suas respectivas funções.

Sinais	Funções
A[0:18]	Barramento de endereçamento
DQ[0:7]	Barramento de dados
$\overline{WE}$	Controle de escrita (ativo em nível lógico baixo).
$\overline{CE}$	Controle de habilitação do chip (ativo em nível baixo).
$\overline{OE}$	Controle de habilitação das saídas (ativo em nível baixo).

Com relação ao funcionamento do dispositivo, é importante observar o comportamento durante os ciclos de escrita e leitura.

A figura abaixo apresenta o ciclo de leitura para o chip BS62LV4001.

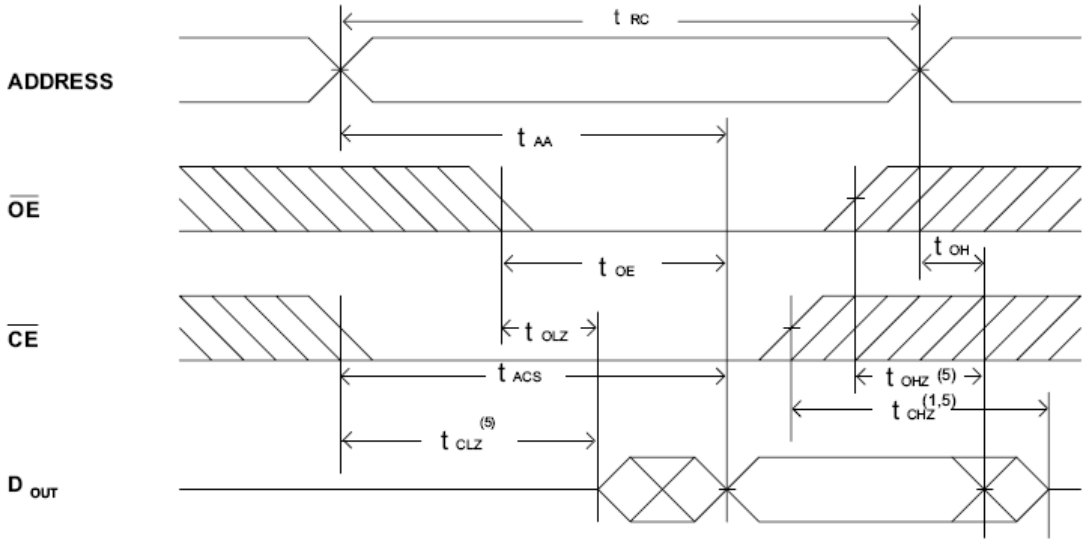

Figura 2.9. .Ciclo de Leitura (*Low Power/Voltage CMOS SRAM 512K x 8 bit BS62LV4001, Datasheet, Read Cycle 3).*

Observa-se pela figura que existe um tempo de espera necessário para que seja disponibilizado o byte válido no barramento de dados da memória após a colocação do endereço válido no barramento de endereços corresponde. Este valor apresentado abaixo.

$$t_{AA} = 70ns \tag{1}$$

É necessário observar esse tempo de acesso antes de submeter o barramento de endereços a outro valor sob pena de obter resultados aleatórios no barramento de dados.

A figura abaixo apresenta o ciclo de escrita para a memória escolhida.

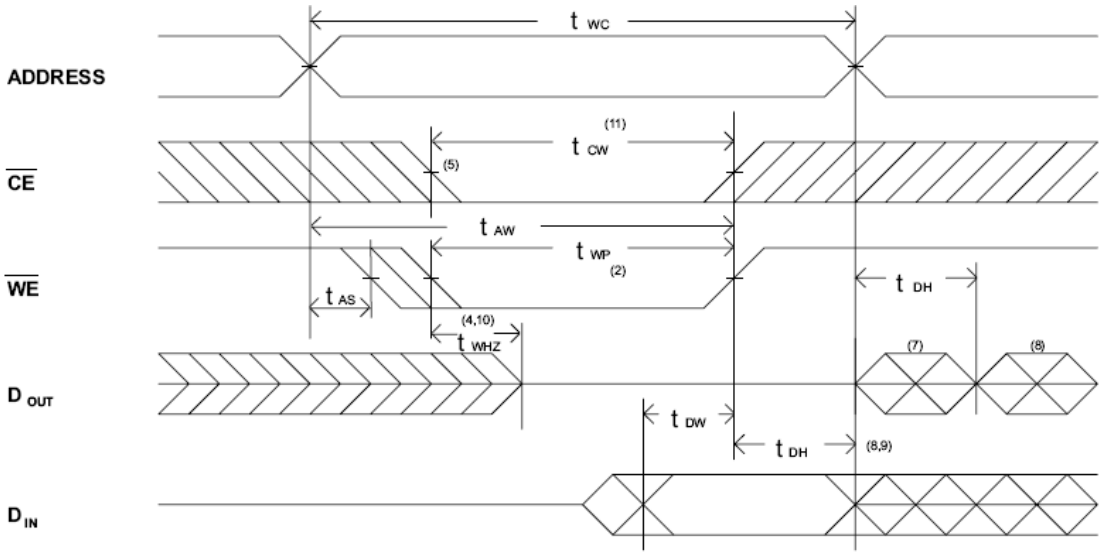

Figura 2.10. .Ciclo de Escrita (*Low Power/Voltage CMOS SRAM 512K x 8 bit BS62LV4001, Datasheet, Write Cycle 2).*

A característica mais importante do ciclo de escrita é que os valores apresentados ao barramento de endereçamento só podem ser alterados quando o sinal de controle de escrita, $\overline{WE}$, se encontrar em nível lógico alto.

3 DESENVOLVIMENTO

3.1 MÓDULO CMOS E LENTE

Como foi citado anteriormente, um dos requisitos do projeto é a utilização de um módulo contendo um sensor CMOS e uma lente. Sendo assim, foi utilizado o conjunto apresentado na figura abaixo.

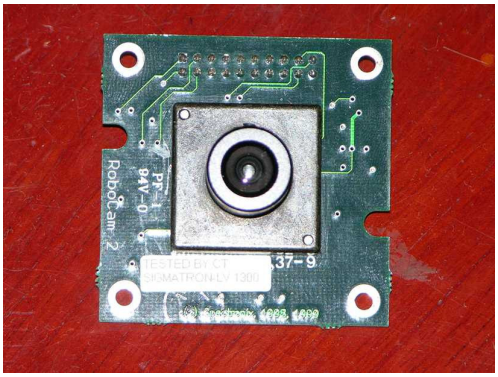

Figura 3.1. Módulo com sensor CMOS e lente.

A placa apresentada na figura é o módulo RoboCam II produzido pela Spectronix. Este módulo provê suporte ao funcionamento do sensor CMOS monocromático produzido pela VISION. O esquemático da placa é apresentado na figura abaixo.

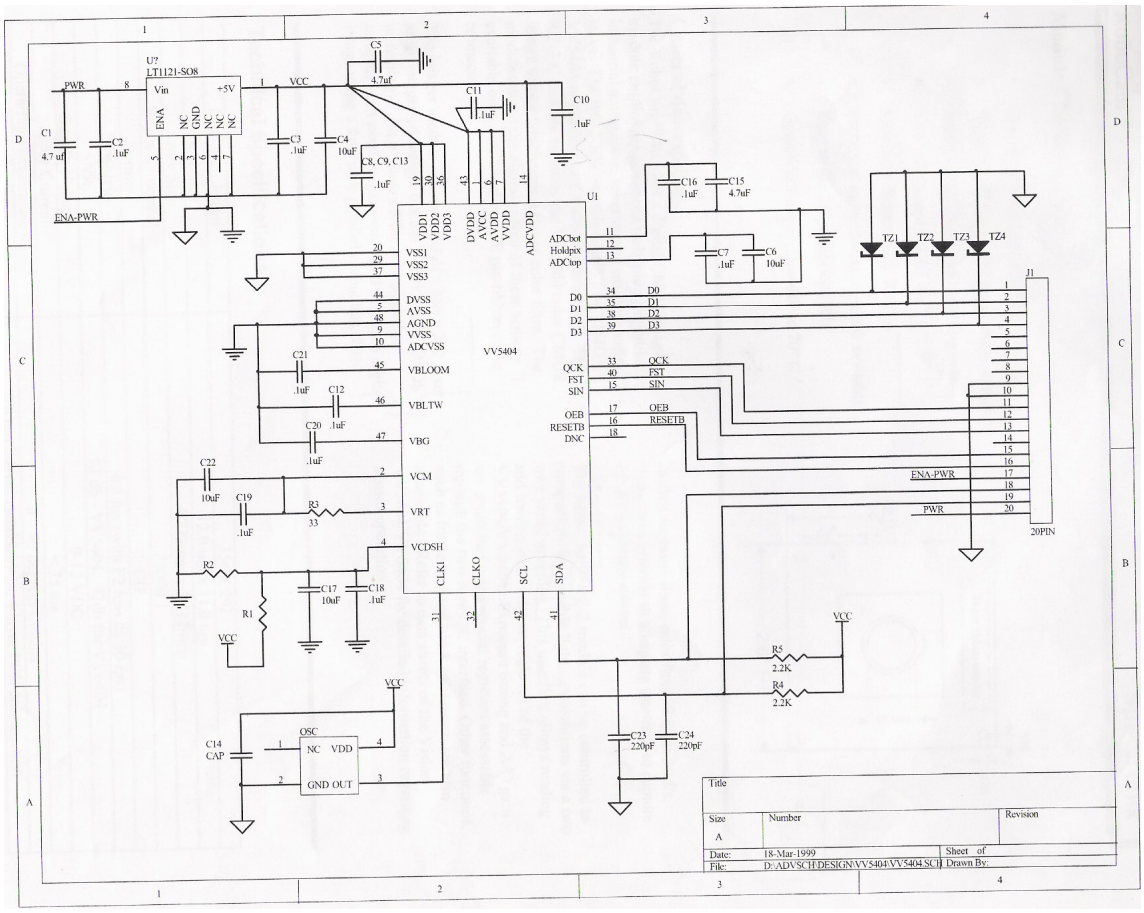

Figura 3.2. Diagrama esquemático da RoboCam II (Spectronix, *RoboCam Image Module RC-II, Datasheet, Schematics).*

Como é possível observar no esquemático, existe um regulador de tensão responsável pela alimentação de 5V da placa. O regulador é habilitado a partir de um sinal externo recebido através da interface de 20 pinos também apresentada na placa.

Verifica-se a presença de um oscilador responsável pelo clock de entrada do sensor. Este oscilador faz com que seja eliminada uma das formas de operação que corresponde aquela em que o processador fornece o clock para o sensor. Esse modo de operação corresponde ao primeiro apresentado na Fig (2.2).

A interface da placa de suporte ao funcionamento do sensor, apresenta, como já foi citado, uma interface de 20 pinos. Esta interface possui os sinais apresentados na tabela abaixo.

Tabela 3.1. Descrições dos sinais disponibilizados na placa de interface do sensor (Spectronix, *RoboCam Image Module RC-II, Datasheet, Table 1)*.

Pin	Name	Type	Description
1	DATA[0]	Output	Image and control data – 8 bit words
2	DATA[1]	Output	
3	DATA[2]	Output	
4	DATA[3]	Output	
5	N.C.		No Connect
6	N.C.		No Connect
7	N.C.		No Connect
8	N.C.		No Connect
9,10	GND	Signa Ground	Signal and Power ground
11	QCK	Output	Image data clock, used to clock data from DATA[0:3]
12	FST	Output	Frame start signal, goes high to indicate start of frame capture data
13	SIN	Input	Rising edge triggers a soft reset of the capture process
14	N.C.		No Connect
15	CE	Input	Chip enable, drive high to enable, normally active if not driven (pull-up)
16	RESETB	Input	System reset, Active low
17	PWR-ENA		Regulator power enable
18	SDA	Input/output	I2C data line used for two wire serial control
19	SCL	Input	I2C clock line used for two wire serial control
20	VDD	Power	9-12 VDC, noise de-coupled.

3.2 PLACA DE CIRCUITO IMPRESSO

Para fazer o apoio à utilização da placa RoboCam II, a memória SRAM, e o CPLD, além da criação da interface digital da câmera CMOS deste projeto, foi desenvolvida uma placa de circuito impresso. A figura abaixo apresenta a placa já montada com seus componentes.

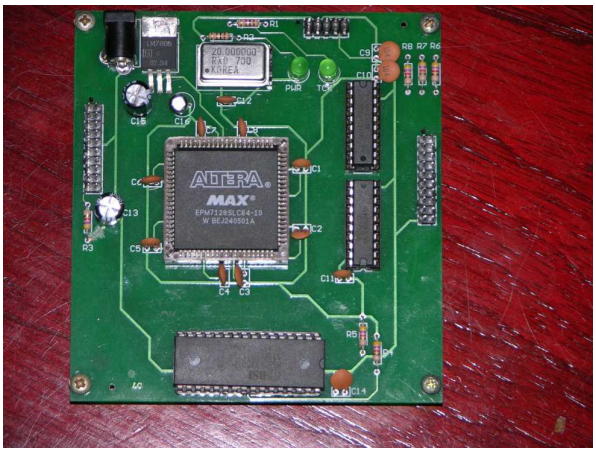

Figura 3.3. Placa de circuito impresso para implementação da câmera digital CMOS.

Como é possível perceber pela Fig. (3.2), a placa possui duas faces. Esta característica da placa foi fruto da dificuldade de roteamento dos diversos pinos do CPLD encontrada para a realização em uma única face.

A partir de um conector jack fêmea para alimentação de 9 a 12 V e um regulador, é feita a alimentação dos componentes da placa com 5V. A alimentação de 12 V também é disponibilizada para a interface com o sensor, juntamente com um capacitor de desacoplamento. Esse capacitor é umas das exigências para alimentação da placa do sensor, conforme pode ser visto na descrição do pino 20 da na Tab. (3.1). Um diodo emissor de luz, LED, indica a alimentação da placa e dos elementos do circuito.

Um oscilador também foi colocado na placa, o qual gera um clock de 20 MHz. Este é necessário uma vez que os circuitos, ou máquinas de estado, que farão o controle das funções da câmera precisam de um sinal de relógio. O oscilador presente na placa do sensor, conforme foi comentado no item anterior, tem como atribuição possibilitar o funcionamento das funções do sensor, como a disponibilização dos bytes de controle e vídeo, assim como os sinais de sincronismo QCK e FST.

O diagrama esquemático da placa é apresentado abaixo.

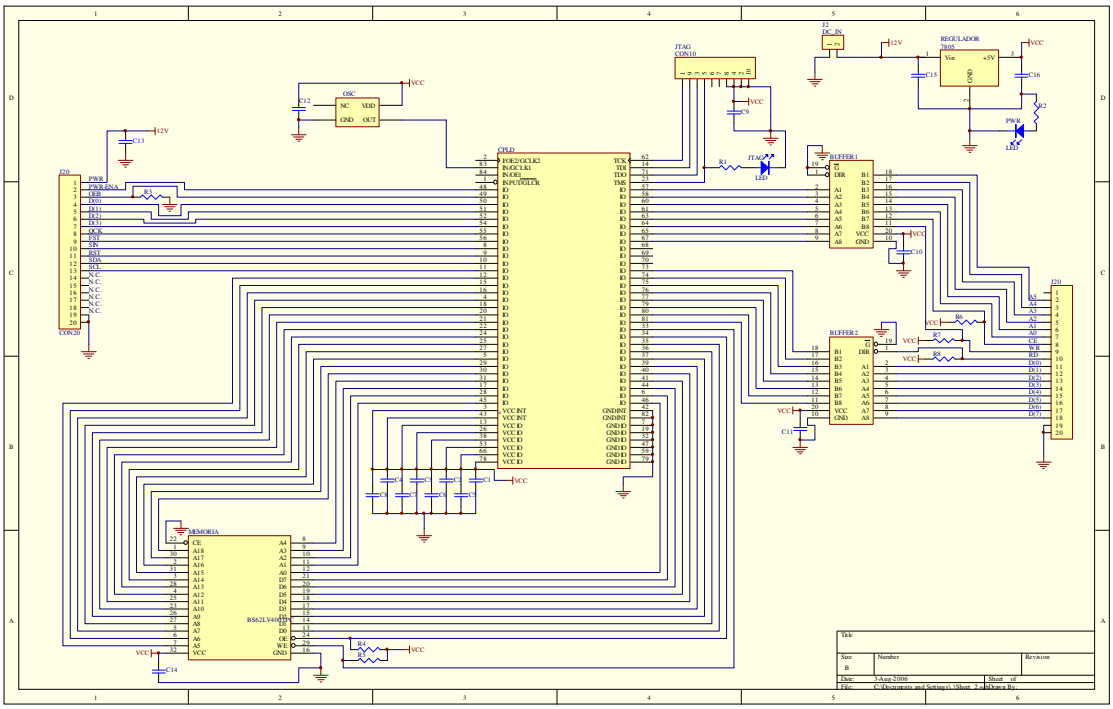

Figura 3.4. Esquemático da placa de circuito impresso.

Observa-se no esquemático da placa que, entre a interface com o processador e o CPLD, existem dois CI's 74AHC245. Estes componentes são responsáveis pelo isolamento entre o circuito do processador e da câmera, e para garantir o sentido correto da comunicação entre os mesmos.

O primeiro 74AHC245, chamado BUFFER1 na serigrafia da placa, está configurado para que apenas um sentido seja contemplado. Este sentido é do processador para o CPLD, onde são implementados os sinais de controle. Os sinais de controle são, diferentemente do planejado, 6 sinais de endereço A[0:5], 1 bit de controle de escrita, e 1 bit de habilitação. O aumento no número de bits de endereço se justifica pela disponibilidade de pinos do *buffer*. No caso de se utilizar 3 pinos para o endereçamento de controle, ainda sobrariam 3 pinos do *buffer*. Sendo assim, optou-se por utilizar toda a capacidade do componente.

O BUFFER2, nome destinado ao segundo 74AHC245, possui uma configuração tal que o sentido da comunicação é escolhido pelo processador. Os sinais bidirecionais de dados D[0:8], e o bit de controle de leitura RD são acessados a partir deste CI.

Os bits de controle de escrita, leitura e habilitação estão conectados a resistores de *pull-up*. Esta medida foi adotada para que ao energizar a placa, houvesse uma garantia de qual nível lógico será apresentado por cada um destes sinais de controle.

Resistores de *pull-up* também foram acoplados aos sinais de controle de escrita e habilitação da memória. Novamente visam a caracterização de um estado conhecido quando do momento inicial de funcionamento da câmera.

Nota-se, ainda no esquemático, a presença de uma interface de 10 bits. Esta interface, chamada de JTAG, permite a configuração do CPLD, e a realização de determinados testes de funcionamento. Existem pinos exclusivos para este tipo de interface no CPLD. Um LED foi colocado para sinalizar o momento da configuração do dispositivo.

Finalmente, capacitores de desacoplamento foram colocados entre os pinos de alimentação dos diversos dispositivos. Essa é uma garantia para o funcionamento adequado da fonte com relação ao nível de tensão fornecido.

3.3 IMPLEMENTAÇÃO DO MÓDULO DE GRAVAÇÃO E LEITURA DA MEMÓRIA

O primeiro módulo implementado em VHDL foi o de gravação e leitura na memória. Acreditava-se que tendo o controle da memória, seria mais fácil detectar se o processo de aquisição de imagens estaria sendo feito de maneira adequada.

Os sinais de entrada e saída do módulo de controle da câmera são apresentados na tabela abaixo.

Tabela 3.2. Sinais do módulo de controle da memória.

Sinais	Tipo
enable	entrada
write	entrada
clock	entrada
datain	entrada
dataout	saída
DQ	entrada e saída
$\overline{WE}$	saída
$\overline{CE}$	saída
$\overline{OE}$	saída

A ausência do barramento de endereçamento se justifica, uma vez que, o controle do endereçamento da memória é feito a partir do módulo que solicita a escrita e leitura na mesma, e controla os procedimentos de aquisição de imagens.

Conforme pode ser visto na Tab. (2.4), a memória possui 1 bit de controle de escrita $\overline{WE}$, 1 bit de habilitação do chip $\overline{CE}$, e um bit de habilitação das saídas $\overline{OE}$. Estes são resultantes do módulo de controle e são enviados para a memória, de forma que a mesma tenha o comportamento desejado. Na implementação deste módulo foram utilizados apenas os sinais $\overline{WE}$ e $\overline{CE}$, sendo o $\overline{OE}$ configurado de forma a permanecer ativo durante todo o funcionamento da câmera.

O barramento de dados DQ corresponde ao mesmo da memória. Assim sendo, justifica apresentar-se como de entrada e saída, uma vez que funcionará ou em um modo ou em outro, de acordo com o tipo de operação a ser executada.

O sinal de clock verificado na Tab. (3.2) é utilizado para a implementação do sincronismo do módulo de controle, lembrando que os tempos para realização dos ciclos de leitura e escrita devem ser respeitados. Sabendo que o clock gerado pelo oscilador da placa apresenta uma freqüência de 20 MHz, pode se calcular o período de um ciclo do clock, conforme abaixo.

$$T = \frac{1}{f} = \frac{1}{20 \times 10^6} = 50ns \tag{2}$$

O período de 50 ns não é suficiente para realização dos ciclos de escrita e leitura na memória. Portanto, o sinal de entrada do módulo de controle correspondente ao clock deve apresentar uma freqüência menor que a do sinal disponibilizado pelo oscilador da placa.

Para gerar tal sinal foi implementado um contador de sete bits sincronizado com a borda de subida do sinal de relógio da placa. Assim a freqüência resultante do bit menos significativo do contador pode ser calculada conforme demonstrado abaixo.

$$f_{contador} = \frac{20 \times 10^6}{2} = 10MHz \tag{3}$$

O período correspondente à freqüência calculada em (3) é, naturalmente, o dobro do valor encontrado em (2), conforme pode ser visto a seguir.

$$T_{contador} = \frac{1}{f_{contador}} = \frac{1}{10 \times 10^6} = 100ns \quad (4)$$

Com um período de 100 ns o tempo de acesso à memória está garantido. Os outros 6 bits do contador foram utilizados para fazer outras divisões do clock gerado pelo oscilador, quando necessário.

Os últimos sinais de controle verificados na entrada do módulo são o write e o enable. O fluxograma apresentado na figura abaixo descreve o controle dos ciclos de escrita e leitura a partir destes.

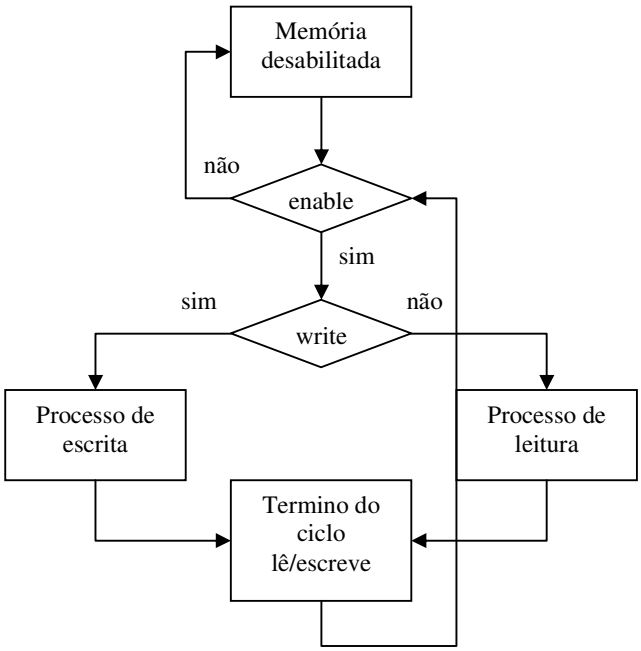

Figura 3.5. Fluxograma do módulo de controle da memória.

No fluxograma, o processo de escrita envolve todo o ciclo de escrita da memória. Neste processo são feitos procedimentos para garantir que o barramento de endereços é atualizado antes do sinal de controle de escrita.

Da mesma forma, o processo de leitura corresponde ao ciclo de leitura por completo. Os procedimentos realizados têm por objetivo garantir que o tempo de acesso aos dados seja respeitado e que os mesmos só sejam retornados pelo módulo de controle quanto estiver realmente concluída a tarefa.

3.4 IMPLEMENTAÇÃO DO CONTROLE DO BARRAMENTO SERIAL DO SENSOR

Tendo alcançado um módulo que faz o comando das funções de gravação e leitura na memória com sucesso, o próximo passo consiste em implementar a comunicação serial com o sensor CMOS. Esta etapa é extremamente importante, uma vez que é necessária uma pré-configuração do dispositivo para que *pixels* de vídeo válidos possam ser disponibilizados.

De acordo com o relatado no item 2.1.4, atualmente o protocolo I2C vem sendo utilizado em larga escala para os mais variados fins. Como conseqüência disso, melhorias neste método de comunicação foram feitas de forma que fosse possível utiliza-lo em um número ainda maior de aplicações. Apesar

disso, a interface serial do sensor VV5404 utiliza apenas uma versão simplificada do pacote de funções oferecidas pelo protocolo I2C de hoje.

Essa diferença entre as possibilidades verificadas e o que de fato é necessário para o projeto da câmera sugerido é relevante na medida que os recursos disponíveis são limitados. Houve uma tentativa de implementar o pacote integral do protocolo I2C disponibilizado em VHDL pelo fabricante LATTICE, porém sem sucesso. O problema encontrado foi que o número de macro-células necessárias ultrapassou o valor disponível no CPLD aqui empregado.

O mesmo problema de número de unidades lógicas programáveis foi encontrado para a maioria das implementações em VHDL do protocolo disponíveis para consulta. Contudo, uma dessas versões apresentava apenas os comandos mais básicos do I2C, e, portanto, foi a escolhida para ser explorada.

Os principais sinais utilizados pelo módulo de controle simplificado I2C são apresentados na tabela abaixo.

Tabela 3.3. Principais sinais do módulo de controle I2C simplificado.

Sinais	Tipo
clk	entrada
start	entrada
stop	entrada
write	entrada
read	entrada
Din	entrada
Dout	saída
SCL	entrada e saída
SDA	entrada e saída

O sinal clk corresponde ao clock de sincronismo da máquina de estado responsável pelo controle do módulo. Conforme exigência do próprio sensor, a freqüência do clock deve ser de no máximo 100 kHz. Para alcançar uma divisão do clock de 20 MHz da placa até o patamar desejado foi utilizado o bit mais significativo do contador de 7 bits, o mesmo que foi destinado a gerar o clock do comando da memória no item 3.3. A freqüência do sinal gerado pelo bit mais significativo do contador é calculada abaixo.

$$f_{contador} = \frac{20 \times 10^6}{2^8} = 78125Hz \tag{5}$$

Como é possível perceber, foi conseguida uma freqüência de 78,125 kHz a partir do bit mais significativo do contador. A taxa de transferência de bits depende diretamente do clock fornecido pelo processador no barramento de interface serial. Sabendo que o valor apresentado em (5) é aquele mais próximo do limite de 100 kHz que pode ser alcançado por uma divisão do clock da placa, pode-se considerar que a grandeza em (5) é bastante razoável, permitindo a atribuição deste sinal ao clk.

Os barramentos de dados Din e Dout correspondem aos utilizados para enviar dados ao sensor e receber do mesmo, respectivamente.

Os sinais SCL e SDA são o resultado da máquina de estados de controle da interface serial. A partir destes sinais, que são disponibilizados diretamente ao sensor, é possível configurar e operar o sensor VV5404.

Finalmente, start, stop, write, e read correspondem aos comandos principais do protocolo I2C. A partir desses sinais é efetuado o controle básico da interface serial. O fluxograma com a seqüência de controle implementada é apresentado abaixo.

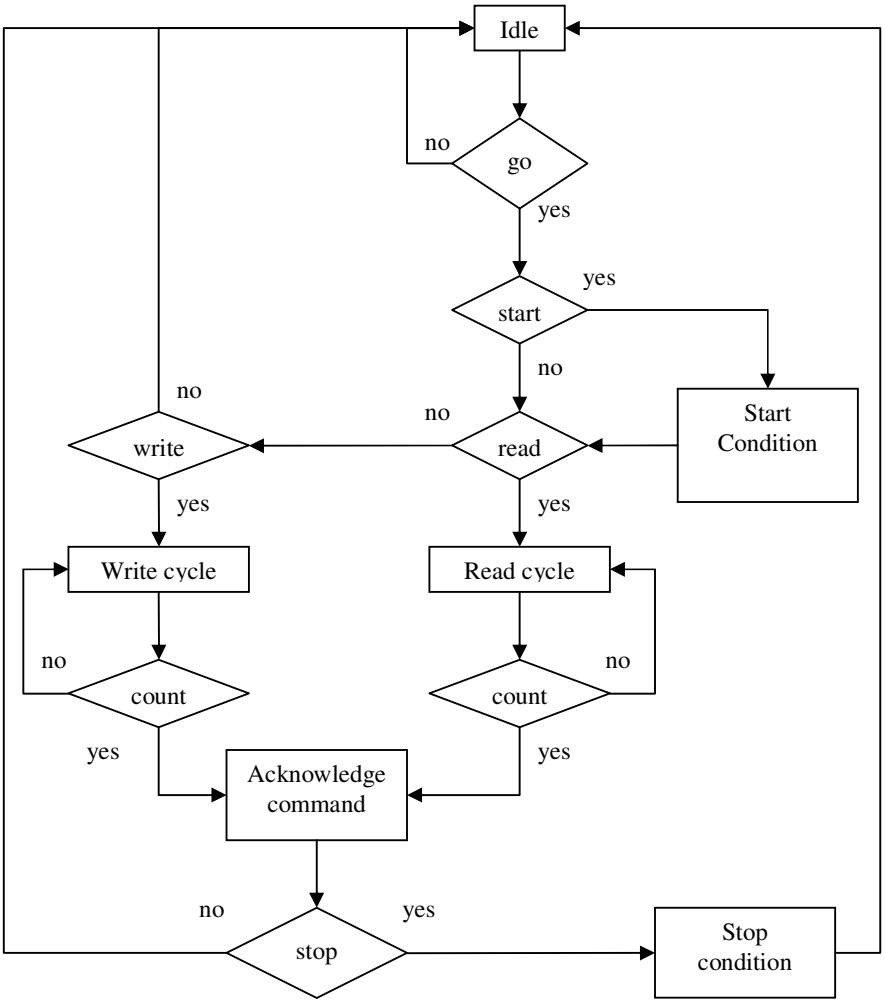

Figura 3.6. Fluxograma de controle da interface serial I2C.

Fazendo uma referência ao que foi citado no item 2.1.4, a comunicação serial do sensor orientada a byte necessita que os dados sejam amostrados a partir do sinal SDA na borda de subida do SCL. Sendo assim, os bytes devem ser enviados e recebidos de forma seqüencial. Para atingir uma situação em que fosse possível a amostragem serial dos bytes foi criado o count. Com este sinal é feito o controle do termino da amostragem dos dados enviados ou recebidos.

Por fim, vale ressaltar que a máquina de estados do I2C simplificado é capaz de controlar comandos simultâneos, de escrita e parada, por exemplo. Isso se deve ao fato de que possui em sua concepção o conhecimento da prioridade de algumas instruções sobre as outras.

3.5 IMPLEMENTAÇÃO DA MÁQUINA DE ESTADOS PRINCIPAL

O problema do número reduzido de macro-células não permitiu que todos os controles necessários fossem implementados. No diagrama da figura abaixo os campos em azul apresentam o que de fato a máquina de estados principal permite que o processador controle.

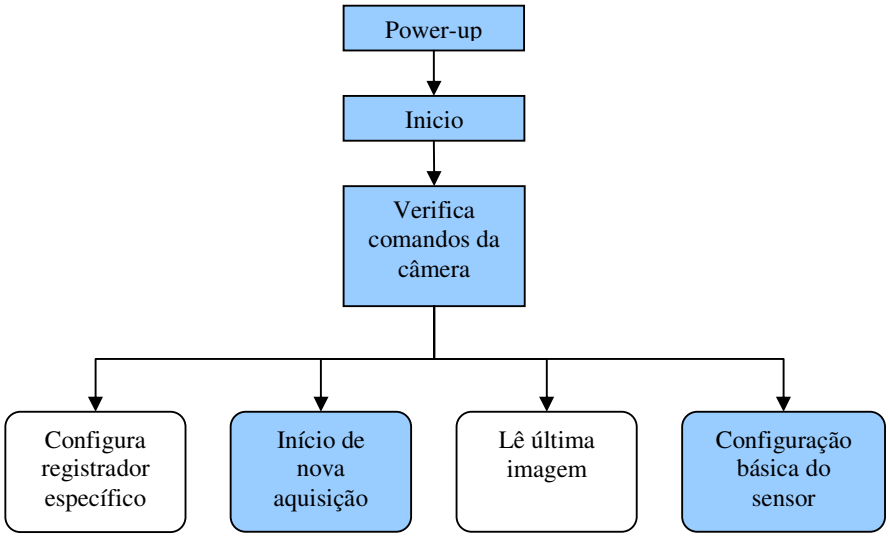

Figura 3.7. Controles implementados no controlador da câmera.

As funções de configuração de registradores específicos e da leitura da última imagem adquirida independente de nova aquisição não foram implementadas.

A figura abaixo apresenta um fluxograma com o controle principal da câmera implementado.

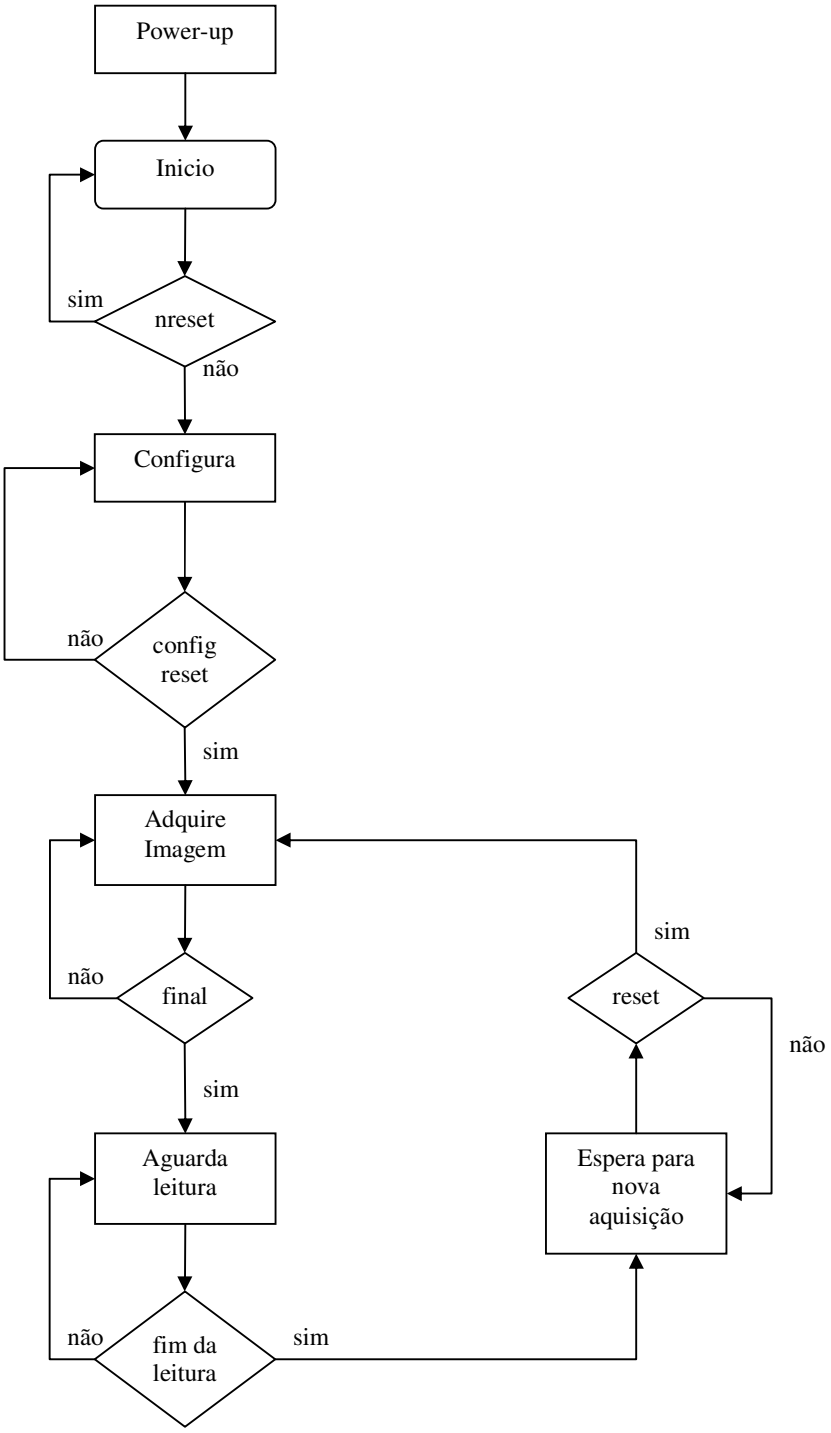

Figura 3.8. Máquina de estados principal da câmera.

Como é possível verificar no fluxograma, após a energização do circuito a máquina de estados vai para um estado inicial, onde fica enquanto o sinal nreset estiver ativo. Este é responsável pelo início da configuração do sensor. Assim que o *nreset* é desativado a configuração básica do sensor é realizada.

A configuração básica consiste na gravação em dois registradores do sensor. O primeiro registrador chamado de *setup* 4 é responsável pelo modo de operação dos sinais de qualificação de dados. Devido às características das placas de suporte ao funcionamento do sensor e da câmera os modos de operação apresentados na tabela abaixo foram escolhidos.

Tabela 3.4. Modo de operação do QCK e FST.

Objeto da configuração	Modo de operação
Freqüência de QCK	*slow* QCK
Modo de operação de QCK	qualificação de dados apenas
Modo de operação de FST	qualificação de início de frame*s*

De acordo com a Tab. (3.4), o sinal QCK deve qualificar em sua borda de subida o *nibble* mais significativo, e na borda de descida o *nibble* menos significativo, conforme a freqüência de operação.. Além disso, este sinal só estará presente quando os *nibbles* recebidos do sensor forem correspondentes aos *pixels* de vídeo, ignorando as seqüências de controle embutido.

O sinal FST terá a função de sinalizar, com sua borda de subida, o inicio de aquisição de um novo quadro. Apesar de permanecer em nível alto durante a linha de início de *frame*, como os dados disponibilizados durante esta linha não correspondem a dados de vídeo válidos, o sinal FST já estará em nível lógico baixo quando for detectada a primeira borda de subida do QCK.

Ainda na configuração básica do sensor é feito o acesso a outro registrador chamado *setup* 0. Toda vez que é energizado, o sensor é colocado em modo de operação de economia de energia. Sendo assim, para que possam ser disponibilizados *pixels* de vídeo ativo no barramento de dados, é necessário que o sensor seja retirado do modo de economia de energia.

Tendo sido realizada a configuração básica necessária, a câmera digital está pronta para iniciar a aquisição de um *frame* e se encontrará no estado final da configuração até que o sinal *reset* seja desabilitado.

Já no procedimento de aquisição de imagem, o que sinaliza o início do processo é a borda de subida do FST. A máquina de estados só interrompe a aquisição da imagem corrente quando for detectada nova borda de subida.

Com relação à aquisição dos *pixels*, o procedimento é regulado por QCK. Na borda de subida deste sinal o *nibble* recebido a partir do barramento de dados é armazenado nos 4 bits mais significativos de um registrador de 8 bits. Assim que uma borda de descida é detectada, o *nibble* recebido é armazenado nos 4 bits menos significativos do mesmo registrador, de forma que um *pixel* tenha sido adquirido. O barramento de endereçamento da memória é incrementado e um comando de escrita na memória é enviado ao módulo de controle desta, de forma que o *pixel* seja armazenado na posição de memória escolhida a partir dos bits de endereço.

O procedimento de aquisição da imagem, como foi dito, é realizado até que nova borda de subida do sinal FST seja detectada, momento em que o procedimento de leitura é iniciado.

O procedimento de leitura é regulado a partir de um *clock* enviado pelo elemento processador da imagem pelos bits de endereço disponíveis na interface da câmera. Na borda de subida do *clock* especificado é incrementado o barramento de endereçamento da memória e enviado um comando de leitura para o módulo controlador da mesma. O byte resultante da leitura é enviado ao processador através dos pinos de dados bidirecionais contidos na interface da câmera. O procedimento de leitura é finalizado no momento em que o barramento de endereçamento tiver o valor correspondente ao último endereço de memória utilizado para armazenamento dos *pixels*.

Terminado o procedimento de leitura a máquina de estados permanecerá em um estado aguardando a habilitação do sinal *reset* para que novo ciclo de aquisição de imagem seja iniciado.

O software utilizado para programação do CPLD é o Quartus II. Esta ferramenta, além de permitir a programação do dispositivo, faz diversas análises do código em VHDL após a compilação do mesmo. A figura abaixo apresenta a primeira página do relatório disponibilizado pelo Quartus II depois de compilado todo o código responsável pelo controle da câmera.

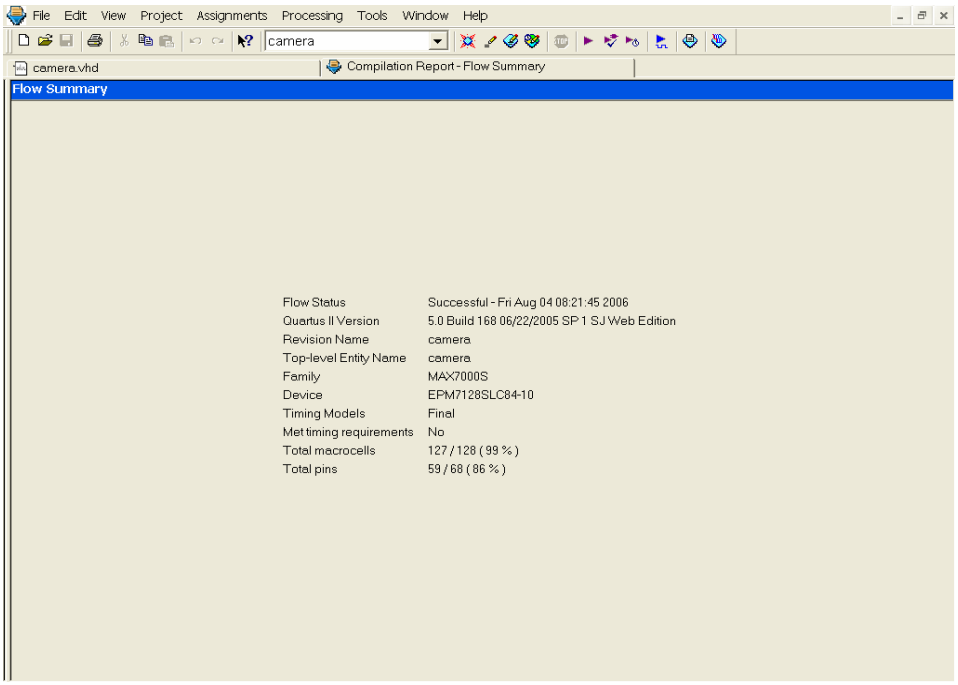

Figura 3.9. Relatório de compilação do código de controle da câmera.

Algumas informações importantes são disponibilizadas nesta primeira página. A informação de destaque no relatório da compilação deste código é o número de macro-células utilizadas, que corresponde a 99% do total. Esta é a principal justificativa para que nem todas as funções desejadas tenham sido implementadas.

4 RESULTADOS EXPERIMENTAIS

4.1 TESTE DA CONFIGURAÇÃO DO SENSOR DA CÂMERA

Alguns testes foram realizados para que fosse possível concluir que a configuração do sensor estava sendo feita de maneira satisfatória. A maioria deles se baseou na observação a partir de um osciloscópio do comportamento dos sinais de qualificação de dados.

O primeiro teste foi configurar o registrador *setup* 4 de forma que ambos os sinais de qualificação de dados estivessem desabilitados. As figuras abaixo apresentam os resultados verificados no osciloscópio.

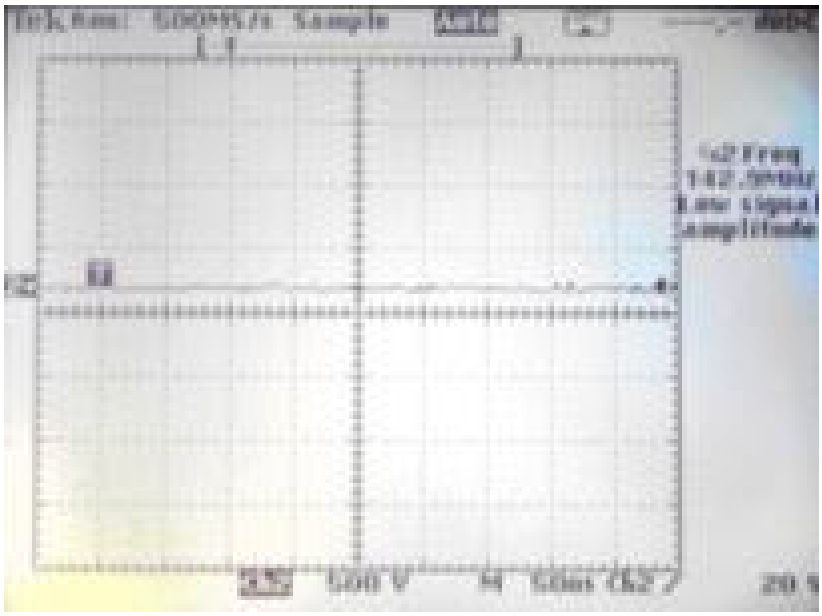

Figura 4.1. Forma de onda para QCK desabilitado.

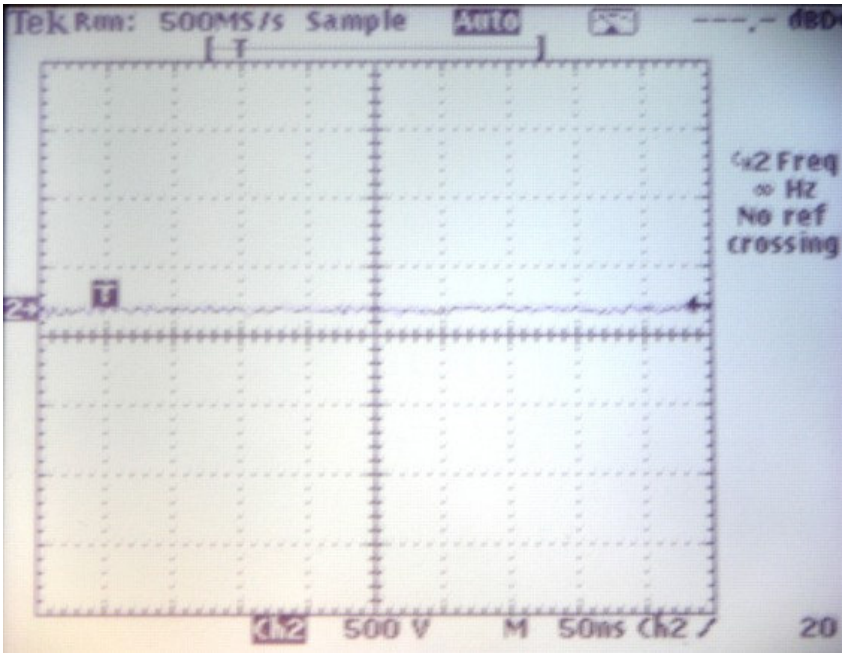

Figura 4.2. Forma de onda para FST desabilitado.

Como é possível observar nas figuras, não é detectado nenhum sinal válido no osciloscópio.

O segundo teste foi configurar o registrador *setup* 4 de forma que os sinais estivessem habilitados e que a freqüência de operação para o QCK fosse o *fast* QCK, e o modo de operação fosse *free running*, ou seja, qualifica todos os *nibbles* disponibilizados pelo sensor. As figuras abaixo apresentam os resultados verificados no osciloscópio.

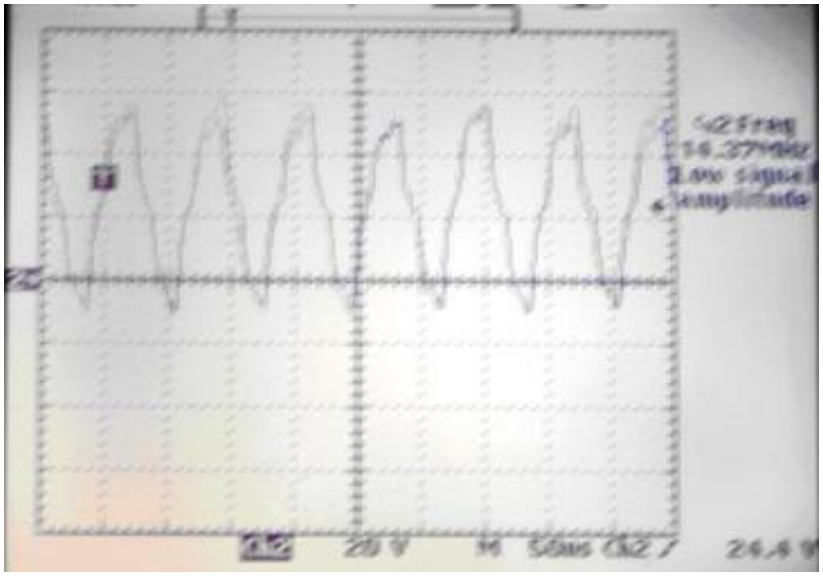

Figura 4.3. Forma de onda para QCK na freqüência *fast* e no modo de operação *free running*.

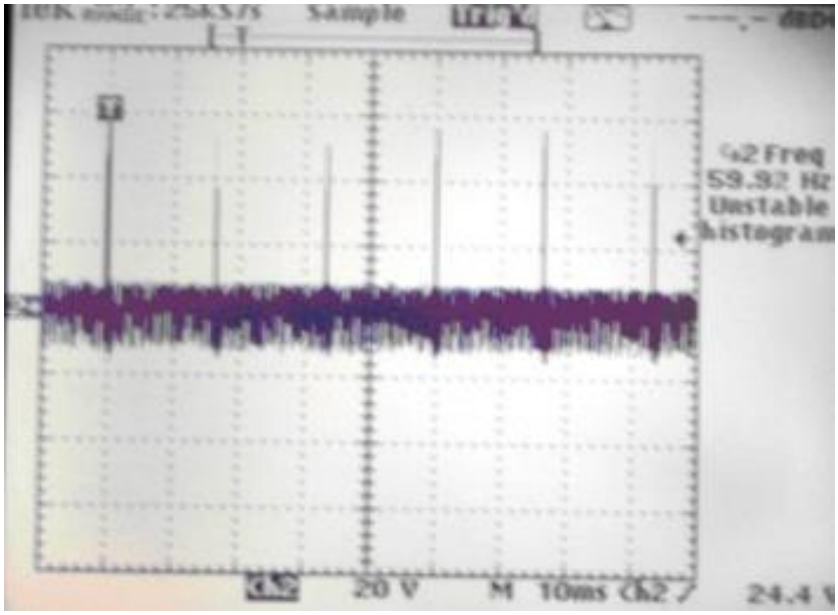

Figura 4.4. Forma de onda para FST habilitado.

Verifica-se pela Fig. (4.3) que para o QCK operando na maior freqüência, a taxa verificada é a de envio de *nibbles*, duas vezes maior que a freqüência de *pixels*.

Analisando a Fig. (4.4) observamos pulsos, que correspondem, provavelmente, ao início de *frames*. Não é possível medir com exatidão a freqüência dos pulsos pela presença de algum ruído no sinal.

O último teste foi realizar a configuração do registrador 4 de modo que o sinal QCK trabalhe na freqüência *slow* QCK, e o sinal disponibilizado no pino FST corresponda ao sinal QCK na freqüência *fast* QCK. Foi mantido o modo de operação do QCK com sendo *free running*. As figura abaixo apresentam os resultados verificados no osciloscópio.

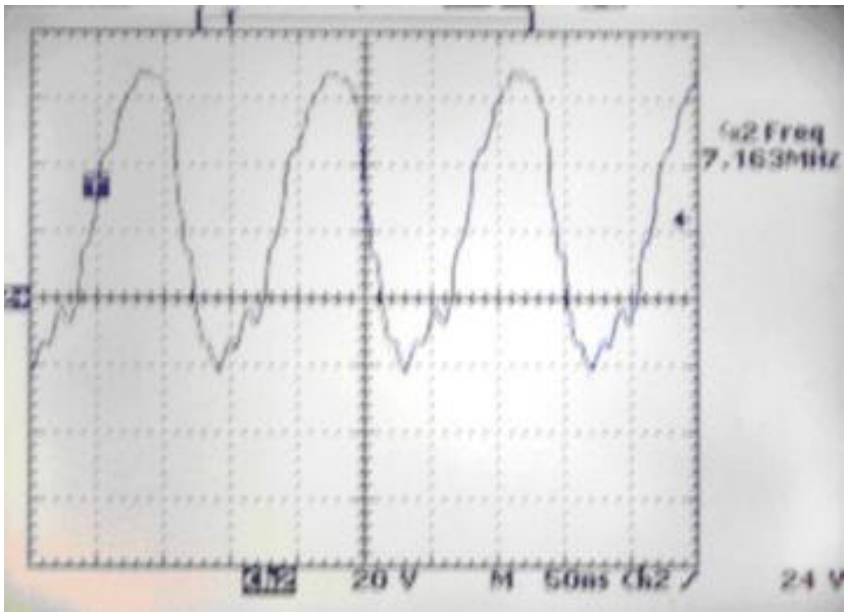

Figura 4.5. Forma de onda para QCK na freqüência *slow* QCK e modo de operação *free running*.

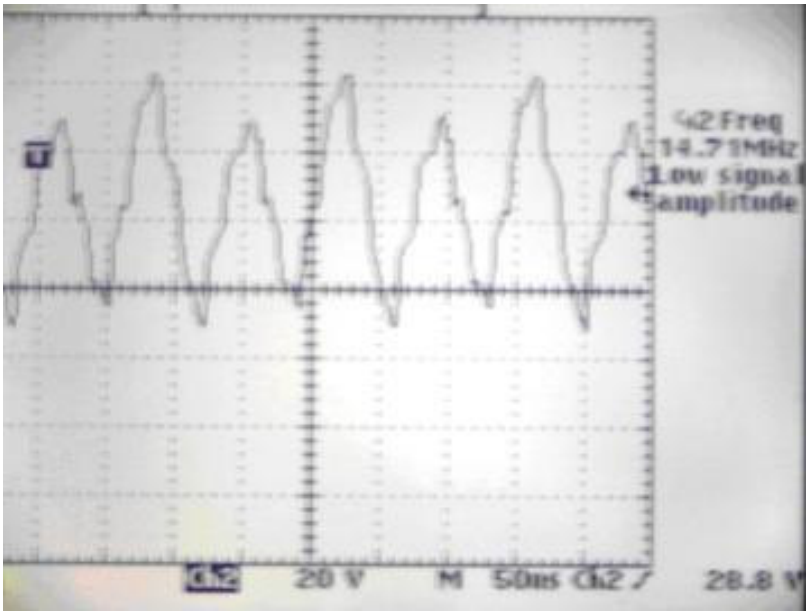

Figura 4.6. Forma de onda para FST operando como QCK na freqüência *fast* QCK.

É possível verificar pelas Fig. (4.5) e Fig. (4.6) que o sinal FST apresenta uma freqüência aproximadamente duas vezes maior que a freqüência de QCK, conforme era esperado.

4.2 VERIFICAÇÃO DA IMAGEM ADQUIRIDA

O teste de verificação da imagem adquirida dependia da implementação de uma interface entre a câmera CMOS implementada e um PC. Para tanto, planejava-se utilizar uma placa de desenvolvimento para microprocessadores do tipo PIC.

No entanto, não foi possível fazer essa interface, de maneira que também não foi possível determinar se a aquisição das imagens estava sendo feita de maneira satisfatória.

O teste que realizado foi a verificação do barramento de saída através de LEDs e do osciloscópio. No osciloscópio verificou-se para o barramento de saída, conforme disposto no manual do sensor, que nível lógico alto era disponibilizado durante períodos bem definidos. Isso corresponde aos momentos de calibração do nível de preto pelo sensor.

Verificou-se, a partir de LEDs conectados ao barramento de dados do sensor, que ao alterar a imagem submetida à lente do módulo a intensidade dos LEDs se alterava, o que sugere a dependência entre os sinais de saída e a imagem de excitação.

Apesar dos testes aqui descritos, não se pode considerar que a aquisição da imagem está sendo feita de maneira satisfatória, uma vez que este processo envolve gravação na memória, entre outros. O que se pode afirmar com certeza é que o sensor esta disponibilizando informações que correspondem a *pixels* de vídeo válidos.

5 CONCLUSÕES

Foi implementada neste projeto uma câmera CMOS com interface digital conforme desejado. No entanto, algumas das características da câmera sugerida não foram preservadas na final do projeto. Isso se deve, principalmente ao fato de que o dispositivo escolhido para funcionar como *hardware* configurável não apresentava capacidade suficiente à implementação de todas as funções vislumbradas.

Houve, a principio, uma falha na designação de um dos componentes do projeto. O critério de escolha, disponibilidade de pinos de I/O, deveria ter sido o número de unidades lógicas programáveis, ou macro-células. Sendo assim, nova pesquisa de mercado foi feita para avaliar a disponibilidade de outros dispositivos com maior capacidade de implementação lógica. Verificou-se que os preços sugeridos para o novo CPLD eram muito maiores que o do dispositivo utilizado, fugindo a premissa de construção de uma câmera digital de baixo custo.

Existem, então, algumas possibilidades para futuros projetos.

A primeira seria otimizar os módulos de controle apresentados de forma que a capacidade do dispositivo em questão seja utilizada de maneira mais racional.

Outra tentativa, seria a escolha de novo CPLD com capacidade de implementação lógica suficiente para as funções desejadas para a câmera. Neste caso, obviamente, seria necessário o desenvolvimento de nova placa de circuito impresso de suporte ao funcionamento da câmera, assim como maiores recursos financeiros, uma vez que o custo final do projeto aumentaria consideravelmente.

Por fim, é possível também modificar o projeto alterando algumas características como o controle embutido da interface serial, de forma que esta função seja realizada pelo processador da imagem e as outras funções continuem sendo controladas a partir do CPLD. Considera-se esta uma alternativa viável na medida em que, por estar bem difundido como protocolo simples de comunicação em CI's, o I2C já se apresenta implementado no *hardware* de muitos microcontroladores e outros dispositivos eletrônicos de controle e comunicação.

REFERÊNCIAS BIBLIOGRÁFICAS

Wakerly, John F., DIGITAL DESIGN: Principles & Practices, terceira edição. Prentice-Hall, 1994. Cap. 7.

Zeidman, Bob, INTRODUCTION TO CPLD AND FPGA DESIGN. Chalkboard Network, in www.chalknet.com

Brown, Stephen, Rose, Jonathan, FPGA AND CPLD ARCHITECTURES: A Tutorial. IEEE Design & Test of Computers, 1996.

Smith, J. Douglas, VHDL & VERILOG COMPARED & CONTRASTED Plus Modeled Example Written in VHDL, Verilog and C. VeriBest Incorporated, in www.ece.msstate.edu/~reese/EE8993/verilog_vhd

ALTERA, UNIVERSITY PROGRAM DESIGN LABORATORY PACKAGE, User Guide. ALTERA, 1997, in www.altera.com/education

Spectronix, ROBOCAM IMAGE MODULE RC-II. Datasheet.

LATTICE, DESIGNING AN I2C MASTER CONTROLLER. Reference Design RD 1005, 2005, in www.latticesemi.com

ALTERA, FLEX10K, Embedded Programmable Logic Device Family. Datasheet, 2003.

ALTERA, MAX7000S, Programmable Logic Device. Datasheet, 2002.

VISION, VV5404 & VV6404 Mono and Colour Digital Vídeo CMOS Image Sensors. Datasheet, 1999.

ALTERA, EPM7128E & EPM7128S Dedicated Pin-Outs, Datasheet.

Texas Instruments, SN54LS245, SN74LS245 Octal Bus Transceivers With 3-State Outputs. Datasheet, 2002.

BSI, BS62LV4001 Low Power/Voltage CMOS SRAM 512K x 8 bit. Datasheet, 2001.

VISION, VV5404 & VV6404 Mono and Colour Digital Vídeo CMOS Image Sensors. Datasheet, 1999, Figure 2,13,14,17,18, Table 10,19

BSI, BS62LV4001 Low Power/Voltage CMOS SRAM 512K x 8 bit. Datasheet, 2001, Read Cycle 3, Write Cycle 2.

Brown, Stephen, Rose, Jonathan, FPGA AND CPLD ARCHITECTURES: A Tutorial. IEEE Design & Test of Computers, 1996, Figure 2.

Spectronix, ROBOCAM IMAGE MODULE RC-II. Datasheet. Schematics, Table 1.

ANEXOS

CAMERA.VHD

```
--
-- VHD principal.
-- Implementa a configuracao basica do sensor via interface serial
-- Aquisicao da imagem
-- Gravacao na memoria
--
--
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

use work.i2c.all;
use work.memory.all;

entity camera is
	port (
		-- sinais de controle das maquinas de estado da camera
		clock : in std_logic;
		nReset : in std_logic;
		reset : in std_logic;

		-- sinais para implementacao da comunicacao serial I2C
		Dout : out std_logic_vector(7 downto 0);
		error : out std_logic; -- no correct ack received
		SCL : inout std_logic;
		SDA : inout std_logic;

		-- sinais para o sensor da camera
		fst		: in	std_logic;
		qck		: in	std_logic;
		DATA	: in	std_logic_vector(3 downto 0);
		sin		: out	std_logic;
		CE		: out	std_logic;
```

```
                RESETB      : out   std_logic;
                PWR_ENA     : out   std_logic;

                -- sinais da memoria
                WEn         : out   std_logic;
                CEn         : out   std_logic;
                OEn         : out   std_logic;
                address : buffer std_logic_vector(18 downto 0);
                DQ          : inout  std_logic_vector(7 downto 0);

                -- sinal para leitura da memória vindo do processador
                proc_clock: in  std_logic
                );
end entity camera;

architecture structural of camera is
        -- constantes para implementacao da comunicacao serial i2c
        constant SLAVE_ADDR : std_logic_vector(6 downto 0) := "0010000";
        constant CLK_CNT : unsigned(7 downto 0) := conv_unsigned(20, 8);
        signal cmd_ack : std_logic;
        signal D : std_logic_vector(7 downto 0);
        signal lack : std_logic;
        signal start, read, write, ack, stop : std_logic;
        signal i2c_dout : std_logic_vector(7 downto 0);
        signal clk              :       std_logic;

        -- sinal para implementacao da divisao do clock da maquina
        signal counter  :       std_logic_vector(7 downto 0);

        -- sinais para maquina de estados de aquisicao dos videos.
        signal grava    :       std_logic;
        signal le               :       std_logic;
        signal config   :       std_logic;
        signal byte             :       std_logic_vector(7 downto 0);

        -- sinais pra controle de gravacao e leitura na memoria
        signal write_mem:       std_logic;
        signal enable_mem:      std_logic;
        signal dataout  :       std_logic_vector(7 downto 0);
```

```
        signal final     :      std_logic;
        signal end_read  :      std_logic;


begin
        -- “conecta” ao controlador I2C
        u1: simple_i2c port map (
                                        clk     => clk,
                                        ena     => '1',
                                        clk_cnt => clk_cnt,
                                        nReset  => nReset,
                                        read    => read,
                                        write   => write,
                                        start   => start,
                                        stop    => stop,
                                        ack_in  => ack,
                                        cmd_ack => cmd_ack,
                                        Din     => D,
                                        Dout    => i2c_dout,
                                        ack_out => lack,
                                        SCL     => SCL,
                                        SDA     => SDA);

        -- “conecta” ao controlador da memoria
        u2: memoria port map (
                                        clock   => counter(0),
                                        write   => write_mem,
                                        enable  => enable_mem,

                                        datain  => byte,
                                        dataout => dataout,
                                        DQ              => DQ,

                                        WEn     => WEn,
                                        CEn     => CEn,
                                        OEn     => OEn
                                                );
```

```
-- atribui valores aos sinais de habilitacao da camera
CE              <=      '1';
RESETB          <=  '1';
PWR_ENA         <=      '1';
SIN             <=      '0';


-- processo para geracao do clock com frequencia menor (= 78,125 kHz)
divisor_clock : process(clock, counter)
        begin
                if (clock'event and clock = '1') then
                        counter <= counter + 1;
                end if;
                clk <= counter(7);
        end process;

-- bloco com maquina de estados para configuracao do sensor da camera
init_statemachine : block
        -- cria as variaveis do tipo states
        type states is (i1, i2, i3, i4, i5, i6, final);

        -- cria o sinal state correspondente ao estado corrente
        signal state : states;
begin
        -- processo responsavel pela atribuicao do proximo estado
        nxt_state_decoder: process(clk, nReset, state, start, read, write, ack, stop, D,
         cmd_ack, lack)
                -- criacao de variaveis intermediarias
                variable nxt_state : states;
                variable iD : std_logic_vector(7 downto 0);
                variable ierr : std_logic;
                variable istart, iread, iwrite, iack, istop : std_logic;
                variable istore_dout : std_logic;
                variable iconfig : std_logic;


        begin
                atribuicao dos valores as variaveis intermediarias e proximo estado
                nxt_state := state;
```

```
ierr := '0';
istore_dout := '0';

istart := start;
iread := read;
iwrite := write;
iack := ack;
istop := stop;
iD := D;
iconfig := '0';

-- logica de atribuicao do proximo estado
case (state) is

        when i1 =>      -- send start condition, sent slave address + write
                nxt_state := i2;
                istart := '1';
                iread := '0';
                iwrite := '1';
                iack := '0';
                istop := '0';
                iD := (slave_addr & '0'); -- write to slave (R/W = '0')
                iconfig := '0';


        when i2 =>      -- send register index (setup4)
                if (cmd_ack = '1') then
                        nxt_state := i3;
                        -- check aknowledge bit
                        if (lack = '1') then
                                ierr := '1'; -- no acknowledge received from
last command, expected ACK
                        end if;

                        istart := '0';
                        iread := '0';
                        iwrite := '1';
                        iack := '0';
                        istop := '0';
```

```
                    iD := "00010100";
                    iconfig :='0';
                end if;

            when i3 =>      -- send config register data, sent stop condition
                if (cmd_ack = '1') then
                    nxt_state := i4;
                    -- check aknowledge bit
                    if (lack = '1') then
                        ierr := '1'; -- no acknowledge received from
last command, expected ACK
                    end if;

                    istart := '0';
                    iread := '0';
                    iwrite := '1';
                    iack := '0';
                    istop := '1';
                    iD := "11000110"; -- modo de operacao fst e qck
                    iconfig := '0';
                end if;

            when i4 =>      -- send start condition, sent slave address + write
                if (cmd_ack = '1') then
                    nxt_state := i5;

                    istart := '1';
                    iread := '0';
                    iwrite := '1';
                    iack := '0';
                    istop := '0';
                    iD := (slave_addr & '0'); -- write to slave (R/W = '0')
                    iconfig := '0';
                end if;

            when i5 =>      -- send register index (setup0)
                if (cmd_ack = '1') then
                    nxt_state := i6;
                    -- check aknowledge bit
```

```
                    if (lack = '1') then
                        ierr := '1'; -- no acknowledge received from last command, expected ACK
                    end if;

                    istart := '0';
                    iread := '0';
                    iwrite := '1';
                    iack := '0';
                    istop := '0';
                    iD := "00010000";
                    iconfig := '0';
                end if;

            when i6 =>      -- send config register data, sent stop condition
                if (cmd_ack = '1') then
                    nxt_state := final;
                    -- check aknowledge bit
                    if (lack = '1') then
                        ierr := '1'; -- no acknowledge received from last command, expected ACK
                    end if;

                    istart := '0';
                    iread := '0';
                    iwrite := '1';
                    iack := '0';
                    istop := '1';
                    iD := "00001000"; -- retira do "Low Power Mode"
                    iconfig := '0';
                end if;

            when final => -- Fim da configuracao. Aguarda reset para reconfiguracao.
                if (nReset = '0') then
                    nxt_state := i1;
                end if;

                iconfig := '1';
```

```
                    when others =>
                        nxt_state := final;

                end case;

                -- logica de atribuicao de valores aos registradores a partir das variaveis
                -- intermediarias
                if (nReset = '0') then
                    state <= i1;
                    error <= '0';

                    start <= '0';
                    read <= '0';
                    write <= '0';
                    ack <= '0';
                    stop <= '0';
                    D <= (others => '0');
                    config <= '0';

                elsif (clk'event and clk = '1') then
                    state <= nxt_state;
                    error <= ierr;

                    start <= istart;
                    read <= iread;
                    write <= iwrite;
                    ack <= iack;
                    stop <= istop;
                    D <= iD;
                    config <= iconfig;
                end if;
            end process nxt_state_decoder;
        end block init_statemachine;


-- sinais para teste com osciloscopio
--          Dout(3 downto 0) <= DATA(3 downto 0);
--          Dout(6) <= fst;
--          Dout(7) <= qck;
```

```
--                  Dout(5) <= final;
--                  address(18 downto 17) <= "00";

-- bloco com maquina de estados para aquisicao do frame de video
maq_estados : block
-- criacao das variaveis do tipo estado
type estados is (inicio, wait_qck, stored_msn, stored_lsn, store_add, prep_read, wait_read, fim);

-- sinal estado correspondente ao estado corrente
signal estado : estados;

-- criação dos registradores para controle da maquina de estados
signal comeca : std_logic;
signal stop : std_logic;
signal zera_add: std_logic;
signal chadd : std_logic;
signal chadd1: std_logic;
signal passo : std_logic;


begin

        -- processo com a maquina de estados de aquisicao de video
        process (clock, estado, grava, zera_add, final, le, end_read, comeca, stop, passo, qck, reset,
DATA, byte)
                -- criacao de variaveis intermediarias
                variable prox_estado : estados;
                variable ibyte    : std_logic_vector(7 downto 0);
                variable igrava  : std_logic;
                variable ifinal    : std_logic;
                variable ile : std_logic;
                variable izera_add: std_logic;

        begin

                - atribuição de valores as variáveis intermediarias
                prox_estado := estado;
                ibyte := byte;
                igrava := grava;
```

```
ile := le;
ifinal := final;
izera_add:= zera_add;

-- lógica de atribuição do próximo estado
case estado is

when inicio => --a partir do estado inicial espera comando para incio da aquisicao
        if (comeca = '1') then
                prox_estado := wait_qck;
        end if;

        igrava := '0';
        ile := '0';
        ifinal := '0';
        izera_add:= '0';

when wait_qck => -- aguarda borda de subida do sinal qck para armazenar o nibble
                -- mais significativo do byte de dados vindo do sensor

        if (qck = '1') then
                ibyte(7 downto 4) := DATA(3 downto 0);
                prox_estado := stored_msn;
        end if;

        igrava := '0';
        ile := '0';
        ifinal := '0';
        izera_add:= '0';

when stored_msn => -- aguarda borda de descida do sinal qck para armazenar o nibble
                -- menos significativo do byte de dados vindo do sensor
        if (qck = '0') then
                ibyte(3 downto 0) := DATA(3 downto 0);
                prox_estado := stored_lsn;
        end if;

        igrava := '0';
        ile := '0';
```

```
        ifinal := '0';
        izera_add:= '0';

when stored_lsn => -- grava na memoria o byte de dados e aguarda sinal de fim de
                   -- aquisição para iniciar a leitura
        if (stop = '1') then
                prox_estado := prep_read;
        else
                prox_estado := wait_qck;
        end if;

        igrava := '1';
        ile := '0';
        ifinal := '0';
        izera_add:= '0';

when store_add => -- estado para armazenar ultimo endereco de memória utilizado
        prox_estado := prep_read;

        igrava := '0';
        ile := '0';
        ifinal := '1';
        izera_add:= '0';

when prep_read => -- estado para inicializar o endereco de memória utilizado
        prox_estado := wait_read;

        igrava := '0';
        ile := '0';
        ifinal := '1';
        izera_add:= '1';


when wait_read => -- estado onde a camera aguarda leitura dos dados a partir da
                  -- memória pelo processador da imagem
        if (end_read = '1') then
                prox_estado := fim;
        end if;
```

```
            igrava := '0';
            ile:= '1';
            ifinal := '1';
            izera_add:= '0';


when fim => -- estado onde a camera aguarda um sinal para inicio de nova aquisicao
            if (reset = '0') then
                        prox_estado := inicio;
            end if;

            ifinal := '1';
            igrava := '0';
            ile := '0';
            izera_add:= '0';

end case;

-- logica de geracao dos registradores a partir das variáveis intermediarias
if (reset = '0') then
            grava <= '0';
            estado <= inicio;
            byte <= "11111111";
            write_mem <= '1';
            enable_mem <= '1';
            final <= '0';
            zera_add <= '1';
elsif (clock'event and clock = '1') then
            estado <= prox_estado;
            byte <= ibyte;
            grava <= igrava;
            le <= ile;
            final <= ifinal;
            zera_add <= izera_add;
            if (grava='1') then
                        write_mem <= '0';
                        enable_mem <= '0';
            elsif (le='1') then
                        write_mem <= '1';
```

```
                enable_mem <= '0';
            else
                enable_mem <= '1';
            end if;

        end if;
end process;

-- processo para controle do barramento de enderecamento da memoria
process(clock,zera_add)
begin
        if (zera_add='1') then
            address(16 downto 0) <= (others => '0');
        elsif (clock'event and clock='1') then
            if (grava='1' or chadd = '1') then
                address(16 downto 0) <= address(16 downto 0) + 1;
            end if;
            if (le='1' and address(16 downto 0)="11111111111111111") then
                end_read <= '1';
            else
                end_read <= '0';
            end if;
        end if;
end process;

-- processo para o sincronismo entre o clock da maquina de leitura da imagem e do clock do
-- processador da imagem
process(clock, proc_clock)
        variable step: std_logic;
begin
        if (clock'event and clock = '1') then
            if (step = '0') then
                if (proc_clock = '1') then
                    chadd1 <= '1';
                    step := '1';
                else
                    chadd1 <= '0';
                end if;
            else
```

```
                        if (proc_clock = '1') then
                                chadd1 <= '0';
                        else
                                chadd1 <= '0';
                                step := '0';
                        end if;
                end if;
        end if;
    end process;

    chadd <= chadd1 and clock;

    -- processo para sinalizacao do incio e fim da aquisicao de videos.
    process(fst, reset)
    begin
        if (reset = '0') then
                passo <= '0';
                stop <= '0';
                comeca <= '0';
        elsif (fst'event and fst = '1') then
                if (passo = '1') then
                        stop <= '1';
                        passo <= '0';
                elsif (config = '1')then
                        comeca <= '1';
                        passo <= '1';
                end if;
        end if;
    end process;


end block;



end architecture structural;
```

I2C.VHD

```
--
-- Simple I2C controller
--
-- 1) No multimaster
-- 2) No slave mode
-- 3) No fifo's
--
-- notes:
-- Every command is acknowledged. Do not set a new command before previous
is acknowledged.
-- Dout is available 1 clock cycle later as cmd_ack
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

package I2C is
      component simple_i2c is
      port (
            clk : in std_logic;
            ena : in std_logic;
            nReset : in std_logic;

            clk_cnt : in unsigned(7 downto 0);  -- 4x SCL

            -- input signals
            start,
            stop,
            read,
            write,
            ack_in : std_logic;
            Din : in std_logic_vector(7 downto 0);

            -- output signals
            cmd_ack : out std_logic;
            ack_out : out std_logic;
            Dout : out std_logic_vector(7 downto 0);

            -- i2c signals
            SCL : inout std_logic;
            SDA : inout std_logic
      );
      end component simple_i2c;
end package I2C;


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity simple_i2c is
      port (
            clk : in std_logic;
            ena : in std_logic;
            nReset : in std_logic;

            clk_cnt : in unsigned(7 downto 0);  -- 4x SCL
```

```
            -- input signals
            start,
            stop,
            read,
            write,
            ack_in : std_logic;
            Din : in std_logic_vector(7 downto 0);

            -- output signals
            cmd_ack : out std_logic;
            ack_out : out std_logic;
            Dout : out std_logic_vector(7 downto 0);

            -- i2c signals
            SCL : inout std_logic;
            SDA : inout std_logic
      );
end entity simple_i2c;

architecture structural of simple_i2c is
      component i2c_core is
      port (
            clk : in std_logic;
            nReset : in std_logic;

            clk_cnt : in unsigned(7 downto 0);

            cmd : in std_logic_vector(2 downto 0);
            cmd_ack : out std_logic;
            busy : out std_logic;

            Din : in std_logic;
            Dout : out std_logic;

            SCL : inout std_logic;
            SDA : inout std_logic
      );
      end component i2c_core;

      -- commands for i2c_core
      constant CMD_NOP  : std_logic_vector(2 downto 0) := "000";
      constant CMD_START      : std_logic_vector(2 downto 0) := "010";
      constant CMD_STOP : std_logic_vector(2 downto 0) := "011";
      constant CMD_READ : std_logic_vector(2 downto 0) := "100";
      constant CMD_WRITE      : std_logic_vector(2 downto 0) := "101";

      -- signals for i2c_core
      signal core_cmd : std_logic_vector(2 downto 0);
      signal core_ack, core_busy, core_txd, core_rxd : std_logic;

      -- signals for shift register
      signal sr : std_logic_vector(7 downto 0); -- 8bit shift register
      signal shift, ld : std_logic;

      -- signals for state machine
      signal go, host_ack : std_logic;
begin
      -- hookup i2c core
      u1: i2c_core port map (clk, nReset, clk_cnt, core_cmd, core_ack,
core_busy, core_txd, core_rxd, SCL, SDA);

      -- generate host-command-acknowledge
      cmd_ack <= host_ack;
```

```
      -- generate go-signal
      go <= (read or write) and not host_ack;

      -- assign Dout output to shift-register
      Dout <= sr;

      -- assign ack_out output to core_rxd (contains last received bit)
      ack_out <= core_rxd;

      -- generate shift register
      shift_register: process(clk)
      begin
            if (clk'event and clk = '1') then
                  if (ld = '1') then
                        sr <= din;
                  elsif (shift = '1') then
                        sr <= (sr(6 downto 0) & core_rxd);
                  end if;
            end if;
      end process shift_register;

      --
      -- state machine
      --
      statemachine : block
            type states is (st_idle, st_start, st_read, st_write, st_ack,
st_stop);
            signal state : states;
            signal dcnt : unsigned(2 downto 0);
      begin
            --
            -- command interpreter, translate complex commands into simpler
I2C commands
            --
            nxt_state_decoder: process(clk, nReset, state, dcnt, core_txd,
core_cmd, go, start,
            read, core_ack, sr, stop, ack_in)
                  variable nxt_state : states;
                  variable idcnt : unsigned(2 downto 0);
                  variable ihost_ack : std_logic;
                  variable icore_cmd : std_logic_vector(2 downto 0);
                  variable icore_txd : std_logic;
                  variable ishift, iload : std_logic;
            begin
                  -- 8 databits (1byte) of data to shift-in/out
                  idcnt := dcnt;

                  -- no acknowledge (until command complete)
                  ihost_ack := '0';

                  icore_txd := core_txd;

                  -- keep current command to i2c_core
                  icore_cmd := core_cmd;

                  -- no shifting or loading of shift-register
                  ishift := '0';
                  iload := '0';

                  -- keep current state;
                  nxt_state := state;
                  case state is
                        when st_idle =>
                              if (go = '1') then
```

```
                                    if (start = '1') then
                                          nxt_state := st_start;
                                          icore_cmd := CMD_START;
                                    elsif (read = '1') then
                                          nxt_state := st_read;
                                          icore_cmd := CMD_READ;
                                          idcnt := "111";
                                    else
                                          nxt_state := st_write;
                                          icore_cmd := CMD_WRITE;
                                          idcnt := "111";
                                          iload := '1';
                                    end if;
                              end if;

                        when st_start =>
                              if (core_ack = '1') then
                                    if (read = '1') then
                                          nxt_state := st_read;
                                          icore_cmd := CMD_READ;
                                          idcnt := "111";
                                    else
                                          nxt_state := st_write;
                                          icore_cmd := CMD_WRITE;
                                          idcnt := "111";
                                          iload := '1';
                                    end if;
                              end if;

                        when st_write =>
                              if (core_ack = '1') then
                                    idcnt := dcnt -1; -- count down
Data_counter
                                    icore_txd := sr(7);
                                    if (dcnt = 0) then
                                          nxt_state := st_ack;
                                          icore_cmd := CMD_READ;
                                    else
                                          ishift := '1';
--                                        icore_txd := sr(7);
                                    end if;
                              end if;

                        when st_read =>
                              if (core_ack = '1') then
                                    idcnt := dcnt -1; -- count down
Data_counter
                                    ishift := '1';
                                    if (dcnt = 0) then
                                          nxt_state := st_ack;
                                          icore_cmd := CMD_WRITE;
                                          icore_txd := ack_in;
                                    end if;
                              end if;

                        when st_ack =>
                              if (core_ack = '1') then
                                    -- generate command acknowledge signal
                                    ihost_ack := '1';

                                    -- Perform an additional shift, needed
for 'read' (store last received bit in shift register)
                                    ishift := '1';
```

```
                                    -- check for stop; Should a STOP command
be generated ?
                                    if (stop = '1') then
                                          nxt_state := st_stop;
                                          icore_cmd := CMD_STOP;
                                    else
                                          nxt_state := st_idle;
                                          icore_cmd := CMD_NOP;
                                    end if;
                              end if;

                        when st_stop =>
                              if (core_ack = '1') then
                                    nxt_state := st_idle;
                                    icore_cmd := CMD_NOP;
                              end if;

                        when others => -- illegal states
                              nxt_state := st_idle;
                              icore_cmd := CMD_NOP;
                  end case;

                  -- generate registers
                  if (nReset = '0') then
                        core_cmd <= CMD_NOP;
                        core_txd <= '0';

                        shift <= '0';
                        ld <= '0';

                        dcnt <= "111";
                        host_ack <= '0';

                        state <= st_idle;
                  elsif (clk'event and clk = '1') then
                        if (ena = '1') then
                              state <= nxt_state;

                              dcnt <= idcnt;
                              shift <= ishift;
                              ld <= iload;

                              core_cmd <= icore_cmd;
                              core_txd <= icore_txd;

                              host_ack <= ihost_ack;
                        end if;
                  end if;
            end process nxt_state_decoder;

      end block statemachine;

end architecture structural;


--
--
-- I2C Core
--
-- Translate simple commands into SCL/SDA transitions
-- Each command has 5 states, A/B/C/D/idle
--
-- start:   SCL   ~~~~~~~~~~\____
--    SDA   ~~~~~~~~\______
```

```
--           x | A | B | C | D | i
--
-- repstart SCL   ____/~~~~\___
--    SDA   __/~~~\______
--           x | A | B | C | D | i
--
-- stop     SCL   ____/~~~~~~~~
--    SDA   ==\____/~~~~~
--           x | A | B | C | D | i
--
--- write   SCL   ____/~~~~\____
--    SDA   ==X=========X=
--           x | A | B | C | D | i
--
--- read    SCL   ____/~~~~\____
--    SDA   XXXX=====XXXX
--           x | A | B | C | D | i
--

-- Timing:        Normal mode Fast mode
------------------------------------------------------------------
-- Fscl           100KHz            400KHz
-- Th_scl         4.0us       0.6us High period of SCL
-- Tl_scl         4.7us       1.3us Low period of SCL
-- Tsu:sta        4.7us       0.6us setup time for a repeated start
condition
-- Tsu:sto        4.0us       0.6us setup time for a stop conditon
-- Tbuf           4.7us       1.3us Bus free time between a stop and start
condition
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity i2c_core is
      port (
            clk : in std_logic;
            nReset : in std_logic;

            clk_cnt : in unsigned(7 downto 0);

            cmd : in std_logic_vector(2 downto 0);
            cmd_ack : out std_logic;
            busy : out std_logic;

            Din : in std_logic;
            Dout : out std_logic;

            SCL : inout std_logic;
            SDA : inout std_logic
      );
end entity i2c_core;

architecture structural of i2c_core is
      constant CMD_NOP  : std_logic_vector(2 downto 0) := "000";
      constant CMD_START      : std_logic_vector(2 downto 0) := "010";
      constant CMD_STOP : std_logic_vector(2 downto 0) := "011";
      constant CMD_READ : std_logic_vector(2 downto 0) := "100";
      constant CMD_WRITE      : std_logic_vector(2 downto 0) := "101";

      type cmds is (idle, start_a, start_b, start_c, start_d, stop_a,
stop_b, stop_c, rd_a, rd_b, rd_c, rd_d, wr_a, wr_b, wr_c, wr_d);
      signal state : cmds;
```

```
	signal SDAo, SCLo : std_logic;
	signal txd : std_logic;
	signal clk_en, slave_wait :std_logic;
	signal cnt : unsigned(7 downto 0) := clk_cnt;
begin
	-- whenever the slave is not ready it can delay the cycle by pulling
SCL low
	slave_wait <= '1' when ((SCLo = '1') and (SCL = '0')) else '0';

	-- generate clk enable signal
	gen_clken: process(clk, nReset, txd)
	begin
		if (nReset = '0') then
			cnt <= (others => '0');
			clk_en <= '1'; --'0';
		elsif (clk'event and clk = '1') then
			if (cnt = 0) then
				clk_en <= '1';
				cnt <= clk_cnt;
			else
				if (slave_wait = '0') then
					cnt <= cnt -1;
				end if;
				clk_en <= '0';
			end if;
		end if;
	end process gen_clken;

	-- generate statemachine
	nxt_state_decoder : process (clk, nReset, state, cmd, SDA, Din, txd)
		variable nxt_state : cmds;
		variable icmd_ack, ibusy, store_sda : std_logic;
		variable itxd : std_logic;
	begin

		nxt_state := state;

		icmd_ack := '0'; -- default no acknowledge
		ibusy := '1'; -- default busy

		store_sda := '0';

		itxd := txd;

		case (state) is
			-- idle
			when idle =>
				case cmd is
					when CMD_START =>
						nxt_state := start_a;
						icmd_ack := '1'; -- command completed

					when CMD_STOP =>
						nxt_state := stop_a;
						icmd_ack := '1'; -- command completed

					when CMD_WRITE =>
						nxt_state := wr_a;
						icmd_ack := '1'; -- command completed
						itxd := Din;

					when CMD_READ =>
						nxt_state := rd_a;
						icmd_ack := '1'; -- command completed
```

```
                              when others =>
                                    nxt_state := idle;
-- don't acknowledge NOP command                                        icmd_ack
:= '1'; -- command completed
                                    ibusy := '0';
                        end case;

                  -- start
                  when start_a =>
                        nxt_state := start_b;

                  when start_b =>
                        nxt_state := start_c;

                  when start_c =>
                        nxt_state := start_d;

                  when start_d =>
                        nxt_state := idle;
                        ibusy := '0'; -- not busy when idle


                  -- stop
                  when stop_a =>
                        nxt_state := stop_b;

                  when stop_b =>
                        nxt_state := stop_c;

                  when stop_c =>
--                      nxt_state := stop_d;

--                when stop_d =>
                        nxt_state := idle;
                        ibusy := '0'; -- not busy when idle

                  -- read
                  when rd_a =>
                        nxt_state := rd_b;

                  when rd_b =>
                        nxt_state := rd_c;

                  when rd_c =>
                        nxt_state := rd_d;
                        store_sda := '1';

                  when rd_d =>
                        nxt_state := idle;
                        ibusy := '0'; -- not busy when idle

                  -- write
                  when wr_a =>
                        nxt_state := wr_b;

                  when wr_b =>
                        nxt_state := wr_c;

                  when wr_c =>
                        nxt_state := wr_d;

                  when wr_d =>
                        nxt_state := idle;
```

```
                        ibusy := '0'; -- not busy when idle

            end case;

            -- generate regs
            if (nReset = '0') then
                  state <= idle;
                  cmd_ack <= '0';
                  busy <= '0';
                  txd <= '0';
                  Dout <= '0';
            elsif (clk'event and clk = '1') then
                  if (clk_en = '1') then
                        state <= nxt_state;
                        busy <= ibusy;

                        txd <= itxd;
                        if (store_sda = '1') then
                              Dout <= SDA;
                        end if;
                  end if;

                  cmd_ack <= icmd_ack and clk_en;
            end if;
      end process nxt_state_decoder;

      --
      -- convert states to SCL and SDA signals
      --
      output_decoder: process (clk, nReset, state, SCLo, SDA, Din)
            variable iscl, isda : std_logic;
      begin
            case (state) is
                  when idle =>
                        iscl := SCLo; -- keep SCL in same state
                        isda := SDA; -- keep SDA in same state

                  -- start
                  when start_a =>
                        iscl := SCLo; -- keep SCL in same state (for
repeated start)
                        isda := '1'; -- set SDA high

                  when start_b =>
                        iscl := '1';      -- set SCL high
                        isda := '1'; -- keep SDA high

                  when start_c =>
                        iscl := '1';      -- keep SCL high
                        isda := '0'; -- sel SDA low

                  when start_d =>
                        iscl := '0'; -- set SCL low
                        isda := '0'; -- keep SDA low

                  -- stop
                  when stop_a =>
                        iscl := '0'; -- keep SCL disabled
                        isda := '0'; -- set SDA low

                  when stop_b =>
                        iscl := '1'; -- set SCL high
                        isda := '0'; -- keep SDA low
```

```
                  when stop_c =>
                        iscl := '1'; -- keep SCL high
                        isda := '1'; -- set SDA high

                  -- write
                  when wr_a =>
                        iscl := '0';      -- keep SCL low
--                      isda := txd; -- set SDA
                        isda := Din;

                  when wr_b =>
                        iscl := '1';      -- set SCL high
--                      isda := txd; -- set SDA
                        isda := Din;

                  when wr_c =>
                        iscl := '1';      -- keep SCL high
--                      isda := txd; -- set SDA
                        isda := Din;

                  when wr_d =>
                        iscl := '0'; -- set SCL low
--                      isda := txd; -- set SDA
                        isda := Din;

                  -- read
                  when rd_a =>
                        iscl := '0'; -- keep SCL low
                        isda := '1'; -- tri-state SDA

                  when rd_b =>
                        iscl := '1'; -- set SCL high
                        isda := '1'; -- tri-state SDA

                  when rd_c =>
                        iscl := '1'; -- keep SCL high
                        isda := '1'; -- tri-state SDA

                  when rd_d =>
                        iscl := '0'; -- set SCL low
                        isda := '1'; -- tri-state SDA
            end case;

            -- generate registers
            if (nReset = '0') then
                  SCLo <= '1';
                  SDAo <= '1';
            elsif (clk'event and clk = '1') then
                  if (clk_en = '1') then
                        SCLo <= iscl;
                        SDAo <= isda;
                  end if;
            end if;
      end process output_decoder;

      SCL <= '0' when (SCLo = '0') else 'Z'; -- since SCL is externally
pulled-up convert a '1' to a 'Z'(tri-state)
      SDA <= '0' when (SDAo = '0') else 'Z'; -- since SDA is externally
pulled-up convert a '1' to a 'Z'(tri-state)
--    SCL <= SCLo;
--    SDA <= SDAo;

end architecture structural;
```

MEMORIA.VHD

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

package memory is
        component memoria is
                port (
                        clock    : in      std_logic;
                        write    : in      std_logic;
                        enable   : in      std_logic;

                        datain   : in      std_logic_vector(7 downto 0);
                        dataout : out     std_logic_vector(7 downto 0);
                        DQ               : inout   std_logic_vector(7 downto 0);

                        WEn              : out     std_logic;
                        CEn              : out     std_logic;
                        OEn              : out     std_logic
                        );

        end component memoria;
end package memory;

-------------------------------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity memoria is

        port (
                        -- sinais de entrada do controlador da memoria
                        clock    : in      std_logic;
                        write    : in      std_logic;
                        enable   : in      std_logic;
```

```
                datain   : in      std_logic_vector (7 downto 0);

                -- sinais de saida do controlador da memoria
                dataout : out     std_logic_vector (7 downto 0);
                DQ              : inout std_logic_vector (7 downto 0);
                WEn             : out     std_logic;
                CEn             : out     std_logic;
                OEn             : out     std_logic
                );
end memoria;

architecture behv_memoria of memoria is
-- criacao de um sinal intermediario do controlador da memoria
signal passo: std_logic;

        begin
                CEn <= enable;
                OEn <= '0';
                process (clock, write)
                        begin

                                if (clock'event and clock='1') then

                                        if (write='1') then   -- inicio do ciclo de leitura
                                                if passo='0' then -- comanda leitura e garante o tempo
                                                                -- de acesso a memoria
                                                        WEn <= '1';
                                                        passo <= '1';

                                                else -- depois do tempo de acesso a memoria le
                                                     -- o barramento de dados
                                                        DQ <= "ZZZZZZZZ"; -- coloca o sinal do
tipo inout em alta impedancia para fazer a leitura.
                                                        dataout <= DQ;
                                                        passo <= '0';

                                                end if;
                                        else
                                                -- inicio do ciclo de escrita
```

```
                                        if passo='0' then  -- garante o enderecamento e
disponibilizacao dos dados antes do comando de escrita
                                            DQ <= datain;
                                            passo <= '1';

                                        else -- comando de escrita
                                            WEn <= '0';
                                            passo <='0';

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