[[1]](#footnote-1)

Desenvolvimento de um

Processador Gráfico em FPGA

Alunos da Disciplina ENGG57 – Laboratório Integrado IV-A

|  |  |
| --- | --- |
|  |  |

*Abstract*— Foi desenvolvido um processador gráfico na plataforma DE2-115 da Altera com o objetivo de exibir uma imagem qualquer em um monitor compatível pelo cabo VGA.

*Index Terms*—FLASH, Processador Gráfico, SDRAM, VGA, FPGA.

# **INTRODUÇÃO**

Este artigo apresenta um protótipo FPGA e uma parte da placa gráfica processadora. Tal processador é capaz de trabalhar com imagens RGB-24 bits (True Color), com a resolução mínima VGA. Foram gerenciadas as partes internas de um processador através do controlador de flash, controlador de SDRAM, controlador de VGA, controlador de FIFO e controlador de imagem.

# **APRESENTAÇÃO DO PROBLEMA**

O presente trabalho tem como objetivo ampliar o conhecimento sobre um processador gráfico 2D e sua troca de informações com o dispositivo de memória e monitor.

Nessa primeira parte do trabalho deverá ser implementado um módulo responsável pela exibição de imagens em formato gráfico, tal módulo deve ser capaz de trabalhar com imagens True Color, para monitores com resolução mínima VGA (640x480), além de conseguir carregar imagens externamente com a utilização de recursos da placa.

As configurações da imagem serão trabalhadas de forma a atender as especificações direcionadas para a parte 1 do trabalho, ou seja, ao considerar que uma tela VGA equivale a um quadro, a imagem a ser armazenada poderá ter o tamanho de no máximo 9 quadros e apresentar as seguintes disposições: horizontal (1x9), vertical (9x1) e igualitária (3x3). Além disso, haverá a possibilidade de colocar a imagem em qualquer coluna, quando situada na disposição horizontal; em qualquer linha quando situada na disposição vertical; em qualquer linha ou coluna quando situada na disposição igualitária. Para todas as disposições devem ser atendidas a restrição de não exceder os limites de exibição.

Para cada pixel mostrado serão apresentadas as coordenadas atuais e as posições de memória serão compartilhadas com o processador principal por meio de um conjunto de regras.

# **organização do projeto**

Foram realizadas reuniões periódicas durante as semanas para avaliar os procedimentos a serem realizados na implementação do projeto. Além disso, foram divididos grupos para projetar diferentes partes do processador e para aprender a gerenciar a memória flash, SDRAM, assim como os mecanismos de uso do dispositivo VGA.

Após a implementação de cada módulo, as equipes de trabalho se reuniram para realizar a integração das partes. É importante ressaltar que o grupo da SDRAM necessitou de mais componentes devido a maior complexidade do problema.

O driver da placa foi instalado no máximo de computadores possíveis para um melhor gerenciamento de recursos disponíveis, uma vez que alguns computadores estavam em uso durante a nossa visita ao laboratório.

# **DESENVOLVIMENTO**

#### **Considerações Iniciais**

Para inicialização do desenvolvimento o driver da placa foi instalado no computador e foi utilizado o Control Panel (software presente no CD que acompanha o FPGA, desenvolvido pela Terasic) para debug e controle da placa. Outra medida necessária foi a instalação da ferramenta Quartus-Altera, software necessário para desenvolvimento dos módulos que compõem o processador gráfico.

A princípio, a equipe pensou em utilizar o cartão SD para armazenamento das imagens, mas, devido à complexidade da criação de um controlador do cartão, optou-se por armazenar o conteúdo na memória flash, já que seria um processo mais rápido levando-se em consideração que não existia muito tempo disponível para a concretização do trabalho.

Escolhemos as imagens em formato de arquivo Bitmap para a aplicação, visto que o formato não utiliza compressão, preservando seus detalhes, embora os arquivos neste formato sejam extremamente grandes. Desse modo, criarmos um conversor em linguagem C, responsável por converter uma imagem de formato Bitmap para um formato definido pelo grupo com extensão .LAB4. Este arquivo consiste em um apenas nos pixels da imagem a ser exibida, no qual se encontra em little-endian (BGR).

O esquemático contendo os módulos implementados e suas conexões entre eles pode ser visualizado no Anexo A. As funcionalidades de cada um deles serão descritas no decorrer do artigo.

#### **Memória Flash**

A memória flash irá atuar como fonte inicial da imagem a ser exibida, armazenando-as e transferindo os dados para a memória SDRAM, salientando que para armazenar as imagens (formato .LAB4) foi utilizado o Control Panel.

Definimos a flash como a memória inicial de armazenamento das imagens, já que foi constatado a compatibilidade de espaço da memória e das imagens a serem guardadas (formato .LAB4), além de ser uma memória não-volátil, nos garantindo a estabilidade das imagens gravadas. Uma vez definida a utilização da flash, foi feito um estudo do seu datasheet objetivando o correto entendimento do seu funcionamento e das suas temporizações.

Foi verificado, na placa, que utilizamos a memória flash SD9GL064N com velocidade de acesso de 90 ns. Essa velocidade refere-se ao tempo máximo de espera necessário para o dado na saída da memória estar pronta após uma mudança de endereço. Cada endereço acessa uma palavra de 8 bits.

Analisada as temporizações temos que o tempo para o reset ser realizado é de 500ns. O tempo de espera de leitura ou escrita depois do reset é de 50ns. O tempo de espera para obter o dado após a ativação das portas de chip e saída ou mudança do endereço de memória é de 90ns.

Foi construída uma FSM (Anexo C) para controlar a flash com o clock com valor de 25 Mhz (40ns de período) e um buffer com tamanho de 24 bits a ser transferido para a memória SDRAM após estar cheio. A máquina de estados é definida da seguinte forma: Foi criado o estado IDLE, no qual é seu estado de reset. Enquanto estiver na IDLE, a máquina de estados fica esperando por um sinal de start para inicializar, vindo do controlador de SDRAM. Uma vez dado o start o processo se inicia indo para o estado de reset. No estado de reset, o programa fica em loop neste estado por 13 períodos de clock (520 ns). Este estado manda o comando de reset para a memória flash, respeitando a temporização previamente estabelecida.

Após este período ele vai para o estado de getting\_ready (ele deve ficar durante 50 ns após o reset sem fazer nada), que é o estado antes da inicialização da leitura, após isso ele vai para o estado de pre\_read, estado este que serve para esperar a saída de dados da flash se atualizar, uma vez que o endereço de memória tenha sido modificado, tal tempo é de 120ns para a máquina, uma vez que a flash espera ao menos 90ns. A partir daí, amáquina de estados faz a leitura do primeiro registro da memória e insere nos bits mais significativos do buffer de transferência, atualiza o endereço para o próximo registro da memória, espera para pegar os próximos 8 bits (estado de pre\_read novamente) e colocar nos bits intermediários do buffer, atualiza o endereço mais uma vez, espera mais 120ns (pre\_read), realiza outra leitura de 8 bits da memória flash e adiciona nos menos significativos do buffer. Um vez que o buffer está completamente preenchido, a máquina fica esperando para enviar essa informação para o controlador da SDRAM, quando a SDRAM informa que ela já pegou os dados que está no buffer, se a memória não tiver sido completamente lida (toda a imagem), o processo de preenchimento do buffer é retomado, até que todos os endereços sejam lidos, que seria o último endereço (8.294.399 ), e a partir desse ponto ocorre o retorno ao estado de idle e terá um sinal de finish que é uma informação para o controlador de SDRAM, que acabou de pegar todos os dados dispostos na memória flash.

#### **Memória SDRAM**

A SDRAM a ser utilizada foi a ISA42S16320D-7TL. O datasheet apresentado pelo CD que veio junto com a placa da Altera é de uma versão diferente da memória utilizada, entretanto as temporizações da versão anterior são mais pessimistas que a versão da própria memória, sendo assim foi possível utilizá-la como base para esse projeto.

A SDRAM apresenta 64MB de dados, sendo dividida em 4 bancos, cada banco apresentando 8.192 linhas, cada linha apresentando 1024 colunas e cada coluna apresentando uma palavra de 16 bits. A placa disponibiliza duas dessas memórias e as acessa simultaneamente, dando a impressão de uma memória de 128 MB com 32 bits de palavra.

As seguintes considerações foram feitas para se construir a máquina de estados, que controla a SDRAM: O CAS (column Acess Strobe) é 2, ou seja, após o comando de leitura, o dado fica disponível dois pulsos de clock depois; e é necessário a execução de um comando refresh a cada 7,8125 us.

As temporizações realizadas nesse trabalho extrapolam o mínimo necessário em quase todos os casos, com exceção do comando de leitura e escrita que tem tempos exatos. Algumas delas são maiores que o necessário, mas não afetam o desempenho final do trabalho, não visto a necessidade de melhorar o desempenho da FSM. A FSM funciona a 75 Mhz (13,33ns de período). Essa frequência foi escolhida para evitar slack entre a comunicação dos controladores da SDRAM e da FLASH. A frequência da FSM e da memória SDRAM é a mesma, mas estão deslocadas entre si 180º. O diagrama de estados para esta FSM pode ser visualizada nos anexos D – F.

Inicialmente a FSM encontra-se no seu estado de reset, Desligado, e espera o sinal de start do controlador geral de imagem. Uma vez inicializado a máquina mantém no mesmo estado durante 24575 pulsos de clock enviando o sinal de NOP para a memória, respeitando os 200 us exigidos. Após, envia-se um sinal de PRECHARGE, para carregar todos os bancos, durante 3 pulsos de clock. Após isso, 8 cliclos de Refresh, com um espaço de 9 pulsos de clock de NOP entre os Refresh. Após isso, é dado o LOADMODE da SDRAM, carregando as configurações da mesma a serem utilizadas: Rajada de apenas 1 palavra por vez, modo sequencial, latência de CAS 2, com rajada de escrita simples.

A FSM então entra em processo de escrita, obtendo dados da Flash e escrevendo na SDRAM. A FSM da SDRAM reseta a FSM da Flash, enviando um sinal de reset por 8 pulsos de clock. Após isso, envia-se um sinal de start para a flash durante 6 pulsos de clock. Com isso o controlador da flash está em funcionamento. O controlador da SDRAM espera o da Flash preencher o buffer de 24 bits e informar a primeira quando estiver pronta. Quando encontra-se pronta, é feito o processo de escrita na RAM: É enviado o comando de ACTIVE para ativar um banco e uma linha, para então gravar o dado numa coluna indicada com o comando de Write. A linha e coluna onde os dados serão escritos é sequencial, começando da linha 0 e da coluna 0, e indo até a linha 2699 e coluna 1023. Dos 32 bits disponíveis, apenas 24 são utilizados. O processo de write sempre verifica o tempo em que o Refresh foi dado pela última vez. Se a FSM ficou sem dar Refresh na RAM por 300 pulsos de clock, um comando Refresh é dado. Após toda imagem ser lida da Flash para a Ram, inicia-se o processo de leitura.

O processo de leitura envolve o controlador geral de imagem, que funciona na mesma frequência do controlador da SDRAM, apenas deslocados em 180º.

O processo de leitura consiste em esperar o comando de next do controlador geral de imagem. O controlador de imagem informa ao controlador de SDRAM a linha e a coluna a serem lidos na memória, e este último os pixels e envia para a FIFO da VGA.

#### **Controlador de imagem**

No anexo C, encontra-se o início da FSM do controlador de imagem.

A FSM do controlador de imagem reseta a FSM da RAM e a manda operar. Após disso, ela é responsável por achar na memória onde estão os pixels para formar a imagem e enviar para a FIFO da VGA.

Como os pixels da imagem estão em ordem da esquerda para a direita e de cima para baixo, é possível saber que numa mesma linha os pixels estão em ordem crescente na memória. Para cada nova linha na VGA, é necessário achar onde está o primeiro pixel daquela linha. A máquina de estados que implementa esta funcionalidade pode ser visualizada no Anexo H.

#### **Controlador da interface VGA**

A tarefa de exibição de pixels na tela VGA deve ser realizada de forma sequencial, do canto superior esquerdo ao canto inferior direito, da esquerda para a direita e de cima para baixo. A resolução proposta e implementada para o trabalho em questão é a de 640x480 pixels a uma frequência de 60Hz. O esquemático do módulo de interface de vídeo pode ser visualizado no Anexo B.

Conforme as especificações do padrão VGA, o clock de operação deve ser de 25,175 MHz, com período muito maior do que o clock de operação da SDRAM. Além deste cenário assíncrono, a SDRAM possui períodos de refresh periódicos nos quais não pode enviar dados; estes fatores justificam a implementação de um buffer operacional na forma de uma FIFO (first-in, first-out) assíncrona para armazenar os pixels fornecidos pela SDRAM numa taxa mais rápida e liberar estes pixels a uma taxa mais lenta e constante para a saída de vídeo.

Inicialmente foi implementada uma FIFO que funcionaria especificamente nas frequências de clock 133 MHz (escrita) e 25 MHz (leitura), mas esta foi descartada em prol da utilização da implementação da FIFO assíncrona encontrada no IP Catalog do Quartus; esta não possui restrição quanto aos clocks utilizados e oferece diversos níveis de proteção contra metaestabilidade. A quantidade de pixels possíveis de serem armazenados ainda no buffer é uma saída deste módulo, de forma que a SDRAM possui esta informação para controlar sua lógica de escrita.

Uma vez que a FIFO não esteja vazia, é possível inicializar o processo de exibição de pixels na tela. Para cuidar disto foi implementado o módulo FIFO\_Reader, que funciona como um buffer para o sinal empty e gera o sinal de reset para o gerador de sincronização. Em etapas mais avançadas do mesmo projeto, este módulo será substituído por apenas um registrador ou simplesmente descartado, visto que a operação do sistema previsto não permite que a FIFO fique vazia novamente, uma vez tendo começado a receber dados.

O módulo denominado PixelLogic é o responsável por gerar os sinais de sincronização (HSYNC, VSYNC e BLANK), bem como decidir quando a tela estará ligada; esta atividade está relacionada à sincronização e as especificações da VGA.

O processo de atualização da tela inicia no canto superior esquerdo e pinta 1 pixel por vez da esquerda para direita. Ao completar a linha faz-se o incremento na variável correspondente à linha e o endereço da coluna é zerado, a fim de se inicializar a plotagem de uma nova linha. Uma vez que toda a tela foi plotada o processo de atualização começa novamente. Para controlar este processo, a especificação prevê que sejam utilizados os sinais de sincronização HSYNC e VSYNC, os quais devem ser ativados ao final de cada linha ou coluna, respectivamente, respeitando requisitos de tempo.

A Figura 1 mostra os requisitos temporais para estes sinais, ativos em 0; o contador h\_count conta a quantidade de colunas a serem consideradas e v\_count a quantidade de linhas. Durante um dado tempo antes (front porch) e depois (back porch) dos sinais de sincronização (sync), é especificado que a saída de vídeo seja obrigatoriamente desligada (R, G e B em zero ou BLANK ativo). Este comportamento pode ser compreendido analisando a Figura 2 para o sinal HSYNC e a Figura 3 para o VSYNC.

O módulo PixelLogic, portanto, possui dois contadores referentes a linha e coluna sendo “exibidas”. As colunas são incrementadas a cada pulso de clock e as linhas a cada fim de uma dada quantidade de colunas. Quando linha ou coluna ultrapassam o tamanho da imagem a ser exibida, a saída de vídeo imediatamente é desligada através do sinal BLANK e dos próprios sinais de cores. No intervalo (de linhas ou colunas) referente à ativação do pulso de sincronização, estes são ativos sem interferir com o restante da contagem. O módulo consiste, de forma geral, em dois contadores e alguns blocos de loop que verificam dados intervalos destes contadores e ativam os sinais de sincronização de acordo com estes intervalos.

#### **Geração de Sinais de Clock**

A placa em questão fornece um sinal de clock interno de 50 MHz e a possibilidade de utilizar PLLs (phase-locked loops) para gerar clocks com períodos múltiplos a este (ou a um sinal de clock externo, que não foi utilizado).

Na prática, o sinal de 25,175 MHz não pôde ser gerado a partir do uso dos PLLs, portanto foi utilizado então um sinal de 25 MHz tanto para alimentar o controlador da Flash quanto o controlador VGA. Os sinais de controle da interface VGA foram modificados em relação à especificação oficial devido a este fato, reduzindo os tempos de sincronização (front-porch e back-porch) para os observados na Figura 4.

O PLL gera também um sinal de clock de 75 MHz, o qual é utilizado no controlador de imagem e na SDRAM. A partir das análises de temporização pelo Quartus, fica claro que para atingir frequências muito mais altas que 75 MHz é necessário uma melhor otimização dos caminhos combinacionais existentes no projeto. Além disto, a utilização de frequências múltiplas diretas umas das outras facilitou o processo de compatibilidade entre os módulos de forma geral.

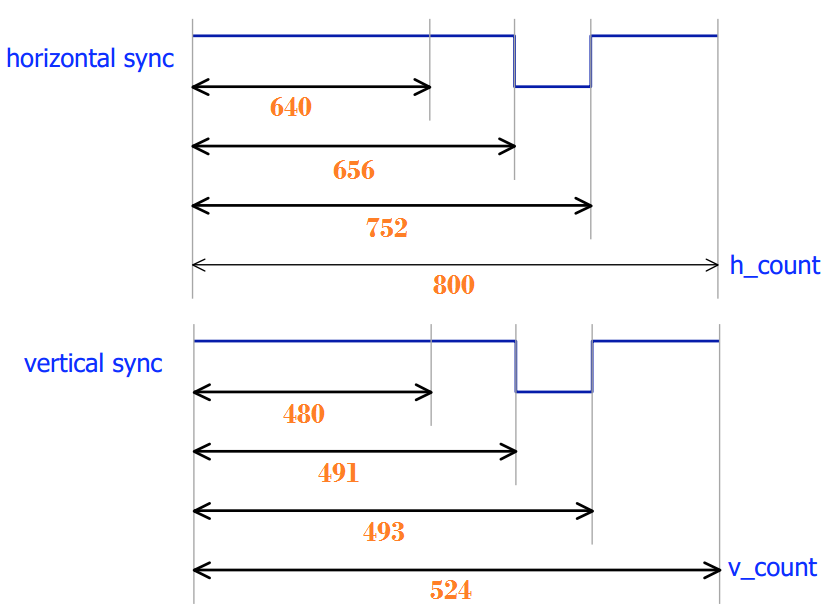


Figura 1 - Temporização VGA 640x480 60Hz ideal

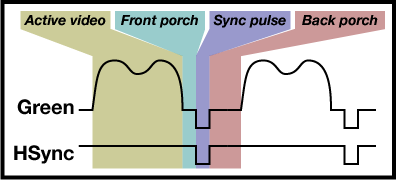


Figura 2 - Funcionamento do HSYNC

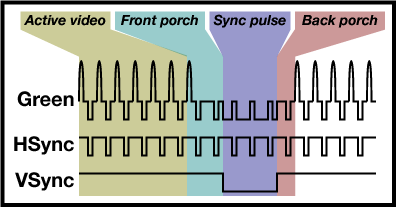


Figura 3 - Funcionamento do VSYNC

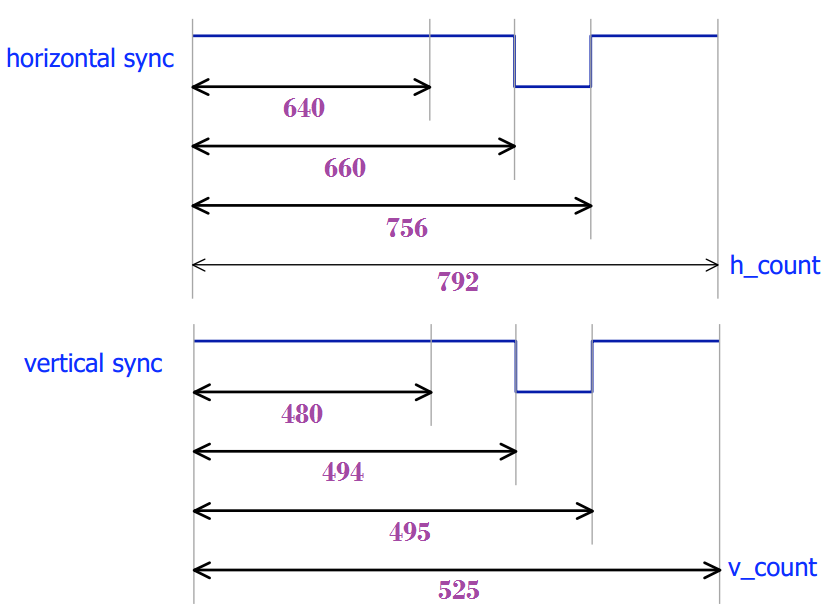


Figura 4 - Temporização VGA 640x480 60Hz implementada

#### **Depuração e análise lógica**

O analisador lógico serve para saber exatamente onde estão as saídas e entradas e fazer a verificação de tamanho, para ter conhecimento dos acontecimentos, ter a noção exata se o que está chegando da flash é o que realmente está sendo esperado e para detectar se o problema que a equipe teve para finalizar o trabalho estava na SDRAM ou na flash. O analisador lógico é uma ferramenta do Quartus para depurar o que foi feito.

# **considerações finais**

O trabalho ainda se encontra em andamento e deverá ser integrado com outros módulos de forma a constituir um processador gráfico; as próximas funcionalidades a serem implementadas são um módulo de gerenciamento de sprites e processador capaz de interpretar um programa e enviar instruções para o módulo gráfico.

# **Contribuições individuais**

### **Caio Abreu**

Participou das reuniões em classe e extra-classe, pesquisando e levando materiais de apoio para informações relevantes ao projeto (documentação do FPGA) e fomentar as discussões. Atuou na equipe de SDRAM, na compreensão e análise de viabilidade da proposta de solução do problema, além de participar de testes nas ferramentas relacionadas a aplicação (Quartus, Qsys e SOPC Builder).

### **Caio Queiroz**

Auxiliou na implementação do módulo SDRAM através de ideias e trabalhou na correção de bugs. Utilizou a ferramenta SOPC Builder e Qsys para facilitar a geração de módulos no Quartus, no início do projeto, embora a ideia foi descartada por um gerador manual.

### **Dhene Arlis**

Buscou informações sobre as ferramentas de debug disponíveis no Quartus. Fez cálculos para exibição da imagem na VGA, utilizou a ferramenta SOPC Builder e Qsys para facilitar a geração de módulos no Quartus, no início do projeto, embora a ideia foi descartada por um gerador manual. Verificou os contadores do h\_sync e v\_sync.

### **Gabriel Almeida**

Elaborou estratégias para evitar problemas com o refresh da SDRAM. Fez análises com as especificações do timing da saída VGA e criou um repositório para inserção de arquivos do grupo. Auxiliou na codificação da SDRAM e VGA. Participou das reuniões e discussões auxiliando na tomada de decisões do projeto.

### **Iago Pattas**

Fez análises do funcionamento geral da SDRAM, participou ativamente da codificação e construção da VGA e FIFO, através da estruturação dos dados. Realizou testes unitários e de integração dos códigos de VGA atualizados.

### **Ícaro Pereira**

Participou da implementação da memória flash através de ideias relevantes e da execução de testes da transferência de imagens estáticas. Realizou testes unitários e de integração no módulo da memória flash. Realizou cálculos para alcançar taxas de processador aceitável. Participou da decisão sobre a escolha da memória SDRAM em relação SRAM.

### **Igor Viana**

Participou da implementação inicial e da finalização do módulo VGA e da construção da estrutura necessária para o módulo. Fez a verificação de como seria a transferência de informações para o código Verilog e como se daria o armazenamento na memória.

### **Leonardo Leão**

Analisou dados sobre a temporização, o ciclo de leitura e o funcionamento geral da SDRAM e atuou efetivamente na implementação dos módulo da SDRAM e da flash, além de atuar na construção das máquinas de estado, na conversão do bitmap para .lab4 e na transferência de dados da memória flash para memória SDRAM. Resolveu problemas, após testes. Criou máquinas de estados para gerenciar a memória RAM e um arquivo.lab4. Realizou testes de partes em separado criando testbenches e teestes e análise de temporizações utilizando Signaltap.

### **Luan Cruz**

Contribuiu com processos de pesquisas para codificação do módulo VGA, a fim de buscar soluções para o desenvolvimento do projeto, participou da resolução de pequenos problemas/dúvidas relacionados a integração dos módulos e auxiliou especificamente no desenvolvimento do código responsável pela distribuição dos quadros (VGA), além disso, ajudou na elaboração e escrita do relatório geral.

### **Luana Vieira**

Realizou a leitura e buscou a compreensão dos datasheets de componentes para auxiliar na implementação dos módulos. Participou da criação de Test Benches e da execução de testes das partes e de integração. Buscou o entendimento da ferramenta SignalTap, como forma de auxiliar na verificação de como exatamente estava acontencendo a transferência de informação entre os módulos, além de organizar dados para a geração do relatório final.

### **Marcos Estrela**

Atuou inicialmente nos estágios iniciais de pesquisa para levantamento de dados e requisitos do projeto, verificando Datasheets e projetos similares. Teve participação no grupo de desenvolvimento da Flash, passando pela análise da FSM desenvolvida. Por fim, teve participação ativa no desenvolvimento do relatório final, coletando informações de cada parte do projeto desenvolvido e expondo-as no mesmo.

### **Pedro Carrano**

Apresentou informações revelantes à equipe sobre o gerenciamento de memória e auxiliou na implementação da SDRAM e na realização de testes para correção de pequenas falhas na transmissão de informações. Analisou a documentação de uso do SDRAM com Qsys. Utilizou a ferramenta SOPC Builder e Qsys para facilitar a geração de módulos no Quartus, no início do projeto, embora a ideia foi descartada por um gerador manual.

### **Rafael Ferrari**

Organização do melhor horário para reuniões extra-classe. Escolha de um arquivo bitmap para teste. Pesquisa sobre os assuntos pertinentes à resolução dos problemas propostos. Participação nas discussões sobre o funcionamento do sistema. Verificação de temporizações da VGA/FIFO. Implementação da VGA/FIFO. Resolução de problemas, após testes. Definição da interface da VGA. Teste dos códigos de VGA atualizados. Testes e implementação da integração do projeto.

# **Conclusão**

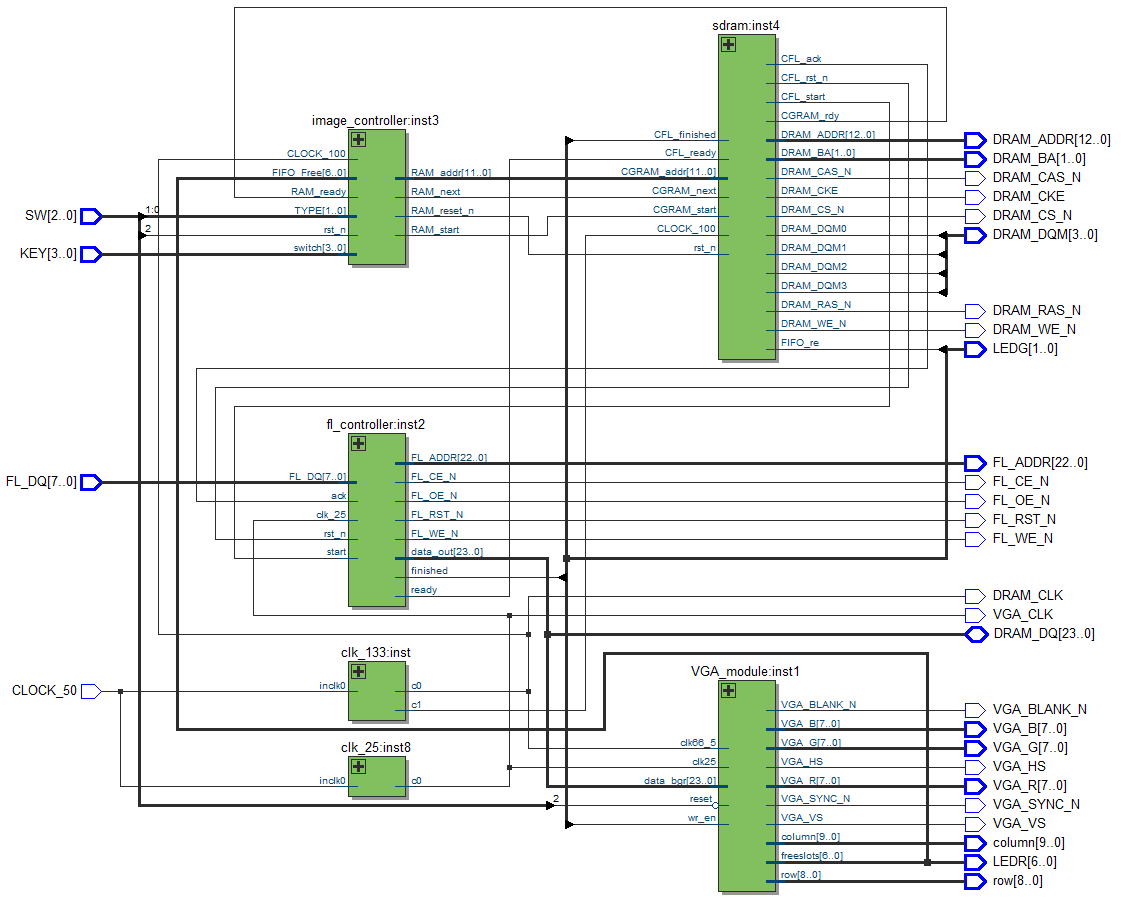
O trabalho implementado foi de extrema importância para colocar em prática os conhecimentos adquiridos durante a graduação.

**REFERENCES**

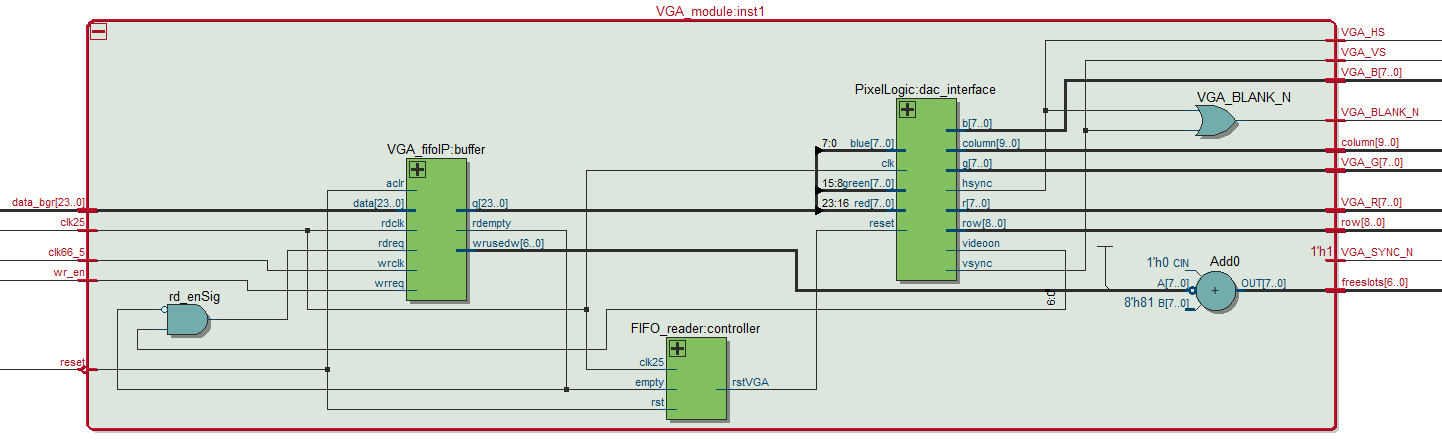
1. <http://lslwww.epfl.ch/pages/teaching/cours_lsl/ca_es/VGA.pdf>
2. <ftp://ftp.altera.com/up/pub/Altera_Material/16.0/Tutorials/Verilog/SignalTap.pdf>
3. <http://web.mit.edu/6.111/www/s2004/NEWKIT/vga.shtml>
4. <https://www.altera.com/documentation.html>
5. <http://de2-115.terasic.com>

# Anexos

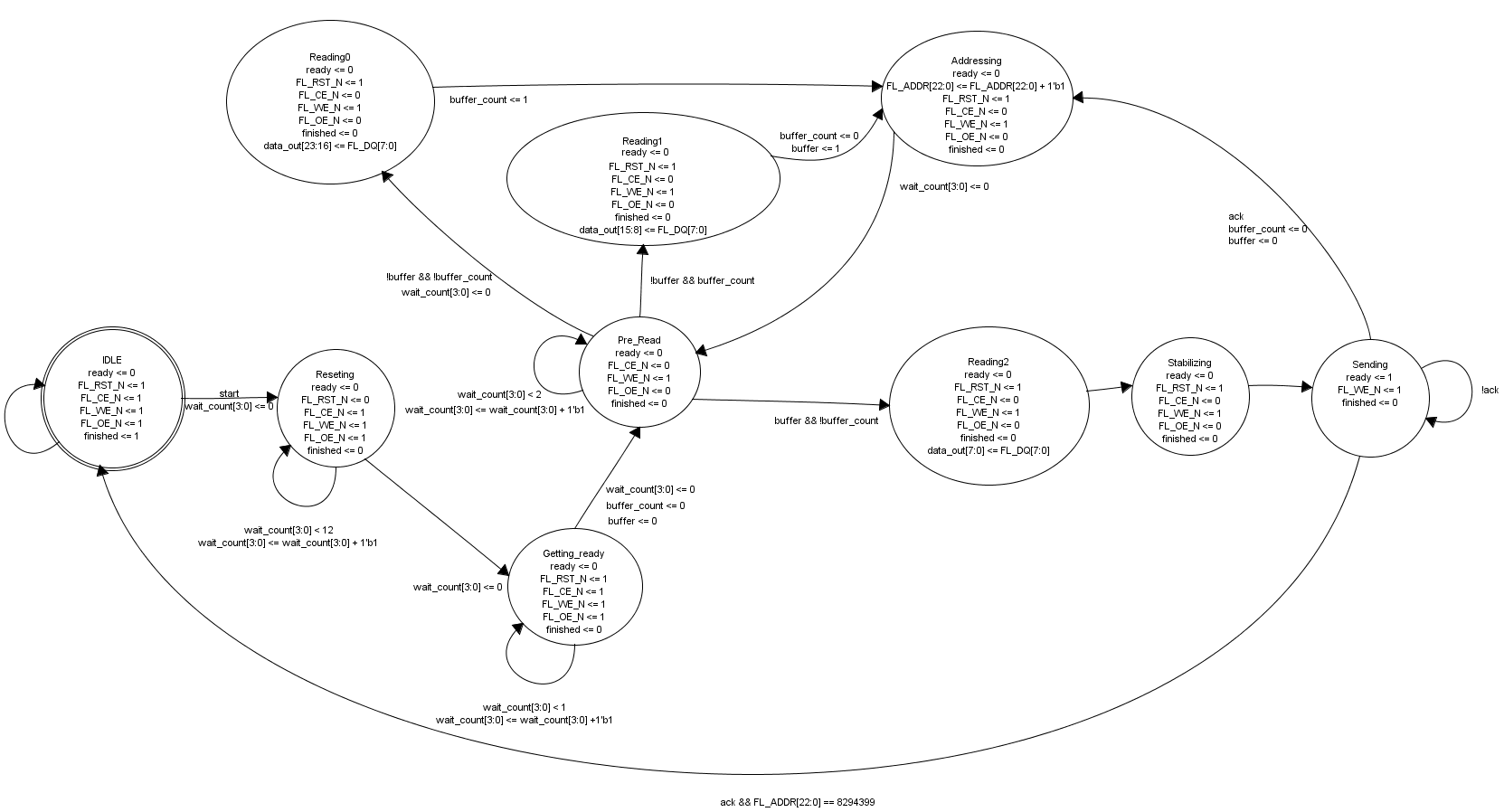
## Esquemático dos Módulos do Projeto



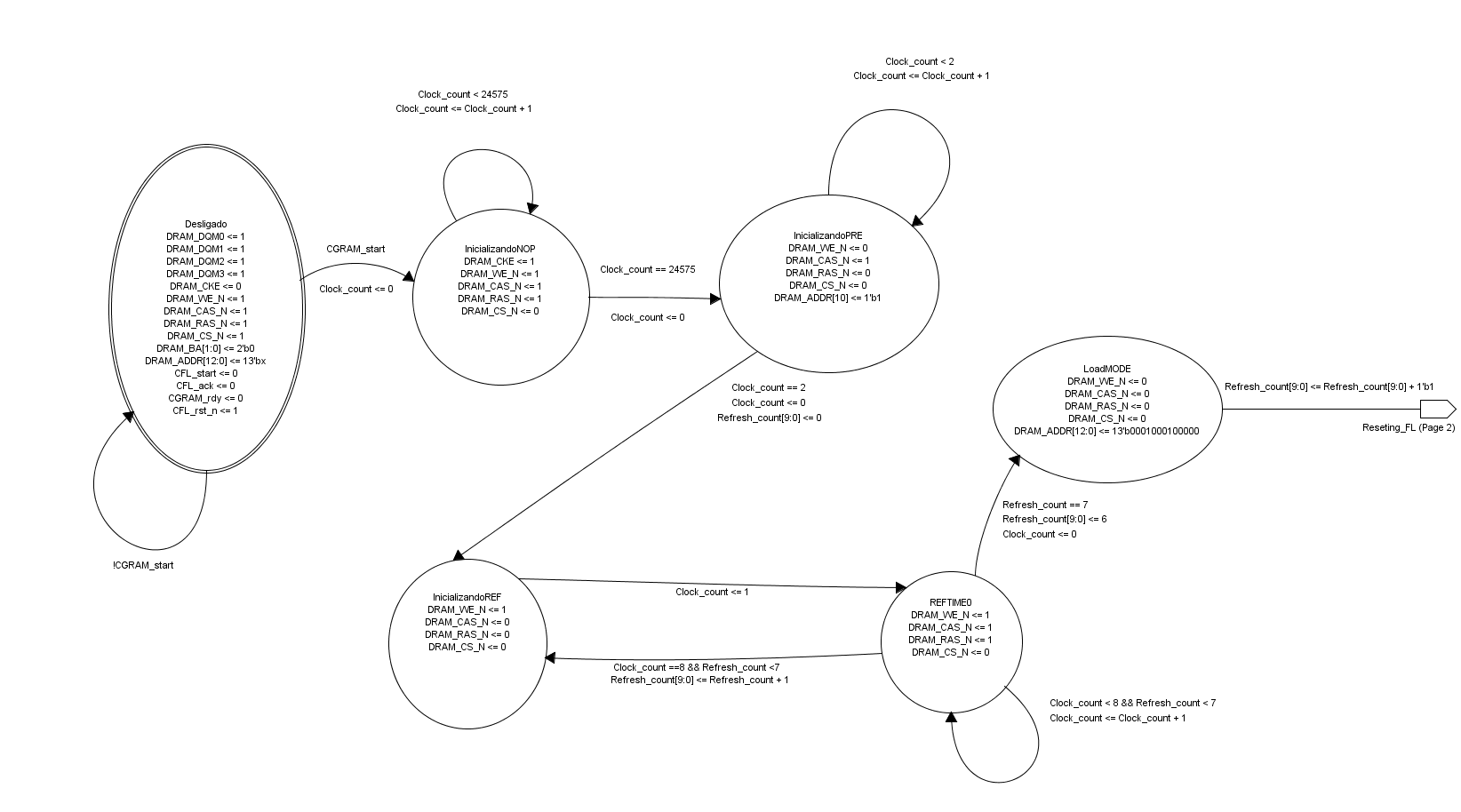
## Esquemático do Módulo de Interface VGA



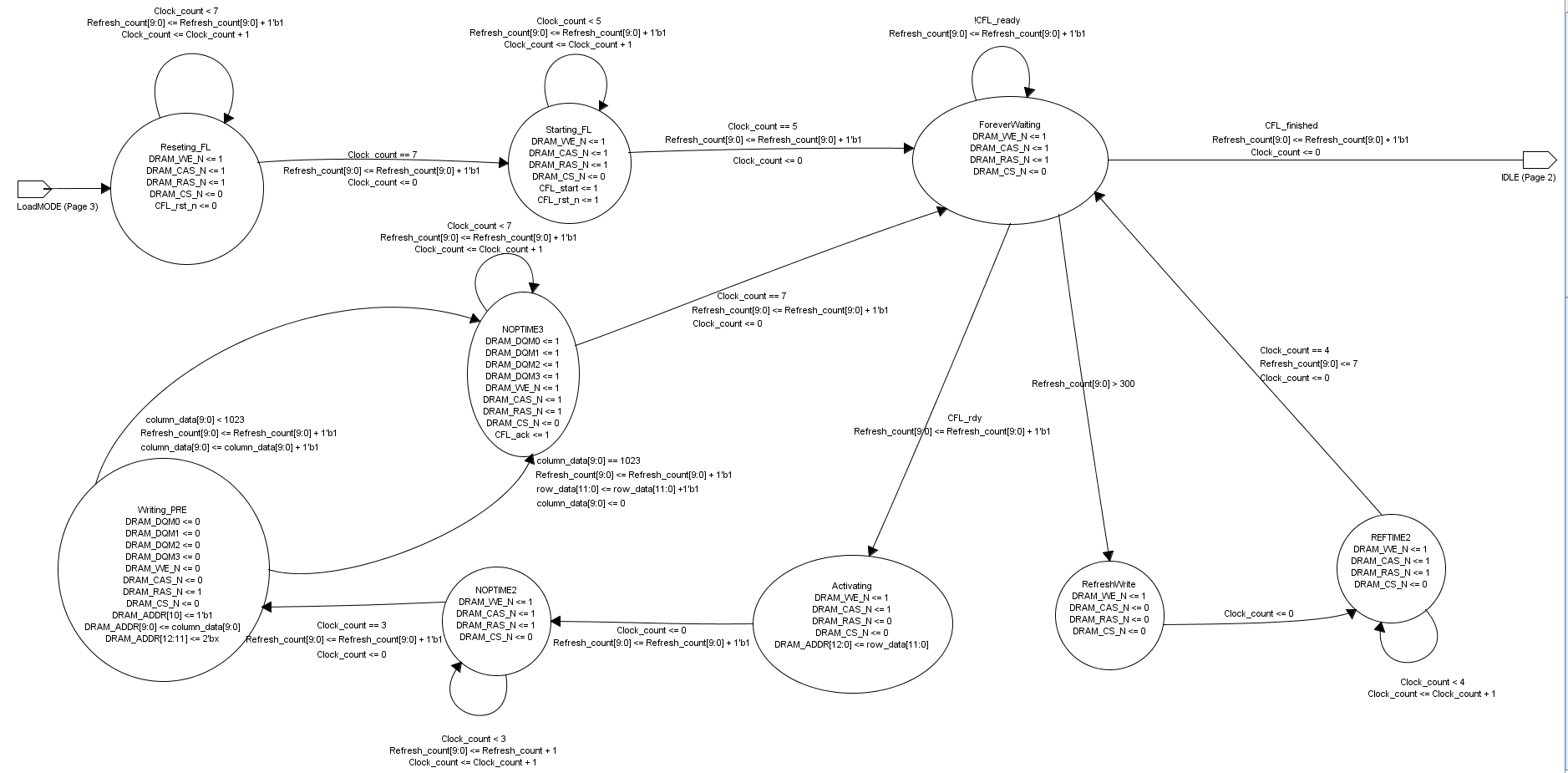
## Máquina de Estados do controlador da Flash



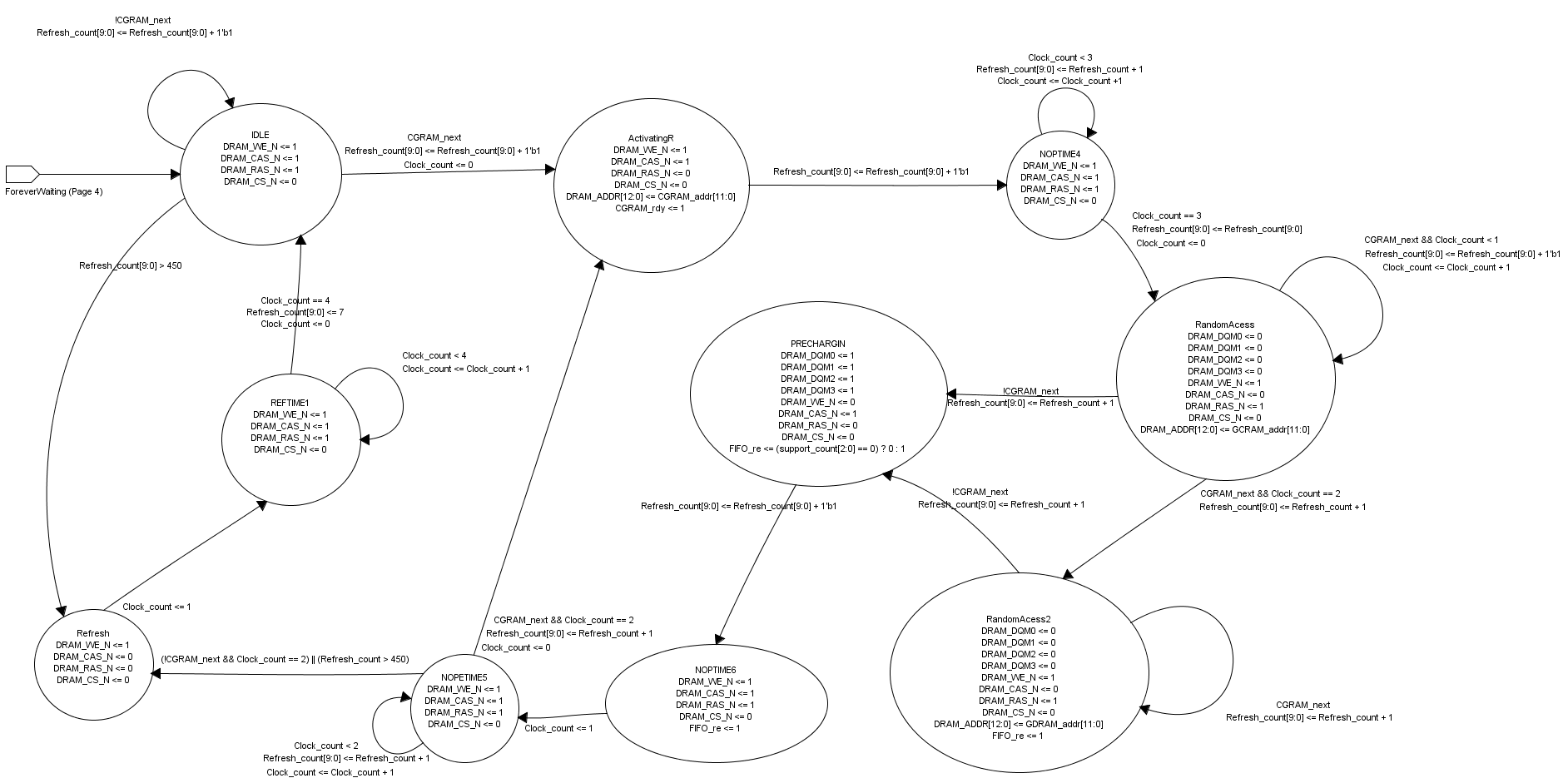
## Máquina de Estados da inicialização da SDRAM



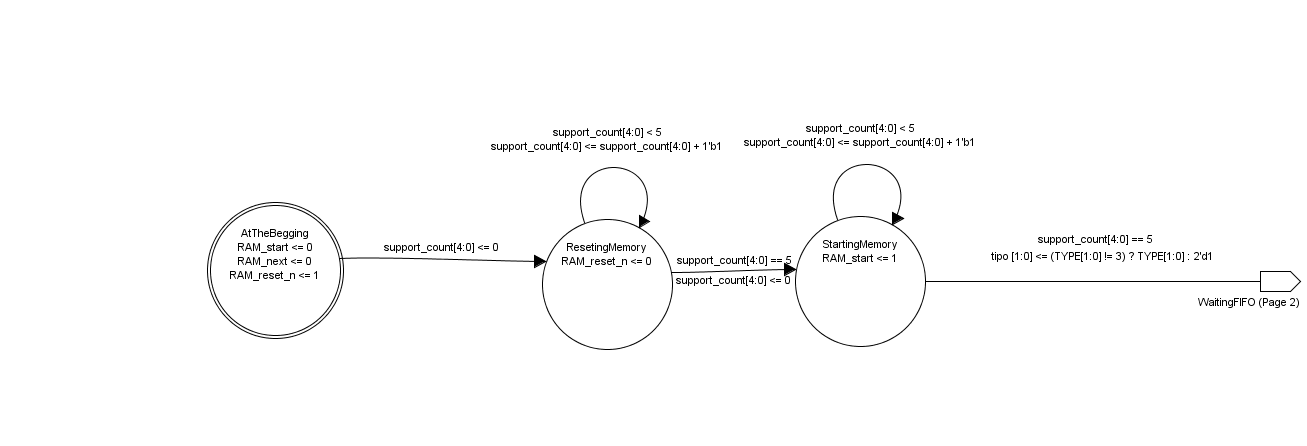
## Máquina de Estados da escrita da SDRAM



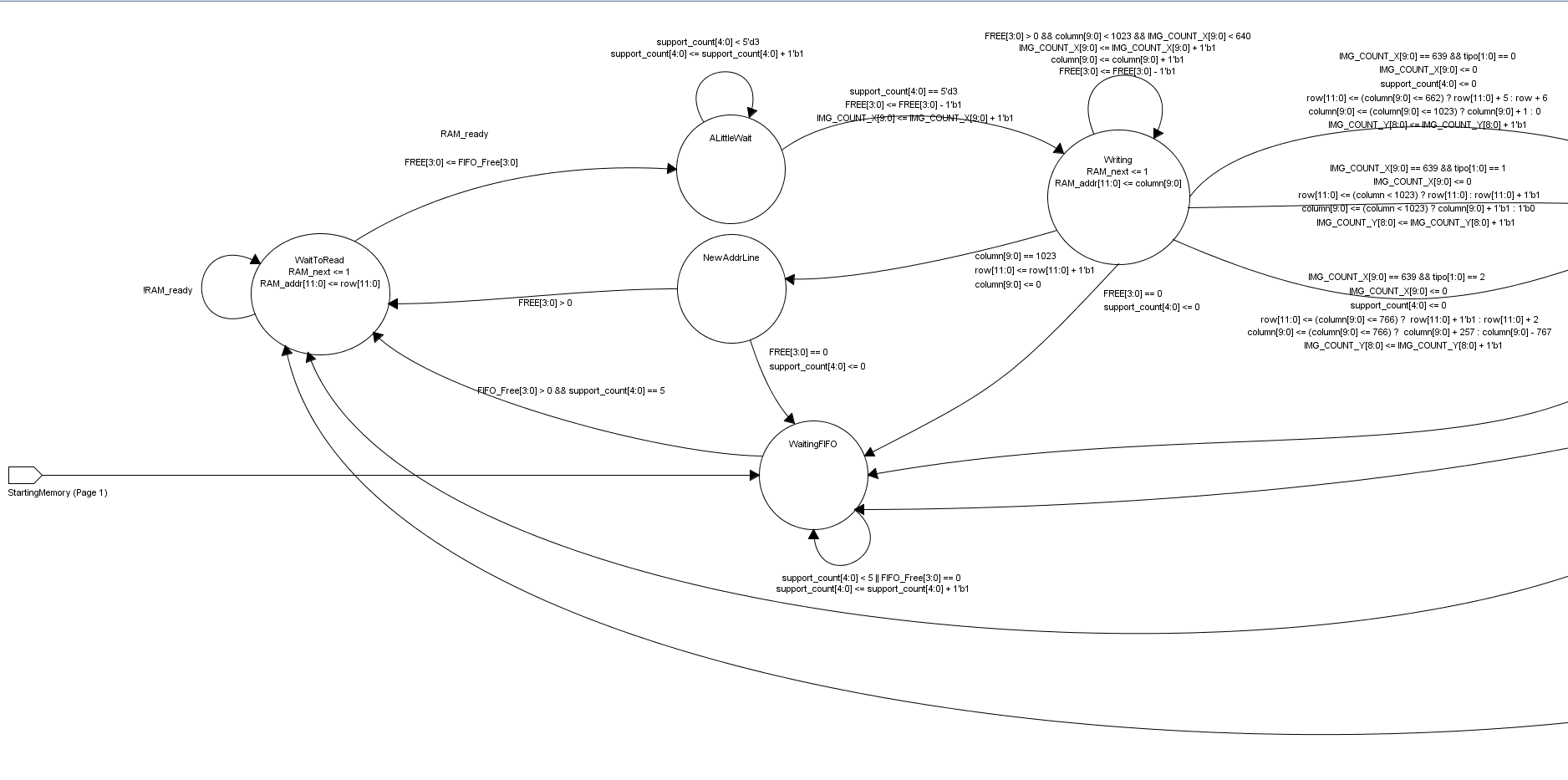
## Máquina de Estados da leitura da SDRAM

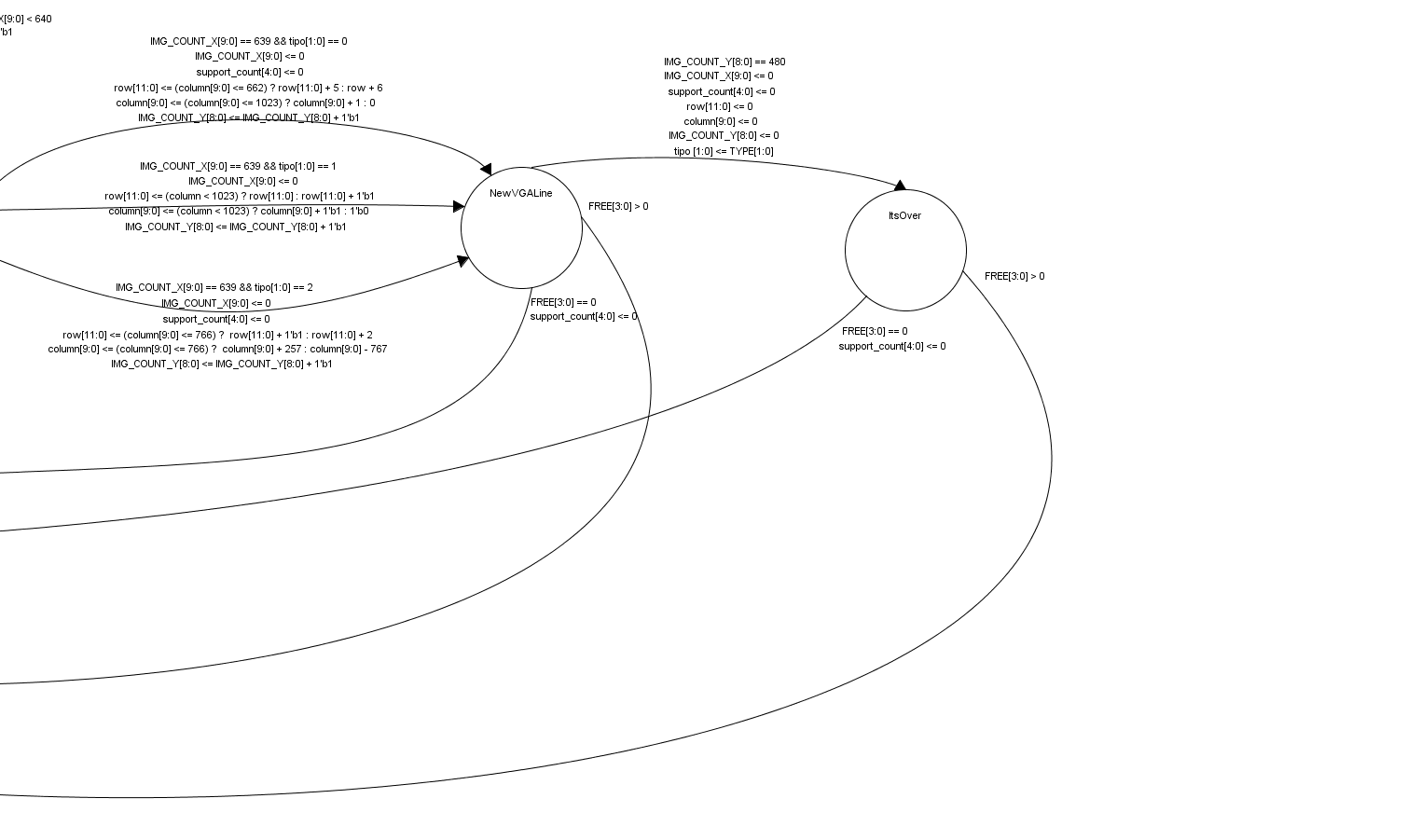


## Início da máquina de estados do controlador de imagem



## Continuação do controlador de imagem





1. [↑](#footnote-ref-1)