FABRÍCIO RIBEIRO TOLOCZKO
THIAGO HENRIQUE MILAN LUQUETA

Desenvolvimento de uma GPU aberta

Trabalho de Conclusão de Curso apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do título de Bacharel em Engenharia

São Paulo

2016

FABRÍCIO RIBEIRO TOLOCZKO
THIAGO HENRIQUE MILAN LUQUETA

Desenvolvimento de uma GPU aberta

Trabalho de Conclusão de Curso apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do título de Bacharel em Engenharia

Orientador: Prof. Dr. Marcelo Knörich Zuffo

São Paulo
2016

Uma imagem vale mais que mil palavras
ou algumas chamadas em OpenGL

RESUMO

Desenvolvimento de um processador gráfico (GPU) sintetizável em FPGA, sendo inteiramente de código aberto. O projeto busca elaborar uma arquitetura escalável, capaz de implementar o *pipeline* gráfico do OpenGL. Deverá ser uma base de desenvolvimento sólida para implementação de outros sistemas computacionais ou especificações de processamento de dados no futuro.

Palavras-Chave: Computação gráfica. Sistemas embutidos. Arquitetura e organização de Computadores.

ABSTRACT

Development of a graphics processor synthesizable on FPGA, being entirely open source. The project seeks to develop a scalable architecture, capable of implementing the OpenGL graphics pipeline. It must be a solid development base for the implementation of other computer systems or data processing specifications in future.

Keywords: computer graphics. embedded systems. organization and architecture of computers.

LISTA DE ABREVIATURA E SIGLAS

API	*Application Programming Interface*
ASIC	*Application Specific Integrated Circuits*
CAD	*Computer Aided Design*
CPU	*Central Processing Unit*
FPGA	*Field Programmable Gate Array*
GNU	*GNU is Not Unix*
GPU	*Graphics Processing Unit*
HDL	*Hardware Description Language*
ISA	*Instruction Set Architecture*
OpenGL	*Open Graphics Library*
PHP	*PHP: Hypertext Preprocessor*
SBC	*Single Board Computer*
SoC	*System on Chip*
USB	*Universal Serial Bus*
Apache 2.0	*Apache License version 2.0*
BSD-2	*The BSD 2-Clause License*
GPL 3.0	*GNU General Public License, version 3*
MIT	*The MIT License*
MPL 2.0	*Mozilla Public License, version 2.0*
CDDL 1.0	*Common Development and Distribution License*
EPL 1.0	*Eclipse Public License 1.0*
PD	*Public Domain*

SUMÁRIO

1. DEFINIÇÃO DO PROBLEMA

As Unidades de Processamento Gráfico (em inglês GPU *Graphics Processing Unit*) são processadores de propósito específico desenvolvidos para operações de processamento gráfico, caracterizado pelo tratamento de um grande volume de dados, que precisam ser exibidos na tela do computador em taxas interativas. Esta necessidade de processamento gráfico é essencial para utilização em jogos, programas de desenvolvimento e edição de imagens e vídeos, exames médicos computadorizados, ferramentas CAD (*Computer Aided Design*), entre outros. As GPUs são amplamente utilizadas em computadores pessoais e portáteis, tais como smartphones e tablets. Pelo fato de tratarem de algoritmos paralelizáveis muito melhor que CPUs (*Central Processing Unit*), as GPUs têm sido utilizadas para outras finalidades no campo da ciência: supercomputadores, processamento de sinais, estudos estatísticos, entre outras simulações científicas.

Os GPUs são uma etapa fundamental no *hardware* moderno: o processamento gráfico está presente em quase todos os segmentos que utilizam computação. Tais processadores são, porém, produzidos por algumas poucas empresas, que dominam praticamente toda tecnologia e todo mercado de GPUs, tanto com *desktops* quanto embarcados(JPR, 2016). Com isso, a maior parte da tecnologia envolvida — desde o hardware ao software das bibliotecas de interface — não está acessível ao domínio público, fazendo com que a utilização e desenvolvimento das GPUs fiquem restritos aos produtos vendidos pelos fabricantes e suas plataformas específicas.

2. ESPECIFICAÇÃO DA NECESSIDADE

É fundamental para o mercado que haja opções de desenvolvimento com código aberto. Um código aberto significa que todo o projeto, desde sua concepção até o produto final, está acessível a qualquer terceiro que deseja utilizá-lo. Isso exige que certos conceitos legais sejam atrelados ao projeto, conforme os padrões existentes. A *Open Source Initiative*(OSI,2016) é a fundação que administra as licenças dos softwares abertos.

O conhecimento disponibilizado em código aberto já demonstrou seu potencial de crescimento tecnológico. Expoentes dessa mentalidade no passado, como o API (*Application Programming Interface*) OpenGL, GNU\Linux ou o projeto software livre, são hoje base de mercados extremamente lucrativos e promissores — Facebook, Twitter, Whatsapp, Android. A escassez de hardware aberto como CPUs, memórias, buses, pipelines e, principalmente, GPUs é mais uma barreira ao desenvolvimento da humanidade hoje. Nesse sentido, a realidade apontada pelo professor Karu Sankaralingam — principal autor do projeto MIAOW, da University of Wisconsin-Madison — revela o sentido da proposta de um GPU aberto(SANKARALINGAM,2015): "(...)RTL-level implementations and low-level detailed microarchitecture specification is lacking", em tradução livre: "implementações em nível RTL e especificação de microarquitetura detalhada em baixo nivel são escassas".

Isso gera a necessidade de criação de novas arquiteturas capazes de concorrer no mercado com maior flexibilidade, a partir de uma tecnologia acessível e aberta. Por essa perspectiva, é preciso que se tenha uma tecnologia escalável, capaz de fornecer resultados confiáveis e com uma estrutura de desenvolvimento simples e robusta — a partir da qual poderá surgir, no futuro, uma GPU comparável em qualidade e tamanho com as já presentes no mercado.

Com isso, foi motivada a realização deste projeto: uma arquitetura GPU aberta, com as características citadas acima. O diferencial deste projeto é que ele pretende ser funcional desde o início, trabalhando gradativamente para se aperfeiçoar e

implementar mais funções conforme avança. Essa implementação gradativa permitiria que, mesmo com a quantidade limitada de mão-de-obra e tempo disponível, tenha-se uma arquitetura que já possui algumas funcionalidades e com um caminho claro e simples para continuar se desenvolvendo.

3. REVISÃO DA LITERATURA

Existem projetos de *hardware* aberto ou ferramentas abertas com enfoque em GPUs, muitos inclusive ainda em desenvolvimento. Alguns deles: MIAOW, Nyuzi, Theia GPU, GPL-GPU. Todos esses projetos implementam parcialmente o que é uma GPU comercial hoje, faltando módulos para gráficos ou funções específicas, o que os torna inviáveis a serem produzidos efetivamente em silício. Embora essa circunstância, cada um definiu bases importantes para que um novo projeto os agrupe e desenvolva integralmente um processador de gráficos, capaz de competir no cenário industrial do futuro.

O custo de desenvolvimento é grande, exigindo dedicação e tempo — a fim de ilustrar, o projeto MIAOW necessitou de 12 meses com 7 pessoas trabalhando para conclusão do *design* inicial. Foi expandido posteriormente por outras 4 pessoas, das quais 3 eram de graduação, resultando em um total de 36 meses para alcançar o estágio atual.

As informações sobre arquiteturas de GPU disponíveis demonstram que a complexidade de um projeto deste porte é grande. Além disso, como a maior parte das tecnologias mais modernas estão sob segredo industrial, a pesquisa se torna relativamente limitada. A maior parte dos estudos fica confinada ao funcionamento teórico ou aos demais projetos, como MIAOW e Nyuzi, que estão em um âmbito mais experimental.

4. TECNOLOGIAS RELEVANTES

4.1. TECNOLOGIAS DE PROCESSAMENTO GRÁFICO

Há algumas formas de viabilizar o processamento gráfico em sistemas computacionais que basicamente estão em duas concepções: software e hardware. Comumente, há uma mescla de ambos dependendo da aplicação.

4.2. IMPLEMENTAÇÃO PURAMENTE EM *SOFTWARE*

A primeira e mais simples é realizar a computação integralmente no CPU via software — modo bastante comum de atuar nos primórdios da computação, sobretudo quando a demanda por esse tipo de tarefa era reduzida em comparação ao cenário contemporâneo de aplicações. Hoje, essa abordagem ainda é útil em sistemas embarcados de pouca exigência visual ou que tenham recursos computacionais disponíveis para esse fim sem prejuízo dos demais[11]. Ela é, no entando, extremamente onerosa ao CPU, dado que ele costuma ser de uso geral — pouca ou nenhuma otimização para processamento tipo SIMD, *single instruction multiple data*, característico de operações em gráficos —, além de ser bastante requisitado para outras funções.

4.3. IMPLEMENTAÇÃO EM *HARDWARE* DEDICADO: ASIC

O modo tradicional de se implementar hardware específico atualmente é desenvolver o sistema em ASIC (*Application Specific Integrated Circuits*, ou Circuito Integrado de Aplicação Especifica), popularmente conhecido como *chip*. Nessa concepção, o projeto é desenvolvido com intuito de fabricar um dispositivo ou fornecer as informações necessárias para que seja integrado a um outro sistema maior, também implementado em ASIC. Essa é a forma comercial disponível atualmente ao se comprar uma placa de vídeo ou um celular, neste o GPU está embutido num chip com outros processadores e periféricos, naquele, o GPU é o circuito integrado propriamente dito.

4.4. IMPLEMENTAÇÃO EM *HARDWARE* DEDICADO: FPGA

O aumento da capacidade computacional dos FPGAs e da redução do seu custo tem viabilizado a implementação de GPUs(FYKSE, 2013), além de inúmeros outros sistemas, puramente ou parcialmente nessa tecnologia(WALLS, 2012). Estudos já mostraram que FPGAs são escolhas razoáveis na implementação de arquiteturas tipo GPU(KINGYENS, 2011).

As vantagens dessa tecnologia são o custo de desenvolvimento bastante reduzido e o tempo de verificação de resultados quase imediato — aumentando a agilidade do projeto. Sua versatilidade também é notável devido à maior margem de erro disponível se comparada ao ASIC. Por outro lado, tempos de atraso maiores, velocidade máxima de trabalho menor e limite de área disponível para computação são desvantagens em relação aos ASICs.

4.5. ESPECIFICAÇÕES DE GRÁFICOS: OPENGL

OpenGL é uma API (*Application Programming Interface*, do inglês, Interface de Programação de Aplicação). Dentro das especificações para gráficos existentes, o OpenGL é uma das mais importantes e cumpre papel fundamental na indústria desde sua primeira versão, lançada em 1992. É definido via *software*, embora esteja profundamente ligado ao *hardware*.

Sob perspectiva de implementação de *hardware*, a OpenGL é um conjunto de operações sobre GPU e CPU (parcialmente ou integralmente). É uma especificação que orienta a implementação em *hardware*, mas não a define rigidamente — cada um de seus blocos podem ser desenvolvidos no *software* de maneira totalmente diferente entre diversos fabricantes. Uma abordagem comum é entender OpenGL como um *pipeline* de alguns estágios programáveis e outros de funções fixas controladas por estados.

Um GPU comercial implementa todas as funções do *pipeline* gráfico ou parte delas — a depender da implementação em *hardware*.

É importante considerar uma especificação de gráficos como o OpenGL ao se desenvolver um GPU — dado que ela acaba definindo objetivos e facilitando o projeto, além de garantir que, se corretamente implementada, o GPU poderá ser integrado a diversos sistemas de *software* já existentes.

4.6. IMPLEMENTAÇÃO ABERTA DA OPENGL: MESA3D E GALLIUM

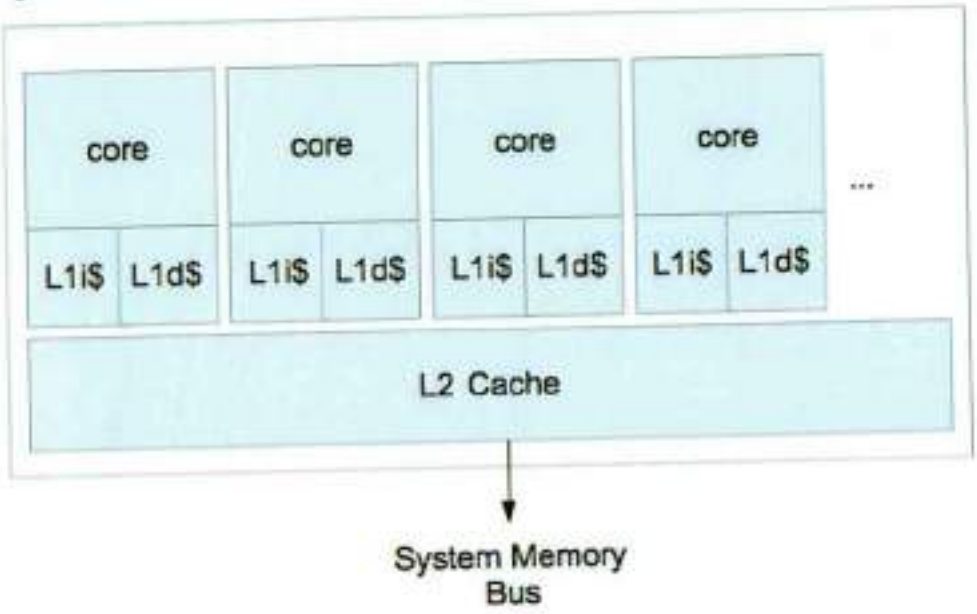
Mesa3D[16], comumente referido somente como Mesa, é um projeto de implementação de adaptadores gráficos abertos. Ele funciona como framework de adaptadores, enquadrando uma base abstrata comum entre diversos GPUs. Entre outras especificações, há drivers, ou adaptadores, disponíveis para implementar o pipeline gráfico através de OpenGL, Direct3D ou equivalente. Cada driver pode comunicar-se com hardware especifico: GPUs de fabricantes como NVIDIA, AMD ou Intel. Há adaptadores capazes de implementar o pipeline gráfico inteiramente ou parcialmente em software, isso é, rodando apenas no CPU.

O Gallium é também um framework, sendo parte do projeto Mesa. Sua proposta é abstrair ainda mais as funções gráficas, tornando o desenvolvimento de drivers para conectar a aplicação gráfica ao hardware uma tarefa mais simples e menos propensa a bugs. Nouveau(NOUVEAU, 2016) é um exemplo de driver gráfico desenvolvido sobre o Gallium que implementa grande parte do processamento gráfico necessário para GPUs da NVIDIA, usando de seu hardware da mesma forma que fazem os drivers proprietários desse fabricante. Atualmente, plataformas Linux utilizam quase exclusivamente o projeto Mesa e os seus drivers para garantir compatibilidade com hardware gráfico disponível comercialmente.

4.7. ARQUITETURAS DE GPU COMERCIAIS E ACADÊMICAS

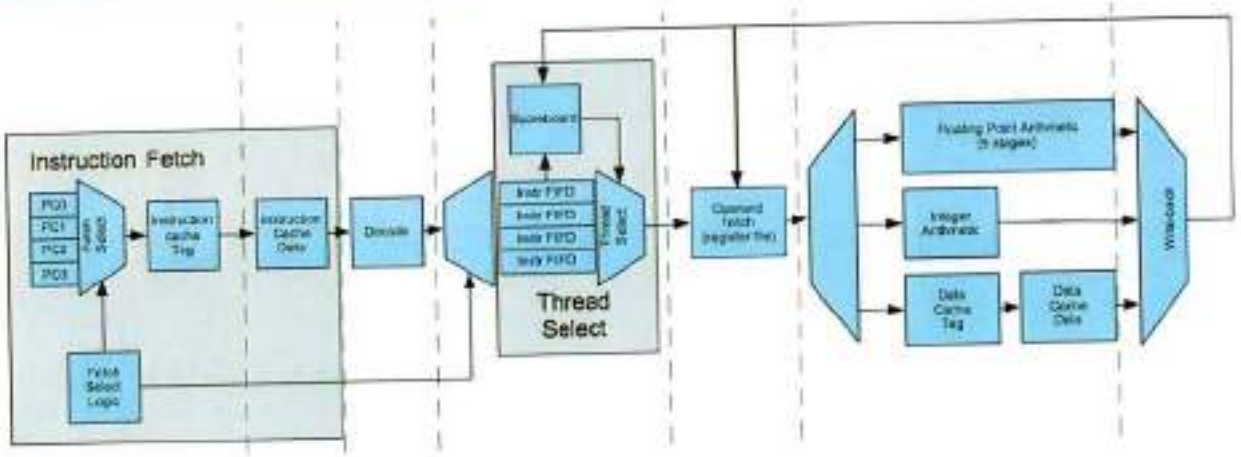
Há projetos relevantes no meio acadêmico atualmente. O Nyuzi(BUSH, 2015) apresenta uma microarquitetura baseada em vários pequenos núcleos de processamento, similar ao que se perseguido por fabricantes de modelos comerciais. A figura a seguir ilustra essa abordagem de *hardware*.

Figura 1 - Arquitetura geral de uma GPU



Fonte: NYUZI

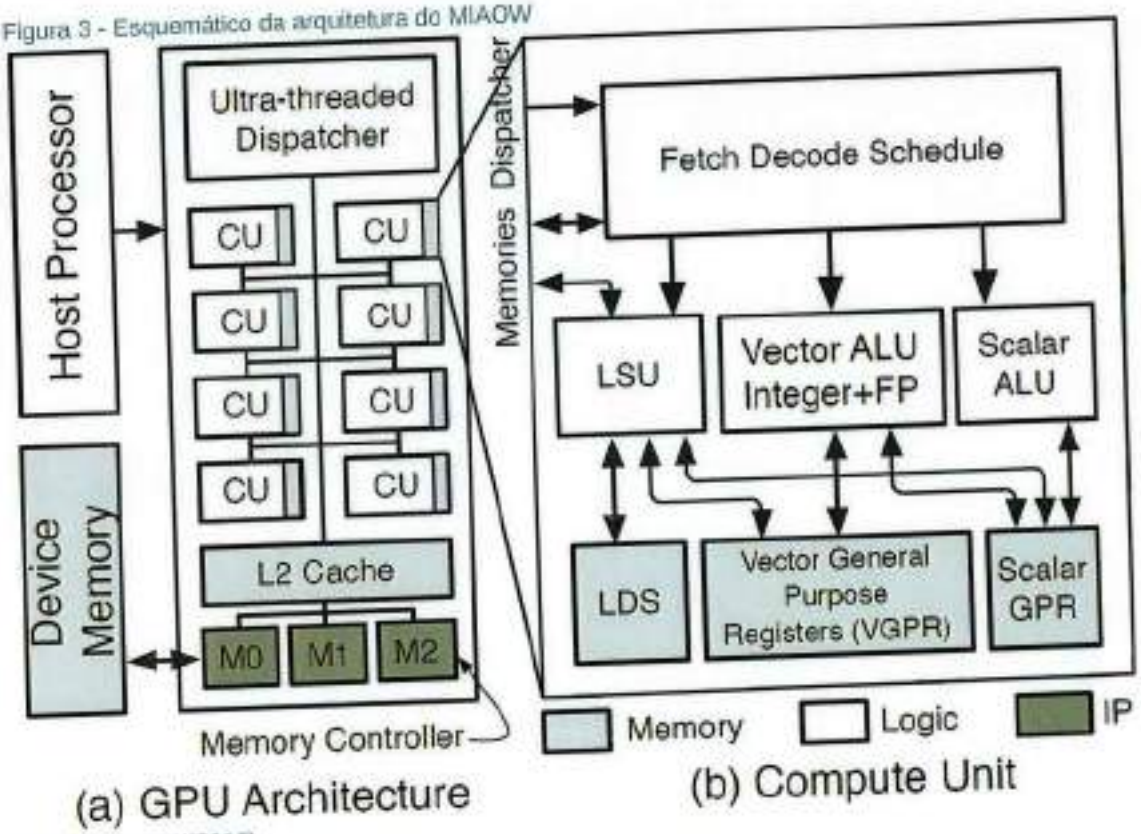
Figura 2 - Esquemático da arquitetura do Nyuzi



Fonte: NYUZI

Similar a um processador convencional (CPU) de poucos ou apenas um núcleo, o GPU segue uma perspectiva de vários núcleos de menor área e maior foco em processamento de grandes quantidades de dados. Na figura do *pipeline* de execução do Nyuzi, observamos da esquerda para a direita uma estrutura de busca de instruções, decodificação, seleção do contexto de processamento, acesso a registradores e execução da instrução.

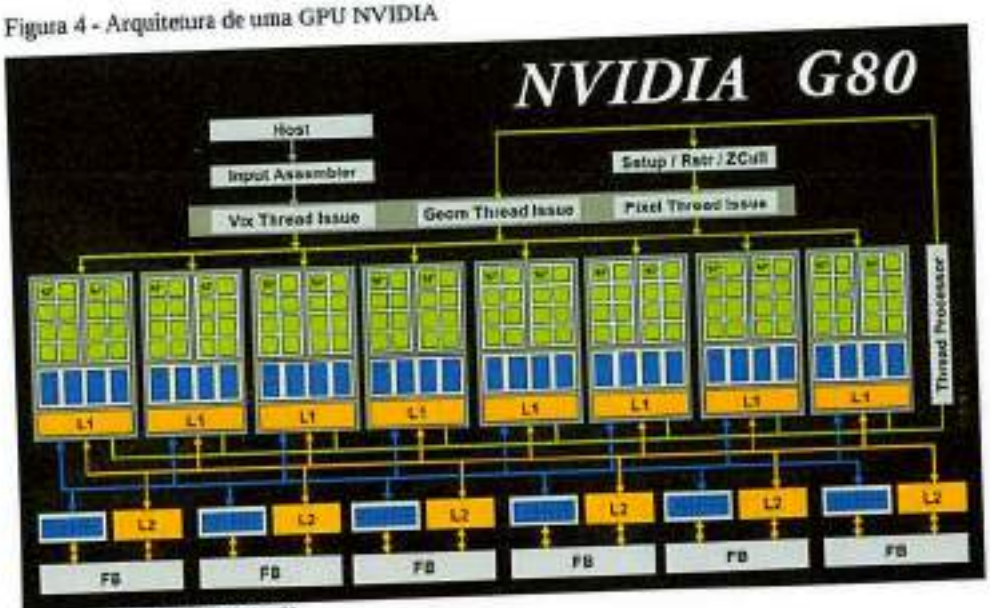
Figura 3 - Esquemático da arquitetura do MIAOW



Fonte: MIAOW(2016)

Na figura, observa-se a arquitetura do projeto MIAOW. À esquerda, o sistema global, contendo o gerente de tarefas no topo, as unidades de computação (CU) no centro e o *cache* associado ao controle de memória abaixo. À direita, observa-se o conteúdo de uma unidade de computação, em que há uma sequência de funções similar àquela citada para o Nyuzi.

Figura 4 - Arquitetura de uma GPU NVIDIA



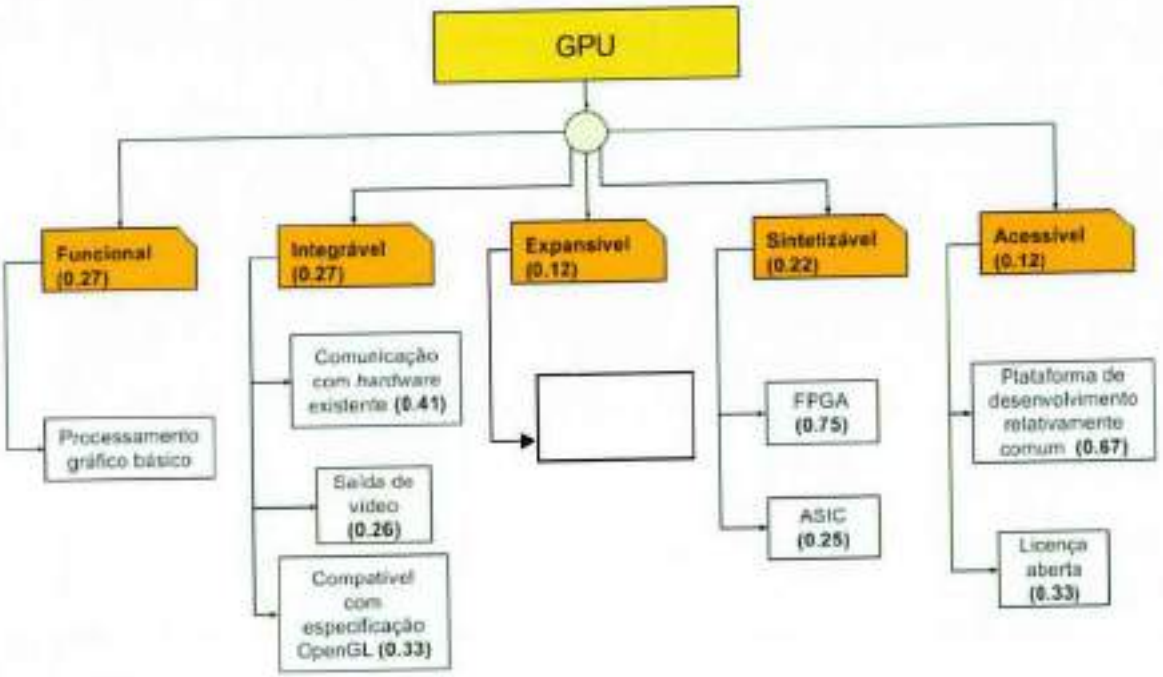
Fonte: BOLOTOFF(2010)

A imagem traz a arquitetura de um GPU NVIDIA. O *Input Assembler* decodifica a instrução do processador (*Host*) transfere para o *Vertex Thread Issue*, que gerencia tarefas relacionadas a processamento de vértices. Os *threads* são executados nos núcleos, armazenando dados temporariamente e criando novas tarefas com auxílio do *Thread Processor* que realoca novos *threads* nos núcleos através dos blocos *Geometry Thread Issue*, *Setup*, *Raster*, e *Pixel Thread Issue*. Através da coordenação adequada, os pixels são gerados e armazenados no *framebuffer* (FB), prontos para serem exibidos na tela num sistema convencional.

Por ser uma arquitetura comercial e voltada para o processamento gráfico, observamos que o GPU da NVIDIA contém uma estrutura diferenciada, além do *hardware* extra em relação aos outros dois projetos de dimensão acadêmica. A característica básica, no entanto, não se altera, que é a presença de vários núcleos coordenados executando tarefas diferentes através de concentrações e de níveis diferentes de memórias (*caches*).

5. ÁRVORE DE OBJETIVOS DO DISPOSITIVO

Figura 5 - Árvore de objetivos



Fonte: autoral

O objetivo desse projeto é o desenvolvimento e teste de uma arquitetura de GPU totalmente aberta (*open source hardware*). Essa nova arquitetura será compatível com padrões programação de interface *hardware* gráfico, como o OpenGL. Há a necessidade também da apresentação de uma arquitetura sólida e eficiente capaz de ser expandida no futuro — característica importante de um projeto aberto.

6. ESPECIFICAÇÃO DE REQUISITOS

Especificamente pela característica deste projeto existe uma primeira escolha, que é a da plataforma tecnológica de desenvolvimento da GPU Aberta. Com isso em mente, os primeiros requisitos de engenharia contemplam analisar se tais plataformas são capazes suprir os requisitos de marketing.

Após definidos todos os requisitos das plataformas de desenvolvimento, os subsequentes passos do projeto consistiam primariamente na familiarização com as ferramentas e na programação efetiva de toda estrutura da GPU, de maneira progressiva. Com isso, tem-se que os requisitos passam a contemplar o bom andamento desse processo.

Os requisitos para esse projeto, a partir deste passo, são dinâmicos e dependem da maturidade do projeto e das necessidades de desenvolvimento para cada determinado momento.

6.1. REQUISITOS PARA O SISTEMA

Requisitos de *marketing* levantados:

1. É funcional e executa tarefas básicas de processamento gráfico;
2. Fácil integração com sistemas existentes;
3. Permite pelo menos uma saída de vídeo totalmente funcional;
4. Suporta funções básicas da interface de programação OpenGL;
5. É desenvolvido de forma estruturada para que possa ser atualizado e aprimorado em versões futuras;
6. Possibilidade de síntese em ASIC;
7. Uso de plataformas de desenvolvimento acessíveis;
8. Distribuído sob licença aberta;

Requisitos *marketing*	Requisito de engenharia	Justificativa
1, 4, 5, 6	Possuir blocos estruturais funcionais que executam as funções de processamento	A divisão em blocos estruturais permite que a arquitetura possua uma boa escalabilidade funcional e que seja mais simples de desenvolver e depurar

1, 2, 3, 5	Comunica-se com CPU e memória através de interface apropriada	Para que a GPU seja compatível com as tecnologias atuais, é necessária a escolha de uma boa interface de comunicação (barramento)
1	Arquitetura otimizada para operar em SIMD (Single Instruction Multiple Data)	O funcionamento básico de uma GPU assim o exige. Com isso, a estrutura precisa ser otimizada para trabalhar com um fluxo grande de dados sofrendo a mesma operação
2, 3	A saída do sistema deve permitir a conexão com um sistema de vídeo (VGA, DVI, HDMI, etc.)	Para que o funcionamento do sistema seja confirmado e para atender ao requisito de *marketing* (3) é necessário que o projeto tenha conexão de vídeo
5, 8	Conformidade com os padrões de software aberto (*Open Source*)	O procedimento de conformidade com os padrões de software aberto permite que o projeto seja enquadrado legalmente como tal. Isso garante que terceiros possam utilizá-lo sem problemas e com poucas restrições
1,4	Implementar *pipeline* das especificações OpenGL	A fim de executar um *software* inteiramente utilizando o GPU e obter uma saída de vídeo, é importante que todas as etapas do *pipeline* gráfico estejam desenvolvidas e implementadas seguindo as especificações OpenGL.
5, 7, 8	As ferramentas de desenvolvimento utilizadas precisam garantir que os códigos gerados sejam acessíveis	Os códigos gerados não podem ficar restritos a plataformas específicas ou em formatos pouco usuais. Isso ajuda na garantia de sua flexibilidade e acessibilidade, bem como evita problemas de licença aberta.

Tabela 1 - Tabela de mapeamento de requisitos de engenharia

6.2. RESTRIÇÕES

6.2.1. Restrições técnicas

A adequação a especificações como OpenGL determinam características de interface que restringem o funcionamento do sistema.

Além disso, a adequação com as licenças de software aberto precisa ser observada. Isso é considerado uma restrição — ainda que tenha como objetivo eliminar empecilhos legais que poderiam dificultar terceiros de se beneficiarem dos resultados do projeto.

6.2.2. Restrições econômicas

Existem limitações quanto à capacidade computacional disponível e o custo nas FPGAs atuais. Nesse sentido, a necessidade de utilização de placas de desenvolvimento muito caras, devido à grande capacidade de área e processamento exigida nesse trabalho, pode se tornar uma grande restrição ao projeto.

Deve-se levar em consideração também o custo dos softwares de desenvolvimento. Utilizar os softwares para os quais a universidade já possui licença de uso é uma solução para esse problema, bem como utilizar ferramentas gratuitas.

6.2.3. Restrições de manufaturabilidade

O escopo do projeto não é desenvolver um circuito integrado físico. Ainda assim a necessidade de ser sintetizável em ASIC precisa ser levada em conta. A arquitetura não pode fazer com que a produção de um circuito integrado de aplicação especifica seja inviável.

7. PROJETO

O primeiro passo do projeto foi definir a estratégia para resolvermos o problema bem como a plataforma que vamos utilizar.

Entre as possibilidades de projetar um ASIC ou uma FPGA, pretendemos utilizar a segunda abordagem. Ela é mais adequada para atendimento do requisito de código aberto e do requisito de escalabilidade funcional, ou seja, gradativamente pretendemos incorporar novas funções à esta GPU — importante notar a diferença entre escalabilidade funcional, relativo ao desenvolvimento do projeto, e escalabilidade de desempenho, relativo à performance do sistema final.

Em se tratando do desenvolvimento em FPGA, poderíamos ter escolhido trabalhar em cima de arquiteturas abertas já prontas (como o MIAOW), tornando-as graficamente funcionais, ou também construir os blocos funcionais de *hardware* a partir de uma implementação em software, ou ainda começar todo o desenvolvimento desde o início.

Partimos para a segunda opção, visto que permite que a arquitetura apresente uma saída de vídeo funcional já no início, é necessariamente realizado em adequação com as funções do OpenGL, possui uma sequência de desenvolvimento mais clara e aparenta ser mais simples.

Optou-se pela Mesa3D como implementação OpenGL pelos seguintes motivos:

- Projeto maduro, largamente testado;
- Extensa documentação e bom suporte da comunidade de usuários;
- Código aberto, inclusive dos seus *drivers*. Excelente referência para implementação de *hardware*;
- Pode usar parcialmente o CPU e parcialmente o GPU, dependendo do *driver* apenas. Isso cria escalabilidade funcional no desenvolvimento do projeto.

Optou-se por implementar a parte equivalente ao rasterizador na softpipe na primeira fase do projeto, que contemplará todo o curso da disciplina de Projeto de Formatura (PSI2594).

7.1. TABELA DE CONCEITOS

Conceitos:

A) Implementar diretamente numa placa de desenvolvimento com FPGA;

B) Utilizar uma placa de desenvolvimento FPGA com um processador integrado no qual rodará uma versão da Mesa3D através de um *driver* desenvolvido especificamente para o projeto;

C) Utilizar uma placa de desenvolvimento com um processador e um GPU comercial. O Mesa3D rodará nesse sistema através de um *driver* desenvolvido especificamente para o projeto. À essa placa, estará conectada uma FPGA, através de USB 3.0 ou outro sistema de comunicação, contendo o GPU projetado;

Tabela 1 - Tabela de conceitos

	Forças		Fraquezas	
A	• Permite maior complexidade • Memória gráfica é acessível	+ ++	• Ausência do Mesa3D • Alto custo	- - - - - -
B	• Baixo custo • Mesa3D garante escalabilidade funcional • Memória gráfica é acessível	+++ +++ ++	• Limita a complexidade	-
C	• Baixo custo • Mesa3D garante escalabilidade funcional	+++ +++	• Limita a complexidade • Possibilidade de acesso restrito à memória gráfica	- - -

Fonte: autoral

Com essa tabela, fica claro que o conceito B é a melhor escolha. O conceito C é uma escolha também muito razoável, perdendo para a segunda opção apenas por apresentar o risco de que o acesso à memória gráfica seja mais difícil (ou até

mesmo impossível), o que pode ocasionar menor performance e maior dificuldade em fazer o sistema funcionar.

7.2. NÍVEL 0

Figura 6 - Nível 0

Fonte: autoral

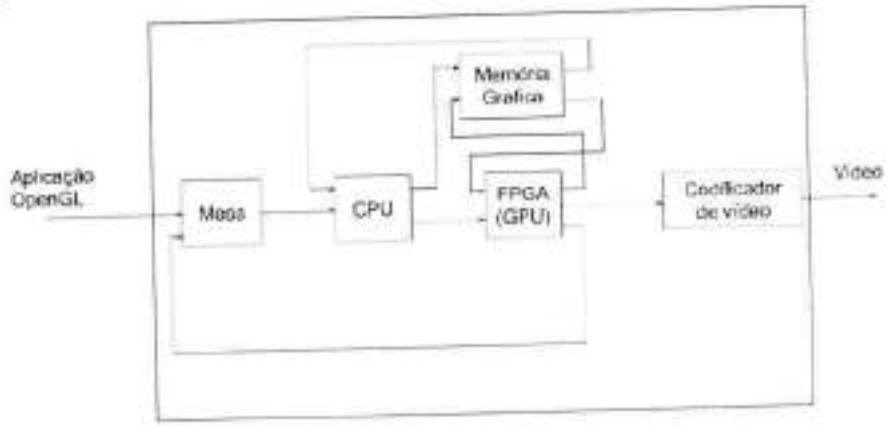
Tabela 2 - Tabela do nível 0

GPU aberta	
Entradas	Aplicação (ISA)
Saídas	Vídeo
Funcionalidades	Realiza as funções de processamento gráfico

Fonte: autoral

7.3. NÍVEL 1

Figura 7 - Nível 1



Fonte: autoral

Tabela 3 - Tabela do módulo Mesa do nível 1

Mesa	
Entradas	Aplicação OpenGL, Realimentação do Mesa
Saídas	InstruçõesCPU

Funcionalidades	Interpreta comandos advindos da aplicação e os converte para funções a serem executadas no CPU. Essas funções podem ou não passar instruções e dados para o GPU(implementado em FPGA).

Fonte: autoral

Tabela 4 - Tabela do módulo CPU do nivel 1

CPU	
Entradas	Barramento de memória gráfica, InstruçõesCPU
Saídas	Barramento de memória gráfica, InstruçõesGPU
Funcionalidades	Comanda todo o sistema através de programas, de suas e entradas e de suas saídas. De maneira direta, o CPU armazena ou lê informações na memória gráfica e instrui GPU implementado no FPGA. Mesa e aplicação em OpenGL são programas que rodam neste CPU.

Fonte: autoral

Tabela 5 - Tabela do módulo Memória gráfica do nivel 1

Memória gráfica	
Entradas	Barramento de memória
Saídas	Barramento de memória
Funcionalidades	Armazena dados relativos à execução do *pipeline* gráfico, seja via *software*, seja via *hardware*.

Fonte: autoral

Tabela 6 - Tabela do módulo FPGA do nivel 1

FPGA(GPU)	
Entradas	Barramento de memória, InstruçõesGPU
Saídas	Barramento de memória, Entrada do codificador de vídeo e Realimentação do Mesa
Funcionalidades	Implementa GPU desenvolvido neste projeto, seja parcial ou totalmente. A Realimentação do Mesa só é utilizada quando o *pipeline* gráfico estiver parcialmente implementado em *hardware*.

Fonte: autoral

Tabela 7 - Tabela do módulo Codificador de Vídeo do nivel 1

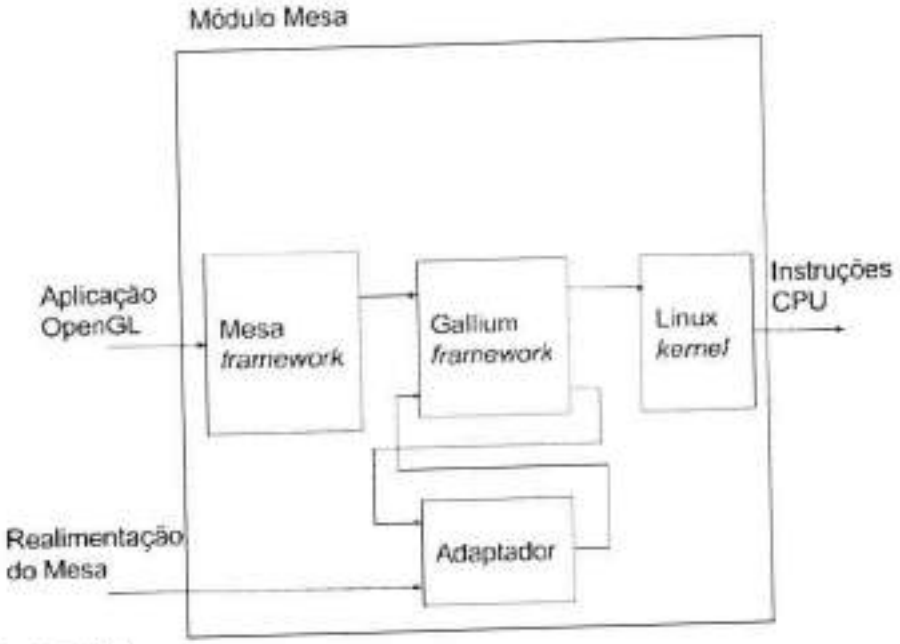
Codificador de	

vídeo	
Entradas	Saída de vídeo não codificada do GPU
Saídas	Vídeo codificado
Funcionalidades	Recebe região de memória gráfica(*framebuffer*) a ser exibida em vídeo e a codifica no padrão lógico e eletrônico adequado(e.g. VGA,HDMI).

Fonte: autoral

7.4. NÍVEL 2

Figura 8 - Módulo mesa no Nível 2



Fonte: autoral

Tabela 8 - Tabela do módulo Mesa Framework do nível 2 do bloco Mesa

Mesa *framework*	
Entradas	Aplicação OpenGL
Saídas	Abstrações Gallium
Funcionalidades	Interpreta comandos advindos da aplicação e os converte para funções de abstração do Gallium

Fonte: autoral

Tabela 9 - Tabela do módulo Gallium Framework do nível 2 do bloco Mesa

Gallium	

framework	
Entradas	Abstrações Gallium, Saída do Adaptador
Saídas	Comunicação com *kernel* do Linux — ou outro sistema operacional capaz de executar o Mesa e seus subsistemas.
Funcionalidades	Converte abstrações típicas do Gallium e converte em instruções e dados para o *kernel* do sistema operacional. O adaptador fornece instruções adequadas para a execução correta dessas funções de abstração.

Fonte: autoral

Tabela 10 - Tabela do módulo Adaptador do nível 2 do bloco Mesa

Adaptador	
Entradas	Saída de controle do Gallium, Realimentação do Mesa
Saídas	Saída do adaptador
Funcionalidades	A partir das abstrações do Gallium, implementa controle e execução do *pipeline* gráfico via *software* ou via *hardware*. Esta é a parte efetivamente modificável do *software* neste projeto.

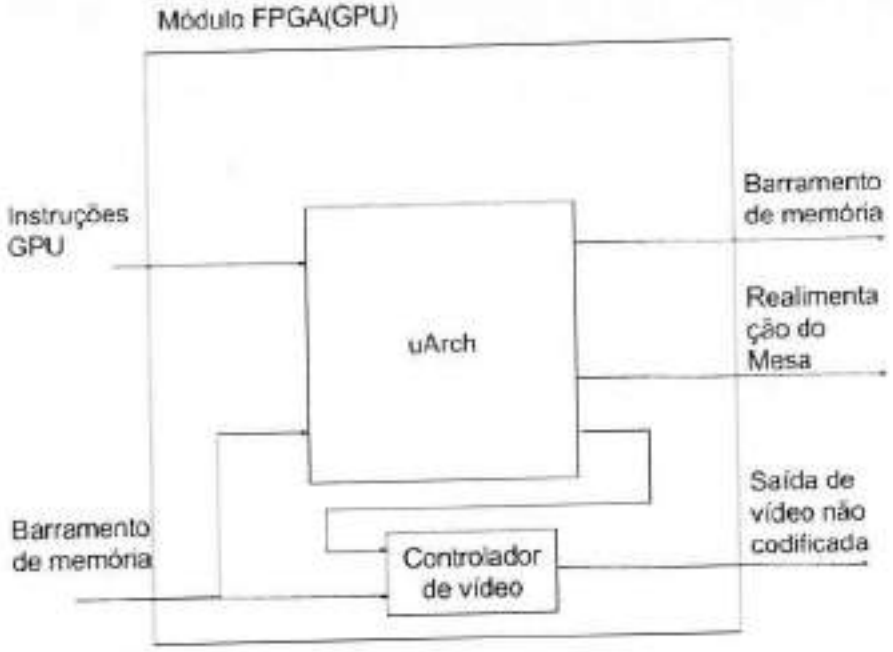
Fonte: autoral

Tabela 11 - Tabela do módulo Linux Kernel do nível 2 do bloco Mesa

Linux *kernel*	
Entradas	Comunicação com *kernel* do Linux — ou de sistema operacional capaz de executar o Mesa e seus subsistemas.
Saídas	InstruçõesCPU
Funcionalidades	Executa controle do *hardware* relacionado a gráficos e do próprio CPU, além de diversas funções relativas ao sistema total.

Fonte: autoral

Figura 9 - Módulo FPGA no Nível 2



Fonte: autoral

Tabela 12 - Tabela do módulo uArch do nível 2 do bloco FPGA

uArch	
Entradas	InstruçõesGPU, Barramento de memória
Saídas	Barramento de memória, Realimentação do Mesa, Controlador de Vídeo
Funcionalidades	Implementa efetivamente a microarquitetura do GPU desse projeto. Recebe instruções vindas do CPU, lê ou escreve em *buffers* na memória gráfica, controla sistema de vídeo e realimenta o Mesa quando necessário.

Fonte: autoral

Tabela 13 – Tabela do módulo Controlador de vídeo do nível 2 do bloco FPGA

Controlador de vídeo	
Entradas	Controlador de vídeo, Barramento de memória
Saídas	Saída de vídeo não codificada
Funcionalidades	Lê *framebuffer* presente na memória gráfica, recebe instruções da microarquitetura e prepara dados em sinais para serem codificados posteriormente.

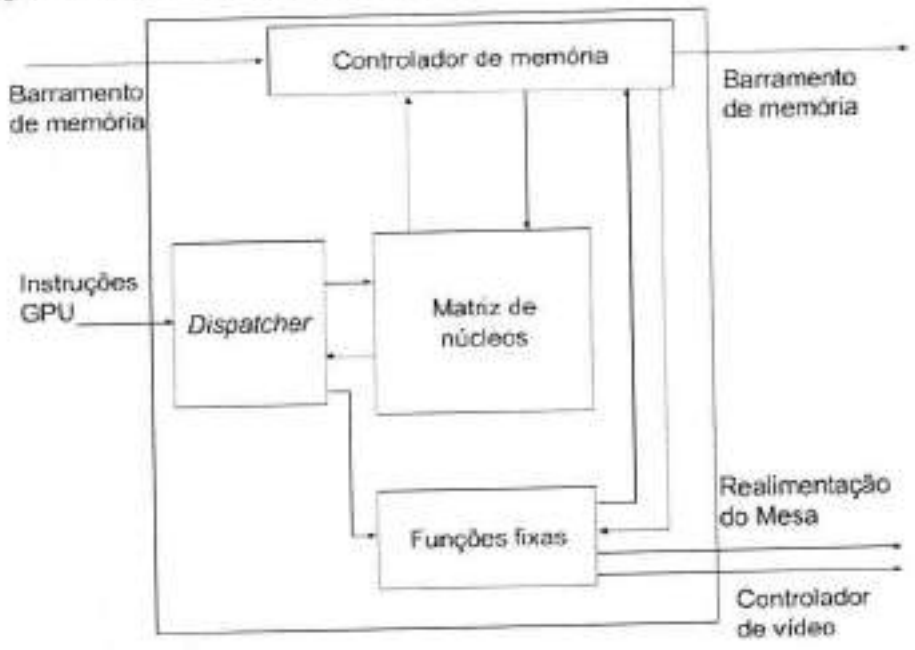
Fonte: autoral

Os módulos CPU, Memória Gráfica e Codificador de Vídeo não foram detalhados neste nível já que um nível de maior de abstração destes não é relevante ao projeto. O CPU executa programas definidos em outros módulos; a memória gráfica armazena dados de maneira organizada e acessível via barramento de memória; e o Codificador de vídeo é *hardware* presente nas placas de desenvolvimento escolhidas, recebendo apenas os sinais adequados do controlador de vídeo, implementados dentro do FPGA em sua malha lógica configurável.

7.5. NÍVEL 3

O módulo uArch é o único que necessita de um terceiro nível de detalhamento no projeto. Ele será constantemente revisado conforme o desenvolvimento do projeto e as necessidades encontradas. Abaixo, há um modelo inicial, baseado em arquiteturas de outros sistemas de GPU, sejam comerciais ou acadêmicos[2,6,20].

Figura 10 - Módulo uArch da FPGA no Nível 3



Fonte: autoral

Tabela 14 - Tabela do módulo Dispatcher do nível 3 do bloco uArch da FPGA

Dispatcher	
Entradas	InstruçõesGPU

Saídas	Barramento da matriz de núcleos, Barramento das funções fixas
Funcionalidades	Recebe instruções do CPU e distribui tarefas para núcleos na Matriz de Núcleos. Interpreta a necessidade de utilizar o bloco Funções Fixas.

Fonte: autoral

Tabela 15 - Tabela do módulo Controlador de Memória do nível 3 do bloco uArch da FPGA

Controlador de memória	
Entradas	Barramento de memória, Barramento de Saída da Matriz de Núcleos, Barramento de Saída de Funções Fixas
Saídas	Barramento de memória, Barramento da Matriz de Núcleos, Barramento de Funções Fixas
Funcionalidades	Media as requisições de leitura e escrita dos *buffers* na memória gráfica, assim como implementa *buffers* temporários de alta velocidade — conhecidos como *caches*.

Fonte: autoral

Tabela 16 - Tabela do módulo Matriz de núcleos do nível 3 do bloco uArch da FPGA

Matriz de Núcleos	
Entradas	Barramento de saída do Dispatcher, Barramento de saída do Controlador de Memória
Saídas	Barramento do *Dispatcher*, Barramento do Controlador de Memória
Funcionalidades	Contém núcleos que executarão os processos definidos pelo *Dispatcher*.

Fonte: autoral

Tabela 17 - Tabela do módulo Funções Fixas do nível 3 do bloco uArch da FPGA

Funções fixas	
Entradas	Barramento de saída do *Dispatcher*
Saídas	Barramento do Controlador de Memória, Realimentação do Mesa, Controlador de Vídeo
Funcionalidades	Interioriza funções de processamento fixo.

Fonte: autoral

7.6. ANÁLISE DE ACOPLAMENTO E COESÃO

Há um alto grau de acoplamento no nível 1, visto que a memória está ligada nos dois sentidos tanto à FPGA e à CPU, como também há uma realimentação da saída da FPGA ao módulo Mesa. Isso implica que falhas no acesso à memória ou problemas na arquitetura do bloco FPGA podem causar contratempos ao projeto dada a natureza mais difícil de depuração nesse tipo de sistema. A coesão no primeiro nível é relativamente elevada, o que implica maior facilidade no desenvolvimento e na respectiva procura de falhas.

Detalhando ainda mais o sistema, no nível 2 do módulo mesa, percebe-se novamente o impacto da realimentação, visto que o correto funcionamento do adaptador (ou *driver*) é essencial para que o Gallium *framework* funcione corretamente, além de todo o *hardware* associado. Devido à natureza do *software*, porém, isso não deve atrapalhar na detecção do erro, pois pode-se verificar quais chamadas de funções específicas falharam e facilmente delimitar o problema.

No nível 3 do módulo uArch da FPGA temos um sistema muito mais complexo e altamente maleável que é a "alma" do projeto. Existe forte acoplamento entre os módulos, além de possibilidade de realimentação em vários caminhos. O desenvolvimento baseado em metas, comparativos, divisão de tarefas, *testbenches*, *benchmarks* e revisões da arquitetura devem minimizar chances de falhas catastróficas tanto no sistema quanto no avanço do projeto. Embora esse cuidado, é importante reconhecer que, a partir deste nível de projeto, habitarão os maiores riscos de atrasos ou mesmo de não cumprimento de objetivos.

7.7. SELEÇÃO DOS DISPOSITIVOS

Em uma etapa anterior, fizemos um levantamento de placas de desenvolvimento FPGA SoC (*System on a Chip*) com CPUs integrados e que permitem a instalação de um sistema operacional (necessário para rodar o Mesa3D). Os dispositivos selecionados eram:

- Mpression Helio SoC Evaluation Kit da Macnica[21]
- DE-1 SoC Board da Terasic[22]

- DE2i-150 FPGA Development Kit da Terasic[23]
- Zynq-7000 All Programmable SoC ZC702 Evaluation Kit da Xilinx[24]
- Z-turn Board da Xilinx[25]

Tabela 18 - Tabela de valores de placas com FPGA SoC

Modelo	CPU	Número de kPLEs	VGA(V) HDMI(H)	Observação	Relação $/LUT
Z-turn Board	ARM A9	53.2	H		2.23
Mpression Helio View Kit	ARM A9	110	H	LCD incluso	5.68
DE1-SoC Board	ARM A9	85	V	$249 preço comum $175 preço acadêmico	2.05
DE2i-150 FPGA Development Kit	Intel Atom	150	H\V	$700 preço comum $614 preço acadêmico	4.09
Zynq-7000 All Programmable SoC ZC702 Evaluation Kit	ARM A9	53.2	H		16.82

Fonte: autoral

Dentre essas opções, foi adquirida a placa DE1-SoC da Terasic. Suas especificações técnicas, relevantes ao projeto, são:

- Cyclone V com ARM Cortex-A9 integrado
- 85000 PLEs
- 1GB de memória RAM DDR3 + 64MB de memória SDRAM na FPGA
- Saída VGA

7.8. EAP

Etapas do projeto:

1 - Preparar ambiente de desenvolvimento

Preparar ambiente de desenvolvimento em *software* e testar funcionamento básico do *hardware*;

2 - Compilar Mesa3D

Compilar Mesa3D para rodar na placa de desenvolvimento escolhida. Essa compilação exige o uso de um *cross compiler* que executa a compilação cruzada entre o sistema atual, que pode conter qualquer CPU, e o sistema de destino, que conterá um CPU específico — ARM, Intel ou equivalente, a depender da placa de desenvolvimento escolhida;

3 - Rodar Mesa3D

Preparar placa de desenvolvimento para rodar implementação Mesa3D totalmente via *software* no CPU e exibir imagem na saída de vídeo;

4 - Pesquisa e escolha de arquitetura

A pesquisa nesta etapa deve seguir critérios definidos durante o desenvolvimento do projeto. Isso porque a arquitetura pode variar de acordo com as restrições técnicas que surjam durante a implementação do *pipeline* gráfico em *hardware* típico[2] de uma GPU.

Esta etapa pode ser revisitada caso as etapas seguintes não tenham os seus objetivos atingidos adequadamente. De maneira geral, subprojetos análogos a este serão elaborados nesta fase e nas seguintes com objetivo de atender aos requisitos da etapa em questão e aos requisitos de engenharia e de *marketing* previstos no projeto principal.

4 - 1) Pesquisa de arquiteturas possíveis

Os requisitos iniciais serão basicamente atender à ideia de processamento paralelo, repetindo o que se observa em arquiteturas comerciais[1, 2] ou acadêmicas[6, 8, 9]. Essa condição inicial torna o processo de escolha abrangente e, portanto, sujeito a dificuldades na previsão da performance do sistema final. Isso deve ser minimizado conforme iteração devido ao *benchmarking* de módulos e de sistema. Outros requisitos possíveis na primeira iteração devem ser ponderados segundo observação das arquiteturas pesquisadas.

O único requisito fixo é a escalabilidade da arquitetura. Isso se deve ao requisito de *marketing* 5, em que o sistema "é desenvolvido de forma estruturada para que possa ser atualizado e aprimorado em versões futuras". Essa escalabilidade poderá estar ligada a *throughput* do sistema ou de blocos específicos do *pipeline* gráfico.

4 - 2) Escolher arquitetura

A arquitetura deve respeitar os critérios definidos anteriormente. Essa etapa exige a ponderação da equipe de desenvolvimento, tendo como resultado matrizes de decisão análogas às construídas neste documento.

5 - *Benchmarking* de arquitetura

5 - 1) Elaborar *benchmark* de arquitetura

O *benchmark* de arquitetura deve verificar a capacidade da arquitetura em produzir determinadas respostas. Aqui, deve-se produzir indicadores de performance tais como *throughput* (do inglês, significando vazão de dados) e uso de área ou de recursos. O objetivo é verificar as limitações da arquitetura escolhida após sua implementação e orientar, portanto, modificações futuras.

5 - 2) Implementar arquitetura escolhida

As etapas internas desta fase são diferentes das demais. Elas definirão um *loop* de desenvolvimento até que se construa toda arquitetura no adaptador, ainda que parcialmente em *software*. Isso se explica pela natureza complexa do *pipeline*

gráfico e a constante necessidade de modificar o módulo em desenvolvimento e seu *driver* a fim de que se conforme ao *testbench*.

A arquitetura atual deve ser implementada em HDL, isso é, linguagem descritora de *hardware*. Isso inclui a respectiva fase de testes e confirmação da funcionalidade da arquitetura. É importante notar que, a fim de seguir os requisitos de *marketing* e os de engenharia, funcionalidades previstas no *pipeline* gráfico podem ser parciais ou efetivamente não implementadas em *hardware* — mas sempre estarão presentes seja *hardware*, seja *software*, assim como garante a estrutura fornecida pelo Mesa e pelo Gallium.

5 – 2.1) Implementar módulo do *pipeline* gráfico

Faz-se necessário o desenvolvimento de um *driver* capaz de coordenar o *hardware* via *software*. Esse *driver* deve ser desenvolvido simultaneamente ao *hardware* em questão.
Implementar módulo do *pipeline* gráfico.

O *pipeline* gráfico é construído por diversos blocos associados[18]. Na versão 4.4 do OpenGL, são ao todo 9 blocos principais — cada um, composto por partes menores, arranjadas em *pipelines* elementares, representando funções dinâmicas ou fixas.

Estes módulos a serem desenvolvidos poderão ou não ser os blocos lógicos do *pipeline* gráfico. Há arquiteturas conhecidas[2,6] que implementam setores desse *pipeline* em módulos únicos ou dentro das unidades de processamento. Isso poderá servir de base para 'concatenar' o *hardware* a fim de que sejam aproveitadas repetições de processamento ou de lógica em partes diferentes do *pipeline*.

As funções do *pipeline* gráfico deverão ser implementadas via *software* através do adaptador quando o *hardware* for incapaz de executar tais tarefas, seja por ainda não terem sido implementadas, seja por não serem objetivo de implementação — excesso de complexidade em relação ao tempo disponível para desenvolvimento, por exemplo.

5 - 2.2) Implementar adaptador

O adaptador de *hardware* implementa as funções de controle do *hardware*, decidindo o fluxo de dados, bem como os estados da máquina comandada. O *pipeline* gráfico estará contido inteiramente nesse *driver*, segundo especificação OpenGL ou equivalente, e terá suas funções executadas via *software* ou via *hardware*. As funções implementadas no adaptador devem concordar com o módulo em desenvolvimento, daí a necessidade de desenvolvimento simultâneo.

5 – 2.3) Desenvolver *testbench* para módulo e seu adaptador

Testbench nessa etapa deve garantir o funcionamento do módulo de acordo com dados de entrada e saída, além de estados de máquina definidos. Para referência, o *driver softpipe*, presente no Gallium do projeto Mesa, fornecerá essas informações. Caso necessário, outros adaptadores poderão ser modificados a fim de garantir o funcionamento dos módulos desenvolvidos.

5 – 2.4) Testar módulo desenvolvido com *testbench*

Testbench deverá aumentar a confiabilidade no desenvolvimento do módulo atual a fim de que não haja erros que comprometam o sistema.

5 – 3) Aplicar *benchmark* a outras arquiteturas

Informações e dados sobre outras arquiteturas deverão estar disponíveis nesta etapa. Em caso negativo, o *benchmark* observará apenas as possibilidades de performance da arquitetura corrente, tais como capacidade de expansão, *throughput* ou outros que se mostrem relevantes para este comparativo. *Softwares* para implementação do *pipeline* gráfico estão disponíveis para uso, por exemplo o *llvmpipe*[19] do Gallium — o qual pode também servir de comparativo quanto à performance da arquitetura escolhida.

5 – 4) Comparar arquitetura usando *benchmark* de arquitetura

Terminada a fase de desenvolvimento da arquitetura corrente, ela deverá ser comparada a arquiteturas de GPUs comerciais ou acadêmicos e de *softwares* para implementação do *pipeline gráfico*.

6 - Testbench do Sistema

Esta etapa deve garantir o funcionamento do sistema completo de acordo com entradas e saídas relevantes — entra a aplicação gráfica por um lado, sai o vídeo exibido por outro.

Dada a limitação de recursos disponíveis no FPGA, é esperado que o sistema não possa ser implementado na sua totalidade em qualquer configuração de arquitetura. Por isso, deve ser escolhida uma versão da arquitetura que permita a avaliação da performance de um sistema completo. Caso isso não seja possível em nenhuma configuração, o sistema deverá conter o máximo de módulos de *hardware* mais relevantes quanto ao aumento de velocidade de processamento, deixando em *software* os demais.

6 – 1) Elaborar *testbench* de Sistema

Testbench de sistema deve garantir o funcionamento total de acordo com entradas e saídas esperadas. Esses dados podem vir de outros sistemas de referência.

6 – 2) Testar sistema com *testbench*

Esta etapa efetivamente fornece um sistema funcional, capaz de atender os requisitos de engenharia previamente afixados.

7 - Benchmarking de Sistema

O *benchmarking* de sistema justificará ou não as decisões tomadas durante a escolha de arquitetura. Isso poderá levar à revisão e reinicialização do processo de desenvolvimento da arquitetura e do sistema. Isso porque nesta fase, sistemas equivalentes servirão de referência a fim de cumprir os requisitos de engenharia — por sua vez, satisfazendo automaticamente aos requisitos de *marketing*.

7 – 1) Elaborar *benchmark* para sistema

O *benchmark* de sistema deve garantir os objetivos iniciais do projeto. A qualidade da análise deverá indicar também os caminhos a serem percorridos a fim de

melhorias na arquitetura e na escolha de sua configuração. *Benchmarks* comerciais ou acadêmicos poderão servir de base para este *benchmark*.

7 – 2) Aplicar benchmark a outros sistemas

O *benchmark* de sistema produzirá dados sobre sistemas equivalentes ao de desenvolvimento corrente. Essa equivalência poderá ser dependente da disponibilidade de tais sistemas de referência ou do acesso aos dados de cada fase do *pipeline* gráfico. Caso não haja sistemas abertos ou acessíveis, os sistemas fechados serão utilizados para esse propósito considerando apenas suas entradas e saídas ou ainda considerando a aplicação de *benchmarks* externos ao projeto — comerciais ou acadêmicos, a depender da necessidade.

7 – 3) Comparar sistema usando benchmark desenvolvido

A comparação entre o sistema atual deve produzir dados sobre a sua performance em relação aos sistemas de referência.

Tabela 19 - EAP do projeto

Etapa	Atividade	Deriverables/Checkpoints	Duração*	Pessoas	Recursos
1	Preparar ambiente de desenvolvimento	Manual da placa de desenvolvimento e executar pequenos exemplos da documentação	¼ semana	Thiago	PC, placa de desenvolvimento e monitor de vídeo
2	Compilar Mesa3D	Saída do *cross-compiler*	¼ semana	Fabrício	PC
3	Rodar Mesa3D	Verificar aplicações OpenGL sendo exibidas na saída de vídeo	½ semana	Fabrício e Thiago	PC, *software* de desenvolvimento, placa de desenvolvimento e monitor de vídeo
4	Pesquisa e escolha de arquitetura	Arquitetura intermediária			PC
4.1	Pesquisa de arquiteturas possíveis	Lista de arquiteturas que atendam aos requisitos correntes	1 semana e 1/2	Fabrício e Thiago	PC
4.2	Escolher arquitetura	Arquitetura intermediária	1/2 semana	Fabrício e Thiago	PC
5	*Benchmarking* de arquitetura	Resultado do *benchmark* de arquitetura			

5.1	Elaborar *benchmark* de arquitetura	*Benchmark* de arquitetura	1 semana	Fabrício e Thiago	PC
5.2	Implementar arquitetura escolhida	Módulos em HDL e adaptador correspondente	9 semanas	Fabrício e Thiago	PC e ambiente de desenvolvimento
5.2.1	Implementar módulo do *pipeline* gráfico	Módulo em HDL	-	Fabrício e Thiago	PC
5.2.2	Implementar adaptador	Complemento do adaptador (*driver*) relativo ao módulo	-	Fabrício e Thiago	PC
5.2.3	Desenvolver *testbench* para módulo e seu adaptador	*Testbench* do módulo e do complemento do adaptador	-	Fabrício e Thiago	PC
5.2.4	Testar módulo desenvolvido com *testbench*	Módulo funcional	-	Fabrício e Thiago	PC
5.3	Aplicar *benchmark* a outras arquiteturas	Listagem da performance das arquiteturas de referência	¼ semana	Thiago	PC
5.4	Comparar arquitetura usando *benchmark* de arquitetura	Performance do sistema desenvolvido em relação aos sistemas de referência	¼ semana	Thiago	PC
6	*Testbench* do sistema	Sistema em HDL e adaptador			PC e ambiente de desenvolvimento
6.1	Elaborar *testbench* do sistema	*Testbench* do sistema e do adaptador	1 semana	Fabrício	PC
6.2	Testar sistema com *testbench*	Sistema funcional	1 semana	Fabrício	PC e ambiente de desenvolvimento
7	*Benchmarking* do sistema	Comparação da performance do sistema com sistemas de referência			PC
7.1	Elaborar *benchmark* para sistema	*Benchmark* de sistema	½ semana	Thiago	PC e ambiente de desenvolvimento
7.2	Aplicar *benchmark* a outros sistemas	Listagem da performance dos sistemas de referência	½ semana	Thiago	PC e ambiente de desenvolvimento
7.3	Comparar sistema usando *benchmark*	Comparação da performance do sistema com sistemas de referência	½ semana	Thiago	PC e ambiente de desenvolvimento

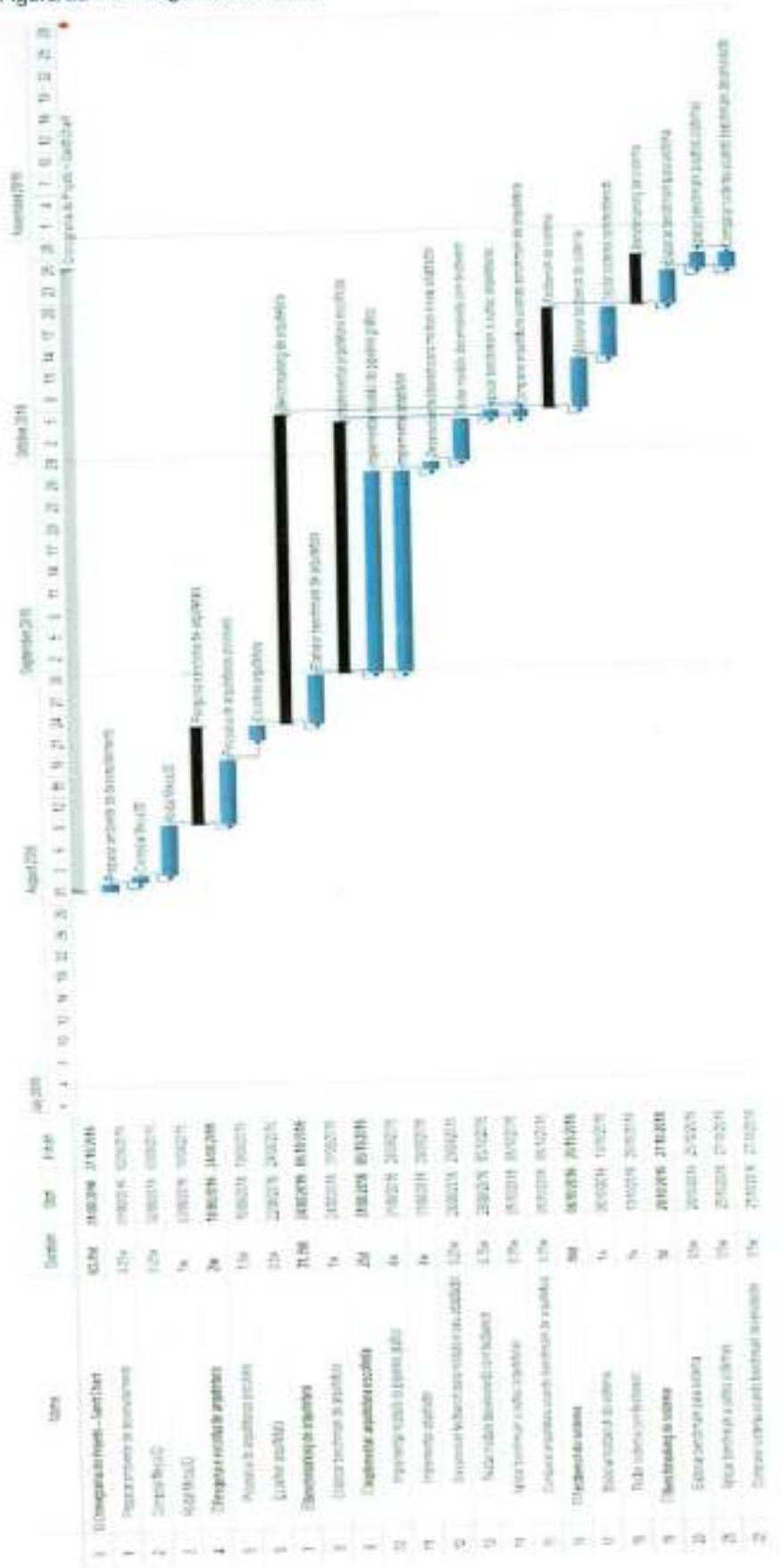
	desenvolvido				

Fonte: autoral

*Duração baseada na carga horária relativa a créditos-trabalho da disciplina Projeto de Formatura II. Sendo 4 créditos-trabalho, semanalmente, são exigidos aproximadamente 7 horas/semana de aluno em dedicação ao projeto fora da sala de aula

7.9. CRONOGRAMA (CARTA DE GANTT)

Figura 11 - Cronograma de Gantt



Fonte: autoral

A observação do cronograma de Gantt leva-nos à noção de redução no tempo total esperado. Se as etapas fossem executadas de forma dependente com relação à anterior, o projeto ocuparia cerca de 4 meses, que é o máximo disponível para disciplina Projeto de Formatura II. Por outro lado, ao lembrar que algumas etapas podem ser realizadas simultaneamente pelos membros da equipe de desenvolvimento, o tempo total de projeto passa a ser de cerca de 3 meses. Isso é benéfico, pois aumenta o tempo disponível para etapas fundamentais como a de desenvolvimento dos módulos e adaptador ou de pesquisa e decisão por arquitetura.

7.10. ATIVIDADES CRÍTICAS E ANÁLISE DE RISCO

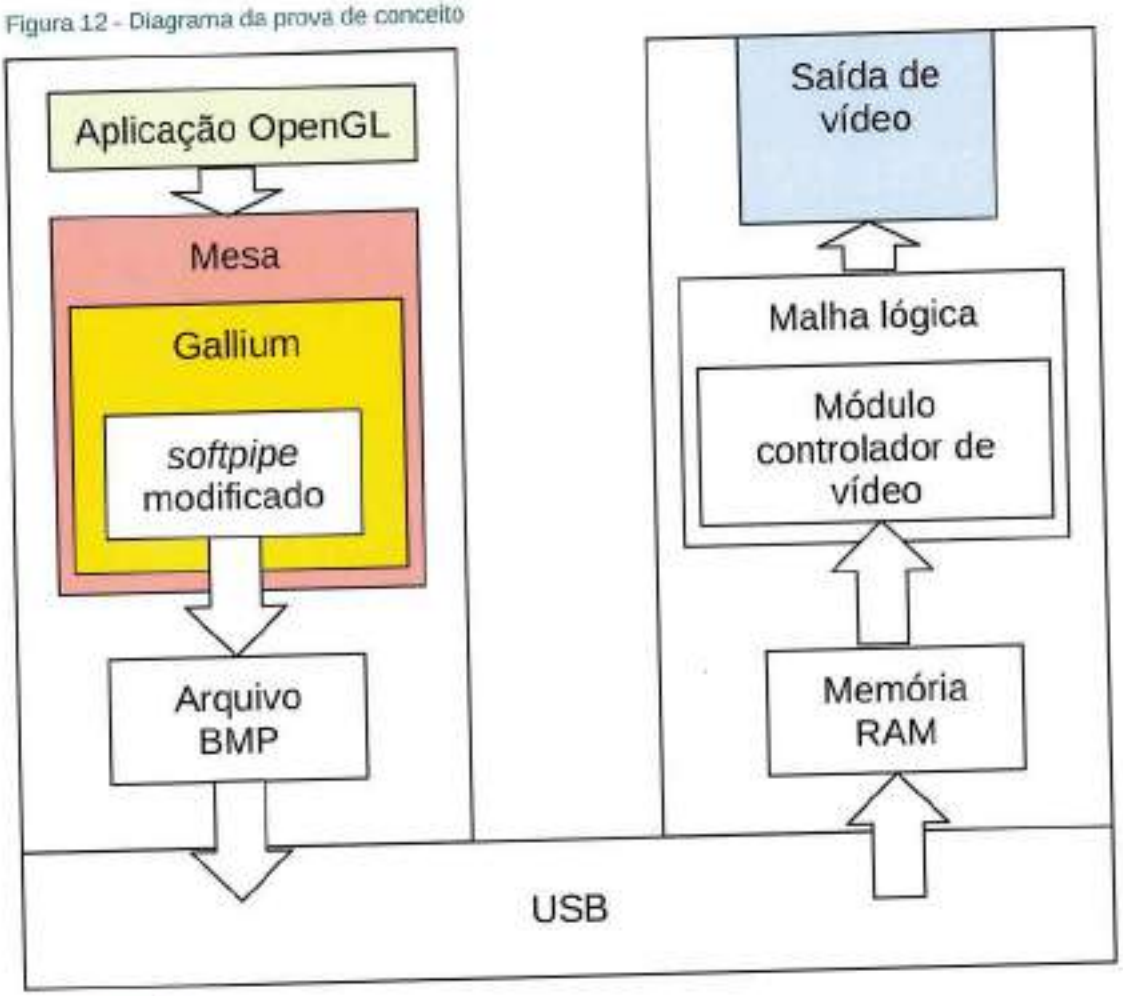
Uma das atividades críticas é a etapa 5.2, de implementação dos blocos da GPU e dos adaptadores (ou drivers) específicos. Isso porque, a partir dessa etapa, o trabalho é maior e todo funcionamento depende do bom projeto de cada bloco. Como os passos a partir do 5º serão realizados mais de uma vez - um para cada bloco do pipeline a ser implementado - esse processo torna-se, de certo modo, iterativo e demanda maior cuidado para evitar problemas desnecessários com a união das partes à medida que o projeto cresce. A vantagem que temos é que o processo todo já é definido nas especificações do OpenGL, o que dá uma boa base para definirmos as entradas, saídas e o funcionamento de cada bloco, evitando problemas de compatibilidade.

Outro ponto crítico é a definição da arquitetura. Dado que não se pode implementar o pipeline em hardware exatamente como ele é logicamente, faz-se necessário a escolha de uma arquitetura capaz de processar os dados de maneira equivalente ao pipeline gráfico e, ao mesmo tempo, ocupar pouca área efetiva ou poucos recursos. A escolha dessa estrutura altera completamente o desenvolvimento seguinte, implicando em alto risco de falha de projeto, dependendo das decisões tomadas.

8. PROVA DE CONCEITO

Antes de entrarmos na etapa de implementação, foi realizada uma prova de conceito para demonstrar a validade do trabalho feito durante a etapa de projeto. Para tal, decidiu-se demonstrar que é possível modificar um adaptador do Mesa extraindo dados do *pipeline* gráfico para processá-los no FPGA e exibi-los em vídeo.

Figura 12 - Diagrama da prova de conceito



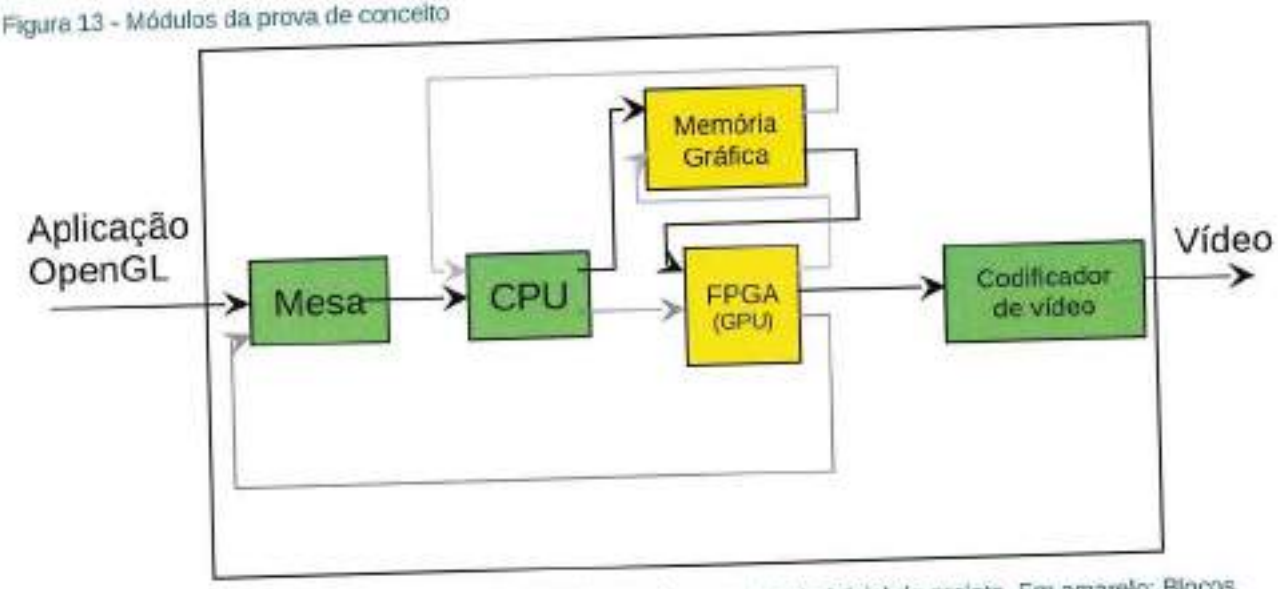
Fonte: autoral

Especificamente, o *driver* softpipe do Gallium é alterado através de sua função de *flush* do framebuffer, equivalente à última etapa do processo de geração de vídeo. Ao invés de ser enviada para a memória de vídeo, a saída do *software* é transferida para um arquivo *bitmap* e transmitida via USB para uma FPGA com saída de vídeo, mostrando um frame no monitor a cada envio. Ainda que essa abordagem seja inadequada para aplicações de vídeo em tempo real, visto que a velocidade de transferência do USB é muito baixa, o experimento serve para provar a viabilidade

do conceito na construção do projeto; bastando apenas a aquisição de plataformas de desenvolvimento mais adequadas — em termos de largura de banda do barramento — para se obter o resultado desejado.

Podemos ver no diagrama a seguir, o caminho percorrido pela prova de conceito apresentada. Em verde, estão destacados os módulos modificados da maneira prevista neste projeto. Em amarelo, os módulos que foram modificados no sentido de atender os objetivos do projeto. Isso porque o FPGA e a Memória Gráfica não implementam, na prova de conceito, os níveis mais detalhados do projeto.

Figura 13 - Módulos da prova de conceito

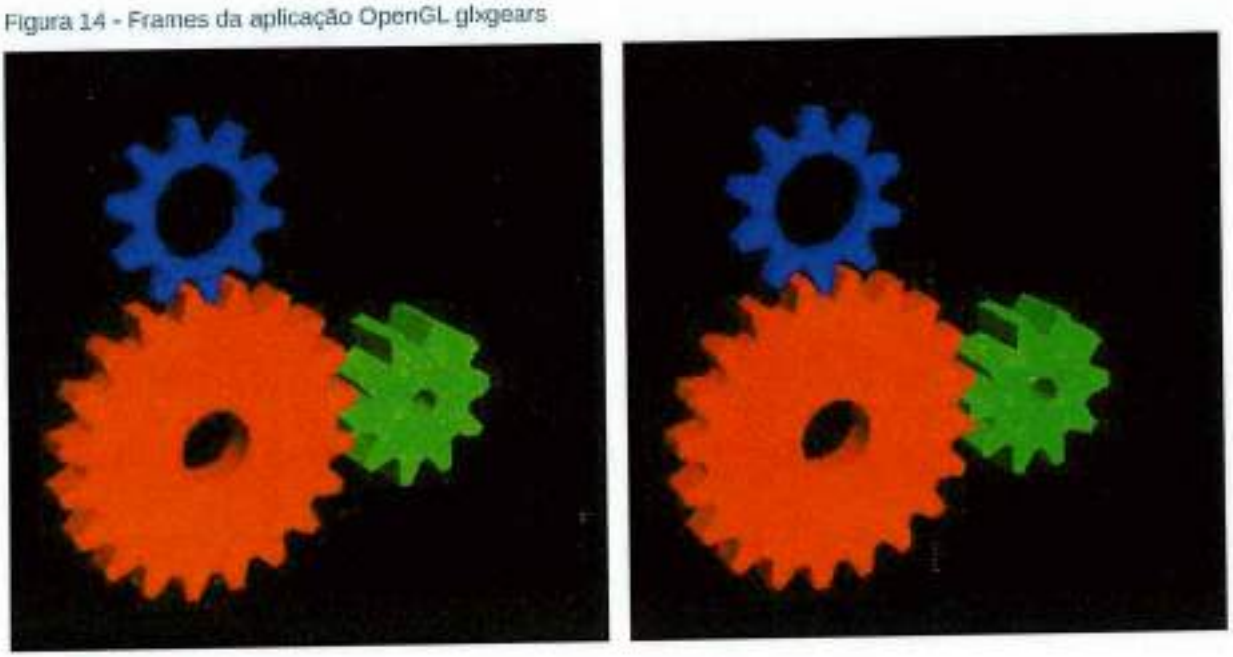


Em verde: Blocos implementados na prova de conceito conforme proposta inicial de projeto. Em amarelo: Blocos que na prova de conceito sofrerão alterações específicas e que não fazem parte da proposta inicial do projeto.
Fonte: autoral.

8.1. PROVA DE CONCEITO: SOFTWARE

A modificação do *driver* softpipe permitiu que fossem obtidas imagens dos *frames* da aplicação OpenGL ao fim do *pipeline*. Na figura, observamos o *frame* de número 1 e número 5.

Figura 14 - Frames da aplicação OpenGL glxgears



Fonte: autoral

O programa de teste utilizado foi o *glxgears*, tradicional como *benchmark* simples de sistemas gráficos. A plataforma de desenvolvimento e testes nessa etapa foi o sistema operacional Linux. A distribuição utilizada foi o Debian 8, assim como a versão Mesa 10.3.2 estável.

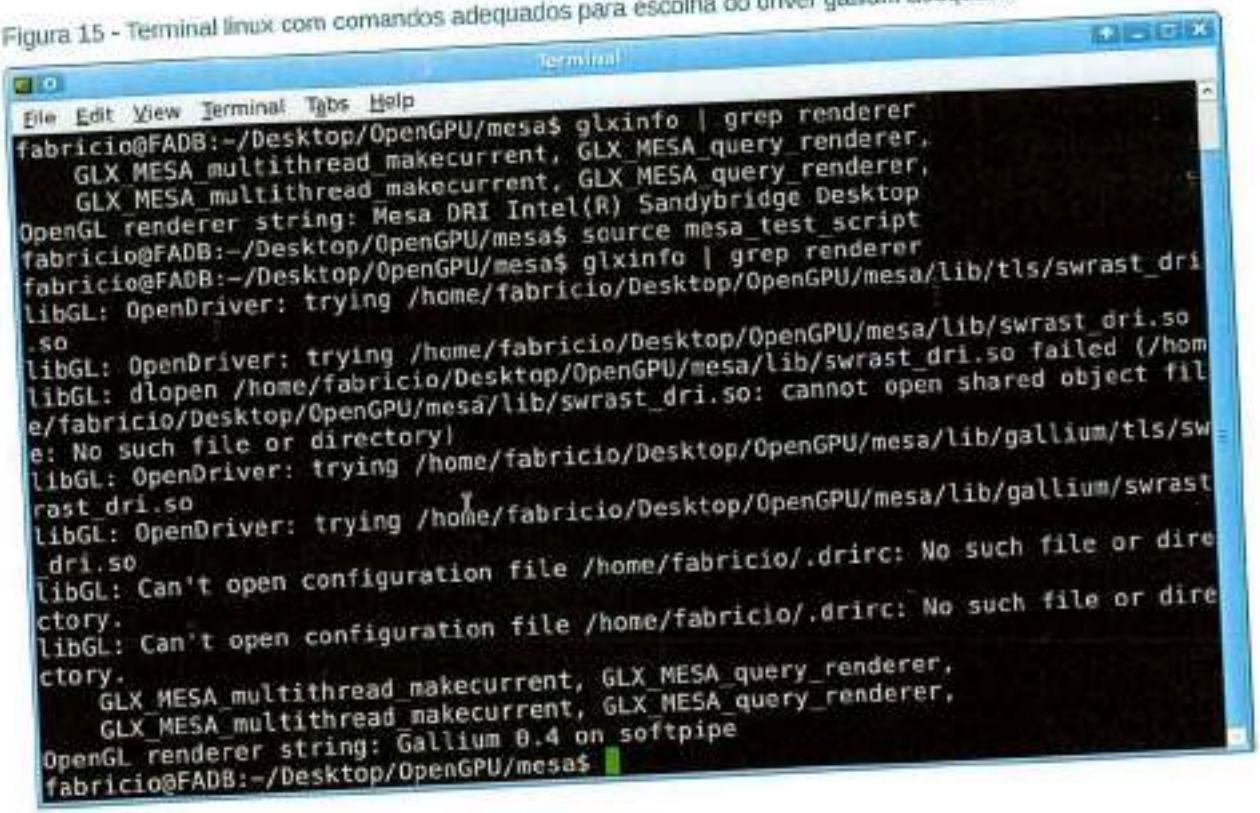
A mudança importante no código do softpipe foi a ativação de dois sinais de depuração (*debug*). A partir deles, é possível armazenar todos os *frames* produzidos pela aplicação. O código modificado está no anexo.

O projeto Mesa é bastante sofisticado e permite algumas configurações importantes. Após compilar o código fonte do Mesa já com a modificação do softpipe, um *script* simples foi desenvolvido a fim de simplificar a declaração das variáveis de ambiente necessárias. Deve-se executá-lo com auxílio do emulador de terminal, conforme instruções a seguir.

```
#!/bin/bash
export LIBGL_DRIVERS_PATH=$PWD/lib:$PWD/lib/gallium
export LD_LIBRARY_PATH=$PWD/lib:${LD_LIBRARY_PATH} $@
export LIBGL_DEBUG=verbose
export LIBGL_ALWAYS_SOFTWARE=1
export GALLIUM_DRIVER=swr
```

O comando *source*, e o *script* acima como argumento, permite que se utilize as variáveis de ambiente sejam declaradas no contexto utilizado. Esse *script* deve ser executado dentro da pasta raiz do código fonte do Mesa. 'LIBGL_DRIVER_PATH' e 'LD_LIBRARY_PATH' definem a localização das bibliotecas compiladas, enquanto 'LIBGL_DEBUG=verbose' garante que mensagens de depuração e análise da execução sejam colocados na saída de texto do terminal. 'LIBGL_ALWAYS_SOFTWARE' indica que o Mesa sempre utilizará o *driver* exclusivamente *software*. 'GALLIUM_DRIVER=swr' decide que será utilizado o *software rasterizer*, que nada mais é que o *pipeline* gráfico executado via *software*. No caso, o sistema escolhe o Gallium Softpipe como *driver* adequado, tal como mostra a imagem a seguir.

Figura 15 - Terminal linux com comandos adequados para escolha do driver gallium adequado



Fonte: autoral

É notável a maior quantidade de informações exibidas, além da mudança do 'renderizador'. Primeiro, o *driver* utilizado pelo GPU integrado ao CPU do sistema, Mesa DRI Intel Sandybridge Desktop. Ele faz parte do pacote i965 no código fonte

Mesa. Na sequência, após configurar as variáveis de ambiente, o sistema passa a utilizar apenas o Gallium sobre o *driver* softpipe, na biblioteca swrast_dri.so.

Um comparativo simples foi feito com três sistemas rodando a aplicação de testes gráficos glmark2, a fim de evidenciar a diferença entre GPU (Intel Sandybridge), CPU com *software* otimizado (llvmpipe) e CPU com *software* de referência (softpipe).

- softpipe: 13 imagens por segundo
- llvmpipe: 369 imagens por segundo
- Mesa DRI Intel Sandy Bridge (GPU, *pipeline* via *hardware*): 980 imagens por segundo

Essa é apenas uma ilustração da diferença entre a performance de cada sistema. Uma avaliação mais precisa exigiria diversos outros testes, inclusive com aplicações gráficas distintas.

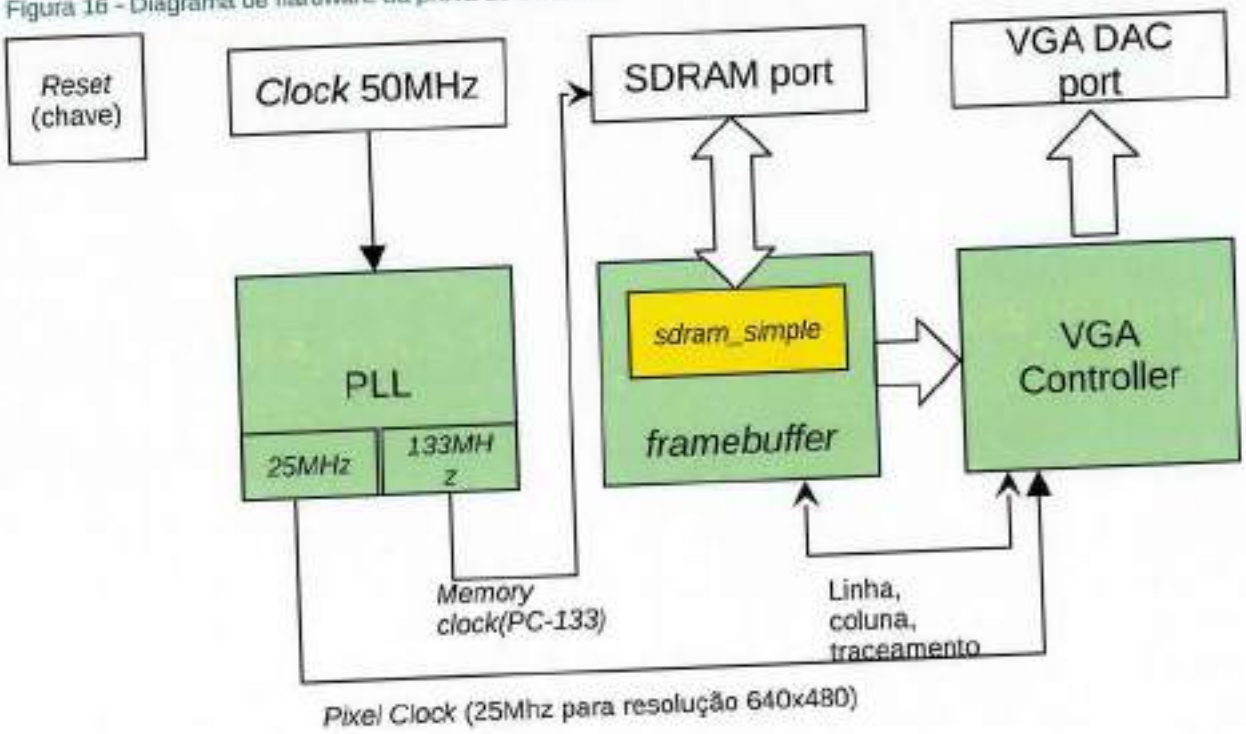
A partir dos arquivos BMP, pode-se prosseguir com a prova de conceito, que é a parte aqui chamada de *hardware*.

8.2. PROVA DE CONCEITO: HARDWARE

O sistema FPGA exige uma percepção diferenciada em relação ao *software*. Entre outras diferenças, uma das mais importantes é a noção de simultaneidade. Linguagens descritoras de *hardware* como VHDL ou Verilog conduzem a eventos ocorrendo de maneira sincronizada ou assíncrona, concorrente ou sequencial — exatamente como se comporta o *hardware* desenvolvido em ASIC ou em circuitos discretos. Por isso, seu desenvolvimento exige habilidades diferentes daquelas vistas em programação de *software*.

A prova de conceito em questão aproveitou da disponibilidade da placa DE2 aos autores. Esta e uma placa de performance razoável, que tem saída VGA, memória RAM e uma quantidade mais que suficiente de células logicas (LE, *logic elements* ou LUT, *lookup table*). O sistema se baseou no seguinte diagrama para ser implementado:

Figura 16 - Diagrama de hardware da prova de conceito



Fonte: autoral

Os blocos em verde indicam módulos implementados em VHDL, totalmente funcionais (testados). O módulo em amarelo, *sdram_simple*, apresentou problemas quanto à conexão com a SDRAM. Ao fim da prova de conceito, este foi o trecho em que não se obteve sucesso. O vídeo pode ser exibido e manipulado na saída VGA, sob resolução configurada. O *framebuffer*, que era responsável por abstrair a SDRAM e torná-la uma memória *plana*, também foi construído adequadamente. Seu endereçamento se propõe a seguir o modelo linha e coluna, como numa matriz de *pixels*. Isso facilita a concepção do controlador VGA.

O intuito inicial era transmitir uma imagem gerada na etapa anterior, *software*, armazená-la na RAM do FPGA e exibi-la no vídeo por VGA. Não foi possível atingir esse objetivo devido ao controlador da SDRAM, embora a maior parte do sistema tenha sido implementado — caracterizando a possibilidade de se desenvolver uma GPU. Os códigos relativos ao desenvolvimento do *hardware* estão no apêndice desse documento.

9. DECISÃO DE ARQUITETURA E ABORDAGEM DE IMPLEMENTAÇÃO

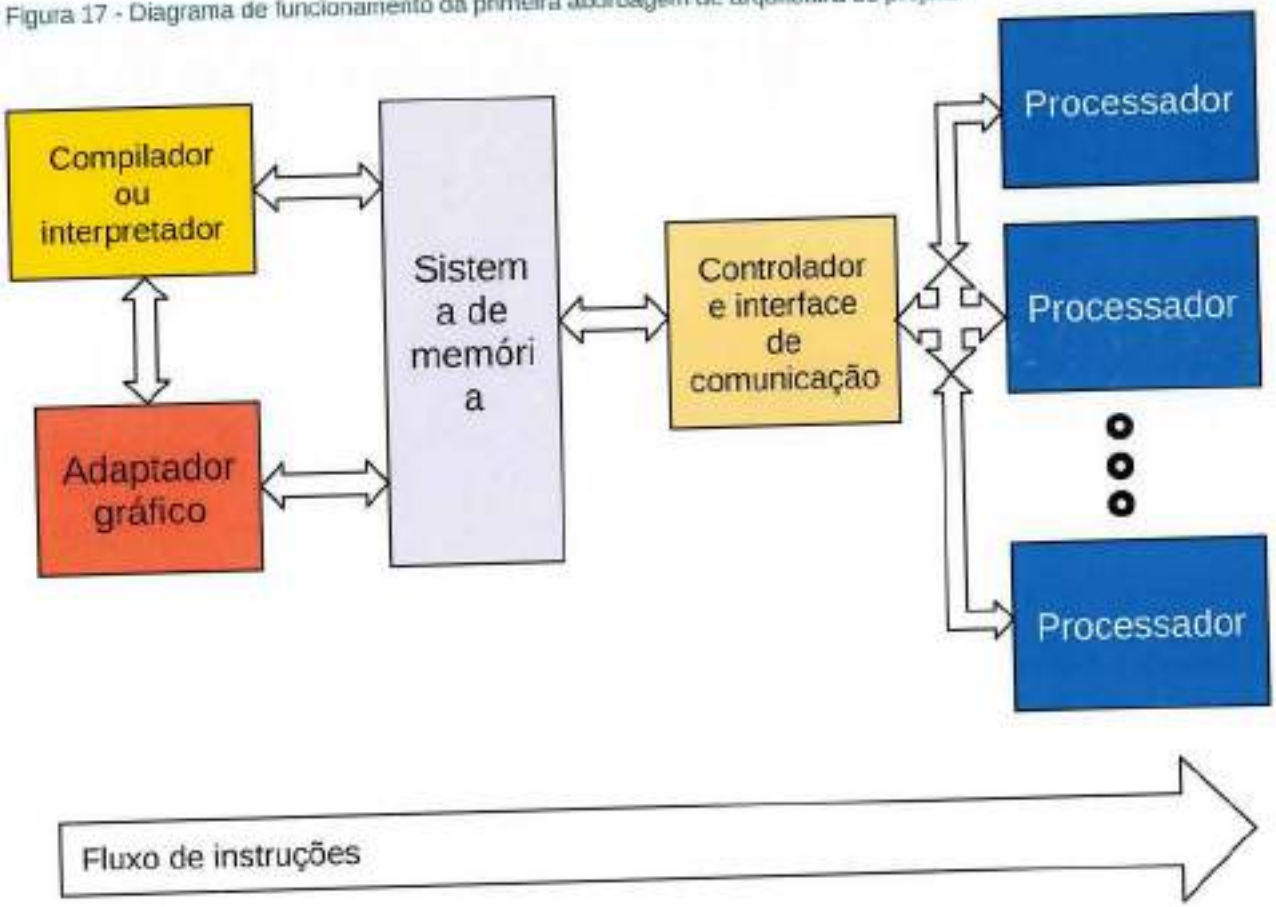
A arquitetura foi discutida após a definição inicial do projeto. Isso porque havia a necessidade de maior compreensão do problema e das suas possíveis soluções. O conceito inicial seguiu a ideia proposta anteriormente de implementação de uma matriz de núcleos de processamento comandadas por uma unidade de controle — enquanto os dados trafegariam pelo sistema de memória e seria alimentado no controlador de vídeo.

Essa perspectiva conduziu à escolha de núcleos de processamento capazes de lidar com os tipos de operações características do *pipeline* gráfico. Uma matriz de possibilidades e funções foi preparada para esta tarefa — inclusa na seção de apêndice deste volume.

9.1. JUSTIFICATIVAS PARA MUDANÇA DE ABORDAGEM

A escolha do núcleo foi fundamental para a conclusão que viria na sequência: implementar e associar processadores em *hardware*, ainda que estivessem prontos, caracterizar-se-ia em uma tarefa inviável no tempo e no esforço humano disponíveis para o presente trabalho. Para entender melhor esta circunstância, é preciso observar a cadeia de desenvolvimento característica de processadores.

Figura 17 - Diagrama de funcionamento da primeira abordagem de arquitetura do projeto



Fonte: autoral

Acima se verifica a necessidade de um compilador ou interpretador compatível com o processador escolhido para compor o *hardware*. O adaptador gráfico precisaria traduzir de uma linguagem de máquina — TGSI, que é a representação usada pelo Gallium — em outra, que seria a linguagem de instruções do núcleo escolhido. Isso aumenta consideravelmente o projeto quanto ao número de módulos a serem desenvolvidos. O sistema de compilação já demanda bastante pesquisa e trabalho a fim de organizar um produto final funcional e de qualidade mínima. As etapas de testes são bastante extensas num primeiro instante, dado não haver ainda uma implementação do GPU. Associar estes módulos convergiria para novas decisões e novos projetos menores.

Apesar desse conceito ter sido debatido e considerado como primeira opção, abandoná-lo foi uma decisão importante para os resultados obtidos posteriormente. Embora isso não descarte que no futuro esta abordagem inicial possa ser revisitada

e implementada — já que é promissora quanto suas características técnicas, além de estar presente na totalidade dos processadores gráficos comerciais aos quais se teve acesso durante a fase de pesquisa deste trabalho.

9.2. NOVA ABORDAGEM

Foi idealizada uma nova abordagem — mais simples, porém de limites mais claros e modestos quanto ao tempo de implementação e à dedicação disponível. Ao invés de implementar a região programável do *pipeline* gráfico, optou-se por desenvolver algum trecho 'fixo' ou de baixa configurabilidade. Então, apresentou-se a ideia de desenvolver o *rasterizador*, um módulo responsável por decidir onde devem ser preenchidos pixels na tela a partir de triângulos. De maneira geral, esta função é importante quanto ao tempo de processamento — embora não seja a de maior consumo de tempo dos processadores, figura entre as principais tarefas dos sistemas de processamento gráfico atuais, já que otimizações nessa seção podem ter impacto notável na performance do sistema total.

A implementação do rasterizador tornou-se, portanto, o objetivo e, de certa forma, o módulo central do GPU desenvolvido em *hardware* neste trabalho. As demais funções gráficas e de processamento geral ficaram em *software*. Apesar disso, espera-se que o ambiente de desenvolvimento forneça o suporte necessário para continuação da implementação de outras funcionalidades esperadas em uma GPU dos padrões contemporâneos a este trabalho. É importante frisar que a arquitetura e abordagem de implementação escolhidas atingiram todos os requisitos de engenharia e *marketing* propostos anteriormente, como será verificado a seguir.

10. PRIMEIRA FASE DE IMPLEMENTAÇÃO

10.1. RELATÓRIO DE RECONHECIMENTO E TESTES NO SOFTPIPE

O softpipe foi investigado ao longo de duas semanas. Usou-se o código fonte, a documentação do gallium, um *debugger* e um *profiler*. Abaixo, há *callgraphs* de uma aplicação gráfica rodando sobre o driver gallium, editado e compilado. A geração destes gráficos foi feita usando o programa *kcachegrind*, uma ferramenta que lê arquivos gerados por um software de *profiling* — conhecido como *callgrind*, parte do sistema de analise valgrind.

O callgrind analisou a execução do programa glxgears rodando com o driver gallium softpipe modificado. O softpipe rasteriza e renderiza um triângulo por vez — dentro da função sp_vbuf_draw_arrays há loops que percorrem um buffer de vértices, dependendo da primitiva (pontos, linhas, triângulos fan, quads, etc). Para cada triângulo, o pipeline calcula informações de projeção e corte de perspectiva através das funções em roxo nos diagramas abaixo. A função subtriangle executa renderização e cortes adequados no triângulo em questão. Esses dados são passados à função flush_spans.

Esta, por sua vez, gera os *fragment quads* (blocos de exatamente 2x2 pixels). Os quads são repassados a funções de teste de profundidade e stencil. Blocos de *fragment quads* são gerados, num total de no máximo 16 por iteração — número relacionado aos testes de bits dentro das funções (16 quads = 32x32 bits ou pixels). Os blocos de *fragment quads* são direcionados para a função shade_quads que executa o *shader* definido pela aplicação — estágio do *fragment shader*. Ao fim do loop de geração de blocos e o respectivo *shading* e escrita dos *color buffers*, a execução retorna para o loop de triângulos — após o qual haverá a finalização do frame, a união dos *color buffers* e o respectivo *flush* da cena para o framebuffer.

A modificação necessária para demonstrar a usabilidade do softpipe no OpenGPU foi alterar o código de modo que não mais o *shading* fosse executado a cada bloco de quads produzido. Ao invés disso, a função shade_quads foi executada somente ao final de todas as iterações de produção de fragment quads para um determinado

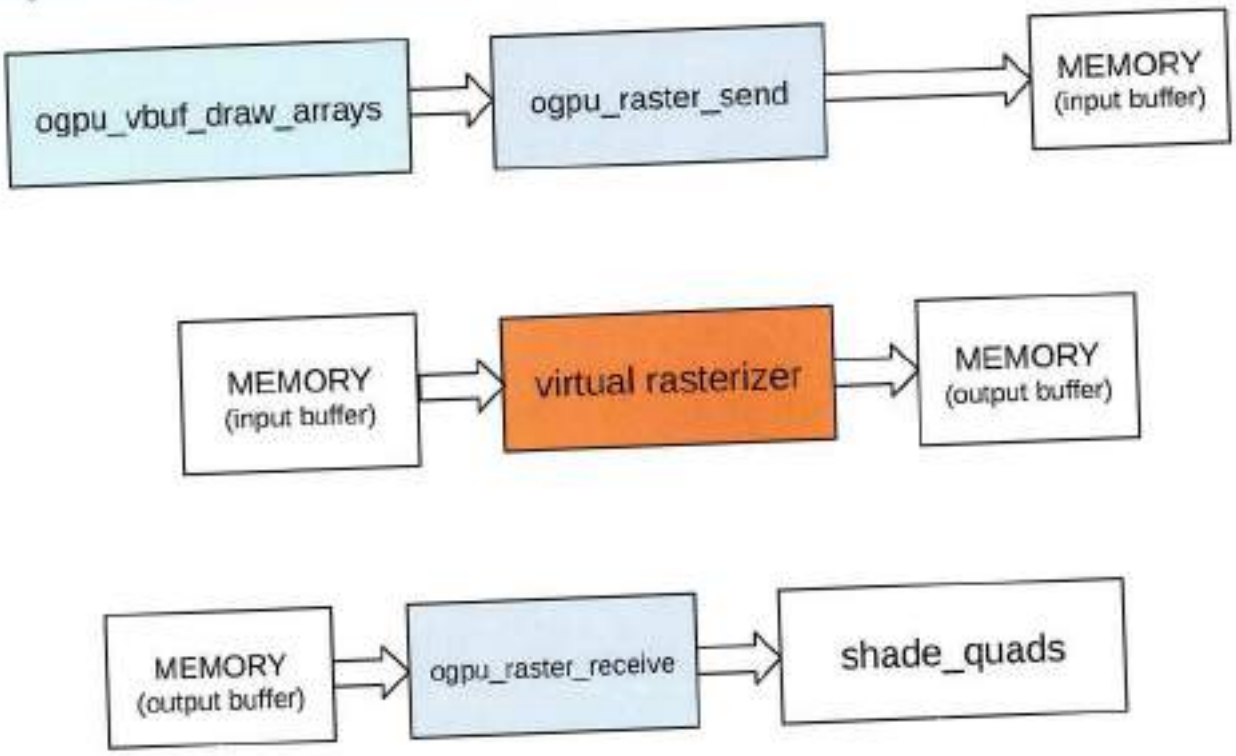
triângulo. Tendo, portanto, um buffer de vértices e uma função que dá continuidade ao pipeline gráfico, podemos construir o processo de rasterização e renderização, obtendo um buffer de fragmentos — prontos para entrar no estágio de *fragment shading*.

A maior dificuldade nesta tarefa foi alocar o buffer corretamente — os quads gerados não podiam ser alimentados de uma vez na função — precisaram ser alimentados por um loop que repetia o número de quads gerados a cada iteração interna. Isso demandou um buffer mais sofisticado que guardava não apenas os *fragment quads* de uma iteração, mas também o número gerado naquele instante — isso foi repetido posteriormente nas chamadas da função shade_quads. Embora não verificado, isso ocorreu possivelmente pela shade_quads supor blocos relativos apenas a uma linha — se passarmos quads relativos a linhas diferentes, mesmo que em ordem de 'Z', horizontalmente do fim do triângulo para o começo, ele executa o shading com falhas visuais. Esta pode ser uma característica a ser investigada futuramente a fim de minimizar a carga de informações transportadas.

A versão do Mesa escolhida para modificação foi a 8.0.2, visto que a distribuição UBUNTU disponibilizada pela Terasic para rodar na DE1 contém exatamente essa versão de drivers de vídeo. Para haver compatibilidade de ambiente, é importante que as versões sejam próximas ou mesmo iguais. Não deve, no entanto, atrapalhar em futuras atualizações do driver OpenGPU, principalmente, porque o Gallium é bastante estável e não sofre modificações fundamentais desde 2010 — enquanto a versão 8.0.2 do Mesa é de 2012.

Com essa situação, espera-se poder construir o rasterizador em software — de modo que seja um espelho para o desenvolvimento em hardware. A seguir, a relação das estruturas e de uso de dados no trecho escolhido para ser implementado:

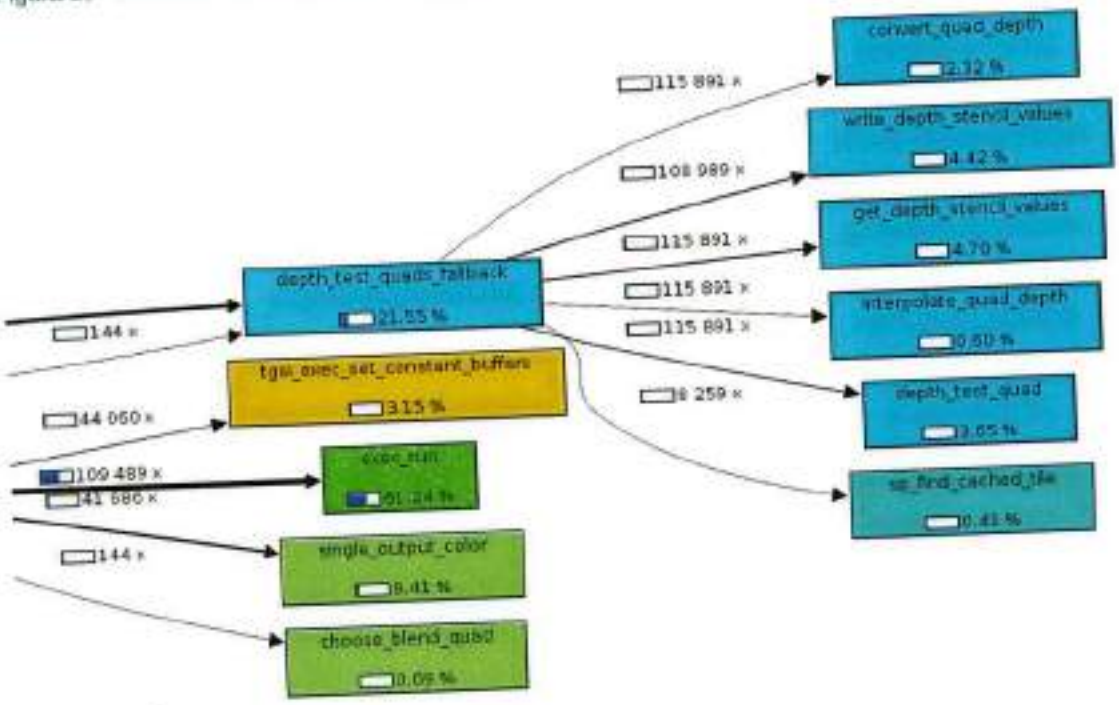
Figura 18 - Diagrama do softpipe modificado e suas funções



Fonte: autoral

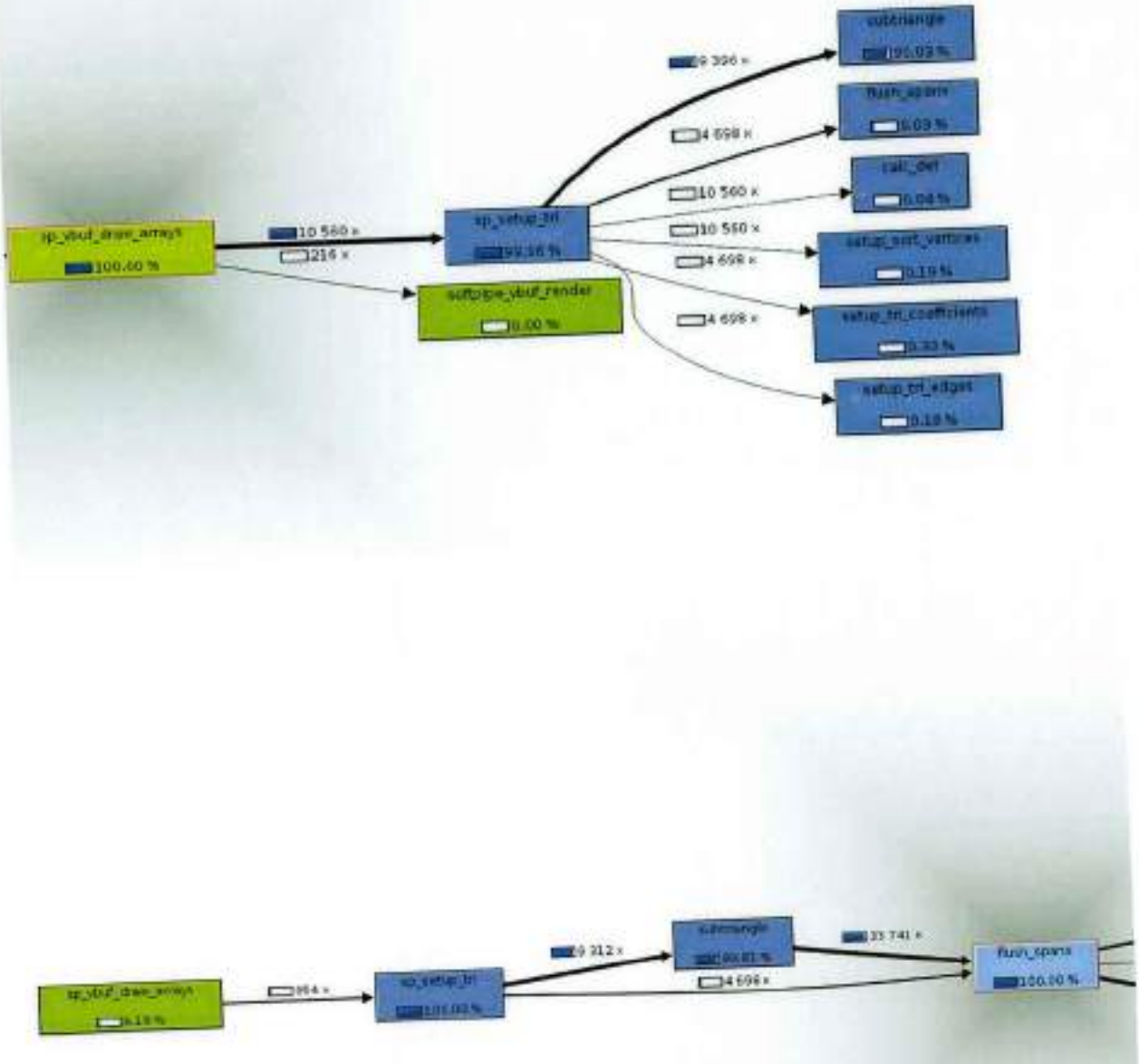
Esta concepção surgiu a partir dos gráficos de chamadas (call graphs) produzidos pela execução do softpipe e pela sua respectiva análise com o software callgrind.

Figura 19 - Fluxo de informações no softpipe parte 1



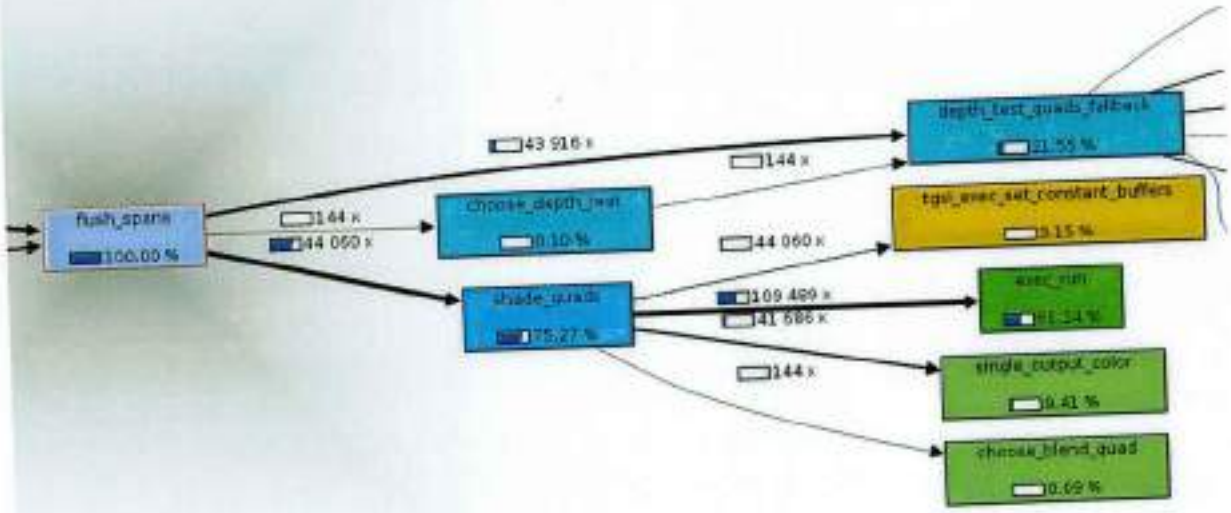
Fonte: autoral

Figura 20- Fluxo de informações no softpipe parte 2 e 3



Fonte: autoral

Figura 21 - Fluxo de informações no softpipe parte 4



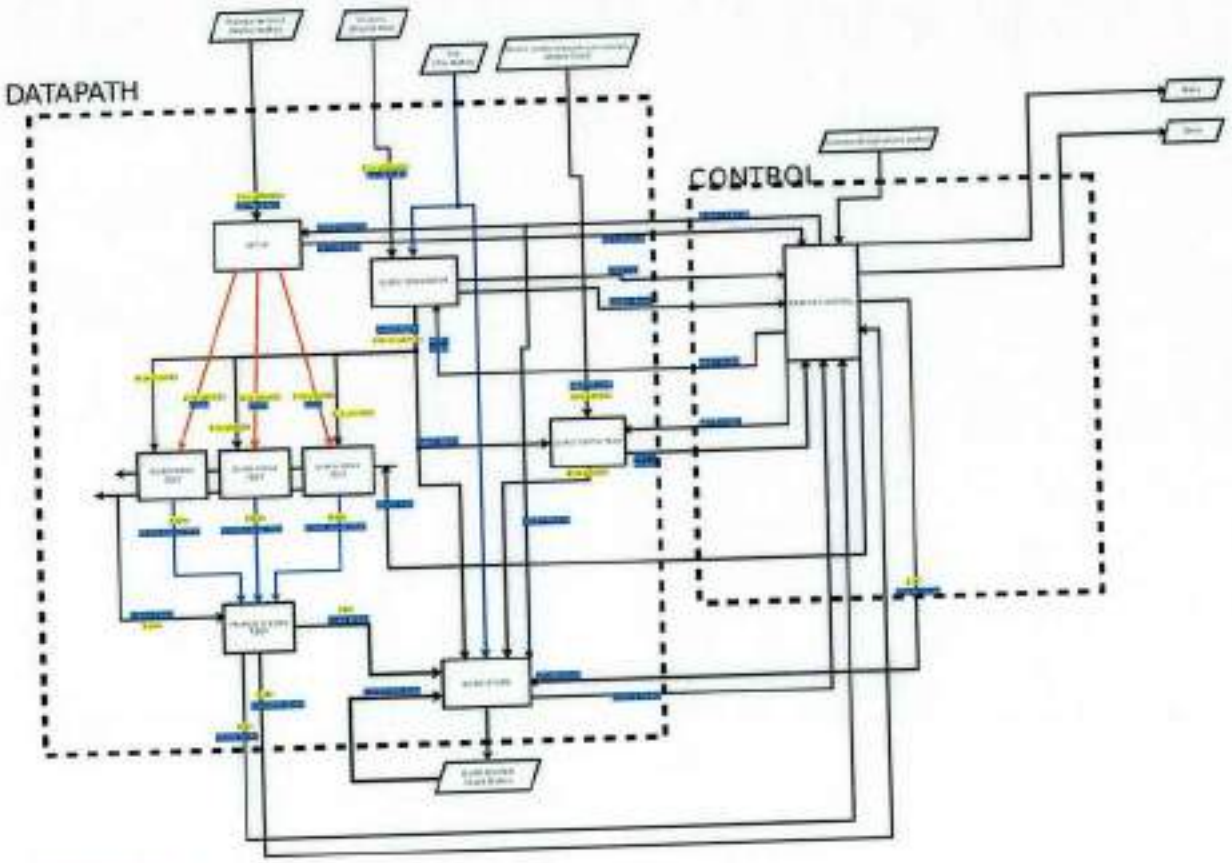
Fonte: autoral

Nas figuras acima se verifica o fluxo de informações na implementação do softpipe. A partir desses gráficos de chamadas foi possível estabelecer o trecho de inserção das modificações. Elas são necessárias para conduzir o fluxo de dados para o rasterizador, bem como recebê-lo de volta do mesmo módulo, dando continuidade ao processamento gráfico.

10.2. IMPLEMENTAÇÃO DO RASTERIZADOR VIRTUAL

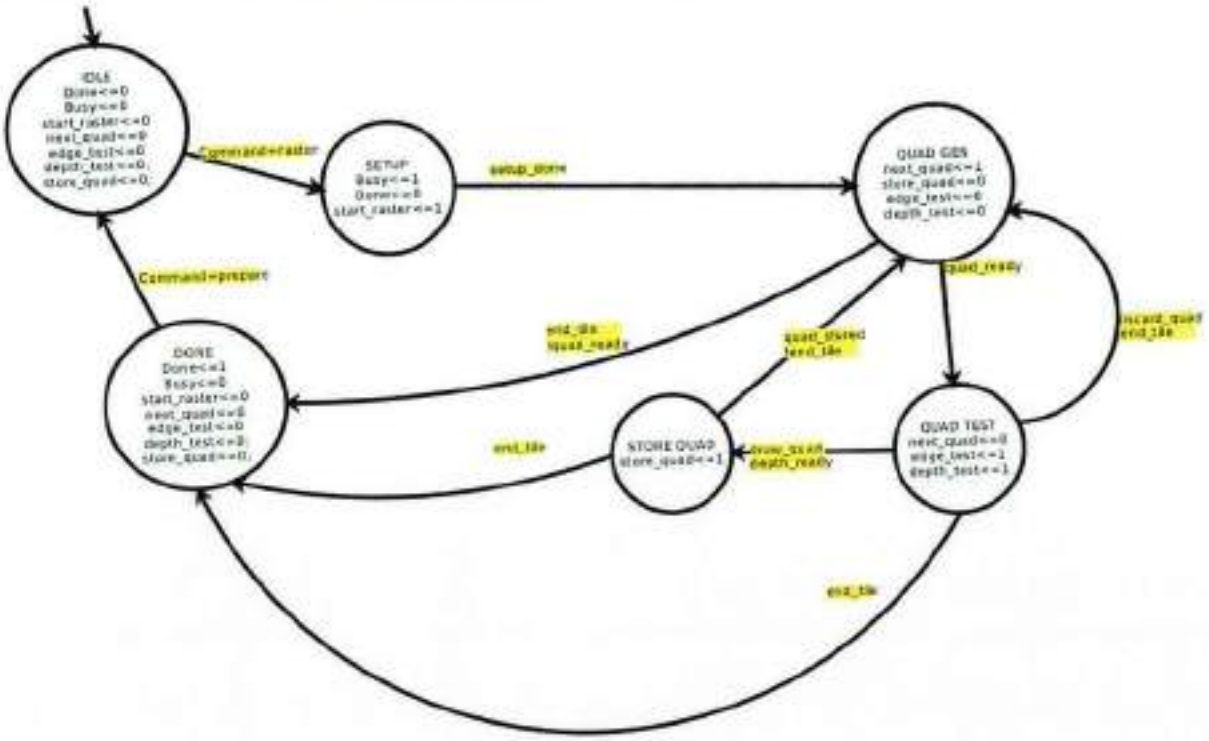
Abaixo, as imagens do hardware desenvolvido e o algoritmo do módulo de controle. Ambos foram testados e executados com sucesso. Apesar de realizar o teste de profundidade (depth testing), os dados produzidos não foram repassados ao softpipe — isso porque há necessidade de modificar algumas funções de chamada como a shade_quads que executa o shading dos quads produzidos pelo rasterizador.

Figura 22 - Diagrama do funcionamento do rasterizador, evidenciando o trecho de dados e o trecho de controle



Fonte: autoral

Figura 23 - Máquina de estados da unidade de controle do rasterizador



Fonte: autoral

Imagens do diagrama do *hardware* e do algoritmo de controle do rasterizador estão presentes no apêndice em maior qualidade.

10.3. FUNCIONAMENTO DO RASTERIZADOR

A implementação atual recebe vértices de um triângulo e informações sobre o *tile* atual, quer dizer, a região da tela que em que se deseja processar o triângulo escolhido. O rasterizador realiza a preparação dos dados, calcula em quais pixels do *tile* dentro da tela o triângulo será preenchido. Esses pixels são associados em blocos de 2x2, chamados de *quad fragments* – fragmento é o nome dado a cada pixel. Para cada *tile* são produzidos fragmentos pertinentes quando houver necessidade de rasterização, carregando informações de máscara (pixels a serem preenchidos) e de profundidade, ou *depth testing*. Para cada *tile*, é executada uma operação de rasterização em cada triângulo – ou, a cada triângulo, são executados diversas operações de rasterização, divididas em *tiles*.

O uso da concepção de *tilling* da tela permite que vários rasterizadores sejam alimentados paralelamente, produzindo informações de diversos *tiles* diferentes. Outra vantagem é a possibilidade de não restringir o tamanho máximo de tela (embora a implementação atual esteja limitada a 65536 pixels de largura ou altura, devido à escolha da representação numérica de 16 *bits*).

10.4. FUNÇÕES IMPLEMENTADAS

O rasterizador virtual foi inteiramente integrado ao softpipe. Na maneira como está, o sistema é capaz de executar qualquer aplicação OpenGL (pelo menos até a especificação 3.0, na versão 8.0.2 do Mesa3D de 2012). É possível que execute aplicações DirectX em Windows, embora não tenham sido testadas — todos os testes foram realizados somente em plataforma Linux.

11. SEGUNDA FASE DE IMPLEMENTAÇÃO

A segunda parte da implementação é caracterizada por adaptar a primeira fase para o hardware, transformando o rasterizador evidenciado na figura 24 em código. No caso, a linguagem escolhida foi o VHDL (*Very High Speed Integrated Circuit Hardware Description Language*).

11.1. CRONOGRAMA

O correr da implementação e da verificação não seguiu o cronograma conforme as expectativas. Isso porque a preparação da DE1-SoC demandou um tempo maior, enquanto o projeto do rasterizador veio tardiamente. Ainda assim, houve uma sequência bem estabelecida e previsível, garantindo que, como veremos, o projeto estaria finalizado até sua apresentação.

O mês de agosto serviu como extensão da fase de projeto, ainda não finalizada naquele momento. As duas primeiras semanas de setembro serviram para configuração do Mesa3D e respectiva compilação. Um estudo sobre técnicas de rasterização e sobre o funcionamento do Mesa3D e do Gallium tomaram as duas últimas semanas. Nas duas primeiras semanas de outubro, foi implementada a versão virtual do rasterizador, já acoplado ao *driver* softpipe do Gallium – na prática, o softpipe foi modificado para que transmitisse e recebesse dados pela interface com o rasterizador OpenGPU. Como será lembrado na seção de conclusão e futuro do projeto, o softpipe modificado deverá dar lugar a um novo *driver* Gallium. Até esse momento, o sistema já exibia qualquer aplicação OpenGL compatível com a versão 8.0.2 do Mesa3D, utilizando o softpipe modificado para o OpenGPU.

Fim de outubro e início de novembro marcaram a implementação do *hardware*. Enquanto as verificações dos módulos foram feitas, a DE1 SoC foi utilizada em pequenos exemplos a fim de verificar o seu uso no teste do *hardware*. Infelizmente, a maior parte do tempo foi consumido na implementação das interfaces de comunicação do FPGA, fazendo com que alguns problemas na implementação do *hardware* não fossem vistas a tempo. Embora isso, o sistema funcionou no início de dezembro, atingindo todos os requisitos de *marketing* e de engenharia.

11.2. PREPARANDO A DE1-SoC

Inicialmente, foi necessário preparar a placa de desenvolvimento para que pudesse rodar os softwares necessários. Para isso, um sistema operacional precisaria ser instalado na placa de desenvolvimento. A fabricante da placa, Terasic, disponibilizava duas opções: o Ubuntu Lite e a LXDE (*Lightweight X11 Desktop Environment*). A vantagem da primeira é a maior facilidade para instalar os pacotes e bibliotecas necessários e a desvantagem é que, por ser mais pesado, a velocidade é um pouco reduzida. A segunda é bem mais leve, porém houve problemas com instalação de pacotes. Uma terceira opção seria criar um sistema operacional customizado.

Como o sistema operacional não faz parte do escopo do projeto e não afeta o seu funcionamento, escolheu-se a primeira opção. O segundo passo foi a instalação das bibliotecas necessárias para o funcionamento do mesa3D. Vale a pena ressaltar que a arquitetura de um processador ARM, que compunha o núcleo da placa, é diferente da arquitetura dos processadores em um desktop comum (x86 64 *bits*). Portanto, programas compilados (ou seja, "montados") para computadores pessoais comuns não funcionam na placa. É necessário um compilador específico e bibliotecas específicas.

Após a instalação de todos os componentes necessários, a biblioteca mesa3D e o driver Gallium modificados foram compilados na placa sem problemas: a preparação estava completa.

11.3. RASTERIZADOR VIRTUAL

A implementação de um rasterizador software diferente daquele presente no softpipe, do Gallium, foi fundamental para verificar a completude do algoritmo de rasterização. Embora seja considerado simples, há diversos pontos propensos a falhas prováveis – como o sistema de tiling e a geração de fragmentos.

Outro ponto interessante, foi a implementação parcial de algumas características de hardware, como clock e sincronismo dos dados – embora rodando em um CPU. Isso permitiu corrigir alguns erros de lógica no algoritmo do rasterizador, que, fundamentalmente, deveria considerar a implementação em hardware.

A implementação virtual foi bem sucedida, não diferindo daquela já existente no softpipe. Toda a sequência do desenvolvimento se deu em cima do algoritmo produzido e testado nesta etapa.

11.4. RASTERIZADOR NO FPGA

A implementação em *hardware* foi bem sucedida de maneira geral. O rasterizador virtual foi replicado em VHDL, passando por testes funcionais em cada módulo – listados nas figuras anteriores. O *software* ModelSIM, associado ao Altera Quartus foi utilizado nessa etapa.

A maior dificuldade surgiu na integração entre o sistema e o FPGA, que tem sistemas de transmissão de dados já definidos. Embora estejam prontos, alguns não tinham configuração tão simples ou – no caso de um modulo específico – não houve meios de funcionar de acordo com a especificação. Cerca de duas semanas de trabalho quase contínuo foram necessárias para garantir que instruções e dados fossem trocados entre o softpipe modificado, dentro do CPU ARM, e o rasterizador OpenGPU, no FPGA.

O sistema dentro do FPGA foi construído com base num exemplo fornecido pela Terasic. Nele, uma implementação em linguagem de *hardware* Verilog(apesar de instanciar módulos em arquivos VHDL) faz a interface entre os módulos gerados pela ferramenta Qsys, da fabricante Altera, e o processador ARM com sua estrutura de *hardware* característica – sistema chamado de HPS pela Altera. Há módulos e configurações para debug, JTAG, e exibição VGA com acesso ao *framebuffer* do sistema via *bus* F2H. O sistema operacional Linux, na versão Ubuntu disponibilizada pela Terasic, foi mantido e modificado conforme as necessidades a fim de executar os testes e as aplicações para o OpenGPU.

O sistema de interface foi definido com base em registradores de entrada e saída através do Qsys. De maneira geral, a implementação em *software* troca dados com o FPGA utilizando o *bus* H2F – comunicação em que HPS comanda e módulos no FPGA. Outra opção seria usar o F2H, criando módulos no FPGA para escrever ou ler diretamente na memória DDR3 do sistema.

As variáveis de comando, dados dos vértices, *tile*, *cliprect*, *reset*, endereço de *buffer* e coeficientes para *depth test*, foram definidos em registradores alocados em

endereços gerados no Qsys. Esses registradores funcionam como entrada e saída(módulos PIO no Qsys) e têm endereços acessíveis pelo *bus* Avalon, da Altera. A saída do sistema deveria ser um *buffer* de RAM no próprio FPGA ou na memória DDR3 do sistema. Nenhum dos dois meios foi viável dentro do tempo disponível para execução do projeto – sobretudo se considerados os atrasos na configuração da DE1 SoC. O módulo de interface genérica, *External Interface to Avalon*, fornecido pela Altera não funcionou conforme documentação disponível – obrigando o uso de um sistema pouco indicado para esse tipo de aplicação, dada a necessidade de grande largura de banda na transmissão de dados.

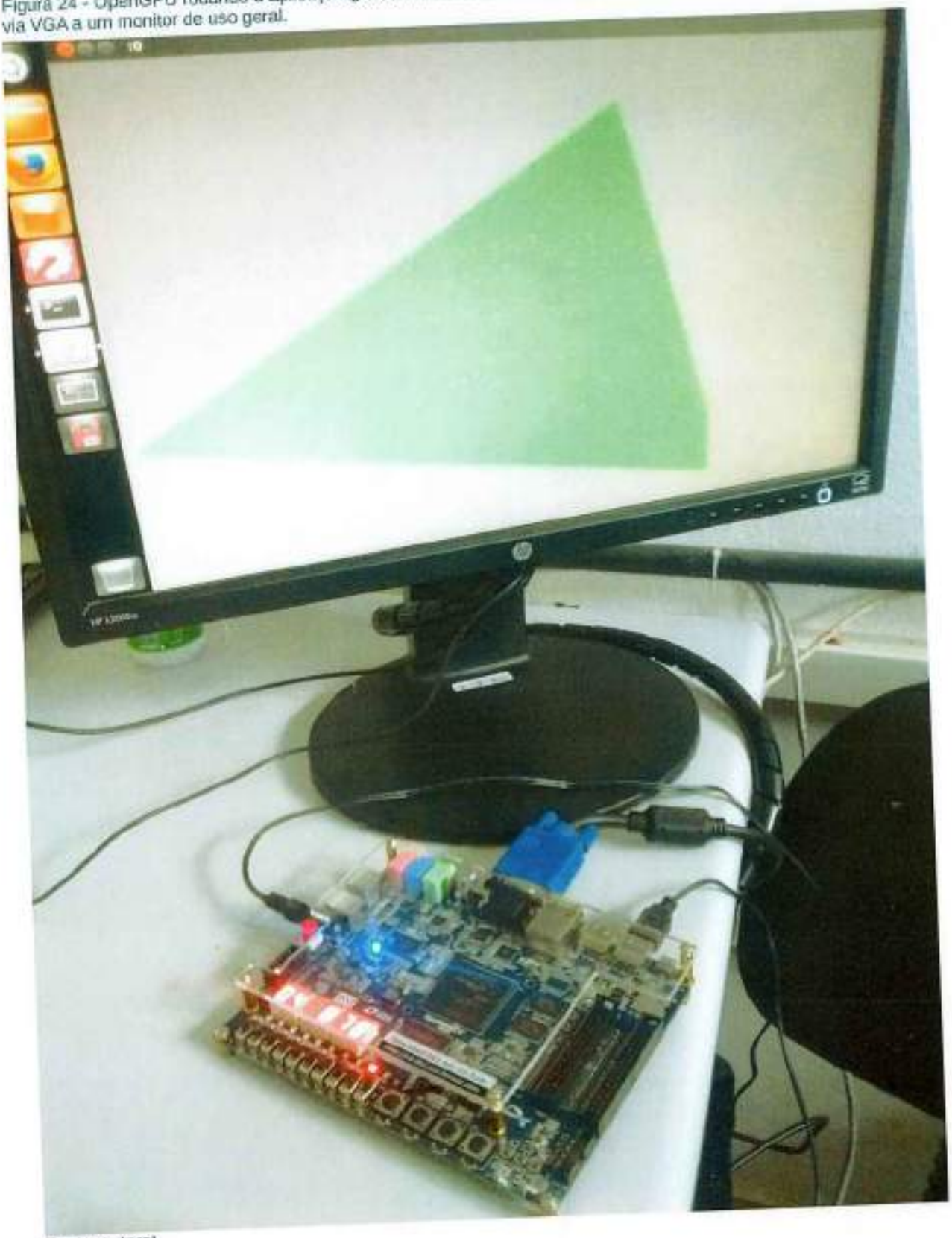
A cada *quad fragment* produzido, um sinal de requisição é gerado pelo módulo de armazenamento(*quad store*). O rasterizador aguarda até que um outro sinal, de *acknowledge* retorne do adaptador OpenGPU no CPU. Isso é, o rasterizador deve esperar a leitura de cada pacote de dados produzido. Cada pacote foi construído para ter 64 bits, representando as informações de cada *quad fragment* – isso significa transmissões volumosas de dados e atrasos consideráveis, já que o acesso do HPS ao sistema do FPGA é bastante tumultuado por tráfego de outros periféricos. Em um *tile*, podem ser produzidos até 1024 *quad fragments*, o que significa uma carga de até 8KB em cada operação de rasterização. Supondo uma tela de 1024x768 pixels, ou 16x12 *tiles*, serão 1.5MB a cada frame apenas para um triângulo. É sabido que uma cena conta com centenas, milhares ou até milhões de triângulos em certas aplicações atuais. Por isso, é fundamental uma comunicação de alta capacidade de transmissão, suportada por *buffers*.

Apesar dessa situação não prevista, o sistema pode funcionar com um pouco menos de performance no caso de uma aplicação OpenGL simples e considerável lentidão em aplicações mais sofisticadas. O objetivo de otimização não estava no escopo do projeto, então esse não é um problema para a implementação atual.

Houve alguns problemas não verificados durante a fase de testes funcionais dos módulos. Assim, o rasterizador não gerou todos os *quad fragments* esperados. Por isso, os triângulos ficaram com aspecto pontilhado na implementação *hardware*. Apesar disso, o sistema é estável e não houve outras falhas durante o período de testes realizado(cerca de uma hora contínua com diferentes aplicações OpenGL).

As chaves DIP da DE1 SoC foram usadas para trocar entre as implementações: original do softpipe, virtual do OpenGPU e *hardware* do OpenGPU.

Figura 24 - OpenGPU rodando a aplicação glxheads sobre a DE1 SoC com uma versão Ubuntu Linux, conectado via VGA a um monitor de uso geral.



Fonte: autoral

12. LICENÇAS ABERTAS DE USO

Com o objetivo de se adequar com o requisito de licença aberta, várias licenças de uso de software foram analisadas. Com isso, uma tabela comparativa foi gerada para facilitar a escolha que melhor se enquadra nos objetivos do projeto, com os seguintes critérios e seus respectivos pesos (em parênteses):

- Restrições (3): Diz respesto ao quão restritivo é aos usuários que eventualmente utilizarão o projeto. Maior nota é menos restritivo.
- Clareza (1): Diz respeito ao quão claro é sua documentação. Maior nota é mais documentado.
- Abrangência (2): Diz respeito à quantidade de tópicos legais que a licença abrange. Maior nota é mais abrangente.
- Uso (3): Diz respeito à popularidade, ou seja, ao quão bem aceito é pela comunidade OpenSource. Maior nota é mais comumente usado.
- Compatibilidade (2): Diz respeito à compatibilidade da licença com outras licenças OpenSouce, importante para evitar problemas no uso do projeto por terceiros. Maior nota é mais compatível.

Tabela 20 - Comparativo entre licenças

Licença	Restrições	Clareza	Abrangência	Uso	Compatibilidade	TOTAL (média)
Apache 2.0	7	8	7	10	5	7,54
BSD-2	9	5	2	7	8	6,64
GPL 3.0	4	10	9	8	3	6,36
MIT	8	5	3	9	10	7,45
MPL 2.0	6	8	6	6	6	6,18
CDDL 1.0	5	6	7	4	5	5,18
EPL 1.0	7	5	6	6	6	6,18
PD	10	4	3	5	10	6,81

Fonte: autoral

Todas essas licenças são aceitas pela Open Source Initiative, que é a entidade responsável por endossar os softwares abertos e suas licenças. Os sítios contendo cada uma dessas licenças se encontra na bibliografia.

A tabela nos indica que a melhor opção para uso de licença é a Apache 2.0. Essa é uma licença usada com muita frequência pelos desenvolvedores de software aberto e que possui um alto grau de liberdade. Suas principais características diferenciadas são as restrições quanto ao uso de marca registrada e sua exigência de que os direitos autorais sejam reconhecidos. Isso é interessante pois certifica que a OpenGPU se enquadra nos padrões de Open Source enquanto mantém o reconhecimento da autoria por parte dos alunos que participaram no projeto.

12.1. LICENÇAS DE SOFTWARE DE TERCEIROS UTILIZADOS NO PROJETO

O projeto utilizou uma versão modificada do Mesa3D e do Gallium. Com isso, é necessária atenção às exigências das licenças desses softwares. No caso, ambos se utilizam da licença MIT, que tem como característica principal ser muito permissiva e simplificada. Dessa forma, não há problemas restritivos ao projeto no tocante a licenças de terceiros e eventuais problemas legais.

13. DISCUSSÃO E CONCLUSÕES

O projeto possui grande capacidade de crescimento e melhoria. A abordagem adotada pode abrir espaço para o desenvolvimento de novas tecnologias, talvez competitivas com sistemas comerciais futuros. A escolha da arquitetura permitiu alcançar os objetivos propostos inicialmente, sob o aspecto de *marketing* e também de engenharia. A abordagem garantiu que os requisitos básicos para que o GPU seja funcional, integrável e expansível fossem alcançados. A presença da licença traz integração ao estado da arte no âmbito de softwares e hardwares abertos.

A prova de conceito demonstrou o funcionamento da plataforma de desenvolvimento e a aplicabilidade da ideia proposta. O estágio de desenvolvimento do rasterizador virtual, denominado primeira fase, garantiu a objetividade necessária para a implementação inicial do projeto. A segunda fase, pela qual o projeto passou, pode ser considerada o embrião de uma implementação em hardware; o passo necessário para que o projeto seja integrado à comunidade.

Os custos envolvidos nesta elaboração ficam restritos ao estabelecimento do ambiente de desenvolvimento e na manutenção da dedicação humana envolvida. Isso torna a empreitada, da forma como foi conduzida, de baixo custo. As dificuldades do projeto são decorrentes da escassa mão-de-obra e tempo disponíveis. Como o time responsável pelo trabalho é reduzido e a dedicação não era exclusiva ao projeto, os cronogramas propostos não puderam ser seguidos. A resposta da equipe foi focar em um trecho específico do funcionamento de uma GPU, o rasterizador, para simplificar o trabalho. Isso, porém, não descaracteriza o escopo inicial e não foge das intenções originais da equipe. Pode-se dizer que o grande sucesso obtido foi o a definição de uma boa estratégia de ataque ao problema, que permite, com o tempo, que um produto final completo, totalmente implementado em hardware, seja desenvolvido.

A expectativa futura é aperfeiçoar a técnica de rasterização, produzir um sistema de *bufferização* adequado – contando com técnicas de compressão ou similares –, implementar *depth testing* pelo lado do *software*, aprofundar o gerenciamento de memória e o seu acesso de modo a compartilhar dados de maneira eficiente. Isso abrirá caminhos para que processadores de uso geral possam ser implementados dentro do OpenGPU – levando-o ao patamar de GPU de uso geral, conhecida como

GPGPU. A abertura à comunidade é o objetivo maior, abrindo oportunidade para que um GPU aberto seja desenvolvido em FPGAs e, no futuro, em ASICs.

Este projeto é importante na área de sistemas eletrônicos, já que o GPU é *hardware* fundamental na elaboração de sistemas mais complexos nas mais variadas aplicações de engenharia. O curso Sistemas Eletrônicos da Escola Politécnica forneceu conhecimentos diversos para a realização desse projeto, como técnicas de projeto de circuitos digitais, organização e arquitetura de computadores, modelagem em processamento de sinais, projeto de circuitos integrados dedicados, semi-dedicados e de sistemas integrados, métodos de concepção de projeto de engenharia e, principalmente, postura e atitude esperadas de profissionais engenheiros. Espera-se com este trabalho de formatura unificar esses e muitos outros conhecimentos adquiridos durante o curso, aplicando-os de maneira integrada e final, a fim de servir como ensaio para o futuro profissional de cada membro da equipe.

REFERÊNCIAS

BUSH, J. **Nyami**(Nyuzi): A synthesizable GPU architectural model for general-purpose and graphics-specific workloads. Binghamton: Universidade de Binghamton, 2015. Disponível em: <http://www.cs.binghamton.edu/~millerti/nyami-ispass2015.pdf>. Acesso em: 29 maio 2016.

FYKSE, E. **Performance comparison GPU, DSP and FPGA implementations of image processing and computer vision algorithms in embedded systems.** Noruega: NTNU, 2013.

JPR. **Nvidia increased everything in Q3 2016**. Total GPU shipments up a whopping 20.4%, from last quarter. California: Jon Peddie Research, 2016. Disponível em: <http://jonpeddie.com/publications/market_watch>. Acesso em: 31 de maio de 2016

JPR. **Qualcomm single largest proprietary GPU supplier**. Imagination technologies the leader in GPU IP, ARM and Vivante growing rapidly, according to latest report from Jon Peddie Research. California: Jon Peddie Research, 2016. Disponível em: <http://jonpeddie.com/press-releases/details/qualcomm-single-largest-proprietary-gpu-supplier-imagination-technologies-t/>. Acesso em: 31 de maio de 2016

KINGYENS, J., STEFFAN, J. G. **The Potential for a GPU-Like Overlay Architecture for FPGAs.** International Journal of Reconfigurable Computing, 2011. Article ID 514581

NOUVEAU. **Noveau**: Accelerated open source driver for nVidia cards. Disponível em: <https://nouveau.freedesktop.org/>. Acesso em: 5 de julho de 2016

NYUZI. **Nyuzi**. Binghamton: Universidade de Binghamton, 2015. Disponivel em: <http://nyuzi.org/>. Acesso em: 29 de maio de 2016.

OSI. **Open source initiative**. 2016. Disponível em: <http://https://opensource.org/>. Acesso em: 29 de maio de 2016.

SANKARALINGAM, K. **MIAOW**: An opensource GPGPU. Madison: Universidade de Wisconsin. Disponível em: <http://www.hotchips.org/wp-content/uploads/hc_archives/hc27/HC27.25-Tuesday-Epub/HC27.25.50-GPU-Epub/HC27.25.512-MIAOW-Balasubramaniam-UWisc-v1.2.pdf>. Acesso em: 29 de maio de 2016

WALLS, C. **Embedded Software**: the torks. Oxford: Elsevier, 2012.

APÊNDICE A

Critérios nas tabelas a seguir:

- Este critério (linha) é tantas vezes mais importante que aquele (coluna)
- Pesos calculados por média geométrica da linha seguida de normalização

Tabela do primeiro nível na hierarquia de necessidades

	Funcional	Integrável	Expansível	Sintetizável	Acessível	Média geométrica	Peso
Funcional	1	1	3	1	2	1.43	0.27
Integrável	1	1	3	1	2	1.43	0.27
Expansível	1/3	1/3	1	1	1	0.64	0.12
Sintetizável	1	1	1	1	2	1.15	0.22
Acessível	1/2	1/2	1	1/2	1	0.66	0.12

Tabela do segundo nível do item 'Integrável' na hierarquia de necessidades

	Comunicação	Saída Vídeo	OpenGL	Média geométrica	Peso
Comunicação	1	2	1	1.26	0.41
Saída Vídeo	1	1	1/2	0.79	0.26
OpenGL	1/2	2	1	1	0.33

Tabela do segundo nível do item 'Sintetizável' na hierarquia de necessidades

	ASIC	FPGA	Média geométrica	Peso
ASIC	1	1/3	0.58	0.25
FPGA	3	1	1.73	0.75

Tabela do segundo nível do item 'Acessível' na hierarquia de necessidades

	Plataforma acessível	Licença aberta	Média geométrica	Peso
Plataforma acessível	1	2	1.41	0.67
Licença aberta	0.5	1	0.71	0.33

APÊNDICE B

Arquivos relativos ao projeto de *hardware* da prova de conceito. Arquivos de autoria externa ao projeto estão com suas respectivas licenças de uso. Todos eles receberam alguma modificação dos autores do presente projeto e por isso encontram-se nesta seção.

```
------ framebuffer.vhd ------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity framebuffer is
    generic( w    :    positive :=640;
                  h    :    positive :=480);
    port(    clock    :    in std_logic;
             reset    :    in    std_logic :='0';
             enable:    in std_logic :='0';
             row    :    in    integer range 0 to h-1 :=0;
             col    :    in integer range 0 to w-1 :=0;
             r,g,b    :    out std_logic_vector(7 downto 0) := (others =>'0');

             sdram_cke       : out std_logic;          -- Clock-enable to SDRAM
             sdram_cs        : out std_logic;          -- Chip-select to SDRAM
             sdram_ras       : out std_logic;          -- SDRAM row address strobe
             sdram_cas       : out std_logic;          -- SDRAM column address strobe
             sdram_we        : out std_logic;          -- SDRAM write enable
             sdram_ba        : out std_logic_vector(1 downto 0);       -- SDRAM bank address
             sdram_addr      : out std_logic_vector(11 downto 0);   -- SDRAM row/column address
             sdram_data      : inout std_logic_vector(15 downto 0); -- Data to/from SDRAM
             sdram_dqmu      : out std_logic;          -- Enable upper-byte of SDRAM databus if true
             sdram_dqml      : out std_logic);         -- Enable lower-byte of SDRAM databus if
true));
end;

architecture framebuffer_rtl of framebuffer is

    component sdram_simple port(
                        -- Host side
                        clk_i                   : in std_logic;          -- Master clock
                        reset_i         : in std_logic := '0';    -- Reset, active high
                        refresh_i       : in std_logic := '0';    -- Initiate a refresh cycle, active
high
                        rw_i            : in std_logic := '0';    -- Initiate a read or write operation,
active high
                        we_i            : in std_logic := '0';    -- Write enable, active low
                        addr_i          : in std_logic_vector(21 downto 0) := (others => '0');   --
Address from host to SDRAM
                        data_i          : in std_logic_vector(15 downto 0) := (others => '0');   -- Data
from host to SDRAM
                        ub_i            : in std_logic;           -- Data upper byte enable, active low
                        lb_i            : in std_logic;           -- Data lower byte enable, active low
                        ready_o         : out std_logic := '0';   -- Set to '1' when the memory is ready
                        done_o          : out std_logic := '0';   -- Read, write, or refresh, operation
is done
                        data_o          : out std_logic_vector(15 downto 0);   -- Data from SDRAM to host

                        -- SDRAM side
                        sdCke_o         : out std_logic;          -- Clock-enable to SDRAM
                        sdCe_bo         : out std_logic;          -- Chip-select to SDRAM
                        sdRas_bo        : out std_logic;          -- SDRAM row address strobe
                        sdCas_bo        : out std_logic;          -- SDRAM column address strobe
                        sdWe_bo         : out std_logic;          -- SDRAM write enable
                        sdBs_o          : out std_logic_vector(1 downto 0);       -- SDRAM bank address
                        sdAddr_o        : out std_logic_vector(11 downto 0);   -- SDRAM row/column
address
                        sdData_io       : inout std_logic_vector(15 downto 0); -- Data to/from SDRAM
                        sdDqmh_o        : out std_logic;          -- Enable upper-byte of SDRAM databus
if true
```

```
                        sdDqml_o        : out std_logic);        -- Enable lower-byte of SDRAM databus
  if true
      end component;

      signal S_ADDRESS : std_logic_vector(21 downto 0):=(others =>'0');
      signal S_DATA_OUTPUT : std_logic_vector(15 downto 0):=(others =>'0');
      signal S_MEMORY_READY : std_logic:='1';
      signal S_DONE : std_logic:='0';
      signal access_next_pixel : std_logic :='0'; -- flag para acessar proximo pixel com row e col
      signal next_pixel_address : std_logic_vector(21 downto 0) :=(others =>'0');
      signal S_READ_NOW : std_logic :='0';
      signal S_REFRESH : std_logic :='0';

      begin
      MEMORY: sdram_simple port map(
                        -- Host side
                        clk_i =>clock,
                        reset_i =>reset,
                        rw_i =>S_READ_NOW,
                        refresh_i=>S_REFRESH,
                        we_i =>'1',
                        addr_i =>S_ADDRESS,
                        ub_i =>'1',
                        lb_i =>'0',
                        done_o =>S_DONE,
                        ready_o =>S_MEMORY_READY,
                        data_o =>S_DATA_OUTPUT,

                        sdCke_o=>sdram_cke,
                        sdCe_bo=>sdram_cs,
                        sdRas_bo=>sdram_ras,
                        sdCas_bo=>sdram_cas,
                        sdWe_bo=>sdram_we,
                        sdBs_o=>sdram_ba,
                        sdAddr_o=>sdram_addr,
                        sdData_io=>sdram_data,
                        sdDqmh_o=>sdram_dqmu,
                        sdDqml_o=>sdram_dqml);

      process(row,col,S_MEMORY_READY)
      begin
            if S_MEMORY_READY = '0' then
                    access_next_pixel<='0';
                    S_READ_NOW <='0';
            end if;
            access_next_pixel<='1';
            next_pixel_address<=std_logic_vector(to_unsigned(row*w+col,next_pixel_address'length));
            S_READ_NOW <='1';
            --next_pixel_address(21 downto 20)<="00";
            --next_pixel_address(19 downto 8)<=std_logic_vector(to_unsigned(row,12));
            --next_pixel_address(7 downto 0)<=std_logic_vector(to_unsigned(col,8));
      end process;


      process(S_DONE,enable)
      begin
      if enable = '1' then
            if S_DONE'event and S_DONE='1' then
                    r(7 downto 3)<=S_DATA_OUTPUT(4 downto 0);
                    r(2 downto 0)<=(others=>S_DATA_OUTPUT(4));
                    g(7 downto 2)<=S_DATA_OUTPUT(10 downto 5);
                    g(1 downto 0)<=(others=>S_DATA_OUTPUT(10));
                    b(7 downto 3)<=S_DATA_OUTPUT(15 downto 11);
                    b(2 downto 0)<=(others=>S_DATA_OUTPUT(15));
            end if;
      else
            r(7 downto 0)<=(others=>'0');
            g(7 downto 0)<=(others=>'0');
            b(7 downto 0)<=(others=>'0');
      end if;
      end process;


end framebuffer_rtl;
------ Fim do arquivo framebuffer.vhd ------
```

```
------ gpu.vhd ------
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity gpu is
    port(   reset : in STD_LOGIC;
            main_clock: in STD_LOGIC;

            sdram_cke       : out std_logic;          -- Clock-enable to SDRAM
            sdram_cs        : out std_logic;          -- Chip-select to SDRAM
            sdram_ras       : out std_logic;          -- SDRAM row address strobe
            sdram_cas       : out std_logic;          -- SDRAM column address strobe
            sdram_we        : out std_logic;          -- SDRAM write enable
            sdram_ba        : out std_logic_vector(1 downto 0);       -- SDRAM bank address
            sdram_addr      : out std_logic_vector(11 downto 0);   -- SDRAM row/column address
            sdram_data      : inout std_logic_vector(15 downto 0); -- Data to/from SDRAM
            sdram_dqmu      : out std_logic;          -- Enable upper-byte of SDRAM databus if true
            sdram_dqml      : out std_logic;          -- Enable lower-byte of SDRAM databus if true);

            vga_clk    : out std_logic;
            vga_sync   : out std_logic;
            vga_blank: out std_logic;
            vga_hs     : out std_logic;
            vga_vs     : out std_logic;

            r,g,b           : out std_logic_vector(9 downto 0) := (others=>'0'));
end;

architecture gpu_structure of gpu is

--component framebuffer_sdram_controller
--    generic( w    :    positive :=640;
--                       h    :    positive :=480);
--    port(   clock    :    in std_logic;
--               reset    :    in    std_logic;
--               enable:    in std_logic;
--               row    :    in    integer range 0 to w-1;
--               col    :    in integer range 0 to h-1;
--               r,g,b    :    out std_logic_vector(7 downto 0);
--
--               sdram_ncs    :    out    std_logic;
--               sdram_nras    :    out    std_logic;
--               sdram_ncas    :    out    std_logic;
--               sdram_nwe    :    out    std_logic;
--               sdram_dqm    :    out    std_logic_vector(1 downto 0);
--               sdram_addr    :    out    std_logic_vector(11 downto 0);
--               sdram_bank    :    out    std_logic_vector(1 downto 0);
--               sdram_clk    :    out    std_logic;
--               sdram_cke    :    out    std_logic;
--               sdram_dq        :    in          std_logic_vector(15 downto 0)
--               );
--end component;
component framebuffer
               generic( w    :    positive :=640;
                        h    :    positive :=480);
               port(           clock    :    in std_logic;
                               reset    :    in    std_logic :='0';
                               enable:    in std_logic :='0';
                               row    :    in    integer range 0 to h-1 :=0;
                               col    :    in integer range 0 to w-1 :=0;
                               r,g,b    :    out std_logic_vector(7 downto 0) := (others
=>'1');

                               sdram_cke       : out std_logic;          -- Clock-enable to
SDRAM
                               sdram_cs        : out std_logic;          -- Chip-select to
SDRAM
                               sdram_ras       : out std_logic;          -- SDRAM row address
strobe
                               sdram_cas       : out std_logic;          -- SDRAM column
address strobe
```

```
                                        sdram_we        : out std_logic;          -- SDRAM write enable
                                        sdram_ba        : out std_logic_vector(1 downto 0);       --
 SDRAM bank address

                                        sdram_addr      : out std_logic_vector(11 downto 0);   -- SDRAM
 row/column address

                                        sdram_data      : inout std_logic_vector(15 downto 0); -- Data
 to/from SDRAM

                                        sdram_dqmu      : out std_logic;          -- Enable upper-byte
 of SDRAM databus if true

                                        sdram_dqml      : out std_logic);         -- Enable lower-byte
 of SDRAM databus if true
 end component;

 component hw_image_generator
     GENERIC(
          pixels_y :  INTEGER := 309;   --row that first color will persist until
          pixels_x :  INTEGER := 680);  --column that first color will persist until
    PORT(
          disp_ena :  IN   STD_LOGIC;  --display enable ('1' = display time, '0' = blanking time)
          row      :  IN   INTEGER; --row pixel coordinate
          column   :  IN   INTEGER; --column pixel coordinate
          red      :  OUT  STD_LOGIC_VECTOR(7 DOWNTO 0) := (OTHERS => '0');  --red magnitude output to DAC
          green    :  OUT  STD_LOGIC_VECTOR(7 DOWNTO 0) := (OTHERS => '0');  --green magnitude output to DAC
          blue     :  OUT  STD_LOGIC_VECTOR(7 DOWNTO 0) := (OTHERS => '0')); --blue magnitude output to DAC
end component;


 component vga_controller
     GENERIC(
          h_pulse  :  INTEGER   := 96;   --horizontal sync pulse width in pixels
          h_bp     :  INTEGER   := 48;   --horizontal back porch width in pixels
          h_pixels :  INTEGER   := 640;  --horizontal display width in pixels
          h_fp     :  INTEGER   := 16;   --horizontal front porch width in pixels
          h_pol    :  STD_LOGIC := '0';   --horizontal sync pulse polarity (1 = positive, 0 = negative)
          v_pulse  :  INTEGER   := 2;       --vertical sync pulse width in rows
          v_bp     :  INTEGER   := 33;      --vertical back porch width in rows
          v_pixels :  INTEGER   := 480;  --vertical display width in rows
          v_fp     :  INTEGER   := 10;      --vertical front porch width in rows
          v_pol    :  STD_LOGIC := '0');  --vertical sync pulse polarity (1 = positive, 0 = negative)
  PORT(
          pixel_clk :  IN   STD_LOGIC;  --pixel clock at frequency of VGA mode being used
          reset_n   :  IN   STD_LOGIC;  --active low asynchronous reset
          h_sync    :  OUT  STD_LOGIC;  --horizontal sync pulse
          v_sync    :  OUT  STD_LOGIC;  --vertical sync pulse
          disp_ena  :  OUT  STD_LOGIC;  --display enable ('1' = display time, '0' = blanking time)
          column    :  OUT  INTEGER range 0 to h_pixels-1 :=0; --horizontal pixel coordinate
          row       :  OUT  INTEGER range 0 to v_pixels-1 :=0; --vertical pixel coordinate
          n_blank   :  OUT  STD_LOGIC;  --direct blacking output to DAC
          n_sync    :  OUT  STD_LOGIC); --sync-on-green output to DAC
end component;


component pixelclock
    PORT
    (
          inclke          : IN STD_LOGIC  := '0';
          c0       : OUT STD_LOGIC ;
          c1       : OUT STD_LOGIC
    );
end component;


signal S_DISP_EN: STD_LOGIC :='1';
signal S_COL: INTEGER range 0 to 639 :=0;
signal S_ROW: INTEGER range 0 to 479 :=0;
signal S_PIXEL_CLOCK: STD_LOGIC;
signal S_MEMORY_CLOCK: STD_LOGIC;


begin
--    MEMORY: hw_image_generator port map(
--                         disp_ena=>S_DISP_EN,
--                         row=>S_ROW,
--                         column=>S_COL,
--                         red=>r(9 downto 2),
--                         green=>g(9 downto 2),
```

```
--                          blue=>b(9 downto 2)
--                          );
    MEMORY: framebuffer port map(
                            clock =>S_PIXEL_CLOCK,
                            reset =>reset,
                            enable =>S_DISP_EN,
                            row =>S_ROW,
                            col =>S_COL,

                            sdram_cke=>sdram_cke,
                            sdram_cs=>sdram_cs,
                            sdram_ras=>sdram_ras,
                            sdram_cas=>sdram_cas,
                            sdram_we=>sdram_we,
                            sdram_ba=>sdram_ba,
                            sdram_addr=>sdram_addr,
                            sdram_data=>sdram_data,
                            sdram_dqmu=>sdram_dqmu,
                            sdram_dqml=>sdram_dqml,

                            r=>r(9 downto 2),
                            g=>g(9 downto 2),
                            b=>b(9 downto 2)
                            );


    VIDEO_CONTROLLER: vga_controller port map(
                           disp_ena=>S_DISP_EN,
                           column=>S_COL,
                           row=>S_ROW,
                           pixel_clk=>S_PIXEL_CLOCK,
                           reset_n=>"NOT"(reset),
                           h_sync=>vga_hs,
                           v_sync=>vga_vs,
                           n_blank=>vga_blank,
                           n_sync=>vga_sync);

    PLL_PIXEL_CLOCK: pixelclock port map(inclk0=>main_clock,c0=>S_PIXEL_CLOCK, c1=>S_MEMORY_CLOCK);

    vga_clk<=S_PIXEL_CLOCK;

end gpu_structure;
------ Fin du gpu.vhd ------
```

```
------ hw_image_generator.vhd ------
--------------------------------------------------------------------------------
--
--   FileName:            hw_image_generator.vhd
--   Dependencies:        none
--   Design Software:  Quartus II 64-bit Version 12.1 Build 177 SJ Full Version
--
--   HDL CODE IS PROVIDED "AS IS."  DIGI-KEY EXPRESSLY DISCLAIMS ANY
--   WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT
--   LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
--   PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL DIGI-KEY
--   BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL
--   DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF
--   PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS
--   BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF),
--   ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.
--
--   Version History
--   Version 1.0 05/10/2013 Scott Larson
--     Initial Public Release
--
--------------------------------------------------------------------------------


LIBRARY ieee;
USE ieee.std_logic_1164.all;


ENTITY hw_image_generator IS
  GENERIC(
        pixels_y :  INTEGER := 300;   --row that first color will persist until
        pixels_x :  INTEGER := 600);  --column that first color will persist until
  PORT(
        disp_ena :  IN   STD_LOGIC;  --display enable ('1' = display time, '0' = blanking time)
        row      :  IN   INTEGER; --row pixel coordinate
        column   :  IN   INTEGER; --column pixel coordinate
        red      :  OUT  STD_LOGIC_VECTOR(7 DOWNTO 0) := (OTHERS => '0');  --red magnitude output to DAC
        green    :  OUT  STD_LOGIC_VECTOR(7 DOWNTO 0) := (OTHERS => '0');  --green magnitude output to DAC
        blue     :  OUT  STD_LOGIC_VECTOR(7 DOWNTO 0) := (OTHERS => '0')); --blue magnitude output to DAC
END hw_image_generator;


ARCHITECTURE behavior OF hw_image_generator IS
BEGIN
  PROCESS(disp_ena, row, column)
  BEGIN


        IF(disp_ena = '1') THEN           --display time
        IF(row < pixels_y AND column < pixels_x) THEN
        red <= (OTHERS => '0');
        green  <= (OTHERS => '0');
        blue <= (OTHERS => '1');
        ELSE
        red <= (OTHERS => '1');
        green  <= (OTHERS => '1');
        blue <= (OTHERS => '0');
        END IF;
        ELSE                              --blanking time
        red <= (OTHERS => '0');
        green <= (OTHERS => '0');
        blue <= (OTHERS => '0');
        END IF;

  END PROCESS;
END behavior;

------ Fim do hw_image_generator ------
```

```
------ pixelclock.vhd ------
-- megafunction wizard: %ALTPLL%
-- GENERATION: STANDARD
-- VERSION: WM1.0
-- MODULE: altpll


-- ============================================================
-- File Name: pixelclock.vhd
-- Megafunction Name(s):
--                altpll
--
-- Simulation Library Files(s):
--                altera_mf
-- ============================================================
-- ************************************************************
-- THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
--
-- 13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
-- ************************************************************



--Copyright (C) 1991-2013 Altera Corporation
--Your use of Altera Corporation's design tools, logic functions
--and other software and tools, and its AMPP partner logic
--functions, and any output files from any of the foregoing
--(including device programming or simulation files), and any
--associated documentation or information are expressly subject
--to the terms and conditions of the Altera Program License
--Subscription Agreement, Altera MegaCore Function License
--Agreement, or other applicable license agreement, including,
--without limitation, that your use is for the sole purpose of
--programming logic devices manufactured by Altera and sold by
--Altera or its authorized distributors.  Please refer to the
--applicable agreement for further details.



LIBRARY ieee;
USE ieee.std_logic_1164.all;


LIBRARY altera_mf;
USE altera_mf.all;


ENTITY pixelclock IS
    PORT
    (
        inclk0          : IN STD_LOGIC  := '0';
        c0      : OUT STD_LOGIC ;
        c1      : OUT STD_LOGIC
    );
END pixelclock;



ARCHITECTURE SYN OF pixelclock IS

    SIGNAL sub_wire0    : STD_LOGIC_VECTOR (5 DOWNTO 0);
    SIGNAL sub_wire1    : STD_LOGIC ;
    SIGNAL sub_wire2    : STD_LOGIC ;
    SIGNAL sub_wire3    : STD_LOGIC ;
    SIGNAL sub_wire4    : STD_LOGIC_VECTOR (1 DOWNTO 0);
    SIGNAL sub_wire5_bv    : BIT_VECTOR (0 DOWNTO 0);
    SIGNAL sub_wire5    : STD_LOGIC_VECTOR (0 DOWNTO 0);
```

```
    COMPONENT altpll
    GENERIC (
          clk0_divide_by            : NATURAL;
          clk0_duty_cycle           : NATURAL;
          clk0_multiply_by          : NATURAL;
          clk0_phase_shift          : STRING;
          clk1_divide_by            : NATURAL;
          clk1_duty_cycle           : NATURAL;
          clk1_multiply_by          : NATURAL;
          clk1_phase_shift          : STRING;
          compensate_clock          : STRING;
          inclk0_input_frequency              : NATURAL;
          intended_device_family              : STRING;
          lpm_hint          : STRING;
          lpm_type          : STRING;
          operation_mode            : STRING;
          port_activeclock          : STRING;
          port_areset       : STRING;
          port_clkbad0      : STRING;
          port_clkbad1      : STRING;
          port_clkloss      : STRING;
          port_clkswitch            : STRING;
          port_configupdate         : STRING;
          port_fbin         : STRING;
          port_inclk0       : STRING;
          port_inclk1       : STRING;
          port_locked       : STRING;
          port_pfdena       : STRING;
          port_phasecounterselect             : STRING;
          port_phasedone            : STRING;
          port_phasestep            : STRING;
          port_phaseupdown          : STRING;
          port_pllena       : STRING;
          port_scanaclr     : STRING;
          port_scanclk      : STRING;
          port_scanclkena           : STRING;
          port_scandata     : STRING;
          port_scandataout          : STRING;
          port_scandone     : STRING;
          port_scanread     : STRING;
          port_scanwrite            : STRING;
          port_clk0         : STRING;
          port_clk1         : STRING;
          port_clk2         : STRING;
          port_clk3         : STRING;
          port_clk4         : STRING;
          port_clk5         : STRING;
          port_clkena0      : STRING;
          port_clkena1      : STRING;
          port_clkena2      : STRING;
          port_clkena3      : STRING;
          port_clkena4      : STRING;
          port_clkena5      : STRING;
          port_extclk0      : STRING;
          port_extclk1      : STRING;
          port_extclk2      : STRING;
          port_extclk3      : STRING
    );
    PORT (
                clk     : OUT STD_LOGIC_VECTOR (5 DOWNTO 0);
                inclk     : IN STD_LOGIC_VECTOR (1 DOWNTO 0)
    );
    END COMPONENT;


BEGIN
    sub_wire5_bv(0 DOWNTO 0) <= "0";
    sub_wire5     <= To_stdlogicvector(sub_wire5_bv);
    sub_wire2     <= sub_wire0(1);
    sub_wire1     <= sub_wire0(0);
    c0   <= sub_wire1;
    c1   <= sub_wire2;
```

```
    sub_wire3    <= inclk0;
    sub_wire4    <= sub_wire5(0 DOWNTO 0) & sub_wire3;


    altpll_component : altpll
    GENERIC MAP (
          clk0_divide_by => 2,
          clk0_duty_cycle => 50,
          clk0_multiply_by => 1,
          clk0_phase_shift => "0",
          clk1_divide_by => 50000000,
          clk1_duty_cycle => 50,
          clk1_multiply_by => 133333333,
          clk1_phase_shift => "0",
          compensate_clock => "CLK0",
          inclk0_input_frequency => 20000,
          intended_device_family => "Cyclone II",
          lpm_hint => "CBX_MODULE_PREFIX=pixelclock",
          lpm_type => "altpll",
          operation_mode => "NORMAL",
          port_activeclock => "PORT_UNUSED",
          port_areset => "PORT_UNUSED",
          port_clkbad0 => "PORT_UNUSED",
          port_clkbad1 => "PORT_UNUSED",
          port_clkloss => "PORT_UNUSED",
          port_clkswitch => "PORT_UNUSED",
          port_configupdate => "PORT_UNUSED",
          port_fbin => "PORT_UNUSED",
          port_inclk0 => "PORT_USED",
          port_inclk1 => "PORT_UNUSED",
          port_locked => "PORT_UNUSED",
          port_pfdena => "PORT_UNUSED",
          port_phasecounterselect => "PORT_UNUSED",
          port_phasedone => "PORT_UNUSED",
          port_phasestep => "PORT_UNUSED",
          port_phaseupdown => "PORT_UNUSED",
          port_pllena => "PORT_UNUSED",
          port_scanaclr => "PORT_UNUSED",
          port_scanclk => "PORT_UNUSED",
          port_scanclkena => "PORT_UNUSED",
          port_scandata => "PORT_UNUSED",
          port_scandataout => "PORT_UNUSED",
          port_scandone => "PORT_UNUSED",
          port_scanread => "PORT_UNUSED",
          port_scanwrite => "PORT_UNUSED",
          port_clk0 => "PORT_USED",
          port_clk1 => "PORT_USED",
          port_clk2 => "PORT_UNUSED",
          port_clk3 => "PORT_UNUSED",
          port_clk4 => "PORT_UNUSED",
          port_clk5 => "PORT_UNUSED",
          port_clkena0 => "PORT_UNUSED",
          port_clkena1 => "PORT_UNUSED",
          port_clkena2 => "PORT_UNUSED",
          port_clkena3 => "PORT_UNUSED",
          port_clkena4 => "PORT_UNUSED",
          port_clkena5 => "PORT_UNUSED",
          port_extclk0 => "PORT_UNUSED",
          port_extclk1 => "PORT_UNUSED",
          port_extclk2 => "PORT_UNUSED",
          port_extclk3 => "PORT_UNUSED"
    )
    PORT MAP (
          inclk => sub_wire4,
          clk => sub_wire0
    );



END SYN;
------ Final omitido para simplificação. Arquivo pixelclock.vhd ------
```

```
------ sdram_simple.vhd ------
-- The MIT License (MIT)
--
-- Copyright (c) 2014 Matthew Hagerty
--
-- Permission is hereby granted, free of charge, to any person obtaining a copy
-- of this software and associated documentation files (the "Software"), to deal
-- in the Software without restriction, including without limitation the rights
-- to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
-- copies of the Software, and to permit persons to whom the Software is
-- furnished to do so, subject to the following conditions:
--
-- The above copyright notice and this permission notice shall be included in all
-- copies or substantial portions of the Software.
--
-- THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
-- IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
-- FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
-- AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
-- LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
-- OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
-- SOFTWARE.


-- Simple SDRAM Controller for Winbond W9812G6JH-75
--
-- Matthew Hagerty, copyright 2014
--
-- Create Date:  18:22:09 March 18, 2014
-- Module Name:  sdram_simple - Rtl
--
-- Change Log:
--
-- Jan 28, 2016
--       Changed to use positive clock edge.
--       Buffered output (read) data, sampled during RAS2.
--       Removed unused signals for features that were not implemented.
--       Changed tabs to space.
--
-- March 19, 2014
--       Initial implementation.



library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.numeric_std.all;
use IEEE.math_real.all;


entity sdram_simple is
   port(
        -- Host side
        clk_i             : in std_logic;           -- Master clock
        reset_i           : in std_logic := '0';    -- Reset, active high
        refresh_i         : in std_logic := '0';    -- Initiate a refresh cycle, active high
        rw_i              : in std_logic := '0';    -- Initiate a read or write operation, active high
        we_i              : in std_logic := '0';    -- Write enable, active low
        addr_i            : in std_logic_vector(21 downto 0) := (others => '0');   -- Address from host to
SDRAM
        data_i            : in std_logic_vector(15 downto 0) := (others => '0');   -- Data from host to
SDRAM
        ub_i              : in std_logic;           -- Data upper byte enable, active low
        lb_i              : in std_logic;           -- Data lower byte enable, active low
        ready_o           : out std_logic := '0';   -- Set to '1' when the memory is ready
        done_o            : out std_logic := '0';   -- Read, write, or refresh, operation is done
        data_o            : out std_logic_vector(15 downto 0);   -- Data from SDRAM to host


        -- SDRAM side
```

```
        sdCke_o         : out std_logic;          -- Clock-enable to SDRAM
        sdCe_bo         : out std_logic;          -- Chip-select to SDRAM
        sdRas_bo        : out std_logic;          -- SDRAM row address strobe
        sdCas_bo        : out std_logic;          -- SDRAM column address strobe
        sdWe_bo         : out std_logic;          -- SDRAM write enable
        sdBs_o          : out std_logic_vector(1 downto 0);        -- SDRAM bank address
        sdAddr_o        : out std_logic_vector(11 downto 0);   -- SDRAM row/column address
        sdData_io       : inout std_logic_vector(15 downto 0); -- Data to/from SDRAM
        sdDqmh_o        : out std_logic;          -- Enable upper-byte of SDRAM databus if true
        sdDqml_o        : out std_logic           -- Enable lower-byte of SDRAM databus if true
    );
end entity;

architecture rtl of sdram_simple is


    -- SDRAM controller states.
    type fsm_state_type is (
    ST_INIT_WAIT, ST_INIT_PRECHARGE, ST_INIT_REFRESH1, ST_INIT_MODE, ST_INIT_REFRESH2,
    ST_IDLE, ST_REFRESH, ST_ACTIVATE, ST_RCD, ST_RW, ST_RAS1, ST_RAS2, ST_PRECHARGE);
    signal state_r, state_x : fsm_state_type := ST_INIT_WAIT;



    -- SDRAM mode register data sent on the address bus.
    --
    -- | A11-A10 |A9       | A8  A7 | A6 A5 A4 |     A3    | A2 A1 A0 |
    -- | reserved| wr burst |reserved| CAS Ltncy|addr mode| burst len|
    --   0  0  0  0        0   0    0  1  0          0        0  0  0
    constant MODE_REG : std_logic_vector(11 downto 0) := "00" & "0" & "00" & "010" & "0" & "000";


    -- SDRAM commands combine SDRAM inputs: cs, ras, cas, we.
    subtype cmd_type is unsigned(3 downto 0);
    constant CMD_ACTIVATE   : cmd_type := "0011";
    constant CMD_PRECHARGE  : cmd_type := "0010";
    constant CMD_WRITE      : cmd_type := "0100";
    constant CMD_READ       : cmd_type := "0101";
    constant CMD_MODE       : cmd_type := "0000";
    constant CMD_NOP        : cmd_type := "0111";
    constant CMD_REFRESH    : cmd_type := "0001";


    signal cmd_r            : cmd_type;
    signal cmd_x            : cmd_type;


    signal bank_s           : std_logic_vector(1 downto 0);
    signal row_s            : std_logic_vector(11 downto 0);
    signal col_s            : std_logic_vector(7 downto 0);
    signal addr_r           : std_logic_vector(11 downto 0);
    signal addr_x           : std_logic_vector(11 downto 0);   -- SDRAM row/column address.
    signal sd_dout_r        : std_logic_vector(15 downto 0);
    signal sd_dout_x        : std_logic_vector(15 downto 0);
    signal sd_busdir_r      : std_logic;
    signal sd_busdir_x      : std_logic;


    signal timer_r, timer_x         : natural range 0 to 20000 := 0;
    signal refcnt_r, refcnt_x       : natural range 0 to 8 := 0;


    signal bank_r, bank_x           : std_logic_vector(1 downto 0);
    signal cke_r, cke_x             : std_logic;
    signal sd_dqmu_r, sd_dqmu_x     : std_logic;
    signal sd_dqml_r, sd_dqml_x     : std_logic;
    signal ready_r, ready_x         : std_logic;


    -- Data buffer for SDRAM to Host.
    signal buf_dout_r, buf_dout_x : std_logic_vector(15 downto 0);
```

```
begin

   -- All signals to SDRAM buffered.

   (sdCe_bo, sdRas_bo, sdCas_bo, sdWe_bo) <= cmd_r;   -- SDRAM operation control bits
   sdCke_o       <= cke_r;         -- SDRAM clock enable
   sdBs_o        <= bank_r;        -- SDRAM bank address
   sdAddr_o      <= addr_r;        -- SDRAM address
   sdData_io   <= sd_dout_r when sd_busdir_r = '1' else (others => 'Z');   -- SDRAM data bus.
   sdDqmh_o      <= sd_dqmu_r;  -- SDRAM high data byte enable, active low
   sdDqml_o      <= sd_dqml_r;  -- SDRAM low date byte enable, active low


   -- Signals back to host.
   ready_o <= ready_r;
   data_o <= buf_dout_r;


   --   21 20 | 19 18 17 16 15 14 13 12 11 10 09 08 | 07 06 05 04 03 02 01 00 |
   -- BS0 BS1 | ROW (A11-A0)  4096 rows             | COL (A7-A0)  256 cols   |
   bank_s <= addr_i(21 downto 20);
   row_s <= addr_i(19 downto 8);
   col_s <= addr_i(7 downto 0);



   process (
   state_r, timer_r, refcnt_r, cke_r, addr_r, sd_dout_r, sd_busdir_r, sd_dqmu_r, sd_dqml_r, ready_r,
   bank_s, row_s, col_s,
   rw_i, refresh_i, addr_i, data_i, we_i, ub_i, lb_i,
   buf_dout_r, sdData_io)
   begin

        state_x  <= state_r;      -- Stay in the same state unless changed.
        timer_x  <= timer_r;      -- Hold the cycle timer by default.
        refcnt_x <= refcnt_r;     -- Hold the refresh timer by default.
        cke_x    <= cke_r;        -- Stay in the same clock mode unless changed.
        cmd_x    <= CMD_NOP;      -- Default to NOP unless changed.
        bank_x   <= bank_r;       -- Register the SDRAM bank.
        addr_x   <= addr_r;       -- Register the SDRAM address.
        sd_dout_x   <= sd_dout_r;         -- Register the SDRAM write data.
        sd_busdir_x <= sd_busdir_r;   -- Register the SDRAM bus tristate control.
        sd_dqmu_x   <= sd_dqmu_r;
        sd_dqml_x   <= sd_dqml_r;
        buf_dout_x  <= buf_dout_r;        -- SDRAM to host data buffer.


        ready_x  <= ready_r;      -- Always ready unless performing initialization.
        done_o   <= '0';          -- Done tick, single cycle.


        if timer_r /= 0 then
        timer_x <= timer_r - 1;
        else


        cke_x    <= '1';
        bank_x   <= bank_s;
        addr_x   <= "0000" & col_s;   -- A10 low for rd/wr commands to suppress auto-precharge.
        sd_dqmu_x   <= '0';
        sd_dqml_x   <= '0';


        case state_r is


        when ST_INIT_WAIT =>


        -- 1. Wait 200us with DQM signals high, cmd NOP.
```

```
-- 2. Precharge all banks.
-- 3. Eight refresh cycles.
-- 4. Set mode register.
-- 5. Eight refresh cycles.


state_x <= ST_INIT_PRECHARGE;
timer_x <= 20000;          -- Wait 200us (20,000 cycles).
--timer_x <= 2;           -- for simulation
sd_dqmu_x <= '1';
sd_dqml_x <= '1';


when ST_INIT_PRECHARGE =>


state_x <= ST_INIT_REFRESH1;
refcnt_x <= 8;             -- Do 8 refresh cycles in the next state.
--refcnt_x <= 2;          -- for simulation
cmd_x <= CMD_PRECHARGE;
timer_x <= 2;              -- Wait 2 cycles plus state overhead for 20ns Trp.
bank_x <= "00";
addr_x(10) <= '1';         -- Precharge all banks.


when ST_INIT_REFRESH1 =>


if refcnt_r = 0 then
        state_x <= ST_INIT_MODE;
else
        refcnt_x <= refcnt_r - 1;
        cmd_x <= CMD_REFRESH;
        timer_x <= 7;              -- Wait 7 cycles plus state overhead for 70ns refresh.
end if;


when ST_INIT_MODE =>


state_x <= ST_INIT_REFRESH2;
refcnt_x <= 8;             -- Do 8 refresh cycles in the next state.
--refcnt_x <= 2;          -- for simulation
bank_x <= "00";
addr_x <= MODE_REG;
cmd_x <= CMD_MODE;
timer_x <= 2;              -- Trsc == 2 cycles after issuing MODE command.


when ST_INIT_REFRESH2 =>


if refcnt_r = 0 then
        state_x <= ST_IDLE;
        ready_x <= '1';
else
        refcnt_x <= refcnt_r - 1;
        cmd_x <= CMD_REFRESH;
        timer_x <= 7;              -- Wait 7 cycles plus state overhead for 70ns refresh.
end if;


--
-- Normal Operation
--


-- Trc  - 70ns - Attive to active command.
-- Trcd - 20ns - Active to read/write command.
-- Tras - 50ns - Active to precharge command.
-- Trp  - 20ns - Precharge to active command.
-- TCas - 2clk - Read/write to data out.
--
--      |<-----------    Trc     ------------>|
--      |<----------- Tras ---------->|
--      |<- Trcd  ->|<- TCas  ->|         |<-  Trp  ->|
```

```
--  T0__  T1__  T2__  T3__  T4__  T5__  T6__  T7__  T8__  T9__ T10__
-- __|  |__|  |__|  |__|  |__|  |__|  |__|  |__|  |__|  |__|  |__
-- IDLE  ACTVT  NOP  RD/WR  NOP   NOP  PRECG IDLE  ACTVT
--       --<Row>--------<Col>------------<Bank>--------<Row>--
--       ---------------<A10>-------------<A10>-------------------
-- ------------------<Din>------------<Dout>--------
--       ---------------<DQM>---------------
--       --<Refsh>-------------
--
-- A10 during rd/wr : 0 = disable auto-precharge, 1 = enable auto-precharge.
-- A10 during precharge: 0 = single bank, 1 = all banks.


--  T0__  T1__  T2__  T3__  T4__  T5__  T6__  T7__  T8__  T9__ T16__
-- __|  |__|  |__|  |__|  |__|  |__|  |__|  |__|  |__|  |__|  |__
-- IDLE  ACTVT  NOP  RD/WR  NOP   NOP  PRECG IDLE  ACTVT
--  T0__  T1__  T2__  T3__  T4__  T5__  T6__  T7__  T8__  T9__ T16__
-- __|  |__|  |__|  |__|  |__|  |__|  |__|  |__|  |__|  |__|  |__
--       IDLE  ACTVT  NOP  RD/WR  NOP   NOP  PRECG IDLE  ACTVT


when ST_IDLE =>
-- 60ns since activate when coming from PRECHARGE state.
-- 10ns since PRECHARGE.  Trp == 20ns min.
if rw_i = '1' then
        state_x <= ST_ACTIVATE;
        cmd_x <= CMD_ACTIVATE;
        addr_x <= row_s;          -- Set bank select and row on activate command.
elsif refresh_i = '1' then
        state_x <= ST_REFRESH;
        cmd_x <= CMD_REFRESH;
        timer_x <= 7;             -- Wait 7 cycles plus state overhead for 70ns refresh.
end if;


when ST_REFRESH =>


state_x <= ST_IDLE;
done_o <= '1';


when ST_ACTIVATE =>
-- Trc (Active to Active Command Period) is 65ns min.
-- 70ns since activate when coming from PRECHARGE -> IDLE states.
-- 20ns since PRECHARGE.
-- ACTIVATE command is presented to the SDRAM.  The command out of this
-- state will be NOP for one cycle.
state_x <= ST_RCD;
sd_dout_x <= data_i;      -- Register any write data, even if not used.


when ST_RCD =>
-- 10ns since activate.
-- Trcd == 20ns min.  The clock is 10ns, so the requirement is satisfied by this state.
-- READ or WRITE command will be active in the next cycle.
state_x <= ST_RW;


if we_i = '0' then
        cmd_x <= CMD_WRITE;
        sd_busdir_x <= '1';       -- The SDRAM latches the input data with the command.
        sd_dqmu_x <= ub_i;
        sd_dqml_x <= lb_i;
else
        cmd_x <= CMD_READ;
end if;


when ST_RW =>
-- 20ns since activate.
-- READ or WRITE command presented to SDRAM.
state_x <= ST_RAS1;
sd_busdir_x <= '0';
```

```
        when ST_RAS1 =>
        -- 38ns since activate.
        state_x <= ST_RAS2;


        when ST_RAS2 =>
        -- 48ns since activate.
        -- Tras (Active to precharge Command Period) 45ns min.
        -- PRECHARGE command will be active in the next cycle.
        state_x <= ST_PRECHARGE;
        cmd_x <= CMD_PRECHARGE;
        addr_x(10) <= '1';        -- Precharge all banks.
        buf_dout_x <= sdData_io;


        when ST_PRECHARGE =>
        -- 58ns since activate.
        -- PRECHARGE presented to SDRAM.
        state_x <= ST_IDLE;
        done_o <= '1';            -- Read data is ready and should be latched by the host.
        timer_x <= 1;             -- Buffer to make sure host takes down memory request before going IDLE.


        end case;
        end if;
    end process;


    process (clk_i)
    begin
        if rising_edge(clk_i) then
        if reset_i = '1' then
        state_r  <= ST_INIT_WAIT;
        timer_r  <= 0;
        cmd_r    <= CMD_NOP;
        cke_r    <= '0';
        ready_r  <= '0';
        else
        state_r  <= state_x;
        timer_r  <= timer_x;
        refcnt_r <= refcnt_x;
        cke_r    <= cke_x;        -- CKE to SDRAM.
        cmd_r    <= cmd_x;        -- Command to SDRAM.
        bank_r   <= bank_x;       -- Bank to SDRAM.
        addr_r   <= addr_x;       -- Address to SDRAM.
        sd_dout_r   <= sd_dout_x;         -- Data to SDRAM.
        sd_busdir_r <= sd_busdir_x;   -- SDRAM bus direction.
        sd_dqmu_r   <= sd_dqmu_x;         -- Upper byte enable to SDRAM.
        sd_dqml_r   <= sd_dqml_x;         -- Lower byte enable to SDRAM.
        ready_r  <= ready_x;
        buf_dout_r  <= buf_dout_x;


        end if;
        end if;
    end process;


end architecture;
------ Fim do sdram_simple.vhd ------
------ vga_controller.vhd ------
--------------------------------------------------------------------------------
--
--   FileName:            vga_controller.vhd
--   Dependencies:        none
--   Design Software:  Quartus II 64-bit Version 12.1 Build 177 SJ Full Version
--
--   HDL CODE IS PROVIDED "AS IS."  DIGI-KEY EXPRESSLY DISCLAIMS ANY
--   WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT
--   LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
--   PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL DIGI-KEY
--   BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL
--   DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF
--   PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS
--   BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF),
--   ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.
```

```
--
--   Version History
--   Version 1.0 05/19/2013 Scott Larson
--       Initial Public Release
--
--------------------------------------------------------------------------------


LIBRARY ieee;
USE ieee.std_logic_1164.all;


ENTITY vga_controller IS
  GENERIC(
        h_pulse  :  INTEGER   := 96;   --horizontal sync pulse width in pixels
        h_bp     :  INTEGER   := 48;   --horizontal back porch width in pixels
        h_pixels :  INTEGER   := 640;  --horizontal display width in pixels
        h_fp     :  INTEGER   := 16;   --horizontal front porch width in pixels
        h_pol    :  STD_LOGIC := '0';   --horizontal sync pulse polarity (1 = positive, 0 = negative)
        v_pulse  :  INTEGER   := 2;       --vertical sync pulse width in rows
        v_bp     :  INTEGER   := 33;      --vertical back porch width in rows
        v_pixels :  INTEGER   := 480;  --vertical display width in rows
        v_fp     :  INTEGER   := 10;      --vertical front porch width in rows
        v_pol    :  STD_LOGIC := '0');  --vertical sync pulse polarity (1 = positive, 0 = negative)
  PORT(
        pixel_clk :  IN   STD_LOGIC;  --pixel clock at frequency of VGA mode being used
        reset_n   :  IN   STD_LOGIC;  --active low asycnchronous reset
        h_sync    :  OUT  STD_LOGIC;  --horizontal sync pulse
        v_sync    :  OUT  STD_LOGIC;  --vertical sync pulse
        disp_ena  :  OUT  STD_LOGIC;  --display enable ('1' = display time, '0' = blanking time)
        column    :  OUT  INTEGER range 0 to h_pixels-1:=0;  --horizontal pixel coordinate
        row       :  OUT  INTEGER range 0 to v_pixels-1 :=0; --vertical pixel coordinate
        n_blank   :  OUT  STD_LOGIC;  --direct blacking output to DAC
        n_sync    :  OUT  STD_LOGIC); --sync-on-green output to DAC
END vga_controller;


ARCHITECTURE behavior OF vga_controller IS
  CONSTANT  h_period  :  INTEGER := h_pulse + h_bp + h_pixels + h_fp;  --total number of pixel clocks in a
row
  CONSTANT  v_period  :  INTEGER := v_pulse + v_bp + v_pixels + v_fp;  --total number of rows in column
BEGIN


  n_blank <= '1';  --no direct blanking
  n_sync <= '0';   --no sync on green

  PROCESS(pixel_clk, reset_n)
        VARIABLE h_count  :  INTEGER RANGE 0 TO h_period - 1 := 0;  --horizontal counter (counts the
columns)
        VARIABLE v_count  :  INTEGER RANGE 0 TO v_period - 1 := 0;  --vertical counter (counts the rows)
  BEGIN

        IF(reset_n = '0') THEN  --reset asserted
        h_count := 0;             --reset horizontal counter
        v_count := 0;             --reset vertical counter
        h_sync <= NOT h_pol;  --deassert horizontal sync
        v_sync <= NOT v_pol;  --deassert vertical sync
        disp_ena <= '0';          --disable display
        column <= 0;              --reset column pixel coordinate
        row <= 0;                 --reset row pixel coordinate

        ELSIF(pixel_clk'EVENT AND pixel_clk = '1') THEN


        --counters
        IF(h_count < h_period - 1) THEN    --horizontal counter (pixels)
        h_count := h_count + 1;
        ELSE
        h_count := 0;
        IF(v_count < v_period - 1) THEN  --veritcal counter (rows)
        v_count := v_count + 1;
        ELSE
        v_count := 0;
        END IF;
        END IF;
```

```
        --horizontal sync signal
        IF(h_count < h_pixels + h_fp OR h_count > h_pixels + h_fp + h_pulse) THEN
        h_sync <= NOT h_pol;      --deassert horiztonal sync pulse
        ELSE
        h_sync <= h_pol;          --assert horiztonal sync pulse
        END IF;

        --vertical sync signal
        IF(v_count < v_pixels + v_fp OR v_count > v_pixels + v_fp + v_pulse) THEN
        v_sync <= NOT v_pol;      --deassert vertical sync pulse
        ELSE
        v_sync <= v_pol;          --assert vertical sync pulse
        END IF;

        --set pixel coordinates
        IF(h_count < h_pixels) THEN  --horiztonal display time
        column <= h_count;        --set horiztonal pixel coordinate
        END IF;
        IF(v_count < v_pixels) THEN  --vertical display time
        row <= v_count;           --set vertical pixel coordinate
        END IF;


        --set display enable output
        IF(h_count < h_pixels AND v_count < v_pixels) THEN  --display time
        disp_ena <= '1';                                    --enable display
        ELSE                                                --blanking time
        disp_ena <= '0';                                    --disable display
        END IF;


        END IF;
    END PROCESS;

END behavior;
```

——— Fim do vga_controller.vhd ———

APÊNDICE C

Decisão de arquiteturas, planos de implementação e critérios considerados estão presentes nesta seção. Estes dados não foram incluídos no corpo principal do texto devido a extensão e a não participação na implementação — no entanto, foram extremamente relevantes no curso do projeto e, por isso, encontram-se abaixo.

Abordagens de projeto

O plano A mostra-se o mais adequado segundo os requisitos e as restrições do projeto OpenGPU. A ideia de construir apenas uma etapa do *pipeline* gráfico já havia sido proposta anteriormente. Essa abordagem facilita o desenvolvimento do sistema, reduzindo a possibilidade de atrasos ou não finalização do plano.

O plano B foi considerado inicialmente como melhor opção, mas na primeira semana demonstrou não ser viável em um tempo tão curto dada a complexidade dos processadores de propósito geral e a arquitetura escolhida — outro ponto negativo é a dificuldade em visualizar a profundidade do trabalho a ser executado durante o desenvolvimento do *driver* Gallium, que deveria conter um compilador no plano B.

De modo geral, deve haver 3 meses disponíveis para execução do plano A. Entretanto, caso essa primeira concepção demonstre atrasos excessivos ou inviabilidade até no máximo 2 meses após início de sua execução, os outros planos serão considerados como substitutos prováveis a fim de concluir o projeto e entregar ao menos alguns dos requisitos propostos.

1. Plano A: Desenvolver módulo de rasterização
    - Rasterizador
    - Gerenciador de tarefas
    - Bus de interconexão
    - Gerenciador de memória
    - Controlador de vídeo

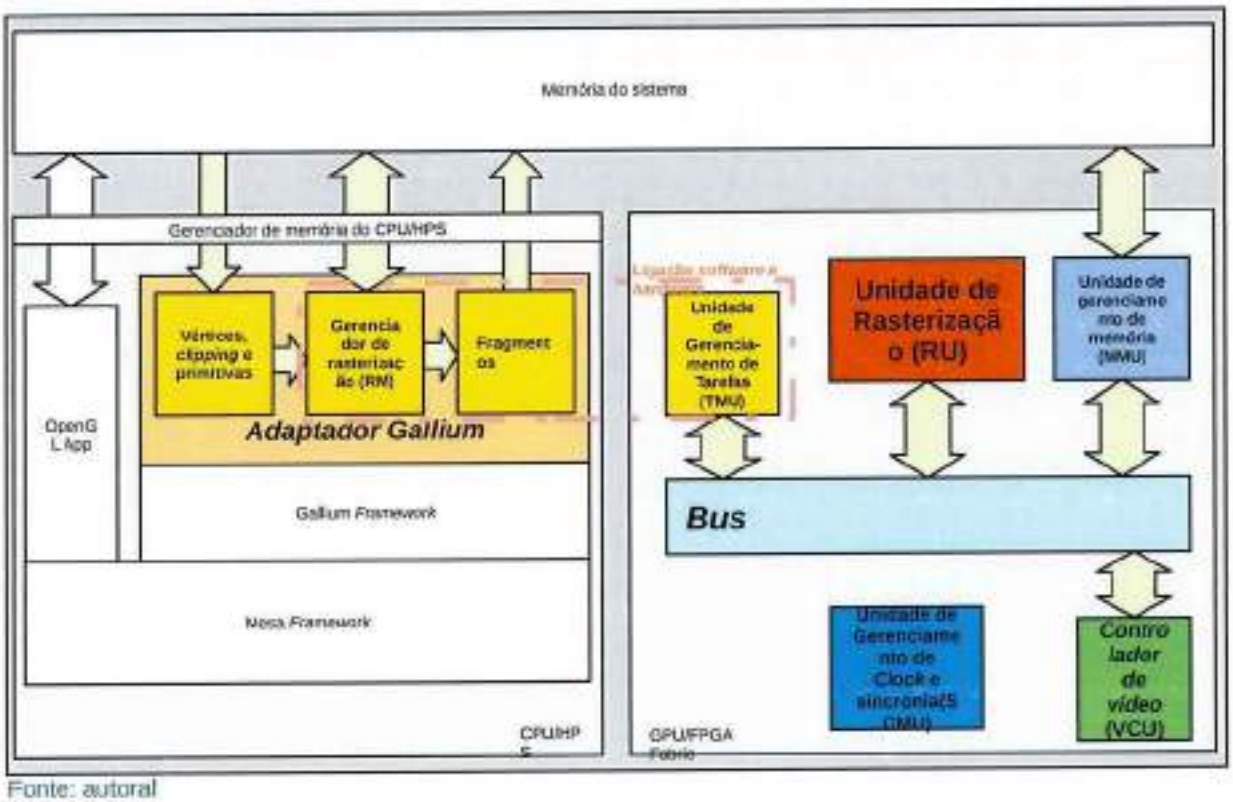
2. Plano B: Unir módulos de projetos diferentes
    - Núcleos
        - AltOr32
        - Amber 23/25
        - ARM4U
        - BERI
        - HiCoVec
        - Hive
        - LEON2/3/4
        - Leros
        - LXP32
        - MB-Lite
        - MicroCore

        - NEO430
        - Nyuzi
        - OpenMSP430
        - OpenPiton
        - OpenRISC
        - Potato Processor
        - RISCV Rocket-Chip Generator
        - Simply RISC S1
        - TotalCPU(TCPU)
        - UCore
        - Zet
        - Zip Cpu
    - Dispatcher
    - GPU ISA
        - Usar a mesma do núcleo
        - Usar a mesma do TGSI, convertendo para a do núcleo
            - Compilador
                - LLVM
    - Controlador de memória
    - Funções fixas
    - Bus de interconexão
    - Modo de transferência de comandos CPU->GPU
        - Interface própria
        - Memória
    - Memória do GPU
        - Compartilhada com CPU(host)
        - Exclusiva
3. Plano C: Adequar projeto Nyuzi para DE1 SoC
4. Plano D: Construir *pipeline* gráfico fixo sobre OpenGL 1.0
5. Plano E: Adequar processador a ISA de um GPU comercial

Descrição dos planos de projeto

1. Plano A: Desenvolver módulo de rasterização

Figura 25 - Sistema projetado



Fonte: autoral

Abaixo, a descrição de cada módulo do projeto:

- RASTERIZADOR(*RU*)

Representa a unidade de rasterização. Neste módulo, dados de polígonos são transformados em pixels. Os dados estarão sempre em algum *buffer* na memória do sistema de modo que sejam acessíveis pelos módulos de interesse.

Optou-se pelo desenvolvimento do *rasterizador* por ter sido demonstrado que ele representa a maior parte do tempo de processamento gráfico[9]. Outro motivo é sua independência dos *shaders* no *pipeline* gráfico, sendo uma função fixa.

Etapas internas:

- Estudo

Há vários métodos de rasterização que serão estudados nesta etapa. O critério de escolha será a facilidade de implementação — dado que outras necessidades, como performance, não são objetivo deste projeto.

- Implementação

Um *testbench* em *hardware* deverá ser desenvolvido para que verifique os vários submódulos necessários para o método de rasterização escolhido.

- Teste

Um *testbench* via simulação deverá ser desenvolvido para que verifique os vários submódulos necessários para o método de rasterização escolhido.

- GERENCIADOR DE CLOCK E SINCRONIA (SCMU, *Clock Synchrony Manager Unit*)

Este módulo deverá distribuir sinais de relógio entre todos os módulos.

  - Pesquisa de sistemas de distribuição de *clock*

A sincronia dos módulos deverá ser observada, assim como um método de configuração.

  - Implementação do *clock manager*

Os sinais de *clock* necessários por cada módulo serão produzidos individualmente e um método que garanta a sincronia será implementado.

  - Testes do *clock manager*

Uma simulação comportamental é suficiente, demonstrando a sincronia dos sinais de *clock*.

- GERENCIADOR DE MEMÓRIA (MMU, *Memory Manager Unit*)
  - Estudo de sistemas de gerenciamento de memória

Etapa fundamental para que sejam definidos os paradigmas de desenvolvimento da MMU. Ela será responsável pela interface entre o *bus* e a memória do sistema.

  - Implementação da MMU

A MMU deverá ter uma interface compatível com o *bus* e outra com a memória do sistema. Pelo fato desse projeto utilizar a DE-1 SoC, será necessário compatibilizar a MMU com o controlador de memória da DDR3.

  - Testes da MMU

A MMU deverá ser capaz de acessar a DDR3 disponível na DE-1 SoC. Uma verificação com padrões de dados deverá ser executada garantindo o funcionamento completo da MMU.

- VIA DE COMUNICAÇÃO (*Bus*)

O *bus* de comunicação deve ser explorado e definido nesta fase.

  - Estudo de viabilidade do Wishbone e alternativas

Wishbone é prioritário dada a disponibilidade de *cores* abertos com esta interface. Sua viabilidade será observada nesta etapa e definirá a necessidade ou não de outros tipos de *bus*. Deverá haver um arbitrador capaz de priorizar e organizar os acessos à memória.

  - Implementação inicial do *bus* escolhido

O *bus* mais adequado de acordo com o previsto anteriormente será implementado, visando a generalidade da conexão e possibilidade de ser modificado futuramente.

- Implementação do arbitrador

O arbitrador deverá seguir as especificações anteriores e ser configurável quanto a prioridade de acesso dos módulos conectados ao *bus*.

- Testes do bus e arbitrador

Um teste será elaborado e aplicado de acordo com o funcionamento elementar, capacidade de transferência — visando orientar modificações futuras a fim de evitar 'gargalos' de performance. Os testes do arbitrador e o *bus* deverão ser executados em conjunto.

- GERENCIADOR DE TAREFAS (TMU, *Task Manager Unit*)

O gerenciador de tarefas deve ser capaz de modificar as regiões de memória relativas a múltiplos *rasterizadores*, assim como verificar a disponibilidade de cada um, equalizando a carga de execução.

O gerenciador de tarefas precisará interpretar comandos advindos do CPU. Para isso, sua inicialização executará uma região pré-determinada na memória, na qual o CPU executará modificações, repassando comandos e ponteiros para dados. Esse sistema de interpretação será atualizado conforme a execução das fases seguintes. Este módulo deverá ser um processador especializado em interpretar comandos e distribuir tarefas, embora não processe os dados em si.

- Estudo de métodos de distribuição de carga

Métodos deverão ser pesquisados e um escolhido — critérios de seleção do método deverão ser coerentes com as necessidades do projeto. A interpretação de comandos deverá ser observada para que o TMU seja capaz de executá-los.

- Implementação inicial do gerenciador de tarefas

O método escolhido anteriormente deverá ser implementado nesta etapa.

- Testes do gerenciador de tarefas

Os métodos anteriores serão verificados nesta etapa. Um teste de comandos e distribução de carga para mútiplos rasterizadores deverá ser preparado e executado.

- ADAPTADOR GALLIUM (*Gallium Driver*)

O *driver* é a interface entre o *framework* Gallium e o *hardware*.

- Implementação do gerenciador de tarefas e rasterizador virtuais

O gerenciador de tarefas virtual será um modelo comportamental do TMU. Ele será executado no CPU (*host*) e servirá de referência à implementação do *driver*, pois funcionará com o rasterizador virtual a fim de simular o comportamento do sistema que será desenvolvido em *hardware*.

Esses modelos comportamentais deverão ser desenvolvidos em C ou C++.

- Implementação do *driver*

O adaptador deve ser capaz de implementar os mínimos recursos exigidos pelo Gallium e comandar os módulos do *pipeline* gráfico. Os módulos anteriores e posteriores à etapa de rasterização deverão ser copiados de outro *driver* Gallium — softpipe, llvmpipe ou outro que apresente melhor facilidade no desenvolvimento geral do projeto. Efetivamente, o módulo responsável por comandar o rasterizador no *hardware* será o principal foco no desenvolvimento.

- Testes do *driver*

O sistema deverá responder adequadamente ao gerenciador de tarefas virtual e exibir gráficos gerados por uma aplicação OpenGL.

- CONTROLADOR DE VÍDEO (*Video Controller Unit*)

Responsável por exibir o vídeo, deverá ser configurável para ler uma região específica da memória — *framebuffer*. Exibirá o conteúdo deste *buffer* na saída VGA — de acordo com as configurações de resolução e frequência de atualização.

- Implementação do *Video Controller*

Seguirá as especificações acima e, adicionalmente, deverá conter um *buffer* FIFO na leitura da memória — responsável por garantir o fluxo constante de pixels para a exibição de vídeo. Esse *buffer* é devido ao sistema estar conectado ao mesmo *bus* dos outros módulos — que a princípio consumirão tempo notável do sistema de memória.

Interiorizará um módulo controlador de VGA. O registro de configurações deverá estar na memória do sistema e será observado num intervalo de tempo de 200ms ou outro que seja pouco custoso ao *bus* e permita mudanças de configuração rápidas para humanos.

- Testes do *Video Controller*

Leitura de uma região especificada da memória contendo o registro de configuração, do *framebuffer* e do *buffer* FIFO na entrada. Resoluções e frequências diferentes devem ser obtidas dentro das limitações do controlador VGA.

CRONOGRAMA DO PLANO A

O tempo estimado para cada fase é definido nesta seção. Considera-se 1 dia de trabalho de 8 horas (ou dois integrantes, 4 horas) e 1 semana com 5 dias. Algumas fases serão realizadas simultaneamente. A expectativa de finalização do projeto é em 8 semanas.

Abaixo as etapas em ordem cronológica e o tempo estimado de conclusão:

- *Gallium driver*: 4 semanas e 4 dias
- RU: 2 semanas e 1 dia
- SCMU: 3 dia
- MMU: 4 dias
- *Bus*: 1 semana
- Teste 1
- TMU: 8 semanas
- VCU: 1 semana
- Teste 2
- *Benchmarking*

Link do cronograma (é necessário ter o app Gantter instalado no Google driver para acessá-lo): https://goo.gl/rkg9lw

TESTES DE MÓDULOS E DE SISTEMA E *BENCHMARKING*

Um *testbench* (Teste 1) será elaborado para verificar RU, *bus*, arbitrador, SCMU e MMU.

Haverá um teste final (Teste 2) após o desenvolvimento de todos os módulos, seguido de um *benchmarking*. Ambos serão desenvolvidos após a finalização dos módulos.

Os módulos só serão considerados finalizados caso passem em testes exclusivos a cada um. Esses testes serão elaborados conforme a compreensão da tecnologia de cara fase.

2. Plano B: Unir módulos de projetos diferentes
    - NÚCLEO: Primeira fase de escolha (28 processadores)

    - Estabilidade
        - Teste em FPGA comprovando funcionamento
        - Metas ou características principais cumpridas
    - Configurabilidade
        - Disponibilidade de configurações que permitam alterar a arquitetura interna do CPU
    - Operações com vetores
    - Operações com ponto flutuante 32 bits (*single precision*)
    - Gerenciamento de memória (MMU)

    - Virtualização no acesso a memória
- Cache
- *Multicore* (SMP, AMP)
- *Multithreading* (SMT)
- Largura
    - 32 bits ou mais
- Documentação
    - Qualidade
- NÚCLEO: Segunda fase de escolha (Nyuzi, OpenRISC e OpenSPARCT2)

- Maturidade do projeto
    - Nyuzi e OpenSPARC T2 tem a maior maturidade, visto que são projetos funcionais e implementaram todas ou a maior parte das especificações.
- Nível de otimizações (área e performance)
    - Otimizações de área ou de performance vêm do projeto com viés mais comercial que é o OpenSPARC T2. Os outros núcleos ainda objetivam terminar especificações ou são de viés acadêmico. O RISC-V BOOM tem demonstrado performance comparável a processadores comerciais.
- Processamento vetorial de ponto flutuante
    - Nyuzi desenvolveu módulos de processamento vetorial de ponto flutuante. OpenSPARC T2 aparentemente realiza a mesma tarefa, mas não fica claro numa primeira leitura se o faz paralelamente como o Nyuzi.
- Configurabilidade
    - Mor1kx é o projeto mais versátil — e este é um dos seus objetivos. Nyuzi e or1200 também apresentam um grau de configuração. OpenSPARC T2 não oferece configurações facilitadas — objetiva ser um projeto final e não algo a ser implementado em outros sistemas, embora não restrinja esta última opção. Diferentemente, os processadores do Rocket Chip Generator oferecem alto grau de modificação.
- Concepção de GPU/CPU multicore
    - Nyuzi e OpenSPARC T2 são CPU multicore, com o Nyuzi modificado para uma concepção GPU mais elaborada. Essas arquiteturas são próximas às utilizadas nos GPU comerciais — conhecidas como SIMT, que significa *Single Instruction Multiple Threads* (do inglês, Instrução Única, Múltiplos Contextos), assentada em mútiplos núcleos. Essa arquitetura permite que diversos contextos de processamento ocorram dentro de um processador, diminuindo seu tempo de ócio. Quando um *thread* acessa um recurso indisponível e precisa aguardar os dados, outro *thread* pode ser processado e, assim, otimizar o funcionamento do sistema. Isso, associado a diversos núcleos,

representa uma arquitetura que tem se mostrado eficiente em relação a processamento paralelo independente.

- Compatibilidade com outras ISAs
  - Mor1kx, or1200 e OpenSPARC T2 são ISAs bem estabelecidas — OpenRISC com o documento publicado da ISA e o OpenSPARC T2 com a SPARC V9. Rocket e BOOM implementam a RISC-V que é uma arquitetura recente, mas promissora — e já bem estabelecida.
- Concepção comercial
  - OpenSPARC T2 tem a maior característica comercial. É um produto finalizado, testado e depurado. Ele implementa a SPARC V9 completamente — a menos de operações *quad float* — e tem otimizações diversas quanto área e performance. Rocket e BOOM também caminham nesse sentido, embora projetos não maduros.
- Documentação
  - A documentação do OpenSPARCT2 é mais completa e detalhada que as outras, embora a do Nyuzi e do or1200 sejam muito claras também.
- Organização/documentação de código
  - O código de todos está bem comentado, apesar do mor1kx e do OpenSPARC T2 terem alguns trechos um pouco complexos sem explicações claras do funcionamento do código.

- NÚCLEO: Decisão baseada na tabela de comparação

Houve um empate entre OpenSPARC T2 e RISC-V BOOM. Por um lado, OpenSPARC T2 oferece processamento vetorial e grande estabilidade; por outro, BOOM oferece perspectiva de ser fortemente compatível com novos módulos de *hardware* no futuro próximo — se a RISC-V se tornar uma ISA popular, será possível manter o desenvolvimento do OpenGPU com auxílio de outros projetos.

Outro ponto negativo do OpenSPARC T2 é seu tamanho excessivo (cerca de 60K LUT na versão OpenSPARC T1, parecida com a T2) para ser executado em um FPGA SoC — a DE1 SoC tem 85K LUT. Isso possivelmente pode ser contornado reduzindo o número de *threads* e tamanho de cache — entretanto, não é claro no presente momento a facilidade de execução dessa tarefa e o impacto no funcionamento geral do sistema.

O projeto Rocket Chip Generator utiliza *Chipsel* como linguagem descritora de *hardware*. Embora gere códigos *Verilog* sintetizaveis nos *toolchain* para FPGA e ASIC, ela representa mais uma etapa de aprendizado — o que é um ponto negativo dada a quantidade conteúdo a ser compreendido (Gallium, *pipeline* gráfica, *dispatcher*, arquitetura do núcleo escolhido, *cores* complementares, ...). Adicionalmente, o processador de ponto flutuante do projeto Rocket não está maduro o bastante — o que poderá representar problemas no desenvolvimento.

O processador utilizado no OpenGPU será o OpenSPARC T2. O RISC-V BOOM será explorado apenas se o desenvolvimento com o OpenSPARC T2 for inviável.

- DISPATCHER E GPU ISA

O *dispatcher* depende da ISA escolhida. Ele é o módulo responsável por dividir tarefas e enviá-las aos núcleos. Este módulo precisa ser desenvolvido, já que não haverá um módulo pronto que atinja as mesmas especificações deste projeto — principalmente a compatibilidade com OpenGL, viabilizada pelo projeto Mesa.

- ISA baseada no *instruction set* do núcleo

A CPU escolhida foi a OpenSPARC T2 e, portanto, a ISA interna será a SPARC V9. Este núcleo implementa quase a totalidade da ISA SPARC V9, o que facilita o desenvolvimento do *dispatcher* quanto a variedade de instruções já preparadas. Outro ponto positivo é a tendência *RISC* da arquitetura SPARC, isso é, de simplificar as instruções ao máximo, ao invés de instruções mais complexas. Instruções simples permitem mais versatilidade ao realizar processamento sem, necessariamente, perder performance. O ponto negativo é a necessidade do compilador ou o *dispatcher* ser mais complexo para aproveitar essas instruções da melhor forma possível.

- ISA baseada no TGSI

TGSI é uma representação intermediária usada pelo Gallium, do projeto Mesa. Ela representa *shaders* e como devem ser processados. Essa linguagem é de baixo nível, sendo parecida com *assembly*. Os *drivers* do Gallium convertem TGSI em código de máquina do GPU a que são destinados.

Uma possibilidade seria utilizar o TGSI como *instruction set* parcial. As outras instruções relativas ao funcionamento do GPU seriam acrescentadas conforme a necessidade. Inicialização e configuração de estados do GPU são exemplos possíveis de instruções não presentes na TGSI.

Isso simplificaria o *driver*, mas complicaria o *dispatcher*, já que todo o trabalho de converter entre TGSI e a ISA SPARC V9 ficaria em *hardware*. Esse tipo de abordagem é a utilizada na maioria dos GPU comerciais — *shaders* são intepretados em tempo real dentro do GPU. O CPU fica, portanto, desobrigado em realizar esta tarefa que é bastante custosa em termos de processamento. Por isso, os GPU comerciais caminham neste sentido, a custo de aumento na complexidade do projeto — o que é um ponto negativo para o projeto OpenGPU, dada a limitação de tempo e recursos humanos.

- Decisão do ISA GPU

Os códigos em TGSI não são otimizados antes de chegar no *driver*[3] — o que implica menor performance se forem utilizados como estão. Daí a necessidade de otimização em *hardware* ou de aceitar a performance reduzida, deixando este tópico ser explorado em um momento futuro. Buscando uma performance mínima, existe uma outra abordagem intermediária em termos de velocidade de processamento — o código TGSI pode ser convertido em uma ISA construída especificamente para este GPU, utilizando um compilador modular.

LLVM é um compilador modular é uma ótima opção para desenvolver ou converter entre linguagens, sendo aplicado inclusive em programas e sistemas comerciais. A Sony tem usado o LLVM no SDK do PS4[4]. O *llvmpipe* é um *driver* Gallium e utiliza o LLVM para converter TGSI em código x86, aproveitando características intrínsecas dos CPU Intel para realizar diversas otimizações nos *shaders* — o que melhorou a performance gráfica dos GPU de baixo custo da Intel. Desenvolver uma linguagem de programação pode ser relativamente rápido[2] com o LLVM.

A opção deste projeto será, portanto, explorar as capacidades do LLVM gerando código para uma ISA desenvolvida especificamente para o OpenGPU. Três razões embasam esta decisão:

- Um sistema de compilação modular abstrai o sistema, criando uma linguagem intermediária largamente difundida, explorada e desenvolvida. Isso contribui para o objetivo do projeto OpenGPU de ser futuramente base de novos projetos;
- Reduz a carga de desenvolvimento, já que boa parte da tecnologia de compilação está presente dentro do LLVM e não precisa ser refeita;
- Aumenta a versatilidade do projeto e a possibilidade de incluir ou remover especificações de *hardware* mais facilmente.

- CONTROLADOR DE MEMÓRIA

Há várias opções de controlador de memória. Há dependência, no entanto, de como os processadores e o *dispatcher* estarão organizados. O *Cyclone V* presente na placa DE-1 SoC é um FPGA com um controlador DDR3. É importante notar que o MMU presente no OpenSPARC T2 utiliza memórias DDR2 — sendo necessário fazer a adaptação para DDR3. Isso pode significar reescrever totalmente o código do MMU para que o sistema funcione de acordo com o previsto.

- FUNÇÕES FIXAS

Há diversas funções fixas importantes de serem implementadas, enquanto outras são opcionais. Para este projeto, apenas o controlador de saída de vídeo será considerado fundamental — já que se espera obter imagem na saída VGA da placa. Os demais itens na lista(não exaustiva) a

seguir figuram em ordem de preferência, mas nenhum deverá ser implementado a menos que haja tempo extra:

- Controlador de Saída de Vídeo;
- Rasterizador — a princípio será implementado via *software* dentro dos núcleos, mas havendo tempo, poderá ser colocado em *hardware* como coprocessador ao lado dos núcleos de propósito geral;
- Compressor/Decompressor de dados;
- Codificador/Decodificador de vídeo — a escolher padrão de codificação.

- BUS DE INTERCONEXÃO

O uso do *Wishbone* é a primeira tentativa. Trata-se de um *bus* de código aberto utilizado em diversos sistemas e é mantido pelo OpenCores. Dada a facilidade de encontrar *IP cores* neste banco de dados, espera-se utilizá-lo de modo a diminuir o esforço de interconexão entre os sistemas. Apesar disso, sistemas de conexão diferentes poderão ser utilizados ou implementados a depender da necessidade e escassez de tempo.

O OpenSPARC T2 oferece sistemas de conexão já implementados como o CCX entre caches L2 e os núcleos. Esta e outras interfaces poderão ser reaproveitadas ou modificadas.

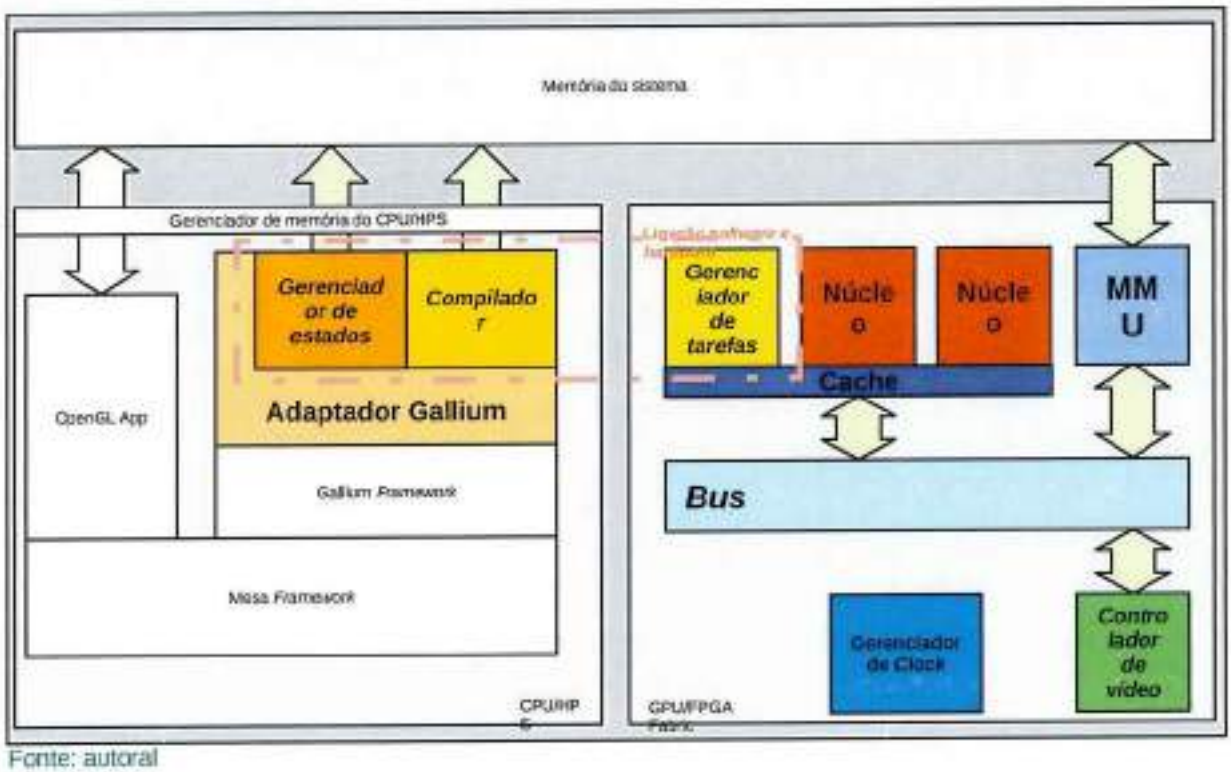
Os *buses* da Altera também são outra opção, sobretudo para facilitar o uso dos recursos dentro do *Cyclone V*.

- TRANSFERÊNCIA DE DADOS ENTRE CPU E GPU

A proposta mais simples e largamente utilizada, inclusive comercialmente, é passar instruções para o GPU via memória, tanto de *shaders* quanto de configurações de estados. Isso assume também o compartilhamento de memória RAM. Atualmente, os sistemas contam com memória RAM bastante abundante em relação às necessidades computacionais, o que diminui a expectativa de tornar exclusiva a memória disponível para o GPU. Esta condição pode ser revista num projeto futuro, já que bastaria acrescentar instruções de transferência de dados ou utilizar um DMA acionado pelo CPU de modo a intermediar as memórias de uso geral e de vídeo.

- ARQUITETURA ESCOLHIDA
-

Figura 26 - Sistema projetado



Fonte: autoral

Observa-se na figura, a arquitetura num nível mais detalhado. As cores representam os módulos que serão desenvolvidos. A ligação *software* e *hardware* indica a relação lógica entre *software (CPU)* e *hardware (GPU)*. Uma opção importante foi a alocação de 2 núcleos — isso se justifica pela necessidade de um GPU ser *multicore*, apesar da limitação de recursos no FPGA disponível. Desta forma, poderão ser testadas rotinas de multiplicação de *cores* que sejam funcionais para um número maior de núcleos — e internamente de *threads*, que deverão ser restritos também a 2 por núcleo pelo mesmo motivo.

Abaixo, a descrição de cada módulo do projeto:

- NÚCLEO (*Core*)

  Representa o núcleo escolhido anteriormente. Nesta fase, ele será modificado para obter a configuração mínima. Depois, será testado através de um *testbench* apropriado. Etapas internas:

  - Estudo das características no núcleo

    O núcleo OpenSPARC T2 será estudado com objetivo de melhorar a compreensão de suas estruturas e das suas interfaces a fim de ser conectado isoladamente ao *bus* de interesse. As possibilidades de configuração também serão exploradas nesta etapa.

  - Elaboração de teste

    Elaboração de teste do núcleo isolado. Um teste preliminar será executado para que se garanta o funcionamento inicial do núcleo isolado.

- Redução do núcleo

  O OpenSPARC T2 será simplificado para que contenha apenas 2 *threads* e o menor cache possível. Como tarefa secundária, serão feitas variáveis externas para facilitar a configuração posterior.

- Execução de teste em núcleo modificado

  Teste do núcleo com a redução anterior.

- Estudo do ISA SPARC V9

  Compreensão da ISA SPARC V9 e suas características. Esta etapa deve servir como prévia para implementação do *dispatcher* e do compilador baseado no LLVM.

- Configuração do núcleo

  Os *threads* deverão executar uma região previamente definida da memória. Para posterior comunicação com o *dispatcher*, cada um deverá registrar na memória o estado de execução — ocupado ou aguardando uma tarefa.

- GERENCIADOR DE CLOCK E SINCRONIA (SCMU, *Clock Synchrony Manager Unit*)

Este módulo deverá distribuir sinais de relógio entre todos os módulos.

- Pesquisa de sistemas de distribuição de *clock*

  A sincronia dos módulos deverá ser observada, assim como um método de configuração.

- Implementação do *clock manager*

  Os sinais de *clock* necessários por cada módulo serão produzidos individualmente e um método que garanta a sincronia será implementado.

- Testes do *clock manager*

  Uma simulação comportamental é suficiente, demonstrando a sincronia dos sinais de *clock*.

- CACHE (CMU, *Cache Manager Unit*)

Os núcleos compartilharão um segundo nível de *cache* (L2) daquele que cada um contém internamente (L1).

- Pesquisa de sistemas de *cache* L2

  O *cache* deve diminuir a espera de dados dos núcleos. Esta informação estará disponível somente com o sistema completo, por isso, será necessário usar um *cache* simples e pequeno, que apenas demonstre sua presença e permita sua configuração, sem se ater à performance.

- Implementação do L2

  O *cache* L2 deverá ser configurável e totalmente compatível com o núcleo e com o *bus*.

- Testes do L2

O *cache* precisará ser testado conforme especificações da etapa anterior.

- GERENCIADOR DE MEMÓRIA (MMU, *Memory Manager Unit*)
  - Estudo de sistemas de gerenciamento de memória

Etapa fundamental para que sejam definidos os paradigmas de desenvolvimento da MMU. Ela será responsável pela interface entre o *bus* e a memória do sistema.

  - Implementação da MMU

A MMU deverá ter uma interface compatível com o *bus* e outra com a memória do sistema. Pelo fato desse projeto utilizar a DE-1 SoC, será necessário compatibilizar a MMU com o controlador de memória da DDR3.

  - Testes da MMU

A MMU deverá ser capaz de acessar a DDR3 disponível na DE-1 SoC. Uma verificação de dados deverá ser executada garantindo o funcionamento completo da MMU.

- VIA DE COMUNICAÇÃO (*Bus*)

O *bus* de comunicação deve ser explorado e definido nesta fase.

  - Estudo de viabilidade do Wishbone e alternativas

Wishbone é prioritário dada a disponibilidade de *cores* abertos com esta interface. Sua viabilidade será observada nesta etapa e definirá a necessidade ou não de outros tipos de *bus* — inclusive aqueles presentes perifericamente ao núcleo OpenSPARC T2. Deverá haver um arbitrador capaz de priorizar e organizar os acessos à memória.

  - Implementação inicial do *bus* escolhido

O *bus* mais adequado de acordo com o previsto anteriormente será implementado, visando a generalidade da conexão e possibilidade de ser modificado futuramente.

  - Implementação do arbitrador

O arbitrador deverá seguir as especificações anteriores e ser configurável quanto a prioridade de acesso dos módulos conectados ao *bus*.

  - Testes do bus e arbitrador

Um teste será elaborado e aplicado de acordo com o funcionamento elementar, capacidade de transferência — visando orientar modificações futuras a fim de evitar 'gargalos' de performance. Os testes do arbitrador e o *bus* deverão ser executados em conjunto.

- GERENCIADOR DE TAREFAS (TMU, *Task Manager Unit*)

O gerenciador de tarefas deve ser capaz de modificar as regiões de memória relativas aos *threads* dos núcleos, assim como verificar a disponibilidade de cada um, equalizando a carga de execução.

O gerenciador de tarefas precisará interpretar comandos advindos do CPU. Para isso, sua inicialização executará uma região pré-determinada na memória, na qual o CPU executará modificações, repassando comandos e ponteiros para dados. Esse sistema de interpretação será atualizado conforme a execução das fases seguintes. Este módulo deverá ser um processador especializado em interpretar comandos e distribuir tarefas, embora não processe os dados em si.

- Estudo de métodos de distribuição de carga

  Métodos deverão ser pesquisados e um escolhido — critérios de seleção do método deverão ser coerentes com as necessidades do projeto. A interpretação de comandos deverá ser observada para que o *dispatcher* seja capaz de executá-los.
- Implementação inicial do gerenciador de tarefas

  O método escolhido anteriormente deverá ser implementado nesta etapa.
- Testes do gerenciador de tarefas

  Os métodos anteriores serão verificados nesta etapa. Um teste de comandos e distribução de carga para os núcleos deverá ser preparado e executado.

- GERENCIADOR DE ESTADOS (SMU, State Manager Unit)

Responsável por configurar o gerenciador de tarefas presente no GPU. Informações como região de memória de vídeo, resolução e controle das tarefas deverá ser preparado por este módulo. Com auxílio do módulo compilador, enviará processos/*shaders* para o GPU.

- Implementação do SMU(*State Manager Unit*)

Esta etapa implementará um código em *software* para controlar o GPU. Será importante estudar detalhes do Gallium *Framework* a fim de receber e enviar os comandos e dados corretos.

- Testes do SMU

  Esta etapa de testes deverá funcionar junto ao TMU.

- COMPILADOR (*Compiler*)

Esta fase representa o módulo de compilação de código TGSI e dos comandos de gerenciamento de estados. A saída deste compilador serão códigos de máquina com a ISA desenvolvida para o OpenGPU. Esta ISA será interpretada pelo TMU.

- Estudo do LLVM

  Fase de compreensão do sistema de compilação LLVM. Esta foi a base escolhida para o compilador.

- Implementação do compilador
  O compilador deve ser capaz de converter os códigos previstos.
- Testes do compilador
  Os testes deverão ser confirmados com auxílio do TMU.

- ADAPTADOR GALLIUM (*Gallium Driver*)
  O *driver* é a interface entre o *framework* Gallium e os módulos *Compiler* e SMU.
  - Implementação do *driver*
    O adaptador deve ser capaz de implementar os mínimos recursos exigidos pelo Gallium e comandar os módulos *Compiler* e SMU.

- CONTROLADOR DE VÍDEO (*Video Controller*)
  Responsável por exibir o vídeo, deverá ser configurável para ler uma região específica da memória — *framebuffer*. Exibirá o conteúdo deste *buffer* na saída VGA — de acordo com as configurações de resolução e frequência de atualização.
  - Implementação do *Video Controller*
    Seguirá as especificações acima e, adicionalmente, deverá conter um *buffer* FIFO na leitura da memória — responsável por garantir o fluxo constante de pixels para a exibição de vídeo. Esse *buffer* é devido ao sistema estar conectado ao mesmo *bus* dos outros módulos — que a princípio consumirão tempo notável do sistema de memória.
    Interiorizará um módulo controlador de VGA. O registro de configurações deverá estar na memória do sistema e será observado num intervalo de tempo de 200ms ou outro que seja pouco custoso ao *bus* e permita mudanças de configuração rápidas para humanos.
  - Testes do *Video Controller*
    Leitura de uma região especificada da memória contendo o registro de configuração, do *framebuffer* e do *buffer* FIFO na entrada. Resoluções e frequências diferentes devem ser obtidas dentro das limitações do controlador VGA.

CRONOGRAMA DO PLANO B

O tempo estimado para cada fase é definido nesta seção. Considera-se 1 dia de trabalho de 8 horas (ou dois integrantes, 4 horas) e 1 semana com 5 dias. Algumas fases serão realizadas simultaneamente.
Abaixo os módulos e o tempo estimado de conclusão:

- *Core*: 10 semana
- SCMU: 1 dia
- CMU: 2 semanas
- MMU: 1 semana
- *Bus*: 1 semana
- TMU: 8 semanas
- SMU: 8 semanas
- *Compiler*: 8 semanas
- *Driver*: 8 semanas
- *Video Controller*: 1 semana

TESTES DE MÓDULOS E DE SISTEMA E *BENCHMARKING*

Um *testbench* será elaborado para verificar os núcleos, o *cache*, o *bus*, o arbitrador e a MMU. Os núcleos deverão ser capazes de acessar a memória em suas regiões previamente configuradas e cada qual executar um programa simultaneamente.

Haverá um teste final após o desenvolvimento de todos os módulos, seguido de um *benchmarking*. Ambos serão desenvolvidos após a finalização dos módulos.

Os módulos só serão considerados finalizados caso passem em testes exclusivos a cada um. Esses testes serão elaborados conforme a compreensão da tecnologia de cara fase.

3. Adequar projeto Nyuzi para DE1 SoC

Outra abordagem é adaptar o projeto Nyuzi complementando-o com um *driver* Gallium adequado. Esse *driver* roda em cima do CPU ARM em cima do sistema Mesa. O *bus* de interconexão deve ser projetado para que o Nyuzi acesse a memória DDR3 da placa e possa receber instruções do CPU.

O Nyuzi utiliza o LLVM como sistema para gerar o compilador. Este, por sua vez, recebe códigos C/C++ e produz código de máquina específico para o processador Nyuzi. Isso mostra que os programas executados no Nyuzi não são gerados em tempo real, definindo a necessidade de embarcar esse compilador no *driver* Gallium produzido neste projeto para que seja possível executar uma *pipeline* gráfica em tempo real. O comparativo com o *driver* llvmpipe e seu funcionamento pode ser produtivo nessa etapa.

O projeto de adequação ao Nyuzi é definido nas seguintes etapas e no seguinte tempo:

1. Preparação e estudo do projeto Nyuzi (1 semana)
    a. Fazer processador Nyuzi rodar na placa DE-1 SoC;
    b. Compreensão do *hardware* do processador Nyuzi;
    c. Compreensão das ferramentas de *software*;
2. Desenvolvimento (3 semanas)
    a. *Hardware* que realize interface entre memória e CPU (*host*);
    b. *Driver* Gallium baseado em compilador LLVM.

Essas etapas devem acompanhadas de fases de testes e depuração previamente preparadas. A princípio, essa tarefa não deve ultrapassar um mês supondo o tempo descrito acima.

4. Construir *pipeline* gráfico fixo sobre OpenGL 1.0

OpenGL 1.0 foi o início das especificações de gráficos. Não havia *shaders* no início dos anos 90 (OSI, 2016). E isso se traduz num ponto positivo desta abordagem: *pipeline* fixo ao invés de programável. Esta perspectiva reduz consideravelmente o trabalho ao dispensar a tarefa de desenvolver processadores para os *shaders*. Os aspectos negativos são: o *pipeline* gráfico não pode ser implementado em trechos tão pequenos e modulares, o que exige o desenvolvimento de blocos extensos de *hardware* e experiência em processamento gráfico; pouca versatilidade do *hardware* de desenvolvido, o que reduz a possibilidade de expandir o projeto dada incapacidade *shaders* — cuja arquitetura necessária é bastante diferente de um *pipeline*.

Pode haver métodos de desenvolver *hardware* desse tipo dentro do prazo esperado pelo OpenGPU. Implementar apenas controladores de memória e o rasterizador em *hardware* pode ser uma destas tentativas. Caso as outras abordagens não sejam suficientes, esta poderá ser outra opção a ser explorada.

5. Adequar processador a ISA de um GPU comercial

Há o projeto MIAOW que implementou minimamente o ISA AMD Southern Islands. Baseado nisso, é possível desenvolver *hardware* apenas pelo ISA, ligando-o em *drivers* já estabelecidos tanto no sistema Mesa quanto nos *softwares* dos fabricantes.

O aspecto positivo está na redução do trabalho com *software* e todo o desenvolvimento conjunto com o *hardware*. O produto final também seria de boa qualidade, já que os *drivers* costumam ser bastante estáveis e completos quanto suas capacidades e especificações. Por outro lado, a complexidade de *hardware* seria bastante elevada, fazendo com o que o tempo de projeto seja alto. Talvez haja meios de implementar uma ISA parcialmente, reduzindo esse esforço e viabilizando o projeto no tempo disponível — por isso também é um plano considerado caso os as outras opções não sejam viáveis.

APÊNDICE D

Código de modificação do *driver* softpipe, parte do projeto Mesa e de seu subprojeto Gallium. O trecho modificado está em destaque abaixo.

Caminho do arquivo: mesa/src/gallium/drivers/softpipe/sp_flush.c

```
/**************************************************************************
 *
 * Copyright 2007 VMware, Inc.
 * All Rights Reserved.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 * copy of this software and associated documentation files (the
 * "Software"), to deal in the Software without restriction, including
 * without limitation the rights to use, copy, modify, merge, publish,
 * distribute, sub license, and/or sell copies of the Software, and to
 * permit persons to whom the Software is furnished to do so, subject to
 * the following conditions:
 *
 * The above copyright notice and this permission notice (including the
 * next paragraph) shall be included in all copies or substantial portions
 * of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
 * OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.
 * IN NO EVENT SHALL VMWARE AND/OR ITS SUPPLIERS BE LIABLE FOR
 * ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
 * TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 *
 **************************************************************************/

/* Author:
 *      Keith Whitwell <keithw@vmware.com>
 */


#include "pipe/p_defines.h"
#include "pipe/p_screen.h"
#include "draw/draw_context.h"
#include "sp_flush.h"
#include "sp_context.h"
#include "sp_state.h"
#include "sp_tile_cache.h"
#include "sp_tex_tile_cache.h"
#include "util/u_memory.h"
#include "util/u_string.h"


void
softpipe_flush( struct pipe_context *pipe,
                unsigned flags,
                struct pipe_fence_handle **fence )
```

```
{
   struct softpipe_context *softpipe = softpipe_context(pipe);
   uint i;

   draw_flush(softpipe->draw);

   if (flags & SP_FLUSH_TEXTURE_CACHE) {
         unsigned sh;

         for (sh = 0; sh < Elements(softpipe->tex_cache); sh++) {
         for (i = 0; i < softpipe->num_sampler_views[sh]; i++) {
         sp_flush_tex_tile_cache(softpipe->tex_cache[sh][i]);
         }
         }
   }

   /* If this is a swapbuffers, just flush color buffers.
         *
         * The zbuffer changes are not discarded, but held in the cache
         * in the hope that a later clear will wipe them out.
         */
   for (i = 0; i < softpipe->framebuffer.nr_cbufs; i++)
         if (softpipe->cbuf_cache[i])
         sp_flush_tile_cache(softpipe->cbuf_cache[i]);

   if (softpipe->zsbuf_cache)
         sp_flush_tile_cache(softpipe->zsbuf_cache);

   softpipe->dirty_render_cache = FALSE;

   /* Enable to dump BMPs of the color/depth buffers each frame */
#if 1
   //if (flags & PIPE_FLUSH_END_OF_FRAME) {
         static unsigned frame_no = 1;
         static char filename[256];
         util_snprintf(filename, sizeof(filename), "cbuf_%u.bmp", frame_no);
         debug_dump_surface_bmp(pipe, filename, softpipe->framebuffer.cbufs[0]);
         util_snprintf(filename, sizeof(filename), "zsbuf_%u.bmp", frame_no);
         debug_dump_surface_bmp(pipe, filename, softpipe->framebuffer.zsbuf);
         ++frame_no;
   //}
#endif

   if (fence)
         *fence = (void*)(intptr_t)1;
}

void
softpipe_flush_wrapped(struct pipe_context *pipe,
                        struct pipe_fence_handle **fence,
                        unsigned flags)
{
   softpipe_flush(pipe, SP_FLUSH_TEXTURE_CACHE, fence);
}

/**
 * Flush context if necessary.
 *
 * Returns FALSE if it would have block, but do_not_block was set, TRUE
```

```
 * otherwise.
 *
 * TODO: move this logic to an auxiliary library?
 */
boolean
softpipe_flush_resource(struct pipe_context *pipe,
                        struct pipe_resource *texture,
                        unsigned level,
                        int layer,
                        unsigned flush_flags,
                        boolean read_only,
                        boolean cpu_access,
                        boolean do_not_block)
{
   unsigned referenced;

   referenced = softpipe_is_resource_referenced(pipe, texture, level, layer);

   if ((referenced & SP_REFERENCED_FOR_WRITE) ||
       ((referenced & SP_REFERENCED_FOR_READ) && !read_only)) {

      /*
       * TODO: The semantics of these flush flags are too obtuse. They should
       * disappear and the pipe driver should just ensure that all visible
       * side-effects happen when they need to happen.
       */
      if (referenced & SP_REFERENCED_FOR_READ)
      flush_flags |= SP_FLUSH_TEXTURE_CACHE;

      if (cpu_access) {
      /*
      * Flush and wait.
      */

      struct pipe_fence_handle *fence = NULL;

      if (do_not_block)
      return FALSE;

      softpipe_flush(pipe, flush_flags, &fence);

      if (fence) {
      /*
               * This is for illustrative purposes only, as softpipe does not
               * have fences.
               */
      pipe->screen->fence_finish(pipe->screen, fence,
                                       PIPE_TIMEOUT_INFINITE);
      pipe->screen->fence_reference(pipe->screen, &fence, NULL);
      }
      } else {
      /*
      * Just flush.
      */
      softpipe_flush(pipe, flush_flags, NULL);
      }
   }
   return TRUE;
}
```

APÊNDICE E

Arquivos VHDL do OpenGPU e respectiva modificação no arquivo setup.c no Mesa3D para funcionamento do OpenGPU na placa DE1 SoC.

Modificações no setup.c:

```
//OGPU
#include "hps_0.h" //EPS FPGA DE1 SoC Board definitions for this project

#define soc_cv_av //TODO: remove this and do cross compiling in altera environment
#include "hwlib.h"//TODO: remove this and do cross compiling in altera environment
#include "socal/socal.h"//TODO: remove this and do cross compiling in altera environment
#include "socal/hps.h"//TODO: remove this and do cross compiling in altera environment
#include "socal/alt_gpio.h"//TODO: remove this and do cross compiling in altera environment


#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <stdlib.h>
#include <stddef.h>
#include <stdint.h>
#include <assert.h>
#include <string.h>
#include <math.h>
#include <time.h>

#define HW_REGS_BASE ( ALT_STM_OFST )
#define HW_REGS_SPAN ( 0x04000000 )
#define HW_REGS_MASK ( HW_REGS_SPAN - 1 )

#define ALT_AXI_FPGASLVS_OFST (0xC0000000) // axi_master
#define HW_FPGA_AXI_SPAN ( 0xFBFFFFFF-ALT_AXI_FPGASLVS_OFST+1 ) // Bridge span
#define HW_FPGA_AXI_MASK ( HW_FPGA_AXI_SPAN - 1 )
//OGPU end

//--OPENGPU
typedef uint8_t ogpu_bit;
typedef uint8_t ogpu_command;

enum OGPU_COMMAND
{
    OGPU_CMD_NOP=0,
    OGPU_CMD_PREPARE=0xA5,
    OGPU_CMD_RASTER=0xAA
};

struct ogpu_edge
{
    uint32_t x0,y0;
    uint32_t x1,y1;
};

struct ogpu_box
{
    float x0,y0;
    float x1,y1;
};

struct ogpu_quad
{
    //Fragment-vector element correspondence
    //  #---#---#
    //  | 0 | 1 |
    //  #---#---#
    //  | 2 | 3 |
    //  #---#---#
    uint16_t m[4][2];
};

struct ogpu_tile
{
    uint16_t x0,y0;
    uint16_t x1,y1;
};
```

```
struct ogpu_depth_coef
{
    int32_t a,b,c;
};

struct ogpu_depth_quad
{
    int32_t m[4];
};

struct ogpu_quad_buffer_cell
{
    uint16_t x,y; //quad (X,Y) top left coord
    uint8_t mask:4; //good mask on same quad order referenced above
    uint8_t stencil[4]; //RESERVED to stencil for future implementation
    float depth[4];
};

struct ogpu_quad_buffer
{
    uint16_t n; //number of elements
    struct ogpu_quad_buffer_cell *b; //buffer pointer
    struct ogpu_tile tile;
};

#define OGPU_DEPTH_DEPTH ((1<<30)-1) //depth buffer precision

static inline int32_t ogpu_fix_float(float f)
{
    return (int32_t)f;
}

static inline uint32_t ogpu_ufix_float(float f)
{
    return (uint32_t)f;
}

static void ogpu_setup(ogpu_bit clock,
                       //INPUTS
                       const ogpu_bit start_raster,
                       const float (*v0)[2],const float (*v1)[2],const float (*v2)[2], //v*: input vertices
                       //OUTPUTS
                       //struct ogpu_box *box, //bound box
                       struct ogpu_edge *e0,struct ogpu_edge *e1,struct ogpu_edge *e2, //e*: edges
                       ogpu_bit *setup_done)
{
    static ogpu_bit _clock=0;
    ////////////////////
    //     COMB     //
    ////////////////////
    //Edges data allocating and fixing floating points
    e0->x0=ogpu_ufix_float(v0[0][0]);    e0->y0=ogpu_ufix_float(v0[0][1]);
    e0->x1=ogpu_ufix_float(v1[0][0]);    e0->y1=ogpu_ufix_float(v1[0][1]);

    e1->x0=ogpu_ufix_float(v1[0][0]);    e1->y0=ogpu_ufix_float(v1[0][1]);
    e1->x1=ogpu_ufix_float(v2[0][0]);    e1->y1=ogpu_ufix_float(v2[0][1]);

    e2->x0=ogpu_ufix_float(v2[0][0]);    e2->y0=ogpu_ufix_float(v2[0][1]);
    e2->x1=ogpu_ufix_float(v0[0][0]);    e2->y1=ogpu_ufix_float(v0[0][1]);

    //Define triangle bound box //SOFTPIPE'S CLIPRECT DEFINES BOUND BOX BEFORE RASTERIZER STAGE
//    ogpu_bit c1,c2,c3;
//    c1=v0[0][0]<=v1[0][0];
//    c2=v0[0][0]<=v2[0][0];
//    c3=v1[0][0]<=v2[0][0];
//    if(c1)
//    {
//        if(c3)
//        {
//            box->x0=v0[0][0];
//            box->x1=v2[0][0];
//        }
//        else
//        {
//            if(c2)
//            {
//                box->x0=v0[0][0];
//                box->x1=v1[0][0];
//            }
//            else
//            {
//                box->x0=v2[0][0];
```

```
//                box->x1=v1[0][0];
//            }
//        }
//    }
//    else
//    {
//        if(c2)
//        {
//            box->x0=v1[0][0];
//            box->x1=v2[0][0];
//        }
//        else
//        {
//            if(c3)
//            {
//                box->x0=v1[0][0];
//                box->x1=v0[0][0];
//            }
//            else
//            {
//                box->x0=v2[0][0];
//                box->x1=v0[0][0];
//            }
//        }
//    }
//    c1=v0[0][1]<=v1[0][1];
//    c2=v0[0][1]<=v2[0][1];
//    c3=v1[0][1]<=v2[0][1];
//    if(c1)
//    {
//        if(c3)
//        {
//            box->y0=v0[0][1];
//            box->y1=v2[0][1];
//        }
//        else
//        {
//            if(c2)
//            {
//                box->y0=v0[0][1];
//                box->y1=v1[0][1];
//            }
//            else
//            {
//                box->y0=v2[0][1];
//                box->y1=v1[0][1];
//            }
//        }
//    }
//    else
//    {
//        if(c2)
//        {
//            box->y0=v1[0][1];
//            box->y1=v2[0][1];
//        }
//        else
//        {
//            if(c3)
//            {
//                box->y0=v1[0][1];
//                box->y1=v0[0][1];
//            }
//            else
//            {
//                box->y0=v2[0][1];
//                box->y1=v0[0][1];
//            }
//        }
//    }
    ////////////////////
    //     seq        //
    ////////////////////
    if(clock!=_clock)
    {
        _clock=clock;
        if(clock)       //CLOCK RISING EDGE
        {
            if(start_raster)
            {
                *setup_done=1;
            }
            else
```

```
            {
                *setup_done=0;
            }
        }
        else      //CLOCK FALLING EDGE
        {
        }
    }
}

static void ogpu_quad_generator(ogpu_bit clock,
                                //INPUTS
                                const ogpu_bit next_quad,
                                 const struct ogpu_box box, //(x0,y0) must be at left and at upper side of
(x1,y1)
                                const struct ogpu_tile tile, //(x0,y0) must be at left and at upper side of
(x1,y1)
                                //OUTPUTS
                                ogpu_bit *quad_ready,
                                ogpu_bit *end_tile,
                                struct ogpu_quad *quad)
{
    static ogpu_bit generate_quads=0;
    static ogpu_bit _clock=0;

    static uint16_t i=0,j=0,x0=0,y0=0,x1=0,y1=0;
    ////////////////////
    //     COMB     //
    ////////////////////


    ////////////////////
    //     SEQ      //
    ////////////////////
    if(clock!=_clock)
    {
        _clock=clock;
        if(clock)      //CLOCK RISING EDGE
        {
            if(next_quad)
            {
                if(!generate_quads)
                {
                    *end_tile=0;
                        if((tile.x1 < box.x0) || (tile.x0 > box.x1)) {*quad_ready=0;*end_tile=1;return;}
//checks if box is not
                        if((tile.y1 < box.y0) || (tile.y0 > box.y1)) {*quad_ready=0;*end_tile=1;return;}
//intersecting tile
                    generate_quads=1; //else generate quads
                    //clip tile: this helps discard void areas
                    i=y0=(tile.y0 <= box.y0)?(uint16_t)box.y0:tile.y0; //y0 is not used?
                    j=x0=(tile.x0 <= box.x0)?(uint16_t)box.x0:tile.x0;
                    x1=(tile.x1 >= box.x1)?(uint16_t)box.x1:tile.x1;
                    y1=(tile.y1 >= box.y1)?(uint16_t)box.y1:tile.y1;
                    //end clip tile
                    i&=~(1<<0);     //guarantee even number clearing first bit, this
                    x0=j&=~(1<<0); //is important to maintain quads at same place
                                    //independently of clipping
                }
                //generate quad coords
                quad->m[0][0]=j;      quad->m[1][0]=j+1;
                quad->m[0][1]=i;      quad->m[1][1]=i;
                quad->m[2][0]=i;      quad->m[3][0]=j+1;
                quad->m[2][1]=i+1;    quad->m[3][1]=i+1;
                j+=2;
                if(j>x1)
                {
                    j=x0;
                    i+=2;
                    if(i>y1)
                    {
                        *end_tile=1;
                        generate_quads=0;
                    }
                }
                *quad_ready=1;
            }
            else
            {
                *quad_ready=0;
            }
        }
        else      //CLOCK FALLING EDGE
```

```
        {
        }
    }
}

static inline ogpu_bit ogpu_edge_test(struct ogpu_edge e,uint32_t x,uint32_t y)
{
    //TODO: implement hardware oriented solution(fast fix point computing)
    if(
       ((int32_t)x-(int32_t)(e.x0))*((int32_t)(e.y1)-(int32_t)(e.y0)) -
       ((int32_t)(e.x1)-(int32_t)(e.x0))*((int32_t)y-(int32_t)(e.y0)))>=0 ) return 1;
    else return 0;
}

static void ogpu_quad_edge_test(ogpu_bit clock,
                                //INPUTS
                                struct ogpu_edge e,struct ogpu_quad quad,
                                ogpu_bit edge_test,
                                //OUTPUTS
                                ogpu_bit (*edge_mask)[4],
                                ogpu_bit *edge_ready)
{
    //static ogpu_bit _clock=0;
    ////////////////////
    //     COMB     //
    ////////////////////
    edge_mask[0][0]=ogpu_edge_test(e,quad.m[0][0],quad.m[0][1]);
    edge_mask[0][1]=ogpu_edge_test(e,quad.m[1][0],quad.m[1][1]);
    edge_mask[0][2]=ogpu_edge_test(e,quad.m[2][0],quad.m[2][1]);
    edge_mask[0][3]=ogpu_edge_test(e,quad.m[3][0],quad.m[3][1]);
    ////////////////////
    //     SEQ      //
    ////////////////////
    //if(clock!=_clock)
    {
        //_clock=clock;
        //if(clock)      //CLOCK RISING EDGE
        {
            *edge_ready=edge_test;
        }
        //else      //CLOCK FALLING EDGE
        {
        }
    }
}

static void ogpu_triangle_edge_test(ogpu_bit clock,
                                    //INPUTS
                                    ogpu_bit (*edge_ready)[3],ogpu_bit (*edge_mask0)[4],
                                    ogpu_bit (*edge_mask1)[4],ogpu_bit (*edge_mask2)[4],
                                    //OUTPUTS
                                    ogpu_bit (*quad_mask)[4],
                                    ogpu_bit *draw_quad,ogpu_bit *discard_quad)
{
    static ogpu_bit _clock=0;
    ogpu_bit _quad_mask[4];
    ogpu_bit _draw_quad;
    ////////////////////
    //     comb     //
    ////////////////////
    //if edge bits from one fragment are 1,1,1 or 0,0,0, draw fragment
    _quad_mask[0]=(edge_mask0[0][0]&&edge_mask1[0][0]&&edge_mask2[0][0])||
                  (!(edge_mask0[0][0]||edge_mask1[0][0]||edge_mask2[0][0]));
    _quad_mask[1]=(edge_mask0[0][1]&&edge_mask1[0][1]&&edge_mask2[0][1])||
                  (!(edge_mask0[0][1]||edge_mask1[0][1]||edge_mask2[0][1]));
    _quad_mask[2]=(edge_mask0[0][2]&&edge_mask1[0][2]&&edge_mask2[0][2])||
                  (!(edge_mask0[0][2]||edge_mask1[0][2]||edge_mask2[0][2]));
    _quad_mask[3]=(edge_mask0[0][3]&&edge_mask1[0][3]&&edge_mask2[0][3])||
                  (!(edge_mask0[0][3]||edge_mask1[0][3]||edge_mask2[0][3]));
    _draw_quad=_quad_mask[0]||_quad_mask[1]||_quad_mask[2]||_quad_mask[3];

    ////////////////////
    //     SEQ      //
    ////////////////////
    if(clock!=_clock)
    {
        _clock=clock;
        if(clock)      //CLOCK RISING EDGE
        {
            if(edge_ready[0][0]&&edge_ready[0][1]&&edge_ready[0][2])
            {
                if(_draw_quad)
                {
```

```
                    *draw_quad=1;
                    *discard_quad=0;
                    quad_mask[0][0]=_quad_mask[0];
                    quad_mask[0][1]=_quad_mask[1];
                    quad_mask[0][2]=_quad_mask[2];
                    quad_mask[0][3]=_quad_mask[3];
                }
                else
                {
                    *draw_quad=0;
                    *discard_quad=1;
                }
            }
            else
            {
                *draw_quad=0;
                *discard_quad=0;
            }
        }
        else      //CLOCK FALLING EDGE
        {
        }
    }
}

static void ogpu_depth_coef(const float (*v0)[4],const float (*v1)[4],const float (*v2)[4],
                            struct ogpu_depth_coef *coef)
{
    float A,B,C; //Cross product components
    float vx,vy,vz,wx,wy,wz;
    vx=(v1[0][0]-v0[0][0]); //V vector
    vy=(v1[0][1]-v0[0][1]);
    vz=(v1[0][2]-v0[0][2]);
    wx=(v2[0][0]-v0[0][0]); //W vector
    wy=(v2[0][1]-v0[0][1]);
    wz=(v2[0][2]-v0[0][2]);
    //This is a simple algorithm for calculate the plane equation from three points.
    //Calculating cross product VxW to obtain normal vector.
    //The normal vector and a point(V0) define a plane
    //Then we calculate the z function -- that returns z value for any x,y
    //Here, we calculate the coefficients a,b,c for function z=a*x+b*y+c
    //The final products and sums are performed inside hardware, while this
    //pre-calculations are performed in software, because they are constant for each
    //triangle. More details, consult OpenGPU project documentation.
    A=vy*wz-vz*wy;
    B=vz*wx-vx*wz;
    C=vx*wy-vy*wx;
    coef->a=ogpu_fix_float((-A/C)*(OGPU_DEPTH_DEPTH));
    coef->b=ogpu_fix_float((-B/C)*(OGPU_DEPTH_DEPTH));
    coef->c=ogpu_fix_float(v0[0][2]+A/C*v0[0][0]+B/C*v0[0][1])*(OGPU_DEPTH_DEPTH)); //c=Pz+A/C*Px+B/C*Py ,
where P=v0
}

static void ogpu_quad_depth_test(ogpu_bit clock,
                            //INPUTS
                            struct ogpu_quad quad,
                            ogpu_bit depth_test,
                            struct ogpu_depth_coef coef,
                            //OUTPUTS
                             ogpu_bit *depth_ready,
                             struct ogpu_depth_quad *depth_quad
                                 )
{
    static ogpu_bit _clock=0;
    static ogpu_bit _depth_test=0;
    ////////////////////
    //     COMB     //
    ////////////////////

    ////////////////////
    //     SEQ      //
    ////////////////////
    if(clock!=_clock)
    {
        _clock=clock;
        if(clock)      //CLOCK RISING EDGE
        {
            if(depth_test != _depth_test) // RISING EDGE
            {
                _depth_test = depth_test;
                if(depth_test)
                {
                    depth_quad->z[0]=coef.a*quad.x[0][0]+coef.b*quad.x[0][1]+coef.c;
```

```
                    depth_quad->m[1]=coef.a*quad.x[1][0]+coef.b*quad.x[1][1]+coef.c;
                    depth_quad->m[2]=coef.a*quad.x[2][0]+coef.b*quad.x[2][1]+coef.c;
                    depth_quad->m[3]=coef.a*quad.x[3][0]+coef.b*quad.x[3][1]+coef.c;
                    *depth_ready=1;
                }
                else
                {
                    *depth_ready=0;
                }
            }
        }
        else      //CLOCK FALLING EDGE
        {
        }
    }
}

static int ogpu_quad_buffer_alloc(struct ogpu_quad_buffer *quad_buffer,const struct ogpu_tile tile)
{
//    static uint32_t _size = 0;
//    uint32_t size=((tile.x1-tile.x0)/2+1)*((tile.y1-tile.y0)/2+1);
//    if(_size==0 || quad_buffer->b==0)
//    {
//                            quad_buffer->b    =    (struct    ogpu_quad_buffer_cell*)malloc(sizeof(struct
ogpu_quad_buffer_cell)*size);
//        if(quad_buffer->b == 0) return -1; //memory allocation failed
//    }
//    else
//    {
//        if(size>_size)
//        {
//            _size=size;
//                quad_buffer->b  =  (struct  ogpu_quad_buffer_cell*)realloc(quad_buffer->b,sizeof(struct
ogpu_quad_buffer_cell)*size);
//            if(quad_buffer->b == 0) return -2; //memory allocation failed
//        }
//
//    }
//    quad_buffer->n=0;
//    quad_buffer->tile=tile;
//    return 0;
    return -1;
}

static void ogpu_quad_buffer_free(struct ogpu_quad_buffer *quad_buffer)
{
    //if(quad_buffer->b) free(quad_buffer->b);
    //quad_buffer->b=0;
}

static inline float ogpu_float_fix(int32_t x)
{
    return ((float)x)/((long)1<<31)-2);
}

static void ogpu_quad_store(ogpu_bit clock,
                            //INPUTS
                            ogpu_bit (*quad_mask)[4],
                            struct ogpu_quad quad,
                            ogpu_bit start_raster,
                            ogpu_bit store_quad,
                            struct ogpu_tile tile,
                            struct ogpu_depth_quad depth_quad,
                            //OUTPUTS
                            ogpu_bit *quad_stored,
                            struct ogpu_quad_buffer *quad_buffer
                            )
{
    static ogpu_bit _clock=0;
    static ogpu_bit _start_raster=0;
    static ogpu_bit _store_quad=0;
    static uint16_t _quad_counter=0;
    ////////////////////
    //     COMB     //
    ////////////////////

    ////////////////////
    //     SEQ      //
    ////////////////////
    if(start_raster!=_start_raster)
    {
        _start_raster=start_raster;
        if(start_raster) // rising edge
```

```
        {
            _quad_counter=0;
        }
    }
    if(clock!=_clock)
    {
        _clock=clock;
        if(clock)       //CLOCK RISING EDGE
        {
            if(store_quad != _store_quad)
            {
                _store_quad = store_quad;
                if(store_quad) // RISING EDGE
                {
                    quad_buffer->b[_quad_counter].x=quad.m[0][0];
                    quad_buffer->b[_quad_counter].y=quad.m[0][1];
                    quad_buffer->b[_quad_counter].mask=
                        quad_mask[0][0]<<0|quad_mask[0][1]<<1|
                        quad_mask[0][2]<<2|quad_mask[0][3]<<3;
                    quad_buffer->b[_quad_counter].stencil[0]=0;
                    quad_buffer->b[_quad_counter].stencil[1]=0;
                    quad_buffer->b[_quad_counter].stencil[2]=0;
                    quad_buffer->b[_quad_counter].stencil[3]=0;
                    quad_buffer->b[_quad_counter].depth[0]=ogpu_float_fix(depth_quad.m[0]);
                    quad_buffer->b[_quad_counter].depth[1]=ogpu_float_fix(depth_quad.m[1]);
                    quad_buffer->b[_quad_counter].depth[2]=ogpu_float_fix(depth_quad.m[2]);
                    quad_buffer->b[_quad_counter].depth[3]=ogpu_float_fix(depth_quad.m[3]);
                    quad_buffer->tile=tile;

                    _quad_counter++;
                    quad_buffer->n=_quad_counter;
                    *quad_stored=1;
                }
                else
                {
                    *quad_stored=0;
                }
            }
        }
        else       //CLOCK FALLING EDGE
        {
        }
    }
}

enum OGPU_RASTER_CONTROL_STATE
{
    OGPU_RASTER_CONTROL_IDLE=0,
    OGPU_RASTER_CONTROL_DONE,
    OGPU_RASTER_CONTROL_SETUP,
    OGPU_RASTER_CONTROL_QUAD_GEN,
    OGPU_RASTER_CONTROL_QUAD_TEST,
    OGPU_RASTER_CONTROL_STORE_QUAD
};

static void ogpu_raster_control(ogpu_bit clock,
                                //INPUTS
                                ogpu_command cmd,
                                ogpu_bit setup_done,
                                ogpu_bit end_tile,
                                ogpu_bit quad_ready,
                                ogpu_bit depth_ready,
                                ogpu_bit quad_stored,
                                ogpu_bit draw_quad,
                                ogpu_bit discard_quad,
                                //OUTPUTS
                                ogpu_bit *start_raster,
                                ogpu_bit *next_quad,
                                ogpu_bit *edge_test,
                                ogpu_bit *depth_test,
                                ogpu_bit *store_quad,
                                ogpu_bit *busy,
                                ogpu_bit *done
                                )
{
    static unsigned _state=OGPU_RASTER_CONTROL_IDLE;
    static ogpu_bit _clock=0;
    if(clock!=_clock)
    {
        _clock=clock;
        if(clock)       //CLOCK RISING EDGE
        {
            // ////
```

```
// CAUTION: 'if' order matters, because it does part of logic in some state transitions
switch(_state)
{
default:
case OGPU_RASTER_CONTROL_IDLE:
    if(cmd==OGPU_CMD_RASTER)
    {
        _state=OGPU_RASTER_CONTROL_SETUP;
        *busy=1;
        *done=0;
        *start_raster=1;
        break;
    }
    break;

case OGPU_RASTER_CONTROL_DONE:
    if(cmd==OGPU_CMD_PREPARE)
    {
        _state=OGPU_RASTER_CONTROL_IDLE;
        *done=0;
        *busy=0;
        *start_raster=0;
        *next_quad=0;
        *edge_test=0;
        *depth_test=0;
        *store_quad=0;
        break;
    }
    break;

case OGPU_RASTER_CONTROL_SETUP:
    if(setup_done)
    {
        _state=OGPU_RASTER_CONTROL_QUAD_GEN;
        *next_quad=1;
        *store_quad=0;
        *edge_test=0;
        *depth_test=0;
        break;
    }
    break;

case OGPU_RASTER_CONTROL_QUAD_GEN:
    if(quad_ready)
    {
        _state=OGPU_RASTER_CONTROL_QUAD_TEST;
        *next_quad=0;
        *edge_test=1;
        *depth_test=1;
        break;
    }
    if(end_tile)
    {
        _state=OGPU_RASTER_CONTROL_DONE;
        *done=1;
        *busy=0;
        *start_raster=0;
        *next_quad=0;
        *edge_test=0;
        *depth_test=0;
        *store_quad=0;
        break;
    }
    break;

case OGPU_RASTER_CONTROL_QUAD_TEST:
    if(draw_quad && depth_ready)
    {
        _state=OGPU_RASTER_CONTROL_STORE_QUAD;
        *store_quad=1;
        break;
    }
    if(end_tile)
    {
        _state=OGPU_RASTER_CONTROL_DONE;
        *done=1;
        *busy=0;
        *start_raster=0;
        *next_quad=0;
        *edge_test=0;
        *depth_test=0;
        *store_quad=0;
        break;
```

```
                }
                if(discard_quad)
                {
                    _state=OGPU_RASTER_CONTROL_QUAD_GEN;
                    *next_quad=1;
                    *store_quad=0;
                    *edge_test=0;
                    *depth_test=0;
                    break;
                }
                break;

            case OGPU_RASTER_CONTROL_STORE_QUAD:
                if(end_tile)
                {
                    _state=OGPU_RASTER_CONTROL_DONE;
                    *done=1;
                    *busy=0;
                    *start_raster=0;
                    *next_quad=0;
                    *edge_test=0;
                    *depth_test=0;
                    *store_quad=0;
                    break;
                }
                if(quad_stored)
                {
                    _state=OGPU_RASTER_CONTROL_QUAD_GEN;
                    *next_quad=1;
                    *store_quad=0;
                    *edge_test=0;
                    *depth_test=0;
                    break;
                }
                break;
            }
        }
        else      //CLOCK FALLING EDGE
        {
        }
    }
}

//HARDWARE FUNCTIONS FOR FPGA DE1 SoC
static int ipow(int base, int exp)
{
    int result = 1;
    while (exp)
    {
        if (exp & 1)
            result *= base;
        exp >>= 1;
        base *= base;
    }

    return result;
}

//Seven Seg display: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F,-, .
uint8_t _seven_mask[]={
                0x3F, //0
                0x06, //1
                0x5B, //2
                0x4F, //3
                0x66, //4
                0x6D, //5
                0x7D, //6
                0x07, //7
                0x7F, //8
                0x6F, //9
                0x77, //A
                0x7c, //B
                0x39, //C
                0x5E, //D
                0x79, //E
                0x71, //F
                0x40, //-
                0x00};//

static uint8_t d2ss(int value,int digit) //returns seven segment 'digit' configuration according entered
'value'
{
        uint8_t res=0xFF;
```

```
        if(digit<0)
        {
                if(value<0) res=_seven_mask[16];
                else res=_seven_mask[17];
        }
        else if(digit<100)
        {
                uint8_t n;
                if(value>0) n=(value%ipow(10,digit+1))/ipow(10,digit);
                else n=17;

                res=_seven_mask[n];
        }
        res=~res;
        return res;
}

static uint8_t h2ss(int value,int digit) //returns seven segment 'digit' configuration according entered
hexadecimal 'value'
{
        uint8_t res=0xFF;
        if(digit<0)
        {
                if(value<0) res=_seven_mask[16];
                else res=_seven_mask[17];
        }
        else if(digit<100)
        {
                uint8_t n;
                if(value>0) n=(value%ipow(10,digit+1))/ipow(10,digit);
                else n=17;

                res=_seven_mask[n];
        }
        res=~res;
        return res;
}
//

/**
 * Do triangle rasterization using OPENGPU VIRTUAL RASTERIZER.
 */
void
ogpu_raster_tri(struct setup_context *setup,
            const float (*v0)[4],
            const float (*v1)[4],
            const float (*v2)[4])
{
        ////SOFTPIPE RASTERIZER SETUP FUNCTIONS
        //TODO: REPLACE THIS FUNCTIONS WITH CUSTOM ONES
        struct quad_stage *pipe = setup->softpipe->quad.first;

        float det;

        uint layer = 0;
        unsigned viewport_index = 0;

        if (setup->softpipe->no_rast || setup->softpipe->rasterizer->rasterizer_discard)
          return;

        det = calc_det(v0, v1, v2);

        if (!setup_sort_vertices( setup, det, v0, v1, v2 ))
          return;

        setup_tri_coefficients( setup );

        if (setup->softpipe->layer_slot > 0) {
          layer = *(unsigned *)setup->vprovoke[setup->softpipe->layer_slot];
          layer = MIN2(layer, setup->max_layer);
        }

        if (setup->softpipe->viewport_index_slot > 0) {
          unsigned *udata = (unsigned*)v0[setup->softpipe->viewport_index_slot];
          viewport_index = sp_clamp_viewport_idx(*udata);
        }
        ////
//TEST DE1
        void *virtual_base,*h2f_virtual_base;
        int fd;
        //int loop_count;
        static int led_direction=0;
        static int led_mask=0x1;
```

```
        void *h2p_lw_led_addr,*h2p_lw_sw_addr;
        void *seven_seg_addr[6];
        void *rl_req,*rl_data_high,*rl_data_low,*rl_ack;
        void *rl_reset;
        void *rl_command;
        void *rl_v0x,*rl_v0y,*rl_v0z;
        void *rl_v1x,*rl_v1y,*rl_v1z;
        void *rl_v2x,*rl_v2y,*rl_v2z;
        void *rl_clip_rect0,*rl_clip_rect1;
        void *rl_tile0,*rl_tile1,*rl_depth_coef_a,*rl_depth_coef_b,*rl_depth_coef_c;
        void *rl_quad_buffer_addr_high,*rl_quad_buffer_addr_low;
        void *rl_status;

        // map the address space for the LED registers into user space so we can interact with them.
        // we'll actually map in the entire CSR span of the HPS since we want to access various registers
within that span

        if( ( fd = open( "/dev/mem", ( O_RDWR | O_SYNC ) ) == -1 ) {
                printf( "ERROR: could not open \"/dev/mem\"...\n" );
                return;
        }

        virtual_base = mmap( NULL, HW_REGS_SPAN, ( PROT_READ | PROT_WRITE ), MAP_SHARED, fd,
HW_REGS_BASE );

        if( virtual_base == MAP_FAILED ) {
                printf( "ERROR: mmap() failed...\n" );
                close( fd );
                return;
        }

        h2p_lw_led_addr=( unsigned long *)(virtual_base + ( ( unsigned long )( ALT_LWFPGASLVS_OFST +
LED_PIO_BASE ) & ( unsigned long)( HW_REGS_MASK ) ));
        h2p_lw_sw_addr=( unsigned long *)(virtual_base + ( ( unsigned long )( ALT_LWFPGASLVS_OFST +
DIPSW_PIO_BASE ) & ( unsigned long)( HW_REGS_MASK ) );
        seven_seg_addr[0]=( unsigned long *)(virtual_base + ( ( unsigned long )( ALT_LWFPGASLVS_OFST +
SEVEN_SEG_0_BASE ) & ( unsigned long)( HW_REGS_MASK ) );
        seven_seg_addr[1]=( unsigned long *)(virtual_base + ( ( unsigned long )( ALT_LWFPGASLVS_OFST +
SEVEN_SEG_1_BASE ) & ( unsigned long)( HW_REGS_MASK ) );
        seven_seg_addr[2]=( unsigned long *)(virtual_base + ( ( unsigned long )( ALT_LWFPGASLVS_OFST +
SEVEN_SEG_2_BASE ) & ( unsigned long)( HW_REGS_MASK ) );
        seven_seg_addr[3]=( unsigned long *)(virtual_base + ( ( unsigned long )( ALT_LWFPGASLVS_OFST +
SEVEN_SEG_3_BASE ) & ( unsigned long)( HW_REGS_MASK ) );
        seven_seg_addr[4]=( unsigned long *)(virtual_base + ( ( unsigned long )( ALT_LWFPGASLVS_OFST +
SEVEN_SEG_4_BASE ) & ( unsigned long)( HW_REGS_MASK ) );
        seven_seg_addr[5]=( unsigned long *)(virtual_base + ( ( unsigned long )( ALT_LWFPGASLVS_OFST +
SEVEN_SEG_5_BASE ) & ( unsigned long)( HW_REGS_MASK ) );


        h2f_virtual_base = mmap( NULL, HW_FPGA_AXI_SPAN, ( PROT_READ | PROT_WRITE ), MAP_SHARED,
fd,ALT_AXI_FPGASLVS_OFST );
        if( h2f_virtual_base == MAP_FAILED ) {
                if( munmap( h2f_virtual_base, HW_FPGA_AXI_SPAN ) != 0 ) {
                                printf( "ERROR: munmap() failed...\n" );
                                close( fd );
                                return;
                }
                printf( "ERROR: mmap() failed for h2f mapping...\n" );
                close( fd );
                return;
        }

        rl_reset=( unsigned long *)(h2f_virtual_base + ( ( unsigned long )( OGPU_RESET_BASE ));
        rl_req=( unsigned long *)(h2f_virtual_base + ( ( unsigned long )( OGPU_QUAD_STORE_REQ_BASE ));
        rl_data_high=( unsigned long *)(h2f_virtual_base + ( ( unsigned long *)(
( OGPU_QUAD_STORE_DATA_HIGH_BASE )));
        rl_data_low=( unsigned long *)(h2f_virtual_base + ( ( unsigned long *)(
( OGPU_QUAD_STORE_DATA_LOW_BASE ));
        rl_ack=( unsigned long *)(h2f_virtual_base + ( ( unsigned long )( OGPU_QUAD_STORE_ACK_BASE ));

        rl_command=( unsigned long *)(h2f_virtual_base + ( ( unsigned long )( OGPU_RASTER_UNIT_COMMAND_BASE
));
        rl_v0x=( unsigned long *)(h2f_virtual_base + ( ( unsigned long )( OGPU_RASTER_UNIT_V0X_BASE ));
        rl_v0y=( unsigned long *)(h2f_virtual_base + ( ( unsigned long )( OGPU_RASTER_UNIT_V0Y_BASE ));
        rl_v0z=( unsigned long *)(h2f_virtual_base + ( ( unsigned long )( OGPU_RASTER_UNIT_V0Z_BASE ));
        rl_v1x=( unsigned long *)(h2f_virtual_base + ( ( unsigned long )( OGPU_RASTER_UNIT_V1X_BASE ));
        rl_v1y=( unsigned long *)(h2f_virtual_base + ( ( unsigned long )( OGPU_RASTER_UNIT_V1Y_BASE ));
        rl_v1z=( unsigned long *)(h2f_virtual_base + ( ( unsigned long )( OGPU_RASTER_UNIT_V1Z_BASE ));
        rl_v2x=( unsigned long *)(h2f_virtual_base + ( ( unsigned long )( OGPU_RASTER_UNIT_V1X_BASE ));
        rl_v2y=( unsigned long *)(h2f_virtual_base + ( ( unsigned long )( OGPU_RASTER_UNIT_V1Y_BASE ));
        rl_v2z=( unsigned long *)(h2f_virtual_base + ( ( unsigned long )( OGPU_RASTER_UNIT_V2Z_BASE ));
        rl_clip_rect0=( unsigned long *)(h2f_virtual_base + ( ( unsigned long *)(
( OGPU_RASTER_UNIT_CLIP_RECT0_BASE ));
```

```
        rl_clip_rectl=(   unsigned   long       )h2f_virtual_base   +   (   (   unsigned   long       )
( OGPU_RASTER_UNIT_CLIP_RECTl_BASE ));
        rl_tile0=( unsigned long  )h2f_virtual_base + ( ( unsigned long  )| OGPU_RASTER_UNIT_TILE0_BASE ));
        rl_tile1=( unsigned long  )h2f_virtual_base + ( ( unsigned long  )| OGPU_RASTER_UNIT_TILE1_BASE ));
        rl_depth_coef_a=(   unsigned   long       )h2f_virtual_base   +   (   (   unsigned   long       )
( OGPU_RASTER_UNIT_DEPTH_COEF_A_BASE ));
        rl_depth_coef_b=(   unsigned   long       )h2f_virtual_base   +   (   (   unsigned   long       )
( OGPU_RASTER_UNIT_DEPTH_COEF_B_BASE ));
        rl_depth_coef_c=(   unsigned   long       )h2f_virtual_base   +   (   (   unsigned   long       )
( OGPU_RASTER_UNIT_DEPTH_COEF_C_BASE ));
        rl_quad_buffer_addr_high=(   unsigned   long      )h2f_virtual_base   +   (   (   unsigned   long      )
( OGPU_RASTER_UNIT_QUAD_BUFFER_ADDR_HIGH_BASE ));
        rl_quad_buffer_addr_low=(   unsigned   long      )h2f_virtual_base   +   (   (   unsigned   long      )
( OGPU_RASTER_UNIT_QUAD_BUFFER_ADDR_LOW_BASE ));
        rl_status=(   unsigned   long       )h2f_virtual_base   +   (   (   unsigned   long       )
( OGPU_RASTER_UNIT_STATUS_BASE ));
```

```
    int sw = alt_read_hword(h2p_lw_sw_addr) & 0x1FF;
    static unsigned f_counter = 0;

    f_counter++;

    if(sw&4) // information ON
    {
        static clock_t t0=0;
        double seconds = ((double)(clock()-t0))/CLOCKS_PER_SEC;
        if(seconds>0.25)
        {
            unsigned fps=(unsigned)((f_counter/seconds)*1000);
            t0=clock();

            if(sw&2) //Total frame number after switching to this mode
            {
                alt_write_word(seven_seg_addr[0],d2ss(f_counter,0));
                alt_write_word(seven_seg_addr[1],d2ss(f_counter,1));
                alt_write_word(seven_seg_addr[2],d2ss(f_counter,2));
                alt_write_word(seven_seg_addr[3],d2ss(f_counter,3));
                alt_write_word(seven_seg_addr[4],d2ss(f_counter,4));
                alt_write_word(seven_seg_addr[5],d2ss(f_counter,5));
            }
            else //FPS
            {

                alt_write_word(seven_seg_addr[0],d2ss(fps,0));
                alt_write_word(seven_seg_addr[1],d2ss(fps,1));
                alt_write_word(seven_seg_addr[2],d2ss(fps,2));
                alt_write_word(seven_seg_addr[3],d2ss(-1,0));
                alt_write_word(seven_seg_addr[4],d2ss(fps,3));
                alt_write_word(seven_seg_addr[5],d2ss(fps,4));
                f_counter=0;
            }
        }

        // control led
        if(sw&1) *(uint32_t *)h2p_lw_led_addr = led_mask;
        else *(uint32_t *)h2p_lw_led_addr = ~led_mask;

        // update led mask
        if (led_direction == 0){
            led_mask <<= 1;
            if (led_mask == (0x01 << (LED_PIO_DATA_WIDTH-1)))
                led_direction = 1;
        }else{
            led_mask >>= 1;
            if (led_mask == 0x01){
                led_direction = 0;
            }
        }
    }

    if(sw&1) //if sw0 is one, do OGPU HARDWARE APPROACH
    {
        static struct ogpu_box box;
        static struct ogpu_tile tile;

        uint8_t command;
        uint16_t v0x=ogpu_ufix_float(v0[0][0]),
                 v0y=ogpu_ufix_float(v0[0][1]),
                 v0z=ogpu_ufix_float(v0[0][2]);
        uint16_t v1x=ogpu_ufix_float(v1[0][0]),
```

```
                    v1y=ogpu_ufix_float(v1[0][1]),
                    v1z=ogpu_ufix_float(v1[0][2]);
        uint16_t v2x=ogpu_ufix_float(v2[0][0]),
                    v2y=ogpu_ufix_float(v2[0][1]),
                    v2z=ogpu_ufix_float(v2[0][2]);
        uint32_t clip_rect0,clip_rect1,tile0,tile1;
        box.x0=setup->softpipe->cliprect[viewport_index].minx;
        box.y0=setup->softpipe->cliprect[viewport_index].miny;
        box.x1=setup->softpipe->cliprect[viewport_index].maxx;
        box.y1=setup->softpipe->cliprect[viewport_index].maxy;
        //printf("clip_rect: p0(%d,%d) p1(%d,%d)\n",x0,y0,x1,y1);
        clip_rect0=(ogpu_fix_float(box.x0)<<16)|(ogpu_fix_float(box.y0)&0xFFFF);//x0|y0
        clip_rect1=(ogpu_fix_float(box.x1)<<16)|(ogpu_fix_float(box.y1)&0xFFFF);//x1|y1

        tile.x0=setup->softpipe->cliprect[viewport_index].minx;
        tile.y0=setup->softpipe->cliprect[viewport_index].miny;
        tile.x1=tile.x0+62;
        tile.y1=tile.y0+62;
        tile0=(tile.x0<<16)|(tile.y0&0xFFFF);//x0|y0 tile of 64x64 pixels
        tile1=(tile.x1<<16)|(tile.y1&0xFFFF);//x1|y1
        //printf("tile: p0(%x) p1(%x)\n",tile0,tile1);

        //printf("status: %x\n",alt_read_word(r1_status));
        alt_write_word(r1_v0x,v0x); alt_write_word(r1_v0y,v0y); alt_write_word(r1_v0z,v0z);
        alt_write_word(r1_v1x,v1x); alt_write_word(r1_v1y,v1y); alt_write_word(r1_v1z,v1z);
        alt_write_word(r1_v2x,v2x); alt_write_word(r1_v2y,v2y); alt_write_word(r1_v2z,v2z);
        alt_write_word(r1_clip_rect0,clip_rect0); alt_write_word(r1_clip_rect1,clip_rect1);
        alt_write_word(r1_tile0,tile0); alt_write_word(r1_tile1,tile1);
        //ogpu_depth_coef(v0,v1,v2,&coef);
        alt_write_word(r1_depth_coef_a,0);
        alt_write_word(r1_depth_coef_b,0);
        alt_write_word(r1_depth_coef_c,0);
        alt_write_word(r1_quad_buffer_addr_high,0); alt_write_word(r1_quad_buffer_addr_low,0);

        alt_write_word(r1_reset,0); // reset gpu(active low)
        alt_write_word(r1_reset,1); // active gpu

        static struct ogpu_quad_buffer quad_buffer;
#define OGPU_HW_TILE_SIZE 64 //by now, it's limited to TILE_SIZE in sp_tile_cache.h
        struct ogpu_quad_buffer_cell __qb[OGPU_HW_TILE_SIZE/2*OGPU_HW_TILE_SIZE/2];
        memset((void*)__qb,0,sizeof(struct
ogpu_quad_buffer_cell)*OGPU_HW_TILE_SIZE/2*OGPU_HW_TILE_SIZE/2);

        quad_buffer.b=__qb;

        do//TILE LOOP
        {
                quad_buffer.n=0;
                quad_buffer.tile=tile;

                //if(ogpu_quad_buffer_alloc(&quad_buffer,tile)<0){    printf("Memory    allocation
failed\n"); return;}

                unsigned ibuf=0;
                uint32_t dataH,dataL,st,rt;
                command=0xAA;
                alt_write_word(r1_command,command);
                st=alt_read_word(r1_status);
                while((st&1)==0) //while DONE bit is zero
                {
                        at=alt_read_word(r1_status);
                        rt=alt_read_word(r1_req);
                        if(rt)
                        {
                                dataH=alt_read_word(r1_data_high);
                                dataL=alt_read_word(r1_data_low);
                                alt_write_word(r1_ack,1);
                                //usleep(10000);
                                while(alt_read_word(r1_req)); // while req signal is high
                                alt_write_word(r1_ack,0);
                                //mem_buf[ibuf++]=(uint64_t)((((uint64_t)dataH)<<32)|(dataL));
                                quad_buffer.b[ibuf].x=(uint16_t)(dataH>>16);
                                quad_buffer.b[ibuf].y=(uint16_t)(dataH&0xFFFF);
                                quad_buffer.b[ibuf].mask=(uint8_t)(dataL&0x0F);
                                ibuf++;
                        }
                        if(ibuf>=8192) break;
                }
                st=alt_read_word(r1_status);

                quad_buffer.n=ibuf;

                command=0xA5; //PREPARE FOR NEXT RASTER
```

```
                        alt_write_word(ri_command,command);
                        do
                        {
                                st=alt_read_word(ri_status);
                                //printf("status(prepare) (%d us): %x\n",i,s);
                                //usleep(i);
                                //i+=100;
                        }while((st&1)!=0); //while DONE bit is one

                //      printf("e0\tx0:%d y0:%d\tx1:%d y1:%d\n"
                //             "e1\tx0:%d y0:%d\tx1:%d y1:%d\n"
                //             "e2\tx0:%d y0:%d\tx1:%d y1:%d\n",e0.x0,e0.y0,e0.x1,e0.y1,
                //                                              e1.x0,e1.y0,e1.x1,e1.y1,
                //                                              e2.x0,e2.y0,e2.x1,e2.y1);
                //      printf("Bf(%.1f,%.1f)\n"
                //             "Bl(%.1f,%.1f)\n",box.x0,box.y0,box.x1,box.y1);

                        unsigned q,s,m,c;
                        #define OGPU_HW_QUAD_SIZE MAX_QUADS
                //      struct quad_header sp_quad[OGPU_SOFT_QUAD_SIZE];
                //      struct quad_header *sp_quad_ptrs[OGPU_SOFT_QUAD_SIZE];
                        m=quad_buffer.n;
                        c=0;
                        s=OGPU_HW_QUAD_SIZE;

                        do//MEMORY LOOP -- QUAD BUFFER READING AND SOFTPIPE NEXT STAGE INTERFACING
                        {
                                if(m<s) s=m;
                                for(q=0;q<s;q++,c++)
                                {
                                        setup->quad[q].input.x0=quad_buffer.b[c].x;
                                        setup->quad[q].input.y0=quad_buffer.b[c].y;

                                        setup->quad[q].input.layer=layer;
                                        setup->quad[q].input.viewport_index=viewport_index;
                                        setup->quad[q].input.coverage[0]=0;
                                        setup->quad[q].input.coverage[1]=0;
                                        setup->quad[q].input.coverage[2]=0;
                                        setup->quad[q].input.coverage[3]=0;
                                        setup->quad[q].input.facing=setup->facing;
                                        setup->quad[q].inout.mask=quad_buffer.b[c].mask;
                                        setup->quad[q].output.depth[0]=0;//quad_buffer.b[c].depth[0];
                                        setup->quad[q].output.depth[1]=0;//quad_buffer.b[c].depth[1];
                                        setup->quad[q].output.depth[2]=0;//quad_buffer.b[c].depth[2];
                                        setup->quad[q].output.depth[3]=0;//quad_buffer.b[c].depth[3];
        //              printf("D%d\t%f\t%f\n\t%f\t%f\n\n",c,quad_buffer.b[c].depth[0],
        //                     quad_buffer.b[c].depth[1],quad_buffer.b[c].depth[2],
        //                     quad_buffer.b[c].depth[3]);
                                        setup-
>quad[q].output.stencil[0]=0;//quad_buffer.b[c].stencil[0];
                                        setup-
>quad[q].output.stencil[1]=0;//quad_buffer.b[c].stencil[0];
                                        setup-
>quad[q].output.stencil[2]=0;//quad_buffer.b[c].stencil[0];
                                        setup-
>quad[q].output.stencil[3]=0;//quad_buffer.b[c].stencil[0];
                                        setup->quad[q].coef=setup->coef;
                                        setup->quad[q].posCoef=&setup->posCoef;

                                        setup->quad_ptrs[q]=&setup->quad[q];
                                }
                                if(s) pipe->run( pipe, setup->quad_ptrs, s );
                                m-=s;
                        }while(m);
                tile.x0+=64;
                tile.x1=tile.x0+62;
                if(tile.x0>=setup->softpipe->cliprect[viewport_index].maxx)
                {
                        tile.x0=0;
                        tile.x1=tile.x0+62;
                        tile.y0+=64;
                        tile.y1=tile.y0+62;
                }
                }while(tile.y0<setup->softpipe->cliprect[viewport_index].maxy);
        }
        else // if sw0 is zero, do software approach
        {
                if(sw&(1<<9)) // if one, ogpu software model
                {


                        //    static unsigned counter=0;
                        //    printf("Tri %d\nv0\tx:%.1f\ty:%.1f\tz:%.1f\tw:%.1f\n"
```

```
                        //          "v1\tx:%.1f\ty:%.1f\tz:%.1f\tw:%.1f\n"
                        //          "v2\tx:%.1f\ty:%.1f\tz:%.1f\tw:%.1f\n\n",counter++,
                        //                                                  v0[0][0],v0[0][1],v0[0]
[2],v0[0][3],
                        //                                                  v1[0][0],v1[0][1],v1[0]
[2],v1[0][3],
                        //                                                  v2[0][0],v2[0][1],v2[0]
[2],v2[0][3]);
                                                                    static              ogpu_bit
start_raster=0,next_quad=0,quad_ready=0,end_tile=0,setup_done=0;
                            static ogpu_bit edge_ready[]={0,0,0},edge_test=0;
                            static ogpu_bit edge_mask0[]={0,0,0,0};
                            static ogpu_bit edge_mask1[]={0,0,0,0};
                            static ogpu_bit edge_mask2[]={0,0,0,0};
                            static ogpu_bit clock=0;
                            static ogpu_bit draw_quad=0;
                            static ogpu_bit discard_quad=0;
                            static ogpu_bit quad_mask[]={0,0,0,0};
                            static ogpu_bit depth_ready=0;
                            static ogpu_bit depth_test=0;
                            static ogpu_bit store_quad=0;
                            static ogpu_bit quad_stored=0;
                            static ogpu_bit done=0;
                            static ogpu_bit busy=0;
                            static ogpu_command cmd=OGPU_CMD_RASTER;
                            static struct ogpu_edge e0,e1,e2;
                            static struct ogpu_box box;
                            static struct ogpu_quad quad;
                            static struct ogpu_tile tile;
                            static struct ogpu_depth_coef coef;
                            static struct ogpu_depth_quad depth_quad;
                            static struct ogpu_quad_buffer quad_buffer;
                        #define OGPU_TILE_SIZE 64 //by now, it's limited to TILE_SIZE in sp_tile_cache.h
                            struct ogpu_quad_buffer_cell __qb[OGPU_TILE_SIZE/2*OGPU_TILE_SIZE/2];

                            ogpu_depth_coef(v0,v1,v2,&coef);
                            tile.x0=setup->softpipe->cliprect[viewport_index].minx;
                            tile.y0=setup->softpipe->cliprect[viewport_index].miny;

                            box.x0=setup->softpipe->cliprect[viewport_index].minx;
                            box.y0=setup->softpipe->cliprect[viewport_index].miny;
                            box.x1=setup->softpipe->cliprect[viewport_index].maxx;
                            box.y1=setup->softpipe->cliprect[viewport_index].maxy;

                            tile.x1=tile.x0+63; tile.y1=tile.y0+63;

                            quad_buffer.b=__qb;
                            do//TILE LOOP
                            {
                                clock=0;
                                quad_buffer.n=0;
                                quad_buffer.tile=tile;

                                unsigned _next_raster=1;

                                    //if(ogpu_quad_buffer_alloc(&quad_buffer,tile)<0){  printf("Memory
allocation failed\n"); return;}
                                do//OGPU LOOP -- behavior algorithm implementation
                                {
                                    if(_next_raster)
                                    {
                                        if(!done)
                                        {
                                            cmd=OGPU_CMD_RASTER;
                                            _next_raster=0;
                                        }
                                        else
                                        {
                                            cmd=OGPU_CMD_PREPARE;
                                        }
                                    }

ogpu_raster_control(clock,cmd,setup_done,end_tile,quad_ready,depth_ready,quad_stored,
                                                                draw_quad,discard_quad,

&start_raster,&next_quad,&edge_test,&depth_test,&store_quad,
                                                                &busy,&done);

                                    ogpu_setup(clock,start_raster,(const float (*)[2])v0,(const float (*)
[2])v1,(const float (*)[2])v2,
                                                        &e0,&e1,&e2,&setup_done);
                                    ogpu_quad_generator(clock,next_quad,box,tile,
                                                        &quad_ready,&end_tile,&quad);
```

```
ogpu_quad_edge_test(clock,e0,quad.edge_test,&edge_mask0,&edge_ready[0]);

ogpu_quad_edge_test(clock,e1,quad,edge_test,&edge_mask1,&edge_ready[1]);

ogpu_quad_edge_test(clock,e2,quad,edge_test,&edge_mask2,&edge_ready[2]);

ogpu_triangle_edge_test(clock,&edge_ready,&edge_mask0,&edge_mask1,&edge_mask2,
                                                    &quad_mask,&draw_quad,&discard_quad);
                            ogpu_quad_depth_test(clock,quad,depth_test,coef,
                                                 &depth_ready,&depth_quad);


ogpu_quad_store(clock,&quad_mask,quad,start_raster,store_quad,tile,depth_quad,
                                          &quad_stored,&quad_buffer);
                            clock^=1;
                        }while(!done || _next_raster);

                    //    printf("e0\tx0:%d y0:%d\tx1:%d y1:%d\n"
                    //           "e1\tx0:%d y0:%d\tx1:%d y1:%d\n"
                    //           "e2\tx0:%d y0:%d\tx1:%d y1:%d\n",e0.x0,e0.y0,e0.x1,e0.y1,
                    //                                            e1.x0,e1.y0,e1.x1,e1.y1,
                    //                                            e2.x0,e2.y0,e2.x1,e2.y1);
                    //    printf("B0(%.1f,%.1f)\n"
                    //           "B1(%.1f,%.1f)\n",box.x0,box.y0,box.x1,box.y1);

                        unsigned q,s,n,c;
                        #define OGPU_SOFT_QUAD_SIZE MAX_QUADS
                    //    struct quad_header sp_quad[OGPU_SOFT_QUAD_SIZE];
                    //    struct quad_header *sp_quad_ptrs[OGPU_SOFT_QUAD_SIZE];
                        n=quad_buffer.n;
                        c=0;
                        s=OGPU_SOFT_QUAD_SIZE;

                            do//MEMORY LOOP -- QUAD BUFFER READING AND SOFTPIPE NEXT STAGE
INTERFACING
                        {
                            if(n<s) s=n;
                            for(q=0;q<s;q++,c++)
                            {
                                setup->quad[q].input.x0=quad_buffer.b[c].x;
                                setup->quad[q].input.y0=quad_buffer.b[c].y;

                                setup->quad[q].input.layer=layer;
                                setup->quad[q].input.viewport_index=viewport_index;
                                setup->quad[q].input.coverage[0]=0;
                                setup->quad[q].input.coverage[1]=0;
                                setup->quad[q].input.coverage[2]=0;
                                setup->quad[q].input.coverage[3]=0;
                                setup->quad[q].input.facing=setup->facing;
                                setup->quad[q].input.mask=quad_buffer.b[c].mask;
                                setup->quad[q].output.depth[0]=0;//quad_buffer.b[c].depth[0];
                                setup->quad[q].output.depth[1]=0;//quad_buffer.b[c].depth[1];
                                setup->quad[q].output.depth[2]=0;//quad_buffer.b[c].depth[2];
                                setup->quad[q].output.depth[3]=0;//quad_buffer.b[c].depth[3];
                    //            printf("Did\t%f\t%f\n\t%f\t%f\n\n",c,quad_buffer.b[c].depth[0],
                    //                   quad_buffer.b[c].depth[1],quad_buffer.b[c].depth[2],
                    //                   quad_buffer.b[c].depth[3]);
                                setup->quad[q].output.stencil[0]=0;//quad_buffer.b[c].stencil[0];
                                setup->quad[q].output.stencil[1]=0;//quad_buffer.b[c].stencil[0];
                                setup->quad[q].output.stencil[2]=0;//quad_buffer.b[c].stencil[0];
                                setup->quad[q].output.stencil[3]=0;//quad_buffer.b[c].stencil[0];
                                setup->quad[q].coef=setup->coef;
                                setup->quad[q].posCoef=&setup->posCoef;

                                setup->quad_ptrs[q]=&setup->quad[q];
                            }
                            if(s) pipe->run( pipe, setup->quad_ptrs, s );
                            n-=s;
                        }while(n);
                    tile.x0+=64;
                    tile.x1=tile.x0+62;
                    if(tile.x0>=setup->softpipe->cliprect[viewport_index].maxx)
                    {
                        tile.x0=0;
                        tile.x1=tile.x0+62;
                        tile.y0+=64;
                        tile.y1=tile.y0+62;
                    }
                    }while(tile.y0<setup->softpipe->cliprect[viewport_index].maxy);

                    //ogpu_quad_buffer_free(&quad_buffer);
```

```
                }
                else sp_setup_tri(setup,v0,v1,v2); // if sw9 is zero, use softpipe original fuaction
        }
        if( munmap( virtual_base, HW_REGS_SPAN ) != 0 ) {
                        if( munmap( h2f_virtual_base, HW_FPGA_AXI_SPAN ) != 0 ) {
                                        printf( "ERROR: h2f munmap() failed...\n" );
                                        close( fd );
                                        return;
                        }
                        printf( "ERROR: munmap() failed...\n" );
                        close( fd );
                        return;
                }
                else
                {
                        if( munmap( h2f_virtual_base, HW_FPGA_AXI_SPAN ) != 0 ) {
                                        printf( "ERROR: h2f munmap() failed...\n" );
                                        close( fd );
                                        return;
                        }
                }

                close( fd );
        //TEST DE1 END-----
}
//--OPENGPU
```

```
<?xml version="1.0" encoding="UTF-8"?>
<system name="$$(FILENAME)">
 <component
   name="$$(FILENAME)"
   displayName="$$(FILENAME)"
   version="1.0"
   description=""
   tags=""
   categories="System" />
 <parameter name="bonusData"><![CDATA[bonusData
{
   element alt_vip_itc_0
   {
      datum _sortIndex
      {
         value = "11";
         type = "int";
      }
      datum sopceditor_expanded
      {
         value = "1";
         type = "boolean";
      }
   }
   element alt_vip_vfr_vga
   {
      datum _sortIndex
      {
         value = "10";
         type = "int";
      }
      datum sopceditor_expanded
      {
         value = "1";
         type = "boolean";
      }
   }
   element alt_vip_vfr_vga.avalon_slave
   {
      datum _lockedAddress
      {
         value = "1";
         type = "boolean";
      }
      datum baseAddress
      {
         value = "256";
         type = "String";
      }
   }
   element button_pio
   {
      datum _sortIndex
      {
         value = "6";
         type = "int";
      }
      datum sopceditor_expanded
      {
         value = "1";
         type = "boolean";
      }
   }
   element button_pio.s1
   {
      datum _lockedAddress
      {
         value = "1";
         type = "boolean";
      }
      datum baseAddress
      {
         value = "65728";
         type = "String";
      }
   }
   element clk_0
   {
      datum _sortIndex
      {
         value = "0";
         type = "int";
      }
```

```
    datum sopceditor_expanded
    {
       value = "1";
       type = "boolean";
    }
}
element dbg_csig
{
    datum _sortIndex
    {
       value = "39";
       type = "int";
    }
}
element dbg_ram
{
    datum _sortIndex
    {
       value = "40";
       type = "int";
    }
}
element dipsw_pio
{
    datum _sortIndex
    {
       value = "5";
       type = "int";
    }
    datum sopceditor_expanded
    {
       value = "1";
       type = "boolean";
    }
}
element dipsw_pio.external_connection
{
    datum _tags
    {
       value = "";
       type = "String";
    }
}
element dipsw_pio.s1
{
    datum _lockedAddress
    {
       value = "1";
       type = "boolean";
    }
    datum baseAddress
    {
       value = "65664";
       type = "String";
    }
}
element hps_0
{
    datum _sortIndex
    {
       value = "1";
       type = "int";
    }
    datum sopceditor_expanded
    {
       value = "1";
       type = "boolean";
    }
}
element hps_0.f2h_axi_slave
{
    datum baseAddress
    {
       value = "0";
       type = "String";
    }
}
element intr_capturer_0
{
    datum _sortIndex
    {
       value = "9";
       type = "int";
```

```
    }
    datum sopceditor_expanded
    {
       value = "1";
       type = "boolean";
    }
}
element intr_capturer_0.avalon_slave_0
{
    datum _lockedAddress
    {
       value = "1";
       type = "boolean";
    }
    datum _tags
    {
       value = "";
       type = "String";
    }
    datum baseAddress
    {
       value = "196608";
       type = "String";
    }
}
element jtag_uart
{
    datum _sortIndex
    {
       value = "7";
       type = "int";
    }
    datum sopceditor_expanded
    {
       value = "1";
       type = "boolean";
    }
}
element jtag_uart.avalon_jtag_slave
{
    datum _lockedAddress
    {
       value = "1";
       type = "boolean";
    }
    datum baseAddress
    {
       value = "131072";
       type = "String";
    }
}
element led_pio
{
    datum _sortIndex
    {
       value = "4";
       type = "int";
    }
    datum sopceditor_expanded
    {
       value = "1";
       type = "boolean";
    }
}
element led_pio.s1
{
    datum _lockedAddress
    {
       value = "1";
       type = "boolean";
    }
    datum baseAddress
    {
       value = "65600";
       type = "String";
    }
}
element master_non_sec
{
    datum _sortIndex
    {
       value = "8";
       type = "int";
```

```
    }
    datum sopeditor_expanded
    {
       value = "1";
       type = "boolean";
    }
}
element master_secure
{
    datum _sortIndex
    {
       value = "2";
       type = "int";
    }
    datum sopeditor_expanded
    {
       value = "1";
       type = "boolean";
    }
}
element ogpu_quad_store_ack
{
    datum _sortIndex
    {
       value = "44";
       type = "int";
    }
}
element ogpu_quad_store_ack.s1
{
    datum baseAddress
    {
       value = "48";
       type = "String";
    }
}
element ogpu_quad_store_data_high
{
    datum _sortIndex
    {
       value = "42";
       type = "int";
    }
}
element ogpu_quad_store_data_high.s1
{
    datum baseAddress
    {
       value = "16";
       type = "String";
    }
}
element ogpu_quad_store_data_low
{
    datum _sortIndex
    {
       value = "43";
       type = "int";
    }
}
element ogpu_quad_store_data_low.s1
{
    datum baseAddress
    {
       value = "32";
       type = "String";
    }
}
element ogpu_quad_store_req
{
    datum _sortIndex
    {
       value = "41";
       type = "int";
    }
}
element ogpu_raster_unit_clip_rect0
{
    datum _sortIndex
    {
       value = "21";
       type = "int";
    }
```

```
}
element ogpu_raster_unit_clip_rect0.s1
{
   datum baseAddress
   {
      value = "65712";
      type = "String";
   }
}
element ogpu_raster_unit_clip_rect1
{
   datum _sortIndex
   {
      value = "22";
      type = "int";
   }
}
element ogpu_raster_unit_clip_rect1.s1
{
   datum baseAddress
   {
      value = "65728";
      type = "String";
   }
}
element ogpu_raster_unit_command
{
   datum _sortIndex
   {
      value = "20";
      type = "int";
   }
}
element ogpu_raster_unit_command.s1
{
   datum baseAddress
   {
      value = "65696";
      type = "String";
   }
}
element ogpu_raster_unit_depth_coef_a
{
   datum _sortIndex
   {
      value = "25";
      type = "int";
   }
}
element ogpu_raster_unit_depth_coef_a.s1
{
   datum baseAddress
   {
      value = "65776";
      type = "String";
   }
}
element ogpu_raster_unit_depth_coef_b
{
   datum _sortIndex
   {
      value = "26";
      type = "int";
   }
}
element ogpu_raster_unit_depth_coef_b.s1
{
   datum baseAddress
   {
      value = "65792";
      type = "String";
   }
}
element ogpu_raster_unit_depth_coef_c
{
   datum _sortIndex
   {
      value = "27";
      type = "int";
   }
}
element ogpu_raster_unit_depth_coef_c.s1
{
```

```
    datum baseAddress
    {
        value = "65808";
        type = "String";
    }
}
element ogpu_raster_unit_quad_buffer_addr_high
{
    datum _sortIndex
    {
        value = "29";
        type = "int";
    }
}
element ogpu_raster_unit_quad_buffer_addr_high.sl
{
    datum baseAddress
    {
        value = "65840";
        type = "String";
    }
}
element ogpu_raster_unit_quad_buffer_addr_low
{
    datum _sortIndex
    {
        value = "28";
        type = "int";
    }
}
element ogpu_raster_unit_quad_buffer_addr_low.sl
{
    datum baseAddress
    {
        value = "65824";
        type = "String";
    }
}
element ogpu_raster_unit_status
{
    datum _sortIndex
    {
        value = "19";
        type = "int";
    }
}
element ogpu_raster_unit_status.sl
{
    datum baseAddress
    {
        value = "65680";
        type = "String";
    }
}
element ogpu_raster_unit_tile0
{
    datum _sortIndex
    {
        value = "23";
        type = "int";
    }
}
element ogpu_raster_unit_tile0.sl
{
    datum baseAddress
    {
        value = "65744";
        type = "String";
    }
}
element ogpu_raster_unit_tile1
{
    datum _sortIndex
    {
        value = "24";
        type = "int";
    }
}
element ogpu_raster_unit_tile1.sl
{
    datum baseAddress
    {
        value = "65760";
```

```
        type = "String";
    }
}
element ogpu_raster_unit_v0x
{
    datum _sortIndex
    {
        value = "30";
        type = "int";
    }
}
element ogpu_raster_unit_v0x.sl
{
    datum baseAddress
    {
        value = "65536";
        type = "String";
    }
}
element ogpu_raster_unit_v0y
{
    datum _sortIndex
    {
        value = "31";
        type = "int";
    }
}
element ogpu_raster_unit_v0y.sl
{
    datum baseAddress
    {
        value = "65552";
        type = "String";
    }
}
element ogpu_raster_unit_v0z
{
    datum _sortIndex
    {
        value = "32";
        type = "int";
    }
}
element ogpu_raster_unit_v0z.sl
{
    datum baseAddress
    {
        value = "65568";
        type = "String";
    }
}
element ogpu_raster_unit_v1x
{
    datum _sortIndex
    {
        value = "33";
        type = "int";
    }
}
element ogpu_raster_unit_v1x.sl
{
    datum baseAddress
    {
        value = "65584";
        type = "String";
    }
}
element ogpu_raster_unit_v1y
{
    datum _sortIndex
    {
        value = "34";
        type = "int";
    }
}
element ogpu_raster_unit_v1y.sl
{
    datum baseAddress
    {
        value = "65600";
        type = "String";
    }
}
```

```
element ogpu_raster_unit_v1z
{
   datum _sortIndex
   {
      value = "35";
      type = "int";
   }
}
element ogpu_raster_unit_v1z.sl
{
   datum baseAddress
   {
      value = "65616";
      type = "String";
   }
}
element ogpu_raster_unit_v2x
{
   datum _sortIndex
   {
      value = "36";
      type = "int";
   }
}
element ogpu_raster_unit_v2x.sl
{
   datum baseAddress
   {
      value = "65632";
      type = "String";
   }
}
element ogpu_raster_unit_v2y
{
   datum _sortIndex
   {
      value = "37";
      type = "int";
   }
}
element ogpu_raster_unit_v2y.sl
{
   datum baseAddress
   {
      value = "65648";
      type = "String";
   }
}
element ogpu_raster_unit_v2z
{
   datum _sortIndex
   {
      value = "38";
      type = "int";
   }
}
element ogpu_raster_unit_v2z.sl
{
   datum baseAddress
   {
      value = "65664";
      type = "String";
   }
}
element ogpu_reset
{
   datum _sortIndex
   {
      value = "45";
      type = "int";
   }
}
element ogpu_reset.sl
{
   datum baseAddress
   {
      value = "64";
      type = "String";
   }
}
element pll_stream
{
   datum _sortIndex
```

```
    {
        value = "12";
        type = "int";
    }
    datum sopceditor_expanded
    {
        value = "1";
        type = "boolean";
    }
}
element seven_seg_0
{
    datum _sortIndex
    {
        value = "13";
        type = "int";
    }
}
element seven_seg_0.xl
{
    datum baseAddress
    {
        value = "80";
        type = "String";
    }
}
element seven_seg_1
{
    datum _sortIndex
    {
        value = "14";
        type = "int";
    }
}
element seven_seg_1.xl
{
    datum baseAddress
    {
        value = "64";
        type = "String";
    }
}
element seven_seg_2
{
    datum _sortIndex
    {
        value = "15";
        type = "int";
    }
}
element seven_seg_2.xl
{
    datum baseAddress
    {
        value = "48";
        type = "String";
    }
}
element seven_seg_3
{
    datum _sortIndex
    {
        value = "16";
        type = "int";
    }
}
element seven_seg_3.xl
{
    datum baseAddress
    {
        value = "32";
        type = "String";
    }
}
element seven_seg_4
{
    datum _sortIndex
    {
        value = "17";
        type = "int";
    }
}
element seven_seg_4.xl
```

```
    {
       datum baseAddress
       {
          value = "16";
          type = "String";
       }
    }
    element seven_seg_5
    {
       datum _sortIndex
       {
          value = "18";
          type = "int";
       }
    }
    element seven_seg_5.s1
    {
       datum baseAddress
       {
          value = "0";
          type = "String";
       }
    }
    element sysid_qsys
    {
       datum _sortIndex
       {
          value = "3";
          type = "int";
       }
       datum sopceditor_expanded
       {
          value = "1";
          type = "boolean";
       }
    }
    element sysid_qsys.control_slave
    {
       datum _lockedAddress
       {
          value = "1";
          type = "boolean";
       }
       datum baseAddress
       {
          value = "65536";
          type = "String";
       }
    }
}
]]></parameter>
 <parameter name="clockCrossingAdapter" value="HANDSHAKE" />
 <parameter name="device" value="5CSEMA5F31C6" />
 <parameter name="deviceFamily" value="Cyclone V" />
 <parameter name="deviceSpeedGrade" value="6" />
 <parameter name="fabricMode" value="QSYS" />
 <parameter name="generateLegacySim" value="false" />
 <parameter name="generationId" value="0" />
 <parameter name="globalResetBus" value="false" />
 <parameter name="hdlLanguage" value="VERILOG" />
 <parameter name="hideFromIPCatalog" value="false" />
 <parameter name="lockedInterfaceDefinition" value="" />
 <parameter name="maxAdditionalLatency" value="1" />
 <parameter name="projectName">DE1_SOC_Linux_FB.qpf</parameter>
 <parameter name="sopcBorderPoints" value="false" />
 <parameter name="systemHash" value="0" />
 <parameter name="testBenchDutName" value="" />
 <parameter name="timeStamp" value="0" />
 <parameter name="useTestBenchNamingPattern" value="false" />
 <instanceScript></instanceScript>
 <interface
   name="alt_vip_itc_0_clocked_video"
   internal="alt_vip_itc_0.clocked_video"
   type="conduit"
   dir="end" />
 <interface
   name="button_pio_external_connection"
   internal="button_pio.external_connection"
   type="conduit"
   dir="end" />
 <interface name="clk" internal="clk_0.clk_in" type="clock" dir="end" />
 <interface
   name="dbg_osic_probes"
```

```
    internal="dbg_csig.probes"
    type="conduit"
    dir="end" />
  <interface
    name="dbg_ram_probes"
    internal="dbg_ram.probes"
    type="conduit"
    dir="end" />
  <interface
    name="dipsw_pio_external_connection"
    internal="dipsw_pio.external_connection"
    type="conduit"
    dir="end" />
  <interface
    name="hps_0_h2f_reset"
    internal="hps_0.h2f_reset"
    type="reset"
    dir="start" />
  <interface name="hps_0_hps_io" internal="hps_0.hps_io" type="conduit" dir="end" />
  <interface
    name="led_pio_external_connection"
    internal="led_pio.external_connection"
    type="conduit"
    dir="end" />
  <interface name="memory" internal="hps_0.memory" type="conduit" dir="end" />
  <interface
    name="ogpu_quad_store_ack_external_connection"
    internal="ogpu_quad_store_ack.external_connection"
    type="conduit"
    dir="end" />
  <interface
    name="ogpu_quad_store_data_high_external_connection"
    internal="ogpu_quad_store_data_high.external_connection"
    type="conduit"
    dir="end" />
  <interface
    name="ogpu_quad_store_data_low_external_connection"
    internal="ogpu_quad_store_data_low.external_connection"
    type="conduit"
    dir="end" />
  <interface
    name="ogpu_quad_store_req_external_connection"
    internal="ogpu_quad_store_req.external_connection"
    type="conduit"
    dir="end" />
  <interface
    name="ogpu_raster_unit_clip_rect0_external_connection"
    internal="ogpu_raster_unit_clip_rect0.external_connection"
    type="conduit"
    dir="end" />
  <interface
    name="ogpu_raster_unit_clip_rect1_external_connection"
    internal="ogpu_raster_unit_clip_rect1.external_connection"
    type="conduit"
    dir="end" />
  <interface
    name="ogpu_raster_unit_command_external_connection"
    internal="ogpu_raster_unit_command.external_connection"
    type="conduit"
    dir="end" />
  <interface
    name="ogpu_raster_unit_depth_coef_a_external_connection"
    internal="ogpu_raster_unit_depth_coef_a.external_connection"
    type="conduit"
    dir="end" />
  <interface
    name="ogpu_raster_unit_depth_coef_b_external_connection"
    internal="ogpu_raster_unit_depth_coef_b.external_connection"
    type="conduit"
    dir="end" />
  <interface
    name="ogpu_raster_unit_depth_coef_c_external_connection"
    internal="ogpu_raster_unit_depth_coef_c.external_connection"
    type="conduit"
    dir="end" />
  <interface
    name="ogpu_raster_unit_quad_buffer_addr_high_external_connection"
    internal="ogpu_raster_unit_quad_buffer_addr_high.external_connection"
    type="conduit"
    dir="end" />
  <interface
    name="ogpu_raster_unit_quad_buffer_addr_low_external_connection"
    internal="ogpu_raster_unit_quad_buffer_addr_low.external_connection"
```

```
    type="conduit"
    dir="end" />
  <interface
    name="ogpu_raster_unit_status_external_connection"
    internal="ogpu_raster_unit_status.external_connection"
    type="conduit"
    dir="end" />
  <interface
    name="ogpu_raster_unit_tile0_external_connection"
    internal="ogpu_raster_unit_tile0.external_connection"
    type="conduit"
    dir="end" />
  <interface
    name="ogpu_raster_unit_tile1_external_connection"
    internal="ogpu_raster_unit_tile1.external_connection"
    type="conduit"
    dir="end" />
  <interface
    name="ogpu_raster_unit_v0x_external_connection"
    internal="ogpu_raster_unit_v0x.external_connection"
    type="conduit"
    dir="end" />
  <interface
    name="ogpu_raster_unit_v0y_external_connection"
    internal="ogpu_raster_unit_v0y.external_connection"
    type="conduit"
    dir="end" />
  <interface
    name="ogpu_raster_unit_v0z_external_connection"
    internal="ogpu_raster_unit_v0z.external_connection"
    type="conduit"
    dir="end" />
  <interface
    name="ogpu_raster_unit_v1x_external_connection"
    internal="ogpu_raster_unit_v1x.external_connection"
    type="conduit"
    dir="end" />
  <interface
    name="ogpu_raster_unit_v1y_external_connection"
    internal="ogpu_raster_unit_v1y.external_connection"
    type="conduit"
    dir="end" />
  <interface
    name="ogpu_raster_unit_v1z_external_connection"
    internal="ogpu_raster_unit_v1z.external_connection"
    type="conduit"
    dir="end" />
  <interface
    name="ogpu_raster_unit_v2x_external_connection"
    internal="ogpu_raster_unit_v2x.external_connection"
    type="conduit"
    dir="end" />
  <interface
    name="ogpu_raster_unit_v2y_external_connection"
    internal="ogpu_raster_unit_v2y.external_connection"
    type="conduit"
    dir="end" />
  <interface
    name="ogpu_raster_unit_v2z_external_connection"
    internal="ogpu_raster_unit_v2z.external_connection"
    type="conduit"
    dir="end" />
  <interface
    name="ogpu_reset_external_connection"
    internal="ogpu_reset.external_connection"
    type="conduit"
    dir="end" />
  <interface name="reset" internal="clk_0.clk_in_reset" type="reset" dir="end" />
  <interface
    name="seven_seg_0_external_connection"
    internal="seven_seg_0.external_connection"
    type="conduit"
    dir="end" />
  <interface
    name="seven_seg_1_external_connection"
    internal="seven_seg_1.external_connection"
    type="conduit"
    dir="end" />
  <interface
    name="seven_seg_2_external_connection"
    internal="seven_seg_2.external_connection"
    type="conduit"
    dir="end" />
```

```
 <interface
   name="seven_seg_3_external_connection"
   internal="seven_seg_3.external_connection"
   type="conduit"
   dir="end" />
 <interface
   name="seven_seg_4_external_connection"
   internal="seven_seg_4.external_connection"
   type="conduit"
   dir="end" />
 <interface
   name="seven_seg_5_external_connection"
   internal="seven_seg_5.external_connection"
   type="conduit"
   dir="end" />
 <module name="alt_vip_itc_0" kind="alt_vip_itc" version="14.0" enabled="1">
  <parameter name="ACCEPT_COLOUR8_IN_SEQ" value="0" />
  <parameter name="ANC_LINE" value="0" />
  <parameter name="AP_LINE" value="0" />
  <parameter name="BPS" value="8" />
  <parameter name="CLOCKS_ARE_SAME" value="0" />
  <parameter name="COLOUR_PLANES_ARE_IN_PARALLEL" value="1" />
  <parameter name="FAMILY" value="Cyclone V" />
  <parameter name="FIELD0_ANC_LINE" value="0" />
  <parameter name="FIELD0_V_BACK_PORCH" value="0" />
  <parameter name="FIELD0_V_BLANK" value="0" />
  <parameter name="FIELD0_V_FRONT_PORCH" value="0" />
  <parameter name="FIELD0_V_RISING_EDGE" value="0" />
  <parameter name="FIELD0_V_SYNC_LENGTH" value="0" />
  <parameter name="FIFO_DEPTH" value="1920" />
  <parameter name="F_FALLING_EDGE" value="0" />
  <parameter name="F_RISING_EDGE" value="1" />
  <parameter name="GENERATE_SYNC" value="0" />
  <parameter name="H_ACTIVE_PIXELS" value="1024" />
  <parameter name="H_BACK_PORCH" value="160" />
  <parameter name="H_BLANK" value="0" />
  <parameter name="H_FRONT_PORCH" value="24" />
  <parameter name="H_SYNC_LENGTH" value="136" />
  <parameter name="INTERLACED" value="0" />
  <parameter name="NO_OF_MODES" value="1" />
  <parameter name="NUMBER_OF_COLOUR_PLANES" value="4" />
  <parameter name="STD_WIDTH" value="1" />
  <parameter name="THRESHOLD" value="1919" />
  <parameter name="USE_CONTROL" value="0" />
  <parameter name="USE_EMBEDDED_SYNCS" value="0" />
  <parameter name="V_ACTIVE_LINES" value="768" />
  <parameter name="V_BACK_PORCH" value="29" />
  <parameter name="V_BLANK" value="0" />
  <parameter name="V_FRONT_PORCH" value="3" />
  <parameter name="V_SYNC_LENGTH" value="6" />
 </module>
 <module name="alt_vip_vfr_vga" kind="alt_vip_vfr" version="14.0" enabled="1">
  <parameter name="AUTO_CLOCK_MASTER_CLOCK_RATE" value="50000000" />
  <parameter name="AUTO_CLOCK_RESET_CLOCK_RATE" value="130000000" />
  <parameter name="BITS_PER_PIXEL_PER_COLOR_PLANE" value="8" />
  <parameter name="CLOCKS_ARE_SEPARATE" value="1" />
  <parameter name="FAMILY" value="Cyclone V" />
  <parameter name="MAX_IMAGE_HEIGHT" value="768" />
  <parameter name="MAX_IMAGE_WIDTH" value="1024" />
  <parameter name="MEM_PORT_WIDTH" value="128" />
  <parameter name="NUMBER_OF_CHANNELS_IN_PARALLEL" value="4" />
  <parameter name="NUMBER_OF_CHANNELS_IN_SEQUENCE" value="1" />
  <parameter name="SMASTER_BURST_TARGET" value="32" />
  <parameter name="SMASTER_FIFO_DEPTH" value="64" />
 </module>
 <module name="button_pio" kind="altera_avalon_pio" version="16.0" enabled="1">
  <parameter name="bitClearingEdgeCapReg" value="true" />
  <parameter name="bitModifyingOutReg" value="false" />
  <parameter name="captureEdge" value="true" />
  <parameter name="clockRate" value="50000000" />
  <parameter name="direction" value="Input" />
  <parameter name="edgeType" value="FALLING" />
  <parameter name="generateIRQ" value="true" />
  <parameter name="irqType" value="EDGE" />
  <parameter name="resetValue" value="0" />
  <parameter name="simDoTestBenchWiring" value="false" />
  <parameter name="simDrivenValue" value="0" />
  <parameter name="width" value="2" />
 </module>
 <module name="clk_0" kind="clock_source" version="16.0" enabled="1">
  <parameter name="clockFrequency" value="50000000" />
  <parameter name="clockFrequencyKnown" value="true" />
  <parameter name="inputClockFrequency" value="0" />
```

```
  <parameter name="resetSynchronousEdges" value="NONE" />
 </module>
 <module
  name="dbg_csig"
  kind="altera_in_system_sources_probes"
  version="16.0"
  enabled="1">
  <parameter name="create_source_clock" value="false" />
  <parameter name="create_source_clock_enable" value="false" />
  <parameter name="device_family" value="Cyclone V" />
  <parameter name="gui_use_auto_index" value="true" />
  <parameter name="instance_id" value="CSIG" />
  <parameter name="probe_width" value="12" />
  <parameter name="sld_instance_index" value="0" />
  <parameter name="source_initial_value" value="0" />
  <parameter name="source_width" value="0" />
 </module>
 <module
  name="dbg_ram"
  kind="altera_in_system_sources_probes"
  version="16.0"
  enabled="1">
  <parameter name="create_source_clock" value="false" />
  <parameter name="create_source_clock_enable" value="false" />
  <parameter name="device_family" value="Cyclone V" />
  <parameter name="gui_use_auto_index" value="true" />
  <parameter name="instance_id" value="RAM0" />
  <parameter name="probe_width" value="91" />
  <parameter name="sld_instance_index" value="0" />
  <parameter name="source_initial_value" value="0" />
  <parameter name="source_width" value="0" />
 </module>
 <module name="dlpsw_pio" kind="altera_avalon_pio" version="16.0" enabled="1">
  <parameter name="bitClearingEdgeCapReg" value="true" />
  <parameter name="bitModifyingOutReg" value="false" />
  <parameter name="captureEdge" value="true" />
  <parameter name="clockRate" value="50000000" />
  <parameter name="direction" value="Input" />
  <parameter name="edgeType" value="ANY" />
  <parameter name="generateIRQ" value="true" />
  <parameter name="irqType" value="EDGE" />
  <parameter name="resetValue" value="0" />
  <parameter name="simDoTestBenchWiring" value="false" />
  <parameter name="simDrivenValue" value="0" />
  <parameter name="width" value="10" />
 </module>
 <module name="hps_0" kind="altera_hps" version="16.0" enabled="1">
  <parameter name="ABSTRACT_REAL_COMPARE_TEST" value="false" />
  <parameter name="ABS_RAM_MEM_INIT_FILENAME" value="meminit" />
  <parameter name="ACV_PHY_CLK_ADD_FR_PHASE" value="0.0" />
  <parameter name="AC_PACKAGE_DESKEW" value="false" />
  <parameter name="AC_ROM_USER_ADD_0" value="0_0000_0000_0000" />
  <parameter name="AC_ROM_USER_ADD_1" value="0_0000_0000_1000" />
  <parameter name="ADDR_ORDER" value="0" />
  <parameter name="ADD_EFFICIENCY_MONITOR" value="false" />
  <parameter name="ADD_EXTERNAL_SEQ_DEBUG_NIOS" value="false" />
  <parameter name="ADVANCED_CK_PHASES" value="false" />
  <parameter name="ADVERTISE_SEQUENCER_SW_BUILD_FILES" value="false" />
  <parameter name="AFI_DEBUG_INFO_WIDTH" value="32" />
  <parameter name="ALTMEMPHY_COMPATIBLE_MODE" value="false" />
  <parameter name="AP_MODE" value="false" />
  <parameter name="AP_MODE_EN" value="0" />
  <parameter name="AUTO_DEVICE_SPEEDGRADE" value="6" />
  <parameter name="AUTO_PD_CYCLES" value="0" />
  <parameter name="AUTO_POWERDN_EN" value="false" />
  <parameter name="AVL_DATA_WIDTH_PORT" value="32,32,32,32,32,32" />
  <parameter name="AVL_MAX_SIZE" value="4" />
  <parameter name="BONDING_OUT_ENABLED" value="false" />
  <parameter name="BOOTFROMFPGA_Enable" value="true" />
  <parameter name="BSEL" value="1" />
  <parameter name="BSEL_EN" value="false" />
  <parameter name="BYTE_ENABLE" value="true" />
  <parameter name="C2P_WRITE_CLOCK_ADD_PHASE" value="0.0" />
  <parameter name="CALIBRATION_MODE" value="Skip" />
  <parameter name="CALIB_REG_WIDTH" value="8" />
  <parameter name="CAN0_Mode" value="N/A" />
  <parameter name="CAN0_PinMuxing" value="Unused" />
  <parameter name="CAN1_Mode" value="N/A" />
  <parameter name="CAN1_PinMuxing" value="Unused" />
  <parameter name="CFG_DATA_REORDERING_TYPE" value="INTER_BANK" />
  <parameter name="CFG_REORDER_DATA" value="true" />
  <parameter name="CFG_TCCD_NS" value="2.5" />
  <parameter name="COMMAND_PHASE" value="0.0" />
```

```
  <parameter name="CONTROLLER_LATENCY" value="5" />
  <parameter name="CORE_DEBUG_CONNECTION" value="EXPORT" />
                                                                                                    <parameter
name="CPORT_TYPE_PORT">Bidirectional,Bidirectional,Bidirectional,Bidirectional,Bidirectional,Bidirectional<
/parameter>
  <parameter name="CSEL" value="0" />
  <parameter name="CSEL_EN" value="false" />
  <parameter name="CTI_Enable" value="false" />
  <parameter name="CTL_AUTOPCH_EN" value="false" />
  <parameter name="CTL_CMD_QUEUE_DEPTH" value="8" />
  <parameter name="CTL_CSR_CONNECTION" value="INTERNAL_JTAG" />
  <parameter name="CTL_CSR_ENABLED" value="false" />
  <parameter name="CTL_CSR_READ_ONLY" value="1" />
  <parameter name="CTL_DEEP_POWERDN_EN" value="false" />
  <parameter name="CTL_DYNAMIC_BANK_ALLOCATION" value="false" />
  <parameter name="CTL_DYNAMIC_BANK_NUM" value="4" />
  <parameter name="CTL_ECC_AUTO_CORRECTION_ENABLED" value="false" />
  <parameter name="CTL_ECC_ENABLED" value="false" />
  <parameter name="CTL_ENABLE_BURST_INTERRUPT" value="true" />
  <parameter name="CTL_ENABLE_BURST_TERMINATE" value="true" />
  <parameter name="CTL_HRB_ENABLED" value="false" />
  <parameter name="CTL_LOOK_AHEAD_DEPTH" value="4" />
  <parameter name="CTL_SELF_REFRESH_EN" value="false" />
  <parameter name="CTL_USR_REFRESH_EN" value="false" />
  <parameter name="CTL_ZQCAL_EN" value="false" />
  <parameter name="CUT_NEW_FAMILY_TIMING" value="true" />
  <parameter name="DAT_DATA_WIDTH" value="32" />
  <parameter name="DEBUGAPB_Enable" value="false" />
  <parameter name="DEBUG_MODE" value="false" />
  <parameter name="DEVICE_DEPTH" value="1" />
  <parameter name="DEVICE_FAMILY_PARAM" value="" />
  <parameter name="DISABLE_CHILD_MESSAGING" value="false" />
  <parameter name="DISCRETE_FLY_BY" value="true" />
  <parameter name="DLL_SHARING_MODE" value="None" />
  <parameter name="DMA_Enable">No,No,No,No,No,No,No,No</parameter>
  <parameter name="DQS_DQSN_MODE" value="DIFFERENTIAL" />
  <parameter name="DQ_INPUT_REG_USE_CLKN" value="false" />
  <parameter name="DUPLICATE_AC" value="false" />
  <parameter name="ED_EXPORT_SEQ_DEBUG" value="false" />
  <parameter name="EMAC0_Mode" value="N/A" />
  <parameter name="EMAC0_PTP" value="false" />
  <parameter name="EMAC0_PinMuxing" value="Unused" />
  <parameter name="EMAC1_Mode" value="RGMII" />
  <parameter name="EMAC1_PTP" value="false" />
  <parameter name="EMAC1_PinMuxing" value="HPS I/O Set 0" />
  <parameter name="ENABLE_ABS_RAM_MEM_INIT" value="false" />
  <parameter name="ENABLE_BONDING" value="false" />
  <parameter name="ENABLE_BURST_MERGE" value="false" />
  <parameter name="ENABLE_CTRL_AVALON_INTERFACE" value="true" />
  <parameter name="ENABLE_DELAY_CHAIN_WRITE" value="false" />
  <parameter name="ENABLE_EMIT_BFM_MASTER" value="false" />
  <parameter name="ENABLE_EXPORT_SEQ_DEBUG_BRIDGE" value="false" />
  <parameter name="ENABLE_EXTRA_REPORTING" value="false" />
  <parameter name="ENABLE_ISS_PROBES" value="false" />
  <parameter name="ENABLE_NON_DESTRUCTIVE_CALIB" value="false" />
  <parameter name="ENABLE_NON_DES_CAL" value="false" />
  <parameter name="ENABLE_NON_DES_CAL_TEST" value="false" />
  <parameter name="ENABLE_SEQUENCER_MARGINING_ON_BY_DEFAULT" value="false" />
  <parameter name="ENABLE_USER_ECC" value="false" />
  <parameter name="EXPORT_AFI_HALF_CLK" value="false" />
  <parameter name="EXTRA_SETTINGS" value="" />
  <parameter name="F2H_AXI_CLOCK_FREQ" value="50000000" />
  <parameter name="F2H_SDRAM0_CLOCK_FREQ" value="100" />
  <parameter name="F2H_SDRAM1_CLOCK_FREQ" value="100" />
  <parameter name="F2H_SDRAM2_CLOCK_FREQ" value="100" />
  <parameter name="F2H_SDRAM3_CLOCK_FREQ" value="100" />
  <parameter name="F2H_SDRAM4_CLOCK_FREQ" value="100" />
  <parameter name="F2H_SDRAM5_CLOCK_FREQ" value="100" />
  <parameter name="F2SCLK_COLDRST_Enable" value="false" />
  <parameter name="F2SCLK_DBGRST_Enable" value="false" />
  <parameter name="F2SCLK_PERIPHCLK_Enable" value="false" />
  <parameter name="F2SCLK_PERIPHCLK_FREQ" value="0" />
  <parameter name="F2SCLK_SDRAMCLK_Enable" value="false" />
  <parameter name="F2SCLK_SDRAMCLK_FREQ" value="0" />
  <parameter name="F2SCLK_WARMRST_Enable" value="false" />
  <parameter name="F2SDRAM_Type" value="" />
  <parameter name="F2SDRAM_Width" value="" />
  <parameter name="F2SINTERRUPT_Enable" value="true" />
  <parameter name="F2S_Width" value="3" />
  <parameter name="FIX_READ_LATENCY" value="8" />
  <parameter name="FORCED_NON_LDC_ADDR_CMD_MEM_CK_INVERT" value="false" />
  <parameter name="FORCED_NUM_WRITE_FR_CYCLE_SHIFTS" value="0" />
  <parameter name="FORCE_DQS_TRACKING" value="AUTO" />
```

```
    <parameter name="FORCE_MAX_LATENCY_COUNT_WIDTH" value="0" />
    <parameter name="FORCE_SEQUENCER_TCL_DEBUG_MODE" value="false" />
    <parameter name="FORCE_SHADOW_REGS" value="AUTO" />
    <parameter name="FORCE_SYNTHESIS_LANGUAGE" value="" />
    <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC0_RX_CLK_IN" value="100" />
    <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC0_TX_CLK_IN" value="100" />
    <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC1_RX_CLK_IN" value="100" />
    <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC1_TX_CLK_IN" value="100" />
    <parameter
       name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC_PTP_REF_CLOCK"
       value="100" />
    <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_I2C0_SCL_IN" value="100" />
    <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_I2C1_SCL_IN" value="100" />
    <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_I2C2_SCL_IN" value="100" />
    <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_I2C3_SCL_IN" value="100" />
    <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_SDIO_CLK_IN" value="100" />
    <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_SPIS0_SCLK_IN" value="100" />
    <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_SPIS1_SCLK_IN" value="100" />
    <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_USB0_CLK_IN" value="100" />
    <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_USB1_CLK_IN" value="100" />
    <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_EMAC0_GTX_CLK" value="100" />
    <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_EMAC0_MD_CLK" value="100.0" />
    <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_EMAC1_GTX_CLK" value="100" />
    <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_EMAC1_MD_CLK" value="100.0" />
    <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_I2C0_CLK" value="100" />
    <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_I2C1_CLK" value="100" />
    <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_I2C2_CLK" value="100" />
    <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_I2C3_CLK" value="100" />
    <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_QSPI_SCLK_OUT" value="100" />
    <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_SDIO_CCLK" value="100" />
    <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_SPIM0_SCLK_OUT" value="100" />
    <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_SPIM1_SCLK_OUT" value="100" />
                                                                                   <parameter
name="GPIO_Enable">No,No,No,No,No,No,No,No,No,Yes,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,
No,No,No,No,No,No,Yes,No,No,No,No,Yes,No,No,No,No,No,No,No,Yes,No,No,No,No,Yes,Yes,No,No,No,No,No,No,Yes,No
,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,N
o,No</parameter>
    <parameter name="GP_Enable" value="false" />
    <parameter name="H2F_AXI_CLOCK_FREQ" value="50000000" />
    <parameter name="H2F_CTI_CLOCK_FREQ" value="100" />
    <parameter name="H2F_DEBUG_APB_CLOCK_FREQ" value="100" />
    <parameter name="H2F_LW_AXI_CLOCK_FREQ" value="50000000" />
    <parameter name="H2F_TPIU_CLOCK_IN_FREQ" value="100" />
    <parameter name="HARD_EMIF" value="true" />
    <parameter name="HCX_COMPAT_MODE" value="false" />
    <parameter name="HHP_HPS" value="true" />
    <parameter name="HHP_HPS_SIMULATION" value="false" />
    <parameter name="HHP_HPS_VERIFICATION" value="false" />
    <parameter name="HLGPI_Enable" value="false" />
    <parameter name="HPS_PROTOCOL" value="DDR3" />
    <parameter name="I2C0_Mode" value="I2C" />
    <parameter name="I2C0_PinMuxing" value="HPS I/O Set 0" />
    <parameter name="I2C1_Mode" value="I2C" />
    <parameter name="I2C1_PinMuxing" value="HPS I/O Set 0" />
    <parameter name="I2C2_Mode" value="N/A" />
    <parameter name="I2C2_PinMuxing" value="Unused" />
    <parameter name="I2C3_Mode" value="N/A" />
    <parameter name="I2C3_PinMuxing" value="Unused" />
    <parameter name="INCLUDE_BOARD_DELAY_MODEL" value="false" />
    <parameter name="INCLUDE_MULTIRANK_BOARD_DELAY_MODEL" value="false" />
    <parameter name="IS_ES_DEVICE" value="false" />
                                                                                   <parameter
name="LOANIO_Enable">No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No
,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,N
o,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No<
/parameter>
    <parameter name="LOCAL_ID_WIDTH" value="8" />
    <parameter name="LRDIMM_EXTENDED_CONFIG">0x0000000000000000</parameter>
    <parameter name="LWH2F_Enable" value="true" />
    <parameter name="MARGIN_VARIATION_TEST" value="false" />
    <parameter name="MAX_PENDING_RD_CMD" value="16" />
    <parameter name="MAX_PENDING_WR_CMD" value="8" />
    <parameter name="MEM_ASR" value="Manual" />
    <parameter name="MEM_ATCL" value="Disabled" />
    <parameter name="MEM_AUTO_LEVELING_MODE" value="true" />
    <parameter name="MEM_BANKADDR_WIDTH" value="3" />
    <parameter name="MEM_BL" value="OTF" />
    <parameter name="MEM_BT" value="Sequential" />
    <parameter name="MEM_CK_PHASE" value="0.0" />
    <parameter name="MEM_CK_WIDTH" value="1" />
    <parameter name="MEM_CLK_EN_WIDTH" value="1" />
    <parameter name="MEM_CLK_FREQ" value="400.0" />
    <parameter name="MEM_CLK_FREQ_MAX" value="800.0" />
```

```
<parameter name="MEM_COL_ADDR_WIDTH" value="10" />
<parameter name="MEM_CS_WIDTH" value="1" />
<parameter name="MEM_DEVICE" value="MISSING_MODEL" />
<parameter name="MEM_DLL_EN" value="true" />
<parameter name="MEM_DQ_PER_DQS" value="8" />
<parameter name="MEM_DQ_WIDTH" value="32" />
<parameter name="MEM_DRV_STR" value="RZQ/7" />
<parameter name="MEM_FORMAT" value="DISCRETE" />
<parameter name="MEM_GUARANTEED_WRITE_INIT" value="false" />
<parameter name="MEM_IF_BOARD_BASE_DELAY" value="10" />
<parameter name="MEM_IF_DM_PINS_EN" value="true" />
<parameter name="MEM_IF_DQSN_EN" value="true" />
<parameter name="MEM_IF_SIM_VALID_WINDOW" value="0" />
<parameter name="MEM_INIT_EN" value="false" />
<parameter name="MEM_INIT_FILE" value="" />
<parameter name="MEM_MIRROR_ADDRESSING" value="0" />
<parameter name="MEM_NUMBER_OF_DIMMS" value="1" />
<parameter name="MEM_NUMBER_OF_RANKS_PER_DEVICE" value="1" />
<parameter name="MEM_NUMBER_OF_RANKS_PER_DIMM" value="1" />
<parameter name="MEM_PD" value="DLL off" />
<parameter name="MEM_RANK_MULTIPLICATION_FACTOR" value="1" />
<parameter name="MEM_ROW_ADDR_WIDTH" value="15" />
<parameter name="MEM_RTT_NOM" value="RZQ/4" />
<parameter name="MEM_RTT_WR" value="RZQ/4" />
<parameter name="MEM_SRT" value="Normal" />
<parameter name="MEM_TCL" value="11" />
<parameter name="MEM_TFAW_NS" value="30.0" />
<parameter name="MEM_TINIT_US" value="500" />
<parameter name="MEM_TMRD_CK" value="4" />
<parameter name="MEM_TRAS_NS" value="35.0" />
<parameter name="MEM_TRCD_NS" value="13.75" />
<parameter name="MEM_TREFI_US" value="7.8" />
<parameter name="MEM_TRFC_NS" value="260.0" />
<parameter name="MEM_TRP_NS" value="13.75" />
<parameter name="MEM_TRRD_NS" value="7.5" />
<parameter name="MEM_TRTP_NS" value="7.5" />
<parameter name="MEM_TWR_NS" value="15.0" />
<parameter name="MEM_TWTR" value="4" />
<parameter name="MEM_USER_LEVELING_MODE" value="Leveling" />
<parameter name="MEM_VENDOR" value="JEDEC" />
<parameter name="MEM_VERBOSE" value="true" />
<parameter name="MEM_VOLTAGE" value="1.5V DDR3" />
<parameter name="MEM_WTCL" value="8" />
<parameter name="NPU_EVENTS_Enable" value="false" />
<parameter name="NRS_MIRROR_PING_PONG_ATSO" value="false" />
<parameter name="NULTICAST_EN" value="false" />
<parameter name="NAND_Mode" value="N/A" />
<parameter name="NAND_PinMuxing" value="Unused" />
<parameter name="NEXTGEN" value="true" />
<parameter name="NIOS_ROM_DATA_WIDTH" value="32" />
<parameter name="NUM_DLL_SHARING_INTERFACES" value="1" />
<parameter name="NUM_EXTRA_REPORT_PATH" value="10" />
<parameter name="NUM_OCT_SHARING_INTERFACES" value="1" />
<parameter name="NUM_OF_PORTS" value="1" />
<parameter name="NUM_PLL_SHARING_INTERFACES" value="1" />
<parameter name="OCT_SHARING_MODE" value="None" />
<parameter name="P2C_READ_CLOCK_ADD_PHASE" value="0.0" />
<parameter name="PACKAGE_DESKEW" value="false" />
<parameter name="PARSE_FRIENDLY_DEVICE_FAMILY_PARAM" value="" />
<parameter name="PARSE_FRIENDLY_DEVICE_FAMILY_PARAM_VALID" value="false" />
<parameter name="PHY_CSR_CONNECTION" value="INTERNAL_JTAG" />
<parameter name="PHY_CSR_ENABLED" value="false" />
<parameter name="PHY_ONLY" value="false" />
<parameter name="PINGPONGPHY_EN" value="false" />
<parameter name="PLL_ADDR_CMD_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_ADDR_CMD_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_ADDR_CMD_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_ADDR_CMD_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_ADDR_CMD_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_ADDR_CMD_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_AFI_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_AFI_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_AFI_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_AFI_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_AFI_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_AFI_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_AFI_HALF_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_AFI_HALF_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_AFI_HALF_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_AFI_HALF_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_AFI_HALF_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_AFI_HALF_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_AFI_PHY_CLK_DIV_PARAM" value="0" />
```

```
<parameter name="PLL_AFI_PHY_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_AFI_PHY_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_AFI_PHY_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_AFI_PHY_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_AFI_PHY_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_C2P_WRITE_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_C2P_WRITE_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_C2P_WRITE_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_C2P_WRITE_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_C2P_WRITE_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_C2P_WRITE_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_CLK_PARAM_VALID" value="false" />
<parameter name="PLL_CONFIG_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_CONFIG_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_CONFIG_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_CONFIG_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_CONFIG_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_CONFIG_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_DR_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_DR_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_DR_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_DR_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_DR_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_DR_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_HR_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_HR_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_HR_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_HR_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_HR_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_HR_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_LOCATION" value="Top_Bottom" />
<parameter name="PLL_MEM_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_MEM_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_MEM_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_MEM_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_MEM_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_MEM_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_NIOS_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_NIOS_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_NIOS_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_NIOS_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_NIOS_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_NIOS_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_P2C_READ_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_P2C_READ_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_P2C_READ_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_P2C_READ_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_P2C_READ_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_P2C_READ_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_SHARING_MODE" value="None" />
<parameter name="PLL_WRITE_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_WRITE_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_WRITE_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_WRITE_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_WRITE_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_WRITE_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="POWER_OF_TWO_BUS" value="false" />
<parameter name="PRIORITY_PORT" value="1,1,1,1,1,1" />
<parameter name="QSPI_Mode" value="1 SS" />
<parameter name="QSPI_PinMuxing" value="HPS I/O Set 0" />
<parameter name="RATE" value="Full" />
<parameter name="RDIMM_CONFIG" value="0000000000000000" />
<parameter name="READ_DQ_DQS_CLOCK_SOURCE" value="INVERTED_DQS_BUS" />
<parameter name="READ_FIFO_SIZE" value="8" />
<parameter name="REFRESH_BURST_VALIDATION" value="false" />
<parameter name="REFRESH_INTERVAL" value="15080" />
<parameter name="REF_CLK_FREQ" value="25.0" />
<parameter name="REF_CLK_FREQ_MAX_PARAM" value="0.0" />
<parameter name="REF_CLK_FREQ_MIN_PARAM" value="0.0" />
<parameter name="REF_CLK_FREQ_PARAM_VALID" value="false" />
<parameter name="S2FCLK_COLDRST_Enable" value="false" />
<parameter name="S2FCLK_PENDINGRST_Enable" value="false" />
<parameter name="S2FCLK_USER0CLK_Enable" value="false" />
<parameter name="S2FCLK_USER1CLK_Enable" value="false" />
<parameter name="S2FCLK_USER1CLK_FREQ" value="100.0" />
<parameter name="S2FCLK_USER2CLK" value="1" />
<parameter name="S2FCLK_USER2CLK_Enable" value="false" />
<parameter name="S2FCLK_USER2CLK_FREQ" value="100.0" />
<parameter name="S2FINTERRUPT_CAN_Enable" value="false" />
<parameter name="S2FINTERRUPT_CLOCKPERIPHERAL_Enable" value="false" />
<parameter name="S2FINTERRUPT_CTI_Enable" value="false" />
<parameter name="S2FINTERRUPT_DMA_Enable" value="false" />
<parameter name="S2FINTERRUPT_EMAC_Enable" value="false" />
```

```
<parameter name="S2FINTERRUPT_FPGAMANAGER_Enable" value="false" />
<parameter name="S2FINTERRUPT_GPIO_Enable" value="false" />
<parameter name="S2FINTERRUPT_I2CEMAC_Enable" value="false" />
<parameter name="S2FINTERRUPT_I2CPERIPHERAL_Enable" value="false" />
<parameter name="S2FINTERRUPT_L4TIMER_Enable" value="false" />
<parameter name="S2FINTERRUPT_NAND_Enable" value="false" />
<parameter name="S2FINTERRUPT_OSCTIMER_Enable" value="false" />
<parameter name="S2FINTERRUPT_QSPI_Enable" value="false" />
<parameter name="S2FINTERRUPT_SDMMC_Enable" value="false" />
<parameter name="S2FINTERRUPT_SPIMASTER_Enable" value="false" />
<parameter name="S2FINTERRUPT_SPISLAVE_Enable" value="false" />
<parameter name="S2FINTERRUPT_UART_Enable" value="false" />
<parameter name="S2FINTERRUPT_USB_Enable" value="false" />
<parameter name="S2FINTERRUPT_WATCHDOG_Enable" value="false" />
<parameter name="S2F_Width" value="2" />
<parameter name="SDIO_Mode" value="4-bit Data" />
<parameter name="SDIO_PinMuxing" value="HPS I/O Set 0" />
<parameter name="SEQUENCER_TYPE" value="NIOS" />
<parameter name="SEQ_MODE" value="0" />
<parameter name="SKIP_MEM_INIT" value="true" />
<parameter name="SOPC_COMPAT_RESET" value="false" />
<parameter name="SPEED_GRADE" value="7" />
<parameter name="SPIM0_Mode" value="N/A" />
<parameter name="SPIM0_PinMuxing" value="Unused" />
<parameter name="SPIM1_Mode" value="Single Slave Select" />
<parameter name="SPIM1_PinMuxing" value="HPS I/O Set 0" />
<parameter name="SPIS0_Mode" value="N/A" />
<parameter name="SPIS0_PinMuxing" value="Unused" />
<parameter name="SPIS1_Mode" value="N/A" />
<parameter name="SPIS1_PinMuxing" value="Unused" />
<parameter name="STARVE_LIMIT" value="10" />
<parameter name="STN_Enable" value="false" />
<parameter name="SYS_INFO_DEVICE_FAMILY" value="Cyclone V" />
<parameter name="TEST_Enable" value="false" />
<parameter name="TIMING_BOARD_AC_EYE_REDUCTION_H" value="0.0" />
<parameter name="TIMING_BOARD_AC_EYE_REDUCTION_SU" value="0.0" />
<parameter name="TIMING_BOARD_AC_SKEW" value="0.03" />
<parameter name="TIMING_BOARD_AC_SLEW_RATE" value="1.0" />
<parameter name="TIMING_BOARD_AC_TO_CK_SKEW" value="0.0" />
<parameter name="TIMING_BOARD_CK_CKN_SLEW_RATE" value="2.0" />
<parameter name="TIMING_BOARD_DELTA_DQS_ARRIVAL_TIME" value="0.0" />
<parameter name="TIMING_BOARD_DELTA_READ_DQS_ARRIVAL_TIME" value="0.0" />
<parameter name="TIMING_BOARD_DERATE_METHOD" value="AUTO" />
<parameter name="TIMING_BOARD_DQS_DQSN_SLEW_RATE" value="2.0" />
<parameter name="TIMING_BOARD_DQ_EYE_REDUCTION" value="0.0" />
<parameter name="TIMING_BOARD_DQ_SLEW_RATE" value="1.0" />
<parameter name="TIMING_BOARD_DQ_TO_DQS_SKEW" value="0.0" />
<parameter name="TIMING_BOARD_ISI_METHOD" value="AUTO" />
<parameter name="TIMING_BOARD_MAX_CK_DELAY" value="0.03" />
<parameter name="TIMING_BOARD_MAX_DQS_DELAY" value="0.02" />
<parameter name="TIMING_BOARD_READ_DQ_EYE_REDUCTION" value="0.0" />
<parameter name="TIMING_BOARD_SKEW_BETWEEN_DIMMS" value="0.05" />
<parameter name="TIMING_BOARD_SKEW_BETWEEN_DQS" value="0.08" />
<parameter name="TIMING_BOARD_SKEW_CKDQS_DIMM_MAX" value="0.16" />
<parameter name="TIMING_BOARD_SKEW_CKDQS_DIMM_MIN" value="0.09" />
<parameter name="TIMING_BOARD_SKEW_WITHIN_DQS" value="0.01" />
<parameter name="TIMING_BOARD_TDH" value="0.0" />
<parameter name="TIMING_BOARD_TDS" value="0.0" />
<parameter name="TIMING_BOARD_TIH" value="0.0" />
<parameter name="TIMING_BOARD_TIS" value="0.0" />
<parameter name="TIMING_TDH" value="55" />
<parameter name="TIMING_TDQSCK" value="255" />
<parameter name="TIMING_TDQSCKDL" value="1200" />
<parameter name="TIMING_TDQSCKDM" value="900" />
<parameter name="TIMING_TDQSCKDS" value="450" />
<parameter name="TIMING_TDQSH" value="0.35" />
<parameter name="TIMING_TDQSQ" value="125" />
<parameter name="TIMING_TDQSS" value="0.25" />
<parameter name="TIMING_TDS" value="10" />
<parameter name="TIMING_TDSH" value="0.2" />
<parameter name="TIMING_TDSS" value="0.2" />
<parameter name="TIMING_TIH" value="140" />
<parameter name="TIMING_TIS" value="180" />
<parameter name="TIMING_TQH" value="0.38" />
<parameter name="TIMING_TQHS" value="300" />
<parameter name="TIMING_TQSH" value="0.4" />
<parameter name="TPIUFPGA_Enable" value="false" />
<parameter name="TPIUFPGA_alt" value="false" />
<parameter name="TRACE_Mode" value="N/A" />
<parameter name="TRACE_PinMuxing" value="Unused" />
<parameter name="TRACKING_ERROR_TEST" value="false" />
<parameter name="TRACKING_WATCH_TEST" value="false" />
<parameter name="TREFI" value="35100" />
```

```
<parameter name="TRFC" value="350" />
<parameter name="UART0_Mode" value="No Flow Control" />
<parameter name="UART0_PinMuxing" value="HPS I/O Set 0" />
<parameter name="UART1_Mode" value="N/A" />
<parameter name="UART1_PinMuxing" value="Unused" />
<parameter name="USB0_Mode" value="N/A" />
<parameter name="USB0_PinMuxing" value="Unused" />
<parameter name="USB1_Mode" value="SDR" />
<parameter name="USB1_PinMuxing" value="HPS I/O Set 0" />
<parameter name="USER_DEBUG_LEVEL" value="1" />
<parameter name="USE_AXI_ADAPTOR" value="false" />
<parameter name="USE_FAKE_PHY" value="false" />
<parameter name="USE_MEM_CLK_FREQ" value="false" />
<parameter name="USE_MM_ADAPTOR" value="true" />
<parameter name="USE_SEQUENCER_BFM" value="false" />
<parameter name="WEIGHT_PORT" value="0,0,0,0,0,0" />
<parameter name="WRBUFFER_ADDR_WIDTH" value="6" />
<parameter name="can0_clk_div" value="1" />
<parameter name="can1_clk_div" value="1" />
<parameter name="configure_advanced_parameters" value="false" />
<parameter name="customize_device_pll_info" value="false" />
<parameter name="dbctrl_stayosc1" value="true" />
<parameter name="dbg_at_clk_div" value="0" />
<parameter name="dbg_clk_div" value="1" />
<parameter name="dbg_trace_clk_div" value="0" />
<parameter name="desired_can0_clk_mhz" value="100.0" />
<parameter name="desired_can1_clk_mhz" value="100.0" />
<parameter name="desired_cfg_clk_mhz" value="100.0" />
<parameter name="desired_emac0_clk_mhz" value="250.0" />
<parameter name="desired_emac1_clk_mhz" value="250.0" />
<parameter name="desired_gpio_db_clk_hz" value="32000" />
<parameter name="desired_l4_mp_clk_mhz" value="100.0" />
<parameter name="desired_l4_sp_clk_mhz" value="100.0" />
<parameter name="desired_mpu_clk_mhz" value="800.0" />
<parameter name="desired_nand_clk_mhz" value="12.5" />
<parameter name="desired_qspi_clk_mhz" value="400.0" />
<parameter name="desired_sdmmc_clk_mhz" value="200.0" />
<parameter name="desired_spi_m_clk_mhz" value="200.0" />
<parameter name="desired_usb_mp_clk_mhz" value="200.0" />
<parameter name="device_name" value="5CSEMA5F31C6" />
  <parameter  name="device_pll_info_manual">(320000000  1600000000)  (320000000  1000000000)  (800000000
400000000 400000000)</parameter>
<parameter name="emac1_clk_mhz" value="25.0" />
<parameter name="emac2_clk_mhz" value="25.0" />
<parameter name="gpio_db_clk_div" value="6249" />
<parameter name="l3_mp_clk_div" value="1" />
<parameter name="l3_sp_clk_div" value="1" />
<parameter name="l4_mp_clk_div" value="1" />
<parameter name="l4_mp_clk_source" value="1" />
<parameter name="l4_sp_clk_div" value="1" />
<parameter name="l4_sp_clk_source" value="1" />
<parameter name="main_pll_c3" value="3" />
<parameter name="main_pll_c4" value="3" />
<parameter name="main_pll_c5" value="15" />
<parameter name="main_pll_m" value="63" />
<parameter name="main_pll_n" value="0" />
<parameter name="nand_clk_source" value="2" />
<parameter name="periph_pll_c0" value="3" />
<parameter name="periph_pll_c1" value="3" />
<parameter name="periph_pll_c2" value="1" />
<parameter name="periph_pll_c3" value="19" />
<parameter name="periph_pll_c4" value="4" />
<parameter name="periph_pll_c5" value="9" />
<parameter name="periph_pll_m" value="79" />
<parameter name="periph_pll_n" value="1" />
<parameter name="periph_pll_source" value="0" />
<parameter name="qspi_clk_source" value="1" />
<parameter name="quartus_ini_hps_emif_pll" value="false" />
<parameter
   name="quartus_ini_hps_ip_enable_all_peripheral_fpga_interfaces"
   value="false" />
<parameter name="quartus_ini_hps_ip_enable_bsel_osel" value="false" />
<parameter
   name="quartus_ini_hps_ip_enable_emac0_peripheral_fpga_interface"
   value="false" />
<parameter
   name="quartus_ini_hps_ip_enable_low_speed_serial_fpga_interfaces"
   value="false" />
<parameter name="quartus_ini_hps_ip_enable_test_interface" value="false" />
<parameter name="quartus_ini_hps_ip_f2sdram_bonding_out" value="false" />
<parameter name="quartus_ini_hps_ip_fast_f2sdram_sim_model" value="false" />
<parameter name="quartus_ini_hps_ip_suppress_sdram_synth" value="false" />
<parameter name="sdmmc_clk_source" value="2" />
```

```
 <parameter name="show_advanced_parameters" value="false" />
 <parameter name="show_debug_info_as_warning_msg" value="false" />
 <parameter name="show_warning_as_error_msg" value="false" />
 <parameter name="spi_m_clk_div" value="8" />
 <parameter name="ush_mp_clk_div" value="0" />
 <parameter name="use_default_mpu_clk" value="true" />
</module>
<module
  name="intr_capturer_0"
  kind="intr_capturer"
  version="100.99.98.97"
  enabled="1">
 <parameter name="AUTO_INTERRUPT_RECEIVER_INTERRUPTS_USED" value="7" />
 <parameter name="NUM_INTR" value="32" />
</module>
<module
  name="jtag_uart"
  kind="altera_avalon_jtag_uart"
  version="16.0"
  enabled="1">
 <parameter name="allowMultipleConnections" value="true" />
 <parameter name="avalonSpec" value="2.0" />
 <parameter name="clkFreq" value="50000000" />
 <parameter name="hubInstanceID" value="0" />
 <parameter name="readBufferDepth" value="64" />
 <parameter name="readIRQThreshold" value="8" />
 <parameter name="simInputCharacterStrean" value="" />
 <parameter name="simInteractiveOptions">INTERACTIVE_ASCII_OUTPUT</parameter>
 <parameter name="useRegistersForReadBuffer" value="false" />
 <parameter name="useRegistersForWriteBuffer" value="false" />
 <parameter name="useRelativePathForSimFile" value="false" />
 <parameter name="writeBufferDepth" value="64" />
 <parameter name="writeIRQThreshold" value="8" />
</module>
<module name="led_pio" kind="altera_avalon_pio" version="16.0" enabled="1">
 <parameter name="bitClearingEdgeCapReg" value="false" />
 <parameter name="bitModifyingOutReg" value="false" />
 <parameter name="captureEdge" value="false" />
 <parameter name="clockRate" value="50000000" />
 <parameter name="direction" value="Output" />
 <parameter name="edgeType" value="RISING" />
 <parameter name="generateIRQ" value="false" />
 <parameter name="irqType" value="LEVEL" />
 <parameter name="resetValue" value="1" />
 <parameter name="simDoTestBenchWiring" value="false" />
 <parameter name="simDrivenValue" value="0" />
 <parameter name="width" value="10" />
</module>
<module
  name="master_non_sec"
  kind="altera_jtag_avalon_master"
  version="16.1"
  enabled="1">
 <parameter name="AUTO_DEVICE" value="5CSEMA5F31C6" />
 <parameter name="AUTO_DEVICE_FAMILY" value="Cyclone V" />
 <parameter name="AUTO_DEVICE_SPEEDGRADE" value="6" />
 <parameter name="COMPONENT_CLOCK" value="0" />
 <parameter name="FAST_VER" value="0" />
 <parameter name="FIFO_DEPTHS" value="2" />
 <parameter name="PLI_PORT" value="50000" />
 <parameter name="USE_PLI" value="1" />
</module>
<module
  name="master_secure"
  kind="altera_jtag_avalon_master"
  version="16.0"
  enabled="1">
 <parameter name="AUTO_DEVICE" value="5CSEMA5F31C6" />
 <parameter name="AUTO_DEVICE_FAMILY" value="Cyclone V" />
 <parameter name="AUTO_DEVICE_SPEEDGRADE" value="6" />
 <parameter name="COMPONENT_CLOCK" value="0" />
 <parameter name="FAST_VER" value="0" />
 <parameter name="FIFO_DEPTHS" value="2" />
 <parameter name="PLI_PORT" value="50000" />
 <parameter name="USE_PLI" value="0" />
</module>
<module
  name="ogps_qsad_store_ack"
  kind="altera_avalon_pio"
  version="16.1"
  enabled="1">
 <parameter name="bitClearingEdgeCapReg" value="false" />
 <parameter name="bitModifyingOutReg" value="false" />
```

```
 <parameter name="captureEdge" value="false" />
 <parameter name="clockRate" value="50000000" />
 <parameter name="direction" value="Output" />
 <parameter name="edgeType" value="RISING" />
 <parameter name="generateIRQ" value="false" />
 <parameter name="irqType" value="LEVEL" />
 <parameter name="resetValue" value="0" />
 <parameter name="simDoTestBenchWiring" value="false" />
 <parameter name="simDrivenValue" value="0" />
 <parameter name="width" value="1" />
</module>
<module
  name="ogpu_quad_store_data_high"
  kind="altera_avalon_pio"
  version="16.0"
  enabled="1">
 <parameter name="bitClearingEdgeCapReg" value="false" />
 <parameter name="bitModifyingOutReg" value="false" />
 <parameter name="captureEdge" value="false" />
 <parameter name="clockRate" value="50000000" />
 <parameter name="direction" value="Input" />
 <parameter name="edgeType" value="RISING" />
 <parameter name="generateIRQ" value="false" />
 <parameter name="irqType" value="LEVEL" />
 <parameter name="resetValue" value="0" />
 <parameter name="simDoTestBenchWiring" value="false" />
 <parameter name="simDrivenValue" value="0" />
 <parameter name="width" value="32" />
</module>
<module
  name="ogpu_quad_store_data_low"
  kind="altera_avalon_pio"
  version="16.0"
  enabled="1">
 <parameter name="bitClearingEdgeCapReg" value="false" />
 <parameter name="bitModifyingOutReg" value="false" />
 <parameter name="captureEdge" value="false" />
 <parameter name="clockRate" value="50000000" />
 <parameter name="direction" value="Input" />
 <parameter name="edgeType" value="RISING" />
 <parameter name="generateIRQ" value="false" />
 <parameter name="irqType" value="LEVEL" />
 <parameter name="resetValue" value="0" />
 <parameter name="simDoTestBenchWiring" value="false" />
 <parameter name="simDrivenValue" value="0" />
 <parameter name="width" value="32" />
</module>
<module
  name="ogpu_quad_store_req"
  kind="altera_avalon_pio"
  version="16.0"
  enabled="1">
 <parameter name="bitClearingEdgeCapReg" value="false" />
 <parameter name="bitModifyingOutReg" value="false" />
 <parameter name="captureEdge" value="false" />
 <parameter name="clockRate" value="50000000" />
 <parameter name="direction" value="Input" />
 <parameter name="edgeType" value="RISING" />
 <parameter name="generateIRQ" value="false" />
 <parameter name="irqType" value="LEVEL" />
 <parameter name="resetValue" value="0" />
 <parameter name="simDoTestBenchWiring" value="false" />
 <parameter name="simDrivenValue" value="0" />
 <parameter name="width" value="1" />
</module>
<module
  name="ogpu_raster_unit_clip_rect0"
  kind="altera_avalon_pio"
  version="16.0"
  enabled="1">
 <parameter name="bitClearingEdgeCapReg" value="false" />
 <parameter name="bitModifyingOutReg" value="false" />
 <parameter name="captureEdge" value="false" />
 <parameter name="clockRate" value="50000000" />
 <parameter name="direction" value="Output" />
 <parameter name="edgeType" value="RISING" />
 <parameter name="generateIRQ" value="false" />
 <parameter name="irqType" value="LEVEL" />
 <parameter name="resetValue" value="0" />
 <parameter name="simDoTestBenchWiring" value="false" />
 <parameter name="simDrivenValue" value="0" />
 <parameter name="width" value="32" />
</module>
```

```
<module
  name="ogpu_raster_unit_clip_rect1"
  kind="altera_avalon_pio"
  version="16.0"
  enabled="1">
 <parameter name="bitClearingEdgeCapReg" value="false" />
 <parameter name="bitModifyingOutReg" value="false" />
 <parameter name="captureEdge" value="false" />
 <parameter name="clockRate" value="50000000" />
 <parameter name="direction" value="Output" />
 <parameter name="edgeType" value="RISING" />
 <parameter name="generateIRQ" value="false" />
 <parameter name="irqType" value="LEVEL" />
 <parameter name="resetValue" value="0" />
 <parameter name="simDoTestBenchWiring" value="false" />
 <parameter name="simDrivenValue" value="0" />
 <parameter name="width" value="32" />
</module>
<module
  name="ogpu_raster_unit_command"
  kind="altera_avalon_pio"
  version="16.0"
  enabled="1">
 <parameter name="bitClearingEdgeCapReg" value="false" />
 <parameter name="bitModifyingOutReg" value="false" />
 <parameter name="captureEdge" value="false" />
 <parameter name="clockRate" value="50000000" />
 <parameter name="direction" value="Output" />
 <parameter name="edgeType" value="RISING" />
 <parameter name="generateIRQ" value="false" />
 <parameter name="irqType" value="LEVEL" />
 <parameter name="resetValue" value="0" />
 <parameter name="simDoTestBenchWiring" value="false" />
 <parameter name="simDrivenValue" value="0" />
 <parameter name="width" value="8" />
</module>
<module
  name="ogpu_raster_unit_depth_coef_a"
  kind="altera_avalon_pio"
  version="16.0"
  enabled="1">
 <parameter name="bitClearingEdgeCapReg" value="false" />
 <parameter name="bitModifyingOutReg" value="false" />
 <parameter name="captureEdge" value="false" />
 <parameter name="clockRate" value="50000000" />
 <parameter name="direction" value="Output" />
 <parameter name="edgeType" value="RISING" />
 <parameter name="generateIRQ" value="false" />
 <parameter name="irqType" value="LEVEL" />
 <parameter name="resetValue" value="0" />
 <parameter name="simDoTestBenchWiring" value="false" />
 <parameter name="simDrivenValue" value="0" />
 <parameter name="width" value="32" />
</module>
<module
  name="ogpu_raster_unit_depth_coef_b"
  kind="altera_avalon_pio"
  version="16.0"
  enabled="1">
 <parameter name="bitClearingEdgeCapReg" value="false" />
 <parameter name="bitModifyingOutReg" value="false" />
 <parameter name="captureEdge" value="false" />
 <parameter name="clockRate" value="50000000" />
 <parameter name="direction" value="Output" />
 <parameter name="edgeType" value="RISING" />
 <parameter name="generateIRQ" value="false" />
 <parameter name="irqType" value="LEVEL" />
 <parameter name="resetValue" value="0" />
 <parameter name="simDoTestBenchWiring" value="false" />
 <parameter name="simDrivenValue" value="0" />
 <parameter name="width" value="32" />
</module>
<module
  name="ogpu_raster_unit_depth_coef_c"
  kind="altera_avalon_pio"
  version="16.0"
  enabled="1">
 <parameter name="bitClearingEdgeCapReg" value="false" />
 <parameter name="bitModifyingOutReg" value="false" />
 <parameter name="captureEdge" value="false" />
 <parameter name="clockRate" value="50000000" />
 <parameter name="direction" value="Output" />
 <parameter name="edgeType" value="RISING" />
```

```
 <parameter name="generateIRQ" value="false" />
 <parameter name="irqType" value="LEVEL" />
 <parameter name="resetValue" value="0" />
 <parameter name="simDoTestBenchWiring" value="false" />
 <parameter name="simDrivenValue" value="0" />
 <parameter name="width" value="32" />
</module>
<module
  name="ogpu_raster_unit_quad_buffer_addr_high"
  kind="altera_avalon_pio"
  version="16.0"
  enabled="1">
 <parameter name="bitClearingEdgeCapReg" value="false" />
 <parameter name="bitModifyingOutReg" value="false" />
 <parameter name="captureEdge" value="false" />
 <parameter name="clockRate" value="50000000" />
 <parameter name="direction" value="Output" />
 <parameter name="edgeType" value="RISING" />
 <parameter name="generateIRQ" value="false" />
 <parameter name="irqType" value="LEVEL" />
 <parameter name="resetValue" value="0" />
 <parameter name="simDoTestBenchWiring" value="false" />
 <parameter name="simDrivenValue" value="0" />
 <parameter name="width" value="32" />
</module>
<module
  name="ogpu_raster_unit_quad_buffer_addr_low"
  kind="altera_avalon_pio"
  version="16.0"
  enabled="1">
 <parameter name="bitClearingEdgeCapReg" value="false" />
 <parameter name="bitModifyingOutReg" value="false" />
 <parameter name="captureEdge" value="false" />
 <parameter name="clockRate" value="50000000" />
 <parameter name="direction" value="Output" />
 <parameter name="edgeType" value="RISING" />
 <parameter name="generateIRQ" value="false" />
 <parameter name="irqType" value="LEVEL" />
 <parameter name="resetValue" value="0" />
 <parameter name="simDoTestBenchWiring" value="false" />
 <parameter name="simDrivenValue" value="0" />
 <parameter name="width" value="32" />
</module>
<module
  name="ogpu_raster_unit_status"
  kind="altera_avalon_pio"
  version="16.0"
  enabled="1">
 <parameter name="bitClearingEdgeCapReg" value="false" />
 <parameter name="bitModifyingOutReg" value="false" />
 <parameter name="captureEdge" value="false" />
 <parameter name="clockRate" value="50000000" />
 <parameter name="direction" value="Input" />
 <parameter name="edgeType" value="RISING" />
 <parameter name="generateIRQ" value="false" />
 <parameter name="irqType" value="LEVEL" />
 <parameter name="resetValue" value="0" />
 <parameter name="simDoTestBenchWiring" value="false" />
 <parameter name="simDrivenValue" value="0" />
 <parameter name="width" value="32" />
</module>
<module
  name="ogpu_raster_unit_tile0"
  kind="altera_avalon_pio"
  version="16.0"
  enabled="1">
 <parameter name="bitClearingEdgeCapReg" value="false" />
 <parameter name="bitModifyingOutReg" value="false" />
 <parameter name="captureEdge" value="false" />
 <parameter name="clockRate" value="50000000" />
 <parameter name="direction" value="Output" />
 <parameter name="edgeType" value="RISING" />
 <parameter name="generateIRQ" value="false" />
 <parameter name="irqType" value="LEVEL" />
 <parameter name="resetValue" value="0" />
 <parameter name="simDoTestBenchWiring" value="false" />
 <parameter name="simDrivenValue" value="0" />
 <parameter name="width" value="32" />
</module>
<module
  name="ogpu_raster_unit_tile1"
  kind="altera_avalon_pio"
  version="16.0"
```

```
    enabled="1">
   <parameter name="bitClearingEdgeCapReg" value="false" />
   <parameter name="bitModifyingOutReg" value="false" />
   <parameter name="captureEdge" value="false" />
   <parameter name="clockRate" value="50000000" />
   <parameter name="direction" value="Output" />
   <parameter name="edgeType" value="RISING" />
   <parameter name="generateIRQ" value="false" />
   <parameter name="irqType" value="LEVEL" />
   <parameter name="resetValue" value="0" />
   <parameter name="simDoTestBenchWiring" value="false" />
   <parameter name="simDrivenValue" value="0" />
   <parameter name="width" value="32" />
  </module>
  <module
    name="ogpu_raster_unit_v0x"
    kind="altera_avalon_pio"
    version="16.0"
    enabled="1">
   <parameter name="bitClearingEdgeCapReg" value="false" />
   <parameter name="bitModifyingOutReg" value="false" />
   <parameter name="captureEdge" value="false" />
   <parameter name="clockRate" value="50000000" />
   <parameter name="direction" value="Output" />
   <parameter name="edgeType" value="RISING" />
   <parameter name="generateIRQ" value="false" />
   <parameter name="irqType" value="LEVEL" />
   <parameter name="resetValue" value="0" />
   <parameter name="simDoTestBenchWiring" value="false" />
   <parameter name="simDrivenValue" value="0" />
   <parameter name="width" value="16" />
  </module>
  <module
    name="ogpu_raster_unit_v0y"
    kind="altera_avalon_pio"
    version="16.0"
    enabled="1">
   <parameter name="bitClearingEdgeCapReg" value="false" />
   <parameter name="bitModifyingOutReg" value="false" />
   <parameter name="captureEdge" value="false" />
   <parameter name="clockRate" value="50000000" />
   <parameter name="direction" value="Output" />
   <parameter name="edgeType" value="RISING" />
   <parameter name="generateIRQ" value="false" />
   <parameter name="irqType" value="LEVEL" />
   <parameter name="resetValue" value="0" />
   <parameter name="simDoTestBenchWiring" value="false" />
   <parameter name="simDrivenValue" value="0" />
   <parameter name="width" value="16" />
  </module>
  <module
    name="ogpu_raster_unit_v0z"
    kind="altera_avalon_pio"
    version="16.0"
    enabled="1">
   <parameter name="bitClearingEdgeCapReg" value="false" />
   <parameter name="bitModifyingOutReg" value="false" />
   <parameter name="captureEdge" value="false" />
   <parameter name="clockRate" value="50000000" />
   <parameter name="direction" value="Output" />
   <parameter name="edgeType" value="RISING" />
   <parameter name="generateIRQ" value="false" />
   <parameter name="irqType" value="LEVEL" />
   <parameter name="resetValue" value="0" />
   <parameter name="simDoTestBenchWiring" value="false" />
   <parameter name="simDrivenValue" value="0" />
   <parameter name="width" value="16" />
  </module>
  <module
    name="ogpu_raster_unit_v1x"
    kind="altera_avalon_pio"
    version="16.0"
    enabled="1">
   <parameter name="bitClearingEdgeCapReg" value="false" />
   <parameter name="bitModifyingOutReg" value="false" />
   <parameter name="captureEdge" value="false" />
   <parameter name="clockRate" value="50000000" />
   <parameter name="direction" value="Output" />
   <parameter name="edgeType" value="RISING" />
   <parameter name="generateIRQ" value="false" />
   <parameter name="irqType" value="LEVEL" />
   <parameter name="resetValue" value="0" />
   <parameter name="simDoTestBenchWiring" value="false" />
```

```
    <parameter name="simDrivenValue" value="0" />
    <parameter name="width" value="16" />
  </module>
  <module
    name="ogpu_raster_unit_v1y"
    kind="altera_avalon_pio"
    version="16.0"
    enabled="1">
    <parameter name="bitClearingEdgeCapReg" value="false" />
    <parameter name="bitModifyingOutReg" value="false" />
    <parameter name="captureEdge" value="false" />
    <parameter name="clockRate" value="50000000" />
    <parameter name="direction" value="Output" />
    <parameter name="edgeType" value="RISING" />
    <parameter name="generateIRQ" value="false" />
    <parameter name="irqType" value="LEVEL" />
    <parameter name="resetValue" value="0" />
    <parameter name="simDoTestBenchWiring" value="false" />
    <parameter name="simDrivenValue" value="0" />
    <parameter name="width" value="16" />
  </module>
  <module
    name="ogpu_raster_unit_v1z"
    kind="altera_avalon_pio"
    version="16.0"
    enabled="1">
    <parameter name="bitClearingEdgeCapReg" value="false" />
    <parameter name="bitModifyingOutReg" value="false" />
    <parameter name="captureEdge" value="false" />
    <parameter name="clockRate" value="50000000" />
    <parameter name="direction" value="Output" />
    <parameter name="edgeType" value="RISING" />
    <parameter name="generateIRQ" value="false" />
    <parameter name="irqType" value="LEVEL" />
    <parameter name="resetValue" value="0" />
    <parameter name="simDoTestBenchWiring" value="false" />
    <parameter name="simDrivenValue" value="0" />
    <parameter name="width" value="16" />
  </module>
  <module
    name="ogpu_raster_unit_v2x"
    kind="altera_avalon_pio"
    version="16.0"
    enabled="1">
    <parameter name="bitClearingEdgeCapReg" value="false" />
    <parameter name="bitModifyingOutReg" value="false" />
    <parameter name="captureEdge" value="false" />
    <parameter name="clockRate" value="50000000" />
    <parameter name="direction" value="Output" />
    <parameter name="edgeType" value="RISING" />
    <parameter name="generateIRQ" value="false" />
    <parameter name="irqType" value="LEVEL" />
    <parameter name="resetValue" value="0" />
    <parameter name="simDoTestBenchWiring" value="false" />
    <parameter name="simDrivenValue" value="0" />
    <parameter name="width" value="16" />
  </module>
  <module
    name="ogpu_raster_unit_v2y"
    kind="altera_avalon_pio"
    version="16.0"
    enabled="1">
    <parameter name="bitClearingEdgeCapReg" value="false" />
    <parameter name="bitModifyingOutReg" value="false" />
    <parameter name="captureEdge" value="false" />
    <parameter name="clockRate" value="50000000" />
    <parameter name="direction" value="Output" />
    <parameter name="edgeType" value="RISING" />
    <parameter name="generateIRQ" value="false" />
    <parameter name="irqType" value="LEVEL" />
    <parameter name="resetValue" value="0" />
    <parameter name="simDoTestBenchWiring" value="false" />
    <parameter name="simDrivenValue" value="0" />
    <parameter name="width" value="16" />
  </module>
  <module
    name="ogpu_raster_unit_v2z"
    kind="altera_avalon_pio"
    version="16.0"
    enabled="1">
    <parameter name="bitClearingEdgeCapReg" value="false" />
    <parameter name="bitModifyingOutReg" value="false" />
    <parameter name="captureEdge" value="false" />
```

```
 <parameter name="clockRate" value="50000000" />
 <parameter name="direction" value="Output" />
 <parameter name="edgeType" value="RISING" />
 <parameter name="generateIRQ" value="false" />
 <parameter name="irqType" value="LEVEL" />
 <parameter name="resetValue" value="0" />
 <parameter name="simDoTestBenchWiring" value="false" />
 <parameter name="simDrivenValue" value="0" />
 <parameter name="width" value="16" />
</module>
<module name="cpu_reset" kind="altera_avalon_pio" version="16.0" enabled="1">
 <parameter name="bitClearingEdgeCapReg" value="false" />
 <parameter name="bitModifyingOutReg" value="false" />
 <parameter name="captureEdge" value="false" />
 <parameter name="clockRate" value="50000000" />
 <parameter name="direction" value="Output" />
 <parameter name="edgeType" value="RISING" />
 <parameter name="generateIRQ" value="false" />
 <parameter name="irqType" value="LEVEL" />
 <parameter name="resetValue" value="1" />
 <parameter name="simDoTestBenchWiring" value="false" />
 <parameter name="simDrivenValue" value="0" />
 <parameter name="width" value="1" />
</module>
<module name="pll_stream" kind="altera_pll" version="16.0" enabled="1">
 <parameter name="debug_print_output" value="false" />
 <parameter name="debug_use_rbc_taf_method" value="false" />
 <parameter name="device" value="5CSEMA5F31C6" />
 <parameter name="device_family" value="Cyclone V" />
 <parameter name="gui_active_clk" value="false" />
 <parameter name="gui_actual_output_clock_frequency0" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency1" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency10" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency11" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency12" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency13" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency14" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency15" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency16" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency17" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency2" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency3" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency4" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency5" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency6" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency7" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency8" value="0 MHz" />
 <parameter name="gui_actual_output_clock_frequency9" value="0 MHz" />
 <parameter name="gui_actual_phase_shift0" value="0" />
 <parameter name="gui_actual_phase_shift1" value="0" />
 <parameter name="gui_actual_phase_shift10" value="0" />
 <parameter name="gui_actual_phase_shift11" value="0" />
 <parameter name="gui_actual_phase_shift12" value="0" />
 <parameter name="gui_actual_phase_shift13" value="0" />
 <parameter name="gui_actual_phase_shift14" value="0" />
 <parameter name="gui_actual_phase_shift15" value="0" />
 <parameter name="gui_actual_phase_shift16" value="0" />
 <parameter name="gui_actual_phase_shift17" value="0" />
 <parameter name="gui_actual_phase_shift2" value="0" />
 <parameter name="gui_actual_phase_shift3" value="0" />
 <parameter name="gui_actual_phase_shift4" value="0" />
 <parameter name="gui_actual_phase_shift5" value="0" />
 <parameter name="gui_actual_phase_shift6" value="0" />
 <parameter name="gui_actual_phase_shift7" value="0" />
 <parameter name="gui_actual_phase_shift8" value="0" />
 <parameter name="gui_actual_phase_shift9" value="0" />
 <parameter name="gui_cascade_counter0" value="false" />
 <parameter name="gui_cascade_counter1" value="false" />
 <parameter name="gui_cascade_counter10" value="false" />
 <parameter name="gui_cascade_counter11" value="false" />
 <parameter name="gui_cascade_counter12" value="false" />
 <parameter name="gui_cascade_counter13" value="false" />
 <parameter name="gui_cascade_counter14" value="false" />
 <parameter name="gui_cascade_counter15" value="false" />
 <parameter name="gui_cascade_counter16" value="false" />
 <parameter name="gui_cascade_counter17" value="false" />
 <parameter name="gui_cascade_counter2" value="false" />
 <parameter name="gui_cascade_counter3" value="false" />
 <parameter name="gui_cascade_counter4" value="false" />
 <parameter name="gui_cascade_counter5" value="false" />
 <parameter name="gui_cascade_counter6" value="false" />
 <parameter name="gui_cascade_counter7" value="false" />
 <parameter name="gui_cascade_counter8" value="false" />
```

```
<parameter name="gui_cascade_counter9" value="false" />
<parameter name="gui_cascade_outclk_index" value="5" />
<parameter name="gui_channel_spacing" value="0.0" />
<parameter name="gui_clk_bad" value="false" />
<parameter name="gui_device_speed_grade" value="2" />
<parameter name="gui_divide_factor_c0" value="1" />
<parameter name="gui_divide_factor_c1" value="1" />
<parameter name="gui_divide_factor_c10" value="1" />
<parameter name="gui_divide_factor_c11" value="1" />
<parameter name="gui_divide_factor_c12" value="1" />
<parameter name="gui_divide_factor_c13" value="1" />
<parameter name="gui_divide_factor_c14" value="1" />
<parameter name="gui_divide_factor_c15" value="1" />
<parameter name="gui_divide_factor_c16" value="1" />
<parameter name="gui_divide_factor_c17" value="1" />
<parameter name="gui_divide_factor_c2" value="1" />
<parameter name="gui_divide_factor_c3" value="1" />
<parameter name="gui_divide_factor_c4" value="1" />
<parameter name="gui_divide_factor_c5" value="1" />
<parameter name="gui_divide_factor_c6" value="1" />
<parameter name="gui_divide_factor_c7" value="1" />
<parameter name="gui_divide_factor_c8" value="1" />
<parameter name="gui_divide_factor_c9" value="1" />
<parameter name="gui_divide_factor_n" value="1" />
<parameter name="gui_dps_cntr" value="C0" />
<parameter name="gui_dps_dir" value="Positive" />
<parameter name="gui_dps_num" value="1" />
<parameter name="gui_dsm_out_sel" value="1st_order" />
<parameter name="gui_duty_cycle0" value="50" />
<parameter name="gui_duty_cycle1" value="50" />
<parameter name="gui_duty_cycle10" value="50" />
<parameter name="gui_duty_cycle11" value="50" />
<parameter name="gui_duty_cycle12" value="50" />
<parameter name="gui_duty_cycle13" value="50" />
<parameter name="gui_duty_cycle14" value="50" />
<parameter name="gui_duty_cycle15" value="50" />
<parameter name="gui_duty_cycle16" value="50" />
<parameter name="gui_duty_cycle17" value="50" />
<parameter name="gui_duty_cycle2" value="50" />
<parameter name="gui_duty_cycle3" value="50" />
<parameter name="gui_duty_cycle4" value="50" />
<parameter name="gui_duty_cycle5" value="50" />
<parameter name="gui_duty_cycle6" value="50" />
<parameter name="gui_duty_cycle7" value="50" />
<parameter name="gui_duty_cycle8" value="50" />
<parameter name="gui_duty_cycle9" value="50" />
<parameter name="gui_en_adv_params" value="false" />
<parameter name="gui_en_dps_ports" value="false" />
<parameter name="gui_en_phout_ports" value="false" />
<parameter name="gui_en_reconf" value="false" />
<parameter name="gui_enable_cascade_in" value="false" />
<parameter name="gui_enable_cascade_out" value="false" />
<parameter name="gui_enable_mif_dps" value="false" />
<parameter name="gui_feedback_clock" value="Global Clock" />
<parameter name="gui_frac_multiply_factor" value="1" />
<parameter name="gui_fractional_cout" value="32" />
<parameter name="gui_mif_generate" value="false" />
<parameter name="gui_multiply_factor" value="1" />
<parameter name="gui_number_of_clocks" value="1" />
<parameter name="gui_operation_mode" value="normal" />
<parameter name="gui_output_clock_frequency0" value="138.0" />
<parameter name="gui_output_clock_frequency1" value="138.0" />
<parameter name="gui_output_clock_frequency10" value="100.0" />
<parameter name="gui_output_clock_frequency11" value="100.0" />
<parameter name="gui_output_clock_frequency12" value="100.0" />
<parameter name="gui_output_clock_frequency13" value="100.0" />
<parameter name="gui_output_clock_frequency14" value="100.0" />
<parameter name="gui_output_clock_frequency15" value="100.0" />
<parameter name="gui_output_clock_frequency16" value="100.0" />
<parameter name="gui_output_clock_frequency17" value="100.0" />
<parameter name="gui_output_clock_frequency2" value="100.0" />
<parameter name="gui_output_clock_frequency3" value="100.0" />
<parameter name="gui_output_clock_frequency4" value="100.0" />
<parameter name="gui_output_clock_frequency5" value="100.0" />
<parameter name="gui_output_clock_frequency6" value="100.0" />
<parameter name="gui_output_clock_frequency7" value="100.0" />
<parameter name="gui_output_clock_frequency8" value="100.0" />
<parameter name="gui_output_clock_frequency9" value="100.0" />
<parameter name="gui_phase_shift0" value="0" />
<parameter name="gui_phase_shift1" value="0" />
<parameter name="gui_phase_shift10" value="0" />
<parameter name="gui_phase_shift11" value="0" />
<parameter name="gui_phase_shift12" value="0" />
```

```
  <parameter name="gui_phase_shift13" value="0" />
  <parameter name="gui_phase_shift14" value="0" />
  <parameter name="gui_phase_shift15" value="0" />
  <parameter name="gui_phase_shift16" value="0" />
  <parameter name="gui_phase_shift17" value="0" />
  <parameter name="gui_phase_shift2" value="0" />
  <parameter name="gui_phase_shift3" value="0" />
  <parameter name="gui_phase_shift4" value="0" />
  <parameter name="gui_phase_shift5" value="0" />
  <parameter name="gui_phase_shift6" value="0" />
  <parameter name="gui_phase_shift7" value="0" />
  <parameter name="gui_phase_shift8" value="0" />
  <parameter name="gui_phase_shift9" value="0" />
  <parameter name="gui_phase_shift_deg0" value="0.0" />
  <parameter name="gui_phase_shift_deg1" value="0.0" />
  <parameter name="gui_phase_shift_deg10" value="0.0" />
  <parameter name="gui_phase_shift_deg11" value="0.0" />
  <parameter name="gui_phase_shift_deg12" value="0.0" />
  <parameter name="gui_phase_shift_deg13" value="0.0" />
  <parameter name="gui_phase_shift_deg14" value="0.0" />
  <parameter name="gui_phase_shift_deg15" value="0.0" />
  <parameter name="gui_phase_shift_deg16" value="0.0" />
  <parameter name="gui_phase_shift_deg17" value="0.0" />
  <parameter name="gui_phase_shift_deg2" value="0.0" />
  <parameter name="gui_phase_shift_deg3" value="0.0" />
  <parameter name="gui_phase_shift_deg4" value="0.0" />
  <parameter name="gui_phase_shift_deg5" value="0.0" />
  <parameter name="gui_phase_shift_deg6" value="0.0" />
  <parameter name="gui_phase_shift_deg7" value="0.0" />
  <parameter name="gui_phase_shift_deg8" value="0.0" />
  <parameter name="gui_phase_shift_deg9" value="0.0" />
  <parameter name="gui_phout_division" value="1" />
  <parameter name="gui_pll_auto_reset" value="Off" />
  <parameter name="gui_pll_bandwidth_preset" value="Auto" />
    <parameter  name="gui_pll_cascading_mode">Create  an  adjpllin  signal  to  connect  with  an  upstream
PLL</parameter>
  <parameter name="gui_pll_mode" value="Fractional-N PLL" />
  <parameter name="gui_ps_units0" value="ps" />
  <parameter name="gui_ps_units1" value="ps" />
  <parameter name="gui_ps_units10" value="ps" />
  <parameter name="gui_ps_units11" value="ps" />
  <parameter name="gui_ps_units12" value="ps" />
  <parameter name="gui_ps_units13" value="ps" />
  <parameter name="gui_ps_units14" value="ps" />
  <parameter name="gui_ps_units15" value="ps" />
  <parameter name="gui_ps_units16" value="ps" />
  <parameter name="gui_ps_units17" value="ps" />
  <parameter name="gui_ps_units2" value="ps" />
  <parameter name="gui_ps_units3" value="ps" />
  <parameter name="gui_ps_units4" value="ps" />
  <parameter name="gui_ps_units5" value="ps" />
  <parameter name="gui_ps_units6" value="ps" />
  <parameter name="gui_ps_units7" value="ps" />
  <parameter name="gui_ps_units8" value="ps" />
  <parameter name="gui_ps_units9" value="ps" />
  <parameter name="gui_refclki_frequency" value="100.0" />
  <parameter name="gui_refclk_switch" value="false" />
  <parameter name="gui_reference_clock_frequency" value="50.0" />
  <parameter name="gui_switchover_delay" value="0" />
  <parameter name="gui_switchover_mode">Automatic Switchover</parameter>
  <parameter name="gui_use_locked" value="true" />
 </module>
 <module
   name="seven_seg_0"
   kind="altera_avalon_pio"
   version="16.0"
   enabled="1">
  <parameter name="bitClearingEdgeCapReg" value="false" />
  <parameter name="bitModifyingOutReg" value="false" />
  <parameter name="captureEdge" value="false" />
  <parameter name="clockRate" value="50000000" />
  <parameter name="direction" value="Output" />
  <parameter name="edgeType" value="RISING" />
  <parameter name="generateIRQ" value="false" />
  <parameter name="irqType" value="LEVEL" />
  <parameter name="resetValue" value="0" />
  <parameter name="simDoTestBenchWiring" value="false" />
  <parameter name="simDrivenValue" value="0" />
  <parameter name="width" value="7" />
 </module>
 <module
   name="seven_seg_1"
   kind="altera_avalon_pio"
```

```
    version="16.0"
    enabled="1">
   <parameter name="bitClearingEdgeCapReg" value="false" />
   <parameter name="bitModifyingOutReg" value="false" />
   <parameter name="captureEdge" value="false" />
   <parameter name="clockRate" value="50000000" />
   <parameter name="direction" value="Output" />
   <parameter name="edgeType" value="RISING" />
   <parameter name="generateIRQ" value="false" />
   <parameter name="irqType" value="LEVEL" />
   <parameter name="resetValue" value="0" />
   <parameter name="simDoTestBenchWiring" value="false" />
   <parameter name="simDrivenValue" value="0" />
   <parameter name="width" value="7" />
  </module>
  <module
    name="seven_seg_2"
    kind="altera_avalon_pio"
    version="16.0"
    enabled="1">
   <parameter name="bitClearingEdgeCapReg" value="false" />
   <parameter name="bitModifyingOutReg" value="false" />
   <parameter name="captureEdge" value="false" />
   <parameter name="clockRate" value="50000000" />
   <parameter name="direction" value="Output" />
   <parameter name="edgeType" value="RISING" />
   <parameter name="generateIRQ" value="false" />
   <parameter name="irqType" value="LEVEL" />
   <parameter name="resetValue" value="0" />
   <parameter name="simDoTestBenchWiring" value="false" />
   <parameter name="simDrivenValue" value="0" />
   <parameter name="width" value="7" />
  </module>
  <module
    name="seven_seg_3"
    kind="altera_avalon_pio"
    version="16.0"
    enabled="1">
   <parameter name="bitClearingEdgeCapReg" value="false" />
   <parameter name="bitModifyingOutReg" value="false" />
   <parameter name="captureEdge" value="false" />
   <parameter name="clockRate" value="50000000" />
   <parameter name="direction" value="Output" />
   <parameter name="edgeType" value="RISING" />
   <parameter name="generateIRQ" value="false" />
   <parameter name="irqType" value="LEVEL" />
   <parameter name="resetValue" value="0" />
   <parameter name="simDoTestBenchWiring" value="false" />
   <parameter name="simDrivenValue" value="0" />
   <parameter name="width" value="7" />
  </module>
  <module
    name="seven_seg_4"
    kind="altera_avalon_pio"
    version="16.0"
    enabled="1">
   <parameter name="bitClearingEdgeCapReg" value="false" />
   <parameter name="bitModifyingOutReg" value="false" />
   <parameter name="captureEdge" value="false" />
   <parameter name="clockRate" value="50000000" />
   <parameter name="direction" value="Output" />
   <parameter name="edgeType" value="RISING" />
   <parameter name="generateIRQ" value="false" />
   <parameter name="irqType" value="LEVEL" />
   <parameter name="resetValue" value="0" />
   <parameter name="simDoTestBenchWiring" value="false" />
   <parameter name="simDrivenValue" value="0" />
   <parameter name="width" value="7" />
  </module>
  <module
    name="seven_seg_5"
    kind="altera_avalon_pio"
    version="16.0"
    enabled="1">
   <parameter name="bitClearingEdgeCapReg" value="false" />
   <parameter name="bitModifyingOutReg" value="false" />
   <parameter name="captureEdge" value="false" />
   <parameter name="clockRate" value="50000000" />
   <parameter name="direction" value="Output" />
   <parameter name="edgeType" value="RISING" />
   <parameter name="generateIRQ" value="false" />
   <parameter name="irqType" value="LEVEL" />
   <parameter name="resetValue" value="0" />
```

```
 <parameter name="simDoTestBenchWiring" value="false" />
 <parameter name="simDrivenValue" value="0" />
 <parameter name="width" value="7" />
</module>
<module
  name="sysid_qsys"
  kind="altera_avalon_sysid_qsys"
  version="16.0"
  enabled="1">
 <parameter name="id" value="-1395322110" />
</module>
<connection
  kind="avalon"
  version="16.0"
  start="alt_vip_vfr_vga.avalon_master"
  end="hps_0.f2h_axi_slave">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x8000" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_axi_master"
  end="oggpu_raster_unit_quad_buffer_addr_high.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x00010130" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_axi_master"
  end="oggpu_raster_unit_quad_buffer_addr_low.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x00010120" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_axi_master"
  end="oggpu_raster_unit_depth_coef_c.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x00010110" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_axi_master"
  end="oggpu_raster_unit_depth_coef_b.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x00010100" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_axi_master"
  end="oggpu_raster_unit_depth_coef_a.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x000100f0" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_axi_master"
  end="oggpu_raster_unit_tile1.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x000100e0" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_axi_master"
  end="oggpu_raster_unit_tile0.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x000100d0" />
 <parameter name="defaultConnection" value="false" />
</connection>
```

```
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_axi_master"
  end="ogpu_raster_unit_clip_rect1.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x000104c0" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_axi_master"
  end="ogpu_raster_unit_clip_rect0.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x000100b0" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_axi_master"
  end="ogpu_raster_unit_command.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x000100a0" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_axi_master"
  end="ogpu_raster_unit_status.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x00010090" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_axi_master"
  end="ogpu_raster_unit_v2z.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x00010080" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_axi_master"
  end="ogpu_raster_unit_v2y.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x00010070" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_axi_master"
  end="ogpu_raster_unit_v2x.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x00010060" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_axi_master"
  end="ogpu_raster_unit_v1z.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x00010050" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_axi_master"
  end="ogpu_raster_unit_v1y.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x00010040" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
```

```
  version="16.0"
  start="hps_0.h2f_axi_master"
  end="ogpu_raster_unit_v1x.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x00010030" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_axi_master"
  end="ogpu_raster_unit_v0z.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x00010020" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_axi_master"
  end="ogpu_raster_unit_v0y.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x00010010" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_axi_master"
  end="ogpu_raster_unit_v0x.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x00010000" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_axi_master"
  end="ogpu_quad_store_req.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x0000" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_axi_master"
  end="ogpu_quad_store_data_high.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x0010" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_axi_master"
  end="ogpu_quad_store_data_low.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x0020" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_axi_master"
  end="ogpu_quad_store_ack.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x0030" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_axi_master"
  end="ogpu_reset.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x0040" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_lw_axi_master"
```

```
  end="jtag_uart.avalon_jtag_slave">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x00020000" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_lw_axi_master"
  end="alt_vip_vfr_vga.avalon_slave">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x0100" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_lw_axi_master"
  end="sysid_qsys.control_slave">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x00010000" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_lw_axi_master"
  end="led_pio.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x00010040" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_lw_axi_master"
  end="button_pio.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x000100c0" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_lw_axi_master"
  end="seven_seg_0.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x0050" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_lw_axi_master"
  end="seven_seg_1.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x0040" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_lw_axi_master"
  end="seven_seg_2.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x0030" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_lw_axi_master"
  end="seven_seg_3.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x0020" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.h2f_lw_axi_master"
  end="seven_seg_4.s1">
 <parameter name="arbitrationPriority" value="1" />
```

```
 <parameter name="baseAddress" value="0x0010" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.s2f_lw_axi_master"
  end="dipsw_pio.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x00010080" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="hps_0.s2f_lw_axi_master"
  end="seven_seg_5.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x0000" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="master_non_sec.master"
  end="jtag_uart.avalon_jtag_slave">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x00020080" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="master_non_sec.master"
  end="alt_vip_vfr_vgs.avalon_slave">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x0100" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="master_non_sec.master"
  end="intr_capturer_0.avalon_slave_0">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x00030080" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="master_non_sec.master"
  end="sysid_qsys.control_slave">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x00010080" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="master_secure.master"
  end="hps_0.f2h_axi_slave">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x0000" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="master_non_sec.master"
  end="led_pio.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x00010040" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="avalon"
  version="16.0"
  start="master_non_sec.master"
  end="button_pio.s1">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x000100c0" />
 <parameter name="defaultConnection" value="false" />
```

```
 </connection>
 <connection
   kind="avalon"
   version="16.0"
   start="master_non_sec.master"
   end="dipsw_pio.s1">
  <parameter name="arbitrationPriority" value="1" />
  <parameter name="baseAddress" value="0x00010080" />
  <parameter name="defaultConnection" value="false" />
 </connection>
 <connection
   kind="avalon_streaming"
   version="16.0"
   start="alt_vip_vfr_vga.avalon_streaming_source"
   end="alt_vip_itc_0.din" />
 <connection kind="clock" version="16.0" start="clk_0.clk" end="master_secure.clk" />
 <connection kind="clock" version="16.0" start="clk_0.clk" end="sysid_qsys.clk" />
 <connection kind="clock" version="16.0" start="clk_0.clk" end="led_pio.clk" />
 <connection
   kind="clock"
   version="16.0"
   start="clk_0.clk"
   end="master_non_sec.clk" />
 <connection kind="clock" version="16.0" start="clk_0.clk" end="dipsw_pio.clk" />
 <connection kind="clock" version="16.0" start="clk_0.clk" end="button_pio.clk" />
 <connection kind="clock" version="16.0" start="clk_0.clk" end="jtag_uart.clk" />
 <connection
   kind="clock"
   version="16.0"
   start="clk_0.clk"
   end="ogpu_raster_unit_v2z.clk" />
 <connection
   kind="clock"
   version="16.0"
   start="clk_0.clk"
   end="ogpu_raster_unit_v2y.clk" />
 <connection
   kind="clock"
   version="16.0"
   start="clk_0.clk"
   end="ogpu_raster_unit_v2x.clk" />
 <connection
   kind="clock"
   version="16.0"
   start="clk_0.clk"
   end="ogpu_raster_unit_v1z.clk" />
 <connection
   kind="clock"
   version="16.0"
   start="clk_0.clk"
   end="ogpu_raster_unit_v1y.clk" />
 <connection
   kind="clock"
   version="16.0"
   start="clk_0.clk"
   end="ogpu_raster_unit_v1x.clk" />
 <connection
   kind="clock"
   version="16.0"
   start="clk_0.clk"
   end="ogpu_raster_unit_v0z.clk" />
 <connection
   kind="clock"
   version="16.0"
   start="clk_0.clk"
   end="ogpu_raster_unit_v0y.clk" />
 <connection
   kind="clock"
   version="16.0"
   start="clk_0.clk"
   end="ogpu_raster_unit_v0x.clk" />
 <connection
   kind="clock"
   version="16.0"
   start="clk_0.clk"
   end="ogpu_raster_unit_quad_buffer_addr_high.clk" />
 <connection
   kind="clock"
   version="16.0"
   start="clk_0.clk"
   end="ogpu_raster_unit_quad_buffer_addr_low.clk" />
 <connection
   kind="clock"
```

```
    version="16.0"
    start="clk_0.clk"
    end="ogpu_raster_unit_depth_coef_c.clk" />
  <connection
    kind="clock"
    version="16.0"
    start="clk_0.clk"
    end="ogpu_raster_unit_depth_coef_b.clk" />
  <connection
    kind="clock"
    version="16.0"
    start="clk_0.clk"
    end="ogpu_raster_unit_depth_coef_a.clk" />
  <connection
    kind="clock"
    version="16.0"
    start="clk_0.clk"
    end="ogpu_raster_unit_tile1.clk" />
  <connection
    kind="clock"
    version="16.0"
    start="clk_0.clk"
    end="ogpu_raster_unit_tile0.clk" />
  <connection
    kind="clock"
    version="16.0"
    start="clk_0.clk"
    end="ogpu_raster_unit_clip_rect1.clk" />
  <connection
    kind="clock"
    version="16.0"
    start="clk_0.clk"
    end="ogpu_raster_unit_clip_rect0.clk" />
  <connection
    kind="clock"
    version="16.0"
    start="clk_0.clk"
    end="ogpu_raster_unit_command.clk" />
  <connection
    kind="clock"
    version="16.0"
    start="clk_0.clk"
    end="ogpu_raster_unit_status.clk" />
  <connection kind="clock" version="16.0" start="clk_0.clk" end="seven_seg_5.clk" />
  <connection kind="clock" version="16.0" start="clk_0.clk" end="seven_seg_4.clk" />
  <connection kind="clock" version="16.0" start="clk_0.clk" end="seven_seg_3.clk" />
  <connection kind="clock" version="16.0" start="clk_0.clk" end="seven_seg_2.clk" />
  <connection kind="clock" version="16.0" start="clk_0.clk" end="seven_seg_1.clk" />
  <connection kind="clock" version="16.0" start="clk_0.clk" end="seven_seg_0.clk" />
  <connection
    kind="clock"
    version="16.0"
    start="clk_0.clk"
    end="ogpu_quad_store_req.clk" />
  <connection
    kind="clock"
    version="16.0"
    start="clk_0.clk"
    end="ogpu_quad_store_data_high.clk" />
  <connection
    kind="clock"
    version="16.0"
    start="clk_0.clk"
    end="ogpu_quad_store_data_low.clk" />
  <connection
    kind="clock"
    version="16.0"
    start="clk_0.clk"
    end="ogpu_quad_store_ack.clk" />
  <connection kind="clock" version="16.0" start="clk_0.clk" end="ogpu_reset.clk" />
  <connection
    kind="clock"
    version="16.0"
    start="clk_0.clk"
    end="intr_capturer_0.clock" />
  <connection
    kind="clock"
    version="16.0"
    start="clk_0.clk"
    end="alt_vip_vfr_vga.clock_master" />
  <connection
    kind="clock"
    version="16.0"
```

```
    start="clk_0.clk"
    end="hps_0.f2h_axi_clock" />
  <connection
    kind="clock"
    version="16.0"
    start="clk_0.clk"
    end="hps_0.h2f_axi_clock" />
  <connection
    kind="clock"
    version="16.0"
    start="clk_0.clk"
    end="hps_0.h2f_lw_axi_clock" />
  <connection kind="clock" version="16.0" start="clk_0.clk" end="pll_stream.refclk" />
  <connection
    kind="clock"
    version="16.0"
    start="pll_stream.outclk0"
    end="alt_vip_vfr_vga.clock_reset" />
  <connection
    kind="clock"
    version="16.0"
    start="pll_stream.outclk0"
    end="alt_vip_itc_0.is_clk_rst" />
  <connection
    kind="interrupt"
    version="16.0"
    start="hps_0.f2h_irq0"
    end="jtag_uart.irq">
   <parameter name="irqNumber" value="0" />
  </connection>
  <connection
    kind="interrupt"
    version="16.0"
    start="hps_0.f2h_irq0"
    end="button_pio.irq">
   <parameter name="irqNumber" value="1" />
  </connection>
  <connection
    kind="interrupt"
    version="16.0"
    start="hps_0.f2h_irq0"
    end="dipsw_pio.irq">
   <parameter name="irqNumber" value="2" />
  </connection>
  <connection
    kind="interrupt"
    version="16.0"
    start="intr_capturer_0.interrupt_receiver"
    end="jtag_uart.irq">
   <parameter name="irqNumber" value="0" />
  </connection>
  <connection
    kind="interrupt"
    version="16.0"
    start="intr_capturer_0.interrupt_receiver"
    end="button_pio.irq">
   <parameter name="irqNumber" value="1" />
  </connection>
  <connection
    kind="interrupt"
    version="16.0"
    start="intr_capturer_0.interrupt_receiver"
    end="dipsw_pio.irq">
   <parameter name="irqNumber" value="2" />
  </connection>
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="master_non_sec.clk_reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="master_secure.clk_reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="alt_vip_vfr_vga.clock_master_reset" />
  <connection
    kind="reset"
    version="16.0"
```

```
    start="clk_0.clk_reset"
    end="alt_vip_vfr_vga.clock_reset_reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="alt_vip_itc_0.is_clk_rst_reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="seven_seg_0.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="seven_seg_5.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="seven_seg_4.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="seven_seg_3.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="seven_seg_2.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="seven_seg_1.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="pll_stream.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="jtag_sort.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="button_pio.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="dipsw_pio.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="led_pio.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="sysid_qsys.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="ogpu_raster_uslt_clip_rect1.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="ogpu_raster_usit_clip_rect0.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="ogpu_raster_unit_quad_buffer_addr_high.reset" />
  <connection
```

```
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="ogpu_raster_unit_status.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="ogpu_raster_unit_command.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="ogpu_raster_unit_tile0.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="ogpu_raster_unit_tile1.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="ogpu_raster_unit_depth_coef_a.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="ogpu_raster_unit_depth_coef_b.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="ogpu_raster_unit_depth_coef_c.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="ogpu_raster_unit_quad_buffer_addr_low.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="ogpu_raster_unit_v0x.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="ogpu_raster_unit_v2z.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="ogpu_raster_unit_v2y.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="ogpu_raster_unit_v2x.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="ogpu_raster_unit_v1z.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="ogpu_raster_unit_v1y.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="ogpu_raster_unit_v1x.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="ogpu_raster_unit_v0z.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
```

```
    end="ogpu_raster_unit_v0y.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="ogpu_quad_store_req.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="ogpu_quad_store_data_high.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="ogpu_quad_store_data_low.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="ogpu_quad_store_ack.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="ogpu_reset.reset" />
  <connection
    kind="reset"
    version="16.0"
    start="clk_0.clk_reset"
    end="instr_capturer_0.reset_sink" />
  <interconnectRequirement for="$system" name="qsys_mm.clockCrossingAdapter" value="HANDSHAKE" />
  <interconnectRequirement for="$system" name="qsys_mm.maxAdditionalLatency" value="1" />
</system>
```

```
// ============================================================================
// Copyright (c) 2013 by Terasic Technologies Inc.
// ============================================================================
//
// Permission:
//
//   Terasic grants permission to use and modify this code for use
//   in synthesis for all Terasic Development Boards and Altera Development
//   Kits made by Terasic.  Other use of this code, including the selling
//   ,duplication, or modification of any portion is strictly prohibited.
//
// Disclaimer:
//
//   This VHDL/Verilog or C/C++ source code is intended as a design reference
//   which illustrates how these types of functions can be implemented.
//   It is the user's responsibility to verify their design for
//   consistency and functionality through the use of formal
//   verification methods.  Terasic provides no warranty regarding the use
//   or functionality of this code.
//
// ============================================================================
//
//  Terasic Technologies Inc
//  9F., No.176, Sec.2, Gongdao 5th Rd, East Dist, Hsinchu City, 30070. Taiwan
//
//
//                     web: http://www.terasic.com/
//                     email: support@terasic.com
//
// ============================================================================
//Date:  Thu Jul 11 11:26:45 2013
// ============================================================================

`define ENABLE_ADC
`define ENABLE_AUD
`define ENABLE_CLOCK2
`define ENABLE_CLOCK3
`define ENABLE_CLOCK4
`define ENABLE_CLOCK
`define ENABLE_DRAM
`define ENABLE_FAN
`define ENABLE_FPGA
`define ENABLE_GPIO
`define ENABLE_HEX
//`define ENABLE_HPS
`define ENABLE_IRDA
`define ENABLE_KEY
`define ENABLE_LEDR
`define ENABLE_PS2
`define ENABLE_SW
`define ENABLE_TD
`define ENABLE_VGA

module DE1_SOC_golden_top(

      /* Enables ADC - 3.3V */
        `ifdef ENABLE_ADC

      output            ADC_CONVST,
      output            ADC_DIN,
      input             ADC_DOUT,
      output            ADC_SCLK,

        `endif

       /* Enables AUD - 3.3V */
        `ifdef ENABLE_AUD

      input             AUD_ADCDAT,
      inout             AUD_ADCLRCK,
      inout             AUD_BCLK,
      output            AUD_DACDAT,
      inout             AUD_DACLRCK,
      output            AUD_XCK,

        `endif

      /* Enables CLOCK2  */
        `ifdef ENABLE_CLOCK2
      input             CLOCK2_50,
        `endif

      /* Enables CLOCK3 */
```

```
  `ifdef ENABLE_CLOCK3
input              CLOCK3_50,
  `endif

/* Enables CLOCK4 */
  `ifdef ENABLE_CLOCK4
input              CLOCK4_50,
  `endif

/* Enables CLOCK */
  `ifdef ENABLE_CLOCK
input              CLOCK_50,
  `endif

 /* Enables DRAM - 3.3V */
  `ifdef ENABLE_DRAM
output      [12:0] DRAM_ADDR,
output      [1:0]  DRAM_BA,
output             DRAM_CAS_N,
output             DRAM_CKE,
output             DRAM_CLK,
output             DRAM_CS_N,
inout       [15:0] DRAM_DQ,
output             DRAM_LDQM,
output             DRAM_RAS_N,
output             DRAM_UDQM,
output             DRAM_WE_N,
  `endif

/* Enables FAN - 3.3V */
  `ifdef ENABLE_FAN
output             FAN_CTRL,
  `endif

/* Enables FPGA - 3.3V */
  `ifdef ENABLE_FPGA
output             FPGA_I2C_SCLK,
inout              FPGA_I2C_SDAT,
  `endif

/* Enables GPIO - 3.3V */
  `ifdef ENABLE_GPIO
inout     [35:0]         GPIO_0,
inout     [35:0]         GPIO_1,
  `endif


/* Enables HEX - 3.3V */
  `ifdef ENABLE_HEX
output      [6:0]  HEX0,
output      [6:0]  HEX1,
output      [6:0]  HEX2,
output      [6:0]  HEX3,
output      [6:0]  HEX4,
output      [6:0]  HEX5,
  `endif

  /* Enables HPS */
  `ifdef ENABLE_HPS
inout              HPS_CONV_USB_N,
output      [14:0] HPS_DDR3_ADDR,
output      [2:0]  HPS_DDR3_BA,
output             HPS_DDR3_CAS_N,
output             HPS_DDR3_CKE,
output             HPS_DDR3_CK_N, //1.5V
output             HPS_DDR3_CK_P, //1.5V
output             HPS_DDR3_CS_N,
output      [3:0]  HPS_DDR3_DM,
inout       [31:0] HPS_DDR3_DQ,
inout       [3:0]  HPS_DDR3_DQS_N,
inout       [3:0]  HPS_DDR3_DQS_P,
output             HPS_DDR3_ODT,
output             HPS_DDR3_RAS_N,
output             HPS_DDR3_RESET_N,
input              HPS_DDR3_RZQ,
output             HPS_DDR3_WE_N,
output             HPS_ENET_GTX_CLK,
inout              HPS_ENET_INT_N,
output             HPS_ENET_MDC,
inout              HPS_ENET_MDIO,
input              HPS_ENET_RX_CLK,
input       [3:0]  HPS_ENET_RX_DATA,
input              HPS_ENET_RX_DV,
```

```
    output      [3:0]  HPS_ENET_TX_DATA,
    output             HPS_ENET_TX_EN,
    inout       [3:0]  HPS_FLASH_DATA,
    output             HPS_FLASH_DCLK,
    output             HPS_FLASH_NCSO,
    inout              HPS_GSENSOR_INT,
    inout              HPS_I2C1_SCLK,
    inout              HPS_I2C1_SDAT,
    inout              HPS_I2C2_SCLK,
    inout              HPS_I2C2_SDAT,
    inout              HPS_I2C_CONTROL,
    inout              HPS_KEY,
    inout              HPS_LED,
    inout              HPS_LTC_GPIO,
    output             HPS_SD_CLK,
    inout              HPS_SD_CMD,
    inout       [3:0]  HPS_SD_DATA,
    output             HPS_SPIM_CLK,
    input              HPS_SPIM_MISO,
    output             HPS_SPIM_MOSI,
    inout              HPS_SPIM_SS,
    input              HPS_UART_RX,
    output             HPS_UART_TX,
    input              HPS_USB_CLKOUT,
    inout       [7:0]  HPS_USB_DATA,
    input              HPS_USB_DIR,
    input              HPS_USB_NXT,
    output             HPS_USB_STP,
`endif

    /* Enables IRDA - 3.3V */
      `ifdef ENABLE_IRDA
    input              IRDA_RXD,
    output             IRDA_TXD,
      `endif

    /* Enables KEY - 3.3V */
      `ifdef ENABLE_KEY
    input       [3:0]  KEY,
      `endif

    /* Enables LEDR - 3.3V */
      `ifdef ENABLE_LEDR
    output      [9:0]  LEDR,
      `endif

    /* Enables PS2 - 3.3V */
      `ifdef ENABLE_PS2
    inout              PS2_CLK,
    inout              PS2_CLK2,
    inout              PS2_DAT,
    inout              PS2_DAT2,
      `endif

    /* Enables SW - 3.3V */
      `ifdef ENABLE_SW
    input       [9:0]  SW,
      `endif

    /* Enables TD - 3.3V */
      `ifdef ENABLE_TD
    input              TD_CLK27,
    input       [7:0]  TD_DATA,
    input              TD_HS,
    output             TD_RESET_N,
    input              TD_VS,
      `endif

    /* Enables VGA - 3.3V */
      `ifdef ENABLE_VGA
    output      [7:0]  VGA_B,
    output             VGA_BLANK_N,
    output             VGA_CLK,
    output      [7:0]  VGA_G,
    output             VGA_HS,
    output      [7:0]  VGA_R,
    output             VGA_SYNC_N,
    output             VGA_VS
      `endif
);

//=============================================================
```

```
//  REG/WIRE declarations
//=============================================================


//=============================================================
//  Structural coding
//=============================================================


endmodule
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

package ogpu_data_record_pkg is

type ogpu_float is
record
-- TODO: implement float point convereion to fix notation
--       sig: std_logic;
--       exp: std_logic_vector(30 downto 23);
--       frac: std_logic_vector(22 downto 0);
        int: unsigned(15 downto 0); -- fixed notation for initial implementation
end record;

type ogpu_vertex is
record
        x,y,z: ogpu_float;
end record;

type ogpu_box is
record
        --x0,y0: ogpu_float;
        --x1,y1: ogpu_float;
        x0,y0: unsigned(15 downto 0);
        x1,y1: unsigned(15 downto 0);
end record;

type ogpu_tile is
record
        x0,y0: unsigned(15 downto 0);
        x1,y1: unsigned(15 downto 0);
end record;

type ogpu_edge is
record
        x0,y0: unsigned(15 downto 0);
        x1,y1: unsigned(15 downto 0);
end record;

type ogpu_quad is
record
        x0,y0,x1,y1: unsigned(15 downto 0);
        x2,y2,x3,y3: unsigned(15 downto 0);
end record;

type ogpu_depth_coefficients is
record
        a,b,c: signed(31 downto 0);
end record;

type ogpu_depth_quad is array(0 to 3) of signed(31 downto 0);

type ogpu_setup_in_type is
record
        vx0: ogpu_float;
        vy0: ogpu_float;
        vx1: ogpu_float;
        vy1: ogpu_float;
        vx2: ogpu_float;
        vy2: ogpu_float;
        start_raster: std_logic;
end record;

type ogpu_setup_out_type is
record
        setup_done: std_logic;
        e0: ogpu_edge;
        e1: ogpu_edge;
        e2: ogpu_edge;
end record;

type ogpu_quad_generator_in_type is
record
        clip_rect: ogpu_box;
        tile: ogpu_tile;
        next_quad: std_logic;
end record;

type ogpu_quad_generator_out_type is
record
        end_tile: std_logic;
        quad_ready: std_logic;
```

```
        quad: ogpu_quad;
end record;

type ogpu_quad_edge_test_in_type is
record
        edge_test: std_logic;
        quad:  ogpu_quad;
        e:  ogpu_edge;
end record;

type ogpu_quad_edge_test_out_type is
record
        edge_ready:  std_logic;
        edge_mask:   std_logic_vector(0 to 3);
end record;

type ogpu_triangle_edge_test_in_type is
record
        edge_ready: std_logic_vector(0 to 2);
        edge_mask0: std_logic_vector(0 to 3);
        edge_mask1: std_logic_vector(0 to 3);
        edge_mask2: std_logic_vector(0 to 3);
end record;

type ogpu_triangle_edge_test_out_type is
record
        draw_quad: std_logic;
        discard_quad: std_logic;
        quad_mask: std_logic_vector(0 to 3);
end record;

type ogpu_depth_test_in_type is
record
        depth_coef: ogpu_depth_coefficients;
        quad: ogpu_quad;
        depth_test: std_logic;
end record;

type ogpu_depth_test_out_type is
record
        depth_ready: std_logic;
        depth_quad: ogpu_depth_quad;
end record;

type ogpu_quad_store_in_type is
record
        quad_mask: std_logic_vector(0 to 3);
        quad: ogpu_quad;
        depth_quad: ogpu_depth_quad;
        start_raster: std_logic;
        store_quad: std_logic;

        buffer_ack: std_logic;
        addr: std_logic_vector(63 downto 0);
end record;

type ogpu_quad_store_out_type is
record
        quad_stored: std_logic;

        quad_buffer_length: std_logic_vector(23 downto 0);
        buffer_address: std_logic_vector(15 downto 0);
        buffer_byte_enable: std_logic_vector(7 downto 0);
        buffer_write: std_logic;
        buffer_write_data: std_logic_vector(63 downto 0);
end record;

type ogpu_command is (OGPU_CMD_NOP,OGPU_CMD_PREPARE,OGPU_CMD_RASTER,OGPU_CMD_ERROR);
function ogpu_std_logic_to_command_func (s: std_logic_vector(7 downto 0)) return ogpu_command;

type ogpu_raster_control_in_type is
record
        command: ogpu_command;
        setup_done: std_logic;
        esd_tile: std_logic;
        quad_ready: std_logic;
        depth_ready: std_logic;
        quad_stored: std_logic;
        draw_quad: std_logic;
        discard_quad: std_logic;
end record;

type ogpu_raster_control_out_type is
```

```
record
        start_raster: std_logic;
        next_quad: std_logic;
        edge_test: std_logic;
        depth_test: std_logic;
        store_quad: std_logic;
        busy: std_logic;
        done: std_logic;
end record;

end package ogpu_data_record_pkg;

package body ogpu_data_record_pkg is

function ogpu_std_logic_to_command_func (s: std_logic_vector(7 downto 0)) return ogpu_command is
        variable r : ogpu_command;
begin
        case s is
                when "00000000" => r:=OGPU_CMD_NOP;
                when "10100101" => r:=OGPU_CMD_PREPARE;
                when "10101010" => r:=OGPU_CMD_RASTER;
                when others => r:=OGPU_CMD_ERROR;
        end case;
        return r;
end function;

end package body;
```

```
library ieee;
use ieee.std_logic_1164.all;
use work.ogpu_data_record_pkg.all;
use ieee.numeric_std.all;

--TODO: implement depth_test algorithm used in software model

entity ogpu_depth_test is
        port(clock: in std_logic;
                 reset: in std_logic;
                 d: in ogpu_depth_test_in_type;
                 q: out ogpu_depth_test_out_type);
end ogpu_depth_test;

architecture depth_test_1 of ogpu_depth_test is
        type reg_type is record
                depth_test: std_logic;
                depth_ready: std_logic;
                depth_quad: ogpu_depth_quad;
        end record;
        signal r,rin     :       reg_type;
begin
comb: process(reset,d,r)
        variable v : reg_type;
        begin
                v := r; --default assignment

                v.depth_test := d.depth_test;
                v.depth_ready := d.depth_test;
                v.depth_quad := (others => to_signed(0,32));

                if reset = '1' then
                        v.depth_ready := '0';
                end if;

                rin <= v;                -- drive register inputs

                q.depth_ready <= r.depth_ready;  -- drive module outputs
                q.depth_quad <= v.depth_quad;

        end process;

seq: process(clock)
        begin
                if rising_edge(clock) then r <= rin; end if;
        end process;
end depth_test_1;
```

```
library ieee;
use ieee.std_logic_1164.all;
use work.ogpu_data_record_pkg.all;

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity ogpu_quad_edge_test is
        port(clock: in std_logic;
                reset: in std_logic;
                d: in ogpu_quad_edge_test_in_type;
                q: out ogpu_quad_edge_test_out_type));
end ogpu_quad_edge_test;

architecture edge_test_1 of ogpu_quad_edge_test is
        function ogpu_edge_test_func (e: ogpu_edge;
                                                                        x:   unsigned(15
downto 0);
                                                                        y:   unsigned(15
downto 0))
                                                                return std_logic is
--              variable xt,yt,x0t,y0t,x1t,y1t,r : signed(31 downto 0);
                variable xt,yt,x0t,y0t,x1t,y1t,i : integer;
                variable r : signed(31 downto 0);
        begin
--              xt := to_signed(to_integer(x),32);
--              yt := to_signed(to_integer(y),32);
--              x0t := to_signed(to_integer(e.x0),32);
--              y0t := to_signed(to_integer(e.y0),32);
--              x1t := to_signed(to_integer(e.x1),32);
--              y1t := to_signed(to_integer(e.y1),32);
                xt := to_integer(x);
                yt := to_integer(y);
                x0t := to_integer(e.x0);
                y0t := to_integer(e.y0);
                x1t := to_integer(e.x1);
                y1t := to_integer(e.y1);
                i := ((xt-x0t)*(y1t-y0t) - (x1t-x0t)*(yt-y0t));
                r := to_signed(i,32);
                if r >= 0 then
                        return '1';
                else
                        return '0';
                end if;
        end ogpu_edge_test_func;

        type req_type is record
                edge_test: std_logic;
                edge_ready:  std_logic;
                edge_mask:   std_logic_vector(0 to 3));
        end record;
        signal r,rin     :        req_type;
begin
comb: process(reset,d,r)
        variable v : req_type;
        begin
                v := r; --default assignment
                v.edge_test := d.edge_test;
                v.edge_ready := r.edge_test;

                v.edge_mask(0) := ogpu_edge_test_func(d.e,d.quad.x0,d.quad.y0);
                v.edge_mask(1) := ogpu_edge_test_func(d.e,d.quad.x1,d.quad.y1);
                v.edge_mask(2) := ogpu_edge_test_func(d.e,d.quad.x2,d.quad.y2);
                v.edge_mask(3) := ogpu_edge_test_func(d.e,d.quad.x3,d.quad.y3);


                if reset = '1' then
                        v.edge_ready := '0';
                        v.edge_mask := (others => '0');
                end if;

                rin <= v;                       -- drive register inputs

                q.edge_ready <= r.edge_ready;  -- drive module outputs
                q.edge_mask <= v.edge_mask;

        end process;

seq: process(clock)
        begin
                if rising_edge(clock) then r <= rin; end if;
```

```
        end process;
end edge_test_1;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.ogpu_data_record_pkg.all;

entity ogpu_edge_test_testbench is
end ogpu_edge_test_testbench;

architecture ogpu_edge_test_tbl of ogpu_edge_test_testbench is
        constant clock_period : time := 20 ns;
        signal clock,reset        : std_logic :='0';
        --
        signal edge_test: std_logic :='0';
        signal edge_ready:  std_logic :='0';
        signal edge_mask:   std_logic_vector(0 to 3);
        -- DATAPATH/CONTROL interface signals
        signal quad:  ogpu_quad;
        signal e:  ogpu_edge;
begin
ET1:  entity work.ogpu_quad_edge_test(edge_test_1) port map(
                                                                -- IN
                                                                clock=>clock,reset=>reset,
                                                                d.edge_test=>edge_test,d.e=>e,
                                                                d.quad=>quad,
                                                                -- OUT
                                                                q.edge_ready=>edge_ready,
                                                                q.edge_mask=>edge_mask);

reset_proc: process
        begin
                --reset <= '0';
                --wait for 5*clock_period;
                reset <= '1';
                wait for 5*clock_period;
                reset <= '0';
                wait;
        end process;

stim_proc: process
        begin
                edge_test <= '0';
                e.x0<=to_unsigned(1,16);
                e.y0<=to_unsigned(1,16);
                e.x1<=to_unsigned(100,16);
                e.y1<=to_unsigned(100,16);

                -- quad at (1,20): at left, under edge
                quad.x0<=to_unsigned(1,16);                 quad.x1<=to_unsigned(2,16);
                quad.y0<=to_unsigned(20,16);                quad.y1<=to_unsigned(20,16);

                quad.x2<=to_unsigned(1,16);                 quad.x3<=to_unsigned(2,16);
                quad.y2<=to_unsigned(21,16);                quad.y3<=to_unsigned(21,16);
                wait for 6*clock_period;
                edge_test <= '1';
                wait for 2*clock_period;
                edge_test <= '0';
                wait for 2*clock_period;

                -- other quad (30,30): exactly over edge
                quad.x0<=to_unsigned(30,16);                quad.x1<=to_unsigned(31,16);
                quad.y0<=to_unsigned(30,16);                quad.y1<=to_unsigned(30,16);

                quad.x2<=to_unsigned(30,16);                quad.x3<=to_unsigned(31,16);
                quad.y2<=to_unsigned(31,16);                quad.y3<=to_unsigned(31,16);
                wait for 6*clock_period;
                edge_test <= '1';
                wait for 2*clock_period;
                edge_test <= '0';
                wait for 2*clock_period;

                -- other quad (60,15): at right, over edge
                quad.x0<=to_unsigned(60,16);                quad.x1<=to_unsigned(61,16);
                quad.y0<=to_unsigned(15,16);                quad.y1<=to_unsigned(15,16);

                quad.x2<=to_unsigned(60,16);                quad.x3<=to_unsigned(61,16);
                quad.y2<=to_unsigned(16,16);                quad.y3<=to_unsigned(16,16);
                wait for 6*clock_period;
                edge_test <= '1';
                wait for 2*clock_period;
                edge_test <= '0';
                wait for 2*clock_period;
```

```
                -- The same, but with another edge
                edge_test <= '0';
                e.x0<=to_unsigned(20,16);
                e.y0<=to_unsigned(10,16);
                e.x1<=to_unsigned(5,16);
                e.y1<=to_unsigned(40,16);


                -- quad at (2,3): at left, over edge
                quad.x0<=to_unsigned(2,16);                quad.x1<=to_unsigned(3,16);
                quad.y0<=to_unsigned(3,16);                quad.y1<=to_unsigned(3,16);

                quad.x2<=to_unsigned(2,16);                quad.x3<=to_unsigned(3,16);
                quad.y2<=to_unsigned(4,16);                quad.y3<=to_unsigned(4,16);
                wait for 6*clock_period;
                edge_test <= '1';
                wait for 2*clock_period;
                edge_test <= '0';
                wait for 2*clock_period;

                -- other quad (60,90): at right, under edge
                quad.x0<=to_unsigned(60,16);               quad.x1<=to_unsigned(61,16);
                quad.y0<=to_unsigned(90,16);               quad.y1<=to_unsigned(90,16);

                quad.x2<=to_unsigned(60,16);               quad.x3<=to_unsigned(61,16);
                quad.y2<=to_unsigned(91,16);               quad.y3<=to_unsigned(91,16);
                wait for 6*clock_period;
                edge_test <= '1';
                wait for 2*clock_period;
                edge_test <= '0';
                wait for 2*clock_period;
        wait;
        end process;

clock_process: process
        begin
                clock <= not clock;
                wait for clock_period/2;
        end process;
end ogpu_edge_test_tb1;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.ogpu_data_record_pkg.all;

--TODO: improve this algorithm, optmize for speed

entity ogpu_quad_generator is
        port(clock: in std_logic;
                reset: in std_logic;
                d: in ogpu_quad_generator_in_type;
                q: out ogpu_quad_generator_out_type);
end ogpu_quad_generator;

architecture generator_1 of ogpu_quad_generator is
        type reg_type is record
                next_quad: std_logic;
                end_tile: std_logic;
                quad_ready: std_logic;
                quad: ogpu_quad;
                generate_quads: std_logic;
                i,j,x0,x1,y1: unsigned(15 downto 0);
        end record;
        signal r,rin     :       reg_type;
begin
comb: process(reset,d,r)
        variable v : reg_type;
        begin
                v := r; --default assignment
                v.next_quad := d.next_quad;
                v.quad_ready := '0';

                if d.next_quad = '1' and r.next_quad = '0' then
                        v.end_tile := '0';
                elsif r.next_quad='1' then
                        if r.generate_quads='0' and r.end_tile = '0' then
                                v.end_tile := '0';
                                v.i:=(others => '0');
                                v.j:=(others => '0');
                                v.x0:=(others => '0');
                                v.x1:=(others => '0');
                                v.y1:=(others => '0');
                                if (d.tile.x1 < d.clip_rect.x0) or (d.tile.x0 > d.clip_rect.x1) then
                                        v.quad_ready := '0';
                                        v.end_tile := '1';
                                else
                                        if (d.tile.y1 < d.clip_rect.y0) or (d.tile.y0 > d.clip_rect.y1)
then
                                                v.quad_ready := '0';
                                                v.end_tile := '1';
                                        else
                                                -- tile is valid, so, generate quads
                                                v.generate_quads := '1';
                                                -- clip tile, discarding void areas
                                                if d.tile.y0 <= d.clip_rect.y0 then
                                                        v.i := d.clip_rect.y0;
                                                else
                                                        v.i := d.tile.y0;
                                                end if;

                                                if d.tile.x0 <= d.clip_rect.x0 then
                                                        v.j := d.clip_rect.x0;
                                                else
                                                        v.j := d.tile.x0;
                                                end if;
                                                v.x0 := v.j;

                                                if d.tile.x1 >= d.clip_rect.x1 then
                                                        v.x1 := d.clip_rect.x1;
                                                else
                                                        v.x1 := d.tile.x1;
                                                end if;

                                                if d.tile.y1 >= d.clip_rect.y1 then
                                                        v.y1 := d.clip_rect.y1;
                                                else
                                                        v.y1 := d.tile.y1;
                                                end if;
                                                -- after clipping, guarantee even number for i or x0
                                                -- this is for avoid quads starting with odd position
                                                v.i(0) := '0';
                                                v.x0(0) := '0';
                                        end if;
```

```
                        end if;
                elsif r.generate_quads='1' and r.end_tile = '0' then
                        v.quad.x0 := v.j;          v.quad.x1 := v.j+1;
                        v.quad.y0 := v.i;          v.quad.y1 := v.i;
                        v.quad.x2 := v.j;          v.quad.x3 := v.j+1;
                        v.quad.y2 := v.i+1;        v.quad.y3 := v.i+1;

                        v.j := v.j+2;
                        if v.j>v.x1 then
                                v.j := v.x0;
                                v.i := v.i+2;
                                if v.i>v.y1 then
                                        v.end_tile := '1';
                                        v.generate_quads := '0';
                                end if;
                        end if;
                        v.quad_ready := '1';
                end if;
        else
                v.quad_ready := '0';
        end if;

        if reset = '1' then
                v.quad_ready := '0';
                v.end_tile := '0';
                v.generate_quads := '0';
                v.i:=(others => '0');
                v.j:=(others => '0');
                v.x0:=(others => '0');
                v.x1:=(others => '0');
                v.y1:=(others => '0');
        end if;

        rin <= v;                -- drive register inputs

        --q.quad_ready <= v.quad_ready;  -- drive module outputs
        q.quad_ready <= r.quad_ready;  -- drive module outputs
        q.end_tile <= v.end_tile;
        q.quad <= v.quad;

    end process;

seq: process(clock)
    begin
        if rising_edge(clock) then r <= rin; end if;
    end process;
end generator_1;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.ogpu_data_record_pkg.all;

entity ogpu_quad_generator_testbench is
end ogpu_quad_generator_testbench;

architecture ogpu_quad_generator_tbl of ogpu_quad_generator_testbench is
        constant clock_period : time := 20 ns;
        signal clock,reset        : std_logic :='0';
        --
        signal quad                         :
        :=(x0=>(others=>'0'),y0=>(others=>'0'),x1=>(others=>'0'),y1=>(others=>'0'),x2=>(others=>'0'),y2=>(o                                    ogpu_quad
thers=>'0'),x3=>(others=>'0'),y3=>(others=>'0'));
        signal tile                         :
        :=(x0=>(others=>'0'),y0=>(others=>'0'),x1=>(others=>'0'),y1=>(others=>'0'));                                                          ogpu_tile
        signal clip_rect :
        :=(x0=>(others=>'0'),y0=>(others=>'0'),x1=>(others=>'0'),y1=>(others=>'0'));                                                           ogpu_box
        -- DATAPATH/CONTROL interface signals
        signal next_quad:                   std_logic:='0';
        signal quad_ready:                  std_logic:='0';
        signal end_tile:                    std_logic:='0';
begin
QG1:  entity work.ogpu_quad_generator(generator_1) port map(
                                                                        -- IN
                                                                        clock=>clock,reset=>reset,
        d.clip_rect=>clip_rect,d.tile=>tile,d.next_quad=>next_quad,
                                                                        -- OUT
        q.end_tile=>end_tile,q.quad_ready=>quad_ready,
                                                                        q.quad=>quad);

reset_proc: process
        begin
                --reset <= '0';
                --wait for 5*clock_period;
                reset <= '1';
                wait for 5*clock_period;
                reset <= '0';
                wait;
        end process;

init_proc: process
        begin
                --wait for 5*clock_period;
                tile.x0 <= to_unsigned(0,16);                   tile.y0 <= to_unsigned(0,16);
                tile.x1 <= to_unsigned(62,16);    tile.y1 <= to_unsigned(62,16);
                clip_rect.x0 <= to_unsigned(0,16);              clip_rect.y0 <= to_unsigned(0,16);
                clip_rect.x1 <= to_unsigned(128,16);                    clip_rect.y1 <= to_unsigned(128,16);
                wait;
        end process;

stim_proc: process
        begin
                next_quad <= '1';
                wait until quad_ready = '1';
                next_quad <= '0';
                wait for clock_period;
        end process;

clock_process: process
        begin
                clock <= not clock;
                wait for clock_period/2;
        end process;
end ogpu_quad_generator_tbl;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.ogpu_data_record_pkg.all;

entity ogpu_quad_store is
        port(clock: in std_logic;
                reset: in std_logic;
                d: in ogpu_quad_store_in_type;
                q: out ogpu_quad_store_out_type);
end ogpu_quad_store;

architecture quad_store_1 of ogpu_quad_store is
        --first implementation: without passing depth_quad
        -- packet size: 64 bits
        -- 63: X(16) (Big endian) | 47: Y(16) (Big endian) | 31: quad_mask(4) | 27-0: reserved(28)
        type reg_type is record
                start_raster: std_logic;
                store_quad: std_logic;
                buffer_ack: std_logic;
                buffer_ready: std_logic;

                qead_stored: std_logic;
                qaad_buffer_length: unsigned(23 downto 0);
                buffer_address: std_logic_vector(15 downto 0);
                buffer_byte_enable: std_logic_vector(7 downto 0);
                buffer_write: std_logic;
                buffer_write_data: std_logic_vector(63 downto 0);

                buffer_pointer: unsigned(15 downto 0);
        end record;
        signal r,rin     :        reg_type;
begin
comb: process(reset,d,r)
        varisble v : reg_type;
        begin
                v := r; --default assignment
                v.start_raster := d.start_raster;
                v.store_quad := d.store_quad;
                v.buffer_ack := d.buffer_ack;

                if d.buffer_ack = '1' and r.buffer_ack = '0' then
                        v.buffer_write := '0';
                end if;

                if d.buffer_ack = '0' and r.buffer_ack = '1' then
                        v.quad_stored := '1';
                end if;

                if r.quad_stored = '1' then
                        v.quad_stored := '0';
                end if;

                if d.store_quad = '0' and r.store_quad = '1' then
                        v.buffer_ready := '1';
                end if;

                if r.store_quad = '1' and r.start_raster = '1' and r.buffer_ready = '1' then
                        v.buffer_ready := '0';
                        v.buffer_address  :=  std_logic_vector(r.buffer_pointer);  --  get  anterior  to
improve speed without calculating +1 in same clock
                        v.buffer_write_data(63 downto 48) := std_logic_vector(d.quad.x0);
                        v.buffer_write_data(47 downto 32) := std_logic_vector(d.quad.y0);
                        v.buffer_write_data(27 downto 4) := (others => '0'); -- reserved = 0
                        v.buffer_write_data(3) := d.quad_mask(3);
                        v.buffer_write_data(2) := d.quad_mask(2);
                        v.buffer_write_data(1) := d.quad_mask(1);
                        v.buffer_write_data(0) := d.quad_mask(0);
                        v.buffer_write := '1';
                        v.buffer_byte_enable := "11111111";

                        v.quad_buffer_length:=r.quad_buffer_length+1; -- TODO restrict to buffer's limit
size
                        v.buffer_pointer:=r.buffer_pointer+8; -- 8 bytes for each memory word
                end if;

                if r.start_raster = '0' then
                        v.buffer_pointer := (others => '0');
                        v.buffer_address := (others => '0');  -- drive module outputs
                        v.buffer_byte_enable := (others => '0');
                        v.buffer_write := '0';
                        v.quad_buffer_length := (others => '0');
                        v.quad_stored := '0';
```

```
                        v.buffer_ready := '1';
                end if;

                if reset = '1' then
                        v.buffer_pointer := (others => '0');
                        v.buffer_address := (others => '0');  -- drive module outputs
                        v.buffer_byte_enable := (others => '0');
                        v.buffer_write := '0';
                        v.buffer_write_data := (others => '0');
                        v.quad_buffer_length := (others => '0');
                        v.start_raster := '0';
                        v.store_quad := '0';
                        v.quad_stored := '0';
                        v.buffer_ack := '0';
                        v.buffer_ready := '1';
                end if;

                rin <= v;                -- drive register inputs
                q.buffer_address <= v.buffer_address;  -- drive module outputs
                q.buffer_byte_enable <= v.buffer_byte_enable;
                q.buffer_write <= r.buffer_write;
                q.buffer_write_data <= v.buffer_write_data;
                q.quad_buffer_length <= std_logic_vector(v.quad_buffer_length);
                q.quad_stored <= r.quad_stored;

        end process;

seq: process(clock)
        begin
                if rising_edge(clock) then r <= rin; end if;
        end process;
end quad_store_1;

architecture quad_store_2 of cgpu_quad_store is
        --first implementation: without passing depth_quad
        -- packet size: 64 bits
        -- 63: X(16) (Big endian) | 47: Y(16) (Big endian) | 31: quad_mask(4) | 27-0: reserved(28)
        type reg_type is record
                start_raster: std_logic;
                store_quad: std_logic;
                buffer_ack: std_logic;
                buffer_ready: std_logic;

                quad_stored: std_logic;
                quad_buffer_length: unsigned(23 downto 0);
                buffer_address: std_logic_vector(15 downto 0);
                buffer_byte_enable: std_logic_vector(7 downto 0);
                buffer_write: std_logic;
                buffer_write_data: std_logic_vector(63 downto 0);

                buffer_pointer: unsigned(15 downto 0);
        end record;
        signal r,rin     :       reg_type;
begin
comb: process(reset,d,r)
        variable v : reg_type;
        begin
                v := r; --default assignment
                v.start_raster := d.start_raster;
                v.store_quad := d.store_quad;
                v.buffer_ack := d.buffer_ack;

                if reset = '1' then
                        v.buffer_pointer := (others => '0');
                        v.buffer_address := (others => '0');  -- drive module outputs
                        v.buffer_byte_enable := (others => '0');
                        v.buffer_write := '0';
                        v.buffer_write_data := (others => '0');
                        v.quad_buffer_length := (others => '0');
                        v.start_raster := '0';
                        v.store_quad := '0';
                        v.quad_stored := '0';
                        v.buffer_ack := '0';
                        v.buffer_ready := '1';
                else
                        if r.start_raster = '1' then
                                if d.store_quad = '1' and r.store_quad = '0' and v.buffer_ready = '1'
then
                                        v.buffer_ready := '0';
                                        v.buffer_address := std_logic_vector(r.buffer_pointer); -- get
anterior to improve speed without calculating +1 in same clock
                                        v.buffer_write_data(63          downto          48)          :=
std_logic_vector(d.quad.x0);
```

```
                                        v.buffer_write_data(47          downto          32)          :=
std_logic_vector(d.quad.y0);
                                        v.buffer_write_data(31) := d.quad_mask(3);
                                        v.buffer_write_data(30) := d.quad_mask(2);
                                        v.buffer_write_data(29) := d.quad_mask(1);
                                        v.buffer_write_data(28) := d.quad_mask(0);
                                        v.buffer_write_data(27 downto 0) := (others => '1'); -- reserved
= 0
                                        v.buffer_write := '1';
                                        v.buffer_byte_enable := "11111111";

                                        v.quad_buffer_length:=r.quad_buffer_length+1;  --  TODO  restrict
to buffer's limit size
                                        v.buffer_pointer:=r.buffer_pointer+8; -- 8 bytes for each memory
word

                                        --v.quad_stored  :=  '1';  --  remove  comment  to  bypass  external
memory control(buffer_ack signal)
                                elsif d.store_quad = '0' then
                                        v.quad_stored := '0';
                                        v.buffer_write := '0';
                                end if;
                        else
                                if d.start_raster = '1' then
                                        v.quad_buffer_length := (others=>'0');
                                        v.buffer_pointer := (others => '0');
                                end if;
                                v.quad_stored := '0';
                                v.buffer_write := '0';
                                v.buffer_ready := '1';
                        end if;
                end if;

                if d.buffer_ack = '1' then
                        v.buffer_ready := '1';
                        v.buffer_write := '0';
                        v.quad_stored := '1';
                        v.buffer_byte_enable := "00000000";
                end if;

                rin <= v;                       -- drive register inputs
                q.buffer_address <= v.buffer_address;  -- drive module outputs
                q.buffer_byte_enable <= v.buffer_byte_enable;
                q.buffer_write <= r.buffer_write;
                q.buffer_write_data <= v.buffer_write_data;
                q.quad_buffer_length <= std_logic_vector(v.quad_buffer_length);
                q.quad_stored <= r.quad_stored;

        end process;

reg: process(clock)
        begin
                if rising_edge(clock) then r <= rin; end if;
        end process;
end quad_store_2;

architecture test_1 of ogpu_quad_store is
        -- test implementation for writing in external buffer
        type reg_type is record
                start_raster: std_logic;
                store_quad: std_logic;
                buffer_ack: std_logic;

                quad_stored: std_logic;
                quad_buffer_length: unsigned(23 downto 0);
                buffer_address: std_logic_vector(15 downto 0);
                buffer_byte_enable: std_logic_vector(7 downto 0);
                buffer_write: std_logic;
                buffer_write_data: std_logic_vector(63 downto 0);

                buffer_pointer: unsigned(15 downto 0);
        end record;
        signal r,rin     :       reg_type;
begin
comb: process(reset,d,r)
        variable v : reg_type;
        begin
                v := r; --default assignment
                v.start_raster := d.start_raster;
                v.store_quad := d.store_quad;
                v.buffer_ack := d.buffer_ack;

                v.quad_stored := '1';
```

```
                if reset = '1' then
                        v.buffer_pointer := (others => '0');
                        v.buffer_address := (others => '0');
                        v.buffer_byte_enable := (others => '0');
                        v.buffer_write := '0';
                        v.buffer_write_data := (others => '0');
                        v.quad_buffer_length := (others => '0');
                        v.start_raster := '0';
                        v.store_quad := '0';
                        v.quad_stored := '0';
                        v.buffer_ack := '0';
                elsif d.buffer_ack = '0' then
                        v.buffer_address  :=  std_logic_vector(r.buffer_pointer);  --  get  anterior  to
improve speed without calculating +1 in same clock
                        v.buffer_write_data(63 downto 32) := (others => '1');
                        v.buffer_write_data(31 downto 16) := (others => '0');
                        v.buffer_write_data(15 downto 0) := std_logic_vector(r.buffer_pointer);
                        v.buffer_write := '1';
                        v.buffer_byte_enable := "11111111";

                        v.quad_buffer_length:=r.quad_buffer_length+1;
                        v.buffer_pointer:=v.buffer_pointer+8; -- 8 bytes for each memory word

                        v.quad_stored:='1';
                else
                        v.buffer_write := '0';
                end if;

                if d.buffer_ack = '1' then
                        v.buffer_write := '0';
                        v.quad_stored := '1';
                        v.buffer_byte_enable := "00000000";
                end if;

                rin <= v;                -- drive register inputs
                q.buffer_address <= v.buffer_address;  -- drive module outputs
                q.buffer_byte_enable <= v.buffer_byte_enable;
                q.buffer_write <= v.buffer_write;
                q.buffer_write_data <= v.buffer_write_data;
                q.quad_buffer_length <= std_logic_vector(v.quad_buffer_length);
                q.quad_stored <= r.quad_stored;

        end process;

seq: process(clock)
        begin
                if rising_edge(clock) then r <= rin; end if;
        end process;
end test_1;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.ogpu_data_record_pkg.all;

entity ogpu_quad_store_testbench is
end ogpu_quad_store_testbench;

architecture ogpu_quad_store_testbench_tbi of ogpu_quad_store_testbench is
        constant clock_period : time := 20 ns;
        signal clock,reset        : std_logic :='0';
        --
        signal quad_mask: std_logic_vector(0 to 3);
        signal quad: ogpu_quad;
        signal depth_quad: ogpu_depth_quad;

        signal buffer_ack: std_logic:='0';
        signal addr: std_logic_vector(63 downto 0);

        signal quad_buffer_length: std_logic_vector(23 downto 0);
        signal buffer_address: std_logic_vector(15 downto 0);
        signal buffer_byte_enable: std_logic_vector(7 downto 0);
        signal buffer_write: std_logic:='0';
        signal buffer_write_data: std_logic_vector(63 downto 0);

        -- DATAPATH/CONTROL interface signals
        signal start_raster: std_logic:='0';
        signal store_quad: std_logic:='0';
        signal quad_stored: std_logic:='0';

        --internal testbench signals
        signal run_proc: std_logic :='0';

begin
QS1:  entity work.ogpu_quad_store(quad_store_1) port map(
                                                                -- IN
                                                                clock=>clock,reset=>reset,
                                                                d.addr=>addr,
                                                                d.quad_mask=>quad_mask,
                                                                d.quad=>quad,
                                                                d.depth_quad=>depth_quad,
                                                                d.start_raster=>start_raster,
                                                                d.store_quad=>store_quad,
                                                                d.buffer_ack=>buffer_ack,
                                                                -- OUT
                                                                q.quad_stored=>quad_stored,

        q.quad_buffer_length=>quad_buffer_length,

        q.buffer_address=>buffer_address,

        q.buffer_byte_enable=>buffer_byte_enable,
                                                                q.buffer_write=>buffer_write,

        q.buffer_write_data=>buffer_write_data);

reset_proc: process
        begin
                --reset <= '0';
                --wait for 5*clock_period;
                reset <= '1';
                wait for 5*clock_period;
                reset <= '0';
                wait;
        end process;

ext_memory_proc: process
        begin
                wait until buffer_write = '1';
                wait until falling_edge(clock) and clock = '0';
                wait for 3*clock_period;
                buffer_ack <= '1';
                wait for clock_period;
                buffer_ack <= '0';
        end process;

init_stim_proc: process
        begin
                wait until reset = '0';
                depth_quad <= (others=>(others=>'0'));
                addr<=(others=>'0');
                run_proc<='1';
                wait;
```

```
        end process;

stim_proc: process
        variable i:integer:=0;
        variable j:integer:=0;
        begin
                if run_proc = '0' then
                        wait until run_proc = '1';
                        start_raster <= '1';
                        wait for 3*clock_period;
                end if;
                quad_mask<=std_logic_vector(to_unsigned(1,4));
                quad.x0<=to_unsigned(i,16);       quad.x1<=to_unsigned(i+1,16);
                quad.y0<=to_unsigned(j,16);       quad.y1<=to_unsigned(j,16);
                quad.x2<=to_unsigned(i,16);       quad.x3<=to_unsigned(i+1,16);
                quad.y2<=to_unsigned(j+1,16);     quad.y3<=to_unsigned(j+1,16);

                store_quad <= '1';
                wait until quad_stored = '1';
                store_quad <= '0';
                i:=i+2;
                if i>63 then
                        i:=0;
                        j:=j+2;
                        if j>63 then
                                j:=0;
                                start_raster<='0';
                                wait for 3*clock_period;
                                start_raster<='1';
                        end if;
                end if;
                wait for 2*clock_period;
        end process;

clock_process: process
        begin
                clock <= not clock;
                wait for clock_period/2;
        end process;
end ogpu_quad_store_testbench_tb1;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.ogpu_data_record_pkg.all;

entity ogpu_raster_control is
        port(clock: in std_logic;
                reset: in std_logic;
                d: in ogpu_raster_control_in_type;
                q: out ogpu_raster_control_out_type);
end ogpu_raster_control;

architecture raster_control_1 of ogpu_raster_control is
        type state_type is (
                        OGPU_RASTER_CONTROL_IDLE,
                        OGPU_RASTER_CONTROL_DONE,
                        OGPU_RASTER_CONTROL_SETUP,
                        OGPU_RASTER_CONTROL_QUAD_GEN,
                        OGPU_RASTER_CONTROL_QUAD_TEST,
                        OGPU_RASTER_CONTROL_STORE_QUAD);

        type reg_type is record
                setup_done: std_logic;
                end_tile: std_logic;
                quad_ready: std_logic;
                depth_ready: std_logic;
                quad_stored: std_logic;
                draw_quad: std_logic;
                discard_quad: std_logic;
                start_raster: std_logic;
                next_quad: std_logic;
                edge_test: std_logic;
                depth_test: std_logic;
                store_quad: std_logic;
                busy: std_logic;
                done: std_logic;

                state : state_type;
        end record;
        signal r,rin     :       reg_type;
begin
comb: process(reset,d,r)
        variable v : reg_type;
        begin
                v := r; --default assignment

                v.setup_done := d.setup_done;
                v.end_tile := d.end_tile;
                v.quad_ready := d.quad_ready;
                v.depth_ready := d.depth_ready;
                v.quad_stored := d.quad_stored;
                v.draw_quad := d.draw_quad;
                v.discard_quad := d.discard_quad;

                case r.state is
                        when OGPU_RASTER_CONTROL_DONE =>
                                if d.command = OGPU_CMD_PREPARE then
                                        v.state := OGPU_RASTER_CONTROL_IDLE;
                                        v.done := '0';
                                        v.busy := '0';
                                        v.start_raster := '0';
                                        v.next_quad := '0';
                                        v.edge_test := '0';
                                        v.depth_test := '0';
                                        v.store_quad := '0';
                                end if;

                        when OGPU_RASTER_CONTROL_SETUP =>
                                if v.setup_done = '1' then
                                        v.state := OGPU_RASTER_CONTROL_QUAD_GEN;
                                        v.next_quad := '1';
                                        v.store_quad := '0';
                                        v.edge_test := '0';
                                        v.depth_test := '0';
                                        --v.start_raster := '0';
                                end if;

                        when OGPU_RASTER_CONTROL_QUAD_GEN =>
                                if v.quad_ready = '1' then
                                        v.state := OGPU_RASTER_CONTROL_QUAD_TEST;
                                        v.next_quad := '0';
                                        v.edge_test := '1';
                                        v.depth_test := '1';
```

```
                elsif v.end_tile = '1' then
                        v.state := OGPU_RASTER_CONTROL_DONE;
                        v.done := '1';
                        v.busy := '0';
                        v.start_raster := '0';
                        v.next_quad := '0';
                        v.edge_test := '0';
                        v.depth_test := '0';
                        v.store_quad := '0';
                end if;

        when OGPU_RASTER_CONTROL_QUAD_TEST =>
                if (v.draw_quad and v.depth_ready) = '1' then
                        v.state := OGPU_RASTER_CONTROL_STORE_QUAD;
                        v.store_quad := '1';
                elsif v.end_tile = '1' then
                        v.state := OGPU_RASTER_CONTROL_DONE;
                        v.done := '1';
                        v.busy := '0';
                        v.start_raster := '0';
                        v.next_quad := '0';
                        v.edge_test := '0';
                        v.depth_test := '0';
                        v.store_quad := '0';
                elsif v.discard_quad = '1' then
                        v.state := OGPU_RASTER_CONTROL_QUAD_GEN;
                        v.next_quad := '1';
                        v.store_quad := '0';
                        v.edge_test := '0';
                        v.depth_test := '0';
                end if;

        when OGPU_RASTER_CONTROL_STORE_QUAD =>
                if v.end_tile = '1' then
                        v.state := OGPU_RASTER_CONTROL_DONE;
                        v.done := '1';
                        v.busy := '0';
                        v.start_raster := '0';
                        v.next_quad := '0';
                        v.edge_test := '0';
                        v.depth_test := '0';
                        v.store_quad := '0';
                elsif v.quad_stored = '1' then
                        v.state := OGPU_RASTER_CONTROL_QUAD_GEN;
                        v.next_quad := '1';
                        v.store_quad := '0';
                        v.edge_test := '0';
                        v.depth_test := '0';
                end if;

        when others => -- including OGPU_RASTER_CONTROL_IDLE
                if d.command = OGPU_CMD_RASTER then
                        v.state := OGPU_RASTER_CONTROL_SETUP;
                        v.busy := '1';
                        v.done := '0';
                        v.start_raster := '1';
                end if;
    end case;

    if reset = '1' then
            v.start_raster := '0';
            v.next_quad := '0';
            v.edge_test := '0';
            v.depth_test := '0';
            v.store_quad := '0';
            v.busy := '0';
            v.done := '0';

            v.state := OGPU_RASTER_CONTROL_IDLE;
    end if;

    rin <= v;                   -- drive register inputs

    -- drive module outputs
    q.start_raster <= r.start_raster;
    q.next_quad    <= r.next_quad;
    q.edge_test    <= r.edge_test;
    q.depth_test   <= r.depth_test;
    q.store_quad   <= r.store_quad;
    q.busy                  <= r.busy;
    q.done                  <= r.done;

end process;
```

```
seq: process(clock)
        begin
                if rising_edge(clock) then r <= rin; end if;
        end process;
end raster_control_1;
```

```
-- OGPU -- RASTER STRUCTURAL MODEL --

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.ogpu_data_record_pkg.all;

entity ogpu_raster_unit is
        port(clock:                                         in std_logic;
                reset:                                      in std_logic;
                command:                                    in std_logic_vector(7 downto 0);
                v0x,v0y,v0z:                                in std_logic_vector(15 downto 0);
                v1x,v1y,v1z:                                in std_logic_vector(15 downto 0);
                v2x,v2y,v2z:                                in std_logic_vector(15 downto 0);
                clip_rect0:                           in std_logic_vector(31 downto 0);
                clip_rect1:                           in std_logic_vector(31 downto 0);
                tile0:                                      in std_logic_vector(31 downto 0);
                tile1:                                      in std_logic_vector(31 downto 0);
                depth_coef_a:           in std_logic_vector(31 downto 0);
                depth_coef_b:           in std_logic_vector(31 downto 0);
                depth_coef_c:           in std_logic_vector(31 downto 0);
                quad_buffer_addr:       in std_logic_vector(63 downto 0);
                done:                                       out std_logic;
                busy:                                       out std_logic;
                quad_buffer_length: out std_logic_vector(23 downto 0);
                --external buffer interface
                ext_buffer_ack:                             in std_logic;
                ext_buffer_address:                   out std_logic_vector(15 downto 0);
                ext_buffer_byte_enable: out std_logic_vector(7 downto 0);
                ext_buffer_write:                           out std_logic;
                ext_buffer_write_data:  out std_logic_vector(63 downto 0);--);
                --debug interface
                     dbg: out std_logic_vector(11 downto 0));
end ogpu_raster_unit;

architecture structure_1 of ogpu_raster_unit is
        --
        signal int_command:                                  ogpu_command;
        signal int_v0:                                               ogpu_vertex;
        signal int_v1:                                               ogpu_vertex;
        signal int_v2:                                               ogpu_vertex;
        signal int_clip_rect:                         ogpu_box;
        signal int_tile:                                      ogpu_tile;
        signal int_depth_coef:                        ogpu_depth_coefficients;
        --
        signal e0                                                    : ogpu_edge
        :=(x0=>(others=>'0'),y0=>(others=>'0'),x1=>(others=>'0'),y1=>(others=>'0'));
        signal e1                                                    : ogpu_edge
        :=(x0=>(others=>'0'),y0=>(others=>'0'),x1=>(others=>'0'),y1=>(others=>'0'));
        signal e2                                                    : ogpu_edge
        :=(x0=>(others=>'0'),y0=>(others=>'0'),x1=>(others=>'0'),y1=>(others=>'0'));
        signal quad                                                  : ogpu_quad
        :=(x0=>(others=>'0'),y0=>(others=>'0'),x1=>(others=>'0'),y1=>(others=>'0'),x2=>(others=>'0'),y2=>(o
thers=>'0'),x3=>(others=>'0'),y3=>(others=>'0'));
        signal edge_mask0                   : std_logic_vector(0 to 3)               :=(others=>'0');
        signal edge_mask1                   : std_logic_vector(0 to 3)               :=(others=>'0');
        signal edge_mask2                   : std_logic_vector(0 to 3)               :=(others=>'0');
        signal edge_ready                   : std_logic_vector(0 to 2)               :=(others=>'0');
        signal quad_mask                    : std_logic_vector(0 to 3)               :=(others=>'0');
        signal depth_quad                   : ogpu_depth_quad                  :=(others=>(others=>'0'));
        -- DATAPATH/CONTROL interface signals
        signal start_raster:                std_logic:='0';
        signal next_quad:                   std_logic:='0';
        signal quad_ready:                  std_logic:='0';
        signal depth_test:                  std_logic:='0';
        signal edge_test:                   std_logic:='0';
        signal store_quad:                  std_logic:='0';
        signal setup_done:                                   std_logic:='0';
        signal end_tile:                                     std_logic:='0';
        signal draw_quad:                                    std_logic:='0';
        signal discard_quad:                                 std_logic:='0';
        signal depth_ready:                                  std_logic:='0';
        signal quad_stored:                                  std_logic:='0';
begin
--debug
dbg(0)<=start_raster;
dbg(1)<=next_quad;
dbg(2)<=quad_ready;
dbg(3)<=depth_test;
dbg(4)<=edge_test;
dbg(5)<=store_quad;
dbg(6)<=setup_done;
dbg(7)<=end_tile;
```

```
dbg(8)<=draw_quad;
dbg(9)<=discard_quad;
dbg(10)<=depth_ready;
dbg(11)<=quad_stored;
-- input/output packing
        int_command <= ogpu_std_logic_to_command_func(command);
        int_v0.x.int<=unsigned(v0x);                int_v0.y.int<=unsigned(v0y);
        int_v0.z.int<=unsigned(v0z);
        int_v1.x.int<=unsigned(v1x);                int_v1.y.int<=unsigned(v1y);
        int_v1.z.int<=unsigned(v1z);
        int_v2.x.int<=unsigned(v2x);                int_v2.y.int<=unsigned(v2y);
        int_v2.z.int<=unsigned(v2z);
        int_clip_rect.x0<=unsigned(clip_rect0(15 downto 0));
        int_clip_rect.y0<=unsigned(clip_rect0(31 downto 16));
        int_clip_rect.x1<=unsigned(clip_rect1(15 downto 0));
        int_clip_rect.y1<=unsigned(clip_rect1(31 downto 16));
        int_tile.x0<=unsigned(tile0(15 downto 0));
        int_tile.y0<=unsigned(tile0(31 downto 16));
        int_tile.x1<=unsigned(tile1(15 downto 0));
        int_tile.y1<=unsigned(tile1(31 downto 16));
        int_depth_coef.a<=signed(depth_coef_a);
        int_depth_coef.b<=signed(depth_coef_b);
        int_depth_coef.c<=signed(depth_coef_c);

-- CONTROL
RC1:  entity work.ogpu_raster_control(raster_control_1) port map(clock=>clock,reset=>reset,
                                                                 -- IN
                                                                 d.command=>int_command,
                                                                 d.setup_done=>setup_done,
                                                                 d.end_tile=>end_tile,
                                                                 d.quad_ready=>quad_ready,
                                                                 d.depth_ready=>depth_ready,
                                                                 d.quad_stored=>quad_stored,
                                                                 d.draw_quad=>draw_quad,
                                                                 d.discard_quad=>discard_quad,
                                                                 -- OUT
                                                                 q.start_raster=>start_raster,
                                                                 q.next_quad=>next_quad,
                                                                 q.edge_test=>edge_test,
                                                                 q.depth_test=>depth_test,
                                                                 q.store_quad=>store_quad,
                                                                 q.busy=>busy,
                                                                 q.done=>done);
-- DATAPATH
ST1:  entity work.ogpu_setup(setup_1) port map(-- IN
                                                                 clock=>clock,reset=>reset,

        d.vx0=>int_v0.x,d.vy0=>int_v0.y,

        d.vx1=>int_v1.x,d.vy1=>int_v1.y,

        d.vx2=>int_v2.x,d.vy2=>int_v2.y,
                                                                 d.start_raster=>start_raster,
                                                                 -- OUT
                                                                 q.setup_done=>setup_done,
                                                                 q.e0=>e0,q.e1=>e1,q.e2=>e2);

QG1:  entity work.ogpu_quad_generator(generator_1) port map(
                                                                 -- IN
                                                                 clock=>clock,reset=>reset,

        d.clip_rect=>int_clip_rect,d.tile=>int_tile,d.next_quad=>next_quad,
                                                                 -- OUT

        q.end_tile=>end_tile,q.quad_ready=>quad_ready,
                                                                 q.quad=>quad);

ET1:  entity work.ogpu_quad_edge_test(edge_test_1) port map(
                                                                 -- IN
                                                                 clock=>clock,reset=>reset,

        d.edge_test=>edge_test,d.e=>e0,
                                                                 d.quad=>quad,
                                                                 -- OUT
                                                                 q.edge_ready=>edge_ready(0),
                                                                 q.edge_mask=>edge_mask0);

ET2:  entity work.ogpu_quad_edge_test(edge_test_1) port map(
                                                                 -- IN
                                                                 clock=>clock,reset=>reset,

        d.edge_test=>edge_test,d.e=>e1,
                                                                 d.quad=>quad,
                                                                 -- OUT
                                                                 q.edge_ready=>edge_ready(1),
                                                                 q.edge_mask=>edge_mask1);
```

```
ET3:  entity work.ogpu_quad_edge_test(edge_test_1) port map(
                                                                    -- IN
                                                                    clock=>clock,reset=>reset,

        d.edge_test=>edge_test,d.e=>e2,
                                                                    d.quad=>quad,
                                                                    -- OUT
                                                                    q.edge_ready=>edge_ready(2),
                                                                    q.edge_mask=>edge_mask2);
TE1:  entity work.ogpu_triangle_edge_test(triangle_test_1) port map(
                                                                    -- IN
                                                                    clock=>clock,reset=>reset,
                                                                    d.edge_ready=>edge_ready,

        d.edge_mask0=>edge_mask0,d.edge_mask1=>edge_mask1,d.edge_mask2=>edge_mask2,
                                                                    -- OUT

        q.drav_quad=>drav_quad,q.discard_quad=>discard_quad,
                                                                    q.quad_mask=>quad_mask);
DT1:  entity work.ogpu_depth_test(depth_test_1) port map(
                                                                    -- IN
                                                                    clock=>clock,reset=>reset,
                                                                    d.depth_coef=>int_depth_coef,
                                                                    d.quad=>quad,
                                                                    d.depth_test=>depth_test,
                                                                    -- OUT
                                                                    q.depth_ready=>depth_ready,
                                                                    q.depth_quad=>depth_quad);
QS1:  entity work.ogpu_quad_store(quad_store_1) port map(
                                                                    -- IN
                                                                    clock=>clock,reset=>reset,
                                                                    d.addr=>quad_buffer_addr,
                                                                    d.quad_mask=>quad_mask,
                                                                    d.quad=>quad,
                                                                    d.depth_quad=>depth_quad,
                                                                    d.start_raster=>start_raster,
                                                                    d.store_quad=>store_quad,
                                                                    d.buffer_ack=>ext_buffer_ack,
                                                                    -- OUT
                                                                    q.quad_stored=>quad_stored,

        q.quad_buffer_length=>quad_buffer_length,

        q.buffer_address=>ext_buffer_address,

        q.buffer_byte_enable=>ext_buffer_byte_enable,

        q.buffer_write=>ext_buffer_write,

        q.buffer_write_data=>ext_buffer_write_data);
end structure_1;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.ogpu_data_record_pkg.all;

entity ogpu_raster_unit_testbench is
end ogpu_raster_unit_testbench;

architecture ogpu_raster_unit_testbench_tb1 of ogpu_raster_unit_testbench is
        constant clock_period : time := 20 ns;
        signal clock,reset        : std_logic :='0';
        --
        signal command:  std_logic_vector(7 downto 0):=(others =>'0');
        signal v0x,v0y,v0z:              std_logic_vector(15 downto 0);
        signal v1x,v1y,v1z:              std_logic_vector(15 downto 0);
        signal v2x,v2y,v2z:              std_logic_vector(15 downto 0);
        signal clip_rect0:                       std_logic_vector(31 downto 0);
        signal clip_rect1:                       std_logic_vector(31 downto 0);
        signal tile0:                            std_logic_vector(31 downto 0);
        signal tile1:                            std_logic_vector(31 downto 0);
        signal depth_coef_a:             std_logic_vector(31 downto 0);
        signal depth_coef_b:             std_logic_vector(31 downto 0);
        signal depth_coef_c:             std_logic_vector(31 downto 0);
        signal quad_buffer_addr: std_logic_vector(63 downto 0);
                                                 std_logic := '0';
        signal done:
        signal busy:                             std_logic := '0';
        signal quad_buffer_length:                       std_logic_vector(23 downto 0);
        --external buffer interface
        signal ext_buffer_ack:                           std_logic := '0';
        signal ext_buffer_address:                       std_logic_vector(15 downto 0);
        signal ext_buffer_byte_enable:           std_logic_vector(7 downto 0);
        signal ext_buffer_write:                         std_logic := '0';
        signal ext_buffer_write_data:            std_logic_vector(63 downto 0);
        --
        signal dbg: std_logic_vector(11 downto 0);

begin

RU1:  entity work.ogpu_raster_unit(structure_1) port map(clock=>clock,
                reset=>reset,
                command=>command,
                v0x=>v0x,v0y=>v0y,v0z=>v0z,
                v1x=>v1x,v1y=>v1y,v1z=>v1z,
                v2x=>v2x,v2y=>v2y,v2z=>v2z,
                clip_rect0=>clip_rect0,
                clip_rect1=>clip_rect1,
                tile0=>tile0,
                tile1=>tile1,
                depth_coef_a=>depth_coef_a,
                depth_coef_b=>depth_coef_b,
                depth_coef_c=>depth_coef_c,
                quad_buffer_addr=>quad_buffer_addr,
                done=>done,
                busy=>busy,
                quad_buffer_length=>quad_buffer_length,
                --external buffer interface
                ext_buffer_ack=>ext_buffer_ack,
                ext_buffer_address=>ext_buffer_address,
                ext_buffer_byte_enable=>ext_buffer_byte_enable,
                ext_buffer_write=>ext_buffer_write,
                ext_buffer_write_data=>ext_buffer_write_data,
                --debug interface
                        dbg=>dbg);

reset_proc: process
        begin
                reset <= '1';

                v0x<=std_logic_vector(to_unsigned(1,16));
                v0y<=std_logic_vector(to_unsigned(1,16));
                v0z<=std_logic_vector(to_unsigned(1,16));

                v1x<=std_logic_vector(to_unsigned(100,16));
                v1y<=std_logic_vector(to_unsigned(1,16));
                v1z<=std_logic_vector(to_unsigned(1,16));

                v2x<=std_logic_vector(to_unsigned(1,16));
                v2y<=std_logic_vector(to_unsigned(100,16));
                v2z<=std_logic_vector(to_unsigned(1,16));

                depth_coef_a<=(others => '0');
                depth_coef_b<=(others => '0');
                depth_coef_c<=(others => '0');
```

```
                quad_buffer_addr<=(others => '0');

                wait for 5*clock_period;
                reset <= '0';
                wait;
        end process;

ext_memory_proc: process
        begin
                wait until ext_buffer_write = '1';
                wait until falling_edge(clock);
                wait for 3*clock_period;
                ext_buffer_ack <= '1';
                wait for clock_period;
                ext_buffer_ack <= '0';
        end process;

stim_proc: process
        variable i:integer:=0;
        variable j:integer:=0;
        begin
                if reset = '1' then
                        wait until reset = '0';
                        wait for 1*clock_period;
                end if;

                clip_rect0(31 downto 16)<=std_logic_vector(to_unsigned(1,16)); -- x0
                clip_rect0(15 downto 0 )<=std_logic_vector(to_unsigned(1,16)); -- y0
                clip_rect1(31 downto 16)<=std_logic_vector(to_unsigned(100,16)); -- x1
                clip_rect1(15 downto 0 )<=std_logic_vector(to_unsigned(100,16)); -- y1

                tile0(31 downto 16)<=std_logic_vector(to_unsigned(0,16)); -- x0
                tile0(15 downto 0 )<=std_logic_vector(to_unsigned(0,16)); -- y0
                tile1(31 downto 16)<=std_logic_vector(to_unsigned(63,16)); -- x1
                tile1(15 downto 0 )<=std_logic_vector(to_unsigned(63,16)); -- y1

                command<=std_logic_vector(to_unsigned(16#AA#,8));

                wait until busy = '1';
                wait until done = '1';

                command<=std_logic_vector(to_unsigned(16#A5#,8));
                wait until done = '0';

                wait;

        end process;

clock_process: process
        begin
                clock <= not clock;
                wait for clock_period/2;
        end process;
end ngps_raster_unit_testbench_th1;
```

```
library ieee;
use ieee.std_logic_1164.all;
use work.ogpu_data_record_pkg.all;

entity ogpu_setup is
        port(clock: in std_logic;
                  reset: in std_logic;
                  d:     in ogpu_setup_in_type;
                  q:     out ogpu_setup_out_type);
end ogpu_setup;

architecture setup_1 of ogpu_setup is
        type reg_type is record
                start_raster: std_logic;
                setup_done: std_logic;
                e0: ogpu_edge;
                e1: ogpu_edge;
                e2: ogpu_edge;
        end record;
        signal r,rin     :       reg_type;
begin
comb: process(reset,d,r)
        variable v : reg_type;
        begin
                v := r;

                v.start_raster := d.start_raster;
                v.setup_done := '0';

                v.e0.x0 := d.vx0.int;
                v.e0.y0 := d.vy0.int;
                v.e0.x1 := d.vx1.int;
                v.e0.y1 := d.vy1.int;

                v.e1.x0 := d.vx1.int;
                v.e1.y0 := d.vy1.int;
                v.e1.x1 := d.vx2.int;
                v.e1.y1 := d.vy2.int;

                v.e2.x0 := d.vx2.int;
                v.e2.y0 := d.vy2.int;
                v.e2.x1 := d.vx0.int;
                v.e2.y1 := d.vy0.int;

                if d.start_raster = '1' then
                        v.setup_done := '1';
                end if;
                if reset='1' then v.setup_done := '0'; end if;

                rin <= v;                -- drive register inputs

                q.setup_done <= r.setup_done;  -- drive module outputs
                q.e0 <= v.e0;
                q.e1 <= v.e1;
                q.e2 <= v.e2;
        end process;

seq: process(clock)
        begin
                if rising_edge(clock) then
                        r <= rin;
                end if;
        end process;
end setup_1;
```

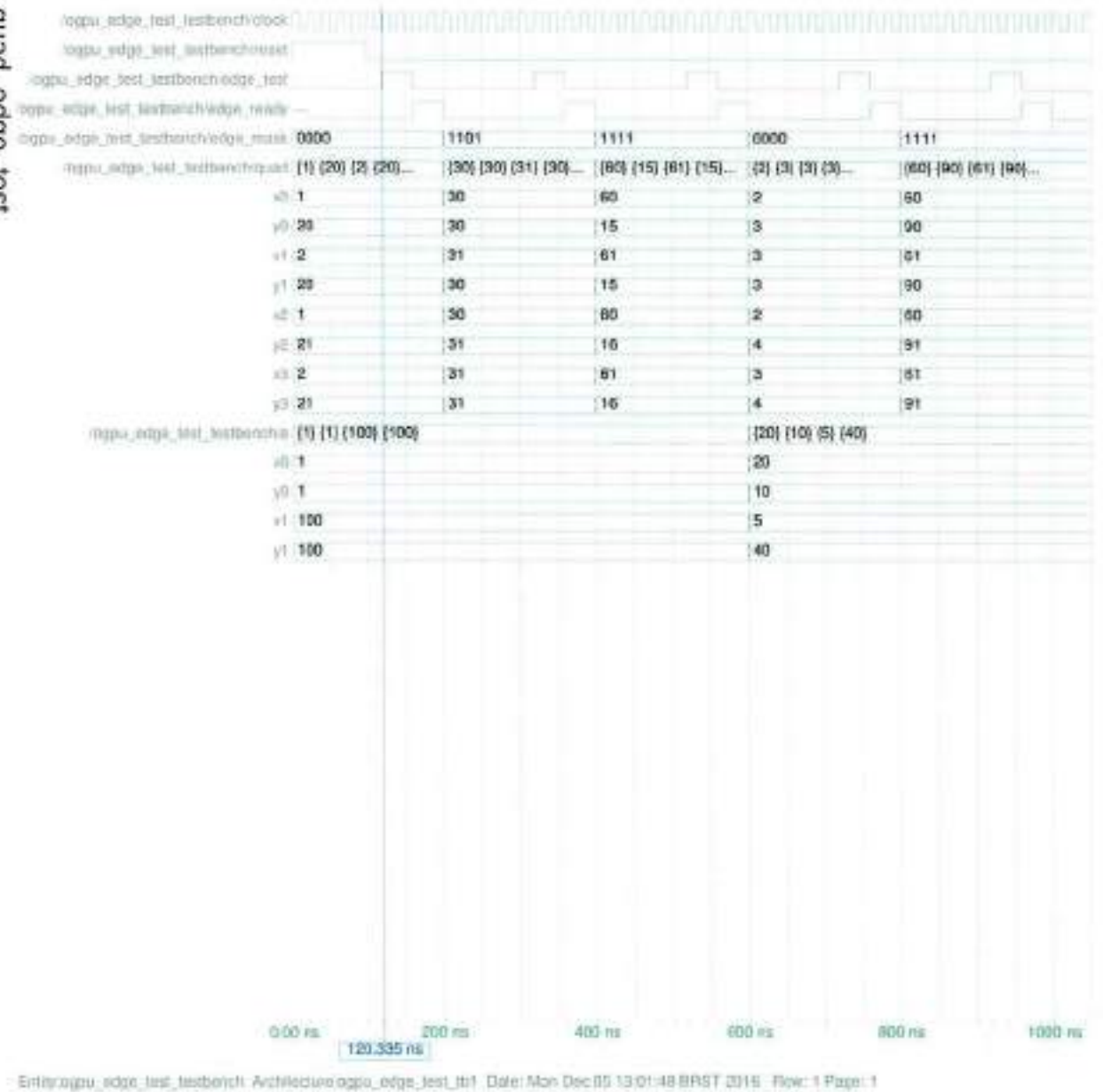
```
-- OpenGPU Testbench --

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.ogpu_data_record_pkg.all;

entity ogpu_setup_testbench is
end ogpu_setup_testbench;

architecture ogpu_setup_tbi of ogpu_setup_testbench is
        constant clock_period : time := 20 ns;
        signal clock,reset        : std_logic :='0';
        --
        signal vx0: ogpu_float;
        signal vy0: ogpu_float;
        signal vx1: ogpu_float;
        signal vy1: ogpu_float;
        signal vx2: ogpu_float;
        signal vy2: ogpu_float;
        signal e0: ogpu_edge;
        signal e1: ogpu_edge;
        signal e2: ogpu_edge;
        -- DATAPATH/CONTROL interface signals
        signal start_raster: std_logic;
        signal setup_done: std_logic;
begin
ST1:  entity work.ogpu_setup(setup_1) port map(
                                                                -- IN
                                                                clock=>clock,reset=>reset,
                                                                d.vx0=>vx0,d.vy0=>vy0,
                                                                d.vx1=>vx1,d.vy1=>vy1,
                                                                d.vx2=>vx2,d.vy2=>vy2,
                                                                d.start_raster=>start_raster,
                                                                -- OUT
                                                                q.setup_done=>setup_done,
                                                                q.e0=>e0,q.e1=>e1,q.e2=>e2);

reset_proc: process
        begin
                --reset <= '0';
                --wait for 5*clock_period;
                reset <= '1';
                wait for 5*clock_period;
                reset <= '0';
                wait;
        end process;

stim_proc: process
        begin
                start_raster <= '1';
                wait for 4*clock_period;
                start_raster <= '0';
                wait for 4*clock_period;
        end process;

stim_proc2: process
        variable i:integer:=1;
        begin
                vx0.int <= to_unsigned(1*i,16); vy0.int <= to_unsigned(1*i,16);
                vx1.int <= to_unsigned(6*i,16); vy1.int <= to_unsigned(2*i,16);
                vx2.int <= to_unsigned(2*i,16); vy2.int <= to_unsigned(5*i,16);
                wait for 6*clock_period;
                i:=i+1;
                if i>10 then i:=0; end if;
        end process;

clock_process: process
        begin
                clock <= not clock;
                wait for clock_period/2;
        end process;
end ogpu_setup_tbi;
```

```
library ieee;
use ieee.std_logic_1164.all;
use work.ogpu_data_record_pkg.all;

entity ogpu_triangle_edge_test is
        port(clock: in std_logic;
                reset: in std_logic;
                d: in ogpu_triangle_edge_test_in_type;
                q: out ogpu_triangle_edge_test_out_type);
end ogpu_triangle_edge_test;

architecture triangle_test_1 of ogpu_triangle_edge_test is
        type reg_type is record
                edge_ready: std_logic_vector(0 to 2);
                draw_quad: std_logic;
                discard_quad: std_logic;
                quad_mask: std_logic_vector(0 to 3);
        end record;

        signal r,rin     :        reg_type;
begin
comb: process(reset,d,r)
        variable et0,et1,et2,et3 : std_logic;
        variable ot0,ot1,ot2,ot3 : std_logic;
        variable v : reg_type;
        begin
                v := r; --default assignment

                v.edge_ready := d.edge_ready;

                if (v.edge_ready(0) and v.edge_ready(1) and v.edge_ready(2))='1' then
                -- this logic comes from algorithm to check need of drawing quad fragment
                        et0 := d.edge_mask0(0) and d.edge_mask1(0) and d.edge_mask2(0);
                        et1 := d.edge_mask0(1) and d.edge_mask1(1) and d.edge_mask2(1);
                        et2 := d.edge_mask0(2) and d.edge_mask1(2) and d.edge_mask2(2);
                        et3 := d.edge_mask0(3) and d.edge_mask1(3) and d.edge_mask2(3);
                        ot0 := d.edge_mask0(0) or d.edge_mask1(0) or d.edge_mask2(0);
                        ot1 := d.edge_mask0(1) or d.edge_mask1(1) or d.edge_mask2(1);
                        ot2 := d.edge_mask0(2) or d.edge_mask1(2) or d.edge_mask2(2);
                        ot3 := d.edge_mask0(3) or d.edge_mask1(3) or d.edge_mask2(3);
                        v.quad_mask(0):=(et0)  or  (not(ot0)); -- if <all< edge tests are (1,1,1) or
(0,0,0) then draw fragment
                        v.quad_mask(1):=(et1) or (not(ot1));
                        v.quad_mask(2):=(et2) or (not(ot2));
                        v.quad_mask(3):=(et3) or (not(ot3));
                        v.draw_quad    :=   v.quad_mask(0)   or   v.quad_mask(1)   or   v.quad_mask(2)   or
v.quad_mask(3);
                        v.discard_quad := not v.draw_quad;
                else
                        v.draw_quad := '0';
                        v.discard_quad := '0';
                end if;

                if reset = '1' then
                        v.draw_quad := '0';
                        v.discard_quad := '0';
                end if;

                rin <= v;                -- drive register inputs

                q.draw_quad <= r.draw_quad;  -- drive module outputs
                q.discard_quad <= r.discard_quad;
                q.quad_mask <= v.quad_mask;

        end process;

seq: process(clock)
        begin
                if rising_edge(clock) then r <= rin; end if;
        end process;
end triangle_test_1;
```

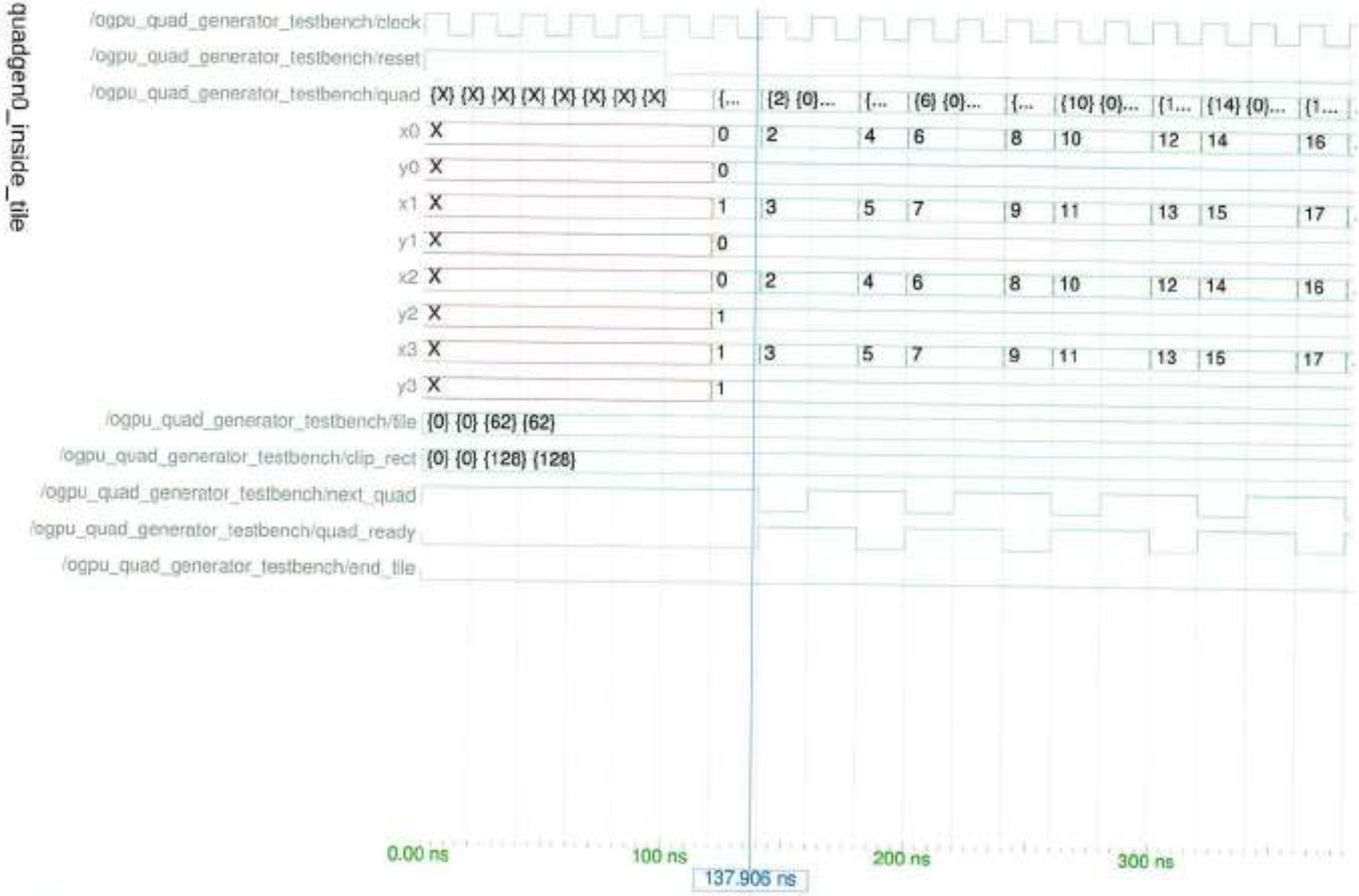
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.ogpu_data_record_pkg.all;

entity ogpu_triangle_edge_test_testbench is
end ogpu_triangle_edge_test_testbench;

architecture ogpu_triangle_edge_test_tb1 of ogpu_triangle_edge_test_testbench is
        constant clock_period : time := 20 ns;
        signal clock,reset        : std_logic :='0';
        --
        signal edge_ready: std_logic_vector(0 to 2):=(others=>'0');
        signal edge_mask0: std_logic_vector(0 to 3);
        signal edge_mask1: std_logic_vector(0 to 3);
        signal edge_mask2: std_logic_vector(0 to 3);
        signal quad_mask: std_logic_vector(0 to 3);
        -- DATAPATH/CONTROL interface signals
        signal draw_quad: std_logic;
        signal discard_quad: std_logic;
begin
TE1:  entity work.ogpu_triangle_edge_test(triangle_test_1) port map(
                                                                    -- IN
                                                                    clock=>clock,reset=>reset,
                                                                    d.edge_ready=>edge_ready,

        d.edge_mask0=>edge_mask0,d.edge_mask1=>edge_mask1,d.edge_mask2=>edge_mask2,
                                                                    -- OUT

        q.draw_quad=>draw_quad,q.discard_quad=>discard_quad,
                                                                    q.quad_mask=>quad_mask);

reset_proc: process
        begin
                --reset <= '0';
                --wait for 5*clock_period;
                reset <= '1';
                wait for 5*clock_period;
                reset <= '0';
                wait;
        end process;

stim_proc: process
        begin
                --wait for 5*clock_period;
                edge_mask0 <= "0110";
                edge_mask1 <= "0101";
                edge_mask2 <= "0110";
                -- expects quad_mask="1100", because only (0,0,0) and (1,1,1) mean "inside triangle"
situation
                wait for 6*clock_period;
                edge_ready(0)<='1';
                wait for clock_period;
                edge_ready(1)<='1';
                wait for clock_period;
                edge_ready(2)<='1';
                wait for 2*clock_period;
                edge_ready<="000";
                wait for clock_period;

                edge_mask0 <= "1111";
                edge_mask1 <= "1111";
                edge_mask2 <= "0000";
                -- expects quad_mask="0000", because only (0,0,0) and (1,1,1) mean "inside triangle"
situation
                wait for 1*clock_period;
                edge_ready<="111";
                wait for 2*clock_period;
                edge_ready<="000";
                wait for clock_period;

                edge_mask0 <= "1110";
                edge_mask1 <= "0100";
                edge_mask2 <= "0110";
                -- expects quad_mask="0101", because only (0,0,0) and (1,1,1) mean "inside triangle"
situation
                wait for 1*clock_period;
                edge_ready<="111";
                wait for 2*clock_period;
                edge_ready<="000";
                wait for clock_period;

                edge_mask0 <= "0011";
```

```
                edge_mask1 <= "0110";
                edge_mask2 <= "0011";
                -- expects quad_mask="1010", because only (0,0,0) and (1,1,1) mean "inside triangle"
situation
                wait for 1*clock_period;
                edge_ready<="111";
                wait for 2*clock_period;
                edge_ready<="000";
                wait for clock_period;

                edge_mask0 <= "1001";
                edge_mask1 <= "0011";
                edge_mask2 <= "1001";
                -- expects quad_mask="0101", because only (0,0,0) and (1,1,1) mean "inside triangle"
situation
                wait for 1*clock_period;
                edge_ready<="111";
                wait for 2*clock_period;
                edge_ready<="000";
                wait for clock_period;

                edge_mask0 <= "1100";
                edge_mask1 <= "1001";
                edge_mask2 <= "1100";
                -- expects quad_mask="1010", because only (0,0,0) and (1,1,1) mean "inside triangle"
situation
                wait for 1*clock_period;
                edge_ready<="111";
                wait for 2*clock_period;
                edge_ready<="000";
                wait for clock_period;

                wait;
        end process;

clock_process: process
        begin
                clock <= not clock;
                wait for clock_period/2;
        end process;
end ogpu_triangle_edge_test_tb1;
```

quad_edge_test

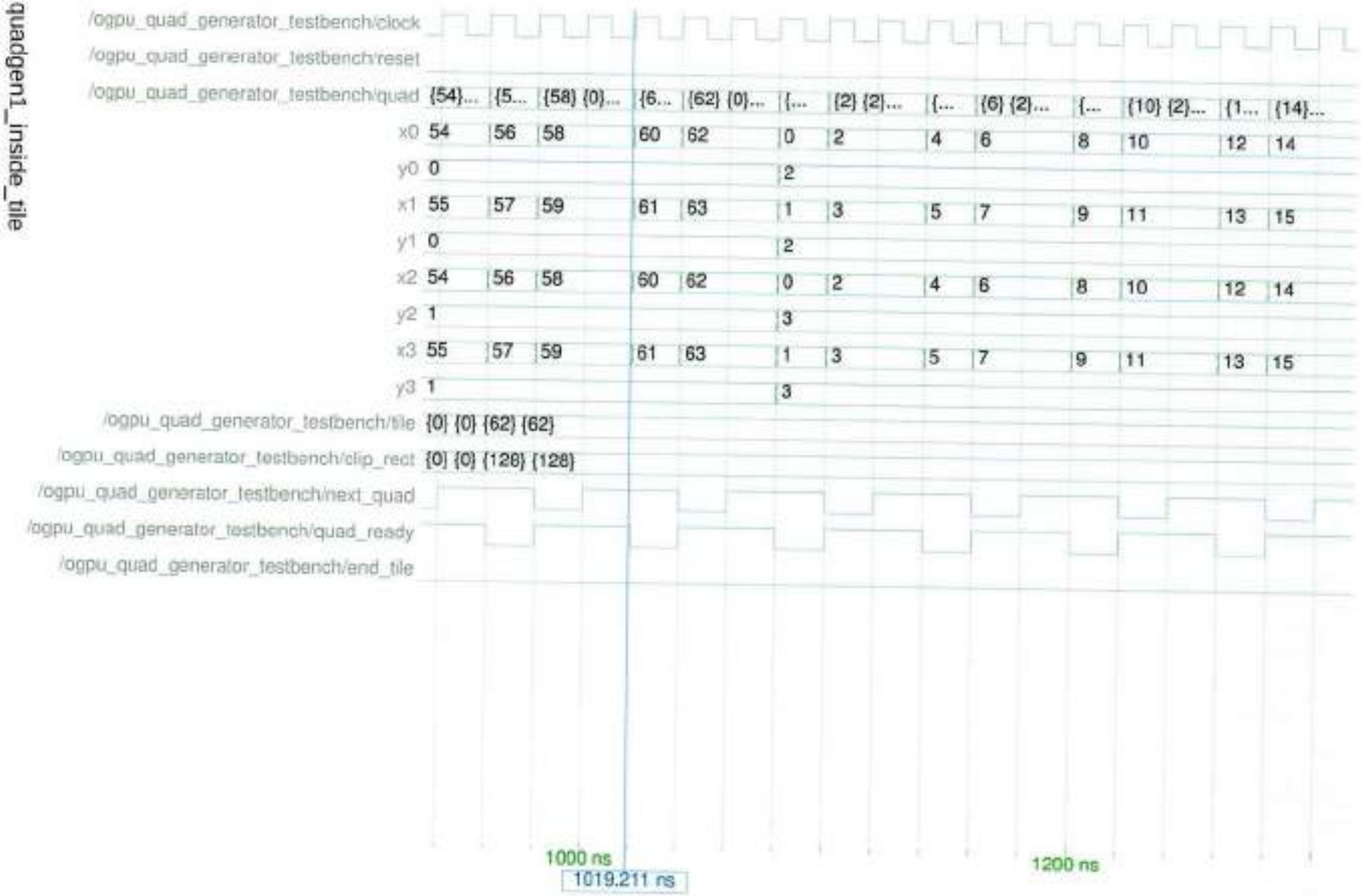


quadgen0_inside_tile



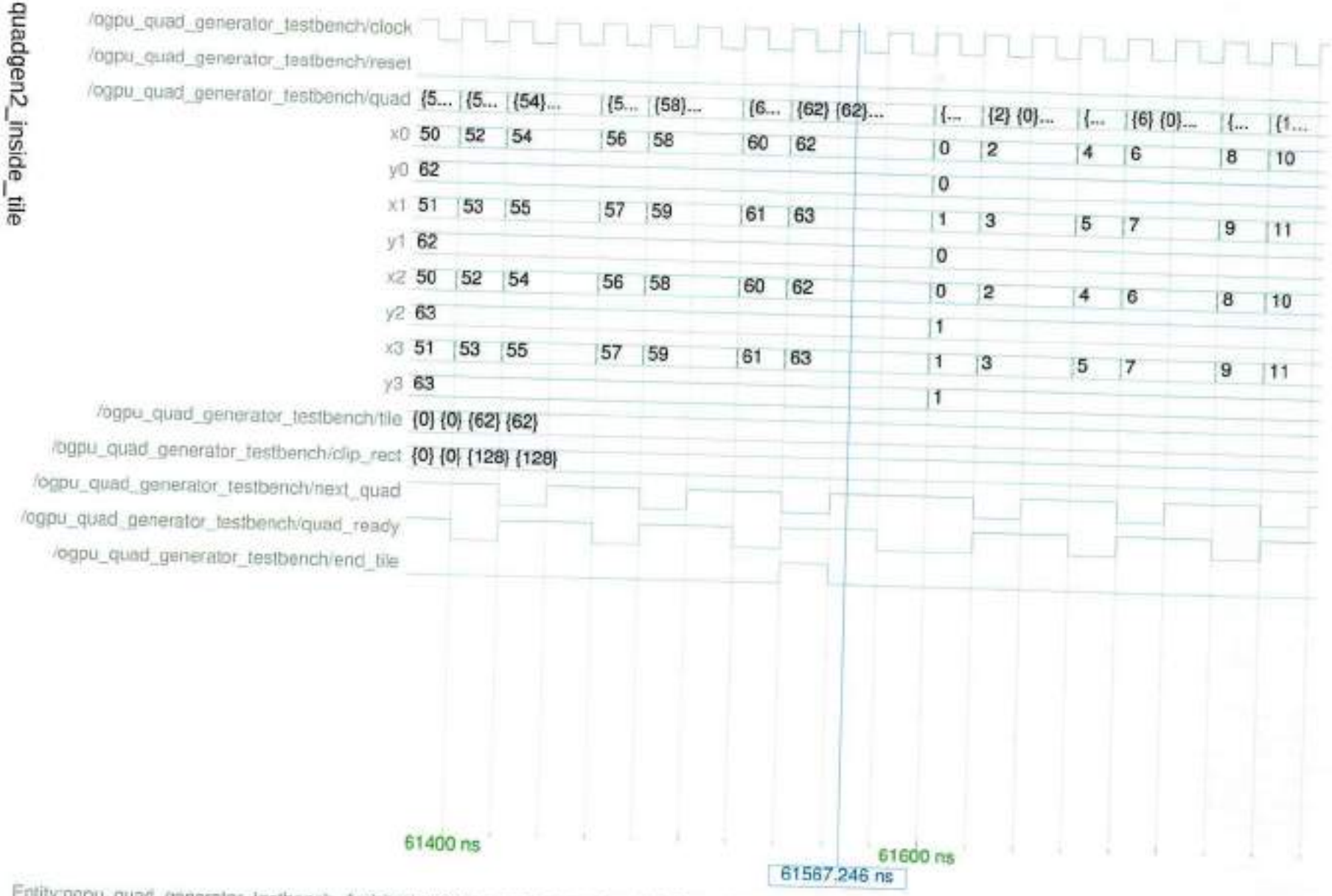
Entity:ogpu_quad_generator_testbench Architecture:ogpu_quad_generator_tb1 Date: Mon Dec 05 12:04:22 BRST 2016 Row: 1 Page: 1

quadgen1_inside_tile



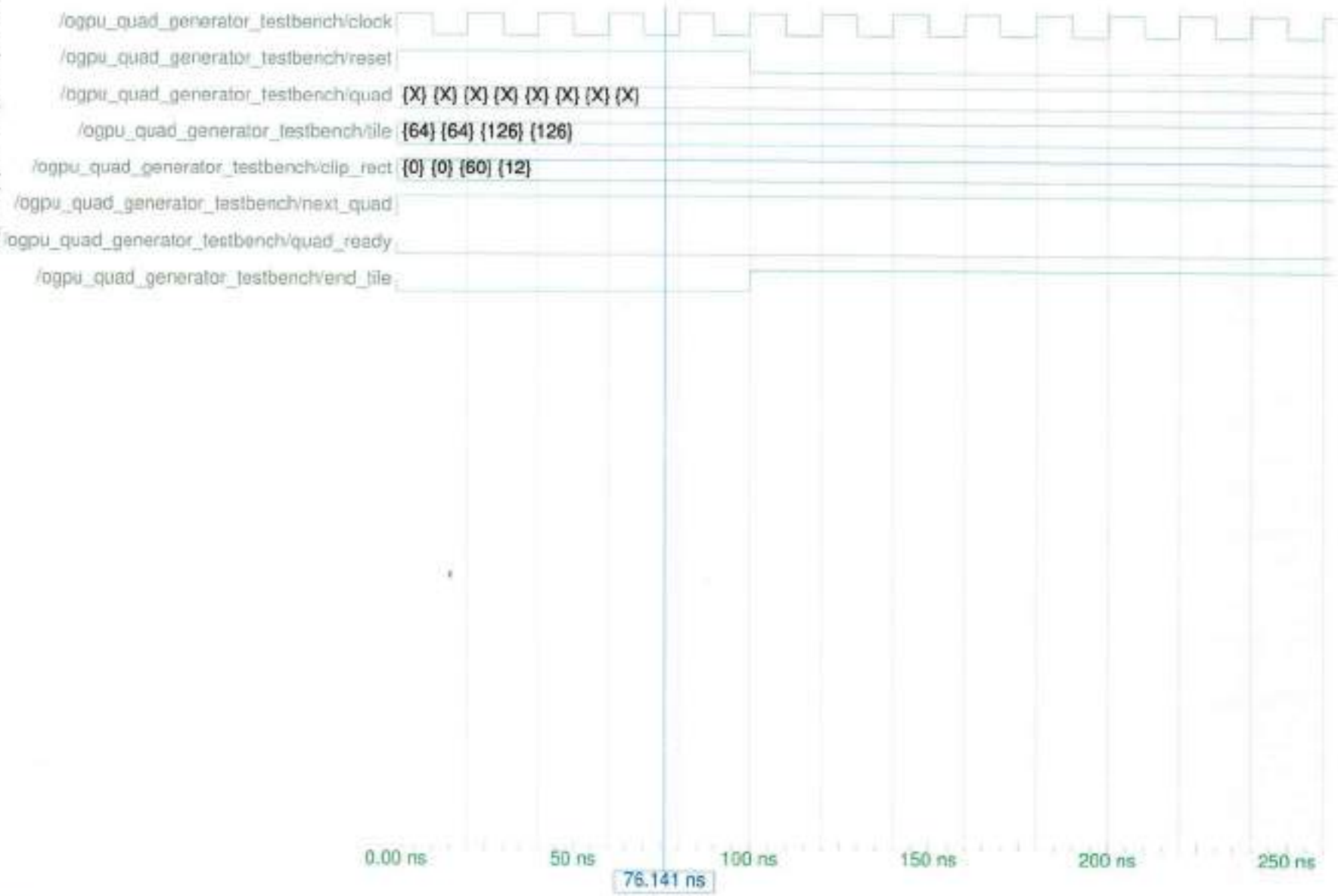
Entity:ogpu_quad_generator_testbench Architecture:ogpu_quad_generator_tb1 Date: Mon Dec 05 12:06:01 BRST 2016 Row: 1 Page: 1

quadgen2_inside_tile



Entity:ogpu_quad_generator_testbench Architecture:ogpu_quad_generator_tb1 Date: Mon Dec 05 12:07:40 BRST 2016 Row: 1 Page: 1

quadgen0_outside_tile



Entity:ogpu_quad_generator_testbench Architecture:ogpu_quad_generator_tb1 Date: Mon Dec 05 12:09:58 BRST 2016 Row: 1 Page: 1

/sgpu_quad_store_testbench/clock
/sgpu_quad_store_testbench/reset
/sgpu_quad_store_testbench/quad_mask
/sgpu_quad_store_testbench/quad
/sgpu_quad_store_testbench/depth_quad
/sgpu_quad_store_testbench/buffer_ack
/sgpu_quad_store_testbench/buffer_addr
/sgpu_quad_store_testbench/quad_buffer_length
/sgpu_quad_store_testbench/buffer_address
/sgpu_quad_store_testbench/buffer_byte_enable
/sgpu_quad_store_testbench/buffer_write
/sgpu_quad_store_testbench/buffer_write_data
/sgpu_quad_store_testbench/clear_index
/sgpu_quad_store_testbench/store_quad
/sgpu_quad_store_testbench/quad_stored
/sgpu_quad_store_testbench/run_proc
XXXX
0000
0010
{0} {0} {1} {0} {0} {1} {1} {1}
{0} {0} {0} {0}
X
0
8
1
2
0000
0008
00
FF
0000000000000000
0002000040000000
0.00 ns
0.00 ns
100 ns
200 ns
300 ns
Entity:sgpu_quad_store_testbench Architecture:sgpu_quad_store_testbench_tb1 Date: Mon Dec 05 15:45:26 EST 2016 Row: 1 Page: 1

quad_store0

/qpu_quad_store_testbench/clock
/qpu_quad_store_testbench/reset
/qpu_quad_store_testbench/quad_mask 1100 1110
/qpu_quad_store_testbench/quad_data {60}... {62} {62} {63} {62} {62} {63} {63} {63} 0000 0010
/qpu_quad_store_testbench/qpu_quad {0} {0} {0} {0} {0} {0} {0} {0} {1} {0} {0} {0} {1} {1} {1} {1} {2}...
/qpu_quad_store_testbench/buffer_ack
/qpu_quad_store_testbench/status 0
/qpu_quad_store_testbench/quad_buffer_length 1023 1024
/qpu_quad_store_testbench/buffer_address 1FF0 1FF8
/qpu_quad_store_testbench/buffer_byte_enable FF
/qpu_quad_store_testbench/buffer_write
/qpu_quad_store_testbench/buffer_write_data 003C... 003E003E70000000 0000000000000000 00...
/qpu_quad_store_testbench/start_raster
/qpu_quad_store_testbench/store_quad
/qpu_quad_store_testbench/quad_stored
/qpu_quad_store_testbench/run_proc
122969.067 ns
143000 ns
182400 ns
Entity:qpu_quad_store_testbench Architecture:qpu_quad_store_testbench_b1 Date:Mon Dec 05 15:47:49 EST 2016 Row: 1 Page: 1

quad_store1

raster_unit0

raster_unit1

raster_unit2

raster_unit3

bpu_setup_testbench.clock
bpu_setup_testbench.reset
0.00 ns
200 ns
400 ns
600 ns
800 ns
179.905 ns

setup

triangle_edge_test0

triangle_edge_test1
150 ns
200 ns
300 ns

/ogpu_triangle_edge_test_tb/benchclock
/ogpu_triangle_edge_test_tb/benchreset
/ogpu_triangle_edge_test_tb/benchtriedge_mask
/ogpu_triangle_edge_test_tb/benchtriedge_mask0
/ogpu_triangle_edge_test_tb/benchtriedge_mask1
/ogpu_triangle_edge_test_tb/benchtriedge_mask2
/ogpu_triangle_edge_test_tb/benchtriangleid_mask
/ogpu_triangle_edge_test_tb/testbenchdraw_done
/ogpu_triangle_edge_test_tb/benchvalidtest_guid
111
000
111
000
1110
0011
1001
0100
0110
0011
1001
0101
1010
0101
triangle_edge_test2
400 ns
Entity:ogpu_triangle_edge_test_tb(bench) Architecture:ogpu_triangle_edge_test_tb1 Date: Mon Dec 05 13:25:00 IRST 2016 Row: 1 Page: 1

triangle_edge_test3