Margareth Mee Gomes de Lima

Arquitetura RTL de um bloco OSD (On-Screen Display)

Campina Grande, Brasil

Maio de 2021

Margareth Mee Gomes de Lima

Arquitetura RTL de um bloco OSD (On-Screen Display)

Relatório de Estágio Integrado submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no domínio da Engenharia Elétrica.

Universidade Federal de Campina Grande - UFCG

Centro de Engenharia Elétrica e Informática - CEEI

Departamento de Engenharia Elétrica - DEE

Coordenação de Graduação em Engenharia Elétrica - CGEE

Orientador: Gutemberg Gonçalves dos Santos Júnior

Campina Grande, Brasil

Maio de 2021

"Não sei por quais caminhos Deus me conduz, mas conheço bem o meu guia"
Martinho Lutero

Resumo

O presente relatório descreve as atividades desenvolvidas durante o período de estágio integrado realizado na Allegro DVT. A Allegro DVT atua na área de vídeo digital e possui diversas soluções para codificação e decodificação de vídeo. O estágio teve como objetivo o desenvolvimento da arquitetura RTL (*Register Transfer Level*) de um bloco OSD (*On-Screen Display*), responsável pela exibição de informações em vídeos, tais como logotipos, legendas, etc. Para tanto, foi realizado uma análise considerando as especificações exigidas e proposta uma arquitetura para o bloco OSD. Inicialmente, foi desenvolvido um modelo de referência em linguagem C, em seguida, esse modelo foi descrito em RTL. Realizou-se ainda o processo de validação do bloco utilizando testes unitários. Por fim, uma síntese foi feita para avaliação do bloco.

Palavras-chave: *hardware*, *Register Transfer Level*, *On-Screen Display*.

Abstract

The present report describes the activities carried out during a 22-week intership at Allegro DVT. Allegro DVT works in the field of digital video and offers many solutions for video encoders and decoders. The goal of this internship is designing a RTL- Register Transfer Level architecture of an On-Screen Display, a hardware component in charge of displaying information on the video, such as logos, captions, etc. Initially, an analysis was made considering the required specifications and an architecture for the OSD component was proposed. Additionally, a reference model was developed in C language, then this model was described in RTL. The hardware component was validated using unit tests. Finally, a synthesis was made to evaluate the OSD.

Keywords: hardware, Register Transfer Level, On-Screen Display.

Lista de tabelas

Lista de ilustrações

Lista de abreviaturas e siglas

ASIC	*Application Specific Integrated Circuits*
CTU	*Coding Tree Unit*
CHROMA	Crominância
IP	*Intellectual Property*
FIFO	*First int first out*
HEVC	*High Efficiency Video Coding*
LCU	*Largest Coding Unit*
LCD	*Liquid Crystal Display*
LUMA	Luminância
OSD	*On-Screen Display*
OLED	*Organic Light-Eitting Diode*
RGB	*red, green, blue*
RGBA	*red, green, blue, alpha*
RTL	*Register Transfer Level*

Sumário

1 Introdução

A codificação de vídeo tem exercido um papel cada vez mais importante no cenário mundial. É impressionante como a vida dos seres humanos está rodeada por informações que são transmitidas por meio de vídeos. A codificação de vídeos tem, portanto, se tornado fundamental, pois permite a compressão de vídeos para que sejam transmitidos usando um menor número de dados. O grande desenvolvimento tecnológico e computacional das últimas décadas permitiu um grande avanço no processo de codificação e decodificação de vídeos. Hoje é possível realizar a transmissão de vídeos sem perder muito em sua qualidade, graças às poderosas técnicas de compressão existentes.

Este relatório descreve as atividades desenvolvidas durante o estágio realizado na empresa Alegro de 18/02/2019 a 26/07/2019. A Alegro é uma empresa que atua nesta área e desenvolve o design e arquitetura de IP's - *Intellectual Property* para codificação e decodificação de vídeos digitais de alta qualidade. A empresa foi fundada em 2003 e está sediada em Grenoble- França. A empresa é composta por duas grandes equipes: certificação e IP. A equipe de certificação fornece sequências de vídeo que certificam decodificadores de vídeo, essas sequências são geradas de acordo com os padrões internacionais que as regulamentam. A segunda equipe é a Equipe IP, responsável pelo desenvolvimento de codificadores e decodificadores de vídeo, esta equipe está, por sua vez, subdividida em *software* e *hardware*.

O tema deste projeto de estágio surgiu a partir da necessidade de se ter um bloco OSD - *On-Screen Display* nos codificadores. O OSD é responsável por exibir informações no vídeo que será codificado. Em muitas situações, faz-se necessário exibir informações como logotipos, legendas, hora, data, etc. São várias as possibilidades de aplicação para o OSD, motivo que levou a equipe de *hardware* a se interessar pelo seu desenvolvimento.

O estágio consiste na proposição e design de uma arquitetura para o OSD. A arquitetura proposta deve levar em consideração as especificações impostas pelo codificador já existente. Além disso, o bloco deve suportar os diferentes modos de operação do

codificador, sendo capaz de lidar com os diversos casos possíveis.

Este relatório aborda o estudo e o trabalho realizado durante o estágio e está organizado em quatro capítulos. O primeiro capítulo apresenta a introdução e os objetivos do trabalho. O segundo capítulo explica os princípios usados para a conversão de formatos de imagem e a combinação dos vídeos. O terceiro capítulo descreve as atividades desenvolvidas durante o estágio, as especificações do bloco OSD, a descrição da arquitetura do OSD e seu processo de validação. O quarto capítulo apresenta as conclusões do trabalho.

1.1 Objetivos

O objetivo deste estágio consiste em projetar a arquitetura RTL de um bloco OSD (On-Screen Display) que será integrado nos codificadores de vídeo desenvolvidos pela Alegro. Para tanto, faz-se necessário estudar o processo de conversão de imagens RGB para YUV, bem como a operação de combinação de dois vídeos. A conversão deve ser feita de forma que atenda às normas de conversão existentes. Além disso, é importante garantir que a imagem tenha uma boa qualidade após a conversão. Em seguida, será necessário desenvolver um código em linguagem C, esta etapa permitirá desenvolver um modelo de referência no qual o hardware se baseará. Os resultados gerados por este código-fonte serão usados para comparar os resultados produzidos pelo hardware.

Será realizado um estudo sobre a arquitetura do bloco no intuito de encontrar a solução mais eficiente e economicamente possível considerando as restrições operacionais do bloco e o ambiente no qual ele será integrado. Testes unitários serão usados para validar cada sub-bloco que compõe o bloco OSD. Finalmente, os sub-blocos serão reunidos para testar e validar o bloco OSD por completo. Após esta validação, o bloco será sintetizado para avaliá-lo em termos de *timing* e área de superfície. Por fim, o bloco será integrado a um codificador de vídeo.

2 Princípios da conversão de vídeo

Este capítulo apresenta os princípios necessários para compreender a conversão de imagens e a operação Alpha Blending.

2.1 Imagens

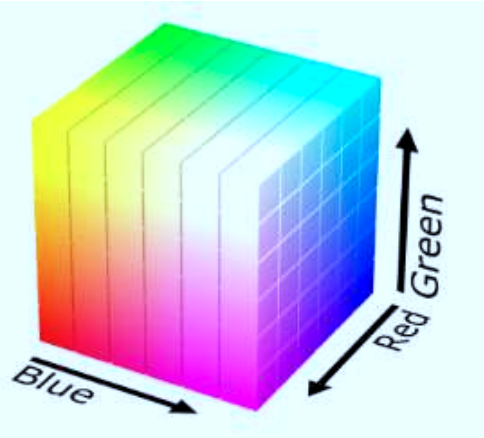
Geralmente, as informações que serão exibidas no vídeo estão no formato RGB - *red, green, blue*, esta é na verdade uma codificação de cores baseada no sistema de cores primárias:

- Vermelho (de comprimento de onda igual a 700,0 nm).
- Verde (de comprimento de onda igual a 546,1 nm).
- Azul (de comprimento de onda igual a 435,8 nm).

Este sistema corresponde à maneira como as cores são geralmente codificadas nos computadores. É comumente utilizado para exibição de cores por meio de tubos de raios catódicos, LCD *(Liquid Crystal Display)*, *Plasma Display* ou OLED *(Organic Light-Emitting Diode)*, estando assim presente em diversos dispositivos eletrônicos como computadores e televisores. Cada pixel é formado pelas três fontes de cores (vermelho, verde e azul) que são aproximadas o suficiente para formar a cor do pixel. A Figura 1 mostra o espaço de cores RGB representado na forma de um cubo. O eixo X representa a cor vermelha, que cresce para a esquerda, o eixo Y representando a cor azul, que cresce para a direita e o eixo Z, representando a cor verde, cresce para cima. Cada ponto do cubo representa uma cor formada a partir da combinação dos três valores (X, Y, Z).

Cada componente é codificada em 8 bits, resultando em 24 bits por pixel. Dessa forma existem 256 intensidades dadas por cada componente, o que gera 16.777.216 cores

Figura 1 – Espaço de cores RGB.



Fonte: http://ecor.ib.usp.br/lib/exe/detail.php.

diferentes para este formato. No entanto, as cores exibidas em uma tela são limitadas por vários fatores, como o *hardware* usado ou até mesmo a redução do número de bits usados para codificar as cores. Outra limitação consiste na capacidade do olho humano de distinguir as cores (cerca de 2 milhões de cores), de forma que o olho humano não consegue diferenciar todas as cores existentes no formato RGB.

É possível ter ainda uma quarta componente chamada A (alfa) para representar a transparência, constituindo assim o formato RGBA. Este formato com 4 componentes será utilizado pelo OSD uma vez que as informações contidas nesta quarta componente são necessárias para combinar os dois vídeos. O alfa geralmente é representado por um número entre 0 e 1, onde 0 significa que o vídeo é completamente transparente e 1 completamente opaco. Esta componente é normalmente codificada em 8 bits, portanto, possui valores entre 0 e 255. A Figura 2 exemplifica a finalidade desta componente, o logotipo "Allegro" é adicionado a uma imagem de base que aparece no fundo. Em regiões onde há informações a serem adicionadas, o valor alfa é diferente de zero (opaco), enquanto que em outras regiões o valor de alfa é zero (transparente). O formato utilizado das informações (logo, texto) que serão adicionadas ao vídeo de base possuem o formato RGBA 32 bits, isto é, 8 bits para cada componente.

2.2 Imagens YUVA

Outro espaço vetorial usado para a representação de cores é o formato YUV. Este espaço tem três componentes: Y, U e V. A primeira componente (Y) representa a luminância (LUMA) e as componentes UV representam a crominância (CHROMA). A

Figura 2 – Logo Allegro.

Fonte: Próprio Autor.

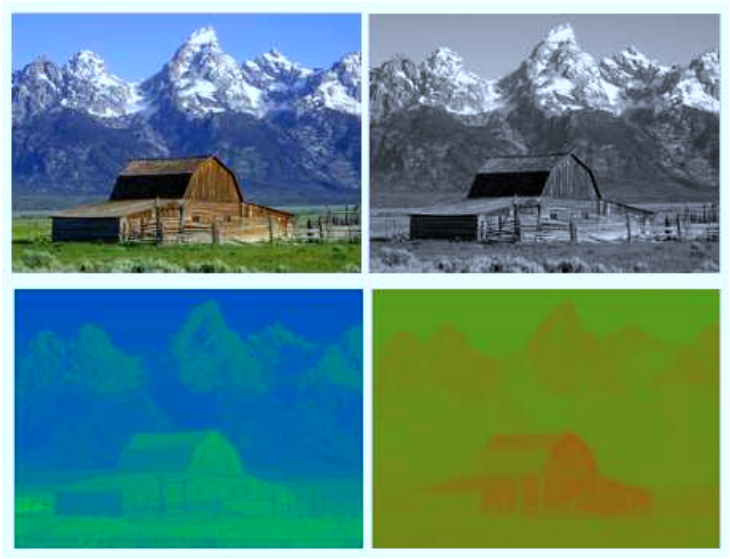
LUMA representa o nível de cinza da imagem, isto é, a variação entre preto e branco, ela é obtida a partir da combinação dos valores de R, G e B. A componente U representa a diferença entre Y e B, enquanto que a componente V representa a diferença entre Y e R.

Este padrão foi desenvolvido para garantir a compatibilidade entre receptores de televisão em preto e branco e receptores em cores. No caso dos receptores preto e branco, apenas a LUMA é levada em conta, enquanto que no caso dos receptores coloridos LUMA e CHROMA são levados em conta.

O formato YUV permite reduzir a largura de banda da crominância considerando alguns aspectos da percepção humana. O olho humano é, na verdade, muito mais sensível ao preto e branco do que à cor. Pode-se remover parte da CHROMA, mantendo a LUMA intacta, e obter uma imagem com ótima qualidade. Isso ocorre porque a LUMA guarda as características mais significativas para o olho humano. Portanto, pode-se diminuir o número de dados usados para representar uma imagem sem, contudo, perder significativamente em qualidade. Este processo de compressão é chamado de decimação, e permite reduzir uma imagem por um fator de 0,5 (decimação 4:2:2), de 0,25 (decimação 4:2:0), ou ainda eliminar completamente a crominância (4:0:0). Naturalmente, o tipo de aplicação é que definirá qual o fator de redução a ser utilizado. A decimação é, portanto, uma vantagem do sistema YUV.

A Figura 3 mostra um exemplo de uma imagem YUV e suas respectivas componentes Y, U e V. A imagem no canto superior esquerdo é a imagem original, que contém as três componentes (YUV). No canto superior direito tem-se uma imagem apenas com a componente Y. No canto inferior esquerdo apenas a componente U e no canto inferior direito apenas a componente V. Essas imagens também mostram que a LUMA de fato contém as informações mais significativas da imagem, ou seja, a LUMA constitui a base da imagem.

Figura 3 – Decomposição de uma imagem em Y,U e V.



Fonte: https://en.wikipedia.org/wiki/YUV/media/File:Barn-yuv.png.

2.3 Conversão RGB-YUV

Os codificadores desenvolvidos pela Allegro utilizam o formato de vídeo YUV. No entanto, as informações que serão adicionadas estão no formato RGB. Para poder combinar os dois vídeos, deverá ser feito primeiramente a conversão do vídeo RGB para YUV. O processo de conversão é regulado por padrões existentes na área de vídeos. A organização responsável por definir os regulamentos usados no campo da codificação de vídeo é o Setor de Padronização de Telecomunicações da ITU.

As equações 3.1, 2.2 e 2.3 para conversão RGB-YUV são encontradas nas normas H.265 (ITU, 2018) e H.264 (ITU, 2017):

$$Y = K_r \cdot R + (1 - K_R - K_B) \cdot G + K_b \cdot B \tag{2.1}$$

$$U = \frac{0.5 \cdot (B - Y)}{(1 - K_B)} \tag{2.2}$$

$$V = \frac{0.5 \cdot (R - Y)}{(1 - K_R)} \tag{2.3}$$

Os coeficientes Kr e Kb são parâmetros definidos pelo padrão H.265. R, G e B são os componentes do formato RGB e seus valores são representados entre [0, 255]. A operação de conversão é evidentemente uma mudança de espaço vetorial. Para melhor

visualizar esta operação, as equações foram colocadas em formato matricial:

$$Y = a_{11} \cdot R + a_{12} \cdot G + a_{13} \cdot B \quad (2.4)$$

$$U = -a_{21} \cdot R - a_{22} \cdot G + a_{23} \cdot B \quad (2.5)$$

$$V = a_{31} \cdot R - a_{32} \cdot G - a_{33} \cdot B \quad (2.6)$$

onde:

$$a_{11} = K_R \quad (2.7)$$

$$a_{12} = (1 - K_R - K_B) \quad (2.8)$$

$$a_{13} = K_B \quad (2.9)$$

$$a_{21} = \frac{K_R}{2 \cdot (1 - K_B)} \quad (2.10)$$

$$a_{22} = \frac{1 - K_R - K_B}{2 \cdot (1 - K_B)} \quad (2.11)$$

$$a_{23} = 0.5 \quad (2.12)$$

$$a_{31} = 0.5 \quad (2.13)$$

$$a_{32} = \frac{1 - K_R - K_B}{2 \cdot (1 - K_R)} \quad (2.14)$$

$$a_{33} = \frac{K_B}{2 \cdot (1 - K_R)} \quad (2.15)$$

Os valores de LUMA obtidos estão contidos entre [0,255], U e V estão contidos entre [-128 ,127]. A fim de se obter a mesma faixa de variação é necessário deslocar U e V de 128, sendo assim, as equações 2.4, 2.5 e 2.6 tornam-se:

$$Y = a_{11} \cdot R + a_{12} \cdot G + a_{13} \cdot B \quad (2.16)$$

$$U = -a_{21} \cdot R - a_{22} \cdot G + a_{23} \cdot B + 128 \quad (2.17)$$

$$V = a_{31} \cdot R - a_{32} \cdot G - a_{33} \cdot B + 128 \quad (2.18)$$

Essas equações apresentadas são válidas se cada componente do formato YUV também for codificado em 8 bits, ou seja, o BIT_DEPTH é 8b. Se o BIT_DEPTH for igual a 10b, será necessário deslocar o resultado 2 bits para a esquerda e então adicionar 512.

2.4 Decimação

Decimação é uma forma de reduzir o volume de dados associados ao vídeo no formato YUV. Esse método leva em consideração o fato de que o olho humano é muito

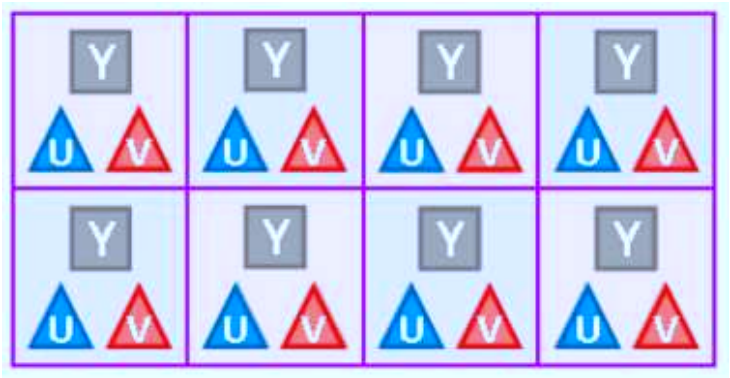
mais sensível às variações de preto e branco do que às cores. Assim, é possível reduzir o tamanho dos dados de U e V sem degradar a qualidade percebida da imagem. Este método preserva a componente Y, onde o brilho está localizado e, portanto, as características mais importantes da imagem.

A Figura 4 ilustra uma estrutura de 8 pixels, cada quadrado representa um pixel e contém uma componente Y, U e V. A decimação é representada por três valores A:B:C.

- A é o número de amostras de luminância (Y) por linha.
- B é o número de amostras de crominância (UV) na primeira linha.
- C é o número de amostras de crominância (UV) na segunda linha.

Assim no caso da decimação 4:4:4, A=B=C=4, isto é, temos 4Y para cada linha, 4UV na primeira linha e 4UV na segunda linha.

Figura 4 – Decimação 4:4:4.



Fonte: http://www.latelierducable.com/tv-televiseur/yuv-420-ycbcr-422-rgb-444-cest-quoi-le-chroma-subsampling/.

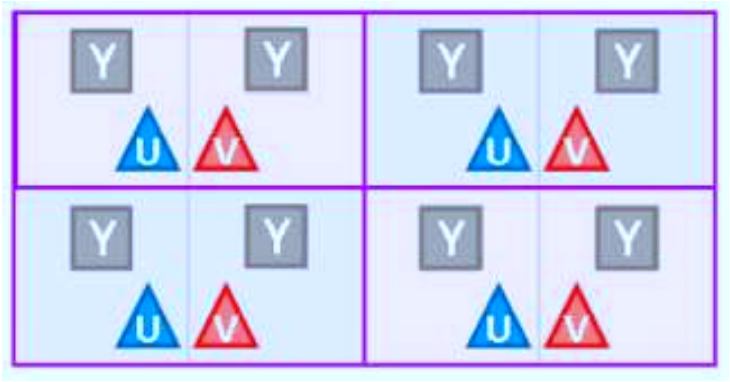
Os quadrados maiores correspondem aos pixels, os quadrados cinzas Y correspondem à luminância, os triângulos U e V correspondem à crominância. De acordo com este sistema, cada pixel corresponde a uma amostra composta por um conjunto de valores de Y, U, V. Se o BIT_DEPTH for igual a 8b, por exemplo, será necessário (8Y + 8U + 8V)*8 = 192 bits para codificar os 8 pixels, já que cada pixel é codificado em 24 bits.

2.4.1 Decimação 4:2:2

A decimação 4:2:2 permite reduzir à metade a crominância (Figura 5).

- A = 4 pois a LUMA não sofre decimação.
- B = 2 porque há duas amostras de UV na primeira linha.
- C = 2 porque há duas amostras de UV na segunda linha.

Figura 5 – Decimação 4:2:2.



Fonte: http://www.latelierducable.com/tv-televiseur/yuv-420-ycbcr-422-rgb-444-cest-quoi-le-chroma-subsampling/.

Se o BIT_DEPTH for igual a 8 bits, o total de bits usados será (8Y + 4U + 4V)*8 = 128 bits. A taxa de bits é reduzida em (192-128)/192 = 1/3. Este tipo de decimação é muito usada porque permite reduzir em 1/3 a quantidade de bits usada, mantendo uma imagem com pouquíssima ou nenhuma diferença visual.

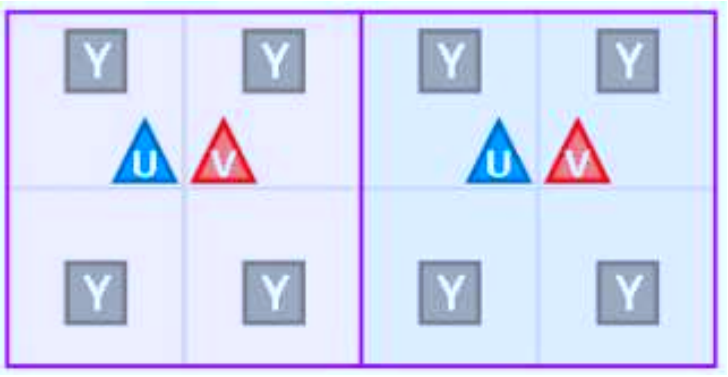
2.4.2 Decimação 4:2:0

A decimação 4:2:0 permite reduzir horizontal e verticalmente a crominância pela metade (Figura 6)

- A = 4 porque a LUMA não sofre modificação.
- B = 2 porque há duas amostras de UV na primeira linha.
- C = 0 porque não há amostras de UV na segunda linha.

Se o BIT_DEPTH for igual a 8 bits, o total de bits usados será (8Y + 2U +2V)*8 = 96 bits. A taxa de bits é reduzida em (192-96)/192 = 0,5. Este tipo de decimação reduz, portanto, significativamente o tamanho dos dados usados para codificar a imagem, porém a diferença entre a imagem original (4:4:4) e a imagem decimada (4:2:0) é mais notável.

Figura 6 – Decimação 4:2:2.



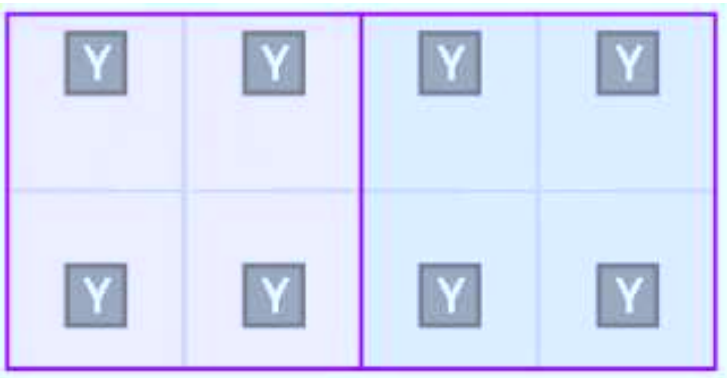
Fonte: http://www.latelierducable.com/tv-televiseur/yuv-420-ycbcr-422-rgb-444-cest-quoi-le-chroma-subsampling/.

2.4.3 Decimação 4:0:0

A decimação de 4:0:0 remove toda a crominância, deixando apenas a LUMA (Figura 7).

- A = 4 porque a LUMA não sofre modificação.
- B = 0 porque a CHROMA é eliminada completamente.
- C = 0 porque a CHORMA é eliminada completamente.

Figura 7 – Decimação 4:0:0.



Fonte: http://www.latelierducable.com/tv-televiseur/yuv-420-ycbcr-422-rgb-444-cest-quoi-le-chroma-subsampling/.

A imagem de saída da decimação 4:0:0 é uma imagem em preto e branco. Se o BIT DEPTH for igual a 8 bits, o total de bits usados será (8Y)*8 = 64 bits. A taxa de bits é reduzida em (192-64)/192 = 2/3.

3 Atividades Desenvolvidas

3.1 Especificações do bloco OSD

O OSD será integrado aos codificadores projetados pela Allegro, por isso é muito importante levar em consideração o funcionamento do ambiente ao qual o OSD será integrado. Ele deve ser compatível com os diferentes modos de operação do codificador.

O OSD deve ser capaz de trabalhar com um BIT_DEPTH de 8 ou 10 bits, pois os codificadores o são. Além disso, ele deve ser capaz de operar com três tipo de decimação: 4:0:0, 4:2:2 e 4:2:0. Essa especificação será importante, por exemplo, para o bloco de conversão RGBA-YUVA.

A imagem que será entregue ao bloco OSD para ser adicionada ao vídeo se encontra no formato RGBA, onde cada pixel é codificado em 32 bits (8 bits por componente). Foi definido que o usuário terá até 8 regiões de exibição dentro dos limites de comprimento e largura do vídeo de base. As regiões não devem exceder os limites do vídeo, caso isso ocorra, o OSD deverá exibir a parte que não depassa a imagem.

Essas regiões não devem se sobrepor, pois isso criará um conflito ao exibir as informações. Devemos, portanto, garantir que as regiões não se sobreponham. As regiões são alinhadas em 32 bytes na memória, isso se deve ao tamanho do barramento AXI usado para transmitir os dados.

O bloco OSD terá uma interface com duas pontes. Uma ponte é um componente que geralmente estabelece a interface de dados entre os componentes do projeto. A Ponte-Source é uma ponte que já existia no codificador, ele é responsável por buscar na memória a LUMA e a CHROMA do vídeo de base. A outra ponte é a Ponte-RGBA, que deve procurar os pixels no formato RGBA e entregá-los ao OSD.

O OSD deve ser projetado em RTL com uma linguagem de descrição de hardware. A operação de cada sub-bloco deve ser validada por meio da co-simulação, usando testes unitários. Esses testes comparam os resultados gerados por um código de referência em

C com o resultado do hardware. Posteriormente, os sub-blocos devem ser instanciados em um único componente chamado OSD, que também deve ser validado. Por fim, será necessário realizar uma síntese ASIC - *Application Specific Integrated Circuits* do bloco para analisar os relatórios gerados e garantir que as restrições de *timing* e área de superfície sejam respeitadas.

3.2 Algoritmo do Bloco OSD

Escrever uma operação em um nível de programação mais alto permite ter uma maior liberdade para desenvolver o algoritmo e testá-lo. O algoritmo foi desenvolvido em linguagem C. O OSD é composto por duas grandes operações: conversão RGBA-YUVA e Alpha Blending. Este capítulo descreve os algoritmos usados nessas duas operações e os resultados obtidos.

3.2.1 Conversão RGBA-YUVA e decimação

A conversão de RGBA para YUVA é uma operação muito importante para o bloco OSD, pois seu resultado decidirá se a imagem exibida no vídeo terá uma boa qualidade ou não. Portanto, é importante garantir que o algoritmo realize uma boa conversão. As principais tarefas que o algoritmo deve realizar são:

- Ler de um arquivo .bitmap, formato em que a imagem RGBA é salva.
- Converter do formato RGBA para YUVA 4:4:4.
- Realizar a decimação de YUVA 4:4:4 para YUVA 4:2:2, YUVA 4:2:0 ou YUVA 4:0:0.
- Gerar arquivos YUV de saída contendo o resultado da conversão.

As funções que executam essas tarefas foram escritas em linguagem C. Os resultados gerados pelo algoritmo de conversão podem ser encontrados na Figura 8. As figuras (a),(b),(c) e (d) são muito semelhantes, isso é a prova de que a decimação pode comprimir a imagem sem perder muita qualidade. No caso dessas imagens, por exemplo, não é realmente possível ver as diferenças entre os diferentes tipos de decimação a olho nu, exceto para a decimação de 4:0:0, onde existe apenas a LUMA. Para que seja possível ver as diferenças na qualidade da imagem para diferentes decimações, a imagem RGB deve ter uma qualidade muito baixa.

3.2.2 Alpha Blending

O Alpha Blending adiciona a imagem YUVA ao vídeo de base. Esta etapa é, portanto, tão importante quanto a última para garantir um bom resultado final. As tarefas que este algoritmo deve realizar são:

- Ler o arquivo no formato YUV que contém o vídeo. Este arquivo pode estar em diferentes formatos: 4:2:2, 4:2:0 ou 4:0:0.
- Combinar as duas imagens de acordo com o parâmetro alfa associado a cada pixel.
- Gerar um arquivo YUV com o vídeo e a imagem exibida no vídeo.

As funções que executam essas tarefas foram escritas em linguagem C. Um exemplo do resultado gerado pelo código pode ser encontrado na Figura 9. A imagem de fundo já estava no formato YUV 4:4:4 e o logotipo (no canto inferior direito) estava no formato RBGA. Primeiramente, a função de conversão transformou o logotipo em YUVA 4:4:4. Em seguida, o logotipo foi combinado à imagem de base por meio da função Alpha Blending. O algoritmo deve obedecer às mesmas especificações de hardware para que as saídas geradas por ambos sejam exatamente as mesmas e possam ser comparadas.

3.3 Concepção do bloco OSD

Este capítulo discute a descrição e as especificações do bloco de hardware OSD. A descrição da arquitetura geral e de cada sub-bloco também são apresentadas.

3.3.1 Especificações do bloco OSD

O bloco On-Screen Display é um bloco que será utilizado nos codificadores de vídeo desenvolvidos pela Allegro DVT. A função do OSD é combinar dois vídeos, um vídeo base com outro vídeo que contém as informações a serem exibidas. Uma aplicação para este bloco é adicionar legendas ou um logotipo a um vídeo. As informações que serão exibidas dependem da finalidade do vídeo.

Uma arquitetura geral do bloco OSD (Figura 10) foi apresentada pela empresa como uma proposta do que deveria ser entregue ao final do estágio. A arquitetura é composta por 4 blocos principais: RGBAtoYUVA, Color Reverse, Resize e Alpha Blending. Além disso, há uma ponte na entrada do bloco que faz a interface do bloco com a fonte de dados.

O bloco RGBAtoYUVA é um bloco que converte as informações que serão exibidas no vídeo. Esta informação está a priori no formato RGBA, por isso deve ser convertida em YUVA a fim de combinar os dois vídeos no bloco Alpha Blending. O bloco Resize permite redimensionar a imagem caso seja necessário. O bloco Color Reverse permite comparar as cores das informações que serão exibidas e as do fundo para saber se são semelhantes. Se as cores forem semelhantes ou iguais, este bloco permite modificar a cor da imagem que será exibida no vídeo para que haja um maior contraste. Por fim, o bloco Alpha Blending realiza a operação de combinação de dois vídeos, esta operação é feita de acordo com os valores do componente alfa associado a cada pixel da imagem RGBA.

3.3.1.1 As regiões de exibição

As informações que serão exibidas no vídeo são fornecidas pelo cliente. É por isso que o formato e as características dessas informações devem ser bem definidas para garantir o funcionamento correto do bloco. Se as especificações não forem bem definidas, a complexidade do bloco de hardware pode aumentar consideravelmente.

As informações estão no formato de imagem bitmap RGBA codificado em 32 bits, onde cada componente é codificado em 8 bits. Este formato foi escolhido pela empresa por ser um formato de imagem amplamente utilizado e de fácil utilização. Uma ponte será necessária para procurar os dados na memória e entregá-los ao bloco OSD.

Foi definido que o vídeo pode ter no máximo 8 regiões de exibição, onde cada região é definida por um tamanho e uma posição relativa (X, Y) ao vídeo de base. As regiões não devem se sobrepor, pois isso causaria problemas ao exibir informações.

Considerando um vídeo composto por imagens de 128x128 pixels, os casos extremos que podem ocorrer são: (1) não há informação a ser exibida no vídeo, portanto não há região. O bloco OSD funciona como um *bypass* e o vídeo no formato YUV é passado para o próximo bloco do codificador; (2) As 8 regiões são usadas para exibição, este é o caso mostrado na figura 11. Esta figura mostra uma imagem 128x128 dividida em 8 regiões. Cada região deve conter coordenadas (X, Y), que indicam seu início. A região 4, por exemplo, possui coordenadas (64,32). Isso significa que o pixel (0,0) da imagem RGBA da região 4 será exibido a partir desta coordenada até o pixel (64 + size_X_region4, 32 + size_Y_region4). Na Figura 14, as regiões ocupam todo o espaço do quadro, mas isso não é uma regra.

A Figura 11 é apenas um exemplo, as regiões podem de fato começar e terminar em qualquer pixel, a única restrição é que as regiões sejam alinhadas em 32 bytes. Esse alinhamento é ditado pelo tamanho do barramento AXI usado na arquitetura.

Outro exemplo que pode exemplificar como as regiões são definidas se encontra na figura 12. Neste caso, existem apenas 2 regiões de exibição e os tamanhos das regiões são diferentes. Em resumo, as regiões devem respeitar estas três regras: pode haver até 8 regiões, as regiões não se sobrepõem e devem ser alinhadas em 32 bytes.

As coordenadas de cada região e seus tamanhos são fornecidas pelo software de controle do bloco, esses valores podem mudar, por exemplo, de um *frame* para outro. O bloco OSD deve ser capaz de lidar com o caso em que o tamanho das regiões exceda o tamanho do vídeo. Nesse caso, o bloco deve truncar a imagem RGBA para mostrar apenas a parte que não está além da borda da imagem. A Figura 13 mostra o funcionamento do bloco quando esta situação ocorre.

O bloco OSD deve ser capaz de lidar com a situação em que a cor de fundo é a mesma ou próxima à da imagem RGB. Nesse caso, o usuário não consegue distinguir as informações exibidas do vídeo em segundo plano. O OSD deve, portanto, inverter ou alterar a cor da imagem RGB para que as informações exibidas sejam identificáveis. Esta etapa é realizada pelo sub-bloco Reverse Color, que identifica automaticamente as áreas que apresentam este tipo de problema e as reverte.

3.3.1.2 Modos de operação do OSD

Os codificadores desenvolvidos pela Allegro permitem trabalhar em diversos modos de operação, o bloco OSD deve ser compatível com todos esses modos de operação. Os parâmetros que definem o modo de operação são descritos na Tabela 1. O parâmetro *al_chroma_mode* define o tipo de decimação usado na codificação. O *al_lcu_size* é um parâmetro que define o tamanho de uma LCU - *Largest Coding Unit*. A LCU é frequentemente chamada de CTU - *Coding Tree Unit*, é a unidade básica de cálculo para o padrão HEVC - *High Efficiency Video Coding*. O tamanho do LCU representa como a imagem é dividida para então ser codificada. Geralmente, quanto maior o tamanho do LCU, maior a eficiência da codificação. O BIT_DEPTH é o parâmetro que define o número de bits usados para codificar os componentes YUV, portanto a arquitetura deve operar em 10 bits ou 8 bits.

3.3.2 Conversor RGBAtoYUVA

O bloco RGBAtoYUVA realiza a conversão do formato RGBA para YUVA. Além disso, ele executa a decimação de pixels, isto é, transforma o formato 4:4:4 em 4:0:0, 4:2:2 ou 4:2:0. A Figura 14 mostra as entradas e saídas deste bloco.

A descrição das entradas e saídas pode ser encontrada na Tabela 2. A entrada *RGBA* consiste em uma linha de 32 bytes da imagem RGBA entregue pela Ponte-RGBA.

Tabela 1 – Modos de operação do OSD.

Parâmetro	Tamanho	Descrição
al_chroma_mode	2 bits	“00”: 4:0:0, “01”: 4:2:0, “10”: 4:2:2
al_max_cu_size	2 bits	“00”: LCU 16x16, “01”: LCU 32x32, “10”: LCU 64x64
BIT_DEPTH	1 bit	’0’: 8b ’1’: 10b

Fonte: Próprio Autor.

Considerando que cada pixel contém 4 componentes (R, G, B e A) e que cada componente é codificado em 8 bits, cada pixel necessita de 4 bytes. Assim, o bloco RGBAtoYUVA recebe 8 pixels por vez. A entrada *mode* define o tipo de decimação que será executada: ”00” corresponde a 4:0:0; “01” corresponde a 4:2:2; “10” corresponde a 4:2:0. As saídas YA e UVA são blocos 8:4, resultado da conversão RGBAtoYUVA.

Tabela 2 – I/Os do RGBAtoYUVA.

Parâmetro	I/O	Tamanho	Descrição
RGBA	I	32 bytes	Entrada de dados no formato RGBA, onde os 32 bytes correspondem a 8 pixels.
mode	I	2 bits	A entrada mode define o tipo de decimação que será utilizado: ı00ȷ − 4 : 0 : 0 ı01ȷ − 4 : 2 : 0 ı10ȷ − 4 : 2 : 2
enable	I	1 bit	Ativa o funcionamento do bloco
reset	I	1 bit	Reset
clk	I	1 bit	Clock
rd_empty	I	1 bit	Verifica se a saída do bridge está empty.
wr_full_YA	I	1 bit	Verifica se a FIFO YA na saída do bloco de conversão está full.
wr_full_UVA	I	1 bit	Verifica se o UVA FIFO na saída do bloco de conversão está full.
YA	O	bloco 8:4	Saída em blocos 8 : 4 da LUMA.
UVA	O	bloc 8:4	Saída em blocos 8 : 4 da CHROMA.

Fonte: Próprio Autor.

A Figura 15 mostra os detalhes da arquitetura do bloco conversor. É um pipeline de 5 estágios. O primeiro estágio do pipeline é a ordenação dos dados feita pelo bloco *DATA ORDER*. No segundo estágio os pixels RGBA são transformados em YUVA. No terceiro estágio o resultado da conversão é comparado com os valores máximo e mínimo permitidos, se esse valor exceder os limites, ele é truncado. No quarto estágio é feita a

decimação dos valores UVA e a transformação das linhas em bloco 8:4 do tipo YA. O último estágio deverá formar blocos 8:4 do tipo UVA.

3.3.3 Alpha Blending

O bloco Alpha Blending é o bloco responsável pela combinação de dois vídeos. A equação matemática que representa a operação realizada neste bloco é mostrada abaixo:

$$video_{blended} = alpha \cdot video_2 + (1 - alpha) \cdot video_1 \quad (3.1)$$

onde $video_{blended}$ é o resultado da operação; $video_1$ é o vídeo de base; $video_2$ é a informação a ser adicionada; alfa é o parâmetro que define a transparência do segundo vídeo em comparação com o primeiro. Em resumo, o bloco Alpha Blending recebe o $video_1$ e $video_2$ na entrada e entrega o $video_{blended}$ na saída. A Figura 16 mostra as principais entradas/saídas do bloco.

A interface deste bloco é feita através de FIFOs - *First in first out*. Nos codificadores desenvolvidos pela empresa, as FIFOs trabalham com blocos de 8x4 pixels, o que não será diferente para o bloco OSD.

3.3.3.1 Arquitetura do Bloco Alpha Blending

A arquitetura do bloco Alpha Blending foi projetada considerando as seguintes características: tamanho da FIFO, ordem dos dados, complexidade e área de superfície. A unidade básica de operação do bloco Alpha Blending é o Pixel Block, este bloco permite combinar os dois pixels utilizando o valor de alfa. Partindo desta unidade de processamento e sabendo que existem 32 pixels para processar cada vez que a FIFO $video_1$ é lida, considerou-se duas soluções para decidir o número de Pixel Blocks que seriam usados.

A primeira possibilidade consiste em processar os dados em único ciclo, ou seja, instanciar 32 unidades do Pixel. Esta solução é mais rápida, mas requer uma área de superfície maior. Além disso, a componente A(alfa), presente no $vídeo_2$, sempre ocupa metade dos blocos 8:4, isso ocorre porque necessita-se do valor alfa para combinar os pixels. Desse modo, seriam necessárias duas palavras da FIFO 2 ($vídeo_2$) para poder processar uma palavra da FIFO 1 ($vídeo_1$). Desse modo, não seria possível processar todos os 32 pixels em um único ciclo.

A segunda possibilidade é usar, por exemplo, um número menor de blocos Pixel Block. Para isso, será necessário um tempo maior para processar os 32 pixels, por outro lado, esta solução requer uma área de superfície menor e considera a taxa de entrada de dados.

A Figura 17 mostra a arquitetura do bloco Alpha Blending, é um pipeline de 3 estágios. O primeiro estágio é um MUX que seleciona a parte do vetor de entrada será processado. Isso ocorre porque não é possível processar todos os 32 pixels em paralelo. O bloco Alpha Blending possui 4 Pixels Block, portanto, é possível processar 4 pixels por vez.

O componente REG salva o resultado de 4 pixels gerados pelo Pixel Block e constrói o bloco de saída 8: 4. O componente Control_XY é o componente responsável por informar se há algo a ser exibido sobre o bloco 8:4. Ele contém em sua entrada as coordenadas (X, Y) para cada região, essas coordenadas são comparadas com as coordenadas (X, Y) do bloco 8:4 atual para definir se há algo a ser exibido sobre aquele bloco.

3.3.4 Visão geral da descrição dos blocos

Os dois blocos principais do OSD são o bloco de conversão e Alpha Blending. Este capítulo apresentou suas principais características e arquitetura. Os outros dois blocos apresentados no início do capítulo, o bloco Color Reverse e Resize, não foram descritos em VHDL. Durante o decorrer do estágio foi necessário fazer uma escolha entre descrever em RTL os dois blocos restantes e validar e integrar o OSD. Como os dois blocos não eram a prioridade do projeto, optou-se por realizar a parte de validação e integração.

3.3.5 Ferramentas utilizadas

Diferentes ferramentas foram utilizadas durante a realização deste estágio. A gestão de arquivos foi feita com o GitLab. O software *Questa Advanced Simulator* da Mentor Graphics foi usada para realizar as simulações dos blocos de hardware. Outra ferramenta chamada PYUV também foi usada para exibir e analisar imagens no formato YUV.

3.4 Validação dos componentes

Após a descrição dos blocos em VHDL, passou-se à validação dos blocos. O método escolhido foi aquele usado pela equipe de hardware para validar seus blocos de hardware. O método envolve o uso de um ambiente de co-simulação de VHDL e C++/C e consiste em escrever testes unitários em C para comparar a saída gerada pelo software e hardware.

No teste unitário, há uma função de referência que faz exatamente a mesma operação que o hardware. Ele também contém a chamada para uma função que envia os dados de entrada para um *wrapper* que faz a interface entre o código C e o código RTL. Isso garante que a função em C e o bloco de hardware contenham as mesmas entradas. Assim,

a função de escrita é chamada, o bloco de hardware realiza seu trabalho. Posteriormente, uma função de leitura é usada para ler as saídas geradas pelo bloco de hardware. O teste unitário, em última análise, usa uma função que compara as saídas de hardware e software e determina se estas são iguais ou não.

3.4.1 Validação do bloco de conversão

Os testes unitários foram usados para validar a operação do bloco de conversão. Vários testes foram feitos para garantir que todo o código escrito em VHDL fosse testado. O bloco de conversão permite escolher vários modos de operação, isso é definido pelos parâmetros do bloco (Tabela 3). A combinação desses dois parâmetros permite obter 6

Tabela 3 – Parâmetros do bloco de conversão

modo	4:0:0, 4:2:0 e 4:2:2.
BIT_DEPTH	8bits, 10 bits

Fonte: Próprio Autor.

modos de operação do bloco de conversão. Estes 6 modos foram testados, variando-se também o vetor RGBA, cujos valores encontram-se entre 0 e 255. Para os testes, foram utilizados valores aleatórios entre 0 e 255, o que permite submeter o bloco a diferentes situações e aumenta a chance de encontrar erros. Os valores mínimo e máximo também foram testados para ver se havia caso de *overflow*.

Além disso, os testes unitários possibilitaram verificar o correto funcionamento do pipeline. Várias situações foram criadas para validar o funcionamento do controle de pipeline, como por exemplo:

- ausência de dados válidos na entrada RGBA.
- saída da FIFO completamente preenchida.

Essas situações geram ”bolhas” no pipeline, ou seja, o estado atual de cada estágio do pipeline é salvo até o momento em que o bloco pode continuar a funcionar (quando os dados válidos chegam na entrada ou a FIFO volta a estar vazia). O bloco de conversão, portanto, foi testado e validado.

3.4.2 Validação do Alpha Blending

Testes unitários também foram usados para validar o bloco Alpha Blending. Como foi feito para o bloco de conversão, diferentes parâmetros foram variados a fim de produzir

diferentes modos de operação para o bloco. Os parâmetros associados a este bloco são:

Tabela 4 – Parâmetros Alpha Blending.

al$_c$*hroma*	'0': dados na entrada são LUMA	'1': dados na entrada são CHROMA
blend	'0': funciona como bypass	'1': combina os dois vídeos
BIT_DEPTH	8 bits	10 bits
`x_begin`, `y_begin`, `x_width`, `y_width`	define os limites das regiões de exibição.	

Fonte: Próprio Autor.

Os diferentes valores possíveis destes parâmetros foram combinados para testar diferentes modos de operação do bloco. Valores aleatórios foram colocados nas entradas $video_1$ e $video_2$. Os valores máximo e mínimo para as componentes Y, U e V também foram testados a fim de encontrar possíveis *overflows*. O funcionamento do pipeline também foi testado nas seguintes situações:

- ausência de dados válidos na entrada de vídeo 1.
- ausência de dados válidos na entrada de vídeo 2.
- saída da FIFO completamente preenchida.

Essas três situações e suas possíveis combinações geram "bolhas" no pipeline, e o controle do pipeline deve ser capaz de gerenciar tais situações. O bloco Alpha Blending foi, portanto, testado e validado.

3.4.3 Validação do bloco OSD e sua integração

Para validação do OSD foi criado um componente *top* onde todos os sub-componentes foram declarados e conectados por meio de FIFOs. Além disso, o bloco foi conectado às duas pontes responsáveis por fornecer os dados de entradas ou pixels das imagens.

3.4.3.1 Conexão do bloco às pontes YUV e RGBA

O bloco OSD é conectado às Pontes YUV e RGBA. A Ponte-YUV é um componente que já existia, este componente é responsável por fornecer a imagem em blocos 8:4 ao codificador. Para isso, a Ponte YUV deve buscar a imagem na memória para então formar os blocos 8:4. A ordem em que os pixels da imagem é dada pelo componente Source Order, também já existente, este componente fornece as coordenadas (X, Y) dos pixels.

O bloco Alpha Blending trabalha com blocos 8:4, então para cada bloco 8:4 que chega na entrada também existem as coordenadas (X, Y) associadas. Essas coordenadas são comparadas com as coordenadas das 8 regiões de exibição para ver se o bloco se encontra em uma das regiões. Além disso, os valores (X, Y) devem ter exatamente a mesma sequência de (X, Y) do Source Order da ponte. Se as sequências forem diferentes, as informações serão exibidas nos locais errados da imagem, por isso foi necessário ter um cuidado redobrado com o funcionamento do bloco Control_RGBA. O Control_RGBA determina se o bloco de 8:4 em questão pertence a alguma região de exibição ou não.

A Ponte-RGBA é responsável por buscar na memória a imagem RGBA e fornecê-la ao bloco de conversão. Esta ponte deve buscar os dados RGBA na mesma sequência da ponte YUV para que os dados cheguem na ordem correta ao bloco Alpha Blending. O Control_RGBA fornece as informações necessárias para que a Ponte-RGBA carregue os pixels na ordem correta.

Uma vez que as conexões externas foram feitas ao bloco OSD, um *Test Bench* foi criado para simular o funcionamento do bloco. O *Test Bench* foi baseado em outros testes já utilizados pela equipe para testar o funcionamento do codificador, um pulso chamado al_start é enviado para um componente onde as pontes, bloco OSD e outros componentes são declarados. O pulso indica que os componentes devem começar a trabalhar e quando finalizam seus cálculos, a saída al_run passa a 0. Para que o resultado do bloco OSD seja comparado com o resultado do software, a saída YUV é escrita em um arquivo de saída.

Vários testes foram realizados para validar o funcionamento do bloco OSD, os seguintes pontos foram validados:

- a conexão das pontes com o bloco OSD.
- a operação do bloco Control_RGBA e sua conexão.
- a sequência correta de dados que chegam ao bloco Alpha Blending e ao bloco de conversão.

No entanto, as saídas de hardware e software ainda apresentaram algumas diferenças para certos testes. A validação do bloco OSD não foi finalizada completamente.

3.5 Síntese ASIC

Este capítulo apresenta uma análise dos resultados gerados pela realização de uma síntese ASIC do bloco OSD. A síntese é uma etapa essencial do projeto, pois apresenta a tradução do circuito descrito em VHDL em células padrão. As células padrão são parte de

uma biblioteca de células da tecnologia escolhida. Durante a síntese, o software gera uma lista de células necessárias para traduzir o circuito digital. Esta fase permite saber se o circuito é sintetizável, uma vez que é possível obter um circuito digital que funciona na simulação, mas que não é sintetizável.

A síntese gera diversos relatórios que permitem que o circuito seja avaliado sob diferentes perspectivas. Um dos relatórios, por exemplo, dá uma visão geral do circuito digital em relação à superfície que será necessária para imprimir o circuito, isso é possível porque cada célula possui características físicas associadas à tecnologia escolhida. Assim, é possível calcular, por exemplo, a área de silício necessária para projetar o circuito. A tecnologia utilizada foi a CLN16_FC de 16nm a uma frequência de 500 MHz. O software usado para realizar a síntese foi o *Design Compiler* da Synopsys.

3.5.1 Alpha Blending

A Figura 18 mostra um gráfico com a distribuição da área do bloco Alpha Blending. A unidade de área utilizada é µm². Em geral, os blocos possuem valores de área semelhantes, exceto para o bloco Control_XY que possui uma área um pouco maior que os outros. Isso se deve ao fato de que este bloco faz várias comparações para saber se as coordenadas (X, Y) pertencem a uma das 8 regiões, isto é, há uma parte combinatória considerável. De fato, o resultado da síntese mostra que 696µm² do total de 776 µm² é combinatória.

3.5.2 Bloco de conversão

A Figura 19 mostra a distribuição da área do bloco de conversão. O bloco de conversão tem uma área total de 6726 µm², o dobro do tamanho do Alpha Blending. Isso pode ser justificado pelo fato do bloco de conversão ser mais complexo do que o Alpha Blending em termos de cálculo e também pela presença de elementos de memória. O bloco que ocupa mais espaço é o DATA ORDER porque armazena palavras de 32 bytes.

3.5.3 Visão geral da síntese

De modo geral, o bloco OSD não ocupa muito espaço, o que é uma vantagem, pois o OSD será uma espécie de recurso adicional que o codificador oferecerá. Os clientes da Allegro serão, portanto, capazes de adicioná-lo como um *plus* ao codificador sem aumentar significativamente sua área de superfície.

A síntese mostrou que o circuito não apresenta um caminho crítico na frequência utilizada, portanto, não há *slack* negativo. Além disso, o design não apresentou *latches*.

Figura 8 – Decimação.

(a) 4:2:2

(b) 4:2:0.

(c) 4:0:0.

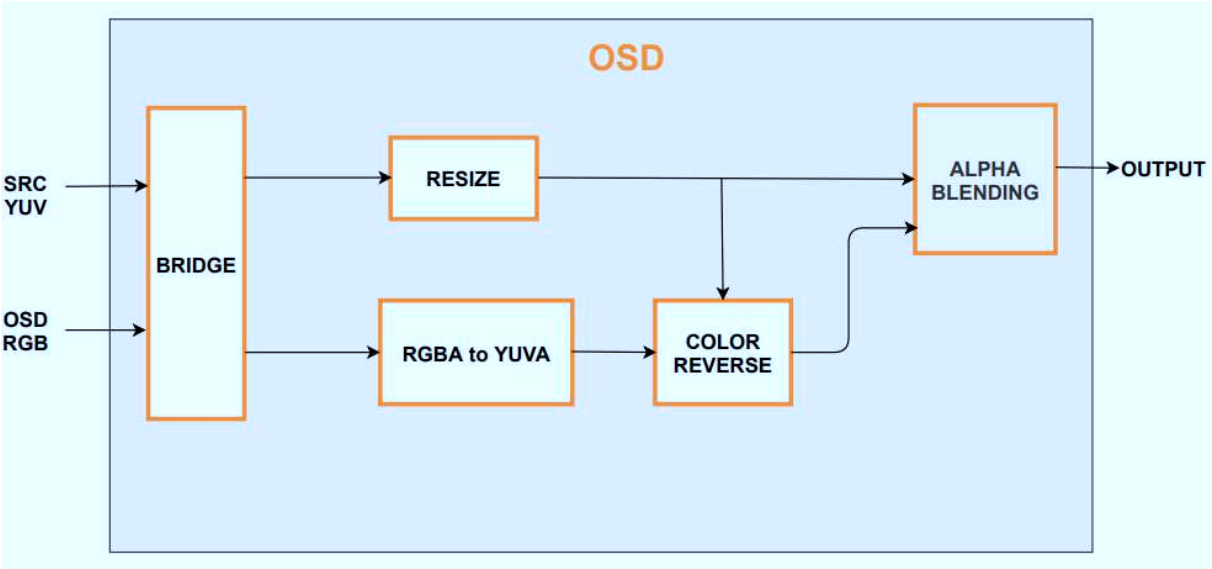
(d) RGB.

Fonte: Próprio Autor.

Figura 9 – Resultado do algoritmo do Alpha Blending.

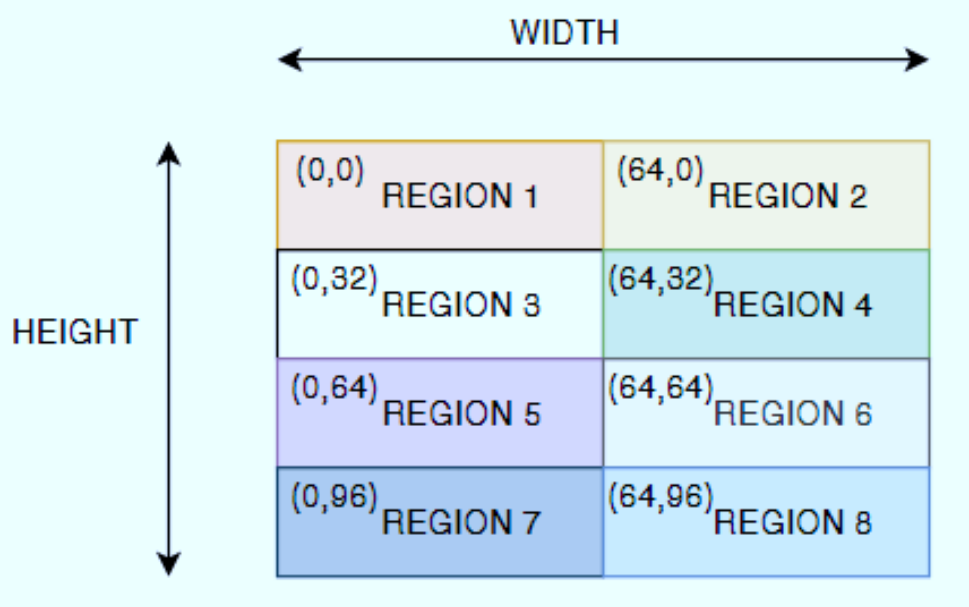
Fonte: Próprio Autor.

Figura 10 – Arquitetura global do blobo OSD.



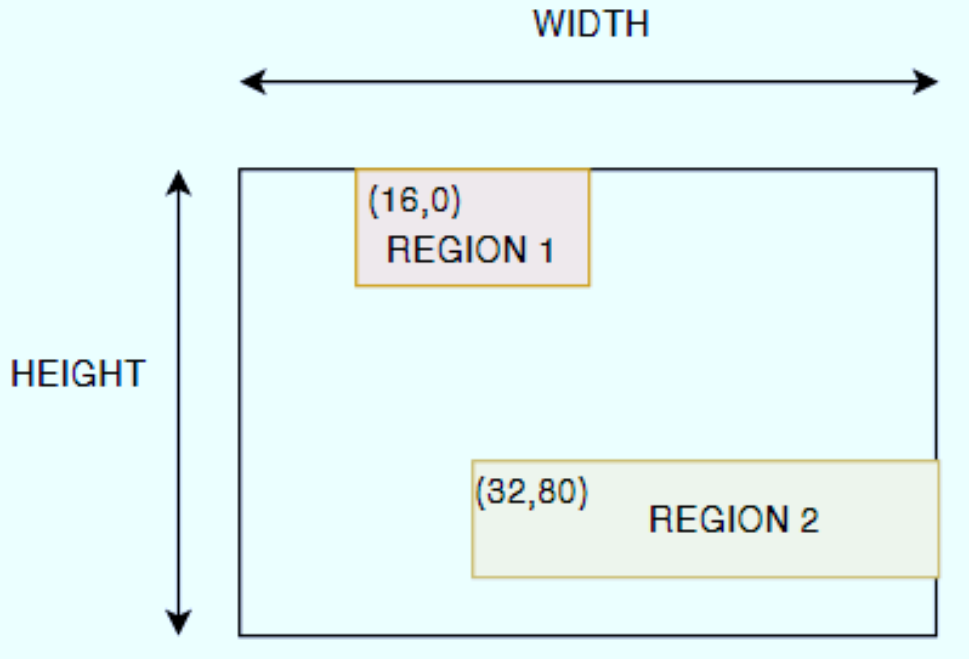
Fonte: Próprio Autor.

Figura 11 – 8 regiões de exibição.



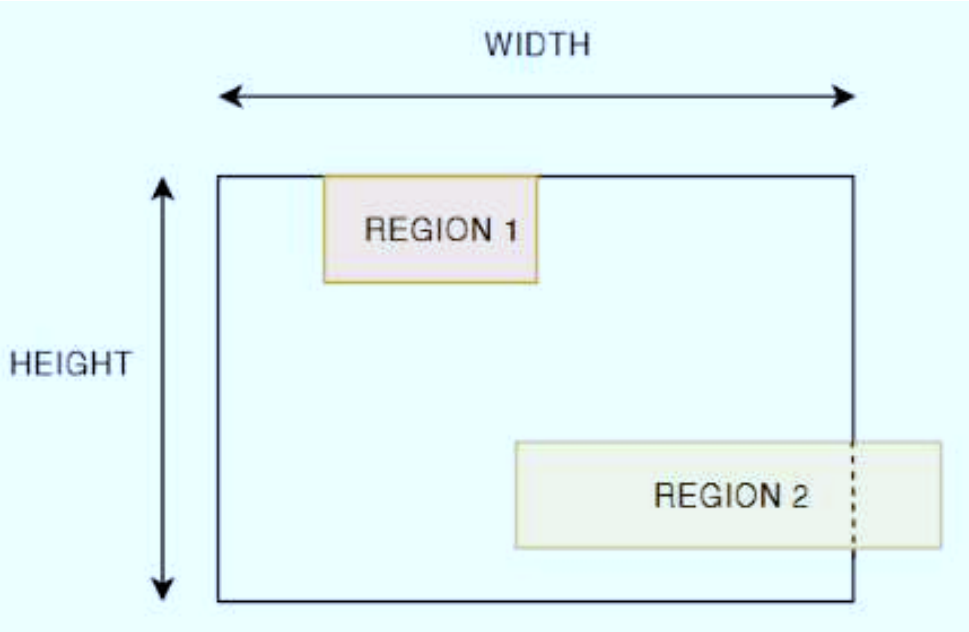
Fonte: Próprio Autor.

Figura 12 – 2 regiões de exibição.



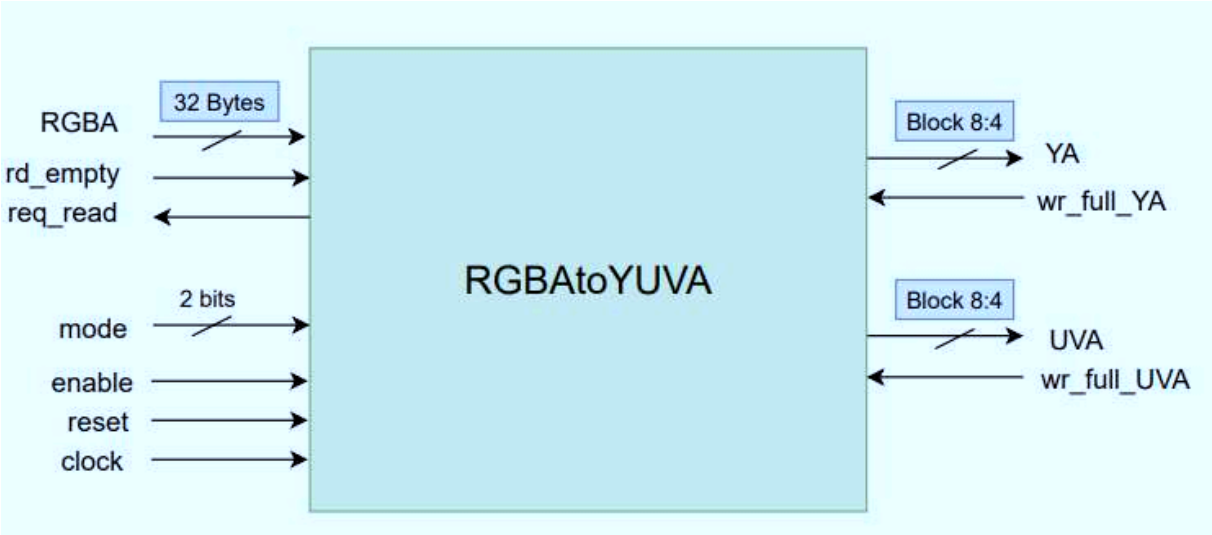
Fonte: Próprio Autor.

Figura 13 – Região maior que a imagem.



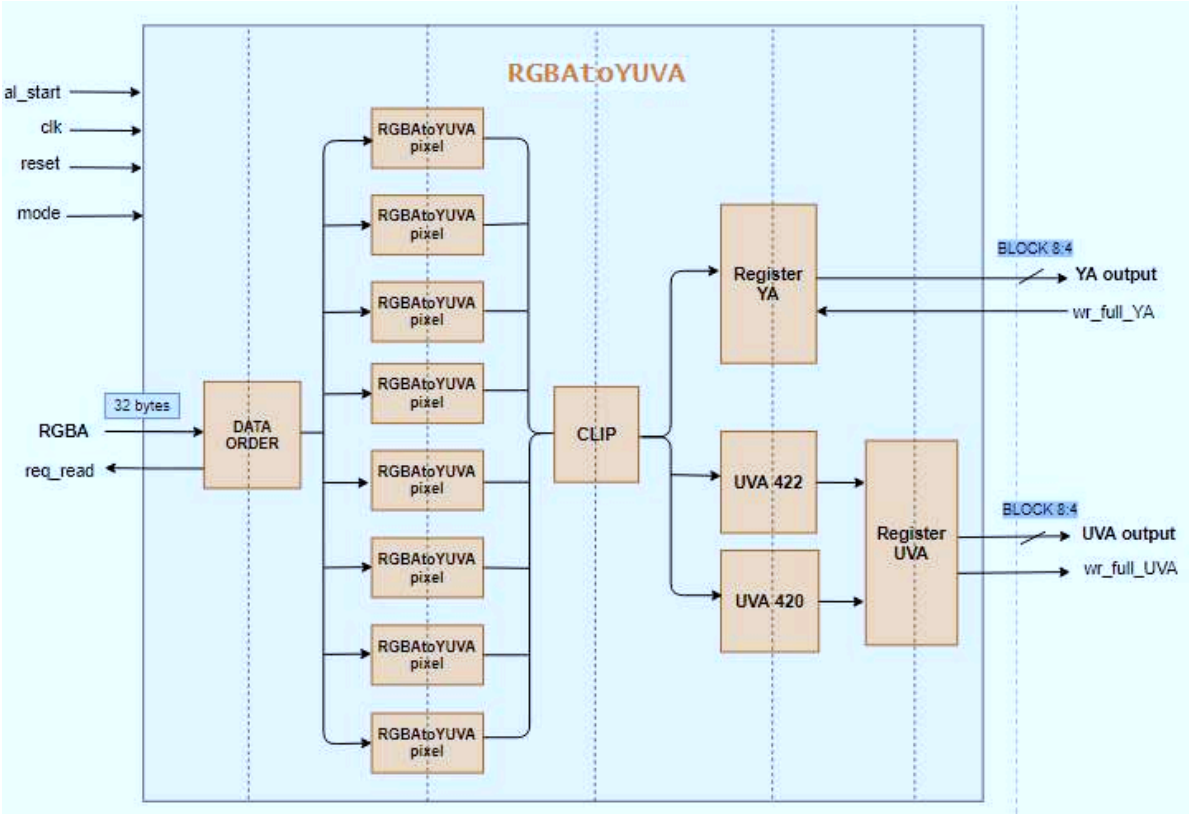
Fonte: Próprio Autor.

Figura 14 – Bloco de conversão.



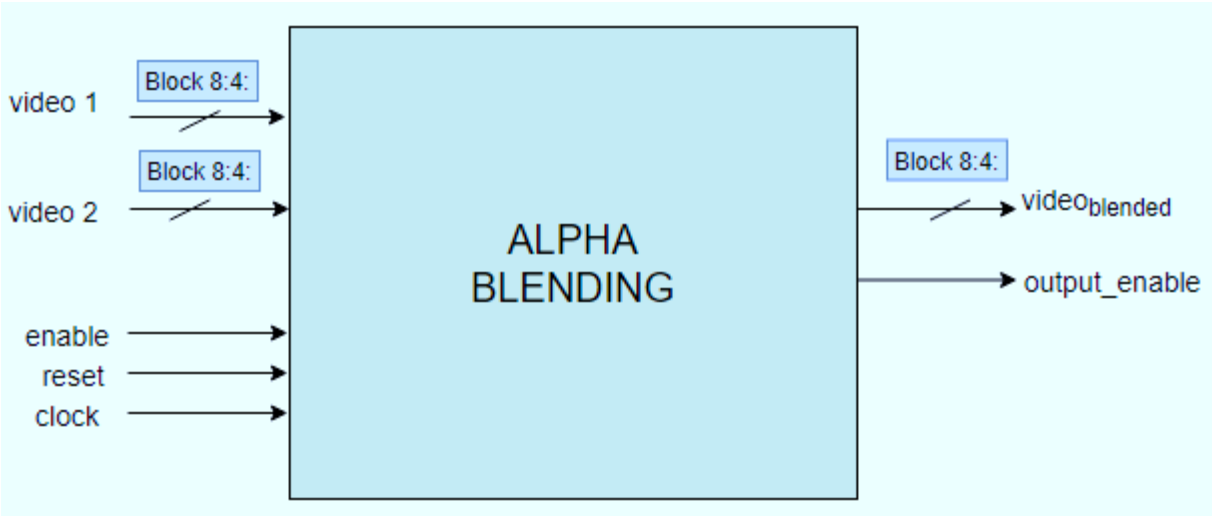
Fonte: Próprio Autor.

Figura 15 – Arquitetura do bloco de conversão.



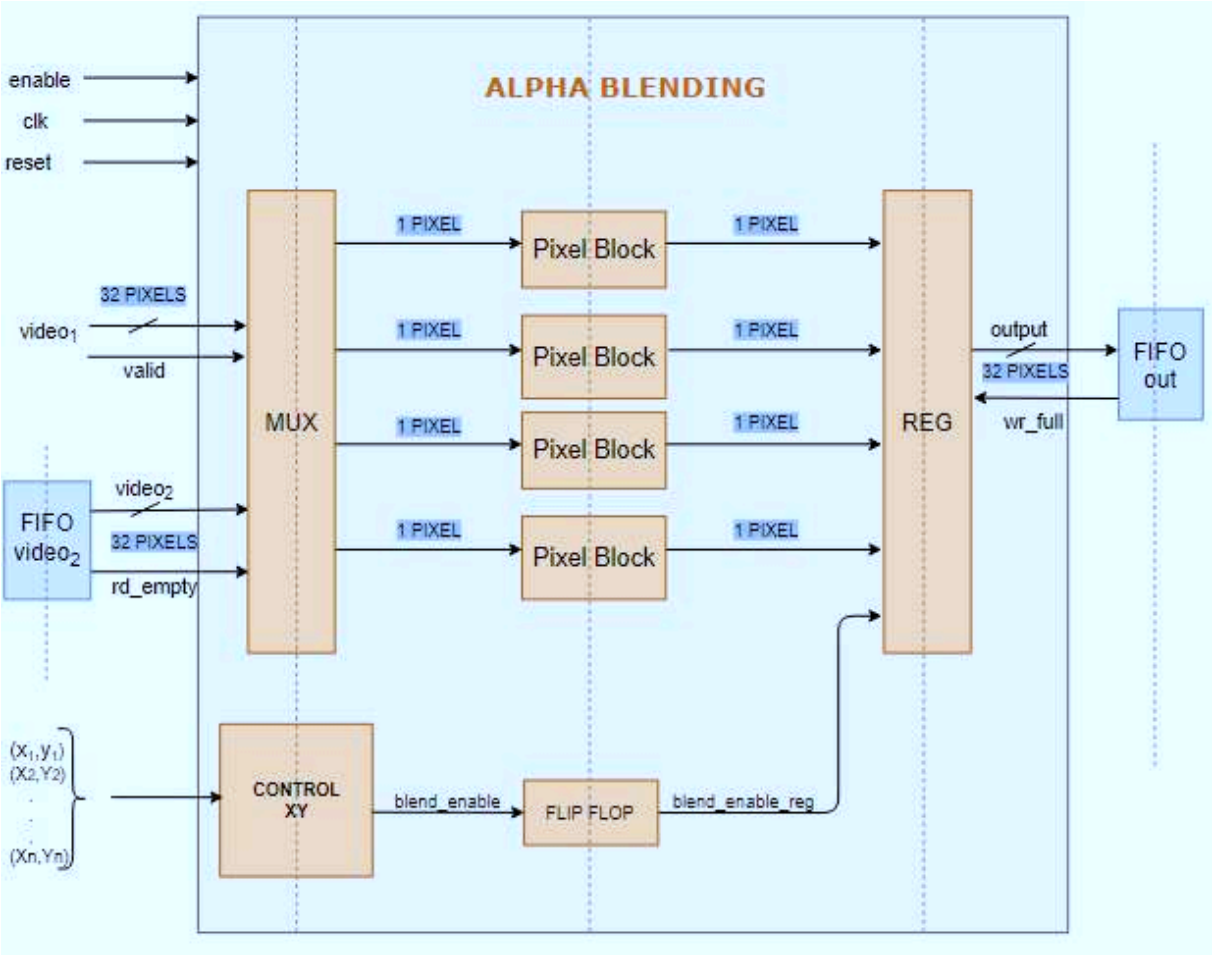
Fonte: Próprio Autor.

Figura 16 – Bloco Alpha Blending.



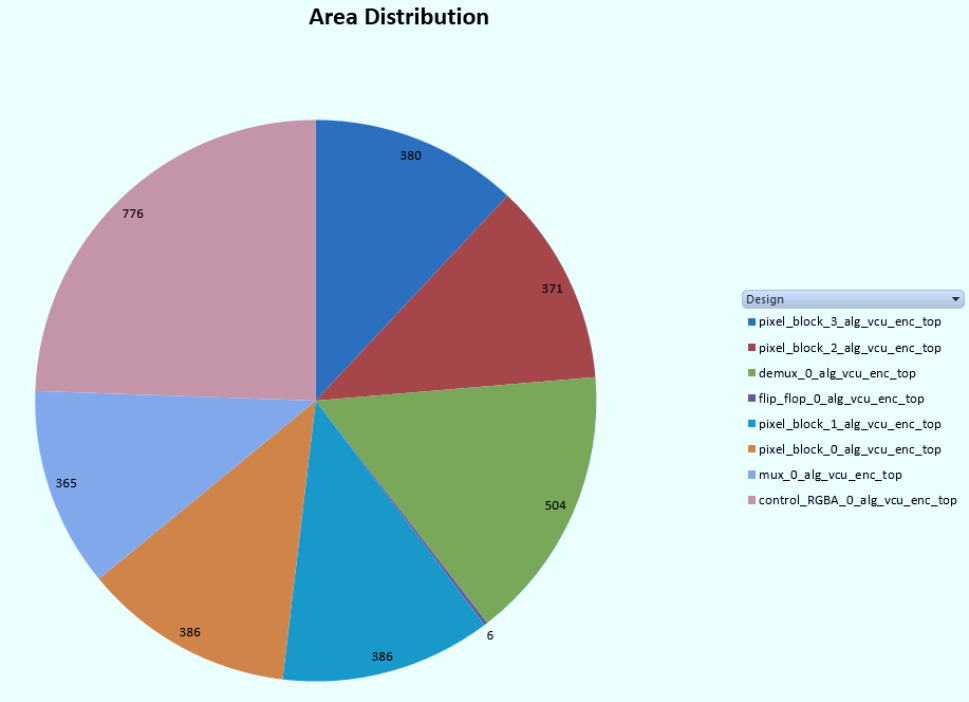
Fonte: Próprio Autor.

Figura 17 – Arquitetura do Alpha Blending.



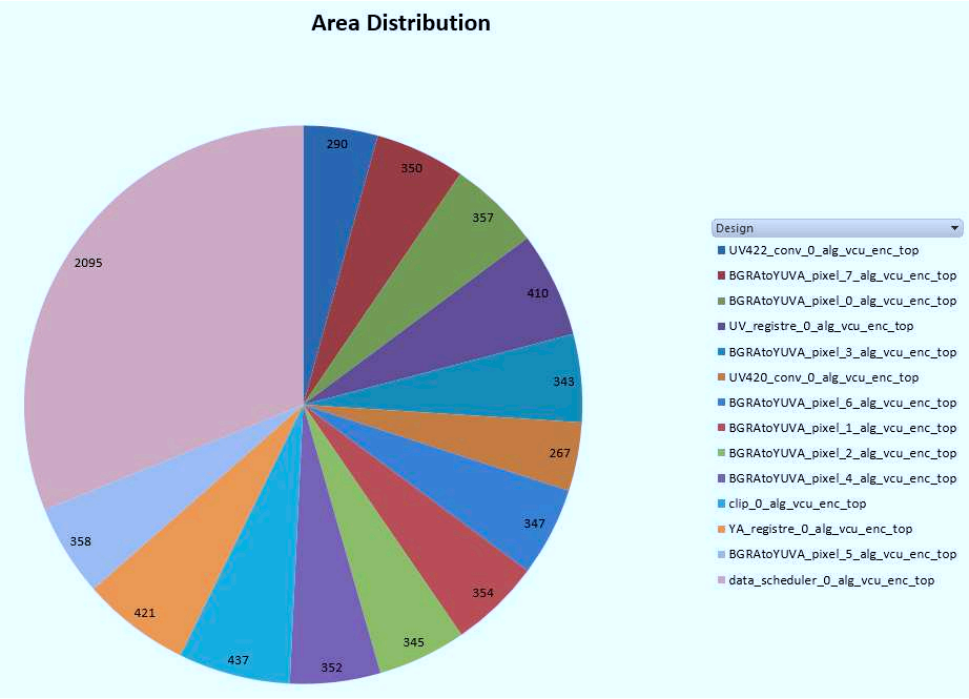
Fonte: Próprio Autor.

Figura 18 – Distribuição de área do bloco Alpha Blending.



Fonte: Próprio Autor.

Figura 19 – Distribuição de área do bloco de conversão.



Fonte: Próprio Autor.

4 Conclusões

O objetivo deste estágio foi propor uma arquitetura para o bloco *On-Screen Display* e descrevê-lo em RTL. Para tanto, foi feito um estudo do processo de conversão de RGBA para YUVA e da operação *alpha blending*. Essas operações foram escritas em C para se obter um modelo de referência. Em seguida, foi proposta uma arquitetura para o OSD, tendo em vista as especificações escolhidas. Todos os blocos *hardware* foram descritos em VHDL, com exceção do bloco Color Reverse. Este bloco tinha menor prioridade quanto aos outros, e devido ao tempo decidiu-se não implementá-lo.

A validação dos blocos de hardware foi dividida em duas etapas: validação de cada sub-componente através de testes unitários; validação do componente OSD. A primeira etapa foi concluída e os blocos foram validados. No que se refere a segunda etapa, a comunicação e transmissão de dados do OSD com as pontes YUV e RGBA foi validada, o funcionamento do bloco OSD foi validado até a saída do bloco RGBAtoYUV. No entanto, a saída final do bloco, isto é, após o Alpha Blending ainda apresenta diferenças quando comparada com o resultado de referência gerado pelo código C.

A síntese do bloco OSD mostrou que ele possui uma pequena área de superfície quando comparado à área do codificador. Este é um bom resultado, pois o OSD será vendido como um recurso adicional do codificador, assim o cliente poderá adicionar este recurso sem aumentar muito a área de superfície total do codificador. A síntese também mostrou que todos os componentes eram sintetizáveis e que não havia slack negativo.

Por fim, a realização deste estágio permitiu colocar em prática os conhecimentos aprendidos ao longo destes anos de formação. Além disso, foi uma grande oportunidade de trabalhar no desenvolvimento de um projeto com aplicações reais. Durante o estágio foi possível desenvolver habilidades como comunicação, relacionamentos interpessoais, autonomia e organização. O estágio também proporcionou a aprendizagem e prática de ferramentas como o Gitlab, *Questa Advanced Simulator*, programação em C e VHDL.

Referências

ITU, T. S. S. O. *Advanced video coding for generic audiovisual services*. [S.l.]: H.264, 2017. Citado na página 6.

ITU, T. S. S. O. *High efficiency video coding*. [S.l.]: H.265, 2018. Citado na página 6.