FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Implementação em FPGA de um conversor HDMI para transmissão em série de alta velocidade

Ana Marisa Oliveira Barbosa

Versão de trabalho

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador: Prof. Doutor João Paulo de Castro Canas Ferreira

Co-orientador: Prof. Dr. Henrique Manuel de Castro Faria Salgado

Supervisor Externo: Dr. Luís Manuel de Sousa Pessoa

6 de Junho de 2017

Resumo

A sociedade atual depende cada vez mais dos serviços de comunicações, exigindo melhores ligações e mais rápidas, prevendo-se num futuro próximo a necessidade de ligações na ordem das centenas de Gb/s. O projeto iBrow que está a ser desenvolvido por vários parceiros, incluindo o INES-TEC, e vem propor uma nova exploração do espetro de frequências permitindo assim comunicações de alta velocidade. Este projeto passa por propor uma metodologia que permite a manufaturação de transcetores de baixo custo capazes de atingir grandes débitos de transmissão.

A interface HDMI é cada vez mais usada em todos os tipos de ambientes: tanto empresariais como domésticos. Por esse motivo acaba por ser uma boa interface para testar os transcetores que estão a ser desenvolvidos. E por isso nesta dissertação é proposto um projeto que visa testar os transcetores desenvolvidos pelo projeto, desenvolvendo e implementando uma arquitetura em FPGA capaz de suportar sinais provenientes de uma fonte HDMI, serializá-los e ainda enviá-los através das saídas de alta velocidade para que possam ser de serguida enviados stravés dos transcetores do projeto iBrow, no sentido de os testar. A arquitetura é capaz de suportar ainda o processo inverso, isto é, receber os dados provenientes dos transcetores do projeto iBrow através das entradas de alta velocidade existentes na FPGA a ser utilizada e voltá-los a enviar para o disposituivo final HDMI.

Inicialmente foi desenvolvida e implementada uma arquitetura em FPGA, recorrendo a *hardware* capaz de descodificar sinais HDMI, que é capaz de transmitir entre dois dispositivos sinais de imagem tanto em formato RGB como YCbCr, e som em formato I2S.

–> Explicar arquitetura da parte HDMI

–> Explicar arquitetura da parte dos GTX

Abstract

Now I have to write it in English

Agradecimentos

Quero agradecer antes de tudo aos meus pais, Fernado e Conceição, não só porque me puserem neste mundo e me deram a melhor educação que me poderiam ter dado, mas também porque me permitiram vir estudar para a melhor Faculdade de Engenharia do país e seguir o melhor caminho possivel, ainda que tal viesse impor algumas restrições e cuidados. Também lhes quero agradecer porque ao longo destes 5 anos conseguiram aguentar as minhas crises exestencias, os meus choros quando as coisas não corriam tão bem, e o foi também graças a isso que consegui continuar a ter força e lutar pelo que sempre quis. Agradeço ainda a minha irmã Elisabete, porque por muito chata e irritante que seja sempre me ajudou não só a nivel pessoal mas também quando era preciso corrijir este tipo de documentos importantes, aliás, ela será a primeira a ler este documento e a queixar-se de que tenho muitos erros e não sei colocar virgulas e acentos. Obrigada por ter paciência para corrigir estes documentos super compridos cheios de termos técnicos e projetos que ela não faz ideia para que servem. Para além dela agradeço também ao meu cunhado Marco, porque em conjunto conseguiram colocar no mundo os meus dois sobrinhos queridos: o Dinis e a Sofia. Ambos foram sempre o meu refugio ao fim de semana, mesmo quando as coisas não corriam bem sabia que no fim de semana chegaria a casa e eles estariam sempre lá. A todo o resto da minha familia enorme, avós, tios, tias, primos, primas, primos emprestados e primas emprestadas, um agradecimento muito especial porque sempre me suportaram nesta minha caminhada na faculdade tornando as coisas mais fáceis de suportar durante a semana.

Marisa Oliveira

'The journey is the reward'

Steve Jobs

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

ARC	*Audio Return Channel*
BER	*Bit Error Rate*
CDR	*Clock Data Recovery*
CEC	*Consumer Electronics Control*
CMU	*Clock Multiplier Unit*
DDC	*Display Data Channel*
DFE	*Decision Feedback Equalizer*
DVI	*Digital Video Interface*
EDID	*Extended Display Identification Channel*
EEPROM	*Electrically erasable programmable read-only memory*
EIA/CEA	*Electronic Industry Alliance/ Consumer Eletronics Association*
FEC	*Forward Error Correction*
FEUP	Faculdade de Engenharia da Universidade do Porto
FIFO	*First-In First-Out*
FMC	*FPGA Mezzanine Cards*
FPGA	*Field-Programmable Gate Array*
HDCP	*High-bandwith Digital Content Protection*
HDMI	*High Definition Multimedia Interface*
HDTV	*High-Definition television*
HEC	*HDMI Ethernet Channel*
HPC	*High Pin Count*
iBrow	*Innovative ultra-BROadband ubiquitous Wireless communications through terahertz transceivers*
INESC-TEC	Instituto de Nacional de Engenharia de Sistema e Computadores Tecnologias e Ciências
LPCM	*Linear Pulse Code Modulation*
MIMO	*Multiple Input Multiple Output*
PCS	*Physical Coding Sublayer*
PISO	*Parallel-Input Serial-Output*
PLL	*Phase-Locked Loop*
PMA	*Physical Medium Attachment Sublayer*
PRBS	*Pseudo Random Bit Sequence*
QAM	*Quadrature Amplitude Modulation*
RTD	*Resonant Tunneling Diode*
SIPO	*Serial-Input Parallel-Output*
TMDS	*Transition- Minimized Differential Signaling*
VESA	*Video Electronics Standards Association*
RGB	*Red Green Blue*
I2S	*Inter-IC Sound*

Capítulo 1

Introdução

Este trabalho surge no contexto da unidade curricular Preparação para a Dissertação, pertencente ao plano de estudos do Mestrado Integrado em Engenharia Eletrotécnica e de Computadores, sendo que esta mesma unidade curricular dá início ao trabalho a ser realizado no semestre seguinte na unidade curricular Dissertação.

1.1 Enquadramento Geral

Ao longo das últimas décadas a sociedade tem vindo a tornar-se cada vez mais dependente das comunicações com e sem fios, não só em termos empresariais, mas também em termos pessoais. Esta tendência tem vindo a vincar-se recentemente, com a crescente utilização de tablets e smartphones, tornando os recursos atuais incapazes de responder a tal procura. E cada vez esta exigência irá aumentar prevendo-se a necessidade de ligações na ordem das centenas de Gb/s no ano de 2020, essencialmente para comunicações a curta distância. Daqui conclui-se que os recursos que existem atualmente não são capazes de responder a esta necessidade crescente de comunicações de alto débito, e como tal é necessário urgentemente o desenvolvimento de tecnologias não só capazes de satisfazer esta procura, mas ao mesmo tempo que o façam de forma eficiente em termos energéticos e financeiros. Neste contexto enquadra-se o projeto iBrow (Innovative ultra-BROadband ubiquitous Wireless communications through terahertz transceivers), o qual está a ser parcialmente desenvolvido pela equipa de investigação de tecnologias óticas e eletrónicas do INESC-TEC, que vem responder a esta necessidade de uma forma eficiente.

Este projeto vem propor o desenvolvimento de uma tecnologia capaz de responder a esta necessidade de comunicações de alto débito através de uma utilização eficaz do espetro de frequências, promovendo a utilização de bandas de frequência mais altas, desde 60 GHz até 1 THz. Para além disso vem também propor uma metodologia, que pela primeira vez permite um baixo custo de manufaturação de transcetores capazes de atingir altos débitos de transmissão para que possam ser perfeitamente integrados em redes de comunicações ótica de grande velocidade.

Toda esta crescente de consumo por parte dos utilizadores de novas e cada vez mais tecnologias não se verifica apenas na necessidade de aumento de largura de banda para as comunicações, mas existe também uma necessidade extrema da existência de interfaces digitais de vídeo e som que não só sejam capazes de fazer chegar ao utilizador sinais de alto débito, mas que ao mesmo tempo o façam de maneira segura no sentido de proteger eventuais cópias não autorizadas. Assim sendo, o desenvolvimento de um conversor HDMI (High Definition Multimedia Interface) de alto débito enquadra-se perfeitamente nesta necessidade sendo que é a interface de vídeo e áudio standard e que implementa o protocolo HDCP (High-bandwith Digital Content Protection) que protege a reprodução de sinais em dispositivos não autorizados.

Existem várias interfaces digitais que implementam o protocolo referido anteriormente, entre elas destacam-se DisplayPort, DVI e HDMI. No entanto, devido ao tremendo sucesso que a interface HDMI obteve, de acordo com In-Stat referido em [9] foram vendidos 5 milhões de exemplares em 2004, 17.4 milhões em 2005, 63 milhões em 2006 e 143 milhões em 2007, tornou-se a interface standard para HDTV (High-Definition television), substituindo a interface DVI (Digital Visual Interface). Relativamente à interface DisplayPort, esta é utilizada em vários equipamentos, mas principalmente no sector dos computadores e vem complementar o HDMI. Contudo, comparando as duas interfaces previamente referidas, o HDMI tem algumas vantagens no que toca à capacidade de transmitir sinais CEC (Consumer Electronics Control) e a compatibilidade elétrica com o DVI. Mas o mais importante na realidade baseia-se na capacidade de transmissão dos sinais, sendo que o HDMI é capaz de fazer transmitir o sinal na sua largura de banda completa até 10 metros, enquanto que a DisplayPort apenas o consegue transmitir até 3 metros.

Através da implementação dos objetivos propostos pela dissertação será possível implementar um conversor HDMI capaz de fazer transmitir sinais de alto débito, tornando mais eficiente este tipo de comunicações e ao mesmo tempo fazendo-o de forma segura, protegendo as cópias e reproduções não autorizadas dos sinais transmitidos.

1.2 Motivação

Com a explosão que se fez sentir nos últimos anos na utilização do espetro de frequências, verifica-se que é necessário tornar a sua utilização mais eficiente no sentido de conseguir satisfazer a necessidade da sociedade de comunicar quase sem limites em termos de velocidade da comunicação em si. Promove-se assim uma nova abordagem do espetro de frequências, de maneira a que se possa utilizá-lo de uma forma mais eficaz. Ao longos dos anos tem-se vindo a verificar melhorias no que toca à eficiência espetral através do desenvolvimento e aplicação de algumas técnicas, tal como referido em [10], como por exemplo o QAM (Quadrature Amplitude Modulation) para modulação do sinal e também técnicas MIMO (Multiple Input Multiple Output) nas entradas e saídas do sistema de comunicação. Verificou-se que o aproveitamento do espetro de facto melhorou, no entanto, estas técnicas

não são suficientes para se conseguir atingir um débito de algumas dezenas ou centena de Gb/s. Assim sendo, a solução passa por promover a utilização de bandas de frequência mais altas, contrariamente ao que se fez no passado.

Por definição, considera-se a banda de ondas mm entre 60 a 100 GHz e a banda THz entre 100 GHz a 1 THz. Estas bandas do espetro de frequências são bandas cuja utilização no passado foi pouca ou até mesmo nenhuma, isto porque para conseguir explorar estas bandas são necessários componentes adequados à operação nas mesmas. Relativamente a banda de ondas mm, apesar de nos últimos anos terem sidos desenvolvidas e aplicadas técnicas que melhoram a eficiência espetral desta região, tal como referido anteriormente, a escassez da largura de banda limita o débito da ligação. Em [10] são referidas implementações realizadas no passado que conseguiram alcançar débitos até 100 GHz em ligações sem fios a uma distância de 1 metro com BER = 1 x 10-3, recorrendo também à utilização de mais de um transmissor e recetor. Apesar de inovadores estes valores revelam-se insuficientes para o que se pretende alcançar.

Quanto à região do espetro que corresponde a uma frequência superior a 10 THz, apesar da grande largura de banda disponível nesta região, existem várias limitações para a comunicação sem fios referidas em [11]. Destaca-se o facto do baixo balanço de potência possível para a transmissão devido aos limites de segurança dos olhos, os impactos atmosféricos na propagação do sinal (chuva, pó e poluição) e ainda o impacto da falta de alinhamento entre transmissores e recetores. Estas são algumas das razões que limitam a comunicação sem fios para frequências superiores a 10 THz.

Assim sendo, segundo [11], torna-se evidente que a banda do espetro com maior potencial para a comunicação sem fios é a banda entre 100 GHz e 1 THz, uma vez que não só oferece uma largura de banda bastante maior (desde GHz até alguns THz) comparativamente a outra bandas, mas também é uma região do espetro que não sofre muito devido às más condições atmosféricas. Para além disso, a utilização destas bandas de frequência altas acabará por aliviar o espetro relativamente à sua escassez e às suas limitações de capacidade.

Tendo em conta esta nova abordagem do espetro, o projeto iBrow tem vindo a desenvolver metodologias que permitem a manufaturação de transcetores para operar a estas frequências de baixo custo, mas que ao mesmo tempo são capazes de atingir altos débitos, para que desta maneira sejam integrados em redes de comunicação com e sem fios de grande velocidade. Os transcetores de baixo custo propostos pelo projeto passam por utilizar díodos ressonantes de efeito túnel (RTD) com formatos de modulação simples e com interligação com fibra ótica. Assim será possível satisfazer as necessidades previstas para 2020 de forma eficaz tanto em termos energéticos como financeiros.

Para que se possa demonstrar o potencial desta tecnologia proposta pelo iBrow, vai-se recorrer à transmissão de vídeo em alta definição descomprimido através destes mesmos dispositivos propostos pelo projeto. Assim sendo, para efetuar a transmissão será utilizada a interface HDMI, que fará transmitir um sinal de alto débito para de seguida o mesmo sinal

ser transmitido pelos transcetores propostos pelo projeto iBrow. Esta transmissão terá de ser realizada em série visto que estes mesmos transcetores apenas suportam transmissão de dados em série, uma vez que esta é a maneira mais eficaz.

O HDMI é uma interface digital que transmite vídeo não comprimido e áudio que poderá ou não estar comprimido. Esta interface implementa vários protocolos entre quais se destaca o protocolo HDCP pois é o responsável pela prevenção de reproduções não autorizadas dos sinais a transmitir, o que é bastante importante hoje em dia dado os inúmeros consumidores que conseguem fazer cópias ilegais. Este protocolo faz uma verificação inicial antes de transmitir os dados encriptados no sentido de perceber se o dispositivo de destino é efetivamente um dispositivo autorizado para a reprodução de sinal. Esta é ainda uma interface que consegue transmitir sinais de alta definição e é ainda compatível com o DVI. Hoje em dia, esta é a interface standard para HDTVs e tem diversas aplicações tais como câmaras digitais, discos Blu-ray e leitores de DVD de alta definição, computadores pessoais, tablets e smartphones.

Em suma, esta implementação tornar-se bastante útil, uma vez que é capaz de abranger um vasto nível de aplicações, acessíveis a todos os utilizadores, tanto em ambientes empresariais como pessoais.

1.3 Descrição do Problema e Objetivos

Este projeto tem como principal objetivo a implementação de uma arquitetura que permita a receção de dados HDMI, o seu tratamento e serialização para o seu posterior envio em alta velocidade. Para além disto, a arquitetura deve também voltar a receber os dados em série, fazer a sua conversão para dados em paralelo e voltar a enviar para um dispositivo HDMI final.

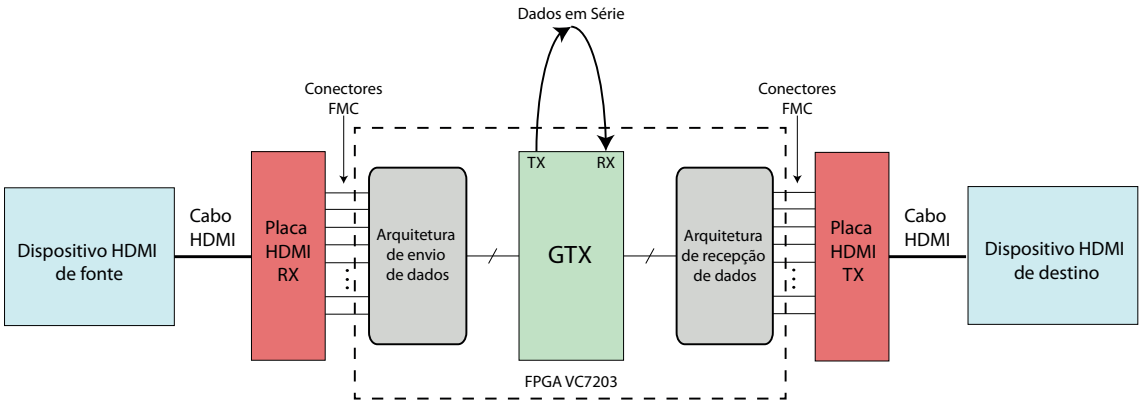

Figura 1.1: Diagrama geral do problema proposto

O projeto faz uso de uma FPGA VC7203 (*Virtex-7*) que possibilita a implementação de uma arquitetura adquada e ao mesmo tempo possui entradas e saídas de alta velocidade

para que se possa testar a arquitetura desenvolvida. Para além disso, serão também utilizadas umas placas HDMI que permitem enviar os dados em paralelo de um fonte HDMI para a FPGA VC7203 através dos conectores FMC, e depois fazer o processo inverso, ou seja, enviar os dados em série para a placa para que a placa os transmita para o dispositivo final HDMI.

A figura 1.1 da página 4 ilustra um diagrama geral do projeto a ser realizado. No sentido de simplificar o desenvolvimento do mesmo, este foi dividido em 2 partes:

1. Concepção e desenvolvimento de arquiteturas que permitam comunicação entre dispositivos HDMI.

2. Concepção e desenvolvimento de arquiteturas que permitem serialização de dados e deserialização dos mesmos.

A primeira parte do projeto consiste em obter comunicações entre dois dispositivos HDMI utilizando para tal as placas HDMI disponiveis. A figura 1.2 da página 5 ilustra um diagrama que descreve em que consiste o problema a ser resolvido na primeira parte do projeto.

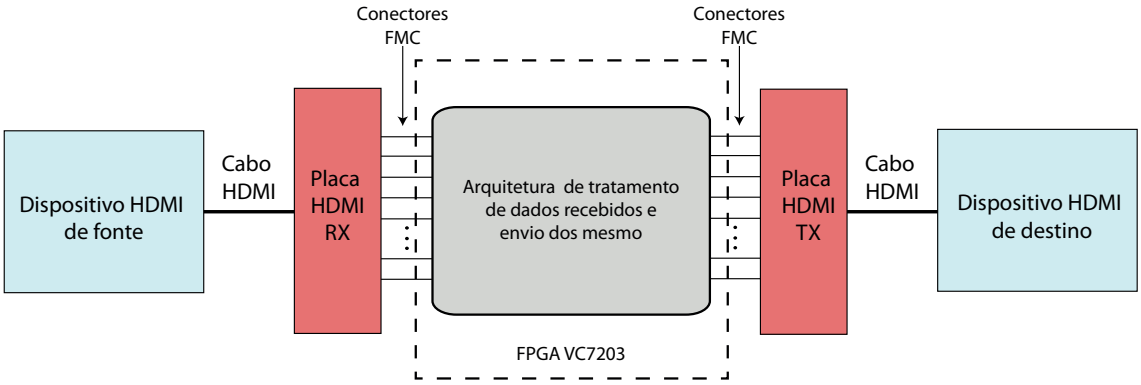

Figura 1.2: Diagrama geral da primeira parte problema

À placa HDMI recetora é ligado um sinal externo via cabo HDMI de uma fonte HDMI e de seguida essa mesma placa enviará para a FPGA através dos conectores FMC os sinais de video a serem transmitidos. Na FPGA VC7203 é desenvolvida uma arquitetura que permita o tratamento dos dados provenientes dos conectores FMC e de seguida, esses mesmo dados, são enviados para a placa HDMI transmissora de maneira a que esta possa envia-los para o dispositivo HDMI de destino.

A segunda parte do projeto consiste em desenvolver uma arquitura na FPGA que permita serializar os dados recebidos, envia-los através das entradas de alta velocidade da mesma e voltar a recebe-los. A figura 1.3 da página 6 ilustra o diagrama do trabalho a ser desenvolvido nesta fase.

Tal como ilustra a figura 1.3, serão recebidos dados em paralelo e deve ser desenvolvida uma arquitetura que organize esses mesmos dados em tramas, e de seguida os envie para

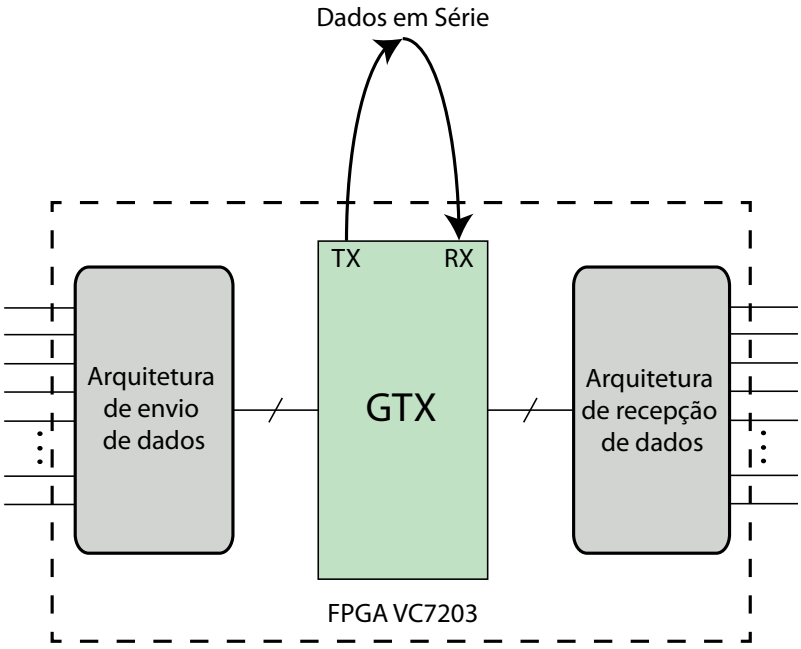

Figura 1.3: Diagrama geral da segunda parte do problema

os transcetores da FPGA que automaticamente fazem a serialização dos dados. Depois os dados em série são transmitidos por um cabo físico, e recebidos nos transcetores onde serão de-serializados novamente. Por fim, as tramas serão recebidas e processadas de modo a obter-se na saída os dados em paralelo tal como recebidos na entrada. Assim que as duas partes do projeto estejam concluídas obtem-se o objetivo final.

1.4 Estrutura da Dissertação

Esta dissertação está organizada em vários capitulos que vão desde uma revisão bibliográfica do problema até à descrição da resposta ao mesmo. No capitulo 2 é feita uma revisão bibliográfica sobre os diversos aspectos que o problema apresenta, apresentando também considerações de diversos autores sobre tal.

No capitulo 3 é descrito toda a concepção e desenvolvimento das arquiteturas referentes à primeira parte do trabalho apresentado em 1.3. São descritas todas as arquiteturas desenvolvidas, as principais dificuldades encontradas no seu desenvolvimento e ainda os resultados obtidos.

O capitulo 4 aborda todas as questões relativas à transmissão em série. São apresentas as arquiteturas desenvolvidas para se obter uma ligação em série, todos os cuidados necessários e ainda são explicadas as decisões tomadas quanto à escolha de determinados parâmetros.

O capitulo 5 expões as conclusões finais de todo o trabalho realizado e ainda aborda

trabalho que pode futuramento ser realizado decorrente do que foi realizado até ao momento.

Capítulo 2

Revisão Bibliográfica

Neste capítulo é realizada uma revisão bibliográfica das interfaces áudio e vídeo existentes, em específico do HDMI, também sobre métodos de codificação/descodificação de sinais HDMI numa FPGA e ainda sobre ligações de alta velocidade em série e cuidados que se deve ter com as mesmas.

2.1 Interfaces de transmissão de video/audio

As interfaces de áudio e vídeo definem parâmetros físicos e interpretações dos sinais recebidos, segundo [12]. Para sinais digitais a interface acaba por definir não só a camada física mas também a camada de ligação de dados e principalmente a camada da aplicação. As características físicas do equipamento (elétrico ou ótico) incluem o número e o tipo de ligações necessárias, tensões, frequências, intensidade ótica e ainda o design físico dos conectores. Relativamente à camada de ligação de dados, esta define como os dados da aplicação serão encapsulados para que, por exemplo, possam ser sincronizados ou para fazer correções de erros. Por fim, a camada da aplicação define o formato do sinal de áudio e vídeo a ser transmitido, normalmente incorporando *codecs* não específicos. No entanto, por vezes esta camada acaba por não definir em concreto o tipo de formato de dados deixando em aberto tal parâmetro para que se possa transmitir dados no geral (é o caso do HDMI). No caso dos sinais analógicos, todas as funções que existem para os sinais digitais definidas em três camadas, são representadas num único sinal.

No caso da transmissão de sinais de áudio e vídeo digital existem várias interfaces que passam a ser analisadas, segundo [12]:

- ***Display Port:*** utiliza um conector do tipo *DisplayPort* e é o principal concorrente do HDMI. Esta interface define uma interconexão sem licenças que foi inicialmente desenhada para ser utilizada numa conexão entre o computador e o monitor do mesmo. O sinal de vídeo não é compatível com DVI ou HDMI, mas um conector *DisplayPort* pode fazer passar estes sinais.

- ***IEEE 1394 "FireWire":*** utiliza um conector do tipo *FireWire* ou i.LINK. Este protocolo de transferência de dados é principalmente utilizado em câmaras digitais, mas também em computadores e em transferências de sinal de áudio. Este tipo de interface é capaz de hospedar vários sinais no mesmo cabo entregando os dados nos devidos destinos.

- ***HDMI (High Definition Multimedia Interface):*** utiliza um conector do tipo HDMI e é uma interface de transmissão de sinal áudio/vídeo comprimida para transmissão de sinal digital descomprimida.

2.2 HDMI (*High Definition Multimedia Interface*)

O HDMI é uma interface de áudio e vídeo de alta definição que transporta dados áudio no formato não comprimido. Suporta num único cabo qualquer formato de vídeo em diversas resoluções e desde 2004 tem vindo a sofrer algumas alterações que vêm melhorar o desempenho da interface.

Esta interface está dividida em diversos canais de comunicação que implementam determinados protocolos, entre os quais se destacam as seguintes de [9]:

2.2.1 DDC - *Display Data Channel*

É um conjunto de protocolos utilizado nas comunicações digitais entre um dispositvo de origem e um dispositivo final que permite a comunicação entre ambos. Estes protocolos permitem que o ecrã comunique com o seu adaptador quais os modos que consegue suportar e também que o dispositivo que liga ao ecrã consiga ajustar alguns parâmetros, como por exemplo o contraste e a luminosidade. EDID (*Extended display identification data*) é a estrutura *standard* para este tipo de comunicações que define as capacidades do monitor e os modos gráficos suportados pelo mesmo. Este protocolo é utilizado pela *source* da comunicação do HDMI para obter os dados necessários do dispoistivo *sink*, no sentido de perceber quais os modos suportados pelo mesmo. Este canal é também ativamente usado para HDCP (*High-Bandwith Digital Content Protection*).

2.2.2 TMDS - *Transition-Minimized Differential Signaling*

É uma tecnologia utilizada para transmissão de dados em série de alta velocidade utilizado em comunicações digitais. O transmissor implementa um algoritmo que reduz as interferências eletromagnéticas nos cabos e permite ainda uma recuperação robusta de sinal de relógio no recetor.

Em específico na interface HDMI, este protocolo divide a informação a transmitir em 3 principais pacotes e intercala a sua transmissão: Período de transmissão de vídeo, período de transmissão de dados e período de controlo. No primeiro período (período de transmissão de vídeo) são transmitidas os pixeis do vídeo em linha. No segundo período

(o período de transmissão de dados) são transmitidos os dados de vídeo e os dados auxiliares à transmissão dentro dos respectivos pacotes. O terceiro período ocorre entre os dois anteriores.

Para além de ser utilizada no HDMI, esta técnica é também utilizada em interfaces DVI.

2.2.3 CEC - *Consumer Electronics Control*

É uma característica do HDMI que permite ao utilizador controlar até 15 dispositivos que tenham esta mesma característica ativa e que estão conectados por HDMI usando apenas um controlo remoto. Também é possivel dispositivos inviduais controlarem outros dispositivos sem intervenção do utilizador

2.2.4 ARC - *Audio Return Channel*

Esta característica do HDMI utiliza 2 pins do conector. É uma ligação de audio que tem como objetivo substituir outros cabos entre a TV e outros recetores ou então sistema de som. Esta direção é usada quando é a TV que gera ou recebe o vídeo mas é outro equipamente que reproduz o som. Esta característica está apenas disponível a partir da versão 1.4 de HDMI.

2.2.5 HEC - *HDMI Ethernet Channel*

Esta especificação do HDMI, tal como a anterior, está também apenas disponível a partir da versão 1.4 do HDMI e é uma tecnologia capaz de consolidar vídeo, audio e dados em série num único cabo HDMI, permitindo também aplicações baseadas em IP sobre o HDMI e uma comunicação *Ethernet* bidireccional até 100 Mbit/s.

Uma das principais características mais recentes das interfaces HDMI prende-se ao facto de permitir que sinais não sejam reproduzidos em dipositivos não autrizados. Isto é, através de um protocolo cujo nome já foi referido anteriormente, HDCP (*High-Bandwith Digital Content Protection*), o sinal HDMI pode ser encriptado e posteriormente transmitido pela *source*, protegendo assim a sua reprodução em dispositivos não autorizados. Esta tem vindo a tornar-se uma característica importante, visto que a reprodução ilegal de vídeos tem vindo a tornar-se reccorrente nos dias atuais.

2.3 HDMI implementado sobre a FPGA

A interface HDMI, tal como descrito no subcapítulo anterior, consiste numa interface que permite a transferência de sinais áudio e vídeo digitais entre dois dispositivos, e como tal será necessário um adaptador que permita a conexão entre os dois dispositivos e que ao mesmo tempo sirva como recurso para a codificação e descodificação do sinal HDMI.

Assim sendo, existe *hardware* disponível que consegue fazer as duas funções descritas, nomeadamente em [2]. Esta interface HDMI consiste num adaptador e descodificador do sinal HDMI para a FPGA, sendo que são necessárias uma placa para a transmissão e outra para a receção do sinal. Cada placa tem respetivamente dois transmissores e dois recetores independentes e faz uso dos conectores FMC para se conectar com a FPGA.

2.3.1 Conexão à FPGA XILINX VC7203 Virtex-7

Na figura 2.1 da página 10 visualiza-se a placa de desenvolvimento a ser utilizada no projeto com numeração para as suas diversas características, sendo que nesta fase se pretende perceber o que são os conectores FMC (*FPGA Mezzanine Card*) e onde estão localizados nesta mesma placa.

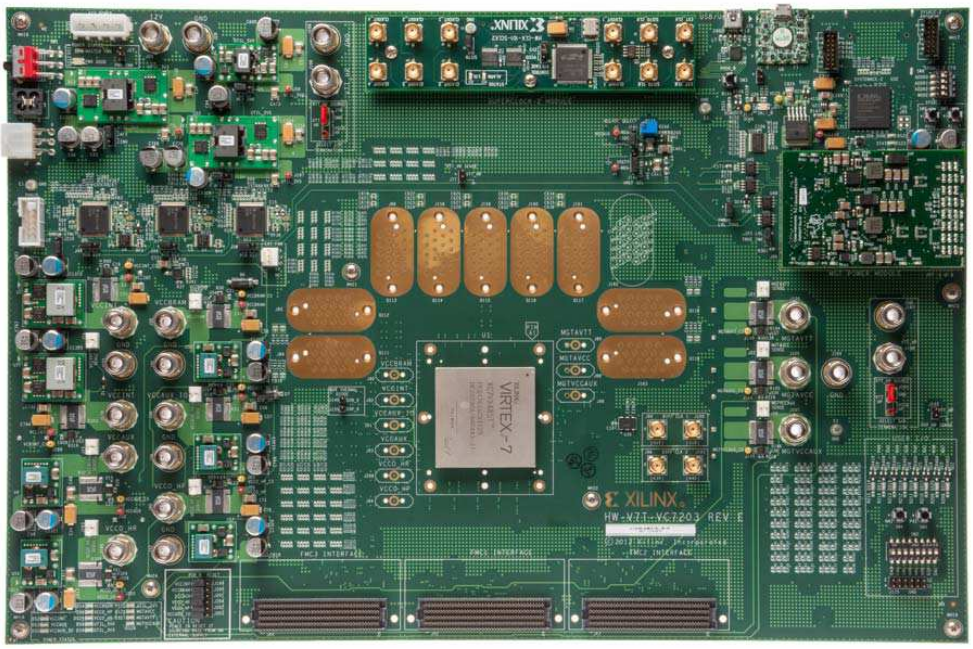

Figura 2.1: Vista Geral da FPGA VC7203 Virtex-7 retirada de [1]

A numeração 27, 28 e 29 correspondem aos conectores FMC disponíveis na FPGA a ser utilizada neste projeto. O número 27 corresponde ao conector JA2, 28 corresponde ao JA3 e 29 corresponde ao JA4. Estes conectores são usados como entradas e saídas de uma ligação, que neste caso em especifico será a uma placa HDMI, pois permitem uma ligação de alta velocidade (até 10 Gb/s). Estes três conectores HPC (High Pin Count) são compostos por 10x40 posições que permitem uma comunicação de alta velocidade por uma I/O cujo tamanho é relativamente pequeno.

O conector JA2 (FMC1 HPC) permite a seguinte conectividade:

- 68 pares que podem ser definidos pelo utilizador:
  - 34 pares LA

  - 17 pares HA
  - 17 pares HB

- 4 sinais de relógio diferenciais

O conector JA3 (FMC2 HPC) permite a seguinte conectividade:

- 68 pares que podem ser definidos pelo utilizador:
  - 34 pares LA
  - 17 pares HA
  - 17 pares HB
- 4 sinais de relógio diferenciais

O conector JA2 (FMC1 HPC) permite a seguinte conectividade:

- 65 pares que podem ser definidos pelo utilizador:
  - 34 pares LA
  - 16 pares HA
  - 15 pares HB
- 4 sinais de relógio diferenciais

Estes serão os conectores a ser utilizados e mais à frente neste relatório será explicado como é que os sinais são transmitidos.

2.3.2 Transmissor e Recetor

Este *hardware*, TB-FMCH-HDMI2, está dividido em 2 placas: o recetor (RX) que recebe o sinal recebido pelo cabo HDMI, faz a descodificação e envia o sinal para a FPGA, e o transmissor (TX) que faz o processo inverso, isto é, recebe o sinal proveniente da FPGA e transmite-o para o cabo HDMI para que possa chegar ao dispositivo de destino.

2.3.2.1 Recetor

Na figura 2.2 na página 12 é possível visualizar o diagrama de blocos do recetor disponível.

As suas principais funções dividem-se nas seguintes:

1. **Receção do Sinal HDMI (ADV7612 para a FPGA localizada na placa)**

   A receção do sinal HDMI é feita por um conector HDMI e usa um circuito integrado ADV7612BSWZ-P que recebe sinal HDMI e retira do mesmo os sinais a serem passados para a FPGA localizada na placa HDMI. O recetor tem também uma memória EEPROM (electrically erasable programmable read-only memory) que é usada para guardar dados EDID.

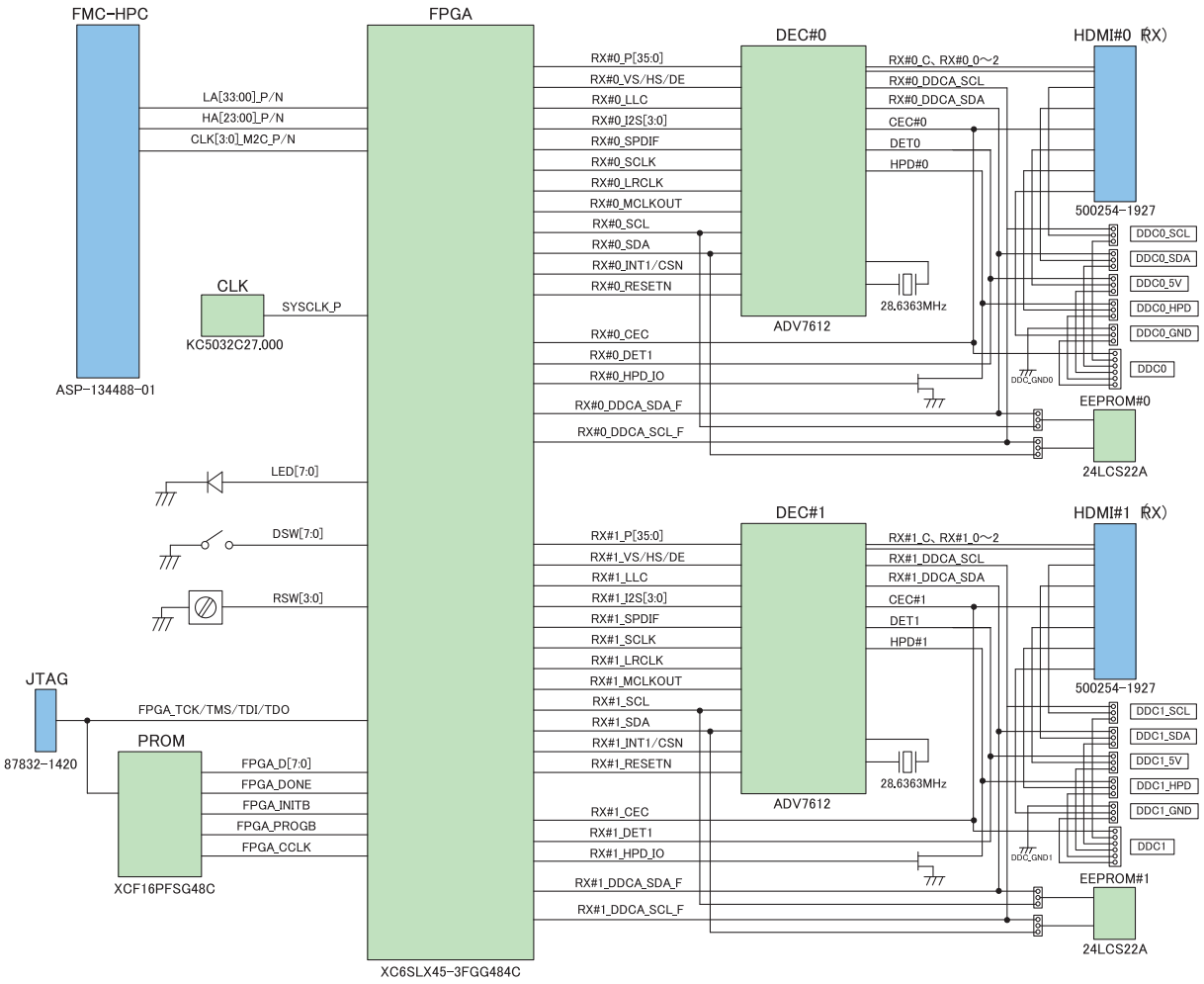

Figura 2.2: Diagrama de blocos de TB-FMCH-HDMI2 RX retirado de [2]

2. **Interface com o conector FMC (da FPGA localizada na placa para o conector FMC)**

Após passarem pela FPGA embebida (configurada por *default*) na placa são passados os seguintes sinais presentes na tabela 2.1 da página 13:

Conclui-se que os dados presentes que são transmitidos para os conectores FMC, para além dos sinais de sincronização são essencialmente dados de vídeo, os do recetor 0 passados entre LA03_P a LA32_P e os do recetor 1 passados entre LA03_N a LA32_P. O sinail "*data enable*" é um sinal que sinaliza a chegada de dados. HSYNC é um sinal que representa a sincronização horizontal e é um pulso que sincroniza o início da linha do dispositivo de destino com a imagem que a originou. Por outro lado, o sinal VSYNC é a representação da sincronização horizontal, que faz o mesmo que HSYNC (mas na vertical), certificando-se de que o dispositivo de destino começa no topo na imagem na altura correta.

Nome do pin	*Input/0utput*	FPGA para FMC	RX para a FPGA
CLK0_M2C_P	*Output*	RX#0_LLC	RX#0 sinal LLC
CLK1_M2C_P	*Output*	RX#1_LLC	RX#1 sinal LLC
LA00_P_CC	*Output*	RX#0_VSYNC	RX#0_VSYNC
LA01_P_CC	*Output*	RX#0_HSYNC	RX#0_HSYNC
LA02_P	*Output*	RX#0_DE	RX#0 data enable
LA03_P a LA32_P	*Output*	RX#0_P0 a RX#0_P29	RX#0 dados de vídeo de 0 a 29
LA33_P	*Input/Output*	Não usado	———-
CLK0_M2C_N	*Input/Output*	Não usado	———-
CLK1_M2C_N	*Input/Output*	Não usado	———-
LA00_N_CC	*Output*	RX#1_VSYNC	RX#1_VSYNC
LA01_N_CC	*Output*	RX#1_HSYNC	RX#1_HSYNC
LA02_N	*Output*	RX#1_DE	RX#1 data enable
LA03_N a LA32_N	*Output*	RX#1_P0 a RX#1_P29	RX#1 dados de vídeo de 0 a 29
LA33_P	*Input/Output*	Não usado	———-
CLK2_M2C_P	*Input/Output*	Não usado	———-
CLK3_M2C_P	*Input/Output*	Não usado	———-
HA00_P a HA23_P	*Input/Output*	Não usado	———-
CLK2_M2C_N	*Input/Output*	Não usado	———-
CLK3_M2C_N	*Input/Output*	Não usado	———-
HA00_N a HA23_N	*Input/Output*	Não usado	———-

Tabela 2.1: Nomes dos pins da interface FMC de TB-FMCH-HDMI2 RX, adaptada de [2]

Uma nota importante ainda sobre a passagem dos sinais através dos conectores FMC é que os dados provenientes da FPGA embebida na placa para os conectores são amostrados na transição de 1 para 0 do sinal de relógio do vídeo, e como tal, estes mesmos dados devem ser lidos na transição de 0 para 1 do sinal do relógio do lado da FPGA principal. A figura 2.3 na página 13 ilustra esta situação.

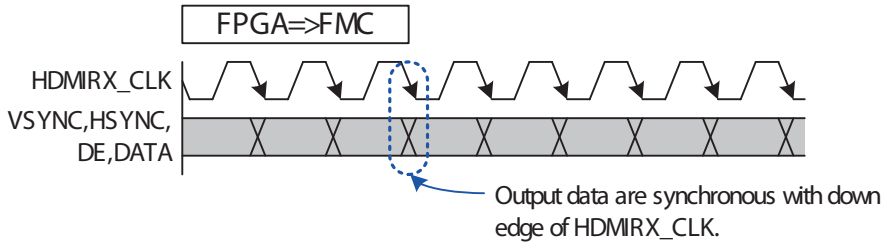

Figura 2.3: Amostragem dos dados provenientes da FPGA no recetor, retirada de [2]

Na figura 2.4 da página 14 é possivel visualizar a placa TB-FMCH-HDMI2 RX referida anteriomente.

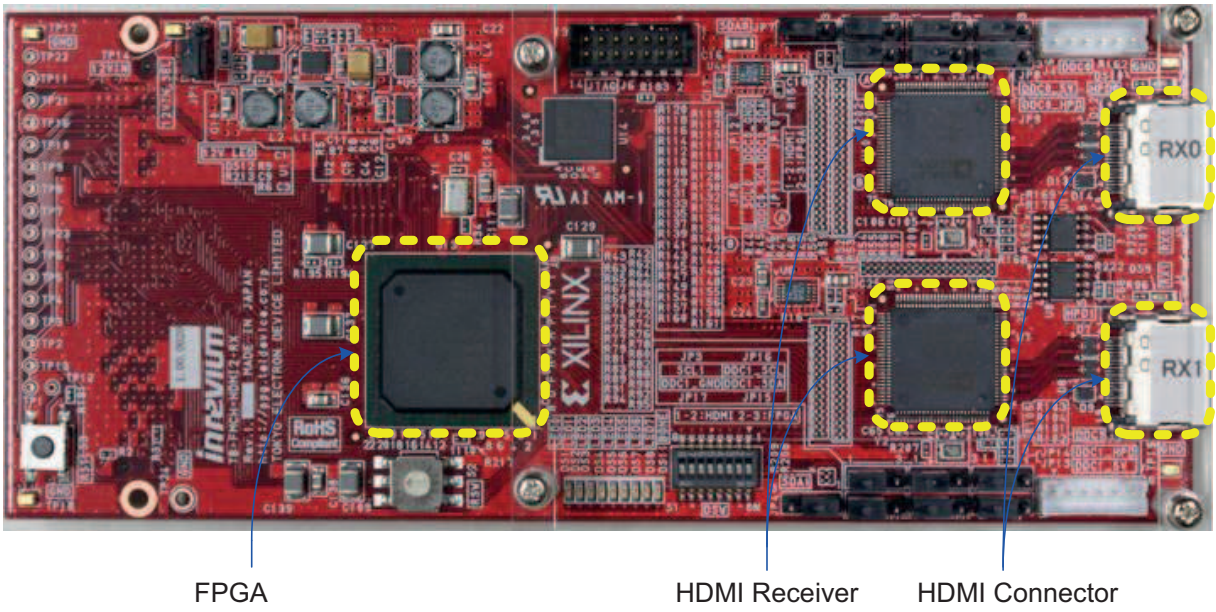

Figura 2.4: TB-FMCH-HDMI2 RX, retirada de [2]

2.3.2.2 Transmissor

O diagrama de blocos do transmissor está representado na figura 2.5 na página 15. Mais uma vez é possível dividir o diagrama em duas principais funções:

1. **Interface com o conector FMC (do conector FMC para a FPGA localizada na placa)**

   Os sinais representados na tabela 2.2 na página 16 são equivalentes aos sinais presentes na tabela 2.1 na página 13, mis uma vez com a placa configurada por *default*.

   Os dados capturados pela FPGA embebida na placa na transição de 0 para 1 do sinal de relógio do vídeo, tal como ilustra a figura 2.6 na página 15. Como tal, o sinal deve ser transferido na FPGA principal transição de 1 para 0 do sinal de relógio do vídeo.

2. **Transmissor HDMI (da FPGA localizada na placa para ADV7511)**

   Após passar pela FPGA embebida na placa o sinal passa pelo bloco ADV7511 para de seguida ser possível o seu envio pelo conector HDMI para o dispositivo de destino.

Na figura 2.7 da página 16 é possivel visualizar a placa TB-FMCH-HDMI2 RX referida anteriomente.

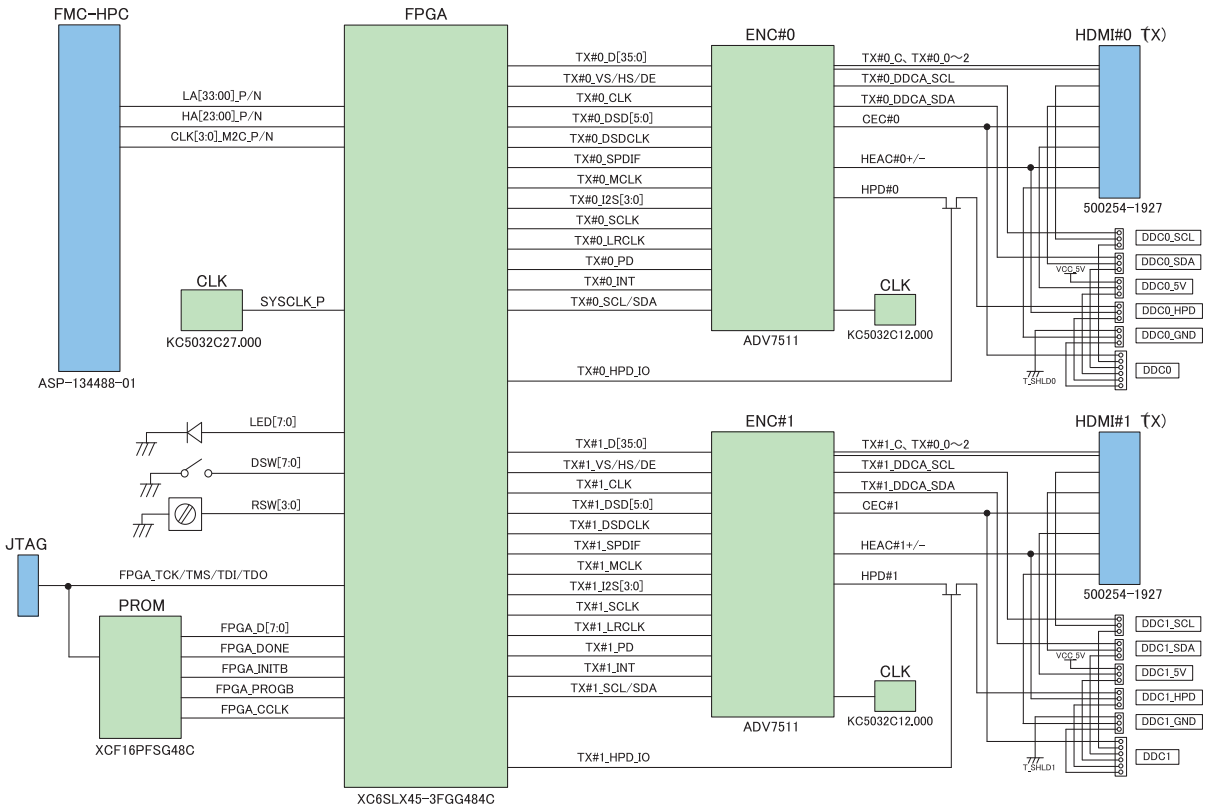

Figura 2.5: Diagrama de blocos de TB-FMCH-HDMI2 TX retirado de [2]

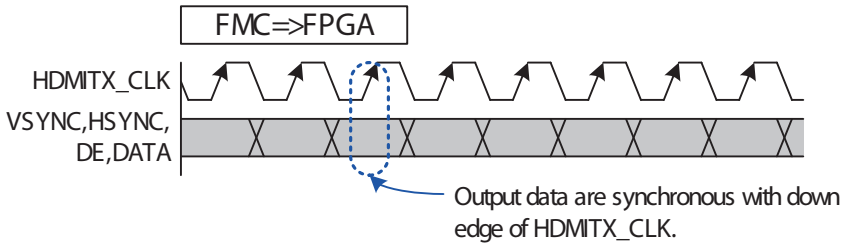

Figura 2.6: Amostragem dos dados provenientes do FMC no recetor, retirada de [2]

2.4 Conexão de alta velocidade em série -> NOVO SUB-CAPITULO

A comunicação de dados de alta velocidade pode ser efetuada tanto em série como em paralelo, sendo que cada uma tem as suas devidas implicações. No caso das comunicações em paralelo é possivel atingir uma velocidade de comuncação maior, no entanto tem um custo mais elevado devido à necessidade de mais recursos físicos. Para além de um custo maior apresentam também

Nome do pin	Input/0utput	FMC para FPGA	FPGA para TX
CLK0_M2C_P	Input	TX#0_DCLK	TX#0 sinal DCLK
CLK1_M2C_P	Input/Output	Não usado	————-
LA00_P_CC	Input	TX#0_VSYNC	TX#0_VSYNC
LA01_P_CC	Input	TX#0_HSYNC	TX#0_HSYNC
LA02_P	Input	TX#0_DE	TX#0 data enable
LA03_P a LA32_P	Input	TX#0_D0 a TX#0_D29	TX#0 dados de vídeo de 0 a 29
LA33_P	Input/Output	Não usado	————-
CLK0_M2C_N	Input	TX#1_DCLK	TX#0 sinal DCLK
CLK1_M2C_N	Input/Output	Não usado	————-
LA00_N_CC	Input	TX#1_VSYNC	TX#1_VSYNC
LA01_N_CC	Input	TX#1_HSYNC	TX#1_HSYNC
LA02_N	Output	TX#1_DE	TX#1 data enable
LA03_N a LA32_N	Output	TX#1_D0 a TX#1_D9	TX#1 dados de vídeo de 0 a 29
LA33_P	Input/Output	Não usado	————-
CLK2_M2C_P	Input/Output	Não usado	————-
CLK3_M2C_P	Input/Output	Não usado	————-
HA00_P a HA23_P	Input/Output	Não usado	————-
CLK2_M2C_N	Input/Output	Não usado	————-
CLK3_M2C_N	Input/Output	Não usado	————-
HA00_N a HA23_N	Input/Output	Não usado	————-

Tabela 2.2: Nomes dos pins da interface FMC de TB-FMCH-HDMI2 TX, adaptada de [2]

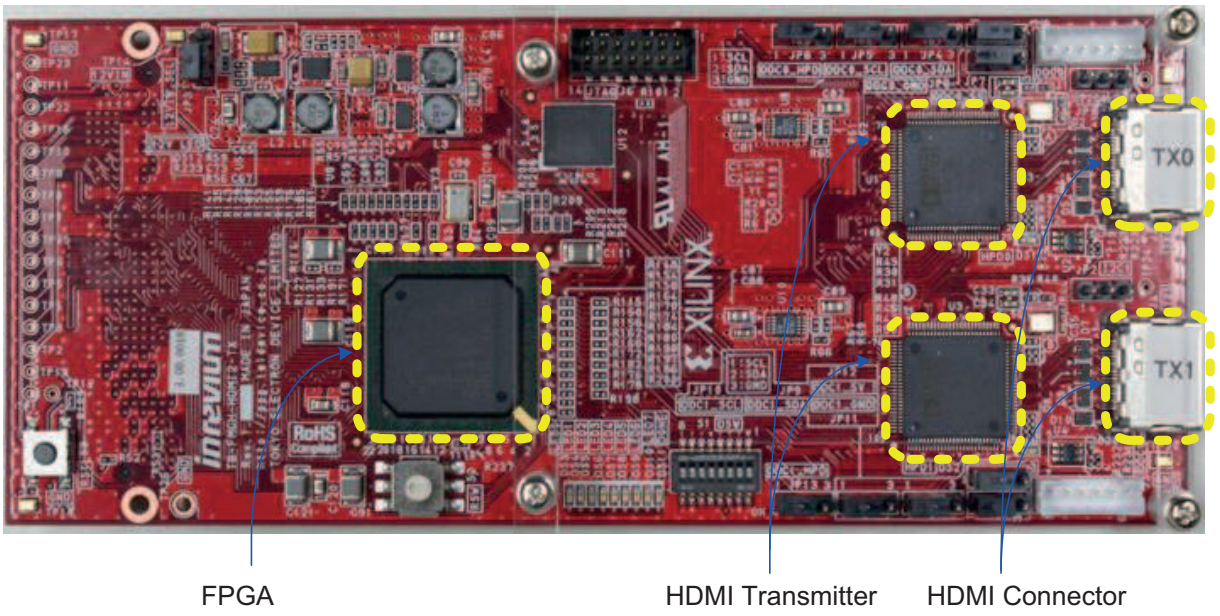

Figura 2.7: TB-FMCH-HDMI2 TX, retirada de [2]

2.5 Conexão de alta velocidade em série

A comunicação de dados de alta velocidade pode ser efetuada tanto em série como em paralelo, no entanto cada uma tem as suas implicações. No caso das comunicações em paralelo permitem uma velocidade de comunicação maior, em contrapartida tem um custo mais elevado devido à necessidade de mais recursos físicos e apresenta ainda problemas no que toca à diferença de tempos de chegada de dados e sinais de relógio (visto que estes podem chegar a tempos diferentes) e também no que toca à interferência entre canais.

Desta maneira, segundo [3], comunicações em série têm vindo a substituir as comunicações em paralelo em ligações de alta velocidade. Apesar disso, as comunicações realizadas dentro dos circuitos integrados são normalmente realizadas em paralelo, visto que permite maior rapidez de comunicação, e como tal é necessário a utilização de serializadores e deserializadores no sentido de transformar os dados nos diferentes domínios em que são utilizados.

2.5.1 Arquitetura de serializador e deserializador

Em [3] é apresentada uma arquitetura simples de um serializador/deserializador que passará a ser explicada em detalhe de seguida.

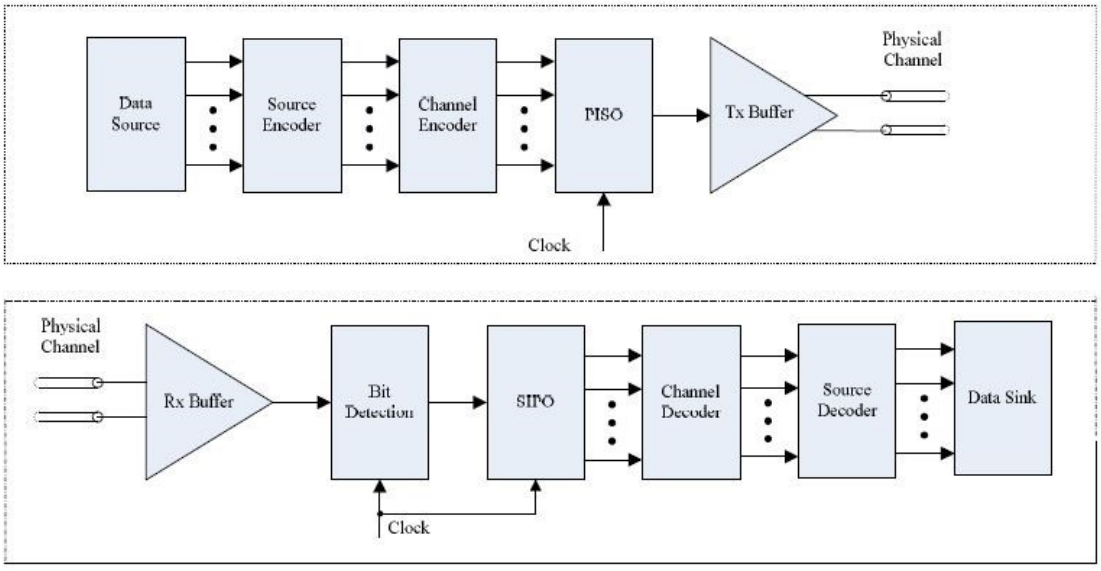

Figura 2.8: Arquitetura simples de um ser/des, retirada de [3]

Na arquitetura representada no topo da figura 2.8 na página 17 está representado o serializador proposto em [3], cujas principais fases passam a ser descritas:

- Chegada do sinal em paralelo ao bloco "*data source*", que corresponde à chegada dos dados em paralelo a serem posteriormente transmitidos.

- Codificação da fonte (*source enconding*) é bloco que se segue nesta arquitetura e inclui construção de tramas, sincronização de padrões e ainda FEC [1].
- O bloco seguinte da arquitetura corresponde à codificação de canal, que é realizada de maneira a que o sinal a ser transmitido consiga um melhor desempenho no que toca a deteção de bits no recetor.
- De seguida, o sinal codificado é enviado para o bloco PISO (*parallel input - serial output)* e de onde sai um sinal em série dos dados a serem enviados.
- Finalmente estes dados são enviados para o *buffer* para que possam ser devidamente convertidos em sinais elétricos ou pulsos óticos para que de seguida sejam transmitidos pela camada física.
- Em alguns casos um equalizador (*pre-emphasis*) para corrigir alguns erros que possam acontecer no canal em ligações de alta velocidade é utilizado. Erros podem acontecer por diversos motivos, entre os quais: a dependência da atenuação e da dispersão relativamente à frequência, interferências e ruído.

O deserializador proposto segue a mesma estrutura que o serializador fazendo, no entanto, o processo inverso.

2.5.2 Considerações na implementação deste tipo de arquitetura

Em [3] são apontados os principais desafios na implementação deste tipo de arquitetura que passarão a ser descritos brevemente e que serão tidos em conta.

1. **Restrições na utilização de circuitos**

   Logo à partida existem grandes restrições no que toca aos circuitos utilizados nestes tipos de serializadores e deserializadores. Isto porque os sinais recebidos em paralelo são sinais digitais, contudo, quando estes sinais passam pelo canal de transmissão sofrem distorções e também lhes é adicionado ruído, o que leva a que o sinal recebido do lado do recetor seja um sinal analógico e que necessite de ser tratado como tal. A sua recuperação tem de ser então baseada na regeneração correta do sinal de relógio e também na amostragem apropriada.

   Ao mesmo tempo, este tipo de dispositivos são normalmente um subsistema de um sistema grande e usados em dispositivos portáteis, e como tal precisam de ter um baixo consumo de energia. Assim sendo, um dos primeiros grandes desafios desta implementação de serializador/deserializador, segundo [3], é conseguir implementar circuitos digitais de alta velocidade e que ao mesmo tempo têm um baixo consumo de potência. Esta mesma referência apresenta duas principais técnicas utilizadas para

[1] *Foward Error Correction* é uma técnica que permite o controlo de erros na transmissão de dados.

alcançar tais objetivos que passam pela utilização de lógica CMOS que permitem a utilização a alta velocidade com baixo consumo de potência.

Outro requisito crítico na implementação deste tipo de arquiteturas é também a adaptação das impedâncias características do *buffer* (de transmissão e receção) com a impedância característica da linha onde é transmitido o sinal. Isto porque, caso estas não estejam adaptadas ocorrerão reflexões no lado do transmissor ou do recetor (consoante a desadaptação) que não permitem a transmissão da potência total do sinal. No entanto, este requisito requer um grande consumo de potência, pois na prática os canais de transmissão têm uma impedância muito baixa.

2. **PISO (*Parallel input – serial output*) e SIPO (*serial input – parallel output*)**

   Estes blocos são necessários para o correto funcionamento deste tipo de arquitetura uma vez que é responsável pela conversão dos dados que se fazem chegar em paralelo em série e vice-versa. Na figura 2.9 na página 19 apresenta-se algumas topologias de PISO e SIPO apresentadas em [3]:

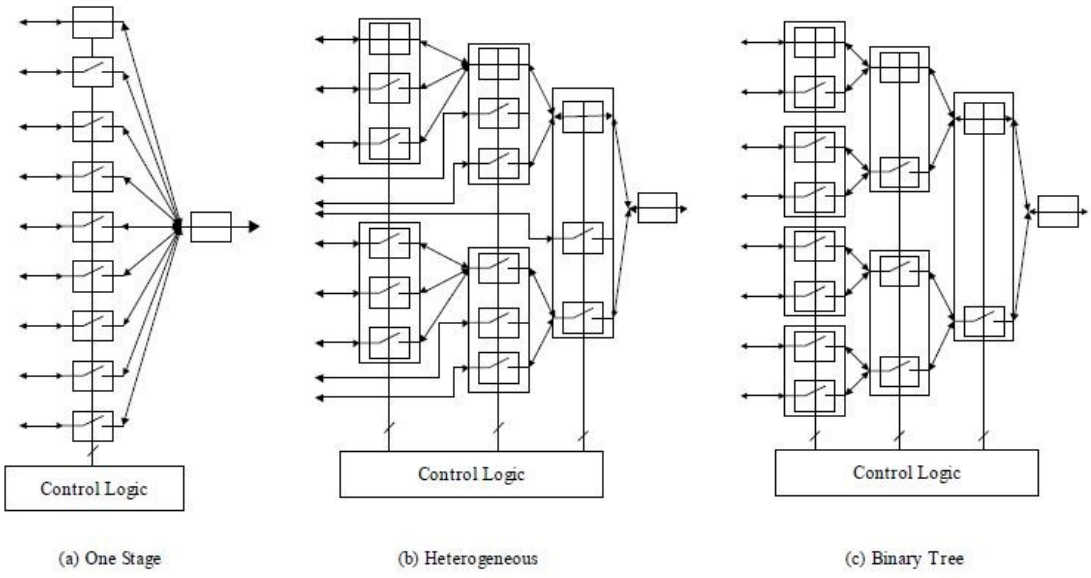

Figura 2.9: Arquitetura de PISO/SIPO, retirada de [3]

No circuito a) visualiza-se uma estrutura de um único andar que é demasiado lenta devido às capacidades intrínsecas largas no nó de conversão. O circuito b) representa uma topologia heterogénea que é mais rápida que a anterior, e no circuito c) está representada uma topologia de árvore binária que é a topologia mais rápida das apresentadas.

Para que estes blocos funcionem é necessário que exista um sinal de relógio de alta frequência (à taxa de débito do canal em série) e um sinal de baixa frequência

também (para a os dados em paralelo). O sinal de relógio mais alto é usado para amostrar na saída os dados provenientes do sinal em paralelo e ao mesmo tempo para amostrar os dados recebidos em série. O sinal de frequência mais baixo, é utilizado para colocar na saída os dados que são amostrados do sinal em série. Deste modo, é necessário a utilização de multiplicadores de sinal de relógio e divisores de frequências que serão de seguida mencionados.

3. **Unidade de Multiplicação de Sinais de relógio (CMU – *Clock Multiplier Unit*)**

   O sinal de relógio de alta frequência é bastante importante na implementação de arquiteturas de serialização e deserialização de alta velocidade, isto porque este sinal é necessário tanto do lado do recetor como do transmissor, segundo a fonte [3]. Do lado do transmissor é necessário para gerar os símbolos a serem transmitidos e do lado do recetor é necessário para que a amostragem do sinal recebido possa ser bem realizada. É comum que este sinal de relógio seja então partilhado entre o recetor e o transmissor, sendo que será necessário um bloco que faça o ajuste de fase deste sinal do lado do recetor. Este é necessário por causa do atraso introduzido pelo canal, que não é conhecido à priori e também devido ao ruído introduzido no canal que tornam a fase do sinal recebido bastante crítica para o desempenho do transcetor. Esta unidade é então responsável por tal procedimento.

4. **Equalização**

   O sinal comunicado ao longo do canal pode sofrer interferências por vários motivos, interferências essas que são críticas no que toca à receção do sinal no recetor. Como tal, existe uma necessidade de utilizar técnicas que melhorem a ligação entre os dois terminais. Segundo [3], esta melhoria pode ser realizada através da utilização de canais de ligação de melhor qualidade. No entanto, esta opção traz custos acrescidos à ligação.

   Por outro lado, também pode ser utilizada uma técnica de equalização que consegue obter bons resultados sem custos acrescidos à ligação. Ainda na mesma referência são apresentadas diferentes combinações de métodos de equalização que passam a ser brevemente descritos:

   - Linear ou não-linear
   - Pode ser implementado do lado transmissor, ou do recetor ou ainda de ambos os lados
   - Pode ser implementado em tempo contínuo ou discreto
   - Pode ser adaptativo ou fixo

Assim sendo, existe um vasto conjunto de opções de equalização que estão diretamente relacionadas com o circuito CDR (*Clock Data Recovery*), sendo que as mais importantes serão referidas mais à frente neste relatório.

5. **CDR (*Clock Data Recovery*)**

   Tal como referido anteriormente, a comunicação de sinais de alta velocidade pode sofrer diversas interferências durante a sua transmissão. Contudo, segundo [3], após a equalização do sinal estas mesmas interferências são parcialmente compensadas permitindo assim uma recuperação dos dados transmitidos. Para fazer a correta recuperação do sinal é necessário recorrer a um circuito que recupere inicialmente o sinal de relógio transmitido do emissor para que o sinal recuperado possa ser usado para recuperação dos dados transmitidos. Uma estrutura deste tipo de circuito será descrita mais à frente neste relatório.

2.5.3 Serializador e Deserializador disponíveis na FPGA

As FPGA de série 7 da XILINX têm disponíveis transcetores capazes de comunicação em série de alta velocidade, tal como é necessário neste projeto. Em específico, na FPGA XILINX VC7203 Virtex-7 estão disponíveis transcetores GTX que permitem uma velocidade de 10 Gb/s e que são os mais adequados para conexão à fibra ótica. Noutros modelos existem outros transcetores, como por exemplo GTZ (que permite até 28 Gb/s), GTH (que permite débitos até 13,1 Gb/s) e GTP (com débitos até 6,6 Gb/s). No entanto apenas serão abordados os transcetores GTX, visto que são os mais adequados para este tipo de comunicações.

Na figura 2.10 na página 22 é possível visualizar a FPGA a ser utilizada no projeto e visualiza-se ainda assinaladas as entradas GTX (QUAD_111, QUAD_112, QUAD_113, QUAD_114, QUAD_115, QUAD_116, QUAD_117, QUAD_118 e QUAD_119). Os transcetores baseiam-se na seguinte arquitetura, segundo [4]:

- **PMA (*Physical Medium Attachment Sublayer*)** que inclui:
  - suporte de taxas de débito até 12,5 Gb/s
  - uma PLL por canal para melhor flexibilidade do sinal de relógio
  - uma interface que faz a conversão de série para paralelo e de paralelo para série (PISO e SIPO)
  - uma PLL (*Phase-Locked Loop*)
  - Equalizador de decisão com feedback (DFE)
  - CDR (*clock data recovery*)
  - Bloco de pré-ênfase e equalização
  - Saída do transmissor programável

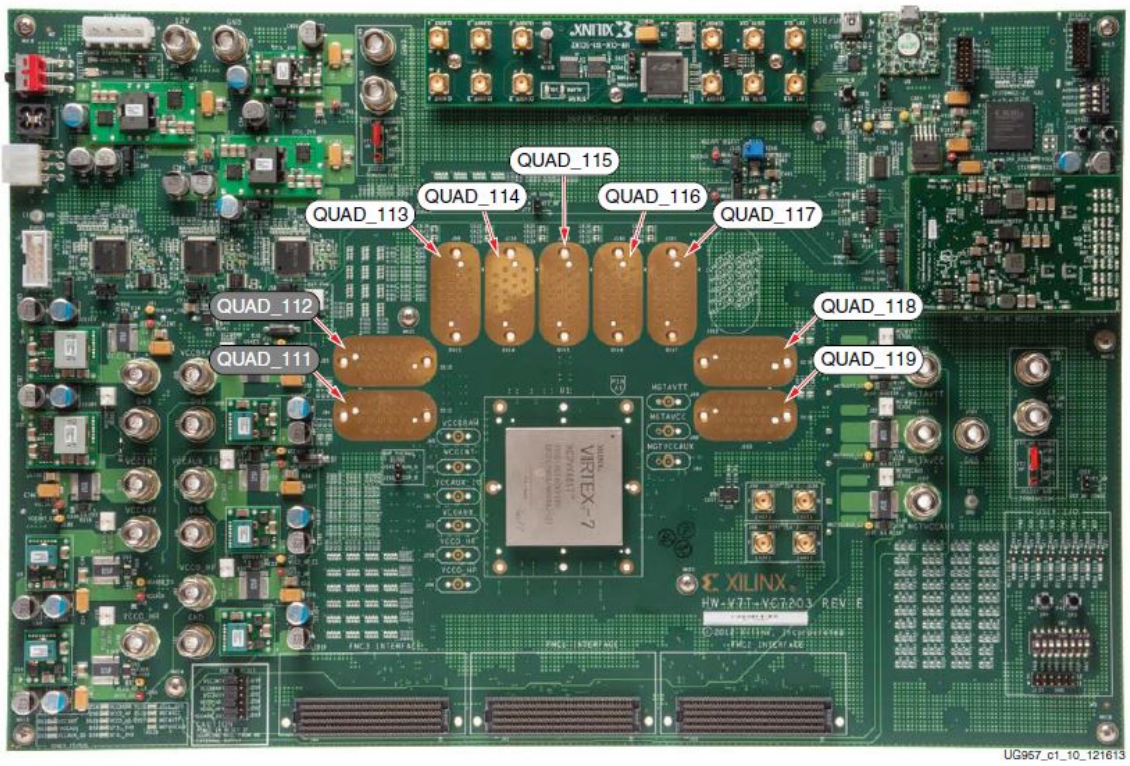

Figura 2.10: Identificação dos transcetores GTX na FPGA VC7203 Virtex-7, retirada de [1]

- **PCS (*Physical Coding Sublayer*) que inclui:**
  - *Datapath* de 2 e 4 byte internos para suportar diferentes taxas de débitos
  - Codificação e descodificação 8B/10B
  - Deteção de vírgula e alinhamento de palavra
  - PRBS (*Pseudo Random Bit Sequence*) gerador e verificador
  - FIFO para correção do sinal de relógio e ligação do canal
  - Lógica que processa os dados em paralelo reconfigurável
  - Este bloco trabalha com taxas de débitos de informação mais baixas.

É possível visualizar um diagrama geral da arquitetura dos transcetores GTX disponíveis na FPGA na figura 2.11 na página 23.

O transmissor e o recetor passam a ser descritos mais detalhadamente de seguida.

2.5.3.1 Transmissor

Cada transcetor GTX inclui um transmissor independente que consiste num módulo PCS e um modulo PMA, tal como referido anteriormente. A figura 2.12 na página 23 representa o diagrama de blocos do transmissor.

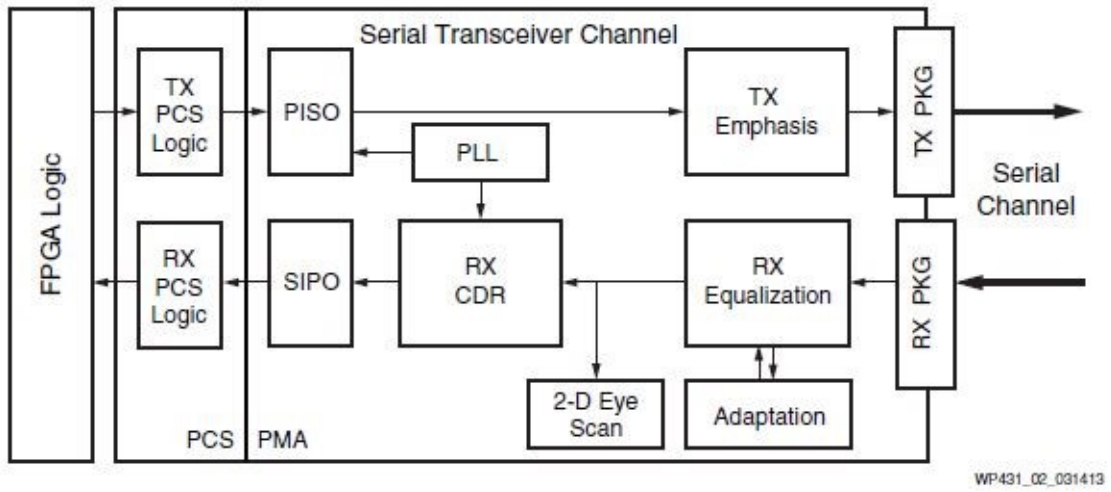

Figure 2: **7 Series Transceiver Channel Architecture**

Figura 2.11: Arquitetura geral dos transcetores GTX, retirada de [4]

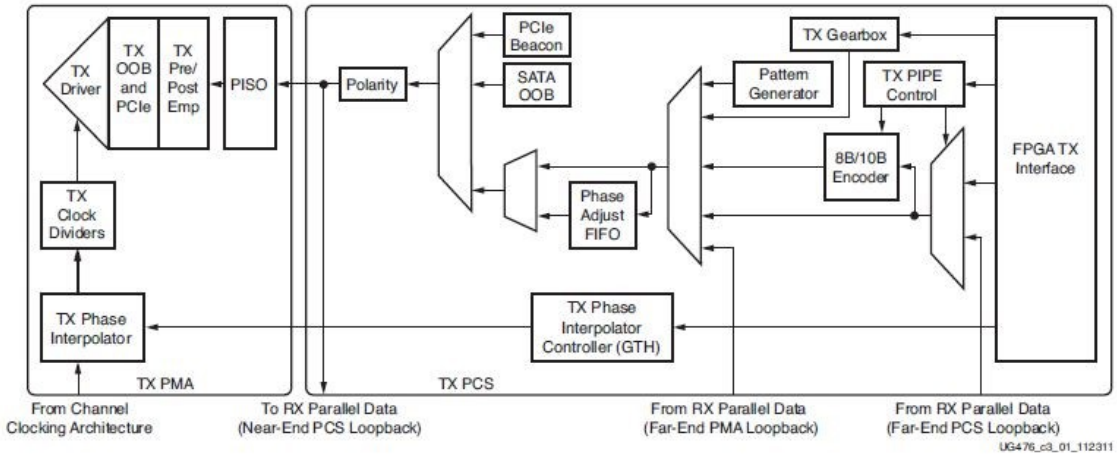

Figura 2.12: Diagrama de blocos de um transmissor GTX, retirada de [5]

Os dados provenientes da FPGA, cujo formato é em paralelo, passam para a interface transmissora, de seguida para os módulos PCS e PMA e, por fim, para a saída pelo driver do transmissor a alta velocidade.

O transmissor contém os seguintes blocos principais, cujas funcionalidades passam a ser brevemente resumidas, segundo [4]:

1. **Interface transmissora da FPGA**

   Esta interface serve como porta de comunicação entre a FPGA e o datapath do transmissor. Esta comunicação é feita através da escrita de dados na porta TXDATA na transição de 0 para 1 do sinal de relógio TXUSRCLK2.

TX8B10BEN	TX_DATA_WIDTH	TX_INT_DATAWIDTH	Tamanho na interface da FPGA (bits)	Tamanho interno dos dados (bits)
1	20	0	16	20
	40	0	32	20
	40	1	32	40
	80	1	64	40
0	16	0	16	16
	20	0	20	20
	32	0	32	16
	32	1	32	32
	40	0	40	20
	40	1	40	40
	64	1	64	32
	80	1	80	40

Tabela 2.3: Configuração do tamanho dos dados de TXDATA, adaptada de [5]

O tamanho do sinal a ser transmitido pode ser configurado para 2, 4 ou 8 bytes. Na realidade este tamanho é definido por TX_DATA_WIDTH e TX_INT_DATAWIDTH e ainda por TX8B19BEN, e o tamanho interno destes sinais pode ser 16, 20, 32, 40, 64 e 80 bits. A tabela 2.3 na página 24 demonstra como esses tamanhos são configurados através das entradas referidas.

Quando o codificador 8B/10B está ativo, então TX_DATA_WIDTH deve estar configurado para 20, 40 ou 80 bits e nesta situação, a interface do transmissor com a FPGA apenas utiliza os dados provenientes da porta TX_DATA_WIDTH. Quando o mesmo está desativo, então TX_DATA_WIDTH pode estar configurado para 16,20,32,40,64 ou 80 bits.

O sinal TX_INT_DATAWITH é um atributo que configura a ativação do datapath de 2 e 4 bytes disponível internamente no transcetor.

Para além do sinal de relógio TXUSRCLK2, que é o sinal de relógio principal para a sincronização dos sinais que chegam ao transmissor, existe um segundo sinal de relógio paralelo que é usado internamente para operações lógicas a realizar no módulo PCS. Este segundo sinal de relógio, TXUSRCLK, irá depender do tamanho interno dos dados usado no datapath e da taxa de transmissão do transmissor GTX. É possível calcular esta mesma taxa através da divisão entre a taxa de transmissão da linha e do tamanho interno dos dados utilizado no datapath. Para além disso, estes dois sinais de relógio têm uma relação fixa que determina os seus valores que depende dos valores presentes em TX_DATA_WIDTH e TX_INT_DATAWIDTH. Essas relações estão apresentadas na tabela 2.4 na página 25:

Assim sendo, é possível fazer uso dos transcetores disponíveis na FPGA, utilizando um tamanho adequado de dados para a transmissão, tendo em conta as configurações necessárias e disponíveis para tal, tal como descrito anteriormente. Por outras

Tamanho na interface da FPGA (byte)	TX_DATA_WIDTH	TX_INT_DATAWIDTH	Frequência de TXUSRCLK2
2	16, 20	0	f(TXUSRCLK2) = f(TXUSRCLK)
4	32, 40	0	f(TXUSRCLK2) = f(TXUSRCLK) / 2
4	32, 40	1	f(TXUSRCLK2) = f(TXUSRCLK)
8	64, 80	1	f(TXUSRCLK2) = f(TXUSRCLK) / 2

Tabela 2.4: Configuração da frequência de TXUSRCLK2, adaptada de [5]

palavras, utilizando tamanhos de entradas devidamente apropriados, será fácil enviar os dados recebidos e descodificados do HDMI através destes transcetores, que pode vir a ser útil numa fase inicial do projeto.

2. **Codificador 8B/10B do transmissor**

   Esta é a codificação utilizada no sinal para de seguida fazê-lo enviar pelas portas de alta velocidade, e é uma codificação *standard* que troca dois bits por byte para alcançar um equilíbrio e obter uma disparidade limitada para que a recuperação de relógio seja razoável. O transmissor possui um *datapath* específico para fazer este tipo de codificação e ao mesmo tempo poupar recursos da FPGA apesar de aumentar a latência no caminho do transmissor.

   A ativação ou não deste bloco é representada no sinal TX8B10BEN. Quando está ativo, o sinal passado na interface do transmissor com a FPGA por TXDATA é codificado antes de ser enviado pelas saídas de alta velocidade, caso contrário, tal não acontece e o sinal é enviado tal como é transmitido.

3. ***Gearbox* do transmissor**

   Este bloco suporta a codificação do sinal 64B/66B e 64B/67B, uma vez que este tipo de codificação é utilizada em alguns protocolos de comunicação de alta velocidade. Esta utilização permite reduzir a sobrecarga da codificação 8B/10B e ao mesmo tempo reter os benefícios de um esquema de codificação. Este bloco suporta interfaces de 2, 4 ou 8 bytes e a codificação dos dados é feita na lógica da FPGA.

4. ***Buffer* do transmissor**

   O transmissor GTX tem disponível também um buffer e um bloco de alinhamento de fase no seu circuito para que possa sincronizar os diferentes domínios dos sinais de relógio. Isto acontece porque internamente o transmissor tem dois sinais de relógios paralelos: o sinal do domínio PMA (XCLK) e o sinal de relógio TXUSRCLK. No entanto, quando a transmissão de dados entre estes dois domínios é realizada é necessário que os sinais estejam sincronizados e as diferenças de fase resolvidas. Como tal, será necessário a utilização deste bloco para se poder realizar a sincronização entre os sinais.

O circuito de alinhamento da fase é utilizado para resolver a diferença entre as fases dos sinais quando o buffer não está ativo. Mas pelo menos um dos blocos deve ser sempre utilizado.

A utilização do *buffer* é mais fácil e é sempre recomendada a sua utilização, enquanto que o bloco de alinhamento de fase é um bloco mais complexo no que toca a lógica e que requer restrições adicionais nas fontes dos sinais de relógio. Por outro lado, o buffer não deve ser utilizado quando a latência é uma questão importante do circuito, uma vez que o bloco de alinhamento de fase consegue alcançar menor latência.

5. **Gerador de padrões do transmissor**

   A geração de sequência pseudoaleatórias é bastante utilizada em sistemas de telecomunicações para testar a integridade do sinal de ligações de grande velocidade. Apesar de estas mesmas sequências parecerem aleatórias à primeira vista, na realidade apresentam determinadas características que são utilizadas para medir a qualidade da ligação. Este bloco do transmissor é responsável por esta ação.

6. **Controlo de polaridade do transmissor**

   Este bloco é responsável por inverter os dados em paralelo antes da sua serialização e transmissão para compensar a inversão de polaridade no par diferencial, isto porque tal pode levar a uma inversão de polarização dos dados transmitidos pelo GTX quando TXN e TXP são acidentalmente trocados na PCB.

7. **Controlo do sinal de relógio de saída do transmissor**

   Este bloco é responsável pelo controlo da divisão do sinal de relógio em série e pelo controlo da divisão e seleção do sinal de relógio em paralelo.

   No que toca à divisão do sinal de relógio em série, cada módulo PMA do transmissor possui um divisor D que divide o sinal de relógio da PLL para suportar taxas de transmissão mais baixas. Este divisor pode ser definido estaticamente para aplicações com uma taxa de transmissão fixa, mas também pode ser utilizado para ligações onde a taxa de transmissão pode variar e como tal D varia com essa mesma variação. Este bloco é responsável por esta divisão do sinal de relógio em série.

   Quanto à divisão e seleção do sinal de relógio paralelo, o sinal de relógio em paralelo que sai deste bloco de controlo de divisão de relógio pode ser utlizado como o relógio de fabrico lógico, dependendo das linhas de transmissão requisitadas. Este bloco controla também essa mesma divisão e seleção.

8. **Driver reconfigurável do transmissor**

   É um *buffer* de saída diferencial de alta velocidade que possui características para maximizar a integridade do sinal, tal como controlo diferencial de tensão, pré-ênfase e resistências de terminação calibradas.

9. **Suporte de deteção de recetor para arquiteturas *PCI Express***

   As especificações das arquiteturas PCI Express incluem características que permitem detetar a presença do recetor para uma determinada ligação. Este bloco é responsável por esta mesma deteção.

2.5.3.2 Recetor

Cada transcetor possui um recetor independente, cujo diagrama de blocos está representado na figura 2.13 na página 27. Mais uma vez, o recetor possui dois módulos principais: PCS e PMA.

O sinal de alta velocidade em série chega ao RX ao modulo PMA, passa por PCS e, por fim, passa para a lógica da FPGA pela interface com a mesma.

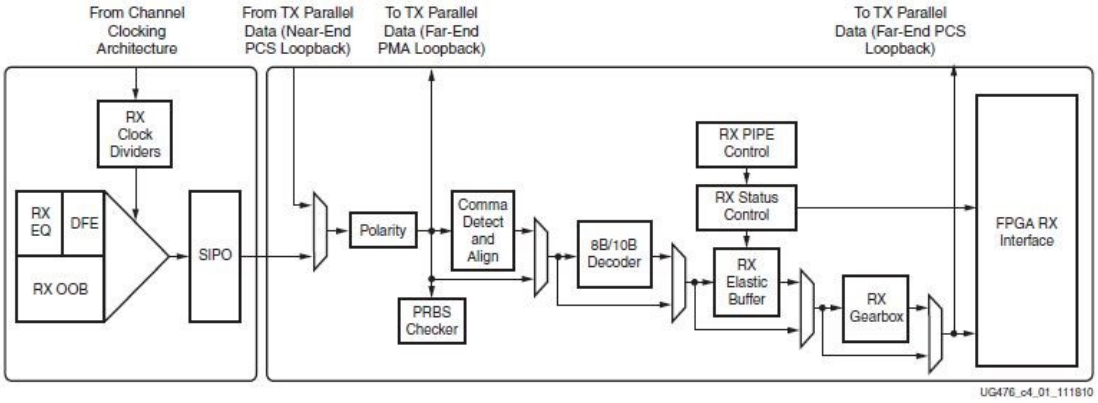

Figura 2.13: Diagrama de blocos de um recetor GTX, retirada de [5]

É possível dividir o recetor GTX nos seguintes principais blocos que passam brevemente a ser descritos:

1. ***Front End* Analógico do recetor**

   É um buffer diferencial de entrada de modo de corrente de alta velocidade que possui determinadas características tais como: reconfiguração de tensão de terminação do recetor e calibração das resistências de terminação.

2. **Equalizador do recetor**

   Ao longo da transmissão, diversos erros podem acontecer nos dados transmitidos e, como tal, são necessários filtros que permitam ou que pelo menos ajudem a realizar a recuperação do sinal recebido corretamente. Os transcetores GTX disponibilizam filtros adaptativos para tal recuperação: LPM que está otimizado para canais com poucas perdas e ainda DFE para canais com perdas maiores. As arquiteturas apresentadas de seguida foram arquiteturas brevemente abordadas anteriormente dentro deste capítulo que estão referidas em [3].

Na figura 2.14 na página 28 encontra-se o diagrama de blocos do equalizador LPM. Este modo é recomendado para aplicações com débitos até 11,2 Gb/s de curto alcance, com perdas de canal até 12 dB à frequência de Nyquist.

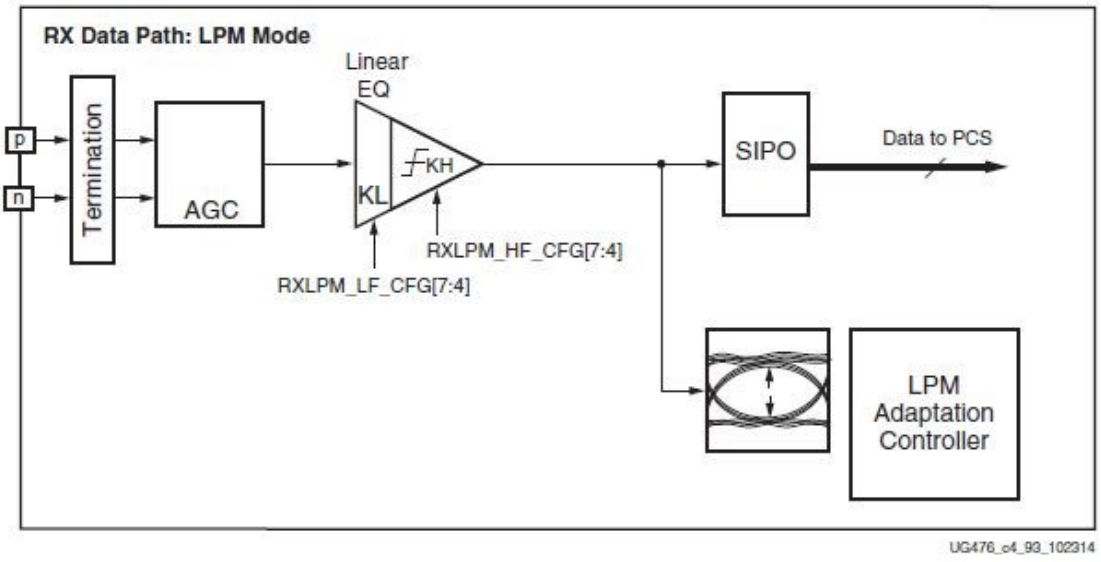

Figura 2.14: Equalizador em modo LPM, retirada de [5]

Na figura 2.15 na página 29 é possível visualizar o diagrama de blocos utilizado para o equalizador DFE (Decision Feedback Equalizer). Este é um filtro que utiliza a realimentação de símbolos detetados para produzir uma estimação da saída do canal. O DFE é alimentado com os símbolos já detetados e produz uma saída que é a combinação da saída do equalizador linear com estes mesmo símbolos já detetados.

Este equalizador é utilizado para ligações de média distância cujas perdas do canal rondam os 8 dB ou mais à frequência de Nyquist.

Vantagens da utilização deste tipo de equalizador:

- Efetua a equalização sem amplificação do ruído ou interferência
- Pode também fazer correções de reflexões causadas pelas descontinuidades do canal
- É vantajosa a sua utilização quando as interferências são preocupantes

Cuidados a ter na utilização deste tipo de equalizador:

- Este tipo de equalização deve ser cuidadosa quando não existe codificação de dados, uma vez que pode levar à não equalização ideal do sinal recebido (pois o filtro pode não se auto adaptar aos dados recebidos).

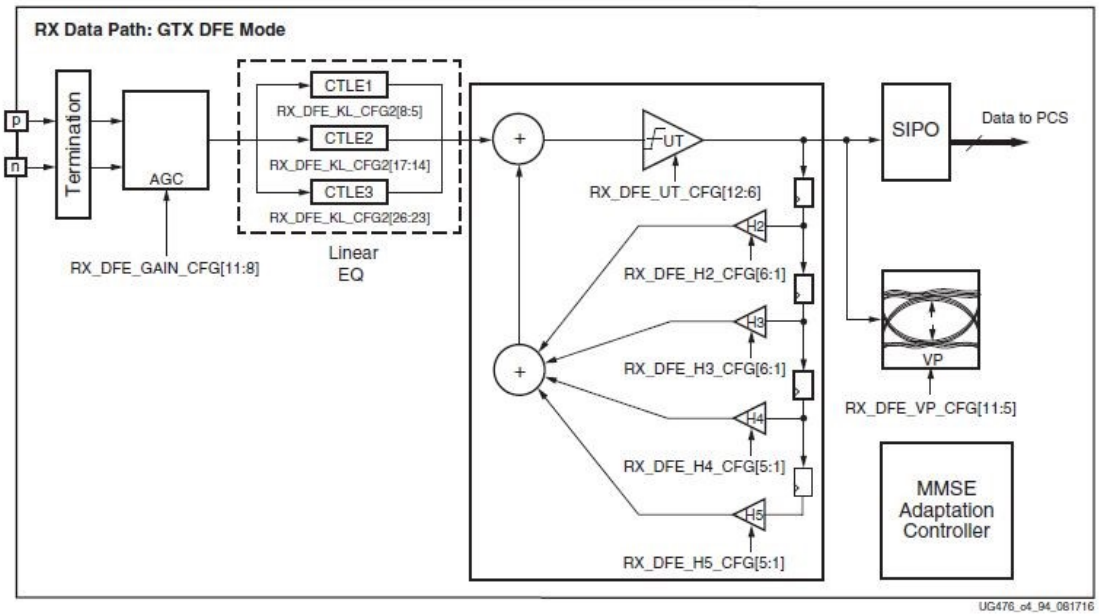

Figura 2.15: Equalizador em modo DFE, retirada de [5]

Visto que neste projeto se pretende realizar comunicações de média/longa distância, deve ser utilizado o equalizador DFE para obter uma boa equalização do sinal recebido.

3. CDR (*Clock Data Recovery*) do recetor

O circuito de CDR faz a recuperação do relógio dos dados recebidos em série. Na figura 2.16 na página 29 é possível encontrar a arquitetura deste mesmo circuito. Este circuito foi também já brevemente referido no subcapítulo anterior que refere as considerações a tomar quando se implementam arquiteturas de serialização e deserialização, e passa de seguida a ser descrito.

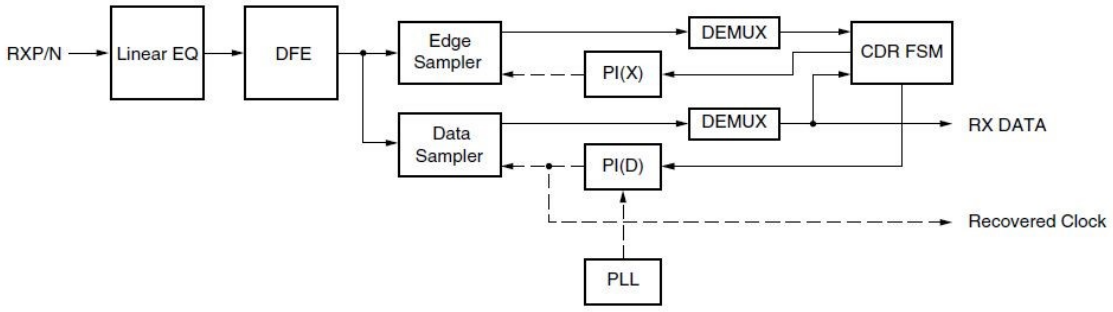

Figura 2.16: Detalhes do circuito CDR (*Clock data recovery*), retirada de [5]

A tracejado encontra-se o caminho feito pelo sinal de relógio até à sua recuperação. Os dados recebidos passam pelo equalizador e de seguida são capturados por um "data sampler" e um "edge sampler". O "edge sampler" captura a fase do sinal recebido em série quando este está na sua região de transição, enquanto que o "data sampler" captura a fase do mesmo sinal a meio do olho dos dados. Estas duas fases são de seguida enviadas para a máquina de estados do CDR para que esta consiga determinar a fase dos sinais que chegam e ao mesmo tempo controlar os interpoladores de fase (PIs).

4. **Controlo do sinal de relógio de saída**

   Tal como no transmissor, o bloco de divisão de sinal de relógio tem dois principais componentes: controlo da divisão do sinal de relógio em série e ainda controlo e seleção da divisão do sinal relógio em paralelo. As suas funções são iguais à do transmissor.

5. **Análise de Margem do Recetor**

   Com o aumento das taxas de débito da transmissão e também da atenuação os equalizadores dos recetores têm mais capacidade de superar a atenuação do canal. Contudo, isto traz um novo desafio, pois nestes casos, a qualidade da ligação não pode ser medida através da abertura do olho no diagrama de olho resultante.

   Para taxas de transmissão muito altas pode acontecer que o diagrama de olho do sinal recebido possa parecer completamente fechado, apesar de que após a equalização esteja aberto. Como tal, esta medida de qualificação da qualidade da ligação realizada pode então ter de ser reavaliada.

   Assim sendo, os transmissores GTX possuem um mecanismo que permite medir e visualizar a margem do diagrama de olho recebido após equalização. Também existem modos que permitem determinar e diagnosticar os efeitos das configurações de equalização.

   Este mecanismo permite que uma correta avaliação da qualidade do canal e para além disso, pode ser feita enquanto os dados estão a ser recebidos, devido ao seu mecanismo que assim o permite, não exigindo nenhuma alteração às configurações do recetor e nem nenhuma lógica extra da FPGA.

6. **Controlo da polarização do recetor**

   Tal como foi mencionado aquando a referência da funcionalidade deste mesmo bloco no transmissor, os sinais RXN e RXP podem ser trocados acidentalmente na PCB e como tal os dados diferencias recebidos pelo recetor estão invertidos. Este bloco é responsável pela inversão dos bytes em paralelo no módulo PCS antes da deserialização do sinal (SIPO) para compensar a polarização inversa do par diferencial.

7. **Verificador de Padrões do recetor**

   Este bloco é responsável pela verificação de determinados padrões PSBR e faz esta mesma verificação antes do alinhamento das palavras ou descodificação. Tal como descrito aquando a referência ao gerador destes mesmos padrões no transmissor, estes servem para verificar a integridade do sinal na ligação.

8. **Alinhamento de *Byte* e palavras do recetor**

   Os dados em série que chegam ao recetor devem ser alinhados com limitações de símbolos antes de poderem ser utilizados como dados em paralelo. Assim sendo, o transmissor envia sequências reconhecíveis (normalmente são chamadas de vírgulas) e o recetor procura essa mesma sequência nos dados recebido. Quando a encontra move-a para os limites das palavras para que as palavras em paralelo recebidas sejam iguais às palavras enviadas pelo transmissor. Para ativar a utilização deste bloco o sinal de entrada RXCOMMADETEN deve ser verdadeiro, mas caso a latência seja um parâmetro critico do circuito então este bloco não deve estar ativo.

   Para definir a sequência que o bloco de alinhamento deve procurar (a vírgula) nos dados que chegam em série ao recetor, então deve-se definir as entradas ALIGB_MCOMMA_VALUE ALIGN_PCOMMA_VALUE e ALIGN_COMMA_ENABLE. Os tamanhos destas sequências dependerão dos valores em RX_DATA_WIDTH, que será explicado mais à frente neste relatório.

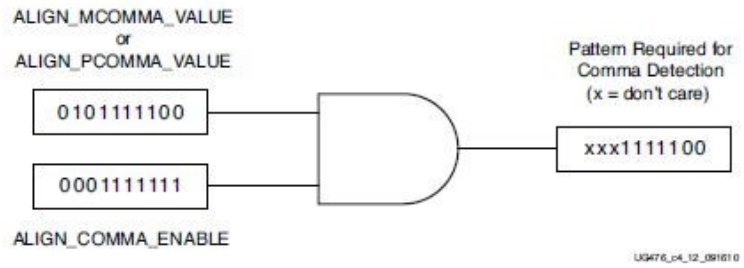

Figura 2.17: Mecanismo de obtenção da "vírgula", retirado de [5]

A figura 2.17 na página 31 ilustra o mecanismo utlizado para obter a sequência de procura dos dados recebidos no recetor quando ALIGN_COMMA_DOUBLE é falso. Quando este mesmo sinal é verdadeiro faz-se então uma extensão do sinal ALIGN_MCOMMA_VALUE e do sinal ALIGN_PCOMMA_VALUE, tal como está representado na figura 2.18 na página 32.

Quando este mesmo sinal está ativo os sinais de entrada ALIGN_MCOMMA_VALUE e ALIGN_PCOMMA_VALUE são combinados e o bloco procura por duas sequências de uma vez nos mesmos dados recebidos. Para ativar o alinhamento de palavras da sequência MCOMMA o sinal RXMCOMMAALIGNEN deve estar ativo, enquanto

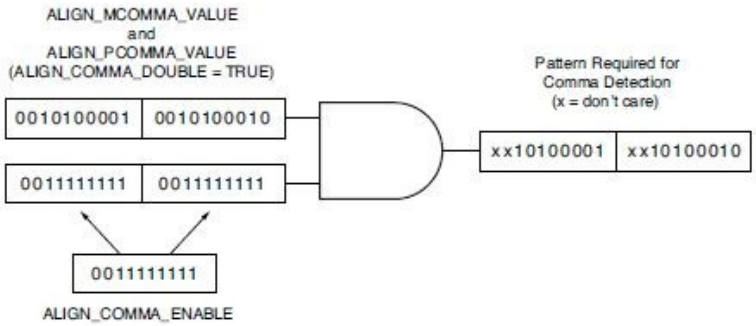

Figura 2.18: Mecanismo de obtenção da "vírgula" quando ALIGN_COMMA_DOUBLE=1, retirado de [5]

que para ativar o alinhamento da palavra PCOMMA o sinal RXPCOMMAALIGNEN deve estar ativo. Quando ambas estão ativas então o alinhamento é realizado com qualquer padrão.

É necessário ter em atenção que em aplicações cuja taxa de débito é superior a 5 Gb/s e que têm demasiado ruído, pode acontecer por vezes um falso alinhamento de palavras que leva à ativação do sinal. Isto indica que as palavras estão alinhadas sem realmente haver dados válidos presentes nelas. Assim sendo, neste tipo de sistemas é necessária a presença de um sistema que faça a verificação da validade destes dados alinhados para prevenir casos como estes.

Deste modo, com esta característica do recetor GTX da FPGA o alinhamento de palavras torna-se simplificado.

9. **Descodificador 8B/10B do recetor**

   Se os sinais estiverem codificados segundo 8B/10B então devem ser descodificados segundo esta norma também. Desta forma, o recetor possui um bloco responsável pela descodificação 8B/10B no recetor sem que gaste recursos adicionais à FPGA. Este mesmo bloco pode não estar ativo caso o sinal tenha codificação 8B/10B.

10. ***Buffer* do recetor**

    Tal como no transmissor o *buffer* é utilizado para possibilitar a sincronização entre o domínio do sinal de relógio do PMA em paralelo e o sinal de relógio RXUSRCLK. Isto porque, para ser possível a transmissão de dados entre os dois domínios a taxa do domínio PMA dever ser suficientemente parecida com a taxa de RXUSRCLK e todas as diferenças de fases entre as mesmas devem estar resolvidas. Este é, então, o bloco responsável por estes ajustes que devem ser feitos. Alternativamente a este buffer pode ser utilizado o circuito de alinhamento de fase, tal como foi referido anteriormente. No entanto, existem algumas vantagens e desvantagens de utilização destas duas opções.

A utilização do *buffer* torna-se mais fácil em termos de operação, enquanto que o circuito de alinhamento exige lógica extra e restrições adicionais relativamente às fontes do sinal de relógio, tal como acontecia para o transmissor. Quanto à utilização de sinais de relógio o buffer pode usar tanto o sinal de relógio recuperado como o sinal de relógio local, enquanto que o circuito de alinhamento de fase apenas pode utilizar o sinal de relógio recuperado pelo recetor. Relativamente aos tempos de estabilização a utilização do buffer não necessita de começar a funcionar imediatamente após a sua inicialização, enquanto que o circuito de alinhamento do sinal necessita de esperar pela estabilização de todos os sinais de relógio antes de conseguir realizar qualquer alinhamento de fase ou atraso. Em contrapartida, o buffer tem uma latência maior do que o circuito de alinhamento de fase, apesar de essa mesma latência depender também de algumas características do mesmo, como por exemplo a correção do sinal de relógio ou a ligação entre os canais do recetor.

11. **Correção do Sinal de relógio do recetor**

    Este bloco é responsável por evitar o overflow do buffer, isto porque o buffer faz a ponte de ligação entre dois domínios de sinal de relógio que apesar de serem muito idênticos nunca serão iguais. Como tal, haverá sempre uma ligeira diferença de fase entre os dois sinais causando acumulação que podem levar a um overflow ou underflow a não ser que seja corrigido.

    Para fazer esta correção, cada transmissor envia periodicamente um ou mais caracteres especiais que permitem que o recetor os elimine ou replique no buffer consoante a necessidade. Através da remoção desses caracteres quando o buffer está muito cheio e a sua replicação quando o *buffer* está vazio o recetor previne o *oveflow* ou *underflow*.

12. **Ligação de canais do recetor**

    Este bloco é responsável por fazer chegar todos os canais ao mesmo tempo ao recetor. Isto acontece porque existem protocolos que combinam múltiplos transcetores para criar um único canal de saída de alta velocidade. A diferença entre os tamanhos dos sinais de cada transcetor pode fazer com que os sinais sejam enviados todos ao mesmo tempo, mas que cheguem a tempos diferentes ao recetor. Este bloco é responsável por eliminar este efeito através do uso de um buffer como um bloco cuja latência é variável.

    Os transmissores GTX enviam um caracter que identifica a ligação entre canais (ou uma sequência de caracteres) simultaneamente. Quando este é recebido o recetor é capaz de determinar a diferença entre cada canal e ajustar a latência do buffer para que os dados cheguem todos ao mesmo tempo a interface com o utilizador.

13. ***Gearbox* do recetor**

RX8B10BEN	RX_DATA_WIDTH	RX_INT_DATAWIDTH	Tamanho na interface da FPGA (bits)	Tamanho interno dos dados (bits)
1	20	0	16	20
	40	0	32	20
	40	1	32	40
	80	1	64	40
0	16	0	16	16
	20	0	20	20
	32	0	32	16
	32	1	32	32
	40	0	40	20
	40	1	40	40
	64	1	64	32
	80	1	80	40

Tabela 2.5: Configuração do tamanho dos dados de RXDATA, adaptada de [5]

Este bloco é similar ao bloco gearbox do transmissor referido anteriormente neste relatório.

14. **Interface do recetor com FPGA**

Este bloco é responsável pela comunicação entre o recetor e a FPGA, ou seja, a FPGA consegue ler os dados recebidos no recetor através da leitura do sinal RXDATA na transição de 0 para 1 do sinal de relógio RXUSRCLK2. O tamanho desta porta pode ser configurado para 2, 4 ou 8 bytes, mas a largura real da porta depende de RX_DATA_WIDTH, RX_DATAWIDTH e RX8B10BEN, tal como no transmissor. Assim, os tamanhos dos dados das portas pode ser 16, 20, 32, 40, 64 ou 80 bits.

O recetor dos transcetores GTX contém datapaths internos de 2 e 4 bytes que são configuráveis através do atributo RX_INT_DATAWIDTH. A largura dos sinais na interface da FPGA é configurável através do sinal RX_DATA_WIDTH que deve ser 20, 40 ou 80 bits no caso de o descodificador 8B/10B estar ativo. Neste caso, a interface do recetor com a FPGA apenas usa as portas RXDATA. Quando o descodificador 8B/10B não é utilizado então RX_DATA_WIDTH pode ser configurado para outro qualquer valor disponível. O valor do tamanho dos sinais para as diferentes configurações no recetor está disponível na tabela2.5 na página 34.

A interface do recetor com a FPGA inclui dois sinais de relógio em paralelo: RXUSRCLK e RXUSRCLK2. A taxa de débito do sinal de relógio paralelo RXUSRCLK2 na interface é determinada pela taxa de débito de linha no recetor, a largura do sinal RXDATA e se a codificação 8B/10B está ativa ou não. Também um segundo sinal de relógio paralelo RXUSRCLK é disponibilizado para lógica interna no PCS do transmissor e depende da largura dos sinais internamente no datapath e da taxa

Tamanho na interface da FPGA (byte)	RX_DATA_WIDTH	RX_INT_DATAWIDTH	Frequência de TXUSRCLK2
2	16, 20	0	f(RXUSRCLK2) = f(RXUSRCLK)
4	32, 40	0	f(RXUSRCLK2) = f(RXUSRCLK) / 2
4	32, 40	1	f(RXUSRCLK2) = f(RXUSRCLK)
8	64, 80	1	f(RXUSRCLK2) = f(RXUSRCLK) / 2

Tabela 2.6: Configuração da frequência de TXUSRCLK2, adaptada de [5]

de débito da linha do recetor. Esta taxa pode ser calculada através da razão entre a taxa de débito da linha e da largura dos dados internamente no *datapath*.

Existe uma relação entre o sinal de relógio RXUSRCLCK E RXUSRCLK2 fixa que se baseia nos sinais RX_DATA_WIDTH e RX_INT_DATAWIDTH. Por exemplo, para uma linha cuja taxa de débito seja superior a 6,6 Gb/s então é necessário recorrer ao datpath interno de 4 byte, ativando o sinal RX_INT_DATAWIDTH. A relação dos valores entre RXUSRCLK e RXUSRCLK2 está representada na tabela 2.6 na página 35.

Após a análise dos recursos existentes já na FPGA a ser utilizada neste trabalho conclui-se que estes transcetores de alto débito permitem não só fazer a transmissão do sinal, mas também incluem técnicas de recuperação fiável dos mesmos dados transmitidos. Uma vez que estes transcetores são também bastantes flexíveis em termos de configurações tornará a implementação da transmissão dos dados pretendidos mais fácil numa fácil inicial, antes da utilização dos transcetores desenvolvidos pelo projeto iBrow.

Capítulo 3

HDMI

Este capítulo descreve o trabalho realizado para cumprir a primeira parte do projeto: obter uma conexão HDMI entre recetor e transmissor. São descritas as várias configurações das placas HDMI disponíveis e ainda as arquiteturas desenvolvidas e implementadas para cumprir esta parte do projeto.

3.1 *Hardware* utilizado

Tal como mencionado no sub-capítulo 2.3, para receber os dados provenientes do cabo HDMI e fazer a sua seleção são utilizadas duas placas HDMI (TB-FMCH-HDMI2 RX E TB-FMCH-HDMI2 TX) que, através das suas entradas e saída FMC de alta velocidade, conseguem enviar para e receber da FPGA os sinais de imagem e som. Nas imagens 2.4 e 2.7 é possível visualizar o recetor (TB-FMCH-HDMI2 RX) e o transmissor (TB-FMCH-HDMI2 TX) HDMI utilizados neste projeto. Em conjunto, estas duas placas são designadas apenas por TB-FMCH-HDMI2. Estas mesmas placas são constituídas por conectores HDMI onde é recebido o sinal HDMI que de seguida é enviado para um recetor ou transmissor, ADV7612 no caso do recetor e ADV7511 no caso do transmissor. Finalmente os sinais provenientes do recetor/transmissor são enviados para uma FGPA embebida na placa (XC6SLX45-3FGG484C) que, consoante a sua configuração, envia pelos conectores FMC os sinais de áudio e vídeo

As placas possuem ainda uma PROM (*Programmable read-only memory*) XCF16PFSG48C de configuração reprogramável que permite armazenar o *bitstream* que configura a FPGA embebida do modo que se pretende. É esta FPGA embebida que em cada placa (RX e TX) é responsável pela seleção e envio ou receção dos dados pretendidos para ou dos conectores FMC, e como tal é necessário que estejam configuradas para realizarem tais procedimentos. O recurso a estas memórias reconfiguráveis vem permitir uma fácil alteração da configuração da FPGA uma vez que, segundo [13], estas memórias de leitura permitem não só armazenar os *bitstreams* de configuração da FPGA, mas também reconfigurá-los, caso se pretenda, de uma forma fácil e eficiente.

As reconfigurações destas memórias são realizadas através de um programador JTAG e ainda recorrendo a um *software*. O *software* utilizado neste projeto tem o nome de *imPACT* e é disponibilizado pela *Xilinx*. Após a conexão do conector JTAG à respectiva placa e ao computador (através de uma porta USB) é necessário inicializar o *software* e programar a memória com o respetivo ficheiro pretendido. Em [14] são detalhadas informações acerca do programador utilizado e ainda sobre o procedimento para se reconfigurar as memórias. As reconfigurações realizadas neste projeto basearam-se nesse documento.

3.1.1 Configurações da FPGA

A FPGA *Spartan-6* (XC6SLX45-3FGG484C) embebida nas placas tem 3 configurações disponíveis. Estas variam não só no suporte que possuem, que pode ser apenas de imagem mas também de áudio, mas variam também no número de bits por imagem que estas podem ter. Nas secções seguintes serão brevemente abordadas as configurações disponíveis e como se pode tirar partido das mesmas no projeto que foi desenvolvido.

3.1.1.1 Configuração por *default*

Esta configuração vem previamente escrita na memória PROM de fábrica e acaba por ser a mais simples de todas. Os dados enviados pelos conectores FMC são apenas referentes aos dados de imagem. As tabela 2.1 e 2.2 nas páginas 13 e 16 respectivamente identificam as portas às quais são atribuídas os sinais de dados de imagem HDMI tanto na placa recetora como na transmissora.

Esta configuração suporta a transmissão de imagens RGB (*Red Green Blue*) com 10 bits. Assim sendo, tal como referido em [2], independentemente da formatação das imagens da fonte HDMI o recetor ADV7612 integrado na placa recetora HDMI converte a imagem para o formato RGB e transmite de maneira a enviar os dados em apenas 10 bits. A tabela 3.1 da página 39, adaptada de [2], apresenta brevemente quais as portas das placas utilizadas e que sinais são transmitidos nas mesmas, no entanto é possível encontrar na tabela A.1 do anexo A mais detalhes relativamente a estes dados. Os nomes dos sinais são referentes aos sinais em TB-FMCH-HDMI2 (tanto TX como RX), e como tal quando se faz referência à FPGA nestas tabelas estas correspondem às que estão embebidas nas placas HDMI.

É de notar ainda que esta configuração é capaz de suportar até dois canais (RX0 e TX0, RX1 e TX1), no entanto nesta tabela apenas são apresentados os dados correspondentes ao canal 0 pois apenas será necessário utilizar um canal neste projeto.

Apesar de ser um configuração simples, uma vez que apenas são transmitidos sinais de imagem em formato RGB, é uma configuração que será utilizada numa fase inicial em algumas arquiteturas implementadas que serão descritas na secção 3.2.

PIN	FPGA -> FMC (RX)	FMC -> (TX)	Descrição
CLK0_M2C_P	RX#O_LLC	TX#O_DCLK	Sinal de relógio dos pixeis
LA00_P_CC	RX#0_VSYNC	TX#0_VSYNC	Sincronização Vertical
LA01_P_CC	RX#0_HSYNC	TX#0_HSYNC	Sincronização Horizontal
LA02_P	RX#0_DE	TX#0_DE	Sinal de dados ativos
LA03_P a LA32_P	RX#0_P0 a RX#0_P29	TX#0_D0 a TX#0_D29	Pixel de Imagem

Tabela 3.1: Descrição e localização dos pinos de TB-FMCH-HDMI2 configurada por *default*

3.1.1.2 Suporte de um canal de imagem e áudio

Para além da configuração descrita anteriormente em 3.1.1.1 que apenas suporta a transmissão de imagem, existe ainda uma configuração capaz de suportar não só a transmissão de imagem mas também de som. A configuração que é escrita na PROM da placa recetora para programar a FPGA embebida controla o recetor ADV7612 de maneira a conseguir transmitir imagens no formato YCbCr ou RGB com 12 bits e também fazer a transmissão do audio em formato I^2S. O mesmo acontece na placa transmissora mas para ser capaz de receber estas configurações.

Assim como referido em [7], neste caso a configuração da imagem está dependente da fonte HDMI, e é transmitida pelas placas tal como é emitida pela fonte, por outras palavras, se a fonte HDMI transmitir uma imagem em formato RGB é nesse mesmo que chega ao destino, no entanto se for transmitida uma imagem no formato YCbCr é nesse que chega ao seu destino. No caso do som, este é sempre transmitido em formato I^2S, o que implica a transmissão dos dados de áudio mas também sinais de relógio necessários à sua transmissão.

Na tabela 3.2 na página 40 são brevemente apresentados as portas e os sinais usados com este tipo de configuração da FPGA embebida. Na tabela A.2 no anexo A é apresentada uma tabela semelhante a esta, mas que inclui mais detalhes relativamente aos pinos usados e ao seu uso. Ambas as tabelas foram adaptadas de [7] onde são apresentados todos os detalhes dos conectores FMC das placas.

Os dados referentes ao som transmitidos pela placa recetora e recebidos de seguida pela placa emissora estão mencionados com mais detalhe na tabela A.2 do anexo A, e tal como indicado anteriormente, esta configuração é capaz de transmitir e receber dados no formato I^2S. Nas especificações deste protocolo, em [15], são definidos os sinais transmitidos aquando a utilização deste formato, que passam a ser descritos:

1. ***Continuous Serial Clock* (SCK):** Este sinal é por vezes reconhecido pelo nome

PIN	FPGA ->FMC (RX)	FMC -> FPGA (TX)	FPGA->HDMI_TX
CLK0_M2C_P	RX#0_LLC	TX#0_DCLK	Sinal de relógio dos pixeis
LA00_P_CC	RX#0_VSYNC	TX#0_VSYNC	Sincronização vertical
LA01_P_CC	RX#0_HSYNC	TX#0_HSYNC	Sincronização horizontal
LA02_P	RX#0_DE	TX#0_DE	Sinal de dados ativos
LA03_P a LA32_P	RX#0_P0 a RX#0_P29	TX#0_D0 a TX#0_D29	Pixel de imagem do bit 0 ao 29
LA00_N_CC a LA01_N_CC	RX#0_InputVideoStatus	TX#0_InputVideoStatus	Formato de video (2D/3D)
LA19_N	RX#0_MCLK	TX#0_MCLK	*Master Clock* de som
LA20_N	RX#0_SCLK	TX#0_SCLK	*Serial Clock* de som
LA21_N a LA26_N	RX#0_AP0 a RX#0_AP5	TX#0_AP0 a TX#0_AP5	Dados de som
LA27_N a LA32_N	RX#0_P30 a RX#0_P35	TX#0_P30 a TX#0_P35	Pixel de imagem do bit 30 ao 35

Tabela 3.2: Descrição e localização dos pinos de TB-FMCH-HDMI2 configurada para um canal de imagem e áudio

de *Bit Clock* e é um sinal de relógio referente aos dados de som em série transmitidos pelos canais AP1, AP2, AP3 e AP4.

2. ***Word Select*(WS)**: Este sinal é por vezes também conhecido por *Left/Right Clock* e é um sinal que indica o canal de som (esquerdo ou direito) que está a ser transmitido através dos dados em série recebidos ou enviados nas portas AP1, AP2, AP3 e AP4. É nomeado de sinal de relógio porque geralmente alterna entre 0 e 1 periodicamente, no entanto tal pode não acontecer, tal como referido em [15].

3. ***Serial Data***: Sinais que transportam os dados de audio.

Na imagem 3.1 são ilustrados os sinais referentes ao audio descritos previamente. O sinal "SCLK"(*Serial Clock*) é referente ao sinal "*Continuous Serial Clock*", o sinal LRCLK (*Left/Right Clock*) refere-se ao sinal "*Word Select*"e ainda ISx refere-se ao sinal "*Serial Data*". É de notar que os dados de som alternam à frequência do sinal "SCLK"que possui uma frequência 64 vezes superior à de "LRCLK". É sabido que a frequência deste é de 48 kHz e por isso o sinal "SCLK"possui uma frequência de aproximadamente de 3,072 MHz.

Na placa recetora HDMI, que envia os dados para a FPGA Virtex-7, é também enviado o sinal *Master Clock* que corresponde a um sinal de relógio de referência do sinais de áudio da entrada e ainda dados de áudio em AP0. É mencionado em [6] que estes dois sinais são referentes ao som no formato SPDIF e como tal não serão abordados neste projeto uma vez que as placas apenas suportam o formato I^2S.

Para além de dados de som e imagem são transmitidos dois bits com informação relativa ao estado do vídeo transmitido. Estes dados indicam o tipo e formato de vídeo que está

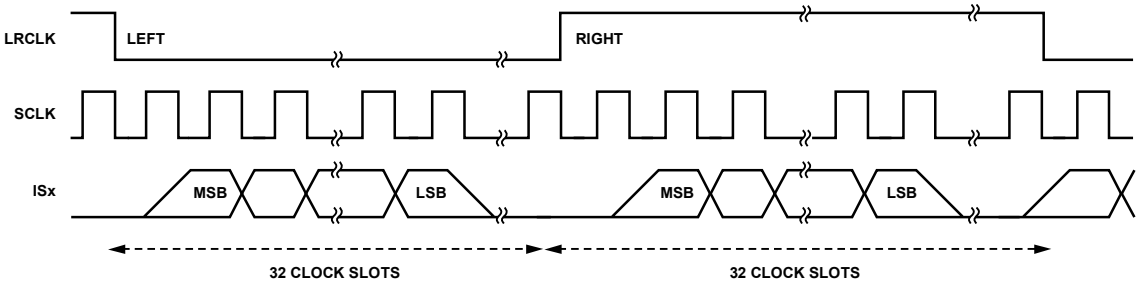

Figura 3.1: Ilustração dos sinais de som transmitidos no formato I^2S, retirada de [6]

a ser transmitidos e devem de seguida ser recebidos na placa transmissora. A combinação dos dois bits definem o estado do vídeo e são detalhadas em [7].

Esta configuração acaba por ser bastante útil e será utilizada em diversas arquiteturas desenvolvidas uma vez que possui duas grandes vantagens: é capaz de suportar som e ao mesmo tempo não limita o formato da imagem transmitida a RGB. Em contrapartida, apenas suporta um canal (ao contrário da anterior), mas tal não é um problema pois apenas se pretende obter a transmissão num único canal entre dispositivo de fonte e dispositivo final HDMI.

3.1.1.3 Suporte de dois canais de imagem melhorado

Esta configuração é capaz de suportar a transmissão de imagens em dois canais, tal como a configuração apresentada em 3.1.1.1 no entanto com alguns melhoramentos. A principal diferença consiste na capacidade de transmitir não só imagens em formato RGB mas também em YCbCr num dos canais. A tabela A.3 do anexo A foi adaptada de [8] e apresenta detalhadamente todos os sinais transmitidos entre as placas HDMI e ainda os nomes dos conectores FMC.

Esta configuração na placa recetora transmite no canal 0 (RX0) imagens tanto no formato RGB como YCbCr de 10 bits por cor e os sinais de controlo respectivos. Relativamente ao canal 1 dessa mesma placa (RX1) apenas é possível transmitir imagens em formato RGB de 10 bits por cor e os seus respetivos controlos. Para além disso, para cada canal são transmitidos dois bits que identificam o estado do vídeo que é transmitido, tal como já acontecia na configuração descrita em 3.1.1.2.

Quando à placa transmissora quando configurada desta forma é capaz de receber nos dois canais (TX0 e TX1) imagens no formato RGB ou YCbCr com 10 bits por cor. Apesar de na tabela A.3 o canal 1 definir os seus bits apenas para o caso de RGB, este canal também suporta na placa transmissora o formato YCbCr (e por isso a atribuição dos bits para TX1 assemelham-se ao canal TX0). Tal não é suportado no canal 1 da placa HDMI recetora (RX1) e por esse motivo a tabela está assim apresentada. À semelhança da placa recetora, a placa transmissora recebe 2 bits relativos à informação do vídeo que está a ser transferido, tal como a tabela A.3 sugere.

3.1.2 Configuração dos interruptores

Neste capitulo serão descritas as configurações dos interruptores presentes nas placas HDMI para cada configuração existente. Tal é necessário definir para que os recetores e transmissores presentes nas placas possam enviar e receber imagens nos formatos que o utilizador pretende. Existem 8 interruptores que podem ser definidos pelo utilizador. Os interruptores entre S1-1 e S1-4 têm como função selecionar o tipo de formato que sai do recetor ADV7612 ou ADV7511 embebido na placa. Relativamente aos outros interruptores, raramente são utilizados e quando são a sua função não é relevante para o projeto e por isso não será especificada.

3.1.2.1 Configuração por *default*

Quando as placas estão configuras de fábrica, relembra-se que as imagens transmitidas correspondem ao formato RGB de 30 bits (10 bits por cor). Como tal, a indicação que vem em [2] sobre as funções dos interruptores da placa HDMI recetora é muito pouca. Apenas é indicado que quando esta configuração está ativa os interruptores se devem encontrar tal como especifica a tabela 3.3 na página 42.

Interruptor	Estado
S1-1	ON
S1-2	ON
S1-3	ON
S1-4	ON
S1-5	Não usado
S1-6	Não usado
S1-7	Não usado
S1-8	ON

Tabela 3.3: Configuração dos interruptores da placa HDMI RX configurada de fábrica, adaptada de [2]

Relativamente à placa HDMI transmissora, é sabido que lhe chegam imagens no formato RGB de 10 bits, no entanto é possivel configurar o ADV7511 de tal forma que na sua saída o número de bits não seja limitado a 10. Para tal é necessário configurar os interruptores da forma que a tabela 3.4 indica.

3.1.2.2 Suporte de um canal de imagem e áudio

Quando se configuram as placas HDMI de forma a obter-se o suporte de áudio, então o formato da imagem transmitida também não é limitado a RGB. Desta maneira, o tabela 3.5 indica como se devem configurar os interruptores de forma a obter-se na saída do ADV7612 as diversas possibilidades relativamente ao formato da imagem.

Interruptor	**Estado**		
S1-1	OFF	ON	ON
S1-2	ON	ON	OFF
S1-3	ON	OFF	ON
S1-4	ON	ON	ON
OUTPUT	8 bits	10 bits	12 bits
S1-5	OFF		
S1-6	Não usado		
S1-7	Não usado		
S1-8	Não usado		

Tabela 3.4: Configuração dos interruptores da placa HDMI RX configurada de fábrica, adaptada de [2]

Interruptor	**Estado**					
S1-1	ON	OFF	ON	ON	OFF	ON
S1-2	ON	ON	OFF	ON	ON	OFF
S1-3	ON	ON	ON	OFF	OFF	OFF
S1-4	ON	ON	ON	ON	ON	ON
OUTPUT	YCbCr 444/422	YCbCr 444/422	YCbCr 444/422	RGB	RGB	RGB
	8 bits	10 bits	12 bits	8 bits	10 bits	12 bits
S1-5	ON					
S1-6	ON					
S1-7	ON					
S1-8	ON					

Tabela 3.5: Configuração dos interruptores da placa HDMI RX configurada para um canal e suporte de áudio, adaptada de [7]

À semelhança da placa recetora para esta configuração, também é possivel configurar o ADV7511 para se obter na sua saída diversos formatos de imagem. A tabela 3.6 apresentada na página 44 indica essas mesmas combinações.

3.1.2.3 Suporte de dois canais de imagem melhorado

Quando se reconfigura as placas para suportarem a versão de transmissão de dois canais melhorada, é necessário ter em conta que existe um canal (canal 0) que tem a possibilidade de transmitir imagens tanto no formato YCbCr como RGB, porém o canal 1 apenas o faz no formato RGB. Na tabela 3.7 da página 44 são apresentadas as configurações dos interruptores que configuram o ADV7612 de forma a enviar diferentes formatos.

Relativamente à placa HDMI transmissora, ambos os canais são capazes de suportar imagens em formato RGB ou YCbCr. A tabela 3.8 da página 44 apresenta as combinações dos interruptores para se poder obter os diversos formatos na saída do ADV7511.

Interruptor	**Estado**								
S1-1	ON	OFF	ON	ON	OFF	ON	ON	OFF	ON
S1-2	ON	ON	OFF	ON	ON	OFF	ON	ON	OFF
S1-3	ON	ON	ON	OFF	OFF	OFF	ON	ON	ON
S1-4	ON	ON	ON	ON	ON	ON	OFF	OFF	OFF
OUTPUT	YCbCr 444	YCbCr 444	YCbCr 444	YCbCr 422	YCbCr 422	YCbCr 422	RGB	RGB	RGB
	8 bits	10 bits	12 bits	8 bits	10 bits	12 bits	8 bits	10 bits	12 bits
S1-5	OFF								
S1-6	Não usado								
S1-7	Não usado								
S1-8	ON								

Tabela 3.6: Configuração dos interruptores da placa HDMI TX configurada para um canal e suporte de áudio, adaptada de [7]

Interruptor	**Estado**			
S1-1	ON	OFF	ON	OFF
S1-2	ON	ON	ON	ON
S1-3	ON	ON	OFF	OFF
S1-4	ON	ON	ON	ON
OUTPUT	YCbCr 444/422	YCbCr 444/422	RGB	RGB
	8 bits	10 bits	8 bits	10 bits
S1-5	ON			
S1-6	ON			
S1-7	ON			
S1-8	ON			

Tabela 3.7: Configuração dos interruptores da placa HDMI RX configurada para dois canais melhorados, adaptada de [8]

Interruptor	**Estado**					
S1-1	ON	OFF	ON	OFF	ON	OFF
S1-2	ON	ON	ON	ON	ON	ON
S1-3	ON	ON	OFF	OFF	ON	ON
S1-4	ON	ON	ON	ON	OFF	OFF
OUTPUT	YCbCr 444	YCbCr 444	YCbCr 422	YCbCr 422	RGB	RGB
	8 bits	10 bits	8 bits	10 bits	8 bits	10 bits
S1-5	OFF					
S1-6	Não usado					
S1-7	Não usado					
S1-8	ON					

Tabela 3.8: Configuração dos interruptores da placa HDMI RX configurada para dois canais melhorados, adaptada de [8]

3.2 Arquiteturas Desenvolvidas

Nesta secção passam a ser descritas as arquiteturas desenvolvidas e implementadas na FPGA referentes à comunicação entre as placas HDMI. Por outras palavras, é feita uma aplicação daquilo que foi explicado sobre as placas HDMI a serem utilizadas até agora em arquiteturas implementadas e testadas em FPGA.

3.2.1 Transmissão de uma imagem gerada na FPGA

Numa fase inicil do projeto, optou-se por simplificar a transmissão e para tal utilizou-se apenas a placa transmissora HDMI configurada por defeito. Contrui-se em Verilog um bloco capaz de gerar uma imagem para ser transmitida, mais especificamente uma barra de cores, e utlizou-se essa imagem para ser transmitida pelos conectores FMC.

O bloco gerador de uma barra de cores foi adaptado de um bloco disponibilizado pela *Inrenvium* aquando a compra das placas. Apesar de ter sido ligeiramente adaptado para este caso em especifico, este baseia-se essencialmente numa maquina de estados que vai contando as linhas e as colunas para que possa enviar não só os valores das cores de cada pixel, mas também os sinais de controlo como a sincronização vertical, a sincronização horizontal e ainda os valor de pixeis ativos.

Para que se entenda mais facilmente como e quando se transmitem os sinais de controlo da imagem e também os valor dos pixeis é demonstrado na imagem 3.2 na página 46 uma exemplo de transmissão de uma imagem gerada na FPGA. Antes de passar para descrição da geração da imagem passam a ser descritos os acrónimos apresentados na figura:

1. **HRES:** *Horizontal Resolution* é o parâmetro que define a resolução horizontal da imagem que vai ser gerada pelo bloco, ou seja o número de pixeis em cada linha de transmissão.

2. **HSW:** *Horizontal Sync Width* é o parâmetro que define o número de ciclos de relógio que o sinal de sincronização horizontal tem.

3. **HBP:** *Horizontal Back Porch* é o parâmetro que define o número de pixeis que não contêm informação útil (relativamente à cor dos mesmos) antes de começar a ser transmitida a linha de imagem.

4. **HFP:** *Horizontal Front Porch* é o parâmetro que define o número de pixeis que não contem informação útil depois de ser transmitida uma linha da imagem.

5. **VRES:** *Vertical Resolution* é o parâmetro que define a resolução vertical da imagem que vai ser gerada pelo bloco, por outras palavras é o número de linhas de pixeis a ser geradas.

6. **VSW:** *Vertical Sync Width* é o parâmetro que define o número de linhas horizontais que o sinal de sincronização vertical está ativo.

7. **VBP:** *Vertical Back Porch* é o parâmetro que define o número de linhas horizontais que não contém informação útil relativamente ao pixeis antes de começarem a ser transmitidas as linhas de pixeis.

8. **VFP:** *Vertical Front Porch* é o parâmetro que define o número de linhas horizontais que não contém informação útil relativamente ao pixeis depois de terem sido transmitidas todas as linhas horizontais da imagem.

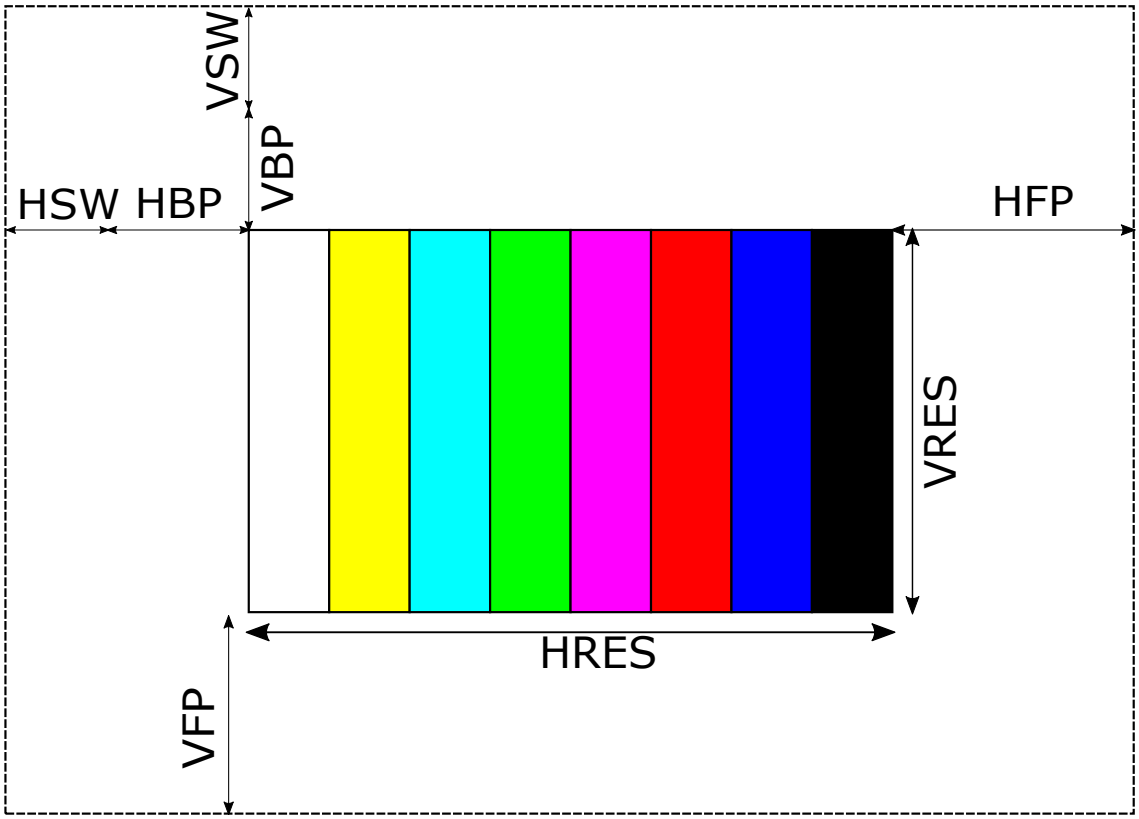

Figura 3.2: Exemplo de imagem gerada pelo modulo desenvolvido

Para gerar uma imagem em *FULL HD* cuja resolução é 1920x1080 pixeis e o sinal de relógio deve ter uma frequência de 148.5 MHz, foram o utilizados os seguintes valores para os parâmetro previamente descritos: HRES = 1920, HSW = 44, HBP = 44, HFP = 148, VRES = 1080, VSW = 5, VBP = 36 e VFP = 4.

A figura 3.3 na página 47 ilustra a maquina de estados desenvolvida para implementar a geração de uma barra a cores na FPGA.

Os registos VCOUNT E HCOUNT de decisão que se visualiza na figura correspondem a contadores que vão contanto pixel a pixel até ao fim de uma linha (no caso do HCOUNT) ou então de uma imagem inteira (no caso do VCOUNT). Os valores de HTOTAL e VTOTAL não são mais do que a soma de todo o tamanho dos dados na horizontal e na vertical respectivamente. Assim sendo, para este caso em especifico obtem-se os seguintes valores:

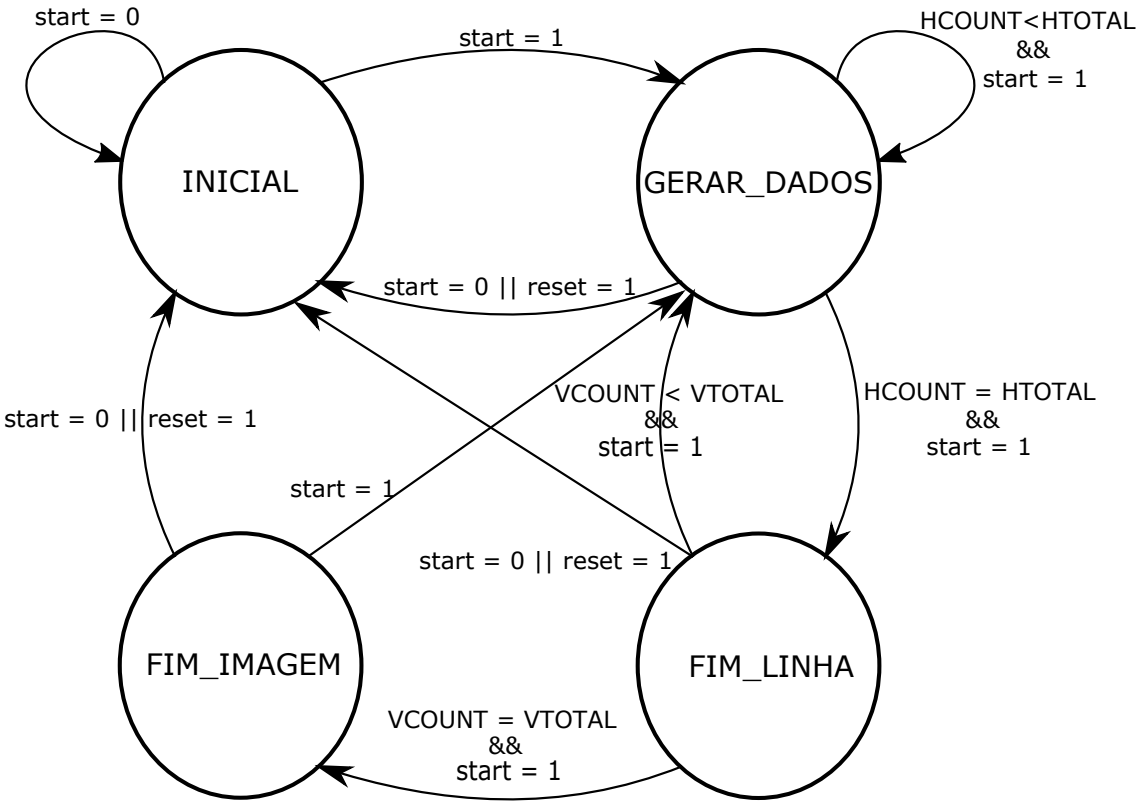

Figura 3.3: Máquina de estados para gerar uma barra de cores

- HTOTAL = HSW + HBP + HRES + HFP = 44 + 44 + 1920 + 148 = 2156
- VTOTAL = VSW + VBP + VRES + VFP = 5 + 36 + 1080 + 4 = 1125

Para além destes sinais de decisão para mudança de estado existem mais dois sinais no diagrama da máquina de estados presente na figura 3.3 que ainda não foram mencionados que são o *reset* e o *start*. Estes dois sinais são botões do utilizador que lhe permitem definir quando se pretende que a transmissão esteja ativa ou desativa (através do botão *start*) ou então quando se pretende restabelecer os dados originais da máquina de estados (através do botão *reset*).

Existem 4 estados nesta máquina eu consistem essencialmente em detecção do final de uma linha, e detecção do final de uma imagem e geração de dados. Os estados passam a ser descritos de seguida:

1. **Estado inicial:** Neste estado são configurados os parâmetros para o inicio de uma transmissão, ou seja, os valores de HCOUNT e VCOUNT são igualados ao valor total do tamanho na horizontal e na vertical respectivamente. Por outras palavras, os valores de HCOUNT e VCOUNT são igualados a HTOTAL e VTOTAL respectivamente. Isto acontece porque é possivel retornar a este estado estando em qualquer

um dos outros desde que seja pressionado o botão de *reset* ou então que a transmissão seja desligada pelo utilizador ($start = 0$).

2. **Estado para gerar dados:** Neste estado, ao flanco positivo do sinal de relógio do sistema, é incrementado o valor de HCOUNT e ao mesmo tempo são gerados os dados a serem transmitidos em cada ciclo de sinal de relógio, consoante o valor de HCOUNT e VCOUT. Quando o valor de HCOUNT se igualar ao valor de HTOTAL, então significa que foi transmitida uma linha inteira da imagem, e por isso a máquina transita de estado e o valor de VCOUNT volta a ser igualado a 1. O processo de geração de dados será explicado em XXXX

3. **Estado de fim de linha:** Quando este estado está ativo, então uma linha da imagem foi transmitida, o que implica que é necessário incrementar o valor de linhas totais transmitidas (incrementando 1 valor em VCOUNT) e ainda verificar se a transmissão de uma imagem completa está realizada. Caso o valor de VCOUNT se iguale ao valor de VTOTAL, então transita-se para o estado de fim de imagem, e coloca-se o valor de VCOUNT a 1. Caso contrário, então a máquina transita para o estado que estava anteriormente.

4. **Estado de fim de imagem** Quando este estado está ativo então significa que ambos os valores de HCOUNT e VCOUNT estão igualados a 1 e que por isso já foi transmitida uma imagem completa e como tal passa-se a transmitir uma próxima imagem, transitando novamente para o estado para gerar dados.

Quando a máquina de estados se encontra no estado para gerar dados, então os dados de controlo são gerados nas seguintes condições :

- **Sinal de sincronização vertical:** O sinal de sincronização vertical é um sinal que como já foi referido anteriormente indica o incio de transmissão de uma nova imagem, e por isso é ativado pela máquina de estados desenvolvida quando o valor em VCOUNT se igual ao valor de VTOTAL e quando o valor de HCOUNT se igual ao valor de HTOTAL, ou seja é ativado no final de uma imagem. Este sinal é ainda desligado quando o valor de VCOUNT se igual a VSW e o valor de HCOUNT se igual ao valor de HTOTAl, isto porque quando estas duas condições se verificam seignifica que o número de linhas em que o sinal de sincronização vertical deve estar ativo já terminou (é mesmo isso que o valor do parâmetro VSW define : *Vertical Sync Width*).

- **Sinal de sincronização horizontal:** O sinal de sincronização horizontal indica o inicio de uma nova linha e como tal deve ser ativo sempre que o valor de HCOUNT se igual e ao valor de HTOTAL (porque indica o fim da emissão de uma linha). Da mesma maneira, este sinal deve ser desativo sempre que o valor de HCOUNT se igual ao valor de HSW, isto porque este valor indica que o período de tempo que este sinal deve estar ativo terminou.

- **Sinal de dados ativos:** Este sinal deve estar ativo sempre que se estiver a transmitir pixeis válidos, e por isso sempre que as condições que serão de seguida apresentadas se verificarem:

  1. O valor de VCCOUNT é maior do que a soma entre VSW e VBP.
  2. O valor de VCOUNT é menor do que a soma entre VSW, VBP, VRES e 1.
  3. O valor de HCOUNT é maior do que a soma entre HSW, HBP subtraída de 1 valor.
  4. O valor de HCOUNT é menor do que a soma HSW, HBP e HRES.

  As duas primeira condições garantem que VCOUNT está na zona vertical que corresponde à transmissão de imagem na figura 3.2, e as duas ultimas condições garantem o mesmo mas na zona horizontal.

- **Valor dos pixeis:** Estes sinais correspondem a um barramento de 30 bits de uma imagem RGB com 10 bits por componente de cor. Como tal, estes valores devem corresponder a cores sempre o sinal de dados ativos estiver ligado e 0 sempre que estiver desligado.

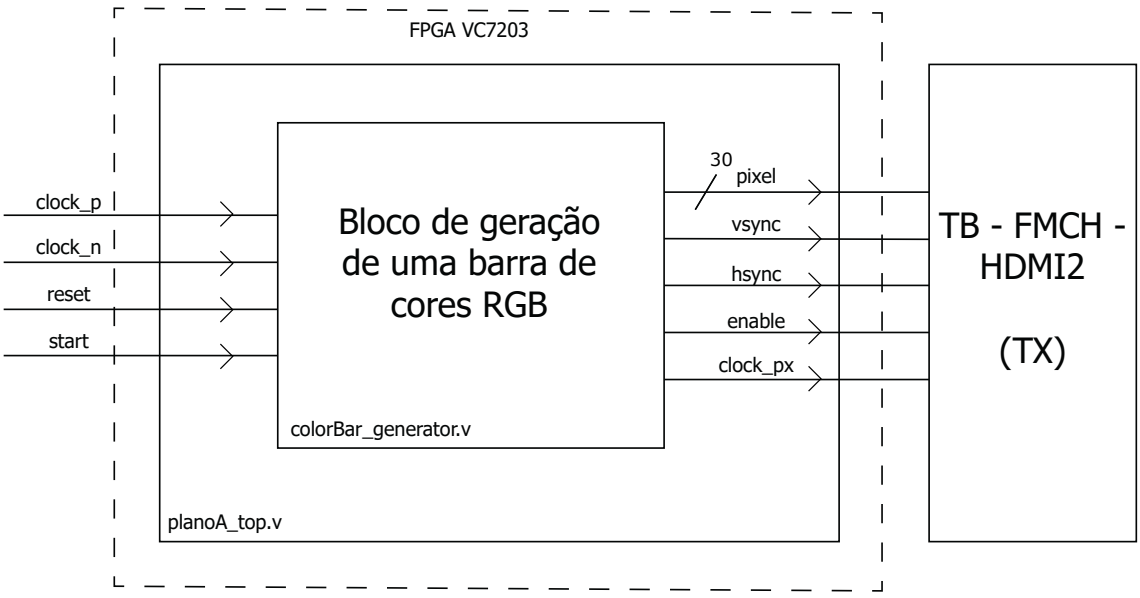

Figura 3.4: Diagrama de blocos de arquitetura implementada utilizando um bloco gerador de barra de cores

Na figura 3.4 é apresentado um diagrama de blocos da arquitetura implementada recorrendo a um bloco gerador de uma barra de cores. Este bloco foi implementado recorrendo-se à maquina de estados apresentada anteriormente.

Nas entradas do bloco estão ligados 4 sinais sendo que dois deles correspondem a um sinal de relógio diferencial de 200 MHz (*clock_p* corresponde ao sinal positivo e *clock_n* ao

sinal negativo), e os outros dois sinais, *start* e *reset*, são sinais relevantes para a máquina de estados do bloco de geração de barras de cores definidos pelo utilizador e, por isso, são atribuídos a botões da FPGA. O sinal de relógio diferencial ligado às entradas deste bloco é proveniente do oscilador presente na FPGA e irá alimentar uma modulo que coloca na sua saída um sinal de relógio de 148.5 MHz. Esse módulo foi criado através do IP disponibilizado no VIVADO *Clocking Wizard* que vem facilitar a geração de um sinal de relógio com a frequência pretendida tendo como uma base um sinal diferencial de 200 MHz. O sinal gerado, de 148.5 MHz, é o sinal de relógio principal do sistema uma vez que é a frequência necessária para gerar uma imagem em *FULL HD*, e como tal é a essa cadência que os sinais serão enviados para a placa HDMI transmissora e é esse ainda o sinal de relógio da mesma.

Relativamente às saída do módulo é possivel visualizar na imagem 3.4 que estas se encontram diretamente ligadas à placa transmissora HDMI através dos conectores FMC. Estes sinais são um barramento de 30 bits que corresponde ao pixel (*pixel*), o sinal de sincronização horizontal (*hsync*), o sinal de sincronização de vertical (*vsync*) e ainda o sinal de dados ativos (*enable*).

Para além do desenvolvimento do código em Verilog é necessário que as portas do modulo de topo, no caso desta arquitetura do modulo "planoA_top.v", estejam atribuidas a portas físicas da FPGA. Para tal é necessário definir onde estão as localizações das portas na FPGA (LOC) e criar um ficheiro de defina essas mesmas restrições fisicas. A tabela 3.9 na página 50 indica quais as localizações fisicas de cada porta existente no modulo de topo. No caso das portas que se conectam com a placa HDMI transmissora, estão representadas de forma abreviada, no entanto na tabela B.1 do anexo B é possivel encontrar todas as portas com mais detalhes e ainda com informação sobre a ligação à placa HDMI transmissora.

I/O	Sinal	LOC na FPGA	Banco na FPGA
I	clk_p	E19	38
I	clk_n	E18	38
I	reset	N41	19
I	start	E42	19
O	cll_px	E34	35
O	enable	K35	34
O	vsync	L31	34
O	hsync	M32	34
O	pixel[0]a[29]	(Ver anexo)	34 e 35

Tabela 3.9: Localização das portas de entrada e saída da arquitetura

O ficheiro com estas restrições fisicas gerado após a atribuição das mesmas é apresentado no sub-capitulo C.0.1.1 do anexo C. Para cada porta são atribuídas duas restrições: uma

que indica a localização fisica na FPGA da porta e outra que indica a norma da mesma (*IOSTANDARD*). A primeira permite atribuir a um determinado lugar físico da FPGA a porta que se pretende e a segunda define a norma dessa mesma porta para que todas as considerações que se tenham de ser tomadas relativamente a essa porta tenham em conta essa mesma norma.

Para além destas restrições fisicas geradas, são também geradas duas restrições temporais quanto aos sinais de relógio à entrada apresentadas no sub-capitulo C.0.1.2 do anexo C. As retrições temporais existentes definem que nas portas de entrada do sinal de relógio diferencial existe um sinal com uma frequência de 200 MHz (periodo de 5ns). Isto porque este sinal de relógio é um sinal primário e como tal é importante que a ferramenta de implementação saiba o seu valor para poder garantir que toda a arquitetura cumpre os requisitos temporais.

Após a definição de todas as restrições e escrita do cófigo em verilg, a arquitetura desenvolvida foi devidamente implementada na FPGA e testada obtendo-se o previsto.

3.2.2 Transmissão de imagem entre dispositivos HDMI

Na arquitetura desenvolvida que é apresentada neste sub-capitulo são utilizadas as placas HDMI recetora e transmissora ambas configuradas por defeito e procede-se à transmissão de uma imagem entre dispositivos HDMI. O objetivo do desenvolvimento desta arquitetura consiste em obter uma ligação entre dois dispositivos ligados às placas HDMI de uma imagem RGB de 10 bits.

Foi desenvolvida uma arquitetura que recebe à cadência do sinal de relógio HDMI proveniente da placa (neste caso em especifico como é uma imagem *FULL HD* é uma frequência de 148,5 MHz) os o resto dos sinais provenientes da mesma, mais especificamente o valor de *pixel*, *vsync*, *hsync* e *enable*. A imagem 3.5 da página 52 ilustra o diagrama de blocos da arquitetura desenvolvida.

É possivel visualizar que nesta arquitetura, tal como já acontecia na anterior, existem na sua saída os os sinais que são enviados para a placa HDMI transmisora e para além disso existem também os sinais provenientes da placa HDMI recetora. Também à semelhança da arquitetura descrita em 3.2.1 existem mais 4 sinais provenientes do exterior: o sinal de relógio diferencia de 200 MHz constituido pelo par positivo $clock\_p$ e pelo par negativo $clock\_n$, e ainda o sinal *start* que define o incio da transmissão da barra de cores em vez dos sinais provebientes da fonte HDMI e, por fim, o sinal de *reset* que permite restabelecer os dados originais do sistema caso se pretenda.

Os sinais que são recebidos à entrada são lidos para registos sincronos com o sinal de relógio proveniente da entrada (da placa HDMI recetora). Quando o sinal definido pelo utilizador *start* está ativo os sinais seleccionados pelo multiplexador visivel na figura 3.5 são os sinais provenientes do modulo desenvolvido anteriormente que gera uma barra de cores. Esse mesmo sinal de start está ligado à entrada do bloco gerador da barra de cores para que quando ativo gere a imagem. Quando o sinal *start* está desativo então obtém-se

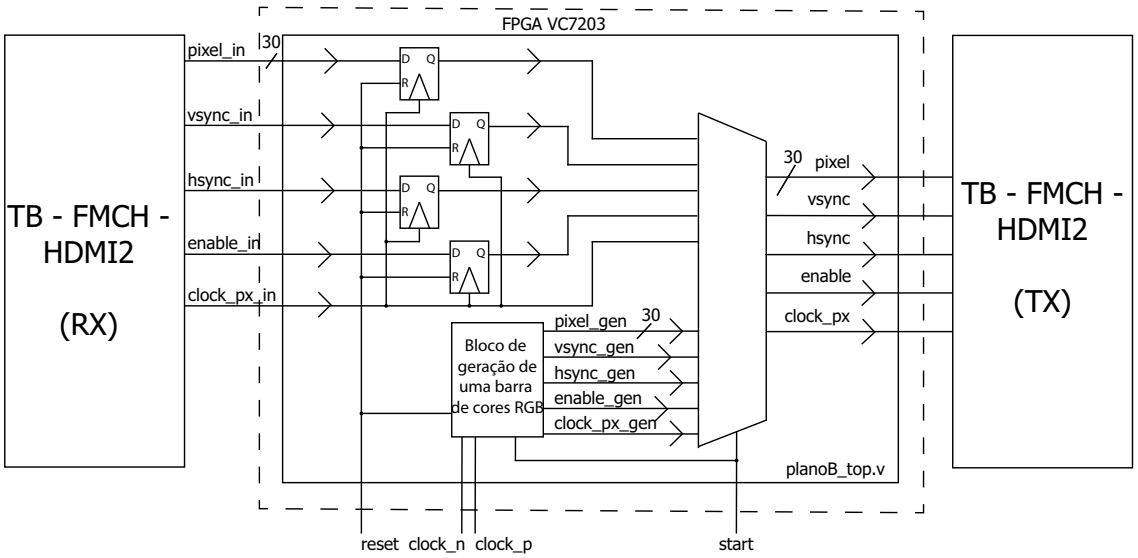

Figura 3.5: Diagram de blocos da arquitetura desenvolvida para transmitir imagem entre dispositivos HDMI

uma ligação entre as placas HDMI recetora e transmissora pois os sinais seleccionados pelo multiplexador são o sinais provenientes da entrada do modulo. Sempre que o sinal de *reset* é ativo então todos os dados são repostos aos originais, como por exemplo os registos voltam ao estado original e também o bloco que produz a barra de cores.

A tabela 3.10 na página 53 especifica as localizações fisicas da FPGA que foram atribuidas a cada porta do modulo desenvolvido e ainda o banco ao qual pertencem. As portas que fazem conexão com as placas HDMI transmissora e recetora são apresentadas nesta tabela de forma abreviada, no entanto na tabela B.2 do anexo B é possivel encontrar mais detalhamente as localizações dessas portas na FPGA bem como informação relativamente a esses sinais nas placas HDMI transmissora e recetora.

I/O	Sinal	LOC na FPGA	Banco na FPGA
I	clk_p	E19	38
I	clk_n	E18	38
I	reset	N41	19
I	start	E42	19
I	clk_px_in	AJ32	14
I	enable_in	AN38	15
I	vsync_in	AU38	15
I	hsync_in	AU39	15
I	pixel_in[0] a [29]	(Ver anexo)	14 e 15
O	clock_px	E34	35
O	enable	K35	34

I/O	Sinal	LOC na FPGA	Banco na FPGA
O	vsync	L31	34
O	hsync	M32	34
O	pixel[0]a[29]	(Ver anexo)	34 e 35

Tabela 3.10: Localização das entradas e saídas das portas da arquitetura

A atribuição destas mesmas localizações das portas gerou um ficheiro de restrições físicas que é apresentado na secção C.0.2.1 do anexo C. Relativamente a estas retrições físicas, existem para cada porta do modulo duas restrições: uma que define a localização fisica e outra a norma da fisica, tal como já foi mencionado anteriormente.

Quanto às restrições temporais aplicadas ao sistema são idênticas às mesmas aplicadas na arquitetura desenvolvida anteriormente e estáo presentes em C.0.2.2 no anexo C.

Após sintese e implementação do código desenvolvido em verilog juntamente com as restrições aplicadas, a FPGA foi programada com esta arquitetura e foram obtidos os resultados esperados: obteve-se uma transmissão de imagem entre dois dispositivos HDMI. Foi utilizado um computador com saída HDMI como fonte de imagem conectado a placa HDMI RX, e desta maneira obteve-se no dispositivo HDMI de destino, que neste caso foi um monitor com entrada HDMI, conectado à placa HDMI transmissora a imagem transmitida.

3.2.3 Transmissão de imagem e som entre dispositivos HDMI

Após se obter uma ligação entre dois dispositivos HDMI de uma imagem, procedeu-se ao desenvolvimento de uma arquitetura capaz de trasmitir imagem e som. Para tal foi necessário reconfigurar as placas HDMI, tal como mencionado anteriormente, para a configuração que suporta apenas um canal mas que permite a transmissão de áudio em formato I^2S. As características desta configuração são apresentadas na secção 3.1.1.2 na página 39 deste documento, mas é de notar que as imagens poderão ser transmitidas e recebidas em dois tipos de formatos (RGB ou YCbCr) e ainda com 8, 10 ou 12 bits por cor (dependendo da configuração dos interruptores das placas HDMI que estão especificados no sub-capitulo 3.1.2.2). Neste caso em especifico serão utilizados 12 bits por cor o que preferá um total de 36 bits por pixel.

Na imagem 3.6 na página 54 é ilustrado um diagrama de blocos da arquitetura desenvolvida para se realizar a transmissão de imagem e som entre dois dispositivos HDMI.

Através de uma breve observação do diagrama de blocos ilustrado é possivel concluir que existem mais portas tanto de entrada como de saída nesta arquitetura comparativamente às arquiteturas descritas previamente. Isto deve-se ao facto de agora haver a transmissão do som o que implica a transmissão de mais sinais. Assim sendo, na entrada encontram-se os sinais relativos às imagens à semelhança das arquiteturas anteriores: 36 bits de pixel, sinais de sincronização horizontal*hsync* e vertical (*vsync*) e ainda sinal que indica sinais

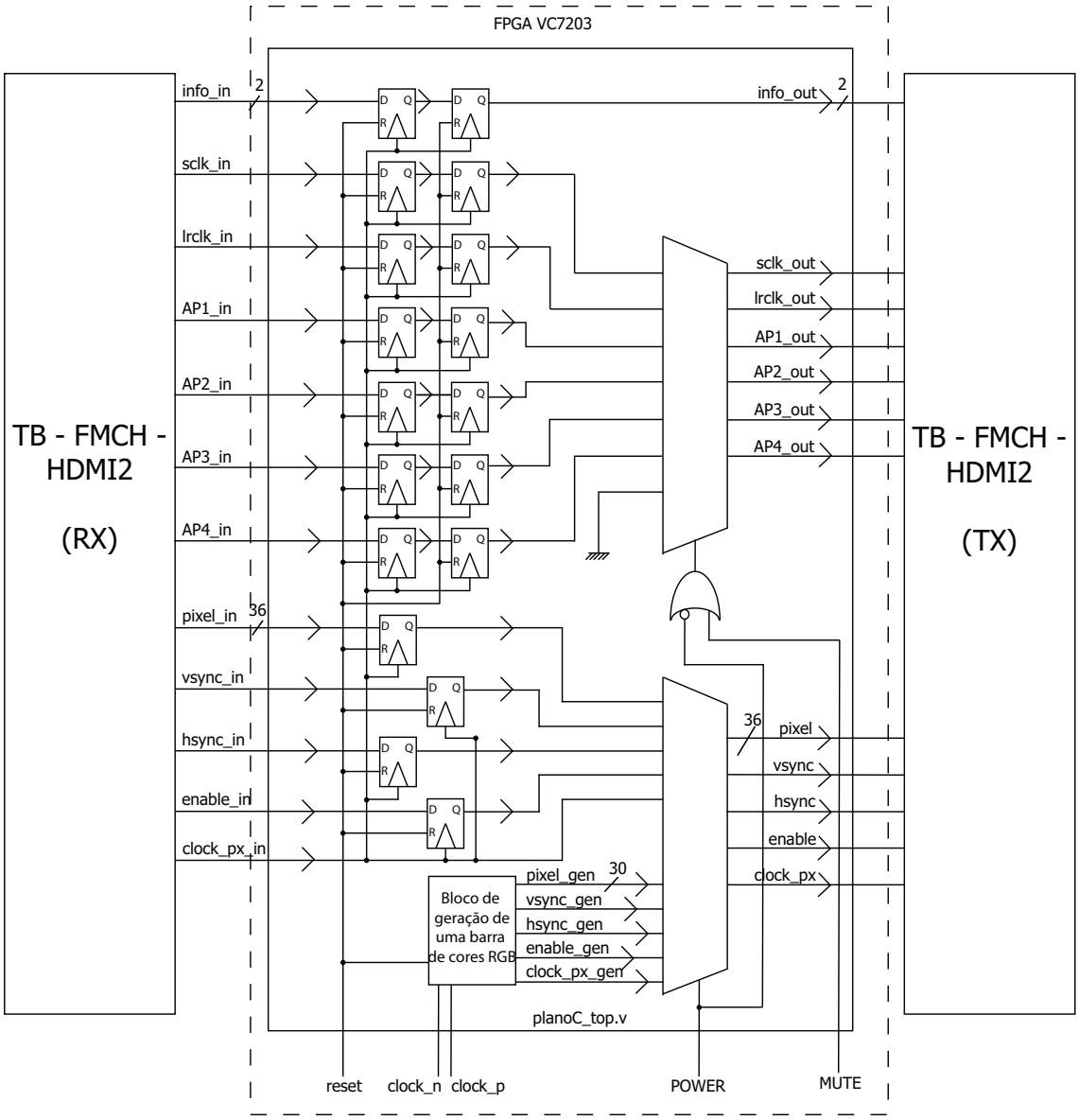

Figura 3.6: Diagrama de blocos da arquitetura desenvolvida para transmitir imagem e som entre dispositivos HDMI

de pixel ativos(*enable*). Para além destes sinais provenientes da placa HDMI recetora, são recebidos os sinais referentes ao som: o sinal de relógio dos dados em série (*sclk*), o sinal referente à seleccção do canal de audio esquerdo ou direto (*lrclk*) e ainda os sinais que transportam os dados de som (de AP1 a AP4). Para além de dados de imagem e som há também um barramento de 2 bits que contém informação relativamente ao tipo de video que é transmitido (tal como já referido na secção 3.1.1.2). Todos estes sinais que se encontram na entrada do modulo encontram-se também na saída pois é necessário enviar todos para a placa HDMI transmissora, no entanto o processo de transmissão dos mesmos passa a ser descrito.

Para além destes sinais provenientes e que são enviados para as palcas HDMI existem ainda mais portas do bloco. Existe o tipico sinal de relógio diferencia de 200 MHz definido na porta com *clock_p* pelo sinal positivo e como *clock_n* pelo sinal negativo. Este sinal de relógio proveniente do oscilador da FPGA serve como fonte para poder ser reproduzido o sinal de 148.5 MHz (frequência de uma imagem *FULL HD*) que irá produzir a barra de cores no modulo "*coloBar_generator*". As outras três portas ainda não mencionadas são três sinais definidos pelo utilizador através de interruptores e botões. O sinal de *reset* serve para repor todos os dados originais do sistema caso o utilizador pretenda. O sinal *POWER* é o sinal que define a transmissão dos dados provenientes da placa HDMI recetora ou então da barra de cores, e o sinal *MUTE* define a transmissão ou não dos sinais de som.

Os sinais de entrada relativos à imagem, tal como nas arquiteturas anteriores, são lidos para registos síncronos com o sinal de relógio de imagem (que no caso desta demonstração será 148.5 MHz pois são transmitidas imagens em *FULL HD*). Estes mesmo sinais são enviados para a placa HDMI transmissora caso o sinal *POWER* esteja ativo. Este sinal é um sinal que quando está desativo em vez de enviar para as saídas do modulo os sinais recebidos da placa HDMI recetora, envia os dados provenientes do bloco que reproduz uma barra de cores.

Relativamente aos dados referentes ao som, estes são também lidos para registos síncronos com o sinal de relógio referente à imagem proveniente da fonte HDMI pois numa fase mais avançada do projeto tal simplificação virá facilitar a transmissão dos dados em série a alta velocidade. Porém é necessário ter em consideração que estes dados variam a uma taxa que não é síncrona com o sinal de relógio da imagem, pois este possui uma frequência de 148,5 MHz para uma imagem no formato *FULL HD*, e os dados de som variam a uma frequência de 3,072 MHz (tal como mencionado no sub-capitulo 3.1.1.2). Assim sendo é necessário tomar as devidas precauções quando se faz o cruzamento entre estes dois domínios de relógio para que não sejam captados dados dos registos quando estes se encontram num estado de meta-estabilidade. Assim sendo, tal como sugerido em [16], são utilizados dois registos síncronos com o sinal de relógio referente à imagem para que não haja propagações de erros.

O sinal de áudio enviado para as placas HDMI transmissora está dependente do valor do sinal de *POWER* e *MUTE*. Caso *MUTE* esteja ativo o sinal de som não é transmitido e as entradas da placa HDMI são definidas com zero. O mesmo acontece quando o sinal de *POWER* esteja desativo uma vez que indica que a transmissão da parte da placa HDMI recetora está desligada.

Mais uma vez é necessário definir as localizações fisicas de cada porta de entrada e saída do modulo principal, e como tal é possivel encontrar descritas essas mesmas atribuições na tabela 3.11 na página 56. Nessa mesma tabela os sinais que se conectam as placas HDMI são brevemente descritas, no entanto na tabela B.3 do anexo B essas mesmas conexões são descritas com mais detalhe.

I/O	Sinal	LOC na FPGA	Banco na FPGA
I	clk_p	E19	38
I	clk_n	E18	38
I	reset	N41	19
I	POWER	E42	19
I	MUTE	G41	19
I	clock_p_in	AJ32	14
I	vsync_in	AU38	15
I	hsync_in	AU39	15
I	enable_in	AN38	15
I	pixel_in[0] a [35]	(Ver anexo)	14 e 15
I	sclk_in	AJ37	14
I	lrclk_in	AL35	14
I	AP1_in	AL37	14
I	AP2_in	AP35	14
I	AP3_in	AM37	14
I	AP4_in	AH33	14
I	info_in[0]	AV38	15
I	info_in[1]	AV39	15
O	clock_p_out	E34	35
O	vsync_out	L31	34
O	hsync_out	M32	34
O	enable_out	K35	34
O	pixel_out[0] a [35]	(Ver anexo)	34 e 35
O	sclk_out	A34	35
O	lrclk_out	B33	35
O	AP1_out	A36	35
O	AP2_out	C39	35
O	AP3_out	B38	35
O	AP4_out	D32	35
O	info_out[0]	K32	34
O	info_out[1]	L32	34

Tabela 3.11: Localização das entradas e saídas das portas da arquitetura

Para que a ferramenta de implementação reconheça as localizações fisicas da FPGA às quais são atribuídas as portas do bloco é gerado um ficheiro de restrições físicas semelhantes aos ficheiros gerados para as outras arquiteturas apresentadas até agora neste documento. O ficheiro que contém as restrições fisicas pode ser encontrado em C.1.0.1 no anexo C.

Também à semelhança das arquiteturas anteriormente descritas foram geradas duas restrições temporais relativamente ao sinal de relógio diferencial à entrada que definem que na entrada existe um sinal com uma frequência de 200 MHz. Estas são apresentadas em C.1.0.2 no anexo C.

Esta arquitetura foi devidamente implementada na FPGA e testada, obtendo-se os resultados esperados. Apesar de os sinais de audio provenientes da placa HDMI recetora serem sincronos com sinal de relógio que não é o dos pixeis, o cruzamento estes dois dominios de relógio diferentes não apresentou qualquer tipo de problemas uma vez que ao ouvido humano os pequenos atrasados que possam eventualemente acontecer não são percetiveis.

Capítulo 4

Transmissão dos dados em série

Capítulo 5

Conclusões e Trabalho Futuro

Adicionar as siglas:
PROM - Programmable read-only memory
SPDIF
imp -> https://www.xilinx.com/support/answers/64340.html

Anexo A

Descrição dos pinos das placas HDMI

A.1 Configuração por *default*

PIN	FPGA ->FMC (RX)	FMC->FPGA (TX)	Descrição
CLK0_M2C_P	RX#O_LLC	TX#O_DCLK	*clock* dos pixeis
LA00_P_CC	RX#0_VSYNC	TX#0_VSYNC	sincr. Vertical
LA01_P_CC	RX#0_HSYNC	TX#0_HSYNC	sincr. Horizontal
LA02_P	RX#0_DE	TX#0_DE	sinal de dados ativos
LA03_P	RX#0_P0	TX#0_D0	Pixel de imagem B[0]
LA04_P	RX#0_P1	TX#0_D1	Pixel de imagem B[1]
LA05_P	RX#0_P2	TX#0_D2	Pixel de imagem B[2]
LA06_P	RX#0_P3	TX#0_D3	Pixel de imagem B[3]
LA07_P	RX#0_P4	TX#0_D4	Pixel de imagem B[4]
LA08_P	RX#0_P5	TX#0_D5	Pixel de imagem B[5]
LA09_P	RX#0_P6	TX#0_D6	Pixel de imagem B[6]
LA10_P	RX#0_P7	TX#0_D7	Pixel de imagem B[7]
LA11_P	RX#0_P8	TX#0_D8	Pixel de imagem B[8]
LA12_P	RX#0_P9	TX#0_D9	Pixel de imagem B[9]
LA13_P	RX#0_P10	TX#0_D10	Pixel de imagem G[0]
LA14_P	RX#0_P11	TX#0_D11	Pixel de imagem G[1]
LA15_P	RX#0_P12	TX#0_D12	Pixel de imagem G[2]
LA16_P	RX#0_P13	TX#0_D13	Pixel de imagem G[3]
LA17_P_CC	RX#0_P14	TX#0_D14	Pixel de imagem G[4]
LA18_P_CC	RX#0_P15	TX#0_D15	Pixel de imagem G[5]
LA19_P	RX#0_P16	TX#0_D16	Pixel de imagem G[6]
LA20_P	RX#0_P17	TX#0_D17	Pixel de imagem G[7]
LA21_P	RX#0_P18	TX#0_D18	Pixel de imagem G[8]
LA22_P	RX#0_P19	TX#0_D19	Pixel de imagem G[9]

PIN	FPGA ->FMC (RX)	FMC->FPGA (TX)	Descrição
LA23_P	RX#0_P20	TX#0_D20	Pixel de imagem R[0]
LA24_P	RX#0_P21	TX#0_D21	Pixel de imagem R[1]
LA25_P	RX#0_P22	TX#0_D22	Pixel de imagem R[2]
LA26_P	RX#0_P23	TX#0_D23	Pixel de imagem R[3]
LA27_P	RX#0_P24	TX#0_D24	Pixel de imagem R[4]
LA28_P	RX#0_P25	TX#0_D25	Pixel de imagem R[5]
LA29_P	RX#0_P26	TX#0_D26	Pixel de imagem R[6]
LA30_P	RX#0_P27	TX#0_D27	Pixel de imagem R[7]
LA31_P	RX#0_P28	TX#0_D28	Pixel de imagem R[8]
LA32_P	RX#0_P29	TX#0_D29	Pixel de imagem R[9]

Tabela A.1: Localização dos pinos de dados utilizados em TB-FMCH-HDMI2 configurado por *default*

A.2 Suporte de um canal de imagem e áudio

PIN	FPGA-> (RX)	FMC -> FPGA (TX)	Descrição
CLK0_M2C_P	RX#O_LLC	TX#O_DCLK	Sinal de relógio dos pixeis
LA00_P_CC	RX#0_VSYNC	TX#0_VSYNC	Sincronização vertical
LA01_P_CC	RX#0_HSYNC	TX#0_HSYNC	Sincronização horizontal
LA02_P	RX#0_DE	TX#0_DE	Sinal de dados ativos
LA03_P	RX#0_P0	TX#0_D0	Pixel de Imagem Cb[0]/B[0]
LA04_P	RX#0_P1	TX#0_D1	Pixel de Imagem Cb[1]/B[1]
LA05_P	RX#0_P2	TX#0_D2	Pixel de Imagem Cb[2]/B[2]
LA06_P	RX#0_P3	TX#0_D3	Pixel de Imagem Cb[3]/B[3]
LA07_P	RX#0_P4	TX#0_D4	Pixel de Imagem Cb[4]/B[4]
LA08_P	RX#0_P5	TX#0_D5	Pixel de Imagem Cb[5]/B[5]
LA09_P	RX#0_P6	TX#0_D6	Pixel de Imagem Cb[6]/B[6]
LA10_P	RX#0_P7	TX#0_D7	Pixel de Imagem Cb[7]/B[7]
LA11_P	RX#0_P8	TX#0_D8	Pixel de Imagem Cb[8]/B[8]
LA12_P	RX#0_P9	TX#0_D9	Pixel de Imagem Cb[9]/B[9]
LA13_P	RX#0_P10	TX#0_D10	Pixel de Imagem Cb[10]/B[10
LA14_P	RX#0_P11	TX#0_D11	Pixel de Imagem Cb[11]/B[1
LA15_P	RX#0_P12	TX#0_D12	Pixel de Imagem Y[0]/G[0]
LA16_P	RX#0_P13	TX#0_D13	Pixel de Imagem Y[1]/G[1]
LA17_P_CC	RX#0_P14	TX#0_D14	Pixel de Imagem Y[2]/G[2]
LA18_P_CC	RX#0_P15	TX#0_D15	Pixel de Imagem Y[3]/G[3]
LA19_P	RX#0_P16	TX#0_D16	Pixel de Imagem Y[4]/G[4]

PIN	FPGA-> (RX)	FMC -> FPGA (TX)	Descrição
LA20_P	RX#0_P17	TX#0_D17	Pixel de Imagem Y[5
LA21_P	RX#0_P18	TX#0_D18	Pixel de Imagem Y[6
LA22_P	RX#0_P19	TX#0_D19	Pixel de Imagem Y[7
LA23_P	RX#0_P20	TX#0_D20	Pixel de Imagem Y[8
LA24_P	RX#0_P21	TX#0_D21	Pixel de Imagem Y[9
LA25_P	RX#0_P22	TX#0_D22	Pixel de Imagem Y[10
LA26_P	RX#0_P23	TX#0_D23	Pixel de Imagem Y[11
LA27_P	RX#0_P24	TX#0_D24	Pixel de Imagem Cr[0
LA28_P	RX#0_P25	TX#0_D25	Pixel de Imagem Cr[1
LA29_P	RX#0_P26	TX#0_D26	Pixel de Imagem Cr[2
LA30_P	RX#0_P27	TX#0_D27	Pixel de Imagem Cr[3
LA31_P	RX#0_P28	TX#0_D28	Pixel de Imagem Cr[4
LA32_P	RX#0_P29	TX#0_D29	Pixel de Imagem Cr[5
LA00_N_CC	RX#0_InputVideoStatus[0]	TX#0_InputVideoStatus[0]	Formato do video (2I
LA01_N_CC	RX#0_InputVideoStatus[1]	TX#0_InputVideoStatus[1]	Formato do video (2I
LA19_N	RX#0_MCLK	TX#0_MCLK	*Master Clock* de s
LA20_N	RX#0_SCLK	TX#0_SCLK	*Serial Clock* de sc
LA21_N	RX#0_AP0	TX#0_AP0	Dados de Som SPI
LA22_N	RX#0_AP1	TX#0_AP1	Dados de Som I2S
LA23_N	RX#0_AP2	TX#0_AP2	Dados de Som I2S
LA24_N	RX#0_AP3	TX#0_AP3	Dados de Som I2S
LA25_N	RX#0_AP4	TX#0_AP4	Dados de Som I2S
LA26_N	RX#0_AP5	TX#0_AP5	Sinal de relógio LR (lef
LA27_N	RX#0_P30	TX#0_D30	Pixel de Imagem Cr[6
LA28_N	RX#0_P31	TX#0_D31	Pixel de Imagem Cr[7
LA29_N	RX#0_P32	TX#0_D32	Pixel de Imagem Cr[8
LA30_N	RX#0_P33	TX#0_D33	Pixel de Imagem Cr[9
LA31_N	RX#0_P34	TX#0_D34	Pixel de Imagem Cr[10
LA32_N	RX#0_P35	TX#0_D35	Pixel de Imagem Cr[11

Tabela A.2: Localização dos pinos de dados utilizados em TB-FMCH-HDMI2 com a configuração de um canal e suporte de audio

A.3 Suporte de dois canais de imagem melhorado

PIN	FPGA-> (RX)	FMC -> FPGA (TX)	Descrição
CLK0_M2C_P	RX#O_LLC	TX#O_DCLK	Sinal de relógio dos pixeis
CLK1_M2C_N	RX#1_LLC	TX#1_DCLK	Sinal de relógio dos pixeis

PIN	**FPGA-> (RX)**	**FMC -> FPGA (TX)**	**Descrição**
LA00_P_CC	RX#0_VSYNC	TX#0_VSYNC	Sincronização vertical
LA01_P_CC	RX#0_HSYNC	TX#0_HSYNC	Sincronização horizontal
LA02_P	RX#0_DE	TX#0_DE	Sinal de dados ativos
LA03_P	RX#0_P0	TX#0_D0	Pixel de Imagem Cb[0]/B[0]
LA04_P	RX#0_P1	TX#0_D1	Pixel de Imagem Cb[1]/B[1]
LA05_P	RX#0_P2	TX#0_D2	Pixel de Imagem Cb[2]/B[2]
LA06_P	RX#0_P3	TX#0_D3	Pixel de Imagem Cb[3]/B[3]
LA07_P	RX#0_P4	TX#0_D4	Pixel de Imagem Cb[4]/B[4]
LA08_P	RX#0_P5	TX#0_D5	Pixel de Imagem Cb[5]/B[5]
LA09_P	RX#0_P6	TX#0_D6	Pixel de Imagem Cb[6]/B[6]
LA10_P	RX#0_P7	TX#0_D7	Pixel de Imagem Cb[7]/B[7]
LA11_P	RX#0_P8	TX#0_D8	Pixel de Imagem Cb[8]/B[8]
LA12_P	RX#0_P9	TX#0_D9	Pixel de Imagem Cb[9]/B[9]
LA13_P	RX#0_P10	TX#0_D10	Pixel de Imagem Y[0]/G[0]
LA14_P	RX#0_P11	TX#0_D11	Pixel de Imagem Y[1]/G[1]
LA15_P	RX#0_P12	TX#0_D12	Pixel de Imagem Y[2]/G[2]
LA16_P	RX#0_P13	TX#0_D13	Pixel de Imagem Y[3]/G[3]
LA17_P_CC	RX#0_P14	TX#0_D14	Pixel de Imagem Y[4]/G[4]
LA18_P_CC	RX#0_P15	TX#0_D15	Pixel de Imagem Y[5]/G[5]
LA19_P	RX#0_P16	TX#0_D16	Pixel de Imagem Y[6]/G[6]
LA20_P	RX#0_P17	TX#0_D17	Pixel de Imagem Y[7]/G[7]
LA21_P	RX#0_P18	TX#0_D18	Pixel de Imagem Y[8]/G[8]
LA22_P	RX#0_P19	TX#0_D19	Pixel de Imagem Y[9]/G[9]
LA23_P	RX#0_P20	TX#0_D20	Pixel de Imagem Cr[0]/R[0]
LA24_P	RX#0_P21	TX#0_D21	Pixel de Imagem Cr[1]/R[1]
LA25_P	RX#0_P22	TX#0_D22	Pixel de Imagem Cr[2]/R[2]
LA26_P	RX#0_P23	TX#0_D23	Pixel de Imagem Cr[3]/R[3]
LA27_P	RX#0_P24	TX#0_D24	Pixel de Imagem Cr[4]/R[4]
LA28_P	RX#0_P25	TX#0_D25	Pixel de Imagem Cr[5]/R[5]
LA29_P	RX#0_P26	TX#0_D26	Pixel de Imagem Cr[6]/R[6]
LA30_P	RX#0_P27	TX#0_D27	Pixel de Imagem Cr[7]/R[7]
LA31_P	RX#0_P28	TX#0_D28	Pixel de Imagem Cr[8]/R[8]
LA32_P	RX#0_P29	TX#0_D29	Pixel de Imagem Cr[9]/R[9]
CLK0_M2C_N	RX#0_Input video status[0]	TX#0_Input video status[0]	Formato do video (2D/3D)
CLK1_M2C_N	RX#0_Input video status[1]	TX#0_Input video status[1]	Formato do video (2D/3D)

PIN	**FPGA-> (RX)**	**FMC -> FPGA (TX)**	**Descrição**
LA33_P	RX#1_Input video status[0]	TX#1_Input video status[0]	Formato do video (2D/3D)
LA33_N	RX#1_Input video status[1]	TX#1_Input video status[1]	Formato do video (2D/3D)
LA00_N_CC	RX#1_VSYNC	TX#1_VSYNC	Sincronização vertical
LA01_N_CC	RX#1_HSYNC	TX#1_HSYNC	Sincronização horizontal
LA02_N	RX#1_DE	TX#1_DE	Sinal de dados ativos
LA03_N	RX#1_P0	TX#1_D0	Pixel de Imagem B[0]
LA04_N	RX#1_P1	TX#1_D1	Pixel de Imagem B[1]
LA05_N	RX#1_P2	TX#1_D2	Pixel de Imagem B[2]
LA06_N	RX#1_P3	TX#1_D3	Pixel de Imagem B[3]
LA07_N	RX#1_P4	TX#1_D4	Pixel de Imagem B[4]
LA08_N	RX#1_P5	TX#1_D5	Pixel de Imagem B[5]
LA09_N	RX#1_P6	TX#1_D6	Pixel de Imagem B[6]
LA10_N	RX#1_P7	TX#1_D7	Pixel de Imagem B[7]
LA11_N	RX#1_P8	TX#1_D8	Pixel de Imagem B[8]
LA12_N	RX#1_P9	TX#1_D9	Pixel de Imagem B[9]
LA13_N	RX#1_P10	TX#1_D10	Pixel de Imagem G[0]
LA14_N	RX#1_P11	TX#1_D11	Pixel de Imagem G[1]
LA15_N	RX#1_P12	TX#1_D12	Pixel de Imagem G[2]
LA16_N	RX#1_P13	TX#1_D13	Pixel de Imagem G[3]
LA17_N_CC	RX#1_P14	TX#1_D14	Pixel de Imagem G[4]
LA18_N_CC	RX#1_P15	TX#1_D15	Pixel de Imagem G[5]
LA19_N	RX#1_P16	TX#1_D16	Pixel de Imagem G[6]
LA20_N	RX#1_P17	TX#1_D17	Pixel de Imagem G[7]
LA21_N	RX#1_P18	TX#1_D18	Pixel de Imagem G[8]
LA22_N	RX#1_P19	TX#1_D19	Pixel de Imagem G[9]
LA23_N	RX#1_P20	TX#1_D20	Pixel de Imagem R[0]
LA24_N	RX#1_P21	TX#1_D21	Pixel de Imagem R[1]
LA25_N	RX#1_P22	TX#1_D22	Pixel de Imagem R[2]
LA26_N	RX#1_P23	TX#1_D23	Pixel de Imagem R[3]
LA27_N	RX#1_P24	TX#1_D24	Pixel de Imagem R[4]
LA28_N	RX#1_P25	TX#1_D25	Pixel de Imagem R[5]
LA29_N	RX#1_P26	TX#1_D26	Pixel de Imagem R[6]
LA30_N	RX#1_P27	TX#1_D27	Pixel de Imagem R[7]
LA31_N	RX#1_P28	TX#1_D28	Pixel de Imagem R[8]
LA32_N	RX#1_P29	TX#1_D29	Pixel de Imagem R[9]

PIN	FPGA-> (RX)	FMC -> FPGA (TX)	Descrição

Tabela A.3: Localização dos pinos de dados utilizados em TB-FMCH-HDMI2 com a configuração de suporte de dois canais melhorado

Anexo B

Localizações das portas provenientes de TB-HDMI

B.1 Transmissão de uma imagem gerada na FPGA

I/O	Sinal	LOC na FPGA	Banco na FPGA	Nome na placa HDMI	PIN da placa HDMI
O	clk_px	E34	35	TX#O_DCLK	CLK0_M2C_P
O	enable	K35	34	TX#0_DE	LA02_P
O	vsync	L31	34	TX#0_VSYNC	LA00_P_CC
O	hsync	M32	34	TX#0_HSYNC	LA01_P_CC
O	pixel[0]	J32	34	TX#0_D0	LA03_P
O	pixel[1]	K33	34	TX#0_D1	LA04_P
O	pixel[2]	L34	34	TX#0_D2	LA05_P
O	pixel[3]	M33	34	TX#0_D3	LA06_P
O	pixel[4]	H34	34	TX#0_D4	LA07_P
O	pixel[5]	K29	34	TX#0_D5	LA08_P
O	pixel[6]	J30	34	TX#0_D6	LA09_P
O	pixel[7]	L29	34	TX#0_D7	LA10_P
O	pixel[8]	J31	34	TX#0_D8	LA11_P
O	pixel[9]	M28	34	TX#0_D9	LA12_P
O	pixel[10]	R28	34	TX#0_D10	LA13_P
O	pixel[11]	N28	34	TX#0_D11	LA14_P
O	pixel[12]	R30	34	TX#0_D12	LA15_P
O	pixel[13]	U31	34	TX#0_D13	LA16_P
O	pixel[14]	C35	35	TX#0_D14	LA17_P_CC
O	pixel[15]	D35	35	TX#0_D15	LA18_P_CC
O	pixel[16]	B36	35	TX#0_D16	LA19_P

I/O	Sinal	LOC na FPGA	Banco na FPGA	Nome na placa HDMI	PIN da placa HDMI
O	pixel[17]	B34	35	TX#0_D17	LA20_P
O	pixel[18]	B39	35	TX#0_D18	LA21_P
O	pixel[19]	A35	35	TX#0_D19	LA22_P
O	pixel[20]	C38	35	TX#0_D20	LA23_P
O	pixel[21]	B37	35	TX#0_D21	LA24_P
O	pixel[22]	E32	35	TX#0_D22	LA25_P
O	pixel[23]	B32	35	TX#0_D23	LA26_P
O	pixel[24]	E33	35	TX#0_D24	LA27_P
O	pixel[25]	C33	35	TX#0_D25	LA28_P
O	pixel[26]	G32	35	TX#0_D26	LA29_P
O	pixel[27]	F36	35	TX#0_D27	LA30_P
O	pixel[28]	F34	35	TX#0_D28	LA31_P
O	pixel[29]	H33	35	TX#0_D29	LA32_P

Tabela B.1: Localização das portas de entrada e saída da arquitetura de transmisão de uma barra de cores para a placa HDMI transmissora

B.2 Transmissão de uma imagem entre dispositivos HDMI

I/O	Sinal	LOC na FPGA	Banco na FPGA	Nome na placa HDMI	PIN da placa HDMI
I	clk_px_in	AJ32	14	RX#0_LLC	CLK0_M2C_P
I	enable_in	AN38	15	RX#0_VSYNC	LA02_P
I	vsync_in	AU38	15	RX#0_HSYNC	LA00_P_CC
I	hsync_in	AU39	15	RX#0_DE	LA01_P_CC
I	pixel_in[0]	AM41	15	RX#0_P0	LA03_P
I	pixel_in[1]	AR38	15	RX#0_P1	LA04_P
I	pixel_in[2]	AN40	15	RX#0_P2	LA05_P
I	pixel_in[3]	AR37	15	RX#0_P3	LA06_P
I	pixel_in[4]	AM39	15	RX#0_P4	LA07_P
I	pixel_in[5]	AP40	15	RX#0_P5	LA08_P
I	pixel_in[6]	AP41	15	RX#0_P6	LA09_P
I	pixel_in[7]	AT39	15	RX#0_P7	LA10_P
I	pixel_in[8]	AR42	15	RX#0_P8	LA11_P
I	pixel_in[9]	AW37	15	RX#0_P9	LA12_P
I	pixel_in[10]	BA37	15	RX#0_P10	LA13_P
I	pixel_in[11]	AW38	15	RX#0_P11	LA14_P

I/O	**Sinal**	**LOC na FPGA**	**Banco na FPGA**	**Nome na placa HDMI**	**PIN da placa HDMI**
I	pixel_in[12]	BB38	15	RX#0_P12	LA15_P
I	pixel_in[13]	BA39	15	RX#0_P13	LA16_P
I	pixel_in[14]	AK34	14	RX#0_P14	LA17_P_CC
I	pixel_in[15]	AJ33	14	RX#0_P15	LA18_P_CC
I	pixel_in[16]	AM36	14	RX#0_P16	LA19_P
I	pixel_in[17]	AJ36	14	RX#0_P17	LA20_P
I	pixel_in[18]	AP36	14	RX#0_P18	LA21_P
I	pixel_in[19]	AK37	14	RX#0_P19	LA22_P
I	pixel_in[20]	AN35	14	RX#0_P20	LA23_P
I	pixel_in[21]	AL36	14	RX#0_P21	LA24_P
I	pixel_in[22]	AG33	14	RX#0_P22	LA25_P
I	pixel_in[23]	AK35	14	RX#0_P23	LA26_P
I	pixel_in[24]	AH31	14	RX#0_P24	LA27_P
I	pixel_in[25]	AH34	14	RX#0_P25	LA28_P
I	pixel_in[26]	AM34	14	RX#0_P26	LA29_P
I	pixel_in[27]	AM31	14	RX#0_P27	LA30_P
I	pixel_in[28]	AM33	14	RX#0_P28	LA31_P
I	pixel_in[29]	AL29	14	RX#0_P29	LA32_P
O	clk_px	E34	35	TX#O_DCLK	CLK0_M2C_P
O	enable	K35	34	TX#0_DE	LA02_P
O	vsync	L31	34	TX#0_VSYNC	LA00_P_CC
O	hsync	M32	34	TX#0_HSYNC	LA01_P_CC
O	pixel[0]	J32	34	TX#0_D0	LA03_P
O	pixel[1]	K33	34	TX#0_D1	LA04_P
O	pixel[2]	L34	34	TX#0_D2	LA05_P
O	pixel[3]	M33	34	TX#0_D3	LA06_P
O	pixel[4]	H34	34	TX#0_D4	LA07_P
O	pixel[5]	K29	34	TX#0_D5	LA08_P
O	pixel[6]	J30	34	TX#0_D6	LA09_P
O	pixel[7]	L29	34	TX#0_D7	LA10_P
O	pixel[8]	J31	34	TX#0_D8	LA11_P
O	pixel[9]	M28	34	TX#0_D9	LA12_P
O	pixel[10]	R28	34	TX#0_D10	LA13_P
O	pixel[11]	N28	34	TX#0_D11	LA14_P
O	pixel[12]	R30	34	TX#0_D12	LA15_P
O	pixel[13]	U31	34	TX#0_D13	LA16_P
O	pixel[14]	C35	35	TX#0_D14	LA17_P_CC

I/O	Sinal	LOC na FPGA	Banco na FPGA	Nome na placa HDMI	PIN da placa HDMI
O	pixel[15]	D35	35	TX#0_D15	LA18_P_CC
O	pixel[16]	B36	35	TX#0_D16	LA19_P
O	pixel[17]	B34	35	TX#0_D17	LA20_P
O	pixel[18]	B39	35	TX#0_D18	LA21_P
O	pixel[19]	A35	35	TX#0_D19	LA22_P
O	pixel[20]	C38	35	TX#0_D20	LA23_P
O	pixel[21]	B37	35	TX#0_D21	LA24_P
O	pixel[22]	E32	35	TX#0_D22	LA25_P
O	pixel[23]	B32	35	TX#0_D23	LA26_P
O	pixel[24]	E33	35	TX#0_D24	LA27_P
O	pixel[25]	C33	35	TX#0_D25	LA28_P
O	pixel[26]	G32	35	TX#0_D26	LA29_P
O	pixel[27]	F36	35	TX#0_D27	LA30_P
O	pixel[28]	F34	35	TX#0_D28	LA31_P
O	pixel[29]	H33	35	TX#0_D29	LA32_P

Tabela B.2: Localização das portas de entrada e saída da arquitura de transmissão de uma imagem RGB de 10 bits entre as placas HDMI transmissora e recetora

B.3 Transmissão de imagem e som entre dispositivos HDMI

I/O	Sinal	LOC na FPGA	Banco na FPGA	Nome na placa HDMI	PIN na placa HDMI
I	clock_p_in	AJ32	AJ32	RX#0_LLC	CLK0_M2C_P
I	vsync_in	AU38	AU38	RX#0_VSYNC	LA00_P_CC
I	hsync_in	AU39	AU39	RX#0_HSYNC	LA01_P_CC
I	enable_in	AN38	AN38	RX#0_DE	LA02_P
I	pixel_in[0]	AM41	AM41	RX#0_P0	LA03_P
I	pixel_in[1]	AR38	AR38	RX#0_P1	LA04_P
I	pixel_in[2]	AN40	AN40	RX#0_P2	LA05_P
I	pixel_in[3]	AR37	AR37	RX#0_P3	LA06_P
I	pixel_in[4]	AM39	AM39	RX#0_P4	LA07_P
I	pixel_in[5]	AP40	AP40	RX#0_P5	LA08_P
I	pixel_in[6]	AP41	AP41	RX#0_P6	LA09_P
I	pixel_in[7]	AT39	AT39	RX#0_P7	LA10_P
I	pixel_in[8]	AR42	AR42	RX#0_P8	LA11_P
I	pixel_in[9]	AW37	AW37	RX#0_P9	LA12_P

I/O	**Sinal**	**LOC na FPGA**	**Banco na FPGA**	**Nome na placa HDMI**	**PIN na placa HDMI**
I	pixel_in[10]	BA37	BA37	RX#0_P10	LA13_P
I	pixel_in[11]	AW38	AW38	RX#0_P11	LA14_P
I	pixel_in[12]	BB38	BB38	RX#0_P12	LA15_P
I	pixel_in[13]	BA39	BA39	RX#0_P13	LA16_P
I	pixel_in[14]	AK34	AK34	RX#0_P14	LA17_P_CC
I	pixel_in[15]	AJ33	AJ33	RX#0_P15	LA18_P_CC
I	pixel_in[16]	AM36	AM36	RX#0_P16	LA19_P
I	pixel_in[17]	AJ36	AJ36	RX#0_P17	LA20_P
I	pixel_in[18]	AP36	AP36	RX#0_P18	LA21_P
I	pixel_in[19]	AK37	AK37	RX#0_P19	LA22_P
I	pixel_in[20]	AN35	AN35	RX#0_P20	LA23_P
I	pixel_in[21]	AL36	AL36	RX#0_P21	LA24_P
I	pixel_in[22]	AG33	AG33	RX#0_P22	LA25_P
I	pixel_in[23]	AK35	AK35	RX#0_P23	LA26_P
I	pixel_in[24]	AH31	AH31	RX#0_P24	LA27_P
I	pixel_in[25]	AH34	AH34	RX#0_P25	LA28_P
I	pixel_in[26]	AM34	AM34	RX#0_P26	LA29_P
I	pixel_in[27]	AM31	AM31	RX#0_P27	LA30_P
I	pixel_in[28]	AM33	AM33	RX#0_P28	LA31_P
I	pixel_in[29]	AL29	AL29	RX#0_P29	LA32_P
I	pixel_in[30]	AJ31	AJ31	RX#0_P30	LA27_N
I	pixel_in[31]	AJ35	AJ35	RX#0_P31	LA28_N
I	pixel_in[32]	AN34	AN34	RX#0_P32	LA29_N
I	pixel_in[33]	AM32	AM32	RX#0_P33	LA30_N
I	pixel_in[34]	AN33	AN33	RX#0_P34	LA31_N
I	pixel_in[35]	AL30	AL30	RX#0_P35	LA32_N
I	sclk_in	AJ37	AJ37	RX#0_SCLK	LA20_N
I	AP1_in	AL37	AL37	RX#0_AP1	LA22_N
I	AP2_in	AP35	AP35	RX#0_AP2	LA23_N
I	AP3_in	AM37	AM37	RX#0_AP3	LA24_N
I	AP4_in	AH33	AH33	RX#0_AP4	LA25_N
I	lrclk_in	AL35	AL35	RX#0_AP5	LA19_N
I	info_in[0]	AV38	AV38	RX#0_Input video status[0]	LA00_N_CC
I	info_in[1]	AV39	AV39	RX#0_Input video status[1]	LA01_N_CC
O	clock_px	E34	35	TX#0_DCLK	CLK0_M2C_P

I/O	Sinal	LOC na FPGA	Banco na FPGA	Nome na placa HDMI	PIN na placa HDMI
O	vsync	L31	34	TX#0_VSYNC	LA00_P_CC
O	hsync	M32	34	TX#0_HSYNC	LA01_P_CC
O	enable	K35	34	TX#0_DE	LA02_P
O	pixel[0]	J32	34	TX#0_D0	LA03_P
O	pixel[1]	K33	34	TX#0_D1	LA04_P
O	pixel[2]	L34	34	TX#0_D2	LA05_P
O	pixel[3]	M33	34	TX#0_D3	LA06_P
O	pixel[4]	H34	34	TX#0_D4	LA07_P
O	pixel[5]	K29	34	TX#0_D5	LA08_P
O	pixel[6]	J30	34	TX#0_D6	LA09_P
O	pixel[7]	L29	34	TX#0_D7	LA10_P
O	pixel[8]	J31	34	TX#0_D8	LA11_P
O	pixel[9]	M28	34	TX#0_D9	LA12_P
O	pixel[10]	R28	34	TX#0_D10	LA13_P
O	pixel[11]	N28	34	TX#0_D11	LA14_P
O	pixel[12]	R30	34	TX#0_D12	LA15_P
O	pixel[13]	U31	34	TX#0_D13	LA16_P
O	pixel[14]	C35	35	TX#0_D14	LA17_P_CC
O	pixel[15]	D35	35	TX#0_D15	LA18_P_CC
O	pixel[16]	B36	35	TX#0_D16	LA19_P
O	pixel[17]	B34	35	TX#0_D17	LA20_P
O	pixel[18]	B39	35	TX#0_D18	LA21_P
O	pixel[19]	A35	35	TX#0_D19	LA22_P
O	pixel[20]	C38	35	TX#0_D20	LA23_P
O	pixel[21]	B37	35	TX#0_D21	LA24_P
O	pixel[22]	E32	35	TX#0_D22	LA25_P
O	pixel[23]	B32	35	TX#0_D23	LA26_P
O	pixel[24]	E33	35	TX#0_D24	LA27_P
O	pixel[25]	C33	35	TX#0_D25	LA28_P
O	pixel[26]	G32	35	TX#0_D26	LA29_P
O	pixel[27]	F36	35	TX#0_D27	LA30_P
O	pixel[28]	F34	35	TX#0_D28	LA31_P
O	pixel[29]	H33	35	TX#0_D29	LA32_P
O	pixel[30]	D33	35	TX#0_D30	LA27_N
O	pixel[31]	C34	35	TX#0_D31	LA28_N
O	pixel[32]	F32	35	TX#0_D32	LA29_N
O	pixel[33]	F37	35	TX#0_D33	LA30_N

I/O	**Sinal**	**LOC na FPGA**	**Banco na FPGA**	**Nome na placa HDMI**	**PIN na placa HDMI**
O	pixel[34]	F35	35	TX#0_D34	LA31_N
O	pixel[35]	G33	35	TX#0_D35	LA32_N
O	sclk_out	A34	35	TX#0_SCLK	LA20_N
O	AP1_out	A36	35	TX#0_AP1	LA22_N
O	AP2_out	C39	35	TX#0_AP2	LA23_N
O	AP3_out	B38	35	TX#0_AP3	LA24_N
O	AP4_out	D32	35	TX#0_AP4	LA25_N
O	lrclk_out	B33	35	TX#0_AP5	LA26_N
O	info_out[0]	K32	34	TX#0_output video status[0]	LA00_N_CC
O	info_out[1]	L32	34	TX#0_output video status[1]	LA01_N_CC

Tabela B.3: Localização das portas de entrada e saída da arquitura de transmissão de imagem e som entre as placas HDMI transmissora e recetora

Anexo C

Ficheiros de restrições

C.0.1 Transmissão de uma imagem gerada na FPGA

C.0.1.1 Restrições Físicas

```
set_property PACKAGE_PIN E18 [get_ports clk_n]
set_property IOSTANDARD DIFF_HSTL_II_18 [get_ports clk_p]
set_property PACKAGE_PIN E34 [get_ports clk_out]
set_property PACKAGE_PIN K35 [get_ports enable]
set_property IOSTANDARD LVCMOS18 [get_ports clk_out]
set_property IOSTANDARD LVCMOS18 [get_ports enable]
set_property IOSTANDARD LVCMOS18 [get_ports vsync]
set_property IOSTANDARD LVCMOS18 [get_ports hsync]
set_property IOSTANDARD LVCMOS18 [get_ports pixel[29]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel[28]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel[27]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel[26]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel[25]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel[24]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel[23]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel[22]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel[21]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel[20]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel[19]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel[18]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel[17]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel[16]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel[15]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel[14]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel[13]]
```

```
set_property IOSTANDARD LVCMOS18 [get_ports pixel[12]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel[11]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel[10]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel[9]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel[8]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel[7]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel[6]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel[5]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel[4]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel[3]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel[2]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel[1]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel[0]]
set_property IOSTANDARD LVCMOS18 [get_ports reset]
set_property IOSTANDARD LVCMOS18 [get_ports start]
set_property PACKAGE_PIN N41 [get_ports reset]
set_property PACKAGE_PIN M32 [get_ports hsync]
set_property PACKAGE_PIN L31 [get_ports vsync]
set_property PACKAGE_PIN J32 [get_ports pixel[0]]
set_property PACKAGE_PIN K33 [get_ports pixel[1]]
set_property PACKAGE_PIN L34 [get_ports pixel[2]]
set_property PACKAGE_PIN M33 [get_ports pixel[3]]
set_property PACKAGE_PIN H34 [get_ports pixel[4]]
set_property PACKAGE_PIN K29 [get_ports pixel[5]]
set_property PACKAGE_PIN J30 [get_ports pixel[6]]
set_property PACKAGE_PIN L29 [get_ports pixel[7]]
set_property PACKAGE_PIN J31 [get_ports pixel[8]]
set_property PACKAGE_PIN M28 [get_ports pixel[9]]
set_property PACKAGE_PIN R28 [get_ports pixel[10]]
set_property PACKAGE_PIN N28 [get_ports pixel[11]]
set_property PACKAGE_PIN R30 [get_ports pixel[12]]
set_property PACKAGE_PIN U31 [get_ports pixel[13]]
set_property PACKAGE_PIN C35 [get_ports pixel[14]]
set_property PACKAGE_PIN D35 [get_ports pixel[15]]
set_property PACKAGE_PIN B36 [get_ports pixel[16]]
set_property PACKAGE_PIN B34 [get_ports pixel[17]]
set_property PACKAGE_PIN B39 [get_ports pixel[18]]
set_property PACKAGE_PIN A35 [get_ports pixel[19]]
set_property PACKAGE_PIN C38 [get_ports pixel[20]]
set_property PACKAGE_PIN B37 [get_ports pixel[21]]
```

```
set_property PACKAGE_PIN E32 [get_ports pixel[22]]
set_property PACKAGE_PIN B32 [get_ports pixel[23]]
set_property PACKAGE_PIN E33 [get_ports pixel[24]]
set_property PACKAGE_PIN C33 [get_ports pixel[25]]
set_property PACKAGE_PIN G32 [get_ports pixel[26]]
set_property PACKAGE_PIN F36 [get_ports pixel[27]]
set_property PACKAGE_PIN F34 [get_ports pixel[28]]
set_property PACKAGE_PIN H33 [get_ports pixel[29]]
set_property PACKAGE_PIN E42 [get_ports start]
```

C.0.1.2 Restrições Temporais

```
create_clock -period 5.000 [get_ports clk_n]
create_clock -period 5.000 [get_ports clk_p]
```

C.0.2 Transmissão de imagem entre dispositivos HDMI

C.0.2.1 Restrições Físicas

```
set_property PACKAGE_PIN E34 [get_ports p_clock_out]
set_property PACKAGE_PIN K35 [get_ports enable_out]
set_property IOSTANDARD LVCMOS18 [get_ports p_clock_out]
set_property IOSTANDARD LVCMOS18 [get_ports enable_out]
set_property PACKAGE_PIN L31 [get_ports vsync_out]
set_property IOSTANDARD LVCMOS18 [get_ports vsync_out]
set_property PACKAGE_PIN M32 [get_ports hsync_out]
set_property IOSTANDARD LVCMOS18 [get_ports hsync_out]
set_property PACKAGE_PIN J32 [get_ports pixel_out[0]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[0]]
set_property PACKAGE_PIN K33 [get_ports pixel_out[1]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[1]]
set_property PACKAGE_PIN L34 [get_ports pixel_out[2]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[2]]
set_property PACKAGE_PIN M33 [get_ports pixel_out[3]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[3]]
set_property PACKAGE_PIN H34 [get_ports pixel_out[4]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[4]]
set_property PACKAGE_PIN K29 [get_ports pixel_out[5]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[5]]
set_property PACKAGE_PIN J30 [get_ports pixel_out[6]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[6]]
set_property PACKAGE_PIN L29 [get_ports pixel_out[7]]
```

```
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[7]]
set_property PACKAGE_PIN J31 [get_ports pixel_out[8]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[8]]
set_property PACKAGE_PIN M28 [get_ports pixel_out[9]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[9]]
set_property PACKAGE_PIN R28 [get_ports pixel_out[10]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[10]]
set_property PACKAGE_PIN N28 [get_ports pixel_out[11]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[11]]
set_property PACKAGE_PIN R30 [get_ports pixel_out[12]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[12]]
set_property PACKAGE_PIN U31 [get_ports pixel_out[13]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[13]]
set_property PACKAGE_PIN C35 [get_ports pixel_out[14]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[14]]
set_property PACKAGE_PIN D35 [get_ports pixel_out[15]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[15]]
set_property PACKAGE_PIN B36 [get_ports pixel_out[16]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[16]]
set_property PACKAGE_PIN B34 [get_ports pixel_out[17]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[17]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[18]]
set_property PACKAGE_PIN B39 [get_ports pixel_out[18]]
set_property PACKAGE_PIN A35 [get_ports pixel_out[19]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[19]]
set_property PACKAGE_PIN C38 [get_ports pixel_out[20]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[20]]
set_property PACKAGE_PIN B37 [get_ports pixel_out[21]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[21]]
set_property PACKAGE_PIN E32 [get_ports pixel_out[22]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[22]]
set_property PACKAGE_PIN B32 [get_ports pixel_out[23]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[23]]
set_property PACKAGE_PIN E33 [get_ports pixel_out[24]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[24]]
set_property PACKAGE_PIN C33 [get_ports pixel_out[25]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[25]]
set_property PACKAGE_PIN G32 [get_ports pixel_out[26]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[26]]
set_property PACKAGE_PIN F36 [get_ports pixel_out[27]]
```

```
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[27]]
set_property PACKAGE_PIN F34 [get_ports pixel_out[28]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[28]]
set_property PACKAGE_PIN H33 [get_ports pixel_out[29]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[29]]
set_property PACKAGE_PIN AJ32 [get_ports p_clock_in]
set_property IOSTANDARD LVCMOS18 [get_ports p_clock_in]
set_property IOSTANDARD LVCMOS18 [get_ports hsync_in]
set_property IOSTANDARD LVCMOS18 [get_ports enable_in]
set_property IOSTANDARD LVCMOS18 [get_ports reset]
set_property IOSTANDARD LVCMOS18 [get_ports start]
set_property IOSTANDARD LVCMOS18 [get_ports vsync_in]
set_property PACKAGE_PIN AN38 [get_ports enable_in]
set_property PACKAGE_PIN AU38 [get_ports vsync_in]
set_property PACKAGE_PIN AU39 [get_ports hsync_in]
set_property PACKAGE_PIN N41 [get_ports reset]
set_property PACKAGE_PIN E42 [get_ports start]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[29]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[28]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[27]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[26]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[25]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[24]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[23]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[22]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[21]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[20]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[19]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[18]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[17]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[16]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[15]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[14]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[13]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[12]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[11]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[10]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[9]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[8]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[7]]
```

```
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[6]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[5]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[4]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[3]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[2]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[1]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[0]]
set_property PACKAGE_PIN AM41 [get_ports pixel_in[0]]
set_property PACKAGE_PIN AR38 [get_ports pixel_in[1]]
set_property PACKAGE_PIN AN40 [get_ports pixel_in[2]]
set_property PACKAGE_PIN AR37 [get_ports pixel_in[3]]
set_property PACKAGE_PIN AM39 [get_ports pixel_in[4]]
set_property PACKAGE_PIN AP40 [get_ports pixel_in[5]]
set_property PACKAGE_PIN AP41 [get_ports pixel_in[6]]
set_property PACKAGE_PIN AT39 [get_ports pixel_in[7]]
set_property PACKAGE_PIN AR42 [get_ports pixel_in[8]]
set_property PACKAGE_PIN AW37 [get_ports pixel_in[9]]
set_property PACKAGE_PIN BA37 [get_ports pixel_in[10]]
set_property PACKAGE_PIN AW38 [get_ports pixel_in[11]]
set_property PACKAGE_PIN BB38 [get_ports pixel_in[12]]
set_property PACKAGE_PIN BA39 [get_ports pixel_in[13]]
set_property PACKAGE_PIN AK34 [get_ports pixel_in[14]]
set_property PACKAGE_PIN AJ33 [get_ports pixel_in[15]]
set_property PACKAGE_PIN AM36 [get_ports pixel_in[16]]
set_property PACKAGE_PIN AJ36 [get_ports pixel_in[17]]
set_property PACKAGE_PIN AP36 [get_ports pixel_in[18]]
set_property PACKAGE_PIN AK37 [get_ports pixel_in[19]]
set_property PACKAGE_PIN AN35 [get_ports pixel_in[20]]
set_property PACKAGE_PIN AL36 [get_ports pixel_in[21]]
set_property PACKAGE_PIN AG33 [get_ports pixel_in[22]]
set_property PACKAGE_PIN AK35 [get_ports pixel_in[23]]
set_property PACKAGE_PIN AH31 [get_ports pixel_in[24]]
set_property PACKAGE_PIN AH34 [get_ports pixel_in[25]]
set_property PACKAGE_PIN AM34 [get_ports pixel_in[26]]
set_property PACKAGE_PIN AM31 [get_ports pixel_in[27]]
set_property PACKAGE_PIN AM33 [get_ports pixel_in[28]]
set_property PACKAGE_PIN AL29 [get_ports pixel_in[29]]
```

C.0.2.2 Restrições Temporais

```
create_clock -period 5.000 [get_ports clk_n]
```

```
create_clock -period 5.000 [get_ports clk_p]
```

C.1 Transmissão de imagem e som entre dispositivos HDMI

C.1.0.1 Restrições Físicas

```
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[35]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[34]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[33]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[32]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[31]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[30]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[29]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[28]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[27]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[26]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[25]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[24]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[23]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[22]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[21]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[20]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[19]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[18]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[17]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[16]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[15]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[14]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[13]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[12]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[11]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[10]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[9]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[8]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[7]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[6]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[5]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[4]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[3]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[2]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[1]]
```

```
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[0]]
set_property IOSTANDARD LVCMOS18 [get_ports info_out[1]]
set_property IOSTANDARD LVCMOS18 [get_ports info_out[0]]
set_property IOSTANDARD LVCMOS18 [get_ports info_in[1]]
set_property IOSTANDARD LVCMOS18 [get_ports info_in[0]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[35]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[34]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[33]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[32]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[31]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[30]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[29]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[28]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[27]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[26]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[25]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[24]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[23]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[22]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[21]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[20]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[19]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[18]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[17]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[16]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[15]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[14]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[13]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[12]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[11]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[10]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[9]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[8]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[7]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[6]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[5]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[4]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[3]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[2]]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[1]]
```

```
set_property IOSTANDARD LVCMOS18 [get_ports pixel_out[0]]
set_property PACKAGE_PIN AM41 [get_ports pixel_in[0]]
set_property PACKAGE_PIN AR38 [get_ports pixel_in[1]]
set_property PACKAGE_PIN AN40 [get_ports pixel_in[2]]
set_property PACKAGE_PIN AR37 [get_ports pixel_in[3]]
set_property PACKAGE_PIN AM39 [get_ports pixel_in[4]]
set_property PACKAGE_PIN AP40 [get_ports pixel_in[5]]
set_property PACKAGE_PIN AP41 [get_ports pixel_in[6]]
set_property PACKAGE_PIN AT39 [get_ports pixel_in[7]]
set_property PACKAGE_PIN AR42 [get_ports pixel_in[8]]
set_property PACKAGE_PIN AW37 [get_ports pixel_in[9]]
set_property PACKAGE_PIN BA37 [get_ports pixel_in[10]]
set_property PACKAGE_PIN AW38 [get_ports pixel_in[11]]
set_property PACKAGE_PIN BB38 [get_ports pixel_in[12]]
set_property PACKAGE_PIN AK34 [get_ports pixel_in[14]]
set_property PACKAGE_PIN AJ33 [get_ports pixel_in[15]]
set_property PACKAGE_PIN AM36 [get_ports pixel_in[16]]
set_property PACKAGE_PIN AJ36 [get_ports pixel_in[17]]
set_property PACKAGE_PIN AP36 [get_ports pixel_in[18]]
set_property PACKAGE_PIN AK37 [get_ports pixel_in[19]]
set_property PACKAGE_PIN AN35 [get_ports pixel_in[20]]
set_property PACKAGE_PIN AL36 [get_ports pixel_in[21]]
set_property PACKAGE_PIN AG33 [get_ports pixel_in[22]]
set_property PACKAGE_PIN AK35 [get_ports pixel_in[23]]
set_property PACKAGE_PIN AH31 [get_ports pixel_in[24]]
set_property PACKAGE_PIN AH34 [get_ports pixel_in[25]]
set_property PACKAGE_PIN AM34 [get_ports pixel_in[26]]
set_property PACKAGE_PIN AM31 [get_ports pixel_in[27]]
set_property PACKAGE_PIN AM33 [get_ports pixel_in[28]]
set_property PACKAGE_PIN AL29 [get_ports pixel_in[29]]
set_property PACKAGE_PIN AJ31 [get_ports pixel_in[30]]
set_property PACKAGE_PIN AJ35 [get_ports pixel_in[31]]
set_property PACKAGE_PIN AN34 [get_ports pixel_in[32]]
set_property PACKAGE_PIN AM32 [get_ports pixel_in[33]]
set_property PACKAGE_PIN AN33 [get_ports pixel_in[34]]
set_property PACKAGE_PIN AL30 [get_ports pixel_in[35]]
set_property PACKAGE_PIN BA39 [get_ports pixel_in[13]]
set_property PACKAGE_PIN AV38 [get_ports info_in[0]]
set_property PACKAGE_PIN AV39 [get_ports info_in[1]]
set_property PACKAGE_PIN AL37 [get_ports AP1_in]
```

```
set_property IOSTANDARD LVCMOS18 [get_ports AP1_in]
set_property PACKAGE_PIN AP35 [get_ports AP2_in]
set_property IOSTANDARD LVCMOS18 [get_ports AP2_in]
set_property PACKAGE_PIN AM37 [get_ports AP3_in]
set_property IOSTANDARD LVCMOS18 [get_ports AP3_in]
set_property PACKAGE_PIN AH33 [get_ports AP4_in]
set_property IOSTANDARD LVCMOS18 [get_ports AP4_in]
set_property PACKAGE_PIN AL35 [get_ports lrclk_in]
set_property IOSTANDARD LVCMOS18 [get_ports lrclk_in]
set_property PACKAGE_PIN AJ37 [get_ports sclk_in]
set_property IOSTANDARD LVCMOS18 [get_ports sclk_in]
set_property PACKAGE_PIN AJ32 [get_ports clock_p_in]
set_property IOSTANDARD LVCMOS18 [get_ports clock_p_in]
set_property PACKAGE_PIN AU38 [get_ports vsync_in]
set_property IOSTANDARD LVCMOS18 [get_ports vsync_in]
set_property PACKAGE_PIN AU39 [get_ports hsync_in]
set_property PACKAGE_PIN AN38 [get_ports enable_in]
set_property PACKAGE_PIN E19 [get_ports clock_p]
set_property IOSTANDARD DIFF_HSTL_II_18 [get_ports clock_p]
set_property PACKAGE_PIN N41 [get_ports reset]
set_property IOSTANDARD LVCMOS18 [get_ports reset]
set_property PACKAGE_PIN E42 [get_ports POWER]
set_property IOSTANDARD LVCMOS18 [get_ports POWER]
set_property PACKAGE_PIN G41 [get_ports MUTE]
set_property IOSTANDARD LVCMOS18 [get_ports MUTE]
set_property PACKAGE_PIN J32 [get_ports pixel_out[0]]
set_property PACKAGE_PIN K33 [get_ports pixel_out[1]]
set_property PACKAGE_PIN L34 [get_ports pixel_out[2]]
set_property PACKAGE_PIN M33 [get_ports pixel_out[3]]
set_property PACKAGE_PIN H34 [get_ports pixel_out[4]]
set_property PACKAGE_PIN K29 [get_ports pixel_out[5]]
set_property PACKAGE_PIN J30 [get_ports pixel_out[6]]
set_property PACKAGE_PIN L29 [get_ports pixel_out[7]]
set_property PACKAGE_PIN J31 [get_ports pixel_out[8]]
set_property PACKAGE_PIN M28 [get_ports pixel_out[9]]
set_property PACKAGE_PIN R28 [get_ports pixel_out[10]]
set_property PACKAGE_PIN N28 [get_ports pixel_out[11]]
set_property PACKAGE_PIN R30 [get_ports pixel_out[12]]
set_property PACKAGE_PIN U31 [get_ports pixel_out[13]]
set_property PACKAGE_PIN C35 [get_ports pixel_out[14]]
```

```
set_property PACKAGE_PIN D35 [get_ports pixel_out[15]]
set_property PACKAGE_PIN B36 [get_ports pixel_out[16]]
set_property PACKAGE_PIN B34 [get_ports pixel_out[17]]
set_property PACKAGE_PIN B39 [get_ports pixel_out[18]]
set_property PACKAGE_PIN A35 [get_ports pixel_out[19]]
set_property PACKAGE_PIN C38 [get_ports pixel_out[20]]
set_property PACKAGE_PIN B37 [get_ports pixel_out[21]]
set_property PACKAGE_PIN E32 [get_ports pixel_out[22]]
set_property PACKAGE_PIN B32 [get_ports pixel_out[23]]
set_property PACKAGE_PIN E33 [get_ports pixel_out[24]]
set_property PACKAGE_PIN C33 [get_ports pixel_out[25]]
set_property PACKAGE_PIN G32 [get_ports pixel_out[26]]
set_property PACKAGE_PIN F36 [get_ports pixel_out[27]]
set_property PACKAGE_PIN F34 [get_ports pixel_out[28]]
set_property PACKAGE_PIN H33 [get_ports pixel_out[29]]
set_property PACKAGE_PIN D33 [get_ports pixel_out[30]]
set_property PACKAGE_PIN C34 [get_ports pixel_out[31]]
set_property PACKAGE_PIN F32 [get_ports pixel_out[32]]
set_property PACKAGE_PIN F37 [get_ports pixel_out[33]]
set_property PACKAGE_PIN F35 [get_ports pixel_out[34]]
set_property PACKAGE_PIN G33 [get_ports pixel_out[35]]
set_property PACKAGE_PIN L32 [get_ports info_out[1]]
set_property PACKAGE_PIN K32 [get_ports info_out[0]]
set_property PACKAGE_PIN A36 [get_ports AP1_out]
set_property IOSTANDARD LVCMOS18 [get_ports AP1_out]
set_property PACKAGE_PIN C39 [get_ports AP2_out]
set_property IOSTANDARD LVCMOS18 [get_ports AP2_out]
set_property PACKAGE_PIN B38 [get_ports AP3_out]
set_property IOSTANDARD LVCMOS18 [get_ports AP3_out]
set_property PACKAGE_PIN D32 [get_ports AP4_out]
set_property IOSTANDARD LVCMOS18 [get_ports AP4_out]
set_property PACKAGE_PIN B33 [get_ports lrclk_out]
set_property IOSTANDARD LVCMOS18 [get_ports lrclk_out]
set_property PACKAGE_PIN A34 [get_ports sclk_out]
set_property IOSTANDARD LVCMOS18 [get_ports sclk_out]
set_property PACKAGE_PIN E34 [get_ports clock_p_out]
set_property IOSTANDARD LVCMOS18 [get_ports clock_p_out]
set_property PACKAGE_PIN K35 [get_ports enable_out]
set_property IOSTANDARD LVCMOS18 [get_ports enable_in]
set_property PACKAGE_PIN M32 [get_ports hsync_out]
```

```
set_property IOSTANDARD LVCMOS18 [get_ports hsync_out]
set_property PACKAGE_PIN L31 [get_ports vsync_out]
set_property IOSTANDARD LVCMOS18 [get_ports vsync_out]
set_property IOSTANDARD LVCMOS18 [get_ports enable_out]
set_property IOSTANDARD LVCMOS18 [get_ports hsync_in]
```

C.1.0.2 Restrições Temporais

```
create_clock -period 5.000 [get_ports clk_n]
create_clock -period 5.000 [get_ports clk_p]
```

Bibliografia

[1] Xilinx and Inc, "VC7203 Virtex-7 FPGA GTX Transceiver Characterization Board User Guide (UG957)," 2014.

[2] Inrevium, *Manual do Utilizador de TB-FMCH-HDMI2 Hardware*. 2012.

[3] D. Chen, "SerDes Transceivers for High-speed Serial Communications,"

[4] Xilinx and Inc, "Xilinx WP431 Leveraging 7 Series FPGA Transceivers for High-Speed Serial I/O Connectivity, White Paper," 2013.

[5] Xilinx and Inc, "7 Series FPGAs GTX/GTH Transceivers User Guide (UG476),"

[6] Analog Devices, "ADV7612 Reference Manual,"

[7] Inrevium, "TB-FMCH-HDMI2 Hardware User Manual 1 IN / OUT + Audio support," pp. 1–59.

[8] Inrevium, "TB-FMCH-HDMI2 Hardware User Manual 2 Ch IN / OUT support," pp. 1–56.

[9] Wikipedia Contributors, "HDMI," 2016.

[10] S. Koenig, D. Lopez-Diaz, J. Antes, F. Boes, R. Henneberger, A. Leuther, A. Tessmann, R. Schmogrow, D. Hillerkuss, R. Palmer, T. Zwick, C. Koos, W. Freude, O. Ambacher, J. Leuthold, and I. Kallfass, "Wireless sub-THz communication system with high data rate enabled by RF photonics and active MMIC technology," *2014 IEEE Photonics Conference, IPC 2014*, vol. 7, no. December 2013, pp. 414–415, 2014.

[11] J. Federici and L. Moeller, "Review of terahertz and subterahertz wireless communications," *Journal of Applied Physics*, vol. 107, no. 11, 2010.

[12] W. contributors, "audio and video interfaces and connectors," 2016.

[13] Xilinx, "Platform Flash In-System," *Memory*, vol. 123, pp. 1–46, 2006.

[14] Xilinx, "Xilinx Platform Cable USB II," vol. 593, pp. 1–36, 2015.

[15] P. Semiconductors and B. I. Timing, "I 2 S bus specification I 2 S bus specification," no. February 1986, pp. 1–7, 1996.

[16] C. E. Cummings, "Clock Domain Crossing (CDC) Design & Verification Techniques Using SystemVerilog," *Snug-2008*, no. Cdc, pp. 1–56, 2008.