



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

13 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 espectro de frequências permitindo assim comunicações de alta velocidade. Este projeto passa por propor uma metodologia que permite a manufaturação de transdutores 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 transdutores que estão a ser desenvolvidos. E por isso nesta dissertação é proposto um projeto que visa testar os transdutores 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 seguida enviados através dos transdutores 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 transdutores do projeto iBrow através das entradas de alta velocidade existentes na FPGA a ser utilizada e voltá-los a enviar para o dispositivo final HDMI.

Inicialmente foi desenvolvida e implementada uma arquitetura em FPGA, recorrendo a *hardware* capaz de decodificar 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

Marisa Oliveira

'The journey is the reward'

Steve Jobs

Conteúdo

1	Introdução	1
1.1	Enquadramento Geral	1
1.2	Motivação	2
1.3	Descrição do Problema e Objetivos	4
1.4	Estrutura da Dissertação	6
2	Revisão Bibliográfica	9
2.1	Interfaces de transmissão de video/audio	9
2.2	HDMI (<i>High Definition Multimedia Interface</i>)	10
2.2.1	DDC - <i>Display Data Channel</i>	10
2.2.2	TMDS - <i>Transition-Minimized Differential Signaling</i>	10
2.2.3	CEC - <i>Consumer Electronics Control</i>	11
2.2.4	ARC - <i>Audio Return Channel</i>	11
2.2.5	HEC - <i>HDMI Ethernet Channel</i>	11
2.3	Transmissão de dados HDMI	11
2.3.1	Conexão à FPGA XILINX VC7203 Virtex-7	12
2.3.2	Recetor	13
2.3.3	Transmissor	15
2.4	Conexão de alta velocidade em série	17
2.4.1	Comunicações em paralelo VS comunicações em série	17
2.4.2	Considerações sobre arquiteturas de transmissão de dados em série	19
2.5	Saídas em série de alta velocidade da FPGA VC7203	33
2.5.1	Localização dos transdutores na FPGA	33
2.5.2	Arquitetura dos transdutores	34
2.6	Sincronização entre diferentes domínios de relógio	35
3	HDMI	39
3.1	<i>Hardware</i> utilizado	39
3.1.1	Configurações da FPGA	41
3.1.2	Configuração dos interruptores	44
3.2	Arquiteturas Desenvolvidas	47
3.2.1	Transmissão de uma imagem gerada na FPGA	48
3.2.2	Transmissão de imagem entre dispositivos HDMI	54
3.2.3	Transmissão de imagem e som entre dispositivos HDMI	56
4	Transmissão dos dados em série	61
5	Conclusões e Trabalho Futuro	63

A	Descrição dos pinos das placas HDMI	65
A.1	Configuração por <i>default</i>	65
A.2	Suporte de um canal de imagem e áudio	66
A.3	Suporte de dois canais de imagem melhorado	67
B	Localizações das portas provenientes de TB-HDMI	71
B.1	Transmissão de uma imagem gerada na FPGA	71
B.2	Transmissão de uma imagem entre dispositivos HDMI	72
B.3	Transmissão de imagem e som entre dispositivos HDMI	74
C	Ficheiros de restrições	79
C.1	Transmissão de uma imagem gerada na FPGA	79
C.2	Restrições Físicas	79
C.2.1	Transmissão de imagem entre dispositivos HDMI	81
C.3	Transmissão de imagem e som entre dispositivos HDMI	85
C.3.1	Restrições Temporais	90

Lista de Figuras

1.1	Diagrama geral do problema proposto	4
1.2	Diagrama geral da primeira parte problema	5
1.3	Diagrama geral da segunda parte do problema	6
2.1	Vista Geral da FPGA VC7203 Virtex-7 retirada de [1]	12
2.2	Diagrama de blocos de TB-FMCH-HDMI2 RX retirado de [2]	13
2.3	Amostragem dos dados provenientes da FPGA no recetor, retirada de [2]	15
2.4	Diagrama de blocos de TB-FMCH-HDMI2 TX retirado de [2]	16
2.5	Amostragem dos dados provenientes do FMC no recetor, retirada de [2]	16
2.6	Arquitetura simples de um serializador e deserializador, retirada de [3]	19
2.7	Ilustração do alinhamento em série quando encontrada a <i>comma</i> , retirada de [4]	24
2.8	Ilustração do alinhamento em paralelo quando encontrada a <i>comma</i> , retirada de [4]	24
2.9	Arquiteturas de PISO/SIPO, retirada de [5]	25
2.10	Arquitetura de um <i>shift-register</i> serializador de 4 bits retirado de [6]	26
2.11	Arquitetura de um <i>shift-register</i> serializador de 4 bits retirado de [6]	27
2.12	Exemplo de um serializador de 10 bits, retirado de [7]	27
2.13	Efeito da Interferência Inter-Simbólica numa transmissão, adaptada de [3]	28
2.14	Localização física na FPGA dos GTX, retirada de [1]	33
2.15	Conectores GTX localizados na FPGA, retirada de [1]	34
2.16	Arquitetura geral dos transctores, retirada de [8]	35
2.17	Representação dos diferentes dominios de sinais de relógio no projeto	36
2.18	Exemplo de metaestabilidade, adaptado de [9]	37
2.19	Exemplo de metaestabilidade, retirado de [9]	37
3.1	TB-FMCH-HDMI2 RX, retirada de [2]	40
3.2	TB-FMCH-HDMI2 TX, retirada de [2]	40
3.3	Ilustração dos sinais de som transmitidos no formato I^2S , retirada de [10]	43
3.4	Exemplo de imagem gerada pelo modulo desenvolvido	49
3.5	Máquina de estados para gerar uma barra de cores	50
3.6	Diagrama de blocos de arquitetura implementada utilizando um bloco gerador de barra de cores	52
3.7	Diagrama de blocos da arquitetura desenvolvida para transmitir imagem entre dispositivos HDMI	54
3.8	Diagrama de blocos da arquitetura desenvolvida para transmitir imagem e som entre dispositivos HDMI	57

Lista de Tabelas

2.1	Nomes dos pins da interface FMC de TB-FMCH-HDMI2 RX, adaptada de [2]	14
2.2	Nomes dos pins da interface FMC de TB-FMCH-HDMI2 TX, adaptada de [2]	17
2.3	Exemplo de codificação 8B/10B, retirada de [3]	22
2.4	Exemplo de palavras de 8 bits codificadas em 8B/10B, retirada de [3]	22
2.5	Caracteres de controlo específicos da codificação 8B/10B, retirada de [3]	23
3.1	Descrição e localização dos pinos de TB-FMCH-HDMI2 configurada por <i>default</i>	41
3.2	Descrição e localização dos pinos de TB-FMCH-HDMI2 configurada para um canal de imagem e áudio	43
3.3	Configuração dos interruptores da placa HDMI RX configurada de fábrica, adaptada de [2]	45
3.4	Configuração dos interruptores da placa HDMI RX configurada de fábrica, adaptada de [2]	45
3.5	Configuração dos interruptores da placa HDMI RX configurada para um canal e suporte de áudio, adaptada de [11]	46
3.6	Configuração dos interruptores da placa HDMI TX configurada para um canal e suporte de áudio, adaptada de [11]	46
3.7	Configuração dos interruptores da placa HDMI RX configurada para dois canais melhorados, adaptada de [12]	47
3.8	Configuração dos interruptores da placa HDMI RX configurada para dois canais melhorados, adaptada de [12]	47
3.9	Localização das portas de entrada e saída da arquitetura	53
3.10	Localização das entradas e saídas das portas da arquitetura	55
3.11	Localização das entradas e saídas das portas da arquitetura	59
A.1	Localização dos pinos de dados utilizados em TB-FMCH-HDMI2 configurado por <i>default</i>	66
A.2	Localização dos pinos de dados utilizados em TB-FMCH-HDMI2 com a configuração de um canal e suporte de áudio	67
A.3	Localização dos pinos de dados utilizados em TB-FMCH-HDMI2 com a configuração de suporte de dois canais melhorado	70
B.1	Localização das portas de entrada e saída da arquitetura de transmissão de uma barra de cores para a placa HDMI transmissora	72
B.2	Localização das portas de entrada e saída da arquitetura de transmissão de uma imagem RGB de 10 bits entre as placas HDMI transmissora e recetora	74
B.3	Localização das portas de entrada e saída da arquitetura de transmissão de imagem e som entre as placas HDMI transmissora e recetora	77

Abreviaturas e Símbolos

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

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 espectro 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 manufatura de transdutores 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-bandwidth 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 [13] 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 espectro 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 espectro de frequências, de maneira a que se possa utilizá-lo de uma forma mais eficaz. Ao longo dos anos tem-se vindo a verificar melhorias no que toca à eficiência espectral através do desenvolvimento e aplicação de algumas técnicas, tal como referido em [14], 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 espectro 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 espectro 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 sido desenvolvidas e aplicadas técnicas que melhoram a eficiência espectral desta região, tal como referido anteriormente, a escassez da largura de banda limita o débito da ligação. Em [14] são referidas implementações realizadas no passado que conseguiram alcançar débitos até 100 GHz em ligações sem fios a uma distância de 1 metro com $BER = 1 \times 10^{-3}$, 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 espectro 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 [15]. 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 [15], torna-se evidente que a banda do espectro 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 outras bandas, mas também é uma região do espectro que não sofre muito devido às más condições atmosféricas. Para além disso, a utilização destas bandas de frequência altas acabará por aliviar o espectro relativamente à sua escassez e às suas limitações de capacidade.

Tendo em conta esta nova abordagem do espectro, o projeto iBrow tem vindo a desenvolver metodologias que permitem a manufatura de transdutores 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 transdutores 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 transdutores propostos pelo projeto iBrow. Esta transmissão terá de ser realizada em série visto que estes mesmos transdutores 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.

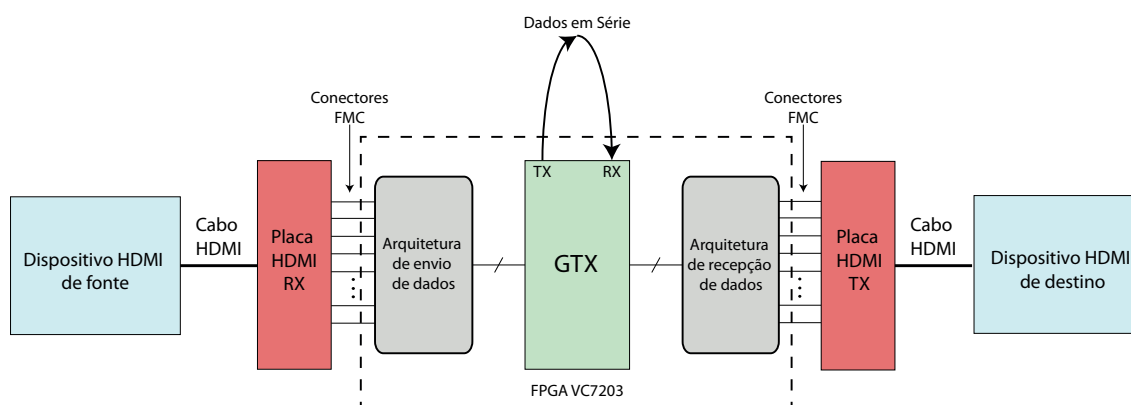


Figura 1.1: Diagrama geral do problema proposto

O projeto faz uso de uma FPGA VC7203 (*Virtex-7*) que possibilita a implementação de uma arquitetura adequada 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 permitam 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 disponíveis. 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.

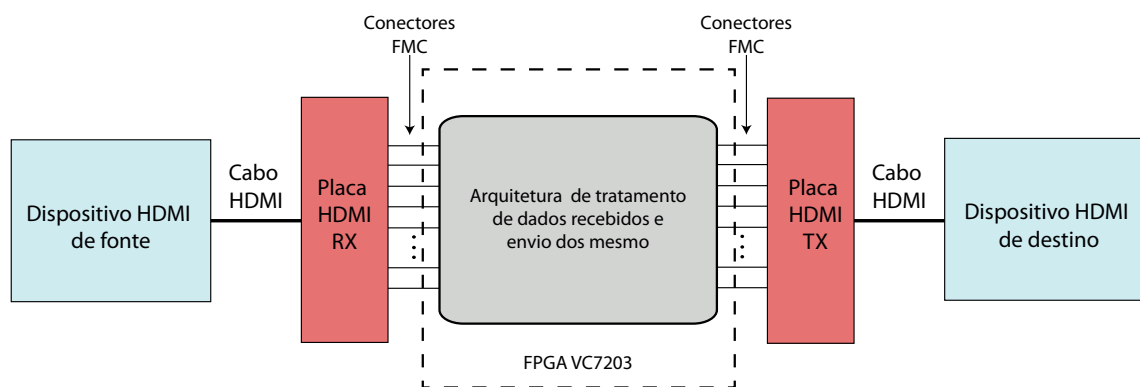


Figura 1.2: Diagrama geral da primeira parte problema

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 arquitetura 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

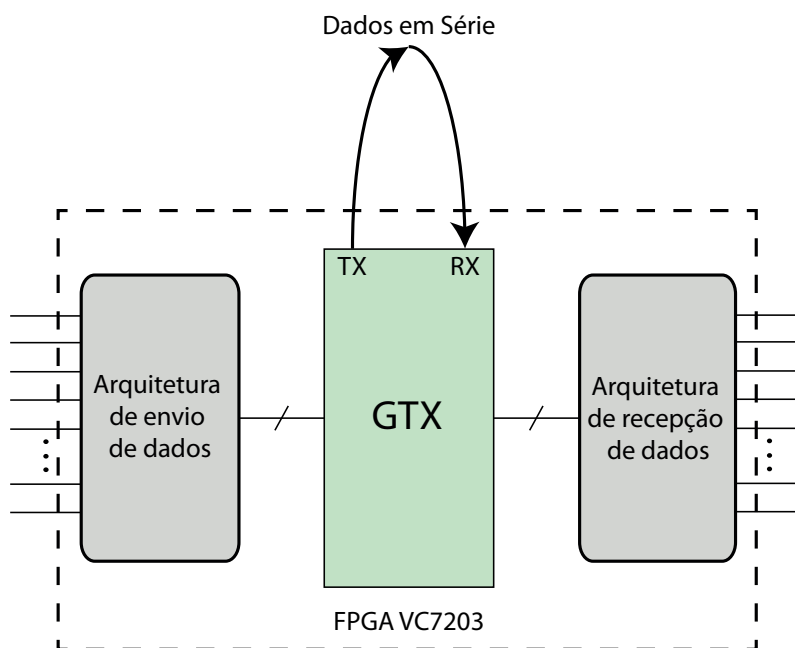


Figura 1.3: Diagrama geral da segunda parte do problema

os transceptores da FPGA que automaticamente fazem a serialização dos dados. Depois os dados em série são transmitidos por um cabo físico, e recebidos nos transceptores 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 capítulos que vão desde uma revisão bibliográfica do problema até à descrição da resposta ao mesmo. No capítulo 2 é feita uma revisão bibliográfica sobre os diversos aspectos que o problema apresenta, apresentando também considerações de diversos autores sobre tal.

No capítulo 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 capítulo 4 aborda todas as questões relativas à transmissão em série. São apresentadas 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 capítulo 5 expõe as conclusões finais de todo o trabalho realizado e ainda aborda

trabalho que pode futuramente 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/áudio

As interfaces de áudio e vídeo definem parâmetros físicos e interpretações dos sinais recebidos, segundo [16]. 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 [16]:

- ***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 [13]:

2.2.1 DDC - *Display Data Channel*

É um conjunto de protocolos utilizado nas comunicações digitais entre um dispositivo 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 dispositivo *sink*, no sentido de perceber quais os modos suportados pelo mesmo. Este canal é também ativamente usado para HDCP (*High-Bandwidth 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 pixels 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 possível dispositivos individuais 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 áudio 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 equipamento 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, áudio 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 dispositivos não autorizados. Isto é, através de um protocolo cujo nome já foi referido anteriormente, HDCP (*High-Bandwidth 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 recorrente nos dias atuais.

2.3 Transmissão de dados HDMI

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. Para que seja possível a conexão entre dois dispositivos HDMI é necessário logo à partida que existam dois conectores HDMI, e ainda que o sinal que vem transportado no cabo seja decodificado para apenas serem transmitidos os dados referentes à imagem e ao som.

O projeto recorre então à utilização de um tipo de *hardware* capaz de cumprir as duas funções descritas, que são duas placas HDMI com o nome de TB-FMC-HDMI2. Ao todo são usadas duas placas, uma recetora do sinal HDMI (RX) e outra transmissora do sinal HDMI (TX) sendo que cada uma tem dois canais (RX0 e RX1, TX0 e TX1). No caso da placa recetora o sinal proveniente da fonte HDMI é recebido e, através dos conectores FMC, envia apenas os dados referentes à imagem e som. No caso da placa transmissora o processo é inverso, ou seja, os conectores FMC recebem os dados apenas referentes à imagem e ao som e a placa envia um sinal HDMI através do seu conector para o dispositivo final HDMI.

2.3.1 Conexão à FPGA XILINX VC7203 Virtex-7

Na figura 2.1 da página 12 visualiza-se a placa de desenvolvimento a ser utilizada no projeto. Assinalado a tracejado vermelho é possível visualizar os 3 conectores FMC (*FPGA Mezzanine Card*) que a placa VC7203 disponibiliza.

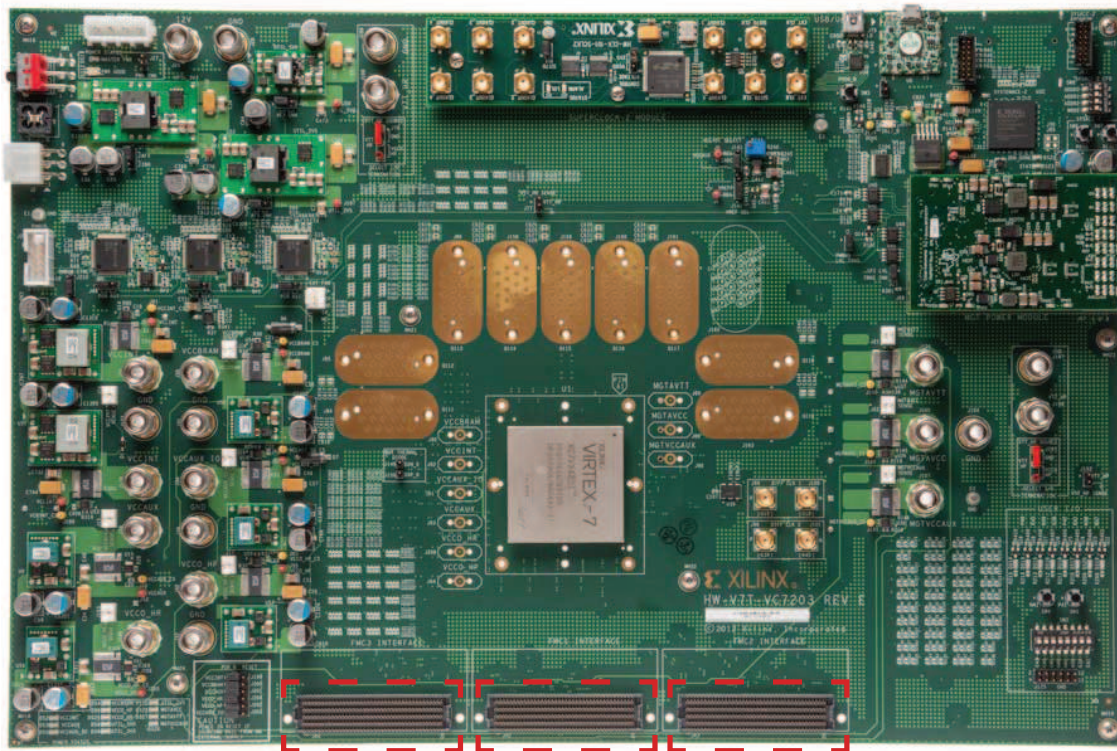


Figura 2.1: Vista Geral da FPGA VC7203 Virtex-7 retirada de [1]

Os conectores assinalados na imagem desta placa tratam-se de conectores FMC HPC (*High Pin Count*) e permitem conectar as placas HDMI com a FPGA. Segundo [17], os conectores FMC implementam determinadas normas que permitem uma rápida transmissão de dados entre placas. Existem dois tipos de conectores FMC : LPC (*Low Pin Count*) que disponibiliza 160 pinos e ainda HPC (*High Pin Count*) que dispõe de 400 pinos. Ainda segundo [17], qualquer tipo dos conectores consegue alcançar uma velocidade de conexão

de até 2 Gb/s para sinais com sinalização diferencial e única. Para além da rápida velocidade de transmissão, outra grande vantagem que a utilização deste tipo de conectores traz é número de conexões que permite (400 pinos no caso de HPC) para a pequena area que ocupam.

Em específico nesta FPGA existem 3 conectores FMC (FMC1, FMC2 e FMC3) que permitem diferentes conectividades. Segundo [1], o conector FMC1 dispõe de 68 pares diferenciais definidos pelo utilizador e ainda 4 sinais de relógio diferenciais. O mesmo acontece para para o conector FMC2. No entanto, o conector FMC3 apenas disponibiliza 65 pares diferenciais que podem ser definidos pelo utilizador e 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 Recetor

Na figura 2.2 na página 13 é possível visualizar o diagrama de blocos do recetor.

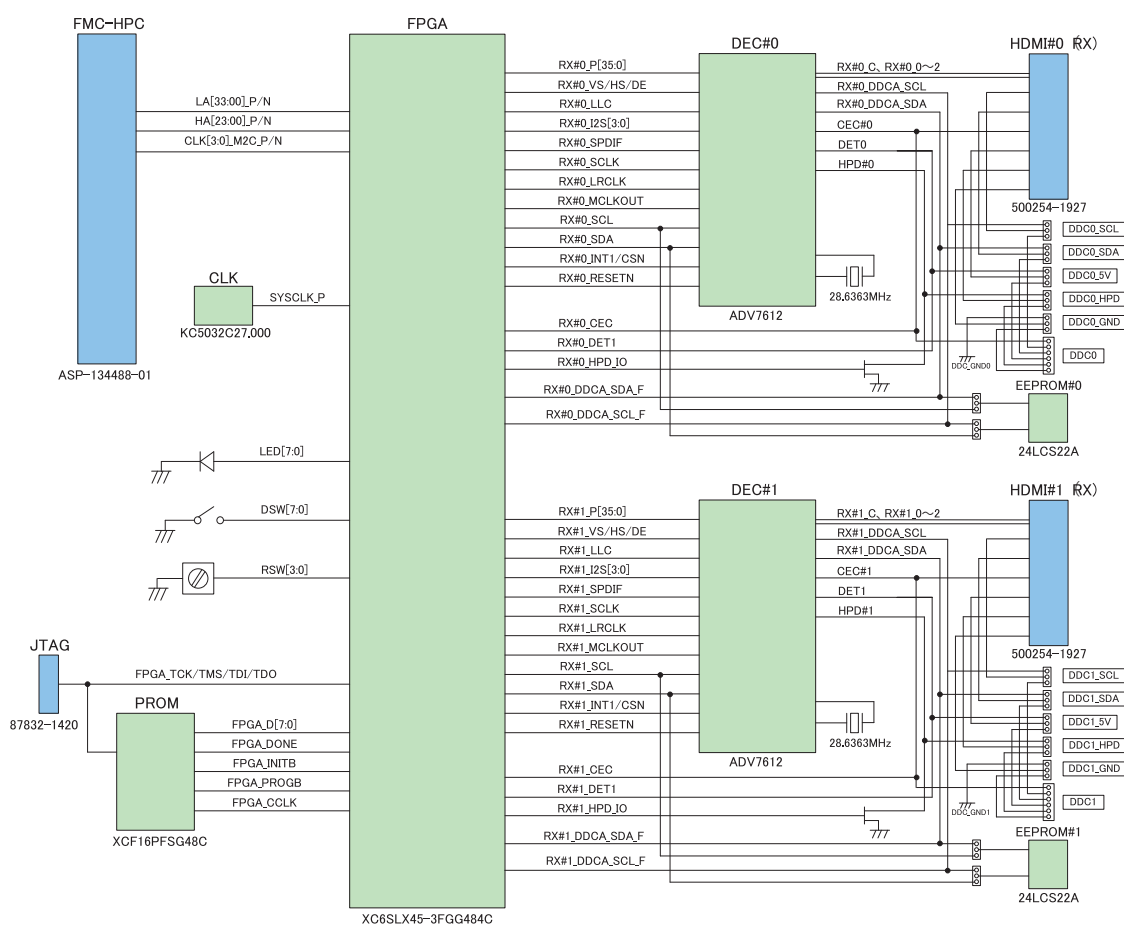


Figura 2.2: Diagrama de blocos de TB-FMCH-HDMI2 RX retirado de [2]

Através de uma rápida observação deste diagrama é possível concluir que se pode dividir as suas principais funções em duas partes que passa a ser descritas nos próximos sub-capítulos.

2.3.2.1 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.

2.3.2.2 Interface com o conector FMC (da FPGA localizada na placa para o conector FMC)

Após passarem pela FPGA embebida na placa são passados os seguintes sinais presentes na tabela 2.1 da página 14 (para o caso em que a FPGA está configurada por *default*):

Nome do Pin	Input/Output	FPGA para FMC	FMC para FPGA
CLK0_M2C_P	Output	RX#0_LL	RX#0 sinal LLC
CLK1_M2C_P	Output	RX#1_LL	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]

Através da análise da tabela 2.1 e do diagrama de blocos da placa na figura 2.2 conclui-se que o integrado ADV7612 é capaz de colocar na sua saída vários sinais (tanto referentes à imagem como ao som), no entanto esses sinais não são todos transmitidos para os conectores FMC. Isto acontece por causa da configuração presente na FPGA embebida que determina que sinais envia para os conectores.

Através da leitura de [10] conclui-se que a configuração presente na FPGA embebida para além de seleccionar os dados a enviar para os conectores FMC, configura também

alguns parâmetros do integrado ADV7612 que permitem que este reproduza sinais num determinado formato e com um determinado número de bits.

Para esta configuração são transmitidos para os conectores dados FMC referentes à imagem e sinais de sincronização do mesmo. Os dados da imagem são enviados do recetor 0 entre os pinos LA03_P a LA32_P, e do recetor 1 entre LA03_N a LA32_P. O sinal “*data enable*” é um sinal que sinaliza a chegada de dados e está ativo quando estão a ser transmitidos os sinais referentes a cada pixel. *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.

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 15 ilustra esta situação.

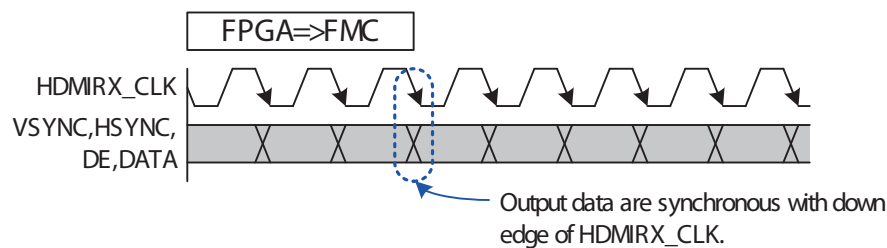


Figura 2.3: Amostragem dos dados provenientes da FPGA no recetor, retirada de [2]

2.3.3 Transmissor

O diagrama de blocos do transmissor está representado na figura 2.4 na página 16. Mais uma vez é possível dividir o diagrama em duas principais funções que passam a ser descritas.

2.3.3.1 Interface com o conector FMC (do conector FMC para a FPGA localizada na placa)

No caso do transmissor o processo é feito no sentido inverso, ou seja, os sinais são lidos dos conectores FMC da placa HDMI e são amostrados para a FPGA embebida na mesma na transição de 0 para 1 do sinal de relógio do HDMI, tal como ilustra a imagem 2.5 da página 16 e depois são processados pela FPGA de maneira a serem enviados para o transmissor HDMI (ADV7511).

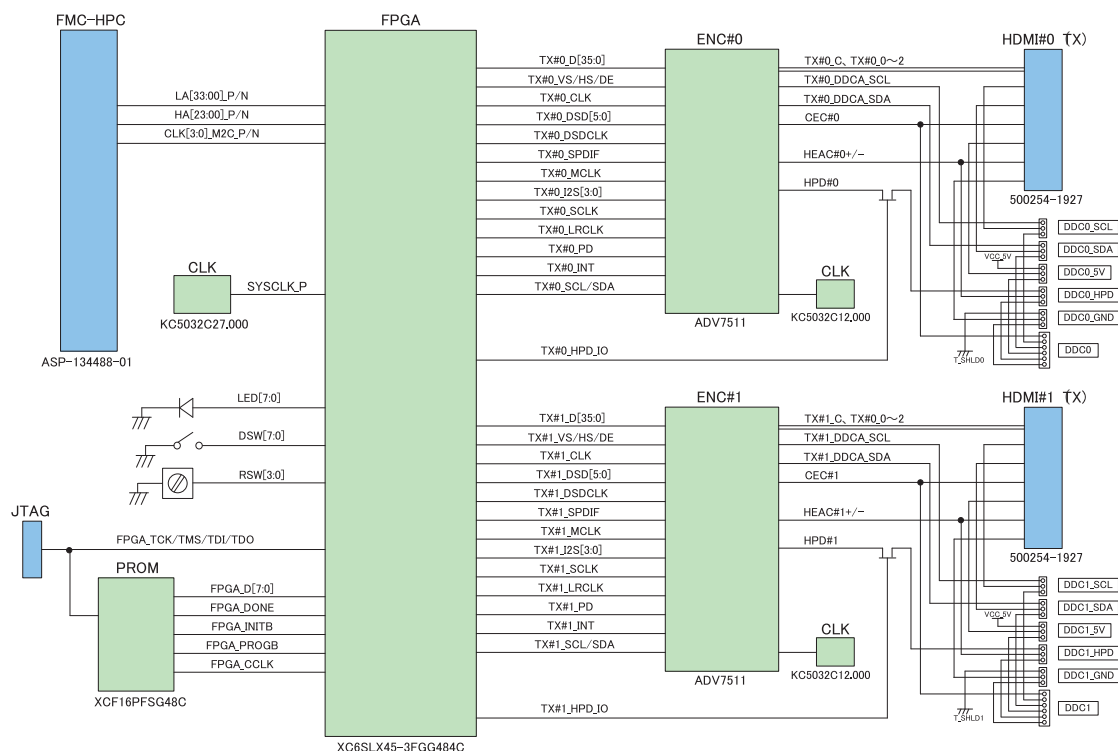


Figura 2.4: Diagrama de blocos de TB-FMCH-HDMI2 TX retirado de [2]

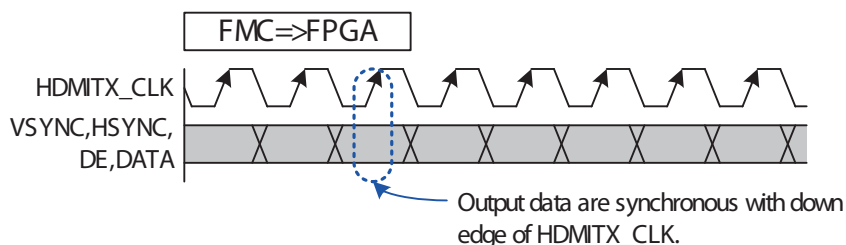


Figura 2.5: Amostragem dos dados provenientes do FMC no recetor, retirada de [2]

Os sinais representados na tabela 2.2 na página 17 são equivalentes aos sinais presentes na tabela 2.1 na página 14, mais uma vez com a placa configurada por *default*, e correspondem aos sinais que a placa transmissora deve receber para enviar para a FPGA.

2.3.3.2 Transmissor HDMI (da FPGA localizada na placa para ADV7511)

Os sinais são então processados de maneira a envia-los para o bloco ADV7511 localizado na placa. Através da análise dos documentos [18] e [19] conclui-se que para além dos dados de imagem e os seus sinais de controlo, são também enviados para o integrado ADV7511 alguns sinais de controlo que indicam ao integrado que tipo de formato de imagem são enviados, ou quantos números de bits tem, entre outras informações. De seguida,

Nome do pin	Input/Output	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]

o ADV7511 converte o sinal para o poder enviar através do cabo HDMI para o dispositivo final.

2.4 Conexão de alta velocidade em série

Nesta secção será abordado o tema de comunicação em série em alta velocidade, desde as suas vantagens e desvantagens até tipos de arquiteturas habitualmente utilizados.

2.4.1 Comunicações em paralelo VS comunicações em série

Desde sempre que tanto a comunicação em série como em paralelo têm vindo a ser utilizadas para as diferentes aplicações que envolvem a transmissão de dados entre módulos, e neste capítulo serão abordadas as diferentes características de cada um. A comunicação de dados em paralelo é utilizada para comunicações relativamente curtas pelo facto de ser mais simples e não trazer tantas implicações. Olhando para um exemplo em concreto deste projeto: a comunicação entre as placas HDMI e a FPGA VC7203 é feita em paralelo através dos conectores FMC, tal como explicado em 2.3.1, porque é uma distância bastante curta e não envolve preocupações no que toca a dobrar ou triplicar a frequência de transmissão e vice-versa. Afinal, para transmitir um determinado número de dados em série é necessário multiplicar esse número de dados pela frequência de transmissão em paralelo do lado do transmissor, e do lado do recetor é necessário lidar com eventuais desalinhamentos.

Quando este tipo de comunicações começaram a ser utilizadas a distâncias maiores e a ser também necessário uma velocidade de comunicação maior, então começaram a haver mais problemas relativamente ao uso desta forma de transmissão. Apesar de, segundo [3], terem sido aplicados métodos que viessem melhorar o desempenho destas comunicações em termos de velocidade, como por exemplo, a sinalização diferencial que veio aumentar a mesma, existem ainda bastantes desvantagens segundo [5]. Há uma desvantagem que se torna bastante óbvia de constatar que é o custo: ter uma ligação em série com um cabo é muito menos dispendioso do que ter 400 cabos (no caso da transmissão entre as placas HDMI e a FPGA) para ter uma ligação em paralelo. Para além disto existem mais três problemas também apontados por [5] : *clock skew*, *data skew* e *crosstalk*. *Clock* e *data skew* tratam-se de pequenos desvios na chegada ao recetor dos dados e dos sinais de relógio, isto porque nem todos são transmitidos exatamente à mesma velocidade e como tal podem provocar pequenos atrasos. Apesar de serem pequenos, podem vir a causar problemas visto que a velocidade de transmissão é bastante alta. Segundo este mesmo autor, existem já técnicas capazes de corrigir estes atrasos relativamente ao sinal de relógio (devido à sua periodicidade), e a correcção dos dados também é possível, no entanto é muito mais problemática. A *crosstalk* trata-se de uma interferência entre cabos adjacentes, inerente à transmissão, que se torna ainda mais problemática com o aumento dos mesmos.

Estas razões, entre outras, têm vindo a motivar o desuso das comunicações em paralelo para transmissões de alta velocidade. Em [3] são mostradas algumas das razões para usar as ligações em série para tal efeito. O autor menciona o facto da utilização de menos pinos para uma maior largura de banda, o que faz com que o custo da transmissão baixe consideravelmente. Este autor considera também um problema das transmissões em paralelo que não acontece em série: as consequências das constantes alternâncias das saídas. Isto é, numa ligação em paralelo o mais provável é que todas as saídas estejam a alterar constantemente e como tal esta alternância na massa acaba por criar ruído que é propagado na ligação. Claro que é possível resolver este problema aplicando sinalização diferencial aos sinais transmitidos, mas em contrapartida aumenta o número de pinos necessários, aumentando por isso o custo da ligação.

Aparentemente, torna-se obvio que as ligações em série devem ser usadas para este efeito uma vez que só acartam consigo vantagens, no entanto é necessário ter em consideração as desvantagens desta utilização.

Logo à partida o primeiro problema que rapidamente o autor de [3] constata é a integridade do sinal, isto é, é expectável que para garantir a integridade do sinal será necessário recorrer a mais lógica que o garanta. Para além disso, também será de esperar que este tipo de comunicações exija placas, cabos e conectores de alta velocidade que são mais caros. E claro, uma vez que estas comunicações trabalham a uma cadência bastante elevada também de esperar que seja necessário fazer simulações digitais em bases de tempo mais pequenas o que pode trazer algumas complicações.

Em conclusão, e tal como o autor de [3] menciona, hoje em dia a utilização de ligações

em série já não são utilizadas apenas na indústria das telecomunicações mas acaba por ser transversal a outras trantas que a usam. O autor remata ainda que o futuro da eletrônica passa por comunicações em série.

2.4.2 Considerações sobre arquiteturas de transmissão de dados em série

Neste sub-capitulo serão apresentadas conseiderações que devem ser tomadas quando se trata de implementar uma arquitetura que permita enviar dados em série a alta velocidade.

Arquitetura de serializadores e deserializadores de alta velocidade

No projeto desenvolvido lida-se com sinais proveninetes da fonte HDMI em paralelo e para que seja possivel transmiti-los em série a alta velocidade é necessário implementar um arquitetura capaz de lidar com este processo de conversão. Neste sub-capitulo serão abordadas algumas técnicas de implementação de serializadores e deserializadores, as suas características e cuidados na sua implementação.

Na figura 2.6 na página 19 é apresentada uma simples arquitetura de um serializador e deserializador proposto pelo autor de [3]. A figura ilustra um diagrama de blocos em que cada bloco tem uma determinada função para se obter uma conversão de dados paralelo para série e vice-versa.

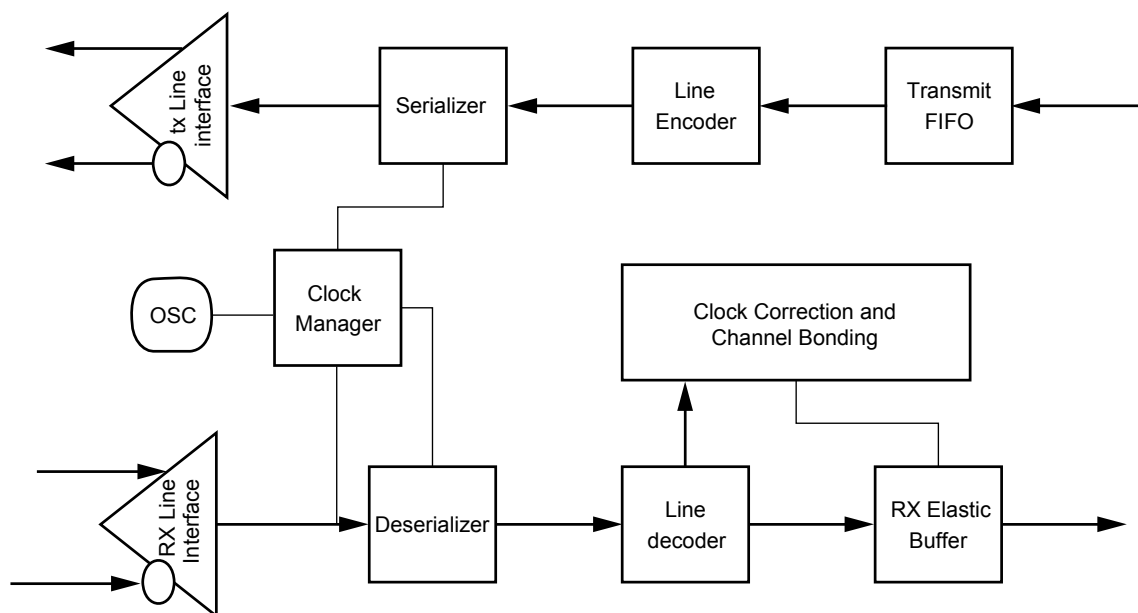


Figura 2.6: Arquitetura simples de um serializador e deserializador, retirada de [3]

As funções de cada bloco do diagrama do serializador apresentado passam a ser brevemente descritas:

- **Transmit FIFO:** Trata-se de uma memória FIFO que guarda os dados em paralelo antes destes serem enviados para o resto da arquitetura. No diagrama poderia também estar representa uma fonte direta de sinais em paralelo (como é o caso deste projeto).
- **Line Encoder:** Este bloco trata-se de um bloco opcional, e nem em todas as arquiteturas de serializadores/deserializadores está presente. Este bloco codifica os dados que recebe para um formato "*line-friendly*" ¹. Trata-se de um formato que ajuda o recetor a recuperar os sinais da dados e relógio, normalmente isto envolve eliminar longas tramas de zeros ou uns, de maneira a garantir que há um equilíbrio entre o número de uns e zeros numa trama.
- **Serializer:** Tal como o nome indica este é um bloco que serializa os dados que recebe, ou seja, quando recebe um determinado número de dados em paralelo (x dados) a uma determinada cadência (frequência y), transforma-o num stream de dados a uma taxa de x multiplicado por y.
- **TX Line Interface:** Este bloco acaba por ser a interface final do serializador com o cabo físico, e geralmente também sofre determinados processos que permitem a melhor recuperação do sinal do lado do recetor.

Por outro lado, o deserializador tem de fazer todo o processo inverso que o bloco serializador faz. As funções de cada bloco passam a ser brevemente descritas.

- **RX Line Interface:** É a interface do deserializador com o cabo físico. Já pode incluir alguma equalização do sinal passiva ou ativa.
- **Deserializer:** Converte os dados em série que recebe a uma cadência de x multiplicado por y, em x dados paralelo a uma cadência de y.
- **Line Decoder:** Descodifica os dados recebidos, se tal processo foi realizado do lado do transmissor.
- **RX Elastic Buffer:** Este bloco permite o alinhamento dos dados recebidos para os respectivos limites. Tal pode ser feito automaticamente ou recorrer-se a palavras de alinhamento, também chamadas de "virgulas".
- **Clock Correction and Channel Bonding:** Este bloco permite que haja correcção entre as diferenças de sinais de relógio, e ainda correcção de atrasos entre diferentes canais (caso haja transmissão em vários canais).

Existe ainda um bloco que é comum tanto ao serializador como ao deserializador que é o **Clock Manager** que essencialmente é responsável pelos diversos processar que os sinais de

¹é o termo usado pelo autor de [3]

relógio necessitarão: desde multiplicações de frequências, divisões e até mesmo recuperação do mesmo.

Esta arquitetura aqui apresentada acaba por apresentar os blocos essenciais para o correcto funcionamento de um serializador e deserializador, no entanto existem outras características que podem ser adicionadas: desde os diferentes tipos de codificações possíveis, até códigos detetores e corretores de erros que podem ser adicionados à arquitetura.

Restrições na utilização de circuitos de serializadores e deserializadores

Quando se passa para a implementação de serializadores e deserializadores é necessário ter em conta algumas considerações relativamente aos circuitos utilizados. E segundo [5], logo à partida existem grandes restrições no que toca aos circuitos utilizados nestes tipos de arquiteturas. 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 [5], é conseguir implementar circuitos digitais de alta velocidade e que ao mesmo tempo têm um baixo consumo de potência. Este mesmo autor apresenta duas principais técnicas utilizadas para 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.

Codificação dos sinais e sua importância

O bloco de codificação de sinais insere-se nesta arquitetura do transmissor devido à sua importância relativamente à recuperação dos dados do lado do recetor. Estes blocos modificam os sinais para um formato que permite ao recetor recuperar os sinais de relógio mais facilmente garantindo que existe um número suficiente de transições entre zero e um. Segundo [3], este bloco para além de fazer o que já foi descrito garante que existe um balanço entre zeros e uns na linha de transmissão (*Direct Current Balance* - *DC Balance*)

e opcionalmente podem também implementar mecanismos de correção de sinal de relógio, sincronização de blocos e eventuais sinalização de canais (quando é usado mais do que um canal físico na transmissão).

Ainda segundo o autor de [3], um dos esquemas de codificação e decodificação mais utilizados em diversos protocolos é 8B/10B. Este tipo de codificação converte palavras de 8 bits em 10 bits garantindo um número suficiente de transições entre zeros e uns nas palavras codificadas. Este mecanismo irá garantir que o sinal de relógio será corretamente recuperado do lado do recetor. Na tabela 2.3 na página 22 é possível encontrar dois exemplos de codificação de palavras de 8 bits em 10 bits. Note-se que palavras que não apresentam transições entre zero e um (no caso da primeira linha da tabela) ou poucas (no caso da segunda linha da tabela) passam a ter transições suficientes para que o sinal de relógio possa ser recuperado do lado do recetor.

Palavra de 8 bits	Símbolo de 10 bits
00000000	1001110100
00000001	0111010100

Tabela 2.3: Exemplo de codificação 8B/10B, retirada de [3]

Este método de codificação garante que existe um balanço entre o número de zeros e uns na linha de transmissão de uma maneira particular, chamada disparidade. Afinal, a maneira mais fácil de garantir este balanço seria limitar os números de zeros e uns em cada palavra (5 zeros e 5 uns), que por acaso são os exemplos apresentados na tabela 2.3. Contudo esta restrição viria limitar o número de palavras de 10 bits possíveis de obter. Em vez disso, este método utiliza dois símbolos para palavra codificada. Isto é, para uma palavra de 8 bits codifica de duas maneiras possíveis em que, geralmente, uma tem 6 zeros e 4 uns e a outra tem 4 zeros e 6 uns. Essencialmente, para cada palavra de 8 bits há duas possibilidades de codificação e durante a transmissão das palavras a seleção é feita com base na necessidade de balanço de zeros e uns na linha. Geralmente os símbolos são reconhecidos por + e -.

Outra vantagem da utilização deste tipo de codificação tem a ver com a detecção de erros do lado do recetor. Isto é, este monitoriza o balanço de zeros e uns tendo em conta a disparidade e caso haja alguma violação nestas regras então o recetor é capaz de detetar a ocorrência de erros.

Hexadecimal	Palavra de 8 bits	Símbolo de 10 bits (-)	Símbolo de 10 bits (+)
EA	11101010	0101011110	0101010001
FF	11111111	1010110001	0101001110
A4	10100100	1101011010	0010101010

Tabela 2.4: Exemplo de palavras de 8 bits codificadas em 8B/10B, retirada de [3]

Na tabela 2.4 da página 22 são apresentadas 3 palavras de 8 bits codificadas nas suas duas possibilidades. A palavra apresentada na 1ª linha, EA em hexadecimal, na sua codificação negativa apresenta 6 uns e 4 zeros, contudo na sua forma positiva apresenta 4 uns e 6 zeros. O mesmo acontece com a palavra apresentada na última linha. No entanto a palavra representada na 2ª linha apresenta o mesmo número de zeros e uns em ambos os formatos. Se hipoteticamente fosse necessário transmitir estas palavras seguidas, o codificador teria em atenção o balanço entre zeros e uns na linha de transmissão aquando a transmissão da palavra EA e A4 para que o mesmo fosse garantido.

Alinhamento da transmissão

Uma função importante do deserializador, segundo o autor de [3] é o alinhamento dos dados que chegam em série. Por outras palavras, é importante que o deserializador reconheça os limites de uma determinada palavra transmitida. Para esse efeito são usados uns caracteres especiais que a codificação 8B/10B disponibiliza. Segundo [3], este tipo de codificação dispõe de 12 caracteres que se codificam em 12 caracteres de controlo geralmente conhecidos por *K-characters*. Estes são geralmente usados para alinhamento e controlo devido às suas particulares características. Esses mesmos caracteres são apresentados na tabela 2.5 da página 23.

Nome	Hexadecimal	Palavra de 8 bits	Símbolo de 10 bits (+)	Símbolo de 10 bits (-)
K28.0	1C	00011100	0011110100	1100001011
K28.1	3C	00111100	0011111001	1100000110
K28.2	5C	01011100	0011110101	1100001010
K28.3	7C	01111100	0011110011	1100001100
K28.4	9C	10011100	0011110010	1100001101
K28.5	BC	10111100	0011111010	1100000101
K28.6	DC	11011100	0011110110	1100001001
K28.7	FC	11111100	0011111000	1100000111
K23.7	F7	11110111	1110101000	0001010111
K27.7	FB	11111011	1101101000	0010010111
K29.7	FD	11111101	1011101000	0100010111
K30.7	FE	11111110	0111101000	1000010111

Tabela 2.5: Caracteres de controlo específicos da codificação 8B/10B, retirada de [3]

Para conseguir alinhar, o recetor procurar por uma palavra de alinhamento, conhecida por "*comma*" (virgula em português), e quando a encontra repõe os limites de alinhamento das palavras. Como esta procura é contínua, a partir da primeira detecção todas as palavras de alinhamento que forem detetadas encontrarão o alinhamento já feito. O valor deste símbolo que delimita as palavras (*comma*) pode à partida ser definido no transceptor, contudo por vezes pode vir já pré-definido. A imagem

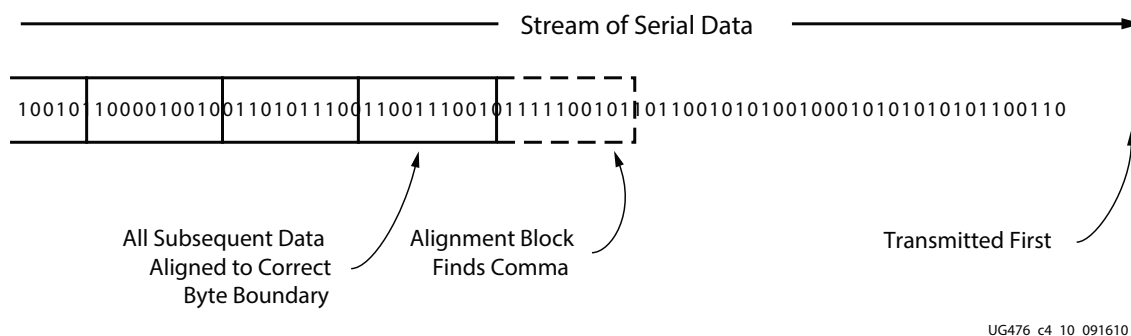


Figura 2.7: Ilustração do alinhamento em série quando encontrada a *comma*, retirada de [4]

A imagem 2.7 da página 24 ilustra o que acontece ao *stream* de bits que chegam ao recetor em série. Aquando a detecção da palavra de alinhamento todos os dados que são retirados daí para a frente já se encontram alinhados pela delimitação da palavra. Na figura 2.8 da página 24 é ilustrado o mesmo exemplo mas em paralelo para melhor entendimento do processo de alinhamento. Todos os dados recebidos no recetor antes da palavra de alinhamento são considerados dados não alinhados.

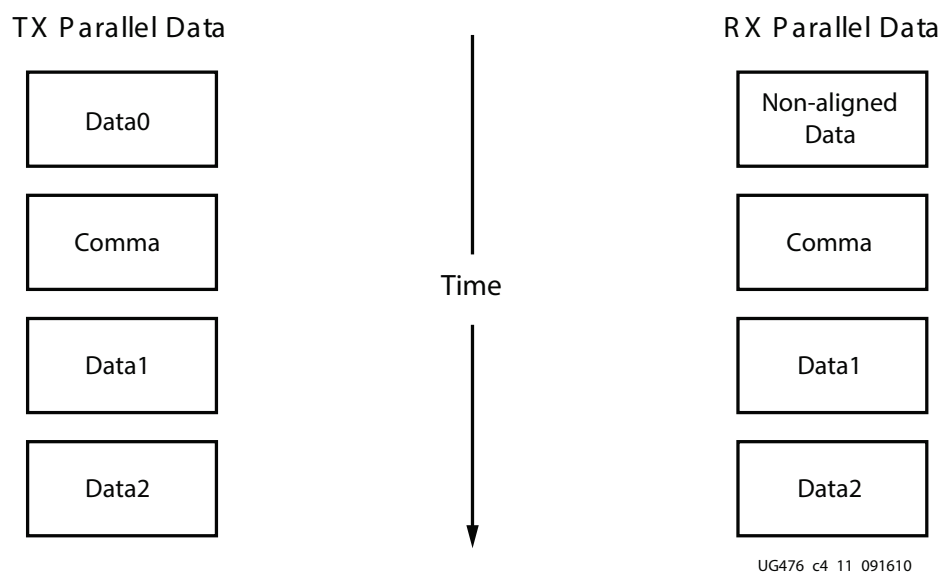


Figura 2.8: Ilustração do alinhamento em paralelo quando encontrada a *comma*, retirada de [4]

Tal como o autor de [3] menciona, a palavra de alinhamento deve ser única e diferenciável de todos os outros dados que possam ser transmitidos. Este mesmo autor sugere que os caracteres especiais *K-characters* mencionados na tabela 2.5 sejam utilizados para tal efeito, sendo que os mais adequados são o K28.1, K28.5 ou K28.7, isto porque todos têm um padrão inicial de 7 bits de 1100000, que apenas é encontrado nestes caracteres. Tal padrão não é encontrado em nenhuma outra sequência ordenada de dados ou em qualquer

outro *K-character*, e por isso é ideal para o alinhamento. Este autor reforça ainda que quando um protocolo de comunicação está a ser contruído, como é o caso deste projeto, a maneira mais segura é "pedir emprestado" este tipo de sinais de controlo a protocolos bem conhecidos.

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

Os blocos de serialização e deserialização da arquitetura têm uma grande importância no correto funcionamento de toda a arquitetura, isto porque, tal como o nome indica, estes convertem os dados em paralelo em série e vice-versa. Tal como já foi referido anteriormente, o serializador recebe N dados a uma frequência de X , e transmite esses N dados a uma frequência de $X \cdot N$. Por outro lado o deserializador, reduz a frequência dos dados, convertendo-os em paralelo, para de seguida ser enviada para o resto da arquitetura para processamento desses mesmos dados. Por isso, é necessário tomar em atenção diversas características quanto à escolha destas arquiteturas, tais como a sua latência.

O autor de [5] propõe 3 tipos de arquiteturas para este tipo de bloco que são apresentadas na imagem 2.9 na página 25 e que passam a ser descritas as suas vantagens e desvantagens.

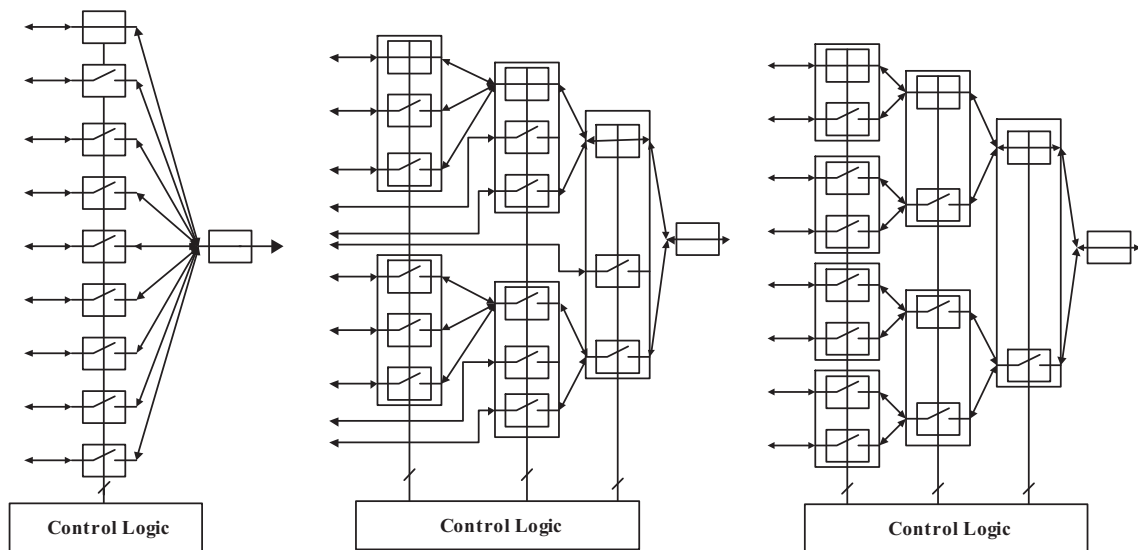


Figura 2.9: Arquiteturas de PISO/SIPO, retirada de [5]

No circuito mais à esquerda, denominada de a) pelo autor de [5], visualiza-se uma estrutura de um único andar, todavia esta é demasiado lenta devido às capacidades intrínsecas largas no nó de conversão. No circuito do centro, b), é representada uma topologia heterogênea que se torna mais rápida que a apresentada em a). Já no último circuito apresentado na imagem (o da mais à direita), c), é representada uma topologia em árvore binária que é a mais rápida segundo o autor de [5]. O autor reforça ainda a ideia de que a utilização de multiplexadores e des-multiplexadores de 2:1 e 1:2 respectivamente são muito

importantes para se obter uma arquitetura de funcionamento rápido. É de notar ainda que quando se utiliza uma arquitetura de serialização/deserialização em árvore binária, tal como o autor sugere, as portas de entrada do serializador terão dezenas de entradas, tal como as portas de saída do deserializador terão dezenas de saída, e por isso alguns pontos que necessitam de circuitos de velocidade elevada e outros que não. Para tal é necessário ter em atenção que tipo de circuitos são usados em cada etapa do serializador dada a sua necessidade de velocidade. O autor sugere a utilização de circuitos lógicos CML para as etapas que necessitam de trabalhar a altas velocidades, e CMOS para as que não necessitam de tanta velocidade.

Em situações em que as taxas de débito são muito elevadas a utilização de *shift registers* torna-se também bastante eficiente para implementar um serializador/deserializador. Na imagem 2.10 da página 26 é possível visualizar o exemplo de um serializador com a base em shift-register de 4 bits e na imagem 2.11 da página 26 um deserializador com base na mesma arquitetura de 4 bits.

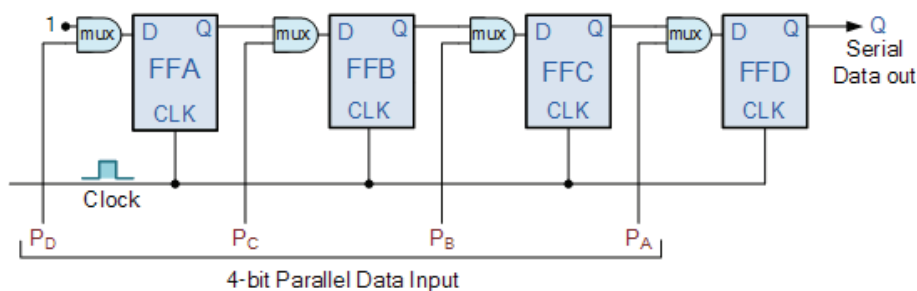


Figura 2.10: Arquitetura de um *shift-register* serializador de 4 bits retirado de [6]

O serializador lê os dados em paralelo a um sinal de relógio cuja frequência é mais baixa e após serem lidos (quando o sinal de escrita na arquitetura estiver desligado) então os sinais saem à cadência de um sinal de relógio múltiplo superior ao anteriormente mencionado, obtendo-se desta maneira um sinal em série de alta velocidade.

No caso do deserializador apresentado na imagem 2.11, e segundo a fonte [6], os dados que chegam em série a uma frequência determinada pelo sinal de relógio *clock*, são movidos do registo A para o B e de seguida para o C e por fim para o D a essa mesma frequência (que corresponde à frequência de transmissão em série). Nas saídas QA, QB, QC e QD obterão-se esses sinais, que chegaram ao início da cascada de registos em série, em paralelo fazendo a sua amostragem a uma frequência mais baixa (múltipla da mais alta, tal como acontecia no serializador).

Os autores de [7] utilizam este tipo de estrutura de serializador para obter uma ligação em série de 1.25 Gb/s. A arquitetura em questão é apresentada na imagem 2.12 da página 27.

Neste caso em específico, a cada ciclo de relógio de "*word-cycle*" é lida uma palavra de 10 proveniente de um multiplexador para um registo com nome na imagem de "*Register*".

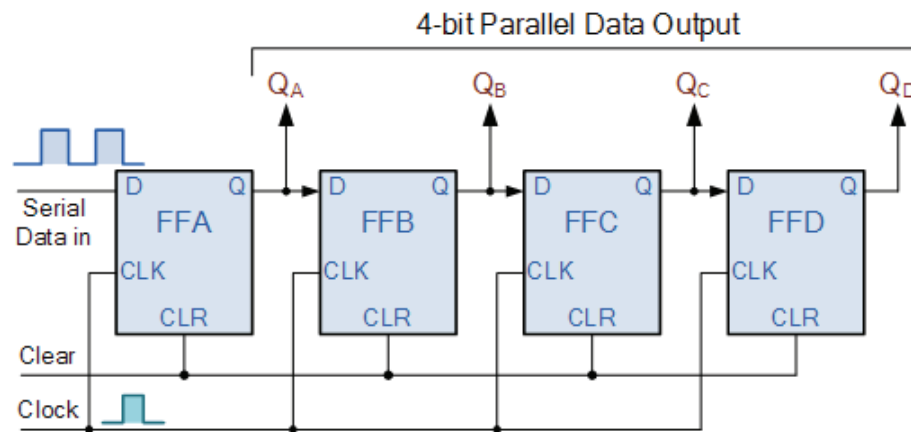


Figura 2.11: Arquitetura de um *shift-register* serializador de 4 bits retirado de [6]

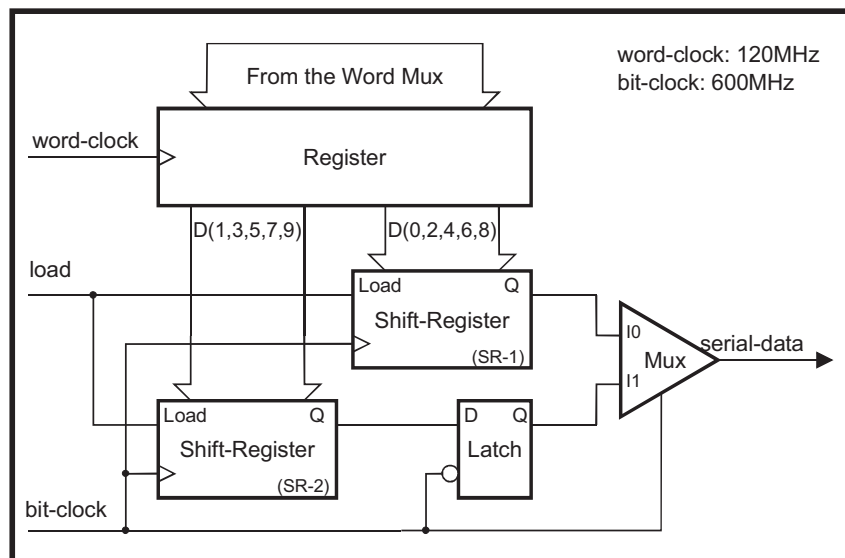


Figura 2.12: Exemplo de um serializador de 10 bits, retirado de [7]

A saída deste módulo é então dividida em dois "*shift-register*" de 5 bits cada. Tal como a figura ilustra, os bits são separados de forma a que bits adjacentes não entrem em *shift-registers* diferentes. A saída do primeiro *shift-register* é a entrada do multiplexador de saída, enquanto que a saída do segundo *shift-register* é a entrada de uma *latch* que serve exatamente para atrasar meio ciclo de relógio a operação de alternância de registos em relação ao primeiro *shift-register*. As saídas do multiplexador de saída são seleccionadas pela alternância do sinal de relógio com frequência mais elevada, fazendo assim uma correcta selecção entre os bits adjacentes e obtendo a saída com a velocidade pretendida para este caso em específico: 1.25 Gb/s. Este exemplo é aqui mencionado para que se possa perceber como é possível tirar partido deste tipo de arquiteturas de serialização.

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 cujo controlo e funcionamento será explicado mais à frente neste relatório .

Interfaces com a camada física

As interfaces com as camadas físicas em ambos os lados da transmissão (recetor e transmissor) incluem circuitos analógicos que permitem transmitir e receber sinais diferenciais. Contudo 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. Por isso existe uma necessidade de utilizar técnicas que melhorem a ligação entre os dois terminais. Segundo [5], esta melhoria poderia ser facilmente obtida através da utilização de canais de ligação de melhor qualidade. No entanto esta opção traz custos acrescidos à ligação.

Assim sendo recorre-se a técnicas implementadas nestas mesmas interfaces que por um lado são responsáveis por "preparar" os sinais a serem transmitidos para um canal ruidoso, precavendo a sua alteração, e por outro são responsáveis de corrigir os dados recebidos de acordo com determinados parâmetros. Estas técnicas que estão aqui a ser mencionadas têm o nome de pré-ênfase do lado do transmissor e equalização do lado do recetor e passarão a ser descritas com mais detalhe.

Durante a transmissão no canal físico é normal haver o fenómeno de interferência inter-simbólica que ocorre devido ao facto de haver longos períodos de transmissão de um determinado valor, seguido de um curto período de tempo com um valor oposto. Segundo a fonte [3], o meio físico tem menos tempo para carregar o valor que é transmitido no curto período de tempo e por isso acaba por produzir uma amplitude mais baixa. Para

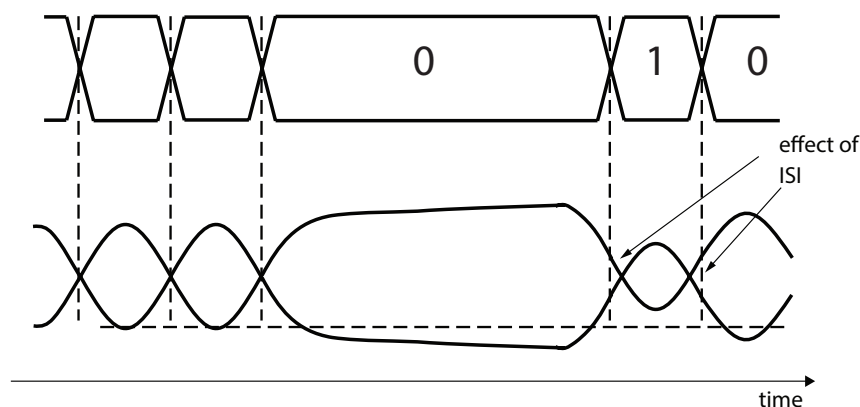


Figura 2.13: Efeito da Interferência Inter-Simbólica numa transmissão, adaptada de [3]

melhor entendimento, a figura 2.13 da página 28 ilustra o fenómeno de interferência inter-simbólica. Considere-se a transmissão de cima da imagem o sinal efetivamente transmitido, e a transmissão de baixo as distorções que o mesmo sofre no canal. Quando há uma longa transmissão de um determinado valor, na imagem ilustrado como 0, e de seguida uma pequena transição do valor oposto então o canal não tem tempo para carregar completamente o valor sofrendo por isso a distorção que se visualiza na imagem. Isto pode vir a causar problema no recetor no que toca à detecção desta transição distorcida.

Torna-se evidente a necessidade de controlar este tipo de interferências, e como tal na interface com o canal físico recorre-se à técnica de pré-ênfase. Segundo [3] esta característica é talvez a mais importante da interface com a camada física e consiste num *overdriving* (dar um ênfase) intencional no início de uma transição e um *underdriving* nos bits seguintes que apresentem o mesmo valor. Este segundo processo também é conhecido por *de-emphasis*. Desta maneira, quando acontece o fenómeno de interferência inter-simbólica, a capacidade do canal recuperar a total amplitude de uma pequena transição não se tornará um problema.

Assim sendo, do lado do transmissor é feito um pré-ênfase do sinal mesmo antes do canal de transmissão. Todavia tal não é certamente suficiente para uma correta reconstituição do sinal e o recetor necessita de ter recorrer a técnicas para a sua correta recuperação.

No lado do recetor é então utilizada a técnica de equalização que vem tentar compensar as distorções que são introduzidas na frequência. Segundo a fonte [3] esta técnica divide-se em dois grandes tipos: equalizador ativos e passivos.

Um equalizador passivo é um circuito passivo que tem uma resposta em frequência complementar as perdas da transmissão, podendo ser equiparado a um filtro. Isto porque efetivamente o que um equalizador passivo faz, segundo a fonte [3], acaba por ser filtrar as frequências que a linha de transmissão não passa e não filtrar as que a mesma passa. Por outro lado, os equalizador ativos, podem ser visto como amplificadores/atenuadores dependentes da frequência. Estes podem ainda ser divididos em dois tipos: equalizadores fixos e adaptativos. Os equalizadores fixos terão sempre a mesma resposta em frequência de acordo com o padrão que lhes é determinado. Segundo o autores de [5] e de [3] um equalizador adaptativo é mais complexo mas também tem mais vantagens comparativamente aos outros tipo de equalizadores.

Um equalizador ativo adaptativo analisa os dados que vão chegando ao recetor e deteta quais são as frequências que estão a ser atenuadas pela linha de transmissão e faz os ajustes necessários de acordo com as medidas que obtém realizadas num *loop* fechado. Essencialmente, para tomar uma decisão tem em conta todos os outros dados chegados anteriormente ao sistema. Este tipo de equalizadores são geralmente usados quando há codificações de linha específicas.

Segundo [3], os equalizadores ativos fixos são melhores para sistemas constantes, como por exemplo ligações *chip-to-chip* ou então ligações cujos cabos têm um comprimento fixo.

Por outro lado, os equalizadores ativos adaptativos são melhores para ligações inconstantes e até mesmo com comprimento de ligação variável.

Sinalização Física

A implementação da camada física dos serializadores e deserializadores de alta velocidade tomam uma forma universal baseada em interfaces eletricamente diferenciais, segundo [3]. Esta forma de transmissão consiste no envio de dois sinais que se complementam eletricamente relativamente a uma massa comum. A principal vantagem que esta metodologia de transmissão acarreta consigo é a uma maior rejeição ao ruído em modo-comum uma vez que o sinal no recetor é obtido através da diferença entre as duas tensões dos sinais diferenciais recebidos. Assim sendo, este tipo de transmissão torna-se vantajosa quando a interferência eletromagnética (EMI) ou a interferência de radio-frequência (RFI) se tornam problemáticas na transmissão, pois caso estas existam a sua interferência será em ambos os sinais (positivo e negativo) e a no recetor a diferença será a mesma, não afetando a recuperação dos dados transmitidos.

Os métodos de implementação de sinalização diferencial podem variar em diversas características, como por exemplo o consumo de energia e tipo de aplicações em que podem ser implementadas. O autor de [3] menciona os três mais comuns:

- **LVDS - *Low-Voltage Differential Signaling***: é uma técnica que opera a uma baixa potência e que atinge velocidades de transmissão até 3,125 Gb/s.
- **LVPELC - *Low-Voltage Pseudo Emitter-Coupled Logic***: é uma técnica que consome entre uma média e alta potência atingindo velocidades de transmissão superiores a 10 Gb/s.
- **CML - *Current Mode Logic***: técnica que consome uma potência de valores médios, atingindo ainda assim velocidades de transmissão superiores a 10 Gb/s.

Os valores de consumo de potência e velocidades de transmissão para cada uma das técnicas apresentadas anteriormente foram retiradas da fonte [20]. Segundo o autor de [3] o método mais adequado para implementação em ligações cujas taxas de ligação rondam os Gb/s deve ser a CML.

Requisitos de sinais de relógio de referência

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. 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. Segundo a fonte [5], é comum que este sinal de relógio seja partilhado entre o recetor e o transmissor, havendo, contudo, a necessidade

de ajuste de fase deste sinal do lado do recetor. Este ajuste de fase deve-se essencialmente por dois motivos: o atraso e ruído do canal. Por um lado, durante a transmissão é introduzido um atraso inerente no sinal que não é conhecido à priori, e por outro durante a transmissão também é introduzido ruído que torna a fase do sinal recebido bastante crítica para o desempenho do transceptor.

Assim sendo, tanto a fonte [3] como a fonte [5] reforçam a ideia de que o sinal de alta velocidade de referência possui requisitos muito particulares para o correto funcionamento do transceptor, uma vez que influencia fortemente o seu desempenho. Entre eles destacam-se o baixo ruído, elevada precisão e uma geração continua do mesmo. Um simples sinal gerado por um oscilador local não é suficiente para cumprir tais critérios, remata o autor de [5]. Desta maneira, é recomendado o uso de osciladores externos, capazes de serem reprogramados para determinadas frequências no sentido de se obter um sinal mais limpo e preciso. Tal será utilizado neste projeto através de um modulo disponível na FPGA VC7203 que será explicado com mais detalhe mais à frente neste relatório.

Importância da criação e uso de pacotes

O autor de [3] menciona a importância de criação de "pacotes" que contribuem para a definição do protocolo de transmissão em série. Antes de se perceber que tipos de tramas constituem um pacote é necessário ter em conta que um "pacote" consiste num conjunto de *bytes* bem definidos que contém num cabeçalho, dados e fim do mesmo.

Em qualquer tipo de transmissão de dados são usados pacotes para os transmitir, e a comunicação em série de alta velocidade não pode fugir a esta realidade ainda por cima porque este tipo de arquiteturas exigem métodos de alinhamento do *stream* de bits chegados em palavras, transmissão de determinados padrões para ser possível a recuperação do relógio entre outros. Assim, é possível aproveitar a necessidade da transmissão destes determinados padrões e criar a partir daqui um pacote de transmissão que sirva para sinalizar as diferentes fases de transmissão dos dados e que ao mesmo tempo permita que o sinal de relógio seja recuperado e os dados alinhados. No fundo é juntar o útil ao útil.

Segundo o autor [3], o pacote de transmissão deve ter bem definidas determinadas tramas, entre as quais:

- **Início de Pacote (*Start of Packet*):** para poder sinalizar a chegada do início de um novo pacote
- **Fim de Pacote (*End of Packet*):** para sinalizar o fim da transmissão de um pacote inteiro
- **Dados :** tramas onde serão transmitidos os dados, pode conter determinados caracteres especiais que indiquem que tipo de dados são transmitidos

- **Símbolo de Espera (*Idle Symbol*):** estas tramas devem ser enviadas sempre que não houver dados para transmitir para garantir que a transmissão contínua dos dados continua alinhada e o sinal de relógio continua a ser recuperado.

Correcção do sinal de relógio

Tal como referido anteriormente, a comunicação de sinais de alta velocidade pode sofrer diversas interferências durante a sua transmissão. Contudo, segundo [5], 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.

Deteção de erros na transmissão

Durante a transmissão em série de alta velocidade, e tal como já foi indicado anteriormente, são introduzidos ruídos e distorções nos sinais e apesar das diversas técnicas já apresentadas anteriormente que evitam estes erros existem códigos detetores e corretores de erros que podem ainda ser aplicados. No caso em específico deste projeto em que se trata da transmissão em série de dados de imagem e som o mais conveniente é a implementação de códigos detetores de erros, pois não traz vantagem ao projeto corrigir esses mesmos erros detetados, por muito curto que seja o tempo que demore tal processo. O autor de [5] menciona mesmo que uma das técnicas mais utilizadas é o CRC (*Cyclic Redundancy Codes*) e que existem já algumas arquiteturas que incluem esta mesma técnica, e como tal esta será brevemente abordada.

O principal objetivo desta técnica é determinar se o sinal que foi transmitido pelo canal com ruído foi corrompido ou não. Segundo [21], a ideia básica por trás deste algoritmo de deteção de erros é tratar a mensagem a ser transmitida como um número binário normal, dividir por um número binário fixo e o resto dessa mesma divisão será o *checksum* da mensagem que será enviado na mesma trama. Do lado do recetor é aplicado o mesmo algoritmo à trama recebida e de seguida verifica-se se o resto da divisão corresponde ao *checksum* recebido. No pior dos casos quando o *checksum* estiver errado então perderá-se a transmissão de um pacote (no caso do projeto será uma imagem) o que ao olho humano à partida nem será detetado.

Ainda segundo a fonte [21], o que torna este algoritmo mais fiável relativamente aos outros é a complexidade de determinação do *checksum*. Se outros algoritmos semelhantes podem receber um *checksum* corrompido e uma mensagem que também esteja corrompida mas de acordo com esse determinado algoritmo o *checksum* está correto para a mensagem recebida, então não é detetado nenhum erro devido à consistência da mensagem. No entanto, como o CRC é um algoritmo mais complexo então tal não acontece tornando-o

num código detetor de erros robusto e que por isso pode ser utilizado neste projeto em concreto.

2.5 Saídas em série de alta velocidade da FPGA VC7203

A FPGA VC7203 disponibiliza de uns transctores de alta velocidade que devido às suas características poderão ser utilizados neste projeto. Esta secção irá abordar quais as características disponíveis e como podem ser utilizadas no projeto.

2.5.1 Localização dos transctores na FPGA

As FPGA de série 7 da XILINX têm disponíveis transctores capazes de comunicação em série de alta velocidade, tal como é necessário neste projeto. Em específico, segundo [8], na FPGA XILINX VC7203 Virtex-7 estão disponíveis transctores GTX que permitem uma velocidade de 12.5 Gb/s e que são os mais adequados para este projeto. Noutros modelos existem outros transctores, 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 transctores GTX, visto que são os mais adequados para este tipo de comunicações.

Na figura 2.14 na página 33 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).

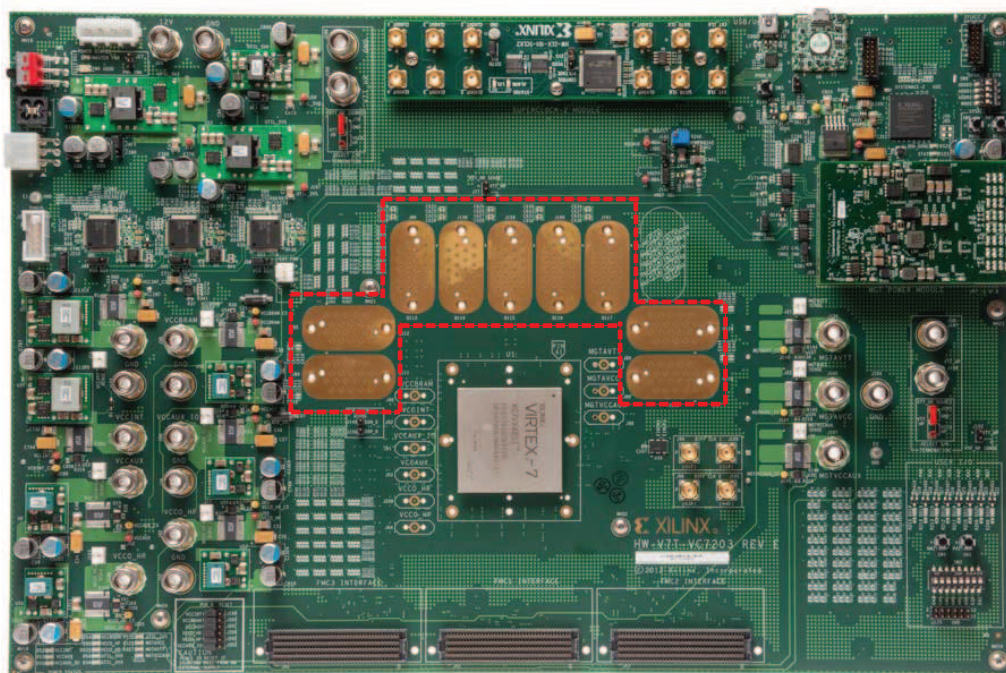


Figura 2.14: Localização física na FPGA dos GTX, retirada de [1]

2.5.2 Arquitetura dos transdutores

Cada GTX é composto por quatro transmissores e quatro receptores. É possível visualizar o conector do GTX na imagem 2.15 na página 34. A imagem A corresponde ao conector e a imagem B faz as correspondência de cada pino do conector. Através da imagem B é possível verificar que efetivamente existem 4 pares diferenciais de transmissores/receptores e ainda 2 pares diferenciais que são usados para conectar os sinais de relógio de referência. Nestes conectores são ligados cabos SMA diferenciais.

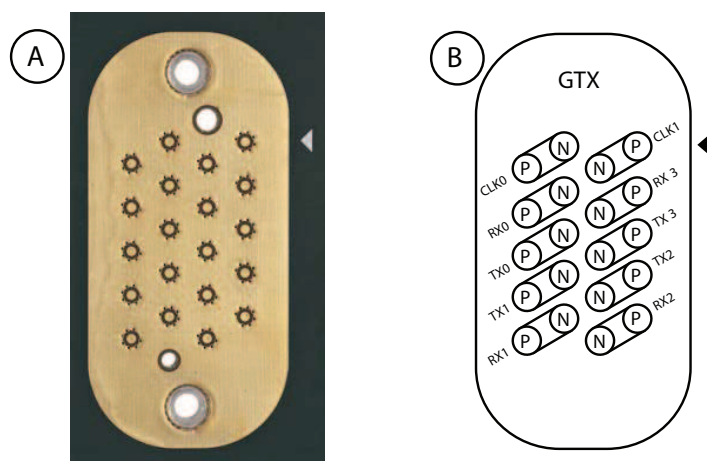


Figura 2.15: Conectores GTX localizados na FPGA, retirada de [1]

Apesar de em cada GTX estarem disponíveis quatro transmissores/receptores, o projeto apenas faz uso de um canal para efetuar a transmissão de dados. Na imagem 2.16 da página 35 é apresentada uma arquitetura geral dos transdutores da FPGA.

A arquitetura dos mesmos passa a ser brevemente descrita, segundo [8] e [4]:

- **Módulo PMA (*Physical Medium Attachment Sublayer*)** que inclui:

- Bloco serie/parelo e vice-versa (PISO e SIPO).
- PLL (*Phase-locked loop*).
- CDR (*Clock Data Recovery*).
- Pré-Ênfase e Bloco de Equalização.
- Suporta uma taxa de débito de saída até 12,5 Gb/s.

- **Módulo PCS (*Physical Coding Sublayer*)** que inclui:

- Lógica de processamento dos dados em paralelo reconfigurável
- *Datapath* de 2 e 4 byte internos para suportar diferentes taxas de débitos
- Codificação e decodificação 8B/10B

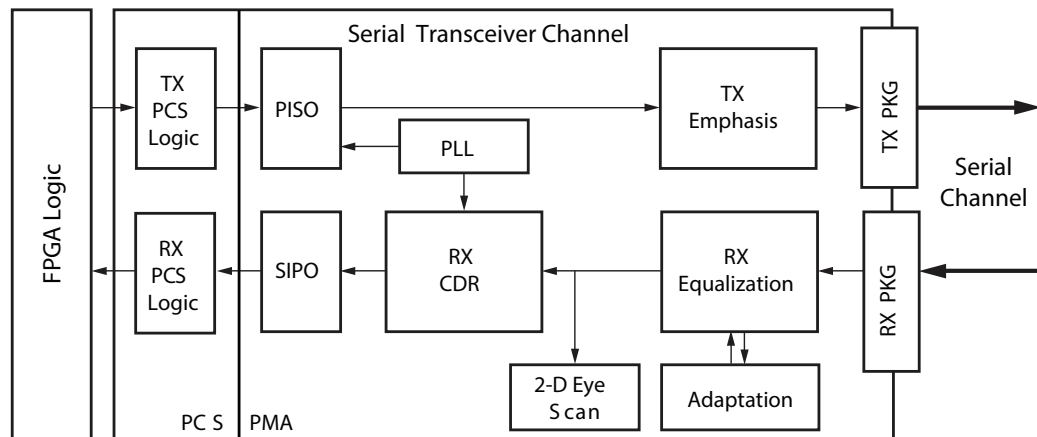


Figura 2.16: Arquitetura geral dos transceptores, retirada de [8]

- Detecção 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
- Este bloco trabalha com taxas de débitos de informação mais baixas.

- **Interface com a lógica da FPGA.**
- **Interface com o canal físico em série diferencial.**

O transmissor e o recetor funcionam separadamente apesar de na imagem 2.16 da página 35 sugerir que funcionam em conjunto. Estes dispõem de blocos necessários referidos em 2.4 e como tal são usados no projeto. Mais detalhes sobre o funcionamento independente do transmissor e recetor serão detalhados quando for abordado o trabalho desenvolvido sobre a comunicação em série.

2.6 Sincronização entre diferentes domínios de relógio

Tal como referido em 1.3 o projeto está dividido em duas partes fundamentais: a primeira em que se trabalha apenas com os dados provenientes do sinal HDMI, e uma segunda parte que conjugará dois domínios de sinais de relógio (do HDMI e do GTX). A imagem 2.17 na página 36 ilustra os diferentes domínios de relógio existentes no projeto.

A placa HDMI RX e a arquitetura de receção de dados HDMI funcionam a um sinal de relógio proveniente da placa HDMI. Todavia a arquitetura responsável pelo envio dos dados para o GTX e posterior receção dos dados provenientes do mesmo funciona a um sinal de relógio proveniente do GTX. Também a arquitetura responsável pelo envio dos dados para a placa HDMI funciona a esse mesmo sinal de relógio proveniente do GTX

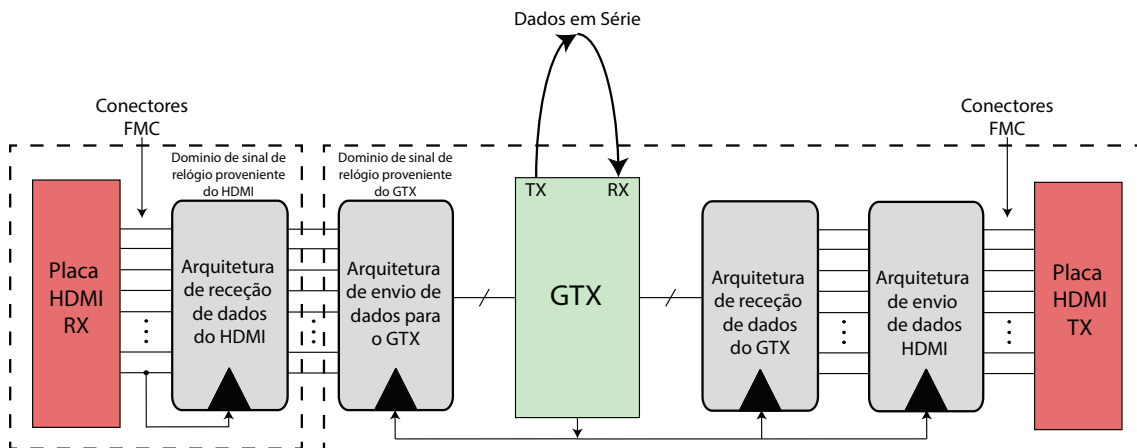


Figura 2.17: Representação dos diferentes domínios de sinais de relógio no projeto

(tal será explicado quando for abordado o trabalho referente à segunda parte do trabalho). Assim sendo, existem aqui dois principais domínios de relógio diferentes o que pode vir a provocar falta de sincronização entre a transição de dados de um para outro.

Ora, segundo a fonte [22] e [9], este problema pode levar a captação de dados que se encontram num estado de metaestabilidade e estes propagam-se ao longo do sistema o que certamente causará danos irreversíveis na transmissão de dados.

Para se perceber o problema de meta-estabilidade é necessário ter em consideração que os dispositivos digitais têm determinados requisitos temporais que necessitam de ser cumpridos para que possam captar corretamente os dados. Esses requisitos passam de seguida a ser enumerados tal como em [9]:

- **Tempo de *Set Up* (T_{su}):** Tempo mínimo em que a entrada do registo deve estar estável antes da subida positiva do sinal de relógio.
- **Tempo de *Hold* (T_H):** Tempo mínimo em que a entrada do registo deve estar estável depois da subida positiva do sinal de relógio.
- **Tempo de *clock-to-output* (T_{co}):** Tempo necessário para o valor estar disponível na saída do registo.

Quando alguma transição viola o T_{su} ou T_H então a saída não conseguirá obter um valor e ficará num estado de metaestabilidade, que nem é 0 nem 1. Estes problemas normalmente acontecem em situações em que os dados são transmitidos entre domínios de relógio diferentes, tal como acontece neste projeto, e podem propagar-se no sistema, tal como já foi referido. A figura 2.18 na página 37 ilustra este problema.

Visto que é possível prever antecipadamente que este tipo de problemas ocorrerão durante o desenvolvimento do projeto, é então necessário levá-los em conta e procurar métodos que os possam resolver.

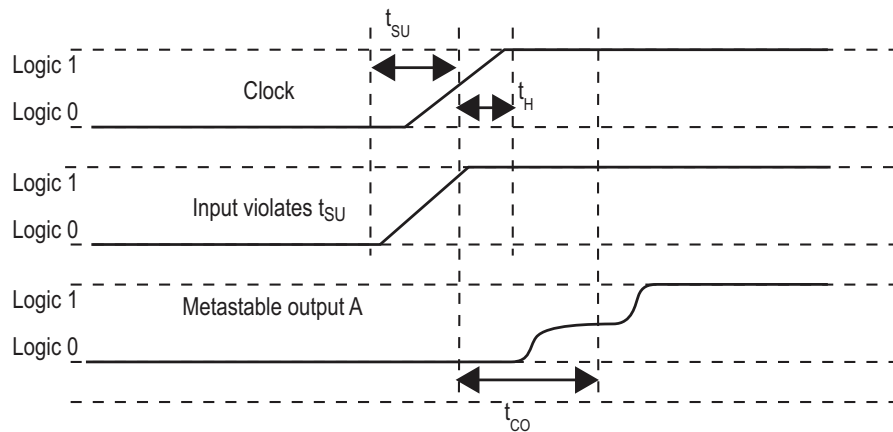


Figura 2.18: Exemplo de metaestabilidade, adaptado de [9]

Segundo [9] e [22], para a correção de problemas de metaestabilidade são tipicamente usadas cadeias de registos sincronização no domínio de relógio de destino para que o sinal seja re-sincronizado nesse mesmo domínio. Isto permitirá que os registos tenham mais tempo para resolver eventuais estados de meta-estabilidade para um valor estável antes do valor ser usado no domínio de destino. O tempo disponível entre os caminhos de registo para registo é o tempo disponível para o sinal metaestável estabilizar. A imagem 2.19 da página 37 ilustra uma cadeia de registo de sincronização.

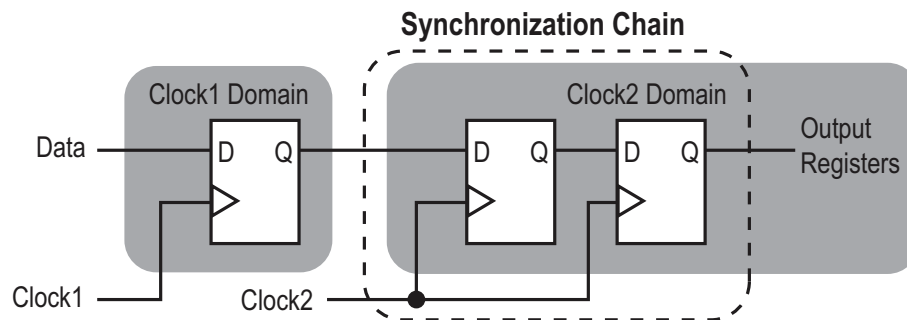


Figura 2.19: Exemplo de metaestabilidade, retirado de [9]

Segundo [9], a cadeia de registos de sincronização funciona da seguinte maneira: todos os registos devem trabalhar ao mesmo sinal de relógio (do domínio de destino) e os registos devem enviar os dados apenas para um único registo na cadeia, excepto o último. Para além disso o primeiro registo da cadeia tem como entrada um sinal proveniente de um domínio de sinal de relógio assíncrono com o de destino.

Assim sendo, a utilização desta técnica virá prevenir eventuais problemas de sincronização que possam vir a existir entre diferentes domínios de relógio existentes no projeto.

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 3.1 e 3.2 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*) XCF16PFG48C 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 [23], 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.

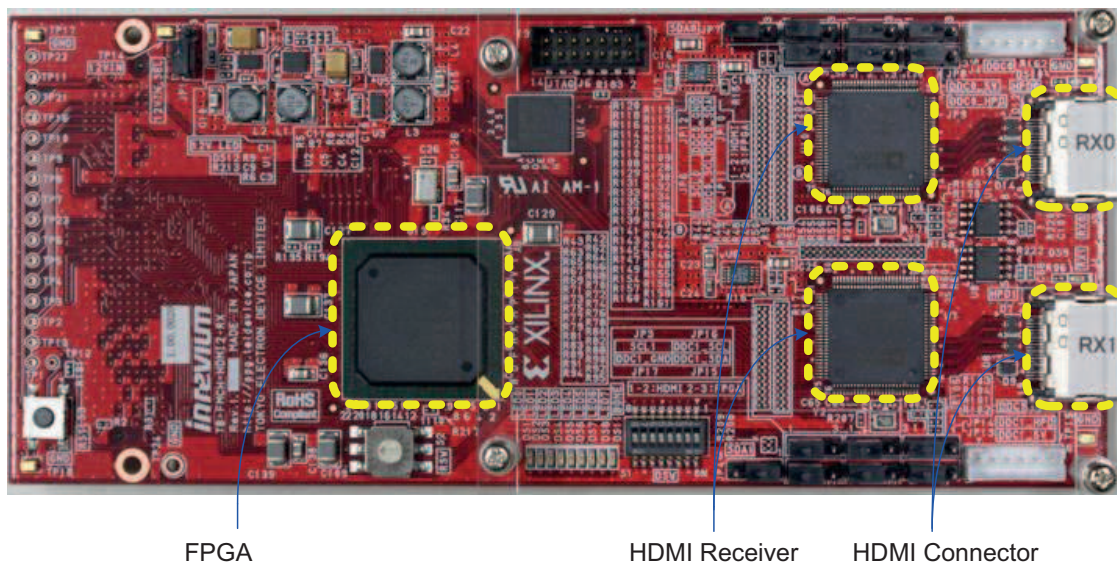


Figura 3.1: TB-FMCH-HDMI2 RX, retirada de [2]

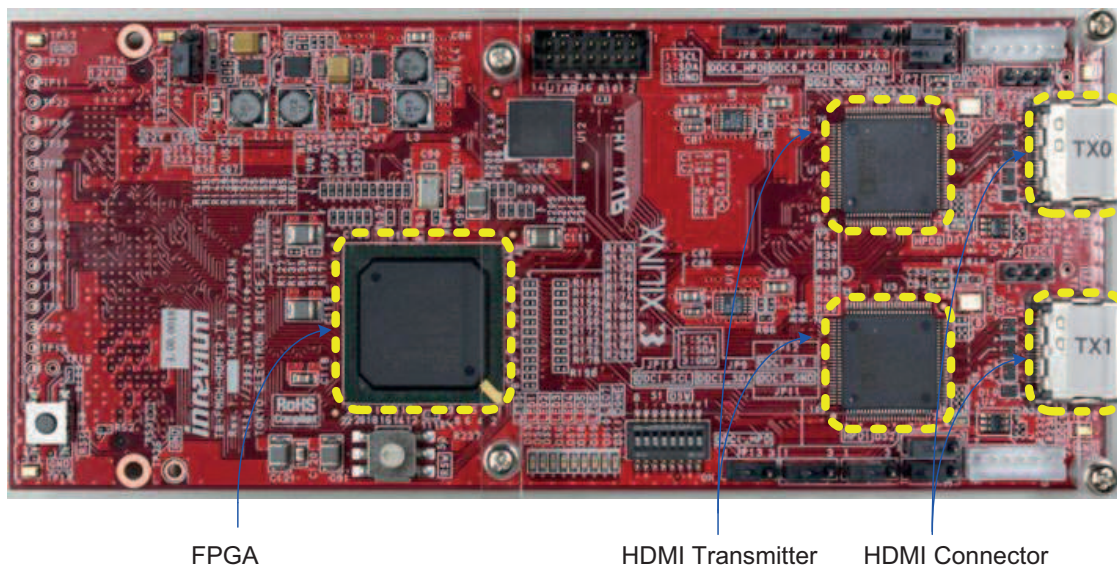


Figura 3.2: TB-FMCH-HDMI2 TX, retirada de [2]

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 [24] 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 tabelas 2.1 e 2.2 nas páginas 14 e 17 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 41, 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.

PIN	FPGA -> FMC (RX)	FMC -> (TX)	Descrição
CLK0_M2C_P	RX#0_LLC	TX#0_DCLK	Sinal de relógio dos pixels
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*

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 uma 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.

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 áudio em formato I^2S . O mesmo acontece na placa transmissora mas para ser capaz de receber estas configurações.

Assim como referido em [11], 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 43 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 [11] 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 [25], 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 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 [25].
3. **Serial Data:** Sinais que transportam os dados de áudio.

PIN	FPGA ->FMC (RX)	FMC -> FPGA (TX)	FPGA->HDMI_TX
CLK0_M2C_P	RX#0_LLC	TX#0_DCLK	Sinal de relógio dos pixels
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	<i>Master Clock</i> de som
LA20_N	RX#0_SCLK	TX#0_SCLK	<i>Serial Clock</i> 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

Na imagem 3.3 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.

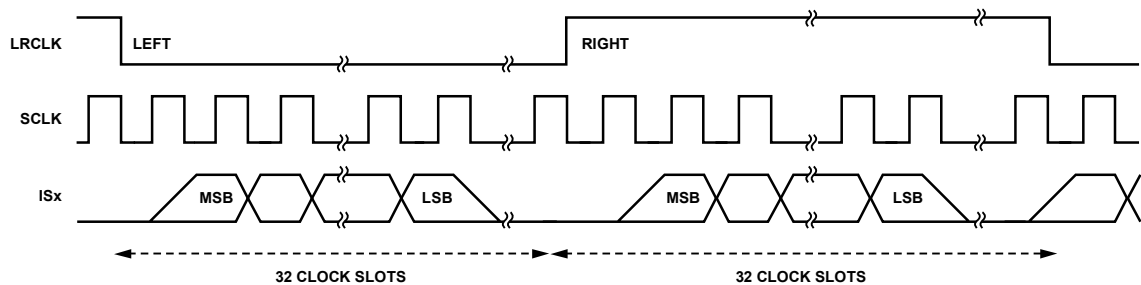


Figura 3.3: Ilustração dos sinais de som transmitidos no formato I^2S , retirada de [10]

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 [10] 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 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 [11].

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 [12] 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 capítulo 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 configuradas 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 receptora 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 45.

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 possível 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.

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]

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	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 [11]

À semelhança da placa recetora para esta configuração, também é possível configurar o ADV7511 para se obter na sua saída diversos formatos de imagem. A [tabela 3.6](#) apresentada na [página 46](#) indica essas mesmas combinações.

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 [11]

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 47 são apresentadas as configurações dos interruptores que configuram o ADV7612 de forma a enviar diferentes formatos.

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 [12]

Relativamente à placa HDMI transmissora, ambos os canais são capazes de suportar imagens em formato RGB ou YCbCr. A tabela 3.8 da página 47 apresenta as combinações dos interruptores para se poder obter os diversos formatos na saída do ADV7511.

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 [12]

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 inicial do projeto, optou-se por simplificar a transmissão e para tal utilizou-se apenas a placa transmissora HDMI configurada por defeito. Construiu-se em Verilog um bloco capaz de gerar uma imagem para ser transmitida, mais especificamente uma barra de cores, e utilizou-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 específico, este baseia-se essencialmente numa máquina 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 valores de pixels ativos.

Para que se entenda mais facilmente como e quando se transmitem os sinais de controlo da imagem e também os valores dos pixels é demonstrado na imagem 3.4 na página 49 um 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 pixels 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 pixels que não contém 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 pixels que não contém 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 pixels a ser geradas.
6. **VSXW:** *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 aos pixels antes de começarem a ser transmitidas as linhas de pixels.
8. **VFP:** *Vertical Front Porch* é o parâmetro que define o número de linhas horizontais que não contém informação útil relativamente aos pixels depois de terem sido transmitidas todas as linhas horizontais da imagem.

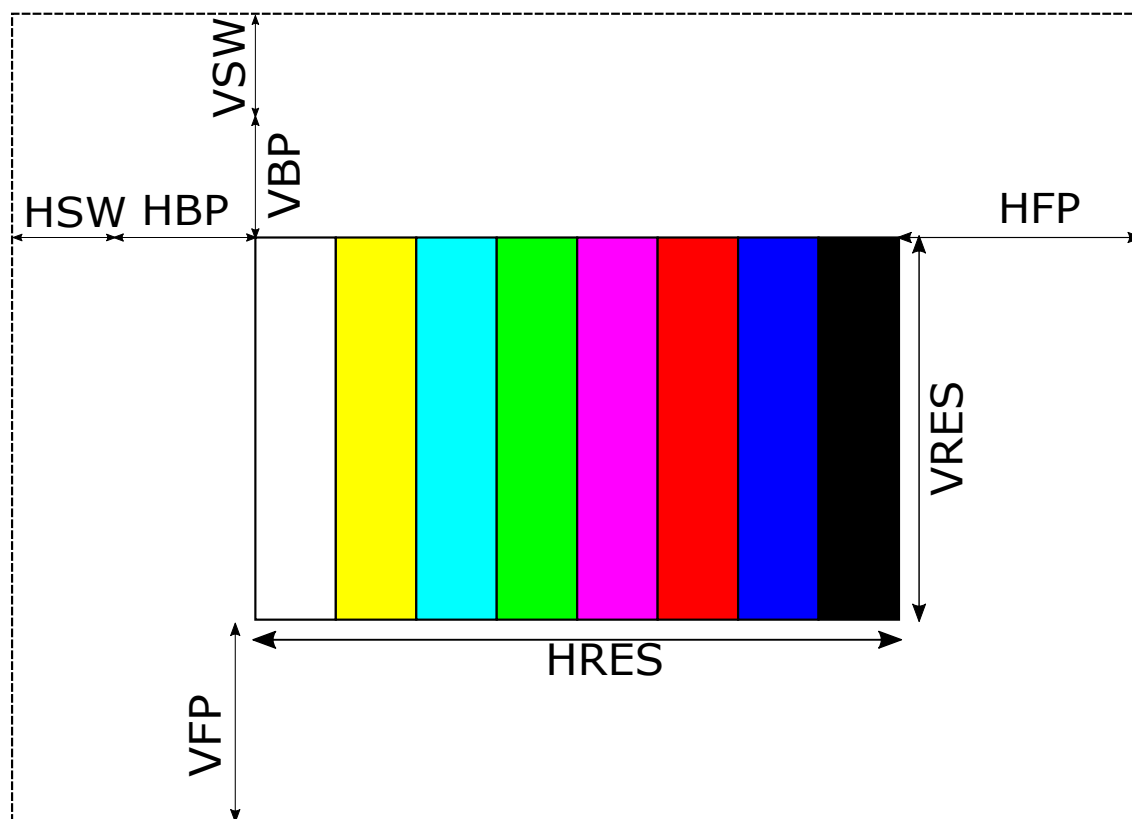


Figura 3.4: Exemplo de imagem gerada pelo módulo desenvolvido

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

A figura 3.5 na página 50 ilustra a máquina 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 contando 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 específico obtêm-se os seguintes valores:

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

Para além destes sinais de decisão para mudança de estado existem mais dois sinais no diagrama da máquina de estados presente na figura 3.5 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

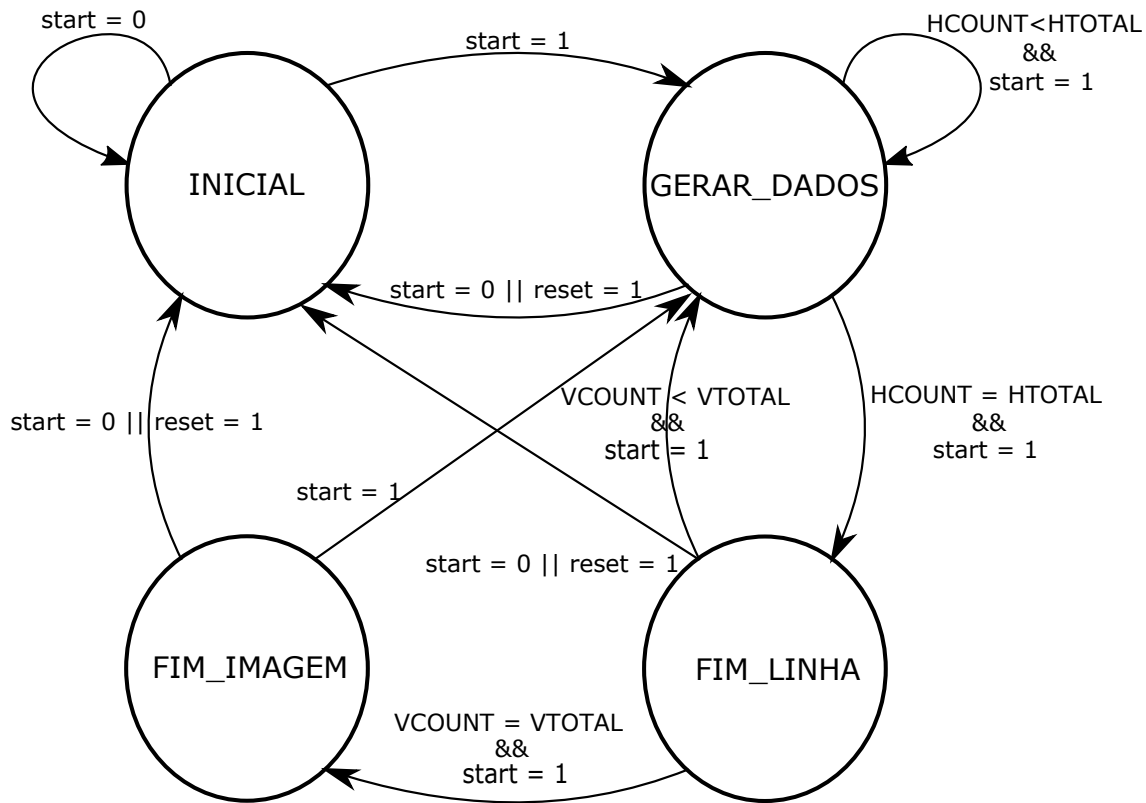


Figura 3.5: Máquina de estados para gerar uma barra de cores

então quando se pretende restabelecer os dados originais da máquina de estados (através do botão *reset*).

Existem 4 estados nesta máquina e 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 início 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 possível 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 VCOUNT. 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 início 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 início 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 pixels 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 3.4, e as duas ultimas condições garantem o mesmo mas na zona horizontal.

- **Valor dos pixels:** 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.

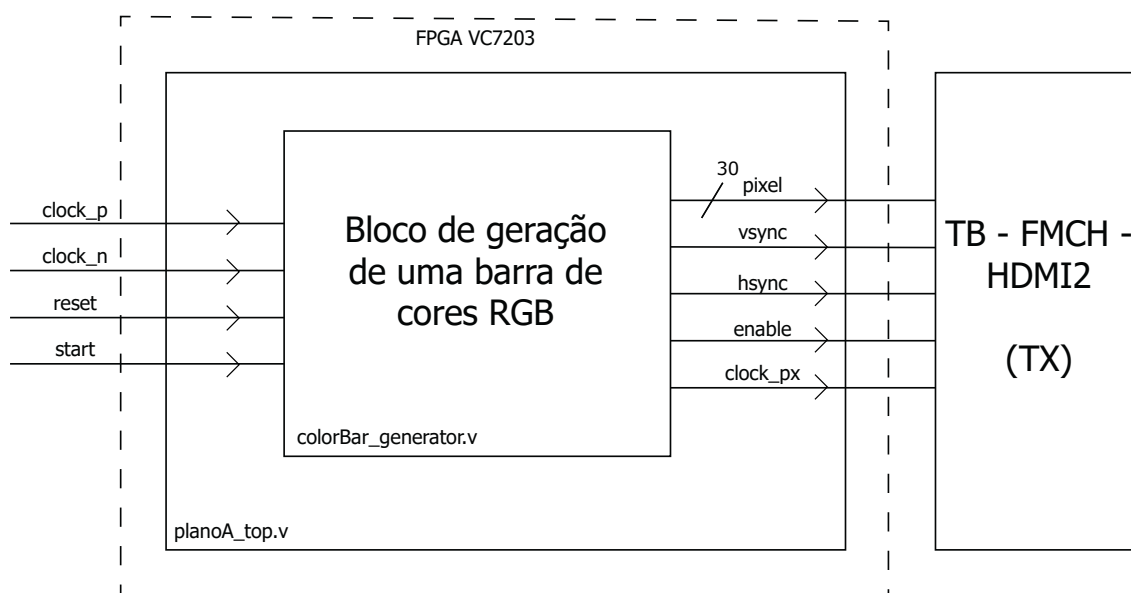


Figura 3.6: Diagrama de blocos de arquitetura implementada utilizando um bloco gerador de barra de cores

Na figura 3.6 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 máquina 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 um módulo 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 à saída do módulo é possível visualizar na imagem 3.6 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 módulo de topo, no caso desta arquitetura do módulo "planoA_top.v", estejam atribuídas a portas físicas da FPGA. Para tal é necessário definir onde estão as localizações das portas na FPGA (LOC) e criar um ficheiro de definições dessas mesmas restrições físicas. A tabela 3.9 na página 53 indica quais as localizações físicas de cada porta existente no módulo 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 possível 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	cpll_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 físicas gerado após a atribuição das mesmas é apresentado no sub-capítulo C.2 do anexo C. Para cada porta são atribuídas duas restrições: uma que indica a localização física 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 físicas geradas, são também geradas duas restrições temporais quanto aos sinais de relógio à entrada apresentadas no sub-capítulo C.2.0.1 do anexo C. As restrições temporais existentes definem que nas portas de entrada do sinal de relógio diferencial existe um sinal com uma frequência de 200 MHz (período 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 código em verilog, 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-capítulo 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 específico 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 3.7 da página 54 ilustra o diagrama de blocos da arquitetura desenvolvida.

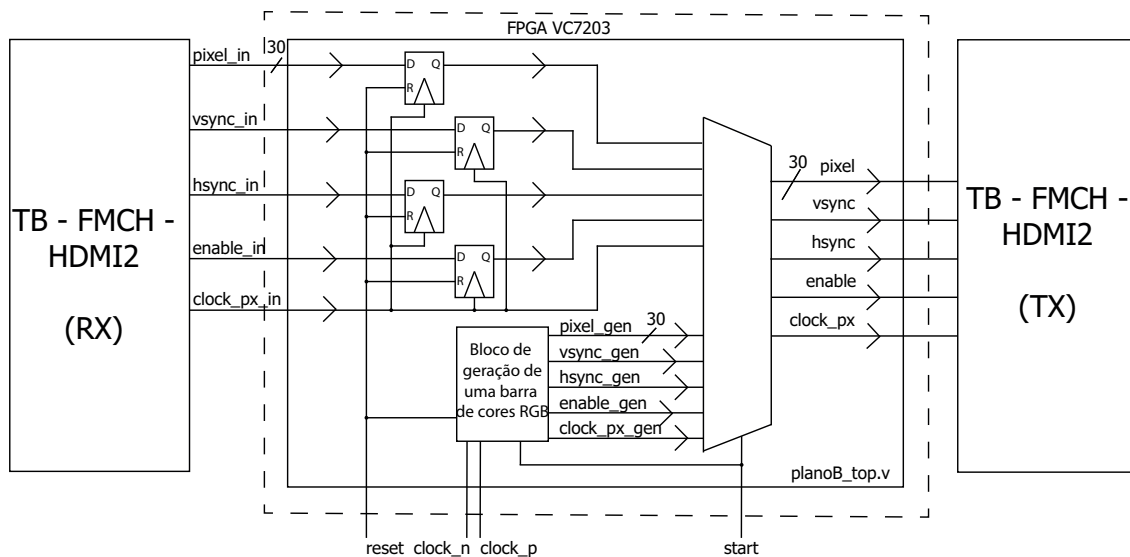


Figura 3.7: Diagram de blocos da arquitetura desenvolvida para transmitir imagem entre dispositivos HDMI

É possível visualizar que nesta arquitetura, tal como já acontecia na anterior, existem na sua saída os sinais que são enviados para a placa HDMI transmissora 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 constituído pelo par positivo *clock_p* e pelo par negativo *clock_n*, e ainda o sinal *start* que define o início da transmissão da barra de cores em vez dos sinais provenientes 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 visível na figura 3.7 são os sinais provenientes do módulo 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 uma ligação entre as placas HDMI recetora e transmissora pois os sinais seleccionados pelo multiplexador são os sinais provenientes da entrada do módulo. 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 55 especifica as localizações físicas da FPGA que foram atribuídas a cada porta do módulo 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 possível encontrar mais detalhadamente 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
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.2.1.1 do anexo C. Relativamente a estas restrições físicas, existem para cada porta do modulo duas restrições: uma que define a localização física e outra a norma da física, tal como já foi mencionado anteriormente.

Quanto às restrições temporais aplicadas ao sistema são idênticas às mesmas aplicadas na arquitetura desenvolvida anteriormente e estão presentes em C.2.1.2 no anexo C.

Após síntese 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 transmitir 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 I²S. As características desta configuração são apresentadas na secção 3.1.1.2 na página 42 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-capítulo 3.1.2.2). Neste caso em específico serão utilizados 12 bits por cor o que preferá um total de 36 bits por pixel.

Na imagem 3.8 na página 57 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 possível 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 de pixel ativos (*enable*). Para além destes sinais provenientes da placa HDMI receptora, são recebidos os sinais referentes ao som: o sinal de relógio dos dados em série (*sclk*), o sinal referente à selecção do canal de audio esquerdo ou direito (*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

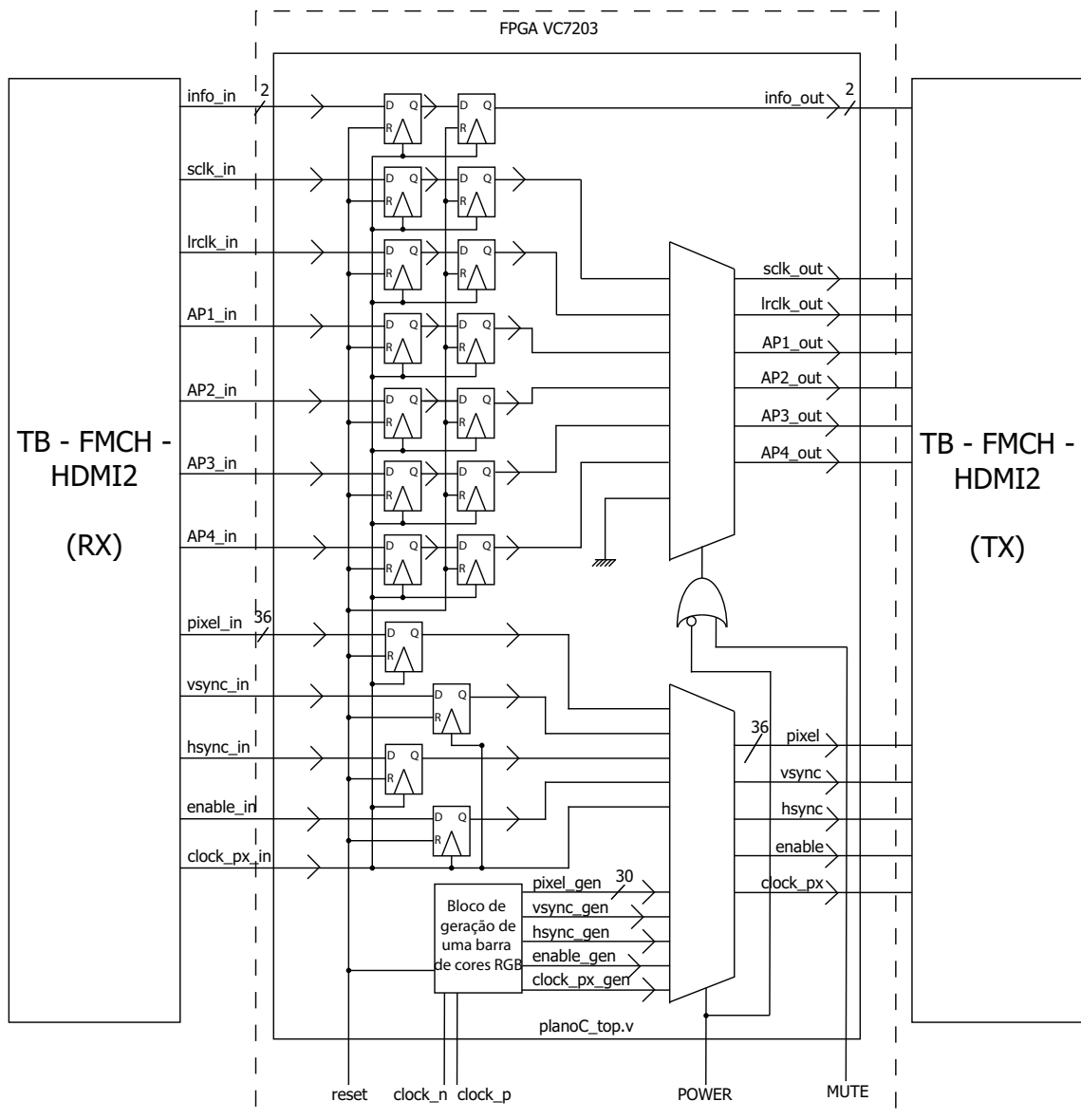


Figura 3.8: Diagrama de blocos da arquitetura desenvolvida para transmitir imagem e som entre dispositivos HDMI

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 placas HDMI existem ainda mais portas do bloco. Existe o típico sinal de relógio diferença 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 módulo "*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-capítulo 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 [22], 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 físicas de cada porta de entrada e saída do modulo principal, e como tal é possível encontrar descritas essas mesmas atribuições na tabela 3.11 na página 59. 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/O	Sinal	LOC na FPGA	Banco na FPGA
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 físicas 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 físicas pode ser encontrado em [C.3.0.1](#) 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.3.1](#) 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 pixels, o cruzamento estes dois

domínios de relógio diferentes não apresentou qualquer tipo de problemas uma vez que ao ouvido humano os pequenos atrasados que possam eventualmente acontecer não são perceptíveis.

Capítulo 4

Transmissão dos dados em série

Em [26] é sugerido que o valor do sinal de relógio de referência seja o mesmo proveniente da fonte HDMI isto porque a frequência do sinal de relógio de referência deve ser exatamente igual (ou múltiplo) da cadência dos dados de entrada.

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	<i>clock</i> dos pixels
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 pixels
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[11]
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 (2D)
LA01_N_CC	RX#0_InputVideoStatus[1]	TX#0_InputVideoStatus[1]	Formato do video (2D)
LA19_N	RX#0_MCLK	TX#0_MCLK	<i>Master Clock</i> de s
LA20_N	RX#0_SCLK	TX#0_SCLK	<i>Serial Clock</i> de sc
LA21_N	RX#0_AP0	TX#0_AP0	Dados de Som SPD
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#0_LLCC	TX#0_DCLK	Sinal de relógio dos pixels
CLK1_M2C_N	RX#1_LLCC	TX#1_DCLK	Sinal de relógio dos pixels

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 transmissão 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#0_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 arquitetura 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 arquitetura de transmissão de imagem e som entre as placas HDMI transmissora e recetora

Anexo C

Ficheiros de restrições

C.1 Transmissão de uma imagem gerada na FPGA

C.2 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.2.0.1 Restrições Temporais

```

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

```

C.2.1 Transmissão de imagem entre dispositivos HDMI

C.2.1.1 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.2.1.2 Restrições Temporais

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

C.3 Transmissão de imagem e som entre dispositivos HDMI**C.3.0.1 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.3.1 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*. 2014.
- [3] A. Athavale and C. Christensen, *High-Speed Serial I/O Made Simple*. 2005.
- [4] Xilinx and Inc, *7 Series FPGAs GTX/GTH Transceivers User Guide (UG476)*. 2016.
- [5] D. Chen, “SerDes Transceivers for High-speed Serial Communications,”
- [6] E. Tutorials, “The Shift Register.”
- [7] P. Moreira, J. Christiansen, A. Marchioro, E. V. D. Bij, K. Kloukinas, M. Campbell, G. Cervelli, and C.-e. Mic, “A 1 . 25Gbit / s Serializer for LHC Data and Trigger Optical Links,”
- [8] Xilinx and Inc, “Xilinx WP431 Leveraging 7 Series FPGA Transceivers for High-Speed Serial I/O Connectivity, White Paper,” 2013.
- [9] Altera, “Understanding Metastability in FPGAs,” *White Paper*, no. July, pp. 1–6, 2009.
- [10] A. Devices, *ADV7612 Reference Manual*.
- [11] Inrevium, *TB-FMCH-HDMI2 Hardware User Manual 1 IN / OUT + Audio support*. 2014.
- [12] Inrevium, *TB-FMCH-HDMI2 Hardware User Manual 2 Ch IN / OUT support*. 2015.
- [13] Wikipedia Contributors, “HDMI,” 2016.
- [14] S. Koenig, D. Lopez-Diaz, J. Antes, F. Boes, R. Henneberger, A. Leuther, A. Tesmann, 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.

- [15] J. Federici and L. Moeller, "Review of terahertz and subterahertz wireless communications," *Journal of Applied Physics*, vol. 107, no. 11, 2010.
- [16] W. contributors, "Audio and video interfaces and connectors," 2016.
- [17] Xilinx, "I/O Design Flexibility with the FPGA Mezzanine Card (FMC)," vol. 315, 2009.
- [18] Analog Devices, *ADV7511 Programming Guide*. No. March, 2012.
- [19] Analog Devices, *ADV7511 Hardware user's guide*. No. July, 2011.
- [20] T. Instruments, *LVDS Owner ' s Manual*. 2008.
- [21] R. Williams, "A Painless Guide To CRC Error Detection Algorithms," *Rocksoft Pty Ltd.*, p. 34, 1993.
- [22] C. E. Cummings, "Clock Domain Crossing (CDC) Design & Verification Techniques Using SystemVerilog," *Snug-2008*, 2008.
- [23] Xilinx, "Platform Flash In-System Programmable Configuration PROMs," 2016.
- [24] Xilinx, "Xilinx Platform Cable USB II," 2015.
- [25] Philips Semiconductors, "I 2 S bus specification," 1996.
- [26] G. Magnaye and J. Snow, "Implementing SMPTE SDI Interfaces with 7 Series Transceivers," vol. 1092, 2013.