



# 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. Doutor Henrique Manuel de Castro Faria Salgado

Supervisor Externo: Doutor Luís Manuel de Sousa Pessoa

22 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, 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 manufatura 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, 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*. 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.

Este projeto é dividido em duas partes para poder cumprir os requisitos. Inicialmente são desenvolvidas arquiteturas que permitem a comunicação entre dois dispositivos HDMI recorrendo-se a duas placas HDMI e uma FPGA de série 7 para realizar tal transmissão. As arquiteturas desta primeira parte do projeto variam entre a transmissão de imagens em diferentes formatos e também de imagem e som. Numa segunda fase do projeto desenvolveu-se uma arquitetura capaz de serializar dados e enviá-los pelas saídas de alta velocidade disponíveis na FPGA que foi utilizada. Esses mesmos dados foram recebidos na FPGA e convertidos novamente para o formato em paralelo. Por fim o projeto junta as duas partes e transmite dados HDMI em série através das entradas e saídas de alta velocidade e recupera o sinal.



# Abstract

Now I have to write it in English



# Agradecimentos

Marisa Oliveira





*'The journey is the reward'*

Steve Jobs



# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>1</b>
1.1	Enquadramento Geral . . . . .	1
1.2	Motivação . . . . .	2
1.3	Descrição do Problema e Objetivos . . . . .	4
1.4	Estrutura da Dissertação . . . . .	6
<b>2</b>	<b>Revisão Bibliográfica</b>	<b>9</b>
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
<b>3</b>	<b>Transmissão de dados HDMI</b>	<b>39</b>
3.1	Infraestrutura do <i>Hardware</i> utilizado . . . . .	39
3.1.1	Configurações da FPGA . . . . .	40
3.1.2	Configuração dos interruptores . . . . .	44
3.2	Arquiteturas Desenvolvidas . . . . .	44
3.2.1	Transmissão de uma imagem gerada na FPGA . . . . .	44
3.2.2	Transmissão de imagem entre dispositivos HDMI . . . . .	51
3.2.3	Transmissão de imagem e som entre dispositivos HDMI . . . . .	53
<b>4</b>	<b>Transmissão dos dados em série</b>	<b>59</b>
4.1	Transmissor . . . . .	59
4.1.1	Interface com a FPGA . . . . .	60

4.1.2	Codificador 8B/10B . . . . .	62
4.1.3	Interface entre os diferentes domínios de sinal de relógio do transmissor . . . . .	63
4.1.4	Interface com a camada física . . . . .	64
4.2	Recetor . . . . .	64
4.2.1	Interface com a camada física . . . . .	64
4.2.2	Equalização . . . . .	65
4.2.3	Alinhamento de Palavras . . . . .	66
4.2.4	Descodificador 8B/10B . . . . .	67
4.2.5	Interfaces entre os diferentes domínios de sinal relógio do recetor . . . . .	67
4.2.6	Interface com a FPGA . . . . .	68
4.3	Análise das características de transmissão . . . . .	69
4.4	Estrutura do transcetor GTX . . . . .	70
<b>5</b>	<b>Transmissão em série de dados HDMI</b>	<b>73</b>
5.1	Abordagem inicial . . . . .	73
5.1.1	Transmissão de uma barra de cores gerada na FPGA em série . . . . .	73
<b>6</b>	<b>Conclusões e Trabalho Futuro</b>	<b>87</b>
6.1	Qualidade dos resultados obtidos . . . . .	87
6.2	Principais dificuldades encontradas . . . . .	87
6.3	Conclusões do projeto desenvolvido . . . . .	87
6.4	Trabalho futuro . . . . .	87
<b>A</b>	<b>Descrição das portas das placas HDMI</b>	<b>91</b>
A.1	Configuração por omissão . . . . .	91
A.2	Suporte de um canal de imagem e áudio . . . . .	91
A.3	Suporte de dois canais de imagem melhorado . . . . .	91
<b>B</b>	<b>Configurações dos interruptores das placas HDMI</b>	<b>93</b>
B.1	Configuração por omissão . . . . .	93
B.2	Suporte de um canal de imagem e áudio . . . . .	94
B.3	Suporte de dois canais de imagem melhorado . . . . .	95
<b>C</b>	<b>Localizações das portas conectadas às placas HDMI</b>	<b>97</b>
C.1	Transmissão de uma imagem gerada na FPGA . . . . .	97
C.2	Transmissão de uma imagem entre dispositivos HDMI . . . . .	98
C.3	Transmissão de imagem e som entre dispositivos HDMI . . . . .	100
<b>D</b>	<b>Ficheiros de restrições</b>	<b>103</b>
D.1	Restrições Físicas relativas às portas de saída das arquiteturas que se conectam à placa HDMI transmissora . . . . .	103
D.2	Restrições Físicas relativas às portas de entrada das arquiteturas que se conectam à placa HDMI recetora . . . . .	104
D.3	Restrições Físicas relativas às portas de entrada e saída exclusivas da arquitetura que transmite uma imagem gerada na FPGA . . . . .	105
D.4	Restrições Temporais relativas à arquitetura que transmite uma imagem gerada na FPGA . . . . .	105
D.5	Restrições Físicas relativas às portas de entrada e saída exclusivas da arquitetura que transmite imagem entre dispositivos HDMI . . . . .	105

D.6	Restrições Temporais relativas à arquitetura que transmite imagem entre dispositivos HDMI . . . . .	106
D.7	Restrições Físicas relativas às portas de entrada e saída exclusivas da arquitetura que transmite imagem e som entre dispositivos HDMI . . . . .	106
D.8	Restrições Temporais relativas à arquitetura que transmite imagem e som entre dispositivos HDMI . . . . .	107



# 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 . . . . .	12
2.2	Diagrama de blocos de TB-FMCH-HDMI2 RX . . . . .	13
2.3	Amostragem dos dados provenientes da FPGA no recetor . . . . .	15
2.4	Diagrama de blocos de TB-FMCH-HDMI2 TX . . . . .	16
2.5	Amostragem dos dados provenientes do FMC no recetor . . . . .	16
2.6	Arquitetura simples de um serializador e deserializador . . . . .	19
2.7	Ilustração do alinhamento em série quando encontrada a <i>comma</i> . . . . .	24
2.8	Ilustração do alinhamento em paralelo quando encontrada a <i>comma</i> . . . . .	24
2.9	Arquiteturas de PISO/SIPO . . . . .	25
2.10	Arquitetura de um <i>shift-register</i> serializador de 4 bits . . . . .	26
2.11	Arquitetura de um <i>shift-register</i> deserializador de 4 bits . . . . .	26
2.12	Exemplo de um serializador de 10 bits . . . . .	27
2.13	Efeito da Interferência Inter-Simbólica numa transmissão . . . . .	28
2.14	Localização física na FPGA dos GTX . . . . .	33
2.15	Conectores GTX localizados na FPGA . . . . .	34
2.16	Arquitetura geral dos transcetores . . . . .	35
2.17	Representação dos diferentes domínios de sinais de relógio no projeto . . . . .	36
2.18	Exemplo de metaestabilidade . . . . .	37
2.19	Exemplo de cadeia de registos de sincronização . . . . .	37
3.1	Placa HDMI recetora TB-FMCH-HDMI2 RX . . . . .	40
3.2	Placa HDMI transmissora TB-FMCH-HDMI2 TX . . . . .	40
3.3	Ilustração dos sinais de som transmitidos no formato $I^2S$ . . . . .	43
3.4	Exemplo de imagem gerada pelo módulo desenvolvido . . . . .	46
3.5	Máquina de estados para gerar uma barra de cores . . . . .	46
3.6	Diagrama de blocos de arquitetura implementada utilizando um bloco gerador de barra de cores . . . . .	49
3.7	<i>Set-Up</i> de teste da arquitetura desenvolvida para transmissão de uma imagem gerada na FPGA para a placa HDMI transmissora . . . . .	51
3.8	Diagrama de blocos da arquitetura desenvolvida para transmitir imagem entre dispositivos HDMI . . . . .	52
3.9	<i>Set-up</i> de teste para as arquiteturas transmissoras de dados entre dispositivos HDMI . . . . .	53

3.10	Diagrama de blocos da arquitetura desenvolvida para transmitir imagem e som entre dispositivos HDMI . . . . .	55
4.1	Arquitetura do transmissor GTX . . . . .	60
4.2	Exemplo simplificado de transmissão com diferentes usos de <i>datapath</i> . . . .	62
4.3	Diferentes domínios de sinal de relógio do transmissor . . . . .	63
4.4	Arquitetura do recetor GTX . . . . .	65
4.5	Exemplo de um alinhamento manual da palavra . . . . .	67
4.6	Diferentes domínios de sinal de relógio no recetor . . . . .	68
4.7	Estrutura geral do módulo GTX gerado no <i>software</i> VIVADO . . . . .	71
5.1	Diagrama de blocos geral da arquitetura que transmite uma barra de cores em série . . . . .	74
5.2	Bloco de sincronização de dados . . . . .	77
5.3	Estrutura das tramas geradas . . . . .	78
5.4	Momentos de transmissão de diferentes tramas . . . . .	79
5.5	Alinhamento das tramas no recetor . . . . .	80
5.6	Máquina de estados de transmissão . . . . .	81
5.7	Máquina de estados leitura de dados . . . . .	82
5.8	Máquina de estados de verificação do alinhamento dos dados . . . . .	84
5.9	Diagrama de blocos de envio de dados para a placa HDMI transmissora . .	85
5.10	Diagrama de blocos da arquitetura de transmissão em série de uma barra de cores gerada na FPGA . . . . .	86
5.11	<i>Setup</i> de teste . . . . .	86



# Lista de Tabelas

2.1	Nomes dos pins da interface FMC de TB-FMCH-HDMI2 RX . . . . .	14
2.2	Nomes dos pins da interface FMC de TB-FMCH-HDMI2 TX . . . . .	17
2.3	Exemplo de codificação 8B/10B . . . . .	22
2.4	Exemplo de palavras de 8 bits codificadas em 8B/10B . . . . .	22
2.5	Caracteres de controlo específicos da codificação 8B/10B . . . . .	23
3.1	Descrição e localização dos pinos de TB-FMCH-HDMI2 configurada por omissão . . . . .	41
3.2	Descrição e localização dos pinos de TB-FMCH-HDMI2 configurada para um canal de imagem e áudio . . . . .	42
3.3	Localização das portas de entrada e saída da arquitetura de transmissão de imagem gerada na FPGA para a placa HDMI transmissora . . . . .	50
3.4	Localização das entradas e saídas das portas da arquitetura de transmissão de uma imagem entre dispositivos HDMI . . . . .	53
4.1	Tamanhos possíveis para a interface da FPGA com o GTX transmissor . . .	60
4.2	Relação entre as frequências dos sinais de relógio <i>TXUSRCLK2</i> e <i>TXUSRCLK</i> . . .	61
4.3	Débitos de transmissão atingíveis para diferentes larguras de porta de entrada do transceptor . . . . .	70
5.1	Sumário do módulo GTX gerado para a transmissão de uma barra de cores gerada na FPGA . . . . .	76
B.1	Configuração dos interruptores da placa HDMI RX configurada de fábrica .	93
B.2	Configuração dos interruptores da placa HDMI TX configurada de fábrica .	94
B.3	Configuração dos interruptores da placa HDMI RX configurada para um canal e suporte de áudio . . . . .	94
B.4	Configuração dos interruptores da placa HDMI TX configurada para um canal e suporte de áudio . . . . .	95
B.5	Configuração dos interruptores da placa HDMI RX configurada para dois canais melhorados . . . . .	95
B.6	Configuração dos interruptores da placa HDMI TX configurada para dois canais melhorados . . . . .	96
C.1	Localização das portas de saída da arquitetura que se conectam à placa HDMI transmissora . . . . .	98
C.2	Localização das portas de entrada da arquitetura provenientes da placa HDMI receptora . . . . .	99

- C.3 Localização de algumas portas de entrada e saída da arquitetura de transmissão de imagem e som entre as placas HDMI transmissora e recetora . . . [101](#)

# 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 [1] 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 [2], 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 [2] 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 [3]. 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 [3], 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.

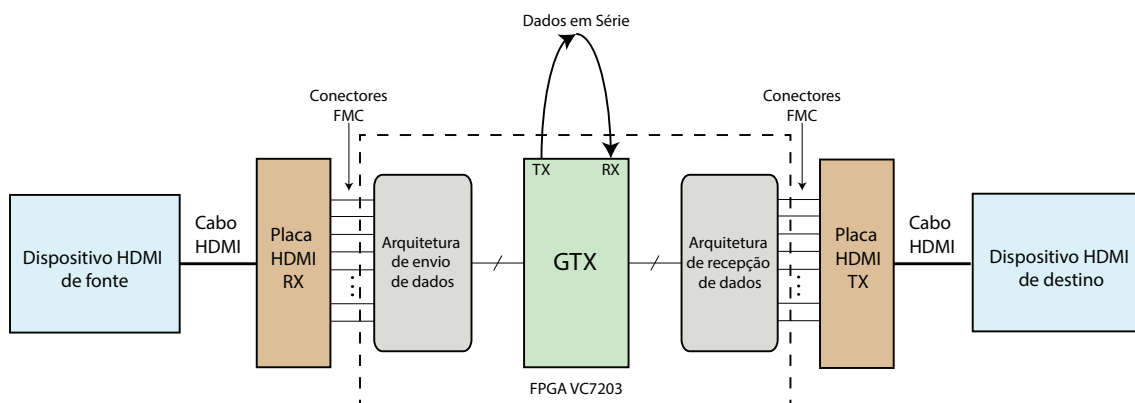


Figura 1.1: Diagrama geral do problema proposto

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 permitem serialização de dados e deserialização dos mesmos.

A primeira parte do projeto consiste em obter comunicações entre dois dispositivos HDMI utilizando para tal as placas HDMI 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.

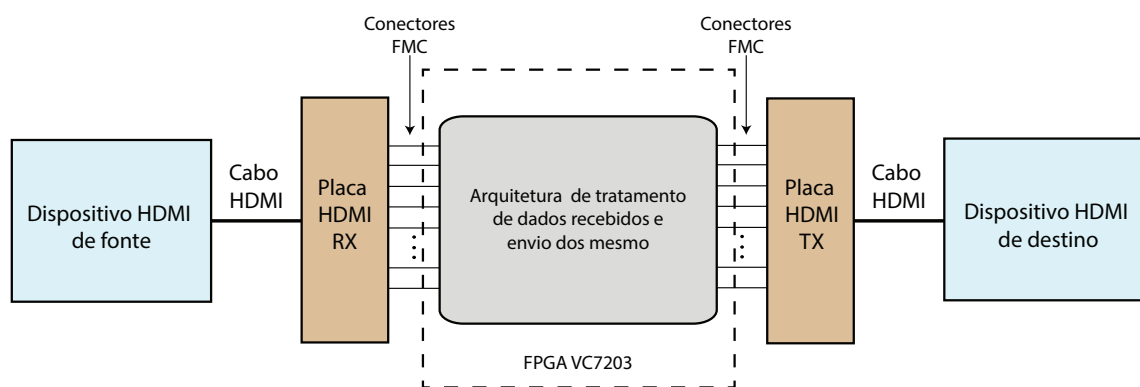


Figura 1.2: Diagrama geral da primeira parte problema

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

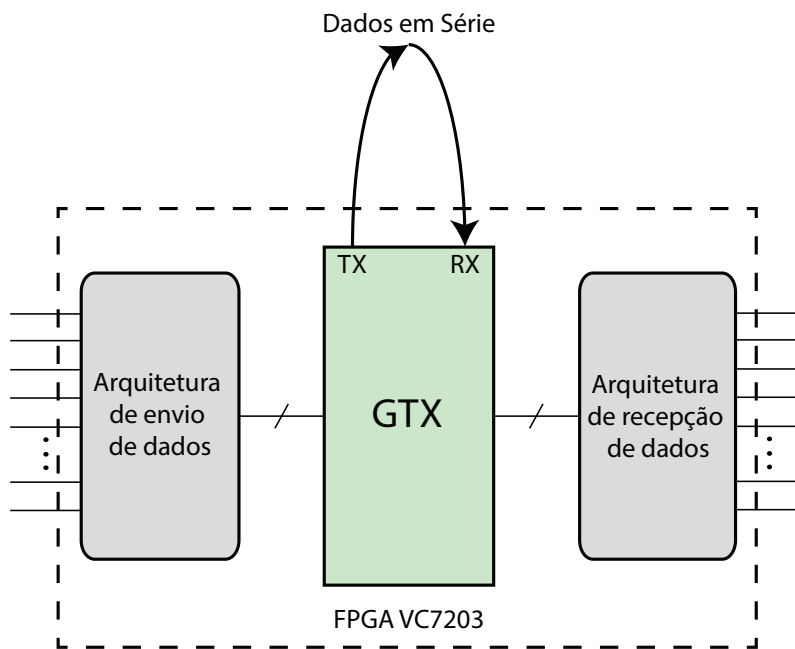


Figura 1.3: Diagrama geral da segunda parte do problema

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 obtém-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 6 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 [4]. 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 [4]:

- ***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 [1]:

### 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.

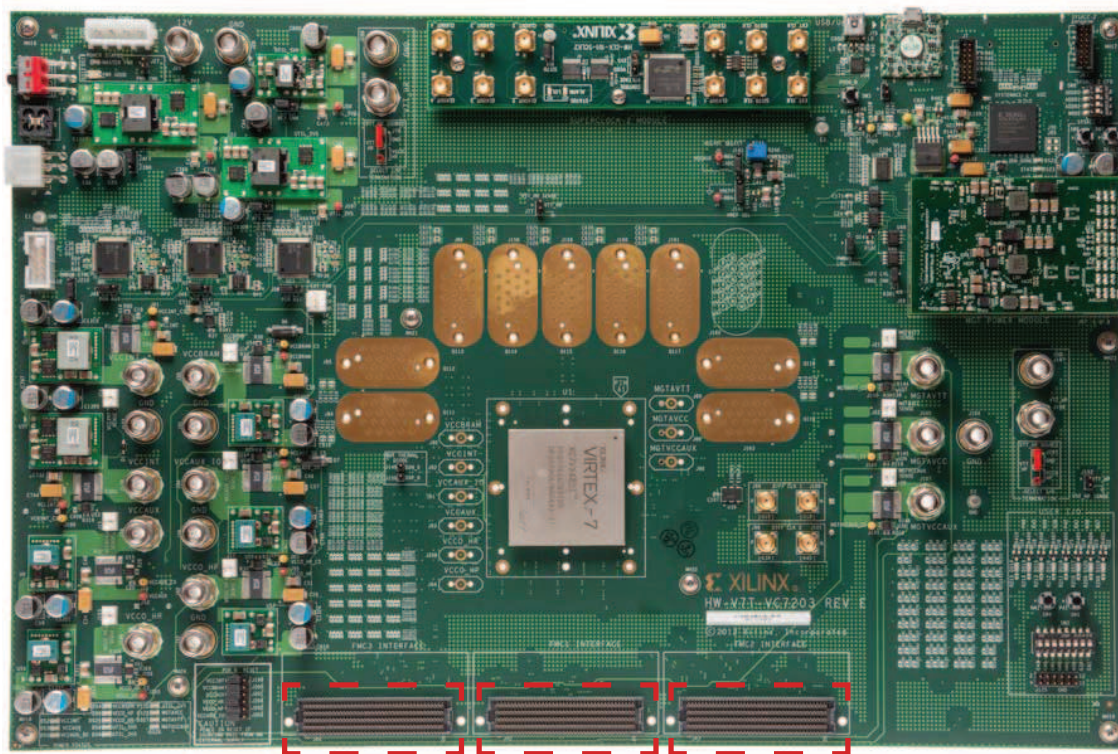


Figura 2.1: Vista Geral da FPGA VC7203 Virtex-7 (retirada de [5])

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 [6], 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 [6], 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 [5], o conector FMC1 dispõe de 68 pares de diferenciais definidos pelo utilizador e ainda 4 sinais de relógio diferenciais. O mesmo acontece para o conector FMC2. No entanto, o conector FMC3 apenas disponibiliza 65 pares de 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.

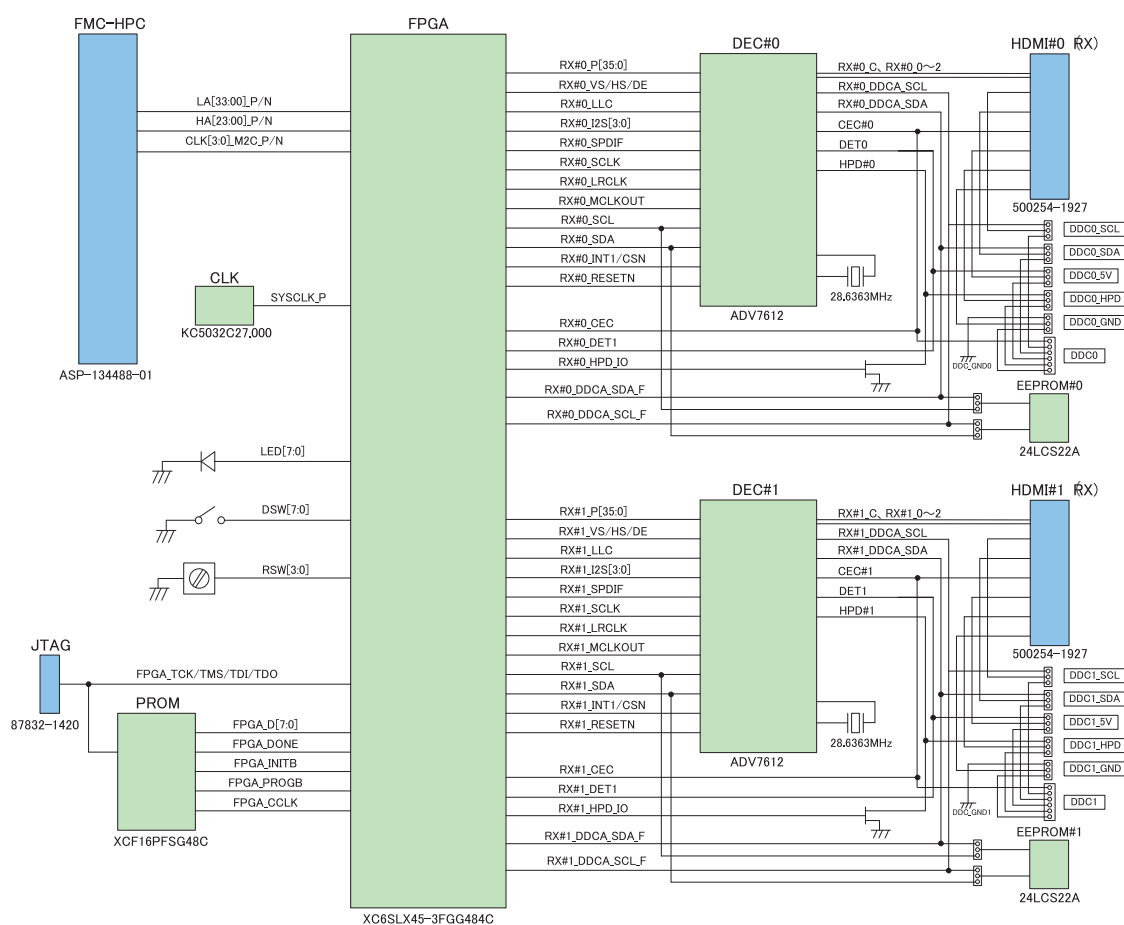


Figura 2.2: Diagrama de blocos de TB-FMCH-HDMI2 RX ()retirado de [7])

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

Tabela 2.1: Nomes dos pins da interface FMC de TB-FMCH-HDMI2 RX (adaptada de [7])

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 [8] conclui-se que a configuração presente na FPGA embebida para além de selecionar 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.

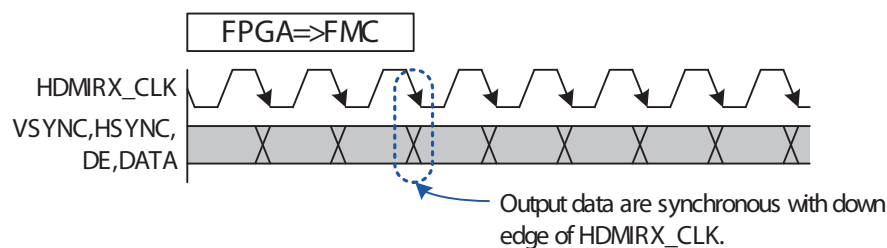


Figura 2.3: Amostragem dos dados provenientes da FPGA no recetor (retirada de [7])

### 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

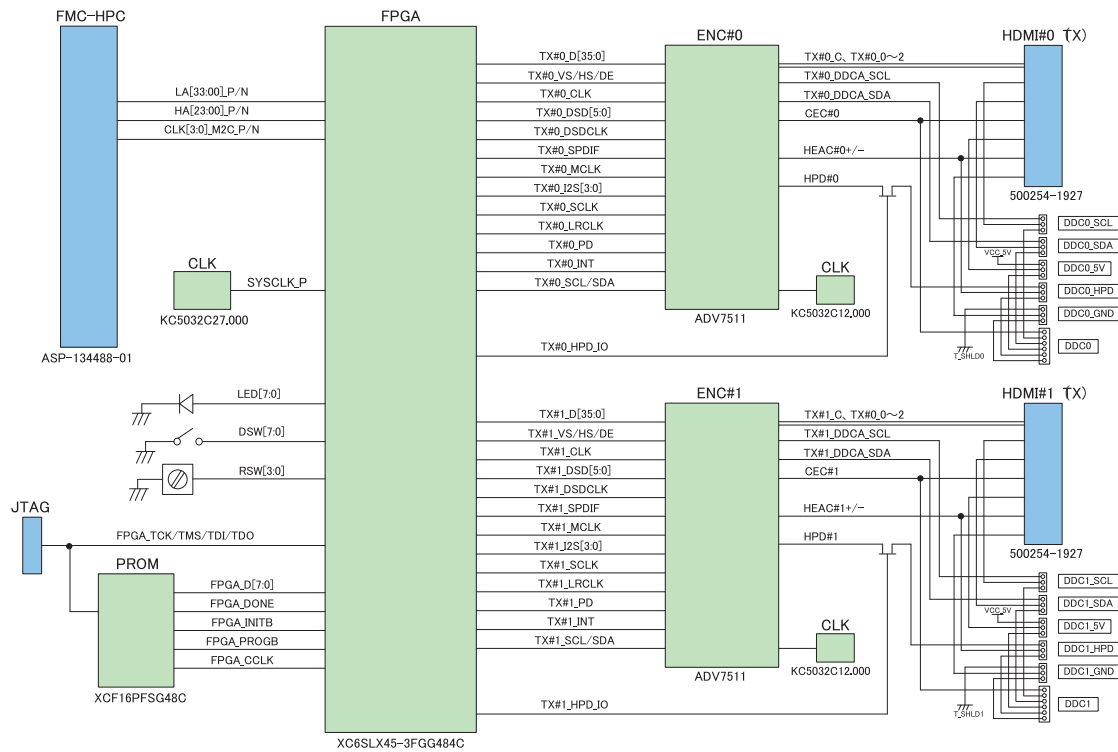


Figura 2.4: Diagrama de blocos de TB-FMCH-HDMI2 TX (retirado de [7])

2.5 da página 16 e depois são processados pela FPGA de maneira a serem enviados para o transmissor HDMI (ADV7511).

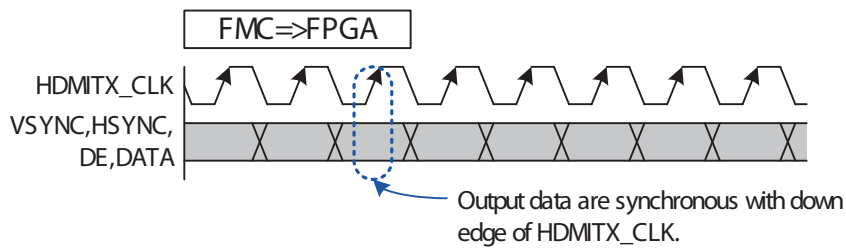


Figura 2.5: Amostragem dos dados provenientes do FMC no recetor (retirada de [7])

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 [9] e [10] conclui-se que para além dos dados de imagem e os seus sinais de controlo, são também enviados para o integrado ADV7511 alguns

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 [7])

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, 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 [11], 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 [12]. 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 [12] : *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 correção 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 [11] 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 [11] 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 [11] 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 eletrónica 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 [11]. 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.

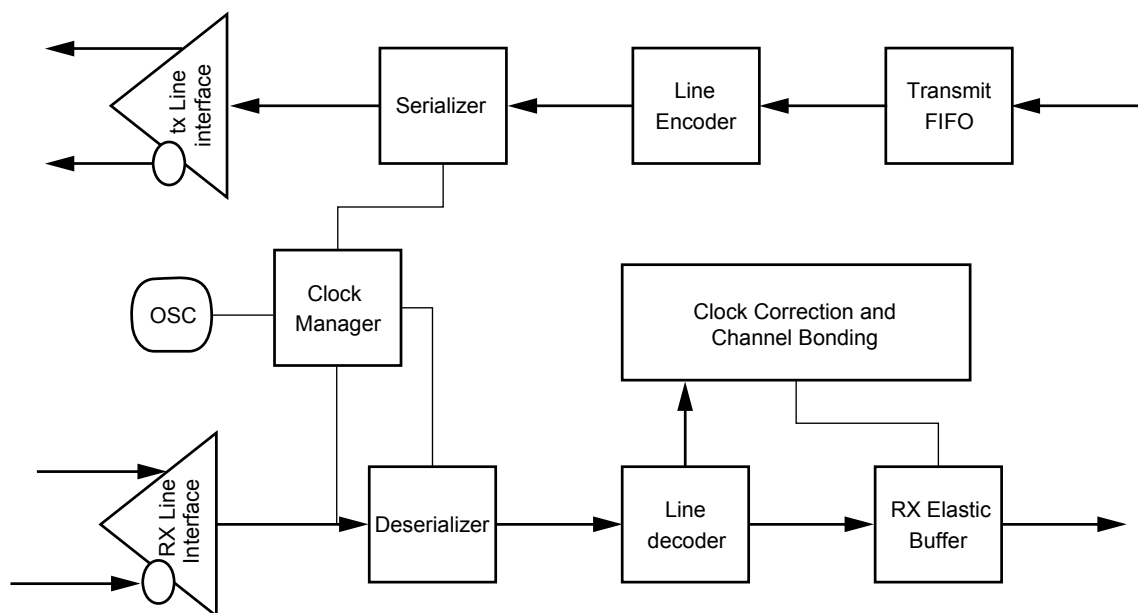


Figura 2.6: Arquitetura simples de um serializador e deserializador (retirada de [11])

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 representada 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*" <sup>1</sup>. Trata-se de um formato que ajuda o recetor a recuperar os sinais dos 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 pelo divisor processar que os sinais de

---

<sup>1</sup>é o termo usado pelo autor de [11]



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 [12], 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 [12], 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 [11], 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 [11], 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 [11])

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 vantagens 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	11010111010	0010101010

Tabela 2.4: Exemplo de palavras de 8 bits codificadas em 8B/10B (retirada de [11])

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 [11] 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 [11], 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 [11])

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

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

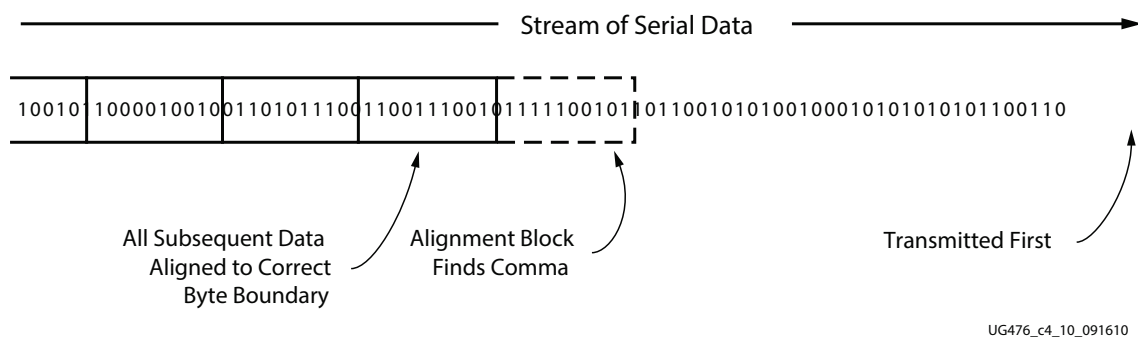


Figura 2.7: Ilustração do alinhamento em série quando encontrada a *comma* (retirada de [13])

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.

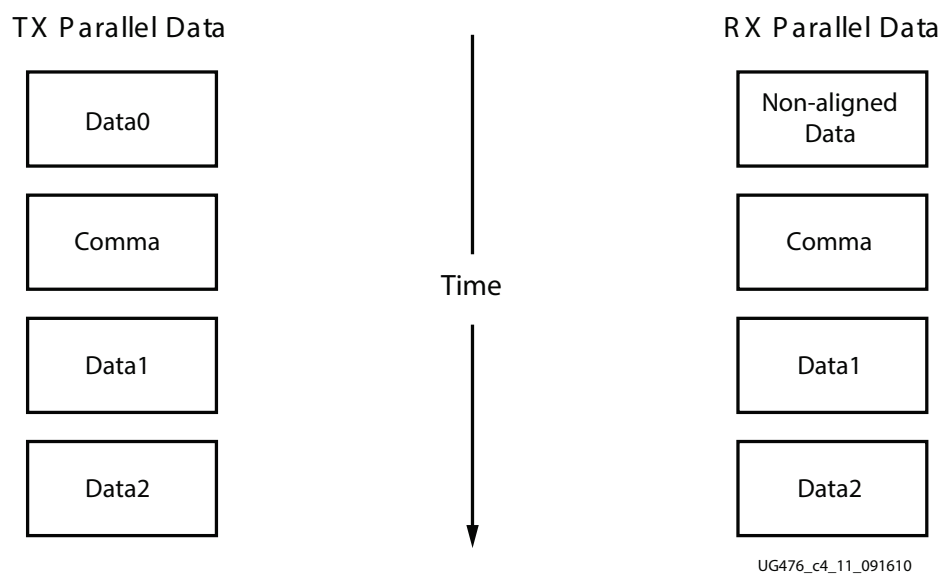


Figura 2.8: Ilustração do alinhamento em paralelo quando encontrada a *comma* (retirada de [13])

Tal como o autor de [11] 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 [12] 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.

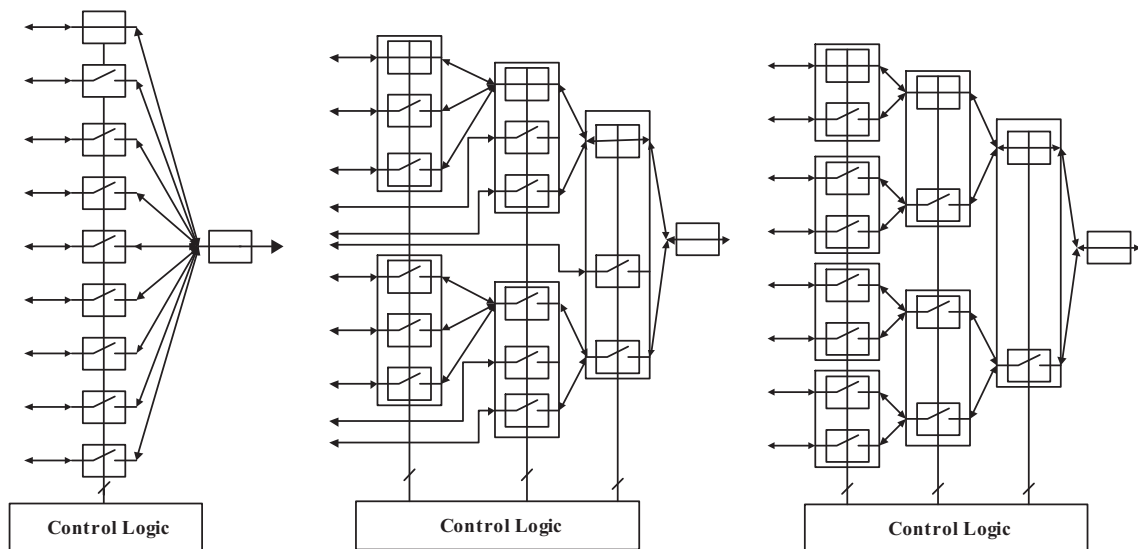


Figura 2.9: Arquiteturas de PISO/SIPO (retirada de [12])

No circuito mais à esquerda, denominada de a) pelo autor de [12], 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 [12]. O autor reforça ainda a ideia de que a utilização de multiplexadores e desmultiplexadores 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 saídas, 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.

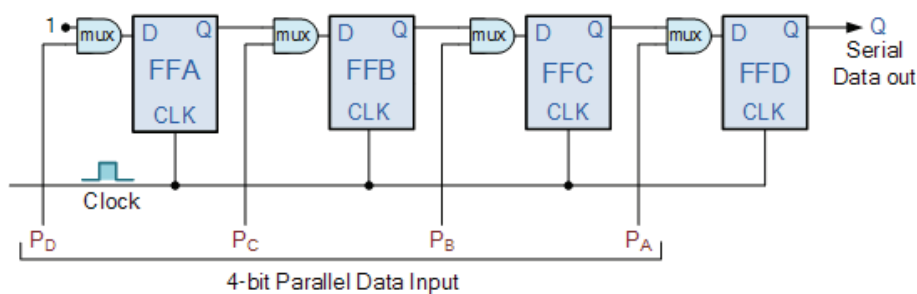


Figura 2.10: Arquitetura de um *shift-register* serializador de 4 bits (retirado de [14])

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.

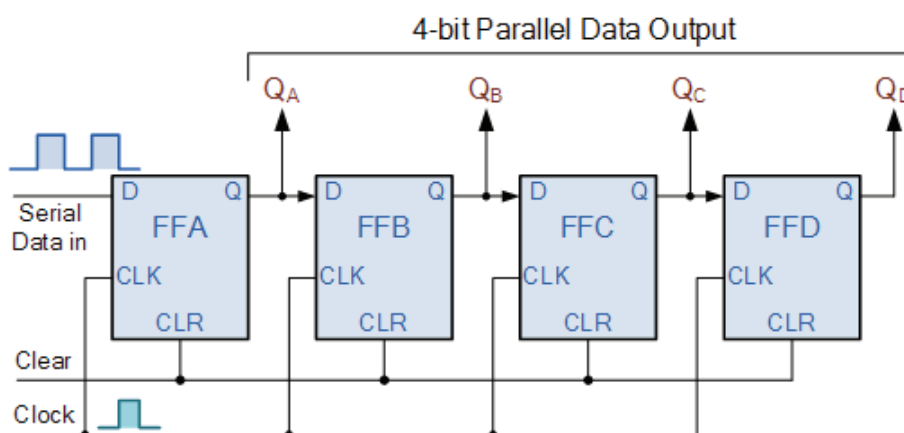


Figura 2.11: Arquitetura de um *shift-register* serializador de 4 bits (retirado de [14])

No caso do deserializador apresentado na imagem 2.11, e segundo a fonte [14], 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 (multipla da mais alta, tal como acontecia no serializador).

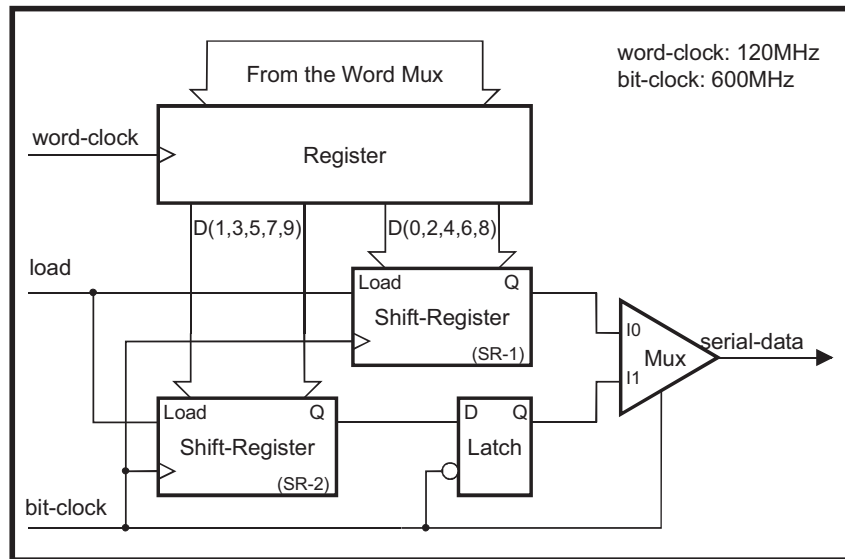


Figura 2.12: Exemplo de um serializador de 10 bits (retirado de [15])

Os autores de [15] 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*". A saída deste modulo 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 correta 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 [12], 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 [11], 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

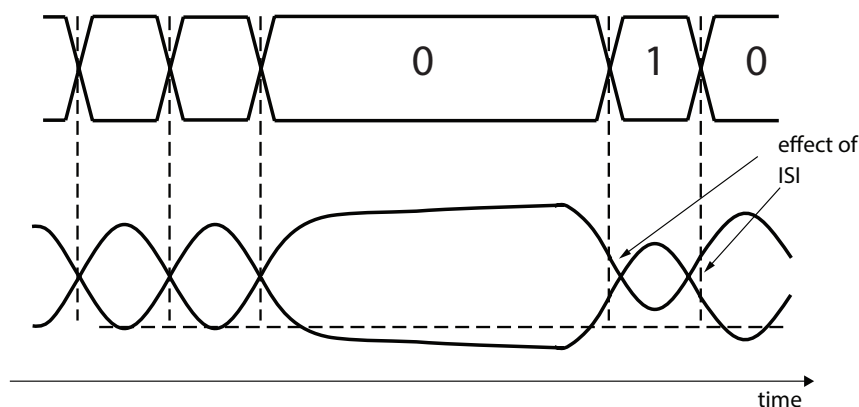


Figura 2.13: Efeito da Interferência Inter-Simbólica numa transmissão (adaptada de [11])



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 [11] 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 [11] 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 [11], 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 [12] e de [11] 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 [11], 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 [11]. 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 acarta 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 [11] 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.
- **LVPECL - *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 [16]. Segundo o autor de [11] 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 [12], 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 [11] como a fonte [12] 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 [12]. 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 [11] 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 [11], 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 [12], 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.

### Detecção 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 [12] 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 [17], 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 [17], 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 transdutores 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 transdutores na FPGA

As FPGA de série 7 da XILINX têm disponíveis transdutores capazes de comunicação em série de alta velocidade, tal como é necessário neste projeto. Em específico, segundo [18], na FPGA XILINX VC7203 Virtex-7 estão disponíveis transdutores GTX que permitem uma velocidade de 12.5 Gb/s e que são os mais adequados para este projeto. Noutros modelos existem outros transdutores, 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 transdutores 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).

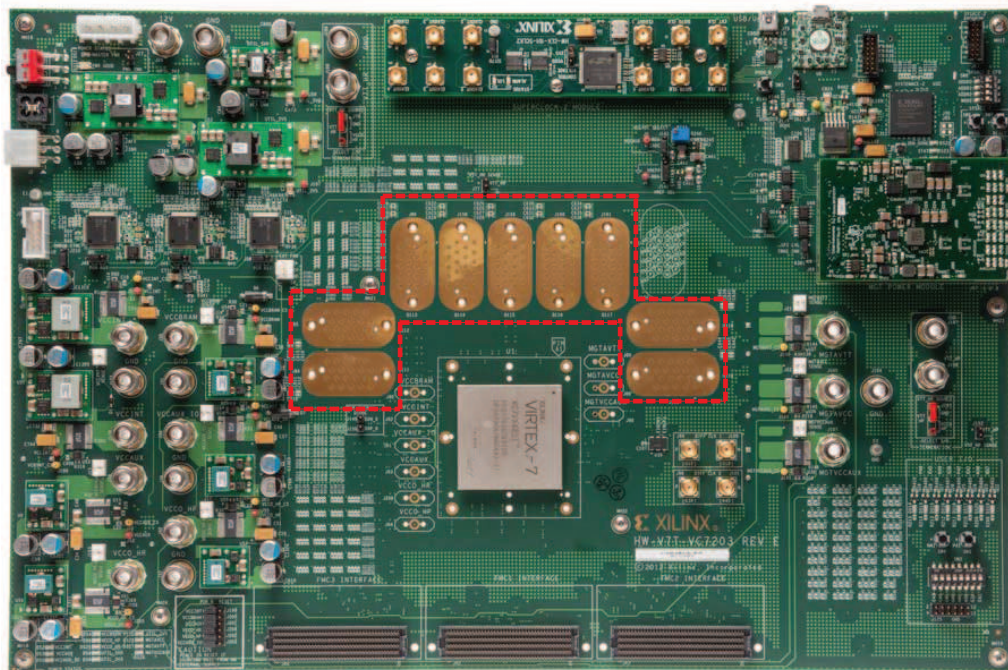


Figura 2.14: Localização física na FPGA dos GTX (retirada de [5])

### 2.5.2 Arquitetura dos transdutores

Cada GTX é composto por quatro transmissores e quatro recetores. 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/recetores e ainda 2 pares diferenciais que são usados para conectar os sinais de relógio de referência. Nestes conectores são ligados cabos SMA diferenciais.

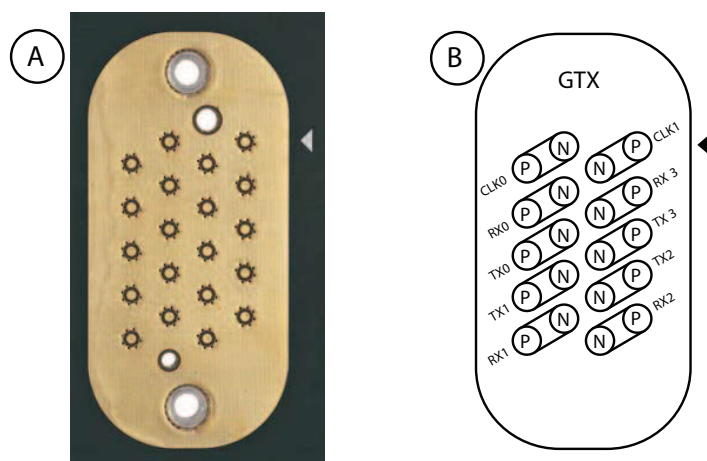


Figura 2.15: Conectores GTX localizados na FPGA (retirada de [5])

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

A arquitetura dos mesmos passa a ser brevemente descrita, segundo [18] e [13]:

- **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



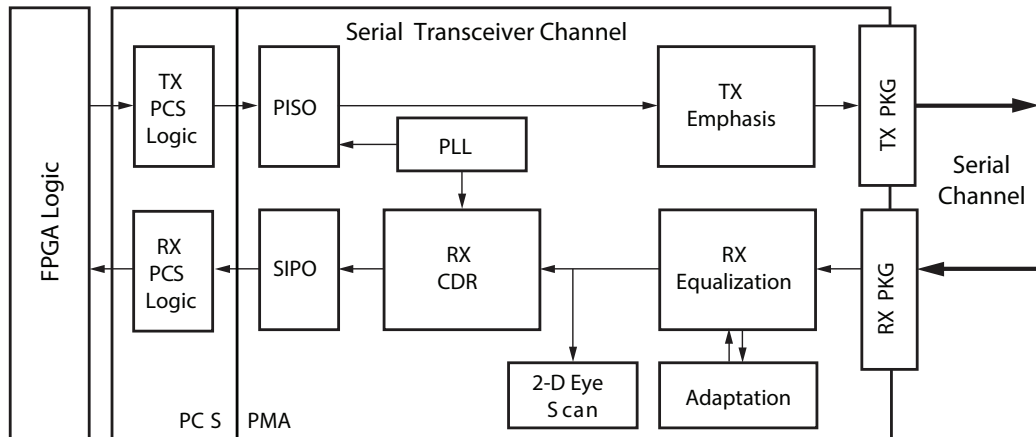


Figura 2.16: Arquitetura geral dos transceptores (retirada de [18])

- 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

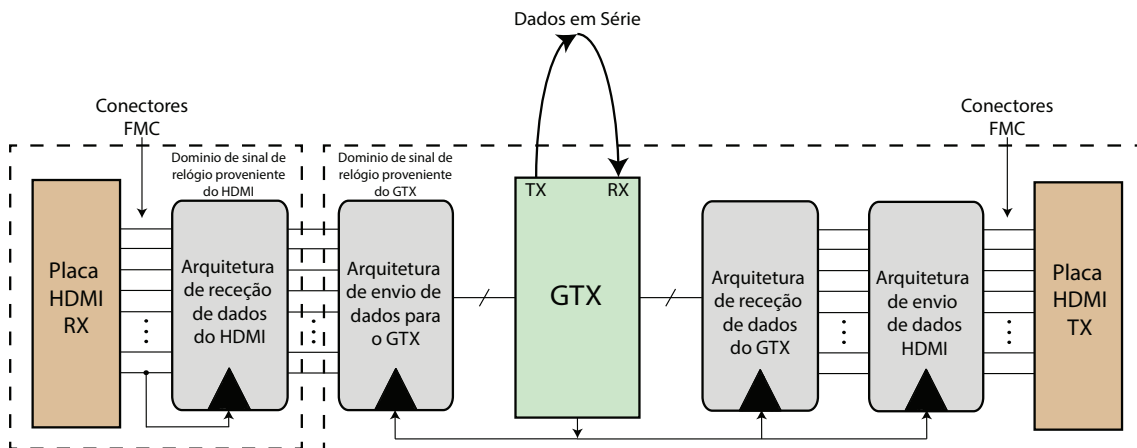


Figura 2.17: Representação dos diferentes domínios de sinais de relógio no projeto

(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 [19] e [20], 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 [20]:

- **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.



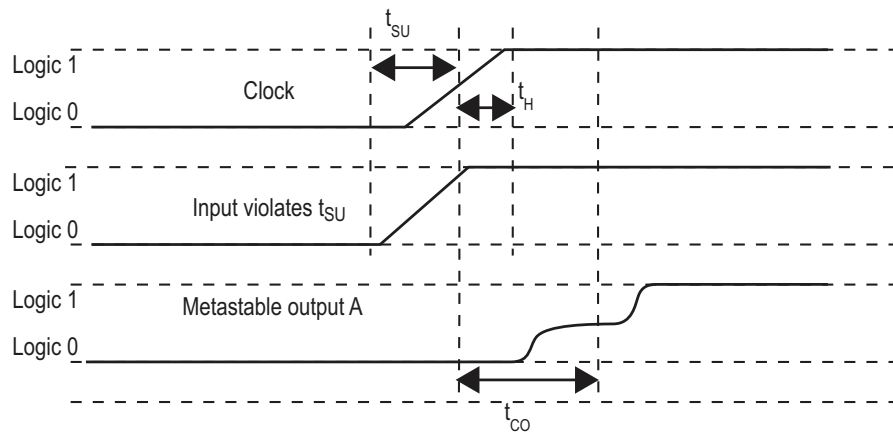


Figura 2.18: Exemplo de metaestabilidade (adaptado de [20])

Segundo [20] e [19], para a correcção 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 registos de sincronização.

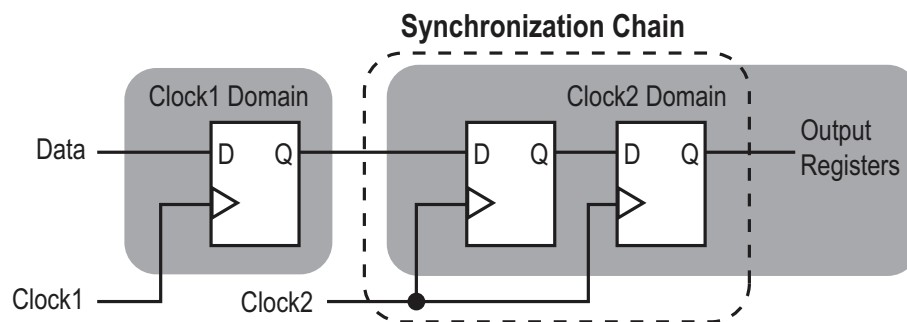


Figura 2.19: Exemplo de cadeia de registos de sincronização (retirado de [20])

Segundo [20], 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

# Transmissão de dados 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 Infraestrutura do *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 saídas 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 (cuja *data sheet* está referenciada em [21]) no caso do recetor e ADV7511 (cuja *data sheet* está referenciada em [22]) no caso do transmissor. Finalmente os sinais provenientes do recetor/transmissor são enviados para uma FPGA da 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 do modo que se pretende. É esta FPGA integrada 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 vêm 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 podem ser reconfiguradas para outros *bitstreams* de forma fácil e eficiente.

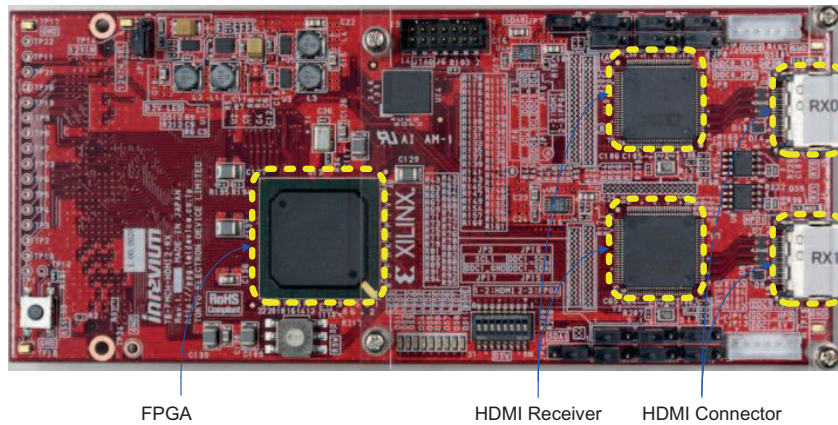


Figura 3.1: Placa HDMI recetora TB-FMCH-HDMI2 RX (retirada de [7])

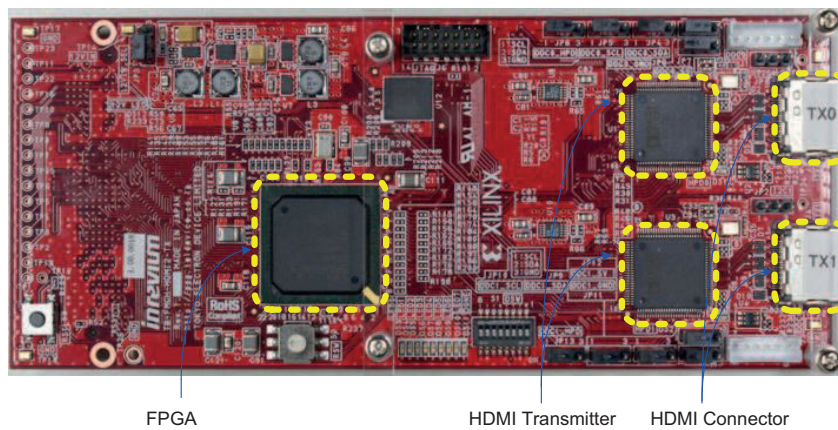


Figura 3.2: Placa HDMI transmissora TB-FMCH-HDMI2 TX (retirada de [7])

As reconfigurações destas memórias são realizadas através de um programador JTAG (*Joint Test Action Group*) 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 - *Universal Serial Bus*) é necessário inicializar o *software* e programar a memória com o ficheiro pretendido. Em [24] são detalhadas informações acerca do programador utilizado e ainda sobre o procedimento para se reconfigurar as memórias. As alterações do modo de funcionamento das placas HDMI neste projeto baseiam-se nesse documento.

### 3.1.1 Configurações da FPGA

A FPGA *Spartan-6* (XC6SLX45-3FGG484C) das placas tem três configurações disponíveis. Estas variam não só no suporte de imagem e som que possuem, mas podem igualmente variar no número de bits por imagem que podem ter. Nas secções seguintes

serão brevemente abordadas as configurações disponíveis e como se pode tirar partido das mesmas no projeto desenvolvido.

### 3.1.1.1 Configuração por omissão

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 à imagem. As tabelas 2.1 e 2.2 nas páginas 14 e 17 respetivamente 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 [7], 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 [7], apresenta brevemente quais as portas das placas utilizadas e que sinais são transmitidos nas mesmas. Todavia é possível encontrar na secção A.1 do anexo A mais detalhes relativamente a estes dados. Os nomes dos sinais apresentados nestas tabelas 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 integradas nas placas HDMI.

PINO	FPGA ->FMC (RX)	FMC ->FPGA (TX)	Descrição
CLK0_M2C_P	RX#0_LLC	TX#0_DCLK	Sinal de relógio dos píxeis
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 omissão

É de notar ainda que esta configuração é capaz de suportar até dois canais (RX0 e TX0, RX1 e TX1). Contudo nesta tabela apenas são apresentados os dados correspondentes ao canal 0 pois apenas é 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 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 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  $I^2S$  (*Inter-IC Sound*). O mesmo acontece na placa transmissora mas para ser capaz de receber estas configurações.

Assim como referido em [25], 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 esse o formato no 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 42 são brevemente apresentadas as portas e os sinais usados com este tipo de configuração. Na secção 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 [25] onde são apresentados todos os detalhes dos conectores FMC das placas.

PINO	FPGA ->FMC (RX)	FMC ->FPGA (TX)	Descrição
CLK0_M2C_P	RX#0_LLC	TX#0_DCLK	Sinal de relógio dos píxeis
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_CCa LA01_N_CC	RX#0_InputVideoStatus	TX#0_InputVideoStatus	Formato de video (2D/3D)
LA19_N	RX#0_MCLK	TX#0_MCLK	Master Clock de som
LA20_N	RX#0_SCLK	TX#0_SCLK	Serial Clock de som
LA21_N a LA26_N	RX#0_AP0 a RX#0_AP5	TX#0_AP0 a TX#0_AP5	Dados de som
LA27_N a LA32_N	RX#0_P30 a RX#0_P35	TX#0_P30 a TX#0_P35	Pixel de imagem do bit 30 ao 35

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

Os dados referentes ao som transmitidos pela placa recetora e recebidos de seguida pela placa emissora estão também mencionados com mais detalhe na tabela ?? 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 [26], 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. Porém, isto pode não acontecer, tal como referido em [26].

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

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.

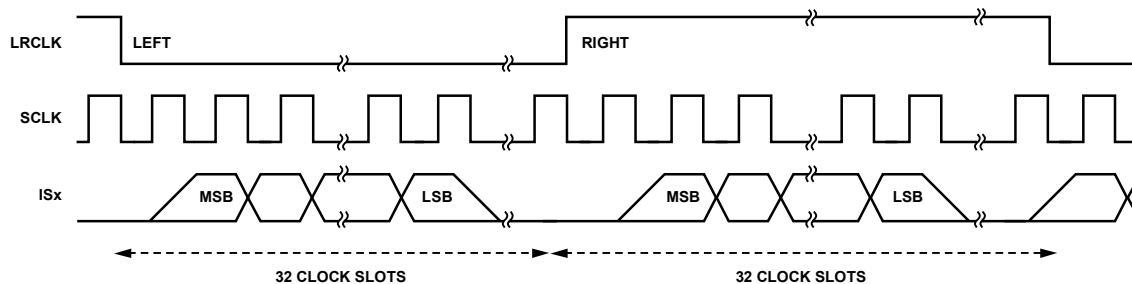


Figura 3.3: Ilustração dos sinais de som transmitidos no formato  $I^2S$  (retirada de [8])

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 dos sinais de áudio da entrada e ainda dados de áudio em AP0. É mencionado em [8] que estes dois sinais são referentes ao som no formato SPDIF (*Sony/Philips Digital Interface Format*) e por isso 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 transmitido e devem de seguida ser recebidos na placa transmissora. As combinações dos dois bits definem o estado do vídeo e são detalhadas em [25].

Esta configuração é 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 mas com alguns melhoramentos. A principal diferença consiste na capacidade de transmitir não só imagens em formato RGB mas também em YCbCr num dos canais. Na secção A.3 do anexo A são apresentados detalhes relativamente a todos os sinais transmitidos pelas placas com esta configuração



Esta configuração na placa recetora transmite no canal 0 (RX0) imagens tanto no formato RGB como YCbCr de 10 bits por cor e respetivos sinais de controlo. Relativamente ao canal 1 dessa mesma placa (RX1) apenas é possível transmitir imagens em formato RGB de 10 bits por cor e os seus sinais de controlo. 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](#).

Quanto 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. Na tabela ?? o canal 1 não é apresentado pois não é relevante para o projeto (visto que só se faz uso de um canal).

### 3.1.2 Configuração dos interruptores

Nesta subsecção são descritas as funcionalidades dos interruptores presentes nas placas HDMI. Estes precisam de estar definidos com determinadas combinações tanto no recetor como no transmissor para que estes possam enviar e receber as 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 seleccionar o tipo de formato que sai do recetor ADV7612 ou do transmissor ADV7511 integrado 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. As diversas combinações estão apresentadas no anexo [B](#).

## 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 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 utilizou-se apenas a placa transmissora HDMI configurada por omissão. 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.

### Concepção e Desenvolvimento

O bloco gerador de uma barra de cores foi adaptado de uma arquitetura 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 píxel, mas também os sinais de controlo como a sincronização vertical (*vsync*), a sincronização horizontal (*hsync*) e ainda os valor de pixeis ativos (*enable*).

Para que se entenda mais facilmente como e quando se transmitem os sinais de controlo da imagem e também o valor dos píxeis é demonstrado na imagem 3.4 na página 46 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 píxeis 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 píxeis 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 píxeis 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 píxeis a ser geradas.
6. **VSW:** *Vertical Sync Width* é o parâmetro que define o número de linhas horizontais que o sinal de sincronização vertical está ativo.
7. **VBP:** *Vertical Back Porch* é o parâmetro que define o número de linhas horizontais que não contém informação útil relativamente aos píxeis antes de começarem a ser transmitidas as linhas de píxeis.
8. **VFP:** *Vertical Front Porch* é o parâmetro que define o número de linhas horizontais que não contém informação útil relativamente aos píxeis depois de terem sido transmitidas todas as linhas horizontais da imagem.

Para gerar uma imagem em *FULL HD* cuja resolução é 1920x1080 píxeis, o sinal de relógio deve ter uma frequência de 148,5 MHz e por isso foram utilizados os seguintes valores para os parâmetros previamente descritos: HRES = 1920, HSW = 148, HBP = 44, HFP = 88, VRES = 1080, VSW = 5, VBP = 36 e VFP = 4. Com estes valores é possível obter uma taxa de refrescamento vertical de 60 Hz, por outras palavras obtém-se 60 imagem por segundo.

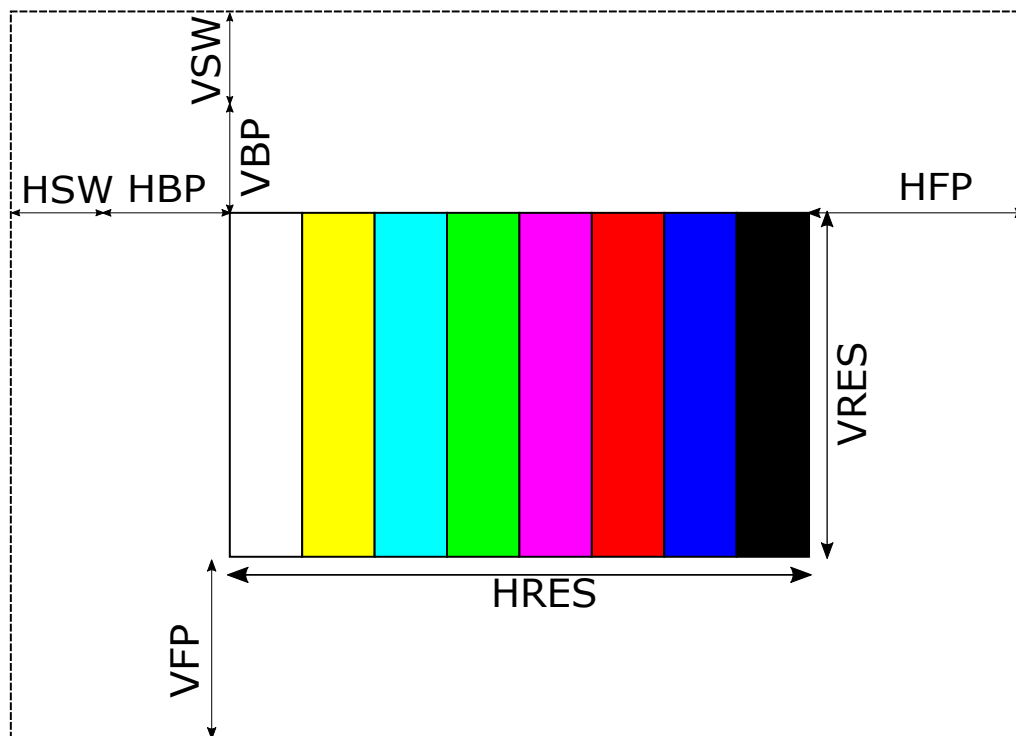


Figura 3.4: Exemplo de imagem gerada pelo módulo desenvolvido

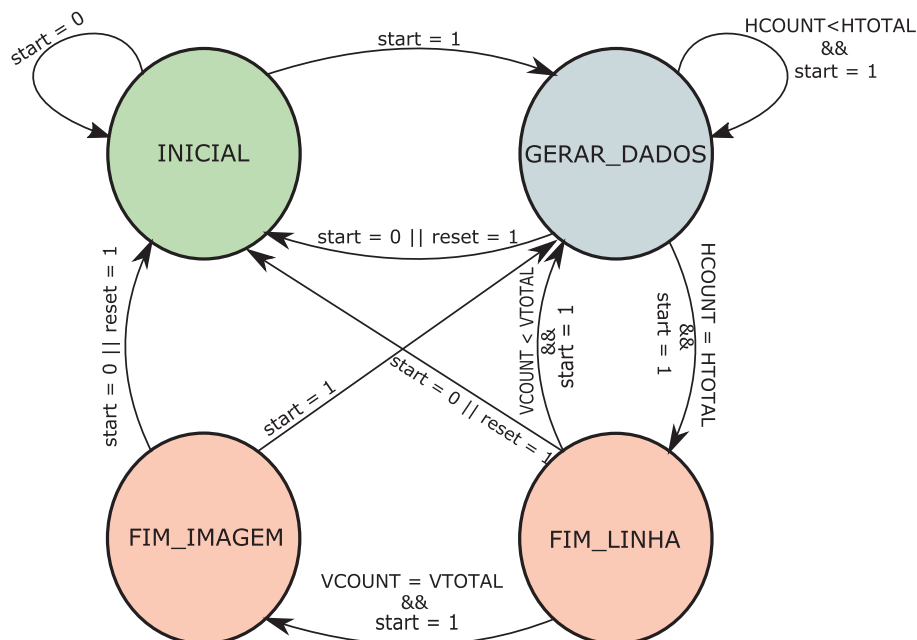


Figura 3.5: Máquina de estados para gerar uma barra de cores

A figura 3.5 na página 46 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 na máquina de estados correspondem a contadores que vão contando píxel a píxel 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 respetivamente. Assim sendo, para este caso em específico obtém-se os seguintes valores:

- $HTOTAL = HSW + HBP + HRES + HFP = 44 + 148 + 1920 + 88 = 2200$
- $VTOTAL = VSW + VBP + VRES + VFP = 5 + 36 + 1080 + 4 = 1125$

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 permite definir quando pretende que a transmissão está ativa ou não (através do botão *start*) ou então quando pretende restabelecer os dados originais da máquina de estados (através do botão *reset*).

Existem 4 estados nesta máquina que consistem essencialmente em deteção do final de uma linha e deteção 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 respetivamente. Por outras palavras, os valores de HCOUNT e VCOUNT são igualados a HTOTAL e VTOTAL respetivamente. 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.
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 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 igualar ao valor de VTOTAL e quando o valor de HCOUNT se igualar ao valor de HTOTAL, ou seja é ativado no final de uma imagem. Este sinal é ainda desligado quando o valor de VCOUNT se igualar a VSW e o valor de HCOUNT se igual ao valor de HTOTAL, isto porque quando estas duas condições se verificam significa 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 por isso deve ser ativo sempre que o valor de HCOUNT se iguale ao valor de HTOTAL (porque indica o fim da emissão de uma linha). Da mesma maneira, este sinal deve ser desligado sempre que o valor de HCOUNT se iguale 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 píxeis válidos e 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 um.
  3. O valor de HCOUNT é maior do que a soma entre HSW, HBP subtraída de um valor.
  4. O valor de HCOUNT é menor do que a soma HSW, HBP e HRES.

As condições 1 e 2 garantem que VCOUNT está na zona vertical que corresponde à transmissão de imagem na figura 3.4 e as condições 3 e 4 garantem o mesmo mas na zona horizontal.

- **Valor dos píxeis:** 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 que o sinal de dados ativos estiver ativo e devem ser igualados a zero sempre que o mesmo estiver inativo.

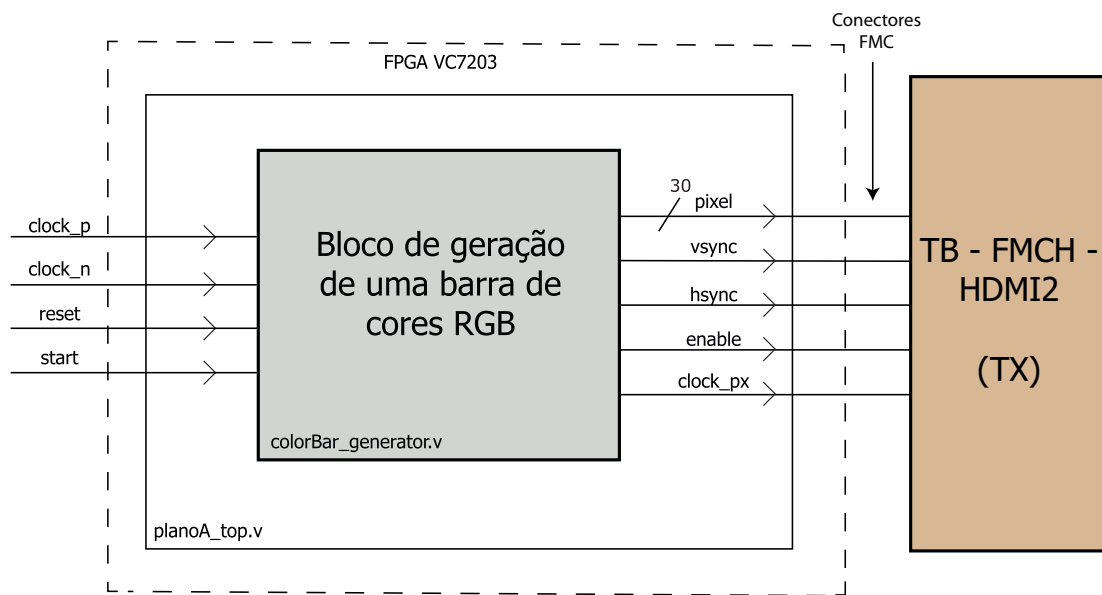


Figura 3.6: Diagrama de blocos de arquitetura implementada utilizando um bloco gerador de barra de cores

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 alimenta um módulo que coloca na sua saída um sinal de relógio de 148,5 MHz. Esse módulo foi criado através do IP (*Intellectual Property*) disponibilizado no *software* VIVADO com o nome de *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 principal sinal de relógio do sistema uma vez que é a frequência necessária para gerar uma imagem em *FULL HD* sendo esta a cadência a que os sinais são enviados para a placa HDMI transmissora e é esse ainda o sinal de relógio da mesma.

Relativamente às saídas do módulo é possível ver na imagem 3.6 que estas se encontram diretamente ligadas à placa transmissora HDMI através dos conectores FMC. Estes sinais constituem um barramento de 30 bits que corresponde ao valor do píxel (*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 isso é necessário definir onde estão as localizações das portas na FPGA (LOC) e criar um ficheiro que defina essas mesmas restrições físicas. A tabela 3.3 na página 50 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. Mas na secção C.1 do anexo C é possível encontrar todas as portas com mais detalhes e ainda com informação sobre a ligação à placa HDMI transmissora.

	Sinal	LOC na FPGA	Banco na FPGA
<b>Entrada</b>	clk_p	E19	38
<b>Entrada</b>	clk_n	E18	38
<b>Entrada</b>	reset	N41	19
<b>Entrada</b>	start	E42	19
<b>Saída</b>	clk_px	E34	35
<b>Saída</b>	enable	K35	34
<b>Saída</b>	hsync	M32	34
<b>Saída</b>	vsync	L31	34
<b>Saída</b>	pixel[0] a pixel [29]	Ver anexo	34 e 35

Tabela 3.3: Localização das portas de entrada e saída da arquitetura de transmissão de imagem gerada na FPGA para a placa HDMI transmissora

O ficheiro com estas restrições físicas gerado após a atribuição das mesmas é apresentado nas secções D.1 e D.3 do anexo D. Em D.1 são apresentados as restrições referentes às saídas da arquitetura que se conectam à placa HDMI transmissora, e em D.3 faz-se referência a todas as outras entradas e saídas desta mesma arquitetura.

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 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 na secção D.4 do anexo D. 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 é importante que a ferramenta de implementação saiba o seu valor para poder garantir que toda a arquitetura cumpre os requisitos temporais.

## Resultados

Após a definição de todas as restrições e escrita do código em Verilog, a arquitetura desenvolvida foi devidamente implementada na FPGA e testada utilizando-se o *set-up* de teste visualizado na imagem 3.7 da página 51.

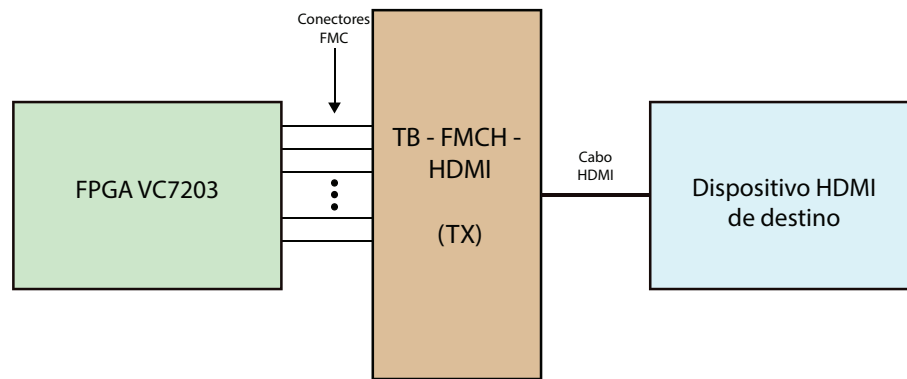


Figura 3.7: *Set-Up* de teste da arquitetura desenvolvida para transmissão de uma imagem gerada na FPGA para a placa HDMI transmissora

Os resultados obtidos foram os esperados, a visualização de uma barra de cores no dispositivo HDMI de destino.

### 3.2.2 Transmissão de imagem entre dispositivos HDMI

Na arquitetura desenvolvida que é apresentada nesta subsecção são utilizadas as placas HDMI recetora e transmissora, ambas configuradas por omissão 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.

#### Concepção e Desenvolvimento

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) e o resto dos sinais provenientes da mesma, mais especificamente o valor de *pixel*, *vsync*, *hsync* e *enable*. A imagem 3.8 da página 52 ilustra o diagrama de blocos da arquitetura desenvolvida.

É possível visualizar que nesta arquitetura, tal como já acontecia na anterior, existe na sua saída os sinais que são enviados para a placa HDMI transmissora e para além disso na sua entrada existe 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 diferencial 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 síncronos com o sinal de relógio proveniente da entrada (da placa HDMI recetora). Quando o sinal definido pelo utilizador *start* está ativo, os sinais selecionados pelo multiplexador visível na figura 3.8 são

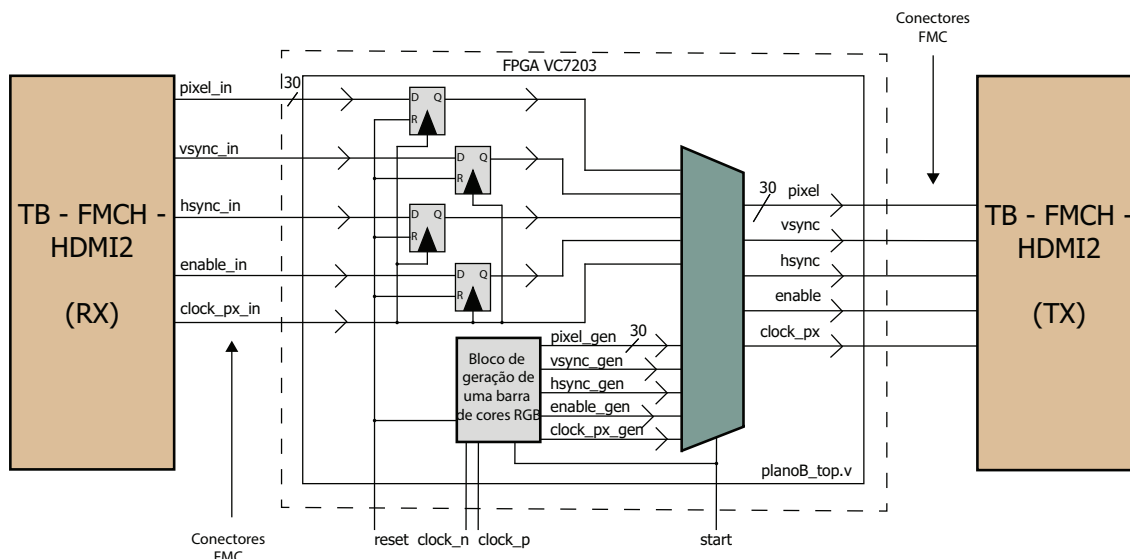


Figura 3.8: Diagrama de blocos da arquitetura desenvolvida para transmitir imagem entre dispositivos HDMI

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 inativo obtém-se uma ligação entre as placas HDMI recetora e transmissora pois os sinais selecionados pelo multiplexador são 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 3.4 na página 53 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. Contudo na secção C.2 do anexo C é possível encontrar mais detalhadamente as localizações dessas portas na FPGA bem como informação relativamente a esses sinais nas placas HDMI transmissora e recetora.

A atribuição destas mesmas localizações das portas gerou um ficheiro de restrições físicas que é apresentado nas secções D.2, D.1 e D.5. A secção D.2 define as portas de entrada que estão conectadas à placa HDMI recetora. A secção D.1 apresenta as restrições das portas de saída da arquitetura que se conectam à placa HDMI transmissora, e por fim a secção D.5 apresenta as restrições das restantes portas de entrada e saída desta arquitetura. Relativamente a estas restrições físicas, existem para cada porta duas: uma que define a localização física e outra a norma da mesma, tal como já foi mencionado anteriormente. Todas estes ficheiros de restrições encontram-se no anexo D.

Quanto às restrições temporais aplicadas ao sistema são idênticas às que foram aplicadas na arquitetura desenvolvida anteriormente e estão presentes em D.6 no anexo D.



	Sinal	LOC na FPGA	Banco na FPGA
<b>Entrada</b>	clk_p	E19	38
<b>Entrada</b>	clk_n	E18	38
<b>Entrada</b>	reset	N41	19
<b>Entrada</b>	start	E42	19
<b>Entrada</b>	clk_px_in	AJ32	14
<b>Entrada</b>	enable_in	AN38	15
<b>Entrada</b>	vsync_in	AU38	15
<b>Entrada</b>	hsync_in	AU39	15
<b>Entrada</b>	pixel_in [0] a pixel_in[29]	Ver anexo	14 e 15
<b>Saída</b>	clk_px	E34	35
<b>Saída</b>	enable	K35	34
<b>Saída</b>	hsync	M32	34
<b>Saída</b>	vsync	L31	34
<b>Saída</b>	pixel [0] a pixel [29]	Ver anexo	34 e 35

Tabela 3.4: Localização das entradas e saídas das portas da arquitetura de transmissão de uma imagem entre dispositivos HDMI

## Resultados

Após síntese e implementação do código desenvolvido em Verilog juntamente com as restrições aplicadas, a FPGA foi programada com esta arquitetura e testada recorrendo-se a um *set-up* de teste como ilustra a imagem 3.9 na página 53. Foi utilizado um computador portátil como fonte HDMI e conectou-se o mesmo à placa recetora. Como dispositivo final utilizou-se um monitor que foi conectado à placa transmissora através de um cabo HDMI também. Os resultados obtidos foram os esperados: transmissão de imagem entre os dois dispositivos.

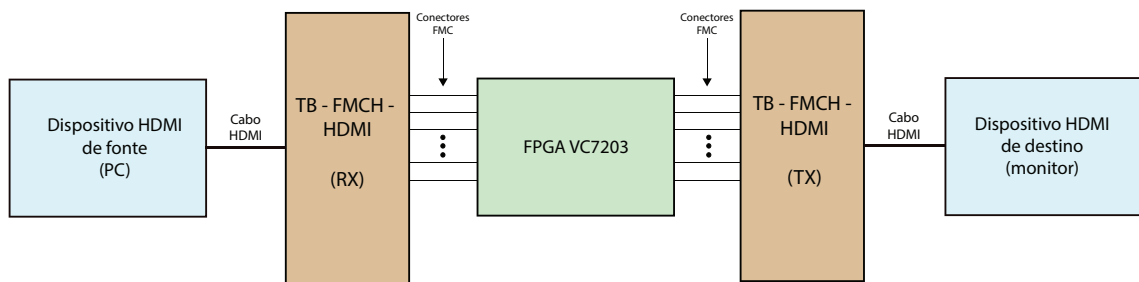


Figura 3.9: *Set-up* de teste para as arquiteturas transmissoras de dados entre dispositivos HDMI

### 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 isso foi necessário reconfigurar as placas HDMI, tal como mencionado anteriormente, para a

configuração que suporta apenas um canal mas que permite a transmissão de áudio em formato  $I^2S$ . As características desta configuração são apresentadas na subsecção 3.1.1.2 na página 41 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 na secção B.2 do anexo B). Neste caso específico são utilizados 12 bits por cor o que perfará um total de 36 bits por píxel.

## Concepção e Desenvolvimento

Na imagem 3.10 na página 55 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: *pixel*, *hsync*, *vsync* e ainda *enable*. Para além destes sinais provenientes da placa HDMI recetora são recebidos os sinais referentes ao som: o sinal de relógio dos dados em série (*sclk*), o sinal referente à seleção do canal de áudio 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 vídeo que é transmitido (tal como já referido na subsecção 3.1.1.2). Todos estes sinais que se encontram na entrada do módulo encontram-se também na saída pois estes são enviados para placa HDMI transmissora.

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 diferencial de 200 MHz definido na porta com *clock\_p* pelo sinal positivo e como *clock\_n* pelo sinal negativo. Este tem a mesma função que na arquitetura anterior: gerar o sinal de relógio para o módulo que produz a barra de cores, tal como previamente descrito.

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 a 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 quando está inativo seleciona no multiplexador as saídas do bloco que produz uma barra de cores em vez dos sinais provenientes da placa HDMI recetora.

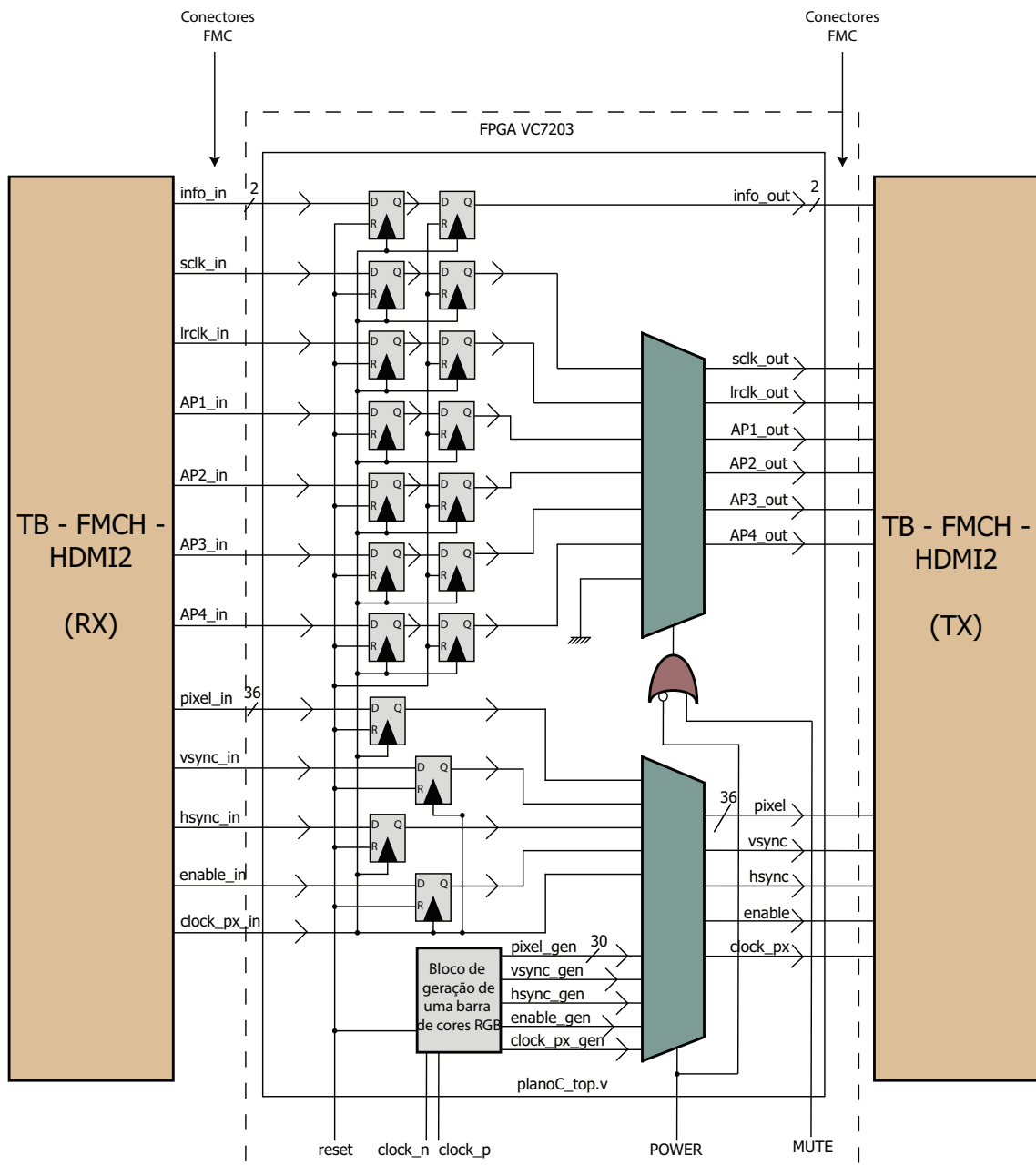


Figura 3.10: Diagrama de blocos da arquitetura desenvolvida para transmitir imagem e som entre dispositivos HDMI

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 esta 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 va-

riam a uma frequência de 3,072 MHz (tal como mencionado na subsecção 3.1.1.2). Assim sendo, apesar de o sinal de relógio do áudio ser mais lento do que o da imagem, é necessário tomar as devidas precauções quando se faz a passagem entre estes dois domínios de relógio para que não sejam captados dados dos registos quando estes se encontram num estado de meta-estabilidade. Tal como sugerido em [19], 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 dos sinais de *POWER* e *MUTE*. Caso *MUTE* esteja ativo o sinal de som não é transmitido e as entradas da placa HDMI são definidas com 0. O mesmo acontece quando o sinal de *POWER* está inativo 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 módulo principal, que estão descritas na tabela ?? na página ?. Nessa mesma tabela os sinais que conectam as placas HDMI são brevemente descritos e na secção C.3 do anexo C são descritos com mais detalhe.

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 conteúdo do ficheiro pode ser encontrado nas secções D.1, D.2 e ainda em D.7 do anexo D. Na secção D.1 é possível encontrar as restrições físicas relativas a algumas portas de saída do módulo, nomeadamente os bits do sinal de píxel entre 0 e 29, o seu sinal de relógio e ainda os sinais de controlo. Na secção D.2 encontram-se as restrições físicas de algumas portas de entrada desta arquitetura provenientes da fonte HDMI. Essas portas correspondem, mais uma vez, aos bits do sinal de píxel entre 0 e 29, o seu sinal de relógio e os respetivos sinais de controlo. Por fim, em D.7 encontram-se as restrições físicas referentes às restantes portas de entrada e saída da arquitetura que se conectam às placas e não foram contempladas e também relativamente às outras portas do módulo que não se conectam às placas HDMI.

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 D.8 no anexo D.

## Resultados

Esta arquitetura foi devidamente implementada na FPGA e testada utilizando-se um *set-up* de teste semelhante ao que se visualiza na figura 3.9 na página 53, tendo em conta que o monitor utilizado neste teste tinha saída de áudio. Assim, os resultados obtidos foram os esperados: a transmissão de imagem e som entre os dois dispositivos foi bem sucedida. Apesar de os sinais de áudio provenientes da placa HDMI recetora serem síncronos com

sinal de relógio que não é o dos píxeis, a passagem de sinais entre estes dois domínios de relógio diferentes não apresentou qualquer tipo de problema uma vez que ao ouvido humano os pequenos atrasados que possam eventualmente acontecer não são perceptíveis.



## Capítulo 4

# Transmissão dos dados em série

Este capítulo aborda o trabalho realizado na segunda parte do projeto, tal como definido em [1.3](#). Esta parte do projeto consiste em transmitir em paralelo para o transmissor GTX existente na FPGA VC7203 que ,através da sua arquitetura, serializa os dados e envia-os através de um cabo físico. Esse mesmo cabo conectado ao recetor GTX irá deserializar para que sejam convertidos novamente para dados em paralelo.

Este processo de serialização e deserialização exige muitos cuidados e aplicações de determinados métodos que permitam manter a integridade do sinal durante a transmissão do mesmo, já como explicado em no capítulo 2 em [2.4.2](#). Todavia nesse mesmo capítulo, em [2.5.2](#) foi já abordado que FPGA VC7203 possui entradas e saídas de alta velocidade com arquiteturas que possuem blocos que permitem a implementação destes mecanismos. Na imagem [2.16](#) da página [35](#) visualiza-se uma arquitetura geral dos transctores, mas estes serão descritos com mais detalhe neste capítulo.

Este capítulo começa por abordar as arquiteturas transmissora e recetora com mais detalhe, quais as características que são vantajosas a este projeto e ainda analisar as vantagens de desvantagens de algumas para poder fazer uma melhor escolha para o projeto. De seguida serão contempladas as arquiteturas desenvolvidas e explicadas todas as decisões no seu desenvolvimento.

### 4.1 Transmissor

Na figura [4.1](#) da página [60](#) visualiza-se a arquitetura do transmissor GTX. Assim como já foi mencionado, esta arquitetura possui muitos blocos com diferentes funções que permitem manter a integridade do sinal durante uma transmissão, pelo menos, daquilo que se pode fazer do lado do transmissor.

Os blocos mais relevantes para o funcionamento do projeto passam de seguida a ser descritos nas próximas subsecções para que se possa entender mais facilmente as arquiteturas desenvolvidas quando estas forem apresentadas.

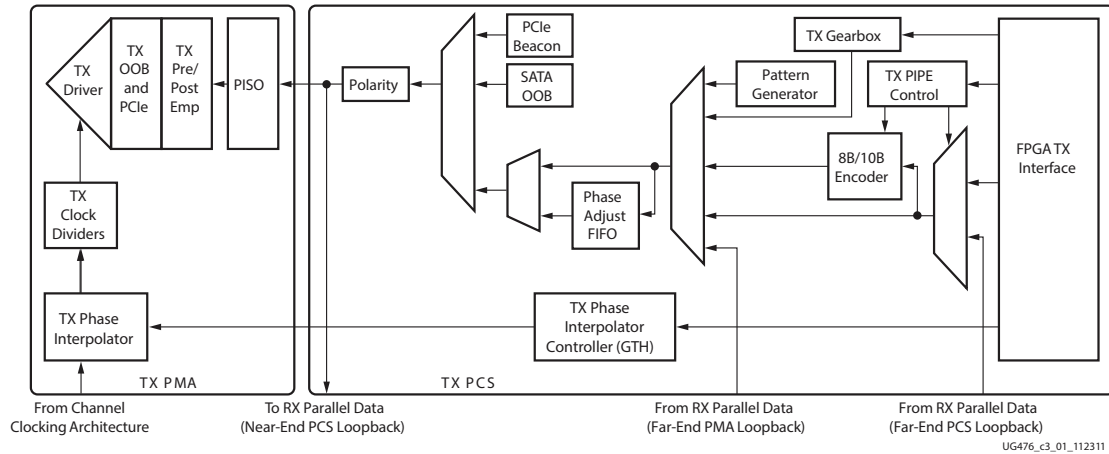


Figura 4.1: Arquitetura do transmissor GTX (retirada de [13])

#### 4.1.1 Interface com a FPGA

Na imagem 4.1 este bloco tem o nome de "*FPGA TX Interface*" e é o bloco por onde entram os dados em paralelo provenientes da FPGA que se pretendem serializar. Segundo [13], o tamanho desta interface depende de vários fatores internos, como por exemplo, o tamanho interno que os dados terão (pode estar dependente da utilização de codificação) e ainda o tamanho do *datapath* que se escolhe usar.

Para se perceber melhor como o tamanho da porta desta interface pode variar é apresentada a table 4.1 na página 60 que expõe os casos possíveis.

Codificação 8B/10B	Tamanho do <i>datapath</i>	Tamanho Interno	Interface com a FPGA
Sim	0 (2 byte)	20	16
Sim	0 (2 byte)	20	32
Sim	1 (4 byte)	40	32
Sim	1 (4 byte)	40	64
Não	0 (2 byte)	16	16
Não	0 (2 byte)	20	20
Não	0 (2 byte)	16	32
Não	1 (4 byte)	32	32
Não	0 (2 byte)	20	40
Não	1 (4 byte)	40	40
Não	1 (4 byte)	32	64
Não	1 (4 byte)	40	80

Tabela 4.1: Tamanhos possíveis para a interface da FPGA com o GTX transmissor (adaptada de [13])

Os diferentes tamanhos que a porta da interface pode tomar estão apresentados na última coluna da tabela tendo em conta que quando é implementada uma codificação então estes apenas podem variar entre 16, 32 ou 64. Todavia quando não há codificação, então o tamanho desta porta pode ser os valores anteriormente referidos ou então até



mesmo 20,40 ou 80. De notar que quando não há codificação e o tamanho da porta é 16, 32 ou 64 existe a possibilidade de estender esses valores (caso se pretenda) para 20, 40 e 80 respetivamente. No entanto, tal não será abordado dado que não é importante para o projeto.

Os valores do tamanho interno do *datapath* podem variar entre 2 e 4 bytes, contudo é necessário tomar em atenção que quando um *datapath* de um determinado tamanho não chega para o número de bits à entrada então é necessário utilizar dois. Mas isto será abordado mais à frente nesta subsecção.

Os dados em paralelo entram neste bloco a uma determinada cadência bem definida : ao flanco positivo do sinal de relógio *TXUSRCLK2*. Este sinal de relógio é o sinal de sincronização entre os dados provenientes da FPGA com o transmissor, no entanto é também necessário outro sinal de relógio para a lógica interna PCS do transmissor : *TXUSRCLK*. De recordar que a lógica interna PCS trabalha com os dados em paralelo ainda e portanto o valor deste sinal de relógio vai depender não só da cadência a que os dados em paralelo (*TXUSRCLK2*) entram no transmissor mas também do *datapath* escolhido.

Por um lado, o sinal de relógio *TXUSRCLK2* é o principal sinal de sincronização entre a lógica da FPGA e a lógica do transmissor, por outro o sinal de relógio *TXUSRCLK* é a cadência determinista dos dados em paralelo no bloco PCS, e por isso existe uma relação entre estes dois sinais muito bem definida que passa a ser apresentada na tabela 4.2 na página 61.

Tamanho na Interface	Tamanho do <i>datapath</i>	Relação entre os sinais de relógio
16, 20 (bits)	0 (2 byte)	$F_{TXUSRCLK} = F_{TXUSRCLK2}$
32, 40 (bits)	0 (2 byte)	$F_{TXUSRCLK} = F_{TXUSRCLK2} * 2$
32, 40 (bits)	1 (4 byte)	$F_{TXUSRCLK} = F_{TXUSRCLK2}$
64, 80 (bits)	1 (4 byte)	$F_{TXUSRCLK} = F_{TXUSRCLK2} * 2$

Tabela 4.2: Relação entre as frequências dos sinais de relógio *TXUSRCLK2* e *TXUSRCLK* (adaptada de [13])

Segundo [13], sabendo estas relações entre os sinais de relógio, o tamanho da porta na interface do transmissor e o tamanho do *datapath* usado é possível obter a velocidade de transmissão dos dados em série através da equação apresentada em 4.1 na página 61.

$$Line\ Rate = F_{TXUSRCLK} * (Internal\ Datapath\ Width) \quad (4.1)$$

Para que se perceba a diferença entre estas frequências é agora apresentado um caso em concreto que é utilizado neste projeto: pretende-se na entrada do transmissor obter 40 bits em paralelo, ou seja, ter uma porta com uma largura de 40 bits, e que estes dados sejam amostrados a uma frequência de 148,5 MHz ( $F_{TXUSRCLK2} = 148,5\ MHz$ ). Assim sendo, existem agora duas hipóteses: a utilização de um *datapath* de 2 byte ou 4 byte.

Se se optar pela utilização de um *datapath* de 4 byte então está-se perante a opção a), ilustrada na imagem 4.2 da página 62, em que o sinal de amostragem à entrada é de 148,5 MHz ( $F_{TXUSRCLK2} = 148,5 \text{ MHz}$ ) e o sinal de lógica do bloco PCS também é 148,5 MHz ( $F_{TXUSRCLK} = 148,5 \text{ MHz}$ ), tal como ilustrado na figura. Neste caso a largura do sinal interno é de 40 bits o que, segundo a equação 4.1, perfaz uma taxa de saída do sinal serializado de 5,94 Gb/s.

Se por outro lado, se optar por um *datapath* interno de 2 byte, então serão necessários 2 *datapaths*, tal como ilustra o caso b) da figura 4.2. Isto implica que a taxa de amostragem nesses sejam o dobro da cadência de entrada dos sinais na interface com transmissor. Assim, mantém-se uma frequência de amostragem à entrada do transmissor de  $F_{TXUSRCLK2} = 148,5 \text{ MHz}$  e uma frequência do sinal de relógio de cada bloco PCS de  $F_{TXUSRCLK} = 297 \text{ MHz}$ . Ou seja, a largura interna de cada *datapath* é de 20 bits e por isso a taxa de saída do sinal serializado será igual a 5,94 Gb/s, segundo a equação 4.1.

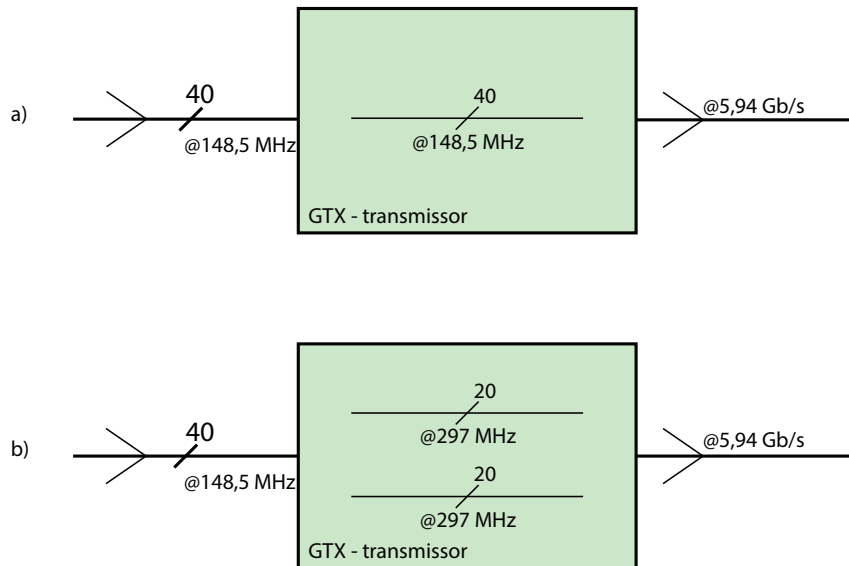


Figura 4.2: Exemplo simplificado de transmissão com diferentes usos de *datapath*

Estes dois sinais de relógio são gerados neste bloco e colocados nas saídas do mesmo para que na FPGA possa ser desenvolvida uma arquitetura que leia os sinais em paralelo para o transmissor à cadência de  $TXUSRCLK2$ . Estes estão alinhados pelo flanco positivo dos mesmos ainda que tenham frequências diferentes. Ambos são gerados com base no sinal de relógio de referência, segundo [13] e [27], através de multiplicadores ou divisores caso possuam frequências diferentes.

#### 4.1.2 Codificador 8B/10B

Este bloco permite que os sinais em paralelo sejam codificados de 8 bits para 10 bits quando está ativo. Por outras palavras, através da codificação estende os sinais de entrada

de 16 bits para 20 bits, os sinais de 32 bits para 40 bits e os sinais de 64 bits para 80 bits. É de notar que apesar de ser vantajosa a utilização de codificação num protocolo de transmissão em série, que a ativação deste bloco aumenta a latência do transmissor, tal como é de esperar.

As características deste tipo de codificação, as suas vantagens e importância foram já abordadas na subsecção com o título "Codificação dos sinais e sua importância" em 2.4.2 no capítulo 2.

### 4.1.3 Interface entre os diferentes domínios de sinal de relógio do transmissor

Já foi referido anteriormente que para o correto funcionamento do transmissor GTX opera com pelo menos dois sinais de relógio diferentes : TXUSRCLK and TXUSRCLK2. O segundo é o sinal de relógio de sincronização principal entre os dados provenientes da FPGA e o transmissor, e o primeiro é o sinal de relógio utilizado para a lógica do bloco PCS.

Contudo, o bloco PCS utiliza dois sinais de relógio internamente para o seu correto funcionamento: TXUSRCLK (como já foi mencionado) e ainda XCLK (sinal de relógio dos dados em paralelo do bloco PMA). A figura 4.3 ilustra os diferentes domínios de relógio presentes no transmissor GTX.

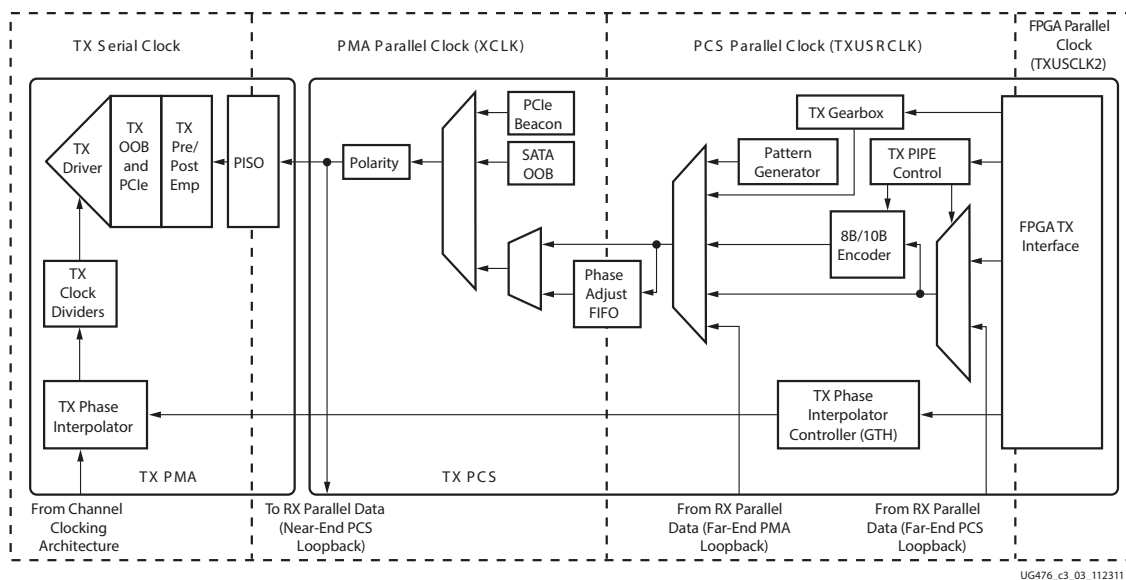


Figura 4.3: Diferentes domínios de sinal de relógio do transmissor (retirada de [13])

Quando os dados se cruzam entre os domínios TXUSRCLK e XCLK é necessário que para além de terem frequências semelhantes, as diferenças de fase que possam existir estejam resolvidas, e por isso existem dois blocos responsáveis por tal que apresentam os seus prós e contras: *buffer* ou bloco de alinhamento de fase. Independentemente da escolha de

um ou de outro, a utilização de um deles é obrigatória e torna-se importante conhecer as suas vantagens e desvantagens, pois tal escolha afeta o funcionamento global do sistema, segundo [13].

Por um lado, a utilização de um *buffer* é mais fácil e ainda assim é robusta, por outro o bloco de alinhamento de fase exige a utilização de mais lógica e restrições de sinais de relógio adicionais. Quando a latência se torna um ponto importante do sistema, então a utilização do *buffer* deve ser posta de lado, visto que o bloco de alinhamento de fase utiliza menos registos obtendo assim uma latência menor. Por outro lado, quando se utilizam várias transmissões em transdutores diferentes à mesma velocidade, o bloco de alinhamento de fase tem a vantagem de diminuir o atraso entre as mesmas. No entanto, isto é apenas uma curiosidade visto que tal não é utilizado no projeto.

#### 4.1.4 Interface com a camada física

Após a serialização dos dados no bloco PISO da figura 4.1 da página 60 os sinais são transmitidos para a camada física através de um *driver* reconfigurável. Esta interface dispõe de várias características que permitem manter a integridade do sinal, que passam de seguida a ser brevemente apresentadas:

- **Sinalização Diferencial:** para diminuir eventuais interferências no sinal durante a sua transmissão no cabo físico, tal como mencionado na subsecção com o título "Sinalização Física" em 2.4.2 no capítulo 2.
- **Pré-ênfase:** para de certa maneira preparar o sinal transmitido para um canal ruidoso, tal como referido na subsecção com o título "Interfaces com a camada física" em 2.4.2 no capítulo 2.
- **Resistências de interface com o cabo calibráveis:** para que a impedância característica da linha esteja adaptada com a interface, evitando assim eventuais reflexões dos sinais, tal como mencionado em na subsecção com o título "Restrições na utilização de circuitos de serializadores e deserializadores" em 2.4.2 do capítulo 2.

## 4.2 Recetor

Na figura 4.4 na página 65 é possível visualizar a arquitetura do recetor GTX. Este inclui diversos blocos que permitem a correta recuperação do sinal, sendo que o funcionamento dos principais já foram mencionados na subsecção 2.4.2 do capítulo 2, contudo passarão de seguida a ser brevemente descritos nas próximas subsecções.

### 4.2.1 Interface com a camada física

A interface o cabo físico do recetor possui duas características que se tornam importantes para a recuperação do sinal:

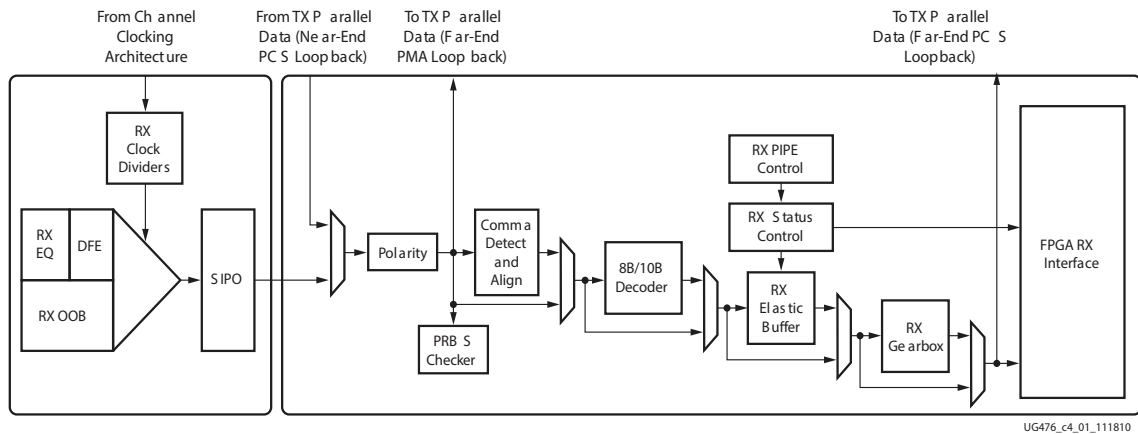


Figura 4.4: Arquitetura do recetor GTX, (retirada de [13])

- **Tensão de terminação configurável:** Esta característica é importante quando a referência do sinal diferencial é variável do lado do transmissor, o que permite recuperar bem diferencial do lado do transmissor.
- **Resistências de interface com o cabo configuráveis:** tal como acontecia do lado do transmissor, esta característica torna-se importante para que a linha de transmissão esteja adaptada ao recetor, evitando assim eventuais reflexões do sinal para a sua correta recuperação.

### 4.2.2 Equalização

Na subsecção com o nome de "Interfaces com a camada física" em 2.4.2 do capítulo 2 foi abordada a importância da utilização de filtros equalizadores para tentar compensar a atenuação e distorção que o sinal sofre durante a sua transmissão no canal físico. O recetor GTX disponibiliza dois tipos de equalizadores adaptativos que devem ser escolhidos consoante as necessidades de consumo de potência do circuito *versus* perdas do canal físico de transmissão.

Segundo [13], para canais cujas perdas não são muito significativas e o consumo de potência torna-se uma característica crítica do circuito existe um filtro adaptativo otimizado para tal com o nome de LPM (*Low-power mode*). Ainda segundo esta fonte o uso deste tipo de equalizador é recomendado para aplicações com débitos até 11,2 Gb/S de curto alcance, e com perdas por canal até 12 dB à frequência de *Nyquist*.

Para canais cujas perdas tornam-se significativas existe disponível um filtro adaptativo com o nome de *Decision Feedback Equalizer* (DFE). Segundo [13], este equalizador é utilizado para ligações de média distância cujas perdas do canal rondam os 8 dB ou mais à frequência de *Nyquist*. Para além disso, a utilização deste tipo de equalizador traz as seguintes vantagens adicionais:

- Efetua a equalização sem amplificação do ruído ou eventuais interferências.

- Pode também fazer correções de reflexões causadas pelas descontinuidades do canal.
- É vantajosa a sua utilização quando as interferências são preocupantes.

Tendo em conta as vantagens dos dois equalizadores disponíveis no recetor GTX, verifica-se que a utilização de cada um deles pode ser utilizada em circunstâncias diferentes: numa fase inicial pode ser utilizado um equalizador DFE, apesar de gastar mais energia, contudo a utilização de um equalizador LPM reduz o consumo. Todavia, numa fase mais avançada do projeto, que envolva a inserção dos sinais em ligações longas e/ou ruidosas, faz mais sentido usar um filtro DFE adaptado para tal.

### 4.2.3 Alinhamento de Palavras

O bloco de alinhamento de palavras é feito antes da deserialização dos sinais, isto porque é necessário definir os limites das palavras antes destes serem convertidos para dados em paralelo. O método de alinhamento de palavras, os caracteres especiais existentes para aplicação neste método e ainda a sua importância foram já abordados na subsecção com o título "Alinhamento da transmissão" em 2.4.2 no capítulo 2.

Nesta subsecção serão abordadas as características que este bloco do recetor possui e ainda como se pode tirar partido das mesmas aquando a sua utilização. O bloco possibilita a escolha da palavra de alinhamento, pode ser uma utilizada em protocolos bastantes conhecidos, já como referido anteriormente e até pode mesmo ser a combinação de duas palavras com uma determinada máscara. Contudo, por uma questão de simplificação, o projeto utiliza apenas uma palavra de alinhamento bem conhecida que será definida quando for apresentado desenvolvimento do mesmo.

Apesar de existirem *flags* disponibilizadas pelo bloco de alinhamento que indicam o estado do mesmo, o autor de [13] alerta que para uma linha de transmissão cuja taxa de débito seja superior a 5 Gb/s, o bloco pode alinhar a palavra falsamente e por isso ativar a *flag* que indica que a palavra está alinhada mesmo que não estando. Isto implica que seja desenvolvido e aplicado um sistema que verifique se a palavra se encontra alinhada ou não.

Este bloco dispõe de uma opção de alinhamento manual que vem substituir o automático, o que pode ser bastante útil em casos em que o transceptor alinha falsamente uma palavra. Na imagem 4.5 na página 67 é possível visualizar o exemplo de alinhamento manual das palavras no recetor que passa a ser brevemente descrito.

Este processo de alinhamento manual é conhecido pelo nome de "*RXSLIDE*" e é usado para mover a palavra em 1 bit. O sinal "*RXSLIDE*" deve estar ativo apenas um ciclo de relógio de *RXUSRCLK2* e de seguida deve ser inativo. Segundo, [13] deve-se esperar pelo menos 32 ciclos de relógio para que esta operação seja realizada outra vez.

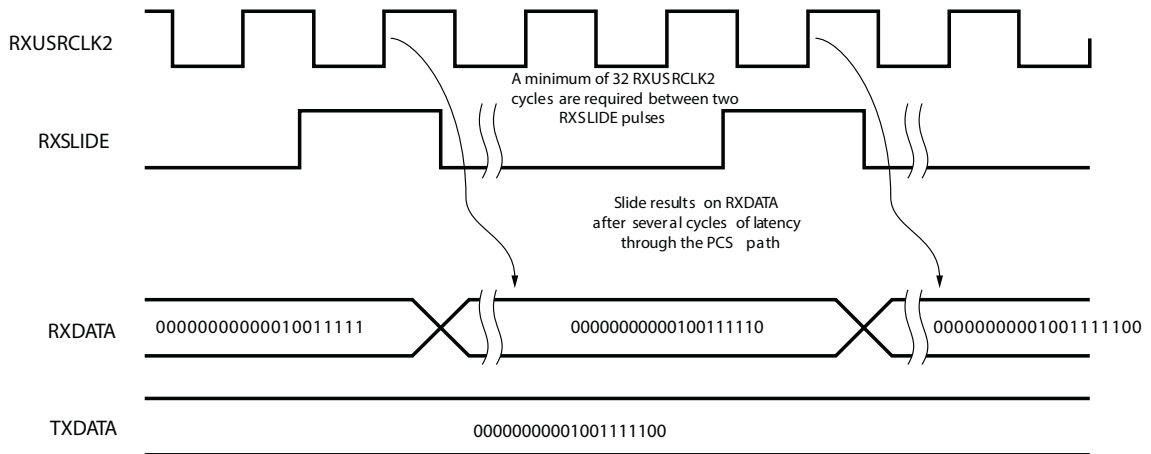


Figura 4.5: Exemplo de um alinhamento manual da palavra (retirada de [13])

#### 4.2.4 Descodificador 8B/10B

Tal como no transmissor, este bloco é responsável pela descodificação 8B/10B caso esta tenha sido realizada do lado do transmissor. Esta operação é realizada depois da conversão dos dados para paralelo e é de notar que a utilização deste bloco aumenta a latência de todo o sistema.

Numa fase inicial do projeto, a utilização deste bloco (tanto do lado do transmissor como do recetor) é posta de lado no sentido de simplificar o projeto e também de diminuir a latência do sistema.

#### 4.2.5 Interfaces entre os diferentes domínios de sinal relógio do recetor

Tal como acontecia no transmissor, o recetor opera a diferentes domínios de sinal de relógio. Particularmente no bloco PCS existem dois domínios de relógio para os dados em paralelo críticos: RXUSCLK e XCLK(o sinal de relógio dos dados em paralelo do bloco PMA). A imagem 4.6 ilustra os diferentes domínios de relógio presentes no recetor.

Quando os dados são transmitidos de um domínio para o outro todas as diferenças de fase entre os sinais de relógio devem estar resolvidas e a frequência não deve variara muito, e por isso deve-se recorrer ao uso de um dos dois seguintes blocos: *elastic buffer* ou bloco de alinhamento de fase. Ambos os blocos resolvem as diferenças de fase entre os dois domínios, no entanto apresentam as suas vantagens e desvantagens que serão de seguida abordadas para se concluir qual deve ser utilizado no projeto.

Por um lado, segundo [13], o *buffer* é uma estrutura robusta e de fácil operação, enquanto que o bloco de alinhamento de fase exige mais lógica e restrições relativamente às fontes de relógio. Por outro lado, o uso do *buffer* implica uma latência maior visto que o bloco de alinhamento de fase foi concebido para diminuir a latência total do sistema.

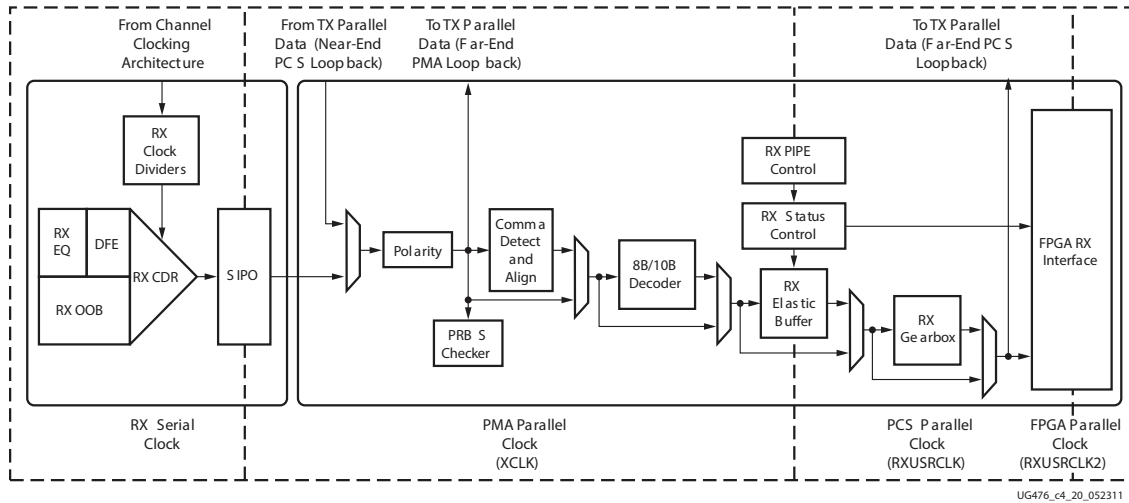


Figura 4.6: Diferentes domínios de sinal de relógio no recetor (retirada de [13])

Relativamente à inicialização, o *buffer* tem a capacidade de se inicializar imediatamente, enquanto que o circuito de alinhamento de fase tem de esperar que todos os sinais de relógio estabilizem para poder conseguir operar corretamente. Quanto à correção do sinal de relógio, o uso do *elastic buffer* é obrigatório, enquanto que quando se utiliza o bloco de alinhamento de palavras, este exige que a lógica de correção do sinal de relógio seja feita fora do transceptor.

Assim sendo, o uso do bloco de alinhamento torna-se útil quando a latência é um requisito crítico do sistema, todavia o uso do *buffer* não requer correção de sinal de relógio externa ao transceptor o que simplifica bastante o sistema. É uma boa opção numa fase inicial do projeto utilizar um *buffer* para fazer as correções de fase, pois, apesar de introduzir um pouco de latência no circuito, não exige lógica externa ao transceptor para correção do sinal de relógio.

#### 4.2.6 Interface com a FPGA

Este é o bloco responsável pela conexão entre os dados do recetor com o resto da FPGA. Toda a lógica desenvolvida na FPGA deve receber os dados provenientes do recetor pela porta "RXDATA" lendo-os ao flanco positivo do sinal de relógio "RXUSRCLK2". O tamanho desta porta depende de vários fatores internos do recetor, tal como acontecia no transmissor, como por exemplo se a codificação está ativa ou não e ainda qual o tamanho do *datapath* usado (2 ou 4 byte). As diferentes possibilidades de larguras da porta de RXDATA assemelham-se às da largura da porta de entrada do transmissor (TXDATA) e por isso é possível encontrá-las na tabela 4.1 na página 60.

Também à semelhança do transmissor existem dois sinais de relógio necessários para o correto funcionamento do recetor: RXUSRCLK e RXUSRCLK2. O segundo já foi mencionado como o sinal de relógio de sincronização entre o recetor e a lógica da FPGA, e o



sinal de relógio RXUSRCLK é o sinal interno para a lógica do bloco PCS (que lida com os dados em paralelo). O valor deste relógio depende da velocidade de transmissão do canal e do tamanho interno do *datapath* usados e é possível calcular o seu valor através da equação presente em 4.1 na página 61.

A relação entre estes dois sinais de relógio mencionados (RXUSRCLK e RXUSRCLK2), é bem definida e depende do tamanho da porta RXDATA e do tamanho do *datapath* usado. A relação entre estes dois sinais de relógio do recetor é idêntica à do transmissor e por isso é possível encontrar essas mesmas relações na tabela 4.2 na página 61.

Segundo [13], é de notar que quando é o mesmo oscilador a fornecer o sinal de relógio de referência para o transmissor e para o recetor, que para além dos sinais de relógio de saída do recetor serem gerados com base no sinal de relógio de referência, podem ainda ser usados os sinais de relógio de saída do transmissor para o recetor e vice-versa.

### 4.3 Análise das características de transmissão

Nesta secção são abordadas as diferentes características de transmissão possíveis de obter (velocidades de transmissão, tamanho de tramas entre outras) tendo em conta as restrições dos GTX, de maneira a otimizar a transmissão.

O projeto pode ser abordado de várias maneiras, desde a restringir a largura de banda base da transmissão até mesmo limitar o número de bits por tramas a transmitir. No sentido de simplificar o projeto e torná-lo eficiente em termos lógicos restringiu-se a cadência de entrada dos dados no transceptor para 148,5 MHz, pois assim não é necessário o uso de memórias, por exemplo FIFO, para armazenar dados antes da sua entrada nos transceptores.

Com esta restrição, e para se retirar algumas conclusões relativamente à otimização de transmissão dos dados em série, efetuaram-se cálculos para as diferentes larguras na porta de dados, com a utilização de codificação ou não através da equação apresentada em 4.1 da página 61. Esses resultados estão apresentados na tabela 4.3 na página 70.

Através de uma rápida observação da tabela 4.3, constata-se que fixando a taxa de amostragem do sinal à entrada do transmissor é possível obter diversas taxas de transmissão para diferentes larguras da porta da interface do transmissor com a FPGA. O projeto pode utilizar diferentes larguras de tramas a transmitir, consoante a transmissão que é realizada: uma transmissão apenas de imagem não necessita de transmitir tantos dados como uma transmissão de imagem e som, e por isso obtém-se diferentes velocidades de transmissão.

A escolha da largura das tramas a ser transmitido bem como a motivação para tal serão abordados aquando a apresentação de cada arquitetura transmitida.

Interface com a FPGA	Codificação 8B/10B	Tamanho do Datapath	Tamanho interno do datapath	$F_{TXUSRCLK}$	$F_{TXUSRCLK2}$	Velocidade em série
16	Sim	2 byte	20	148,5 MHz	148,5 MHz	2,97 Gb/s
	Não	2 byte	16	148,5 MHz	148,5 MHz	2,376 Gb/s
20	Não	2 byte	20	148,5 MHz	148,5 MHz	2,97 Gb/s
32	Sim	2 byte	20	297 MHz	148,5 MHz	5,94 Gb/s
	Não	2 byte	16	297 MHz	148,5 MHz	4,752 Gb/s
	Sim	4 byte	40	148,5 MHz	148,5 MHz	5,94 Gb/s
	Não	4 byte	32	148,5 MHz	148,5 MHz	4,752 Gb/s
40	Não	2 byte	20	297 MHz	148,5 MHz	5,94 Gb/s
	Não	4 byte	40	148,5 MHz	148,5 MHz	5,94 Gb/s
64	Sim	4 byte	40	297 MHz	148,5 MHz	11,88 Gb/s
	Não	4 byte	32	297 MHz	148,5 MHz	9,5 Gb/s
80	Não	4 byte	40	297 MHz	148,5 MHz	11,88 Gb/s

Tabela 4.3: Débitos de transmissão atingíveis para diferentes larguras de porta de entrada do transceptor

## 4.4 Estrutura do transceptor GTX

Esta secção apresenta a estrutura geral do IP disponibilizado pela *Xilinx* gerado no *software* VIVADO, mais em concreto todas as suas entradas e saídas, no sentido de se poder perceber todas as características do mesmo antes de se expor as arquiteturas desenvolvidas. Para gerar este módulo existe uma interface do *software* VIVADO que permite configurar as diversas características do transmissor e do recetor GTX apresentadas nas secções 4.1 e 4.2 respetivamente deste capítulo.

Na imagem 4.7 da página 71 visualiza-se a estrutura geral do IP GTX gerado pelo *software* VIVADO. O módulo de topo representado na imagem é constituído por diversos sub-módulos, tal como indica o manual da interface que gera este IP referenciado em [27]. Contudo, esses sub-módulos são de gestão interna do transceptor e como tal não são relevantes para o projeto. Por esse motivo, nesta imagem apenas são apresentadas as portas de entrada e saída de maior importância para o projeto, que passam de seguida a ser detalhadas:

- **GTREFCLK\_PAD\_N\_IN** e **GTREFCLK\_PAD\_P\_IN**: Entrada do sinal de relógio diferencial externo de referência.
- **SYSCLK\_IN**: Entrada do sinal de relógio a que o resto da arquitetura opera, ou seja, o sinal de relógio de todos os outros blocos da arquitetura que não o GTX.
- **RXN\_IN** e **RXP\_IN**: Par diferencial de entrada do recetor dos dados em série.
- **TXN\_OUT** e **TXP\_OUT**: Par diferencial de saída do transmissor dos dados em série.

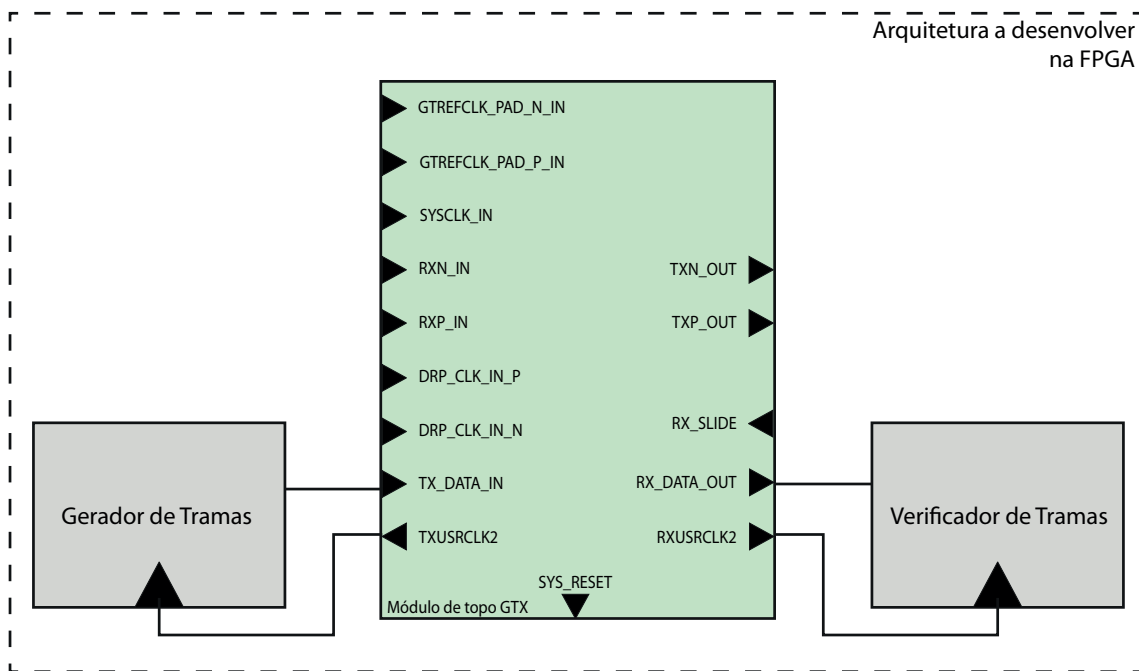


Figura 4.7: Estrutura geral do módulo GTX gerado no *software* VIVADO

- **DRP\_CLK\_IN\_P e DRP\_CLK\_IN\_N**: Entrada do par diferencial externo para a interface DRP (*Dynamic Reconfiguration Port*).
- **TX\_DATA\_IN**: Entrada dos dados em paralelo a serem serializados.
- **RX\_DATA\_OUT**: Saída dos dados em paralelo depois de recebidos e deserializados.
- **TXUSRCLK2**: Sinal de relógio de amostragem dos dados para o transmissor.
- **RXUSRCLK2**: Sinal de relógio de amostragem dos dados provenientes do recetor.
- **RXSLIDE**: Entrada do sinal que ativa o alinhamento manual do recetor.
- **SYS\_RESET**: Sinal de *reset* ativado pelas máquinas de estados do recetor e transmissor.

Através da observação da imagem 4.7 é possível ver que a entrada TX\_DATA\_IN e a saída RX\_DATA\_OUT estão conectadas a dois blocos: um bloco gerador de tramas a enviar para o transmissor e um bloco que verifica as tramas que chegam do recetor. O funcionamento destes blocos pode variar de arquitetura para arquitetura, contudo em todas estas blocos operarão aos sinais de relógio TXUSRCLK2 e RXUSRCLK2 e são obrigatórios para o correto funcionamento do sistema globalmente. Todos os outros sinais não estão diretamente conectados a nenhuma entrada nem saída porque podem variar entre as arquiteturas.

A interface DRP é uma característica do transceptor que até agora não foi abordada porque não é utilizada no projeto. Esta interface permite configurar dinamicamente algumas características dos transceptores, o que é útil em alguns projetos. Apesar de não ser utilizada neste projeto, é impossível desativar esta característica na interface do *software* que gera o módulo GTX, e por esse motivo está aqui brevemente apresentada.

É de notar ainda que a porta RXSLIDE apresentada na imagem 4.7 da página 71 é uma porta que pode não existir se assim se pretender. Todavia optou-se por ativar esta porta pois é esta que ativa o alinhamento manual das palavras recebidas, e tal como mencionado em 4.2.3 na página 66 deste mesmo capítulo, esta opção de alinhamento manual pode vir a ser útil para transmissões cuja taxa de é superior a 5 Gb/s. Através da análise da tabela 4.3 verifica-se que maior parte das taxas de débito calculadas para tramas com 32 bits ou mais são superiores a 5 Gb/s e como tal, é de esperar que tal venha a ser necessário.

## Capítulo 5

# Transmissão em série de dados HDMI

Neste capítulo são contempladas todas as arquiteturas desenvolvidas para a transmissão dos dados HDMI em série, explicando todas as decisões tomadas para se obter o produto final.

### 5.1 Abordagem inicial

Numa fase inicial do projeto optou-se por abordar de uma maneira simples a transmissão dos dados em série, sem o recurso à definição de todas as tramas de uma pacote. Tal decisão foi tomada, ciente da importância das tramas num protocolo de comunicação, pois o módulo GTX disponibilizado pela *Xilinx* é muito complexo e completo e foi necessário uma familiarização com o mesmo.

#### 5.1.1 Transmissão de uma barra de cores gerada na FPGA em série

A arquitetura desenvolvida passa por um conjunto de fases que vão desde a criação do módulo GTX através da interface disponibilizada no *software* para tal, e todas as tomadas de decisões que isso envolve, até a concepção de arquiteturas que criem e verifiquem as tramas. Todas essas fases serão devidamente explicadas nesta subsecção.

#### Considerações sobre a arquitetura

A arquitetura desenvolvida gera uma barra de cores em *FULL HD* na FPGA, tal como descrito em 3.2.1 no capítulo 3, a uma taxa de atualização vertical de 60 Hz. A placa HDMI transmissora utilizada nesta arquitetura deve receber imagens no formato RGB de 30 bits e por isso está programada para a configuração por omissão, referida em 3.1.1.1 na página 41. Para além disto, a arquitetura também tem um bloco que gera as tramas e envia-as para o GTX, do mesmo modo que tem um bloco que recebe as tramas e retira a

informação das mesmas, enviando-as para uma arquitetura que envie esses mesmos dados para a placa HDMI transmissora. Este diagrama geral da arquitetura esta representado na figura 5.1.

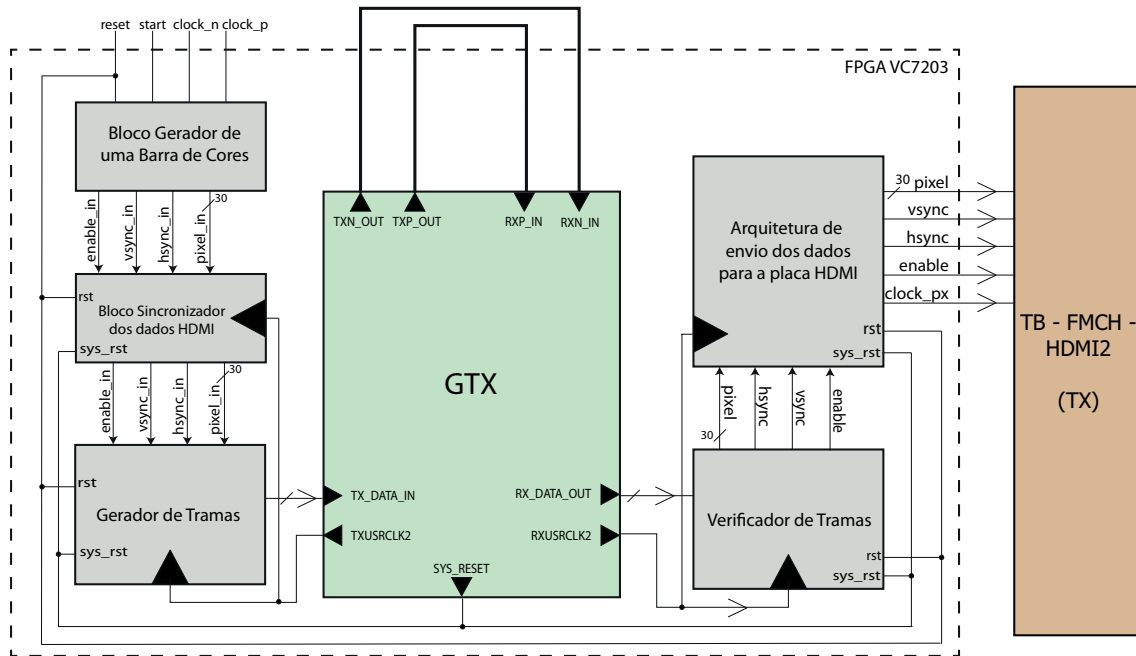


Figura 5.1: Diagrama de blocos geral da arquitetura que transmite uma barra de cores em série

Tal como é de esperar, e foi referido em 2.6 na página 35, os sinais de relógio provenientes do GTX e do módulo que gera a barra de cores não são o mesmo, e por isso, existe um bloco entre a geração e a criação de tramas que permite a sincronização entre os diferentes domínios de sinais de relógio através de *shift-registers*.

Tendo em conta a arquitetura global desenvolvida, foram tomadas 2 decisões importantes para definir as características do transceptor: o número de bits por trama e a frequência a que estas serão lidas para o mesmo. Tal como já foi referido, a maneira mais eficiente no que toca à escolha da frequência de amostragem para o transceptor, é escolher a própria frequência da imagem HDMI, ou seja, 148,5 MHz para uma imagem *FULL HD*. Relativamente ao número de bits por trama, tendo em conta que é necessário enviar os sinais todos (*pixel*, *vsync*, *hsync* e *enable*) e visto que nesta fase do projeto não houve a preocupação da criação de diferentes tramas para os diversos momentos de transmissão, escolheu-se enviar tramas de 40 bits.

Observando novamente a tabela 4.3 na página 70, independentemente do tamanho do *datapath* escolhido, obtém-se uma taxa de transmissão de dados de 5,94 Gb/s.

## Geração do Módulo GTX

Quando se gera o módulo GTX através da interface disponibiliza pelo *software* VIVADO existem algumas decisões que necessitam de ser tomadas para além das que já foram. Essas passam de seguida a ser detalhadas:

Do lado do transmissor tomaram-se as seguintes principais decisões:

1. **Tamanho interno dos dados:** Esta escolha envolve o número de *datapath* que são utilizados e ainda o valor da frequência de TXUSRCLK. Foi selecionado 40 bits o que implica o uso de apenas um *datapath* e obtém-se as frequências de TXUSRCLK E TXUSRCLK2 idênticas.
2. **Tipo de codificação:** Neste caso não se escolheu codificação porque não é possível (a interface com a FPGA é de 40 bits) e também numa fase inicial optou-se por simplificar o projeto.
3. **Escolha ente *Buffer* ou Bloco de Alinhamento de Fase:** Tal como indicado na secção 4.1.3 foi escolhido a utilização do *buffer* uma vez que é de mais fácil utilização não requerendo o uso de lógica extra (comparativamente ao bloco de alinhamento de fase) e ainda assim é robusto.

Do lado do recetor as principais decisões tomadas foram as seguintes:

1. **Tipo de equalização:** Na subsecção 4.2.2 são apresentadas as vantagens de cada um dos tipos de equalizadores disponíveis. Apesar de este ser um projeto simples em que não se pretende inserir o sinal em canais ruidosos, optou-se por utilizar um equalizador DFE uma vez que traz mais vantagens do a utilização do equalizador LPM.
2. **Alinhamento de palavras:** Como palavra de alinhamento escolheu-se o caractere K28.3 da tabela 2.5 na página 23. Contudo é necessário ter em conta que não se está a utilizar codificação e, como tal, para efetuar o alinhamento da palavra o recetor não alinhará pelo caractere K28.3 codificado, mas sim, não codificado. Ou seja, na realidade quando encontrar a palavra "7C" assume que é a palavra de alinhamento e alinha a trama para esse limite. Para além disso, optou-se por ativar a porta "RXSLIDE" que ativa o alinhamento manual dos bits, pois tal como referido em 4.2.3 é de esperar que seja necessário ativar o alinhamento manual para ligações cuja taxa de débito seja superior a 5 Gb/s.
3. **Tipo de descodificação:** Não foi utilizada nenhuma descodificação, pois do lado do transmissor também não há codificação.
4. **Escolha ente *Buffer* ou Bloco de Alinhamento de Fase:** Optou-se pela escolha do *buffer*, porque no caso do recetor para além não requerer lógica extra e ter uma

inicialização mais rápida (comparativamente ao bloco de alinhamento de fase), também não exige que a correcção do sinal de relógio seja realizada fora do transceptor, tal como mencionado em 4.2.5.

Na tabela 5.1 é apresentada uma tabela sumário do módulo gerado. Esta tabela foi adaptada da interface que gera o módulo.

Característica	GT
<b>TX Line Rate (Gb/s)</b>	5,94
<b>TX Reference Clock (MHz)</b>	148,5
<b>Encoding</b>	None
<b>TX Internal Data Width</b>	40
<b>TX External Data Width</b>	40
<b>TXUSRCLK (MHz)</b>	148,5
<b>TXUSRCLK2 (MHz)</b>	148,5
<b>TX Buffer Enabled</b>	TRUE
<b>RX Line Rate (Gb/s)</b>	5,94
<b>RX Reference Clock (MHz)</b>	148,5
<b>Decoding</b>	None
<b>RX Internal Data Width</b>	40
<b>RX External Data Width</b>	40
<b>RXUSRCLK (MHz)</b>	148,5
<b>RXUSRCLK2 (MHz)</b>	148,5
<b>RX Buffer Enabled</b>	TRUE

Tabela 5.1: Sumário do módulo GTX gerado para a transmissão de uma barra de cores gerada na FPGA (adaptada do *software*)

#### 5.1.1.1 Conceção e Desenvolvimento

Na imagem 5.1 da página 74 é representado um protótipo da arquitetura geral apresentada. Nesta subsecção são apresentados com detalhe todos os blocos utilizados e suas entradas e saídas para que se possa perceber as ligações entre os mesmos.

#### Bloco gerador de uma Barra de Cores

Este bloco gerador de barra de cores é exatamente igual ao bloco descrito em 3.2.1 na página 44: gera uma barra de cores em *FULL HD* com uma frequência de 148,5 MHz. Nas suas entradas encontram-se as portas vindas diretamente do exterior, tal como nas outras arquiteturas que utilizam este bloco, que são botões definidos pelo utilizador (*reset* e *start*) e também um sinal de relógio diferencial de 200 MHz. Este bloco coloca nas suas saídas sinais que são enviados para um bloco sincronizador dos dados HDMI que é de seguida detalhado.



### Bloco Sincronizador dos dados HDMI

O recurso a este bloco sincronizador dos dados HDMI é necessário devido aos diferentes domínios de relógio que existem no sistema: por um lado existe um bloco que gera uma barra de cores a uma determinada cadência, e por outro existe um bloco que gera as tramas a serem enviadas para o transceptor a outra cadência. Estes dois sinais de relógio podem ser iguais, no entanto podem não estar em fase o que é suficiente para haver problemas de meta-estabilidade.

Por este motivo, este bloco de sincronização é apenas constituído por dois registos de deslocamento (*shift-registers*) para que problemas de sincronização que possam eventualmente existir sejam resolvidos, tal como apresentado na figura 5.2.

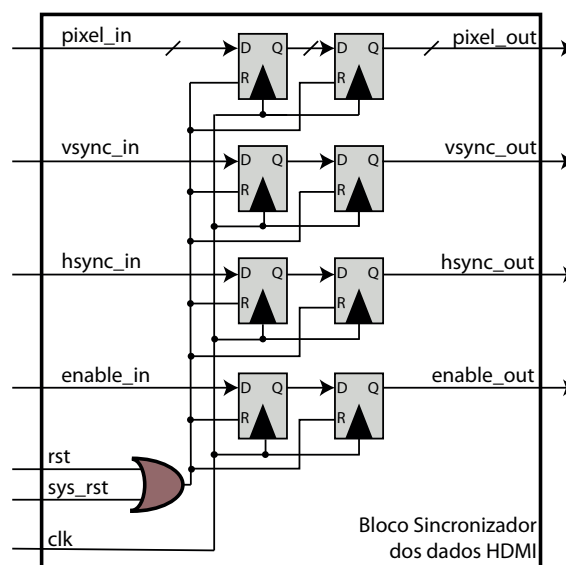


Figura 5.2: Bloco de sincronização de dados

Tal como se visualiza na figura, para além de nas suas entradas se encontrar os dados provenientes da fonte HDMI (que nesta arquitetura é o bloco gerador de barra de cores), há também dois sinais : *rst* e *sys\_rst*. O primeiro sinal refere-se ao sinal de *reset* controlado pelo utilizador e o segundo corresponde a um sinal de *reset* ativado automaticamente pelo módulo GTX.

O sinal de relógio a que opera este bloco trata-se do sinal de relógio proveniente do módulo GTX, tal como se visualiza na figura 5.1 na página 74.

### Gerador de Tramas

Este bloco é responsável pela criação das tramas de 40 bits a serem enviadas para o módulo GTX. Relembra-se que nesta abordagem inicial do projeto não se teve em conta a criação de tramas bem definidas para todos os momentos de transição, apenas se definiu a

trama que corresponde ao início de transmissão para que fosse possível do lado do recetor manter o alinhamento das tramas.

A trama que define o início da transmissão (SOP - *Start of Packet*) é uma trama que serve essencialmente para o alinhamento das palavras do lado do recetor. Ou seja, são aproveitados momentos de transmissão nulos para enviar esta trama e alinhar a mesma. Isto porque neste momento se está a lidar com uma taxa de transmissão superior a 5 Gb/s e por isso é de esperar que é necessário do lado do recetor verificar se as tramas estão realmente alinhadas. Esta trama SOP são 40 bits que em hexadecimal corresponde a: 000605047c. Para além de ter sido a trama de SOP sugerida pelo exemplo que a *Xilinx* disponibiliza, esta trama tem as seguintes particularidades:

1. No final da mesma se encontra a palavra de alinhamento(são os primeiros bits a ser transmitidos), que permite ao recetor alinhar a trama internamente.
2. A trama numa vista geral nunca será encontrada no meio de outros dados, que será de seguida explicado

Esta trama é transmitida sempre que os sinais de controlo se encontram inativos, ou seja que as seguintes condições se verificarem:

1.  $vsync = 0$
2.  $hsync = 0$
3.  $enable = 0$

Quando estas três condições não se verificam em conjunto, então é porque há algum sinal de imagem ativo, e por isso é transmitida uma trama com o formato que se visualiza na figura 5.3.

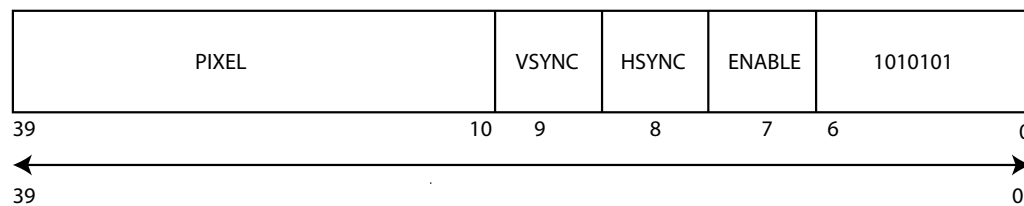


Figura 5.3: Estrutura das tramas geradas

Este formato de trama vem validar aquilo que foi dito anteriormente quando se justificou o formato da trama SOP: os últimos 7 bits da trama SOP são 1111100, enquanto que os últimos 7 bits de todas as outras tramas são 1010101. Portanto, em conjunto com o bloco verificador de tramas que garante que as tramas estão sempre alinhadas, dificilmente a trama SOP será encontrada no meio dos dados transmitidos enquanto dados da transmissão e não como trama de alinhamento.

Assim sendo, consoante os dados que recebe nas suas entradas (*pixel*, *vsync*, *hsync* e *enable*), este bloco ou envia SOP ou envia tramas no formato da imagem 5.3. A imagem 5.4 exemplifica os diversos momentos de transmissão das diferentes tramas numa imagem. A cinza correspondem os momentos da imagem em que os sinais de controlo estão inativos, e por isso são transmitidos SOP, e a castanho correspondem os momentos de transmissão em que algum dos sinais de controlo está ativo e por isso são enviadas tramas no formato previamente exemplificado. Sempre o sinal *rst* ou *sys\_rst* estiver ativo então os dados são repostos e as tramas enviadas são SOP.

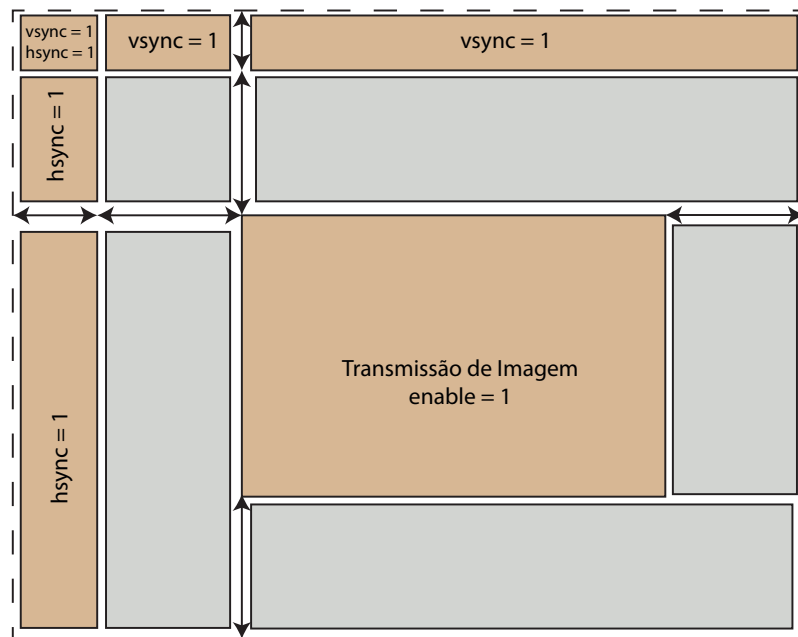


Figura 5.4: Momentos de transmissão de diferentes tramas

É de esperar que a transmissão demore algum tempo a alinhar do lado do recetor, e por isso, é necessário que as tramas SOP sejam enviadas durante algum tempo, por isso se escolheram os momentos de transição de dados de controlo nulos, pois são suficientemente longos para que as tramas possam ser alinhadas. Assim, no pior dos casos, perde-se a primeira imagem completa que é transmitida, pois não são recebidos os primeiros dados de controlo (*vsync* e *hsync*) mas garante-se que todas as outras estão alinhadas e os dados serão corretamente recebidos do lado do transmissor.

### Verificador de Tramas

Este bloco é responsável pela re-organização das tramas recebidas que são recebidas à cadência RXUSRCLK2. Apesar de poder parecer que com o alinhamento interno as tramas chegariam ao recetor tal como são enviadas do transmissor, mesmo com o alinhamento interno ideal, tal não acontece. As tramas são alinhadas para um determinado limite

assim que o recetor encontra as palavra de alinhamento, no entanto não significa que a trama recebida é exatamente à trama enviada.

O que o alinhamento interno faz, segundo [27] e [13], quando deteta a palavra de alinhamento é assumir que dessa palavra para a frente são dados válidos, tal como explicado anteriormente, e alinhar os dados para um determinado limite. Esse limite pode ser selecionado aquando a criação do módulo na interface específica para tal, e o que foi escolhido é que alinha os dados para o byte mais próximo. Apesar de não se estar a lidar com codificação, o bloco de alinhamento das palavras considera que 1 byte são 10 bits (pois com a codificação 8B/10B codifica 8 bits em 10 bits) porque o *datapath* é de 40 bits, e por isso quando deteta a palavra de alinhamento, as tramas recebidas no recetor podem chegar de 4 maneiras diferentes ilustradas na imagem 5.5.

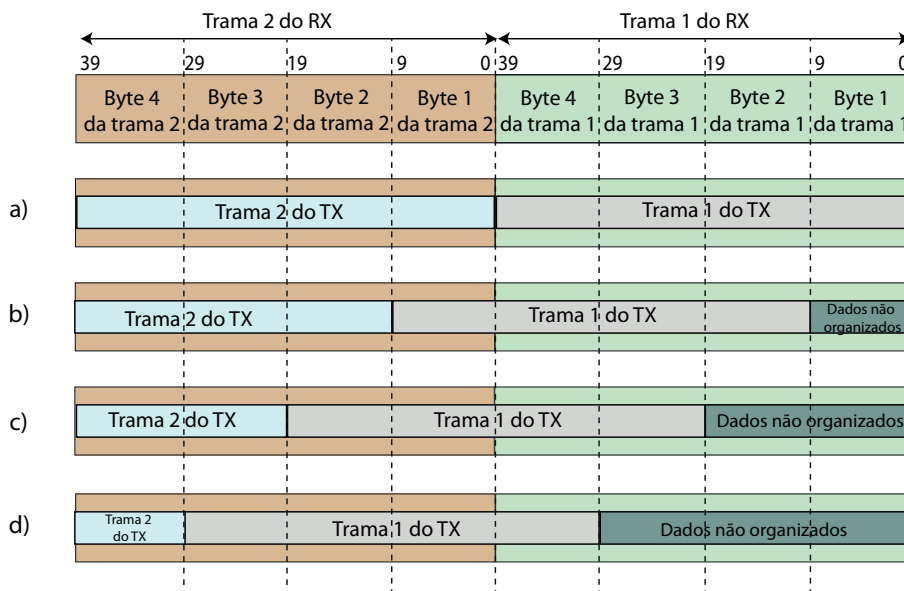


Figura 5.5: Alinhamento das tramas no transceptor

Daí a importância da trama de SOP: quando é detetada em algum dos limites assinalados a cinzento dá-se início à transmissão dos restantes dados. Para se proceder à organização dos dados recebidos é necessário recorrer a uma máquina de estados. Esta máquina de estados utilizada neste exemplo foi adaptada de uma disponibilizada pela *Xilinx*.

A máquina de estados principal é dividida em três por questões de simplificação e de melhor entendimento da mesma. Cada uma delas passará a ser detalhada, sem nunca esquecer que funcionam como um todo e apenas foram divididas para explicação do sistema global.

### 1. Máquina de estados geral da recepção de dados:

Esta máquina define o estado geral da recepção de dados. Estes estados podem ser três e passam de seguida a ser descritos:

- **Begin:** Estado inicial que está à espera da receção de dados válidos do recetor. Quando a trama de SOP é detetada pela primeira vez, então a máquina transita para um estado em que recebe dados válidos.
- **Track\_data:** Neste estado já foi detetado o início da transmissão (SOP) e por isso todos os dados que estão a ser recebidos são válidos. A transmissão encontra-se "à procura de dados". Quando é detetado algum erro (com ajuda de uma máquina de estados que deteta os erros nos dados recebidos) transita de estado.
- **Data\_error\_detected:** Estado de detecção de erro que transita imediatamente para o estado inicial para aguardar a chegada de dados válidos novamente.

Sempre que for necessário fazer *reset* ao sistema, quer pela ativação do utilizador ou então pelo transcetor volta-se ao estado inicial "begin".

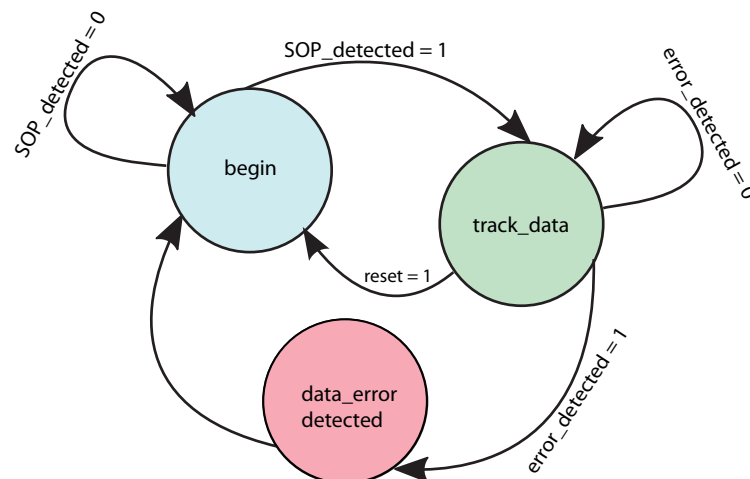


Figura 5.6: Máquina de estados de transmissão

A figura 5.6 na página 81 representa a máquina de estados detalhada anteriormente. De notar que na imagem estão representadas algumas *flags* que decidem de transição, mas que não são mais do que já foi explicado. A *flag* "SOP\_detected" indica que se a trama de início de pacote foi encontrada nos dados recebidos e "error\_detected" indica que foi detetado um erro nos dados recebidos. Ambas são definidas por máquinas de estados que serão explicadas de seguida.

## 2. Máquina de leitura de dados

Esta máquina de estados é responsável pela constante procura de dados recebidos pelo recetor. Isto é, os dados vão sendo recebidos e guardados em registos que guardam as duas últimas tramas chegadas do recetor. Isto porque na combinação dessas duas tramas pode-se encontrar a indicação que dá início à transmissão: a

trama SOP. Essas duas últimas tramas vão sendo constantemente verificadas sendo o seu conteúdo a motivação para a transição de estados.

A figura 5.7 da página 82 ilustra a máquina de estados referida anteriormente. Contudo antes de descrever a máquina de estados, faz-se uma breve descrição das *flags* de transição de estados presentes na figura:

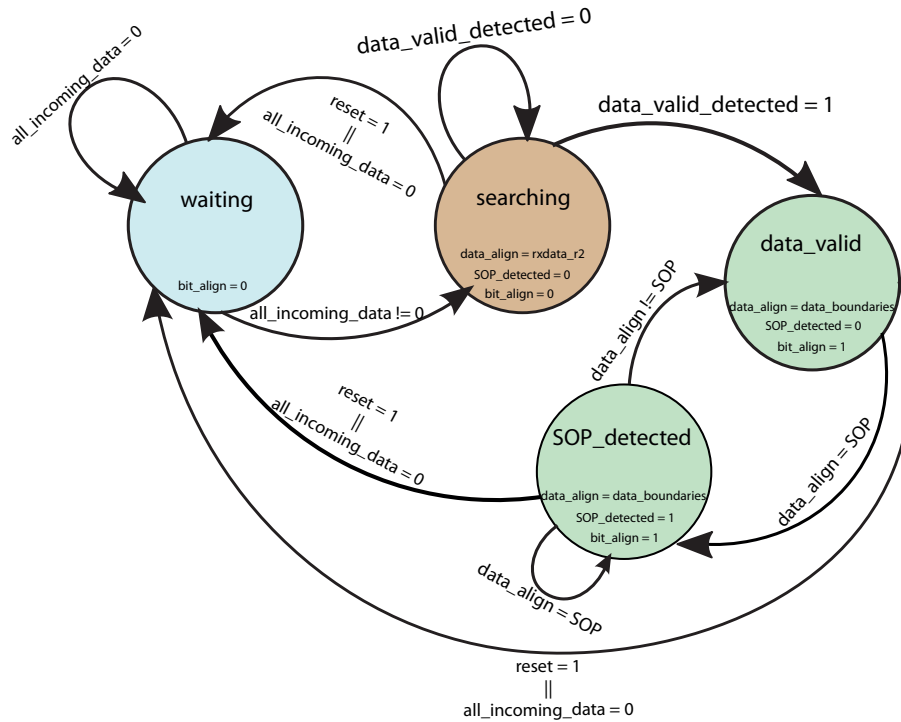


Figura 5.7: Máquina de estados leitura de dados

- *all\_incoming\_data*: Esta *flag* indica se todos os bits presentes nas duas últimas tramas recebidas no recetor são 0 ou não. Se sim, então a *flag* é igual zero, senão é um.
- *data\_valid\_detected*: Esta *flag* indica se foi encontrado nos dados recebidos (nas duas últimas tramas) a trama SOP em alguma das situações assinaladas a cinza presentes na figura 5.5 na página 80.
- *reset*: indica se foi ativado o sinal para repôr os dados originais da máquina de estados.

Estas são as principais flags de transição de estados da máquina, de seguida serão detalhados todos os estados da mesma:

- **Waiting:** Este estado indica que o recetor ainda não está a receber dados, isto porque a *flag all\_incoming\_data* está inativa. Por defeito quando não há transmissão de dados do lado do transmissor, o recetor recebe apenas dados

iguais a zero. Assim que esta *flag* se ativa então procede-se para o estado de procura da trama de início de transmissão.

- **Searching:** Neste estado, a máquina procura pela trama que dá início à transmissão de dados (SOP). Uma vez que ela pode vir alinhada em diferentes bytes nas tramas do recetor, então a máquina procura por SOP nas quatro situações diferentes apresentadas na figura 5.5 na página 80. Assim que encontra SOP, memoriza os limites das tramas em que foi esta foi encontrada para que possa de seguida retirar todos os outros dados de transmissão e transita de estado.
- **Data\_valid:** Este estado serve essencialmente para guardar os dados devidamente alinhados, uma vez que já foi dado o início de transmissão, segundo os limites da trama SOP. Ativa-se a flag *bit\_align* para que o sistema reconheça que os dados se encontram alinhados. Se os dados que estão a ser guardados em *data\_align* corresponderem à trama SOP então transita-se de estado para indicar que esta trama foi detetada.
- **SOP\_detected:** Quando este estado está ativo, então os dados alinhados correspondem à trama SOP, e por isso deve-se ativar a flag "*SOP\_detected*". Este estado mantém-se ativo enquanto os dados alinhados corresponderem à trama SOP e transita para o estado "*data\_valid*" assim que tal deixar de ser verdade.

Sempre que transmissão for interrompida (*all\_incoming\_data* se iguala a zero) ou o sinal de *reset* for ativo (quer por opção do utilizador ou pelo módulo GTX), então a máquina retorna ao estado inicial e repõe os dados originais do sistema.

Inicialmente o estado ativo é "*waiting*", e todas as *flags* de decisão de mudança de estado (*all\_incoming\_data* e *data\_valid\_detected*) estão igualadas a zero.

### 3. Máquina de estados de verificação do alinhamento dos dados

Tal como já foi mencionado várias vezes ao longo do documento, para taxas de transmissão superiores a 5 Gb/s, o transceptor pode alinhar falsamente os dados. Por esse motivo, as tramas recebidas podem não vir dentro dos limites apresentados na figura 5.5 na página 80, o que leva ao uso de uma máquina de estados que verifique tal.

A máquina de estados apresentada na figura 5.8 na página 84 é responsável pela detecção do devido alinhamento das tramas recebidas de acordo com a figura 5.5 na página 80. Caso tal não se verifique, então é também responsável pelo alinhamento manual das tramas.

Antes de se passar à descrição de cada estado, é de notar que existem algumas *flags* de decisão de transição de estados:

- *bit\_align*: Esta *flag* indica se a palavra de se encontra devidamente alinhada, tal como ilustrada na figura 5.5 na página 80.

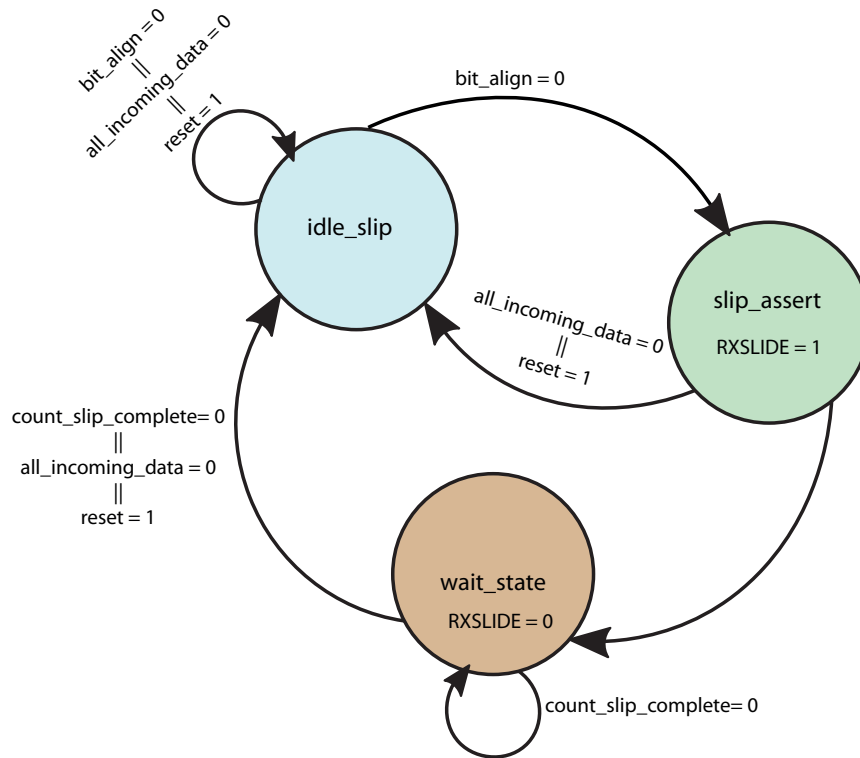


Figura 5.8: Máquina de estados de verificação do alinhamento dos dados

- *all\_incoming\_data*: Esta *flag* já foi mencionada e detalhada na máquina de estado que faz a leitura de dados.
- *count\_slip\_complete*: Esta é uma *flag* que fica ativa quando um contador termina (será explicado mais à frente).
- *reset*: Indica a reposição dos dados originais da máquina de estados quando ativa.

Esta máquina funciona apenas com a utilização de três estados:

- **Idle\_slip**: Este estado é essencialmente uma espera pela chegada de dados e também pela detecção do desalinhamento da palavra. Há transição de estado quando se verifica que as tramas que chegam ao recetor não estão alinhadas de acordo com os limites que deveriam estar.
- **Slip\_assert**: Este estado está ativo quando há detecção de falso alinhamento de palavras, e por isso ativa-se a saída que liga ao GTX: RXSLIDE. A ativação deste sinal vai permitir que as tramas sejam deslocadas em 1 bit,. Este estado transita imediatamente para o estado de *Wait\_state*.
- **Wait\_state**: Neste estado o sinal de saída RXSLIDE volta a estar inativo e aguarda-se que a operação de deslocamento de 1 bit se verifique. Como estamos



perante um *datapath* de 40 bits, aguarda-se 64 ciclos de relógio. Após esta espera, a flag *count\_slip\_complete* fica ativa e há transição de estado.

De notar que sempre que houver falha de transmissão ou o sinal de *reset* for ativo, então os sinais são repostos para os originais e volta-se ao estado *Idle\_slip*.

Inicialmente o estado ativo é o *Idle\_slip* e as *flags* de sinalização de transição de estado são igualadas a 0.

Para além destas máquinas de estados apresentadas, existe a possibilidade de implementação de uma outra que verifique os erros das tramas chegadas. Esta máquina torna-se útil quando há a implementação de códigos detetores de erros nas tramas, ativando a flag "*error\_detected*" sempre que o *checksum* recebido na trama não corresponde à mensagem. Contudo, nesta abordagem inicial tal não é realizado e por isso essa *flag* é globalmente igualada a zero.

### Bloco de envio de dados para a placa HDMI

Este bloco é responsável pela recepção dos dados alinhados provenientes do bloco verificador de tramas e o seu posterior envio para a placa HDMI transmissora. A imagem 5.9 na página 85 o diagrama de blocos deste módulo.

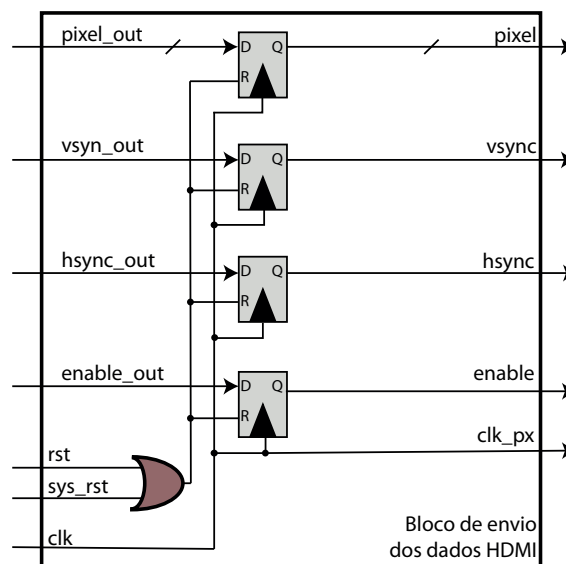


Figura 5.9: Diagrama de blocos de envio de dados para a placa HDMI transmissora

Os dados provenientes do bloco anterior são lidos para um registo ao flanco positivo do sinal de relógio que alimenta este módulo (RXUSRCLK2). De seguida, todos os dados presentes nos registos são enviados para a placa HDMI transmissora, incluindo o sinal de relógio RXUSRCLK2

## Localizações das portas de saída do módulo de topo

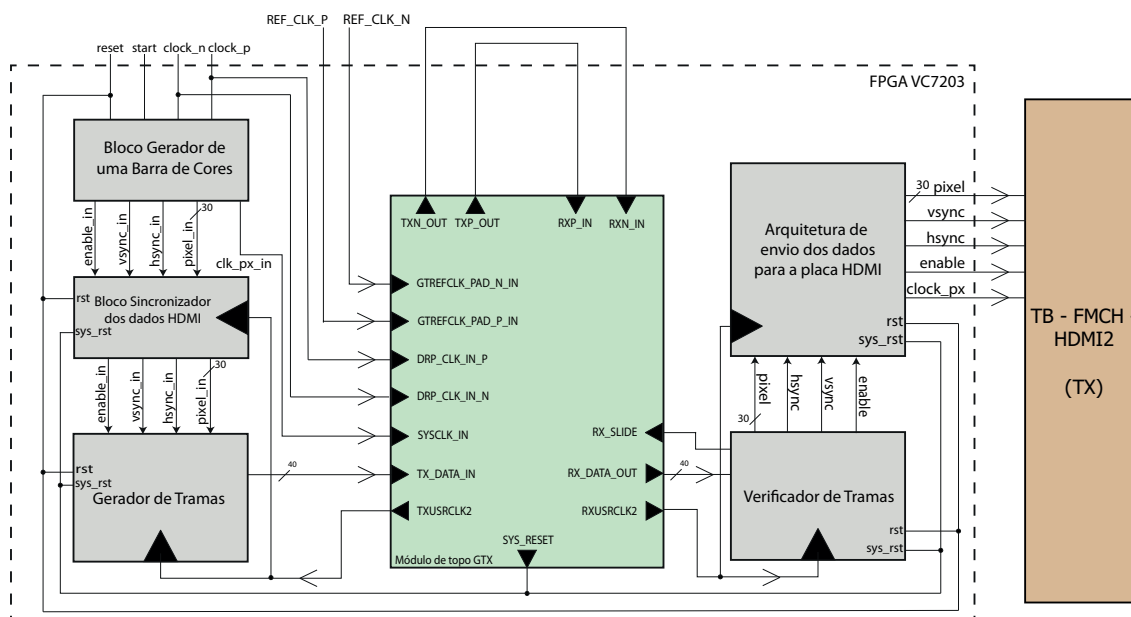


Figura 5.10: Diagrama de blocos da arquitetura de transmissão em série de uma barra de cores gerada na FPGA

## Resultados

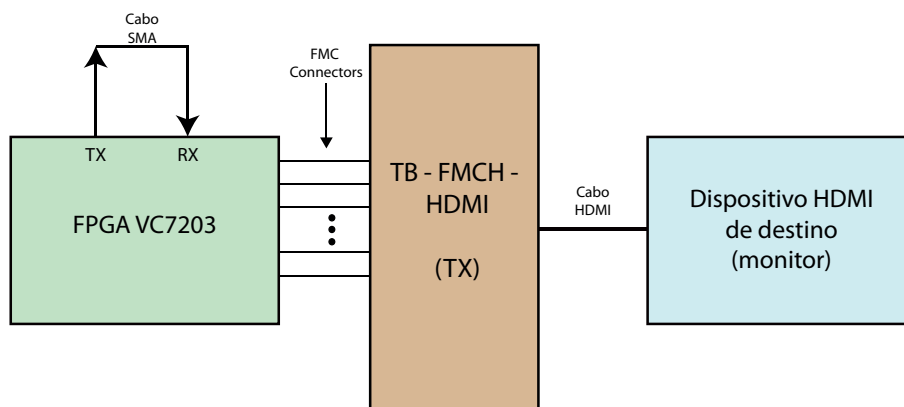


Figura 5.11: Setup de teste

## Capítulo 6

# Conclusões e Trabalho Futuro

### 6.1 Qualidade dos resultados obtidos

- > Parte do HDMI está boa
- > Parte da serialização podia estar melhor

### 6.2 Principais dificuldades encontradas

- > Tempos de simulação comportamental, pos síntese, pos implementação elevados.
- > Tempos de síntese e implementação elevados
- > Informação pouco clara sobre o GTX
- > Muito tempo perdido a perceber os módulos GTX

### 6.3 Conclusões do projeto desenvolvido

- > Parte HDMI está toda ok !
- > Efetivamente conseguiu-se uma transmissão em série, com um pequeno erro que pode ser melhorado com algum tempo através de técnicas de amostragem a velocidades mais altas e isso.
- > Ou entao tbm pode ser corrigido através da criação de um pacote de comunicação robusto
- > No fundo e um bom trabalho pq vai ser incluído num maior e vem testar cenas que vao dar jeito num futuro bastante proximo

### 6.4 Trabalho futuro

- > Tornar a parte de serialização mais robusta
- > Implementar códigos detetores de erros
- > Aproveitar os diferentes canais dos transdutores

-> Passar o som

# Bibliografia

- [1] Wikipedia Contributors, “HDMI,” 2016.
- [2] 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.
- [3] J. Federici and L. Moeller, “Review of terahertz and subterahertz wireless communications,” *Journal of Applied Physics*, vol. 107, no. 11, 2010.
- [4] W. contributors, “Audio and video interfaces and connectors,” 2016.
- [5] Xilinx and Inc, *VC7203 Virtex-7 FPGA GTX Transceiver Characterization Board User Guide (UG957)*. 2014.
- [6] Xilinx, “I/O Design Flexibility with the FPGA Mezzanine Card (FMC),” vol. 315, 2009.
- [7] Inrevium, *Manual do Utilizador de TB-FMCH-HDMI2 Hardware*. 2014.
- [8] A. Devices, *ADV7612 Reference Manual*.
- [9] Analog Devices, *ADV7511 Programming Guide*. No. March, 2012.
- [10] Analog Devices, *ADV7511 Hardware user’s guide*. No. July, 2011.
- [11] A. Athavale and C. Christensen, *High-Speed Serial I/O Made Simple*. 2005.
- [12] D. Chen, “SerDes Transceivers for High-speed Serial Communications,”
- [13] Xilinx and Inc, *7 Series FPGAs GTX/GTH Transceivers User Guide (UG476)*. 2016.
- [14] E. Tutorials, “The Shift Register.”
- [15] 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,”

- [16] T. Instruments, *LVDS Owner's Manual*. 2008.
- [17] R. Williams, "A Painless Guide To CRC Error Detection Algorithms," *Rocksoft Pty Ltd.*, p. 34, 1993.
- [18] Xilinx and Inc, "Xilinx WP431 Leveraging 7 Series FPGA Transceivers for High-Speed Serial I/O Connectivity, White Paper," 2013.
- [19] C. E. Cummings, "Clock Domain Crossing ( CDC ) Design & Verification Techniques Using SystemVerilog," *Snug-2008*, 2008.
- [20] Altera, "Understanding Metastability in FPGAs," *White Paper*, no. July, pp. 1–6, 2009.
- [21] Analog Devices, *Data Sheet ADV7612*.
- [22] Analog Devices, "Data Sheet ADV7511."
- [23] Xilinx, "Platform Flash In-System Programmable Configuration PROMs," 2016.
- [24] Xilinx, "Xilinx Platform Cable USB II," 2015.
- [25] Inrevium, *TB-FMCH-HDMI2 Hardware User Manual 1 IN / OUT + Audio support*. 2014.
- [26] Philips Semiconductors, *I2S bus specification*, 1996.
- [27] Xilinx, *7 series FPGAs Transceivers Wizard v3.5*. 2015.
- [28] Inrevium, *TB-FMCH-HDMI2 Hardware User Manual 2 Ch IN / OUT support*. 2015.

## Anexo A

# Descrição das portas das placas HDMI

Este anexo aborda detalhadamente as conexões que as placas HDMI têm disponíveis para conexão com a FPGA consoante a configuração das mesmas. São apresentados todos os pinos que transmitem dados, qual o nome da ligação desses pinos nas placas HDMI e ainda a descrição dos sinais.

### A.1 Configuração por omissão

Esta secção apresenta todas as portas das placas HDMI configuradas por omissão (tanto recetora como transmissora) que transmitem dados que são utilizados no projeto. De mencionar que esta configuração suporta dois canais (0 e 1), no entanto apenas são apresentadas as portas relativas ao canal 0 pois é o único utilizado no projeto.

### A.2 Suporte de um canal de imagem e áudio

Nesta secção são apresentadas todas as portas das placas HDMI configuradas para terem suporte de imagem e áudio num canal que enviam dados para a FPGA utilizados no projeto. São apresentados os nomes das portas nas placas, os nomes dos sinais tanto do lado do recetor como do transmissor e a sua descrição.

### A.3 Suporte de dois canais de imagem melhorado

Nesta secção são apresentados os dados enviados pelas placas HDMI configuradas para a versão de suporte de imagem em dois canais melhorada. Para cada porta são apresentados os nomes dos sinais tanto na placa transmissora como recetora e também é descrito o seu funcionamento. De notar que esta configuração suporta dois canais, mas na tabela ??

apenas é apresentado o canal 0, visto que o canal 1 não é utilizado.



## Anexo B

# Configurações dos interruptores das placas HDMI

Este anexo contempla as diversas configurações possíveis dos interruptores presentes nas placas HDMI para as diferentes configurações das mesmas.

### B.1 Configuração por omissão

Quando as placas estão configuradas de fábrica relembra-se que as imagens transmitidas correspondem ao formato RGB de 30 bits (10 bits por cor). Deste modo, a indicação que vem em [7] sobre as funções dos interruptores da placa HDMI recetora é muito pouca. Apenas é indicado que quando esta configuração está ativa os interruptores se devem encontrar tal como especifica a tabela B.1 na página 93, adaptada de [7].

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 B.1: Configuração dos interruptores da placa HDMI RX configurada de fábrica

Relativamente à placa HDMI transmissora é sabido que lhe chegam imagens no formato RGB de 10 bits. Todavia é possível configurar o integrado 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 B.2 indica, adaptada de [7].

Interruptor	Estado		
<b>S1-1</b>	OFF	ON	ON
<b>S1-2</b>	ON	ON	OFF
<b>S1-3</b>	ON	OFF	ON
<b>S1-4</b>	ON	ON	ON
<b>Output</b>	8 bits	10 bits	12 bits
<b>S1-5</b>	Não usado		
<b>S1-6</b>	Não usado		
<b>S1-7</b>	Não usado		
<b>S1-8</b>	Não usado		

Tabela B.2: Configuração dos interruptores da placa HDMI TX configurada de fábrica

## B.2 Suporte de um canal de imagem e áudio

Quando se configuram as placas HDMI de forma a obter-se o suporte de imagem e som o formato da imagem transmitida também não é limitado ao RGB. Desta maneira, a tabela B.3 indica como se devem configurar os interruptores de forma a obter-se na saída do ADV7612 as diversas possibilidades relativamente ao formato da imagem e foi adaptada de [25].

Interruptor	Estado					
<b>S1-1</b>	ON	OFF	ON	ON	OFF	ON
<b>S1-2</b>	ON	ON	OFF	ON	ON	OFF
<b>S1-3</b>	ON	ON	ON	OFF	OFF	OFF
<b>S1-4</b>	ON	ON	ON	ON	ON	ON
<b>Formato output</b>	YCbCr 444/422	YCbCr 444/422	YCbCr 444/422	RGB	RGB	RGB
<b>Número de bits de output</b>	8 bits	10 bits	12 bits	8 bits	10 bits	12 bits
<b>S1-5</b>	ON					
<b>S1-6</b>	ON					
<b>S1-7</b>	ON					
<b>S1-8</b>	ON					

Tabela B.3: Configuração dos interruptores da placa HDMI RX configurada para um canal e suporte de áudio

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 B.4 apresentada na página 95 indica essas mesmas combinações e foi adaptada de [25].

Interruptor	Estado								
<b>S1-1</b>	ON	OFF	ON	ON	OFF	ON	ON	OFF	ON
<b>S1-2</b>	ON	ON	OFF	ON	ON	OFF	ON	ON	OFF
<b>S1-3</b>	ON	ON	ON	OFF	OFF	OFF	ON	ON	ON
<b>S1-4</b>	ON	ON	ON	ON	ON	ON	OFF	OFF	OFF
<b>Formato de <i>output</i></b>	YCbCr	YCbCr	YCbCr	YCbCr	YCbCr	YCbCr	RGB	RGB	RGB
	444	444	444	422	422	422			
<b>Número de bits de <i>output</i></b>	8 bits	10 bits	12 bits	8 bits	10 bits	12 bits	8 bits	10 bits	12 bits
<b>S1-5</b>	OFF								
<b>S1-6</b>	Não usado								
<b>S1-7</b>	Não usado								
<b>S1-8</b>	ON								

Tabela B.4: Configuração dos interruptores da placa HDMI TX configurada para um canal e suporte de áudio

### B.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 B.5 da página 95 são apresentadas as configurações dos interruptores que configuram o ADV7612 de forma a enviar diferentes formatos, esta tabela foi adaptada [28].

Interruptor	Estado				
<b>S1-1</b>	ON	OFF	ON	OFF	
<b>S1-2</b>	ON	ON	ON	ON	
<b>S1-3</b>	ON	ON	OFF	OFF	
<b>S1-4</b>	ON	ON	ON	ON	
<b>Formato <i>output</i></b>	YCbCr	444/422	YCbCr	444/422	RGB
<b>Número de bits de <i>output</i></b>	8 bits	10 bits	8 bits	10 bits	
<b>S1-5</b>	ON				
<b>S1-6</b>	ON				
<b>S1-7</b>	ON				
<b>S1-8</b>	ON				

Tabela B.5: Configuração dos interruptores da placa HDMI RX configurada para dois canais melhorados

Relativamente à placa HDMI transmissora, ambos os canais são capazes de suportar imagens em formato RGB ou YCbCr. A tabela B.6 da página 96 apresenta as combinações dos interruptores para se poder obter os diversos formatos na saída do ADV7511.

Interruptor	Estado					
<b>S1-1</b>	ON	OFF	ON	OFF	ON	OFF
<b>S1-2</b>	ON	ON	ON	ON	ON	ON
<b>S1-3</b>	ON	ON	OFF	OFF	ON	ON
<b>S1-4</b>	ON	ON	ON	ON	OFF	OFF
<b>Formato <i>output</i></b>	YCbCr 444	YCbCr 444	YCbCr 422	YCbCr 422	RGB	RGB
<b>Número de bits de <i>output</i></b>	8 bits	10 bits	8 bits	10 bits	8 bits	10 bits
<b>S1-5</b>	OFF					
<b>S1-6</b>	Não usado					
<b>S1-7</b>	Não usado					
<b>S1-8</b>	ON					

Tabela B.6: Configuração dos interruptores da placa HDMI TX configurada para dois canais melhorados

## Anexo C

# Localizações das portas conectadas às placas HDMI

Este anexo aborda todos os detalhes relativamente às arquiteturas que se conectam às placas HDMI. São detalhadas para as várias arquiteturas desenvolvidas as suas conexões, tendo em conta que para algumas arquiteturas estas variam bastante, mas para outras as conexões são semelhantes.

### C.1 Transmissão de uma imagem gerada na FPGA

Esta secção contempla as ligações entre a arquitetura desenvolvida com a placa HDMI transmissora. Nesta arquitetura apenas se transmite uma imagem gerada pela a FPGA e por isso todas as suas conexões com a placa transmissora são de saída, tal como ilustra a tabela [C.1](#).

Esta tabela detalha todas as localizações físicas na FPGA de todas as portas de saída bem como a que banco de entradas/saídas pertencem. Para além disso, é também indicado o nome que essas mesmas saídas têm na placa HDMI transmissora.

	Sinal	LOC na FPGA	Banco na FPGA	Nome na placa HDMI
<b>Saída</b>	clk_px	E34	35	TX#0_DCLK
<b>Saída</b>	enable	K35	34	TX#0_DE
<b>Saída</b>	hsync	M32	34	TX#0_HSYNC
<b>Saída</b>	vsync	L31	34	TX#0_VSYNC
<b>Saída</b>	pixel [0]	J32	34	TX#0_D0
<b>Saída</b>	pixel [1]	K33	34	TX#0_D1
<b>Saída</b>	pixel [2]	L34	34	TX#0_D2
<b>Saída</b>	pixel [3]	M33	34	TX#0_D3
<b>Saída</b>	pixel [4]	H34	34	TX#0_D4
<b>Saída</b>	pixel [5]	K29	34	TX#0_D5
<b>Saída</b>	pixel [6]	J30	34	TX#0_D6
<b>Saída</b>	pixel [7]	L29	34	TX#0_D7

	Sinal	LOC na FPGA	Banco na FPGA	Nome na placa HDMI
Saída	pixel [8]	J31	34	TX#0_D8
Saída	pixel [9]	M28	34	TX#0_D9
Saída	pixel [10]	R28	34	TX#0_D10
Saída	pixel [11]	N28	34	TX#0_D11
Saída	pixel [12]	R30	34	TX#0_D12
Saída	pixel [13]	U31	34	TX#0_D13
Saída	pixel [14]	C35	35	TX#0_D14
Saída	pixel [15]	D35	35	TX#0_D15
Saída	pixel [16]	B36	35	TX#0_D16
Saída	pixel [17]	B34	35	TX#0_D17
Saída	pixel [18]	B39	35	TX#0_D18
Saída	pixel [19]	A35	35	TX#0_D19
Saída	pixel [20]	C38	35	TX#0_D20
Saída	pixel [21]	B37	35	TX#0_D21
Saída	pixel [22]	E32	35	TX#0_D22
Saída	pixel [23]	B32	35	TX#0_D23
Saída	pixel [24]	E33	35	TX#0_D24
Saída	pixel [25]	C33	35	TX#0_D25
Saída	pixel [26]	G32	35	TX#0_D26
Saída	pixel [27]	F36	35	TX#0_D27
Saída	pixel [28]	F34	35	TX#0_D28
Saída	pixel [29]	H33	35	TX#0_D29

Tabela C.1: Localização das portas de saída da arquitetura que se conectam à placa HDMI transmissora

## C.2 Transmissão de uma imagem entre dispositivos HDMI

Nesta secção são detalhadas todas as portas de entrada e saída da arquitetura desenvolvida de transmissão de imagem entre dois dispositivos HDMI que se conectam às placas HDMI. A tabela C.2 apresenta todas as portas de entrada da arquitetura que estão conectadas à placa HDMI recetora. Indica as suas localizações físicas na FPGA, ao banco de entradas/saídas a que essas pertencem e ainda o nome que essas ligações correspondem na placa HDMI recetora. A tabela C.1, apresentada neste mesmo anexo, representa as portas de saída desta arquitetura que se conectam à placa HDMI transmissora detalhando toda a informação relativa às mesmas.

	Sinal	LOC na FPGA	Banco na FPGA	Nome na placa HDMI
Entrada	clk_px_in	AJ32	14	RX#0_LLC

	Sinal	LOC na FPGA	Banco na FPGA	Nome na placa HDMI
<b>Entrada</b>	enable_in	AN38	15	RX#0_DE
<b>Entrada</b>	vsync_in	AU38	15	RX#0_VSYNC
<b>Entrada</b>	hsync_in	AU39	15	RX#0_HSYNC
<b>Entrada</b>	pixel_in [0]	AM41	15	RX#0_P0
<b>Entrada</b>	pixel_in [1]	AR38	15	RX#0_P1
<b>Entrada</b>	pixel_in [2]	AN40	15	RX#0_P2
<b>Entrada</b>	pixel_in [3]	AR37	15	RX#0_P3
<b>Entrada</b>	pixel_in [4]	AM39	15	RX#0_P4
<b>Entrada</b>	pixel_in [5]	AP40	15	RX#0_P5
<b>Entrada</b>	pixel_in [6]	AP41	15	RX#0_P6
<b>Entrada</b>	pixel_in [7]	AT39	15	RX#0_P7
<b>Entrada</b>	pixel_in [8]	AR42	15	RX#0_P8
<b>Entrada</b>	pixel_in [9]	AW37	15	RX#0_P9
<b>Entrada</b>	pixel_in [10]	BA37	15	RX#0_P10
<b>Entrada</b>	pixel_in [11]	AW38	15	RX#0_P11
<b>Entrada</b>	pixel_in [12]	BB38	15	RX#0_P12
<b>Entrada</b>	pixel_in [13]	BA39	15	RX#0_P13
<b>Entrada</b>	pixel_in [14]	AK34	14	RX#0_P14
<b>Entrada</b>	pixel_in [15]	AJ33	14	RX#0_P15
<b>Entrada</b>	pixel_in [16]	AM36	14	RX#0_P16
<b>Entrada</b>	pixel_in [17]	AJ36	14	RX#0_P17
<b>Entrada</b>	pixel_in [18]	AP36	14	RX#0_P18
<b>Entrada</b>	pixel_in [19]	AK37	14	RX#0_P19
<b>Entrada</b>	pixel_in [20]	AN35	14	RX#0_P20
<b>Entrada</b>	pixel_in [21]	AL36	14	RX#0_P21
<b>Entrada</b>	pixel_in [22]	AG33	14	RX#0_P22
<b>Entrada</b>	pixel_in [23]	AK35	14	RX#0_P23
<b>Entrada</b>	pixel_in [24]	AH31	14	RX#0_P24
<b>Entrada</b>	pixel_in [25]	AH34	14	RX#0_P25
<b>Entrada</b>	pixel_in [26]	AM34	14	RX#0_P26
<b>Entrada</b>	pixel_in [27]	AM31	14	RX#0_P27
<b>Entrada</b>	pixel_in [28]	AM33	14	RX#0_P28
<b>Entrada</b>	pixel_in [29]	AL29	14	RX#0_P29

Tabela C.2: Localização das portas de entrada da arquitetura provenientes da placa HDMI recetora

### C.3 Transmissão de imagem e som entre dispositivos HDMI

Nesta secção são apresentadas todas as portas de entrada e saída da arquitetura que permite a transmissão de imagem e som entre dispositivos HDMI. Na tabela C.2 são apresentadas algumas portas de entrada desta arquitetura que correspondem aos bits do sinal de píxel entre 0 e 29, o seu sinal de relógio e ainda os sinais de controlo referentes à imagem. As restantes portas de entrada desta arquitetura são apresentadas na tabela C.3 que correspondem essencialmente aos bits do sinal de píxel entre 30 e 35, os sinais relativos ao som (*sclk*, *lrcclk* e dados de som) e ainda os dois bits relativos à informação do vídeo que é transmitido.

Algumas das saídas desta mesma arquitetura são apresentadas nas tabelas C.1 que correspondem aos bits do sinal de píxel entre 0 e 29, o seu sinal de relógio e ainda os sinais de controlo da imagem (*vsync*, *hsync* e *enable*). Os restantes sinais de saída da arquitetura que se conectam à placa HDMI transmissora são apresentados na tabela C.3 e correspondem aos bits do sinal de píxel entre 30 e 35, os sinais referentes ao som e ainda os dois bits relativos à informação do vídeo transmitido.

Todas as tabelas aqui mencionadas apresentam as localizações físicas na FPGA de cada porta, o banco de entradas/saídas a que pertencem e ainda os nomes correspondentes de cada sinal nas placas HDMI (tanto transmissora como recetora).

	Sinal	LOC na FPGA	Banco na FPGA	Nome na placa HDMI
<b>Entrada</b>	pixel_in [30]	AJ31	14	RX#0_P30
<b>Entrada</b>	pixel_in [31]	AJ35	14	RX#0_P31
<b>Entrada</b>	pixel_in [32]	AN34	14	RX#0_P32
<b>Entrada</b>	pixel_in [33]	AM32	14	RX#0_P33
<b>Entrada</b>	pixel_in [34]	AN33	14	RX#0_P34
<b>Entrada</b>	pixel_in [35]	AL30	14	RX#0_P35
<b>Entrada</b>	sclk_in	AJ37	14	RX#0_SCLK
<b>Entrada</b>	lrcclk_in	AL35	14	RX#0_AP5
<b>Entrada</b>	AP1_in	AL37	14	RX#0_AP1
<b>Entrada</b>	AP2_in	AP35	14	RX#0_AP2
<b>Entrada</b>	AP3_in	AM37	14	RX#0_AP3
<b>Entrada</b>	AP4_in	AH33	14	RX#0_AP4
<b>Entrada</b>	info_in [0]	AV38	15	RX#0_Input video status[0]
<b>Entrada</b>	info_in [1]	AV39	15	RX#0_Input video status[1]
<b>Saída</b>	pixel [30]	D33	35	TX#0_D30
<b>Saída</b>	pixel [31]	C34	35	TX#0_D31
<b>Saída</b>	pixel [32]	F32	35	TX#0_D32



	Sinal	LOC na FPGA	Banco na FPGA	Nome na placa HDMI
<b>Saída</b>	pixel [33]	F37	35	TX#0_D33
<b>Saída</b>	pixel [34]	F35	35	TX#0_D34
<b>Saída</b>	pixel [35]	G33	35	TX#0_D35
<b>Saída</b>	sclk_out	A34	35	TX#0_SCLK
<b>Saída</b>	lrclk_out	B33	35	TX#0_AP5
<b>Saída</b>	AP1_out	A36	35	TX#0_AP1
<b>Saída</b>	AP2_out	C39	35	TX#0_AP2
<b>Saída</b>	AP3_out	B38	35	TX#0_AP3
<b>Saída</b>	AP4_out	D32	35	TX#0_AP4
<b>Saída</b>	info_out [0]	K32	34	TX#0_output video status[0]
<b>Saída</b>	info_out [1]	L32	34	TX#0_output video status[1]

Tabela C.3: Localização de algumas portas de entrada e saída da arquitetura de transmissão de imagem e som entre as placas HDMI transmissora e recetora



## Anexo D

# Ficheiros de restrições

### D.1 Restrições Físicas relativas às portas de saída das arquiteturas que se conectam à placa HDMI transmissora

```
set_property PACKAGE_PIN E34 [get_ports clk_px]
set_property PACKAGE_PIN K35 [get_ports enable]
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 IOSTANDARD LVCMOS18 [get_ports clk_px]
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[0]]
...
set_property IOSTANDARD LVCMOS18 [get_ports pixel[29]]

```

## D.2 Restrições Físicas relativas às portas de entrada das arquiteturas que se conectam à placa HDMI recetora

```

set_property PACKAGE_PIN AJ32 [get_ports clk_px_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 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]]
set_property IOSTANDARD LVCMOS18 [get_ports clk_px_in]
set_property IOSTANDARD LVCMOS18 [get_ports enable_in]
set_property IOSTANDARD LVCMOS18 [get_ports vsync_in]
set_property IOSTANDARD LVCMOS18 [get_ports hsync_in]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[0]]
...
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[29]]
```

### **D.3 Restrições Físicas relativas às portas de entrada e saída exclusivas da arquitetura que transmite uma imagem gerada na FPGA**

```
set_property PACKAGE_PIN E18 [get_ports clk_n]
set_property PACKAGE_PIN N41 [get_ports reset]
set_property PACKAGE_PIN E42 [get_ports start]
set_property IOSTANDARD DIFF_HSTL_II_18 [get_ports clk_n]
set_property IOSTANDARD LVCMOS18 [get_ports reset]
set_property IOSTANDARD LVCMOS18 [get_ports start]
```

### **D.4 Restrições Temporais relativas à arquitetura que transmite uma imagem gerada na FPGA**

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

### **D.5 Restrições Físicas relativas às portas de entrada e saída exclusivas da arquitetura que transmite imagem entre dispositivos HDMI**

```
set_property PACKAGE_PIN N41 [get_ports reset]
set_property PACKAGE_PIN E42 [get_ports start]
set_property PACKAGE_PIN E18 [get_ports clk_n]
set_property IOSTANDARD LVCMOS18 [get_ports reset]
set_property IOSTANDARD LVCMOS18 [get_ports start]
set_property IOSTANDARD DIFF_HSTL_II_18 [get_ports clk_n]
```

## D.6 Restrições Temporais relativas à arquitetura que transmite imagem entre dispositivos HDMI

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

## D.7 Restrições Físicas relativas às portas de entrada e saída exclusivas da arquitetura que transmite imagem e som entre dispositivos HDMI

```
set_property PACKAGE_PIN E19 [get_ports clock_p]
set_property PACKAGE_PIN N41 [get_ports reset]
set_property PACKAGE_PIN E42 [get_ports POWER]
set_property PACKAGE_PIN G41 [get_ports MUTE]
set_property IOSTANDARD DIFF_HSTL_II_18 [get_ports clock_p]
set_property IOSTANDARD LVCMOS18 [get_ports reset]
set_property IOSTANDARD LVCMOS18 [get_ports POWER]
set_property IOSTANDARD LVCMOS18 [get_ports MUTE]
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 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 PACKAGE_PIN AP35 [get_ports AP2_in]
set_property PACKAGE_PIN AM37 [get_ports AP3_in]
set_property PACKAGE_PIN AH33 [get_ports AP4_in]
set_property PACKAGE_PIN AL35 [get_ports lrclk_in]
set_property PACKAGE_PIN AJ37 [get_ports sclk_in]
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[30]]
...
set_property IOSTANDARD LVCMOS18 [get_ports pixel_in[35]]
set_property IOSTANDARD LVCMOS18 [get_ports info_in[0]]
set_property IOSTANDARD LVCMOS18 [get_ports info_in[1]]
set_property IOSTANDARD LVCMOS18 [get_ports AP1_in]
set_property IOSTANDARD LVCMOS18 [get_ports AP2_in]
set_property IOSTANDARD LVCMOS18 [get_ports AP3_in]
set_property IOSTANDARD LVCMOS18 [get_ports AP4_in]
set_property IOSTANDARD LVCMOS18 [get_ports lrclk_in]
set_property IOSTANDARD LVCMOS18 [get_ports sclk_in]
set_property PACKAGE_PIN D33 [get_ports pixel[30]]
```

```
set_property PACKAGE_PIN C34 [get_ports pixel[31]]
set_property PACKAGE_PIN F32 [get_ports pixel[32]]
set_property PACKAGE_PIN F37 [get_ports pixel[33]]
set_property PACKAGE_PIN F35 [get_ports pixel[34]]
set_property PACKAGE_PIN G33 [get_ports pixel[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 PACKAGE_PIN C39 [get_ports AP2_out]
set_property PACKAGE_PIN B38 [get_ports AP3_out]
set_property PACKAGE_PIN D32 [get_ports AP4_out]
set_property PACKAGE_PIN B33 [get_ports lrclk_out]
set_property PACKAGE_PIN A34 [get_ports sclk_out]
set_property IOSTANDARD LVCMOS18 [get_ports pixel[30]]
...
set_property IOSTANDARD LVCMOS18 [get_ports pixel[35]]
set_property IOSTANDARD LVCMOS18 [get_ports AP1_out]
set_property IOSTANDARD LVCMOS18 [get_ports AP2_out]
set_property IOSTANDARD LVCMOS18 [get_ports AP3_out]
set_property IOSTANDARD LVCMOS18 [get_ports AP4_out]
set_property IOSTANDARD LVCMOS18 [get_ports lrclk_out]
set_property IOSTANDARD LVCMOS18 [get_ports sclk_out]
```

## **D.8 Restrições Temporais relativas à arquitetura que transmite imagem e som entre dispositivos HDMI**

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