

João Pedro Aleixo Duarte

# AQUISIÇÃO DE IMAGENS ATRAVÉS DE FPGA

(IMAGE ACQUISITION WITH FPGA)

Dissertação de Mestrado  
Mestrado Integrado em Engenharia Electrotécnica e de Computadores

Setembro de 2015

• U • C •



UNIVERSIDADE DE COIMBRA





Universidade de Coimbra  
Faculdade de Ciências e Tecnologia  
*Departamento de Engenharia Eletrotécnica e de Computadores*

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

# **Aquisição de Imagens através de FPGA**

## **(Image Acquisition with FPGA)**

João Pedro Aleixo Duarte

Trabalho desenvolvido sob supervisão de:  
Tony Richard de Oliveira de Almeida (DEEC - FCTUC)  
Jorge Manuel Simões de Almeida (CRITICAL Software, SA)

Júri:  
António Paulo Mendes Breda Dias Coimbra  
Gabriel Falcão Paiva Fernandes  
Tony Richard de Oliveira de Almeida

Coimbra, Setembro de 2015



Com o apoio de iTGROW  
Trabalho desenvolvido nas instalações da CRITICAL Software, SA  
no Parque Industrial de Taveiro

iTGROW





*A todos aqueles sem os quais este trabalho não seria possível...*





# Agradecimentos

Queria começar por agradecer àqueles que contribuíram de forma mais directa para este trabalho. Ao meu orientador, o Professor Tony Richard, pela disponibilidade e simpatia que sempre demonstrou ao longo de todo o processo. A todos os colaboradores da CRITICAL Software, em especial ao Álvaro Lopes, Humberto Rodrigues, Jorge Almeida, José Rui Simões e Paulo Fernandes pela ajuda, aconselhamento e apoio que foram dando durante estes últimos meses. Aos outros, onde incluo os “iTGrow’ers”, obrigado pela estima e carinho que tenho recebido.

Agradeço também àqueles que me apoiam e amparam: à minha família e amigos por estarem sempre ao meu lado.

Aos que me acompanharam neste percurso académico: desde colegas, professores e à restante comunidade do DEEC, por terem, todos eles, contribuído de alguma forma para o meu crescimento.

A todos, o meu sincero obrigado.



# Abstract

In this work, it was intended to proceed with the capture of images that are sent to the display of a completely digital dashboard for automotive applications, by the graphic processor, for future comparison. The project also wants to confirm that FPGAs (Field-Programmable Gate Arrays) can be a viable alternative to frame grabbers, ASICs (Application Specific Integrated Circuits) or simply other processors and microcontrollers in this type of application, for being able to acquire images in real-time at the same frame rates that they are being sent, *i.e.* without down sampling. The FPGAs also allow to reduce costs and development times due to its flexibility and efficiency.

It was used an FPGA ZedBoard (Zynq Evaluation and Development Board) for developing this project and it was also necessary to design and produce a printed circuit board (PCB) which is an essential element in the connection between the FPGA, the instrument cluster board and its display.

At the time, it was designed and ready to implement in FPGA a hardware and software platform, capable of capturing images. The system was successfully tested in simulation environment and it is ready for tests in real situations that will confirm its correct operation. Therefore, there is a baseline for concrete results, in the near future.

The next step would be an image comparison (image difference or pattern comparison) between the captured images and reference images, to confirm that the information presented by the instrument cluster's display is in accordance with the "stimuli" sent (mostly CAN messages), simulating the various vehicle equipments.

**Keywords:** display, FPGA, instrument cluster, PCB, real-time image acquisition, ZedBoard.



# Resumo

Neste trabalho pretendia-se proceder à captura das imagens a serem enviadas para o ecrã de um painel de instrumentos de aplicação na indústria automóvel, completamente digital, pelo processador gráfico, para futura comparação. Quer também confirmar que as FPGAs (*Field-Programmable Gate Arrays*) podem ser uma alternativa viável, a *frame grabbers*, ASICs (*Application Specific Integrated Circuit*) ou outros processadores e microcontroladores, neste tipo de aplicação ao serem capazes de adquirir as imagens em tempo real, ao mesmo ritmo a que as *frames* estão a ser enviados, *i.e.* sem ser necessário qualquer tipo de subamostragem. As FPGAs permitem também reduzir os custos e períodos de desenvolvimento devido à sua flexibilidade e eficiência.

Usou-se uma FPGA ZedBoard (*Zynq Evaluation and Development Board*) para o desenvolvimento deste projecto, tendo sido também necessário projectar e produzir uma placa de circuito impresso (PCB) que é um elemento essencial na ligação entre a FPGA, a placa do painel de instrumentos e o ecrã do mesmo.

À data, ficou projectada e pronta a implementar na FPGA uma plataforma de *hardware* e *software*, capaz de capturar imagens. O sistema foi testado com sucesso em ambientes simulados e está pronto para testes em situações reais que confirmarão a sua correcta operação. Está criada a base para chegar a resultados concretos, num futuro próximo.

O próximo passo será uma comparação das imagens (ou pela diferença das imagens ou por comparação de padrões) entre as imagens capturadas e imagens de referência, para confirmar que a informação apresentada pelo ecrã está de acordo com os “estímulos” enviados (sobretudo mensagens CAN), simulando os vários equipamentos do veículo.

**Palavras-chave:** ecrã, FPGA, painel de instrumentos, PCB, aquisição de imagens em tempo-real, ZedBoard.



# Índice

Agradecimentos .....	iii
Abstract .....	v
Resumo .....	vii
Índice .....	ix
Lista de Figuras .....	xiii
Lista de Figuras (Anexos) .....	xiv
Lista de Tabelas.....	xiv
Acrónimos e Unidades .....	xvii
1 Introdução.....	1
1.1 Motivação.....	1
1.2 Objetivos .....	2
1.3 Estrutura da dissertação.....	2
2 Estado da arte .....	5
2.1 Principais contribuições .....	6
3 Painel de instrumentos .....	9
4 Ligação com o ecrã .....	17
5 Escolha da FPGA .....	23
6 Arquitectura.....	27
6.1 Arquitectura física .....	27
6.2 Arquitectura do caminho de dados ( <i>datapath</i> ) .....	30
7 Implementação .....	35
7.1 Vivado Design Suite.....	35
7.2 Construção do <i>datapath</i> .....	36
7.2.1 Desenvolvimento de software .....	41
7.2.2 Notas sobre Interfaces AXI .....	42
7.2.3 Módulos criados.....	43
7.3 Testes ao <i>datapath</i> .....	51
7.3.1 <i>Test benches</i> .....	51
7.3.2 Integração dos módulos de debug na plataforma de hardware.....	55
7.3.3 Mark Debug .....	55
7.3.4 Testes ao <i>datapath</i> completo .....	56
8 Montagem do sistema.....	63
8.1 Projecto da PCB de interface.....	63

8.2 Ligação dos equipamentos e testes ao sistema.....	66
9 Trabalho complementar.....	69
9.1 Módulos adicionais implementados.....	69
9.2 Possíveis melhoramentos.....	70
9.2.1 Envio da informação para outra máquina.....	70
9.2.2 Multiprocessamento assimétrico.....	72
9.2.3 Interfaces Pmod.....	72
9.2.4 Saídas de vídeo da ZedBoard.....	73
9.2.5 Projecto de PCBs de interface alternativas.....	74
10 Conclusões.....	77
Referências.....	81
Anexos.....	89
Anexo A - Exemplos de código desenvolvido.....	91
A - 1) pattern_generator.vhd.....	91
A - 2) frame_ram.vhd (versão inicial).....	95
A - 3) frame_ram.vhd (versão 2, com integração de FIFO e interface com RAM).....	98
A - 4) pattern_generator_tb.vhd.....	102
A - 5) FrameRAMwAXI_v1_0.vhd.....	104
A - 6) main.c (aplicação que configura o módulo VDMA e o controlador HDMI via I <sup>2</sup> C).....	110
Anexo B - Imagens de alguns projectos criados.....	121
B - 1) Diagrama de blocos - datapath completo (versão minimal).....	121
B - 2) Pormenor do diagrama de blocos - datapath completo (versão minimal).....	122
B - 3) Diagrama de blocos - bloco “image_input”.....	123
B - 4) Relatório sumário do projecto - datapath completo (versão minimal).....	124
B - 5) Diagrama de blocos - datapath completo (versão com entrada de vídeo externa e saída HDMI).....	125
B - 6) Relatório sumário do projecto - datapath completo (versão com entrada de vídeo externa e saída HDMI).....	126
Anexo C - Desenhos esquemáticos das PCB (alternativas).....	129
C - 1) PCB - A1.....	129
C - 2) PCB - A2.....	130







# Lista de Figuras

Figura 1 - Fotografia do painel de instrumentos (frente) .....	9
Figura 2 - Fotografia do painel de instrumentos (trás).....	10
Figura 3 - Diagrama com a definição das fronteiras e interfaces do painel de instrumentos.....	11
Figura 4 - Fotografia com a placa do painel de instrumentos .....	12
Figura 5 - Fotografia com a placa do painel de instrumentos (com anotações).....	13
Figura 6 - Esquema mais detalhado das interfaces entre componentes do painel de instrumentos e/ou restantes elementos do veículo.....	13
Figura 7 - Foto do FPC principal do ecrã.....	18
Figura 8 - Foto do FPC principal do ecrã (com anotações) .....	19
Figura 9 - Ilustração da primeira hipótese de arquitectura física .....	29
Figura 10 - Ilustração da segunda alternativa de arquitectura física .....	29
Figura 11 - Diagrama de blocos da arquitectura do caminho de dados .....	30
Figura 12 - Interface principal da suite Vivado com destaque para o “Flow Navigator” .....	36
Figura 13 - Diagrama de blocos (esquemático) da implementação básica .....	37
Figura 14 - (a) Diagrama de blocos da implementação básica; (b) Pormenor do diagrama.....	38
Figura 15 - Composição do bloco “image_input” .....	39
Figura 16 - Representação dos padrões de barras de cor (versão original) (formato 8:3) .....	47
Figura 17 - Representação dos padrões de barras de cor (versão final) (formato 8:3).....	48
Figura 18 - Representação do segundo padrão (18 bits menos significativos da posição (RGB666)) (formato 8:3).....	50
Figura 19 - Aspecto da aplicação aquando a criação de IP cores.....	50
Figura 20 - Diagrama temporal dos sinais do módulo “gerador de padrões” (1 frame, 8x8) .....	54
Figura 21 - Diagrama temporal dos sinais do módulo “gerador de padrões” (1 linha, 8 colunas) .....	54
Figura 22 - Linha assinalada com a opção “Mark Debug” .....	55
Figura 23 - Diagrama de blocos esquemático do datapath (com saída HDMI) .....	56
Figura 24 - Diagrama de blocos esquemático do datapath (monitores de memória).....	57
Figura 25 - Diagrama de blocos do datapath (versão com entrada externa e saída HDMI) .....	60
Figura 28 - Desenho da PCB criada (board) .....	65
Figura 28 - Fotografia da PCB criada (face inferior) .....	65

Figura 28 - Fotografia da PCB criada (face superior).....	65
Figura 29 - Ilustração da montagem efectuada .....	66
Figura 30 - Fita FFC utilizada na ligação .....	67
Figura 31 - Fotografia do ecrã ligado à saída de vídeo externa (HDMI).....	74
Figura 32 - Desenho da nova PCB (A1) .....	75
Figura 33 - Desenho da nova PCB (A2) .....	75

## Lista de Figuras (Anexos)

1 - B - 1) Diagrama de blocos - datapath completo (versão minimal).....	121
2 - B - 2) Promenor do diagrama de blocos - datapath completo (versão minimal).....	122
3 - B - 3) Diagrama de blocos - bloco “image_input” .....	123
4 - B - 4) Relatório sumário do projecto - datapath completo (versão minimal).....	124
5 - B - 5) Diagrama de blocos - datapath completo (versão com entrada de vídeo externa e saída HDMI).....	125
6 - B - 6) Relatório sumário do projecto - datapath completo (versão com entrada de vídeo externa e saída HDMI) .....	126
7 - C - 1) Desenho esquemático da PCB alternativa 1 (PCB A1).....	129
8 - C - 2) Desenho esquemático da PCB alternativa 2 (PCB A2).....	130

## Lista de Tabelas

Tabela 1 - Lista de algumas FPGAs e suas características .....	24
--	----





# Acrónimos e Unidades

<b>Acrónimo</b>	<b>Significado</b>
AMBA	Advanced Microcontroller Bus Architecture
AMS	Analog Mixed Signal
AP	All Programmable
APIX	Automotive Pixel Link
ARM	Acorn RISC Machine, posteriormente Advanced RISC Machine
ASIC	Application Specific Integrated Circuit
AXI	Advanced eXtensible Interface
BOM	Bill of Materials
BSP	Board Support Package
CAN	Controller Area Network
CCD	Charge-Coupled Device
CD	Compact Disc
CI / IC	Circuito Integrado, Integrated Circuit
CMOS	Complementary Metal-Oxide-Semiconductor
DAC	Digital-to-Analog Converter
DC	Direct Current
DDR	Double Data Rate
DEEC	Departamento de Engenharia Electrotécnica e de Computadores
DMA	Direct Memory Access
(D)RAM	(Dynamic) Random Access Memory
DRC	Design Rule Check(ing)
DSP	Digital Signal Processor
(EEP)ROM	(Electrically-Erasable Programmable) Read-Only Memory
EMC	Electromagnetic Compatibility
EOL	End of Line
ESCL	Electronic Steering Column Lock
FCTUC	Faculdade de Ciências e Tecnologia da UC
FFC	Flexible Flat Cable
FIFO	First In, First Out
FMC	FPGA Mezzanine Card
FPC	Flexible Printed Circuit
FPD-Link	Flat Panel Display Link
FPGA	Field-Programmable Gate Arrays
FPS	Frames Per Second
GDC	Graphics Display Controllers
GIP	Graphic Interface Processor
GND	Ground
GP	General Purpose
GPIO	General Purpose Input/Output
HDMI	High-Definition Multimedia Interface
HP	High Performance

HS-CAN	High Speed CAN
I/O	Input/Output
I2C / IIC	Inter-Integrated Circuit
IP	Intellectual Property
ISE	Integrated Synthesis Environment, (ferramenta de software produzida pela Xilinx)
ISO	International Organization for Standardization
LCD	Liquid-Crystal Display
LED	Light-Emitting Diode
LIFO	Last In, First Out
LIN	Local Interconnect Network
LPC	Low-Pin Count
LSB	Least Significant Bit
LSD	Laboratório de Sistemas Digitais
LTPS	Low-Temperature Polycrystalline Silicon
LVDS	Low-Voltage Differential Signaling
MCU	Microcontroller Unit
MIEEC	Mestrado Integrado em Engenharia Electrotécnica e de Computadores
MMCM	Mixed-Mode Clock Manager
MOST	Media Oriented Systems Transport
MS-CAN	Medium Speed CAN
NGI	Next-Generation Interface
PATS	Passive Anti-Theft System
PCB	Printed Circuit Board
PL	Programmable Logic
PLL	Phase-Locked Loop
Pmod	Peripheral Modules
PS	Processing System
PSD	Projecto de Sistemas Digitais
PWM	Pulse-Width Modulation
RGB	Red, Green and Blue
RISC	Reduced Instruction Set Computing
ROI	Region of Interest
RSDS	Reduced Swing Differential Signal
RTL	Register Transfer Level
SA	Sociedade Anónima
SDK	Software Development Kit
SMD	Surface-Mount Device
SO / OS	Sistema Operativo, Operating System
SoC	System-on-a-Chip
SPI	Serial Peripheral Interface
SVF	Software Validation Facility
TFT	Thin-Film-Transistor
TTL	Transistor-Transistor Logic
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
VCC	Tensão de alimentação positiva
VCOM	Common Voltage



VDMA	Video DMA
VGA	Video Graphics Array
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuits
VIP	Vehicle Interface Processor
XADC	Xilinx Analog-to-Digital Converter
ZedBoard	Zynq Evaluation and Development Board

---

<b>Grandeza</b>	<b>Unidade</b>	<b>Símbolo</b>
Comprimento	metro	m
Corrente eléctrica	ampere	A
Frequência	hertz	Hz
Informação digital	bit, byte	b, B
Potencial eléctrico	volt	V
Resistência eléctrica	ohm	$\Omega$
Resolução	píxel	px
Temperatura	celsius	$^{\circ}\text{C}$
Tempo	segundo	s



# 1 Introdução

## 1.1 Motivação

Os painéis de instrumentos usados em automóveis têm de ser bastante robustos em termos de segurança, já que deles podem inclusivamente depender a vida de pessoas. Têm de cumprir diversas normas e padrões internacionais, entre os quais a ISO 26262 [1] no que diz respeito à segurança funcional. É bastante importante testar este tipo de equipamentos de forma a confirmar a sua resposta em situações de possíveis falhas.

De forma a validar o seu comportamento, queríamos proceder à captura das imagens digitais que são enviadas para o ecrã do painel de instrumentos. Posteriormente estas imagens poderiam ser alvo de comparação com imagens de referência e, em caso de uma avaliação positiva, validaríamos assim o desempenho do equipamento.

Para tal, teria de recorrer a uma FPGA, já que era uma das restrições imposta pela empresa em que o projecto foi inserido, devido ao bom compromisso preço/desempenho que é espectável neste tipo de equipamentos quando aplicados a este tipo de problemas. Permitia ainda agilizar o processo de desenvolvimento e desenvolver uma solução adaptável e flexível.

Em termos pessoais, parte do interesse para desenvolver este trabalho passava pelas condições em que se ia realizar, ao tratar-se de um estágio a ser desenvolvido nas instalações da CRITICAL Software, num ambiente mais próximo do que poderei encontrar num futuro a curto prazo.

## 1.2 Objetivos

O objectivo principal do trabalho passava por capturar as imagens que estão a ser enviadas para o ecrã do painel de instrumentos. Pretendia-se implementar uma solução que capturasse as imagens a serem enviadas para o ecrã, para posterior validação. O sistema deveria ser obrigatoriamente implementado numa FPGA. Deveria ser desenhada toda a arquitectura do sistema. Teria de ser desenvolvida toda a plataforma de *hardware*, assim como *software* adicional necessário para a configuração do *hardware*.

Para implementar o sistema por completo, em condições reais, era ainda necessário projectar a interface entre os equipamentos (FPGA, placa do painel de instrumentos e respectivo ecrã). Para o efeito teria de ser desenhada uma PCB.

Como objectivos adicionais, poderia ser implementada parte da fase de validação na FPGA ou serem estudados mecanismos de transferência das imagens capturadas para outro computador, para serem aí processadas.

## 1.3 Estrutura da dissertação

Anteriormente a esta secção temos parte da introdução, onde são apresentadas a motivação para este trabalho e os seus objectivos.

No capítulo seguinte é abordado o estado da arte, onde são apresentados outros trabalhos desta área e as suas conclusões, e as principais contribuições deste trabalho, nessa perspectiva.

O terceiro capítulo corresponde à apresentação do painel de instrumentos, com base numa inspecção visual e na análise da documentação. Segue-se (capítulo 4) o estudo da interface com o ecrã e a apresentação, já no capítulo subsequente (capítulo 5), dos factores que levaram à escolha da FPGA que viria a ser adquirida para dar continuidade ao projecto.

Numa outra parte é apresentada a arquitectura do sistema (capítulo 6) que pretendíamos implementar dividida em duas partes: arquitectura da parte “física” onde são apresentadas as possibilidades quanto à ligação dos equipamentos e a arquitectura do caminho de dados onde é exposto o *datapath* que foi idealizado para este projecto.

Segue-se o capítulo onde é descrita a implementação do sistema na FPGA (capítulo 7): começa por uma introdução ao Vivado da Xilinx, são mostrados mais alguns detalhes da implementação da plataforma de *hardware* e do *software* criado para essa plataforma, surgem algumas notas sobre o trabalho realizado na elaboração dos módulos iniciais, sucedendo-se a explicação dos testes realizados.

Posteriormente, é apresentado o projecto da PCB de interface, surgindo depois a secção onde é apresentado o sistema completo, com a ligação de todos os equipamentos (capítulo 0). Na fase final surgem algumas referências relativamente ao trabalho complementar que foi desenvolvido que incluem outros módulos que foram criados e possíveis melhoramentos (envio de informação para outras máquinas, possibilidades de multiprocessamento assimétrico, uso das interfaces Pmod e/ou saídas de vídeo, terminado com a apresentação de projectos de PCBs alternativas).

Para terminar, surgem as conclusões deste trabalho onde são expostas as notas finais sobre o mesmo, o que foi feito e o que pode ainda vir a ser desenvolvido de forma a dar continuidade a este projecto.

Informação complementar relativa ao trabalho desenvolvido pode ser encontrado nos anexos incluídos no final desta dissertação.



## 2 Estado da arte

É possível encontrar uma quantidade significativa de trabalhos científicos com objectivos semelhantes com alguma pesquisa. Por objectivos semelhantes, foi considerada a captura e/ou processamento de imagens com recurso a FPGAs independentemente da proveniência dos dados [2, 3, 4, 5, 6, 7, 8, 9, 10, 11], [12, 13, 14, 15, 16, 17, 18, 19, 20, 21] e [22, 23].

Embora não tenham sido encontrados grandes detalhes sobre a implementação destes projectos, a descrição das vantagens acaba por ser semelhante em todos eles. Alguns destes estudos afirmam que é possível fazer a captura de imagens em tempo-real com recurso exclusivo a FPGAs, noutros a FPGA surge como um complemento ao sistema permitindo assim aceleração por *hardware*, sendo depois as imagens processadas por DSPs (*Digital Signal Processor*) ou por outros processadores. A maioria refere as seguintes características como benefícios que o uso de FPGAs acrescenta aos seus projectos: a flexibilidade deste tipo de equipamentos é, por norma, mencionada como uma vantagem, ao permitir uma prototipagem mais rápida e com menos custos, podendo normalmente ser adaptada consoante os restantes equipamentos do qual provêm as imagens (com a nota de que, na maior parte dos projectos, as imagens provêm de câmaras/sensores de imagem (CCD ou CMOS)). Por vezes são também referidas as capacidades de portabilidade da solução, a sua escalabilidade ou possibilidade de expansão, e os baixos consumos energéticos.

Nalgumas das ligações incluídas nas referências é também possível aceder a outros projectos semelhantes nas citações ou, no caso inverso, trabalhos em que o projecto é citado.

Numa perspectiva um pouco diferente, foram encontradas algumas referências, principalmente dentro da oferta dos principais fabricantes, em que as FPGAs são consideradas como a alternativa a seguir no controlo e criação da própria interface gráfica destes painéis de instrumentos para automóveis [24, 25, 26, 27, 28, 29]. Mais uma vez são salientadas as potencialidades que este tipo de tecnologia pode providenciar: flexibilidade, redução dos custos e dos tempos de desenvolvimento entre outras.

## 2.1 Principais contribuições

O trabalho desenvolvido surge de certa forma em linha com alguns dos trabalhos encontrados na pesquisa realizada para a elaboração do estado da arte. É possível afirmar que a captura de imagens em tempo-real com recurso a FPGAs é viável, independentemente da origem dos dados, com as devidas adaptações a este caso específico.

Ficaram também comprovadas as capacidades deste tipo de equipamentos no que diz respeito à sua flexibilidade, às possibilidades de reutilização de código/componentes que permitem tempos de desenvolvimento mais reduzidos, diminuição de custos e exportação para outras plataformas.

A escalabilidade do projecto também é viável uma vez que o equipamento utilizado ainda possuía muitos recursos disponíveis e possibilidades que não foram exploradas por completo. Estes recursos podem vir a ser utilizados para atingir os objectivos adicionais tais como a validação das imagens capturadas, feita na própria FPGA, ou a implementação do envio das imagens para outra máquina, com o mesmo objectivo de validação.

A principal singularidade do trabalho acaba por ser a aplicação a que se destina, à captura de imagens provenientes de um painel de instrumentos de automóvel com propósitos de validação. Esta acaba por se tornar a principal distinção relativamente aos restantes trabalhos e poderá inclusivamente tornar-se numa vantagem competitiva para as empresas em que este trabalho foi inserido.







### 3 Painel de instrumentos

A exposição do trabalho desenvolvido irá começar por uma análise do painel de instrumentos fornecido, talvez o principal elemento de todo este estudo.

Todo o equipamento é protegido por um corpo de plástico, sendo que na parte frontal, o plástico é côncavo e transparente permitindo a visibilidade de um ecrã (Fig. 1 - A1) que ocupa a grande maioria da sua extensão e ainda quatro elementos situados nas laterais do ecrã (dois de cada lado) (Fig. 1 - A2): dois sensores ópticos que permitem ajustar a luminosidade do ecrã em função da luz ambiente e dois indicadores, iluminados por LEDs, de avaria no *Airbag* e do sistema de segurança PATS (*Passive Anti-Theft System*). Destes elementos destaca-se o ecrã, sobre o qual recai o objectivo do trabalho: a captura das imagens que estão a ser emitidas.



Figura 1 - Fotografia do painel de instrumentos (frente)

As restantes partes são de plástico preto, com os encaixes que permitem a montagem do painel no veículo, e tem gravadas as referências do produto e as logomarcas dos construtores automóveis (que foram dissimuladas na Figura 2 para que não possam ser identificadas).

Imediatamente atrás do ecrã encontra-se exposto um altifalante que permitirá emitir avisos sonoros ao condutor (Fig. 5 - A3) e na parte traseira conseguimos distinguir três conectores (Fig. 2 - B1, B2 e B3), estando um deles colocado de forma lateral (Fig. 2 - B3): orientado para a lateral do painel de instrumentos temos um conector MOST (*Media Oriented Systems Transport*) (Fig. 2 - B3) para a ligação com equipamentos multimédia (navegação, *infotainment*, telemóvel, etc.), existe também um conector LVDS (*Low-Voltage Differential Signaling*) (Fig. 2 - B2) que permite a entrada de um sinal de vídeo externo e por último mas de substancial relevância temos o conector principal do painel de instrumentos (Fig. 2 - B1) que implementa todas as principais comunicações com os restantes equipamentos do veículo (essencialmente via CAN e LIN) assim como a alimentação do próprio painel.

Para além de disponibilizar os pinos para as comunicações CAN, LIN e alimentação, temos ainda linhas para alimentação e regulação de uma possível ventoinha para efeitos de dissipação do calor gerado pelos CI que não se encontra montada no modelo disponibilizado (se bem que é facilmente perceptível qual seria a sua posição no painel), linhas para a transmissão dos sinais de teste (a realizar no final da linha de produção) identificadas como EOL (*End of Line*) assim como pinos para o bloqueio da coluna de direcção do veículo (ESCL - *Electronic Steering Column Lock*).

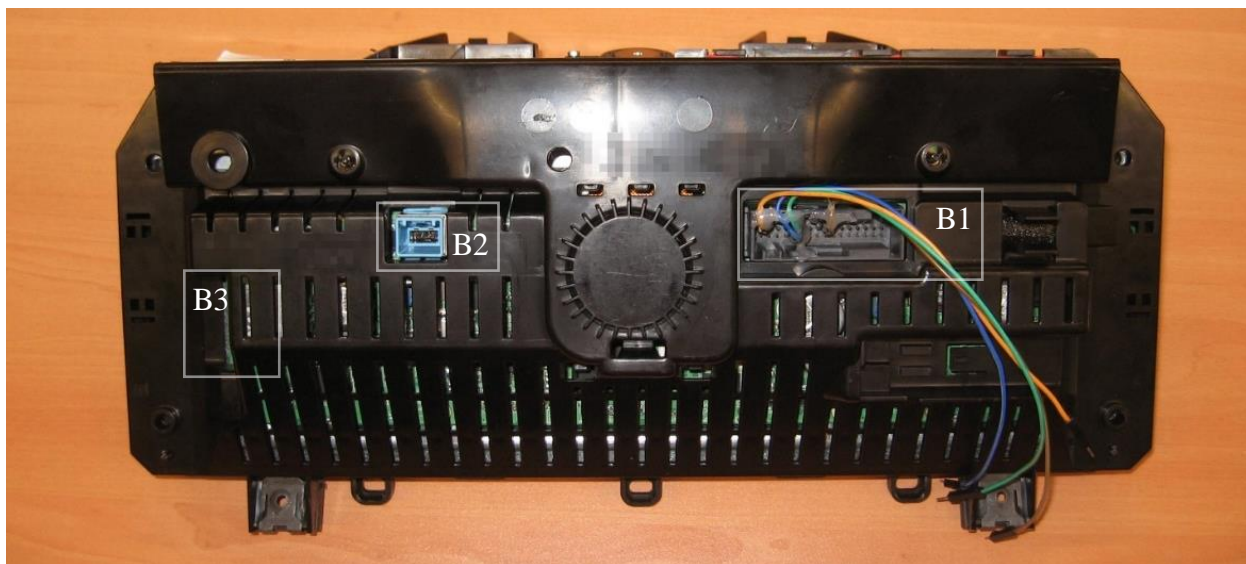


Figura 2 - Fotografia do painel de instrumentos (trás)

Para efeitos deste trabalho, foram apenas utilizadas as ligações com os *jumper cables*, sendo que, como referido no parágrafo anterior, um par se destina a alimentação de todo o painel e o outro à ligação CAN de alta velocidade (também essencial para que o painel inicie o seu funcionamento, permitindo também enviar mensagens que alterem a resposta do sistema e, conseqüentemente, as imagens apresentadas no ecrã).

Para completar e clarificar o que acabou de ser escrito, foi acrescentado o seguinte diagrama, presente na documentação fornecida, onde são apresentadas as diversas interfaces de entrada e saída do painel de instrumentos.

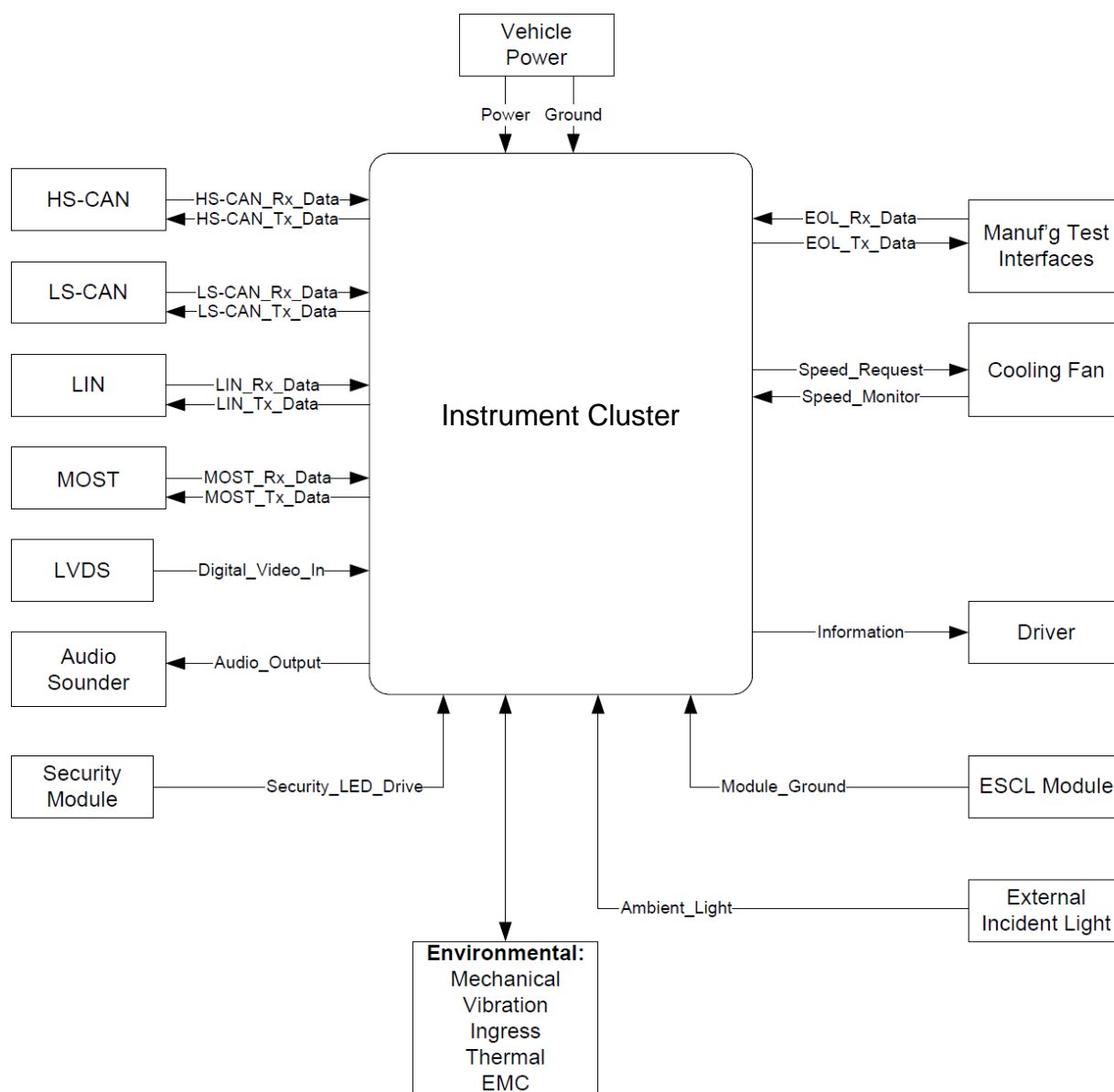


Figura 3 - Diagrama com a definição das fronteiras e interfaces do painel de instrumentos

Prossigamos a análise para o interior do painel de instrumentos. Para expor a placa principal do painel é necessário separar o corpo de plástico em duas partes. Deparamo-nos com uma quantidade significativa de circuitos integrados e elementos passivos. Apenas foram inspecionados os componentes montados na face exposta da PCB, uma vez que não foi possível desaparafusar manualmente a placa do seu suporte metálico. Ainda assim, foram certamente analisados os componentes principais, que necessitam de estar na face visível por motivos de espaço e de dissipação de calor.

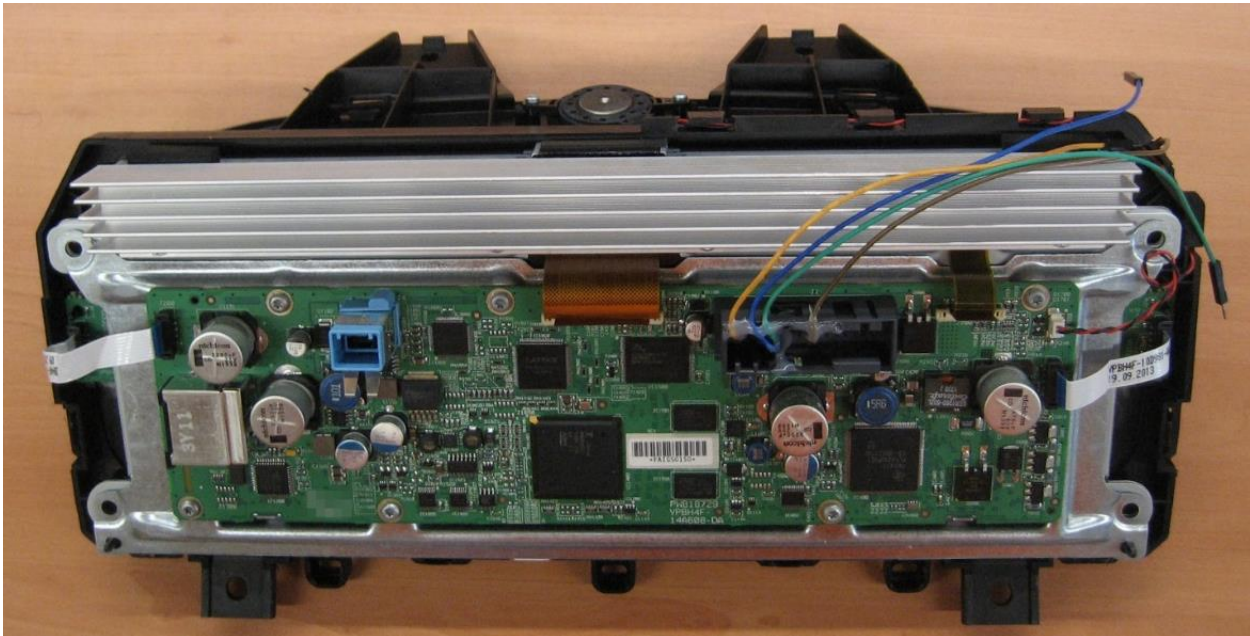


Figura 4 - Fotografia com a placa do painel de instrumentos

Para além dos três conectores anteriormente referidos, que servem de ligação com equipamento exterior ao painel de instrumentos conseguimos agora ver cinco conectores adicionais para ligar os diferentes elementos do painel de instrumentos, dois para ligar a PCB principal aos componentes nas laterais do ecrã (Fig. 5 - A2), dois para a ligação com o ecrã (Fig. 5 - A1) e o último para a conexão com o altifalante (Fig. 5 - A3).

Atendendo à perceptível importância que as ligações da placa principal com o ecrã iriam ter neste trabalho, foi nestas que se concentraram as primeiras tarefas desenvolvidas. Ainda relativamente a estas, o conector maior (e respectiva fita) apresentava-se, presumivelmente, como sendo responsável pela transmissão dos dados e dos sinais de sincronismo para o ecrã sendo que o outro teria a seu cargo funções secundárias, como a alimentação dos LEDs (*Light-Emitting Diode* - Díodo Emissor de Luz) de retroiluminação do ecrã, já que o mesmo acontece em equipamentos semelhantes.

Embora não tenha sido possível aceder ao desenho ou esquemático da placa, nem à lista dos materiais, foi feito o levantamento dos principais CIs montados na PCB, na expectativa de perceber um pouco melhor a função de cada um e o funcionamento geral do painel de instrumentos.

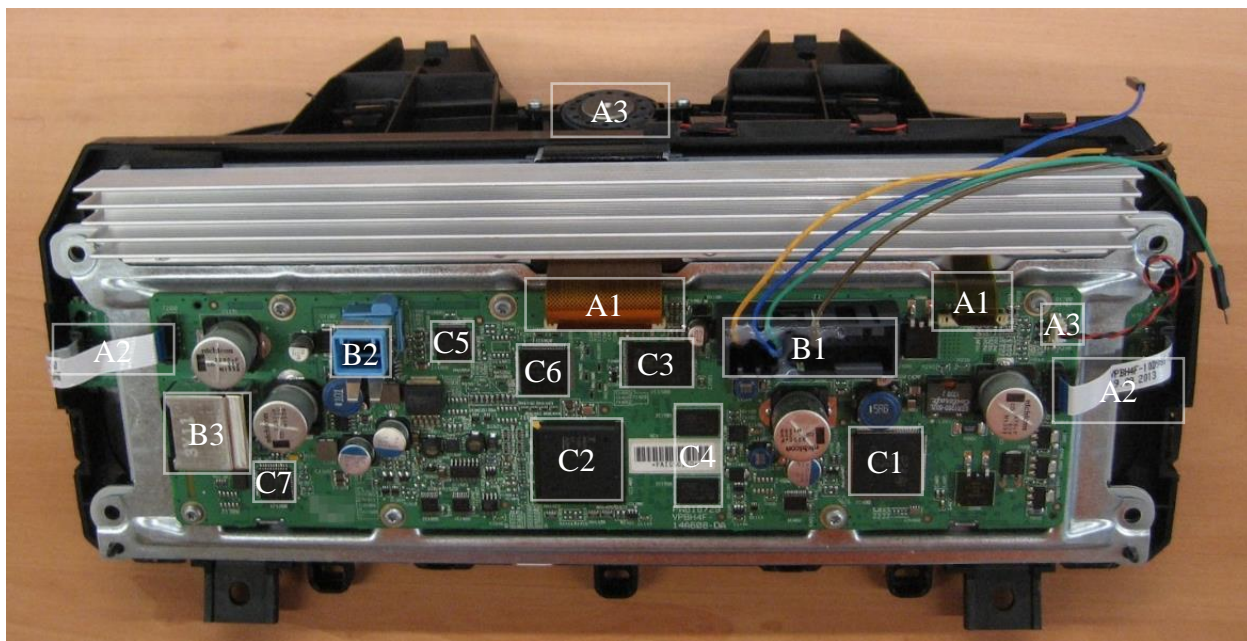


Figura 5 - Fotografia com a placa do painel de instrumentos (com anotações)

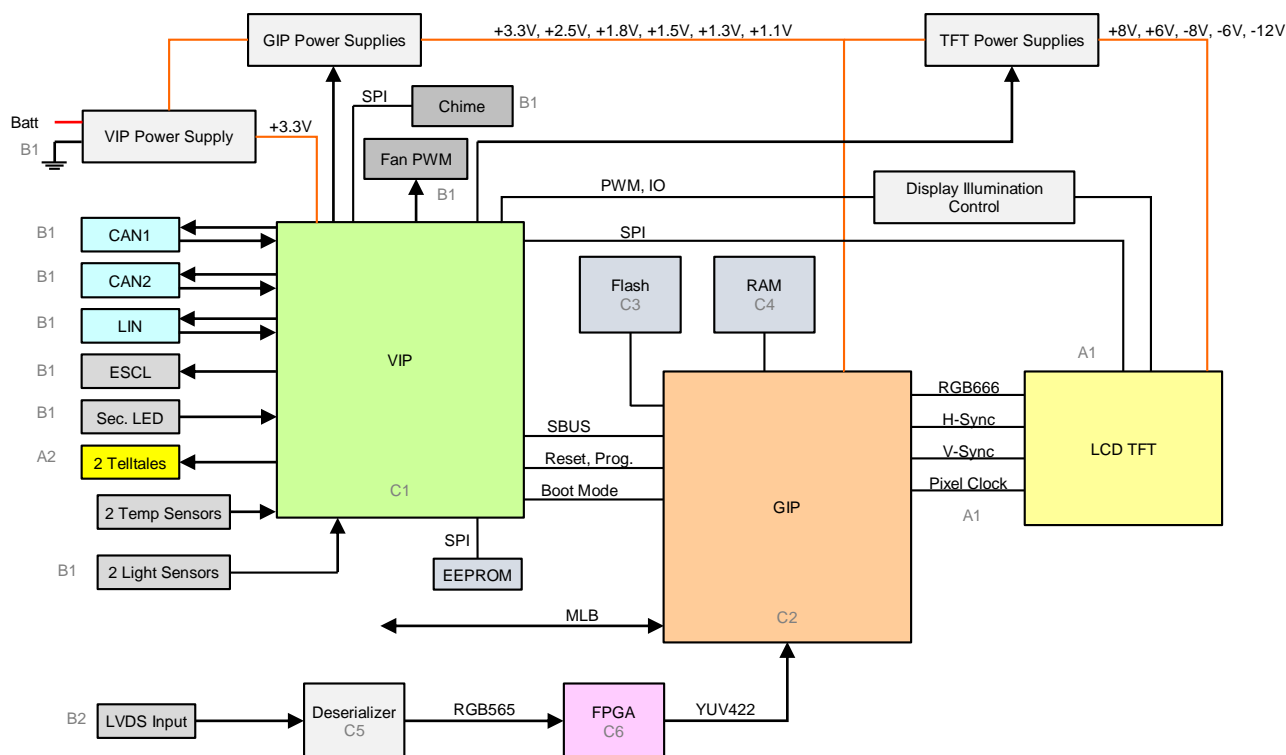


Figura 6 - Esquema mais detalhado das interfaces entre componentes do painel de instrumentos e/ou restantes elementos do veículo

É possível cruzar a notação da Figura 5 com o diagrama da Figura 6, onde o painel de instrumentos é apresentado com algum detalhe adicional, sendo agora visíveis (de forma esquemática) alguns dos componentes que fazem parte do painel assim como as ligações que existem entre os mesmos.

O esquema foi adaptado de um que se encontra na documentação do painel de instrumentos [30] e que aparenta referir-se à próxima geração de interfaces (mencionada como NGI (*Next-Generation Interface*) e que já apresenta uma série de componentes diferentes dos que constituem o modelo disponibilizado. É possível identificar dois dos componentes principais, designados como o VIP (*Vehicle Interface Processor*) e o GIP (*Graphic Interface Processor*). O VIP é o microcontrolador TMS470 da Texas Instruments, responsável pela maior parte das interfaces (CAN, LIN, etc.) do painel de instrumentos com os restantes equipamentos do veículo e o GIP é o Fujitsu MB86R01 Graphics SoC (*System on a Chip*) “Jade” responsável pelo controlo da parte gráfica do painel.

Além destes elementos temos ainda de destacar a referência ao ecrã TFT-LCD, pelo facto de ser neste diagrama que surge pela primeira vez a alusão à profundidade de cor do mesmo, RGB666 que se traduz na utilização de 6 *bits* para a representação de cada uma das cores primárias aditivas RGB (*Red, Green and Blue* - Vermelho, Verde e Azul) que resulta na utilização de 18 *bits* no total por cada *pixel* da imagem do ecrã. São também mencionadas algumas das características previamente referidas: 12.3 polegadas de diagonal, num formato 8:3, e uma resolução de 1280 por 480 *pixéis*. Aparecem ainda destacadas as tensões necessárias na alimentação do ecrã LCD (+8V, +6V, -8V, -6V e -12V) que poderiam limitar ou influenciar a FPGA a escolher caso fosse necessário receber ou fazer o controlo “directo” (sem condicionamento) destes sinais.

Foi ainda acrescentada a parte do esquemático relacionada com a entrada de vídeo LVDS, que passa pelo *Deserializer* da National Semiconductor (DS90UR124) e pela FPGA Lattice (A307RRF8) para a conversão do espaço de cor antes de entrar no GIP. Tudo isto foi aferido com base a breves referências na documentação do painel de instrumentos [30].



Para além da análise feita por inspecção visual, o estudo relativo ao painel de instrumentos baseou-se também na documentação disponibilizada, e da qual foram retirados e adaptados alguns dos esquemas apresentados [31], contudo, a informação fornecida encontrava-se por vezes limitada por questões de confidencialidade e/ou segredo industrial. Da documentação é possível retirar informações importantes como a resolução (2XVGA de 1280 pixéis horizontais e 480 verticais), o *aspect ratio* (proporção da tela) (8:3) e a dimensão (12.3” de diagonal do ecrã, com uma área activa de 291.84 mm por 109.44 mm).

Fica ainda aqui uma nota relativa à resolução do ecrã identificada como 2XVGA ( [32], [33] e [34]). Depreende-se, devido à escassez de referências, que se trata de uma resolução pouco comum e fora dos vários *standard VGA (Video Graphics Array)* e suas extensões. Tal poderá dever-se ao facto de a proporção da tela (8:3) ser algo incomum mesmo para ecrãs de tamanho mais amplo (*wide*) que são geralmente de formato 16:9.



## 4 Ligação com o ecrã

Neste capítulo pretende-se dar conta do trabalho de pesquisa desenvolvido para obter uma melhor compreensão sobre a interface do painel de instrumentos com o ecrã LCD. Eram necessárias mais informações para além das características já anotadas neste documento (resolução, proporções e dimensões). Assim, começou-se novamente por uma análise visual do equipamento, na procura de alguma referência ou outras informações que pudessem úteis cruzando-as posteriormente com os detalhes apresentados na documentação. Analisando a fita principal que liga o ecrã à placa principal do painel de instrumentos podemos verificar, de forma quase imediata, duas particularidades: A primeira é de que estamos perante um circuito impresso flexível (FPC - *Flexible Printed Circuit*) e não apenas uma simples fita de ligação; a segunda é que a disposição das “linhas” de ligação e da preocupação de aproximar algumas delas a dado ponto sugere que os sinais possam estar a ser transmitidos em pares diferenciais (em “acoplamento apertado” (*tight coupling*)).

É praticamente possível distinguir as linhas de sinais de dados, de relógio e sincronismo, alimentação e massa (*ground*) olhando para a sua espessura e disposição das mesmas, sendo que as primeiras (dados, relógio e sinc.) são as mais estreitas (é mesmo possível verificar que linhas de massa consecutivas se encontram todas elas ligadas, formando um pequeno “plano” ao mesmo potencial enquanto as outras (não consecutivas) acabam por ser as linhas mais largas de forma a isolar melhor as restantes).

Relativamente à fita secundária, as linhas são todas idênticas e não dá para aferir muito mais a partir da aparência da mesma. Mantinha-se a convicção de que esta ligação se destinava a funções secundárias do ecrã, em especial para a alimentação dos LEDs de retroiluminação.

Em termos visuais não foi possível analisar melhor estes circuitos uma vez que depois acabam por passar por debaixo do dissipador de calor que se encontra tanto aparafusado como colado à restante estrutura e que não foi retirado de forma a não danificar o equipamento.

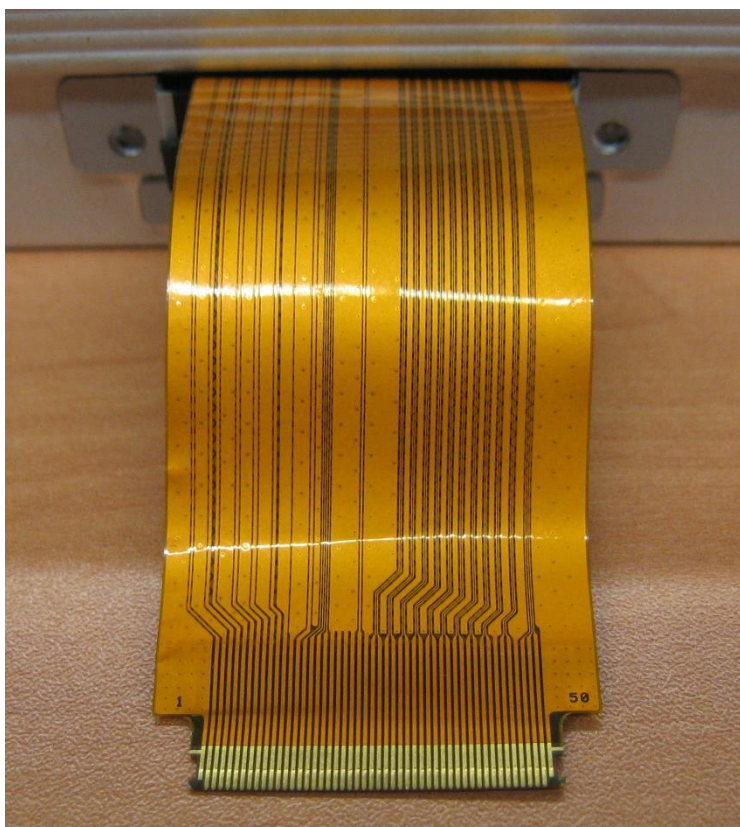


Figura 7 - Foto do FPC principal do ecrã

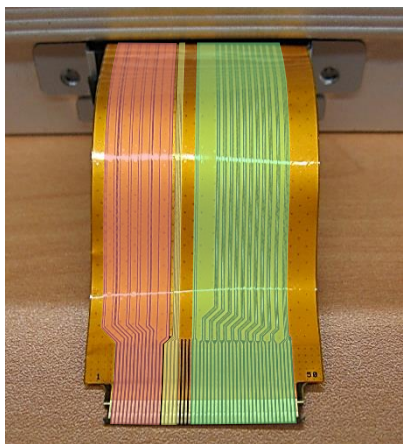
Ao retirar a placa do painel de instrumentos e o seu suporte metálico, revelam-se as referências do ecrã.

A partir da *datasheet* do ecrã LCD [31], confirma-se a maior parte das informações entretanto recolhidas: estamos perante um ecrã de 12.3”, 8:3 de 1280x480 pixéis RGB666 e cuja interface é feita através de dois FPCs, um de 50 pinos e 0.5mm de espaçamento entre pinos (*pitch*) para o painel propriamente dito e outro de 20 pinos com o mesmo *pitch* para a retroiluminação (*backlight*).

Ainda, desta mesma documentação, percebe-se que é possível configurar o ecrã através de uma comunicação SPI (*Serial Peripheral Interface*) e temos também informação sobre medidas e detalhes construtivos. Depois, de relevante, temos acesso às tabelas de “atribuição de pinos” (*pin assignment*) de ambos os FPC, sugerindo também os conectores que devem ser usados para a ligação (série 9687S da IRISO).

Temos a informação completa sobre o FPC de 50 pinos e quais os pinos dedicados à alimentação do ecrã, massa (*ground* (GND)), comunicação SPI, dados sobre a cor de cada *pixel*,

relógio e sincronismo entre outros (teste e selecção de modo de funcionamento) que define se o ecrã se encontra a funcionar em modo RSDS (*Reduced Swing Differential Signal*) ou modo TTL (*Transistor-Transistor Logic*) (*single-ended signaling*) apresentando posteriormente a informação analógica para o FPC de 20 pinos.



Legenda e notas:

#### Alimentação

Tensões potencialmente perigosas para FPGAs, abaixo de 0V e acima de 3.3V (ou 5V).

#### Dados da cor e sinais de sincronismo

Informação crucial para o sistema que pretendíamos implementar.

#### Comunicação SPI

Usado para configuração do controlador do ecrã.

Poderia ser interessante “interceptar” para determinar alguns parâmetros de configuração do ecrã.

Figura 8 - Foto do FPC principal do ecrã (com anotações)

A confirmação relativa ao modo de funcionamento do ecrã só poderia vir a ser esclarecida determinando o valor do sinal do pino “TR” com recurso a um osciloscópio ou criando um circuito digital de teste na FPGA, sendo que as notas presentes na *datasheet* do ecrã apontam no sentido de os sinais estarem a ser transmitidos em modo TTL (*single-ended*) e não de forma diferencial uma vez que indicam que as informações sobre o modo RSDS são apenas para referência e que seriam necessários alguns testes e alterações nas instalações de produção. Segundo o fabricante, a interface do ecrã tanto poderá ser configurada para funcionar em modo LVDS/RSDS ou TTL, consoante os requisitos do sistema (embora a versão parcial da *datasheet*, disponibilizada pelo fabricante, apenas referisse o modo TTL). De forma a precaver ambas as situações seria importante escolher uma FPGA que pudesse lidar com ambos os tipos de sinal.

No *datasheet* destaca-se ainda a informação relativa ao valor recomendado de VCC ser 3.3V o que faz com que o valor máximo para os sinais SPI, os dados, sinais de sincronismo e relógio esteja limitado por este valor, sendo que os restantes valores variam entre -12V a 8V), consumos de corrente, etc. [31].

Temos ainda os diagramas temporais relativos à transmissão de dados nos diversos modos que são também eles muito importantes de forma a sincronizar a informação que pretendo receber, permitindo também desprezar os dados enviados nos períodos de *blanking*, acompanhada por uma tabela com todos os parâmetros temporais entre os quais se deve realçar a frequência do sinal de relógio (41.34MHz) e a “frequência vertical” (58.71Hz aproximadamente 60Hz que é o padrão neste tipo de equipamentos uma vez que se traduz numa “cadência” de *frames* (*frame rate*) de 60FPS (*Frames Per Second*)) [31].

Tanto os valores de tensões eléctricas dos sinais, como a sua frequência máxima tiveram influência na definição dos requisitos para a FPGA, assim como no projecto da interface entre os sistemas.

No final temos outras informações, como as sequências necessárias para ligar e desligar o ecrã de forma correcta, uma tabela que faz corresponder a algumas cores a respectiva combinação RGB, os procedimentos para detecção de falhas e funções de autoprotecção, dados sobre as características ópticas aos quais se sucedem as informações sobre a comunicação SPI que podem assumir alguma importância caso quiséssemos capturar estes sinais usando a FPGA e ficar assim com acesso completo às configurações do ecrã, terminando com alguma informação complementar sobre a luminância ao longo do tempo de vida do ecrã, testes ambientais e de fiabilidade, requisitos de qualidade, etc. [31].







## 5 Escolha da FPGA

A escolha da FPGA que servisse os propósitos do presente trabalho deveria cumprir os seguintes requisitos:

(i) a FPGA tem de disponibilizar pinos de entrada/saída (*I/O – Input/Output*) suficientes para capturar, pelo menos, os sinais de dados, sincronismo e de relógio do sinal de vídeo a ser enviado para o ecrã (tendo ainda em conta a frequência dos mesmos, devendo precaver também a possibilidade de estes sinais estarem a ser transmitidos em pares diferenciais);

(ii) a FPGA tem também de ter uma memória externa RAM (*Random Access Memory*) de 512MB a 1GB de forma a conseguir armazenar alguns dos *frames* que fossem enviados para o ecrã;

(iii) a FPGA deve ter além da parte de lógica programável (PL - *Programmable Logic*), característica comum a todas as FPGAs, um outro processador *hard-core* (processador ARM (*Acorn RISC (Reduced Instruction Set Computing) Machine*, posteriormente *Advanced RISC Machine*), por exemplo) de forma a ter mais possibilidades de processamento e executar acções paralelamente às implementadas na parte de PL;

(iv) necessita ainda de um conector de alta-densidade de pinos (e, conseqüentemente, *pitch* baixo) de forma a ser possível fazer todas as ligações necessárias num espaço reduzido;

(v) ter uma interface USB (*Universal Serial Bus*) (2.0 ou 3.0) ou Gigabit Ethernet para que possa cumprir com objetivos adicionais, ao enviar os *frames* para outro computador para posterior processamento (como a SVF – *Software Validation Facility*, disponível na CRITICAL) com uma cadência suficientemente rápida completando assim a parte de captura de imagens;

(vi) e por fim as ferramentas de desenvolvimento e suporte deveriam, obrigatoriamente, ser gratuitas.

Foram procurados equipamentos que cumprissem com os requisitos apresentados dentro da oferta dos principais fabricantes de FPGAs, a Altera e a Xilinx. Por recomendação de engenheiros mais seniores da CRITICAL Software foi aconselhada a escolha de uma FPGA da Xilinx devido a experiências positivas que tiveram com produtos desta marca e com as suas ferramentas de desenvolvimento, para além de ser a empresa líder do mercado das FPGAs. De todas as opções consideradas houve uma que se destacou, a placa ZedBoard, ao cumprir com todos os parâmetros enunciados ao preço mais competitivo e incluir uma das soluções mais avançadas que a Xilinx disponibiliza de momento, um SoC (*System-on-a-Chip*) Zynq-7000.

Na tabela 1 são apresentadas as FPGAs que foram alvo de pesquisa para a realização do presente trabalho.

Tabela 1 - Lista de algumas FPGAs e suas características

Nome	Preço	RAM	Proc. h. c.	USB	GbE	Exp. a. d.	IDE
Nexys2 Spartan-3E FPGA Board	\$200	16MB	×	✓	×	Hirose FX2	ISE
Nexys3 Spartan-6 FPGA Board	\$270	16MB	×	✓	✓	VHDC	ISE
Spartan 3E Starter Board	\$299	64MB	×	×	×	Hirose FX2	ISE
Nexys4 Artix-7 FPGA Board	\$320	16MB	×	✓	✓	×	ISE
ZedBoard Zynq-7000 Development Board	\$395	512MB	✓	✓	✓	FMC - LPC	Vivado
Atlys Spartan-6 FPGA Development Board	\$419	256MB	×	✓	✓	VHDC	ISE
Xilinx Zynq-7000 All Programmable SoC ZC702 Evaluation Kit	\$895	1GB	✓	✓	✓	2xFMC - LPC	Vivado
Genesys Virtex-5 FPGA Development Board	\$1 099	128MB	×	✓	✓	2xVHDC	ISE
Xilinx Artix-7 FPGA AC701 Evaluation Kit	\$1 295	1GB	×	✓	✓	FMC-HPC	Vivado
Xilinx Kintex UltraScale FPGA KCU105 Evaluation Kit	\$2 495	2GB	×	✓	✓	FMC-HPC	Vivado

A ZedBoard resulta de uma parceria entre a Xilinx, a Avnet e a Digilent para reduzir os custos de produção e distribuição e torná-la assim mais acessível ao público. Possui também um *site* e comunidade próprios (provavelmente a comunidade mais vasta dentro desta categoria de produtos) tornando-se assim mais fácil de aceder a suporte técnico, documentação, projectos de referência e fórum de discussão [35].





# 6 Arquitectura

Muito do trabalho que aqui será descrito influenciou, de certa forma, a escolha do material necessário para o desenvolvimento deste trabalho. Contudo, só depois de escolhida e encomendada a FPGA é que se poderia definir por completo a arquitectura do sistema que se pretendia implementar.

Podemos separar a definição da arquitectura do sistema em duas partes distintas: uma “física” onde definimos a forma como iríamos ligar “electromecanicamente” os sistemas (PCB do painel de instrumentos, ecrã e FPGA) e outra que se prende com um funcionamento do sistema do ponto de vista da programação da FPGA, com a definição do caminho de dados.

## 6.1 Arquitectura física

Relativamente à parte física, era agora possível, cruzando as informações sobre a FPGA adquirida e o ecrã LCD, definir quais os tipos de conectores e cabos de que iriam ser necessários para efectuar as ligações. A FPGA disponibiliza três tipos de conectores de expansão, expondo pinos de entrada/saída (I/O):

- 1 Conector FMC (*FPGA Mezzanine Card*) LPC (*Low-Pin Count*) que expõe 68 pinos de I/O digitais quando configurados no modo *single-ended*, podendo também ser configurados como 34 pares diferenciais. No caso da Zedboard estes pinos são alimentados por uma tensão variável definida por um *jumper* na placa (pode alternar entre 1.8V, 2.5V ou 3.3V) [36]

- 5 Conectores compatíveis com Digilent Pmod™ (*Peripheral Modules*) (2x6) cada um expondo 8 pinos de I/O (mais 2 pinos de alimentação 3.3V e 2 de massa), sendo que um se encontra ligado directamente ao processador ARM (PS – *Processing System*) enquanto os restantes quatro permitem interface com a lógica programável (PL) do Zynq-7000 AP (*All Programmable*) SoC. Apenas dois destes suportam pares diferenciais.
- 1 Conector AMS (*Analog Mixed Signal*) ou XADC (*Xilinx Analog-to-Digital Converter*) que permite a entrada de alguns sinais tanto digitais como analógicos.  
(Informação adicional pode ser consultada em [37], págs. 21 a 25)

Optou-se pela ligação FMC por se tratar da interface de expansão principal, uma vez que providência um maior número de pinos de I/O, suportando a interface com pares diferenciais, não colocando portanto nenhum entrave à ligação com o ecrã, neste aspecto. Permite também desenhar uma solução mais compacta em termos de espaço já que este conector apresenta uma maior densidade de pinos quando comparado com os conectores Pmod que são mais indicados para outro tipo de periféricos (que por norma necessitam de menos pinos de I/O).<sup>1</sup>

Existia desde início a necessidade de fazer uma PCB (em alternativa poderia ser desenhado um FPC, mas, neste caso, tal não se justificava e representaria custos adicionais) para tratar da interface entre os três sistemas referidos (PCB do *painel de instrumentos*, ecrã e FPGA) e, escolhida a interface de expansão da FPGA a utilizar, estavam definidos os conectores de que iríamos precisar de montar nesta placa. Seriam necessários 3 conectores: um conector FMC LPC “macho” para a ligação com a FPGA, e dois conectores FFC (*Flexible Flat Cable*)/FPC de 50 pinos (da série IRISO 9687S ou compatível) para a ligação com a PCB do *painel de instrumentos* e com o ecrã. Seria também indispensável uma fita FFC de 50 pinos de forma a ligar a PCB do painel de instrumentos com a PCB que iríamos projectar (esta fita deveria ter, de preferência, os mesmos entalhes (*notches*) que o FPC do ecrã apresenta (Figura 7) de forma a que o encaixe seja mais seguro.

---

<sup>1</sup> Podemos encontrar alguns exemplos de periféricos de expansão já existentes que utilizam a interface FMC ou Pmod nas seguintes ligações (FMC [51], Pmod [52])

Para a arquitectura física do nosso sistema foram consideradas as seguintes alternativas:

- Uma ligação da FPGA seria ligada em paralelo com o ecrã criando como que uma derivação (em “T”) nas linhas de relógio, dados, e sincronismo, para que estas entrassem directamente tanto no ecrã como na FPGA (Fig. 9).

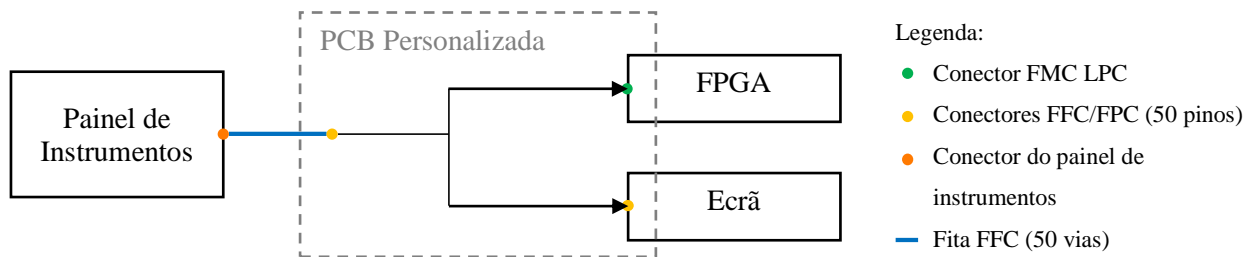


Figura 9 - Ilustração da primeira hipótese de arquitectura física

- Receber os sinais a serem enviados pelo painel de instrumentos na FPGA, em determinados pinos de I/O a definir (a funcionarem como entradas) e replicar os sinais recebidos noutros pinos distintos (que seriam definidos como saídas) enviando-os para o ecrã. Os três sistemas seriam ligados em série de acordo com a seguinte ordem: Painel de Instrumentos → FPGA → Ecrã (Fig. 10).

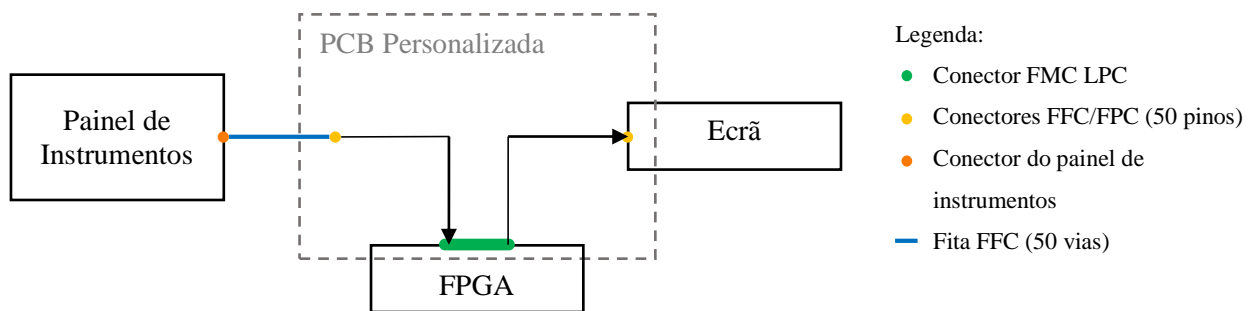


Figura 10 - Ilustração da segunda alternativa de arquitectura física

A decisão recaiu sobre a segunda opção uma vez que, na primeira, a impedância das linhas de entrada pode ser alterada significativamente, já que os sistemas se encontrariam ligados em paralelo, o que poderia comprometer a integridade destes sinais. A segunda alternativa pretende compensar este aspecto negativo mas em contrapartida irá necessitar de utilizar o dobro dos pinos de entrada/saída: a primeira metade irá servir como pinos de entrada (tal como na primeira opção) e a segunda como pinos de saída de sinais.

## 6.2 Arquitectura do caminho de dados (*datapath*)

A arquitectura do caminho de dados foi, inicialmente, definida de forma conceptual, para que servisse, posteriormente, de referência para implementação na FPGA. Deve-se referir que a maior parte das decisões tomadas, relativamente a esta arquitectura, assim como à arquitectura física apresentada na secção anterior, tiveram por base o que foi discutido em algumas reuniões com engenheiros seniores da CRITICAL Software e acabam por transpor para o papel muitas das suas sugestões para este trabalho. Segue-se apresentação da arquitectura do sistema.

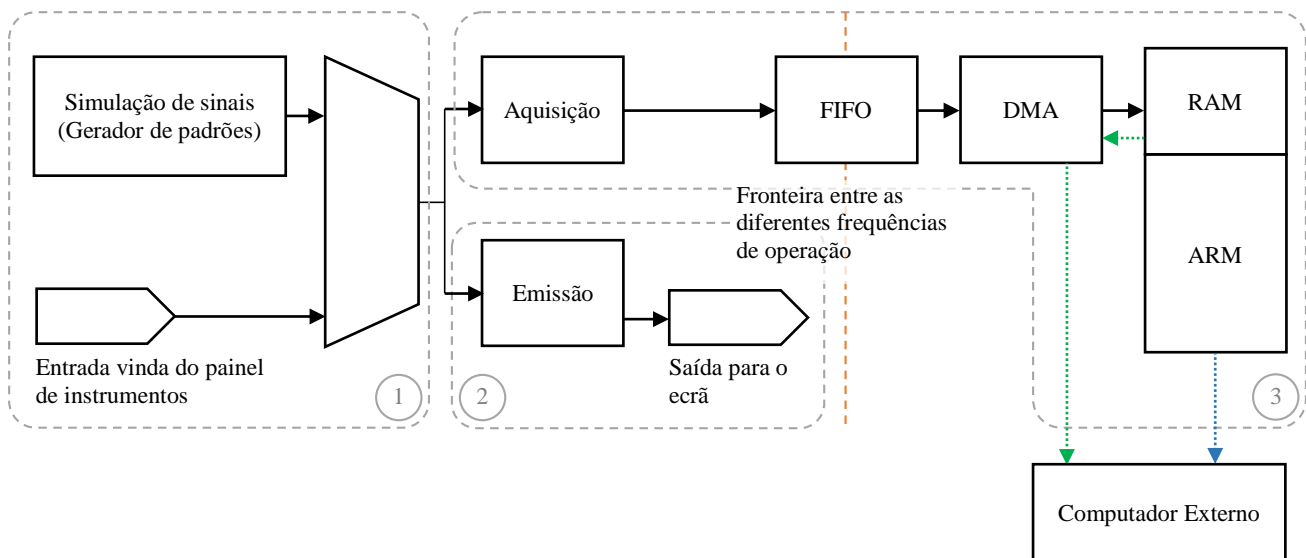


Figura 11 - Diagrama de blocos da arquitectura do caminho de dados

O sistema deve, antes de mais, permitir alternar entre a captura de imagens vindas da placa do painel de instrumentos ou das imagens vindas de um simulador (gerador) de sinais implementado na FPGA que permita a geração de padrões para efeitos de teste do sistema. Os sinais passariam por dois caminhos distintos: um que se limitava a emitir os sinais para o ecrã do painel de instrumentos, para que continuassem a aparecer neste ecrã, enquanto o segundo trata da aquisição dos sinais de forma a passá-los para a memória da FPGA. Como os sistemas funcionam a diferentes frequências de relógio, é praticamente imperativo que nesta parte de aquisição das imagens fosse usado um *buffer* FIFO (*First In, First Out*) [38] de forma a lidar com esta situação, sendo que a parte de entrada funcionaria à frequência de relógio do ecrã e a parte de saída poderá funcionar às frequências permitidas pela FPGA, podendo ainda acumular temporariamente algumas amostras em caso de possíveis paragens (*stall*) causadas por indisponibilidade dos recursos a que pretendemos aceder (p. ex. acessos concorrentes à memória) guardando-as até que estas possam prosseguir pelo restante caminho de dados (*datapath*).



Posteriormente, pretende-se enviar os dados recebidos para a RAM existente na placa ZedBoard (512MB DDR3) em detrimento da possibilidade de usar BRAM (*Block RAM*) disponíveis no “tecido” de lógica programável (PL) da FPGA (140 blocos de 36Kb cada, 560KB no total, de acordo com [39] (a versão do SoC Zynq-7000 que a ZedBoard disponibiliza é a Z-7020, Package CLG484)) para criar uma memória nesta região, poupando-os para os restantes elementos necessários ao *datapath*. Esta opção foi tomada já que, mesmo usando todos os blocos BRAM disponíveis para criar uma memória, estes seriam, ainda assim, insuficientes para armazenar uma única imagem do fluxo de vídeo que está a ser enviado para o ecrã (1350 kB por *frame*, ver expressão (1)), não considerando possibilidades de compressão ou redução do espaço de cor da imagem que vão contra os objectivos do trabalho, onde se inclui a preservação total das imagens e de toda a informação que estas representam.

$$\begin{aligned} \frac{\text{n}^\circ \text{ de bytes úteis}}{\text{frame}} &= 1280 \times 480 \left( \frac{\text{píxeis}}{\text{frame}} \right) \times 18 \left( \frac{\text{bits}}{\text{píxeis}} \right) = \\ &= 11059200 \frac{\text{bits}}{\text{frame}} = 1350 \frac{\text{kB}}{\text{frame}} \end{aligned} \quad (1)$$

Uma solução que fizesse apenas uso dos blocos BRAM poderia trazer vantagens em termos de desempenho (do ponto de vista temporal) em detrimento da possibilidade de armazenar imagens completas. Poderia ser interessante num projecto de integração da FPGA com outros computadores em que o armazenamento das imagens poderia ser parcial (limitado a algumas linhas da imagem) e os dados fossem enviados para outra máquina para um armazenamento mais completo e efectivo e para posterior processamento. Esta solução surge na linha daquilo que se pretende como possível evolução e/ou desenvolvimento deste trabalho.

Neste trabalho o sistema foi abordado numa perspectiva em que deveria funcionar de forma isolada (*stand-alone*) e para tal deveria armazenar imagens completas. Assim é impreterivelmente necessário recorrer à memória RAM da placa e a mecanismos de DMA (*Direct Memory Access*) de forma a libertar o processador ARM (*dual-core*) da maioria do “peso” deste procedimento [40]. Uma vez na RAM os dados estarão também acessíveis pelo processador ARM. Aí é possível proceder a posterior processamento ou enviá-los para outra máquina com o mesmo intuito, por USB ou via Ethernet, (linha a azul - Figura 11) sendo que esta tarefa também poderia ser desempenhada por módulos implementados na FPGA (PL) (linha a verde - Figura 11). Estes procedimentos, de processamento posterior e/ou envio para outras máquinas, encontravam-se além

da fronteira do âmbito definido para este trabalho que se limitava à captura das imagens. Como já foi referido, estes serão os passos seguintes deste trabalho, no que se trata a um possível desenvolvimento futuro.

Esta arquitectura serviu de base para o desenvolvimento de todo o trabalho, mas à medida que a sua implementação foi ficando concluída foram acrescentados alguns elementos adicionais que não estavam ainda previstos como a utilização das saídas de vídeo (VGA, HDMI), interfaces de comunicação (USB, Ethernet) da ZedBoard (capítulo 9).





# 7 Implementação

Neste capítulo é apresentada a implementação da arquitectura, apresentada no capítulo anterior, na FPGA ZedBoard.

## 7.1 Vivado Design Suite

Grande parte do trabalho foi desenvolvido usando a Vivado Design Suite<sup>2</sup> da Xilinx de forma a projectar a plataforma de *hardware* que desejávamos carregar na FPGA. Este *software* integra todas as fases do processo de desenvolvimento: para criar podemos optar por escrever o código na própria aplicação, carregar ficheiros já existentes ou em alternativa usar o “IP *Integrator*” que permite criar um diagrama de blocos composto por “IP (*Intellectual Property*) *Cores*”, seguindo-se a possibilidade de simular (como no caso de *test benches*), abrir o diagrama correspondente à análise RTL (*Register Transfer Level*), sintetizar, implementar e por último programar a parte de lógica programável. Todas estas opções podem ser vistas na barra que se situa do lado esquerdo da interface principal do programa denominada “*Flow Navigator*”.

A Figura 12 apresenta ainda o sumário de um projecto criado especificamente para a ZedBoard, aparecendo inclusivamente a foto da placa, já que é preciso definir qual o dispositivo em que queremos posteriormente implementar o projecto para que o programa possa lidar com as diferenças existentes ente os diversos dispositivos de forma automática. Depois de completada a fase de síntese e de implementação de um projecto, este sumário também apresenta o relatório das mesmas, indicando possíveis erros assim como as estimativas de consumo energético, utilização de recursos, características temporais (atrasos, caminho crítico, etc.) ou violações das DRC (*Design Rule Check(ing)*).

---

<sup>2</sup> Os procedimentos descritos irão estar de acordo com a versão 2015.2 do Vivado, a última disponível à data da escrita deste documento

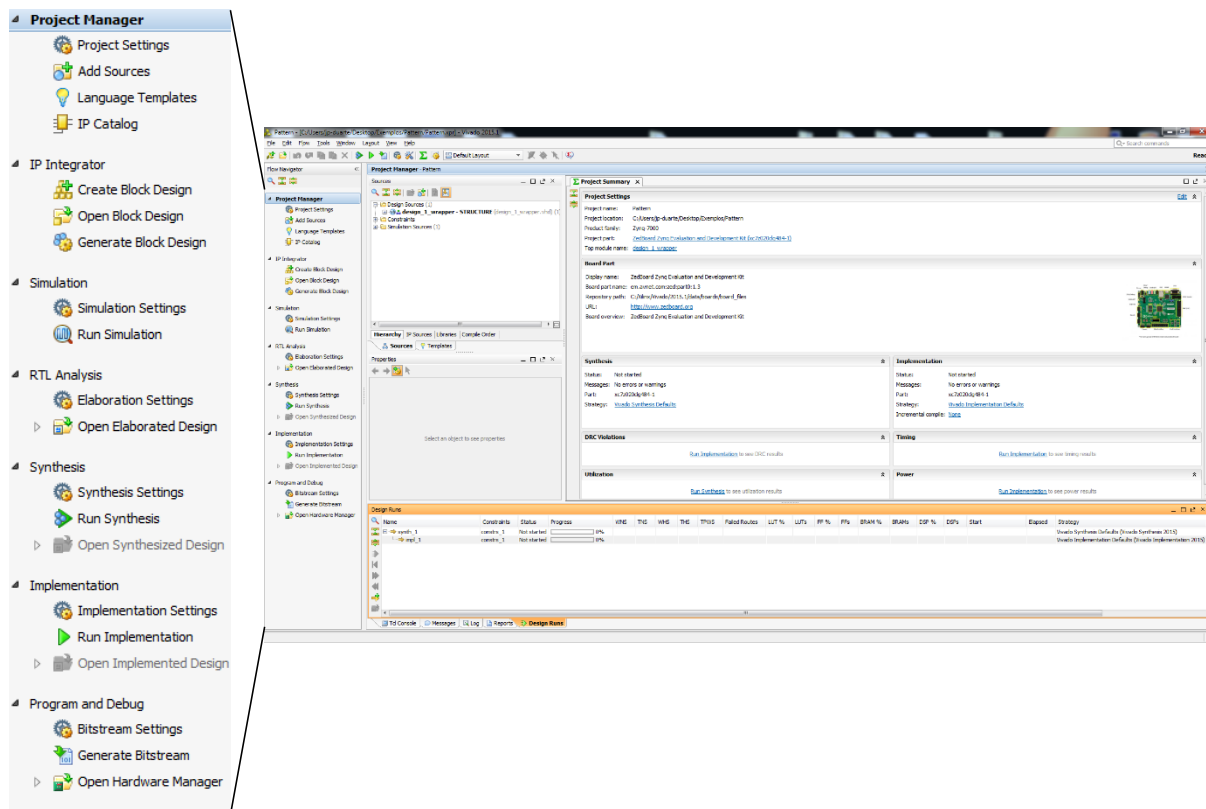


Figura 12 - Interface principal da *suite* Vivado com destaque para o “*Flow Navigator*”

## 7.2 Construção do *datapath*

Definida a arquitetura, avançou-se para a definição do circuito digital. Os módulos começaram a ser incluídos de forma gradual e iam sendo executados testes à medida que iam sendo acrescentados.

Será aqui apresentada uma versão simplificada do caminho de dados em que se procede à captura das imagens provenientes de um módulo de simulação que gera padrões. A base criada para esta situação é então comum a versões posteriores mais elaboradas em que já foi incluída a possibilidade de receber os sinais vindos do painel de instrumentos, e onde também foram acrescentados os módulos de emissão, quer para o ecrã do painel quer para ecrãs externos através das saídas de vídeo da ZedBoard (HDMI, VGA).

De seguida, apresentam-se os diagramas de blocos desta implementação, primeiro na sua versão esquemática (Figura 13), para a enquadrar na arquitetura, e depois na versão criada em “*Vivado Integrator*” (Figura 14).

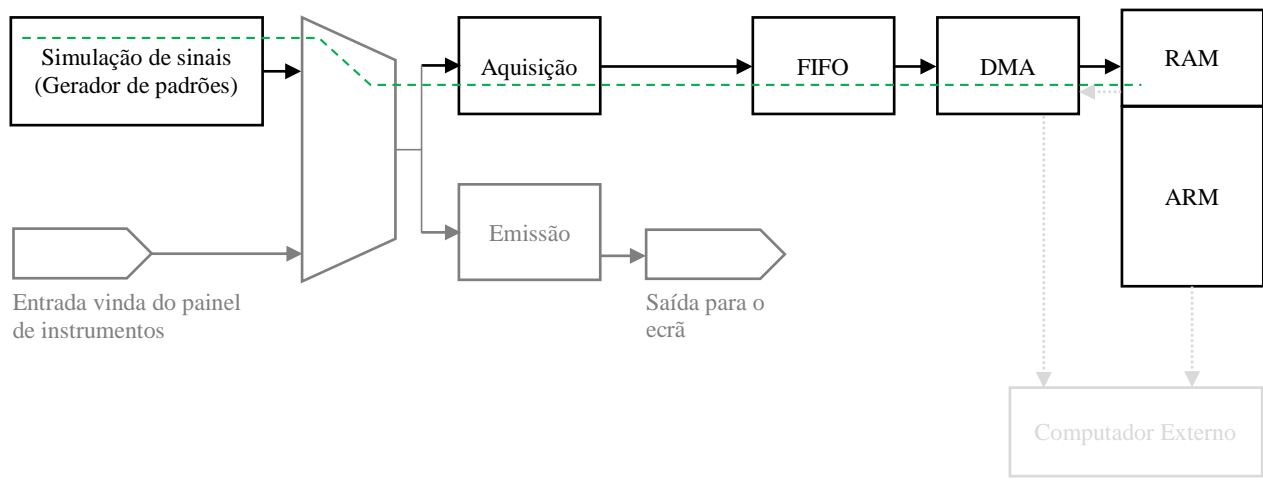


Figura 13 - Diagrama de blocos (esquemático) da implementação básica

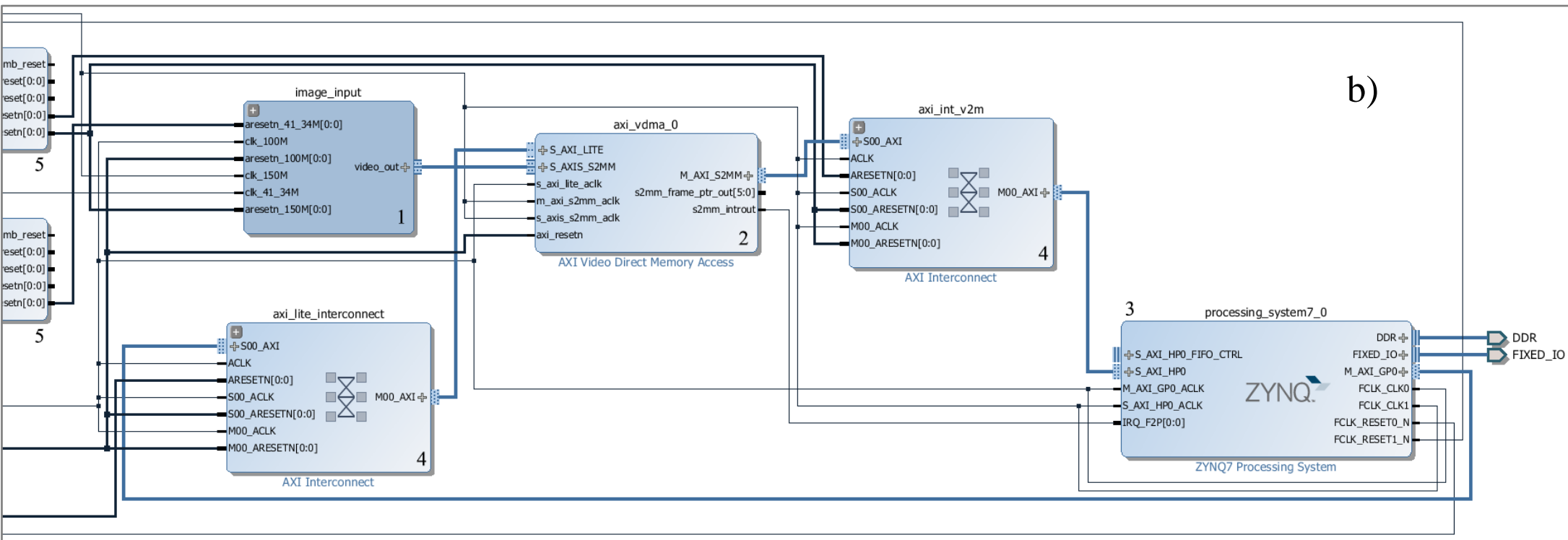
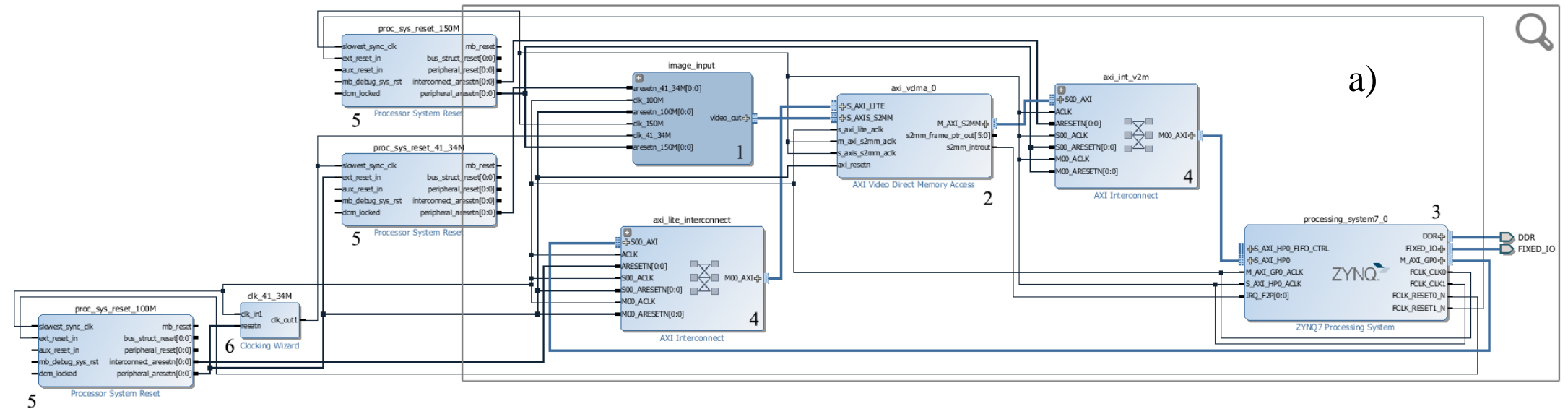


Figura 14 - (a) Diagrama de blocos da implementação básica; (b) Pormenor do diagrama



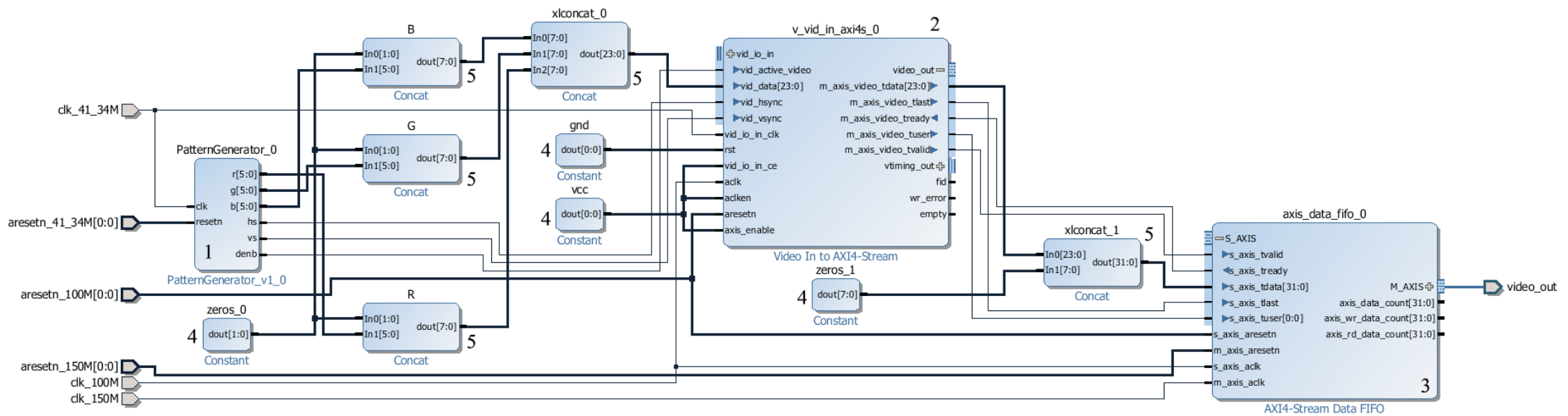


Figura 15 - Composição do bloco “image\_input”

Na Figura 14 é possível visualizar o caminho de dados completo onde se destacam 3 blocos: O bloco composto “image\_input” que, como pode ser visto na Figura 15, contém o bloco “Video In to AXI4-Stream” e o bloco “AXI4-Stream Data FIFO” permitindo assim a captura das imagens recebidas, neste caso geradas pelo gerador de padrões que pode ser substituído por entradas externas vindas do painel de instrumentos via interface FMC. São ainda usados vários módulos “Concat” e “Constant” de forma a preencher as linhas de dados. Tanto o bloco “Video In to AXI4-Stream” como o bloco “AXI4-Stream Data FIFO” integram FIFOs com a profundidade configurada para 2048 “palavras” para que pudessem armazenar pelo menos uma linha completa da imagem do painel de instrumentos (1280 *pixéis* de largura). No módulo “AXI4-Stream Data FIFO” foi activada a possibilidade de serem usados relógios assíncronos, e assim, a saída já está de acordo com relógio do restante caminho de dados. O bloco “AXI Video Direct Memory Access” gere a transferência dos dados para a memória RAM da placa tratando da conversão dos sinais de uma interface AXI4-Stream para uma AXI4 (*Memory-Mapped*). Os dados seguem para o bloco seguinte. Foi activada a funcionalidade de sincronizar o módulo usando o sinal “tuser” que tem vindo a ser propagado pelos módulos anteriores (“Video In...” e “...FIFO”) e identifica a primeira amostra de cada imagem fazendo com que esta surja alinhada correctamente na memória (primeira amostra de cada *frame* no respectivo endereço de memória), algo que não acontecia nos primeiros testes realizados.

Por último, o bloco “ZYNQ7 Processing System” que representa, como o nome indica, a parte correspondente ao processador (PS) ARM deste SoC Zynq. É essencial a todos os projectos que queiram fazer uso deste processador. Foram activadas duas interfaces AXI: uma HP (*High Performance*) que recebe os dados vindos do módulo VDMA (*Video DMA*) e outra GP (*General Purpose*) para a configuração do mesmo módulo (VDMA). É ainda responsável pela gestão dos sinais de relógio (PLL (*Phase-Locked Loop*)) e pelos sinais de *reset*.

Os restantes blocos presentes são na maioria gerados automaticamente pelo “IP Integrator” e servem essencialmente para sincronizar os sinais de *reset* com os respectivos relógios (blocos “Processor System Reset”) intermediar as interfaces AXI (blocos “AXI Interconnect”) permitindo que a mesma interface AXI do PS seja usada por vários IP *cores* do restante *datapath*, o que não se verifica neste projecto mas é um procedimento bastante comum (especialmente para ligar várias interfaces AXI4-Lite a um único porto AXI do PS, permitindo assim a configuração, via PS, de diversos IP *cores*). O bloco “Clocking Wizard” permite simular a entrada do sinal de relógio (neste caso foi usado o módulo MMCM (*Mixed-Mode Clock Manager*) para converter o sinal vindo do PS para a frequência desejada).

Depois de concluído o desenho, é necessário criar a *bitstream* com que a FPGA irá ser, posteriormente, programada. Para tal, é usada a opção “*Generate Bitstream*” que pressupõe a prévia execução das fases de síntese e implementação (caso ainda não tenham sido executadas, o programa irá fazê-lo de forma automática). Finalizada esta operação, é agora possível exportar a plataforma de *hardware* para o SDK (*Software Development Kit*) da Xilinx e criar o *software* que irá correr no processador ARM (PS).

### 7.2.1 *Desenvolvimento de software*

Para que uma aplicação faça uso simultâneo da lógica programável (PL) e do *processing system* (PS) no modo *standalone*, *i.e.* sem carregar um sistema operativo completo (já que algumas versões do Linux são também suportadas por este processador), são necessárias 3 partes distintas:

- (i) a plataforma de *hardware* exportada do Vivado para o SDK;
- (ii) a aplicação propriamente dita que será desenvolvida no SDK e pode ser escrita em C ou C++;
- (iii) e o chamado *Board Support Package* (BSP) que introduz uma camada de *software* intermédia que providencia acesso às funcionalidades básicas do processador como as caches, interrupções e excepções permitindo também interações de entrada e de saída, abortar ou terminar a execução, etc. incluindo ainda os controladores dos módulos criados na lógica programável<sup>3</sup>.

Este BSP é normalmente gerado de forma automática quando se pretende criar uma nova aplicação podendo ser partilhado por diversas aplicações. Pode ainda ser configurado, adicionando bibliotecas de forma a acrescentar funcionalidades sendo possível escolher os controladores e editar os comandos e *flags* de compilação.

Existem alguns *templates* de aplicação (“Hello World”, testes de memória e periféricos e alguns *benchmarks*) e é também possível importar alguns exemplos de código para controlar os diferentes periféricos que tenhamos incluído na lógica programável.

---

<sup>3</sup> De acordo com o que é apresentado no SDK quando aberto o ficheiro “system.mss”, parte integrante do BSP

### 7.2.2 Notas sobre Interfaces AXI

A grande maioria dos IP *cores* que já se encontravam disponíveis no “IP *Integrator*” do Vivado utiliza interfaces AXI (*Advanced eXtensible Interface*). [41]

Alguma pesquisa revela que estes protocolos fazem parte da especificação AMBA (*Advanced Microcontroller Bus Architecture*) que descreve aqueles que são os *standards “de facto”* desta indústria.

Na quarta versão desta especificação (AMBA 4) são definidos os seguintes protocolos que viríamos a utilizar, entre outros [42]:

- AXI4 – Para ligações mapeadas em memória (*memory-mapped*), que providenciam uma performance elevada. É fornecido um endereço seguido da transferência de um *burst* de dados até 256 “palavras” (*words*).
- AXI4-Lite – Uma versão simplificada da ligação que suporta apenas a transferência de uma “palavra” por ligação. Também é *memory-mapped* o que implica que, neste caso, sejam transferidos um endereço e uma “palavra”.
- AXI4-Stream – Para um fluxo de dados de alta velocidade, suportando *burst’s* sem restrições de tamanho. Neste protocolo não existe mecanismo de endereçamento e é indicado para um fluxo de dados directo entre a origem e o destino (*non memory mapped*). [43]

Seria necessário integrar interfaces deste tipo nos módulos criados para que a comunicação com os restantes *cores* fosse possível. Olhando para a breve descrição destes três tipos de interface AXI4 era possível afirmar que uma interface do tipo AXI4-Stream é a mais adequada ao fluxo de vídeo que pretendíamos capturar.

Contudo, não fazia sentido, adaptar desde logo o módulo gerador de padrões para esta interface já que o objectivo deste bloco passava também por simular a forma como os dados iriam entrar, no futuro, na placa e que não se enquadra logo no formato usado neste protocolo. Pareceu-me mais lógica a possibilidade de dotar a parte de aquisição de dados com este tipo de interface convertendo assim o sinal vindo do painel de instrumentos (ou do simulador) para este padrão e assim poder usar os módulos já existentes no restante *datapath*.

Para tratar desta conversão foi utilizado o IP *core* “Video In to AXI4-Stream” que transforma o conjunto de sinais que compõe o sinal de vídeo (dados, relógio sincronismo), num sinal de acordo com o padrão AXI4-Stream.

### 7.2.3 Módulos criados

Ainda antes da escolha e aquisição da FPGA foram criados alguns módulos genéricos que poderiam vir a ser úteis no seguimento do projecto:

- (i) um módulo que gerasse um padrão para que fosse mais fácil verificar o correcto funcionamento dos restantes elementos do *datapath*;
- (ii) um módulo que recebesse estes dados e os guardasse sendo também responsável por validar esta informação, ignorando os pixéis enviados nos períodos de sincronismo e esperando por o início de uma nova imagem
- (iii) e um módulo que seria responsável pela interface com a memória e que deveria ser do tipo FIFO, como referido anteriormente, por se adequar melhor a este caso. Estes módulos foram desenvolvidos em VHDL (VHSIC (*Very High Speed Integrated Circuits*) *Hardware Description Language*) para que pudessem ser implementados independentemente da FPGA que viesse a ser adquirida, uma vez que esta linguagem é, a par da linguagem Verilog, das mais utilizadas para o projecto circuitos digitais e é suportada por praticamente todas as ferramentas de desenvolvimento para equipamentos deste tipo. Acresce-se, a título pessoal, que a linguagem VHDL era, das duas, aquela com que estava mais familiarizado, já que é leccionada e utilizada ao longo do curso (MIEEC).

Destes módulos o único que viria a integrar o *datapath* é o módulo responsável pela gerações de padrões e que pode ser visto na imagem 15 com o nº 1. Todos os restantes módulos incluídos no caminho de dados apresentado fazem parte da biblioteca padrão de IP *cores* do Vivado.

O gerador de padrões irá ser apresentado de seguida enquanto os restantes módulos desenvolvidos, de aquisição e interface com a memória, irão apenas ser apresentados no capítulo 9 destinado ao trabalho complementar, uma vez que não vieram a ser incluídos no *datapath*, tendo sido substituídos pelos módulos “Video In to AXI4-Stream” e “AXI4-Stream Data FIFO” da biblioteca padrão.

### 7.2.3.1 Gerador de Padrões

Com este módulo pretendíamos criar de padrões para efeitos de teste do restante *datapath*.

Em anexo é apresentado o código desenvolvido (Anexo A - 1) sem interrupções.

```
-----  
-- Company: iTGrow  
-- Engineer: João Pedro Aleixo Duarte  
--  
-- Create Date: 12/15/2014 02:11:13 PM  
-- Design Name:  
-- Module Name: PatternGenerator - Behavioral  
-- Project Name:  
-- Target Devices:  
-- Tool Versions:  
-- Description:  
--  
-- Dependencies:  
--  
-- Revision:  
-- Revision 0.01 - File Created  
-- Revision 1.00 - PatternGenerator Created - Detected Sync Error  
-- Revision 1.01 - Sync Error Corrected  
-- Revision 1.02 - Last Column Error Corrected  
-- Additional Comments:  
--  
-----  
library IEEE;  
use IEEE.STD_LOGIC_1164.all;  
use IEEE.NUMERIC_STD.all;  
  
library UNISIM;  
use UNISIM.VComponents.all;  
  
entity PatternGenerator is  
  generic (  
    WIDTH : natural := 1280; --horizontal display width in pixels  
    HFRONTPORCH : natural := 2; --horizontal front porch width in pixels  
    HSYNCWIDTH : natural := 2; --horizontal sync pulse width in pixels  
    HBACKPORCH : natural := 2; --horizontal back porch width in pixels  
    HPOL : std_logic := '0';  
    --horizontal sync pulse polarity (1 = positive, 0 = negative)  
  
    HEIGHT : natural := 480; --vertical display width in rows  
    VFRONTPORCH : natural := 2; --vertical front porch width in rows  
    VSYNCWIDTH : natural := 2; --vertical sync pulse width in rows  
    VBACKPORCH : natural := 2; --vertical back porch width in rows  
    VPOL : std_logic := '0';  
    --vertical sync pulse polarity (1 = positive, 0 = negative)  
  
    PTRN : natural := 0;  
  
    RWIDTH : natural := 6;  
    GWIDTH : natural := 6;  
    BWIDTH : natural := 6  
  );  
port (  
  clk : in std_logic;  
  resetn : in std_logic; -- Active low  
  r : out std_logic_vector(RWIDTH - 1 downto 0);  
  g : out std_logic_vector(GWIDTH - 1 downto 0);  
  b : out std_logic_vector(BWIDTH - 1 downto 0);
```

```

    hs : out std_logic;
    vs : out std_logic;
    denb : out std_logic
);
end PatternGenerator;

```

Até este ponto temos os comentários iniciais que fazem alusão a algumas revisões a que o código foi sujeito. A versão inicial deste módulo apresentava alguns erros e menos parâmetros configuráveis: não considerava a largura dos pulsos de sincronismo (*pulse width*), não permitia ajustar o número de *bits* usados para representar cada uma das cores nem a polaridade dos pulsos de sincronismo. Na declaração da entidade “PatternGenerator” é possível encontrar todos os parâmetros configuráveis (largura e altura da imagem a criar (“WIDTH” e “HEIGHT”), etc.) assim como as declarações das entradas e saídas do bloco. O bloco terá então como uma entrada de relógio (“clk”) e outra para que se reinicie (“resetn”), e à saída apresentará os dados de cor (“r”, “g”, “b”) assim como saídas de sincronismo (“hs”, “vs”, “denb”).

```

architecture Behavioral of PatternGenerator is

```

```

    constant HPERIOD : natural := HFRONTPORCH + HBACKPORCH + WIDTH;
    --total number of pixel clocks in a row
    constant VPERIOD : natural := VFRONTPORCH + VBACKPORCH + HEIGHT;
    --total number of rows in column

    signal hcount : natural range 0 to HPERIOD - 1 := 0;
    --horizontal counter (counts the columns)
    signal vcount : natural range 0 to VPERIOD - 1 := 0;
    --vertical counter (counts the rows)
    signal position : natural range 1 to WIDTH * HEIGHT := 1;
    --pixel counter (counts active pixels)

```

```

begin

```

```

    control : process (clk)
    begin
        if (rising_edge(clk)) then
            if (resetn = '0') then --reset asserted
                hcount <= 0; --reset horizontal counter
                vcount <= 0; --reset vertical counter
                position <= 1; --reset pixel counter

                hs <= not HPOL; --deassert horizontal sync
                vs <= not VPOL; --deassert vertical sync
                denb <= '0'; --disable display
            else
                --counters
                if (hcount < HPERIOD - 1) then --horizontal counter (pixels)
                    hcount <= hcount + 1;
                else
                    hcount <= 0;
                    if (vcount < VPERIOD - 1) then --vertical counter (rows)
                        vcount <= vcount + 1;
                    else
                        vcount <= 0;
                        position <= 1; --reset pixel counter
                    end if;
                end if;
            end if;
        end if;
    end process;

```

```

--horizontal sync signal
if (hcount >= HFRONTPORCH - 1 and hcount < HFRONTPORCH + HSYNCWIDTH
- 1) then
    hs <= HPOL; --assert horizontal sync pulse
else
    hs <= not HPOL; --deassert horizontal sync pulse
end if;
--vertical sync signal
if (vcount > VFRONTPORCH - 1 and vcount <= VFRONTPORCH + VSYNCWIDTH
- 1) then
    vs <= VPOL; --assert vertical sync pulse
else
    vs <= not VPOL; --deassert vertical sync pulse
end if;

--set display enable output
if ((hcount >= HFRONTPORCH + HBACKPORCH - 1 and hcount < HPERIOD -
1) and (vcount >= VFRONTPORCH + VBACKPORCH)) then --display time
    denb <= '1'; --enable display
    position <= position + 1; --count pixel
else --blanking time
    denb <= '0'; --disable display
end if;
end if;
end if;
end process control;

```

Inicialização de variáveis para o controlo dos processos, assim como o processo de reinicialização, que ocorre quando o sinal de *reset* é asserido (valor lógico zero, neste caso). Processos que controlam os contadores de linha e de coluna assim como os sinais de sincronismo *horizontal sync*, *vertical sync* e *display enable*.

```

--PatternGenerator (8 color bars or position least significant bits)
pattern : process(clk)
    variable position_vector : std_logic_vector(RWIDTH + GWIDTH + BWIDTH - 1
downto 0);
begin
    if (rising_edge(clk)) then
        case PTRN is
            when 0 =>
                --White
                if (hcount >= ((0 * (WIDTH/8)) + HFRONTPORCH + HBACKPORCH - 1) and
hcount < ((1 * (WIDTH/8)) + HFRONTPORCH + HBACKPORCH - 1)) then
                    r <= (others => '1');
                    g <= (others => '1');
                    b <= (others => '1');
                --Yellow
                elsif (hcount >= ((1 * (WIDTH/8)) + HFRONTPORCH + HBACKPORCH - 1)
and hcount < ((2 * (WIDTH /8)) + HFRONTPORCH + HBACKPORCH - 1)) then
                    r <= (others => '1');
                    g <= (others => '1');
                    b <= (others => '0');
                --Cyan
                elsif (hcount >= ((2 * (WIDTH/8)) + HFRONTPORCH + HBACKPORCH - 1)
and hcount < ((3 * (WIDTH/8)) + HFRONTPORCH + HBACKPORCH - 1)) then
                    r <= (others => '0');
                    g <= (others => '1');
                    b <= (others => '1');
                --Green

```



```

        elsif (hcount >= ((3 * (WIDTH/8)) + HFRONTPORCH + HBACKPORCH - 1)
and hcount < ((4 * (WIDTH/8)) + HFRONTPORCH + HBACKPORCH - 1))      then
            r <= (others => '0');
            g <= (others => '1');
            b <= (others => '0');
            --Magenta
        elsif (hcount >= ((4 * (WIDTH/8)) + HFRONTPORCH + HBACKPORCH - 1)
and hcount < ((5 * (WIDTH/8)) + HFRONTPORCH + HBACKPORCH - 1))      then
            r <= (others => '1');
            g <= (others => '0');
            b <= (others => '1');
            --Red
        elsif (hcount >= ((5 * (WIDTH/8)) + HFRONTPORCH + HBACKPORCH - 1)
and hcount < ((6 * (WIDTH/8)) + HFRONTPORCH + HBACKPORCH - 1))      then
            r <= (others => '1');
            g <= (others => '0');
            b <= (others => '0');
            --Blue
        elsif (hcount >= ((6 * (WIDTH/8)) + HFRONTPORCH + HBACKPORCH - 1)
and hcount < ((7 * (WIDTH/8)) + HFRONTPORCH + HBACKPORCH - 1))      then
            r <= (others => '0');
            g <= (others => '0');
            b <= (others => '1');
            --Black to White Gradient
        else
            r <= std_logic_vector(to_unsigned((((vcount - VFRONTPORCH -
VBACKPORCH) * (2 ** RWIDTH - 1))/(HEIGHT - 1)), RWIDTH));
            g <= std_logic_vector(to_unsigned((((vcount - VFRONTPORCH -
VBACKPORCH) * (2 ** GWIDTH - 1))/(HEIGHT - 1)), GWIDTH));
            b <= std_logic_vector(to_unsigned((((vcount - VFRONTPORCH -
VBACKPORCH) * (2 ** BWIDTH - 1))/(HEIGHT - 1)), BWIDTH));
        end if;

```

Parte do processo que cria o padrão de barras de cor (que é um padrão bastante usual neste tipo de testes) em que a última barra era, numa versão inicial, toda preta (Figura 16<sup>4</sup>) e foi alterada, na versão actual, para um gradiente vertical de preto para branco (Figura 17), já que a primeira configuração não permitia ver com toda a clareza se o padrão estava de facto bem ajustado aos ecrãs em que testámos.

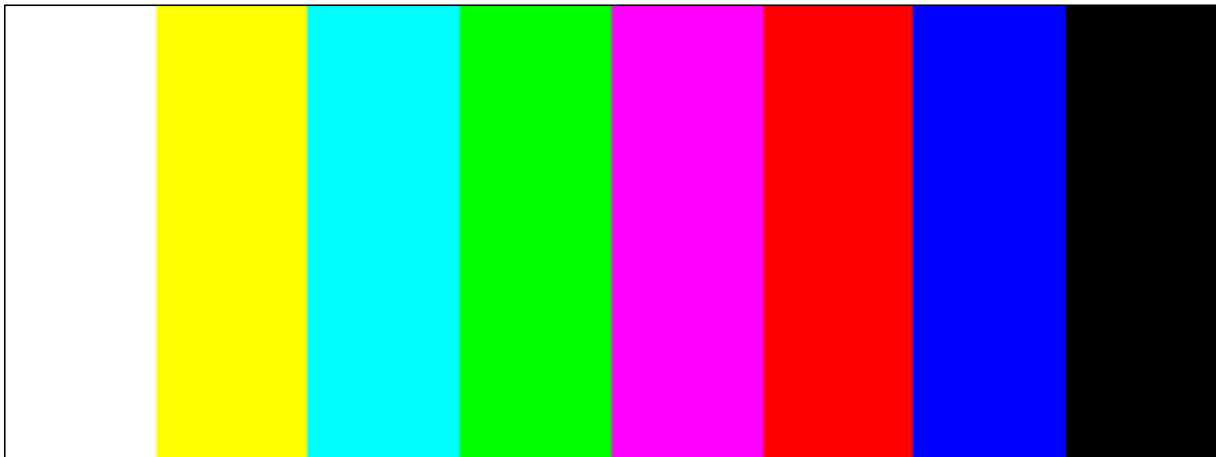


Figura 16 - Representação dos padrões de barras de cor (versão original) (formato 8:3)

---

<sup>4</sup> O contorno preto não faz parte do padrão e foi colocado para distinguir a barra branca do fundo da página

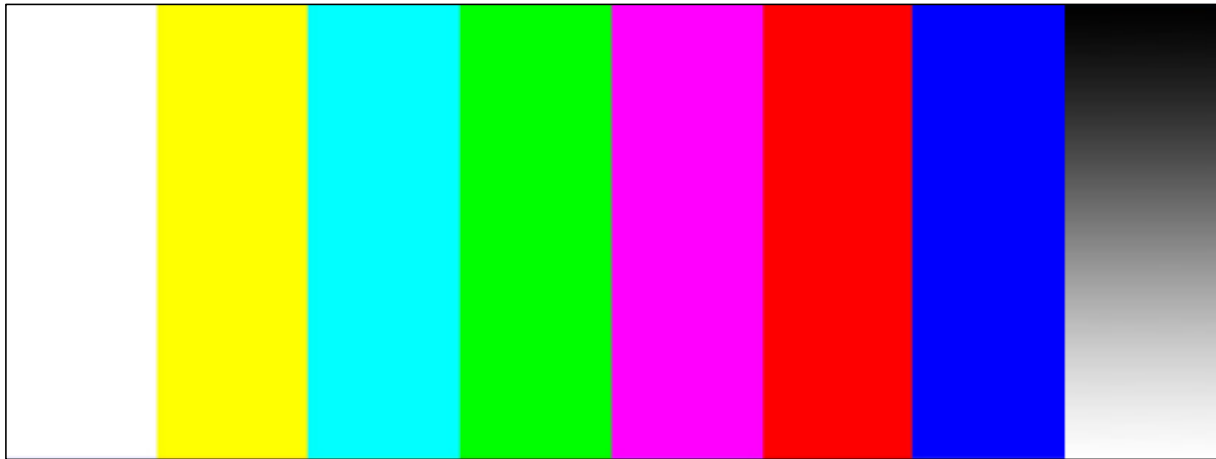


Figura 17 - Representação dos padrões de barras de cor (versão final) (formato 8:3)

```

when others =>
  -- r & g & b = position_vector(15 downto 0)
  position_vector := std_logic_vector(to_unsigned(position, RWIDTH +
  GWIDTH + BWIDTH));
  r <= position_vector(RWIDTH + GWIDTH + BWIDTH - 1 downto GWIDTH +
  BWIDTH);
  g <= position_vector(GWIDTH + BWIDTH - 1 downto BWIDTH);
  b <= position_vector(BWIDTH - 1 downto 0);
end case;
end if;
end process pattern;

end Behavioral;

```

Este bloco de código implementa o segundo padrão que o gerador é capaz de gerar. Este padrão faz corresponder a cor de cada *pixel* com os *bits* menos significativos da posição na tela. Dito de uma forma mais detalhada: a posição é contada avançando ao longo das colunas, linha a linha, da esquerda para a direita, de cima para baixo. A cor do *pixel* será função da posição, função essa em que a todos os *bits* disponíveis para o campo da cor são atribuídos os *bits* menos significativos da posição. Isto pode ser expresso da seguinte maneira:

$$\text{cor} = \text{concatenação}(\text{R}, \text{G}, \text{B}) \quad (2)$$

$$n = n^{\circ} \text{ de } \textit{bits} \text{ disponíveis}(\text{cor}) \quad (3)$$

$$\text{cor}(\text{posição}) = \text{LSB}(\text{posição}, n) \quad (4)$$

A expressão (2) traduz o que se pretende dar a entender por *bits* disponíveis para a cor, considerando-se a concatenação dos *bits* que temos para representar a cor vermelha, seguidos dos *bits* para o verde e por último para o azul. Já (4) apresenta o que foi dito acima já que a função “LSB” (de *Least Significant Bits*) representa, na prática, a recolha dos “n” *bits* menos significativos da posição, em que “n” é o número de *bits* que temos disponíveis para a representação da cor (3).

Imaginemos que temos apenas três *bits*, num hipotético “RGB111”:

$$\begin{aligned}
 &\text{posição} = 1; \\
 &\text{binário}(\text{posição}) = "1"; \\
 &\text{cor}("1") = \text{LSB}("1", 3) = "001" \Leftrightarrow \quad (5) \\
 &\Leftrightarrow \begin{cases} R = "0" \\ G = "0" \\ B = "1" \end{cases}
 \end{aligned}$$

$$\begin{aligned}
 &\text{posição} = 2; \\
 &\text{binário}(\text{posição}) = "10"; \\
 &\text{cor}("10") = \text{LSB}("10", 3) = "010" \Leftrightarrow \quad (6) \\
 &\Leftrightarrow \begin{cases} R = "0" \\ G = "1" \\ B = "0" \end{cases}
 \end{aligned}$$

$$\begin{aligned}
 &\text{posição} = 8; \\
 &\text{binário}(\text{posição}) = {}_{\text{MSB}}"1000"{}_{\text{LSB}}; \\
 &\text{cor}("1000") = \text{LSB}("1000", 3) = "000" \quad (7)
 \end{aligned}$$

Em (5) temos dois casos “normais” em que o número de *bits* da posição não excede os *bits* disponíveis para a cor e assim podemos associar directamente à cor, o valor da posição. No caso apresentado em (7), para representar a posição são necessários mais *bits* dos que os que temos disponíveis para a cor e assim foi necessário suprimir os *bits* mais significativos da posição, sobrando os menos significativos. Dá-se um *overflow*, i.e. o valor transpõe os limites de representação fazendo com que se reinicie. Do ponto de vista visual é um pouco difícil interpretar este padrão, mas olhando também para estas expressões podemos afirmar que à medida que a posição vai aumentando as cores ficando mais vivas reiniciando-se de forma periódica (quando se dá o *overflow* da componente de cor respectiva). Estes períodos são, no entanto, diferentes para as 3 cores: a cor azul varia mais rapidamente, enquanto a cor vermelha é a mais lenta neste processo (Olhando para o exemplo do hipotético espaço de cores RGB111, a cor azul varia a cada avanço da posição, o verde a cada dois avanços e o vermelho a cada quatro).

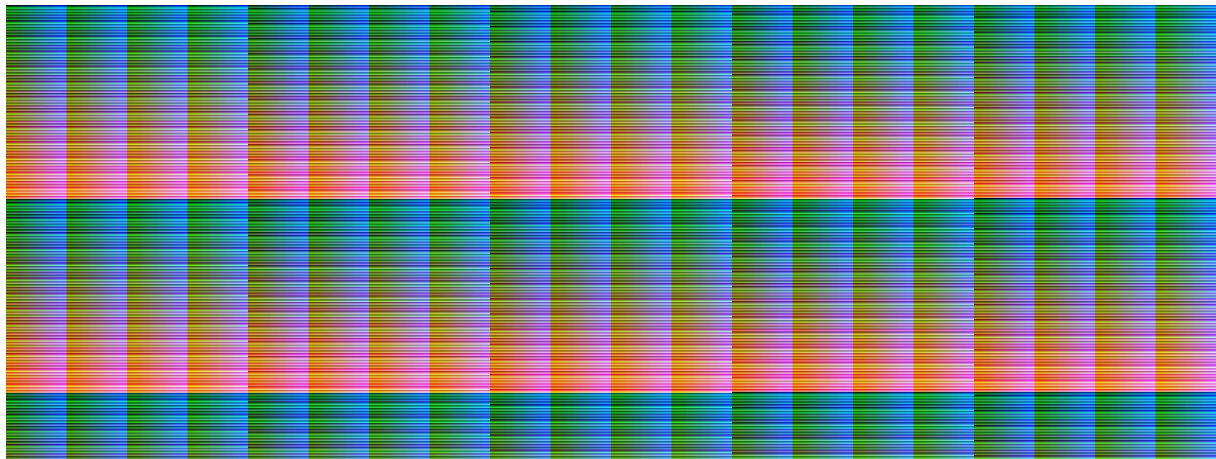


Figura 18 - Representação do segundo padrão  
(18 bits menos significativos da posição (RGB666)) (formato 8:3)

Para criar os IP *cores* para módulos concebidos de raiz bastou criar um projecto com os ficheiros de código que compunham o módulo no Vivado Design Suite. Grande parte das tarefas é executada de forma automática. A imagem seguinte (Figura 19) pretende ilustrar o aspecto da aplicação quando se encontra neste modo que permite criar os nossos próprios cores. Feito isto, e depois de incluir a biblioteca que contem estes módulos, é então possível integrar os nossos próprios IP *cores* no “IP Integrator” juntamente com os restantes, desenvolvidos pela Xilinx e por terceiros.

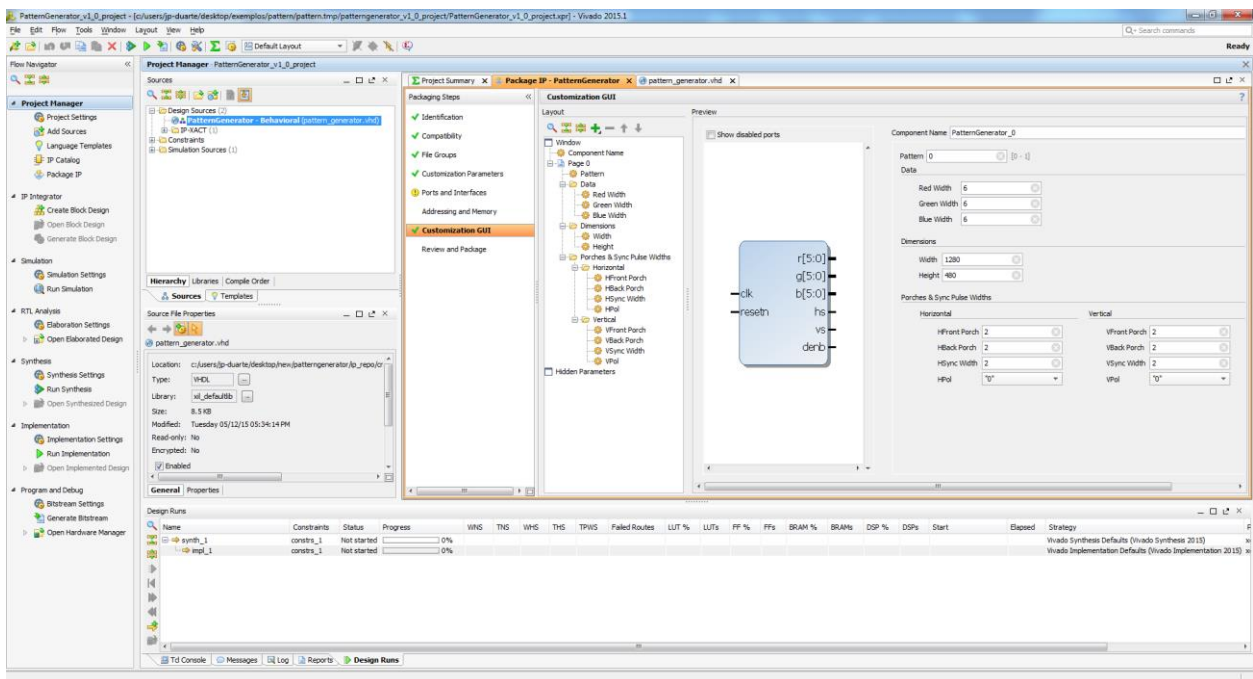


Figura 19 - Aspecto da aplicação aquando a criação de IP *cores*

## 7.3 Testes ao *datapath*

Na Vivado Design Suite, para além de ser possível criar projectos baseados no código desenvolvido, podemos também visualizar simulações do comportamento dos módulos, de forma individual, ou de partes do *datapath*. Para tal, é possível recorrer a vários métodos:

- Criação de *test benches*,
- Integração dos módulos de *debug* na plataforma de *hardware*,
- Demarcação dos sinais a analisar com a opção “*Mark Debug*”.

### 7.3.1 *Test benches*

As *test benches* virtuais traduzem-se, na prática, num ficheiro (VHDL neste caso) onde os módulos são instanciados, tratando da ligação (virtual) dos mesmos e onde se declara também o comportamento das entradas ao longo do tempo. Trata-se de um processo muito útil que permite avaliar o código desenvolvido, possibilitando a detecção de comportamentos anómalos, antes e depois dos testes “reais” com os módulos a serem executados na FPGA.

Segue-se o código desenvolvido para uma *test bench* do módulo gerador de padrões e o diagrama temporal que resulta desta simulação:

```
-----  
-- Company: iTGrow  
-- Engineer: João Pedro Aleixo Duarte  
--  
-- Create Date: 12/17/2014 06:59:10 PM  
-- Design Name:  
-- Module Name: pattern_generator_tb - Behavioral  
-- Project Name:  
-- Target Devices:  
-- Tool Versions:  
-- Description:  
--  
-- Dependencies:  
--  
-- Revision:  
-- Revision 0.01 - file Created  
-- Additional Comments:  
--  
-----  
  
library IEEE;  
use IEEE.STD_LOGIC_1164.all;  
use IEEE.NUMERIC_STD.all;  
  
library UNISIM;  
use UNISIM.VComponents.all;  
  
entity pattern_generator_tb is  
end pattern_generator_tb;
```

```
architecture Behavioral of pattern_generator_tb is
```

```
component PatternGenerator
  generic (
    WIDTH : natural := 1280; --horizontal display width in pixels
    HFRONTPORCH : natural := 2; --horizontal front porch width in pixels
    HSYNCWIDTH : natural := 2; --horizontal sync pulse width in pixels
    HBACKPORCH : natural := 2; --horizontal back porch width in pixels
    HPOL : std_logic := '0';
    --horizontal sync pulse polarity (1 = positive, 0 = negative)

    HEIGHT : natural := 480; --vertical display width in rows
    VFRONTPORCH : natural := 2; --vertical front porch width in rows
    VSYNCWIDTH : natural := 2; --vertical sync pulse width in rows
    VBACKPORCH : natural := 2; --vertical back porch width in rows
    VPOL : std_logic := '0';
    --vertical sync pulse polarity (1 = positive, 0 = negative)

    PATTRN : natural := 0;

    RWIDTH : natural := 6;
    GWIDTH : natural := 6;
    BWIDTH : natural := 6
  );
  port (
    clk : in std_logic;
    resetn : in std_logic; -- Active low
    r : out std_logic_vector(RWIDTH - 1 downto 0);
    g : out std_logic_vector(GWIDTH - 1 downto 0);
    b : out std_logic_vector(BWIDTH - 1 downto 0);
    hs : out std_logic;
    vs : out std_logic;
    denb : out std_logic
  );
end component;
```

O código começa novamente com alguns comentários e com a declaração do componente “PatternGenerator” com todos os seus parâmetros configuráveis (largura e altura, etc.) assim como as declarações das entradas (clk, resetn) e saídas (r, g, b, hs, vs, denb) do bloco.

```
--Inputs
signal clk : std_logic := '0';
signal resetn : std_logic := '0'; -- Active low

--Outputs
signal r : std_logic_vector(5 downto 0);
signal g : std_logic_vector(5 downto 0);
signal b : std_logic_vector(5 downto 0);
signal hs : std_logic;
signal vs : std_logic;
signal denb : std_logic;

constant clk_period : time := 10 ns;
```

Declaram-se, como sinais, as entradas de forma que seja possível interagir com estas e as saídas para que estas apareçam no diagrama temporal da simulação.

```
begin
  --Instantiate the Unit Under Test (UUT)
  uut : PatternGenerator
```

```

generic map(
  WIDTH => 8,
  HEIGHT => 8,
  PTRN => 0
)
port map(
  clk => clk,
  resetn => resetn,
  r => r,
  g => g,
  b => b,
  hs => hs,
  vs => vs,
  denb => denb
);

```

Instancia-se o módulo com a configuração desejada<sup>5</sup> e faz-se a ligação entre os sinais declarados e as entradas e saídas.

```

--Clock process definition
clk_process : process
begin
  clk <= '0';
  wait for clk_period/2;
  clk <= '1';
  wait for clk_period/2;
end process;

--Stimulus process
stim_process : process
begin
  wait for 20 ns;
  resetn <= '1';
end process;

end Behavioral;

```

Criam-se os processos para estimular as entradas de acordo com o intuito do teste.

Foram desenvolvidos *test benches* para outros módulos, sempre que foi necessário, em que o processo é em tudo semelhante ao que foi aqui apresentado.

O primeiro diagrama corresponde ao período de um *frame* completo (entre pulsos de sincronismo vertical) (Figura 20) e o segundo ao período de uma linha da imagem (entre pulsos de sincronismo vertical) onde podemos inclusivamente verificar a cor de cada uma das colunas do padrão gerado (padrão de barras de cor) olhando para os valores dos “vectores” “r”, “g” e “b” (Figura 21). Permite ainda verificar se as larguras dos pulsos de sincronismo se encontram de acordo com o que era pretendido.

---

<sup>5</sup> As dimensões da imagem a gerar pelo gerador de padrões foram reduzidas para oito linhas e colunas (8 x 8) para que fosse possível observar o comportamento do módulo sem que a simulação se estendesse por muito tempo (virtual).

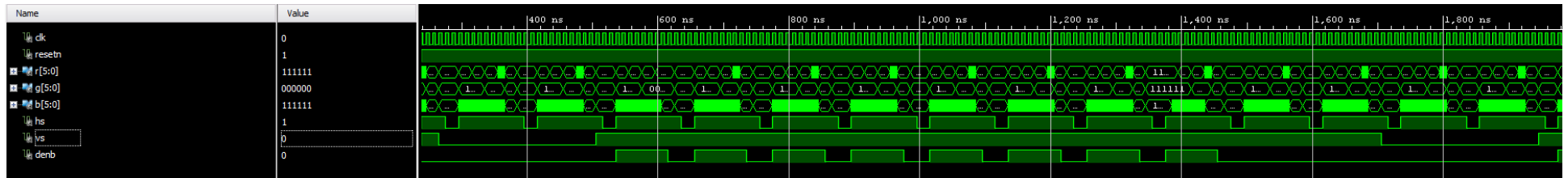


Figura 20 - Diagrama temporal dos sinais do módulo “gerador de padrões” (1 frame, 8x8)

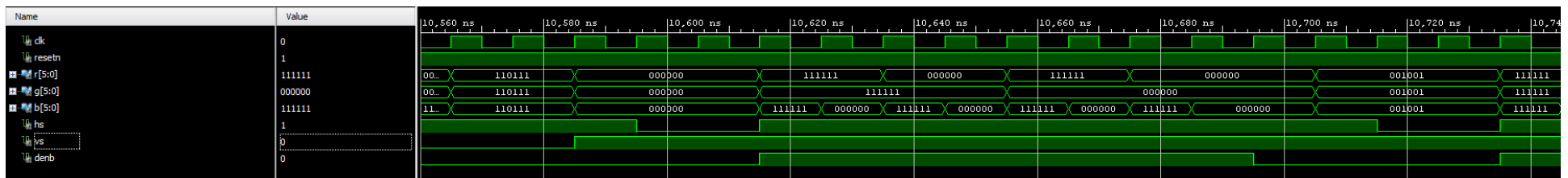


Figura 21 - Diagrama temporal dos sinais do módulo “gerador de padrões”  
(1 linha, 8 colunas)



### 7.3.2 Integração dos módulos de debug na plataforma de hardware

Dentro da biblioteca de IP *cores* da Xilinx existem alguns módulos que se destinam a tarefas de simulação/depuração de erros. Destacam-se dois *cores*: o ILA (*Integrated Logic Analyzer*) e o VIO (*Virtual Input/Output*). O ILA, quando inserido no diagrama de blocos, permite ao utilizador analisar, de forma virtual, os sinais lógicos que pretende. Os sinais surgem, em termos visuais, de forma semelhante às imagens já apresentadas (Figura 20 e 21). O VIO permite criar, como o nome indica, entradas e saídas virtuais permitindo assim analisar o comportamento de outros módulos sob teste. Neste caso específico, o mesmo efeito pode ser alcançado recorrendo às entradas e saídas físicas da placa ZedBoard.

### 7.3.3 Mark Debug

No IP *Integrator* é possível marcar alguns sinais que se pretenda analisar com a opção “Mark Debug”. O sinal fica então destacado (Figura 22), no diagrama, e o bloco ILA é integrado de forma implícita de forma a monitorizar este sinal.

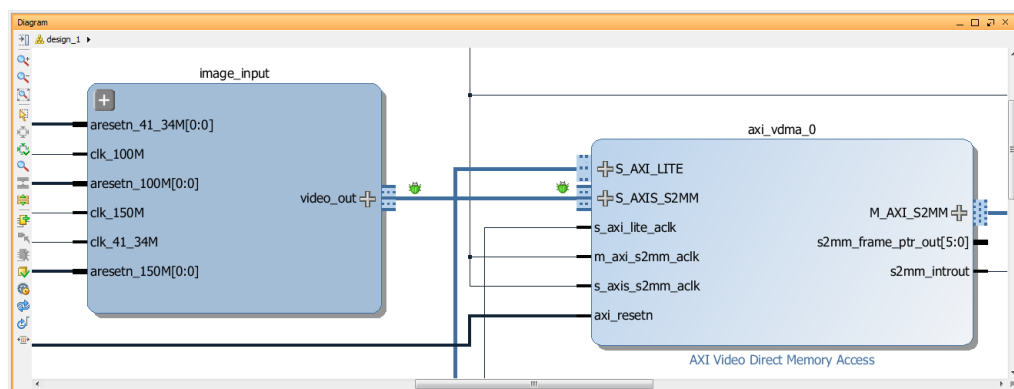


Figura 22 - Linha assinalada com a opção “Mark Debug”

Todos estes procedimentos se provaram úteis na avaliação do código ou da plataforma de *hardware* desenvolvidos, possibilitando a detecção e correção de comportamentos anómalos, antes e depois dos testes “reais” executados na FPGA.

### 7.3.4 Testes ao datapath completo

De forma a testar o *datapath* completo foi criada uma nova versão do mesmo, com base na que foi apresentada anteriormente, ao qual foram acrescentados alguns módulos com vista à utilização da saída HDMI da ZedBoard, para que as imagens capturadas fossem apresentadas num monitor externo confirmando assim a captura correcta das mesmas.

Além da inclusão da saída HDMI, decidiu-se entregar ao processador ARM algumas tarefas computacionais, além da configuração inicial dos módulos do sistema. Estas tarefas pretendiam testar o comportamento e performance do sistema nestas circunstâncias em que o processador ARM também tem uma utilização contínua. Neste teste, o módulo DMA assegura a escrita dos dados numa primeira região de memória. Por sua vez, os dados são lidos pelo processador ARM e escritos numa segunda região donde são lidos por um canal DMA de leitura e enviados para a saída HDMI. Toda esta segunda parte do processo é, como facilmente se compreende, pouco eficiente se comparado com processos exclusivamente de DMA. Contudo, como referido, quis-se neste caso implementar uma situação em que o processador ARM estivesse sob mais esforço computacional.

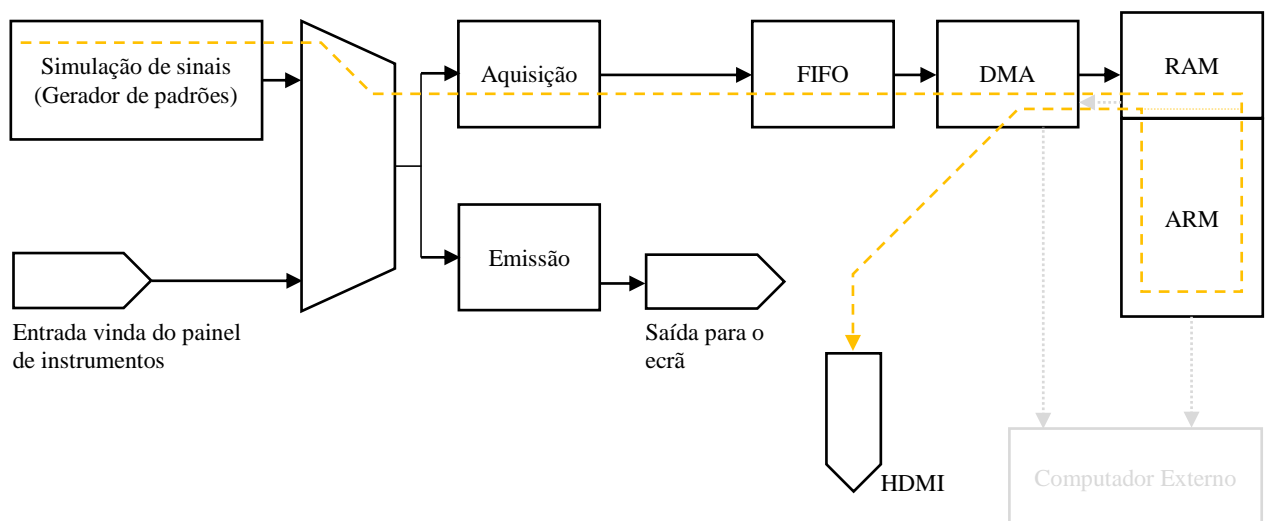


Figura 23 - Diagrama de blocos esquemático do *datapath* (com saída HDMI)

Para confirmar a correcta aquisição dos dados foram também utilizados “monitores de memória” (Figura 24) que podem ser adicionados quando a aplicação é executada em modo de depuração de erros (*Debug*). Estes “monitores” permitem visualizar os dados escritos na memória RAM, num determinado momento da execução do programa<sup>6</sup>.

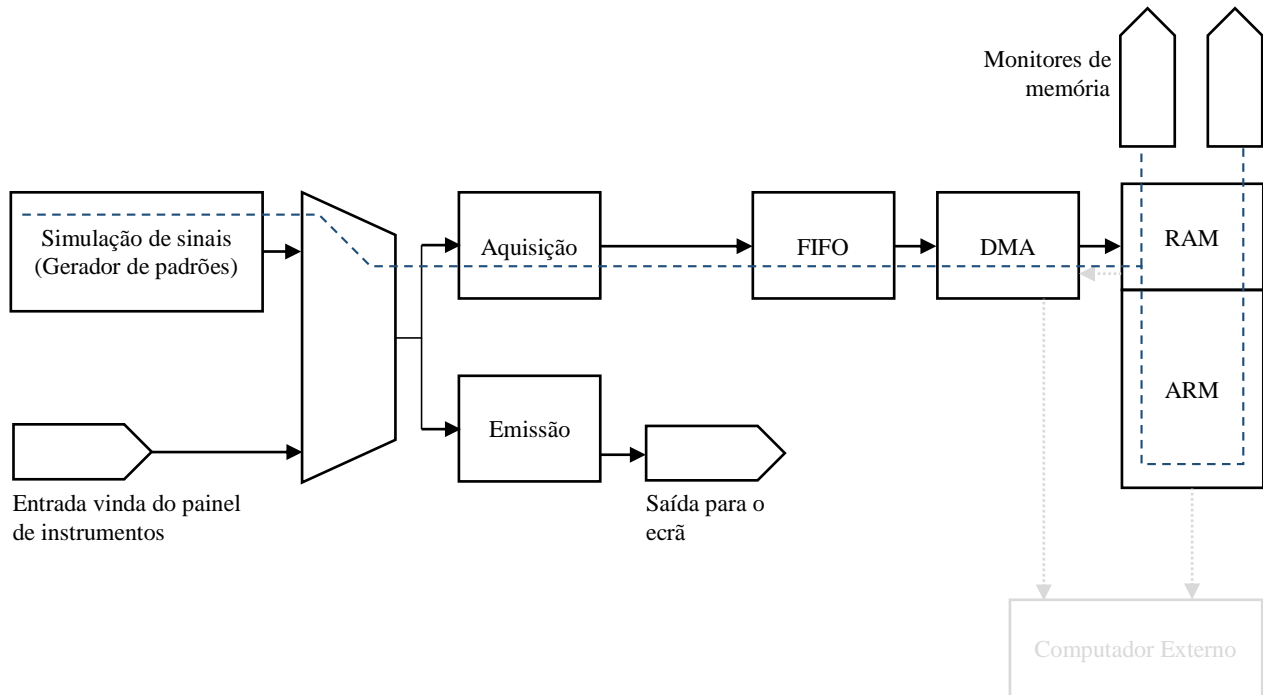


Figura 24 - Diagrama de blocos esquemático do *datapath* (monitores de memória)

Segue-se um excerto do código da aplicação desenvolvida (Xilinx SDK) para este teste, com algumas notas adicionais (a aplicação pode ser consultada na íntegra no A - 6):

```

...

/*****
 *
 * Main function
 *
 *****/

int main(void)
{
    xil_printf("\n\r");
    xil_printf("-----\n\r");
    xil_printf("--      ZedBoard HDMI Display Controller      --\n\r");
    xil_printf("-----\n\r");
    xil_printf("\n\r");

    int Status;

    // Configure VDMA engine
    Status = Vdma_Initialize(&AxiVdma);
}

```

<sup>6</sup> Só é possível visualizar os dados em memória quando execução do programa se encontra pausada (*breakpoint*, interrupção do utilizador, execução “passo a passo”, etc.)

```

if (Status != XST_SUCCESS)
{
    xil_printf( "ERROR : Failed to initialize VTC controller\n\r" );
    return XST_FAILURE;
}

```

Impressões em linha de comandos e configuração do módulo de DMA de vídeo. Todos os parâmetros estão definidos anteriormente em macros e em variáveis globais. Estão definidas regiões de memória em que os dados se irão situar. Algumas características dessas regiões:

Região “Aquisição → DMA → RAM”

- 1280 colunas,
- 480 linhas,

Região “RAM → DMA → HDMI”

- 1920 colunas,
- 1080 linhas,

*Words* de 32 bits (4 bytes) e *frame buffer* de 4 *frames*.

A região para a saída HDMI apresenta maiores dimensões para que a resolução de saída seja “1080p”. Já a resolução de “entrada” corresponde à resolução das imagens vindas do painel de instrumentos.

```

// Configure VTC on output data path
xil_printf( "Video Timing Controller (generator) Initialization ...\n\r" );
Vtc_Initialize(&Vtc);

```

Chamada da função que configura o módulo VTC (*Video Timing Controller*) para que os sinais de sincronismo sejam gerados e adequados à saída HDMI (em 1080p).

```

// Configure ADV7511 via IIC
xil_printf( "HDMI IIC Initialization ...\n\r" );
Status = Iic_Initialize(&Iic);
if (Status != XST_SUCCESS)
{
    xil_printf( "ERROR : Failed to initialize IIC driver\n\r" );
    return XST_FAILURE;
}

```

Configuração do controlador HDMI (ADV7511) via I<sup>2</sup>C. A lista de mensagens/configurações encontra-se definida antes da função “*main*”. Sem esta configuração, as imagens não seriam apresentadas de forma correcta na saída HDMI.

```

int a, f, c, r;
a = 1;
u32 data, address_read, address_write, address_write_sof; // sof = start of
frame

data = 0;
while(a)
{
    // Wait for DMA to synchronize.
    Xil_DCacheFlush();
    for(f = 0; f < NUMBER_OF_FRAMES; f++)
    {
        address_read = WR_BASE_ADDR + (WR_FRAME_HORIZONTAL_SIZE *
WR_FRAME_VERTICAL_SIZE * f);
        address_write_sof = RD_BASE_ADDR + (RD_FRAME_HORIZONTAL_SIZE *
RD_FRAME_VERTICAL_SIZE * f) + FRAME_START_OFFSET;
        for(r = 0; r < WR_FRAME_VERTICAL_SIZE; r++)
        {
            address_write = address_write_sof + (RD_FRAME_HORIZONTAL_SIZE * r);
            for(c = 0; c < (WR_FRAME_HORIZONTAL_SIZE/4); c++)
            {
                data = Xil_In32(address_read);
                Xil_Out32(address_write, data);
                address_read += 4;
                address_write += 4;
            }
        }
    }
}
return 0;
}
...

```

Ciclo infinito em que o processador (ARM) copia as imagens da região “Aquisição → DMA → RAM” para a região “RAM → DMA → HDMI”. O ciclo infinito permite uma aquisição contínua e ininterrupta de imagens, de acordo com os objectivos. Os endereços de leitura e escrita têm de ser actualizados de forma cuidadosa devido à discrepância entre as resoluções das imagens e, conseqüentemente, do tamanho das regiões de memória.

Na figura 25 encontra-se apresentado o diagrama de blocos (IP *Integrator*) do *datapath* para este teste. Nesta implementação destaca-se a utilização de entrada de imagens “externa” (painel de instrumentos) e a activação do canal de leitura no módulo VDMA para o envio das imagens para a saída HDMI (que implica a inclusão de módulos adicionais para a configuração do controlador via I<sup>2</sup>C e da criação dos sinais de sincronismo pelo módulo “VTC”)

A saída “O\_TR” iria permitir determinar com absoluta certeza o modo de funcionamento do ecrã do painel de instrumentos (se TTL (*single-ended*) ou RSDS (diferencial)).

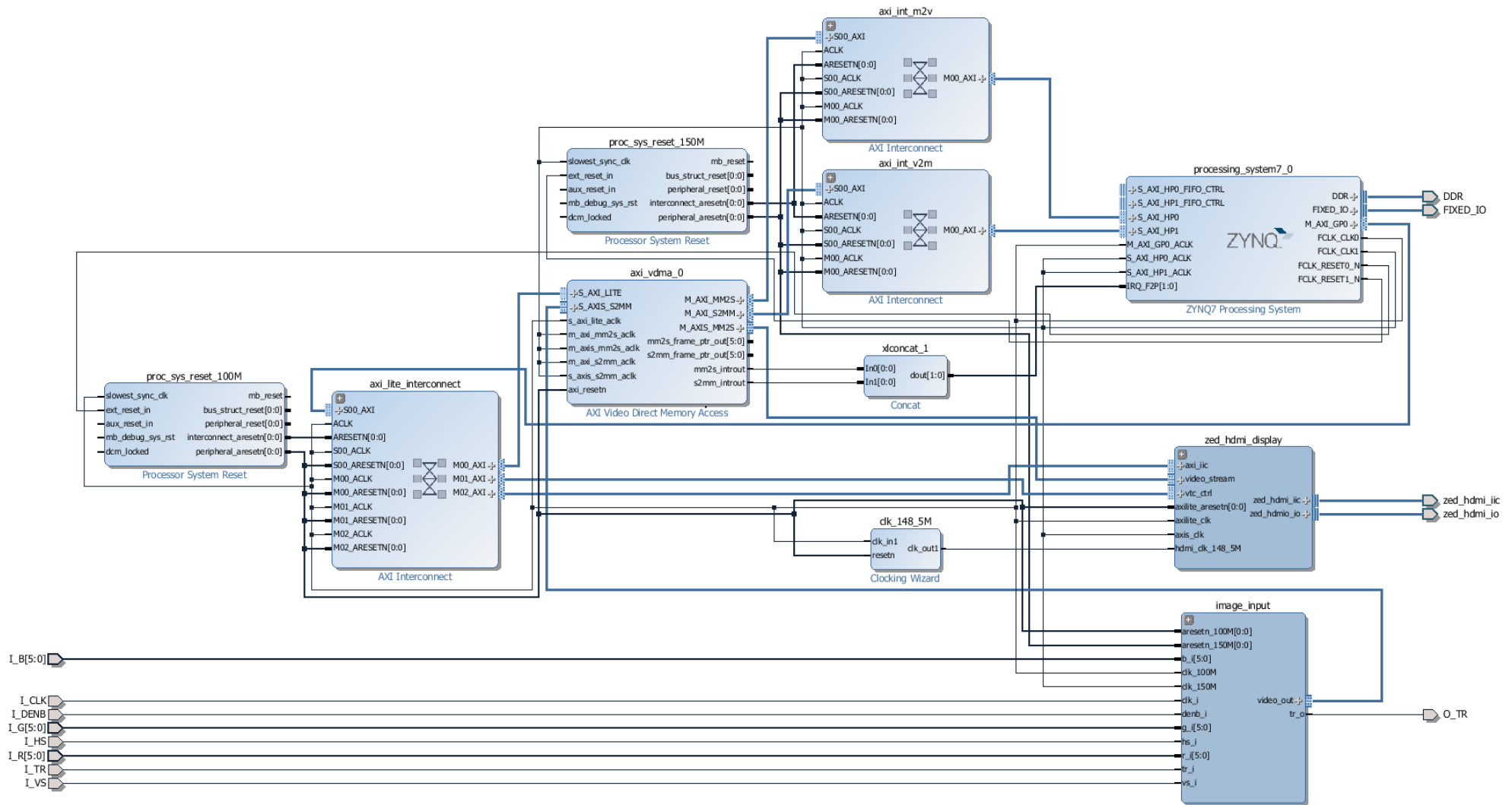


Figura 25 - Diagrama de blocos do *datapath* (versão com entrada externa e saída HDMI)







# 8 Montagem do sistema

## 8.1 Projecto da PCB de interface

O projecto da PCB foi desenvolvido usando o programa EAGLE, da CadSoft, na sua versão “Light Edition”, que pode ser obtida através da ligação <http://www.cadsoftusa.com/download-eagle/>. Esta versão tem algumas limitações mas provou-se suficiente para este projecto [44]:

- A área útil da placa está limitada a um rectângulo de 100 por 80 mm (4 x 3.2 polegadas).
- Apenas podem ser usadas duas camadas (*layers*) de sinal (camada superior e inferior)
- No editor de esquemático apenas pode ser criada uma “folha”.
- O suporte está apenas disponível via *e-mail* ou no fórum.
- O uso está limitado a aplicações sem fins lucrativos e propósitos de avaliação.

O projecto de uma PCB divide-se em duas partes: desenho do esquemático e desenho da placa. Na primeira ligamos os componentes de forma esquemática sendo possível fazer ligações através do nome das “malhas” (*net*).

O desenho da placa teve de ser alvo de uma atenção especial uma vez que era necessário criar a *footprint* (ou *package* no EAGLE) dos conectores necessários: um conector FMC - LPC macho da Molex (45970 SMT Plug Connector) e dois conectores IRISO da série 11501S de 50 pinos. Apesar de estarem disponíveis conectores idênticos aos da placa do painel de instrumentos (da série 9687S da IRISO), optou-se por utilizar conectores da série 11501S uma vez que o novo sistema de retenção automática que esta série disponibiliza permite uma maior acessibilidade, sendo mais fácil montar e desmontar as ligações.

No desenho da placa, procedeu-se ao roteamento cuidado das pistas de acordo com as ligações efectuadas no esquemático.

Durante a concepção do PCB um cuidado especial com os sinais de relógio os quais entram através dos pinos denominados com o sufixo “CC” (de *Clock Capable*) de acordo com as seguintes referências [45] (p. 2) [46] (pg. 29, 34 e 47), [47] (p. 26) e [48] (p. 15) (se o relógio for *single-ended* deverá entrar pelo pino positivo (“P”) do par “CC”). Foi também decidido que deveria ser precavida a possibilidade de os sinais serem transmitidos em pares diferenciais e assim deveriam ser ligados num par de pinos do conector FMC disponível, devendo ainda ser minimizada a diferença de comprimento entre as pistas de cada par de forma a minimizar o desfasamento (*skew*) entre os sinais do par.

Fora ainda tidas em conta as seguintes considerações:

- Quando as pistas se encontram muito próximas (quer no mesmo plano quer em planos diferentes) podem ocorrer fenómenos de indução de ruído por questões de acoplamento indutivo e capacitivo. Este fenómeno, vulgarmente conhecido por *crosstalk*, pode ser ainda mais problemático a frequências mais elevadas (ordem igual ao superior a MHz). No nosso caso a frequência máxima é de aproximadamente 41,34 MHz (contudo trata-se de um sinal digital de relógio, conseqüentemente não sinusoidal, o que se traduz, certamente, no aparecimento de harmónicos de maior frequência) que ainda não é muito gravosa. Foram no entanto tomadas precauções neste sentido, conservando uma distância mínima entre as pistas, tentando garantir o isolamento das mesmas.
- Outra possível causa de ruído são os chamados “*loops* de massa” onde surgem diversas ligações desnecessárias à mesma massa, podendo originar correntes indesejadas devido às possíveis diferenças de potencial, ainda que mínimas, que possam porventura existir entre as diferentes massas do sistema. Caso não sejam tomadas medidas que contrariem este problema, podem ser criadas situações em que o ruído gerado possa influenciar de forma significativa os sinais de pistas próximas. No nosso caso particular, tivemos esta possibilidade em consideração e a placa foi desenhada com o intuito de preservar as diferentes massas do sistema o melhor possível.

Os planos de massa da PCB estão ao mesmo potencial, assegurado pela colocação “estratégica” de vias entre as duas camadas. Existem ligação destes planos aos pontos de montagem dos conectores FFC/FPC e a um contacto no conector FMC/LPC que se encontra ligado à massa da FPGA.

As figuras 26, 27 e 28 apresentam, respectivamente, o desenho da placa, e as fotografias do produto final.

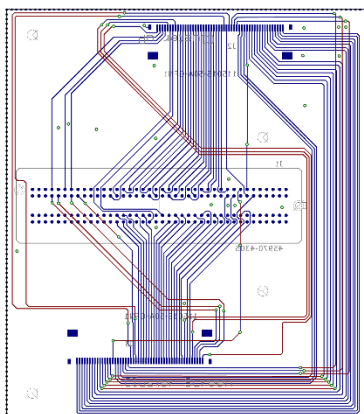


Figura 28 -  
Desenho da PCB criada  
(board)

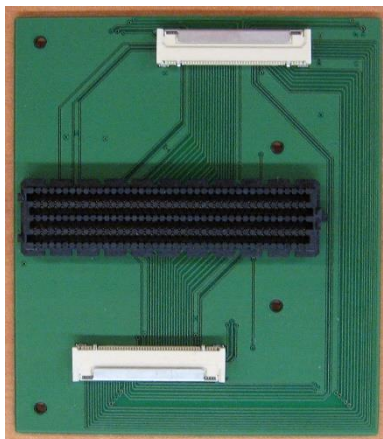


Figura 28 -  
Fotografia da PCB criada  
(face inferior)

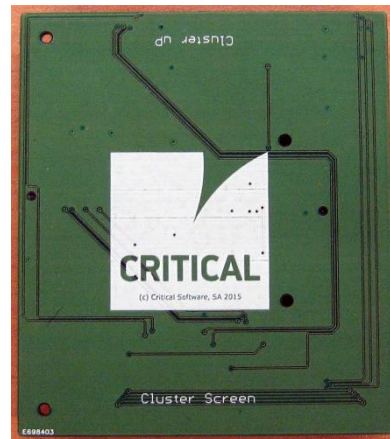


Figura 28 -  
Fotografia da PCB criada  
(face superior)

No desenho da placa (Figura 28) podemos ver a vermelho as pistas na camada superior da placa e a azul as pistas e os contactos dos conectores da camada inferior (trata-se de uma PCB de apenas 2 camadas, pelo que se encontra ainda dentro das restrições da versão do EAGLE utilizada). A verde encontra-se as vias: aquelas que não são utilizadas para fazer passar a pista da camada superior para a camada inferior servem para colocar os planos de cobre ao mesmo potencial eléctrico (*ground*). A fotografia da figura 27 foi espelhada verticalmente de forma a ser mais fácil a comparação com o desenho da placa, não correspondendo à disposição real dos elementos.

É possível verificar que as linhas de dados e sincronismo são direccionadas para o conector FMC enquanto as linhas de alimentação e massa são desviadas deste conector unindo directamente os conectores FFC. As linhas de alimentação poderiam, inclusivamente, danificar os pinos da FPGA uma vez que têm potenciais (de -12V a 8V) fora dos limites tolerados pela placa (0V a 3.3V).

Foram produzidas duas placas idênticas de forma a precaver algum erro de fabrico de alguma delas, mas só após a chegada das mesmas verificámos um erro de projecto que dificultava em muito a ligação de todos os sistemas: os conectores FFC/FPC ficaram orientados para o interior da placa quando a ideia inicial seria estarem virados para o exterior e assim ficarem mais acessíveis.

## 8.2 Ligação dos equipamentos e testes ao sistema

Mesmo com a contrariedade referida, em que os conectores FFC/FPC estão orientados para o interior da PCB, era ainda assim possível ligar todos os dispositivos com o material que tínhamos disponível: placa do painel de instrumentos, placa ZedBoard, PCB de interface, ecrã e fita FFC de 50 pinos.

Existiam algumas fitas à disposição que tinham sido enviadas pelos fornecedores dos conectores: a Molex enviou uma fita de comprimento médio, com os contactos em faces opostas (*opposite-side contacts*), sem entalhes de encaixe e com as fitas de suporte dos contactos um pouco espessas demais para os conectores escolhidos, enquanto a IRISO enviou várias fitas da Sumitomo todas elas com os contactos na mesma face (*same-side contacts*) umas maiores que a da Molex e outras bastante curtas mas que permitiam um encaixe perfeito tanto do lado da placa do painel de instrumentos (conector IRISO da série 9687S) como do lado da PCB (IRISO 11501S) devido aos entalhes na zona de retenção e fitas de suporte de espessura adequadas.

Assim, o sistema foi montado como apresentado na figura 29:

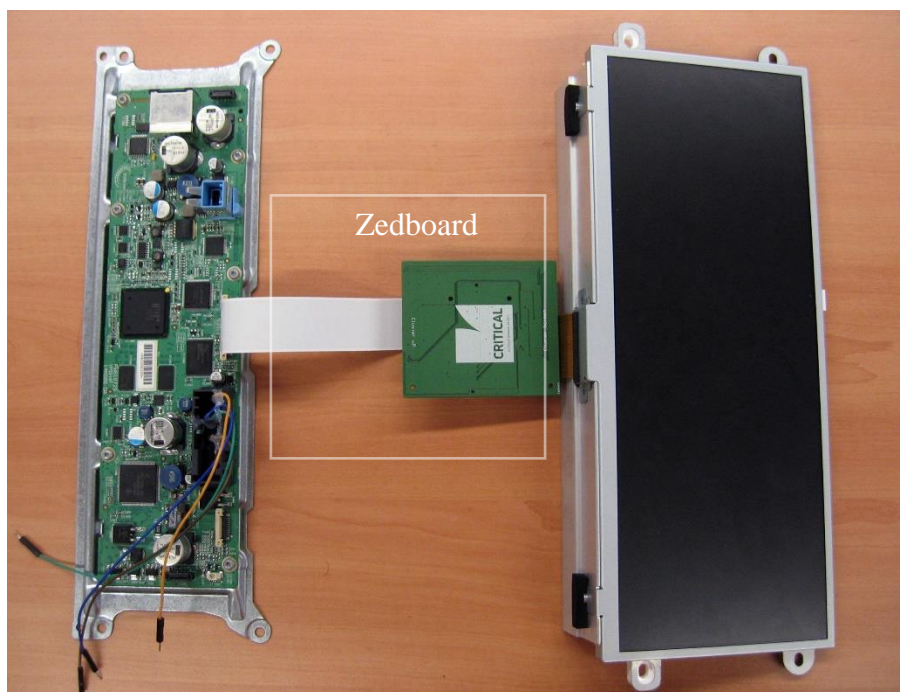


Figura 29 - Ilustração da montagem efectuada

Foi usada uma das fitas Sumitomo disponíveis (Figura 30), já que eram as que, aparentemente, tornavam a ligação mais segura, ao serem as mais compridas permitiam afastar os componentes para que não se sobrepusessem e o encaixe com os conectores era o mais estável.



Figura 30 - Fita FFC utilizada na ligação

Foi preparada uma plataforma de *hardware*, baseada na que foi apresentada anteriormente no capítulo 7 ao qual foi adicionada a retransmissão dos sinais para o ecrã do painel de instrumentos e a interface com outro ecrã ligado à ZedBoard via HDMI.

A tensão ajustável que alimenta a maior parte dos pinos da interface FMC foi configurada para 3.3V através do *jumper* de selecção de forma a suportar os valores esperados nas entradas vindas do painel de instrumentos.

Para alimentar o painel de instrumentos foi usado um transformador (saída de 12V DC) e para o ligar foi usada uma interface CAN externa ligada ao computador. (Esta interface CAN poderia vir a ser integrada na placa ZedBoard posteriormente, já que é possível criar este tipo de interface usando pinos de um dos conectores Pmod).

Com todo o sistema ligado física e electricamente, era esperada a visualização das imagens correspondentes aos avisos iniciais do *painel de instrumentos* tanto no ecrã do painel como no ecrã ligado à ZedBoard via HDMI e tal não aconteceu. Repetimos todo o processo com os mesmos resultados. Analisando melhor as ligações efectuadas verificou-se um erro: ao utilizar uma fita com os contactos na mesma face e tendo os conectores uma posição de encaixe fixa acabámos por ligar inadvertidamente os conectores do painel de instrumentos com o respectivo conector da PCB ao contrário (ao pino 1 do painel fizemos corresponder o pino 50 do conector da PCB).

De forma a evitar o erro cometido poderiam ter sido seguidas as seguintes alternativas:

- a. Utilizar uma fita FFC com contactos em faces opostas. Existia uma fita deste tipo disponível mas não providencia uma ligação muito segura devido a diferenças no formato.
- b. Redesenhar a PCB de interface prevendo esta situação, para que todos os equipamentos ficassem correctamente ligados.

O restante trabalho desenvolvido no âmbito desta dissertação foi orientado de acordo com as alternativas apresentadas.



# 9 Trabalho complementar

Neste capítulo irá ser apresentado algum do trabalho complementar desenvolvido e alguns possíveis melhoramentos a introduzir no projecto.

## 9.1 Módulos adicionais implementados

Nesta secção irão ser apresentados os módulos que foram desenvolvidos numa fase inicial, ainda antes da escolha da FPGA a adquirir, e que não vieram a integrar o *datapath* final.

Nesta fase, foi desenvolvido um módulo que guarda uma imagem que estiver a ser enviada para este bloco (por exemplo, pelo painel de instrumentos ou pelo simulador). Ignora também dados inválidos, ao esperar por *frames* completos, desprezando assim os dados que recebe inicialmente de forma a garantir que a captura não começa a meio de uma imagem que estiver, porventura, a ser enviada.

Inicialmente o módulo limitava-se a guardar a imagem internamente enquanto as versões que se seguiriam deveriam preocupar-se também com a interface com a FIFO e com a memória.

Em anexo (A - 2)) encontra-se uma das primeiras versões do código. Assim como o módulo anterior, este permite configurar uma série de parâmetros como a largura e altura da imagem, o número de *bits* de cada cor, assim como o número de ciclos que devem ser ignorados após os pulsos de sincronismo (declarados como *HFRONTPORCH* e *VFRONTPORCH* embora não seja a designação mais correcta, já que não correspondem aos intervalos homónimos do diagrama presente na *datasheet* o ecrã) e por último o número de ciclos ao fim dos quais damos por perdido o sincronismo com os sinais de entrada. Olhando para o processo responsável pelo sincronismo podemos verificar que só ao fim de dois pulsos na linha que corresponde ao sinal de sincronismo vertical é que começamos a considerar que os sinais estão “sincronizados”. Nota ainda para o cálculo dos endereços para escrita no *array* que desempenha a função de memória neste módulo: os endereços são calculados de forma sequencial, através de incrementos unitários, de forma a escrevermos de forma contínua no *array*, aplicando ainda a operação do resto da divisão inteira a estes endereços de forma a evitar o endereçamento inválido, para além dos limites.

A versão que segue pretendia integrar a FIFO e a interface com a RAM usando blocos pré-feitos da biblioteca “gh vhdl library” da comunidade OpenCores (disponíveis em [http://opencores.org/project,gh\\_vhdl\\_library](http://opencores.org/project,gh_vhdl_library), sendo necessário estar registado no *site* para fazer o *download*). O código também se encontra disponível no anexo A - 3). O desenvolvimento deste módulo acabou por ser descontinuado, e deve servir apenas como referência da primeira tentativa de combinar a aquisição de imagem pela FIFO com a memória RAM. O processo responsável por “encher” a memória (identificado com o comentário “RAM filler”) alternaria entre dois estados: um estado inactivo (*idle*) onde a FIFO vai enchendo até que a *flag* “fifo\_full” seja sinalizada que corresponde a termos metade da FIFO cheia (sinal “hfull”) e um estado (*stream*) em que enviamos um conjunto de dados (*burst*) para a memória caso esta esteja disponível (“rwstall = ‘0’”). Como referi, esta parte encontra-se apenas parcialmente implementada, mas as bases para o desenvolvimento estavam criadas, caso viessem a ser necessárias.

Depois de escolhida a FPGA e instalado o *software* de desenvolvimento surgiu a necessidade de integrar uma interface AXI (mais concretamente AXI4-Stream) na saída do módulo (Anexo A - 5)) de modo a que se tornasse compatível com módulos disponíveis na biblioteca que vem por defeito. Os resultados dos testes com este módulo foram satisfatórios mas os módulos “Video In to AXI4-Stream” e “AXI4-Stream Data FIFO” da biblioteca original garantiam mais confiança num funcionamento correcto.

## 9.2 Possíveis melhoramentos

Depois de criada e testada a plataforma de *hardware* que cumpria com os requisitos mínimos propostos que consistiam na captura de imagens, passando-as para a memória da FPGA, foram idealizadas algumas funcionalidades adicionais que poderiam também ser incluídas no projecto, que a seguir são descritas.

### 9.2.1 Envio da informação para outra máquina

A primeira ideia consistia em passar a informação que tínhamos em memória para outro computador e para tal foram estudadas duas possibilidades, recorrendo à interface USB-UART ou à Gigabit Ethernet:



### 9.2.1.1 USB-UART

O uso da interface USB-UART provou-se uma solução bastante simples para comunicar entre a placa e o computador enviando mensagens entre ambos via esta interface série usando um terminal (foi usada a aplicação PuTTY já que o terminal integrado no SDK da Xilinx apresenta alguns erros que não permitem enviar mensagens para a placa). Contudo não se verificou se esta interface conseguiria acompanhar o mesmo ritmo (663552000 *bits* úteis/s (8)) do fluxo de dados que capturámos e temos disponível para enviar.

$$\begin{aligned} \frac{\text{n}^\circ \text{ de } \textit{bits} \text{ \u00fasteis}}{\text{segundo}} &= 1280 \times 480 \left( \frac{\text{p\u00edxeis}}{\textit{frame}} \right) \times 60 \left( \frac{\textit{frame}}{\text{s}} \right) \times 18 \left( \frac{\textit{bits}}{\text{p\u00edxel}} \right) = \\ &= 663552000 \textit{ bits/s} \end{aligned} \quad (8)$$

### 9.2.1.2 Gigabit Ethernet

Foi estudada, ainda que de forma superficial, a pilha TCP (*Transmission Control Protocol*)/IP (*Internet Protocol*) que a biblioteca lwIP (*lightweight IP*) providencia [49]. Foram implementados alguns exemplos de teste desta biblioteca mas sem grande aplicabilidade para o nosso caso e o desenvolvimento acabou por n\u00e3o ser prosseguido.

Ambas as solu\u00e7\u00f5es apresentadas - o uso da USB-UART e a Gigabit Ethernet - ficaram com a sua implementa\u00e7\u00e3o pendente, uma vez que o envio das imagens para outra m\u00e1quina j\u00e1 se encontra fora do \u00e2mbito concreto deste trabalho, que se limita \u00e0 sua captura. Contudo, esta tarefa sempre foi encarada como um passo crucial a desenvolver num futuro pr\u00f3ximo, em caso de continuidade do projecto.

Embora o uso da interface Gigabit Ethernet implique uma solu\u00e7\u00e3o mais complexa em termos de desenvolvimento, esta oferece muito mais garantias de sucesso relativamente \u00e0 USB-UART, uma vez que permite um desempenho mais elevado ao ser capaz de atingir taxas de transfer\u00eancia superiores.

### 9.2.2 *Multiprocessamento assimétrico*

Foram criados projectos que permitem implementar soluções AMP (*Asymmetric MultiProcessing*). Estas soluções, por sua vez, permitem correr aplicações distintas em cada um dos *cores* do processador ARM (*dual-core*) presente nos SoC Zynq. Foram implementadas de acordo com as “Application Notes” da Xilinx XAPP1078 (*Linux+Bare-Metal*) [50] e XAPP1079 (*2xBare-Metal*) [51]. Assim passaria a ser possível correr aplicações distintas nos diferentes cores podendo um ser responsável pelo envio das imagens para outro computador e o outro por manipular ou comparar as imagens com outras de referência, por exemplo. Depois de implementados os projectos descritos nas “Application Notes” a sua integração no projecto não foi continuada.

### 9.2.3 *Interfaces Pmod*

Antes da chegada da PCB que faria a interface com o conector FMC foi estudada a possibilidade de utilizar os conectores Pmod para criar um protótipo de interface com o painel de instrumentos e com o ecrã. A interface através destes conectores seria possível para apenas uma das operações em simultâneo: (receber do painel ou enviar para o ecrã (a não ser que o envio fosse montado em paralelo com a recepção, um pouco como o que acontece na primeira hipótese apresentada na secção 6.1). Estes conectores e interfaces não são tão adequados para o projecto sendo que a única possível vantagem identificada foi a possibilidade de criar um protótipo de forma mais acelerada quando comparada com utilização da interface FMC, já que os componentes que podem ser utilizados são mais comuns (*jumper wires*, conectores de 1.27mm de espaçamento, etc.). Foram feitos apenas alguns testes às interfaces Pmod da placa (testes de entrada e de saída de sinais, sinais de relógio, etc.) já que ideia de avançar com o protótipo não se justificava, até porque as placas PCB para interface FMC já se encontravam em produção aquando a elaboração deste estudo.

#### 9.2.4 Saídas de vídeo da ZedBoard

De forma a tornar o projecto visualmente mais interessante estudou-se agora a possibilidade de integrar no projecto uma das saídas de vídeo que a placa ZedBoard disponibiliza: VGA e HDMI. Era assim necessário fazer alguns *upgrades* à plataforma de *hardware*: era necessário activar o canal de leitura da memória do módulo VDMA e encaminhar o fluxo de dados para as saídas de vídeo. As imagens capturadas passariam a ser então apresentadas tanto no ecrã do painel de instrumentos como num ecrã externo.

##### 9.2.4.1 VGA

Começou-se por testar a interface VGA. Os sinais digitais são convertidos para sinais analógicos através de DACs (*Digital-to-Analog Converters*) compostos apenas por resistências que vão diminuindo sucessivamente para metade à medida que se avança para os *bits* mais significativos fazendo com que a corrente e consequente peso do *bit* seja maior. Está limitado a 4 *bits* por cor.

Foram criados alguns testes para esta interface, alguns deles não corresponderam completamente às expectativas: por vezes a imagem aparecia muito escura ou aparentemente dessincronizada. O módulo “AXI4-Stream to Video Out” apresentava alguns problemas na interface entre o sinal AXI4-Stream e a saída tendo sido substituído pelo módulo “AxiStreamVGA” disponível na ligação <http://arbot.cz/post/2013/05/11/AXI-stream-VGA.aspx>.

##### 9.2.4.2 HDMI

Embora a interface VGA fosse de facto mais simples, a interface HDMI garantia, à partida, melhores resultados, com uma qualidade de imagem superior. O uso desta interface implica no entanto a inclusão de alguns módulos adicionais. É necessário configurar o controlador HDMI (ADV7511 da Analog Devices) via I<sup>2</sup>C (ou IIC - *Inter-Integrated Circuit*), e converter o espaço de cor de RGB para YCrCb422 (conversão e reamostragem das componentes cromáticas). Tudo isto porque, embora o controlador ADV7511 permitisse a ligação de mais linhas de dados e assim dispensar a conversão, apenas parte delas está ligada aos pinos do SoC. O SoC pode assim ser ligado a um maior número de periféricos [45]. O módulo de reamostragem (“*Chroma Resampler*”) faz parte de um conjunto (“*Video IP Cores*”) que implica uma licença especial, sendo que foi feito o *download* de uma licença de avaliação de forma gratuita. De acordo com os testes realizados tudo se encontrava a funcionar correctamente e a interface HDMI foi assim integrada no projecto principal. (Figura 31)

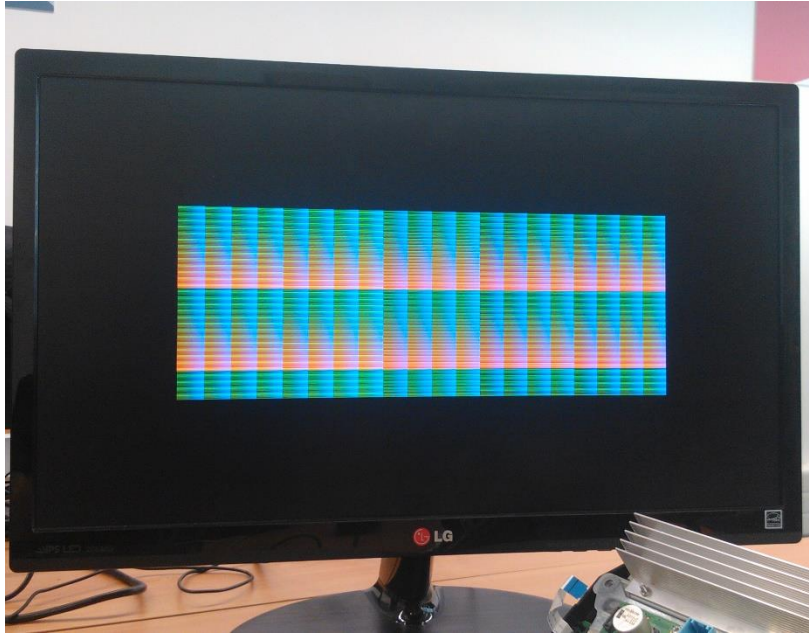


Figura 31 - Fotografia do ecrã ligado à saída de vídeo externa (HDMI)

#### 9.2.5 *Projecto de PCBs de interface alternativas*

Como foi dito no final da secção 8.2 o projecto de uma PCB alternativa poderia ser encarado como uma forma de corrigir o erro de ligações que foi verificado. A outra opção passava por adquirir uma fita FFC semelhante à que foi utilizada (Figura 30) com a *nuance* de ter os contactos em faces opostas.

De forma complementar projectaram-se duas PCBs de interface em alternativa à que foi apresentada no capítulo 7. Os componentes seriam os mesmos: um conector FMC - LPC macho e dois conectores FFC/FPC IRISO 11501S de 50 pinos por placa. Em ambos os projectos foi corrigida a posição e/ou orientação dos conectores tornando a placa mais compacta, com os conectores mais acessíveis e mais bem posicionados, relativamente à ZedBoard. Num dos projectos (Figura 33) foram invertidas as ligações num dos conectores FFC (o pino 1 foi ligado ao 50, o 2 ao 49, 3-48, etc.) e assim seria possível usar as fitas que já tínhamos disponíveis para a ligação. No outro (Figura 32), ainda seria necessária a utilização de uma fita com os contactos em lados opostos. Os desenhos esquemáticos de ambos os circuitos podem ser encontrados em anexo (Anexo C).

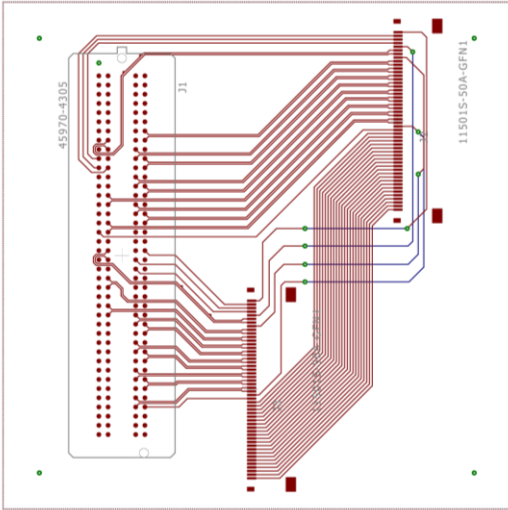


Figura 32 - Desenho da nova PCB (A1)

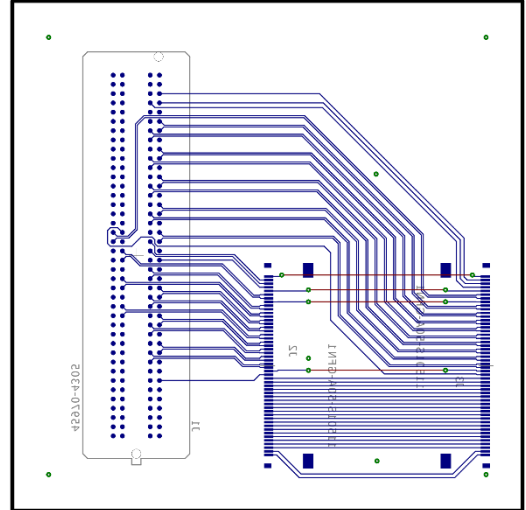


Figura 33 - Desenho da nova PCB (A2)



# 10 Conclusões

O trabalho foi desenvolvido no âmbito de um estágio curricular proporcionado pela iTGROW nas instalações da CRITICAL Software em Taveiro.

O objectivo do trabalho passava por capturar as imagens geradas por um painel de instrumentos digital, usado em automóveis. Para proceder à captura deveria ser utilizada, obrigatoriamente, uma FPGA, já que esse era um dos requisitos/objectivos do projecto.

Depois de estudada a interface da placa do painel de instrumentos e o respectivo ecrã, procedeu-se ao projecto da arquitectura do sistema que iria ser implementado, quer do ponto de vista da ligação de todos os elementos, quer da idealização do caminho de dados que se queria implementar.

Foi desenvolvida a plataforma de *hardware* (assim como o *software* que, essencialmente, configura alguns dos IP *cores* implementados na lógica programável) que permite a captura de imagens. Foram executados alguns testes, em ambiente de simulação, que apresentaram os resultados pretendidos.

As vantagens deste sistema passam pela redução de custos, tanto em termos directamente monetários, como os custos com equipamento, assim como em termos de tempo de desenvolvimento, uma vez que se trata de uma solução flexível, que permite ir sendo adaptada. A plataforma criada pode ainda ser utilizada noutras FPGAs. A desvantagem pode passar por um decréscimo mínimo na performance quando comparada com soluções específicas para o problema, que conseguem atingir frequências de processamento superiores. Os objectivos do projecto não são, no entanto, comprometidos já que as capacidades da FPGA utilizada provaram-se suficientes.

Uma avaria surgida durante os primeiros testes reais, devida a um erro nas ligações dos equipamentos, inviabilizou a continuação dos ensaios previstos. A falta de resultados em situações reais é, de certa forma, compensada pelos bons resultados em ambientes de simulação.

Foram ainda identificadas possibilidades de melhoramentos a incorporar no projecto que podem ser implementadas no futuro.

Como próximos passos, pretende-se confirmar que a imagem que está a ser apresentada num determinado momento corresponde aos comandos enviados para o painel por outros equipamentos do veículo (através de mensagens CAN). Tal confirmação poderá passar por comparar as imagens capturadas em tempo real com imagens de referência, previamente adquiridas. Esta comparação poderá ser feita ou na própria FPGA ou noutra computador (sendo necessário implementar a transmissão das imagens, capturadas pela FPGA, para esta nova máquina).







# Referências

- [1] ISO, “ISO 26262 - Road vehicles - Functional safety - Part 10: Guideline on ISO 26262,” 2012. [Online]. Available: <https://www.iso.org/obp/ui/#iso:std:iso:26262:-10:ed-1:v1:en>. [Acedido em Setembro 2015].
- [2] M. Geier, “Development of a Hardware Framegrabber for Network (GigE Vision) Cameras and Evaluation of fast Image Processing Algorithms for Visual Servoing Applications,” [Online]. Available: <https://www.rcs.ei.tum.de/en/courses/theses/fpga-based-image-acquisition-and-processing/>. [Acedido em Setembro 2015].
- [3] S. McBader e P. Lee, “An FPGA implementation of a flexible, parallel image processing architecture suitable for embedded vision systems,” Abril 2003. [Online]. Available: [http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=1213415&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxppls%2Fabs\\_all.jsp%3Farnumber%3D1213415](http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=1213415&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxppls%2Fabs_all.jsp%3Farnumber%3D1213415). [Acedido em Setembro 2015].
- [4] Y. Lei, K. N. U. D. Sch. of Mech. Eng., Z. Gang, R. Si-Heon e L. Choon-Young, “The Platform of Image Acquisition and Processing System Based on DSP and FPGA,” Abril 2008. [Online]. Available: [http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=4505567&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxppls%2Fabs\\_all.jsp%3Farnumber%3D4505567](http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=4505567&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxppls%2Fabs_all.jsp%3Farnumber%3D4505567). [Acedido em Setembro 2015].
- [5] C. Li, U. o. J. J. C. Sch. of Inf. Sci. & Eng., Y.-l. Zhang e Z.-n. Zheng, “Design of Image Acquisition and Processing Based on FPGA,” Maio 2009. [Online]. Available: [http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5232072&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxppls%2Fabs\\_all.jsp%3Farnumber%3D5232072](http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5232072&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxppls%2Fabs_all.jsp%3Farnumber%3D5232072). [Acedido em Setembro 2015].
- [6] H. Zhihai, W. Dejun, Z. Qiaoyun e Z. Qiaozhi., “Design of portable intelligent image acquisition system based on SOPC,” Fevereiro 2010. [Online]. Available: [http://en.cnki.com.cn/Article\\_en/CJFDTOTAL-YQXB201002024.htm](http://en.cnki.com.cn/Article_en/CJFDTOTAL-YQXB201002024.htm). [Acedido em Setembro 2015].

- [7] D. Wang e L. Kang, “Image Capture and Preprocessing System Based on FPGA,” Março 2011. [Online]. Available: [http://en.cnki.com.cn/Article\\_en/CJFDTOTAL-DSSS201103013.htm](http://en.cnki.com.cn/Article_en/CJFDTOTAL-DSSS201103013.htm). [Acedido em Setembro 2015].
- [8] H. Song, L. Tang e X. Xie, “Image Acquisition and VGA Display Based on OV7620 and FPGA,” Maio 2011. [Online]. Available: [http://en.cnki.com.cn/Article\\_en/CJFDTOTAL-DSSS201105014.htm](http://en.cnki.com.cn/Article_en/CJFDTOTAL-DSSS201105014.htm). [Acedido em Setembro 2015].
- [9] Y.-d. Zhu e Y.-b. Fang, “Image acquisition and VGA display system based on FPGA,” Maio 2011. [Online]. Available: [http://en.cnki.com.cn/Article\\_en/CJFDTOTAL-JSJY201105031.htm](http://en.cnki.com.cn/Article_en/CJFDTOTAL-JSJY201105031.htm). [Acedido em Setembro 2015].
- [10] B. Yan, Y. Sun, F. Ding e H. Yuan, “Design of CMOS image acquisition system based on FPGA,” Junho 2011. [Online]. Available: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5975870>. [Acedido em Setembro 2015].
- [11] H. Hou, W. Zhang, D. Huang e T. Zhang, “Design and Realization of Real-Time Image Acquisition and Display System Based on FPGA,” Novembro 2011. [Online]. Available: [http://link.springer.com/chapter/10.1007%2F978-3-642-27329-2\\_78](http://link.springer.com/chapter/10.1007%2F978-3-642-27329-2_78). [Acedido em Setembro 2015].
- [12] S. R. Kong, “FPGA + DSP-Based Real-Time Image Acquisition System Research and Design,” Janeiro 2012. [Online]. Available: <http://www.scientific.net/AMR.433-440.5482>. [Acedido em Setembro 2015].
- [13] P. M. C. Y. L. Qiang, “FPGA based on the image acquisition system design,” Março 2012. [Online]. Available: [http://en.cnki.com.cn/Article\\_en/CJFDTOTAL-GWCL201203020.htm](http://en.cnki.com.cn/Article_en/CJFDTOTAL-GWCL201203020.htm). [Acedido em Setembro 2015].
- [14] G. Han, Z. Li e X. Liu, “Design of Image Acquisition System Based on FPGA,” Novembro 2012. [Online]. Available: [http://link.springer.com/chapter/10.1007/978-1-4471-4790-9\\_8](http://link.springer.com/chapter/10.1007/978-1-4471-4790-9_8). [Acedido em Setembro 2015].
- [15] National Instruments, “Introduction to FPGA Vision Using the NI LabVIEW FPGA Module,” Janeiro 2014. [Online]. Available: <http://www.ni.com/white-paper/11417/en/>. [Acedido em Setembro 2015].
- [16] X. Guo e T. Liu, “Application of FPGA in high-speed CMOS digital image acquisition and color recognition system,” Junho 2014. [Online]. Available: <http://connection.ebscohost.com/c/articles/97004766/application-fpga-high-speed-cmos-digital-image-acquisition-color-recognition-system>. [Acedido em Setembro 2015].

- [17] C. L. Ramirez, R. A. Enseñat e F. J. M. Mas, “Acquisition and Digital Images Processing, Comparative Analysis of FPGA, DSP, PC for the Subtraction and Thresholding.,” em *Image Processing*, -, Croácia, InTech, 2009, pp. 437-456.
- [18] X. Guo e T. Liu, “Application of FPGA in high-speed CMOS digital image acquisition and color recognition system,” *Journal of Chemical and Pharmaceutical Research*, n° 6(6), pp. 791-798, 2014.
- [19] J. V. Vourvoulakis, J. Lygouras e J. A. Kalomiros, “Design and Implementation of a novel FPGA – based Image Acquisition System with two CMOS Sensors for Advanced Processing Techniques,” 2012.
- [20] M. Shand, “Flexible Image Acquisition using Reconfigurable Hardware,” IEEE Workshop on FPGAs for Custom Computing Machines, 1995.
- [21] L. Tian, X. Liu, J. Li e X. Guo, “Image Preprocessing of CMOS Image Acquisition System Based on FPGA,” *International Journal of Digital Content Technology and its Applications (JDCTA)*, vol. 6, n° 20, pp. 130-139, 2012.
- [22] V. Martin, G. Dunand, V. Moncada e J. Michel, “New FPGA image-oriented acquisition and real-time,” em *18th Topical Conference on High-Temperature*, Wildwood, NJ, United States, 2010.
- [23] I. Bravo, J. Baliñas, A. Gardel, J. L. Lázaro, F. Espinosa e J. García, “Efficient Smart CMOS Camera Based on FPGAs Oriented to Embedded Image Processing,” *Sensors*, n° 11, pp. 2282-2303, 2011.
- [24] J. Day, “FPGAs: addressing the graphics revolution for automotive instrumentation design,” John Day's Automotive Electronics, 19 Abril 2011. [Online]. Available: <http://johndayautomotiveelectronics.com/fpgas-addressing-the-graphics-revolution-for-automotive-instrumentation-design/>. [Acedido em Setembro 2015].
- [25] Lattice, “Lattice Semiconductor - Instrument Cluster,” Lattice, [Online]. Available: <http://www.latticesemi.com/en/Solutions/Solutions/SolutionsDetails01/InstrumentCluster.aspx>. [Acedido em Setembro 2015].
- [26] Xylon logicBRICKS, “Xylon logicBRICKS - Re-Configurable Cluster,” Xylon logicBRICKS, [Online]. Available: Xylon logicBRICKS. [Acedido em Setembro 2015].
- [27] Xilinx , “Xilinx - High Resolution Video and Graphics,” Xilinx , [Online]. Available: <http://www.xilinx.com/applications/automotive/high-resolution-video-and-graphics.html>. [Acedido em Setembro 2015].

- [28] N. Difiore, “Addressing the Graphics Revolution for Automotive Instrumentation Design Using FPGAs,” Xilinx, 30 Agosto 2011. [Online]. Available: [http://www.xilinx.com/support/documentation/white\\_papers/wp400\\_Graphics\\_Auto.pdf](http://www.xilinx.com/support/documentation/white_papers/wp400_Graphics_Auto.pdf). [Acedido em Setembro 2015].
- [29] Altera , “Altera - Infotainment - Instrument Cluster,” [Online]. Available: <https://www.altera.com/solutions/industry/automotive/applications/infotainment/aut-ins-cluster.html>. [Acedido em Setembro 2015].
- [30] Visteon Automotive Systems, “VE-FW93-10849-AA\_RelV1\_0 (Documento interno),” 2014.
- [31] INNOLUX, “TJ123NP01CA - Product Specification (Documento Interno),” 2010.
- [32] Wikipedia, “Video Graphics Array - Wikipedia, the free encyclopedia,” [Online]. Available: [https://en.wikipedia.org/wiki/Video\\_Graphics\\_Array](https://en.wikipedia.org/wiki/Video_Graphics_Array). [Acedido em Junho 2015].
- [33] Wikipedia, “Graphics display resolution - Wikipedia, the free encyclopedia,” [Online]. Available: [https://en.wikipedia.org/wiki/Graphics\\_display\\_resolution](https://en.wikipedia.org/wiki/Graphics_display_resolution). [Acedido em Junho 2015].
- [34] Wikipedia, “List of common resolutions - Wikipedia, the free encyclopedia,” [Online]. Available: [https://en.wikipedia.org/wiki/List\\_of\\_common\\_resolutions](https://en.wikipedia.org/wiki/List_of_common_resolutions). [Acedido em Junho 2015].
- [35] Zedboard, “Zedboard,” [Online]. Available: <http://zedboard.org/>. [Acedido em Agosto 2015].
- [36] Wikipedia, “FPGA Mezzanine Card - Wikipedia, the free encyclopedia,” [Online]. Available: [https://en.wikipedia.org/wiki/FPGA\\_Mezzanine\\_Card](https://en.wikipedia.org/wiki/FPGA_Mezzanine_Card). [Acedido em Julho 2015].
- [37] Avnet, Inc., “ZedBoard\_HW\_Users\_Guide - ZedBoard\_HW\_UG\_v2\_2.pdf,” 27 Janeiro 2014. [Online]. Available: [http://zedboard.org/sites/default/files/documentations/ZedBoard\\_HW\\_UG\\_v2\\_2.pdf](http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf). [Acedido em Julho 2015].
- [38] Wikipedia, “FIFO – Wikipédia, a enciclopédia livre,” [Online]. Available: <https://pt.wikipedia.org/wiki/FIFO>. [Acedido em 2015 Julho].
- [39] Xilinx, “Zynq-7000 AP SoC Product Table (XMP087) - Zynq-7000-combined-product-table.pdf,” [Online]. Available:

- [http://www.xilinx.com/publications/prod\\_mktg/zynq7000/Zynq-7000-combined-product-table.pdf](http://www.xilinx.com/publications/prod_mktg/zynq7000/Zynq-7000-combined-product-table.pdf). [Acedido em Julho 2015].
- [40] Wikipedia, “Direct memory access - Wikipedia, the free encyclopedia,” [Online]. Available: [https://en.wikipedia.org/wiki/Direct\\_memory\\_access](https://en.wikipedia.org/wiki/Direct_memory_access). [Acedido em Julho 2015].
- [41] Xilinx, “Xilinx - AXI4 IP,” [Online]. Available: [http://www.xilinx.com/ipcenter/axi4\\_ip.htm](http://www.xilinx.com/ipcenter/axi4_ip.htm). [Acedido em 2015 Julho].
- [42] Wikipedia, “Advanced Microcontroller Bus Architecture - Wikipedia, the free encyclopedia,” [Online]. Available: [https://en.wikipedia.org/wiki/Advanced\\_Microcontroller\\_Bus\\_Architecture](https://en.wikipedia.org/wiki/Advanced_Microcontroller_Bus_Architecture). [Acedido em Julho 2015].
- [43] L. Crockett, R. Elliot, M. Enderwitz, B. Stewart e D. Northcote, “The Zynq Book,” [Online]. Available: <http://www.zynqbook.com>. [Acedido em Julho 2015].
- [44] CadSoft, “Freeware | CadSoft EAGLE,” [Online]. Available: <http://www.cadsoftusa.com/download-eagle/freeware/>. [Acedido em Julho 2015].
- [45] Digilent Inc., “ZedBoard Schematics Rev. D.2,” 04 Março 2013. [Online]. Available: [http://zedboard.org/sites/default/files/documentations/ZedBoard\\_RevD.2\\_Schematic\\_130516.pdf](http://zedboard.org/sites/default/files/documentations/ZedBoard_RevD.2_Schematic_130516.pdf). [Acedido em Julho 2015].
- [46] Xilinx, “7 Series FPGAs, Clocking Resources,” 12 Junho 2015. [Online]. Available: [http://www.xilinx.com/support/documentation/user\\_guides/ug472\\_7Series\\_Clocking.pdf](http://www.xilinx.com/support/documentation/user_guides/ug472_7Series_Clocking.pdf).
- [47] Xilinx, “7 Series FPGAs, Packaging and Pinout,” 13 Novembro 2014. [Online]. Available: [http://www.xilinx.com/support/documentation/user\\_guides/ug475\\_7Series\\_Pkg\\_Pinout.pdf](http://www.xilinx.com/support/documentation/user_guides/ug475_7Series_Pkg_Pinout.pdf). [Acedido em Julho 2015].
- [48] Xilinx, “Zynq-7000 All Programmable SoC, Packaging and Pinout,” 17 Novembro 2014. [Online]. Available: [http://www.xilinx.com/support/documentation/user\\_guides/ug865-Zynq-7000-Pkg-Pinout.pdf](http://www.xilinx.com/support/documentation/user_guides/ug865-Zynq-7000-Pkg-Pinout.pdf). [Acedido em Julho 2015].
- [49] Wikipedia, “LwIP - Wikipedia, the free encyclopedia,” [Online]. Available: <https://en.wikipedia.org/wiki/LwIP>. [Acedido em Julho 2015].
- [50] J. McDougall, “Simple AMP Running Linux and Bare-Metal System on Both Zynq SoC Processors,” 14 Fevereiro 2013. [Online]. Available: [http://www.xilinx.com/support/documentation/application\\_notes/xapp1078-amp-linux-bare-metal.pdf](http://www.xilinx.com/support/documentation/application_notes/xapp1078-amp-linux-bare-metal.pdf). [Acedido em Maio 2015].

- [51] J. McDougall, "Simple AMP: Bare-Metal System Running on Both Cortex-A9 Processors," 24 Janeiro 2014. [Online]. Available: [http://www.xilinx.com/support/documentation/application\\_notes/xapp1079-amp-bare-metal-cortex-a9.pdf](http://www.xilinx.com/support/documentation/application_notes/xapp1079-amp-bare-metal-cortex-a9.pdf). [Acedido em Maio 2015].
- [52] Digilent Inc., "FMC Cards, Digilent Inc. - Digital Design Engineer's Source," [Online]. Available: <http://www.digilentinc.com/Products/Catalog.cfm?NavPath=2,1095&Cat=21>. [Acedido em 2015 Julho].
- [53] Digilent Inc., "Sensors / Interfaces / Peripheral Modules (Pmods™), Digilent Inc. - Digital Design Engineer's Source," [Online]. Available: <http://www.digilentinc.com/Products/Catalog.cfm?NavPath=2,401&Cat=9>. [Acedido em Julho 2015].







# Anexos



## Anexo A - Exemplos de código desenvolvido

### A - 1) *pattern\_generator.vhd*

A versão anotada encontra-se na secção 7.2.3.1.

```
-----  
-- Company: iTGrow  
-- Engineer: João Pedro Aleixo Duarte  
--  
-- Create Date: 12/15/2014 02:11:13 PM  
-- Design Name:  
-- Module Name: PatternGenerator - Behavioral  
-- Project Name:  
-- Target Devices:  
-- Tool Versions:  
-- Description:  
--  
-- Dependencies:  
--  
-- Revision:  
-- Revision 0.01 - File Created  
-- Revision 1.00 - PatternGenerator Created - Detected Sync Error  
-- Revision 1.01 - Sync Error Corrected  
-- Revision 1.02 - Last Column Error Corrected  
-- Additional Comments:  
--  
-----  
library IEEE;  
use IEEE.STD_LOGIC_1164.all;  
use IEEE.NUMERIC_STD.all;  
  
library UNISIM;  
use UNISIM.VComponents.all;  
  
entity PatternGenerator is  
  generic (  
    WIDTH : natural := 1280; --horizontal display width in pixels  
    HFRONTPORCH : natural := 2; --horizontal front porch width in pixels  
    HSYNCWIDTH : natural := 2; --horizontal sync pulse width in pixels  
    HBACKPORCH : natural := 2; --horizontal back porch width in pixels  
    HPOL : std_logic := '0';  
    --horizontal sync pulse polarity (1 = positive, 0 = negative)  
  
    HEIGHT : natural := 480; --vertical display width in rows  
    VFRONTPORCH : natural := 2; --vertical front porch width in rows  
    VSYNCWIDTH : natural := 2; --vertical sync pulse width in rows  
    VBACKPORCH : natural := 2; --vertical back porch width in rows  
    VPOL : std_logic := '0';  
    --vertical sync pulse polarity (1 = positive, 0 = negative)  
  
    PTRN : natural := 0;  
  
    RWIDTH : natural := 6;  
    GWIDTH : natural := 6;  
    BWIDTH : natural := 6  
  );  
port (  
  clk : in std_logic;
```

```

    resetn : in std_logic; -- Active low
    r : out std_logic_vector(RWIDTH - 1 downto 0);
    g : out std_logic_vector(GWIDTH - 1 downto 0);
    b : out std_logic_vector(BWIDTH - 1 downto 0);
    hs : out std_logic;
    vs : out std_logic;
    denb : out std_logic
);
end PatternGenerator;

architecture Behavioral of PatternGenerator is

    constant HPERIOD : natural := HFRONTPORCH + HBACKPORCH + WIDTH;
    --total number of pixel clocks in a row
    constant VPERIOD : natural := VFRONTPORCH + VBACKPORCH + HEIGHT;
    --total number of rows in column

    signal hcount : natural range 0 to HPERIOD - 1 := 0;
    --horizontal counter (counts the columns)
    signal vcount : natural range 0 to VPERIOD - 1 := 0;
    --vertical counter (counts the rows)
    signal position : natural range 1 to WIDTH * HEIGHT := 1;
    --pixel counter (counts active pixels)

begin
    control : process(clk)
    begin
        if (rising_edge(clk)) then
            if (resetn = '0') then --reset asserted
                hcount <= 0; --reset horizontal counter
                vcount <= 0; --reset vertical counter
                position <= 1; --reset pixel counter

                hs <= not HPOL; --deassert horizontal sync
                vs <= not VPOL; --deassert vertical sync

                denb <= '0'; --disable display

            else
                --counters
                if (hcount < HPERIOD - 1) then --horizontal counter (pixels)
                    hcount <= hcount + 1;
                else
                    hcount <= 0;
                    if (vcount < VPERIOD - 1) then --vertical counter (rows)
                        vcount <= vcount + 1;
                    else
                        vcount <= 0;
                        position <= 1; --reset pixel counter
                    end if;
                end if;

                --horizontal sync signal
                if (hcount >= HFRONTPORCH - 1 and hcount < HFRONTPORCH + HSYNCWIDTH
- 1) then
                    hs <= HPOL; --assert horizontal sync pulse
                else
                    hs <= not HPOL; --deassert horizontal sync pulse
                end if;
                --vertical sync signal
                if (vcount > VFRONTPORCH - 1 and vcount <= VFRONTPORCH + VSYNCWIDTH
- 1) then
                    vs <= VPOL; --assert vertical sync pulse
                else

```

```

    vs <= not VPOL; --deassert vertical sync pulse
end if;

--set display enable output
if ((hcount >= HFRONTPORCH + HBACKPORCH - 1 and hcount < HPERIOD -
1) and (vcount >= VFRONTPORCH + VBACKPORCH)) then --display time
    denb <= '1'; --enable display
    position <= position + 1; --count pixel
else --blanking time
    denb <= '0'; --disable display
end if;
end if;
end if;
end process control;

--PatternGenerator (8 color bars or position least significant bits)
pattern : process(clk)
    variable position_vector : std_logic_vector(RWIDTH + GWIDTH + BWIDTH - 1
downto 0);
begin
    if (rising_edge(clk)) then
        case PTRN is
            when 0 =>
                --White
                if (hcount >= ((0 * (WIDTH/8)) + HFRONTPORCH + HBACKPORCH - 1) and
hcount < ((1 * (WIDTH/8)) + HFRONTPORCH + HBACKPORCH - 1)) then
                    r <= (others => '1');
                    g <= (others => '1');
                    b <= (others => '1');
                --Yellow
                elsif (hcount >= ((1 * (WIDTH/8)) + HFRONTPORCH + HBACKPORCH - 1)
and hcount < ((2 * (WIDTH /8)) + HFRONTPORCH + HBACKPORCH - 1)) then
                    r <= (others => '1');
                    g <= (others => '1');
                    b <= (others => '0');
                --Cyan
                elsif (hcount >= ((2 * (WIDTH/8)) + HFRONTPORCH + HBACKPORCH - 1)
and hcount < ((3 * (WIDTH/8)) + HFRONTPORCH + HBACKPORCH - 1)) then
                    r <= (others => '0');
                    g <= (others => '1');
                    b <= (others => '1');
                --Green
                elsif (hcount >= ((3 * (WIDTH/8)) + HFRONTPORCH + HBACKPORCH - 1)
and hcount < ((4 * (WIDTH/8)) + HFRONTPORCH + HBACKPORCH - 1)) then
                    r <= (others => '0');
                    g <= (others => '1');
                    b <= (others => '0');
                --Magenta
                elsif (hcount >= ((4 * (WIDTH/8)) + HFRONTPORCH + HBACKPORCH - 1)
and hcount < ((5 * (WIDTH/8)) + HFRONTPORCH + HBACKPORCH - 1)) then
                    r <= (others => '1');
                    g <= (others => '0');
                    b <= (others => '1');
                --Red
                elsif (hcount >= ((5 * (WIDTH/8)) + HFRONTPORCH + HBACKPORCH - 1)
and hcount < ((6 * (WIDTH/8)) + HFRONTPORCH + HBACKPORCH - 1)) then
                    r <= (others => '1');
                    g <= (others => '0');
                    b <= (others => '0');

```

```

        --Blue
        elsif (hcount >= ((6 * (WIDTH/8)) + HFRONTPORCH + HBACKPORCH - 1)
and hcount < ((7 * (WIDTH/8)) + HFRONTPORCH + HBACKPORCH - 1))      then
            r <= (others => '0');
            g <= (others => '0');
            b <= (others => '1');
            --Black to White Gradient
        else
            r <= std_logic_vector(to_unsigned((((vcount - VFRONTPORCH -
VBACKPORCH) * (2 ** RWIDTH - 1))/(HEIGHT - 1)), RWIDTH));
            g <= std_logic_vector(to_unsigned((((vcount - VFRONTPORCH -
VBACKPORCH) * (2 ** GWIDTH - 1))/(HEIGHT - 1)), GWIDTH));
            b <= std_logic_vector(to_unsigned((((vcount - VFRONTPORCH -
VBACKPORCH) * (2 ** BWIDTH - 1))/(HEIGHT - 1)), BWIDTH));
            end if;
            when others =>
                -- r & g & b = position_vector(15 downto 0)
                position_vector := std_logic_vector(to_unsigned(position, RWIDTH +
GWIDTH + BWIDTH));
                r <= position_vector(RWIDTH + GWIDTH + BWIDTH - 1 downto GWIDTH +
BWIDTH);
                g <= position_vector(GWIDTH + BWIDTH - 1 downto BWIDTH);
                b <= position_vector(BWIDTH - 1 downto 0);
            end case;
        end if;
    end process pattern;

end Behavioral;

```



## A - 2) *frame\_ram.vhd* (versão inicial)

```
-----  
-- Company: iTGrow  
-- Engineer: João Pedro Aleixo Duarte  
--  
-- Create Date: 02/11/2015 12:40:19 PM  
-- Design Name:  
-- Module Name: FrameRAM - Behavioral  
-- Project Name:  
-- Target Devices:  
-- Tool Versions:  
-- Description:  
--  
-- Dependencies:  
--  
-- Revision:  
-- Revision 0.01 - File Created  
-- Additional Comments:  
--  
-----  
  
library IEEE;  
use IEEE.STD_LOGIC_1164.all;  
use IEEE.NUMERIC_STD.all;  
  
library UNISIM;  
use UNISIM.VComponents.all;  
  
entity FrameRAM is  
  generic (  
    WIDTH : natural := 1280;  
    HEIGHT : natural := 480;  
    HFRONTPORCH : natural := 2;  
    VFRONTPORCH : natural := 2;  
  
    RWIDTH : natural := 6;  
    GWIDTH : natural := 6;  
    BWIDTH : natural := 6;  
  
    HSYNC_LOST : natural := 2048;  
    VSYNC_LOST : natural := 512  
  );  
  port (  
    clk : in std_logic;  
    rst : in std_logic; -- Active low  
    r : in std_logic_vector(RWIDTH - 1 downto 0);  
    g : in std_logic_vector(GWIDTH - 1 downto 0);  
    b : in std_logic_vector(BWIDTH - 1 downto 0);  
    hs : in std_logic;  
    vs : in std_logic;  
    denb : in std_logic  
  );  
end FrameRAM;
```

Até aqui surgem os comentários iniciais e, na declaração da entidade “FrameRAM”, é possível encontrar todos os parâmetros configuráveis (largura e altura da imagem a guardar, etc.) assim como as declarações das entradas do bloco (esta versão não apresenta saídas).

```

architecture Behavioral of FrameRAM is
    constant RAMSIZE : natural := WIDTH * HEIGHT;

    signal col : natural := 0;
    signal lin : natural := 0;
    signal address : natural := 0;

    signal valid : bit := '0';

    signal sync_lost, sync_lost_q : std_logic := '1';
    signal hs_q, hs_edge : std_logic;
    signal vs_q, vs_edge : std_logic;

    type ram_type is array (0 to (RAMSIZE - 1)) of std_logic_vector(RWIDTH +
    GWIDTH + BWIDTH - 1 downto 0);
    signal ram : ram_type;

begin
    process (clk)
    begin
        if rising_edge(clk) then
            hs_q <= hs;
            vs_q <= vs;
        end if;
    end process;

    hs_edge <= '1' when hs_q = '1' and hs = '0' else '0';
    vs_edge <= '1' when vs_q = '1' and vs = '0' else '0';

```

Inicialização de variáveis e detecção de vertentes negativas dos sinais de sincronismo.

```

-- Processo que verifica sincronismo vertical e horizontal
sync : process(clk)
begin
    if rising_edge(clk) then
        if hs_edge = '1' then
            col <= 0;
        elsif hs = '1' then
            col <= col + 1;
            if (col = HSYNC_LOST) then
                sync_lost <= '1';
                sync_lost_q <= '1';
            end if;
        end if;

        if vs_edge = '1' then
            lin <= 0;
            sync_lost <= sync_lost_q;
            sync_lost_q <= '0';
        elsif (vs = '1' and hs_edge = '1') then
            lin <= lin + 1;
            if (lin = VSYNC_LOST) then
                sync_lost <= '1';
                sync_lost_q <= '1';
            end if;
        end if;
    end if;
end process sync;

```

Detecção de perdas de sincronismo. Quando a contagem do número de colunas ou linhas ultrapassar um parâmetro configurável assume-se que o sincronismo foi perdido.

```

-- Valida pixel
vldt : process (col, lin, sync_lost)
begin
  if (sync_lost = '0') then
    if ((col >= HFRONTPORCH and col < (HFRONTPORCH + WIDTH)) and (lin >=
VFRONTPORCH and lin < (VFRONTPORCH + HEIGHT))) then
      valid <= '1';
    else
      valid <= '0';
    end if;
  else
    valid <= '0';
  end if;
end process vldt;

```

Validação dos pixels das imagens. O pixel é válido caso exista sincronismo e esteja dentro do espaço útil da imagem, ignorando assim os *blank pixels*.

```

-- Processo que guarda os pixels do frame
ram_process : process (clk)
begin
  if (rising_edge(clk)) then
    if (sync_lost = '0') then
      if (valid = '1') then
        ram(address) <= r & g & b;
        address <= ((address + 1) rem (RAMSIZE));
      elsif (vs_edge = '0') then
        address <= 0;
      end if;
    else
      address <= 0;
    end if;
  end if;
end process ram_process;

end Behavioral;

```

Processo que guarda a informação na variável “ram”. O endereço é calculado através de incrementos unitários seguido da operação correspondente ao resto da divisão inteira pelo tamanho da imagem de forma a prevenir o endereçamento fora dos limites do espaço definido. O sinal de sincronismo vertical ou a perda de sincronismo reiniciam a contagem.

### A - 3) *frame\_ram.vhd* (versão 2, com integração de FIFO e interface com RAM)

```
-----  
-- Company: iTGrow  
-- Engineer: João Pedro Aleixo Duarte  
--  
-- Create Date: 12/16/2014 05:16:04 PM  
-- Design Name:  
-- Module Name: FrameRAM - Behavioral  
-- Project Name:  
-- Target Devices:  
-- Tool Versions:  
-- Description:  
--  
-- Dependencies:  
--  
-- Revision:  
-- Revision 0.01 - File Created  
-- Additional Comments:  
--  
-----  
  
library IEEE;  
use IEEE.STD_LOGIC_1164.all;  
use IEEE.NUMERIC_STD.all;  
  
entity FrameRAM is  
  generic (  
    WIDTH : natural := 1280;  
    HEIGHT : natural := 480;  
    HFRONTPORCH : natural := 2;  
    VFRONTPORCH : natural := 2  
  );  
  port (  
    r : in std_logic_vector(4 downto 0);  
    g : in std_logic_vector(5 downto 0);  
    b : in std_logic_vector(4 downto 0);  
    clock : in std_logic;  
    hsync : in std_logic;  
    vsync : in std_logic;  
    -- RAM interface  
    rclk : in std_logic;  
    rrst : in std_logic;  
    raddr : out std_logic_vector(31 downto 0);  
    rwdat : out std_logic_vector(31 downto 0);  
    rwen : out std_logic;  
    rwstall : in std_logic  
  );  
end FrameRAM;
```

Comentários iniciais e declaração da entidade com a definição de parâmetros e de entradas e saídas. Destaque para as entradas e saídas de interface com uma memória RAM genérica.

```
architecture Behavioral of FrameRAM is  
  
  component gh_fifo_async_rrd_sr_wf is  
    generic (  
      add_width : integer := 8; -- min value is 2 (4 memory locations)  
      data_width : integer := 8  
    ); -- size of data bus  
    port (  
      clk_WR : in STD_LOGIC; -- write clock
```

```

clk_RD : in STD_LOGIC; -- read clock
rst : in STD_LOGIC; -- resets counters
srst : in STD_LOGIC := '0'; -- resets counters (sync with clk_WR)
WR : in STD_LOGIC; -- write control
RD : in STD_LOGIC; -- read control
D : in STD_LOGIC_VECTOR (data_width - 1 downto 0);
Q : out STD_LOGIC_VECTOR (data_width - 1 downto 0);
empty : out STD_LOGIC;
qfull : out STD_LOGIC;
hfull : out STD_LOGIC;
qqqfull : out STD_LOGIC;
afull : out STD_LOGIC;
full : out STD_LOGIC
);
end component;

```

Declaração de componente (FIFO disponível na comunidade OpenCores).

```

signal col : natural := 0;
signal lin : natural := 0;
signal address : natural := 0;

signal valid : std_logic := '0';

signal sync_lost, sync_lost_q : std_logic := '1';
signal hsync_q, hsync_edge : std_logic;
signal vsync_q, vsync_edge : std_logic;

signal fifo_reset : std_logic := '0';
signal fifo_read, fifo_write : std_logic := '0';
signal fifo_full, fifo_empty : std_logic;
signal fifo_data_in, fifo_data_out : std_logic_vector(15 downto 0);

type rfstate_type is (idle, stream);

type rfregs_type is record
state : rfstate_type;
baseaddress : unsigned(31 downto 0);
currentaddress : unsigned(31 downto 0);
burstcnt : unsigned(4 downto 0);
end record;

signal rfregs : rfregs_type;

```

Declaração de sinais para o controlo dos processos.

```

begin
process (clock)
begin
if rising_edge(clock) then
hsync_q <= hsync;
vsync_q <= vsync;
end if;
end process;

hsync_edge <= '1' when hsync_q = '1' and hsync = '0' else '0';
vsync_edge <= '1' when vsync_q = '1' and vsync = '0' else '0';

```

Deteção das vertentes (negativas) dos sinais de sincronismo.

```

-- Processo que verifica sincronismo vertical e horizontal
sync : process(clock)
begin
  if rising_edge(clock) then
    if hsync_edge = '1' then
      col <= 0;
    elsif hsync = '1' then
      col <= col + 1;
      --if (col = 2048) then
      if (col = 127) then
        sync_lost <= '1';
        sync_lost_q <= '1';
      end if;
    end if;

    if vsync_edge = '1' then
      lin <= 0;
      sync_lost <= sync_lost_q;
      sync_lost_q <= '0';
    elsif (vsync = '1' and hsync_edge = '1') then
      lin <= lin + 1;
      --if (lin = 512) then
      if (lin = 67) then
        sync_lost <= '1';
        sync_lost_q <= '1';
      end if;
    end if;
  end if;
end process sync;

-- Valida pixel
vldt : process (col, lin, sync_lost)
begin
  if (sync_lost = '0') then
    if ((col >= HFRONTPORCH and col < (HFRONTPORCH + WIDTH)) and (lin >=
VFRONTPORCH and lin < (VFRONTPORCH + HEIGHT))) then
      valid <= '1';
    else
      valid <= '0';
    end if;
  end if;
end process vldt;

```

Processos que gerem a perda de sincronismo e a validação dos pixéis.

```

fifo_data_in <= r & g & b;
fifo_write <= valid;

fifo_reset <= sync_lost;

myfifo : gh_fifo_async_rrd_sr_wf
generic map(
  add_width => 8,
  data_width => fifo_data_in'LENGTH
) -- size of data bus
port map(
  clk_WR => clock,
  clk_RD => rclk,
  rst => fifo_reset,
  srst => fifo_reset,
  WR => fifo_write,
  RD => fifo_read,

```

```

D => fifo_data_in,
Q => fifo_data_out,
empty => fifo_empty,
qfull => open,
hfull => fifo_full,
qqqfull => open,
afull => open,
full => open
);

rwdat <= fifo_data_out & fifo_data_out;
raddr <= std_logic_vector(rfregs.currentaddress);

-- RAM filler
process (rclk, rfregs, rwstall, rrst)
variable w : rfregs_type;
begin
    w := rfregs;

    case rfregs.state is
        when idle =>
            rwen <= '0';
            fifo_read <= '0';
            w.burstcnt := (others => '1');

            if fifo_full = '1' then
                w.state := stream;
            end if;

        when stream =>
            rwen <= '1';
            fifo_read <= not rwstall;
            if rwstall = '0' then
                w.currentaddress := rfregs.currentaddress + 1;
                if rfregs.burstcnt = 0 then
                    rwen <= '0';
                    fifo_read <= '0';
                    w.state := idle;
                else
                    w.burstcnt := rfregs.burstcnt - 1;
                end if;
            end if;

        when others => null;
    end case;
    if rrst = '1' then
        w.state := idle;
        w.currentaddress := (others => '0');
    end if;

    if rising_edge(rclk) then
        rfregs <= w;
    end if;

end process;
end Behavioral;

```

Atribuição a sinais. Instanciação de “myfifo”. Processo que gere a máquina de estados do sistema (2 estados: *idle* e *stream*). Em *idle* a leitura está suspensa e a FIFO vai enchendo até ser levantado o sinal “fifo\_full” passando ao estado *stream*. Em *stream* é enviado um *burst* de dados para a memória com um tamanho definido. No final do *burst* regressamos ao estado *idle*.

#### A - 4) *pattern\_generator\_tb.vhd*

A versão anotada encontra-se na secção 7.3.1.

```
-----  
-- Company: iTGrow  
-- Engineer: João Pedro Aleixo Duarte  
--  
-- Create Date: 12/17/2014 06:59:10 PM  
-- Design Name:  
-- Module Name: pattern_generator_tb - Behavioral  
-- Project Name:  
-- Target Devices:  
-- Tool Versions:  
-- Description:  
--  
-- Dependencies:  
--  
-- Revision:  
-- Revision 0.01 - file Created  
-- Additional Comments:  
--  
-----  
  
library IEEE;  
use IEEE.STD_LOGIC_1164.all;  
use IEEE.NUMERIC_STD.all;  
  
library UNISIM;  
use UNISIM.VComponents.all;  
  
entity pattern_generator_tb is  
end pattern_generator_tb;  
  
architecture Behavioral of pattern_generator_tb is  
  
    component PatternGenerator  
        generic (  
            WIDTH : natural := 1280; --horizontal display width in pixels  
            HFRONTPORCH : natural := 2; --horizontal front porch width in pixels  
            HSYNCWIDTH : natural := 2; --horizontal sync pulse width in pixels  
            HBACKPORCH : natural := 2; --horizontal back porch width in pixels  
            HPOL : std_logic := '0';  
            --horizontal sync pulse polarity (1 = positive, 0 = negative)  
  
            HEIGHT : natural := 480; --vertical display width in rows  
            VFRONTPORCH : natural := 2; --vertical front porch width in rows  
            VSYNCWIDTH : natural := 2; --vertical sync pulse width in rows  
            VBACKPORCH : natural := 2; --vertical back porch width in rows  
            VPOL : std_logic := '0';  
            --vertical sync pulse polarity (1 = positive, 0 = negative)  
  
            PPTRN : natural := 0;  
  
            RWIDTH : natural := 6;  
            GWIDTH : natural := 6;  
            BWIDTH : natural := 6  
        );  
    port (  
        clk : in std_logic;  
        resetn : in std_logic; -- Active low  
        r : out std_logic_vector(RWIDTH - 1 downto 0);  
        g : out std_logic_vector(GWIDTH - 1 downto 0);  
        b : out std_logic_vector(BWIDTH - 1 downto 0);  
    );  
end architecture;
```



```

    hs : out std_logic;
    vs : out std_logic;
    denb : out std_logic
);
end component;

--Inputs
signal clk : std_logic := '0';
signal resetn : std_logic := '0'; -- Active low

--Outputs
signal r : std_logic_vector(5 downto 0);
signal g : std_logic_vector(5 downto 0);
signal b : std_logic_vector(5 downto 0);
signal hs : std_logic;
signal vs : std_logic;
signal denb : std_logic;

constant clk_period : time := 10 ns;

begin
--Instantiate the Unit Under Test (UUT)
 uut : PatternGenerator
 generic map(
    WIDTH => 8,
    HEIGHT => 8,
    PTRN => 0
 )
 port map(
    clk => clk,
    resetn => resetn,
    r => r,
    g => g,
    b => b,
    hs => hs,
    vs => vs,
    denb => denb
 );

--Clock process definition
clk_process : process
begin
    clk <= '0';
    wait for clk_period/2;
    clk <= '1';
    wait for clk_period/2;
end process;

--Stimulus process
stim_process : process
begin
    wait for 20 ns;
    resetn <= '1';
end process;

end Behavioral;

```

## A - 5) FrameRAMwAXI\_v1\_0.vhd

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

entity FrameRAMwAXI_v1_0_M00_AXIS is
  generic (

    WIDTH : natural := 1280; --horizontal display width in pixels
    HFRONTPORCH : natural := 2; --horizontal front porch width in pixels
    HSYNCWIDTH : natural := 2; --horizontal sync pulse width in pixels
    HBACKPORCH : natural := 2; --horizontal back porch width in pixels
    HPOL : std_logic := '0'; --horizontal sync pulse polarity (1 = positive, 0 =
negative)

    HEIGHT : natural := 480; --vertical display width in rows
    VFRONTPORCH : natural := 2; --vertical front porch width in rows
    VSYNCWIDTH : natural := 2; --vertical sync pulse width in rows
    VBACKPORCH : natural := 2; --vertical back porch width in rows
    VPOL : std_logic := '0'; --vertical sync pulse polarity (1 = positive, 0 =
negative)

    RWIDTH : natural := 6;
    GWIDTH : natural := 6;
    BWIDTH : natural := 6;

    HSYNC_LOST : natural := 2048;
    VSYNC_LOST : natural := 512;

    -- Should be equal to the SPI register R7 of the TFT-LCD module
    MODE : std_logic_vector(1 downto 0) := "11"; -- "11" => HS+VS+DE mode; "10"
=> HS+VS mode; "01" => DE only mode; "00" => NA

    -- Width of S_AXIS address bus. The slave accepts the read and write
addresses of width C_M_AXIS_TDATA_WIDTH.
    C_M_AXIS_TDATA_WIDTH : integer := 32
  );
  port (
    -- Users to add ports here

    r : in std_logic_vector(RWIDTH - 1 downto 0);
    g : in std_logic_vector(GWIDTH - 1 downto 0);
    b : in std_logic_vector(BWIDTH - 1 downto 0);
    hs : in std_logic;
    vs : in std_logic;
    denb : in std_logic;

    -- User ports ends
    -- Do not modify the ports beyond this line

    -- Global ports
    M_AXIS_ACLK : in std_logic;
    M_AXIS_ARESETN : in std_logic;
    -- Master Stream Ports. TVALID indicates that the master is driving a valid
transfer, A transfer takes place when both TVALID and TREADY are asserted.
    M_AXIS_TVALID : out std_logic;
    -- TDATA is the primary payload that is used to provide the data that is
passing across the interface from the master.
    M_AXIS_TDATA : out std_logic_vector(C_M_AXIS_TDATA_WIDTH - 1 downto 0);
    -- TSTRB is the byte qualifier that indicates whether the content of the
associated byte of TDATA is processed as a data byte or a position byte.
    M_AXIS_TSTRB : out std_logic_vector((C_M_AXIS_TDATA_WIDTH/8) - 1 downto 0);
```

```

    -- TLAST indicates the boundary of a packet.
    M_AXIS_TLAST : out std_logic;
    -- TREADY indicates that the slave can accept a transfer in the current
cycle.
    M_AXIS_TREADY : in std_logic
);
end FrameRAMwAXI_v1_0_M00_AXIS;

```

Declaração da entidade com a definição de parâmetros e de entradas e saídas. Destaque para a inclusão das entradas e saídas de acordo com as interfaces AXI4-Stream.

```

architecture implementation of FrameRAMwAXI_v1_0_M00_AXIS is

    constant HPERIOD : natural := HFRONTPORCH + HBACKPORCH + WIDTH;
    constant VPERIOD : natural := VFRONTPORCH + VBACKPORCH + HEIGHT;

    constant DENB_ENABLE : real := real((((VFRONTPORCH + VBACKPORCH) * HPERIOD)
+ (HFRONTPORCH + HBACKPORCH) - 1));
    constant DENB_ENABLE_L : integer := integer(DENB_ENABLE - (0.03 *
DENB_ENABLE)); -- 97% of DENB_ENABLE
    constant DENB_ENABLE_U : integer := integer(DENB_ENABLE + (0.03 *
DENB_ENABLE)); -- 103% of DENB_ENABLE

    -- constant FILL_PATTERN: std_logic_vector((C_M_AXIS_TDATA_WIDTH-
(RWIDTH+GWIDTH+BWIDTH)-1) downto 0) := "10101010101010";
    constant FILL_PATTERN : std_logic_vector((C_M_AXIS_TDATA_WIDTH - (RWIDTH +
GWIDTH + BWIDTH) - 1) downto 0) := (others => '0');

    signal col : natural := 0;
    signal lin : natural := 0;

    signal denb_counter1 : natural := 0; -- Counts denb rising edges
    signal denb_counter2 : natural := 0; -- Counts clock cycles between denb
falling edge and rising edge

    signal array_counter : natural := 0;
    signal stream_counter : natural := 0;

    signal valid : std_logic := '0';

    signal sync_lost, sync_lost_q : std_logic := '1';
    signal hs_q, hs_edge : std_logic := '0';
    signal vs_q, vs_edge : std_logic := '0';
    signal denb_q, denb_redge, denb_fedge : std_logic := '0';

    signal data_q : std_logic_vector(RWIDTH + GWIDTH + BWIDTH - 1 downto 0) :=
(others => '0');

    type ram_type is array (0 to (WIDTH - 1)) of std_logic_vector(RWIDTH +
GWIDTH + BWIDTH - 1 downto 0);
    signal ram : ram_type := ((others => (others => '0')));

    -- Define the states of state machine
    -- The control state machine oversees the writing of input streaming data to
the FIFO,
    -- and outputs the streaming data from the FIFO
    type state is (IDLE_STATE, -- This is the initial/idle state
STREAM_STATE); -- in this state the stream data is output through
M_AXIS_TDATA

    -- State variable
    signal mst_exec_state : state := IDLE_STATE;

```

```

-- AXI Stream internal signals
-- Streaming data valid
signal axis_tvalid : std_logic := '0';
-- Last of the streaming data
signal axis_tlast : std_logic := '0';

--FIFO implementation signals
signal stream_data_out : std_logic_vector(C_M_AXIS_TDATA_WIDTH - 1 downto 0)
:= (others => '0');

```

Declaração de sinais para o controlo dos processos.

```

begin
-- I/O Connections assignments
M_AXIS_TDATA <= stream_data_out;
M_AXIS_TSTRB <= (others => '1');
M_AXIS_TVALID <= axis_tvalid;
M_AXIS_TLAST <= axis_tlast;

-- Delayed signals
process (M_AXIS_ACLK)
begin
if (rising_edge(M_AXIS_ACLK)) then
hs_q <= hs;
vs_q <= vs;
denb_q <= denb;
data_q <= r & g & b;
end if;
end process;

-- Edge detector
hs_edge <= '1' when ((hs_q = not HPOL) and hs = HPOL) else '0';
vs_edge <= '1' when ((vs_q = not VPOL) and vs = VPOL) else '0';
denb_redge <= '1' when (denb_q = '0' and denb = '1') else '0';
denb_fedge <= '1' when (denb_q = '1' and denb = '0') else '0';

```

Atribuições às saídas e detecção das vertentes dos sinais de sincronismo.

```

-- process that verifies sync signals and assigns counters
sync : process(M_AXIS_ACLK)
begin
if (rising_edge(M_AXIS_ACLK)) then
if (M_AXIS_ARESETN = '0') then
col <= 0;
lin <= 0;
denb_counter1 <= 0;
denb_counter2 <= 0;
sync_lost <= '1';
sync_lost_q <= '1';
else
case (MODE) is -- "11" => HS+VS+DE mode; "10" => HS+VS mode; "01" => DE
only mode; "00" => NA
when "11" | "10" => -- HS+VS+DE mode or HS+VS mode
if hs_edge = '1' then
col <= 0;
lin <= lin + 1;
else
col <= col + 1;
if (col >= HSYNC_LOST) then
sync_lost <= '1';
sync_lost_q <= '1';

```

```

        end if;
    end if;

    if vs_edge = '1' then -- vs_edge is earlier than hs_edge (lin almost
starts at 1)
        lin <= 0;
        sync_lost <= sync_lost_q;
        sync_lost_q <= '0';
        if (lin >= VSYNC_LOST) then
            sync_lost <= '1';
            sync_lost_q <= '1';
        end if;
    end if;

when "01" => -- DE only mode
    if denb_redge = '1' then
        denb_counter1 <= denb_counter1 + 1;
    end if;
    if ((denb_counter1 >= HEIGHT) or (sync_lost = '0')) then
        if denb_fedge = '1' then
            denb_counter2 <= 0;
        elsif denb = '0' then
            denb_counter2 <= denb_counter2 + 1;
        end if;
        if denb_redge = '1' then
            if ((denb_counter2 >= DENB_ENABLE_L) and (denb_counter2 <=
DENB_ENABLE_U)) then -- Valid window
                denb_counter1 <= 1;
                sync_lost <= sync_lost_q;
                sync_lost_q <= '0';
            elsif (denb_counter2 > DENB_ENABLE_U) then -- Above valid window
                denb_counter1 <= 0;
                sync_lost <= '1';
                sync_lost_q <= '1';
            end if;
        end if;
    end if;

when others => -- Reset equivalent
    col <= 0;
    lin <= 0;
    denb_counter1 <= 0;
    denb_counter2 <= 0;
    sync_lost <= '1';
    sync_lost_q <= '1';
end case;
end if;
end if;
end process sync;

-- Validates pixel
vldt : process (M_AXIS_ACLK)
begin
    if (rising_edge(M_AXIS_ACLK)) then
        if (sync_lost = '0') then
            case (MODE) is -- "11" => HS+VS+DE mode; "10" => HS+VS mode; "01" => DE
only mode; "00" => NA
                when "11" | "01" => -- HS+VS+DE mode or DE only mode
                    if denb = '1' then
                        valid <= '1';
                    else
                        valid <= '0';
                    end if;
                when "10" => -- HS+VS mode

```

```

        if (((col >= HBACKPORCH - 1) and (col < HBACKPORCH + WIDTH - 1)) and
((lin > VBACKPORCH) and (lin <= VBACKPORCH + HEIGHT)))) then -- vs_edge is
earlier than hs_edge (lin almost starts at 1)
            valid <= '1';
        else
            valid <= '0';
        end if;
        when others => -- Reset equivalent
            valid <= '0';
        end case;
    else
        valid <= '0';
    end if;
end if;
end process vldt;

```

Processos que gerem a perda de sincronismo e a validação dos pixéis.

```

-- process that stores pixel data in the array (FIFO)
ram_process : process (M_AXIS_ACLK)
begin
    if (rising_edge(M_AXIS_ACLK)) then
        if (sync_lost = '0') then
            if (valid = '1') then
                ram(array_counter) <= data_q;
                array_counter <= ((array_counter + 1) rem (WIDTH));
            elsif ((hs_edge = '1') or (denb_fedge = '1')) then
                array_counter <= 0;
            end if;
        else
            array_counter <= 0;
        end if;
    end if;
end process ram_process;

-- Control state machine implementation
stt_machine : process (M_AXIS_ACLK)
begin
    if (rising_edge (M_AXIS_ACLK)) then
        if (M_AXIS_ARESETN = '0') then
            -- Synchronous reset (active low)
            mst_exec_state <= IDLE_STATE;
            stream_data_out <= (others => '0');
        else
            case (mst_exec_state) is
                when IDLE_STATE =>
                    stream_counter <= 0;
                    axis_tvalid <= '0';
                    axis_tlast <= '0';
                    if valid = '1' then
                        mst_exec_state <= STREAM_STATE;
                    else
                        mst_exec_state <= IDLE_STATE;
                    end if;
                when STREAM_STATE =>
                    axis_tvalid <= '1';
                    stream_data_out <= FILL_PATTERN & ram(stream_counter);
                    if stream_counter >= WIDTH - 1 then
                        axis_tlast <= '1';
                        mst_exec_state <= IDLE_STATE;
                    else
                        if M_AXIS_TREADY = '1' then
                            stream_counter <= stream_counter + 1;
                        end if;
                    end if;
                end case;
            end if;
        end process stt_machine;

```

```
        end if;
        mst_exec_state <= STREAM_STATE;
    end if;
    when others =>
        mst_exec_state <= IDLE_STATE;
    end case;
end if;
end if;
end process stt_machine;

end implementation;
```

Processos que trata da gestão do *array* de dados (FIFO) e da máquina de estados do sistema (2 estados: “IDLE\_STATE” e “STREAM\_STATE”).

## A - 6) *main.c* (aplicação que configura o módulo VDMA e o controlador HDMI via I<sup>2</sup>C)

Na secção 7.3.4 podemos encontrar uma versão compacta deste código, com algumas anotações. Aqui podemos observar todas as configurações efectuadas, assim como a definição de algumas funções de inicialização/configuração dos módulos presentes no *datapath*, para além do programa principal que já foi apresentado.

```
#include "zed_iic.h"
#include "video_resolution.h"
#include "video_generator.h"

#include <stdio.h>
#include "platform.h"
#include "xuartps_hw.h"

#include "xaxivdma.h"
#include "xvtc.h"
#include "xil_cache.h"

#include "xparameters.h"

/***** Constant Definitions *****/

/*
 * HDMI I2C Configuration
 */

#ifndef ADV7511_ADDR
#define ADV7511_ADDR 0x72
#endif

#ifndef CARRIER_HDMI_OUT_CONFIG_LEN
#define CARRIER_HDMI_OUT_CONFIG_LEN (40)
#endif

Xuint8 carrier_hdmi_out_config[CARRIER_HDMI_OUT_CONFIG_LEN][3] =
{
    {ADV7511_ADDR>>1, 0x15, 0x01}, // Input YCbCr 4:2:2 with separate syncs
    {ADV7511_ADDR>>1, 0x16, 0x38}, // Output format 444, Input Color Depth = 8
    // R0x16[ 7] = Output Video Format = 0 (444)
    // R0x16[5:4] = Input Video Color Depth = 11 (8 bits/color)
    // R0x16[3:2] = Input Video Style = 10 (style 1)
    // R0x16[ 1] = DDR Input Edge = 0 (falling edge)
    // R0x16[ 0] = Output Color Space = 0 (RGB)
    // HDTV YCbCr (16to235) to RGB (16to235)
    {ADV7511_ADDR>>1, 0x18, 0xAC},
    {ADV7511_ADDR>>1, 0x19, 0x53},
    {ADV7511_ADDR>>1, 0x1A, 0x08},
    {ADV7511_ADDR>>1, 0x1B, 0x00},
    {ADV7511_ADDR>>1, 0x1C, 0x00},
    {ADV7511_ADDR>>1, 0x1D, 0x00},
    {ADV7511_ADDR>>1, 0x1E, 0x19},
    {ADV7511_ADDR>>1, 0x1F, 0xD6},
    {ADV7511_ADDR>>1, 0x20, 0x1C},
    {ADV7511_ADDR>>1, 0x21, 0x56},
    {ADV7511_ADDR>>1, 0x22, 0x08},
    {ADV7511_ADDR>>1, 0x23, 0x00},
    {ADV7511_ADDR>>1, 0x24, 0x1E},
    {ADV7511_ADDR>>1, 0x25, 0x88},
    {ADV7511_ADDR>>1, 0x26, 0x02},

```



```

{ADV7511_ADDR>>1, 0x27, 0x91},
{ADV7511_ADDR>>1, 0x28, 0x1F},
{ADV7511_ADDR>>1, 0x29, 0xFF},
{ADV7511_ADDR>>1, 0x2A, 0x08},
{ADV7511_ADDR>>1, 0x2B, 0x00},
{ADV7511_ADDR>>1, 0x2C, 0x0E},
{ADV7511_ADDR>>1, 0x2D, 0x85},
{ADV7511_ADDR>>1, 0x2E, 0x18},
{ADV7511_ADDR>>1, 0x2F, 0xBE},
{ADV7511_ADDR>>1, 0x41, 0x10}, // Power down control
// R0x41[ 6] = PowerDown = 0 (power-up)
{ADV7511_ADDR>>1, 0x48, 0x08}, // Video Input Justification
// R0x48[8:7] = Video Input Justification = 01 (right justified)
{ADV7511_ADDR>>1, 0x55, 0x00}, // Set RGB in AVinfo Frame
// R0x55[6:5] = Output Format = 00 (RGB)
{ADV7511_ADDR>>1, 0x56, 0x28}, // Aspect Ratio
// R0x56[5:4] = Picture Aspect Ratio = 10 (16:9)
// R0x56[3:0] = Active Format Aspect Ratio = 1000 (Same as Aspect Ratio)
{ADV7511_ADDR>>1, 0x98, 0x03}, // ADI Recommended Write
{ADV7511_ADDR>>1, 0x9A, 0xE0}, // ADI Recommended Write
{ADV7511_ADDR>>1, 0x9C, 0x30}, // PLL Filter R1 Value
{ADV7511_ADDR>>1, 0x9D, 0x61}, // Set clock divide
{ADV7511_ADDR>>1, 0xA2, 0xA4}, // ADI Recommended Write
{ADV7511_ADDR>>1, 0xA3, 0xA4}, // ADI Recommended Write
{ADV7511_ADDR>>1, 0xAF, 0x04}, // HDMI/DVI Modes
// R0xAF[ 7] = HDCP Enable = 0 (HDCP disabled)
// R0xAF[ 4] = Frame Encryption = 0 (Current frame NOT HDCP encrypted)
// R0xAF[3:2] = 01 (fixed)
// R0xAF[ 1] = HDMI/DVI Mode Select = 0 (DVI Mode)
//{ADV7511_ADDR>>1, 0xBA, 0x00}, // Programmable delay for input video
clock = 000 = -1.2ns
{ADV7511_ADDR>>1, 0xBA, 0x60}, // Programmable delay for input video clock
= 011 = no delay
{ADV7511_ADDR>>1, 0xE0, 0xD0}, // Must be set to 0xD0 for proper operation
{ADV7511_ADDR>>1, 0xF9, 0x00} // Fixed I2C Address (This should be set to
a non-conflicting I2C address)
};

/*
 * Device related constants. These need to be defined as per the HW system.
 */
#define DMA_DEVICE_ID XPAR_AXI_VDMA_0_DEVICE_ID
#define VTC_DEVICE_ID XPAR_ZED_HDMI_DISPLAY_V_TC_0_DEVICE_ID
#define HDMI_IIC_BASEADDR XPAR_ZED_HDMI_DISPLAY_ZED_HDMI_IIC_0_BASEADDR
#define TPG_DEVICE_ID XPAR_IMAGE_INPUT_V_TPG_0_DEVICE_ID

#define DDR_BASE_ADDR 0x01000000
#define DDR_HIGH_ADDR 0x0F000000

/* Memory space for the frame buffers
 */
#define WR_BASE_ADDR (DDR_BASE_ADDR + 0x01000000)
#define WR_HIGH_ADDR (DDR_BASE_ADDR + 0x05000000)
#define WR_SPACE (WR_HIGH_ADDR - WR_BASE_ADDR)

#define RD_BASE_ADDR (DDR_BASE_ADDR + 0x06000000)
#define RD_HIGH_ADDR (DDR_BASE_ADDR + 0x0A000000)
#define RD_SPACE (RD_HIGH_ADDR - RD_BASE_ADDR)

/* Frame size related constants
 */
#define WR_FRAME_START_OFFSET 0 /* No offset */

```

```

#define WR_FRAME_HORIZONTAL_SIZE 1280*4 /* 1280 pixels (columns), each pixel 4
bytes */
#define WR_FRAME_VERTICAL_SIZE 480 /* 480 pixels (rows) */

#define RD_FRAME_START_OFFSET 0 /* No offset */
#define RD_FRAME_HORIZONTAL_SIZE 1920*4 /* 1920 pixels (columns), each pixel 4
bytes */
#define RD_FRAME_VERTICAL_SIZE 1080 /* 1080 pixels (rows) */

#define FRAME_START_OFFSET ((300*1920)+320)*4 /* Center image (1280c*480r*4b)
*/
#define VERTICAL_OFFSET 300
#define HORIZONTAL_OFFSET 320*4

/* Number of frames to work on, this is to set the frame count threshold
*/
#define NUMBER_OF_FRAMES 4

/* Delay timer counter
*
* WARNING: If you are using fsync, please increase the delay counter value
* to be 255. Because with fsync, the inter-frame delay is long. If you do
not
* care about inactivity of the hardware, set this counter to be 0, which
* disables delay interrupt.
*/
#define DELAY_TIMER_COUNTER 0

/*
* Device instance definitions
*/
XAxiVdma AxiVdma;
XVtc Vtc;
zed_iic_t Iic;

/* DMA channel setup
*/
static XAxiVdma_DmaSetup ReadCfg;
static XAxiVdma_DmaSetup WriteCfg;

/***** Function Prototypes *****/
static int Iic_Initialize(zed_iic_t *InstancePtr);

void Vtc_Initialize(XVtc *InstancePtr);

static int Vdma_Initialize(XAxiVdma *InstancePtr);
static int Vdma_ReadSetup(XAxiVdma *InstancePtr);
static int Vdma_WriteSetup(XAxiVdma * InstancePtr);
static int Vdma_StartTransfer(XAxiVdma *InstancePtr);

/*****
*
* Main function
*
*****/
int main(void)
{
    xil_printf("\n\r");
    xil_printf("-----\n\r");
    xil_printf("--          ZedBoard HDMI Display Controller          --\n\r");
    xil_printf("-----\n\r");

```

```

xil_printf("\n\r");

int Status;

// Configure VDMA engine
Status = Vdma_Initialize(&AxiVdma);
if (Status != XST_SUCCESS)
{
    xil_printf( "ERROR : Failed to initialize VTC controller\n\r" );
    return XST_FAILURE;
}

// Configure VTC on output data path
xil_printf( "Video Timing Controller (generator) Initialization ... \n\r" );
Vtc_Initialize(&Vtc);

// Configure ADV7511 via IIC
xil_printf( "HDMI IIC Initialization ... \n\r" );
Status = Iic_Initialize(&Iic);
if (Status != XST_SUCCESS)
{
    xil_printf( "ERROR : Failed to initialize IIC driver\n\r" );
    return XST_FAILURE;
}

int a, f, c, r;
a = 1;
u32 data, address_read, address_write, address_write_sof; // sof = start of
frame

data = 0;
while(a)
{
    // Wait for DMA to synchronize.
    Xil_DCacheFlush();
    for(f = 0; f < NUMBER_OF_FRAMES; f++)
    {
        address_read = WR_BASE_ADDR + (WR_FRAME_HORIZONTAL_SIZE *
WR_FRAME_VERTICAL_SIZE * f);
        address_write_sof = RD_BASE_ADDR + (RD_FRAME_HORIZONTAL_SIZE *
RD_FRAME_VERTICAL_SIZE * f) + FRAME_START_OFFSET;
        for(r = 0; r < WR_FRAME_VERTICAL_SIZE; r++)
        {
            address_write = address_write_sof + (RD_FRAME_HORIZONTAL_SIZE * r);
            for(c = 0; c < (WR_FRAME_HORIZONTAL_SIZE/4); c++)
            {
                data = Xil_In32(address_read);
                Xil_Out32(address_write, data);
                address_read += 4;
                address_write += 4;
            }
        }
    }
}

return 0;
}
/*****
**
*
* This function sets up the IIC transfers to ADV7511
*
* @param InstancePtr is the instance pointer to the IIC device.
*
**
*****/

```

```

* @return XST_SUCCESS if the setup is successful, XST_FAILURE otherwise.
*
* @note None.
*
*****/
static int Iic_Initialize(zed_iic_t *InstancePtr)
{
    int Status;

    Status = zed_iic_axi_init(InstancePtr,"ZED HDMI I2C Controller",
HDMI_IIC_BASEADDR);
    if (Status != XST_SUCCESS)
    {
        xil_printf( "ERROR : Failed to initialize IIC driver\n\r" );
        return XST_FAILURE;
    }

    xil_printf( "HDMI Output Initialization ...\n\r" );
    int i;
    for ( i = 0; i < CARRIER_HDMI_OUT_CONFIG_LEN; i++ )
    {
        Iic.fpIicWrite(InstancePtr, carrier_hdmi_out_config[i][0],
carrier_hdmi_out_config[i][1], &(carrier_hdmi_out_config[i][2]), 1 );
    }

    return XST_SUCCESS;
}

/*****
**
*
* This function sets up the Video Timing Controller
*
* @param InstancePtr is the instance pointer to the VTC device.
*
* @return XST_SUCCESS if the setup is successful, XST_FAILURE otherwise.
*
* @note None.
*
*****/
void Vtc_Initialize(XVtc *InstancePtr)
{
    vgen_init(InstancePtr, VTC_DEVICE_ID);
    vgen_config(InstancePtr, VIDEO_RESOLUTION_1080P, 1);
}

/*****
**
*
* This function sets up the Video DMA
*
* @param InstancePtr is the instance pointer to the DMA engine.
*
* @return XST_SUCCESS if the setup is successful, XST_FAILURE otherwise.
*
* @note None.
*
*****/
static int Vdma_Initialize(XAxiVdma *InstancePtr)
{
    XAxiVdma_Config *VdmaCfgPtr = NULL;
    int Status;

    /* The information of the XAxiVdma_Config comes from hardware build.

```

```

    * The user IP should pass this information to the AXI DMA core.
    */
VdmaCfgPtr = XAxiVdma_LookupConfig(DMA_DEVICE_ID);
if (!VdmaCfgPtr)
{
    xil_printf("No video DMA found for ID %d\r\n", DMA_DEVICE_ID);
    return XST_FAILURE;
}

/* Initialize DMA engine */
Status = XAxiVdma_CfgInitialize(InstancePtr, VdmaCfgPtr, VdmaCfgPtr->
BaseAddress);
if (Status != XST_SUCCESS)
{
    xil_printf("Configuration Initialization failed %d\r\n", Status);
    return XST_FAILURE;
}

/*
 * Setup your video IP that writes and reads to/from the memory
 */

/* Setup the write channel
 */
Status = Vdma_WriteSetup(InstancePtr);
if (Status != XST_SUCCESS)
{
    xil_printf("Write channel setup failed %d\r\n", Status);
    if(Status == XST_VDMA_MISMATCH_ERROR)
        xil_printf("DMA Mismatch Error\r\n");

    return XST_FAILURE;
}

/* Setup the read channel
 */
Status = Vdma_ReadSetup(InstancePtr);
if (Status != XST_SUCCESS)
{
    xil_printf("Read channel setup failed %d\r\n", Status);
    if(Status == XST_VDMA_MISMATCH_ERROR)
        xil_printf("DMA Mismatch Error\r\n");

    return XST_FAILURE;
}

/*
 * WARNING: In free-run mode, interrupts may overwhelm the system.
 * In that case, it is better to disable interrupts.
 */
XAxiVdma_IntrDisable(InstancePtr, XAXIVDMA_IXR_COMPLETION_MASK,
XAxiVdma_Read);

/* Start the DMA engine to transfer
 */
Status = Vdma_StartTransfer(InstancePtr);
if (Status != XST_SUCCESS)
{
    if(Status == XST_VDMA_MISMATCH_ERROR)
        xil_printf("DMA Mismatch Error\r\n");

    return XST_FAILURE;
}

```

```

    return XST_SUCCESS;
}

/*****
**
* This function sets up the read channel
*
* @param InstancePtr is the instance pointer to the DMA engine.
*
* @return XST_SUCCESS if the setup is successful, XST_FAILURE otherwise.
*
* @note None.
*
*****/
static int Vdma_ReadSetup(XAxiVdma *InstancePtr)
{
    int Index;
    u32 Addr;
    int Status;

    ReadCfg.VertSizeInput = RD_FRAME_VERTICAL_SIZE;
    ReadCfg.HoriSizeInput = RD_FRAME_HORIZONTAL_SIZE;

    ReadCfg.Stride = RD_FRAME_HORIZONTAL_SIZE;
    ReadCfg.FrameDelay = 0; /* Does not test frame delay */

    ReadCfg.EnableCircularBuf = 1;
    ReadCfg.EnableSync = 0; /* No Gen-Lock */

    ReadCfg.PointNum = 0; /* No Gen-Lock */
    ReadCfg.EnableFrameCounter = 0; /* Endless transfers */

    ReadCfg.FixedFrameStoreAddr = 0; /* We are not doing parking */

    Status = XAxiVdma_DmaConfig(InstancePtr, XAXIVDMA_READ, &ReadCfg);
    if (Status != XST_SUCCESS)
    {
        xil_printf("Read channel config failed %d\r\n", Status);
        return XST_FAILURE;
    }

    /* Initialize buffer addresses
    *
    * These addresses are physical addresses
    */
    Addr = RD_BASE_ADDR + RD_FRAME_START_OFFSET;
    for(Index = 0; Index < NUMBER_OF_FRAMES; Index++)
    {
        ReadCfg.FrameStoreStartAddr[Index] = Addr;
        Addr += RD_FRAME_HORIZONTAL_SIZE * RD_FRAME_VERTICAL_SIZE;
    }

    /* Set the buffer addresses for transfer in the DMA engine
    * The buffer addresses are physical addresses
    */
    Status = XAxiVdma_DmaSetBufferAddr(InstancePtr, XAXIVDMA_READ,
    ReadCfg.FrameStoreStartAddr);
    if (Status != XST_SUCCESS)
    {
        xil_printf("Read channel set buffer address failed %d\r\n", Status);
        return XST_FAILURE;
    }
}

```

```

/* Clear data buffer
 */
memset((void *)RD_BASE_ADDR, 0x00, RD_SPACE);

return XST_SUCCESS;
}

/*****
**
*
* This function sets up the write channel
*
* @param InstancePtr is the instance pointer to the DMA engine.
*
* @return XST_SUCCESS if the setup is successful, XST_FAILURE otherwise.
*
* @note None.
*
*****/
static int Vdma_WriteSetup(XAxiVdma * InstancePtr)
{
    int Index;
    u32 Addr;
    int Status;

    WriteCfg.VertSizeInput = WR_FRAME_VERTICAL_SIZE;
    WriteCfg.HoriSizeInput = WR_FRAME_HORIZONTAL_SIZE;

    WriteCfg.Stride = WR_FRAME_HORIZONTAL_SIZE;
    WriteCfg.FrameDelay = 0; /* This example does not test frame delay */

    WriteCfg.EnableCircularBuf = 1;
    WriteCfg.EnableSync = 0; /* No Gen-Lock */

    WriteCfg.PointNum = 0; /* No Gen-Lock */
    WriteCfg.EnableFrameCounter = 0; /* Endless transfers */

    WriteCfg.FixedFrameStoreAddr = 0; /* We are not doing parking */

    Status = XAxiVdma_DmaConfig(InstancePtr, XAXIVDMA_WRITE, &WriteCfg);
    if (Status != XST_SUCCESS)
    {
        xil_printf("Write channel config failed %d\r\n", Status);
        return XST_FAILURE;
    }

    /* Initialize buffer addresses
     *
     * Use physical addresses
     */
    Addr = WR_BASE_ADDR + WR_FRAME_START_OFFSET;
    for(Index = 0; Index < NUMBER_OF_FRAMES; Index++)
    {
        WriteCfg.FrameStoreStartAddr[Index] = Addr;
        Addr += WR_FRAME_HORIZONTAL_SIZE * WR_FRAME_VERTICAL_SIZE;
    }

    /* Set the buffer addresses for transfer in the DMA engine
     */
    Status = XAxiVdma_DmaSetBufferAddr(InstancePtr, XAXIVDMA_WRITE,
WriteCfg.FrameStoreStartAddr);
    if (Status != XST_SUCCESS)

```

```

{
    xil_printf("Write channel set buffer address failed %d\r\n", Status);
    return XST_FAILURE;
}

/* Clear data buffer
 */
memset((void *)WR_BASE_ADDR, 0x00, WR_SPACE);

return XST_SUCCESS;
}

/*****
**
*
* This function starts the DMA transfers. Since the DMA engine is operating
* in circular buffer mode, video frames will be transferred continuously.
*
* @param InstancePtr points to the DMA engine instance
*
* @return XST_SUCCESS if both read and write start succesfully
*         XST_FAILURE if one or both directions cannot be started
*
* @note None.
*
*****/
static int Vdma_StartTransfer(XAxiVdma *InstancePtr)
{
    int Status;

    Status = XAxiVdma_DmaStart(InstancePtr, XAXIVDMA_WRITE);
    if (Status != XST_SUCCESS)
    {
        xil_printf("Start Write transfer failed %d\r\n", Status);
        return XST_FAILURE;
    }

    Status = XAxiVdma_DmaStart(InstancePtr, XAXIVDMA_READ);
    if (Status != XST_SUCCESS)
    {
        xil_printf("Start read transfer failed %d\r\n", Status);
        return XST_FAILURE;
    }

    return XST_SUCCESS;
}

```

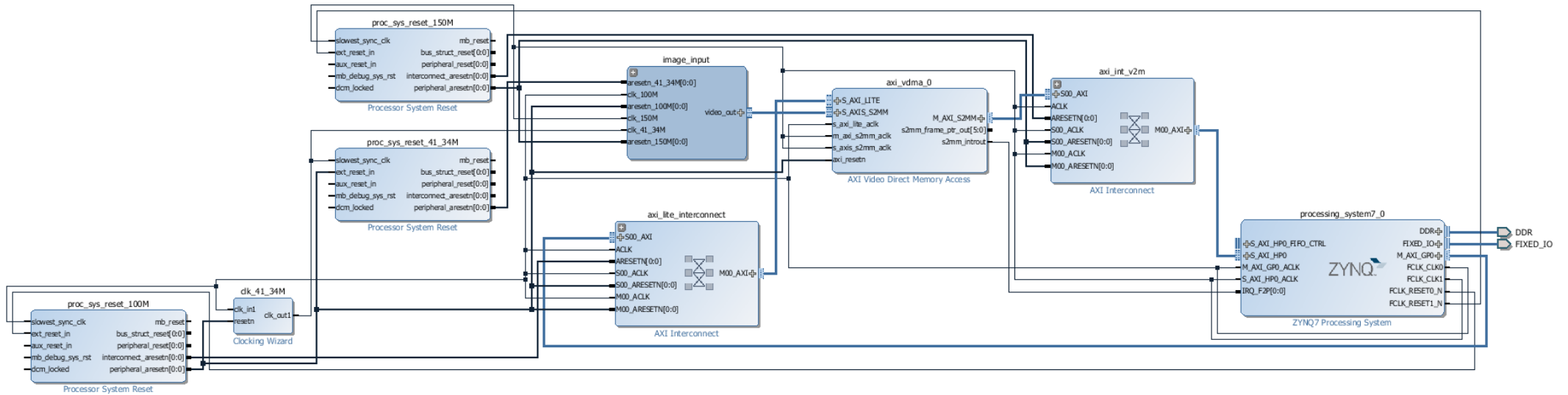




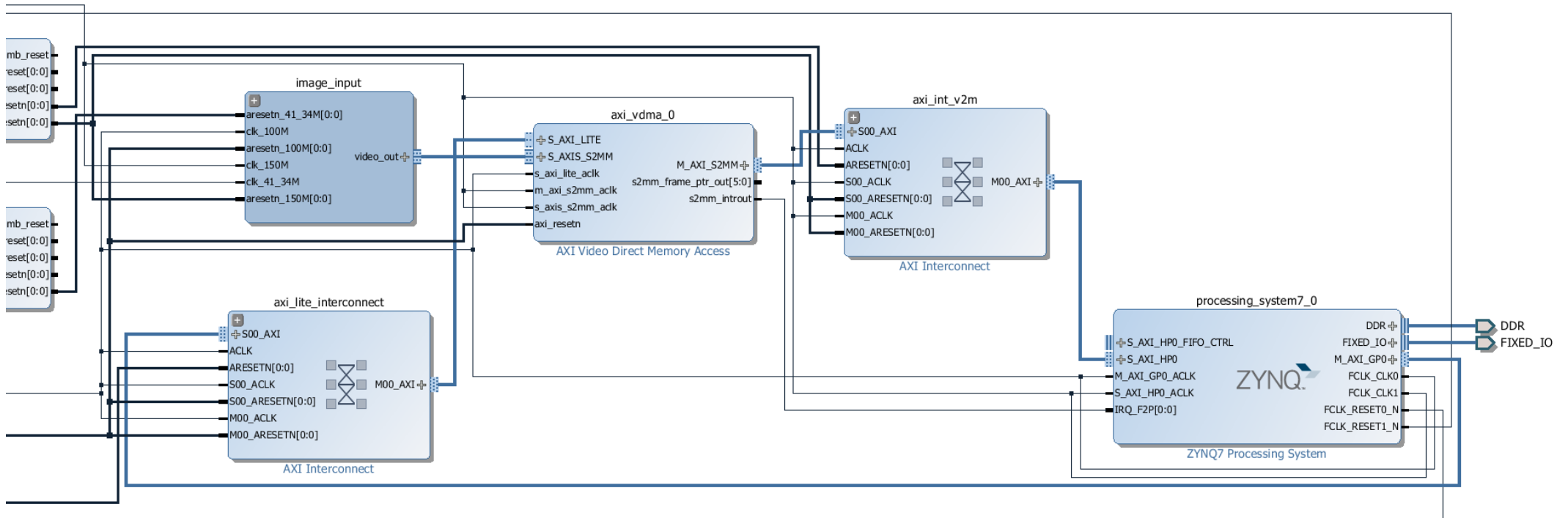


## Anexo B - Imagens de alguns projectos criados

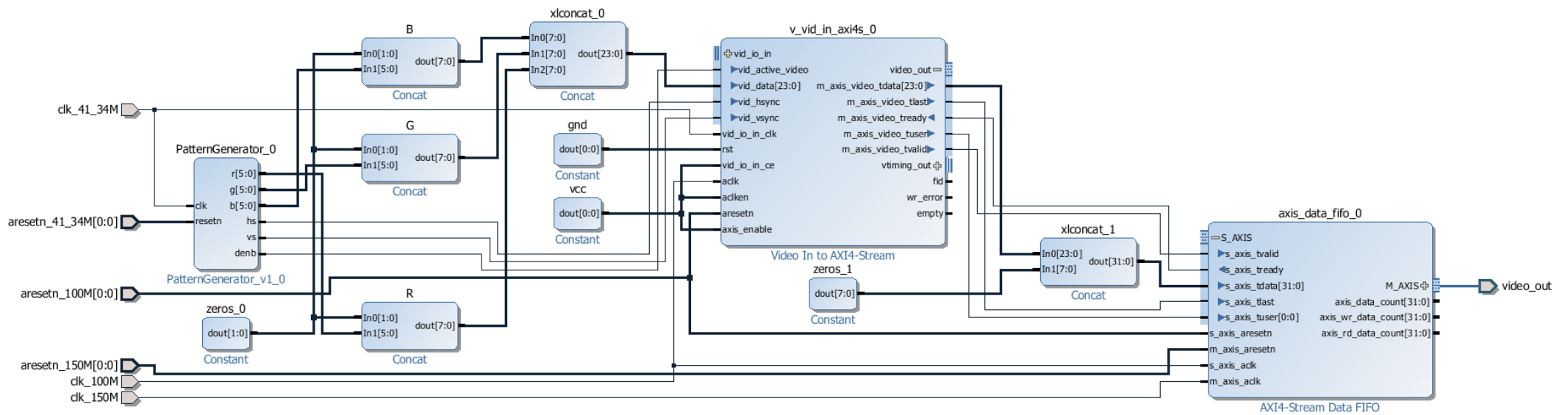
B - 1) Diagrama de blocos - datapath completo (versão minimal)



B - 2) Pormenor do diagrama de blocos - datapath completo (versão minimal)



B - 3) Diagrama de blocos - bloco "image\_input"



B - 4) Relatório sumário do projecto - datapath completo (versão minimal)

Project Summary \_ □ ↻ ×


**Project Settings** Edit ↗

Project name: Datapath  
 Project location: C:/Users/jp-duarte/Desktop/Exemplos/Datapath\_completo  
 Product family: Zynq-7000  
 Project part: [ZedBoard Zynq Evaluation and Development Kit \(xc7z020d4g484-1\)](#)  
 Top module name: [design\\_1\\_wrapper](#)

---

**Board Part** ↗

Display name: ZedBoard Zynq Evaluation and Development Kit  
 Board part name: em.avnet.com:zed:part0:1.3  
 Repository path: C:/Xilinx/Vivado/2015.1/data/boards/board\_files  
 URL: <http://www.zedboard.org>  
 Board overview: ZedBoard Zynq Evaluation and Development Kit



---

**Synthesis** ↗

Status: ✔ Complete  
 Messages: ! [182 warnings](#)  
 Part: xc7z020d4g484-1  
 Strategy: [Vivado Synthesis Defaults](#)

**Implementation** ↗

Status: ✔ Complete  
 Messages: ! [94 warnings](#)  
 Part: xc7z020d4g484-1  
 Strategy: [Vivado Implementation Defaults](#)  
 Incremental compile: [None](#)  
Summary | [Route Status](#)

---

**DRC Violations** ↗

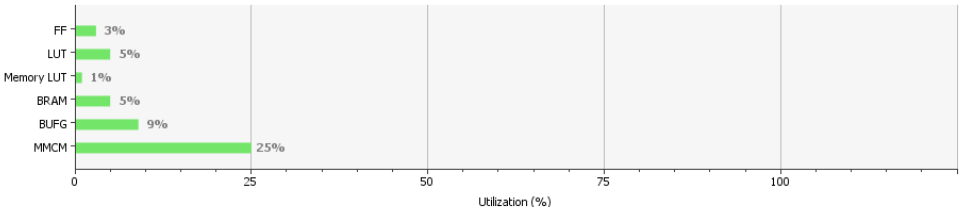
Summary: ! [48 warnings](#)

**Timing** ↗

Worst Negative Slack (WNS): 0.319 ns  
 Total Negative Slack (TNS): 0 ns  
 Number of Failing Endpoints: 0  
 Total Number of Endpoints: 8955  
[Implemented Timing Report](#)  
Setup | [Hold](#) | [Pulse Width](#)

---

**Utilization - Post-Implementation** ↗



Resource	Utilization (%)
FF	3%
LUT	5%
Memory LUT	1%
BRAM	5%
BUFG	9%
MMCM	25%

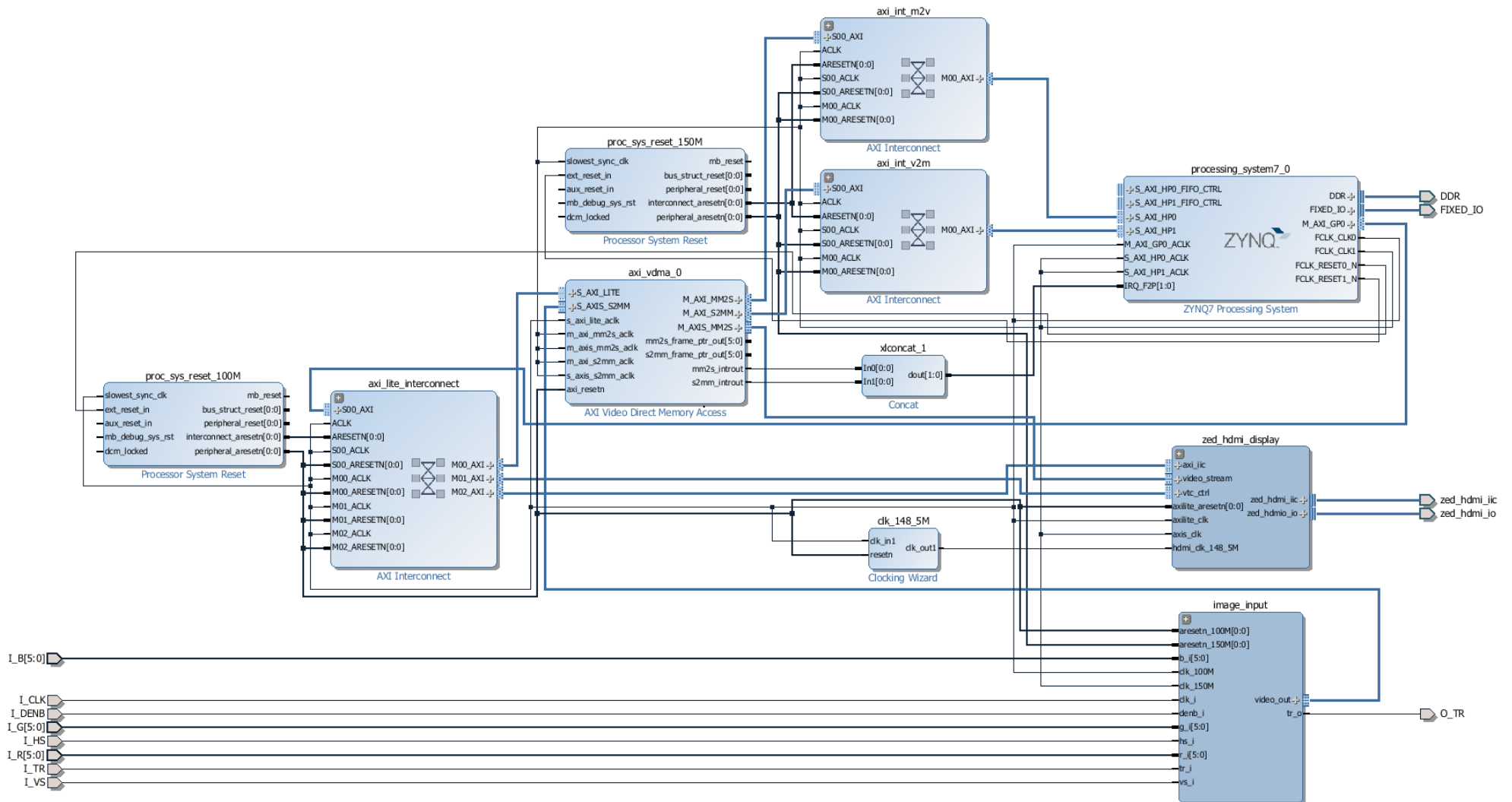
Graph | Table

**Power** ↗

**Total On-Chip Power:** 1.739 W  
**Junction Temperature:** 45,1 °C  
 Thermal Margin: 39,9 °C (3,3 W)  
 Effective  $\theta$ JA: 11,5 °C/W  
 Power supplied to off-chip devices: 0 W  
 Confidence level: [Low](#)  
Summary | [On-Chip](#)

Post-Synthesis | Post-Implementation

B - 5) Diagrama de blocos - datapath completo (versão com entrada de vídeo externa e saída HDMI)



B - 6) Relatório sumário do projecto - datapath completo (versão com entrada de vídeo externa e saída HDMI)

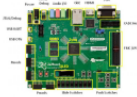
**Project Summary**

**Project Settings**

Project name: TransmitterHDMI  
 Project location: C:/Users/jp-duarte/Desktop/Final/Projects/TransmitterHDMI  
 Product family: Zynq-7000  
 Project part: [ZedBoard Zynq Evaluation and Development Kit \(xc7z020dq484-1\)](#)  
 Top module name: [design\\_1\\_wrapper](#)

**Board Part**

Display name: ZedBoard Zynq Evaluation and Development Kit  
 Board part name: em.avnet.com:zed:part0:1.3  
 Repository path: C:/Xilinx/Vivado/2015.1/data/boards/board\_files  
 URL: <http://www.zedboard.org>  
 Board overview: ZedBoard Zynq Evaluation and Development Kit



**Synthesis**

Status: ✔ Complete  
 Messages: ! 277 warnings  
 Part: xc7z020dq484-1  
 Strategy: [Vivado Synthesis Defaults](#)

**Implementation**

Status: ✔ Complete  
 Messages: ! 3 critical warnings  
! 105 warnings  
 Part: xc7z020dq484-1  
 Strategy: [Vivado Implementation Defaults](#)  
 Incremental compile: None  
**Summary** | Route Status | Failed Nets

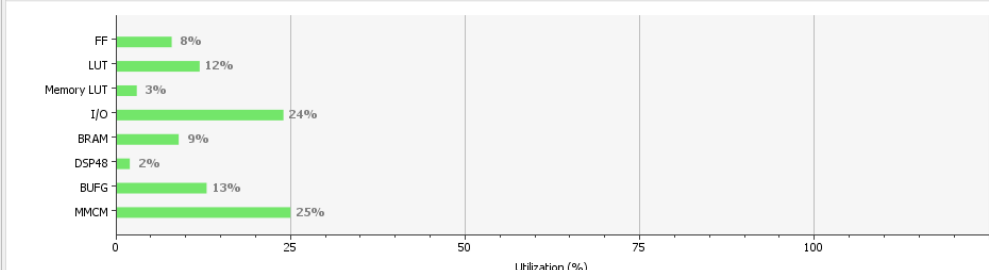
**DRC Violations**

Summary: ! 58 warnings

**Timing**

Worst Negative Slack (WNS): 0.311 ns  
 Total Negative Slack (TNS): 0 ns  
 Number of Falling Endpoints: 0  
 Total Number of Endpoints: 20813  
[Implemented Timing Report](#)  
**Setup** | Hold | Pulse Width

**Utilization - Post-Implementation**



Resource	Utilization (%)
FF	8%
LUT	12%
Memory LUT	3%
I/O	24%
BRAM	9%
DSP48	2%
BUFG	13%
MMCM	25%

**Power**

**Total On-Chip Power:** 1.851 W  
**Junction Temperature:** 46,3 °C  
 Thermal Margin: 38,7 °C (3,2 W)  
 Effective  $\theta_{JA}$ : 11,5 °C/W  
 Power supplied to off-chip devices: 0 W  
 Confidence level: [Low](#)  
**Summary** | On-Chip

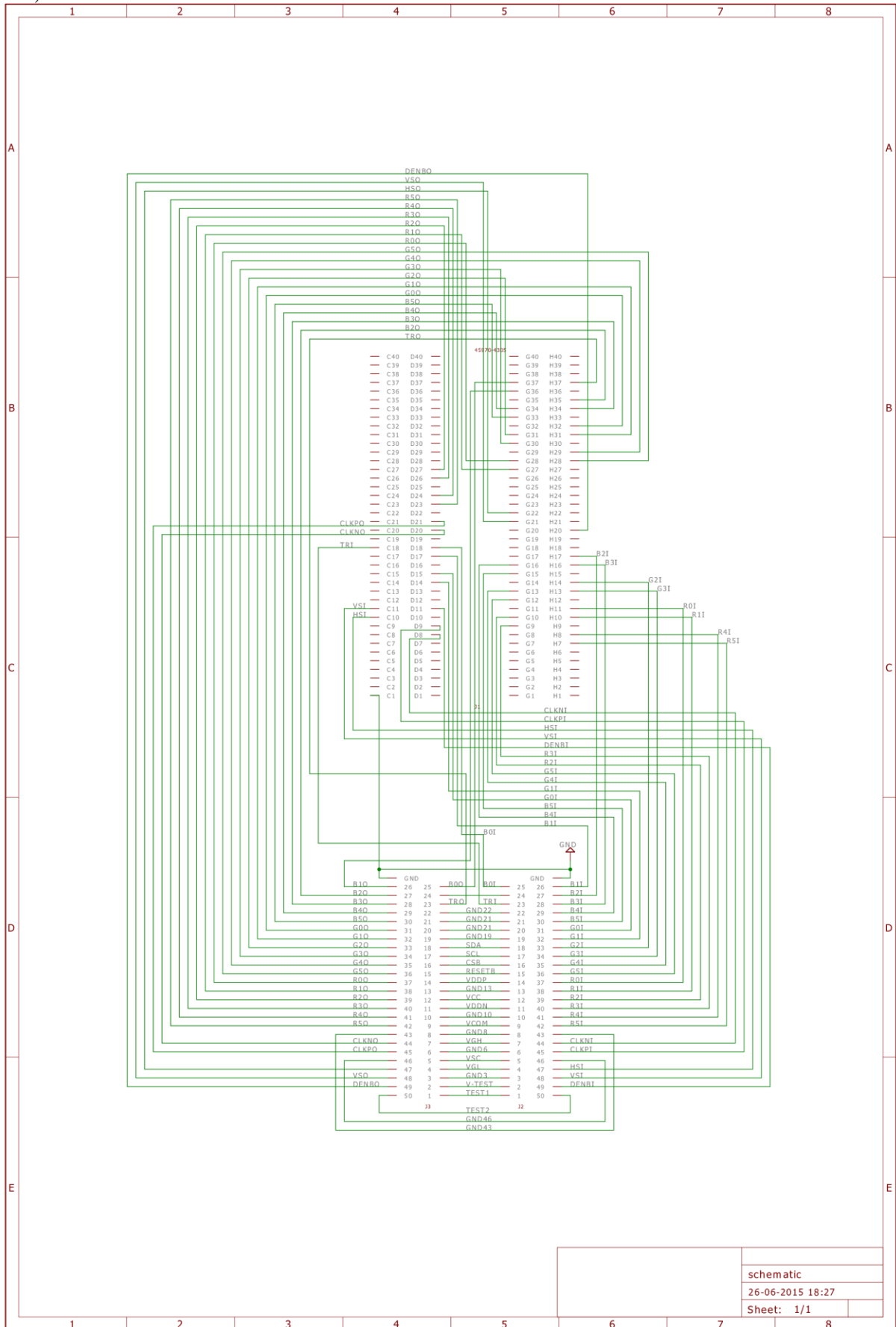






# Anexo C - Desenhos esquemáticos das PCB (alternativas)

## C - 1) PCB - A1



schematic  
26-06-2015 18:27  
Sheet: 1/1



