FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Integração de processador ARC EM para aplicações CBUS MHL 2.2

António Alberto Loureiro da Silva

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador: Professor João Canas Ferreira

Supervisor Externo: Engenheiro Rui Rainho Almeida

13 de julho de 2015

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

MIEEC - MESTRADO INTEGRADO EM ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES | **2014/2015**

A Dissertação intitulada

"Integração de Processador ARC EM para Aplicações CBUS MHL 2.2"

foi aprovada em provas realizadas em 13-07-2015

o júri

Presidente Professor Doutor António José Duarte Araújo
Professor Auxiliar do Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

Professor Doutor Manuel Gradim de Oliveira Gericota
Professor Adjunto do Departamento de Engenharia Eletrotécnica do Instituto Superior de Engenharia do Porto

Professor Doutor João Paulo de Castro Canas Ferreira
Professor Auxiliar do Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

O autor declara que a presente dissertação (ou relatório de projeto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extratos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são corretamente citados.

Autor - António Alberto Loureiro da Silva

Faculdade de Engenharia da Universidade do Porto

Resumo

Atualmente, os dispositivos móveis desempenham um importante papel na forma como o mercado da eletrónica de consumo se movimenta, levando a que as empresas tecnológicas invistam cada vez mais neste ramo. Derivado da evolução dos atuais *smartphones* e *tablets*, começou a haver, necessariamente, uma maior preocupação com a potência consumida por estes dispositivos, sem deixar de ter em conta a sua performance.

O uso destes dispositivos passa, muitas vezes, pela criação, visualização e partilha de conteúdos multimédia. Desta forma, torna-se necessária a interligação com dispositivos de reprodução desses mesmos conteúdos. O HDMI (*High-Definition Multimedia Interface*) é atualmente uma das tecnologias mais utilizadas para partilha de conteúdos multimédia. No entanto, este protocolo não foi inicialmente desenvolvido para aplicações mobile e, como tal, não teve em conta as necessidades de consumo energético inerentes às mesmas. Para responder a esta necessidade de um protocolo melhor adaptado aos dispositivos mobile, foi desenvolvido o MHL (*Mobile High-Definition Link*).

A especificação mais recente de MHL (3.0) permite o transporte de vídeo com resoluções até 4Kx2K @ 30Hz e contém mecanismos auxiliares que permitem o carregamento da bateria do dispositivo com uma capacidade de 900mA @ 5V. Uma das maiores vantagens deste protocolo é a utilização dos conectores já existentes tanto nos dispositivos móveis como nas televisões (USB 2.0 e HDMI respetivamente).

O mecanismo de controlo do protocolo MHL, denominado CBUS (*Control Bus*) divide-se logicamente em duas camadas protocolares : *Link Layer e Translation Layer*. Com o atual core transmissor MHL da Synopsys, a *Translation Layer* é implementada, em software, pelo processador do SoC (System on Chip) onde o core é integrado pelo cliente.

A presente Dissertação demonstra a validade da integração de um processador dedicado, de forma a dotar a solução Synopsys da capacidade de implementar a *Translation Layer* internamente. Desta forma, é possível oferecer uma solução mais completa e robusta, isolando a implementação do protocolo CBUS ao core desenvolvido. É uma solução mais completa porque o core passa a implementar a *Translation Layer* internamente, retirando a carga de processamento respetiva do processador central do sistema. Existe também um importante aumento de robustez devido à implementação e verificação da interface entre as duas camadas do protocolo passarem a ser feitas internamente na Synopsys.

Abstract

Currently, mobile devices play an important role in how the market of consumer electronics moves, leading to that technology companies have been investing more in this field. Due to the evolution of today smartphones and tablets, there is necessarily a greater concern with power consumed by these devices, while taking into account their performance.

The use of mobile devices passes often for creating, viewing and sharing of multimedia content. Thus, it becomes necessary interconnection with playback devices of these same contents. The HDMI (High-Definition Multimedia Interface) is currently one of the most widely used technology for sharing of multimedia content. However, this protocol was not initilly designed for mobile applications and as such, did not take into account the power requirements inherent to them. To answer this need for better protocol adapted to mobile devices has been developed the MHL (*Mobile High-Definition Link*).

The latest MHL specification (3.0) enables video transport with resolutions up to 4Kx2K @ 30Hz and contains auxiliary mechanisms allow charging the device's battery with a capacity of 900mA @ 5V. One major advantage of this protocol is the use of connectors already existing both on mobile devices and in televisions (USB 2.0 and HDMI respectively).

The control mechanism of MHL protocol, called CBUS (Control Bus) is logically divided into two protocol layers: Link Layer and Translation Layer. With the current MHL trasmitter core of Synopsys, Translation Layer is implemented, in software, by the processor of the SoC (System on Chip) where the core is integrated by the customer.

This Dissertation demonstrates the validity of integrating a dedicated processor, in order to provide the Synopsys solution the ability to implement the Translation Layer internally. Thus, it is possible to provide a more complete and robust solution, isolating the implementation of CBUS protocol to the developed core. It is a more complete solution because the core begins to implement the Translation Layer internally removing the processing load respective central system processor. It is also a more robust solution due to Implementation and Verification of the interface between the two layers of the protocol to be passed made internally at Synopsys.

Agradecimentos

Ao meu orientador pelo tempo e apoio durante a dissertação. Obrigado Professor João Canas Ferreira pela disponibilidade e pelo entusiasmo contagiante que tem por aquilo que faz.

À equipa de HDMI da Synopsys que me recebeu e ajudou no desenvolvimento desta dissertação. Obrigado Rui, Vítor Antunes e companhia pela disponibilidade, pelas críticas e por me oferecerem a oportunidade de fazer parte da Synopsys.

Aos meus pais, por me terem ensinado a escolher os melhores caminhos, por confiarem em mim e por me terem dado sempre tudo. Obrigado Mãe pelo espírito lutador e obrigado Pai pelo sentido crítico.

À minha irmã por me obrigar a estudar para quando for engenheiro lhe dar grandes prendas. Obrigado Ana pelo orgulho que tens em mim.

À minha namorada pelo apoio incondicional e pela motivação que me deu quando esta diminuía. Obrigado Cláudia pelo carinho, pelas críticas e, sobretudo, por me fazeres sonhar.

À minha tia Cristina, por todo apoio durante o meu percurso académico. Obrigado Tia, pela vontade de descobrir novas coisas.

Aos meus avós por se orgulharem de mim e me passarem conhecimentos fundamentais. Obrigado por ter tido a oportunidade de vos conhecer a todos.

Aos meus colegas de Faculdade por termos partilhados piadas nerds, sushi, noites de estudo, resistências e condensadores. Obrigado malta pelo companheirismo e por terem feito de mim um melhor jogador de matraquilhos.

António Silva

"Why is it that when one man builds a wall,
the next man immediately needs to know what's on the other side?"

George R. R. Martin, A Game of Thrones

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

AHB	Advanced High-performance Bus
AMBA	Advanced Microcontroller Bus Architecture
APB	Advanced Peripheral Bus
ASIC	Application-Specific Integrated Circuit
ATPG	Automatic Test Pattern Generation
BVCI	Basic Virtual Component Interface
CBUS	MHL Link Control BUS
CEC	Consumer Electronic Control
DCCM	Data Close Couple Memory
DDC	Display Data Channel
DW	DesignWare
EDA	Electronic Design Automation
EDID	Extended Display Identification Data
FPGA	Field Programmable Gate Array
HDCP	High-bandwidth Digital Content Protection
HDMI	High-Definition Multimedia Interface
HW	Hardware
ICCM	Instruction Close Couple Memory
IP	Intelectual Property
ISA	Instruction Set Architecture
JTAG	Joint Test Action Group
MHL	Mobile High-Definition Link
RISC	Reduced Instruction Set Computing
RTL	Register Transfer Level
RX	Receiver
STA	Static Timing Analysis
SW	Software
SoC	System-on-Chip
TX	Transmitter
USB	Universal Serial Bus

Capítulo 1

Introdução

O presente documento foi elaborado no contexto da Unidade Curricular "Dissertação" do Mestrado Integrado em Engenharia Eletrotécnica e de Computadores. O seu objetivo é relatar o trabalho desenvolvido ao longo da Unidade Curricular, demonstrando o estudo que o mesmo implicou, as metodologias que foram seguidas, os resultados obtidos e as conclusões retiradas.

A dissertação a que este documento se refere tem como título "Integração de processador ARC EM para aplicações CBUS MHL 2.2" e foi proposta pela Synopsys, uma empresa externa à Faculdade de Engenharia da Universidade do Porto, líder no mercado de EDA (*Electronic Design Automation*) e que possui também um amplo portfólio de soluções IP (*Intelectual Property*).

1.1 Contexto

Tal como referido acima, a presente dissertação surge da proposta efetuada pela Synopsys, neste caso pela Synopsys Porto, que se encontra instalada no polo empresarial TecMaia, situado na Maia, Porto.

A Synopsys Porto é responsável pelo desenvolvimento de soluções ao nível de interfaces como HDMI, MHL ou MIPI. Esta dissertação enquadra-se, como o seu título indica, na solução MHL, e tem como objetivo a integração de um processador dedicado a fim de tornar mais completa e robusta a solução já existente. O processador a integrar pertence à família DesignWare ARC EM, propriedade da Synopsys.

Atualmente, a tecnologia HDMI (*High-Definition Multimedia Interface*) [1] é a interface multimédia com maior adoção pelo mercado para oferecer a partilha de conteúdos multimédia e de aplicações de entretenimento. A última especificação HDMI (2.0) contempla o transporte de resoluções de vídeo que podem chegar a 4Kx2K @ 60Hz (Ultra HD) e de áudio com frequências de amostragem até 1536KHz.

Embora a interface HDMI seja utilizada em aplicações mobile, esta não foi desenvolvida originalmente para os atuais dispositivos móveis, como *smartphones* e *tablets*, e, por esse motivo, não teve em conta as limitações de baixo consumo energético que os mesmos possuem, derivado de serem alimentados geralmente por baterias.

Por forma a dar reposta à necessidade de um protocolo de interface multimédia melhor adaptado às exigências referidas no parágrafo anterior, surgiu o protocolo MHL (*Mobile High-Definition Link*) [2]. A sua mais recente especificação (3.0) permite o transporte de vídeo com resoluções até 4Kx2K @ 30Hz e contém mecanismos auxiliares que permitem o carregamento da bateria do dispositivo com capacidade de 900mA @ 5V. Uma das maiores vantagens deste protocolo é a reutilização dos conectores já existentes, tanto nos dispositivos móveis como nas televisões (USB 2.0 e HDMI, respetivamente).

Atualmente, as televisões têm embutidos sistemas que disponibilizam a interface MHL, para além da HDMI, tal como o SoC proposto em [3]. Uma das diferenças entre os protocolos HDMI e MHL está no seu mecanismo de controlo. O protocolo HDMI utiliza a tecnologia Consumer Electronic Control (CEC), enquanto que a interface MHL especifica um mecanismo de controlo próprio - Control Bus (CBUS).

1.2 Objetivos

O objetivo da dissertação, tal como já foi referido, é proceder à integração de um processador ARC da família DesignWare ARC EM na solução, já existente, DesignWare MHL TX Interface IP da Synopsys. A implementação foi feita na linguagem de descrição de hardware (*Hardware Description Language* - HDL) Verilog e é sintetizável nas tecnologias de 40nm e 28nm. O propósito desta integração é dotar a solução DesignWare MHL TX da capacidade de implementar a *Translation Layer* do protocolo MHL com uma solução de hardware/software (HW/SW) robusta, escalável e flexível.

O protocolo CBUS, referido na secção anterior como mecanismo de controlo adotado pela tecnologia MHL, permite que os dispositivos troquem entre si informação relativa às suas capacidades operacionais, configuração, estado operacional e encriptação. A especificação do protocolo contempla a sua divisão em camadas distintas, tal como é hábito em outros protocolos de comunicação, proporcionando diferentes níveis de abstração.

Os dois níveis em que o protocolo CBUS se encontra dividido são denominados de *Link Layer*, camada de mais baixo nível, responsável pelo acesso ao meio físico e que trata da gestão dos sinais digitais, e *Translation Layer*, camada de nível imediatamente superior à anterior e que gere a forma como os bits são interpretados em comandos próprios do protocolo.

O propósito do trabalho é dotar a solução DesignWare MHL TX Interface IP da Synopsys da capacidade de implementar a *Translation Layer* do protocolo CBUS através de um processador ARC , tarefa essa que atualmente está ao encargo do processador central do sistema, que gere as tarefas dos vários blocos que constituem o SoC.

Os desafios envolvidos no trabalho encontram-se essencialmente no *design* da interface entre o processador integrado e o atual core transmissor MHL, por forma a que a solução proposta possa ser integrada da forma menos evasiva possível, isto é, permitindo a sua inserção/remoção facilmente, de forma a poderem ser oferecidas ambas as opções (com ou sem implementação interna da *Translation Layer*). Dentro da solução, existem, de modo geral, duas interfaces com

o processador ARC. A primeira está relacionada com o seu controlo por parte do processador principal do sistema, bem como a forma como é feita a troca de informação entre eles. A segunda prende-se com o acesso aos periféricos CBUS, através de uma interface que permita a troca de informação entre *Translation Layer* e *Link Layer*. É expectável que a solução desenvolvida permita a sua integração em outros ambientes, para além do protocolo CBUS, desde que as interfaces se mantenham.

1.3 Contribuições do Trabalho

A solução desenvolvida permite à Synopsys oferecer uma solução de transmissor MHL com a *Translation Layer* do protocolo CBUS implementada internamente, através da integração do processador ARC EM4.

Foi validada a correta funcionalidade do bloco desenvolvido que inclui, para além do processador integrado, blocos de interface com o processador principal do SoC e com os periféricos CBUS, a fim de transmitir os comandos da *Translation Layer* até à *Link Layer*, responsável pelo acesso ao meio físico.

O módulo implementado não altera a estrutura da solução já existente, facilitando a sua integração e remoção, o que permite disponibilizar as duas versões do transmissor MHL, com possibilidade de *upgrade* da solução anterior.

A reutilização do bloco desenvolvido para outros propósitos que não o protocolo CBUS MHL é possível, desde que as interfaces para o processador principal do SoC e para os periféricos se mantenham. Ainda assim, se as mesmas tiverem que ser alteradas ou adaptadas, a solução implementada serve como base para novos projetos.

Por fim, foi validada a síntese do bloco para ASIC e para FPGA, atingindo-se com sucesso as frequências de operação utilizadas pelo core transmissor MHL da Synopsys. Desta forma, é possível garantir que, com a integração do módulo desenvolvido, a solução atual não terá de sofrer alterações ao nível dos seus requisitos temporais.

1.4 Estrutura do documento

Para além da introdução, este documento contém mais cinco capítulos, apresentando a seguinte estrutura:

Capítulo 2

Aqui é feito um enquadramento técnico, apresentando as tecnologias utilizadas ao longo do projeto e explicando onde foram utilizadas.

Capítulo 3

Neste capítulo é apresentada a solução desenvolvida durante a dissertação, através da sua especificação funcional.

Capítulo 4

Ao logo deste capítulo é documentado o fluxo de projeto que foi seguido durante a elaboração da dissertação, apresentando as suas etapas e as ferramentas utilizadas em cada uma delas.

Capítulo 5

Aqui encontram-se os resultados que foram obtidos com o projeto, tanto na sua simulação e verificação funcional, como na síntese para ASIC e FPGA.

Capítulo 6

Por fim, são documentadas as conclusões retiradas no final da elaboração do projeto, apresentando o resultado final obtido e as suas contribuições. Neste capítulo é também previsto o trabalho futuro a realizar após a presente dissertação.

Capítulo 2

Enquadramento Técnico

Neste capítulo é feito um enquadramento técnico sobre o projeto desenvolvido, através da apresentação das tecnologias envolvidas no mesmo, explicando as suas características e funcionamento.

Nas secções seguintes é introduzido o protocolo CBUS da especificação MHL, com o intuito de esclarecer onde se enquadra o processador integrado, é apresentado o processador ARC EM4, que foi inserido na solução e, por fim, são expostos os protocolos AMBA APB e AHB-lite, utilizados pela solução MHL e pelo processador ARC para implementarem as respetivas interfaces externas.

2.1 O protocolo Control Bus (CBUS)

O CBUS é um protocolo de comunicação ponto-a-ponto bidirecional documentado na especificação MHL, pela entidade "*MHL Promoters*", especificado com o intuito de oferecer um mecanismo de controlo para o protocolo MHL. Este permite a troca de comandos entre o aparelho transmissor e o recetor, sendo por isso um protocolo de comunicação *inter-board* com uma linha dedicada na interface MHL, operando sobre o cabo MHL que interliga os dois aparelhos.

Devido à necessidade de mapear o protocolo MHL em conectores USB, em concreto conectores micro USB, existe uma preocupação inerente com a gestão das linhas disponíveis. Desta forma, o CBUS apresenta apenas uma única linha de comunicação (*one wire*) assíncrona.

Este mecanismo de controlo providencia as seguintes funcionalidades:

- Um mecanismo para descoberta de conectividade entre fonte e recetor de sinal que indica se ambos os dispositivos oferecem uma interface MHL;

- Um canal para a comunicação de comandos DDC (*Display Data Channel*), utilizado para o dispositivo fonte determinar quais são as capacidades e características do dispositivo recetor, lendo a estrutura de dados EDID (*Extended Display Identification Data*);

- Um canal para comunicação de comandos DDC, utilizado pelo dispositivo fonte para iniciar todas as escritas e leituras de registos necessárias para proteção de conteúdo HDCP (*High-Bandwidth Digital Content Protection*);

- Um canal denominado de *MHL Sideband Channel* (MSC) que oferece serviços de mais alto nível, como tarefas a executar automaticamente ou tarefas relacionadas, geralmente, com a utilização do controlo remoto por infravermelhos (por exemplo, o comando da televisão).

Tal como já foi referido no capítulo anterior, o protocolo CBUS é composto por duas camadas: *Link Layer* e *Translation Layer*. A camada *Link Layer*, de mais baixo nível, descreve o método e o protocolo a seguir para enviar dados através de uma interface de um único fio. A camada *Translation Layer*, construída em cima da anterior, descreve a forma como os comandos (DDC, por exemplo) devem ser mapeados na interface.

O camada de *Link Layer* do protocolo utiliza codificação bifásica (*bi-phase coding*) para enviar os bits. O protocolo é orientado à transmissão de byte, isto é, a unidade básica de dados na comunicação é o byte, sendo que cada byte contém uma estrutura própria.

Após o arranque do sistema, a lógica de controlo entra na fase de descoberta "*discovery phase*" por forma a detetar a existência de conectividade e as funcionalidades suportadas. Uma vez que a conexão seja estabelecida, tanto o dispositivo recetor como o transmissor podem iniciar o processo de arbitragem da linha. O aparelho que ganhe a arbitragem é considerado o "*initiator*", enquanto que o outro é denominado de "*follower*".

Neste capítulo serão usados os termos *requester* e *responder* para designar, respetivamente, quem inicia a comunicação através do envio de um ou mais pacotes e quem responde a essa solicitação, respondendo aos comandos recebidos. Não confundir estes termos com os termos *initiator* e *follower*. Quando um pacote é enviado do *requester* para o *responder*, o *requester* é o *initiator* e o *responder* o *follower*. Quando um pacote é enviado do *responder* para o *requester* (para completar um comando), o *responder* é o *initiator* desse pacote e o *requester* é o *follower*.

2.1.1 *Link Layer*

2.1.1.1 Codificação bifásica

Os dados a comunicar na linha CBUS devem ser codificados de acordo com a codificação bifásica, segundo a especificação MHL. A Figura 2.1 demonstra o diagrama temporal desta codificação.

Neste tipo de codificação, cada símbolo é logicamente divido em dois estados, o primeiro estado ocorre em metade da duração do símbolo e o segundo estado na segunda metade da duração total do símbolo. Tal como se pode observar na Figura 2.1, o primeiro estado de um símbolo (neste caso um bit) é sempre diferente do segundo estado do símbolo anterior. O segundo estado de um símbolo é idêntico ao primeiro estado do mesmo símbolo, caso o que se pretenda transmitir seja o bit "0". Pelo contrário, caso o bit a transmitir seja um "1", então o segundo estado terá que ser diferente do primeiro, havendo assim uma transição a meio da duração do símbolo.

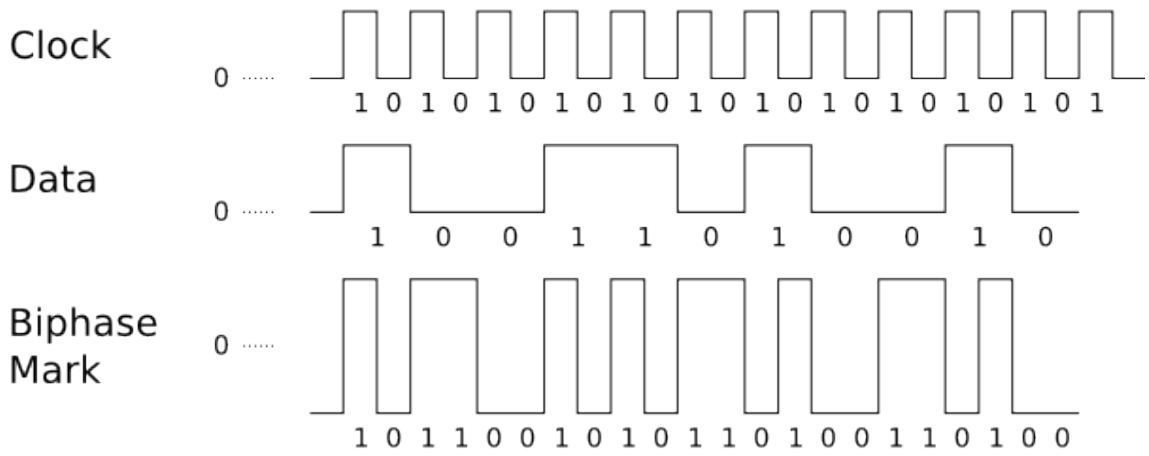


Figura 2.1: Diagrama temporal com um exemplo de codificação bifásica

2.1.1.2 Estrutura dos pacotes

Ao nível da *Link Layer*, o começo de cada pacote responsável pela transmissão de um byte de dados é sinalizado por 1 pulso de *sync* (com duração de 2 bits), seguido de um *header* de 2 bits, 1 bit de controlo, 8 bits de dados, 1 bit de paridade e um pulso de *acknowledge* (duração de 2 bits). No total, por cada byte de informação, é necessário o tempo de 16 bits. A Figura 2.2 demonstra graficamente o formato de uma trama CBUS.

2.1.2 Requisitos temporais

O protocolo CBUS, visto ser uma comunicação assíncrona que utiliza apenas uma linha de comunicação, necessita que sejam especificados os seus requisitos temporais, por forma a que haja um acordo em relação, por exemplo às frequências de amostragem, entre os dois dispositivos que forem interligados via cabo MHL.

As especificações temporais do protocolo CBUS dependem da versão MHL a que está associado, sendo que para o caso do CBUS MHL 2.2, que é o utilizado neste projeto, o *bit rate* especificado para o protocolo CBUS é de 1 MHz. Outras características temporais que devem ser cumpridas podem ser consultadas na especificação MHL 2.2 [2].

2.1.3 *Translation Layer*

A camada *Translation Layer* implementa uma interface com a camada *Link Layer* e atua como interpretadora dos comandos MSC e DDC recebidos, provenientes dos respetivos canais virtuais, respondendo a estes de acordo com o protocolo CBUS.

A *Translation Layer* é também responsável por fazer a gestão do acesso ao canal CBUS. A *Translation Layer* deve arbitrar entre as possíveis solicitações da linha MSC ou DDC, dando acesso à *Link Layer*. A arbitragem do acesso pode ser feita de pacote em pacote, sendo que cada

SYNC (2 bit times)	Header (2 bits)	Control (1 bit)	Data or Command (8 bits)	Parity (1 bit)	ACK (2 bit times)

Figura 2.2: Formato de trama do protocolo CBUS

byte da *Translation Layer*, proveniente de um dos canais virtuais, é mapeado num pacote da *Link Layer* com a duração de 16 tempos de bit, em que os 2 bits do *header* identificam se é um comando MSC ou DDC, permitindo que sejam entrelaçados pacotes de diferentes canais virtuais.

O controlo do fluxo ao longo da linha CBUS é feito recorrendo a pacotes de ACK, NACK e ABORT, existindo *timeouts*, que definem a validade da troca de comandos. O pacote de ACK é gerado pelo *responder* em resposta a um pedido feito previamente pelo *requester* notificando-o que o comando foi completo. O pacote de NACK tem o propósito de notificar o *requester* que o comando enviado não pode ser concluído. O pacote ABORT simboliza que o comando enviado não pode ser concluído devido à ocorrência de um erro. Em relação a este último, um pacote ABORT pode também ser enviado do *requester* para o *responder*, notificando o segundo que a sua resposta ao pedido feito não é coerente.

O canal virtual DDC é responsável pela leitura da estrutura de dados EDID, que contém as capacidades e características do dispositivo recetor, e também pela troca de comandos relacionados com a autenticação HDCP (*High-bandwidth Digital Content Protection*). Nesta comunicação, o dispositivo transmissor é o *master* da comunicação que inicia todas as trocas de pacotes. Tanto na leitura da estrutura de dados EDID como na troca de comandos relacionados com a autenticação HDCP, o dispositivo transmissor não necessita de saber a localização física dos respetivos registos do recetor, a *Translation Layer* do dispositivo recetor é responsável por interpretar os pedidos recebidos e responder com a informação pedida.

O canal virtual MSC transporta comandos de mais alto nível relacionados com o controlo do dispositivo, muitas vezes associados a comandos originários do controlo remoto. Para além da leitura e escrita do estado do dispositivo, o canal virtual MSC possui ainda os seguintes sub-comandos:

- **RAP (*Request Action Protocol*)** – O protocolo RAP permite que se requira uma ação ao dispositivo recetor, por exemplo, habilitando ou não a troca de conteúdo (vídeo e som);

- **RCP (*Remote Control Protocol*)** – O conjunto de comandos desta categoria permite a troca de mensagens despoletadas pelo controlo remoto;

- **UCP (*UTF-8 Character Protocol*)** – As mensagens deste protocolo permitem a troca de caracteres codificados em UTF-8, permitindo desta forma a troca de informação acerca de teclas premidas num teclado, por exemplo.

Faz parte do protocolo CBUS um espaço de endereçamento de registos de cada dispositivo. São esses registos que são lidos e/ou escritos durante a troca de comandos entre os dois dispositivos e que despoletam a sua alteração de estado. O espaço de endereçamento pode ser divido nas seguintes quatro categorias:

- ***Device Capability registers*** – Neste conjunto de registos são guardadas informações acerca do estado do dispositivo, bem como o conjunto de ligações de vídeo e áudio suportadas e a própria identificação do dispositivo (*Device ID*);
- ***Device Interrupt registers*** – Os registos desta categoria incluem informação acerca de interrupções causadas pela alteração do estado de outros registos (de outra categoria), ou pela alteração do estado do dispositivo em si (*Register Change Interrupt* e *Device State Change Interrupt*, respetivamente).
- ***Device Status registers*** – Aqui são guardadas informações acerca do estado da conexão do dispositivo e do estado da ligação MHL.
- ***Scratchpad registers*** – Estes registos são de carácter genérico e neste momento a única funcionalidade associada é a visualização de conteúdos 3D.

2.2 Processador ARC® EM

DesignWare ARC EM é uma família de processadores, propriedade da Synopsys, desenvolvida para aplicações em sistemas embarcados onde o consumo energético e a área ocupada são critérios a ter em conta. Os processadores ARC são configuráveis e extensíveis, permitindo diversas formas de implementação com o objetivo de otimizar a combinação de performance, área e consumo energético de acordo com a aplicação desejada.

O processador a utilizar neste trabalho, que será integrado na solução DesignWare MHL TX Interface IP será o processador ARC EM4 [4], que apresenta as seguintes características:

- Baixa área ocupada – menos que 10K portas lógicas;
- Performance de 1.77 DMIPS/MHz;
- ARCv2 32-bit RISC ISA;
- Big ou Little Endian;
- Até 240 interrupções, com 16 níveis;
- 512B–2MB de memória de instruções – *Instruction Closely Coupled Memory* (ICCM);
- 512–2MB de memória de dados – *Data Closely Coupled Memory* (DCCM);
- Interfaces ARM® AMBA® AHB™, AHB-lite™ e BVCI;

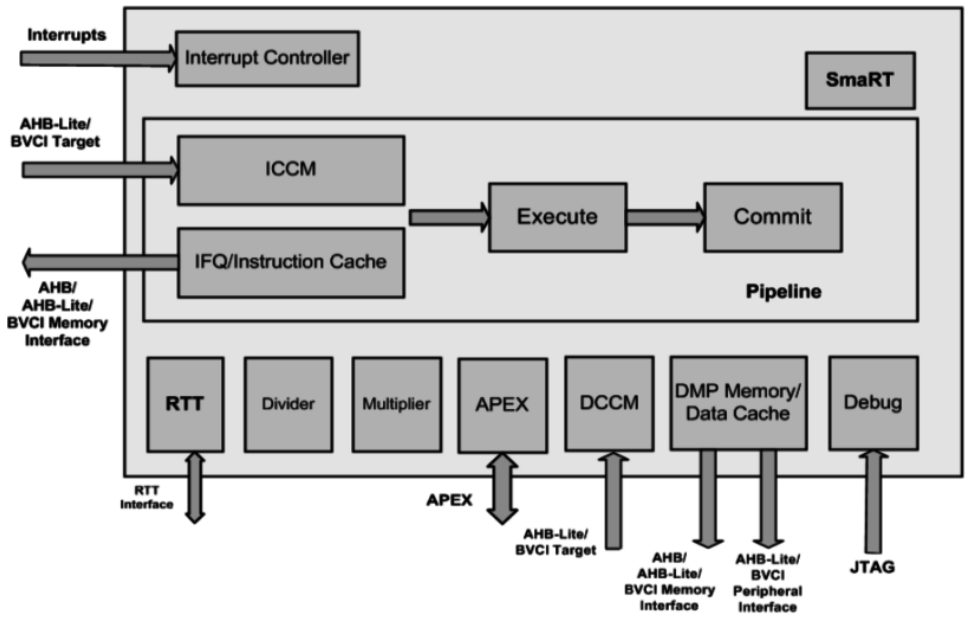


Figura 2.3: Diagrama de blocos do processador ARC EM4 e suas interfaces. Fonte: ARC EM Databook [4]

- Multiplicadores 32x32 ou 16x16, ou ambos;
- Suporte para possíveis extensões a englobar no sistema;
- Depuração por JTAG, com sistema de *Actionpoints*, permitindo a monotorização do processador em tempo real.

A Figura 2.3 apresenta as seguintes interfaces oferecidas pelo processador ARC EM4, que têm as seguintes funcionalidades:

- A interface "*Interrupts*" refere-se às linhas de interrupções externas que o processador ARC EM4 oferece. O processador pode ser configurado para ter até 240 interrupções externas;
- A interface "AHB-Lite/BVCI Target" que se liga diretamente, tanto ao bloco "ICCM" como ao bloco "DCCM", representa a interface de acesso direto à memória de instruções ICCM e à memória de dados DCCM. O acesso por DMI (*Direct Memory Interface*) às memórias pode ser configurado, durante a configuração do processador, para ser através do protocolo AMBA AHB-Lite ou BVCI;
- A interface "AHB/AHB-Lite/BVCI Memory Interface" permite o acesso aos blocos IFQ (*Intruction Fetch Queue*) ou *Instruction Cache*, conforme a configuração do processador. O bloco IFQ permite o acesso direto à pilha de instruções a serem executadas e o bloco *Intruction Cache* implementa uma *cache* da memória de instruções, oferecendo também uma interface externa que pode ser configurada para implementar o protocolo AMBA AHB, AHB-Lite ou BVCI;

- A interface "RTT Interface" permite fazer a monitorização do estado interno do processador em tempo real (*Real Time Trace*) através da ligação do processador a um módulo externo específico, pertencente à família de cores externos *ARC RTT IP Library*;
- A interface "APEX" permite extender as funcionalidades base do processador (*ARC Processor Extensions*) com a adição de blocos dedicados a executar determinadas tarefas que se preveem serem executadas várias vezes, aumentando assim o desempenho do sistema global. Podem ser adicionados, por exemplo, blocos em hardware, responsáveis pela execução de um cálculo de forma eficiente, que podem ser mapeados em instruções próprias, adicionais às da ISA (*Instruction Set Arquitecture*) do processador;
- A interface "AHB/AHB-Lite/BVCI Memory Interface" que liga ao bloco "DMP Memory/Data Cache" permite o acesso direto ao bloco "Data Cache" que, tal como o seu nome indica, representa uma *cache* da memória de dados;
- A interface "AHB-Lite/BVCI Peripheral Interface" que liga ao bloco "DMP Memory/Data Cache" representa o acesso aos periféricos através do seu mapeamento direto em memória (*Directly Memory-mapped Peripherals*). Esta interface pode ser configurada para operar segundo o protocolo AHB-Lite ou BVCI;
- A interface JTAG permite o acesso ao processador para efeitos de *debug* segundo o protocolo JTAG.

2.3 Interfaces AMBA APB e AHB-Lite

Tal como se pôde verificar na secção anterior, o processador ARC EM oferece várias interfaces externas que podem operar segundo o protocolo AMBA AHB-lite , em particular a interface para acesso direto à memória (DMI - *Direct Memory Interface*) e a interface de acesso aos periféricos.

A interface DMI permite o carregamento inicial da memória ICCM com as instruções do programa a correr, bem como o acesso à memória DCCM, que oferece um espaço para troca de mensagens entre o processador principal do SoC e o processador integrado ARC EM.

A interface de acesso a periféricos, diretamente mapeados em memória do processador ARC EM, permite o acesso aos registos da *Link Layer*, servindo assim de interface entre *Translation Layer* e *Link Layer*.

O core transmissor MHL da Synospsys, por sua vez, utiliza na sua interface externa com o processador principal do sistema o protocolo AMBA APB , visto que esta é a interface mais utilizada para cores de periféricos, como é o caso do controlador MHL de transmissão.

A família AMBA, propriedade da ARM, a que pertencem o AHB-lite e o APB , é composta por protocolos de transmissão de dados "*on-chip*", isto é, permitem a troca de informação entre registos, de forma síncrona, através de protocolos especificados e de acesso livre, sendo por isso amplamente utilizados na indústria de EDA (*Electronic Design Automation*). A versão utilizada como referência neste projeto, tanto para o protocolo AHB-lite como para o APB , foi o AMBA 3.

2.3.1 Interface AMBA AHB-lite

A interface AHB-lite foi criada com o objetivo de endereçar soluções com requisitos de alto desempenho. O protocolo suporta a existência de um único *master* e oferece operações com uma maior largura de banda, em comparação, por exemplo, com o protocolo APB .

Segundo a especificação do protocolo AMBA AHB-lite [5], para oferecer as características requeridas por um sistema de elevado desempenho, com frequências de relógio superiores às do protocolo APB , a interface AHB-lite apresenta as seguintes características:

- Transferência em modo *burst*, através da operação em *pipeline*;
- Operações síncronas com um único flanco do relógio, o flanco ascendente;
- Não utilização de interfaces que possam estar em modo "*tristate*";
- Configuração da largura do barramento de dados para 32, 64, 128, 256, 512 ou 1024 bits.

Os *slaves* mais comuns que utilizam a interface AHB-lite são dispositivos internos de memória, interfaces externas de memória e periféricos com requisitos de elevada largura de banda. Apesar de dispositivos que operam com baixa largura de banda poderem ser endereçados via interface AHB-lite , tipicamente estes residem num barramento do tipo APB . Fazer a passagem do protocolo AHB-lite para APB é possível através de um *slave* AHB-lite que faça essa tradução.

2.3.2 Interface AMBA APB

A interface APB oferece um protocolo síncrono otimizado para o consumo de potência e de reduzida complexidade no seu controlo. É indicada para periféricos que não necessitam de uma performance muito elevada, pois esta não oferece transferências em *pipeline*, ao contrário da interface AHB-lite discutida na secção anterior. Cada transferência (de leitura ou escrita) demora, no mínimo, dois ciclos de relógio, onde a transição de todos os sinais se encontra relacionada com o flanco ascendente desse mesmo relógio, facilitando a integração do protocolo em qualquer *design*.

O protocolo APB pode ainda, segundo a sua especificação [6], ser interligado a barramentos AMBA AHB-lite e AMBA AXI (*Advanced Extensible Interface*). A tradução do protocolo AHB-lite para APB e vice-versa é utilizada ao longo deste projeto.

2.4 Resumo

O protocolo ponto-a-ponto MHL, que veio responder ao problema da troca de comunicação multimédia entre dispositivos móveis e outros dispositivos fixos, apresenta-se como uma alternativa ao HDMI, oferecendo características que têm em conta a necessidade de baixo consumo energético, inerente aos dispositivos móveis que são alimentados a bateria. O mecanismo de controlo utilizado pelo protocolo MHL utiliza apenas uma linha de comunicação, denominada de

Link Control Bus (CBUS) que permite a configuração da comunicação MHL, bem como a troca de informação acerca do estado dos dispositivos. O protocolo CBUS, apesar de utilizar apenas uma única linha, faz troca de mensagens provenientes de dois canais virtuais (DDC e MSC). O canal DDC é utilizado para troca de informação relativa à autenticação HDCP e à leitura da estrutura de dados EDID, que contém as capacidades e características do dispositivo recetor. O canal MSC permite a execução de serviços de mais alto nível, por exemplo, relacionados com o controlo remoto do dispositivo. A gestão dos dois canais virtuais é feita pela *Translation Layer*, que atua sobre a camada de *Link Layer*, responsável pela gestão dos sinais elétricos.

A Synopsys, produtora de soluções IP para interfaces MHL, contém no seu portfólio, entre outros, o core DesignWare MHL TX Interface IP, responsável pela parte de transmissão deste protocolo. Neste core, a implementação da *Translation Layer* do protocolo CBUS está ao encargo do processador central do sistema onde a solução se englobará. O objetivo da dissertação é dotar a solução de um processador dedicado, responsável pela gestão da *Translation Layer* e otimizado para essa função, obtendo-se assim uma solução mais integrada, oferecendo ao cliente um método de implementação da *Translation Layer* mais seguro e robusto e retirando carga de processamento ao processador central do sistema.

O processador integrado é um ARC EM4, da família ARC EM, desenvolvido para aplicações em sistemas embarcados ou SoC, permitindo ajustar o *tradeoff* performance, área e consumo energético de acordo com a aplicação desejada. O processador ARC EM4 é altamente configurável e contém interfaces ARM® AMBA® AHB™, AHB-lite™ e BVCI, que irão permitir o acesso aos registos necessários para a gestão da *Translation Layer*.

As interfaces entre o processador principal do sistema e o processador integrado ARC são realizadas através de protocolos AMBA APB e AHB-lite , havendo a necessidade de integrar e/ou implementar *bridges* responsáveis pela tradução de um protocolo para o outro. A necessidade desta tradução está relacionada com o facto do processador ARC não conter interfaces APB nativas para ligação direta à interface APB do core transmissor MHL da Synopsys.

Capítulo 3

Especificação Funcional

Neste capítulo é documentada a especificação funcional do projeto desenvolvido, onde é apresentado, da forma mais pormenorizada possível, aquilo que se pretende que o conjunto dos blocos de hardware sejam capazes de realizar, tendo em conta as suas características temporais e funcionais.

3.1 Integração do processador ARC EM4

A Figura 3.1 apresenta a arquitetura de topo da solução DesignWare MHL TX Interface IP, mostrando os vários blocos constituintes. O bloco CBUS, representado a azul, é aquele onde se encontra integrado o processador ARC . De notar que o controlo dos registos associados ao CBUS, bem como dos restantes associados a outros blocos, é feito através de uma interface APB . Esta interface é a utilizada maioritariamente na indústria de dispositivos multimédia para controlar os vários blocos que compõem o sistema. A sua especificação é propriedade da empresa ARM [6]. Desta forma, a integração do core DesignWare MHL TX Interface torna-se mais simples e rápida para o cliente.

3.2 *Top-level*: DWC_mhl_tx_cb_tl_arcem

3.2.1 Visão global do bloco

Este módulo é responsável por tratar da *Translation Layer* do protocolo CBUS que implementa os comandos vindos dos canais virtuais DDC (*Display Data Channel*) e MSC (*MHL Sideband Channel*).

O "*toplevel*" do projeto irá conter, de forma geral, duas interfaces com outros dois blocos pertencentes à solução MHL TX Interface IP. Uma interface realiza a conexão com o microprocessador principal do sistema onde o core MHL TX será integrado. A outra interface é feita com o banco de registos responsável pela interface entre as duas camadas protocolares do protocolo CBUS (*Translation Layer* e *Link Layer*), que controla os periféricos de entrada e saída da interface CBUS.

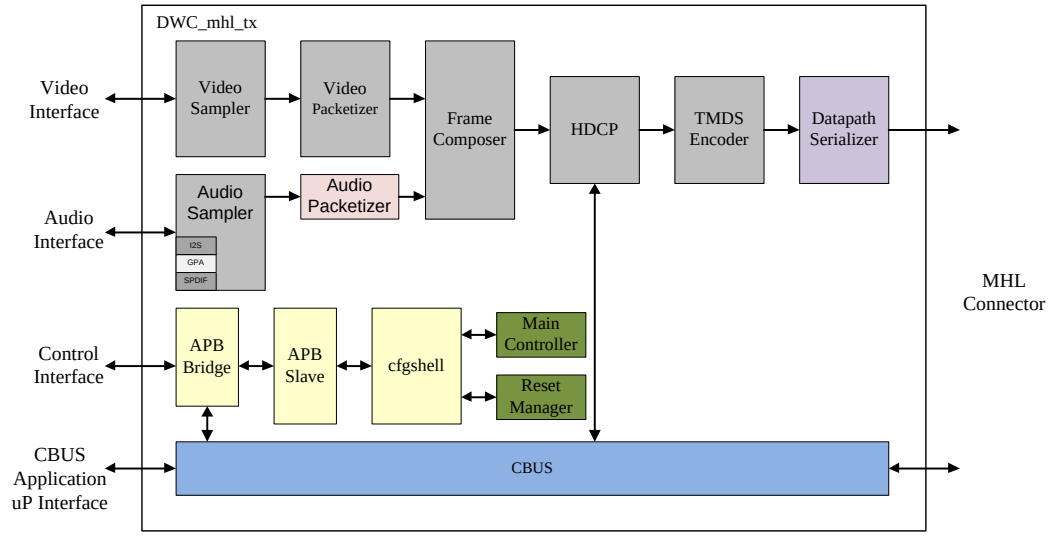


Figura 3.1: MHL TX toplevel

O processador principal do sistema tem acesso a uma API em software que lhe concede o acesso aos serviços da *Translation Layer* com um maior nível de abstração. O processador dedicado ARC EM é responsável por tratar de toda a lógica relacionada com a *Translation Layer* , respondendo aos pedidos do processador principal do sistema, mas também notificando-o acerca de mensagens espontâneas vindas da interface CBUS.

O acesso ao bloco interno de encriptação "HDCP" é feito através do banco de registos da interface CBUS e é usado para o canal virtual DDC, quando este está a realizar troca de mensagens relacionadas com autenticação HDCP.

O controlo dos periféricos CBUS é feito manipulando o banco de registos do bloco CBUS que providencia acesso à linha CBUS para troca de informação entre o aparelho transmissor de conteúdo multimédia e os recetores do mesmo.

A comunicação entre o processador principal do sistema e o processador dedicado ARC EM é feita por troca de mensagens através da memória DCCM que contém regiões alocadas especificamente para esse efeito. Esta abordagem através de regiões de memória dedicadas para troca de mensagens é controlada pelo processador ARC EM que tem assim a função de mestre na comunicação. O controlo do acesso aos *buffers* de comunicação é feito através da troca de interrupções entre os dois processadores que irão notificar ambos do estado da comunicação.

3.2.2 Integração no sistema

O bloco "toplevel" encontra-se integrado dentro do bloco "DWC_mhl_tx_cbus" e contém uma interface com a *bridge* APB principal do core MHL TX, permitindo assim uma forma de comunicação com o processador principal do sistema. O bloco também contém uma interface com o banco registos do CBUS, bem como com um bloco responsável pela geração e tratamento das interrupções relacionadas com o protocolo (*Interrupt Handler*).

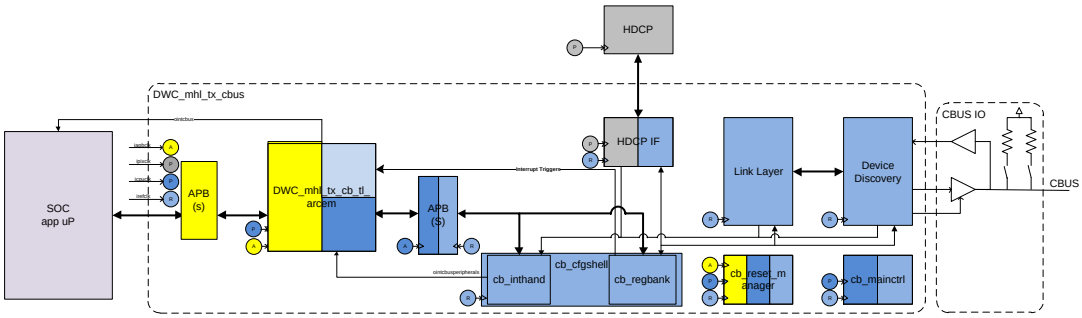


Figura 3.2: Integração do bloco DWC_mhl_tx_cb_tl_arcem no bloco CBUS da solução MHL TX

A Figura 3.2 representa graficamente a integração do bloco desenvolvido na solução MHL TX CBUS existente.

3.2.3 Interface entre o processador principal do sistema e o processador dedicado

Como a interface do core DesignWare MHL TX Interface IP para o processador principal do sistema é feita através de um barramento AMBA APB e o processador ARC EM4 implementa apenas as interfaces externas AMBA AHB-lite e BVCI, é necessária a inclusão de uma *bridge* APB – AHB-lite que faça a tradução dos protocolos e assim possibilite o acesso do processador principal do sistema às memórias ICCM e DCCM, para carregamento inicial do programa e troca de mensagens relacionadas com a implementação da *Translation Layer*.

Uma solução alternativa ao desenvolvimento e inclusão de uma *bridge* APB – AHB-lite seria alterar por completo a interface CBUS do core DesignWare MHL TX Interface IP com o processador central do sistema, passando esta a ser feita através de um barramento AHB-lite .

Com a inclusão de uma *bridge* APB – AHB-lite , não é necessário que o processador principal do sistema tenha que lidar com duas interfaces diferentes: APB para acesso à interface de vídeo e áudio do core MHL e AHB-lite para acesso ao controlo do CBUS, o que facilita a remoção/adição do bloco com o processador integrado. Para além disso, a verificação do core MHL TX não sofre muitas alterações pois a interface principal de acesso (APB) mantém-se

Por outro lado, com a alteração do protocolo de interface com o processador principal do sistema para AHB-lite , a ligação com o processador integrado é direta, podendo ser usado o protocolo AHB-lite na totalidade, incluindo transferências em modo *burst* que permitem uma maior largura de banda.

A Tabela 3.1 apresenta a análise das vantagens de cada uma das duas alternativas, relativamente à outra.

Após análise dos prós e contras de cada uma das soluções possíveis, optou-se pela decisão do desenvolvimento e inclusão de uma *bridge* APB – AHB-lite .

Tabela 3.1: Análise das vantagens de cada uma das soluções relativamente à outra: "Inclusão de uma *bridge* APB – AHB-Lite" ou "Alteração da interface principal de APB para AHB-Lite"

Inclusão de uma *bridge* APB – AHB-Lite	**Alteração da interface principal para AHB-Lite**
• Não é necessário alterar a interface já existente, facilitando a remoção/adição do bloco com o processador ARC , podendo-se fornecer assim ambas as soluções. • Não é necessário o processador principal do sistema fazer a ligação com duas interfaces diferentes – APB e AHB-lite . • Verificação mais fácil de se fazer, dada a maior simplicidade do protocolo APB relativamente ao protocolo AHB-lite .	• Ligação ao processador ARC EM4 mais fácil (direta). • Maior largura de banda. • Carregamento da memória de instruções mais rápida, devido às escritas em *burst* possibilitadas pela interface AHB-lite .

Desta forma, a integração do processador ARC EM4 torna-se mais fácil, sem que haja a necessidade de remodelar toda a interface do core DesignWare MHL TX Interface. A principal vantagem que a solução de alterar a interface para AHB-lite oferecia (aumento da largura de banda) não teria um impacto notório, visto que a troca de informação através desta interface (carregamento da memória de programa e troca de mensagens pela DCCM) não é expectável que necessite de uma frequência mais elevada que a que a interface APB oferece.

3.2.4 Interface entre o processador dedicado e os periféricos CBUS

Tal como a interface entre o processador principal do sistema e o processador ARC, a interface de acesso aos periféricos por parte do processador ARC apenas permite a configuração para o protocolo AMBA AHB-lite ou BVCI. Visto que a solução existente apresenta uma interface APB acedível diretamente pelo processador do SoC, foi inserida uma *bridge* AHB-lite – APB [7] existente na biblioteca de cores da Synopsys, não alterando desta forma a estrutura já existente.

Com a inclusão do bloco "AHB-lite – APB ", beneficia-se de este ser um módulo já verificado e amplamente utilizado nas soluções Synopsys.

3.2.5 Espaço de endereçamento

O mapeamento de memória pode ser observado na Figura 3.3, onde se verifica um espaço de endereçamento contíguo acedível através de um endereço de 24 bits.

Na anterior solução, o espaço de endereçamento do core MHL TX, sem a presença do processador ARC EM4, é dividido em dois blocos de memória, o bloco "*MHL Datapath*", responsável

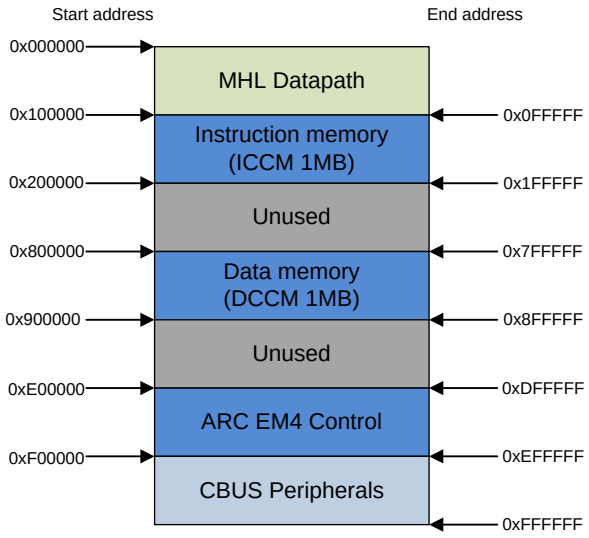


Figura 3.3: Espaço de endereçamento total

pelo controlo dos blocos relacionados com a transferência de vídeo e áudio, e o bloco "*CBUS Peripherals*", que permite a gestão dos periféricos da interface CBUS, providenciando uma interface com a *Link Layer* do protocolo. A Figura 3.4 representa o espaço de endereçamento da solução sem processador ARC EM4.

Com a integração do processador ARC EM4, o espaço de endereçamento que não é utilizado na solução anterior, marcado na Figura 3.4 como "*Unused*", é usado para mapear as memórias DCCM e ICCM, bem como o bloco *cb_arc_cfgshell*, responsável pelo controlo do processador. O espaço de endereçamento da solução com processador ARC EM4 dedicado, visto pelo processador principal do SoC, encontra-se representado na Figura 3.5

O espaço de endereçamento, visto pelo processador ARC EM4 encontra-se representado na Figura 3.6 e é composto pelas memórias DCCM e ICCM, e pelos periféricos CBUS ("*CBUS Peripherals*"), responsáveis pelo controlo do canal CBUS. Embora o processador principal do SoC veja a ICCM na região 0x100000 - 0x1FFFFF, o processador ARC EM vê essa mesma memória mapeada em 0x000000 - 0x0FFFFF. Esta tradução de endereços é feita internamente e está relacionada com um bug no software de configuração do processador ARC que não permite a mudança da região inicial para a memória ICCM.

3.2.6 Sub-blocos constituintes

A Figura 3.7 representa os sub-blocos que constituem o bloco de topo DWC_mhl_tx_cb_tl_arcem bem como as suas conexões com os restantes blocos da solução MHL TX CBUS.

Os sub-blocos presentes na Figura 3.7 contém as seguintes funções:

TL uP ARC EM4

Este módulo é implementado usando um microprocessador ARCv2EM4 e é responsável por

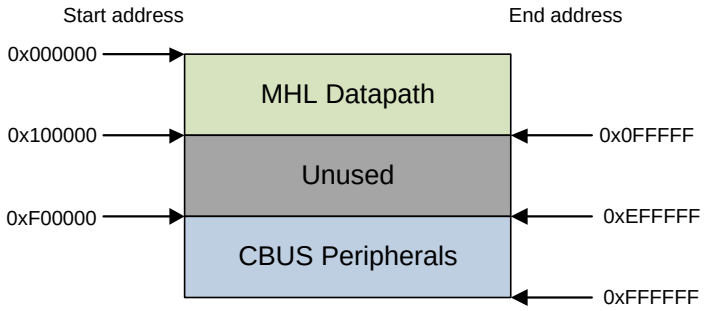


Figura 3.4: Espaço de endereçamento da solução sem processador ARC EM4

correr o software necessário à implementação da *Translation Layer* do protocolo CBUS.

cb_arc_regbank

Banco de registos responsável pelo controlo do microprocessador ARC EM4, que é feito pelo processador principal do SoC. Inclui registos que fazem o o controlo da interface de *Halt/Run* e que despoletam interrupções para o processador ARC EM4. Estas interrupções são tratadas pelo sub-bloco *cb_inthand* que é responsável por todas as interrupções relacionadas com a interface CBUS.

cb_arc_inthand

Este bloco faz o tratamento das interrupções vindas do processador ARC EM4, que pretendem notificar o processador principal do SoC (*SoC app uP*). Este último é responsável por limpar, forçar ou mascarar as interrupções, conforme seja necessário.

APB(S)

APB(S) representa um *slave* APB que faz a transferência de dados entre a *bridge* APB e o bloco *cb_arc_cfgshell*. Este bloco é responsável pela sincronização de dados do domínio "APB clock" para o domínio "REF clock" e vice-versa.

APB 2 AHB Lite

Este bloco converte os sinais da interface APB para sinais de acordo com o interface AHB-Lite. Esta tradução é necessária devido às restrições de interfaces do processador ARC EM4 que não suporta o protocolo APB.

AHB Lite 2 APB

Este módulo é constituído por uma *bridge* que faz a tradução do protocolo AHB-lite para APB . O bloco pertence ao portfólio de cores da interface AMBA da Synopsys, tendo sido

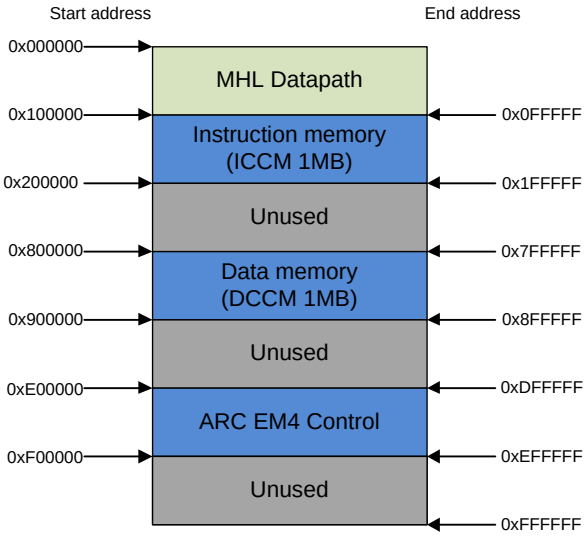


Figura 3.5: Espaço de endereçamento da solução com processador ARC EM4, visto do processador principal do sistema

incorporado na solução. A sua necessidade advém do bloco *cb_cfgshell* apresentar uma interface APB ao passo que a interface de periféricos do processador AHB-lite apenas permite a interface AHB-lite .

cb_regbank

Embora este bloco seja parte da solução MHL TX já existente, sofre algumas adições de forma a acomodar registos dedicados a despoletar interrupções no bloco *cb_arc_inthand* - interrupções essas geradas pelo processador ARC EM4 para notificar o processador central do sistema.

cb_inthand

Tal como o bloco anterior, este também já existia na solução MHL TX mas é alterado para conter a lógica de tratamento das interrupções geradas pelo processador central do sistema a fim de notificar o microprocessador ARC EM4. Estas interrupções são geradas através da manipulação do banco de registos *cb_arc_regbank*.

3.2.7 Frequências de relógio e tecnologias-alvo

A Tabela 3.2 apresenta as frequências que são necessárias atingir com a solução para os vários domínios de relógio.

As tecnologias-alvo são 40nm e 28nm e pretende-se que as restrições de frequência de relógio sejam cumpridas para ambas as tecnologias.

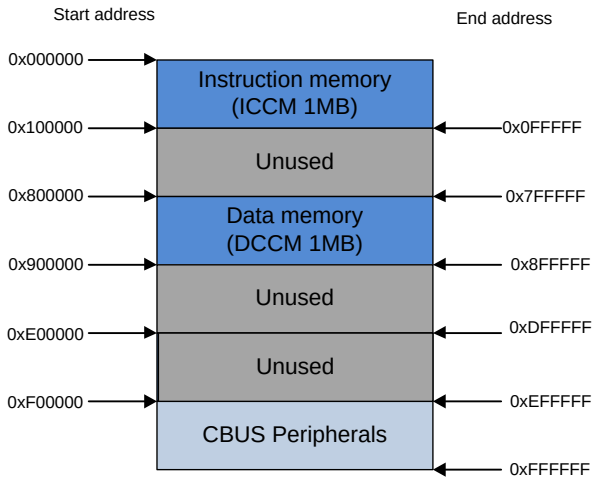


Figura 3.6: Espaço de endereçamento da solução com processador ARC EM4, visto do processador ARC EM4

3.3 Sub-bloco: *TL uP ARC EM4*

Este módulo é composto por um microprocessador ARC EM4 e é responsável pela implementação da *Translation Layer* do protocolo CBUS, correndo software específico.

As configuração do processador e dos seus periféricos, apresenta as seguintes características:

- Espaço de endereçamento contíguo, acedido por endereços de 24bits;
- Duas memórias *Closely Coupled Memories*, indicadas para o processador ARC EM4:
  - 1MByte ICCM (*Instruction Closely Coupled Memory*) para instruções;
  - 1MByte DCCM (*Data Closely Coupled Memory*) para dados.
- Duas interfaces de acesso direto a memória AHB-lite para acesso às memórias de dados e de instruções, referidas no ponto anterior;
- Interface AHB-lite para acesso do processador principal do sistema;
- Modo "*Halt Upon Reset*" que permite que o processador fique em modo de espera após um sinal de *reset*, permitindo o carregamento inicial das memórias de instruções e de dados;
- Uma linha de interrupções externas;
- Controlo *Halt/Run* externo;
- Controlo exterior de proteção de código;
- Utilização de 8 *Actionpoints*, verificação da *stack* e depuração por JTAG.

Tabela 3.2: Frequências de relógio para ASIC 40nm e 28nm

Domínio de relógio	**Frequência mínima**	**Frequência típica**	**Frequência máxima**
icpuclk	Deve ser maior ou igual que a frequência do domínio *irefclk* por forma a não ter impacto na performance do sistema	–	Depende do tipo de tecnologia e dos requisitos do SoC
irefclk	14 MHz	27 MHz	40 MHz
iapbclk	18 MHz	–	Depende do tipo de tecnologia usada

O microprocessador está configurado para entrar em modo *halt* após o seu *reset*. Esta capacidade permite que o processador principal do SoC possa descarregar o programa para a memória ICCM através da interface DMI (*Direct Memory Interface*). Após o pedido para o processador arrancar o seu funcionamento normal, o mecanismo de proteção de código é ativado prevenindo a escrita e leitura indevidas da memória de instruções durante o tempo de execução. O mecanismo de bloqueio da memória só pode ser desligado durante o *reset* total de todo o core MHL TX, dando assim algum nível de proteção ao processador ARC EM4. A interface DMI da memória DCCM é acessível durante a execução do processador ARC , permitindo a troca de mensagens entre os dois processadores. Este método de troca de mensagens permitirá a implementação de:

- Comandos MSC:
  - RAP - *Request Action Protocol*;
  - RCP - *Remote Control Protocol*;
  - UCP - *UTF8 Character Protocol*.
- Comandos DDC;
- Espaço de registos do aparelho (*Device Register Space*);
- Acesso ao *Device Register Space*.

3.3.1 Mapeamento em memória

3.3.1.1 Mapeamento da *Translation Layer* na memória DCCM

A Tabela 3.3 representa o mapeamento dos registos específicos do protocolo CBUS na memória DCCM. O processador ARC irá utilizar esses registos para compor comandos da *Translation Layer* . A atualização e o preenchimento destes registos encontra-se ao encargo tanto do processador principal do sistema, como do processador ARC .

Tabela 3.3: Mapeamento da *Translation Layer* na memória DCCM

Offset	**Nome**	**Descrição**
0x00 - 0x0F	*Capability Registers*	"*MHL Device Registers*"
0x10 - 0x1F	Reservado	
0x20 - 0x2F	*Interrupt Registers*	"*MHL Device Registers*"
0x30 - 0x3F	*Status Registers*	"*MHL Device Registers*"
0x40 - 0x7F	*Scratchpad Registers*	"*MHL Device Registers*"
0x80 - 0xFF	Reservado	
0x100 - 0x10F	*ARCv2EM4 Status Register*	Registo de estado do processador ARC EM4
0x110 - 0x11F	*ARCv2EM4 Control Register*	Registo de controlo do processador ARC EM4
0x120 - 0x121	*Vendor ID*	Registo da *Translation Layer* , identificativo do fornecedor do aparelho
0x122 - 0x17F	Reservado	
0x180 - 0x1FF	*ARCv2EM4 Debug Registers*	Registos genéricos para efeito de depuração de software
0x200 - 0x281	*Command Transmit Buffer*	*Buffer* para troca de comandos da *Translation Layer*
0x282 - 0x304	*Command Reply Buffer*	*Buffer* para troca de comandos da *Translation Layer*
0x305 - 0x386	*Command Receive Buffer*	*Buffer* para troca de comandos da *Translation Layer*

3.3.1.2 Requisitos do registo *Command Transmit Buffer*

O registo *Command Transmit Buffer* é utilizado para o envio de mensagens do processador principal do sistema para o microprocessador ARC EM4 e encontra-se divido logicamente da seguinte forma:

- Tipo de comando - 8 bits;
- Tamanho ocupado pelo *buffer* - 7 bits;
- Dados contidos no *buffer* - 128 bytes;

O tamanho máximo ocupado por dados é limitado pelo tamanho de um bloco da estrutura EDID (versão 1.3).

3.3.1.3 Requisitos do registo *Command Reply Buffer*

O registo *Command Reply Buffer* é utilizado para o envio de mensagens de resposta do microprocessador ARC EM4 para o processador central do sistema. Logicamente encontra-se dividido da seguinte forma:

- Tipo de comando - 8 bits;

- Tamanho ocupado pelo *buffer* - 7 bits;
- Estado do comando, representando possíveis ocorrências de *timeout* ou erros;
- Dados contidos no *buffer* - 128 bytes;

3.3.1.4 Requisitos do registo *Command Reply Buffer*

O registo *Command Receive Buffer* é utilizado para o envio de mensagens do microprocessador ARC EM4 para o processador principal do sistema. Estas mensagens são originadas por informação detetada na linha CBUS e que não é uma resposta direta a um pedido prévio por parte do processador principal do sistema.

- Tipo de comando - 8 bits;
- Tamanho ocupado pelo *buffer* - 7 bits;
- Dados contidos no *buffer* - 128 bytes;

3.3.1.5 Mapeamento da *Translation Layer* na memória ICCM

A Tabela 3.4 representa algumas regiões da memória ICCM que serão dedicadas para dados do *firmware* responsável pela implementação da *Translation Layer* .

O restante espaço de endereçamento é utilizado para o carregamento da memória de programa.

Tabela 3.4: Mapeamento da *Translation Layer* na memória ICCM

Offset	**Nome**	**Descrição**
0x00 - 0x03	*Firmware version*	Versão do software carregado
0x04 - 0x07	*Firmware Build Number*	Número da versão compilada do software carregado
0x08 - 0x0B	*Firmware Status*	Estado do software carregado
0x0C - 0x0F	*Firmware Control*	Controlo do software carregado
0x10 - 0x13	*Firmware Debug Control*	Depuração do controlo do software carregado

3.4 Sub-bloco: *cb_arc_regbank*

Este bloco representa um banco de registos que permite o controlo do processador ARC EM4. Contém registos responsáveis pelo tratamento da interface de *Halt/Run* e pelo despoletar de interrupções para o processador ARC EM4. Estes últimos, contêm barramentos de saída com o valor do registo que são diretamente ligados à entrada do bloco *cb_inthand*, responsável por todas as interrupções relacionadas com a interface CBUS.

3.4.1 Controlo da execução do processador

Uma das funções do sub-bloco *cb_arc_regbank* é o controlo do estado *Halt/Run* do processador ARC EM4, através da manipulação dos sinais de uma interface dedicada. A Tabela 3.5 representa os sinais dessa interface e o seu mapeamento no banco de registos. A direção dos sinais apresentada encontra-se do ponto de vista do processador ARC EM4.

Tabela 3.5: Sinais de controlo *Halt/Run* e seu mapeamento em memória

Nome	**Direção**	**Descrição**	**Mapeamento no banco de registos *cb_arc_regbank***
sys_halt_sts	Saída	Indica que o processador está parado em modo *halt*	*halt_run_status[0]*
sys_sleep_sts	Saída	Indica que o processador está em modo *sleep*	*halt_run_status[1]*
sys_sleep_mode_sts[2:0]	Saída	Indica qual o tipo de estado *sleep* em que o processador se encontra	*halt_run_status[4:2]*
sys_halt_req	Entrada	Registo "*write only*"[1] responsável por requerer que o processador entre em modo *Halt*	*halt_run_req[0]*
sys_run_req	Entrada	Registo "*write only* responsável por requerer que o processador entre em modo *Run*	*halt_run_req[1]*
rst_n	Entrada	Registo "*write only* responsável por requerer que que seja feito um *reset* ao processador	*halt_run_req[2]*

3.4.2 Geração de Interrupções

Outra função do sub-bloco *cb_arc_regbank* é fornecer uma interface ao processador principal do sistema para que este possa gerar interrupções no ARC EM4, relacionadas com a troca de mensagens entre eles através da memória DCCM. A Tabela 3.6 demonstra o mapeamento no banco de registos dos vários registos que permitem despoletar as referidas interrupções.

3.5 Sub-bloco: *cb_arc_inthand*

Este bloco é responsável pela implementação do mecanismo de disparo de interrupções. Este mecanismo já existe na solução sem o processador ARC EM4 integrado e irá ser replicado dado que é um método já testado e amplamente utilizado nos restantes cores, facilitando a sua integração.

[1] Neste contexto e em futuras referências ao longo deste documento, um registo "*write only*" é um registo cuja escrita apenas é válida durante um ciclo de relógio, sendo o registo limpo no ciclo imediatamente a seguir

Tabela 3.6: Mapeamento dos registos responsáveis pelo despoletar de interrupções no bloco *cb_arc_regbank*

Mapeamento no banco de registos *cb_arc_regbank*	**Direção**	**Descrição**
int_cb_access_req[0]	Do processador principal do sistema para o ARC EM4	Processador principal do sistema pede acesso à memória DCCM
int_cb_access_rpl_read[1]	Do processador principal do sistema para o ARC EM4	Indica que o *reply buffer* foi consumido
int_cb_access_rcv_read[2]	Do processador principal do sistema para o ARC EM4	Indica que o *receive buffer* foi consumido
int_cb_access_tx_write[3]	Do processador principal do sistema para o ARC EM4	Indica que o *transmit buffer* foi escrito
int_cb_access_devreg_read[4]	Do processador principal do sistema para o ARC EM4	Indica que o *device register space* foi lido
int_cb_access_devreg_write[5]	Do processador principal do sistema para o ARC EM4	Indica que o *device register space* foi escrito

Os pedidos de interrupções serão gerados através de registos específicos presentes no banco de registos *cb_regbank*. Esses pedidos serão depois concatenados pelo bloco *cb_arc_inthand* numa única linha de interrupção direcionada ao processador principal do sistema.

As interrupções tratadas por este bloco estão relacionadas com a troca de mensagens entre o processador ARC EM4 e o processador central do sistema. Limpar estas interrupções ou, opcionalmente, forçá-las ou mascará-las é da exclusiva responsabilidade do processador principal do sistema que o pode fazer através da interface APB que lhe é oferecida. Esta interface também permite a leitura do estado das interrupções.

A Tabela 3.7 representa o mapeamento das interrupções no banco de registos *cb_regbank*.

3.6 Sub-bloco: *cb_regbank*

Este bloco já existe na solução MHL TX CBUS sem o processador ARC EM4 e é utilizado para configurar os blocos de hardware relacionados com o protocolo CBUS, permitindo a implementação da *Translation Layer* .

O banco de registos existente é modificado por forma a inserir mais um registo responsável pela geração de interrupções, através do mapeamento dos bits do registo em várias linhas de pedido de interrupções que depois irão ser concatenadas pelo bloco *cb_arc_inthand*, documentado na secção anterior.

Estas interrupções estão relacionadas com a troca de mensagens entre o processador principal do sistema e o ARC EM4 e encontra-se na Tabela 3.7 o seu mapeamento no banco de registos.

Tabela 3.7: Mapeamento dos registos responsáveis pelo despoletar de interrupções no bloco *cb_regbank*

Mapeamento no banco de registos *cb_regbank*	**Direção**	**Descrição**
int_cb_access_grt[0]	Do ARC EM4 para o processador principal do sistema	ARC EM4 concede acesso à memória DCCM para o processador principal do sistema
int_cb_noti_rcv[1]	Do ARC EM4 para o processador principal do sistema	Notificação para o processador principal do sistema que existe uma nova mensagem no *receive buffer*
int_cb_noti_rpl[2]	Do ARC EM4 para o processador principal do sistema	Notificação para o processador principal do sistema que existe uma nova mensagem no *reply buffer*

3.7 Sub-bloco: *cb_inthand*

Este bloco já existe na solução sem o processador ARC EM4 e implementa a lógica em volta das interrupções relacionadas com o protocolo CBUS. Em adição às interrupções já existentes, é implementado um outro vetor relacionado com interrupções vindas do processador principal do sistema em direção ao ARC EM4. Este vetor de interrupções recebe pedidos do banco de registos *cb_arc_regbank*, controlado pelo processador principal do sistema, através da interface APB que lhe é oferecida.

O mapeamento das interrupções no banco de registos, encontra-se representado na Tabela 3.6.

3.8 Sub-bloco: Interface APB 2 AHB-Lite

O módulo, tal como o seu nome indica, realiza a tradução do protocolo APB para AHB-lite [5], fazendo a ponte entre a *bridge* APB do bloco de topo e cada uma das interfaces AHB-lite das memórias DCCM e ICCM do processador ARC EM4.

Apesar dos dois protocolos não serem iguais, é possível fazer um mapeamento de alguns dos sinais APB para os sinais AHB-lite , de acordo com a sua função. A Tabela 3.8 demonstra esse mapeamento, descrevendo a função dos sinais.

3.8.1 Sincronização de sinais APB e AHB-lite

Os sinais APB e AHB-lite estão relacionados com diferentes domínios de relógio. A interface APB está relacionada com o relógio APB enquanto que a interface AHB-lite está relacionada com o relógio interno do processador ARC EM4. Não deve ser assumida qualquer relação entre os dois domínios de relógio e, por isso, é necessário implementar um mecanismo de sincronização entre as duas interfaces.

O método utilizado para sincronizar as duas interfaces é baseado na geração de um sinal de pedido pela interface APB , quando esta quer realizar uma leitura/escrita, ao que se sucede um sinal de concessão (*acknowledgement*) por parte da interface AHB-lite . Ambos os sinais, de pedido e concessão, são pulsos de um ciclo de relógio que são sincronizados por células próprias propriedade da Synopsys.

Cada transferência demora no mínimo dois ciclos de relógio APB. No caso do relógio AHB-Lite ser mais lento que o relógio APB, é necessário que a interface AHB-lite use o sinal APB PREADY, por forma a introduzir os ciclos de atraso necessários à conclusão da transferência.

3.8.2 Diagrama de blocos

A Figura 3.8 representa o diagrama de blocos da solução implementada, encontrando-se na Tabela 3.9 a descrição dos dois sub-blocos constituintes.

3.8.2.1 Sub-bloco *DWC_mhl_tx_apb2ahbl_engine*

A Figura 3.9 representa os sinais da interface deste bloco. Do lado esquerdo encontra-se a interface APB enquanto que do lado direito encontra-se a interface com o módulo *DWC_mhl_tx_apb2ahbl_ctrl*, responsável pelo controlo da interface AHB-lite .

Este módulo é responsável por:

- Amostrar o endereço, sinal de leitura/escrita e dados a escrever durante a fase de *setup* do protocolo APB :
  - *oapb2ahbl_addr* mantém o sinal *iapb_paddr*, amostrado durante a fase de *setup*.
  - *oapb2ahbl_data* mantém os dados do barramento *iapb_pwdata*, amostrados durante a fase de *setup*.
  - *oapb2ahbl_wrrden* mantém o sinal *iapb_pwrite*, amostrado durante a fase de *setup*.
- Gerar o sinal de pedido para o bloco *DWC_mhl_tx_apb2ahbl_ctrl*.
- Registar os dados lidos e a resposta vindos da interface AHB-lite , de acordo com o sinal de *acknowledge* vindo do bloco *DWC_mhl_tx_apb2ahbl_ctrl*:
  - *oapb_prdata* mantém os dados do barramento *iahbl2apb_data*, amostrados após receção do sinal de *acknowledgement*.
  - *oapb_pslverr* mantém o sinal *iahbl2apb_resp*, amostrado após receção do sinal de *acknowledgement*.
- Gerar o sinal de *ready* para a interface APB , simplesmente atrasando o sinal de *acknowledge* vindo do bloco *DWC_mhl_tx_apb2ahbl_ctrl*.

3.8.2.2 Sub-bloco *DWC_mhl_tx_apb2ahbl_ctrl*

A Figura 3.10 representa os sinais da interface deste bloco. Do lado esquerdo encontra-se a interface com o módulo *DWC_mhl_tx_apb2ahbl_engine*, enquanto que do lado direito é representada a interface AHB-lite .

Este módulo é responsável por:

- Curto-circuitar o sinal *oahbl_hready* ao sinal *iahbl_hready_out*, devido à interface AHB-lite ter um único escravo.
- Fixar o barramento *oahbl_hburst* ao valor 3'b000, significando um único "*burst*". A transferência em *burst* não será usada, devido às restrições na interface APB .
- Fixar o barramento *oahbl_hsize* ao valor 3'b010, indicando que a transferência é constantemente de 4 bytes.
- Fixar o sinal *oahbl_hsel* a 1'b1, selecionando sempre o único escravo presente na interface AHB-lite .
- Após a receção o pulso de pedido no sinal *ipreq*, manter o os sinais de controlo necessários para a transferência:
  - *oahbl_htrans* é colocado com o valor 2'b10 (NONSEQ), indicando uma transferência não sequencial, tal como indicado no protocolo AHB-lite . Este barramento é colocado com o valor 2'b00 (IDLE) quando não está a decorrer nenhuma transferência.
  - *oahbl_hwrite* é o sinal que indica se a transferência é uma escrita (1 lógico) ou uma leitura (0 lógico) e é resultado da amostragem do sinal de *iapb2ahbl_wrrden* quando o sinal *ipreq* é ativo.
- Após a deteção do sinal *ipreq*, manter a linha do endereço e dos dados a escrever:
  - *oahbl_haddr* mantém os dados do barramento *iapb2ahbl_addr*, amostrados após receção do sinal de pedido (*ipreq*).
  - *oahbl_hwdata* mantém os dados a seres escritos, provenientes do barramento *iapb2ahbl_data*, amostrados um ciclo após a receção do sinal de pedido (*ipreq*).
- Amostrar os dados lidos através da interface AHB-lite :
  - Devido às características temporais da interface AHB-lite , os dados podem ser amostrados um ciclo após ter sido colocado o endereço na interface AHB-lite .
- Gerar o sinal de *acknowledgement* (*ohack*) e o sinal de resposta (*oahbl2apb_resp*), que indicam ao módulo *DWC_mhl_tx_apb2ahbl_engine* que a operação foi finalizada na interface AHB-lite , comunicando possíveis erros:

- Devido às características temporais da interface AHB-lite , o sinal de *acknowledgement* é enviado quando o sinal *iahbl_hready_out* é ativo.
- O sinal *oahbl2apb_resp* contém a mesma informação do sinal *iahbl_hresp*.

3.8.3 Diagramas Temporais

A Figura 3.11 mostra o diagrama temporal dos sinais da interface *APB 2 AHB-Lite* durante uma leitura, enquanto que a Figura 3.12 representa o diagrama temporal dos sinais da interface durante uma escrita.

3.9 Plano de verificação

Nesta secção é apresentado o Plano de Verificação elaborado antes da implementação do projeto, com os casos de uso a testar depois de implementados todos os blocos. Este Plano serve como guia durante o desenvolvimento do *testbench* para o projeto. A Tabela 3.10 representa o plano de testes elaborado.

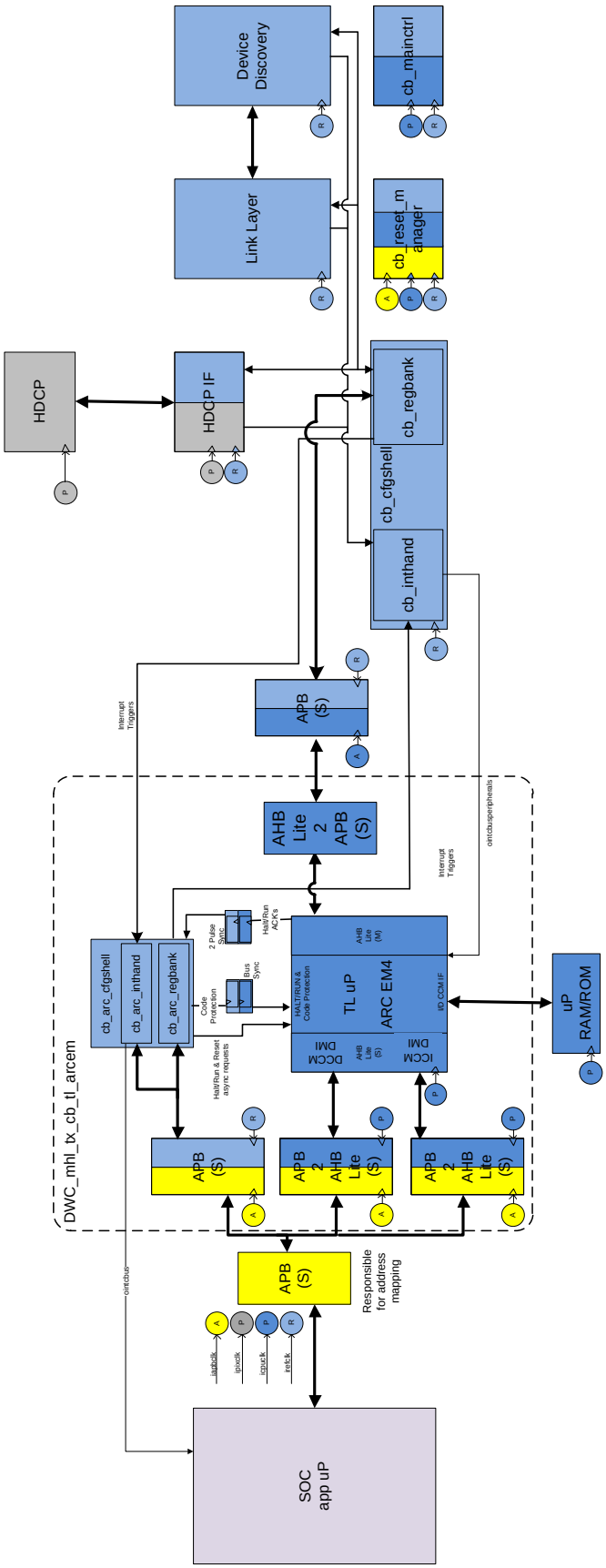


Figura 3.7: Representação do bloco DWC_mhl_tx_cb_tl_arcem em sub-blocos e das suas interfaces

Tabela 3.8: Mapeamento dos sinais da interface *APB 2 AHB-Lite*

Sinal APB	**Sinal AHB-lite**	**Descrição**	**Observações**
iapb_pwrite	*oahbl_hwrite*	Indica se a transferência é uma leitura (0 lógico) ou uma escrita (1 lógico)	N/A
iapb_paddr[15:0]	*oahbl_haddr[15:0]*	Indica o endereço a ler ou escrever	N/A
iapb_pwdata[31:0]	*oahbl_hwdata[31:0]*	Dados a escrever	N/A
oapb_prdata[31:0]	*iahbl_hrdata[31:0]*	Dados lidos	N/A
iapb_psel	*oahbl_hsel*	Sinal de seleção do escravo	N/A
oapb_pslverr	*iahbl_hresp*	Resposta indicando um possível erro que tenha ocorrido com a transferência	N/A
oapb_pready	*iahbl_hready_out*	Sinal indicando que a resposta se encontra pronta	Este sinal pode ser usado pelo escravo para introduzir atrasos durante a transferência
iapb_penable	N/A	Sinal indicando o escravo que pode colocar/retirar dados da linha	Este sinal não é usado na interface AHB-lite mas é usado pela interface *APB 2 AHB-Lite* para controlar os sinais de saída do lado AHB-lite
N/A	*oahbl_hready*	Sinal enviado para todos os escravos e para o mestre indicando que a transferência anterior foi concluída	Este sinal será curto-circuitado ao sinal *iahbl_hready_out*, devido a só existir um escravo nesta interface.
N/A	*oahbl_hburst[2:0]*	Indica o tipo de *burst*. Encontra-se relacionado com a possibilidade da interface AHB-lite permitir transferências em *burst*	Este sinal será fixado ao valor 3'b000, indicando que a transferência não é em modo *burst*
N/A	*oahbl_hsize[2:0]*	Tamanho da transferência	Este sinal será fixado ao valor 3'b010, indicando que a transferência é de 4 bytes (32 bits)
N/A	*oahbl_htrans[1:0]*	Tipo da transferência	Este sinal será gerado pela interface e fixado ao valor 2'b00 (IDLE) quando não estiver a decorrer nenhuma transferência ou a 2'b10 (NONSEQUENTIAL), indicando que a transferência é não sequencial

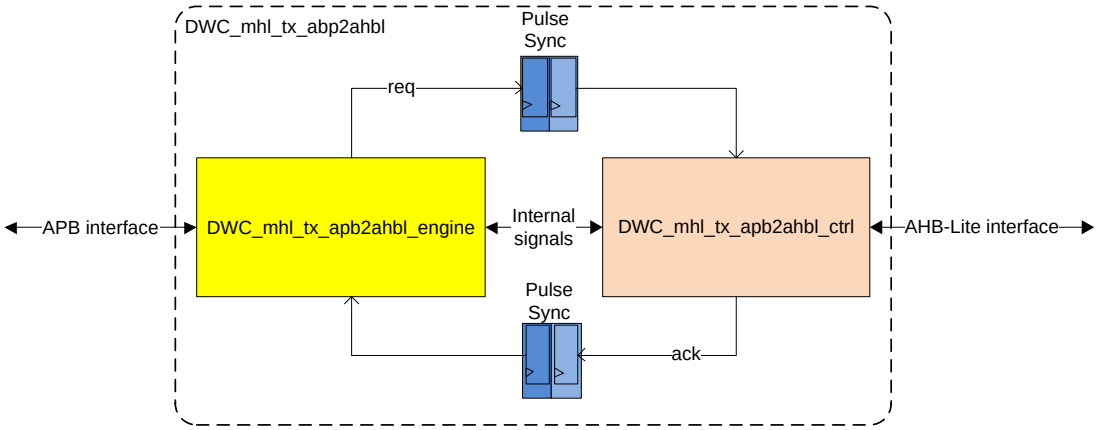


Figura 3.8: Diagrama de blocos do módulo *APB 2 AHB-Lite*

Tabela 3.9: Mapeamento dos sinais da interface *APB 2 AHB-Lite*

Nome do bloco	**Descrição**
DWC_mhl_tx_apb2ahbl_engine	Este módulo é responsável pela deteção das transações nos sinais APB e amostrar o endereço, os dados e o sinal de *ready*. A operação de leitura/escrita é requerida ao sub-bloco *DWC_mhl_tx_apb2ahbl_ctrl* através do sinal *opreq*. Os valores dos sinais amostrados são segurados até que a transação seja completa na interface AHB-lite .
DWC_mhl_tx_apb2ahbl_ctrl	Este bloco é responsável por implementar as transações de leitura/escrita na interface AHB-lite . Depois de estarem concluídas, este notifica o bloco *DWC_mhl_tx_apb2ahbl_engine* da conclusão, utilizando para isso o sinal *ohack*.

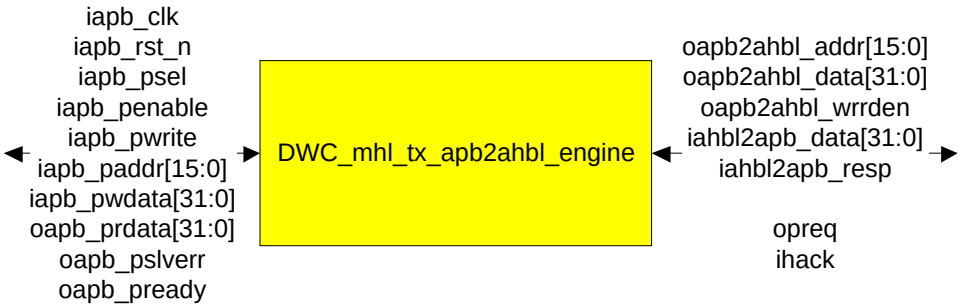


Figura 3.9: Diagrama de blocos do sub-bloco *DWC_mhl_tx_apb2ahbl_engine*

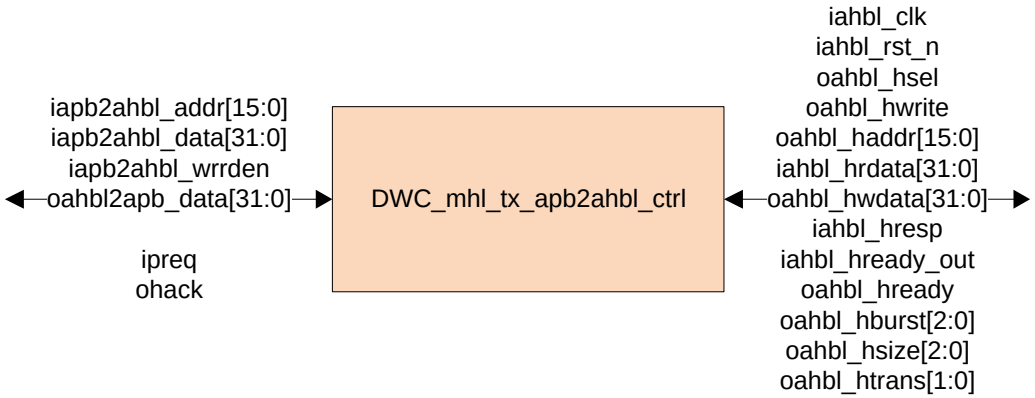


Figura 3.10: Diagrama de blocos do sub-bloco *DWC_mhl_tx_apb2ahbl_ctrl*

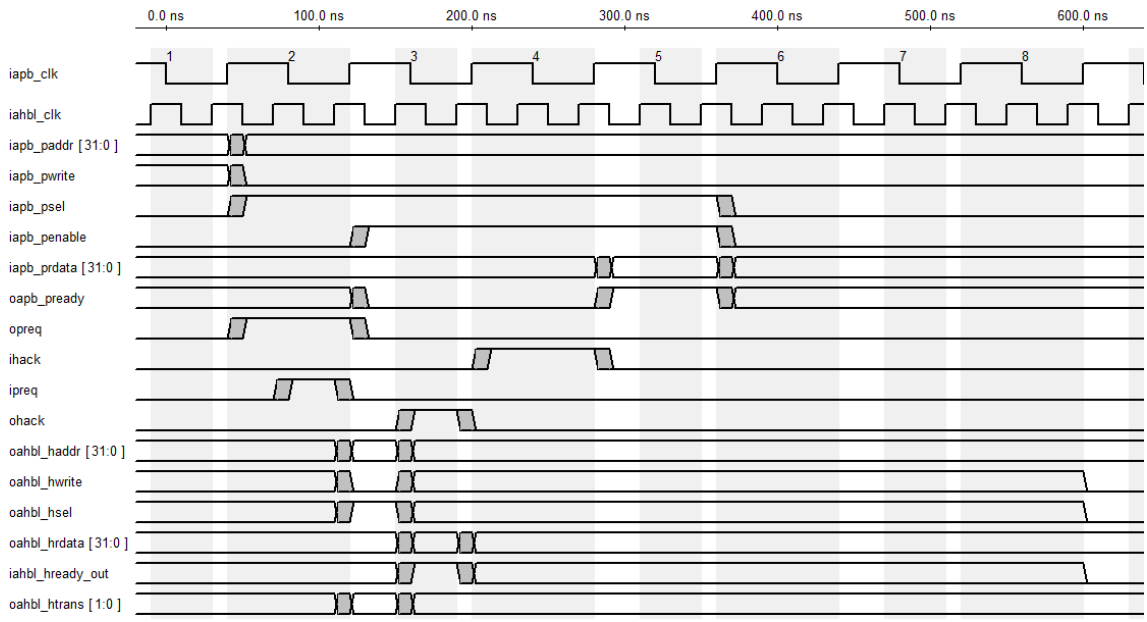


Figura 3.11: Diagrama temporal dos sinais da interface *APB 2 AHB-Lite* durante a operação de leitura

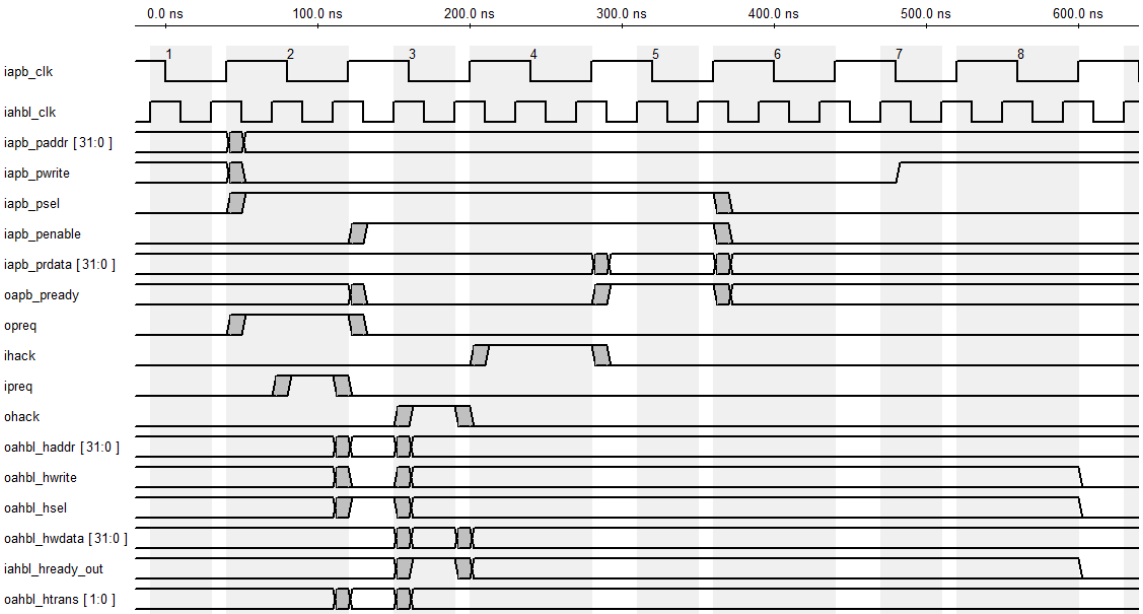


Figura 3.12: Diagrama temporal dos sinais da interface *APB 2 AHB-Lite* durante a operação de escrita

Tabela 3.10: Plano de Verificação do projeto

Componente	**Teste**	**Comentário**
APB Bridge	Verificar a correta operação quando são colocados na interface endereços de registos inexistentes.	–
Banco de registos	Comparar a implementação do banco de registos com o mapeamento planeado na Especificação Funcional	Teste a realizar para os dois bancos de registos da solução (*cb_arc_regbank* e *cb_regbank*)
	Verificar a geração dos sinais de controlo *Halt/Run* através da manipulação do banco de registos	Teste a realizar apenas para o banco de registos *cb_arc_regbank*
Interrupt Handler	Comparar a implementação dos blocos de interrupções com o mapeamento planeado na Especificação Funcional	Teste a realizar para os dois blocos de interrupções da solução (*cb_arc_inthand* e *cb_inthand*)
APB 2 AHB-Lite	Verificar o funcionamento normal com a frequência APB maior que a frequência de CPU	–
	Verificar o funcionamento normal com a frequência de CPU maior que a frequência APB	–
	Verificar transferências de escrita e leitura sequenciais, sem desativar o sinal de seleção (*back to back transactions*)	–
ICCM/DCCM	Verificar a correta escrita e leitura nas memórias ICCM/DCCM	–
	Verificar a funcionalidade de proteção de código. Ativar esta funcionalidade e verificar o seu correto funcionamento através de tentativas de leitura e escrita	Teste a realizar apenas para a memória ICCM, onde esta funcionalidade se encontra disponível
AHB-Lite 2 APB	Verificar a correta operação quando são colocados na interface endereços de registos inexistentes.	–
ARC EM4	Verificar o correto funcionamento da interface de periféricos	A interface dos periféricos é uma interface AHB-lite
	Verificar o correto funcionamento da interface de interrupções	O processador tem apenas uma linha de interrupção externa
	Verificar a funcionalidade de *sleep* do processador	Executar a instrução de *sleep* e verificar o estado dos sinais externos que indicam se o processador se encontra em modo *sleep*
Lock mechanism	Verificar o correto funcionamento do mecanismo de bloqueio da interface de proteção de código do processador	Tentar alterar a interface de proteção de código com esta bloqueada e verificar que esta não se altera.

Capítulo 4

Metodologia e ferramentas utilizadas

Neste capítulo é apresentada a metodologia seguida durante o projeto, bem como as ferramentas utilizadas para o seu desenvolvimento: verificação funcional, síntese e verificação pós síntese.

A metodologia seguida vai de encontro àquele que é o *design flow* da equipa de desenvolvimento de controladores HDMI/MHL, onde o projeto se insere. As ferramentas utilizadas foram aplicações propriedade da Synopsys, utilizadas em cada uma das fases do projeto de acordo com a metodologia seguida pela equipa.

4.1 Enquadramento

As fases de projeto de um circuito digital são, de forma comum, dividas em dois tipos: o *design Front-End* e o *design Back-End*. Enquanto que o primeiro engloba as tarefas desde a especificação funcional do projeto até à sua síntese numa *netlist* que representa o circuito em portas lógicas, o segundo agrupa as fases desde a *netlist "gate-level"* até à sua translação em *layout*.

O projeto em causa insere-se no *design Front-End*, sendo que os resultados finais são o RTL do projeto e sua síntese numa *netlist "gate-level"*. A Figura 4.1 mostra graficamente as várias fases do fluxo de projeto, enquanto que a Tabela 4.1 apresenta as várias ferramentas utilizadas em cada uma das fases.

Tabela 4.1: Ferramentas utilizadas durante o fluxo de projeto

Fase do *design*	**Ferramentas**
Especificação Funcional	Processador de documentos (por exemplo, MS Word)
Codificação em HDL *Hardware Description Language*	Editor de texto (por exemplo, Vim ou Emacs)
Simulação/Verificação	VCS
Síntese Lógica	Design Compiler
Verificação Formal	Formality
Automatic Test Pattern Generation (ATPG)	Tetramax
Análise Temporal Estática	PrimeTime

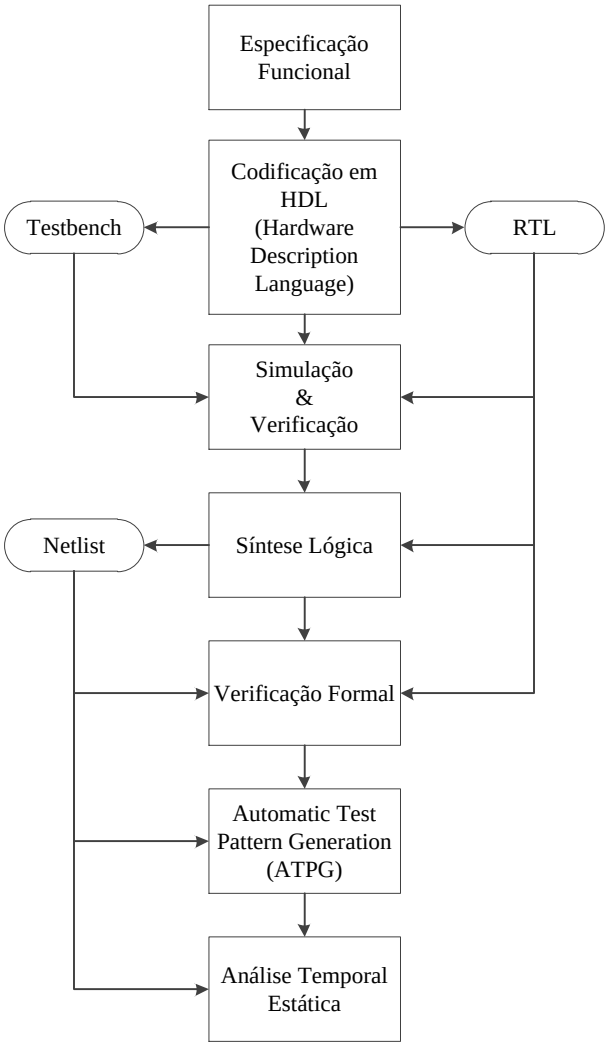


Figura 4.1: Fluxo de projeto para *design Front-End*

4.2 Especificação Funcional

A Especificação Funcional, apresentada no capítulo anterior, tem o intuito de, com base nos casos de uso possíveis para a solução, desenvolver um plano detalhado de como será feita a implementação do bloco de topo e de cada um dos seus sub-blocos constituintes. Este é um documento que serve de apoio durante a implementação em código Verilog.

Este elemento deve especificar o correto funcionamento dos blocos a desenvolver, definindo restrições temporais e funcionais.

4.3 Codificação HDL

Nesta fase pretende-se que os requisitos e a descrição funcional documentados na Especificação Funcional sejam traduzidos em hardware, através de uma linguagem HDL (*Hardware Description Language*).

Na Synopsys, a linguagem de descrição de hardware adotada é o Verilog [8], pelo que foi esta a usada no projeto. Aqui são desenvolvidos o código dos blocos de hardware do projeto, comummente designado como RTL (*Register Transfer Level*), e o *testbench*, responsável por efetuar a bateria de testes especificada no Plano de Verificação ao hardware implementado.

4.4 Simulação/Verificação

Aqui é feita a simulação do bloco de hardware desenvolvido, através da sua simulação instanciada pelo *testbench* elaborado. É objetivo do *testbench* estimular todas as entradas do bloco, que nesta fase é referido como DUT (*Design Under Test*), com todas as combinações lógicas possíveis dos sinais de entrada e que possam vir a ser casos de uso reais.

Nesta fase pretende-se que sejam detetados todos os possíveis erros funcionais, que podem advir de uma má implementação, de casos de uso que não foram contemplados na Especificação Funcional ou de interpretações erradas da mesma por parte do *designer*. Normalmente, quem efetua o teste ao bloco RTL desenvolvido é outra equipa, para que a interpretação dos requisitos não esteja, à partida, viciada. Neste caso, dada a natureza individual do projeto, os testes foram feitos pelo próprio *designer*.

O *testbench* desenvolvido faz auto-teste, isto é, ele próprio estimula o DUT com sinais de entrada, verificando que os sinais de saída são os esperados. Quando algum dos testes não corre como esperado, então a sua depuração pode ser feita recorrendo à análise das formas de onda que forem necessárias.

Como forma de medida da qualidade do *testbench*, é feita a extração da cobertura do código, em inglês designada por *code coverage extraction*. A extração de *coverage* é feita de três formas:

Line coverage
: Este teste calcula a percentagem de linhas de código HDL que foram excitadas durante a simulação.

Condition coverage
: Aqui é calculada a percentagem do total de condições lógicas passíveis de estimular no *design*;

Toggle coverage
: Este tipo de *coverage* verifica qual a percentagem do total de combinações possíveis foram atribuídas aos sinais do bloco.

As ferramentas utilizadas nesta fase do fluxo de projeto são o Synopsys VCS [9] para simulação, e o Synopsys DVE *Discovery Visualization Environment*, para visualização e análise das formas de onda, bem como apresentação dos resultados de *coverage*.

4.5 Síntese Lógica

Nesta fase é feita a síntese do código Verilog, traduzindo o código RTL (*Register Transfer Level*) numa *netlist* onde a solução é definida em termos de portas lógicas. Desta forma, a descrição do projeto passa do nível de registos, combinações lógicas e operações sequenciais para o nível em que o mesmo é descrito em portas lógicas. Daqui advém o nome de "Síntese Lógica".

Ora, como as portas lógicas podem ter diferentes tamanhos, características temporais e de consumo energético, conforme a tecnologia utilizada, nesta fase é escolhida uma tecnologia alvo para o ASIC a sintetizar. No caso deste projeto, foram escolhidas as tecnologias de 40nm e 28nm. Para que a ferramenta de síntese possa compilar o *design* em portas lógicas, é necessário que lhe seja fornecida a biblioteca da tecnologia alvo que contém, para além de portas lógicas elementares, *standard cells* que são células lógicas que agrupam várias portas lógicas formando células que são comummente usadas no desenvolvimento de hardware, como por exemplo *flip-flops*.

Para além do código RTL e da biblioteca da tecnologia alvo, é também necessário fornecer à ferramenta de síntese uma lista das restrições de *design*, relacionadas com requisitos temporais, de área, potência, etc. Esta informação é utilizada para a escolha das *standard cells* mais apropriadas a fim de cumprir os requisitos dados.

Adicionalmente à síntese lógica, nesta fase é feita a inserção automática de cadeias de *scan* que tornam possível a análise da testabilidade dos nós internos. A inserção deste hardware auxiliar permite também que possam ser feitos testes do tipo ATPG (*Automatic Test Pattern Generation*), que é uma das fases posteriores no fluxo de projeto.

A ferramenta utilizada para esta tarefa é o Synopsys Design Compiler [10].

4.6 Verificação Formal

Nesta etapa, é confirmada a equivalência funcional entre a descrição RTL e a *netlist gate-level* gerada pela ferramenta de síntese. Durante este processo, nenhuma simulação é executada. A ferramenta apenas compara cada um dos nós internos do circuito, verificando a correspondência funcional entre as duas descrições.

A ferramenta utilizada nesta fase do projeto é o Synopsys Formality. Caso a verificação não seja concluída com sucesso, é necessário proceder à depuração do mesmo. Para isso, a ferramenta disponibiliza uma interface gráfica onde mostra detalhadamente qual ou quais as células lógicas que não corresponderam, apresentando o esquemático elétrico inferido do RTL e aquele que se esperava ser igual, inferido da *netlist gate-level*.

4.7 Geração Automática de Padrões de Teste

Nesta fase é feito o teste ATPG *Automatic Test Pattern Generation*, através das cadeias de *scan* previamente inseridas durante a síntese lógica. Este tipo de teste, que é feito em simulação, verifica a testabilidade dos nós internos do circuito através da cadeia de *scan*. Assim é possível inferir a percentagem de possíveis erros que podem ser detetados após o fabrico do chip.

Para avaliar a testabilidade de um circuito, são analisados os critérios de observabilidade e controlabilidade. Um nó é controlável se este tomar qualquer valor lógico desejado, "1" ou "0", através de estímulos impostos às entradas do circuito. É considerado observável se o valor lógico desse nó puder ser propagado até uma das saídas do circuito, por forma a poder observar o seu valor.

O teste ATPG baseia-se na geração de padrões de teste aplicados às entradas do circuito com o objetivo de estimular todos os nós internos da solução e propagar o seu valor para as saídas [11]. Desta forma, é avaliada a observabilidade e controlabilidade do circuito, podendo-se detetar falhas do tipo:

stuck-at

Um nó interno fica preso a um determinado valor lógico constantemente. Estas falhas são representadas em modelos comportamentais onde uma falha *stuck-at 0* é representada por um curto-circuito à massa e uma falha *stuck-at 1* por um curto-circuito ao valor da fonte de alimentação. No entanto, estes modelos são apenas comportamentais, podendo a falha ter origem física num outro defeito, por exemplo, um curto-circuito entre dois sinais.

transition delay

Este tipo de falhas ocorre quando uma transição de um dado valor lógico para o outro demora mais do que o previsto durante a síntese, devido a erros de fabrico. Existem dois tipos de faltas possíveis: *slow-to-rise* que modela o atraso da transição de "0" para "1" e *slow-to-fall* que modela o atraso de "1" para "0". A análise da testabilidade destes defeitos é feita em duas fases [12]. A primeira fase utiliza um padrão de entrada para inicializar o valor do nó em análise, por exemplo o valor lógico "0" para detetar uma falta *slow-to-rise*. A segunda fase utiliza um padrão de entrada para propagar a falta até à saída, aplicando o valor lógico contrário ao da inicialização, por exemplo o valor lógico "1" para uma falta do tipo *slow-to-rise*. De notar que o teste apenas verifica a possibilidade de detetar uma falha de atraso pós fabrico, caso esta ocorra, não dando qualquer tipo de indicação acerca do valor desse possível atraso.

A ferramenta utilizada para aplicação dos padrões de teste e avaliação da sua testabilidade é o Synopsys TetraMAX.

4.8 Análise Temporal Estática

Apesar deste tipo de análise já ser efetuada pela ferramenta Synopsys Design Compiler responsável pela Síntese Lógica, nesta fase é utilizada uma ferramenta dedicada a este tipo de análise, que permite obter resultados mais rigorosos. A ferramenta que é utilizada é denominada de Synopsys PrimeTime e é aquela que é usada pela Synopsys para STA (*Static Timing Analysis*), tanto para *design Front-End* como para *design Back-End*.

Uma das diferenças entre o Synopsys Design Compiler e o Synopsys PrimeTime está nos atrasos dependentes da carga. Segundo o manual de utilizador do PrimeTime [13], "Uma das diferenças está nos atrasos dependentes da carga. O Design Compiler, na geração de um relatório temporal originado a partir do início de um caminho segmentado, transfere a porção do atraso dependente da porta lógica de origem para o atraso da "net". Este método pode fornecer melhores resultados de síntese, mas não é o mais apropriado para análise temporal estática." O PrimeTime não adiciona o atraso dependente da carga para a "net".

Nesta fase são analisados todos os caminhos possíveis para os sinais, procurando por violações temporais de *setup*, *hold* ou caminhos demasiados lentos que não permitem cumprir os requisitos temporais definidos antes da síntese lógica.

Capítulo 5

Resultados

Neste capítulo serão apresentados os resultados obtidos após verificação funcional, através da simulação do bloco, bem como após a síntese da solução desenvolvida. A solução foi sintetizada para ASIC nas tecnologias de 40nm e 28nm e para FPGA. A FPGA alvo utilizada foi uma Xilinx Virtex-7, versão XC7VX690T.

A metodologia utilizada para a verificação funcional, para a síntese em ASIC, bem como para a sua verificação e teste, seguiu o *design flow* apresentado no capítulo "Metodologia e ferramentas utilizadas".

5.1 Relatório de Verificação

Aqui é apresentado o Relatório de Verificação com os resultados do *testbench* e extração da cobertura de código (*Code Coverage Extraction*) verificado.

5.1.1 APB Bridge

De seguida são apresentados os testes efetuados à *bridge* APB , a metodologia utilizada e os resultados obtidos.

Teste

Verificar a correta operação quando são colocados endereços de registos inexistentes.

Metodologia

São feitas escritas ou leituras de forma aletória de um valor aleatório, de 32 bits, para todo o espaço endereçamento vazio. Para cada uma das escritas/leituras, é verificado se a *bridge* coloca o sinal APB PSLVERR no valor lógico "1", demonstrando assim a invalidade da transferência desejada.

Resultado

Passou no teste sem qualquer erro.

5.1.2 Banco de registos

Aqui são apresentados os testes efetuados a fim de avaliar especificamente o correto funcionamento dos bancos de registos.

Teste

Comparar a implementação do banco de registos com o mapeamento planeado na Especificação Funcional.

Metodologia

Verificação feita através da comparação visual entre as tabelas de mapeamento dos registos e o código RTL desenvolvido correspondente.

Resultado

Verificada a correspondência.

Teste

Verificar a geração dos sinais de controlo *Halt/Run* através da manipulação do banco de registos.

Metodologia

Escrita nos registos correspondentes à ativação do estado de *Halt* e *Run* e verificação da alteração do seu estado nos registos de estado dos mesmos.

Resultado

Verificação bem sucedida.

5.1.3 *Interrupt Handler*

Apresentam-se de seguida os testes realizados para avaliar o correto funcionamento dos módulos responsáveis pelo tratamento das interrupções.

Teste

Comparar a implementação do bloco de interrupções com o mapeamento planeado na Especificação Funcional.

Metodologia

Verificação feita através da comparação visual entre as tabelas de mapeamento dos registos e o código RTL desenvolvido correspondente.

Resultado

Verificada a correspondência.

5.1.4 *APB 2 AHB-Lite*

Aqui são apresentados os testes efetuados para avaliar o funcionamento da *bridge APB 2 AHB-Lite* desenvolvida especificamente para este projeto.

Teste

Verificar o funcionamento normal com a frequência APB maior que a frequência de CPU.

Metodologia

Todas as simulações foram testadas com a frequência APB igual a 40MHz e a frequência de CPU igual a 14MHz.

Resultado

Verificou-se o correto funcionamento de todas as operações efetuadas durante a simulação como, por exemplo, o carregamento da memória ICCM através que passa por este bloco.

Teste

Verificar o funcionamento normal com a frequência de CPU maior que a frequência APB.

Metodologia

Todas as simulações foram testadas com a frequência APB igual a 14MHz e a frequência de CPU igual a 40MHz.

Resultado

Verificou-se o correto funcionamento de todas as operações efetuadas durante a simulação como, por exemplo, o carregamento da memória ICCM através que passa por este bloco.

Teste

Verificar transferências de escrita e leitura sequenciais, sem desativar o sinal de leitura (back to back transactions).

Metodologia

As memórias ICCM e DCCM foram escritas sequencialmente na sua totalidade, verificando-se, após cada escrita, se a leitura do mesmo endereço retornava o valor previamente escrito (aleatório).

Resultado

Verificou-se o correto funcionamento das transações de escrita e leitura sequenciais.

5.1.5 Memórias ICCM/DCCM

Para avaliar o correto funcionamento da interface de memória para instruções (ICCM) e dados (DCCM), foram feitos os seguintes testes:

Teste

Verificar a correta escrita e leitura nas memórias ICCM e DCCM.

Metodologia

As memórias ICCM e DCCM foram escritas sequencialmente na sua totalidade, verificando-se, após cada escrita, se a leitura do mesmo endereço retornava o valor previamente escrito (aleatório).

Resultado

Verificou-se o correto funcionamento do acesso às memórias.

Teste

Verificar a funcionalidade de proteção de código. Ativar esta funcionalidade e verificar o seu correto funcionamento através de tentativas de leitura e escrita.

Metodologia

A memória ICCM, onde esta funcionalidade será utilizada, foi escrita/lida sequencialmente, verificando-se que a leitura sempre retornou o valor "0", tal como esperado.

Resultado

Verificou-se o correto funcionamento do mecanismo de proteção de código.

5.1.6 *AHB-Lite 2 APB*

Os seguintes testes foram feitos para avaliar o funcionamento da *bridge AHB-Lite – APB* incorporada no *design*.

Teste

Verificar a correta operação quando são colocados na interface endereços de registos inexistentes.

Metodologia

Para verificar este teste, foram feitas leituras e escritas de endereços fora da região onde se encontram mapeados os periféricos CBUS.

Resultado

Verificou-se com através da observação do sinal AHB-lite HRESP que as transações desencadeavam erros na interface AHB-lite , devido a esta estar configurada para apenas um escravo APB na região de memória especificada para os periféricos CBUS.

5.1.7 *ARC EM4*

Os seguintes testes foram feitos para avaliar o funcionamento da *bridge AHB-Lite – APB* incorporada no *design*.

Teste

Verificar o correto funcionamento da interface de periféricos.

Metodologia

Escrita de software de teste para efetuar escritas e leituras através da interface de periféricos sequencialmente.

Resultado

Através da visualização das formas de onda na interface AHB-lite e APB após a respetiva *bridge*, verificou-se o correto funcionamento do acesso aos periféricos CBUS.

Teste

Verificar o correto funcionamento da interface de interrupções.

Metodologia

Escrita de software de teste com a inclusão de uma função responsável pelo tratamento da interrupção que faz uma determinada escrita na interface de periféricos após a ocorrência de uma interrupção externa.

Resultado

Após o estimulo de interrupção na linha dedicada do processador ARC , verificou-se a escrita na interface dos periféricos despoletada pela rotina de atendimento da interrupção.

Teste

Verificar a funcionalidade de *sleep* do processador.

Metodologia

Executar a instrução de *sleep* e verificar o estado dos sinais externos que indicam se o processador se encontra em modo *sleep*.

Resultado

Verificação feita com sucesso.

5.1.8 *Lock mechanism*

De seguida mostra-se a metodologia usada e o resultado obtido com o teste ao mecanismo de bloqueio da interface de proteção de código da memória de instruções.

Teste

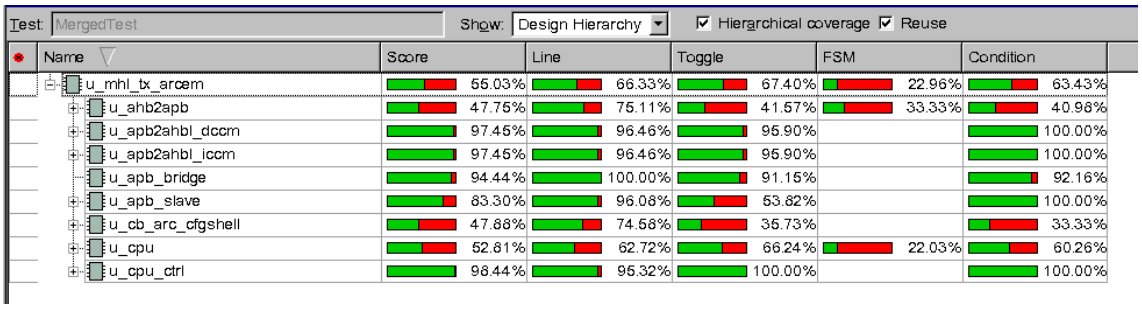
Verificar o correto funcionamento do mecanismo de bloqueio da interface de proteção de código do processador.

Metodologia

Tentar alterar a interface de proteção de código com esta bloqueada e verificar que esta não se altera.

Resultado

Verificação executada com sucesso.



Name	Score	Line	Toggle	FSM	Condition
u_mhl_tx_arcem	55.03%	66.33%	67.40%	22.96%	63.43%
u_ahb2apb	47.75%	75.11%	41.57%	33.33%	40.98%
u_apb2ahbl_dccm	97.45%	96.46%	95.90%		100.00%
u_apb2ahbl_iccm	97.45%	96.46%	95.90%		100.00%
u_apb_bridge	94.44%	100.00%	91.15%		92.16%
u_apb_slave	83.30%	96.08%	53.82%		100.00%
u_cb_arc_cfgshell	47.88%	74.58%	35.73%		33.33%
u_cpu	52.81%	62.72%	66.24%	22.03%	60.26%
u_cpu_ctrl	98.44%	95.32%	100.00%		100.00%

Figura 5.1: Resultado da extração de *coverage*

5.1.9 *Code Coverage Extraction*

A Figura 5.1 demonstra os resultados de *coverage* que foram obtidos com o *testbench* desenvolvido.

O módulo "u_ahb2apb" representa a instância da *bridge* "AHB-Lite 2 APB" e o seu *coverage* global apresenta-se abaixo dos 50% devido ao bloco ser utilizado quando o software do processador ARC acede à interface de periféricos e este apenas efetuar escritas ou leituras simples (não em modo *burst*). Desta forma é impossível aumentar o *coverage*, visto que não existem casos de uso reais suficientes para tal.

Os módulos "u_apb2ahbl_dccm" e "u_apb2ahbl_iccm" são ambos instâncias da *bridge* "APB 2 AHB-Lite", daí terem a mesma percentagem de *coverage*, que está relacionado com os testes à memória ICCM e DCCM serem idênticos. A percentagem não se encontra nos 100% devido ao *toggle coverage* detetar que não são estimuladas todas as combinações do endereço, como era de esperar, pois antes é feita uma filtragem e a estes blocos já só são requeridas transferências para o espaço de endereçamento das memórias ICCM e DCCM, respetivamente.

O módulo "u_apb_bridge" é a instância da *bridge* inicial APB que faz o redirecionamento para a memória ICCM, DCCM e *ARC EM Control*. Esta não apresenta um *toggle coverage* de 100% devido ao *slave* APB da interface de controlo do processador não implementar o sinal de PSLVERR, mantendo este constantemente no nível lógico "0".

O bloco "u_apb_slave" representa uma instância de um *slave* APB . Este não apresenta um *line coverage* de 100% devido a células de sincronização de sinais que contém funcionalidades não utilizadas nesta aplicação. O *toggle coverage* está relacionado com alguns registos que não podem ser escritos ou lidos e que influenciam os sinais de dados e endereço a ficar com bits constantemente ao nível lógico "0".

O bloco "u_cb_arc_cfgshell" contém um banco de registos e um bloco de tratamento de interrupções relacionados com o controlo do processador ARC . Este não apresenta um *coverage* de 100% devido a alguns registos que não podem ser escritos ou lidos e que influenciam os sinais de dados, endereço e controlo de interrupções a ficar com bits constantemente ao nível lógico "0".

O módulo "u_cpu" é a instância do processador ARC EM4. Visto este ser um bloco complexo, seria impensável estimular todo o seu RTL para obter um *coverage* de 100%, devido aos casos

de uso necessários que não seriam reais de acordo com a aplicação do projeto. De qualquer das formas, este não é um fator crítico, visto este processador ter sido desenvolvido independentemente e o seu correto funcionamento estar assegurado à partida.

O módulo "u_cpu_ctrl" é um módulo que encapsula a lógica responsável pelo controlo do funcionamento do processador ARC , nomeadamente a interface *Halt/Run* e o mecanismo de proteção de código e seu bloqueio após *reset*. Este não apresenta um *line coverage* de 100% devido a células de sincronização de sinais que contém funcionalidades não utilizadas nesta aplicação.

5.2 Síntese para ASIC em 40nm

5.2.1 Síntese Lógica

A Síntese Lógica para a tecnologia de 40nm foi efetuada com recurso à ferramenta Synospsys Design Compiler e os resultados obtidos encontram-se expressos na Tabela 5.1

Tabela 5.1: Resultados obtidos pela Síntese Lógica para ASIC em 40nm

Parâmetro	**Valor**
REF clock	40 MHz
CPU clock	100 MHz
APB clock	40 MHz
Área do design	48725 portas lógicas[2]
Consumo de potência (estimado)	1.4202 mW

De notar que nesta fase foi inserida automaticamente pela ferramenta uma cadeia de *scan* para permitir o teste ATPG.

Nesta fase, foram ignoradas algumas violações de tempo de *hold*, de forma consciente. A razão é a destas violações serem resolvidas na fase de *Back-End* do *design* onde é feito o *Place and Route* das células, inserindo os *buffers* necessários para atrasar sinais que possam estar a violar restrições de *hold time*.

5.2.2 Verificação Formal

A Verificação Formal foi efetuada com a ferramenta Synopsys Formality que conseguiu fazer a correta correspondência entre a solução desenvolvida em código RTL e a sua *netlist gate-level* gerada após a síntese lógica.

5.2.3 Geração Automática de Padrões de Teste

Durante o teste ATPG, foi avaliada a controlabilidade e observabilidade do circuito desenhado.

[2]Na representação de área em unidades de "portas lógicas", uma porta lógica equivale à menor porta lógica NAND de duas entradas da tecnologia em questão

Relativamente às faltas do tipo *stuck-at*, o teste permitiu obter um *coverage* de 99,88%, que é considerado ser um bom resultado (acima de 99%), sendo que os nós não cobertos estão relacionados com pontos de sincronização de sinais. Em relação a faltas do tipo *transition delay faults*, foi obtido um *coverage* de 95,03%, considerado também um bom resultado pela equipa da Synopsys.

5.2.4 Análise Temporal Estática

Após a análise temporal estática (*Static Timing Analysis*), esta não retornou qualquer tipo de violações. De notar que nesta fase, a ferramenta descartou possíveis violações no *hold time*, visto que este problema é tratado apenas na fase de *Back-End* do projeto.

5.3 Síntese para ASIC em 28nm

5.3.1 Síntese Lógica

A Síntese Lógica para a tecnologia de 28nm foi efetuada com recurso à ferramenta Synospsys Design Compiler e os resultados temporais encontram-se expressos na Tabela 5.2

Tabela 5.2: Resultados obtidos pela Síntese Lógica para ASIC em 28nm

Parâmetro	**Valor**
REF clock	40 MHz
CPU clock	100 MHz
APB clock	40 MHz
Área do design	46347 portas lógicas[2]
Consumo de potência (estimado)	0.4537 mW

De notar que nesta fase foi inserida automaticamente pela ferramenta uma cadeia de *scan* para permitir o teste ATPG.

Nesta fase, foram ignoradas algumas violações de tempo de *hold*, de forma consciente. A razão é a destas violações serem resolvidas na fase de *Back-End* do *design* onde é feito o *Place and Route* das células, inserindo os *buffers* necessários para atrasar sinais que possam estar a violar restrições de *hold time*.

5.3.2 Verificação Formal

A Verificação Formal foi efetuada com a ferramenta Synopsys Formality que conseguiu fazer a correta correspondência entre a solução desenvolvida em código RTL e a sua *netlist gate-leve* gerada após a síntese lógica.

5.3.3 Geração Automática de Padrões de Teste

Durante o teste ATPG, foi avaliada a controlabilidade e observabilidade do circuito desenhado.

Relativamente às faltas do tipo *stuck-at*, o teste permitiu obter um *coverage* de 99,89% (acima de 99%), que é considerado um bom resultado, sendo que os nós não cobertos pelo teste dizem respeito a pontos de sincronização de sinais. Em relação a faltas do tipo *transition delay faults*, foi obtido um *coverage* de 95,28%, considerado um bom resultado pela equipa da Synopsys.

5.3.4 Análise Temporal Estática

Após a análise temporal estática (*Static Timing Analysis*), esta não retornou qualquer tipo de violações. De notar que nesta fase, a ferramenta descartou possíveis violações no *hold time*, visto que este problema é tratado apenas na fase de *Back-End* do projeto.

5.4 Síntese para FPGA

Para além da síntese em ASIC, foi também feita a síntese para FPGA com o intuito de verificar que a mesma era possível e quais as frequências de relógio atingíveis. A ferramenta utilizada para a compilação e mapeamento é o Synopsys Synplify. Para *Place & Route* foi utilizado o Xilinx Vivado.

A Tabela 5.3 mostra as frequências atingidas com a síntese e apresenta os recursos da FPGA utilizados pela solução. A solução demonstrou-se assim ser sintetizável para a FPGA utilizada para efeitos de teste pela equipa da Synopsys, tendo sido atingidas as frequências de operação necessárias ao correto funcionamento não só do bloco desenvolvido, como do core transmissor MHL onde este se integra.

Tabela 5.3: Resultados obtidos em Síntese pós *Place and Route* para a FPGA Virtex-7

Parâmetro	**Valor**
REF clock	40 MHz
CPU clock	100 MHz
APB clock	40 MHz
Register bits	3855 (~0%)
Lookup tables	7631 (~1%)
Blocos de RAM	512/1470 (~35%)

Capítulo 6

Conclusões e Trabalho Futuro

Neste capítulo são apresentadas as conclusões do trabalho, onde são expostos os resultados obtidos e as suas contribuições, retirando ilações sobre a satisfação dos objetivos propostos. Aqui são também propostos tópicos para trabalho futuro, na sequência daquilo que foi desenvolvido durante esta dissertação.

6.1 Satisfação dos Objetivos

Com o desenvolvimento do projeto desta dissertação, os objetivos propostos inicialmente foram cumpridos. A integração do processador ARC EM na solução DesignWare MHL TX Interface IP foi feita com sucesso, tendo sido desenvolvido o seguinte:

- Implementação da interface entre o processador integrado ARC EM4 e o processador principal do SoC, para que este último possa controlar o funcionamento do processador integrado, bem como trocar mensagens com ele, relacionadas com a implementação da *Translation Layer* do protocolo CBUS;

- Implementação da interface entre o processador integrado ARC EM4 e os periféricos CBUS, por forma a permitir a implementação da *Translation Layer* e sua interação com a *Link Layer*;

- Desenvolvimento dos blocos de controlo do processador ARC EM4, relacionados com a sua configuração, modo de operação e geração de interrupções.

Para além dos objetivos propostos alcançados, foi ainda desenvolvido *firmware* para o processador ARC EM4, que permitiu verificar:

- A correta inicialização do processador, através do desenvolvimento de um *bootloader* compilado num ficheiro binário a ser descarregado para a memória de instruções através da interface de acesso direto à memória;

- A correta operação da interface de periféricos, diretamente mapeada em memória, que permite o acesso aos periféricos CBUS, por forma a implementar a *Translation Layer* em software;
- O atendimento da interrupção externa por parte do processador ARC EM4 integrado, através do desenvolvimento da respetiva rotina de atendimento em software.

Através dos resultados obtidos apresentados, conclui-se que os objetivos propostos foram alcançados e superados através do desenvolvimento dos blocos de hardware necessários, mas também de pedaços de software que permitem validar a solução e que servem como base para a implementação da *Translation Layer*.

Desta forma, a Synopsys poderá integrar o bloco desenvolvido na sua solução atual por forma a oferecer uma solução mais completa através da implementação interna de todas as camadas protocolares do CBUS. Para além do ambiente CBUS MHL, o módulo desenvolvido pode ser facilmente adaptado para ser integrado em outras soluções de arquitetura semelhante.

6.2 Trabalho Futuro

Como trabalho futuro, encontra-se a necessidade de desenvolver o *firmware* responsável pela implementação da *Translation Layer* do protocolo CBUS no processador ARC EM4. O *firmware* desenvolvido poderá basear-se nas rotinas desenvolvidas durante o projeto que validaram funções específicas interligadas ao hardware.

A reserva de regiões de memória de dados para troca de mensagens entre o processador principal do sistema e o processador ARC EM4 é também uma tarefa a realizar futuramente, para assegurar que o programa contido no processador não faz escritas ou leituras nessas regiões de memória indevidamente.

A solução poderá ser testada noutros ambientes a fim de avaliar a sua fácil integração em sistemas com arquiteturas semelhantes: onde existe uma interface com um processador do sistema que o controla e um conjunto de periféricos a serem controlados de forma dedicada por um outro processador integrado na solução desenvolvida.

A síntese da solução poderá ainda ser feita em outros modelos de FPGA, por forma a validar e comparar a performance da solução para diferentes tecnologias.

Referências

[1] HDMI Consortium. High-Definition Multimedia Interface Specification Version 1.4b, Outubro 2011. CONFIDENTIAL.

[2] MHL Promoters. Mobile High-Definition Link Specification Version 2.2, Janeiro 2014. CONFIDENTIAL.

[3] Chi-Cheng Ju, Tsu-Ming Liu, Huaide Wang, Yung-Chang Chang, Chih-Ming Wang, Chang-Lin Hsieh, B. Liu, Hue-Min Lin, Chia-Yun Cheng, Chun-Chia Chen, Min-Hao Chiu, Sheng-Jen Wang, Ping Chao, M.J. Hu, R. Yeh, T. Chuang, Hsiu-Yi Lin, e Chung-Hung Tsai. A 4k #x00d7;2k@60fps multi-standard TV SoC processor with integrated HDMI/MHL receiver. Em *2014 Symposium on VLSI Circuits Digest of Technical Papers*, páginas 1–2, Junho 2014. doi:10.1109/VLSIC.2014.6858389.

[4] Synopsys, Inc. DesignWare ARC EM Databook, Agosto 2014. CONFIDENTIAL.

[5] ARM. AMBA™ 3 AHB-Lite Protocol, Junho 2006. URL: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0033a/index.html.

[6] ARM. AMBA™ 3 APB Protocol, Agosto 2004. URL: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0024b/index.html.

[7] Synopsys, Inc. DesignWare DW_apb Databook, Junho 2014. CONFIDENTIAL.

[8] IEEE Standard for Verilog Hardware Description Language. *IEEE Std 1364-2005 (Revision of IEEE Std 1364-2001)*, páginas 0_1–560, 2006. doi:10.1109/IEEESTD.2006.99495.

[9] Synopsys, Inc. VCS®/VCSi™ User Guide Version J-2014.12-SP1, Março 2015.

[10] Synopsys, Inc. Design Compiler® User Guide Version J-2014.09, Setembro 2014.

[11] *Electronic Design Automation for Integrated Circuits Handbook*. Industrial Information Technology. CRC Press, Abril 2006. URL: http://www.crcnetbase.com/doi/book/10.1201/NOE0849330964.

[12] J.A. Waicukauski, E. Lindbloom, Barry K. Rosen, e V.S. Iyengar. Transition Fault Simulation. *IEEE Design Test of Computers*, 4(2):32–38, Abril 1987. doi:10.1109/MDT.1987.295104.

[13] Synopsys, Inc. PrimeTime® User Guide Version J-2014.12-SP2, Março 2015.