Francisco Alves Fonseca de Oliveira Silveira

Licenciado em Ciências de Engenharia
Electrotécnica e de Computadores

Aplicações das IOPT Tools em plataformas System-on-Chip

Dissertação para obtenção do Grau de Mestre em
Engenharia Electrotécnica e de Computadores

Orientador: Doutora Anikó Katalin Horváth da Costa, Professora Auxiliar da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Júri:

Presidente: Doutor Paulo da Costa Luís da Fonseca Pinto, Professor Catedrático da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Arguente: Doutor Rogério Alexandre Botelho Campos Rebelo, Investigador do Centro de Tecnologias e Sistemas, Uninova – Instituto de Desenvolvimento de Novas Tecnologias

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2017

Utilização das IOPT Tools em plataformas System-on-Chip

Para a minha família e amigos!

Agradecimentos

A conclusão desta dissertação de mestrado assinala a realização de uma etapa importante para mim, pelo que gostaria de agradecer a todos os que, de forma direta ou indireta me ajudaram a concretizar este objetivo.

Em primeiro lugar gostaria de agradecer à Professora Doutora Anikó Costa pela oportunidade de orientação e disponibilidade dedicada, pelos desafios que me propôs, pelo apoio material e técnico e pela motivação ao longo do desenvolvimento da dissertação.

Ao Doutor Rogério Campos Rebelo que sempre se mostrou disponível para a partilha de conhecimentos, pela boa disposição no laboratório de investigação e pela orientação ao longo do estágio curricular.

Quero agradecer também a todos os meus amigos que me deram força para concluir esta etapa e a partilha de bons momentos.

Um especial agradecimento à minha família, pela oportunidade e privilégios que me ofereceram e pelos incentivos e a paciência ao longo destes anos.

Resumo

No âmbito da conceção de sistemas digitais é importante o desenvolvimento de ferramentas para proporcionar maior eficiência no processo de elaboração do sistema. As IOPT-Tools permitem modelar e validar os controladores digitais, com a possibilidade de gerar os códigos C e VHDL de forma automática, o que acelera as fases de verificação e implementação dos controladores.

O problema identificado reside na utilização das ferramentas IOPT-Tools para co-design de sistemas, que se pretendam implementar em plataformas System-on-Chip. É um problema interessante do ponto de vista da engenharia porque envolve a utilização de uma metodologia baseada em modelos para o desenvolvimento de sistemas co-design que se pretendem embutir num só dispositivo. Para validar a metodologia referida definiu-se o desenvolvimento de uma aplicação que requer a interoperabilidade entre a partição hardware e software. Nomeadamente, o desenvolvimento de um rádio definido por software com base em modelos de modulação QPSK disponíveis no Matlab, e a interacção com o utilizador por uma aplicação web.

Portanto, o sistema consiste num controlador co-design para definir a configuração da partição hardware e controlar o funcionamento do processador de sinais digitais com a afectação dos sinais de entrada pela aplicação web. Desta forma, possibilita ao utilizador a escolha da configuração da FPGA, sendo possivel escolher a implementação do sistema referente ao emissor, recetor, ou transcetor.

Palavras-chave: IOPT Tools, co-design, Zynq, Software Defined Radio, Red Pitaya, Vivado, Matlab

Abstract

In the field of digital systems conception the tools development for such systems is very important, once it can increase the development efficiency of these. The IOPT-Tools allow to model and validate digital controllers, besides the capability to automaticaly generate the controller C and VHDL code. So, these tools may accelarate the verification and implementation phases of the developed controllers.

The identified problem resides in the utilization of the IOPT-Tools in systems co-design wich are intended to implement into System-on-Chip platforms. It's an interesting problem by the engineering point of view, once it embraces a model based methodology to co-design systems developing wich are intended to implement in a single chip. In order to validate the mentioned methodology, an application wich requires interoperability between software and hardware partitions was defined. Specifically, the development of a software defined radio based in models of QPSK modulation available in Matlab, and its interaction via web application.

Therefore, the defined system consists in a co-design controller wich can define the configuration of the hardware and controls the digital signal processor wich inputs are affected by the user from the web application. Thus, the user can choose the FPGA configuration that is implemented, being available to choose between the implementations of the system that can be the use of the emitter, the use of receiver, or both as a transceiver.

Keywords: IOPT-Tools, co-design, Zynq, Software Defined Radio, Red Pitaya, Vivado Matla

Conteúdos

Lista de Figuras

Lista de Tabelas

Notações e simbologias

ADC – Analog Digital Converter
ARM – Advanced RISC Machine
ASK – Amplitude Shift Keying
AXI - Advanced eXtensible Interface
BPSK – Binary Phase Shift Keying
CLK – Clock
CPU – Central Processing Unit
DAC – Digital Analog Converter
DSP – Digital Signal Processor
EDK – Embedded Development Kit
ESL – Electronic System Level
FFT – Fast Fourier Transform
FIR – Finite Impulse Response
FMC – FPGA Mezzanine Card
FPGA – Field Programmable Gate Array
FSK – Frequency Shift Keying
GPIO – General Purpose Input/Output
GPP – General Purpose Processor
HDMI – High-Definition Multimedia Interface
IoT – Internet of Things
IP – Intelectual Property
I2C – Inter-Integrated Circuit

JTAG – Joint Test Action Group

MIMO – Multi Input Multi Output

MISO – Multi Input Single Output

MOSI – Multi Output Single Input

NVRAM – Non-Volatile RAM

OFDM – Orthogonal Frequency Division Multiplexing

OLED – Organic Light-Emitting Diode

OSCI – Open SystemC Initiative

OTG – On-The-Go

PLA – Programable Logic Array

QAM – Quadrature Amplitude Modulation

QPSK – Quadrature Phase Shift Keying

RAM - Random Access Memory

RdP – Rede(s) de Petri

RFIC – Radio Frequency Integrated Circuit

RISC – Reduced Instruction Set Computer

RTL – Register Transfer Level

RX – Receiver

SCL – Serial Clock Line

SD – Secure Digital

SDA – Serial Data Line

SDR – Software Defined Radio

SDRAM – Synchronous Dynamic RAM

SoB – System-on-Board

SoC – System-on-Chip

SPI – Serial Peripheral Interface

SPU – Specialized Processor Unit

SSH – Secure Shell

TLM – Transaction Level Modeling

TX – Transmitter

UART – Universal Asynchronous Receiver/Transmitter

USB – Universal Serial Bus

UML – Unified Modeling Language

USRP – Universal Software Radio Peripheral

VGA – Video Graphcs Array

XADC – Xilinx Analog to Digital Converter

1

1 Introdução

1.1 Motivação

As IOPT Tools viabilizam o desenvolvimento de controladores digitais hardware/software e co-design, utilizando uma metodologia baseada em modelos. A utilização destas ferramentas permite a redução de tempo e custos no processo de desenvolvimento de protótipos. As funcionalidades de edição e validação de modelos, geração de espaço de estados e geração de código automático, facilitam a implementação de sistemas nos dispositivos físicos.

Nesta dissertação pretende-se dar ênfase à utilização das IOPT Tools para o desenvolvimento de aplicações orientadas a plataformas System-on-Chip, nomeadamente, a plataforma Red Pitaya. Com o estudo sobre as aplicações das FPGA (Field Programable Gate Array) em sistemas embutidos identificaram-se as áreas de *Machine Learning*, *5G Wireless*, *Embedded Vision*, *Industrial IoT* e *Cloud Computing*. Portanto, verificou-se que uma das principais áreas é no sentido da evolução de dispositivos ligados às telecomunicações. Neste sentido, surgiu o interesse no desenvolvimento de um SDR (*Software-Defined Radio)* e consequentemente definiu-se a implementação de um transcetor com modulação QPSK (Quadrature Phase Shift Keying) como objetivo de estudo.

Este tipo de sistemas tem sido desenvolvido considerando a implementação em software, o que pode comprometer as capacidades de processamento e de adaptabilidade necessárias. No entanto com a utilização das FPGAs podem implementar-se estes sistemas de forma híbrida, isto é, implementando uma parte do sistema em hardware e outra em software. Sendo assim, para a parte do sistema com maior necessidade de processamento de dados utiliza-se a partição

hardware, e para a parte do sistema para interacção com o utilizador utiliza-se a partição software.

1.2 Objetivos

A composição do sistema escolhida para utilização do SDR baseia-se na interação do utilizador por uma aplicação web e um controlador em co-design para controlar o funcionamento do transcetor e escolher quais os módulos de processamento de sinais digitais que estão em funcionamento dinâmicamente. Considerando as necessidades da capacidade de processamento de sinais definiu-se a implementação do DSP (Digital Signal Processor) e parte do controlador em hardware e a implementação da aplicação web e parte do controlador em software.

Para desenvolver a aplicação web pretende-se utilizar como base as aplicações disponíveis na plataforma Red Pitaya, para que sejam utilizados os mesmos standards de comunicação entre o cliente e servidor. Neste sentido, também viabiliza a partilha da aplicação desenvolvida com os investigadores e entusiastas da utilização da plataforma Red Pitaya. Para o controlador co-design pretende-se utilizar as IOPT Tools, nas quais se integram ferramentas para o desenvolvimento de controladores orientados a sistemas embutidos.

O desenvolvimento dos componentes a integrar no DSP pretende-se utilizar a ferramenta Matlab. A escolha desta ferramenta (Matlab) é devida à existência de bibliotecas de funções e módulos de processamento de sinais digitais optimizados para que possam ser geradas as descrições de hardware em VHDL. Posteriormente, segue-se a implementação do DSP e do controlador co-design no dispositivo. Com recurso à ferramenta Vivado pretende-se validar o funcionamento do sistema a implementar em hardware e gerar o *bitstream* para configurar a FPGA. A utilização do sistema operativo Linux na partição software do Zynq permite escolher a configuração da FPGA em funcionamento, executar a parte do controlador IOPT que se pretende que funcione em software e implementar a aplicação web. Pretende-se também explorar a utilidade da funcionalidade de reconfiguração parcial dinâmica do Zynq no contexto da aplicação.

1.3 Estrutura do documento

Estruturou-se a dissertação em cinco capítulos nos quais se inclui a introdução para que seja claro o objetivo de estudo. Apresenta-se de seguida uma breve descrição dos capítulos que se seguem.

No segundo capítulo são apresentados os conceitos e tecnologias que são necessárias para o desenvolvimento da aplicação. É apresentado o formalismo de modelação redes de Petri e a classe IOPT utilizadanas IOPT Tools. São apresentadas as funcionalidades das IOPT Tools que se pretende explorar para o desenvolvimento do controlador. De seguida é apresentado o conceito de plataformas System-on-Chip seguido de duas plataformas deste tipo, nomeadamente a Red Pitaya e a ZedBoard. Apresentam-se de forma sucinta as linguagens de descrição de hardware e de programação utilizadas, seguidos de alguns exemplos de aplicações que integram o conceito de SDR.

O terceiro capítulo é iniciado com uma descrição geral da aplicação, à qual se segue a arquitectura completa do sistema modelado com as IOPT Tools. Ao longo do capítulo é apresentado o desenvolvimento de cada parte do sistema assim como a respetiva validação com as ferramentas de simulação associadas.

No quarto capítulo são apresentados resultados da aplicação em funcionamento utilizando a interface gráfica desenvolvida, comparando aos resultados das simulações dos geradores de sinais no Matlab e no Vivado. Além disso, é apresentada uma discussão face à reconfiguração dinâmica de FPGAs no âmbito da aplicação desenvolvida.

Para concluir o documento, segue-se o quinto capítulo onde se apresentam as conclusões do projeto, algumas ideias para melhorias e possíveis trabalhos futuros.

2

2 Conceitos e tecnologias

2.1 Redes de Petri não autónomas - classe IOPT

Como formalismo de modelação pretende-se dar ênfase às redes de Petri para modelar sistemas que utilizem a ideologia de controlar o comportamento e/ou fluxo dos sistemas.

A elaboração do conceito de Rede de Petri (RdP) tem origem na Alemanha, mais concretamente na Technical University of Darmstadt, com a divulgação da dissertação de C. A. Petri. A sua tese de doutoramento teve como principal objetivo desenvolver uma solução para computação de funções recursivas, na prática o interesse é de agregar novos componentes ao sistema sem que o torne mais lento. A solução mencionada em (Brauer & Reisig, 2009) passa por agregar o componente novo ao último componente agregado e com a condição de que qualquer componente do sistema deve atuar de forma autónoma e que o sistema completo atua de forma assíncrona.

Segundo (Murata, 1989) a representação gráfica, torna a modelação e análise da rede de Petri bastante simples, o que simplifica o desenvolvimento de um projeto de controladores digitais nas diferentes fases de modelação, verificação, validação ou descrição compacta de sistemas. Por sua vez, para (Peterson, 1977) e (Murata, 1989), as características dos sistemas em que as RdP se tornam essencialmente aplicáveis são a computação paralela de múltiplos processos concorrentes, sistemas de controlo distribuídos, sistemas assíncronos ou processos estocásticos.

As redes de Petri não autónomas são uma extensão do modelo clássico constituído por lugares e transições, com a possibilidade de incluir condições ao disparo das transições, avali-

ando valores dos sinais e disparo de eventos de entrada e saída. Os sinais de saída podem ser modelados por duas formas, associadas à máquina de Moore ou à máquina de Mealy.

Assim, no caso de se associar à máquina de Moore, os sinais de saída estão associados à permanência da marcação no lugar, assim quando o lugar deixa de estar marcado o sinal de saída é interrompido. No caso de se associar à máquina Mealy, os eventos de saída são produzidos ao disparo da transição associada, mantendo assim o sinal de saída ativo até que haja o disparo do mesmo evento com o sinal contrário.

As características da classe de RdP IOPT (*input-output place-transition)* são identificadas em (Gomes, Barros, Costa, & Nunes, 2007). Ao longo do artigo mencionado é justificada a utilização deste formalismo para modelar sistemas de automação e sistemas embutidos. A classe é uma extensão da classe lugar-transição em que é permitido modelar o controlador e as interações físicas do ambiente em que se pretende inserir o controlador. Esta classe das redes de Petri é classificada como não autónoma pelas razões anteriormente mencionadas.

A classe IOPT é caracterizada essencialmente pelos sinais de entrada e de saída e por eventos associados aos sinais ou eventos autónomos. Os sinais ou eventos de entrada estão associados ao disparo de transições, e os sinais ou eventos de saída estão associados às marcações dos lugares, disparo de transições ou ambos. Os sinais devem formar um conjunto finito e podem ser descritos como sinais booleanos ou sinais do tipo inteiro com os limites superior e inferior definidos.

Podem-se associar os sinais de entrada a interruptores manipulados por um agente de forma presencial ou remota, à detecção de um valor crítico indicado por sensores presentes no sistema ou sinais de comunicação provenientes de outros controladores.

Os sinais de saída produzidos pelo controlador estão associados por norma ao acionamento de atuadores presentes no sistema como o movimento de um braço robótico ou o movimento de tapetes rolantes. Também podem ser associados à emissão de alertas num ambiente fabril ou sinais de comunicação para outros sistemas ou subsistemas.

Os eventos discretos associados aos sinais de entrada são utilizados para identificar alterações nos sinais de entrada. Quando são verificadas alterações aos sinais de entrada, como um sinal booleano em que o seu valor é alterado de falso para verdadeiro, então o evento é disparado no instante da alteração do sinal.

Além dos eventos associados aos sinais do sistema, podem ser definidos eventos autónomos. Este tipo de eventos pode ser utilizado para a comunicação entre subsistemas, como por exemplo, a partilha de informação entre controladores. A definição formal de uma rede de Petri

da classe IOPT defini-se pelo tuplo N = (P, T, A, T A, M, *weight*, *weightTest*, *priority*, isg, ie, oe, osc), os requisitos de cada parâmetro constituinte do tuplo é apresentado na tabela seguinte (Gomes et al., 2007). Além do tuplo apresentado é relevante mencionar outra característica da classe IOPT, o *Maximal Step*, o que permite o disparo no mesmo instante de tempo de todas as transições que estejam habilitadas.

Tabela 2-1 - Apresentação dos requisitos inerentes aos parâmetros do tuplo que define uma RdP da classe IOPT.

Parâmetro	Requisitos
P	Conjunto finito de lugares.
T	Conjunto finito de transições.
A	Conjunto de arcos tal que $A \subseteq ((P \times T) \cup (T \times P))$.
T A	Conjunto de arcos de teste tal que $TA \subseteq (P \times T)$.
M	Conjunto de marcas tal que $M: P \rightarrow N0$.
*Weigh*t	Peso dos arcos tal que $A \rightarrow N0$.
WeightTest	Peso dos arcos de teste tal que $TA \rightarrow N$.
Priority	Prioridades associadas às transições.
InputSignalGuard	Guardas aplicadas às transições, sendo compostas por expressões booleanas nas quais se verificam os valores de sinais de entrada.
InputEvent	Eventos de entrada aplicados a transições, sendo que condicionam o disparo da transição.
OutputEvent	Eventos de saída aplicados em transições.
OutputSignalCondition	Conjunto de regras aplicadas nos lugares para alterar valores de sinais de saída.

As características de maior importância definidas para a classe IOPT, são a introdução de sinais e eventos de entrada e saída, a definição de transições com prioridades, a definição de um limite da quantidade de marcadores em cada lugar e a possibilidade de definição de arcos de teste. A possibilidade de definir transições com prioridades e arcos de teste permite resolver conflitos inerentes às RdP, de forma injusta ou justa, respetivamente. A definição do limite de

marcas nos lugares permite que as ferramentas de geração de código automático criam os recursos de memória necessários à descrição dos lugares.

Na Figura 2-1 apresenta-se uma rede de Petri que modela o comportamento de um sistema de entrada e saída de um parque de estacionamento. Os sinais de entrada são identificados por círculos preenchidos de azul claro, sinalizando a receção do bilhete de entrada para o parque, presença de um automóvel na posição de entrada e pagamento. O sinal de saída é identificado pelo círculo preenchido a verde referente à cancela de entrada para o parque. Os lugares e transições são identificados na própria figura.

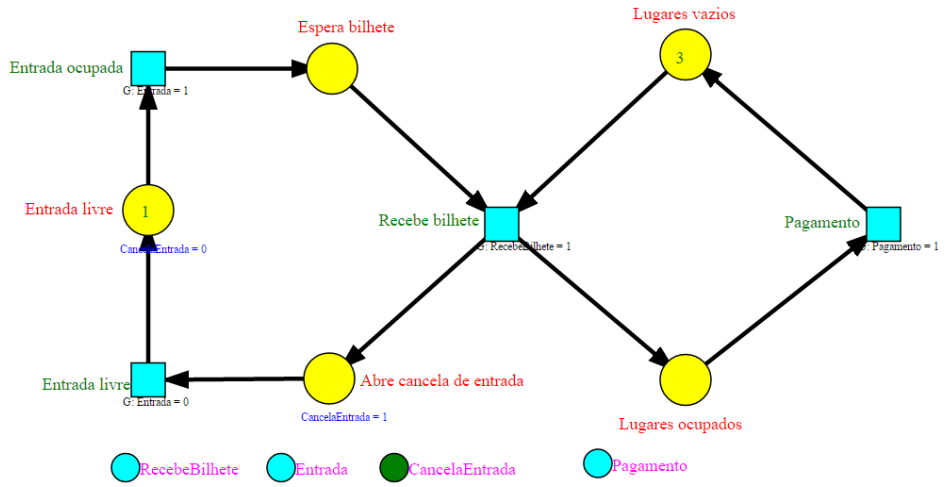

Figura 2-1 - Exemplo de rede de Petri e sinais associados.

2.2 IOPT Tools

2.2.1 Introdução à aplicação IOPT Tools

A ferramenta IOPT-Tools (Pereira, Moutinho, & Gomes, 2012) está em constante desenvolvimento e disponibiliza uma série de ferramentas mencionadas em (Moutinho, 2014) para a modelação de controladores digitais com redes de Petri da classe IOPT via web, a qual pode ser acedida a partir de diversos *browsers* de Internet e por diversos aparelhos como computadores, tablets e telemóveis. O ambiente gráfico disponível para o utilizador é apresentado na Figura 2-2.

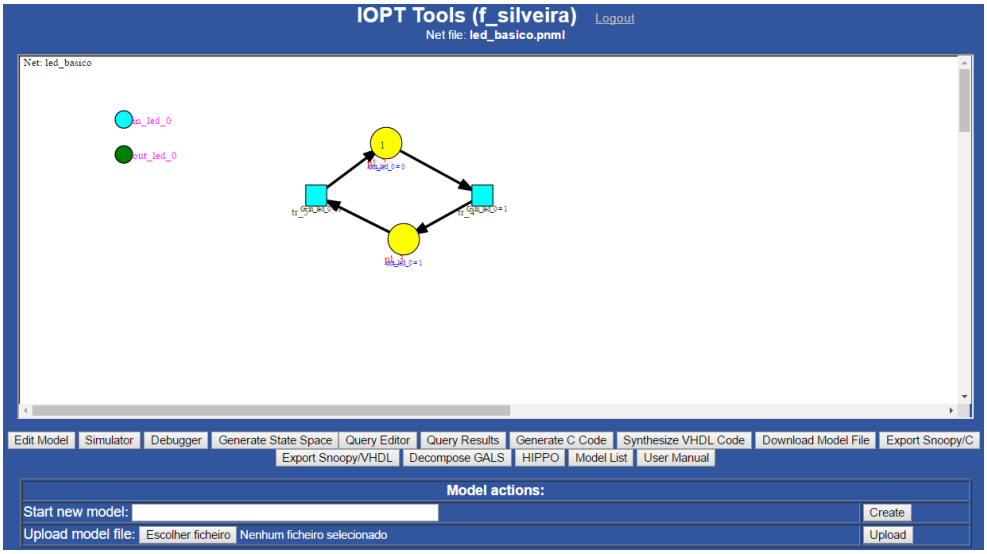

Figura 2-2 - Apresentação do ambiente gráfico das IOPT Tools.

Esta aplicação disponibiliza uma área para edição de redes de Petri assim como a definição dos sinais e eventos associados ao ambiente em que se pretende inserir o controlador, como é apresentado na Figura 2-3.

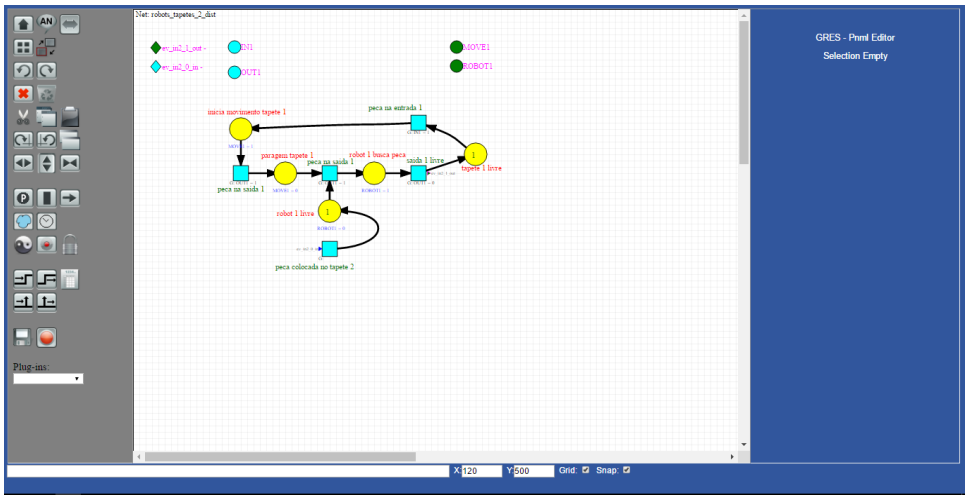

Figura 2-3 - Apresentação do ambiente de edição de modelos.

Dispõe também de uma interface de simulação do modelo criado. A simulação é realizada com um passo de execução e pela alteração dos sinais e eventos de entrada escolhidos pelo utilizador, o que permite a validação do comportamento da rede de Petri. O modelo criado pode também ser descarregado para o aparelho electrónico utilizado para o seu desenvolvimento, ou guardado na lista de modelos associada à conta do utilizador.

Na ferramenta de edição do modelo estão presentes essencialmente dois painéis de controlo, no lado esquerdo o painel para inserção de componentes das IOPT e no seu lado direito o painel para edição dos componentes.

Assim, no painel do lado esquerdo podem ser escolhidos componentes básicos das IOPT como lugares, transições, arcos, sinais e eventos de entrada/saída ou autónomos. Como componentes mais detalhados estão presentes ferramentas para criar canais assíncronos de comunicação para sistemas GALS, criar um lugar complementar ao selecionado, criar semáforos para secções criticas entre lugares, criar vetores/tabelas para registo de valores, criar um lugar e um conjunto de arcos para fixar uma secção de marcação invariante e definição de domínios de tempo GALS para lugares, transições ou arcos específicos.

No painel do lado direito é possível alterar as propriedades dos componentes das redes de Petri, como a legenda dos lugares e transições, a definição de conjuntos de regras nas guardas de transição ou disparo de eventos. Permite definir alterações nos valores dos sinais quando um lugar é marcado. Podem-se definir propriedades dos arcos, como a de atribuir o peso maior do que um, ou definir como arco de teste. No caso da utilização de um arco de teste, a marca não é consumida do lugar, mas permite o disparo das transições associadas.

Além das ferramentas de edição e simulação do modelo estão disponíveis ferramentas essenciais para validação do modelo como a geração do espaço de estados, simulação e *querys* sobre o comportamento do modelo. Portanto, de forma automatizada, podem detectar-se *deadlocks*, conflitos e se as marcações em cada lugar são limitadas. Para facilitar a fase de implementação dos modelos desenvolvidos nas IOPT Tools em plataformas físicas, estão disponíveis geradores de código automático nas linguagens C (Rebelo, Pereira, Moutinho, & Gomes, 2011) e VHDL (Pereira & Gomes, 2013).

2.2.2 Gerador de código VHDL

O gerador de código VHDL permite que o modelo desenvolvido com rede de Petri IOPT seja implementado numa plataforma de hardware (reprogramável) e verificado o comportamento do modelo numa questão de poucos minutos. Com a utilização desta ferramenta são gerados três ficheiros, nos quais são adicionados os sinais lógicos Enable, Reset e Clock.

No primeiro documento: o ficheiro _main em que é definida a estrutura da entidade pelas portas lógicas de entrada e saída de sinais e eventos autónomos. Este ficheiro deve ser definido

como o Top Module na ferramenta Xilinx ISE ou Vivado. O segundo documento: o ficheiro _controller contém a descrição da arquitetura interna comportamental da identidade desenvolvida no modelo representado pela rede de Petri, o qual não carece de qualquer modificação para verificação do sistema. No terceiro documento: o ficheiro _defs contém a definição das portas lógicas do componente e uma possível instância do componente e respetiva associação das portas lógicas da identidade com sinais lógicos.

Concluindo, pode entender-se que mesmo para uma pessoa com conhecimentos reduzidos em linguagens de descrição de hardware consegue realizar a verificação comportamental do sistema desenvolvido com a rede de Petri.

2.2.3 Gerador de código C

A ferramenta de geração automática de código C é utilizada para gerar o controlador modulado com a rede de Petri em código C.

Com efeito, é gerado um ficheiro comprimido na extensão zip, composto por cinco ficheiros. Apenas o ficheiro net_io.c requer ligeiras alterações. Dependendo da arquitetura da plataforma escolhida para implementação, o ficheiro net_io.c deve ser adequado para definirem-se os portos da plataforma escolhida aos sinais de entrada/saída e eventos.

Na Tabela 7-9 dos Anexos apresentam-se as descrições de cada ficheiro gerado pela ferramenta.

Com o desenvolvimento da plataforma IOPT-Tools, foram adicionados novos ficheiros identificados na Tabela 7-10 dos Anexos, para possibilitar a implementação em diferentes plataformas, como exemplo em sistemas Linux ou na plataforma Raspberry PI um e dois. Portanto podem-se compilar apenas os ficheiros gerados que interessam.

2.2.4 Wave4IOPT

A ferramenta Wave4IOPT (Lima, 2015) representa o Editor e Visualizador Web de Formas de Onda. O editor pode receber dados de duas formas distintas. No primeiro caso, o Editor Wave4IOPT recebe os dados provenientes do simulador da ferramenta IOPT Tools referentes aos lugares, transições, sinais e eventos de entrada e saída. No segundo caso, pode ser aberto um ficheiro do tipo *Javascript Object Notation* JSON devidamente estruturado com dados guardados de simulações ou edições anteriores. Pode ser exportado o ficheiro JSON resultante da simulação para o computador pessoal, para posterior edição ou análise dos sinais.

Um aspecto interessante da ferramenta é a portabilidade da ferramenta para outros tipos de sinais digitais. Desde que a estrutura do ficheiro JSON seja cumprida, a ferramenta oferece funcionalidades de edição, visualização e análise de sinais.

2.3 Plataformas System-On-Chip

O conceito das plataformas SoC (System-on-Chip) caracterizado por (Rajesvari, Manoj, Angelin Ponrani, & Annie Joy, 2013) consiste na utilização de um único circuito integrado no qual podem ser implementados todos os componentes de um sistema electrónico. As principais vantagens de utilizar uma plataforma SoC para implementação de um sistema são o alto desempenho do dispositivo, a utilização de tecnologias avançadas de processamento, a redução da escala física do sistema, a eficiência da bateria e o custo reduzido deste tipo de solução.

Na arquitetura tradicional, uma plataforma SoB (System-on-board) é baseada em ASIC (Aplication Specific Integrated Circuit) e com a utilização de um microprocessador ou microcontrolador, memórias, periféricos, lógica personalizada e respetiva combinação numa placa de circuito impresso. Com a implementação de ASICs surgem desvantagens, tais como, o tempo do desenvolvimento destes circuitos e a sua falta de flexibilidade, os quais são aspectos críticos em dispositivos nos quais se preveem melhoramentos. De acordo com (Crockett, Elliot, Enderwitz, & Stewart, 2014), as plataformas SoC apresentam uma solução viável para processadores de aparelhos electrónicos com grande volume de mercado e tempo de vida limitado como PCs, tabletes, smartphones. Na Figura 2-4 apresenta-se uma comparação entre SoB e SoC. A solução mais escolhida para implementar SoCs tem sido as FPGAs, pois podem ser reconfiguradas as vezes que sejam necessárias e pode-se embutir qualquer tipo de sistema (processadores, memória, interface gráfica/som).

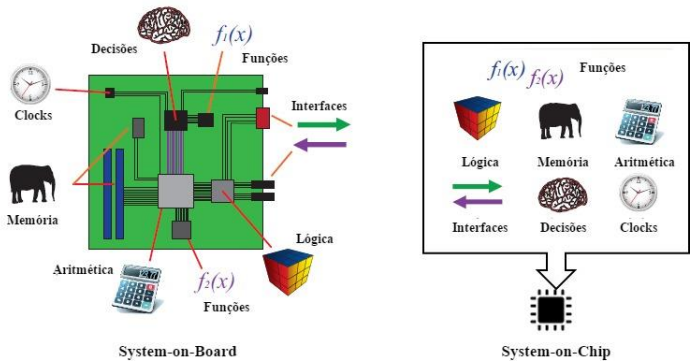

Figura 2-4 - Comparação entre System-on-Board e System-on-Chip. Adaptado de figura 1.1. Capítulo 1 de (Crockett et al., 2014).

2.3.1 Evolução dos SoC

O termo System-on-Chip (SoC) surgiu em meados do ano 1990, inicialmente foi apenas considerado como um termo de marketing devido à limitação da sua escala entre os 350nm e 250nm o que possibilitava apenas a integração de circuitos digitais simples. Inicialmente o termo SoC segundo (Chao, 2001) referia-se a circuitos integrados para aplicações específicas (ASIC) desde que tivesse apenas um circuito integrado na sua composição com o devido sistema embutido. Com a evolução das tecnologias utilizadas no processo de fabrico de componentes semicondutores, em meados do ano 2000, segundo (Crockett et al., 2014; Kuusilinna et al., 2003) conseguiu-se produzir os componentes com cerca de metade da escala, reduzindo o tamanho para a gama entre 180nm e 130nm, o que possibilitou a construção de verdadeiros dispositivos SoC. Finalmente, como referido em (Qian, 2015) iniciou-se a utilização de transístores com dimensões aproximadas a 90, 65 e 45nm.

2.3.2 Arquitetura de um SoC

A arquitetura típica de um dispositivo SoC é composta essencialmente por um processador, memórias, periféricos e as ligações de interconexão entre os elementos. Pode perceber-se esta arquitetura de uma forma simplificada pela Figura 2-5. Sendo assim, o elemento central deste tipo de sistema é o processador, no qual devem ser implementadas duas camadas de *software*. Na camada de mais alto nível temos o sistema operativo e na camada inferior as funcionalidades, interfaces do sistema com o hardware e interação com os componentes periféricos.

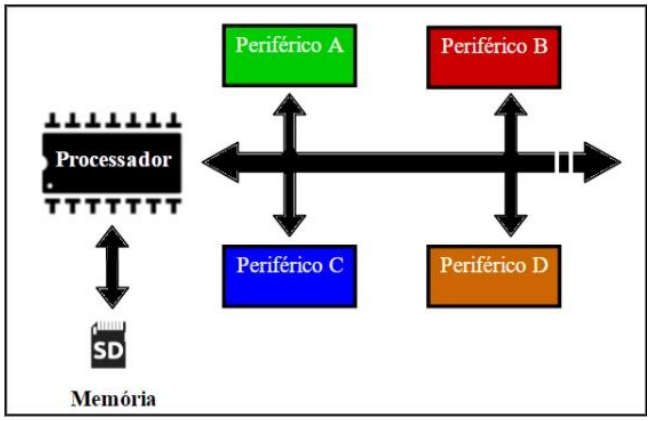

Figura 2-5 - Arquitetura simplificada de um System-on-Chip, figura adaptada de (Crockett et al., 2014).

Os componentes periféricos podem ser divididos essencialmente em três grupos de funcionalidades. O primeiro grupo é constituído por coprocessadores, tendo como principal função a complementaridade do processador principal sendo que o coprocessador deve ser optimizado para uma tarefa específica. O segundo grupo é constituído por núcleos interação com interfaces externas, ligações de elementos como leds e interruptores por exemplo. Por fim o terceiro grupo tem como função a adição de elementos de memória, como cartões de memória SD.

2.3.3 Dispositivo Zynq-7000

O componente Zynq(Xilinx, 2016b) é um dispositivo SoC que é dividido em duas partes, o sistema de processamento (SW) e lógica programável (HW).

Na partição de software a maioria dos dispositivos Zynq dispõe de um processador ARM Cortex-A9 dual core, uma unidade de processamento de aplicações, interfaces de periféricos (MIO-*multiplexed input/output*), memória cache, interfaces de memória e circuitos de geração de clocks. A composição da interface de componentes periféricos envolve protocolos de comunicação como SPI, I2C, CAN, UART, GPIO, SD, USB, GigE.

A partição de lógica programável é baseada na lógica de fábrica das FPGAs Artix-7 e Kintex-7, que é composta essencialmente por blocos de lógica configurável (CLB), blocos de *input/ouput* (IOB) e por interconexões programáveis (termos específicos do Xilinx). Adicionalmente a esta arquitetura típica existem mais dois componentes, blocos de RAMs reconfiguráveis e múltiplas instâncias de DSP48E1 o que permite a implementação de aritmética de alta velocidade. Ambos os componentes podem ser configurados para funcionamento à frequência máxima de clock que é permitida pelo dispositivo.

De acordo com (Crockett et al., 2014) para que ambas as partições comuniquem entre si, são utilizadas interfaces e interconexões com base no standard AXI. A sigla AXI é representativa para *Advanced eXtensible Interface*, neste dispositivo específico é utilizada a versão AXI4 que faz parte do standard ARM AMBA* 3.0 que é o protocolo mais utilizado por fabricantes de dispositivos e blocos IP. Este standard é descrito pela empresa ARM como o standard para comunicações “on-chip” e é definido pelo Xilinx como uma tecnologia óptima de interconexões no caso do seu uso em arquiteturas de FPGAs.

2.3.4 Ferramenta Vivado para o desenho de sistemas com o dispositivo Zynq

A ferramenta Vivado (Xilinx, 2017) pertence ao grupo Xilinx e tem sido desenvolvida para programação dos dispositivos mais recentes como Artix®-7 (7A35T - 7A200T), Kintex®-7 (7K70T, 7K160T) e Zynq®-7000 All Programmable SoC Devices (XC7Z7010 - XC7Z7030).

À semelhança do Xilinx ISE, a ferramenta Vivado permite realizar a síntese, simulação, análise do desenho de sistemas escritos em HDL e implementação no dispositivo alvo. Uma das novidades desta ferramenta é permitir o desenvolvimento de sistemas em linguagens de programação baseadas em C o que viabiliza a utilização da ferramenta por parte de *developers* de *software* com pouco conhecimento sobre linguagens de descrição *hardware*.

As vantagens de utilização da ferramenta são identificadas em (Xilinx, 2017) estão referidas para diferentes fases de um projeto, nomeadamente no desenho de sistemas de alto-nível, verificação e implementação. As vantagens de cada fase do projeto inerentes à ferramenta Xilinx ISE são enunciadas na Tabela 2-2.

Tabela 2-2 - Vantagens proporcionadas pela utilização do Vivado em diferentes fases do desenvolvimento de sistemas.

Aceleraçãodo desenho de sistemas em alto-nível	Aceleração da Verificação	Aceleração da Implementação
Permite gerar blocos IP desenvolvidos em *software* nas linguagens C, C++ e SystemC. *Vivado High-Level Synthesis*	*Vivado Logic Simulation* Verificação acelerada 100x com C, C++ ou SystemC com o Vivado HLS.	Implementação quatro vezes mais rápida.
Integração de blocos IP. *Vivado IP Integrator*	*Integrated Mixed Language Simulator*	Melhoramento da densidade do desenho do sistema em 20%.
Integração de DSP baseado em modelos. *System Generator for DSP*	*Integrated & Standalone Programming and Debug Environments*	Melhoria de desempenho até 3-Speedgrade para *low-end* & *mid-range* e melhoria de 35% de consumo energético para soluções *high-end*.

2.3.5 Plataforma Red Pitaya

A plataforma Red Pitaya(StemLabs, 2016b) é um dispositivo que dispõe de interfaces de medição, controlo, comunicação e armazenamento de dados. A plataforma utiliza o sistema operativo Linux. Na plataforma estão incluídas tecnologias para gerar e adquirir sinais de radio frequência, um módulo FPGA , módulo de processamento de sinais digitais e um CPU (Central Processing Unit). A plataforma Red Pitaya dispõe de duas portas de entrada e duas portas de saída Fast-Analog, um módulo de conexão Ethernet por cabo RJ45, um módulo para comunicação USB e outro para micro USB. Também dispõe de uma ligação micro USB para ser utilizada como fonte de alimentação e um slot para cartão de memória Micro SD. A localização dos periféricos mencionados apresenta-se na Figura 2-6.

Figura 2-6 - Plataforma Red Pitaya e identificação de alguns periféricos, imagem editada de (Leban, 2014).

A plataforma é constituída também por duas extensões de conectores devidamente identificados na Figura 2-7. O extensor um tem 16 pinos que podem funcionar com sinais de entrada ou de saída e que funcionam com valores de tensão eléctrica entre 0V e 3.3V, dois pinos ligados à massa e dois pinos alimentados por 3.3V. O extensor dois tem quatro pinos configurados para sinais de entrada analógicos e quatro pinos configurados para sinais de saída analógicos. É composto também por seis pinos ligados à massa, dois pinos UART dos quais um RX e um TX, dois pinos I2C o pino SDA e o pino SCL, quatro pinos SPI sendo estes SPI_CLK, SPI_MOSI, SPI_CS e SPI_MISO. Também dispõe de um pino ligado a -4V e um pino ligado a +5V.

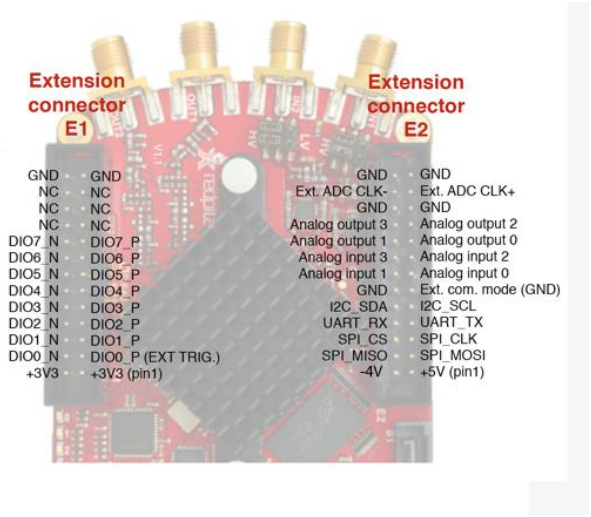

Figura 2-7 - Localização e descrição das duas extensões de conectores, figura retirada de (StemLabs, 2017).

2.3.6 Plataforma ZedBoard

A plataforma ZedBoard(Avnet, 2017) à semelhança da Red Pitaya tem na sua constituição o dispositivo XC7Z020 Zynq baseado na lógica de fabrico da Artix-7. Esta plataforma pode ser utilizada para desenvolvimento de projetos na indústria e por grupos de investigação, mas também tem como objetivo promover a aprendizagem deste tipo de sistemas a estudantes. De forma a facilitar a utilização da plataforma e o desenvolvimento de projetos, foi criada uma plataforma na forma de website para partilha de conhecimentos em (Avnet, 2017).

A plataforma tem uma arquitetura preparada com uma série de interfaces para periféricos, o que facilita o desenvolvimento das mais variadas aplicações sejam de ambiente académico ou industrial.

As interfaces GPIO existentes na plataforma apresentados em (Crockett et al., 2014) são: nove leds, oito interruptores, sete botões de pressão e cinco interfaces Pmod (Peripheral modules). Ao nível de multimédia contem interfaces para Codec de áudio (Analog Devices A-DAU1761) que suporta sinais de áudio como sinais de entrada ou sinais de saída, vídeo HDMI e VGA e ecrã OLED. Relativamente a interfaces de comunicação contem periféricos para Ethernet, USB-OTG, USB-JTAG, USB-UART.

Podem também encontrar-se outros periféricos mais genéricos na arquitetura deste tipo de plataformas como slot para cartão de memória SD, interfaces FMC, XADC header e Xilinx JTAG header.

2.4 Linguagens de descrição hardware, web e de programação

2.4.1 VHDL (Very High-Speed Hardware Description Language)

A linguagem VHDL(Lipsett, Schaefer, & Ussery, 1989) é utilizada para descrever hardware, facilitando a compreensão por parte do ser humano e da máquina. A linguagem permite o uso de uma variedade de ferramentas no sentido do desenvolvimento de projetos de sistemas embutidos, o que permite diminuir o tempo de desenvolvimento de projetos.

Essencialmente, permite descrever modelos nos domínios de estruturas e funcionalidades e possibilita a reutilização de bibliotecas de blocos IP comuns e os testes de quais blocos IP são os mais adequados para o projeto. Numa fase inicial de projeto, as equipas de desenvolvimento podem partilhar descrições de alto nível acerca dos subsistemas que pretendem desenvolver permitindo o desenvolvimento de cada subsistema de forma independente, seja o do comportamento central, ou o do progresso dos diferentes subsistemas. Adicionalmente, com a possibilidade de reutilização de blocos de propriedade intelectual e encapsulamento de blocos, o tempo de desenvolvimento de projetos é diminuído.

Os modelos VHDL comportam-se como circuitos hardware o que permite a utilização destes modelos como especificação de fabrico. É entendida como uma boa linguagem principalmente para as fases de desenvolvimento de projeto: especificação e simulação. Foi desenvolvido para suportar diferentes metodologias de desenho de sistemas como: *top down* vs. *bottom up*, síncrono vs. assíncrono, PLA vs. lógica aleatória.

2.4.2 Linguagens web HTML e CSS

A linguagem HTML (Hypertext Markup Language) permite estruturar um documento digital, descrever o conteudo de uma página web e especificar elementos como hyperlinks. Todos os websites são consituidos por um ou um conjunto de documentos escritos em HTML. Neste tipo de documentos podem-se incluir elementos como imagens, reprodutores de videos ou de sons. É uma linguagem que está em constante evolução sendo o HTML5 a versão mais recente.

A evolução do HTML tem como principal propósito facilitar a integraçao de elementos de multimédia sem que seja necessário integrar outros tipos de tecnologia, mantendo a facilidade de interpretação do código por seres humanos. Novas APIs (Aplication Programming Inter-

face) têm sido desenvolvidas para facilitar a integração de objectos mais complexos e o processo de encontrar defeitos no código.

A linguagem CSS (Cascading Style Sheets) desenvolveu-se para que o estilo de apresentação dos elementos do HTML fosse definido num documento à parte. Esta necessidade surgiu ao longo da versão HTML 3.2, para reduzir o tamanho dos documentos HTML e para reutilizar a definição do estilo dos elementos em páginas web diferentes.

2.4.3 Linguagens de programação JavaScript, C e C++

No seguimento do subcapítulo anterior apresenta-se uma breve descrição da linguagem Javascript, uma vez que a sua utilização tem como foco a interacção entre as páginas web e o cliente a partir de um *script*. O *script* desenvolvido pode ser embutido no documento HTML ou escrito num documento à parte e associado ao documento HTML. O Javascript é uma linguagem orientada a objectos e estando associada a páginas web permite definir os comportamentos e estados dos elementos existentes no documento HTML. Para a utilização desta linguagem é necessário que o *browser* do utilizador seja compativel.

A linguagem de programação C(Ritchie & Kernighan, 2002) é uma das mais conhecidas, criada por Dennis Ritchie entre 1969 e 1973. A linguagem C desenvolveu-se para implementar sistemas de software, nomeadamente operativos e aplicações. Os paradigmas abordados pela linguagem C são o paradigma de estrutura e paradigma imperativo. Neste sentido um documento escrito em C especifica a sequência de instruções utilizando iterações e decisões que o programa deve seguir para alcançar um estado determinado. Apesar de ser uma linguagem utilizada para implementar sistemas em software, podem ser incluidas bibliotecas de funções, como *Dynamic-Link Library* para que se possa ter acesso a sinais de hardware.

A linguagem C++ densenvolveu-se com base na linguagem C com o objetivo de adicionar o paradigma de orientacao a objectos à linguagem C. Este tipo de abordagem permite facilmente criar classes de objectos compostos por atributos e métodos de acesso e alteração dos atributos. As classes de objectos são caracterizadas também por conceitos como herança, associação e polimorfismo. O mecanismo de herança permite que uma classe possa estender outra, utilizando o mesmo tipo de atributos e métodos. O conceito de associação permite que um objeto utilize métodos de outros objectos. O princípio de polimorfismo consiste em duas subclasses herdarem métodos de uma classe, mas a invocação de um método com a mesma identificação tenha comportamentos diferentes.

2.5 Aplicação Software Defined Radio

2.5.1 Características de SDR ideal

Um SDR não tem qualquer tipo de processamento de sinal analógico, exceto a antena, o amplificador de potência, microfone e altifalante à semelhança do modelo apresentado na Figura 2-8. Pelo documento (Software, 2004) entende-se que um SDR permite o controlo por software dos métodos de modulação de sinais, escolha de largura de banda, funções de segurança (*frequency hopping*), requisitos de forma de onda dos standards para uma gama específica de frequências.

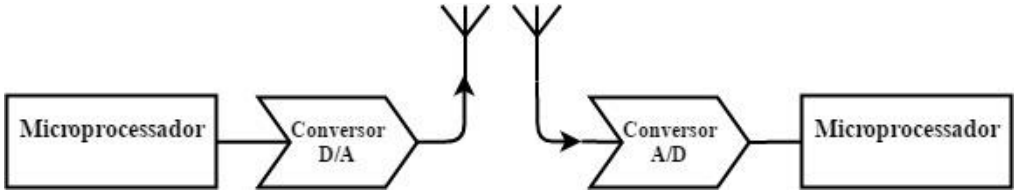

Figura 2-8 - Arquitetura ideal para um SDR, figura adaptada de (Software, 2004).

Os telemóveis são considerados como *"software-controlled radio"*, pois apesar de poderem funcionar em mais do que um tipo de rádio (2G/3G), as funcionalidades são limitadas pelo desenho do sistema. Os SDRs podem ser reprogramados para funcionalidades que não foram antecipadas utilizando diferentes formas de onda com qualquer frequência portadora a qualquer largura de banda.

As antenas devem ter capacidade para capturar ondas eletromagnéticas desde frequências muito baixas (< 1 MHz) a muito altas (> 60 GHz).

Nos SDR deve ser considerada a seleção de canal de comunicação e rejeição de interferências. Uma vez que não é utilizado um *front-end* RF, toda gama de frequências deve ser digitalizada, o que seguindo o critério de Nyquist, a um ritmo de duas vezes a largura de banda escolhida (Grayver, 2013).

As vantagens mais significantes da utilização de um SDR são:

- Quando existe a necessidade de interoperabilidade entre dispositivos.
- Para utilização eficiente dos recursos em função das condições do funcionamento.
- Para reutilização de gamas de frequência conforme oportunidade.
- A redução de dispositivos obsoletos.
- Custos reduzidos.
- Investigação científica

2.5.2 Processador de Sinais Digitais (DSP)

Um processador de sinais digitais (Grayver, 2013) é um microprocessador optimizado para funções direcionadas ao processamento de sinais digitais. Na comparação à utilização de um processador genérico (GPP- General Purpose Processor), este (DSP) tem um consumo de energia mais reduzido. O que se deve à eliminação de uma série de transístores que seriam utilizados para cache ou subsistemas periféricos.

A principal vantagem dos DSP em relação aos GPP está no consumo de energia. Podem ser desenvolvidos processadores com vírgula flutuante ou fixa, no caso da vírgula fixa a utilização de energia será mais reduzida. Os DSP não são muito úteis para implementar sistemas de controlo. A boa utilização destes processadores passa pelo processamento de dados como na implementação de protocolos de comunicação. Um SDR típico combina um DSP com um GPP para implementar a camada de rede.

Os ambientes de desenvolvimento de um DSP são de alguma forma mais complexos do que os ambientes de desenvolvimento para GPP. Além disso, a utilização de um sistema operativo para implementação de um DSP está bem limitada. O que leva a que muitos DSP não utilizem qualquer tipo de sistema operativo e interajam diretamente com o hardware. No caso de ser necessária a utilização de um sistema operativo, é necessário que seja de tempo real.

Os dispositivos DSP são utilizados extensivamente em células de comunicação e em rádios que requerem pouca energia para o funcionamento e têm requisitos moderados relativamente à taxa de processamento de dados.

2.5.3 Utilização do Zynq para desenvolvimento de SDR

Sendo o dispositivo escolhido para o desenvolvimento do SDR, o Zynq-7000, a solução passa pela utilização de um GPP (Dual Core com Debian) com interface gráfica e controlador co-design com FPGA, onde se pretende implementar o DSP. A comparação entre os dispositivos que se adaptam ao desenvolvimento de um SDR prende-se com o consumo de energia, custos, esforço para desenvolvimento e taxa de transferência de dados, tal como é apresentada na Figura 2-9.

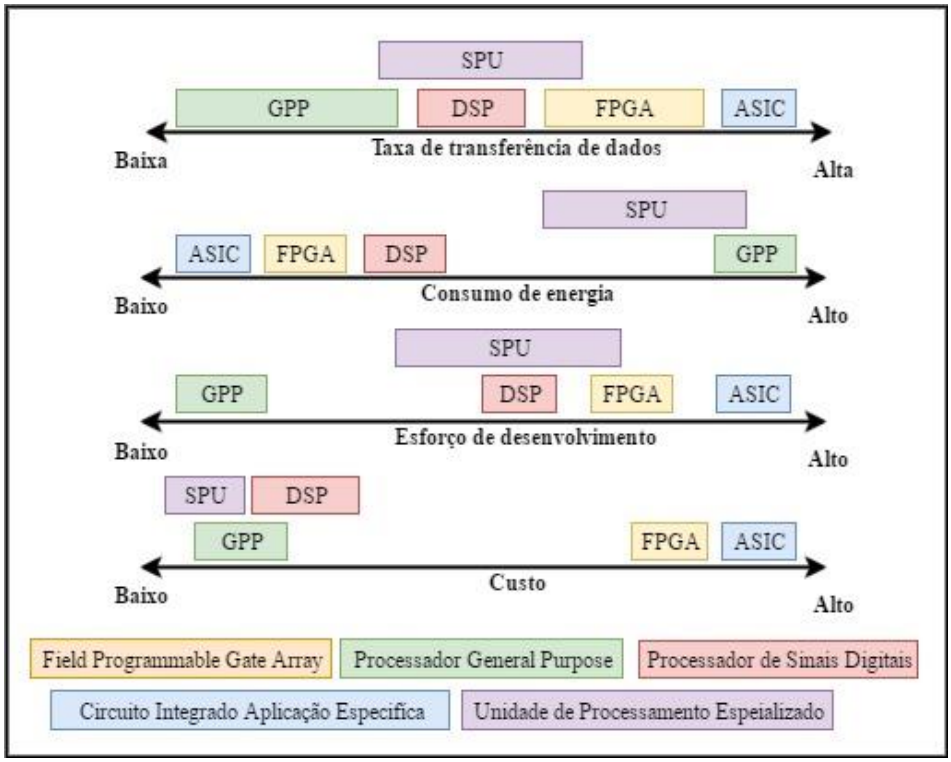

Figura 2-9 - Comparações entre os dispositivos possíveis para implementação de SDR, adaptado de (Grayver, 2013).

Apresentam-se de seguida algumas considerações relativamente às configurações possíveis, mencionadas em (Grayver, 2013):

- Necessidade de grande largura de banda ao acesso da memória no caso de utilização de reconfiguração parcial e no caso da necessidade de alteração rápida das formas de onda.
- O GPP deve ler o ficheiro de configuração e escrever os dados para a FPGA. Existem duas formas básicas para guardar os ficheiros de configuração: pela utilização de *NVRAM (FLASH)*, em que o *bus* de memória é partilhado com a FPGA, e pela utilização de *SDRAM* na qual se utilizam canais de *bus* separados. Sendo certo que a utilização de *SDRAM* é motivada para aplicações de reconfiguração parcial.
- Reconhecimento da necessidade de utilização de memória FLASH (*FPGA-friendly*) (Xilinx, 2010) e (Intel Corporation, 2016) para configurações simples e rápidas. O que permite ao GPP controlar e iniciar a configuração sem transferência de dados. Esta consideração segue o intuito da reconfiguração total.
- O acesso à memória pode não ser uma preocupação no caso de configurações de múltiplas formas de onda, uma vez que poucos registos têm de ser escolhidos para a escolha da forma de onda.

Algumas alternativas dos dispositivos presentes no mercado são comparadas na Tabela 2-3, apresentada no documento (Wyglinski, Orofino, Ettus, & Rondeau, 2016).

Tabela 2-3 - Comparação entre dispositivos para o desenvolvimento de SDR, tabela adaptada de (Wyglinski et al., 2016).

	Ettus USRP N200/N210	ZedBoard Xilinx Zynq-7000 FPGA & AD-FMCOMMSs-EBZ	NooElec NESDR Mini SDR USB stick	Ettus USRP E300
Interface com computador anfitrião	Gigabit Ethernet	Conectores Dual FMC	USB 2.0	Interface AXI4-MM para proceesador ARM dual-core Cortex-A9
RF front-end	USRP daughterboards	RFIC Integrado	RFIC Integrado	RFIC Integrado
Largura de banda	25-50Mhz	56 MHz	3.2 MHz	56 MHz
Gama de frequências RF	DC- 6GHz	70 MHz a 6 GHz	24 a 1766 MHz	70 MHz a 6 GHz
MIMO	Unidades de 1x1 a 8x8	4x4	Não suporta	2x2
Full-duplex	Sim	Sim	Apenas Rx	Sim
ADC	Dual 14-bit 100MS/s	Dual/quad 12-bit 61.44 MS/s	8-bit 3.2 MS/s	Dual/quad 12-bit 61.44 MS/s
DAC	Dual 16-bit 600 MS/s	Dual/quad 12-bit 61.44 MS/s	Não tem	Dual/quad 12-bit 61.44 MS/s
FPGA	Xilinx Spartan 3ª DSP	Xilinx Zynq-7000	Não tem	Xilinx Zynq-7000
Compatibilidade RFNoC	Não suporta	Não suporta	Não suporta	Suporta
Custo	1800€	500€ + 1000€	20€	3000€

2.5.4 Ferramentas para desenvolvimento DSP

Com a importância e aumento de popularidade dos SDRs no âmbito das telecomunicações, a quantidade de ferramentas para o desenvolvimento de soluções tem sido exponencial. No artigo (Oliveira & Costa, 2017) são mencionadas uma série de ferramentas como: OPEN Iris (Sutton, 2015), OpenBTS (OpenBTS, 2017), REDHAWK (REDHAWK, 2016), GNU Radio (Rondeau, 2015), Labview (2017 National Instruments Corporation, 2017), Matlab (1994-2017 The MathWorks Inc., 2017a). Uma vez que o dispositivo escolhido é o Zynq-7000 e a plataforma de desenvolvimento é o Vivado, pretende-se a utilização do *System Generator for DSP*, que é outra ferramenta da Xilinx, a qual, com o apoio das ferramentas do Matlab facilita o desenvolvimento do DSP.

O System Generator for DSP (Xilinx, 2016a) é líder nas ferramentas alto nível para desenvolvimento de sistemas que utilizem DSPs de alto-desempenho para dispositivos *All Programable*. Esta ferramenta permite o desenvolvimento de algoritmos DSP numa fracção de tempo comparativamente ao tradicional RTL.

A ferramenta de apoio presente no Matlab para apoio ao desenvolvimento de unidades DSP é o Simulink (1994-2017 The MathWorks Inc., 2017b) que segue a metodologia baseada em modelos. Os modelos desenvolvidos podem ser traduzidos de forma automática para descrições de Hardware, um breve exemplo é descrito em (Grout, 2001). Assim, permite a utilização dos modelos desenvolvidos na ferramenta Vivado para posterior implementação.

As bibliotecas do Simulink incluem funções básicas de processamento de sinais digitais, lógica de controlo, funções aritméticas, armazenamento de dados, funções de comunicação, entre outros. A utilização das ferramentas do Simulink permitem também, a simulação, teste e verificação dos modelos desenvolvidos.

2.5.5 Trabalhos relacionados

No artigo (Prata et al., 2017) é utilizada uma arquitetura de SDR *All-Digital Transmitter* (ADTx) em que todo o processamento de sinais é realizado de forma digital com a utilização de DSP. Além de técnicas e propostas para o desenvolvimento de transmissores de RF flexíveis, apresentam-se vantagens da utilização de FPGAs relativamente ao tempo e custos associados.

No artigo (Ferreira & Ferreira, 2017) apresentam-se soluções de nível avançado relativamente à flexibilidade, adaptabilidade e eficiência de recursos, utilizando-se a reconfiguração parcial dinâmica de FPGAs. O que é uma das vantagens da utilização dos dispositivos All Programable SoC da Xilinx. A reconfiguração dinâmica de um modulador OFDM (Orthogonal Frequency-Division Multiplexing) é discutida sendo os resultados comparados aos standards impostos pelo protocolo LTE (Longterm Evolution). As modulações de sinais consideradas foram QPSK 16-QAM e 64-QAM.

O artigo realizado por (Junior, Oliveira, & Junior, 2015) apresenta uma metodologia para implementação de SDR num dispositivo Zynq. A solução apresentada passa pela implementação em VHDL de um dispositivo modulador e desmodulador QPSK half-duplex. Nesta solução verificam-se as vantagens de simplificação do módulo de processador, módulo de interface de rede e módulo para FFT (Fast Fourier Transform) (Brigham, 1974) dos sinais. Uma das sugestões para melhoria de desempenho e consumo energético é a separação do módulo do modulador e desmodulador para minimizar a área de utilização da FPGA. Para simulação do modem utilizou-se a ferramenta Simulink presente no ambiente da aplicação Matlab. A implementação realizou-se com a ferramenta Xilinx EDK.

O artigo (Kumar, 2016) apresenta uma solução para a classificação de sinais de rádio utilizando redes neuronais artificiais. Este tipo de rádio integra-se nos SDR inteligentes, o que é uma mais-valia para a identificação do ambiente em que o rádio está em funcionamento e adaptar a sua definição consoante as classificações obtidas. A classificação é realizada para identificar o tipo de modulação utilizado nos sinais recebidos sejam por modulação de frequência, amplitude ou fase em diferentes valores de SNR (Signal to Noise Ratio). Na implementação em FPGA, nomeadamente ARTIX-7, utilizaram-se compiladores para aplicar a FFT e filtro do tipo FIR(Finite Impulse Response) para redução do tempo de reconhecimento dos sinais. Na Tabela 2-4 apresentam-se os resultados da classificação utilizando este método. No entanto não está identificado o ambiente em que foram realizados os testes.

Tabela 2-4 - Resultados das classificações de sinais realizado no artigo (Kumar, 2016).

Detecção de modulação	%
BPSK	98
QPSK	94
FSK	95
ASK	90
QAM	85

Comparando os objetivos da dissertação com os trabalhos referenciados, podem ser identificadas semelhanças no âmbito da implementação do DSP em hardware utilizando FPGAs nomeadamente com a utilização do dispositivo Zynq. Outra característica identificada é a utilização de ferramentas que utilizam modelos para facilitar o desenvolvimento do DSP. Como principais diferenças destaca-se a utilização de RdPs para modelar o sistema, o desenvolvimento de uma aplicação web para facilitar a interacção do utilizador e para que o sistema possa ser controlado remotamente. Além disso, sendo todo o sistema implementado no dispositivo Zynq permite que o funcionamento da plataforma não dependa de estar ligada a um computador. Outra diferença para com os trabalhos mencionados é a preparação do sistema desde a fase inicial considerando a possibilidade de reconfiguração parcial dinâmica para o aumento da flexibilidade do sistema.

3

3 Desenvolvimento do projeto

3.1 Descrição global do sistema

O principal objetivo do sistema é a modulação de mensagens introduzidas no sistema e desmodulação de mensagens recebidas, utilizando a modulação QPSK. Neste projeto o recetor está directamente ligado ao emissor na FPGA, sendo que as mensagens recebidas são o eco das mensagens enviadas. A utilização de antenas na plataforma não foi validada, no entanto o processador de sinais digitais foi desenvolvido para este efeito.

O sistema é composto por três partes, sendo estas a parte da aplicação web pela qual o utilizador pode interagir com o sistema, a parte de controlo do sistema para gerir o funcionamento do sistema e a parte do processador de sinais digitais, como sugerido na Figura 3-1. A aplicação web e parte do controlador implementaram-se na partição software do dispositivo Zynq e a outra parte do controlador assim como o processador de sinais digitais implementou-se em hardware.

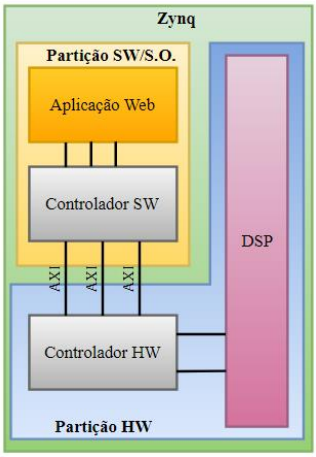

Figura 3-1 - Estrutura do sistema desenvolvido.

A execução de cada parte do sistema é identificada na Figura 3-2 onde também são identificadas as comunicações entre estas. A comunicação entre a aplicação web e o controlador é realizada com a partilha de ficheiros de texto e a comunicação entre as duas partes do controlador é realizada com base na especificação de interconexão *on-chip* AMBA(*Advanced Microcontroler Bus Architechture*), utilizando a interface AXI4.

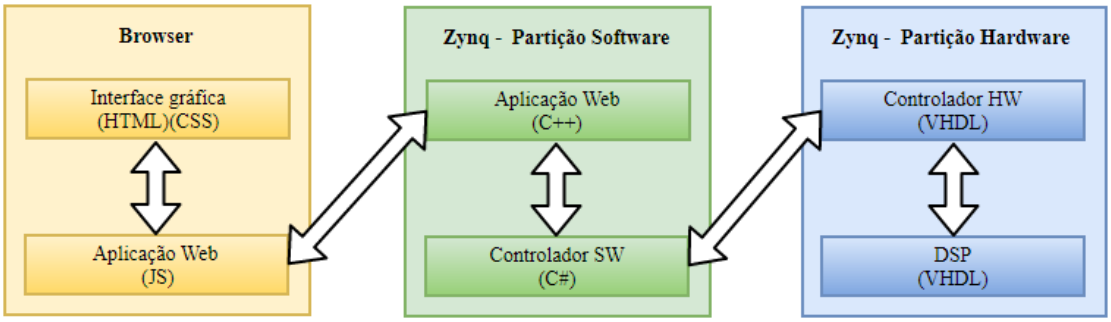

Figura 3-2 – Identificação de onde cada parte do sistema é executada.

As limitações do sistema na partição hardware são o limite do tamanho do pacote de mensagens, as capacidades de armazenamento do *buffer*, o limite de bits na transferência de informação entre o hardware e o software devido à utilização do mesmo bus de dados para escrita e leitura nos registos de memória da FPGA. Na aplicação web definiu-se o limite de caracteres no envio de mensagens para que o limite de tamanho do pacote de mensagem seja cumprido. Também na aplicação web há a necessidade de utilizar o botão de "refresh signals" para que o utilizador tenha acesso aos valores actuais dos sinais do controlador.

Os sinais de entrada e de saída do sistema disponíveis para interacção do utilizador, são apresentados de seguida na Figura 3, os quais são correspondentes à escolha da configuração da FPGA e o controlador associado. Na Figura 4 e Figura 5 apresentam-se os sinais referentes às funcionalidades

para interacção com o emissor e recetor, respectivamente. Os sinais de entrada estão identificados com o preenchimento de fundo verde e os sinais de saída com o fundo de cor roxa.

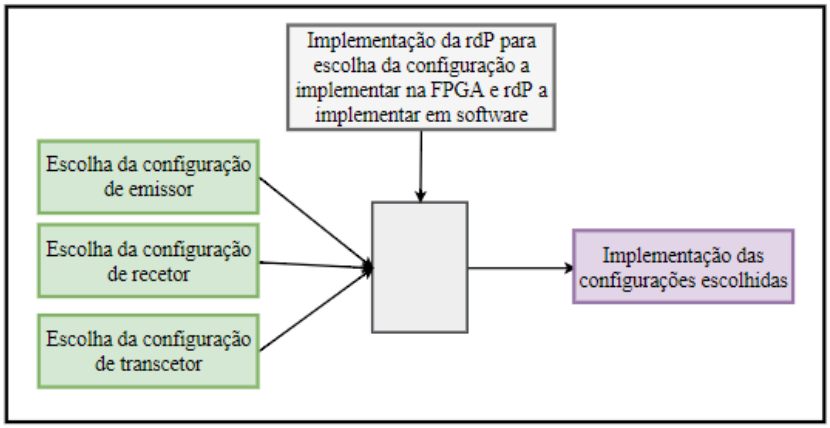

Figura 3 – Apresentação dos sinais de entrada e saída do sistema para a implementação da configuração da FPGA e execução do controlador associado.

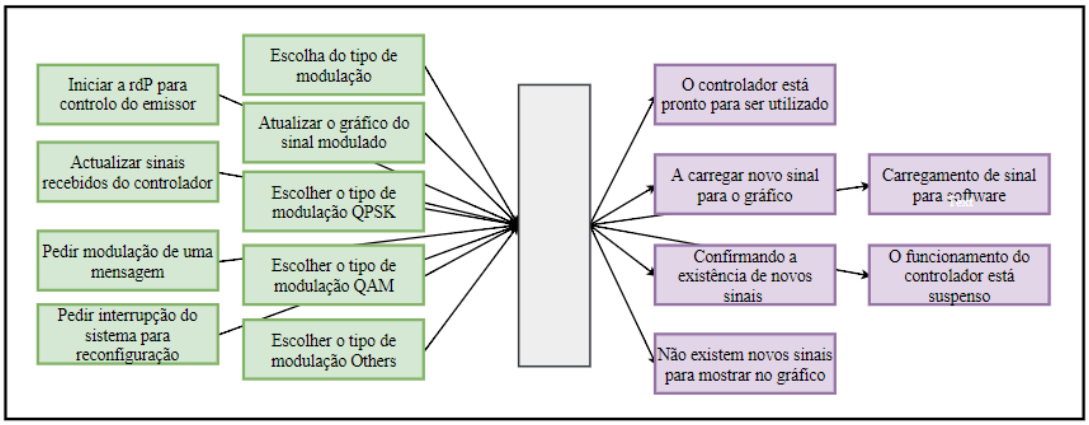

Figura 4 – Apresentação dos sinais de entrada e saída do sistema referentes ao emissor.

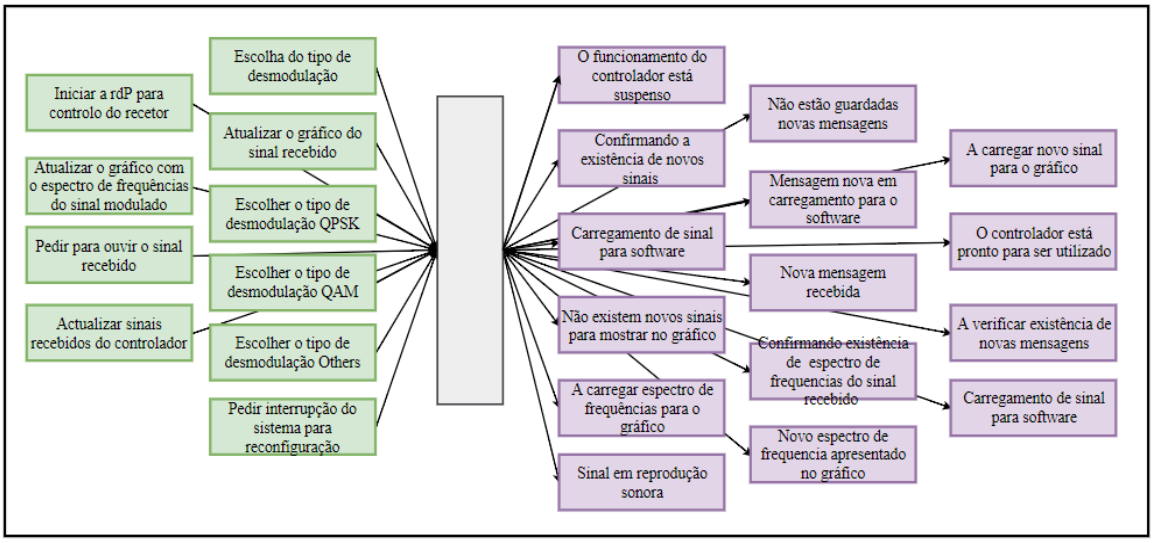

Figura 5 - Apresentação dos sinais de entrada e saída do sistema referentes ao recetor.

3.2 Controladores IOPT

3.2.1 Controlador para escolha do sistema implementado

Desenvolveu-se um controlador para que consoante a escolha do utilizador quanto ao funcionamento do sistema a respetiva configuração da FPGA seja implementada no dispositivo e para que o controlador referente ao emissor ou recetor ou ambos seja executado. A Figura 3-6 apresenta a rede de Petri referente a este controlador.

Na Tabela 3-1 indicam-se os eventos de saída as expressões utilizadas nas guardas de cada transição e na Tabela 3-2 indicam-se os sinais de saída utilizados nos lugares.

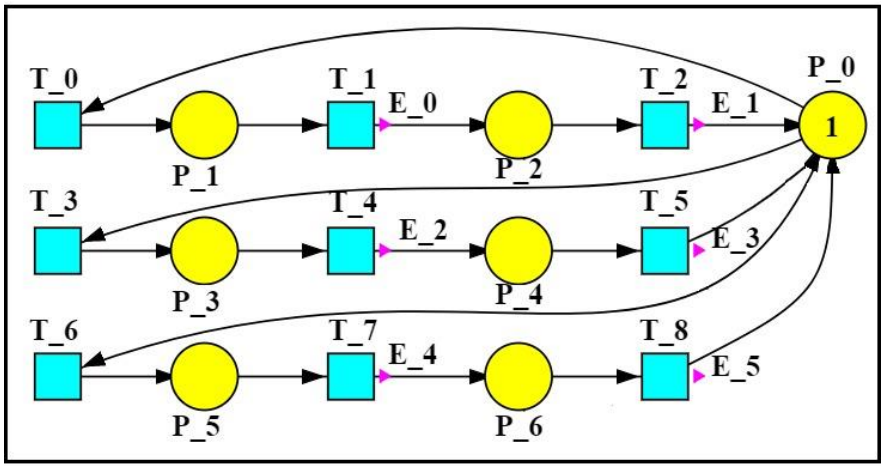

Figura 3-6 – Rede de Petri referente à escolha de funcionamento do sistema.

Tabela 3-1 - Identificação dos eventos de saída e expressões utilizadas nas guardas de cada transição.

Identificação da transição	Expressão utilizada na guarda de transição	Eventos de saída
T_0	choose_tx=1	
T_1	rx_stopped=1 ∧ tx_stopped=1	imp_tx
T_2	tx_pn_ok=1	able_tx
T_3	choose_rx=1	
T_4	rx_stopped=1 ∧ tx_stopped=1	imp_rx
T_5	rx_pn_ok=1	able_rx
T_6	choose_txrx=1	

T_7	rx_stopped=1 ∧ tx_stopped=1	imp_txrx
T_8	rx_pn_ok=1 ∧ tx_pn_ok=1	able_txrx

Tabela 3-2 - Identificação dos sinais de saída utilizados nos lugares da rede de Petri.

Identificação do lugar	Sinais de saída
P_1	change_mod=1
P_3	change_mod=1
P_5	change_mod=1

3.2.2 Descrição da RdP referente ao controlador para o emissor

Na rede de Petri que modela o controlador do emissor de mensagens pretende-se essencialmente controlar os comportamentos do emissor. Os comportamentos modelados são identificados na Figura 3-7, com cores diferentes, sendo estes: a visualização gráfica dos sinais guardados (A) com a cor azul-claro, iniciar ou interromper o funcionamento do controlador (B) com a cor vermelha, escolher o tipo de modelação integrado no emissor (C) com a cor laranja, detectar-se a existência de sinais modelados para guardar em software (D) com a cor azul-escuro e a sinalização de existência de novas mensagens para serem processadas pelo emissor de mensagens (E) com a cor verde.

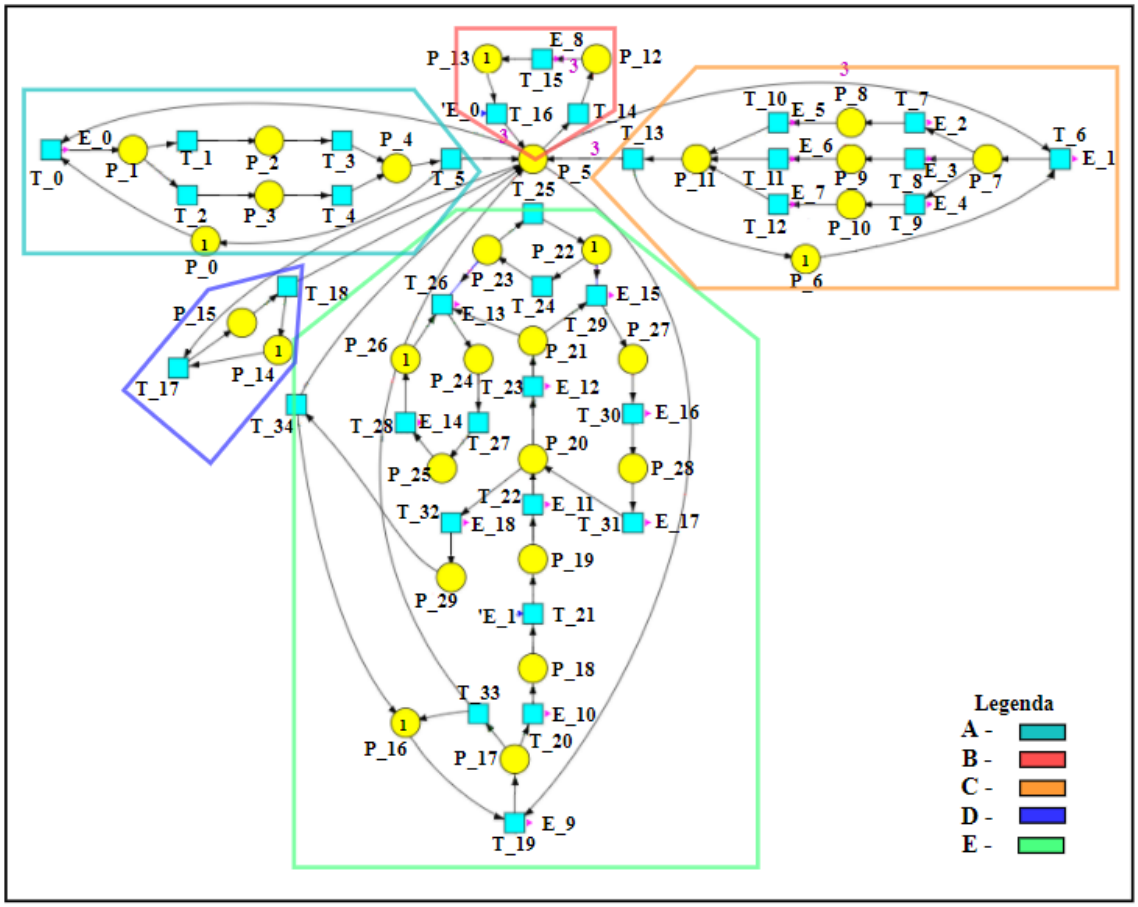

Figura 3-7 - Rede de Petri referente ao controlador do emissor.

Nos anexos, as Tabela 7-1 Tabela 7-2, Tabela 7-3 e Tabela 7-4 apresentam a descrição dos sinais de entrada, sinais de saída, eventos de entrada e eventos de saída respetivamente. Nas tabelas identifica-se também a fonte e destino de cada sinal e evento. Para a compreensão do modelo desenvolvido separou-se cada comportamento em sub-redes para facilitar a descrição.

Na Figura 3-8 apresenta-se a sub-rede utilizada para modelar o comportamento de mostrar na aplicação web o sinal modulado com a mensagem que se pretende enviar. O comportamento é iniciado assim que o utilizador utiliza o botão referente na aplicação web e de seguida o controlador apresenta o sinal após a verificação de que ele existe. Nas Tabela 3-3 e Tabela 3-4 indicam-se os sinais de saída utilizados nos lugares e as expressões utilizadas nas guardas de transição, respetivamente.

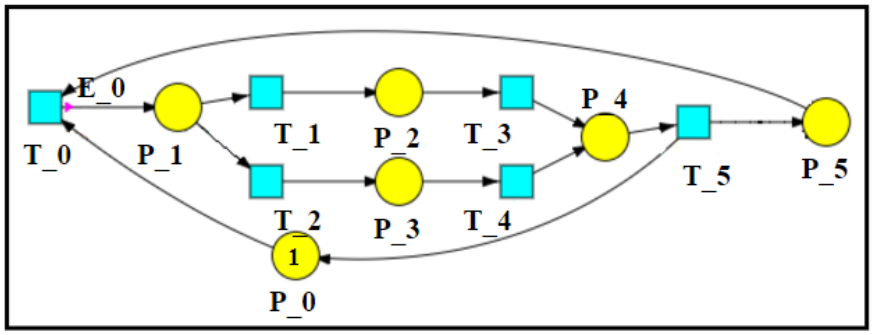

Figura 3-8 – Rede de Petri para modelar o comportamento de mostrar o sinal enviado ao utilizador.

Tabela 3-3 - Identificação dos sinais de saída utilizados nos lugares da rede de Petri.

Identificação do lugar	Sinais de saída
P_1	tx_check_new_sig
P_2	tx_notloadGrafico
P_3	tx_loadGrafico

Tabela 3-4- Identificação das expressões utilizadas nas guardas de transição utilizadas nas transições da rede de Petri.

Identificação da transição	Expressão utilizada na guarda de transição
T_0	VeSinal=1 ∧ EscModelacao=0 ∧ tx_changemode=0
T_1	tx_sig_unavailable=1
T_2	tx_sig_available=1

Na Figura 3-9 apresenta-se a sub-rede utilizada para modelar o comportamento de escolher qual o tipo de modulação que está em funcionamento. Este comportamento parte da acção do utilizador no botão referente na aplicação. A sub-rede desenvolvida permite activar o módulo de gerador de sinais, no entanto ao longo do projeto apenas o módulo QPSK foi desenvolvido. Assim, na escolha de outro tipo de modulação, o funcionamento do emissor desenvolvido fica em suspensão. Além disso, quando o utilizador pretende mudar o módulo de emissor em funcionamento, todas as outras funcionalidades ficam suspensas após a respetiva conclusão. Na Tabela 3-5 indicam-se os eventos de saída as expressões utilizadas nas guardas de cada transição.

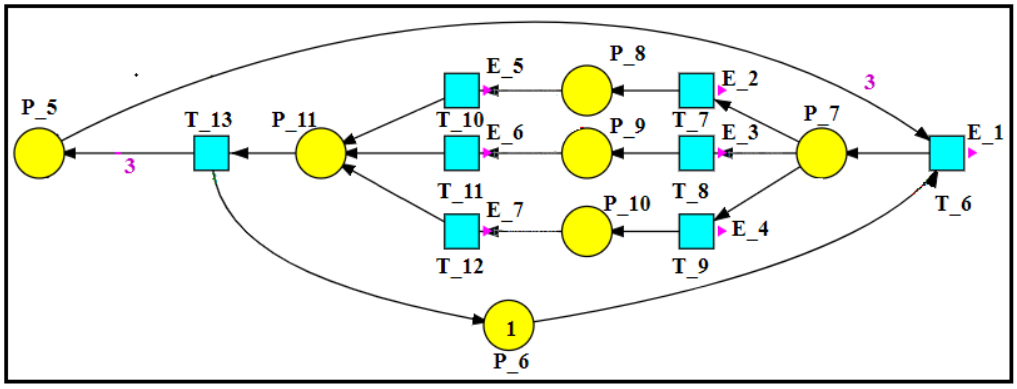

Figura 3-9 - Rede de Petri para modelar o comportamento de escolher o tipo de modulação utilizado no emissor.

Tabela 3-5 - Identificação dos eventos de saída e expressões utilizadas nas guardas de cada transição.

Identificação da transição	Expressão utilizada na guarda de transição	Eventos de saída
T_6	EscModelacao=1 ∧ tx_changemode=0	Tx_hwdspEnd
T_7	M_qpsk=1	Tx_imp_qpsk
T_8	M_qam=1	Tx_imp_qam
T_9	M_others=1	Tx_imp_other
T_10	impQpskEnd=1	Tx_hwdspStart
T_11	impQamEnd=1	Tx_hwdspStart
T_12	impOtherEnd=1	Tx_hwdspStart

Na Figura 3-10 apresenta-se a sub-rede utilizada para modelar o início e fim do da utilização do controlador do emissor, suspendendo todas as outras sub-redes. Na prática este comportamento é im-

portante para o caso de ser necessário alterar a configuração implementada na FPGA. Na Tabela 3-6 indicam-se os eventos de entrada, saída e as expressões utilizadas nas guardas de cada transição e o sinal de saída utilizado no lugar.

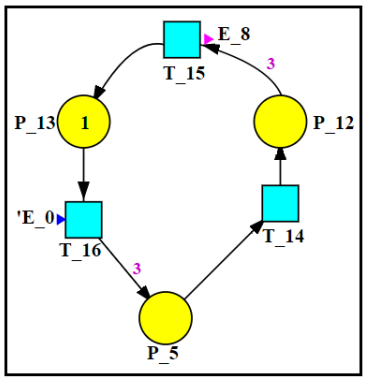

Figura 3-10 - Rede de Petri para modelar o comportamento de iniciar ou interromper o emissor.

Tabela 3-6 - Indicação dos eventos de entrada, saída e as expressões utilizadas nas guardas de cada transição e o sinal de saída utilizado no lugar.

Identificação do nó da rede	Expressão utilizada na guarda de transição	Sinais de saída	Eventos de entrada	Eventos de saída
T_14	tx_changemode=1			
T_15				Tx_stopped
T_16			Tx_start	
P_13		Rx_ready		

Na Figura 3-11 apresenta-se a sub-rede utilizada para detectar a existência de sinais modulados para guardar em software. Este comportamento não está dependente do utilizador, assim quando o controlador implementado em hardware sinaliza a existência de sinais para guardar em software, os dados são gravados automaticamente. Na Tabela 3-7 indicam-se as expressões utilizadas nas guardas de cada transição e o sinal de saída utilizado no lugar.

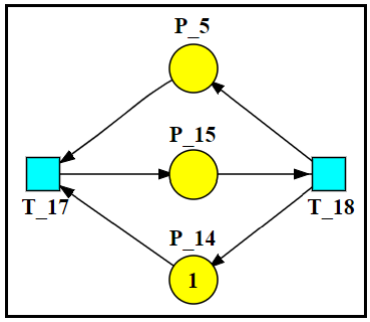

Figura 3-11 - Rede de Petri para modelar o comportamento de guardar novos sinais.

Tabela 3-7 - Identificação das expressões utilizadas nas guardas de cada transição e o sinal de saída utilizado no lugar.

Identificação do nó da rede	Expressão utilizada na guarda de transição	Sinais de saída
T_17	Hwtx_stored_sig=1 ∧ EscModelacao=0 ∧ tx_changemode=0	
T_18	Hwtx_end_sig=1	
P_15		Hwtx_read_sig

Na Figura 3-12 apresenta-se a sub-rede utilizada para modular as mensagens definidas pelo utilizador. O funcionamento da sub-rede inicia-se com a escrita e submissão da mensagem na aplicação web por parte do utilizador e parte da sub-rede implementada em software divide a mensagem de forma a enviar oito bits de cada vez para a partição hardware.

Em seguida, a parte da sub-rede imlpementada em hardware sinaliza ao *buffer* para guardar a mensagem e quando o módulo emissor estiver no início de modulação de um pacote de mensagem, transfere-se a mensagem armazenada. Enquanto a mensagem está a ser modulada emite-se um sinal ao *buffer* de sinais para começar a guardar o sinal e no fim de modulação do sinal emite-se um sinal para concluir a gravação do sinal.

Assim que o controlador detecta a conclusão da modulação do sinal, é enviado um sinal ao *buffer* de sinais para que se conclua a gravação do sinal e é enviado um sinal para o software para que se possa dar início ao carregamento do sinal guardado para a partição de software. Considerou-se também na sub-rede os sinais referentes ao carregamento do sinal para um conversor digital/analógico apesar de não ter sido implementado. Para concluir, se o *buffer* tiver alguma mensagem para ser modulada é repetido o processo descrito, se o *buffer* não tiver mais mensagens são retomadas as marcações iniciais da sub-rede.

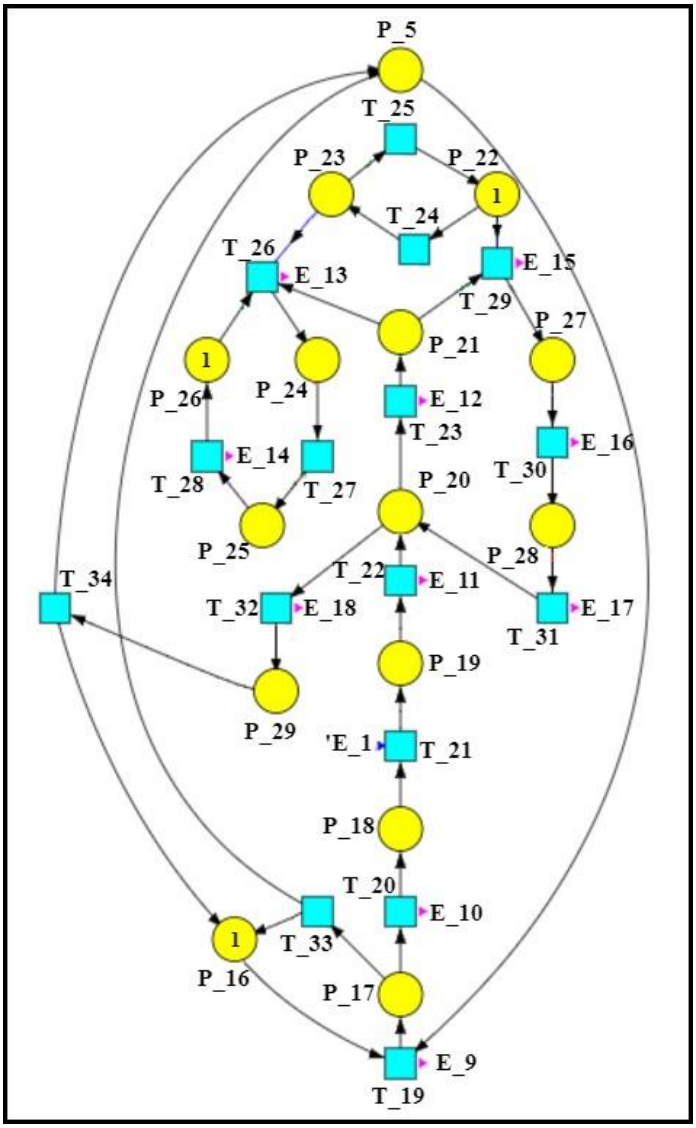

Figura 3-12 - Rede de Petri para modelar o comportamento de enviar mensagens.

Sendo que a aplicação web é executada na partição software e o emissor de mensagens implementado na partição hardware, entende-se a necessidade de unidades de controlo na partição software e na partição hardware para troca de mensagens entre estas. Inicialmente desenvolveu-se o controlador seguindo uma visão geral, a qual foi partida conforme as necessidades de controlo em cada partição do Zynq. A parte da RdP identificada como "E" foi separada em duas pelas transições T_19, T_33 e T_34. Para realizar o conceito de particionar, duplicaram se as transições mencionadas e utilizaram-se eventos de entrada e saída para que o disparo das transições seja realizado em ambas as partições.

Na Tabela 3-8 indicam-se os eventos de entrada, saída e as expressões utilizadas nas guardas de cada transição e o sinal de saída utilizado no lugar da sub-rede. Todos os outros lugares não têm sinais de saída associados.

Tabela 3-8 - Indicação dos eventos de entrada, saída e as expressões utilizadas nas guardas de cada transição e o sinal de saída utilizado no lugar.

Identificação do nó da rede	Expressão utilizada na guarda de transição	Sinais de saída	Eventos de entrada	Eventos de saída
T_19	Envia=1 ∧ EscModelacao=0 ∧ tx_changemode=0			Tx_checkMsg
T_20	Tx_msgAvailable=1			Tx_ascii2hex
T_21			Tx_start_write_buffer	
T_22	Tx_buffer_loaded=1			Tx_unbuff Tx_loadedBuff
T_23	Tx_Frame_available=1			Tx_modelFrame
T_26				Tx_saveSignal
T_27	Tx_endedModelF=1			
T_28				Tx_savedSignal
T_29	Tx_endedModelF=1			Tx_DAC
T_30	Tx_dac_end=1			Tx_bit2antena
T_31				Tx_unbuff
T_32	Tx_Frame_unavailable=1			Tx_empty_frame
P_19		Tx_write_buffer		

3.2.3 Descrição da RdP referente ao controlador para o recetor

A modelação dos comportamentos realizou-se com a utilização da rede de Petri apresentada na Figura 3-13, identificados com cores diferentes, sendo estes: a visualização do espectro de frequências do sinal na aplicação web (A) com a cor amarelo, dar início ou interromper o funcionamento do recetor (B) com a cor vermelho, escolher o tipo de modelação integrado no receptor (C) com a cor laranja, a chegada de nova mensagem (D) com a cor rosa e respectivos sinais (E) com a cor azul-claro, para

que ambos sejam guardados em software, a visualização de forma gráfica de novos sinais recebidos (F) com a cor azul-escuro e actualizar as mensagens recebidas na aplicação web (G) com a cor verde.

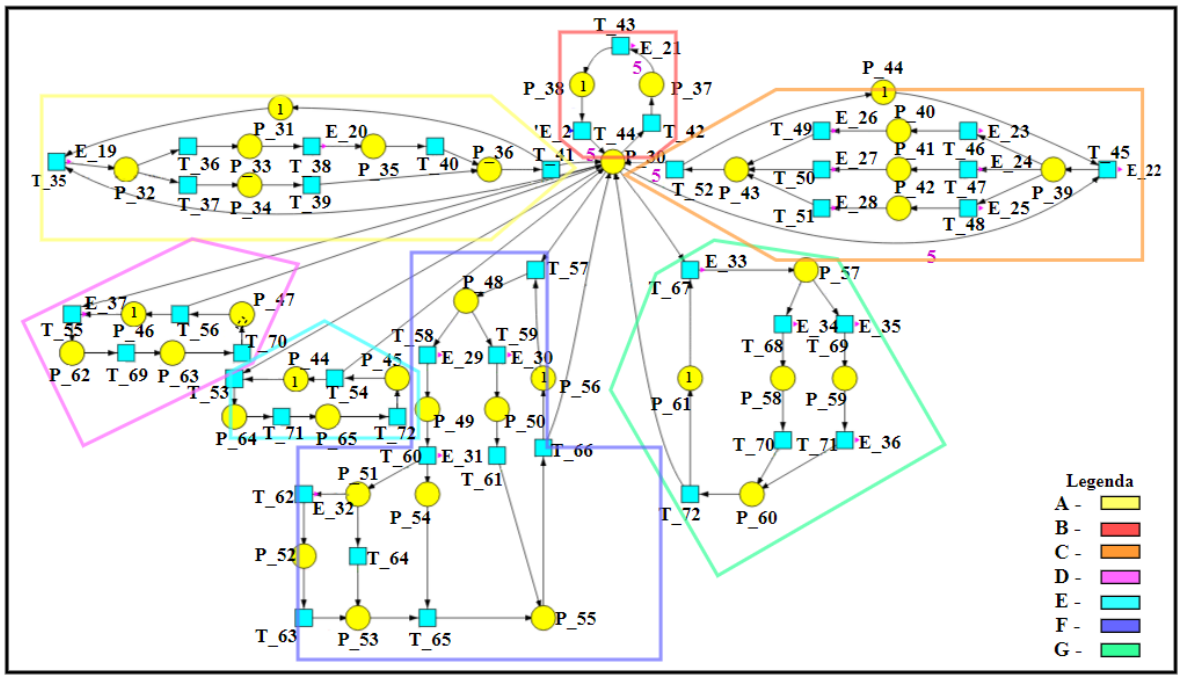

Figura 3-13 - Rede de Petri referente ao controlador do recetor.

Nos Anexos as Tabela 7-5, Tabela 7-6, Tabela 7-7 e Tabela 7-8 apresentam a descrição dos sinais de entrada, sinais de saída, eventos de entrada e eventos de saída respetivamente. Nas tabelas identifica-se também a fonte e destino de cada sinal e evento.

Para a compreensão do modelo desenvolvido separou-se cada comportamento em sub-redes para facilitar a descrição. De acordo com a legenda dos nós e eventos utilizados nas figuras, apresentam-se as tabelas com a indicação dos sinais de saída utilizados em cada lugar, as expressões utilizadas nas guardas de transição e os eventos de entrada e saída utilizados em cada transição.

Na Figura 3-14 apresenta-se a sub-rede utilizada para modelar o comportamento de mostrar na aplicação web o sinal recebido no espectro de frequência. Esta funcionalidade não foi implementada, no entanto o funcionamento seria semelhante à RdP apresentada na Figura 3-8. Na Tabela 3-9 indicam-se os sinais de saída utilizados nos lugares, os eventos de saída e as expressões utilizadas nas guardas de cada transição.

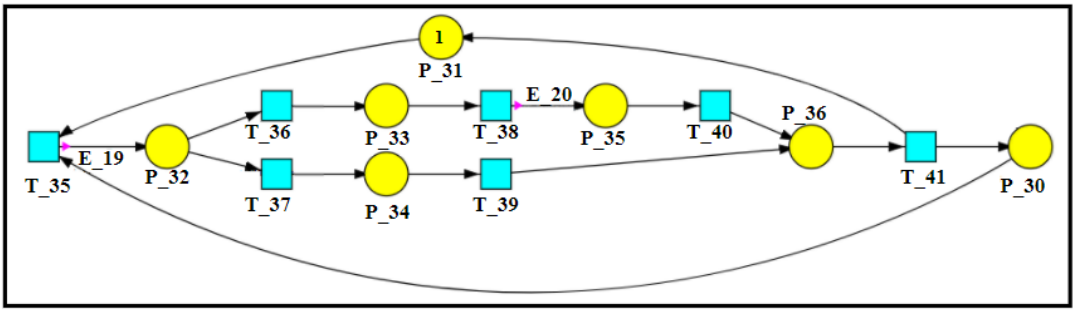

Figura 3-14 - Rede de Petri para modelar o comportamento de mostrar o espectro de frequencia do sinal recebido.

Tabela 3-9 - Indicação dos eventos de saída e as expressões utilizadas nas guardas de cada transição e os sinais de saída utilizados em cada lugar.

Identificação do nó da rede	Expressão utilizada na guarda de transição	Sinais de saída	Eventos de saída
T_35	Rx_esc_espectro=1 ∧ Rx_esc_mod=0 ∧ rx_changemode=0		Tx_checkSig
T_36	Rx_sig_available=1		
T_37	Rx_sig_unavailable=1		
T_38	Rx_loadedEspec=1		Rx_printEspec
P_32		Rx_check_new_sig	
P_33		Rx_app_espec	

Na Figura 3-15 apresenta-se sub-rede utilizada para modelar o início e fim do da utilização do controlador do recetor, suspendendo todas as outras sub-redes. A importancia desta sub-rede é a mesma da sub-rede apresentada na Figura 3-10. Na

Tabela 3-10 indicam-se os sinais de saída utilizados nos lugares, os eventos de entrada, saída e as expressões utilizadas nas guardas de cada transição.

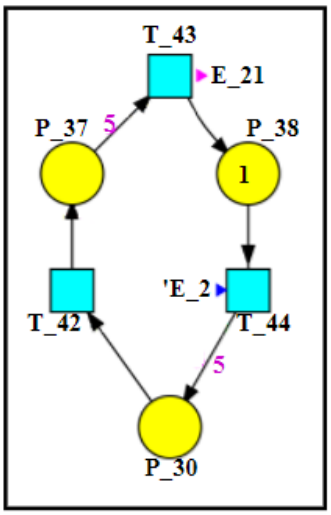

Figura 3-15 - Rede de Petri para modelar o comportamento de iniciar ou interromper o recetor

Tabela 3-10 - Indicação dos eventos de entrada, saída e as expressões utilizadas nas guardas de cada transição e os sinais de saída utilizados em cada lugar.

Identificação do nó da rede	Expressão utilizada na guarda de transição	Sinais de saída	Eventos de entrada	Eventos de saída
T_42	rx_changemode=1			
T_43				Rx_Stopped
T_44			Rx_start	
P_38		Rx_ready		Rx_printEspec

Na Figura 3-16 apresenta-se a sub-rede utilizada para escolher qual o tipo de desmodulação que está em funcionamento. A sub-rede foi desenvolvida segundo o mesmo princípio da sub-rede apresentada na Figura 3-9 e por isso o seu funcionamento é homólogo. Na

Tabela 3-11 indicam-se os eventos de saída e as expressões utilizadas nas guardas de cada transição.

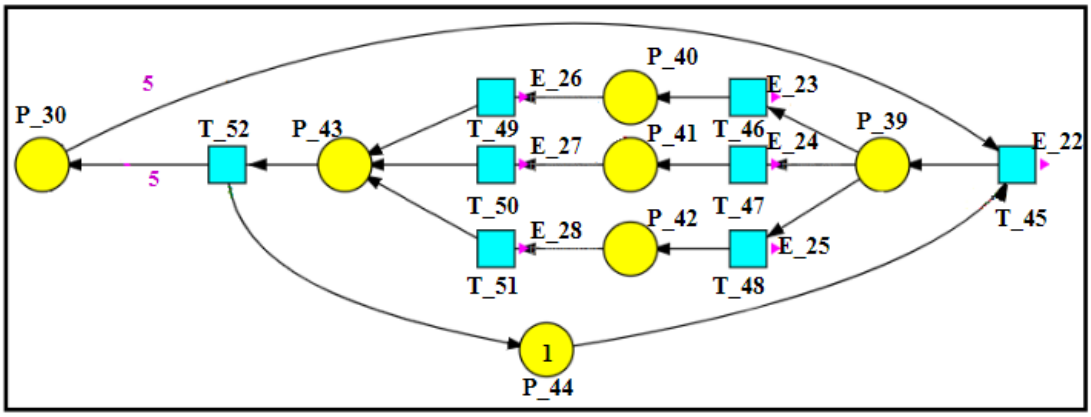

Figura 3-16 - Rede de Petri para modelar o comportamento de escolher o tipo de modelação utilizado no recetor.

Tabela 3-11 - Indicação dos eventos de saída e as expressões utilizadas nas guardas de cada transição.

Identificação da transição	Expressão utilizada na guarda de transição	Eventos de saída
T_45	EscModelacao=1 ∧ tx_changemode=0	rx_hwdspEnd
T_46	Rx_m_qpsk=1	rx_imp_qpsk
T_47	Rx_m_qam=1	rx_imp_qam
T_48	Rx_m_others=1	rx_imp_other
T_49	Rx_impQpskEnd=1	rx_hwdspStart
T_50	Rx_impQamEnd=1	rx_hwdspStart
T_51	Rx_impOtherEnd=1	rx_hwdspStart

Na Figura 3-17 apresenta-se a sub-rede utilizada para detectar a existência de sinais desmodulados para guardar em software. A descrição do funcionamento da sub-rede é identico ao funcionamento da sub-rede apresentada na Figura 3-11, mas neste caso são os sinais recebidos no módulo recetor. Neste projeto, os sinais são iguais aos sinais guardados no *buffer* referente ao emissor porque os módulos emissor e recetor estão ligados em série. No entanto para o caso de se utilizarem antenas entre os módulos emissor e recetor, considerou-se importante esta sub-rede para o caso de se querer comparar os sinais enviados e recebidos. Na

Tabela 3-12 indicam-se as expressões utilizadas nas guardas de cada transição e o sinal de saída utilizado no lugar.

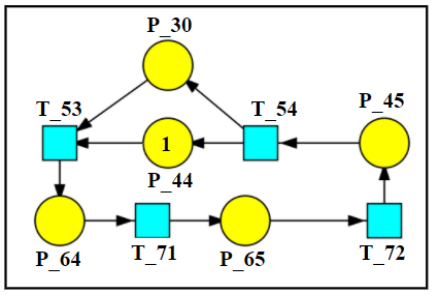

Figura 3-17 - Rede de Petri para modelar o comportamento de guardar novos sinais recebidos.

Tabela 3-12 - Indicação das expressões utilizadas nas guardas de cada transição e o sinal de saída utilizado no lugar.

Identificação do nó da rede	Expressão utilizada na guarda de transição	Sinais de saída
T_53	rxhw_sig_stored=1 ∧ rx_esc_mod=0 ∧ rx_changemode=0	
P_45		readSig_hw

Na Figura 3-18 apresenta-se a sub-rede utilizada para detectar a existência de mensagens recebidas para guardar em software. O funcionamento da sub-rede é idêntico ao funcionamento da sub-rede apresentada na Figura 3-17. Neste caso a utilização da sub-rede considera-se essencial para que todas as mensagens recebidas sejam guardadas em software, para que sejam comparadas às mensagens enviadas. Na Tabela 3-3 indicam-se as expressões utilizadas nas guardas de cada transição e o sinal de saída utilizado no lugar.

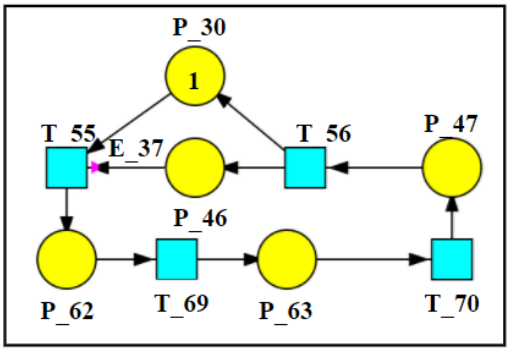

Figura 3-18 - Rede de Petri para modelar o comportamento de guardar novas mensagens recebidas.

Tabela 3-13 - Indicação das expressões utilizadas nas guardas de cada transição e o sinal de saída utilizado no lugar.

Identificação do nó da rede	Expressão utilizada na guarda de transição	Sinais de saída
T_55	rxhw_msg_stored=1 ∧ rx_esc_mod=0 ∧ rx_changemode=0	
P_47		readMsg_hw

Na Figura 3-19 apresenta-se a sub-rede utilizada para modelar o comportamento de mostrar na aplicação web o sinal recebido e considerou-se a possibilidade de em trabalhos futuros ser utilizado um desmodelador FM para ouvir sinais de rádio. Assim sendo, o funcionamento da secção da sub-rede inicia-se pelo pedido do utilizador para actualizar o gráfico de sinais recebidos no caso de existirem novos sinais guardados.

Na Tabela 3-14 indicam-se os sinais de saída utilizados nos lugares, os eventos de saída e as expressões utilizadas nas guardas de cada transição.

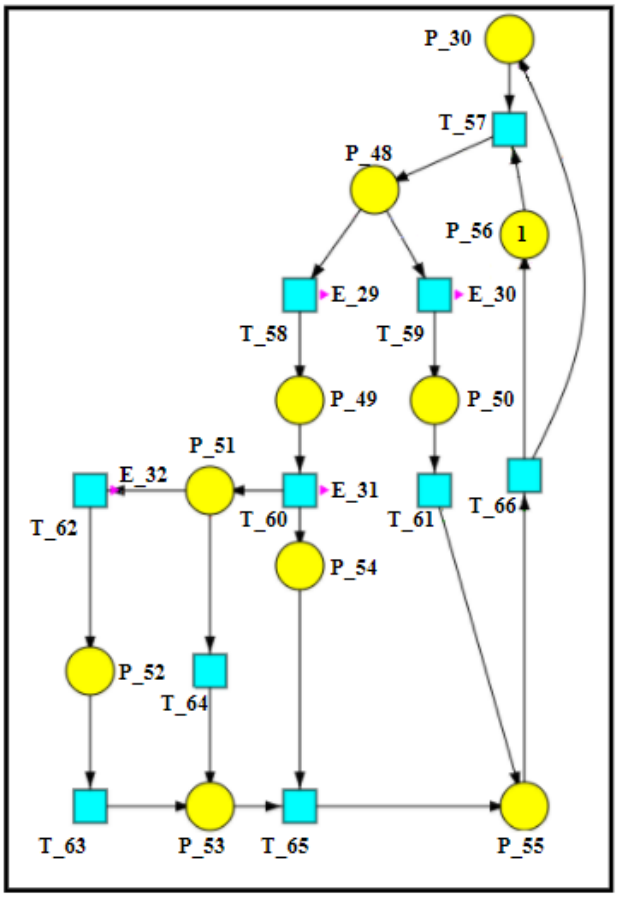

Figura 3-19 - Rede de Petri para modelar o comportamento de mostrar o sinal recebido ao utilizador.

Tabela 3-14 - Indicação dos eventos de saída e as expressões utilizadas nas guardas de cada transição e os sinais de saída utilizados em cada lugar.

Identificação do nó da rede	Expressão utilizada na guarda de transição	Sinais de saída	Eventos de saída
T_57	Rx_esc_view=1 ∧ rx_esc_mod=0 ∧ rx_changemode=0		
T_58	Rx_sig_available=1		Rx_loadSig
T_59	Rx_sig_unavailable=1		Rx_printNoSig
T_60			Rx_printSig
T_62	Rx_listenSig=1		Rx_sig2Listen
T_63	Rx_ListenEnd=1		
T_64	Rx_ListenSig=0		
P_48		Check_sigs_file	
P_52		Rx_audio_sig	

Na Figura 3-20 apresenta-se a sub-rede utilizada para modelar o comportamento de mostrar ao utilizador pela aplicação web as novas mensagens recebidas. O funcionamento da sub-rede inicia-se com o pedido do utilizador de actualizar as mensagens recebidas na aplicação web e no caso da existencia de novas mensagens guardadas, são então apresentadas no bloco de texto referente. Esta sub-rede é implementada apenas na partição software do dispositivo. Na Tabela 3-15 indicam-se os sinais de saída utilizados nos lugares, os eventos de saída e as expressões utilizadas nas guardas de cada transição.

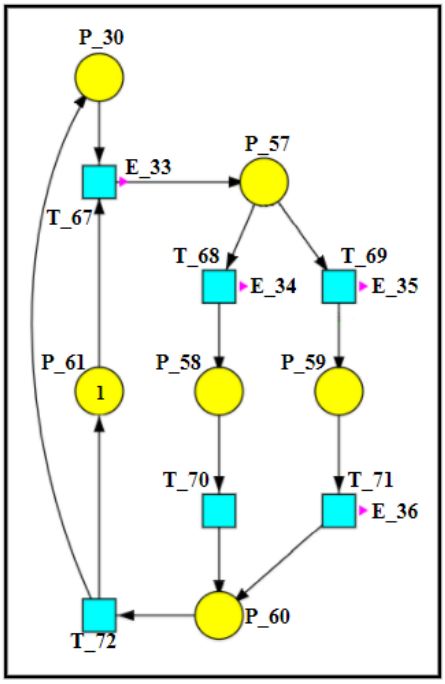

Figura 3-20 - Rede de Petri para modelar o comportamento para mostrar nova mensagem recebida ao utilizador.

Tabela 3-15 - Indicação dos eventos de saída e as expressões utilizadas nas guardas de cada transição e os sinais de saída utilizados em cada lugar.

Identificação do nó da rede	Expressão utilizada na guarda de transição	Sinais de saída	Eventos de saída
T_67	Rx_esc_msg=1 ∧ rx_esc_mod=0 ∧ rx_changemode=0		
T_68	Rx_MsgUnavailable=1		Rx_printNoMsg
T_69	Rx_MsgAvailable=1		Rx_loadFrame
T_71			Rx_printMsg
P_57		Rx_checkMsg_available	

À semelhança do controlador referente ao emissor de mensagens desenvolveu-se o controlador seguindo uma visão geral, que posteriormente se dividiu para se implementar em cada partição do Zynq. No controlador do recetor a parte da RdP "A" utilizada em hardware foi da transição T_36 à T_38, a parte da RdP "E" implementou-se em hardware pelos nós desde a transição T_55 à T_70, a

parte da RdP “F” implementou-se em hardware os nós desde a transição T_62 à T_63 e a parte da RdP “D” implementou-se em hardware pelos nós desde a transição T_53 à T_72.

3.3 Desenvolvimento do DSP com a ferramenta System Generator for DSP

3.3.1 Desenvolvimento do emissor

Para o desenvolvimento do emissor QPSK utilizou-se o exemplo disponibilizado pelo Matlab como base. A utilização deste modelo viabiliza a implementação em hardware por estar optimizado para este efeito. O modelo é constituído por dois módulos, sendo estes o módulo emissor QPSK e o módulo para medir a qualidade dos sinais gerados.

Na Figura 3-21 é apresentado o exemplo mencionado no parágrafo anterior. O módulo preenchido a verde é referente ao emissor QPSK otimizado para HDL e o módulo preenchido a azul é referente à medição da qualidade do sinal à saída do emissor.

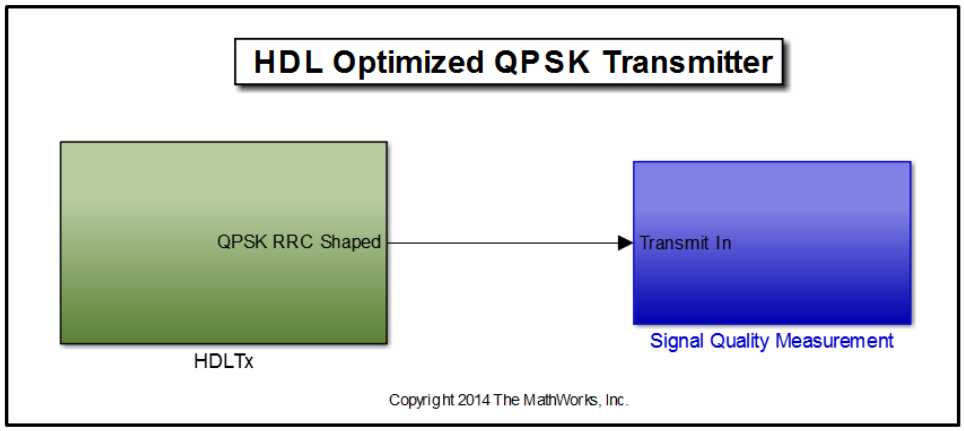

Figura 3-21 - Modelo HDL Optimized QPSK Transmitter, imagem retirada da ferramenta Matlab.

O bloco Signal Quality Measurement corresponde ao bloco de medição da qualidade do sinal, apresentando a localização geométrica dos símbolos gerados para validar a qualidade do emissor, como exemplo na Figura 3-22. Os símbolos recebidos são identificados pela cor verde e como se pode verificar estão localizados dentro dos quadrados de cor vermelha, o que garante a qualidade do emissor. Este módulo utiliza um filtro RRC (Root Raised Cosine) para filtrar os sinais recebidos, no entanto este módulo não foi utilizado ao gerar o código VHDL do sistema, porque a sua função é de validar o modelo de uma forma gráfica, utilizando um diagrama de constelação.

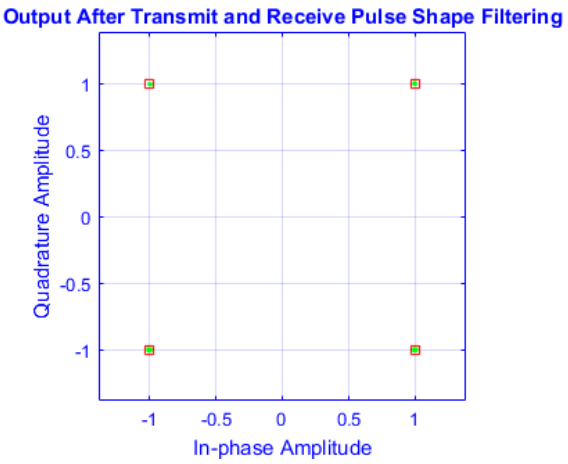

Figura 3-22 - Diagrama da constelação QPSK integrado no módulo de medição da qualidade do sinal.

O sistema HDLTx, apresentado na Figura 3-23, é constituído por três subsistemas, nomeadamente: a fonte de dados e construção de pacotes de informação, o mapeamento dos símbolos QPSK e a modulação do sinal a partir dos símbolos gerados utilizando um filtro do tipo *Root Raised Cosine*.

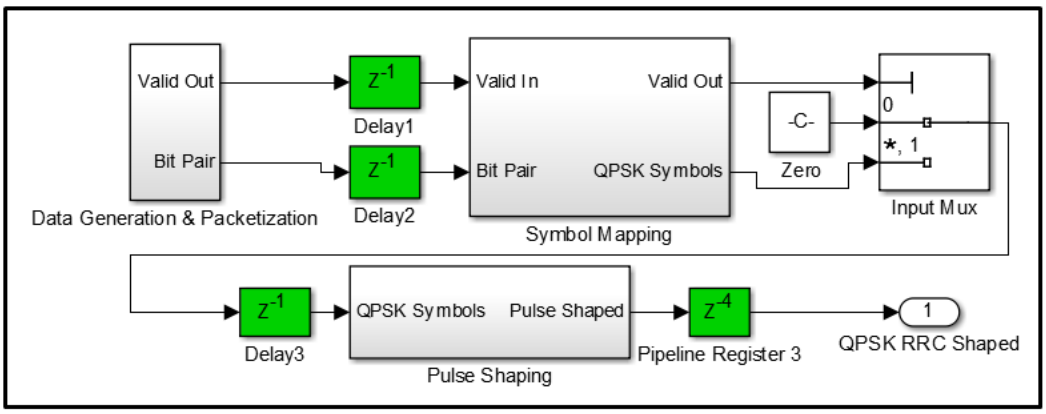

Figura 3-23 - Diagrama de blocos referente ao módulo HDLTx, imagem retirada de (The MathWorks, 2015b)**.**

Para possibilitar a escolha da mensagem a enviar, alterou-se o módulo Data Generation & Packetization. O diagrama de blocos referente ao módulo mencionado apresenta-se na Figura 3-24.

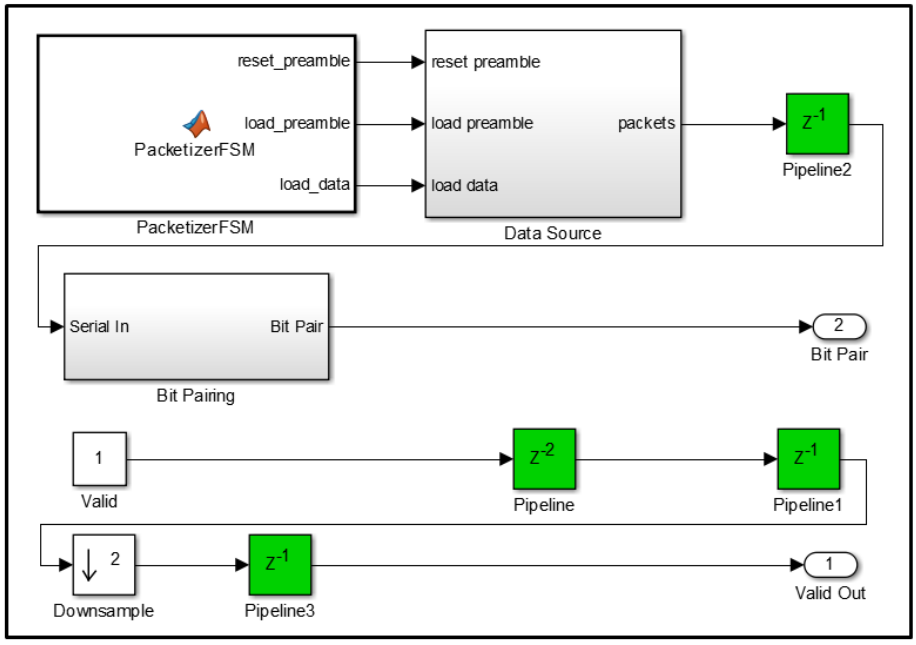

Figura 3-24 – Diagrama de blocos referente ao módulo "Data Generation & Packetization" presente no sistema HDLTx, imagem retirada de (The MathWorks, 2015b)**.**

O módulo *PacketizerFSM* integra uma função programada à semelhança de uma máquina de Moore. Neste sistema estão definidos dois estados, sendo estes o estado de preâmbulo e o estado para anexar dados ao pacote de informação. A mudança entre os estados altera o modo de funcionamento do módulo Data Source. É definido que o estado de preambulo está ativo primeiro e com a duração de 26 ciclos, para que o Barker code seja introduzido no início de cada pacote de informação. O Barker code predefinido utiliza a sequência de 13 bits, mas utilizando 26 bits e considerando o bit -1 como 0. O Barker code utilizado é apresentado na Tabela 3-16. De seguida o sistema segue para o estado em que os dados são anexados ao pacote de informação, no qual permanece durante 174 ciclos.

Tabela 3-16 - Barker code 13 bits adaptado a 26 bits.

Barker code utilizando 13 bits	1 1 1 1 1 0 0 1 1 0 1 0 1
Barker code utilizando 26 bits	1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 1 1 0 0 1 1

O funcionamento do módulo Data Source consiste na construção do pacote de informação utilizando a string "Hello World 0" seguido da numeração do pacote de 0 a 99. Para possibilitar o envio de mensagens escolhidas pelo utilizador removeu-se a LUT contornada com o quadrado de cor verde na Figura 3-25. Foi adicionada uma porta de entrada ao sistema HDLTx, para que o utilizador introduza um vetor com os seus dados previamente convertidos em bits, e um módulo integrando uma função

matlab para ler o valor do vetor na posição escolhida. As alterações mencionadas apresentam-se na Figura 3-26. Alteraram-se os limites de contagem do *data counter* para iniciar na posição 1 e acabar em 174, para que não seja necessário ter como sinal de entrada um vetor de grandes dimensões. No entanto, se o tamanho da mensagem for superior ao tamanho de um pacote de informação, a mensagem é particionada no buffer de mensagens e forma-se uma fila para emitir cada parte da mensagem separadamente.

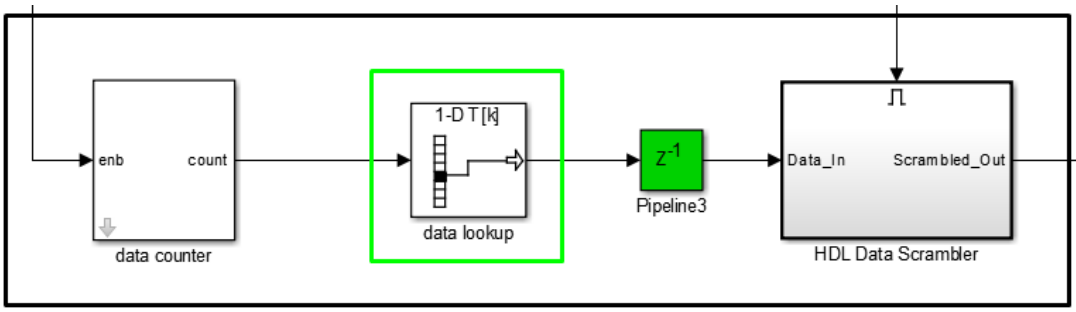

Figura 3-25 – LUT que foi removida do modelo base HDLTx, imagem alterada de (The MathWorks, 2015b)**.**

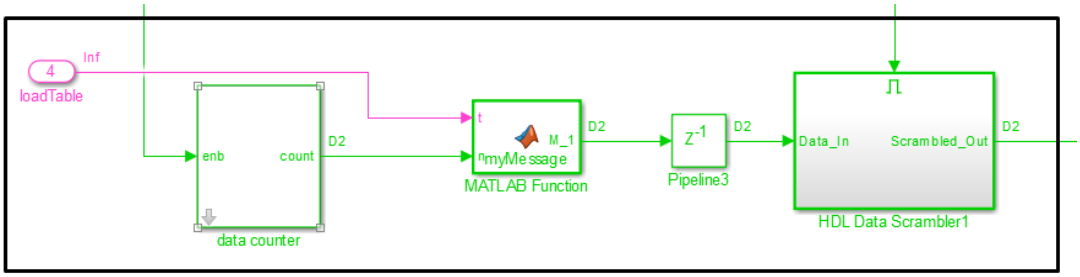

Figura 3-26 - Alterações realizadas ao modelo para que o utilizador possa introduzir a sua mensagem.

Com a geração do código VHDL concluída resultou o modelo apresentado na Figura 3-27, e a lista de recursos utilizados pelo modelo é apresentada na Figura 3-28 a qual é gerada de forma automática pelo HDL Workflow Advisor, selecionando a opção avançada para gerar um relatório para visualização na web.

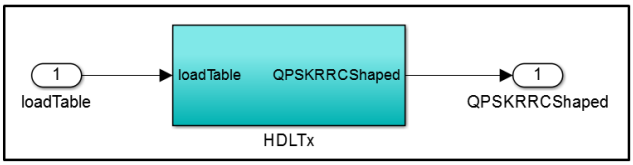

Figura 3-27 - Modelo final do emissor QPSK, desenvolvido a partir do modelo base do Matlab.

Multipliers	22
Adders/Subtractors	24
Registers	205
RAMs	0
Multiplexers	24

Figura 3-28 - Lista de recursos utilizados pelo modelo, gerada pela ferramenta HDL Workflow Advisor.

3.3.2 Desenvolvimento do receptor

Para o desenvolvimento do receptor QPSK utilizou-se como base um exemplo disponível no catálogo de modelos do Matlab. A utilização deste modelo viabiliza a implementação em hardware por estar optimizado para este efeito. O modelo é constituído pelo módulo de amostras de dados, o módulo receptor QPSK, o módulo de descodificação e impressão da mensagem na linha de comandos do Matlab e o módulo com os parâmetros de configuração do recetor QPSK.

Na Figura 3-29 é apresentado o exemplo mencionado no parágrafo anterior. O módulo preenchido a verde é referente ao recetor QPSK otimizado para HDL e o módulo preenchido a amarelo é referente aos parametros de configuração do receptor QPSK.

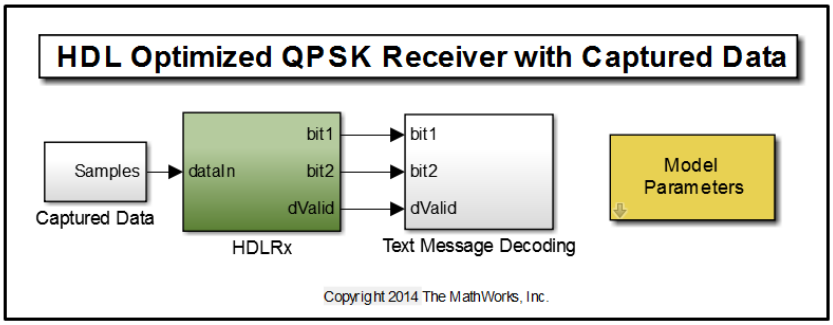

Figura 3-29 - Modelo HDL Optimized QPSK Receiver with Captured Data, imagem retirada de (The MathWorks, 2015a)**.**

O sistema interno do módulo HDLRx é constituído por sete subsistemas e apresenta-se na Figura 3-30. Os módulos apresentados são identificados por: Controlo de ganho automático, Filtro RRC, Compensador de frequência abrupto, Compensador de frequência fino, Recuperação temporal, Descodificador de símbolos e um módulo para apresentar os símbolos sincronizados com o módulo de Recuperação temporal.

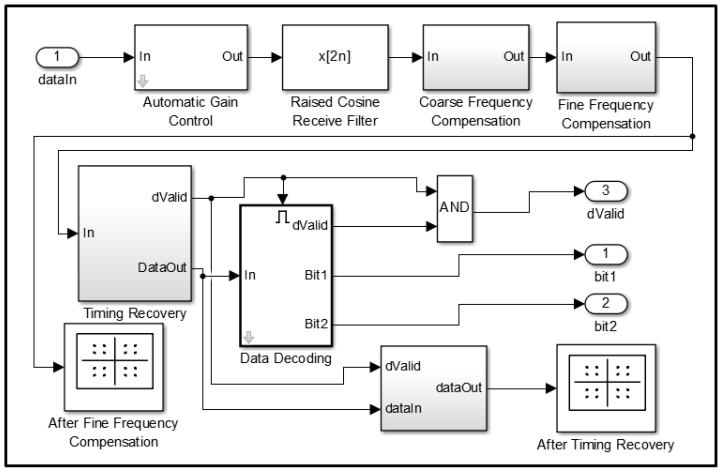

Figura 3-30 - Sistema interno do modelo HDLRx QPSK, imagem retirada de (The MathWorks, 2015a)**.**

O subsistema que recebe o sinal de entrada, o controlador de ganho, é responsavel por assegurar que o sinal de entrada é estável fixando a amplitude do sinal para o subsistema seguinte com o valor de 1/2. Desta forma garante-se que os ganhos nos módulos de detecção de erros de fase e de Recuperação temporal se mantêm. Além disso, este subsistema é utilizado antes do filtro de interpolação para que a amplitude do sinal seja medida com um factor de sobreamostragem de 4.

O funcionamento do subsistema *Root Raised Cosine Filter* utiliza um factor de decimação dois ao sinal de entrada, o que equivale a considerar os valores do sinal de entrada de duas em duas amostras. O filtro utiliza um factor de *rollof* β=0.5 definido como o valor intermédio para reduzir o excesso da largura de banda do filtro e a enlongação do impulso à saída do filtro, melhorando assim a relação sinal/ruído o que facilita o processamento do sinal aos subsistemas seguintes.

De seguida o sinal passa pelo compensador de frequencia abrupto, o qual gera um *offset* de frequência estimado com base no algoritmo de Luise (Luise & Reggiannini, 1995). A utlização deste algoritmo sugere reduzir o numero de recursos necessários em hardware comparativamente ao algoritmo FFT.

De forma a continuar a compensação de frequencia segue-se o compensador de frequencia fino. Neste caso é utilizado um PLL(Phase Locked Loop) para identificar e corrigir o offset de frequencias residuais e o offset associado à fase do sinal de entrada. A presença de um offset relativo às frequencias residuais é identificada por uma rotação lenta dos símbolos QPSK, a qual pode ser observada no gráfico da constelação.

Consequentemente são identificados e corrigidos os erros de temporização do sinal utilizando outro PLL. Dependendo do atraso detectado entre o emissor e o recetor é alterada a sequencia de valo-

res do sinal de saída “dValid”, identificando os instantes óptimos para a tomada de decisões acerca dos símbolos recebidos.

Do sistema *Data decoding* de texto considerou-se apenas a utilização do subsistema Dataframer, apresentado na Figura 3-31 pelas razões que se seguem no parágrafo seguinte. O funcionamento do subsistema Dataframer inicia-se com a confirmação do *barker code* introduzido no início do pacote de dados. Posteriormente segue-se um subsistema para resolver eventuais problemas de ambiguidade na identificação da fase do sinal de entrada devido à utilização do PLL no Compensador de frequencia fina para corrigir o offsets referentes à fase dos sinais de entrada.

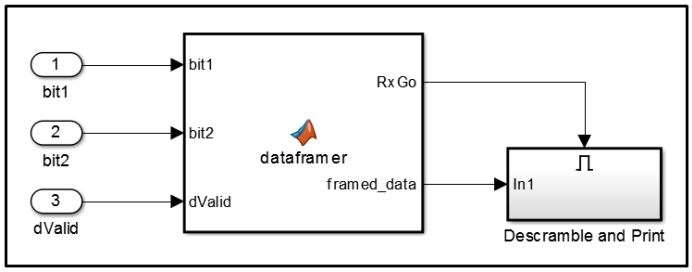

Figura 3-31 - Sistema interno do módulo Data decoding, imagem retirada de (The MathWorks, 2015a)**.**

Para a implementação em hardware escolheram-se os módulos HDLRx e *Data Framer*. O módulo *Descrambler* não tem compatibilidade com o gerador de código HDL, por isso decidiu-se desenvolver este módulo utilizando como base o módulo *Scrambler* apresentado no sistema interno do HDLTx. O desenvolvimento do *Descrambler* realizou-se no ambiente do Simulink, que se apresenta na Figura 3-32. O comportamento do módulo *Descrambler* foi testado, utilizando os modelos de *Scrambler* e *Descrambler* ligados em série para verificar que a mensagem na porta de entrada do *Scrambler* é igual à mensagem apresentada na porta de saída do *Descrambler*.

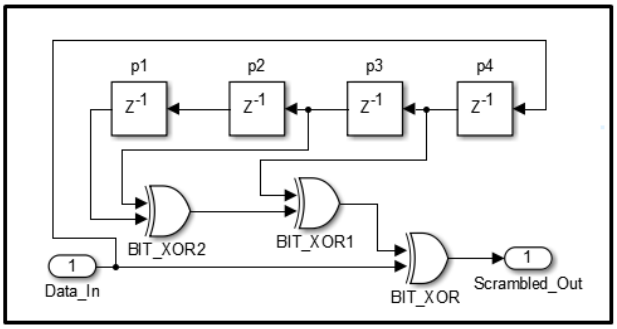

Figura 3-32 - Sistema referente ao módulo *Descrambler* para utilização no recetor QPSK.

Posteriormente foi gerado o código VHDL referente aos módulos HDLTx, Data Framer e do *Descrambler* desenvolvido. Para este efeito não houve necessidade de alterar os módulos *Data Framer* e *Descrambler*. No entanto, para o módulo HDLRx surgiram alguns erros devido ao tipo de dados utilizado no sistema interno do módulo de *Automatic Gain Control*, nomeadamente no módulo identificado por *Modulus*. Neste sentido inclui-se um bloco de conversão de dados onde se identificou o erro, como está apresentado na Figura 3-33. Com a utilização deste bloco identificaram-se situações de overflow e saturação na receção das primeiras duas mensagens, mas funcionando corretamente de seguida, o que sucede também na utilização do modelo tal e qual como disponibilizado no catálogo de modelos do Matlab. Com esta alteração conseguiu-se utilizar o modelo para gerar o respectivo código VHDL.

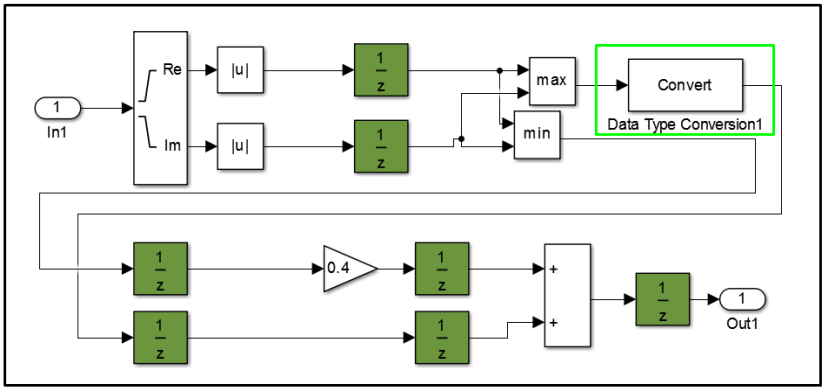

Figura 3-33 - Utilização de um bloco conversor de dados no sistema interno do bloco de Controlo de ganho automático.

Com a utilização do gerador de código VHDL concluída apresenta-se na Figura 3-34 o modelo utilizado e na Figura 3-35 a lista de recursos utilizados, seguindo a mesma metodologia utilizada para gerar a lista de recursos utilizados pelo emissor QPSK. Os sinais de saída do modelo são referentes à existência de uma nova mensagem recebida no sinal Out1 e o sinal Out2 apresensta os valores dos bits descodificados, um bit por instante de tempo.

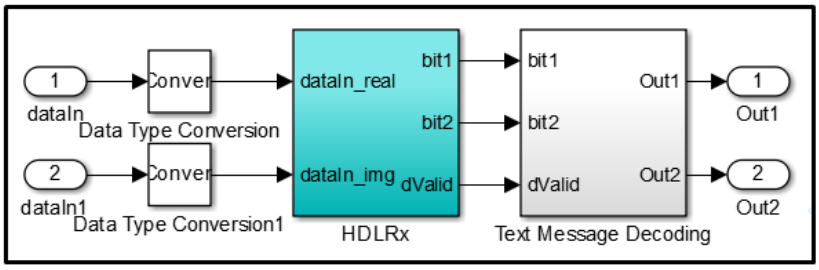

Figura 3-34 - Modelo utilizado para gerar o código VHDL.

Multipliers	122
Adders/Subtractors	247
Registers	817
RAMs	0
Multiplexers	521

Figura 3-35 - Lista de recursos necessários para utilização do modelo em hardware.

3.4 Implementação do DSP e controlador IOPT no Vivado

3.4.1 Implementação do DSP

Para implementar os blocos IP gerados em VHDL de forma automatica no ambiente de modelação simulink surgiu a necessidade de desenvolver o bloco sincronizador de mensagens com o emissor QPSK, apresentado na Figura 3-36, tem como objetivo detectar a sinalização do controlador IOPT pedir a modelação de uma mensagem e colocar a mensagem na entrada do emissor QPSK, quando este está no ciclo de criar uma nova mensagem. O sinal "data_scrambled" identifica que todos os 174 bits da mensagem foram modelados e enviados pelo emissor QPSK. A identificação do novo ciclo de escrita no emissor QPSK é detectado pelo valor do contador "sync_ok" que é atualizado pelo sinal definido no módulo HDLTx/Data_Generation/Data_Soure/Matlab_Function.

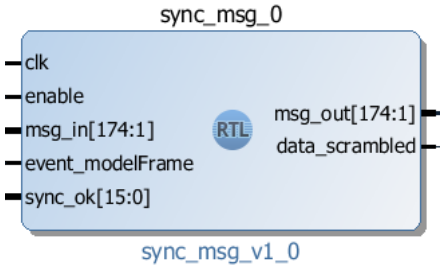

Figura 3-36 - Bloco IP sincronizador de mensagens com o emissor QPSK.

O bloco IP Check vetor, apresentado na Figura 3-37, tem o seu funcionamento dependente do controlador IOPT com a sinalização de chegada de uma nova mensagem. Este módulo tem como funções inicializar o polinómio utilizado no módulo Descrambler no início de implementação, enviar cada bit da mensagem ordenadamente para o módulo Descrambler e reiniciar o módulo no fim do funcionamento. O desenvolvimento deste módulo deve-se à impossibilidade de gerar o código VHDL do bloco Selector presente no modelo do recetor QPSK disponível no Matlab.

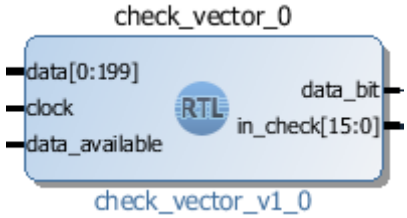

Figura 3-37 - Bloco IP Check_vector.

O bloco IP Data framed, apresentado na Figura 3-38, inicia-se com a validação do sinal "data_available" que é alterado pelo controlador IOPT referente ao recetor QPSK. O objetivo do módulo é concatenar os bits gerados pelo módulo Descrambler para que a mensagem recebida seja formada. Os sinais concatenados formam os símbolos ASCII, em que cada simbolo é representado por um conjunto de oito bits. A concatenação dos bits termina-se com a verificação da quantidade de bits introduzidos no descrambler ser igual a 174.

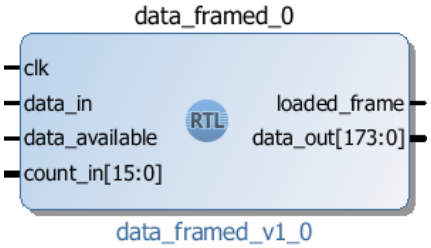

Figura 3-38 -Bloco IP Data framed.

3.4.2 Desenvolvimento de blocos IP para codificação/descodificação de sinais e *buffers*

Para a utilização dos controladores desenvolvidos nas IOPT Tools, utilizou-se um bloco IP para descodificar os sinais que são transmitidos pela partição software e um bloco IP para codificar os sinais que se pretende enviar para a partição software. Estes sinais são transmitidos entre as partições hardware/software com recurso a blocos IP de intercomunicação disponíveis no catálogo de blocos IP do Vivado. Os blocos IP utilizados para a troca de mensagens entre a partição de hardware e de software utilizam o protocolo de comunicação AXI (Advanced eXtensible Interface). Utilizou-se uma porta de *Master* no bloco IP referente à partição software e a porta *Slave* no bloco IP que está ligado à partição hardware. No caso de escrita de comandos da partição software para hardware, é enviado o comando "monitor", como indicado nas especificações da plataforma utilizada, seguido do endereço de memória da FPGA e o valor do sinal, o qual será descodificado pelo bloco IP desenvolvido

"sw_signalsto_hw", apresentado na Figura 3-39. Para o caso de leitura de sinais para a partição software é enviado o comando "monitor" seguido do endereço de memória que se pretende aceder, o qual é identificado pelo bloco IP desenvolvido "hw_signalsto_sw" que coloca na porta de saída os valores referentes ao endereço de memória que se pretende aceder. Este bloco é apresentado na Figura 3-40.

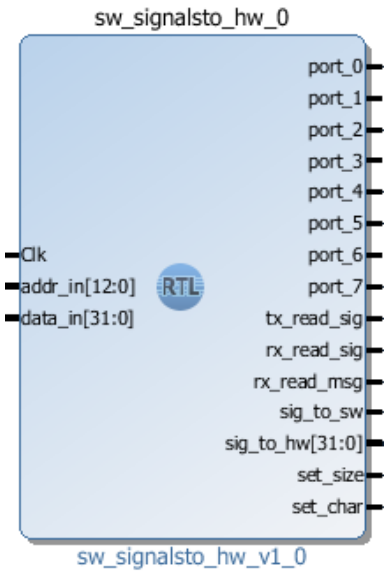

Figura 3-39 - Bloco IP descodificador de sinais recebidos da partição software.

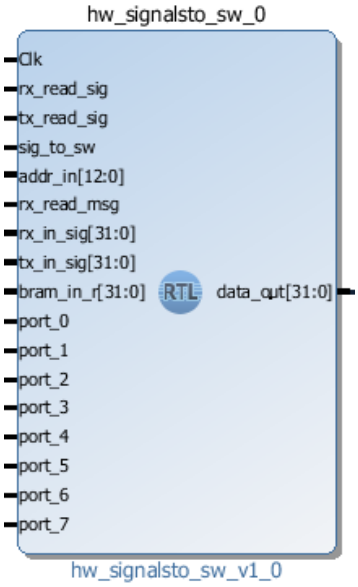

Figura 3-40 - Bloco IP codificador de sinais que se pretende enviar para a partição software.

Neste bloco IP, além dos sinais provenientes dos controladores IOPT identificam-se nas portas de entrada três sinais compostos por 32 bits, os quais são referentes aos valores guardados nos *buffer*s de sinais e mensagens recebidas e aos valores guardados no *buffer* de sinais emitidos. Uma vez que

cada valor do sinal seja enviado ou recebido é composto por 32 bits, considera-se útil a utilização de *buffer*s de sinais e mensagens. Justifica-se a necessidade destes elementos no sistema pela existência de apenas um bus de dados entre as partições hardware e software, o qual utiliza sinais de 32 bits e colocando-se a hipótese de se enviar e receber sinais ao mesmo tempo poderia perder-se informação.

Desenvolveram-se três blocos IP com função de *buffer*, sendo dois utilizados com o emissor e um utilizado com o recetor. O *buffer* de mensagens utilizado com o emissor, apresentado na Figura 3-41, tem como função guardar as mensagens chegadas da partição software para posteriormente, de forma sincronizada, colocar como mensagem de entrada no bloco Emissor.

O funcionamento do *buffer* inicia-se com a sinalização do sinal de escrita no *buffer*, "tx_write_buffer", enviado pelo controlador do emissor implementado em hardware e pela receção da mensagem proveniente do software incluindo o tamanho da mensagem que se pretende guardar. O endereço de memória utilizado para este efeito é o 18 (notação em hexadecimal).

De seguida os valores do sinal que chegam do software são concatenados num vetor até que a mensagem esteja completa. O *buffer* permite a concatenação de até 14 símbolos por segmento de mensagem, sendo cada caracter caracterizado por dois símbolos codificados na notação hexadecimal.

Por cada sinalização de escrita no *buffer* desenvolvido, permite-se guardar até 28 símbolos. Os segmentos de mensagem são transferidos para o sincronizador de mensagens à medida que o evento de unbuff ("out_evt_tx_unbuff") é disparado pelo controlador IOPT.

Quanto aos sinais de saída utilizaram-se três para sinalização do estado do *buffer* ao controlador IOPT e uma para o segmento de mensagem que se pretende enviar. Para caracterizar o estado do *buffer* utilizaram-se os sinais referentes a ter sido recebido do software todos os caracteres previstos e sinais referentes à presença, ou não, de segmentos de mensagem por colocar na entrada do emissor QPSK.

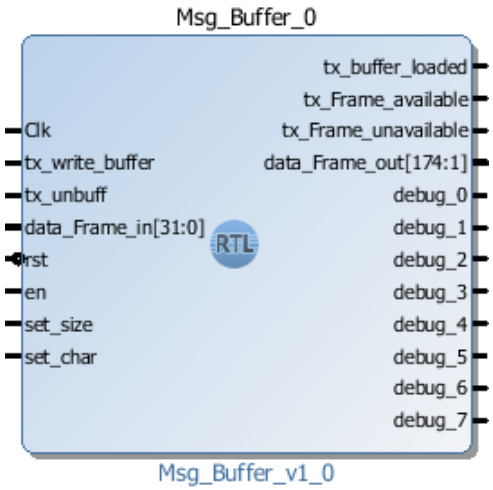

Figura 3-41 - Bloco IP *buffer* de mensagens para o emissor.

O *buffer* de sinais utilizado com o emissor QPSK, apresentado na Figura 3-42, tem como função guardar os sinais gerados referentes à mensagem que se pretende transmitir para que possam ser visualizados no ambiente gráfico desenvolvido para o utilizador. O funcionamento deste *buffer* é controlado pelo controlador referente ao emissor IOPT e pelo bloco IP de sincronização de mensagens.

A sinalização para gravar os sinais é iniciada quando o controlador identifica que o bloco emissor dá inicio à modulação. Quando o controlador identifica que os bits do segmento da mensagem estão modulados, é entao concluido o processo de gravar os sinais. Na identificação do valor um no sinal "tx_sig_r" para leitura dos sinais guardados, estes são transmitidos para o bloco "hw_signalsto_sw".

Os sinais são guardados em vetores de vetores, sendo as partes reais e imaginárias guardadas em diferentes variáveis. No processo de envio dos sinais para o bloco "hw_signalsto_sw", os sinais são concatenados 16 bits de cada parte, formando os vetores de sinais de 32 bits.

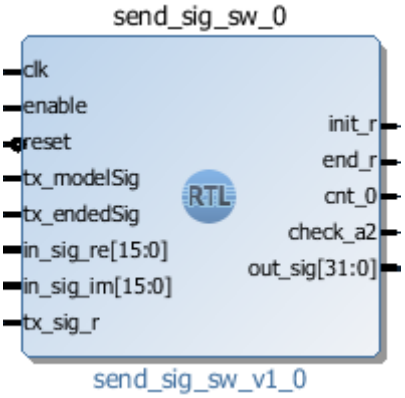

Figura 3-42 - Bloco IP com função de *buffer* de sinais enviados.

O bloco IP com função de *buffer* de sinais e mensagens utilizado com receptor QPSK, apresentado na Figura 3-43, tem como função guardar os sinais e mensagens recebidas para que sejam posteriormente enviadas para a partição software. O funcionamento deste bloco IP é dependente do controlador referente ao receptor IOPT. As mensagens são guardadas pela validação do sinal "rx_msg_store". Os sinais recebidos são guardados após a validação do sinal "rx_sig_store". Na sequência das validações identificou-se o número de sinais utilizados para caracterizar uma mensagem, utilizando o sinal "loaded_frame" definido no bloco IP "data_framed" para que se pudesse definir qual o tamanho necessário dos vetores para guardar os sinais recebidos. Ao terminar a gravação dos sinais é transmitida esta informação ao controlador IOPT, para que posteriormente seja transmitido ao controlador implementado na partição software. À semelhança do *buffer* apresentado no parágrafo anterior, estes sinais e mensagens são transferidos para a partição software passando pelo bloco "hw_signalsto_sw", quando o controlador IOPT implementado em software faz o pedido de leitura. Os endereços utilizados para leitura de mensagens e sinais são o "40600030" e "40600010" respetivamente.

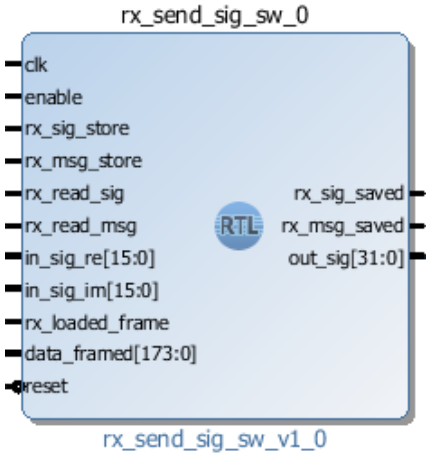

Figura 3-43 - Bloco IP com função de *buffer* de sinais e mensagens recebidas.

3.5 Interface gráfica para utilização do protótipo (Aplicação web)

3.5.1 Estrutura do modelo da aplicação web (modelo base para aplicação na red pitaya)

A estrutura da aplicação web é baseada no modelo de aplicação indicado no guia de desenvolvimento de aplicações web desenvolvido pelo (StemLabs, 2016a), para que a aplicação possa ser parti-

lhada no “bazar” de aplicações da Red Pitaya e utilizada ou modificada por outros investigadores. Na Figura 3-44 é apresentado o modelo de funcionamento de uma aplicação web sugerido em (StemLabs, 2016a).

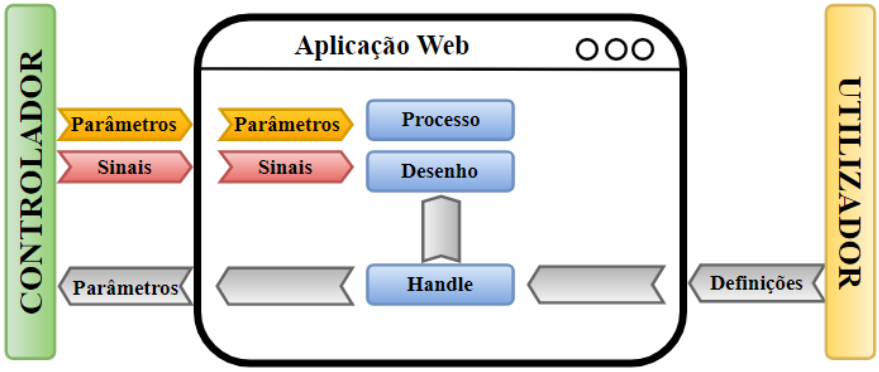

Figura 3-44 - Modelo de funcionamento de uma aplicação web para utilização na plataforma Red Pitaya, imagem adaptada de (StemLabs, 2016a)**.**

Para o desenvolvimento da aplicação alteraram-se os documentos HTML, CS e JS, disponíveis na página github referente a aplicação de alterar o estado de um led. No documento HTML é estruturado o *layout* da página web, sendo estruturadas as divisões e subdivisões de cada parte da interface gráfica. No documento CSS é definido o estilo e localizaçao de cada divisão definida no documento HTML. No documento JS está implementada a lógica referente aos elementos constituintes da interface gráfica, além da troca de informação formatada em JSON com o controlador por conexão WebSocket.

3.5.2 Aplicação web

A interface gráfica para o utilizador desenvolveu-se como aplicação web e considera-se o frontend do projeto. Para o desenvolvimento da aplicação web, além dos documentos CSS, HTML e JS também é necessário modificar o controlador associado à aplicação para a transmissão de sinais e valores para a aplicação web. No início da aplicação é estabelecida a comunicação WebSocket entre a aplicação e o controlador. Para a comunicação de informação entre o controlador e a aplicação utilizaram-se as classe básicas de objectos para sinais e parâmetros disponível na biblioteca rp_sdk presente no ecossistema da plataforma Red Pitaya.

Todos os sinais envolvidos nos controladores IOPT podem ser alterados ou visualizados na aplicação web. As mensagens recebidas ou que se pretendem enviar podem-se visualizar ou editar res-

petivamente. Quando a conexão entre a aplicação e o controlador é estabelecida, inicia-se o ambiente em que se dispõe das opções de funcionamento da aplicação seja a escolha de emissor, recetor ou transmissor. Consoante a escolha do utilizador é inicializado o controlador IOPT correspondente e é carregada a configuração FPGA pretendida.

Na Figura 3-45 apresentam-se os conjuntos de sinais referentes ao controlador do emissor com a cor verde e referentes ao controlador do recetor com a cor azul, apresentam-se também os campos de texto referentes ao envio e receção de mensagens.

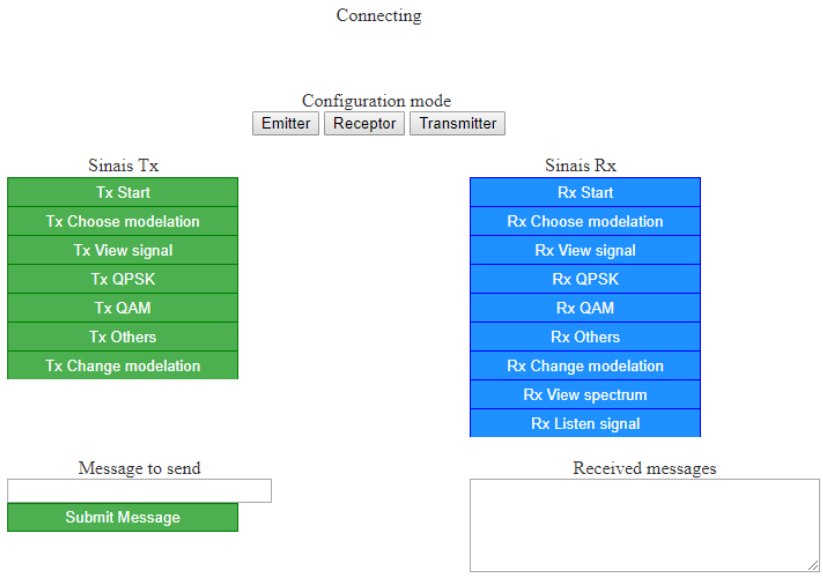

Figura 3-45 - Ambiente da aplicação web referente à escolha dos módulos em utilização, Sinais de entrada para o controlador e os blocos de texto.

Os sinais de saída do controlador IOPT são mostrados na parte inferior da página web como apresentado na Figura 3-46. Para actualizar o valor dos sinais de saída do controlador que estão activos tem de se utilizar o botão “Refresh Signals”. Assim, os sinais que não estão activos, não aparecem na lista. A imagem apresentada identifica todos os sinais que podem estar activos referentes ao controlador do emissor na lista do lado esquerdo e os sinais referentes ao controlador do recetor na lista do lado direito.

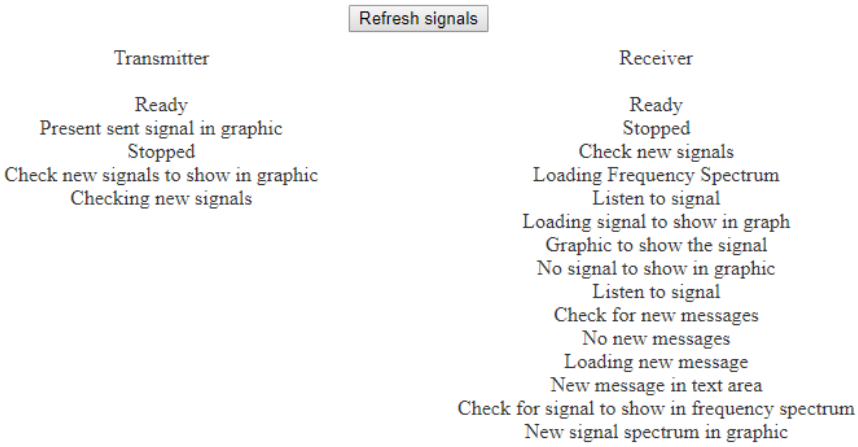

Figura 3-46 - Conjuntos dos sinais de saída referentes ao controlador.

A transmissão de mensagens e sinais entre o controlador da aplicação web e o controlador IOPT é realizada pela escrita e leitura de ficheiros. Assim, estão definidos dois ficheiros para troca de mensagens e dois para troca de sinais. Nos ficheiros em que a aplicação web interage com a função de escrever os valores dos sinais ou mensagens, o controlador IOPT interage com o mesmo ficheiro com a função de leitura e vice-versa.

4

4 Discussão

4.1 Resultados

Para validar o comportamento do emissor e receptor utilizados no gerador de código VHDL da ferramenta simulink compararam-se os sinais reais e imaginários gerados no ambiente de simulação do simulink com os sinais gerados no ambiente de simulação do Vivado. Na Figura 4-1 apresenta-se o diagrama de blocos utilizado no simulink no qual se utilizou o segmento de mensagem "ffffffffffff" como mensagem a transmitir entre o emissor e o recetor. Na Figura 7-12 dos Anexos apresenta-se o diagrama de blocos utilizado no Vivado.

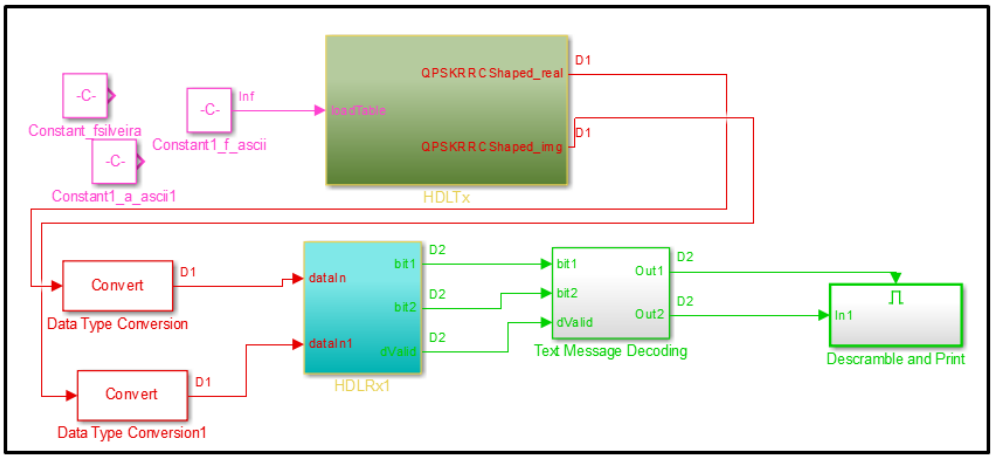

Figura 4-1 - Diagrama de blocos utilizando o emissor e receptor no ambiente do simulink.

Com o propósito de poder-se comparar a forma dos sinais gerados no simulink e no vivado ampliaram-se os sinais referentes a uma mensagem. Apresentam-se de seguida os sinais gerados pela simulação no Simulink sendo o sinal da Figura 4-2 referente à parte real e o sinal da Figura 4-3 referente à parte imaginária.

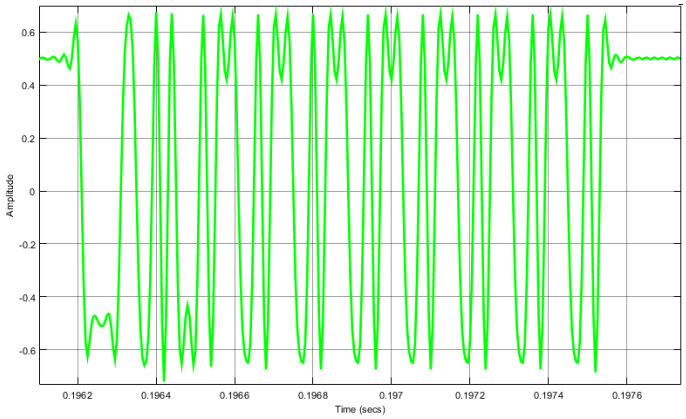

Figura 4-2 - Sinal referente à parte real do sinal gerado pelo emissor.

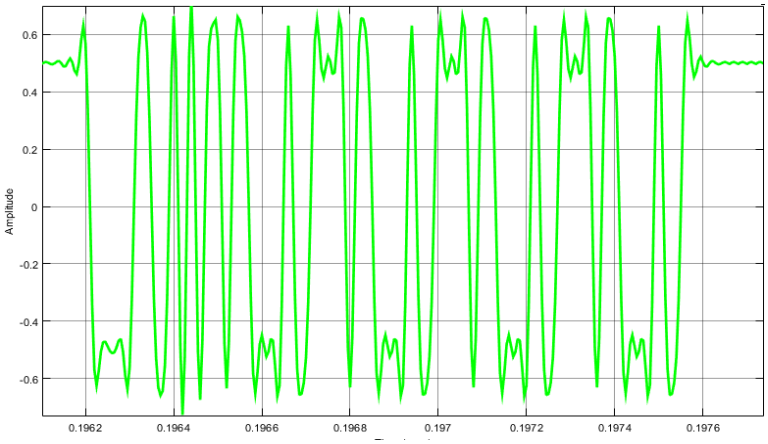

Figura 4-3 - Sinal referente à parte imaginária do sinal gerado pelo emissor.

Na Figura 4-4 apresenta-se o resultado da simulação utilizando o ambiente de simulação do Vivado, a amostra dos sinais apresentada é referente a uma mensagem modulada à semelhança da simulação no ambiente do Simulink. O primeiro sinal da figura é referente à parte real e o segundo à parte imaginária.

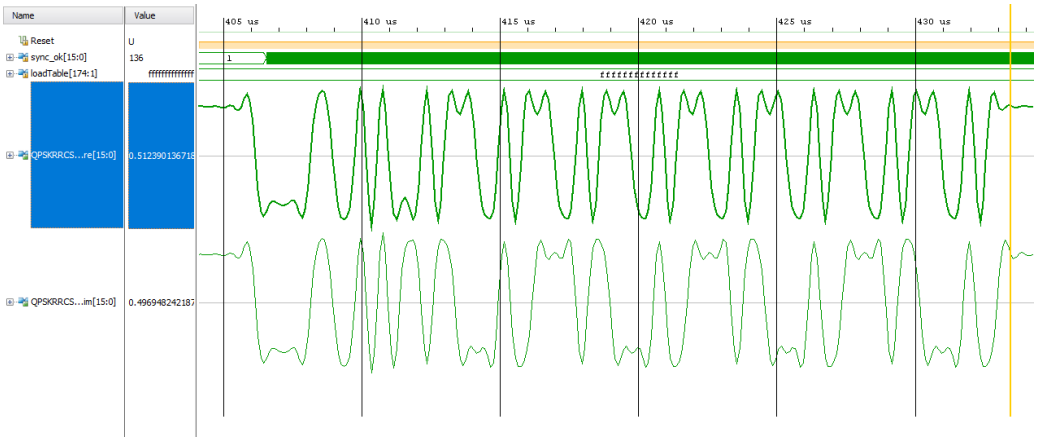

Figura 4-4 – Sinais gerados pelo emissor desenhado no Simulink e utilizado pelo ambiente de simulação do Vivado.

Como se podem observar os sinais gerados têm a mesma forma e utilizando o mesmo conjunto de valores, pertencendo ao intervalo de valores [-0.5 0.5] como esperado.

Com a implementação final do projeto afectaram-se os sinais necessários para o envio da mesma mensagem de teste pela aplicação web, consistindo no segmento "fffffffffffff". Como se pode observar pelos gráficos apresentados na Figura 4-5 e Figura 4-6, os sinais têm a mesma forma e aproximadamente os mesmos valores que os sinais apresentados referentes às simulações realizadas no Simulink e no Matlab. Adicionalmente também se verificou que o segmento de mensagem enviado é igual ao segmento de mensagem recebido.

Figura 4-5 – Sinal referente à parte real apresentado na aplicação web.

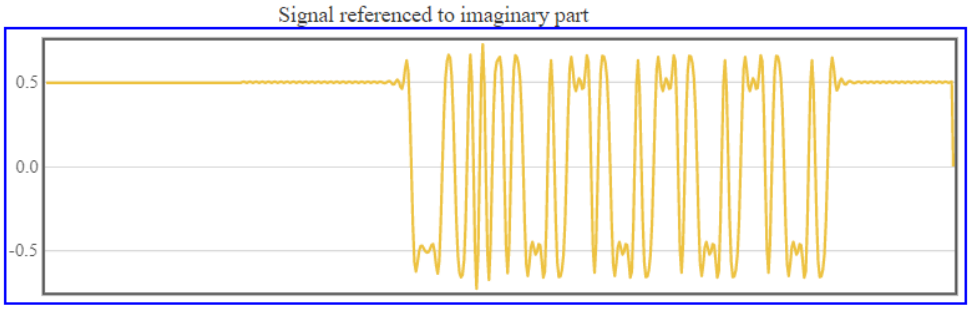

Figura 4-6 - Sinal referente à parte imaginária apresentado na aplicação web.

Considerando a utilização da aplicação desenvolvida em duas plataformas e conectando os pins digitais entre estas, a aplicação funciona como um transcetor de mensagens via web utilizando a modulação QPSK.

4.2 Análise da utilização de reconfiguração completa e reconfiguração parcial no projeto

O dispositivo utilizado permite a reconfiguração parcial dinâmica da FPGA enquanto a parte estática do sistema está em funcionamento. A utilização de uma área da FPGA que permite a reconfiguração parcial dinâmica viabiliza a implementação de diferentes sub-sistemas consoante as necessidades do projeto, como sugerido na Figura 4-7. No entanto apenas uma das configurações pode ser implementada de cada vez.

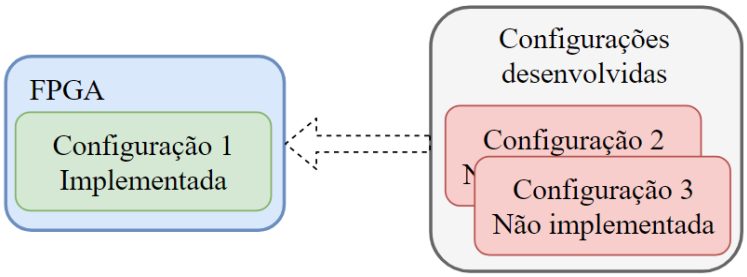

Figura 4-7 - Premissa base da reconfiguração dinâmica.

Assim, pode-se garantir maior flexibilidade do dispositivo, pois mantendo o tipo de portas lógicas entre a área estática e a área reconfigurável da FPGA podem-se utilizar configurações diferentes. A implementação de um sistema com este tipo de reconfiguração pode fazer diferença em caracterísi-

cas como o consumo de energia do dispositivo, a possibilidade de utilizar dispositivos de menor dimensão e a facilidade de introduzir novas funcionalidades ao sistema sem ser necessário gerar e implementar o *bitstream* referente a todo o projeto.

No projeto desenvolvido há a possibilidade de optar pelas funcionalidades do sistema entre emissor, recetor ou ambas. No entanto não se utilizou a reconfiguração parcial dinâmica, pelo que se implementaram os módulos em hardware e com a utilização da aplicação web decidem-se quais os módulos que estão em funcionamento, como sugere a Figura 4-8. Para adicionar novos modos de funcionamento, nomeadamente a utilização de outros tipos de modulação e desmodulação de sinais digitais, sugere-se adicionar os novos módulos ao projeto no Vivado e utilizar os sinais do controlador para decidir quais os módulos em funcionamento.

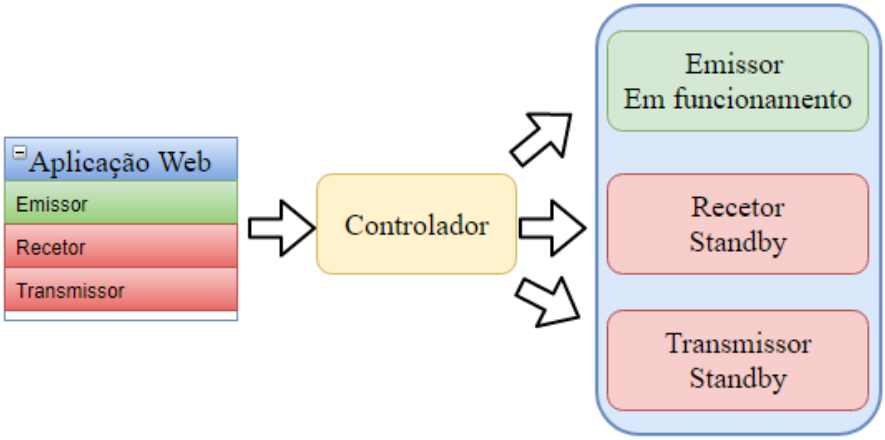

Figura 4-8 - Configuração de implementação do sistema.

Utilizando a reconfiguração parcial dinâmica num sistema garante maior flexibilidade do dispositivo, como exemplo pode-se utilizar uma aplicação web para remotamente reconfigurar a FPGA, por configurações existentes no dispositivo ou pela possibilidade de carregar novos *bitstreams*. No entanto os projetos desenvolvidos utilizando esta habilidade reduzem a flexibilidade de escolha dos dispositivos que podem ser utilizador por ser uma habilidade específica aos dispositivos desenvolvidos pela empresa Xilinx, Inc.

Quanto à utilização de recursos, entende-se que pode ser reduzida com a utilização da reconfiguração dinâmica, assim como os custos de energia, relativamente à abordagem de ter todas as configurações possíveis implementadas e escolher-se qual está em funcionamento. Por outro lado é necessário considerar que há a necessidade de utilizar recursos adicionais ao sistema, sendo estes um processador Microblaze, *bus* AXI, controlador ICAP e *buffer*s, o que pode adicionar complexidade à arquitectura de um sistema simples.

A alteração da configuração na partição de reconfiguração dinâmica implica custos de tempo, o que na abordagem escolhida o custo de tempo é um ciclo de clock. No caso de carregar-se uma nova imagem total para a FPGA o custo de tempo é teoricamente superior ao de reconfigurar uma pequena parte. Um caso em que esta diferença de custo de tempo pode ter menos importância é no caso de uma parte do sistema implementado na partição de reconfiguração dinâmica tornar-se essencial ao sistema e aí seria necessário implementar na partição estática o que implicaria carregar a imagem total para o dispositivo.

Assim surgem aspectos de estudo interessantes, como a necessidade de implementação de mecanismos de segurança, para que não sejam transferidas para o dispositivo configurações de extracção de dados ou que possam causar o mau funcionamento do sistema. Um estudo neste sentido apresenta-se no documento (Johnson, Patranabis, Chakraborty, & Mukhopadhyay, 2017). Outros aspetos que surgem neste sentido são a comparação dos custos de tempo entre cada tipo de reconfiguração. Uma inovação quanto à parte do sistema implementado na partição de reconfiguração dinâmica tornar-se essencial ao sistema poderia passar pelo cálculo da área e recursos utilizados pela configuração que está implementada e a possibilidade de dividir esta área e adicionar novas configurações. Quanto à necessidade de adicionar uma série de recursos mencionados pela Xilinx, Inc. têm sido desenvolvidas novas formas como o processador HoMade apresentado em (Perier et al., 2015).

5

5 Conclusão

5.1 Conclusões

Ao longo da dissertação desenvolveu-se um sistema híbrido considerando a aplicação web para interacção do utilizador com o sistema, o controlador com recurso à utilização das IOPT-Tools e o transmissor optimizado utilizando a modulação de sinais QPSK.

Verificou-se a relevância da utilização das IOPT-Tools para o desenho do sistema que se pretende implementar em plataformas System-on-Chip. Desde a fase inicial do projeto para modelar o sistema e na fase de implementação. Recorrendo ao ambiente de simulação da ferramenta validou-se o funcionamento do controlador e utilizando os geradores de código C e VHDL verificou-se a facilidade de implementação de parte do controlador em software e outra parte em hardware. Este tipo de procedimentos descomplica o desenvolvimento de um sistema híbrido, o qual se considera essencial para aproveitar as caracteristicas vantajosas das plataformas System-on-Chip. Neste sentido escreveu-se um artigo científico, nomeadamente "Utilização das IOPT-Tools no Co-design e implementação em plataformas System-on-Chip" com a co-autoria da Prof. Doutora Anikó Costa. O artigo foi apresentado na conferência "REC 2017 – XIII Jornadas sobre Sistemas Reconfiguráveis" e publicado nas atas referentes(Oliveira & Costa, 2017).

Durante o desenvolvimento do projeto surgiram adversidades, nomeadamente devido à constante evolução e à ausência de prática com o Vivado no início do projeto. Outras adversidades surgiram devido à ausência de prática com o a ferramenta de gerar o código VHDL referente aos modelos do Matlab. Associado a esta adversidade surgiram também problemas devido ao

limite de recursos disponíveis no dispositivo Zynq apesar das características terem sido especificadas na ferramenta de apoio ao gerador de código. Outra adversidade superada foi de conseguir reaproveitar o código das aplicações web disponibilizado pela comunidade associada à Red Pitaya, para desenvolver a nova aplicação web.

Concluindo, identificaram-se as vantagens da utilização das IOPT-Tools para o desenvolvimento de aplicações em que se pretende aproveitar toda a arquitectura das plataformas System-on-Chip. Além da aplicação desenvolvida pode-se considerar o projeto como uma solução base para outras aplicações com caracteristicas semelhantes. Naturalmente surgiram ideias para melhorias do projeto as quais são apresentadas com alusão às vantagens que oferecem ao projeto no subcapítulo 5.2.

5.2 Trabalhos Fututos

Para maior valorização da aplicação reconhecem-se algumas melhorias e novas funcionalidades ou subsistemas que poderiam ser agregados ao projeto.

As melhorias identificadas para a aplicação web são o controlo do acesso de utilizadores para que não estejam dois utilizadores ao mesmo tempo a afectar os sinais enviados para o sistema. Neste sentido poderia-se criar uma sessão de utilização e colocar em espera um segundo utilizador. Outro aspecto para melhorar a flexibilidade da aplicação seria viabilizar o carregamento de novas configurações para a FPGA remotamente. Portanto, além da possibilidade de escolher remotamente a configuração da FPGA em funcionamento, poderiam-se adicionar configurações sendo guardadas pelo sistema operativo. No entanto, este aspecto levanta questões de segurança quanto ao correcto funcionamento do dispositivo, uma vez que poderiam ser carregadas configurações que interrompessem o funcionamento da aplicação ou que modificassem os ficheiros essenciais à aplicação. Portanto poderia-se condicionar o acesso aos recursos com a classificação de um utilizador como administrador. Outra abordagem de reconfiguração da FPGA seria implementarem-se os recursos necessários em hardware para utilização da função de reconfiguração parcial dinâmica cujas implicações se apresentaram no subcapítulo 4.2.

Um aspecto a considerar no caso de ser possivel transferir novas configurações para a plataforma remotamente, seria alterar o funcionamento da Red Pitaya como *Access Point*. Neste sentido, seria desnecessária a conexão por cabo Ethernet ligado a um computador, em vez disso, pode ligar-se a plataforma diretamente a um router. Este conceito remete a aplicação para uma abordagem semelhante à de um laboratório remoto.

A melhoria identificada na implementação hardware que poderia valorizar a aplicação seria a implementação dos DAC e ADC agregados ao emissor e recetor respetivamente, e a utilização de antenas para tornar possivel a transmissão de mensagens por comunicação sem fios. Além disso, seria também importante a implementação de um subsistema para introduzir uma onda portadora ao sinal modulado no emissor e o respetivo filtro no recetor. Adicionalmente também seria interessante a implementação de um desmodulador FM para permitir ouvir rádios locais. Relativamente à melhoria da capacidade dos *buffer*s de mensagens e sinais, seria interessante a implementação dos recursos necessários para utilizar o acesso directo à memória. A utilização desta funcionalidade permitiria manter o tamanho dos *buffer*s e sendo uma funcionalidade independente do CPU, não compromete a capacidade de processamento do dispositivo. Adicionalmente poderia-se agregar ao projeto diferentes processos de modulação e standards de comunicação sem-fios.

6

6 Referências

1994-2017 The MathWorks Inc. (2017a). Matlab. Retrieved February 20, 2017, from https://www.mathworks.com/products/matlab.html

1994-2017 The MathWorks Inc. (2017b). Simulink. Retrieved February 23, 2017, from https://www.mathworks.com/products/simulink.html

2017 National Instruments Corporation. (2017). Labview. Retrieved February 20, 2017, from http://www.ni.com/labview/pt/

Avnet, I. (2017). ZedBoard.org. Retrieved February 20, 2017, from http://zedboard.org/

Brauer, W., & Reisig, W. (2009). Carl Adam Petri and "Petri Nets." *Fundamental Concepts in Computer Science*, *29*(5), 129. http://doi.org/10.1007/s00287-006-0107-7

Brigham, E. O. (1974). *THE FAST FOURIER TRANSFORM*.

Chao, L. (2001). The Pentium ® 4 Processor – Advanced Technology for the Internet and Beyond.

Crockett, L., Elliot, R., Enderwitz, M., & Stewart, B. (2014). The Zynq Book. Retrieved from http://www.zynqbook.com/

Ferreira, M. L., & Ferreira, J. C. (2017). Evaluation of Resource Utilization of a Dynamically Reconfigurable OFDM Baseband Transmitter. Aveiro.

Gomes, L., Barros, J. P., Costa, A., & Nunes, R. (2007). The input-output place-transition petri net class and associated tools. *IEEE International Conference on Industrial Informatics (INDIN), 1*, 509–514. http://doi.org/10.1109/INDIN.2007.4384809

Grayver, E. (2013). *Implementing Software Defined Radio*. Springer Science+Business Media New York 2013.

Grout, I. A. (2001). Modeling , simulation and synthesis : From Simulink to VHDL generated hardware. *Proceedings of the 5th World Multi-Conference on Systemics, Cybernetics and Informatics*, *15*, 443–448.

Intel Corporation. (2016). Parallel Flash Loader IP Core User Guide.

Johnson, A. P., Patranabis, S., Chakraborty, R., & Mukhopadhyay, D. (2017). Remote dynamic partial reconfiguration : A threat to Internet-of-Things and embedded security applications.

Microprocessors and Microsystems, *52*, 131–144. http://doi.org/10.1016/j.micpro.2017.06.005

Junior, S. B., Oliveira, V. C. De, & Junior, G. B. (2015). Software Defined Radio Implementation of a QPSK Modulator / Demodulator in an Extensive Hardware Platform Based on FPGAs Xilinx ZYNQ, *4*. http://doi.org/10.3844/jcssp.2015.598.611

Kumar, A. (2016). module for Cognitive Radios, (July), 87–92.

Kuusilinna, K., Chang, C., Bluethgen, H. M., Davis, W. R., Richards, B., Nikolic, B., & Brodersen, R. W. (2003). *Winning the SoC Revolution, Experiences in Real Design*.

Leban, P. (2014). Red Pitaya User Manual.

Lima, R. (2015). *Wave4IOPT - Editor e Visualizador Web de Formas de Onda*. Universidade Nova de Lisboa.

Lipsett, R., Schaefer, C. F., & Ussery, C. (1989). *VHDL : hardware description and design*.

Luise, M., & Reggiannini, R. (1995). Carrier Frequency Recovery in All-Digital Modems for Burst-Mode Transmissions. *IEEE Transactions on Communications, Vol. 43 No. 2/3/4*, *43*(2), 1169–1178.

Moutinho, F. (2014). IOPT Tools User Manual, *2014*(C), 1–50.

Murata, T. (1989). Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, *77*(4), 541–580. http://doi.org/10.1109/5.24143

Oliveira, A. S. R., & Costa, A. (2017). *REC 2017 - Atas*. Aveiro: REC 2017 - Jornadas sobre Sistemas Reconfiguráveis.

OpenBTS. (2017). OpenBTS. Retrieved February 20, 2017, from http://openbts.org/

Pereira, F., & Gomes, L. (2013). Automatic synthesis of VHDL Hardware Components from IOPT Petri Net models. *IEEE Industrial Electronic Society*, 2214–2219.

Pereira, F., Moutinho, F., & Gomes, L. (2012). IOPT-Tools. Retrieved November 29, 2016, from http://gres.uninova.pt

Perier, J., Chouchene, W., Dekeyser, J., Perier, J., Chouchene, W., & Circuit, J. D. (2015). Circuit Merging versus Dynamic Partial Reconfiguration -The HoMade Implementation. *I-Manager's Journal on Embedded Systems*, *4*(1), 14–23.

Peterson, J. L. (1977). Petri Nets*, *9*(3).

Prata, A., Cordeiro, R. F., Dinis, D. C., Oliveira, A. S. R., Vieira, J., & Carvalho, N. B. (2017). Recent Advances and Trends on FPGA-based All-Digital Transceivers. Aveiro.

Qian, K. (2015). Variability Modeling and Statistical Parameter Extraction for CMOS Devices, (UCB/EECS-2015-165). Retrieved from http://www.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-165.html

Rajesvari, R., Manoj, G., Angelin Ponrani, M., & Annie Joy, M. (2013). Core Based Architecture ofTelecommand System-on-Chip (SoC) for Spacecraft applications, 1–6.

Rebelo, R., Pereira, F., Moutinho, F., & Gomes, L. (2011). From IOPT Petri nets to C : an Automatic Code Generator Tool. *Industrial Informatics (INDIN), 2011 9th IEEE International Conference On, Caparica, Lisbon, Portugal*, 390–395.

REDHAWK. (2016). REDHAWK. Retrieved February 20, 2017, from https://redhawksdr.github.io/Documentation/

Ritchie, D., & Kernighan, B. (2002). *The C programming Language* (Vol. 8).

Rondeau, T. W. (2015). On the GNU Radio Ecosystem. In O. Holland, H. Bogucka, & A. Medeisis

(Eds.), *Opportunistic Spectrum Sharing and White Space Access: the practical reality* (pp. 25–47). John Wiley and Sons, Inc.

Software, F. (2004). Software defined radios Software defined radios – overview and hardware (1), *182*(182), 58–61.

StemLabs. (2016a). Developers guide. Retrieved September 18, 2017, from http://redpitaya.readthedocs.io/en/latest/developerGuide/software/webApps.html

StemLabs. (2016b). Red Pitaya. Retrieved January 3, 2016, from redpitaya.com

StemLabs. (2017). Red Pitaya. Retrieved February 27, 2017, from https://redpitaya.readthedocs.io/en/latest/doc/developerGuide/125-14/extent.html#extension-connector

Sutton, P. (2015). *Opportunistic Spectrum Sharing and White Space Access - The Practical Reality A Dynamically Reconfigurable Software Radio Framework: Iris*. (O. Holland, H. Bogucka, & A. Medeisis, Eds.). New Jersey and simulatniously published in Canada: John Wiley & Sons, Inc,.

The MathWorks, I. (2015a). hdl optimized qpsk receiver with captured data. Retrieved March 12, 2017, from https://www.mathworks.com/help/comm/examples/hdl-optimized-qpsk-receiver-with-captured-data.html

The MathWorks, I. (2015b). HDL Optimized QPSK Transmitter. Retrieved March 12, 2017, from https://www.mathworks.com/help/comm/examples/hdl-optimized-qpsk-transmitter.html

Wyglinski, A. M., Orofino, D. P., Ettus, M. N., & Rondeau, T. W. (2016). Revolutionizing Software Defined Radio : Case Studies in Hardware , Software , and Education, (January), 68–75.

Xilinx, I. (2010). Platform Flash XL / Platform Flash Memory for Virtex-5 FPGAs Bitstream. Retrieved February 20, 2017, from https://www.xilinx.com/publications/...mem/PROMs_Virtex.pdf

Xilinx, I. (2016a). System Generator for DSP. Retrieved February 20, 2017, from https://www.xilinx.com/products/design-tools/vivado/integration/sysgen.html

Xilinx, I. (2016b). Zynq-7000 Product Specification. Retrieved February 20, 2017, from https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf

Xilinx, I. (2017). Vivado Design Tool. Retrieved February 20, 2017, from http://www.xilinx.com/products/design-tools/vivado.html

7 Anexos

7.1 Anexos A

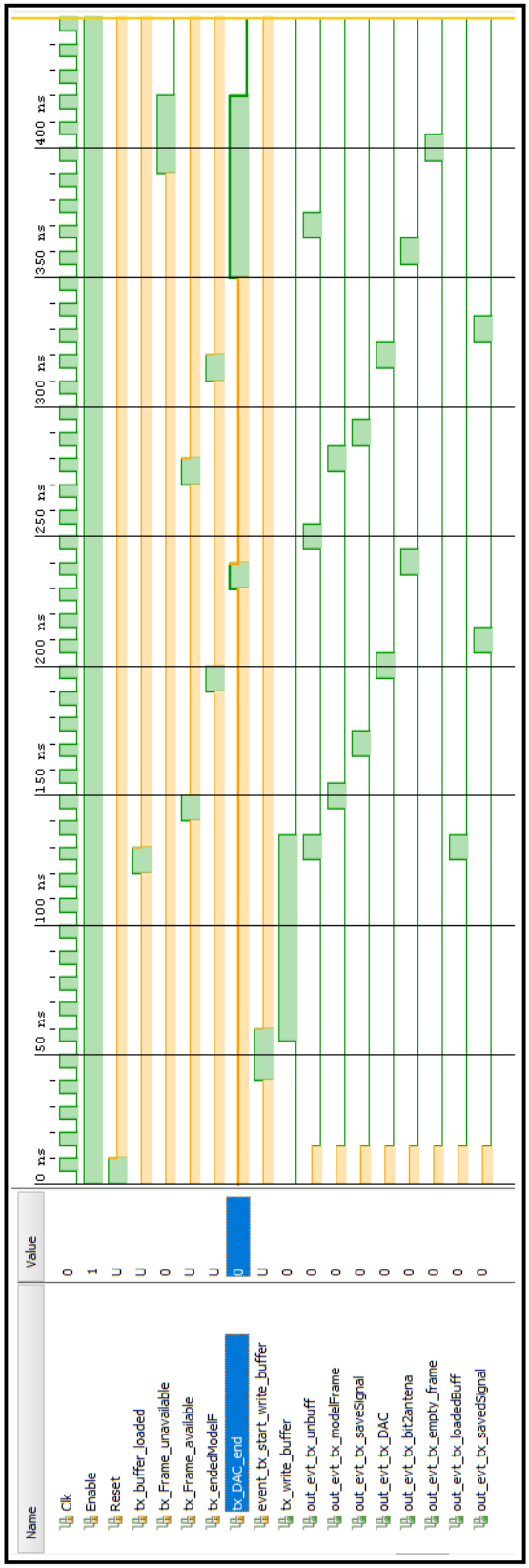

Figura 7-1 - Simulação do controlador referente ao emissor.

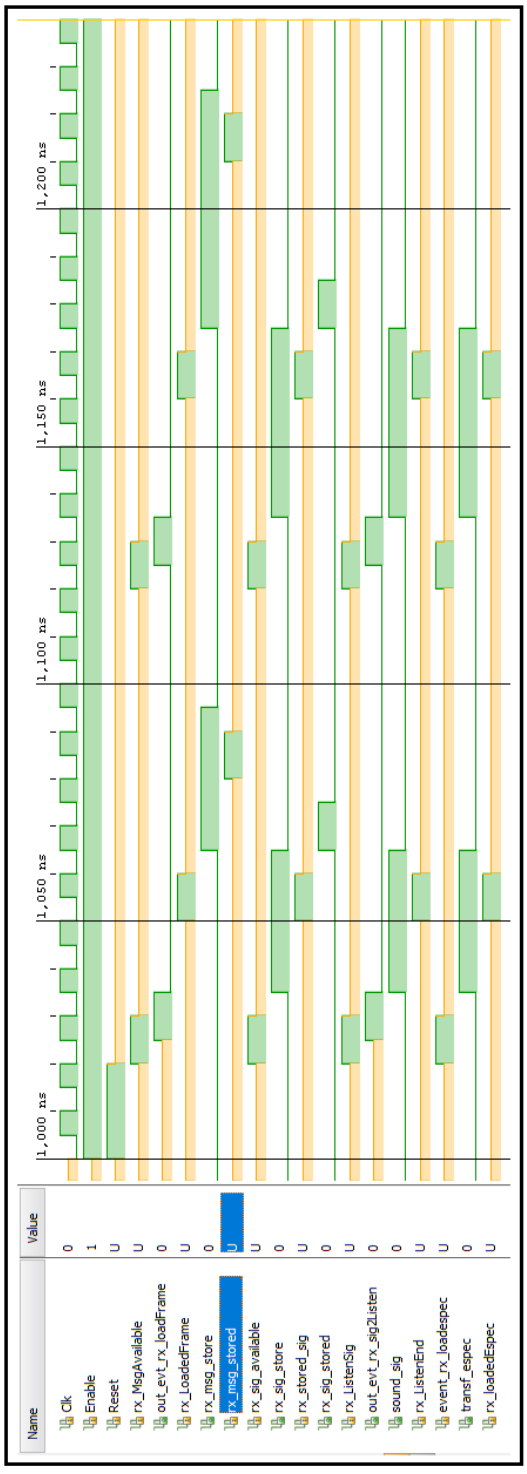

Figura 7-2 - Simulação do controlador junto ao emissor.

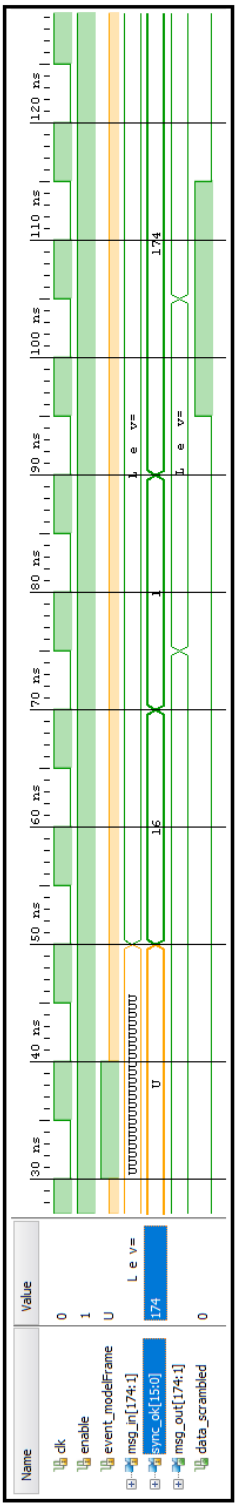

Figura 7-3 - Simulação do bloco Sync_msg.

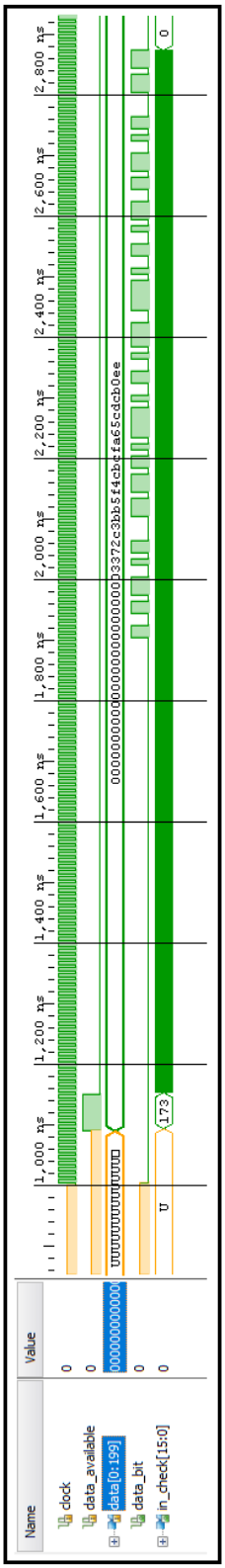

Figura 7-4 - Simulação do bloco Check_vector.

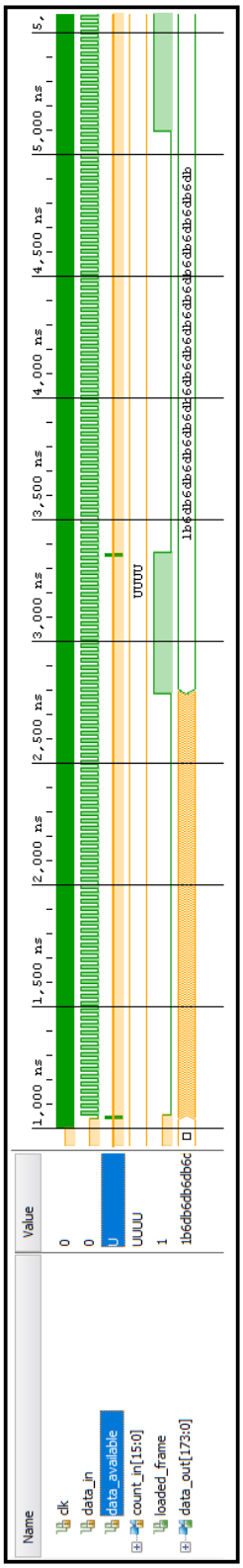

Figura 7-5 - Simulação do bloco data_framed.

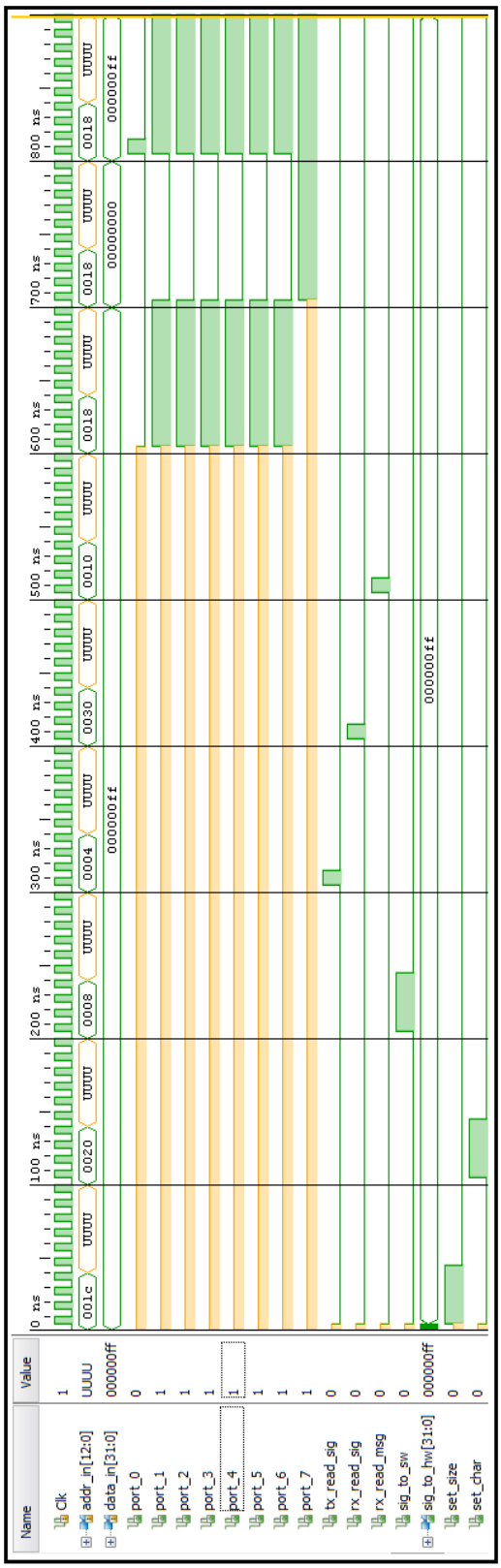

Figura 7-6 - Simulação do bloco Sw_signals_hw.

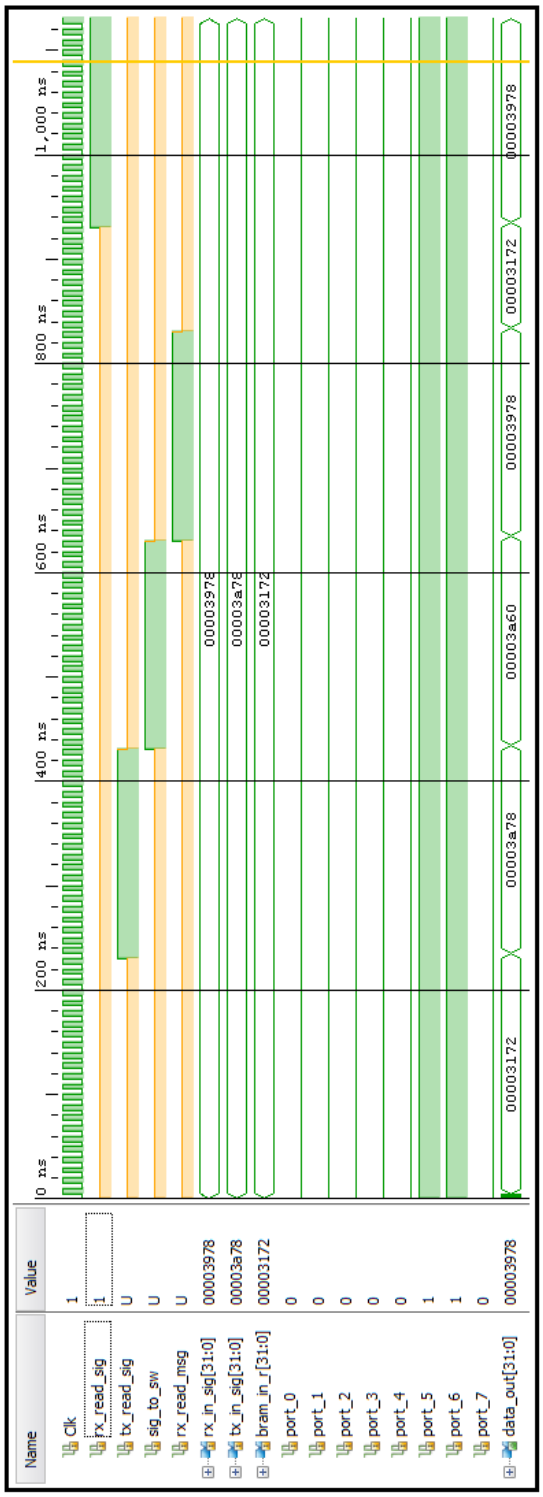

Figura 7-7 - Simulação do bloco hw_signals_sw.

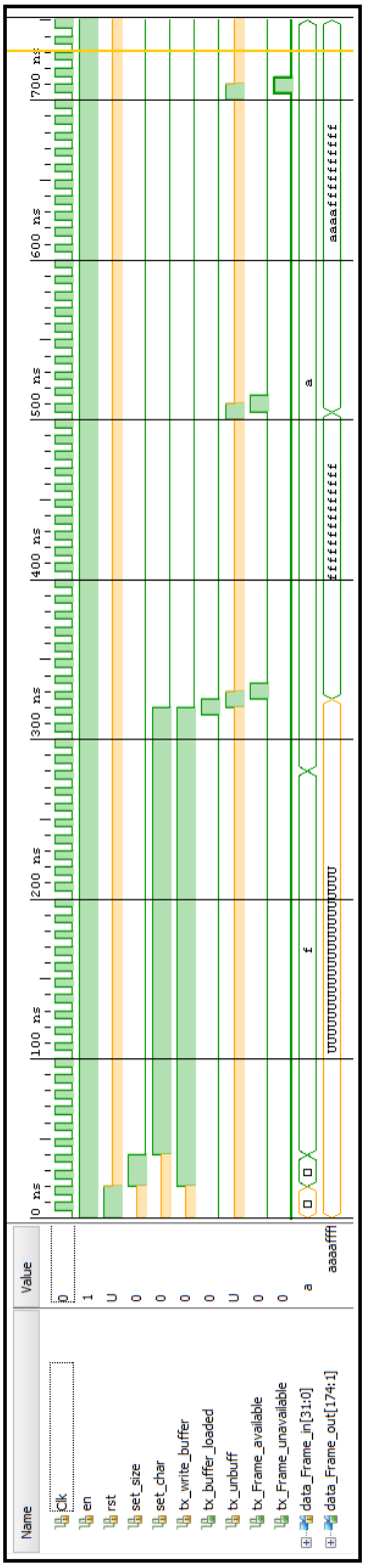

Figura 7-8 - Simulação do bloco Msg_buffer.

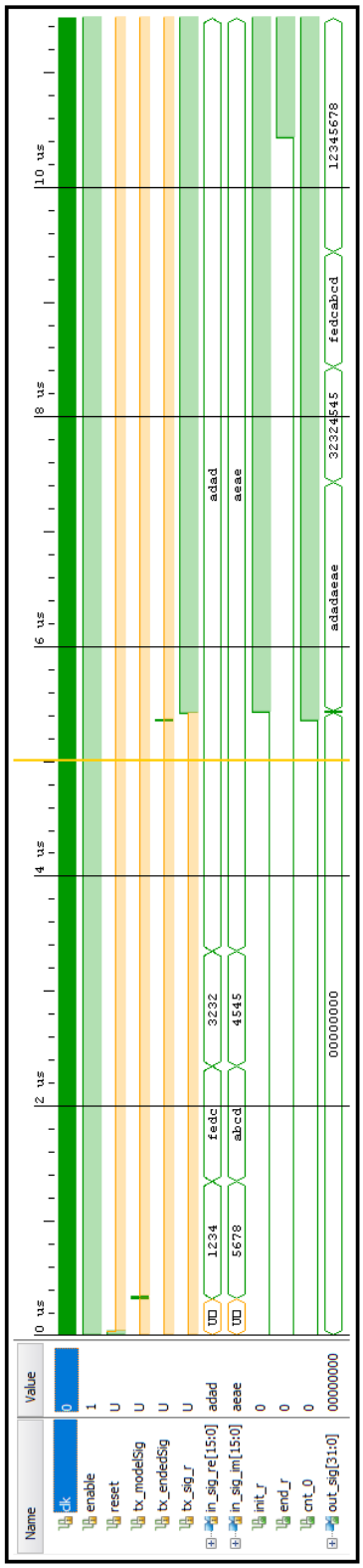

Figura 7-9 - Simulação do bloco send_sigsw.

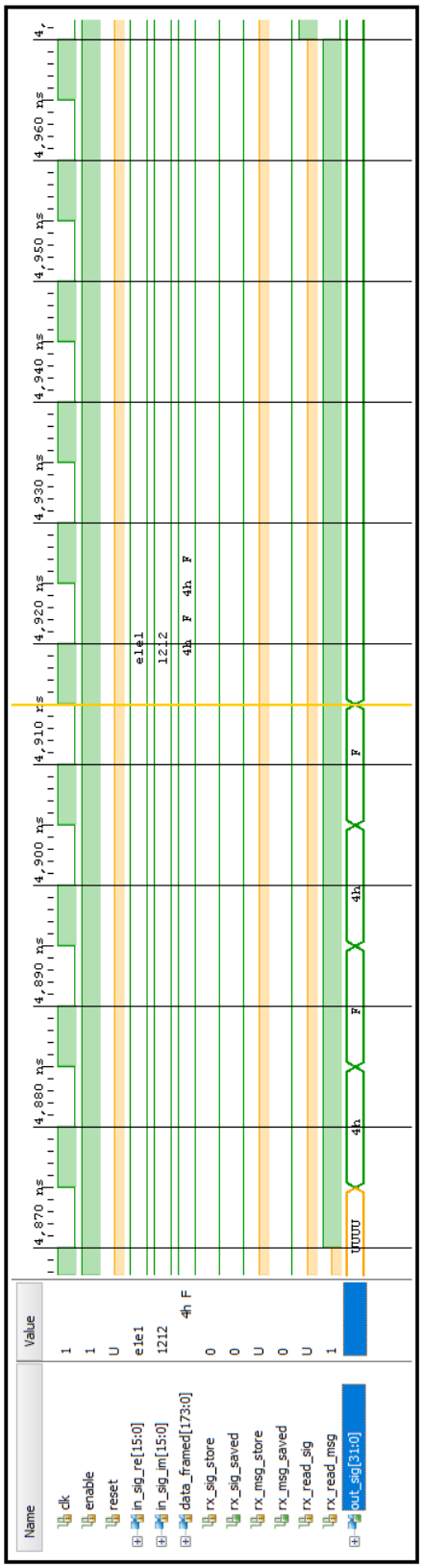

Figura 7-10 - Simulação do bloco Rx_send_sig_sw (função de ler a mensagem guardada).

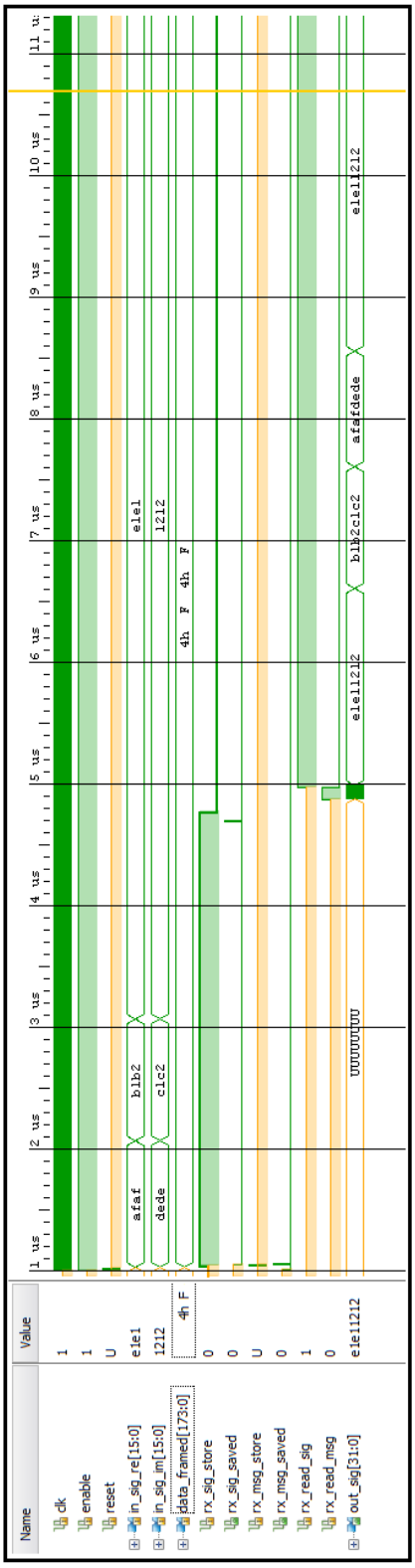

Figura 7-11 - Simulação do bloco Rx_send_Sig_sw (função de ler os sinais guardados).

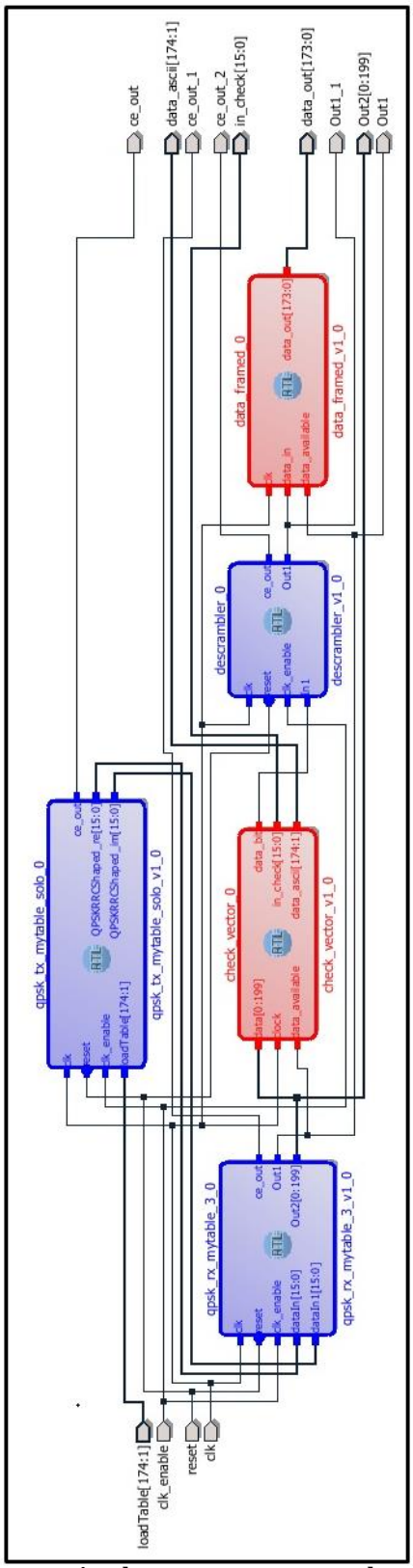

Figura 7-12 - Modelo utilizado para simular o comportamento do emissor e recetor gerados no Simulink.

7.2 Anexos B

Tabela 7-1 - Descrição dos sinais de entrada utilizados na rede de Petri referente ao controlador do emissor.

Sinais de entrada	Fonte	Descrição do sinal
VeSinal	WebApp	O utilizador pede para visualizar o sinal na aplicação web.
EscModelacao	WebApp	O utilizador pede para escolher o tipo de modelação em funcionamento
Tx_changemode	WebApp	O utilizador pede para interromper o funcionamento do emissor
Tx_sig_unavailable		O controlador não identificou um novo sinal guardado para visualizar.
Tx_sig_available		O controlador identificou um novo sinal guardado para visualizar.
M_qpsk	WebApp	Escolha de modelação QPSK foi identificada.
M_qam	WebApp	Escolha de modelação QAM foi identificada.
M_others	WebApp	Escolha de outra modelação que possa ser adicionada ao projeto foi identificada.
impQpskEnd		O controlador hardware sinaliza que o emissor QPSK está pronto para funcionar.
impQamEnd		O controlador hardware sinaliza que o emissor QAM está pronto para funcionar.
impOtherEnd		O controlador hardware sinaliza que o emissor posteriormente adicionado ao projeto está pronto para funcionar.
hwtx_stored_sig	C_hw	Indica a existência de sinais guardados em hardware que possam ser transferidos para o software.
hwtx_end_sig	C_hw	Indica que a transferência do sinal foi concluida.
Envia	WebApp	O utilizador pede para enviar uma mensagem.
Tx_msgAvailable		Sinaliza que existe uma mensagem para enviar que é composta por pelo menos um caracter.
Tx_buffer_loaded	C_hw	Indica que os caracteres da mensagem foram todos transferidos para a partição de hardware.
Tx_frame_available		Indica que existe um segmento de mensagem para ser modulado.
Tx_endedModelF		A modulação do segmento está completa.

Tx_DAC_end		Conversão do segmento modulado D/A está completa.
Tx_frame_unavailable		Indica que não existe um segmento de mensagem para ser modulado.

Tabela 7-2 - Descrição dos sinais de saída utilizados na rede de Petri referente ao controlador do emissor.

Sinais de saída	Destino	Descrição do sinal
Tx_check_new_sig	WebApp	O controlador confirma se há novos sinais para serem visualizados.
Tx_loadGrafico	WebApp	O controlador transfere os dados do sinal para a aplicação web.
Tx_ready	WebApp	O controlador foi executado e está à espera que o utilizador de o sinal para início de funcionamento.
Hwtx_read_sig	C_hw	Transfere-se o sinal guardado na partição hardware é transferido para a partição software.
Tx_write_buffer		É transferido o segmento de mensagem da partição software para a partição software.

Tabela 7-3 - Descrição dos eventos de entrada utilizados na rede de Petri referente ao controlador do emissor.

Eventos de entrada	Fonte	Descrição do evento
Tx_start	WebStart	O utilizador inicia o funcionamento do emissor.

Tabela 7-4 - Descrição dos eventos de saída utilizados na rede de Petri referente ao controlador do emissor.

Eventos de saída	Destino	Descrição do evento
Tx_start_write_buffer	C_hw	Indicação ao controlador IOPT na partição hardware que se pretende transferir um segmento de mensagem.
Tx_checksignal		O controlador confirma se há novos sinais para serem visualizados.
Tx_hwdspEnd		Indicar ao controlador implementado em hardware para interromper o funcionamento do emissor.
Tx_imp_qpsk		Indicação ao controlador para que o módulo emissor QPSK esteja em funcionamento.

Tx_imp_qam		Indicação ao controlador para que o módulo emissor QAM esteja em funcionamento.
Tx_imp_other		Indicação ao controlador para que o módulo emissor posteriormente adicionado esteja em funcionamento.
Tx_hwdspStart		Indicação para o controlador implementado em hardware dar início ao funcionamento do emissor escolhido.
tx_stopped	WebApp	Todas as funcionalidades possíveis do emissor estão interrompidas e não podem voltar a funcionar ate o utilizador sinalizar o start.
Tx_checkMsg		O controlador confirma se há novas mensagens para serem enviadas.
Tx_ascii2hex		Segmento de mensagem convertido em símbolos hexadecimal. Este evento serve como notificação.
Tx_unbuff		Indicação para retirar um segmento de mensagem do *buffer*.
Tx_loadedBuff		Indicação que a mensagem enviada para a partição hadware está gravada no *buffer*.
Tx_modelFrame		Indicação para iniciar a modelação de um segmento de mensagem.
Tx_saveSignal		Indicação para inciar a gravação do segmento de mensagem modelado.
Tx_savedSignal		Indicação que o sinal referente ao segmento de mensagem foi gravado.
Tx_DAC		Indicação para converter o sinal modelado em analogico.
Tx_bit2antena		Indicação para passar o sinal para as portas analogicas da plataforma.
Tx_empty_frame		Indicação de não haver segmentos de mensagem para modelar.

Tabela 7-5 - Descrição dos sinais de entrada utilizados na rede de Petri referente ao controlador do recetor.

Sinais de entrada	Fonte	Descrição do sinal
Rx_Esc_espectro		O utilizador pede para ver o espectro de frequencia do sinal guardado.
Rx_esc_mod	WebApp	O utilizador pede para escolher a modelação.
rx_changemode	WebApp	O utilizador pede para interromper o funcionamento do modulador.
rx_sig_unavailable		O controlador identificou um novo sinal guardado para visualizar.
rx_sig_available		O controlador não identificou um novo sinal guardado para visualizar.
Rx_loadedEspec		Indicação que o sinal foi transferido para a aplicação web.
Rx_M_qpsk	WebApp	Escolha de desmodelação QPSK foi identificada.
Rx_M_qam	WebApp	Escolha de desmodelação QAM foi identificada.
Rx_M_others	WebApp	Escolha de outra desmodelação que possa ser adicionada ao projeto foi identificada.
Rx_impQpskEnd		O controlador hardware sinaliza que o recetor QPSK está pronto para funcionar.
Rx_impQamEnd		O controlador hardware sinaliza que o recetor QAM está pronto para funcionar.
Rx_impOtherEnd		O controlador hardware sinaliza que o recetor posteriormente adicionado ao projeto está pronto para funcionar.
Rxhw_msg_stored		Indica a existência de mensagens guardadas em hardware que podem ser transferidas para o software.
Rxhw_sig_stored		Indica a existência de sinais guardados em hardware que podem ser transferidos para o software.
Rx_esc_view	WebApp	O utilizador pede para visualizar o sinal na aplicação web.
Rx_sig_available		O controlador identificou um novo sinal guardado para visualizar.
Rx_sig_unavailable		O controlador não identificou um novo sinal guardado para visualizar.
Rx_ListenSig	WebApp	O utilizador pretende ouvir o sinal recebido.
Rx_ListenEnd		Indicação do fim de reprodução do sinal.

Rx_esc_msg	WebApp	O utilizador pede para mostrar novas mensagens recebidas.
Rx_MsgUnavailable		Sinaliza que não existem novas mensagens gravadas.
Rx_MsgAvailable		Sinaliza que existe pelo menos uma mensagem nova gravada.

Tabela 7-6 - Descrição dos sinais de saída utilizados na rede de Petri referente ao controlador do recetor.

Sinais de saída	Destino	Descrição do sinal
rx_check_new_sig	WebApp	O controlador confirma se há novos sinais para serem visualizados.
Rx_app_espec	WebApp	Leitura de sinais representados no dominio de frequencia guardados em hardware
rx_ready	WebApp	O controlador foi executado e está à espera que o utilizador de o sinal para dar início ao funcionamento.
ReadMsg_hw	C_hw	Leitura de mensagem guardada em hardware.
ReadSig_hw	C_hw	Leitura de sinais guardados em hardware
Check_sigs_file		Verificação de novos sinais guardados em software.
Rx_audio_sig	WebApp	Reprodução sonora de sinal
Rx_checkMsg_available		Verificação de novas mensagens guardadas em software.

Tabela 7-7 - Descrição dos eventos de entrada utilizados na rede de Petri referente ao controlador do recetor.

Eventos de entrada	Fonte	Descrição do evento
rx_start	WebApp	O utilizador inicia o funcionamento do recetor.

Tabela 7-8 - Descrição dos eventos de saída utilizados na rede de Petri referente ao controlador do recetor.

Eventos de saída	Destino	Descrição do evento
rx_checksig	WebApp	O controlador verifica se á novos sinais no dominio da frequencia para serem visualizados.
Rx_printEspec	WebApp	Indicação para imprimir o gráfico referente ao espectro de frequencia do sinal.

Rx_hwdspEnd		Indicar ao controlador implementado em hardware para interromper o funcionamento do recetor.
Rx_imp_qpsk		Indicação ao controlador para que o módulo recetor QPSK esteja em funcionamento.
rx_imp_qam		Indicação ao controlador para que o módulo emissor QAM esteja em funcionamento.
rx_imp_other		Indicação ao controlador para que o módulo recetor posteriormente adicionado esteja em funcionamento.
rx_hwdspStart		Indicação para o controlador implementado em hardware dar início ao funcionamento do recetor escolhido.
rx_stopped	WebApp	Todas as funcionalidades possíveis do recetor estão interrompidas e não podem voltar a funcionar ate o utilizador sinalizar o start.
Rx_loadSig	WebApp	Indicação para transferir o sinal guardado em hardware para software.
Rx_printNoSig	WebApp	Indicação ao utilizador que não existem sinais novos guardados em hardware.
Rx_printSig	WebApp	Indicação à aplicação web para imprimir o sinal no gráfico.
Rx_sig2Listen	WebApp	Indicação ao utilizador que se vai dar início à reprodução sonora do sinal.
Rx_checkMsg	WebApp	O controlador verifica se há novas mensagens no para serem apresentadas.
Rx_printNoMsg	WebApp	Indicação que não existem novas mensagens para serem apresentadas.
Rx_loadFrame	WebApp	Indicação para a mensagem ser transmitida para a aplicação web.
Rx_printMsg	WebApp	Indicação para a mensagem ser impressa na lista de mensagens da aplicação web.
Rx_loadFrame		Retira a mensagem do módulo recetor

Tabela 7-9 - Descrição dos ficheiros gerados pela ferramenta Gerador de código C.

Nome do ficheiro	Descrição

Net_types.h	Definição dos tipos de dados para a marcação de cada lugar, sinais de entrada ou saída e eventos. Declaração dos métodos associados ao comportamento da rede. Como exemplo, adição e remoção de marcas a um lugar, gerar sinais de saída associados à marcação de um lugar.
Net_main.c	Define-se o ciclo de execução da rede de Petri.
Net_io.c	Preparação para leitura e escrita de sinais de entrada e saída nos pinos da plataforma hardware. É necessário que o utilizador faça a declaração dos portos da plataforma a utilizar e associação dos mesmos aos sinais pretendidos.
Net_functions.c	São declaradas as funções de implementação das semânticas e regras da rede de Petri IOPT.
Net_exec_step.c	Função que executa um passo de execução da rede usando as funções definidas pelo documento net_functions.c.
Makefile	A utilização deste documento é opcional. Construtor de projeto Gnu Make/Unix com instruções para construção do projeto.

Tabela 7-10 - Descrição dos ficheiros adicionalmente gerados automaticamente na versão mais recente da ferramenta.

Dummy_gpio.c	A utilização deste documento permite que os sinais de entrada sejam gerados a partir de um documento de texto, e o registo das saídas seja realizado para um documento de texto.
http_server.c	O documento é constituído pela implementação de *sockets*, monitorização do servidor e HTTP debug.
http_server.h	O documento serve de *header* para o documento http_server.c, são declaradas as funções de debug e definida a estrutura do argumento do pedido de debug.
Linux_sys_gpio.c	A utilização deste documento permite a utilização do controlador em sistemas Linux. Sendo implementadas as fun-

	ções para definição dos pinos como entrada ou saída e respetivamente a escrita e leitura de sinais nos pinos.
Net_dbginfo.c	O documento serve para realização de debug da rede, permitindo forçar entradas e obter informação sobre os sinais de entrada e saída, marcas e disparos de transições.
Net_server.c	O documento é constituído pelas funções associadas à comunicação da rede de Petri como servidor Ethernet, com a implementação do protocolo HTTP.
Net_server.h	O documento serve de header para o documento net_server.c, são declaradas as funções para implementação do servidor HTTP.
Raspi_mmap_gpio.c	A utilização deste documento permite a utilização do controlador nas plataformas Raspberry um e dois. São implementadas as funções para definição dos pinos como pinos de entrada ou de saída e respetivamente a escrita e leitura de sinais nos pinos.