

Faculdade de Engenharia da Universidade do Porto



**Biblioteca de módulos Verilog para interface de
FPGAs com periféricos I/O**

Ricardo Jorge dos Santos Machado

Dissertação realizada no âmbito do
Mestrado Integrado em Engenharia Electrotécnica e de Computadores
Major Telecomunicações

Orientador: Prof. Dr. Hélio Mendes de Sousa Mendonça

Junho 2010

© Ricardo Jorge dos Santos Machado, 2010

Resumo

As FPGAs têm assumido um papel cada vez mais importante no ramo da microelectrónica. Isto deve-se ao facto destes circuitos integrados poderem desempenhar as mesmas funções lógicas de um ASIC, podendo no entanto ser reconfiguradas, mesmo depois de já terem sido enviadas ao cliente. As vantagens são ainda maiores numa fase de desenvolvimento de um novo sistema, em que para a correcção de um protótipo ASIC é necessária a produção de um novo circuito integrado, o que leva a enormes custos de produção.

No âmbito desta dissertação era pretendida uma biblioteca de módulos em Verilog, de fácil integração e utilização, que permitissem a interligação entre a FPGA e vários dispositivos periféricos de I/O, pois apesar de as plataformas de desenvolvimento, utilizadas ao longo de todo o Mestrado Integrado de Engenharia Electrotécnica e Computadores (MIEEC), possuírem os conectores para ligar estes periféricos, não existem módulos disponíveis para a utilização dos mesmos.

Foram criados módulos que possibilitassem a utilização dos botões e LEDs da plataforma de desenvolvimento e do LCD já integrado na mesma. Também foram criados módulos que permitissem a conexão de teclados e ratos que utilizem o protocolo PS/2, assim como módulos que possibilitam a comunicação série usando os protocolos UART/RS-232. Foram ainda desenvolvidos dois módulos VGA, gráfico e texto. O primeiro permite o controlo total, pixel a pixel, podendo-se construir graficamente qualquer figura. O segundo permite a impressão de caracteres no monitor.

A principal utilização prevista para estes módulos é a integração dos mesmos em projectos no âmbito do plano de estudos do MIEEC, tornando esses projectos mais completos e apelativos aos estudantes, motivando-os a aprender mais nesta área.

Para além dos módulos anteriores, que foram definidos como objectivos mínimos para o trabalho desta dissertação de mestrado, foi ainda construído um conjunto de módulos extra, dois que implementam o protocolo SVGA, em modo gráfico e de texto, outros módulos que permitem alternar entre o protocolo VGA e SVGA, em modo gráfico e de texto, e um módulo que permite a integração de uma coluna de som à plataforma de desenvolvimento, reproduzindo um conjunto de sons.

Para efectuar a verificação e validação dos módulos foram desenvolvidas duas aplicações demonstrativas dos mesmos, que implementam dois jogos clássicos: *Breakout*, também conhecido por *Brick Breaker* ou *Arkanoid*, e *Pong*.

Foram ainda desenvolvidos manuais dos módulos e aplicações desenvolvidas, que indicam as suas ligações externas e o seu significado, assim como o seu funcionamento interno detalhado, permitindo a futuros utilizadores e projectistas escolher entre usar os módulos disponibilizados, ou altera-los conforme os seus próprios objectivos.

Palavras-chave: FPGAs, Verilog, Módulos, Periféricos de I/O.

Abstract

The FPGAs are assuming an increasingly role in the field of microelectronics. This is due to the fact that these integrated circuits can perform the same logic functions of an ASIC, having the advantage of being reconfigurable, even after they had been sent to the customer. The advantages are even greater in the process of developing a new system, where the correction of an ASIC prototype implies the fabrication of a new integrated circuit, leading to huge production costs.

As part of this dissertation it was intended the creation of a library of Verilog modules, easily integratable and usable, allowing the interconnection between various I/O devices and the FPGA, since the development platforms used throughout the MIEEC have connectors to these devices, there are no modules available to use them.

Modules were created to enable the use of buttons and LEDs on the development platform and LCD already integrated in it. Modules were also created to allow to connect keyboards and mice that use the PS/2 protocol, as well as modules that enable serial communication using the UART/RS-232 protocols. There were also two VGA modules, graphics and text. The first allows complete control, pixel by pixel and can graphically build any figure. The second allows the insertion of characters in the monitor.

The main intended use is the integration of the modules in projects within the curriculum of MIECC, making them more complete and appealing to students, motivating them to learn more in this area.

Besides the previous modules, that were considered de minimum target of the work of this dissertation, some other modules were developed, two allowing the implementation of the SVGA protocol, in graphical and text mode, other modules that allowed the transition between VGA and SVGA protocols, in both modes, and a module that enabled the connection of a small sound speaker to the platform, playing a set of different sounds.

To perform the verification and validation of the modules, were developed two example applications that implement two classic games: Breakout, also known as Brick Breaker or Arkanoid, and Pong.

Manuals of the modules and applications were also created, indicating their external connections and functions, as well as their detailed internal logic, allowing future users and designers to choose between the uses of the provided modules, or changing them according to their own objectives.

Keywords: *FPGAs, Verilog, Modules, I/O Peripherals.*

Agradecimentos

Aproveito o momento para agradecer a todas as pessoas que, de forma directa ou indirecta, me ajudaram na elaboração desta dissertação de mestrado.

Ao meu orientador, Professor Hélio Mendes de Sousa Mendonça, queria agradecer toda a disponibilidade, orientação, ideias, sugestões e argumentos que me guiaram na elaboração de todo o trabalho desta dissertação.

Queria agradecer também à minha mãe e avós por toda a educação e carinho que me deram e que me possibilitaram o acesso a um ciclo de estudos superior. A eles agradeço o inestimável apoio familiar que contribuiu decisivamente na minha educação e realização pessoal e profissional.

À minha namorada, Sandra, que sempre me apoiou, motivou e inspirou com a sua presença, nunca desistindo de me ajudar em tudo o que pudesse e que me encorajou a superar todos os momentos menos bons durante esta dissertação e durante todo o percurso académico superior. A ti, muito obrigado.

Aos meus amigos mais próximos, por todos os momentos de descontração e pela sua amizade, que sem dúvida ajudam a enfrentar os desafios futuros de uma forma mais alegre e motivada.

“If at first you don't succeed, before you try again, stop to figure out what you did wrong.”

Leo Rosten

Índice

Resumo	iii
Abstract.....	v
Agradecimentos	vii
Índice.....	ix
Lista de figuras	xiii
Lista de tabelas	xvii
Abreviaturas	xxi
Capítulo 1	1
Introdução.....	1
1.1 - Enquadramento	1
1.2 - Motivação	2
1.3 - Objectivos.....	2
1.4 - Estrutura da Dissertação.....	3
Capítulo 2	5
Estado da Arte.....	5
2.1 - Introdução.....	5
2.2 - FPGAs	5
2.3 - Plataforma de Desenvolvimento	7
2.4 - Verilog	9
2.5 - Ferramentas de Desenvolvimento	9
2.6 - Trabalhos Realizados.....	11
Capítulo 3	13
Desenvolvimento e Implementação	13
3.1 - Introdução.....	13
3.2 - Pesquisa	13
3.2.1 - Botões e LEDs	14
3.2.2 - LCD.....	16
3.2.3 - PS/2.....	19
3.2.4 - UART e RS-232	20
3.2.5 - VGA	21

3.2.6 - SVGA	22
3.2.7 - Wii Cam	23
3.3 - Criação dos Módulos	24
3.3.1 - Botões e LEDs	24
3.3.2 - LCD	26
3.3.3 - PS/2	29
3.3.4 - Teclado	35
3.3.5 - Rato	39
3.3.6 - UART	43
3.3.7 - VGA (Modo Gráfico)	49
3.3.8 - VGA (Modo Texto)	50
3.3.9 - SVGA (Modo Gráfico)	53
3.3.10 - SVGA (Modo Texto)	54
3.3.11 - VGA e SVGA (Modo Gráfico)	57
3.3.12 - VGA e SVGA (Modo Texto)	59
3.3.13 - Som	61
3.4 - Criação das Aplicações	62
3.4.1 - Breakout	63
3.4.2 - Pong	70
3.5 - Criação dos Manuais	79
Capítulo 4	81
Resultados	81
4.1 - Botões e LEDs	82
4.2 - LCD	82
4.3 - Teclado	83
4.4 - Rato	84
4.5 - UART	84
4.6 - VGA (Modo Gráfico)	85
4.7 - VGA (Modo Texto)	86
4.8 - SVGA (Modo Gráfico)	86
4.9 - SVGA (Modo Texto)	87
4.10 - VGA e SVGA (Modo Gráfico)	88
4.11 - VGA e SVGA (Modo Texto)	89
4.12 - Som	91
4.13 - Breakout	92
4.14 - Pong	93
Capítulo 5	95
Conclusões	95
5.1 - Conclusões Finais	95
5.2 - Trabalhos Futuros	97
Anexo A	99
Manuais	99
A.1 - Botões e LEDs	100
A.2 - LCD	102
A.3 - Teclado	104
A.4 - Rato	110
A.5 - UART	116
A.6 - VGA (Modo Gráfico)	121
A.7 - VGA (Modo Texto)	123
A.8 - SVGA (Modo Gráfico)	125
A.9 - SVGA (Modo Texto)	127
A.10 - VGA e SVGA (Modo Gráfico)	130
A.11 - VGA e SVGA (Modo Texto)	132
A.12 - Som	134
A.13 - Breakout	136

A.14 - Pong	142
Referências	149

Lista de figuras

Figura 2.1 - Esquemático dos fios e das ligações programáveis [15].....	6
Figura 2.2 - Dispositivos disponíveis na família Spartan-3E da Xilinx[16].	7
Figura 2.3 - Famílias de FPGAs disponibilizadas pela Xilinx.	8
Figura 2.4 - Plataforma de desenvolvimento usada e seus conectores [17].	8
Figura 2.5 - Aplicativo Xilinx ISE 11 visualizando as estatísticas de utilização.	10
Figura 2.6 - Aplicativo ModelSIM XE III 6.4b visualizando os sinais.	11
Figura 3.1 - Esquema de ligação dos botões de pressão [11].	14
Figura 3.2 - Esquema de ligação do botão rotativo [11].	14
Figura 3.3 - Esquema de ligação dos LEDs [11].	15
Figura 3.4 - Esquemático de ligação do LCD e do Intel StrataFlash [12].	16
Figura 3.5 - Rotina de inicialização do LCD [13].	17
Figura 3.6 - Descrição do parâmetro Entry Mode do LCD [13].	17
Figura 3.7 - Descrição do parâmetro Display Control do LCD [13].	17
Figura 3.8 - Lista de comandos do LCD disponíveis [12].	18
Figura 3.9 - Sequência de escrita de 4 bits no LCD [12].	18
Figura 3.10 - Sequência de escrita de um byte no LCD [12].	19
Figura 3.11 - Transmissão do periférico PS/2 para o anfitrião [7].	19
Figura 3.12 - Transmissão do anfitrião para o periférico PS/2 [7].	20
Figura 3.13 - Esquemático da ligação física da Wii Cam [18].	23
Figura 3.14 - Interface de ligação da Wii Cam utilizada.	24
Figura 3.15 - Módulo “Buttons.v”.	25
Figura 3.16 - Módulo “LCD.v”.	27

Figura 3.17 - Endereços das posições de memória do LCD [12].	27
Figura 3.18 - Máquina de estados do módulo "LCD.v".	28
Figura 3.19 - Hierarquia dos módulos que implementam o protocolo PS/2.	29
Figura 3.20 - Módulo "PS2_tx.v".	30
Figura 3.21 - Máquina de Estados do módulo "PS2_tx.v".	32
Figura 3.22 - Módulo "PS2_rx.v".	33
Figura 3.23 - Máquina de estados do módulo "PS2_rx.v".	34
Figura 3.24 - Módulo "PS2.v".	34
Figura 3.25 - Hierarquia dos módulos que controlam o teclado PS/2.	35
Figura 3.26 - Módulo "Keyboard.v".	36
Figura 3.27 - Máquina de estados do módulo "Keyboard.v".	38
Figura 3.28 - Hierarquia dos módulos que controlam o rato PS/2.	39
Figura 3.29 - Módulo "Mouse.v".	40
Figura 3.30 - Máquina de estados do módulo "Mouse.v".	42
Figura 3.31 - Hierarquia dos módulos que implementam o protocolo UART.	43
Figura 3.32 - Módulo "Baud.v".	44
Figura 3.33 - Módulo "UART_rx.v".	44
Figura 3.34 - Máquina de estados do módulo "UART_rx.v".	46
Figura 3.35 - Módulo "UART_tx.v".	46
Figura 3.36 - Máquina de estados do módulo "UART_tx.v".	48
Figura 3.37 - Módulo "UART.v".	48
Figura 3.38 - Módulo "VGA_sync.v".	49
Figura 3.39 - Hierarquia dos módulos que implementam o protocolo VGA em modo de texto.	51
Figura 3.40 - Módulo "VGA.v".	52
Figura 3.41 - Módulo "SVGA_sync.v".	53
Figura 3.42 - Hierarquia dos módulos que implementam o protocolo SVGA em modo de texto.	55
Figura 3.43 - Módulo "SVGA_text.v".	56
Figura 3.44 - Hierarquia dos módulos que implementam os protocolos VGA e SVGA.	57
Figura 3.45 - Módulo "VGA_SVGA.v".	58

Figura 3.46 - Hierarquia dos módulos que implementam os protocolos VGA e SVGA em modo de texto.	59
Figura 3.47 - Módulo “VGA_SVGA_text.v”.	59
Figura 3.48 - Módulo “Sound.v”.	61
Figura 3.49 - Ligação da coluna à plataforma de desenvolvimento.	62
Figura 3.50 - Hierarquia dos módulos que implementam a aplicação “Breakout”.	63
Figura 3.51 - Módulo “Breakout_VGA.v”.	64
Figura 3.52 - Aspecto do LCD durante o jogo “Breakout”.	66
Figura 3.53 - Módulo “Breakout_Move.v”.	67
Figura 3.54 - Módulo “Breakout.v”.	69
Figura 3.55 - Aspecto do monitor VGA durante a aplicação “Breakout”.	70
Figura 3.56 - Hierarquia dos módulos que implementam a aplicação “Pong”.	71
Figura 3.57 - Esquemático do conversor de linha implementado [14].	72
Figura 3.58 - Conversor de linha ligado à plataforma de desenvolvimento.	72
Figura 3.59 - Vista da parte inferior do conversor de linha.	73
Figura 3.60 - Módulo “Pong_VGA.v”.	73
Figura 3.61 - Aspecto do LCD durante o jogo “Pong”.	75
Figura 3.62 - Módulo “Pong_Move.v”.	76
Figura 3.63 - Módulo “Pong.v”.	78
Figura 3.64 - Aspecto do monitor VGA durante a aplicação “Pong”.	79
Figura 3.65 - Montagem da plataforma de desenvolvimento para a aplicação “Pong”.	79
Figura A.1 - Módulo “Buttons.v”.	100
Figura A.2 - Módulo “LCD.v”.	102
Figura A.3 - Módulo “Keyboard.v”.	104
Figura A.4 - Módulo “PS2.v”.	105
Figura A.5 - Módulo “PS2_rx.v”.	106
Figura A.6 - Módulo “PS2_tx.v”.	107
Figura A.7 - Módulo “Mouse.v”.	110
Figura A.8 - Módulo “PS2.v”.	111
Figura A.9 - Módulo “PS2_rx.v”.	112

Figura A.10 - Módulo “PS2_tx.v”.	113
Figura A.11 - Módulo “UART.v”.	116
Figura A.12 - Módulo “Baud.v”.	117
Figura A.13 - Módulo “UART_rx.v”.	118
Figura A.14 - Módulo “UART_tx.v”.	119
Figura A.15 - Módulo “VGA_sync.v”.	121
Figura A.16 - Módulo “VGA.v”.	123
Figura A.17 - Módulo “SVGA_sync.v”.	125
Figura A.18 - Módulo “SVGA_text.v”.	127
Figura A.19 - Módulo “VGA_SVGA.v”.	130
Figura A.20 -Módulo “VGA_SVGA_text.v”.	132
Figura A.21 - Módulo “Sound.v”.	134
Figura A.22 - Módulo “Breakout.v”.	136
Figura A.23 - Módulo “Breakout_Move.v”.	137
Figura A.24 - Módulo “Breakout_VGA.v”.	138
Figura A.25 - Módulo “Pong.v”.	142
Figura A.26 - Módulo “Pong_Move.v”.	143
Figura A.27 - Módulo “Pong_VGA.v”.	145

Lista de tabelas

Tabela 2.1 – Temporizações Horizontais do modo VGA.	21
Tabela 2.2 – Temporizações Verticais do modo VGA.	21
Tabela 2.3 – Temporizações Horizontais do modo SVGA.	22
Tabela 2.4 – Temporizações Verticais do modo SVGA.	22
Tabela 4.1 – Recursos disponíveis na FPGA Spartan-3E.	81
Tabela 4.2 – Ocupação de recursos do módulo “Buttons.v”.	82
Tabela 4.3 – Ocupação de recursos do módulo “LCD.v”.	83
Tabela 4.4 – Ocupação de recursos dos módulos do teclado PS/2.	83
Tabela 4.5 – Ocupação de recursos dos módulos do rato PS/2.	84
Tabela 4.6 – Ocupação de recursos dos módulos de comunicação série.	85
Tabela 4.7 – Ocupação de recursos do módulo “VGA_sync.v”.	85
Tabela 4.8 – Ocupação de recursos dos módulos do modo de texto em VGA.	86
Tabela 4.9 – Ocupação de recursos do módulo “SVGA_sync.v”.	87
Tabela 4.10 – Ocupação de recursos dos módulos do modo de texto em SVGA.	87
Tabela 4.11 – Ocupação de recursos dos módulos que implementam VGA + SVGA (Gráfico).	88
Tabela 4.12 – Ocupação de recursos dos módulos que implementam VGA + SVGA (Gráfico), estando fixo a VGA.	88
Tabela 4.13 – Ocupação de recursos dos módulos que implementam VGA + SVGA (Gráfico), estando fixo a SVGA.	89
Tabela 4.14 – Ocupação de recursos dos módulos que implementam VGA + SVGA (Texto). .	90

Tabela 4.15 – Ocupação de recursos dos módulos que implementam VGA + SVGA (Texto), estando fixo a VGA.	90
Tabela 4.16 – Ocupação de recursos dos módulos que implementam VGA + SVGA (Texto), estando fixo a SVGA.	91
Tabela 4.17 – Ocupação de recursos do módulo “Sound.v”.	91
Tabela 4.18 – Ocupação de recursos dos módulos mais importantes.	92
Tabela 4.19 – Ocupação de recursos dos módulos que implementam o jogo Breakout.	92
Tabela 4.20 – Percentagem de contribuição de cada um dos módulos na ocupação de recursos do jogo Breakout.	93
Tabela 4.21 – Ocupação de recursos dos módulos que implementam o jogo Pong.	93
Tabela 4.22 – Percentagem de contribuição de cada um dos módulos na ocupação de recursos do jogo Pong.	94
Tabela A.1 – Sinais do módulo “Buttons.v”.	100
Tabela A.2 – Ocupação de recursos do módulo “Buttons.v”.	101
Tabela A.3 – Sinais do módulo “LCD.v”.	102
Tabela A.4 – Ocupação de recursos do módulo “LCD.v”.	103
Tabela A.5 – Sinais do módulo “Keyboard.v”.	104
Tabela A.6 – Sinais do módulo “PS2.v”.	105
Tabela A.7 – Sinais do módulo “PS2_rx.v”.	106
Tabela A.8 – Sinais do modulo “PS_tx.v”.	107
Tabela A.9 – Ocupação de recursos dos módulos do teclado PS/2.	108
Tabela A.10 – Sinais do módulo “Mouse.v”.	110
Tabela A.11 – Sinais do módulo “PS2.v”.	111
Tabela A.12 – Sinais do módulo “PS2_rx.v”.	112
Tabela A.13 – Sinais do módulo “PS2_tx.v”.	113
Tabela A.14 – Ocupação de recursos dos módulos do rato PS/2.	114
Tabela A.15 – Sinais do módulo “UART.v”.	116
Tabela A.16 – Sinais do módulo “Baud.v”.	117
Tabela A.17 – Sinais do módulo “UART_rx.v”.	118
Tabela A.18 – Sinais do módulo “UART_tx.v”.	119
Tabela A.19 – Ocupação de recursos dos módulos de comunicação série.	120
Tabela A.20 – Sinais do modulo “VGA_sync.v”.	121

Tabela A.21 – Ocupação de recursos do módulo “VGA_sync.v”.	122
Tabela A.22 – Sinais do módulo “VGA.v”.	123
Tabela A.23 – Ocupação de recursos dos módulos do modo de texto em VGA.	124
Tabela A.24 – Sinais do módulo “SVGA_sync.v”.	126
Tabela A.25 – Ocupação de recursos do módulo “SVGA_sync.v”.	126
Tabela A.26 – Sinais do módulos “SVGA_text.v”.	127
Tabela A.27 – Ocupação de recursos dos módulos do modo de texto em SVGA.	128
Tabela A.28 – Sinais do módulo “VGA_SVGA.v”.	130
Tabela A.29 – Ocupação de recursos dos módulos que implementam VGA + SVGA (Gráfico).	131
Tabela A.30 – Sinais do módulo “VGA_SVGA_text.v”.	132
Tabela A.31 – Ocupação de recursos dos módulos que implementam VGA + SVGA (Texto).	133
Tabela A.32 – Sinais do módulo “Sound.v”.	135
Tabela A.33 – Ocupação de recursos do módulo “Sound.v”.	135
Tabela A.34 – Sinais do módulo “Breakout.v”.	136
Tabela A.35 – Sinais do módulo “Breakout_Move.v”.	137
Tabela A.36 – Sinais do módulo “Breakout_VGA.v”.	139
Tabela A.37 – Ocupação de recursos dos módulos que implementam o jogo Breakout.	140
Tabela A.38 – Sinais do módulo “Pong.v”.	142
Tabela A.39 – Sinais do módulo “Pong_Move.v”.	143
Tabela A.40 – Sinais do módulo “Pong_VGA.v”.	145
Tabela A.41 – Ocupação de recursos dos módulos que implementam o jogo Pong.	147

Abreviaturas

Lista de abreviaturas (ordenadas por ordem alfabética)

ADC	<i>Analog to Digital Converter</i>
ASCII	<i>American Standard Code for Information Interchange</i>
ASIC	<i>Application-Specific Integrated Circuit</i>
DAC	<i>Digital to Analog Converter</i>
FEUP	Faculdade de Engenharia da Universidade do Porto
FPGA	<i>Field-Programmable Gate Array</i>
FSM	<i>Finite State Machine</i>
HDL	<i>Hardware Description Language</i>
I/O	<i>Input / Output</i>
I ² C	<i>Inter-Integrated Circuit</i>
LCD	<i>Liquid Crystal Display</i>
LED	<i>Light Emitting Diode</i>
LUT	<i>Look Up Table</i>
MIEEC	Mestrado Integrado de Engenharia Electrotécnica e Computadores
PS/2	<i>IBM Personal System /2</i>
RAM	<i>Random Access Memory</i>
ROM	<i>Read Only Memory</i>
RTL	<i>Register Transfer Level</i>
UART	<i>Universal Asynchronous Receiver/Transmitter</i>
VESA	<i>Video Electronics Standards Association</i>
VGA	<i>Video Graphics Array</i>
VHDL	<i>VHSIC Hardware Description Language</i>
VHSIC	<i>Very High Speed Integrated Circuits</i>
SVGA	<i>Super Video Graphics Array</i>

Capítulo 1

Introdução

Esta dissertação de mestrado foi realizada no âmbito do Mestrado Integrado de Engenharia Electrotécnica e Computadores, na Faculdade de Engenharia da Universidade do Porto (FEUP).

Neste capítulo será apresentada a temática da dissertação, de forma a enquadrar a importância do trabalho realizado no desenvolvimento de novas aplicações baseadas em FPGAs, assim como a motivação e objectivos associados à sua elaboração. Este capítulo inclui ainda uma breve descrição da estrutura adoptada para a dissertação.

1.1 - Enquadramento

No decorrer dos últimos anos tem-se observado um crescimento exponencial dos equipamentos que utilizam circuitos integrados com aplicação específica (ASICs). Todos esses circuitos integrados têm de ser projectados, construídos, verificados, validados e testados, e caso não correspondam aos requisitos, melhorados. Caso o protótipo fosse elaborado sobre a forma de um ASIC, qualquer alteração que implicasse uma nova análise ao circuito levaria a produção de um novo protótipo. Neste caso o custo de produção seria muito elevado, já que na criação destes protótipos existem muitos custos associados. Por essa razão as FPGA têm assumido um papel cada vez mais importante nesta fase de criação de novos produtos, tendo como principal vantagem o facto de ser reprogramável. Como uma FPGA pode desempenhar as funções de um ASIC, e ambos são usualmente especificados por uma HDL, como por exemplo o Verilog, as FPGAs são as ferramentas ideais para a fase de projecto de um novo ASIC.

O Verilog foi a HDL escolhida para o desenvolvimento deste trabalho de dissertação, pois é a linguagem de descrição de hardware leccionada no decorrer do plano de estudos do MIEEC.

No entanto, as FPGAs não se destacam apenas nessa fase de projecto. Actualmente equipamentos electrónicos usados pelos consumidores comuns já incluem no seu interior FPGAs, de forma a se poder fazer actualizações aos equipamentos.

Tendo em conta estes factos, surge a necessidade de interacção entre o projectista/utilizador e a FPGA. É na resolução deste problema que esta dissertação se enquadra. Ao criar módulos que possibilitem a comunicação com a FPGA através de dispositivos de I/O já utilizados noutras situações, facilitamos a integração da FPGA no projecto, ou no equipamento.

1.2 - Motivação

Tendo em conta a crescente importância das FPGAs, torna-se necessária uma maior formação dos projectistas nesta área. Considerando o plano actual de estudos do MIEEC, o qual eu frequentei, a experiência com estes recursos é limitada, o que torna difícil a integração de um graduado neste mercado de trabalho específico. Tendo em vista melhorar o contacto dos alunos deste curso com esta ferramenta, foram criados os módulos, descritos posteriormente nesta dissertação. Com a introdução destes módulos o contacto dos alunos com as FPGAs pode ser mais aprofundado e motivador, e os projectos que estes desenvolvem nestas ferramentas podem tornar-se mais complexos, aumentando o seu nível de conhecimento, sem no entanto aumentar em demasia o seu grau de dificuldade.

Considerando estes factos, os módulos teriam que ser criados de forma a funcionar da forma mais flexível e versátil possível, de forma a serem reutilizados em diferentes projectos da maneira mais simples possível.

1.3 - Objectivos

Assumindo então a necessidade de criação destes módulos, definiram-se objectivos para o trabalho a realizar no âmbito desta dissertação de mestrado. Os objectivos propostos dividem-se em quatro partes. A primeira relativa às características de todos os módulos, a segunda relativa aos módulos específicos em si, a terceira relativa aos manuais dos módulos desenvolvidos e a quarta referente à criação de uma aplicação final que demonstrasse a utilização dos módulos desenvolvidos.

Tendo em conta esta divisão, serão de seguida apresentados os objectivos.

Os módulos desenvolvidos devem ser de fácil integração e simples de utilizar, proporcionando aos futuros utilizadores uma utilização transparente, e sem necessidade de pesquisa extensa sobre o seu funcionamento. Devem ainda ser versáteis e apresentar o maior número de funcionalidades possível, facilmente seleccionáveis, de modo a cobrirem a maior parte, se não a totalidade dos casos de utilização pretendidos.

Cada um dos módulos deve respeitar o respectivo protocolo de comunicação, abrangendo o maior número de configurações e parâmetros possível. Isto possibilita que mais dispositivos periféricos sejam abrangidos, desde que cumpram as normas do protocolo de comunicação. Os protocolos de comunicação que foram propostos a serem objecto de criação de módulos são:

- PS\2 (teclado).
- PS\2 (rato).
- VGA (gráfico).
- SVGA (gráfico).
- VGA (texto).

- SVGA (texto).
- UART.
- Som.

Para além destes protocolos, propôs-se ainda criar módulos para interagir com os seguintes dispositivos:

- LCD da plataforma de desenvolvimento.
- Botões da plataforma de desenvolvimento (unbouncing).

Para efectuar a verificação e validação dos módulos criados, foram desenvolvidas duas aplicações, que demonstram também exemplos de utilização dos módulos. Estas duas aplicações implementam os jogos *Breakout* e *Pong*.

Os manuais devem de respeitar um formato *standard*. Devem de estar disponíveis de forma permanente, de fácil consulta e navegação. Devem incluir vários esquemáticos que indiquem as ligações externas e sinais/módulos internos, que serão descritos, para além de toda a informação sobre a lógica implementada, para que um utilizador mais experiente, que queira alterar o módulo para outros objectivos, o compreenda mais facilmente.

As aplicações finais têm como objectivo exemplificar a utilização dos módulos de comunicação desenvolvidos, de uma forma interessante e demonstrativa das capacidades dos mesmos. A aplicação deve também conter um manual próprio.

1.4 - Estrutura da Dissertação

No capítulo 2 serão apresentados o estado da arte das FPGAs, a plataforma de desenvolvimento utilizada na elaboração da parte prática desta dissertação, um breve resumo sobre a linguagem Verilog, a sua importância e as suas vantagens, os protocolos de comunicação pesquisados de forma a construir os módulos e um breve resumo dos trabalhos que foram encontrados nesta área.

No capítulo 3 será descrita toda a fase de desenvolvimento e implementação dos módulos, referindo os dados importantes recolhidos durante a pesquisa sobre os protocolos de comunicação, o processo de criação dos módulos, a fase de criação dos manuais dos módulos desenvolvidos. Será ainda descrita a criação das aplicações finais e os seus manuais.

No capítulo 4, designado por Resultados, será descrito em detalhe o funcionamento de cada um dos módulos, assim como serão analisados vários aspectos técnicos resultantes da implementação dos mesmos.

No capítulo 5, apresentar-se-ão as conclusões obtidas com a realização deste trabalho. Será também feita uma análise crítica dos resultados obtidos, apresentando os erros cometidos e aspectos que poderiam ser melhorados, em perspectivas de trabalhos futuros.

No apêndice A serão ainda divulgados os manuais dos respectivos módulos desenvolvidos.

Capítulo 2

Estado da Arte

2.1 - Introdução

Neste capítulo serão apresentados diversos aspectos relacionados com as FPGAs, bem como uma descrição detalhada da plataforma de desenvolvimento usada no decorrer desta dissertação. Serão ainda apresentados alguns aspectos da linguagem HDL Verilog, assim como o grupo de programas utilizados, e a informação sobre a actualidade dos protocolos utilizados na construção dos módulos. Para concluir este capítulo serão referidos alguns trabalhos já realizados nesta área que foram pesquisados.

2.2 - FPGAs

As FPGAs são, como já referido, dispositivos com muitas vantagens na área do *design* de lógica digital. O utilizador pode criar o seu projecto no seu computador, e implementá-lo na FPGA em poucos minutos, sendo assim uma ferramenta de uso prático e eficiente, mantendo a sua extrema flexibilidade.

Com o decréscimo no custo inicial desta ferramenta, e com a disponibilização gratuita de *software* de desenvolvimento pelos fabricantes juntamente com a disponibilização de manuais e fóruns *on-line*, permite que mais profissionais e entusiastas se aventurem neste universo dos sistemas digitais.

Com a principal desvantagem frente aos ASICs, a velocidade de funcionamento, a ser cada vez menor, a FPGA torna-se a ferramenta de eleição no desenvolvimento de novos projectos devido à sua reduzida complexidade no desenvolvimento, permitindo um *time-to-market* menor, reduzido custo de investigação, e diminuição dos custos em engenharia não recursiva. Se a isto se juntar o facto de algumas FPGAs poderem ser parcialmente reconfiguradas, permitindo que só se altere parte do circuito programado, verifica-se que as FPGAs são de facto a ferramenta ideal para o desenvolvimento de projectos de lógica digital, no enquadramento tecnológico e económico actual.

Internamente as FPGAs são constituídas por células de I/O, ligações reconfiguráveis e Blocos Lógicos que podem ter nomes e funções diferentes conforme os fabricantes (CLBs no

caso da *Xilinx*). Geralmente os *Logic Blocks* são constituídos por algumas células (*Slices* no caso da *Xilinx*) sendo cada uma constituída, geralmente, por uma *Lookup Table* (LUT) de 4 entradas, um *Full-Adder* (FA) e um *Flip-Flop* do tipo D. Alguns fabricantes têm introduzido nos seus modelos de topo LUTs de 6 entradas, reivindicando uma melhor performance.

Uma vez que o sinal de relógio é, em circuitos síncronos (predominantes), o sinal mais importante e utilizado, este é encaminhado por ligações especificamente desenhadas, suportando assim o elevado *fan-out*, e distribuídos ao longo da FPGA com uma topologia específica, de forma a reduzir ao máximo o *Clock Skew* (atraso do sinal de relógio em algumas células comparado com outras). Ao conduzir este sinal separadamente dos outros reduz-se ainda a ocorrência de *glitches* no sinal de relógio, provenientes de capacidades parasitas entre ligações.

As ligações de outros sinais são normalmente fios, cuja ligação entre eles é programável, permitindo encaminhar os sinais para o seu destino específico.

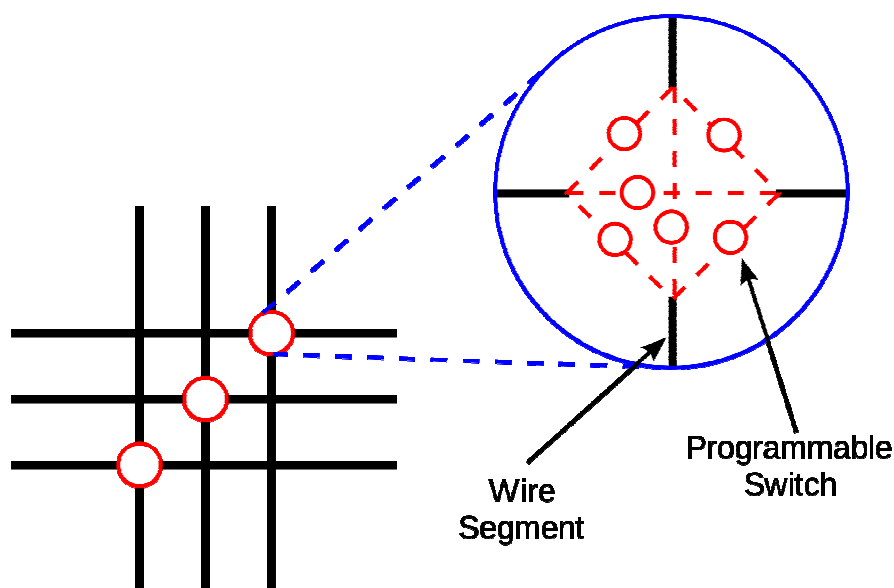


Figura 2.1 - Esquemático dos fios e das ligações programáveis [15].

Durante o processamento do projecto o *software* de desenvolvimento deverá configurar o circuito para uma plataforma específica, pois as ligações e blocos lógicos estão organizados de forma diferente em quase todas as FPGAs. Compete ainda ao programa gerir as ligações e blocos lógicos de forma a obter o melhor desempenho possível, que pode ser melhorado usando regras simples de optimização na construção do projecto.

A medição da capacidade de uma FPGA é difícil de efectuar, devido aos diversos factores que podem condicionar um projecto, desde as ligações disponíveis, aos componentes das *slices* até aos blocos de memória RAM. No entanto é frequente a *Xilinx* (fabricante da maioria das FPGAs utilizadas na FEUP e nesta dissertação) medir as capacidades destes dispositivos pelo número de *Logic Blocks*, ou pelo número de *System Gates*.

2.3 - Plataforma de Desenvolvimento

Durante o desenvolvimento dos módulos propostos para esta dissertação, foi usada uma plataforma de desenvolvimento que incluía, para além da FPGA, outros sistemas que permitem vários tipos de conexões com periféricos.

Esta plataforma de desenvolvimento da *Xilinx* designada por *Spartan-3E Starter Kit* contém um dispositivo com a referência XC3S500E e enquadra-se da seguinte forma na família *Spartan-3E*.

Device	XC3S100E	XC3S250E	XC3S500E	XC3S1200E	XC3S1600E
System Gates	100K	250K	500K	1200K	1600K
Logic Cells	2,160	5,508	10,476	19,512	33,192
Dedicated Multipliers	4	12	20	28	36
Block RAM Blocks	4	12	20	28	36
Block RAM Bits	72K	216K	360K	504K	648K
Distributed RAM Bits	15K	38K	73K	136K	231K
DCMs	2	4	4	8	8
Max Differential I/O	40	68	92	124	156
Max Single Ended I/O	108	172	232	304	376
VQ100	66	66			
CP132		92	92		
TQ144	108	108			
PQ208		158	158		
FT256		172	190	190	
FG320			232	250	250
FG400				304	304
FG484					376

Figura 2.2 - Dispositivos disponíveis na família *Spartan-3E* da *Xilinx* [16].

As FPGAs da família *Spartan* pertencem ao segmento de FPGAs *low-cost* da *Xilinx*, apresentando no entanto boas características de funcionamento para as situações, maioritariamente educacionais, que ocorrem nos trabalhos desenvolvidos na FEUP.

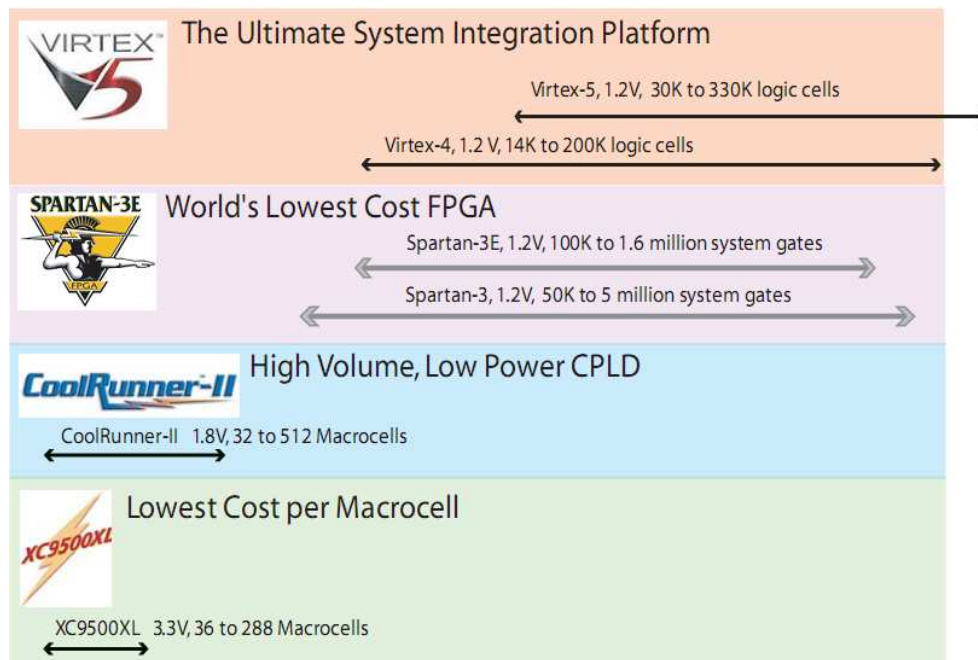


Figura 2.3 - Famílias de FPGAs disponibilizadas pela Xilinx.

Integrados nesta plataforma existem vários conectores a dispositivos de I/O, tais como um conector PS\2, dois conectores DB9 destinados a comunicação série (um conector macho e outro fêmea), um conector VGA, um conector de saída de um DAC, a entrada de um ADC e ainda vários portos de expansão, podendo ser usados para múltiplas ligações.



Figura 2.4 - Plataforma de desenvolvimento usada e seus conectores [17].

Com todas estas características esta plataforma de desenvolvimento é a indicada para se realizar o trabalho desta dissertação, pois possui todas as capacidades necessárias para o trabalho a desenvolver.

2.4 - Verilog

O Verilog, linguagem HDL usada no design, teste e implementação de lógica digital, foi criado em 1983/4 por Phil Moorby e Prabhu Goel. No Verilog a descrição de um circuito situa-se no nível de abstracção *Register Transfer Level* (RTL), ou seja, um circuito digital síncrono é implementado através do encaminhamento de sinais entre registos (usualmente implementados como *Flip-Flops* tipo D) e a realização de operações lógicas sobre esses sinais.

É no controlo sobre a propagação de sinais e dependências destes, que reside a diferença entre as HDL e as linguagens de programação comuns. Para controlar os sinais são usados dois tipos de operadores: *blocking* (=) e *non-blocking* (<=). O uso de atribuições do tipo *blocking* geralmente implica que o circuito não seja síncrono, pois obriga a que a instrução seguinte só seja processada, quando o sinal já tiver o valor atribuído. Este tipo de atribuições é muito usado quando existem cálculos recursivos, em que os valores têm que estar correctos antes da operação a realizar. As atribuições do tipo *non-blocking* originam normalmente circuitos síncronos com o relógio, sendo possível definir se a alteração do valor de todos os sinais ocorre no flanco ascendente ou descendente do relógio.

Para a criação de circuitos lógicos mais avançados, é necessário mais do que alterar o valor dos sinais, e para isso existe um conjunto de instruções que ajudam ao controlo do sistema pretendido, tais como o *if/else* e *case*. Em comparação com a linguagem C existem apenas duas grandes diferenças no uso destes comandos, que são a necessidade de especificação da base numérica dos dados a utilizar na condição (binária, decimal, hexadecimal, etc.) e o início e fim da condição é determinado por *begin/end* em vez de *{}*.

Um projecto normalmente assenta na criação de um ou mais módulos, visto que se pode utilizar um módulo dentro de outro módulo. Os módulos possuem um conjunto de ligações que podem ser de entrada (*input*), saída (*output*) ou bidireccionais (*inout*). Os sinais internos de um módulo podem ser de vários tipos, no entanto os mais utilizados são os fios (*wire*) e os registos (*reg*). Nos primeiros não se podem atribuir valores directamente, pois apenas servem para ligar registos, ligações externas do módulo ou verificar condições. Existem 4 tipos de valores que os fios podem assumir: '1', '0', alta impedância 'z' e indefinido 'x'. Os registos são utilizados no armazenamento de dados temporários, e o seu valor pode ser alterado directamente. Para além destes dois tipos é ainda usado na criação dos módulos desta dissertação outro tipo de dados designado por parâmetro (*parameter*) que são sinais que têm um valor pré-definido, que não pode ser alterado através da lógica combinatória do sistema, pois esse valor é fixado durante a síntese do circuito. No entanto, se o módulo que contém o parâmetro estiver dentro de outro módulo, o módulo principal pode passar, como argumento, o valor pretendido para o parâmetro, mas essa alteração tem que se efectuar sempre antes da sintetização.

A sintetização é um processo realizado pela ferramenta de desenvolvimento, que transforma o código Verilog no mapa do circuito a implementar na FPGA.

2.5 - Ferramentas de Desenvolvimento

Tal como já foi referido, actualmente os fabricantes de FPGAs já disponibilizam gratuitamente ferramentas de desenvolvimento e teste destinadas aos seus produtos. No caso

desta dissertação, com o uso da plataforma *Spartan-3E* da *Xilinx*, a ferramenta disponibilizada pelo fabricante e utilizada na fase de desenvolvimento dos módulos foi a *Xilinx ISE Design Suite 11*. Esta ferramenta contém um conjunto de programas essencial para a criação de qualquer projecto baseado numa FPGA da *Xilinx*, uma vez que suporta praticamente todos os modelos fabricados. Dentro desse conjunto de programas, e no decorrer da parte de implementação desta dissertação, destaca-se o uso de dois: *Xilinx ISE 11* e o *IMPACT*. Estes dois programas têm objectivos diferentes na implementação.

O *Xilinx ISE 11* é uma ferramenta que proporciona a criação e desenvolvimento de um projecto. Esta aplicação permite editar os ficheiros em linguagem Verilog, que contém os módulos a desenvolver, verificar e corrigir a sua sintaxe, navegar entre módulos entre outras funcionalidades. No entanto, as principais funções deste aplicativo destinam-se à criação de um ficheiro (.bit) que contém todas as informações necessárias para implementar o projecto na FPGA. Durante a criação desse ficheiro várias etapas são percorridas tais como a síntese, implementação do design e a criação do ficheiro em si. No final destes processos o programa apresenta ainda um resumo dos componentes utilizados da FPGA, assim como detalhes sobre a propagação dos sinais, tais como a frequência máxima de funcionamento e o *fan-out* médio. Para além dos ficheiros Verilog dos módulos é ainda necessária a criação de um ficheiro (.ucf) que contenha as ligações externas da FPGA, bem como as suas restrições. Este ficheiro pode ser criado manualmente, como foi criado durante esta dissertação, ou, em casos de projectos com um grande volume de sinais, utilizando uma ferramenta disponibilizada na *Design Suite* designada *PlanAhead*.

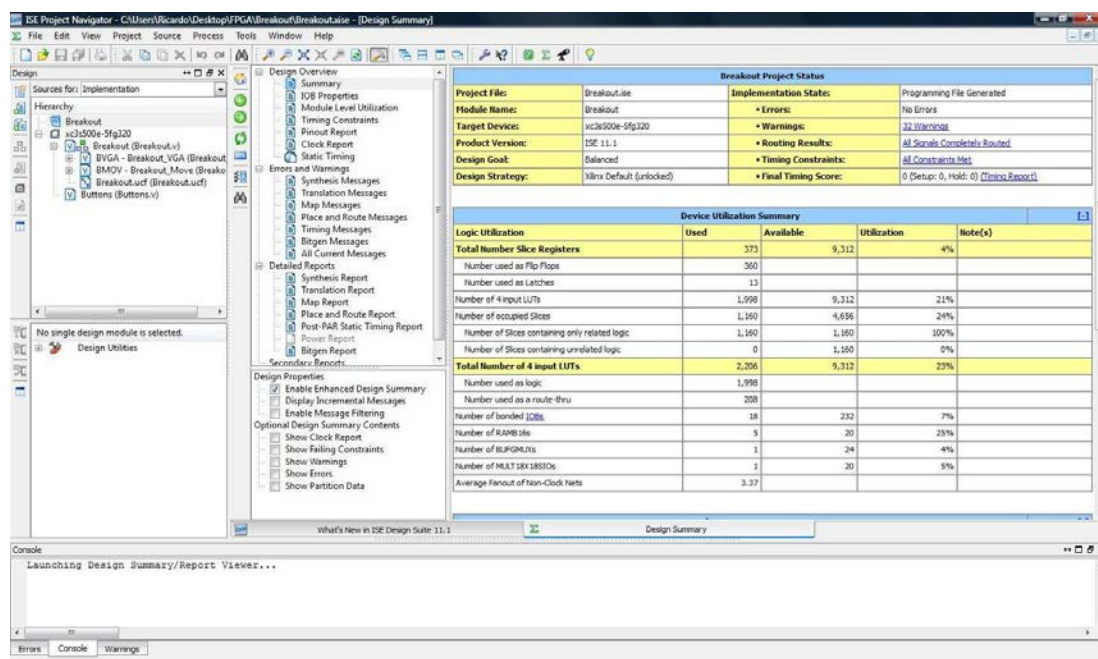


Figura 2.5 - Aplicativo *Xilinx ISE 11* visualizando as estatísticas de utilização.

Outra ferramenta utilizada, mas em menor escala, foi o *ModelSIM XE III 6.4b* que não pertence ao aglomerado de programas fornecidos pela *Xilinx*. Este programa permite a realização de simulações do módulo desenvolvido, através da criação de um outro módulo, *test-bench*, que cria estímulos destinados aos sinais de entrada do módulo testado, e permite observar os sinais de saída, sobre a forma de ondas, com referência temporal.

Este programa foi utilizado em menor escala, apenas em situações em que não era possível obter resultados através da implementação física dos módulos desenvolvidos. Isto porque a simulação de protocolos de comunicação por *test-bench* seria muito mais complexo do que a experimentação prática, que também permite a fácil detecção de pormenores a melhorar no módulo em desenvolvimento.

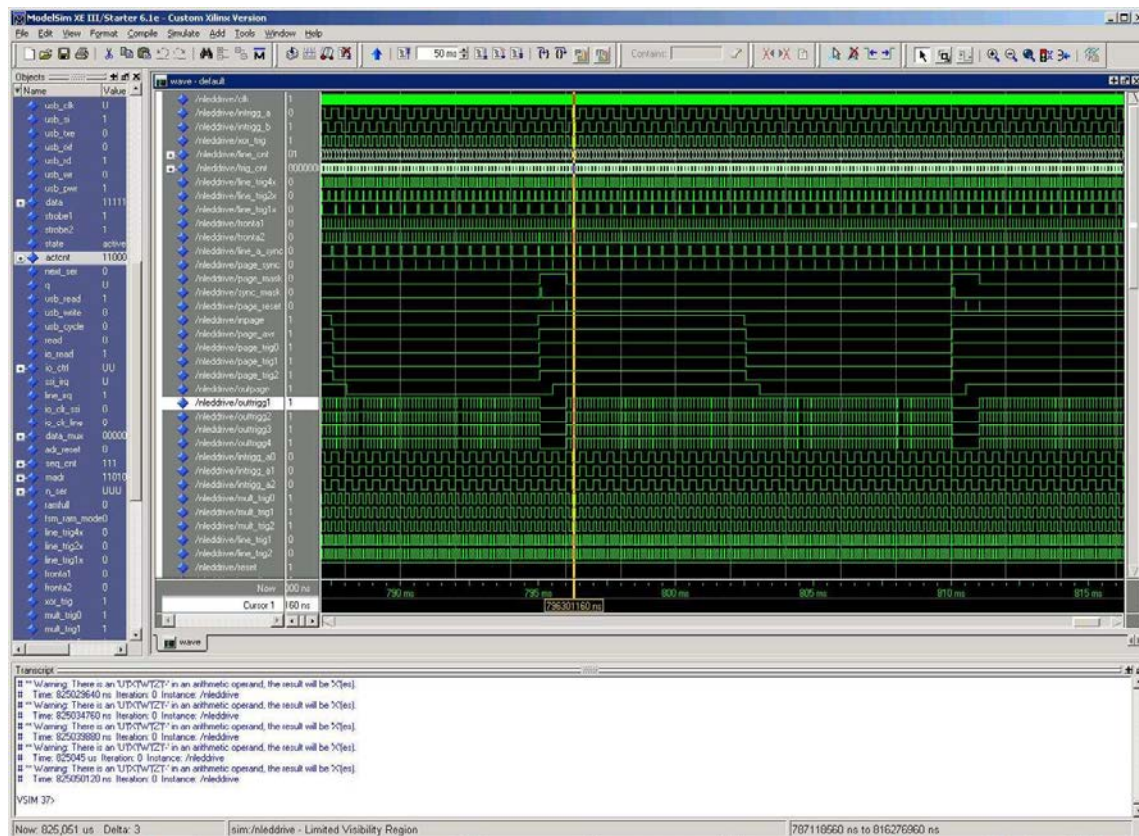


Figura 2.6 - Aplicativo ModelSIM XE III 6.4b visualizando os sinais.

2.6 - Trabalhos realizados

No decorrer da pesquisa para o trabalho desta dissertação foram encontrados vários projectos que cumpriam parte dos objectivos desta dissertação, sendo no entanto a maioria deles realizados em VHDL, que é também uma HDL e provavelmente a maior concorrente com o Verilog. A maior parte destes projectos encontrados focavam-se apenas na interface de um componente, e a reutilização dos módulos, quer estivessem eles projectados em Verilog ou em VHDL, era de elevada complexidade e por vezes bastante confusa.

Para além disso, a falta de conhecimentos em VHDL levou a um esforço extra, para tentar perceber esses módulos e aperfeiçoar e adicionar novos extras aos desenvolvidos durante esta dissertação. Alguns desses módulos foram nesse aspecto de muita utilidade, pois permitiram simplificar alguns processos, e abriram caminho a novas ideias a desenvolver. Esses projectos consultados serão agora enumerados e referidas as ideias retiradas para a construção dos módulos desta dissertação.

No desenvolvimento do módulo para o LCD da FPGA foi consultado um no fórum edaboard [4], escrito em Verilog. Deste projecto foram retiradas as ideias para a estrutura da máquina

de estados do módulo (que depois foi alterada para contemplar mais casos de utilização), e da organização dos dados enviados para o LCD.

No desenvolvimento do módulo para o *unbouncing* dos botões da FPGA foi analisado o módulo de demonstração da *Xilinx* para a plataforma de desenvolvimento utilizada [5], desenvolvido em VHDL, onde foram analisados os intervalos de tempo necessários para a eliminação do ruído mecânico introduzido no botão rotativo.

No desenvolvimento dos módulos para comunicação série, PS/2, VGA e SVGA (gráfico e texto) foram consultados vários aspectos teóricos sobre estes protocolos, de uma forma mais orientada ao desenvolvimento de projectos em FPGAs, assim como alguns diagramas que especificavam como deveriam funcionar as máquinas de estados de alguns módulos, no livro “*FPGA prototyping by VHDL examples: Xilinx Spartan-3 Version*” [6].

Para obter detalhes mais profundos sobre o protocolo PS/2, o funcionamento de teclados e ratos, foi consultada a página Web “Computer-Engineering.org” [7], [8] e [9].

Os detalhes sobre os protocolos VGA e SVGA também foram pesquisados em “*Hardware Level VGA and SVGA Video Programming Information Page*” [10].

Para a obtenção dos caracteres no modo de texto do controlador VGA desenvolvido foi utilizada uma ferramenta já desenvolvida pelo Professor Doutor Hélio Mendes de Sousa Mendonça, o orientador desta dissertação. Essa pequena aplicação produziu os dados necessários à reprodução dos caracteres contemplados no código ASCII, numa resolução de 8x8 pixéis.

Capítulo 3

Desenvolvimento e Implementação

3.1 - Introdução

Neste capítulo serão apresentados todos os dados relevantes recolhidos sobre os protocolos de comunicação que foram implementados. O procedimento de criação de cada um dos módulos é também descrito neste capítulo, fazendo referência à estrutura pretendida, à descrição do módulo e às suas características importantes. Será também apresentada a estrutura utilizada na criação dos manuais dos módulos. A criação das aplicações demonstrativas tem também relevo neste capítulo, sendo explicado os objectivos pretendidos com as suas criações, bem como os recursos nelas utilizados.

3.2 - Pesquisa

Numa fase precedente do trabalho desta dissertação de mestrado, existe uma unidade curricular no plano académico do MIEEC no *major* de telecomunicações, designada “Preparação para a Dissertação”, que consiste na presença em seminários que ajudam os estudantes a enfrentar os desafios futuros da dissertação de mestrado, bem como efectuar a pesquisa de dados necessários para que esta corra de uma forma mais suave. Todos os protocolos aqui referenciados foram pesquisados no âmbito dessa unidade curricular, e estão também presentes no relatório final elaborado para essa unidade curricular. Apenas pequenos detalhes dos protocolos, relacionados com a implementação dos módulos foram pesquisados durante a duração dos trabalhos desta dissertação, sendo que a restante informação utilizada, já estava descrita no referido relatório final.

De seguida serão apresentados os aspectos mais importantes de cada um dos protocolos de comunicação. No entanto, podem ser omissos alguns pormenores devido à sua pouca importância no âmbito desta dissertação.

3.2.1 - Botões e LEDs

Presentes na plataforma de desenvolvimento existem quatro interruptores, cinco botões de pressão e um botão rotativo. Os quatro interruptores, devido à sua natureza funcional, não necessitam da criação de um módulo que funcione como interface entre o seu circuito físico e o processamento lógico da FPGA. No entanto, os restantes necessitam da criação de um módulo que realize o *unbouncing* dos botões, devido ao ruído mecânico e de transição que estes introduzem nas mudanças de estado. Os cinco botões de pressão que existem encontram-se a norte, sul, este e oeste do botão rotativo, sendo que ao pressionar o mesmo se obtêm o quinto botão. Este botão rotativo também gira sobre si mesmo, daí a sua designação, no sentido horário e anti-horário.

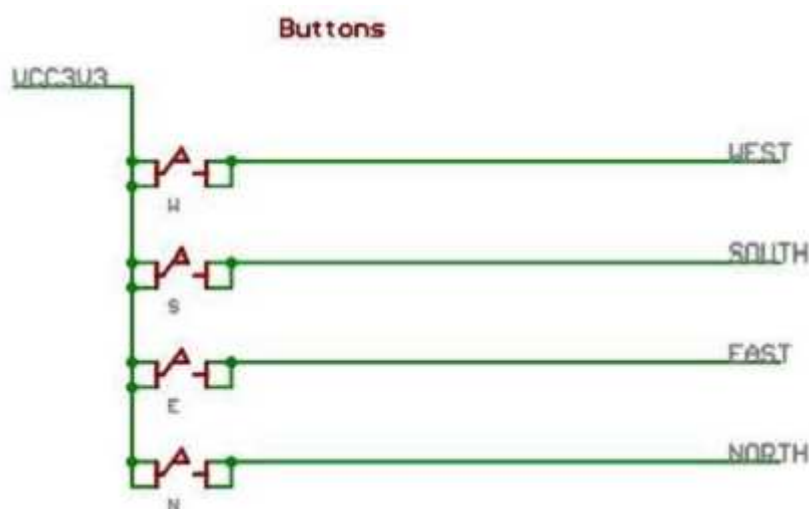


Figura 3.1 - Esquema de ligação dos botões de pressão [11].

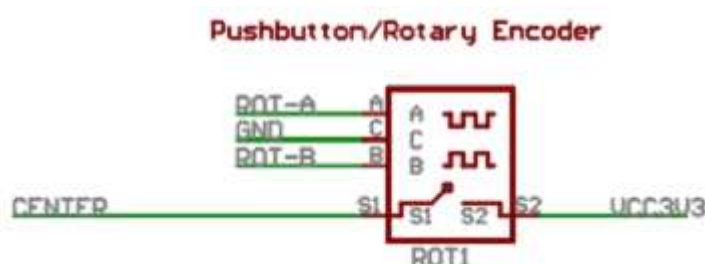


Figura 3.2 - Esquema de ligação do botão rotativo [11].

Para os cinco botões de pressão é necessário que exista um módulo que elimine o ruído de transição de estado, e o ruído mecânico originado pela pressão/libertação do botão. Também é necessário, devido à elevada frequência de funcionamento da FPGA comparado com a velocidade do corpo humano na pressão/libertação dos botões, que à saída do módulo de *unbouncing* apenas se obtenha um impulso com a duração de um ciclo de relógio, por cada vez que o botão é premido. Desta forma, a pressão do botão só é processada uma vez, independentemente do tempo que este esteja premido.

Para o botão rotativo é necessário um módulo que converta os dois sinais (ROT_A e ROT_B) em outros dois sinais, um que indique uma nova rotação, e outro que indique o sentido. É também necessário um mecanismo de *unbouncing* pois o ruído mecânico neste dispositivo é muito elevado.

Existem ainda na placa oito LEDs, que apesar de não necessitarem de qualquer rotina de *unbouncing*, seriam mais facilmente controlados através de um registo de oito bits, do que através de oito registos diferentes cada um com um bit.

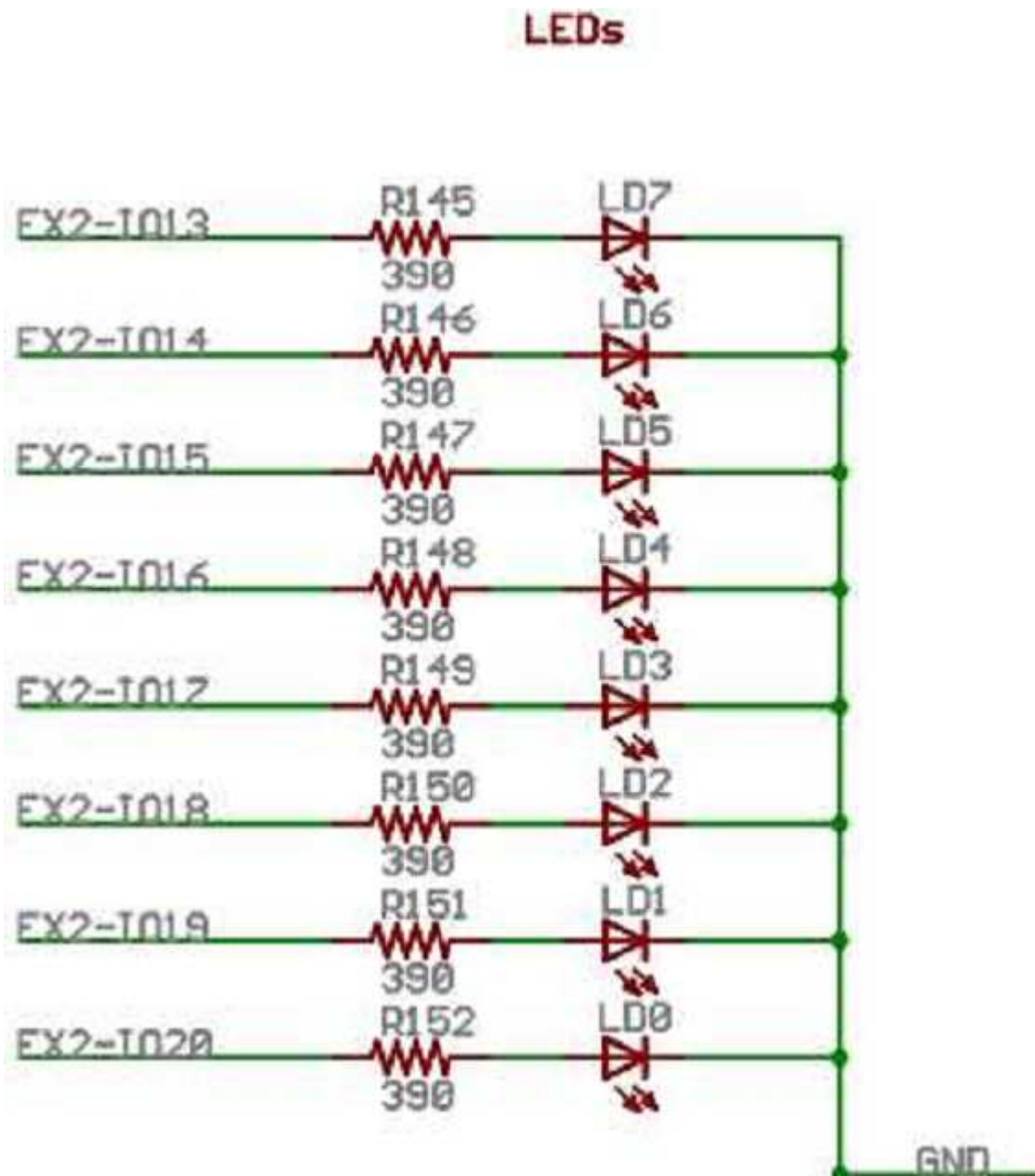


Figura 3.3 - Esquema de ligação dos LEDs [11].

3.2.2 - LCD

A plataforma de desenvolvimento utilizada durante o trabalho desta dissertação de mestrado possui integrada um LCD de duas linhas, cada uma com dezasseis caracteres visíveis e memória para um total de quarenta caracteres por linha.

A interface entre a FPGA e o LCD é constituída por quatro linhas de dados, três linhas de controlo e duas de alimentação. No entanto, essas quatro linhas de dados são partilhadas com o dispositivo *Intel StrataFlash* que tem que estar desactivado para o LCD funcionar.

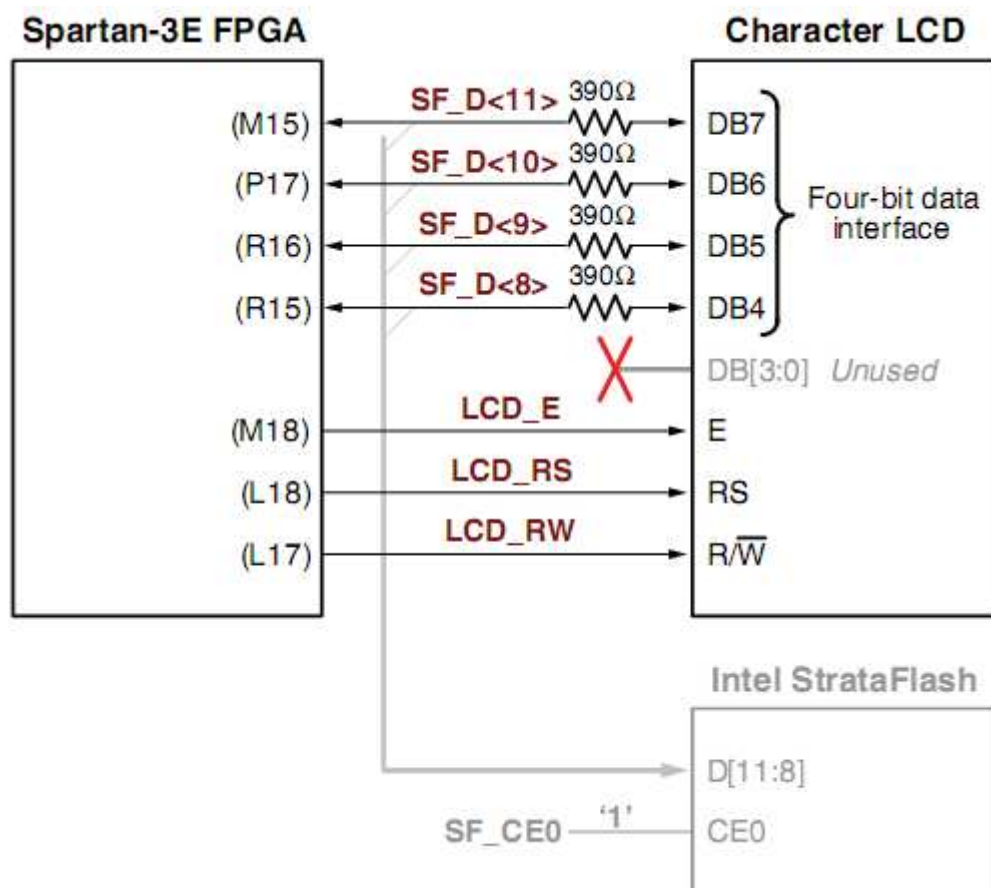


Figura 3.4 - Esquemático de ligação do LCD e do *Intel StrataFlash* [12].

Para o LCD funcionar correctamente é necessário percorrer uma sequência de inicialização, em que são configurados vários parâmetros do LCD.

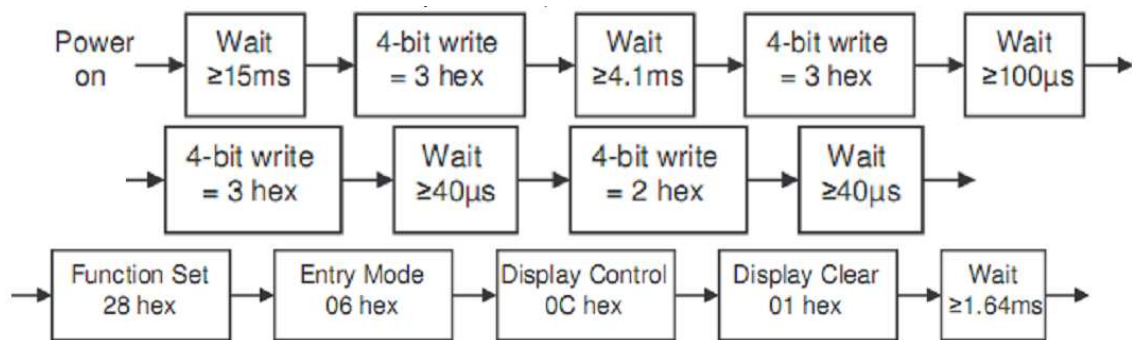
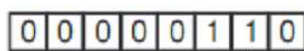


Figura 3.5 - Rotina de inicialização do LCD [13].

A plataforma de desenvolvimento apenas suporta uma função do LCD, que é definida pelo parâmetro *Function Set* na rotina de inicialização. Este parâmetro define a largura do barramento de dados, o número de linhas e o mapa de caracteres a usar.

O parâmetro *Entry Mode* possui quatro configurações possíveis, definidas pelos dois bits menos significativos. O bit menos significativo (DB0) quando está a '1' activa o *Shift* do LCD após a introdução de um carácter. O segundo bit menos significativo (DB1) indica para qual dos sentidos o cursor se vai deslocar após a introdução de um novo carácter.

Entry Mode = 06 hex



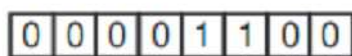
Do not shift display left or right when writing a character

Increment cursor position when writing a character

Figura 3.6 - Descrição do parâmetro *Entry Mode* do LCD [13].

Para definir várias definições de visualização existe o parâmetro *Display Control* que permite activar/desactivar o cursor (DB0), activar/desactivar a linha por baixo do carácter actual (DB1) e ligar/ desligar o LCD (DB2).

Display Control = 0C hex



Cursor blinking off (character flashes)

Cursor (line under character) is off

Display is ON

Figura 3.7 - Descrição do parâmetro *Display Control* do LCD [13].

Para além dos comandos utilizados na sequência de inicialização é possível utilizar outros durante o funcionamento do LCD.

Function	LCD_RS	LCD_RW	Upper Nibble				Lower Nibble			
			DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Clear Display	0	0	0	0	0	0	0	0	0	1
Return Cursor Home	0	0	0	0	0	0	0	0	1	-
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S
Display On/Off	0	0	0	0	0	0	1	D	C	B
Cursor and Display Shift	0	0	0	0	0	1	S/C	R/L	-	-
Function Set	0	0	0	0	1	0	1	0	-	-
Set CG RAM Address	0	0	0	1	A5	A4	A3	A2	A1	A0
Set DD RAM Address	0	0	1	A6	A5	A4	A3	A2	A1	A0
Read Busy Flag and Address	0	1	BF	A6	A5	A4	A3	A2	A1	A0
Write Data to CG RAM or DD RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0
Read Data from CG RAM or DD RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0

Figura 3.8 - Lista de comandos do LCD disponíveis [12].

Como todos os comandos e dados a escrever são registos de oito bits e a interface com o LCD apenas tem quatro linhas de comunicação, todos os dados têm que ser enviados para o LCD em duas partes de quatro bits (*nibble*), sendo que os quatro bits mais significativos são os primeiros a transmitir.

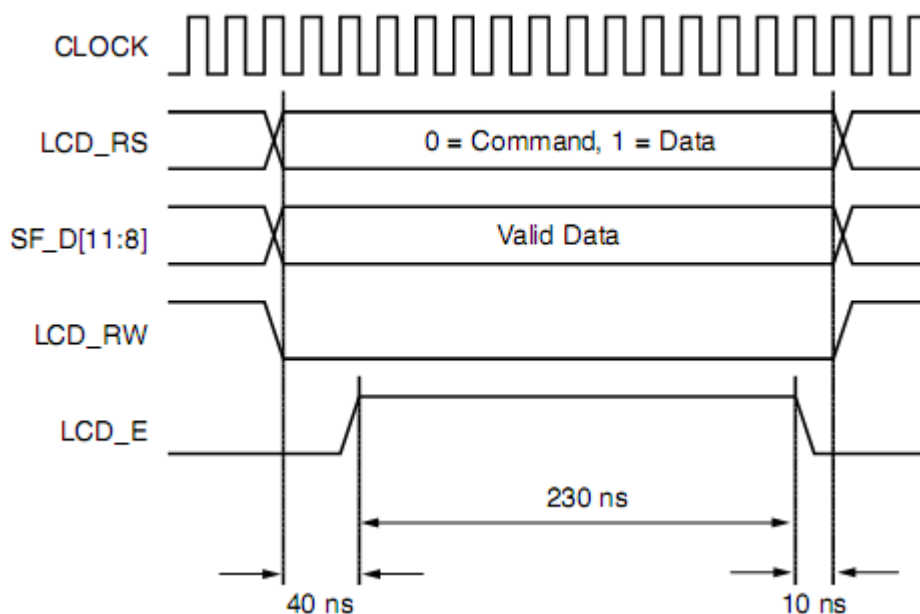


Figura 3.9 - Sequência de escrita de 4 bits no LCD [12].

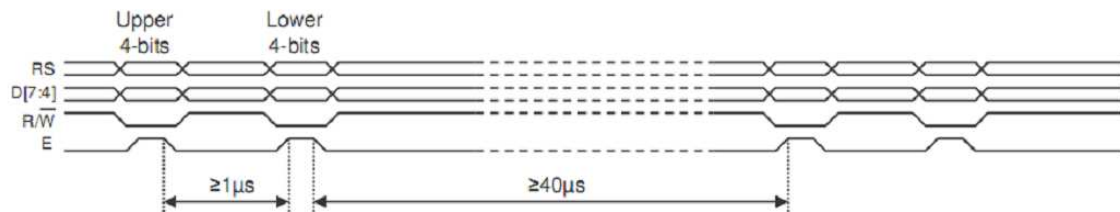


Figura 3.10 - Sequência de escrita de um byte no LCD [12].

Assim, é necessária a construção de um módulo que realize a inicialização do LCD, indique quando este está disponível para uma nova instrução e permita o envio de comandos e a escrita de caracteres.

3.2.3 - PS/2

O protocolo PS/2 [7], muito utilizado em ratos e teclados, permite a comunicação bidireccional (série) entre o anfitrião e o periférico.

São utilizadas duas linhas de comunicação, uma para o relógio e outra para os dados, e duas de alimentação, 5 e 0V.

Os dados são transmitidos em tramas de onze ou doze bits, dependendo do sentido da transmissão, sendo constituídas por um bit de início, oito bits de dados (bit menos significativo primeiro), um bit de paridade, um bit de fim e um bit de *acknowledge* (usado na comunicação do anfitrião para o periférico).

O relógio tem uma frequência entre 10 e 16.7 kHz, e é gerado pelo periférico (excepto quando o anfitrião bloqueia a recepção ou pede para iniciar transmissão 'Relógio = 0'). No anfitrião os dados são lidos na descida do relógio e no periférico os dados são lidos na subida do relógio.

Na transmissão de dados do periférico para o anfitrião, o dispositivo verifica primeiro se a linha de relógio está livre. Se estiver, envia o bit de início, o byte de dados e respectivo bit de paridade e o bit de paragem.

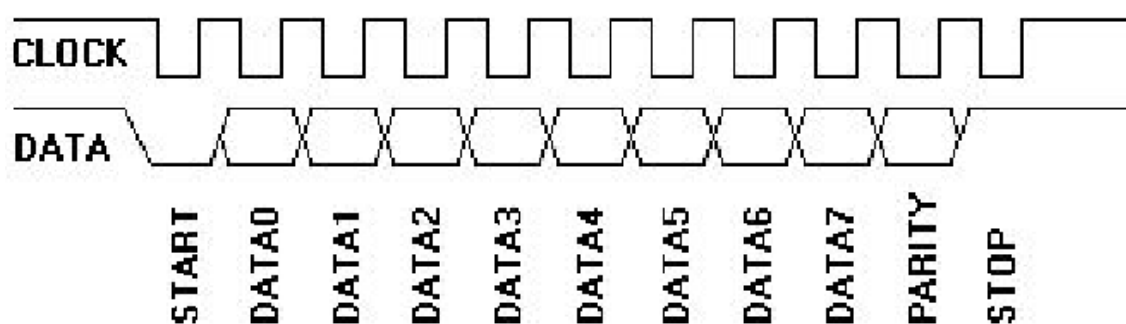


Figura 3.11 - Transmissão do periférico PS/2 para o anfitrião [7].

Quando a transmissão é do anfitrião para o periférico, o anfitrião deve ocupar a linha de relógio pelo menos durante 100 μs. Depois deverá enviar o bit de início e esperar que o periférico assuma controlo da linha de relógio, enviando de seguida o byte de dados, o bit de paridade e o bit de fim. No final deverá esperar pelo *acknowledge* enviado pelo periférico.

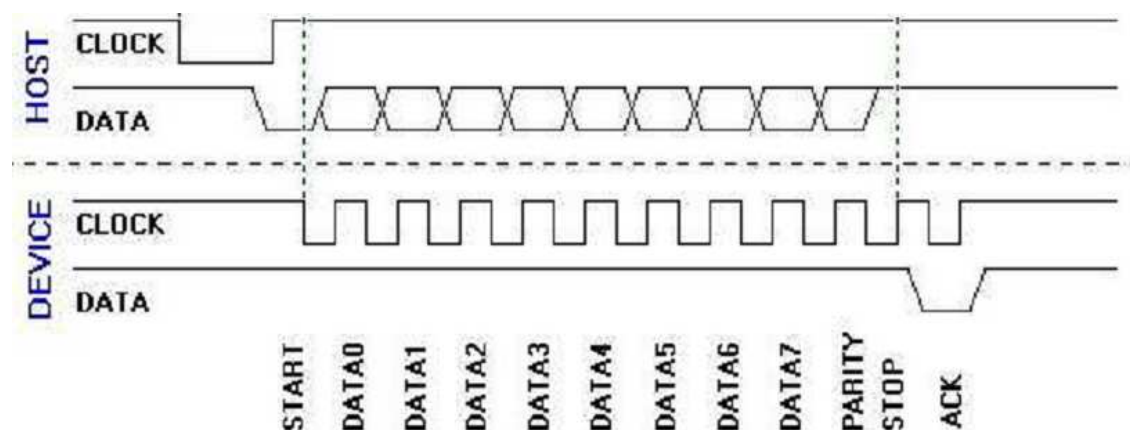


Figura 3.12 - Transmissão do anfitrião para o periférico PS/2 [7].

É necessária então a criação de um módulo para transmissão e outro para recepção, sendo depois englobados por um módulo que regule os conflitos entre os dois módulos referidos anteriormente.

Estes módulos devem funcionar apenas respeitando o protocolo PS/2, sendo que o processamento de dados provenientes conforme o dispositivo ligado (teclado ou rato) deverá ser efectuado num módulo de hierarquia superior, para que estes módulos sejam iguais para os dois tipos de periféricos.

3.2.4 - UART e RS-232

Os protocolos UART/RS-232 combinados originam o que vulgarmente se designa por comunicação através da porta série, sendo que o protocolo UART especifica o modo como os dados são transmitidos e recebidos e o protocolo RS-232 especifica as características físicas da comunicação.

O protocolo RS-232 já se encontra implementado pelo circuito integrado presente na plataforma de desenvolvimento. Este circuito é muito usado e denomina-se MAX232, permitindo a utilização de duas portas série separadas e na plataforma de desenvolvimento estão implementados um conector macho e um conector fêmea, ambos no formato DE-9.

Este protocolo apenas define as características eléctricas, as interfaces de ligação e as funções de cada circuito nas interfaces de ligação. Todas as restantes definições necessárias para comunicar são estabelecidas pelo protocolo UART.

O protocolo UART (*Universal Asynchronous Receiver/Transmitter*) permite uma comunicação assíncrona, podendo ser *full-duplex* ou *half-duplex*. Este protocolo especifica que a trama deve ser constituída por um bit de início, cinco a oito bits de dados (bit menos significativo primeiro), um bit de paridade opcional, um, um e meio ou dois bits de fim.

Como se trata de uma comunicação assíncrona, não existe linha de relógio, pelo que os dispositivos a comunicar entre si, devem ter definidas previamente as condições temporais de amostragem das linhas, designado *Baudrate*. Este *Baudrate* pode assumir vários valores diferentes e é expresso em impulsos por segundo. Existe ainda outro parâmetro temporal designado *Oversample* que indica o número de vezes que cada bit deve ser amostrado.

Assim sendo é necessário uma rotina para converter a *Baudrate* e o *Oversample*, que poderão ser alterados, em ciclos de relógio da FPGA, para saber quando se deve amostrar a linha.

3.2.5 - VGA

O protocolo VGA foi criado pela IBM em 1987 [10] juntamente com o PS/2, e foi adoptado como o standard de visualização analógica em computadores, sendo usado actualmente em quase todos os sistemas operativos durante o arranque e antes de serem carregados outro tipo de drivers.

O seu modo de funcionamento comum permite uma resolução de 640x480 pixéis refrescando o ecrã com uma frequência de 60 Hz. Para obter este modo de funcionamento é necessária uma frequência de pixel (*Pixel Rate*) de 25.175MHz, o que pode ser aproximado para 25MHz, que é metade da frequência do relógio interno da plataforma de desenvolvimento, sem qualquer inconveniente.

No entanto, ainda é necessário saber os parâmetros horizontais e verticais estabelecidos na norma VGA, para implementar este protocolo.

Tabela 2.1 – Temporizações Horizontais do modo VGA.

Descrição	Nº. De Pixéis	Tempo [μ s]
<i>Visible Area</i>	640	25.42
<i>Front Porch</i>	16	0.64
<i>Retrace</i>	96	3.81
<i>Back Porch</i>	48	1.90
<i>Total</i>	800	31.78

Tabela 2.2 – Temporizações Verticais do modo VGA.

Descrição	Nº. De Pixéis	Tempo [μ s]
<i>Visible Area</i>	480	15.25
<i>Front Porch</i>	10	0.32
<i>Retrace</i>	2	0.06
<i>Back Porch</i>	33	1.05
<i>Total</i>	535	16.68

O conector mais usado para implementar o protocolo VGA e também o presente na plataforma de desenvolvimento utilizada nesta dissertação é o DE-15, sendo que o lado do gerador do sinal de vídeo é sempre o conector fêmea. Neste conector destacam-se cinco sinais essenciais para a implementação do protocolo VGA: hsync, vsync, R, G e B.

Os sinais hsync e vsync são sinais de sincronização horizontal e vertical, respectivamente, utilizados para sincronizar o sinal de vídeo. Os sinais R, G e B são os sinais que simbolizam o vermelho, verde e azul respectivamente. É possível recombina estas cores, perfazendo um total de oito cores.

Ao agrupar um conjunto de pixéis (células), normalmente numa resolução de 8x8 ou 16x16 pixéis, e com acesso a uma biblioteca de caracteres e outros recursos de implementação, é possível incluir um modo de texto em VGA, com a impressão dos caracteres nas células.

3.2.6 - SVGA

O *Super Video Graphics Array* ou SVGA é um protocolo que estende o VGA, possibilitando uma maior resolução e *pixel rate* que este. O seu modo de funcionamento original, aprovado pela *Video Electronics Standard Association* (VESA) em 1989, permite uma resolução de 800 por 600 pixéis sendo que cada um tinha quatro bits permitindo atribuir ao pixel uma de dezasseis cores. Este protocolo pode ser aplicado na plataforma de desenvolvimento, ao contrário de muitos outros, devido ao seu *pixel rate*, que têm uma frequência igual à do relógio interno. Existem outras resoluções com um *pixel rate* menor. Porém, essas não podem ser utilizadas pois não é possível dividir o sinal de relógio para essas frequências. Outros modos de funcionamento são também excluídos por possuírem um *pixel rate* superior ao relógio interno da plataforma de desenvolvimento.

Assim sendo, o modo SVGA 800x600 pixéis foi escolhido para ser implementado na FPGA, sendo a única alteração em relação ao protocolo original o facto de apenas estarem disponíveis três bits para definir a cor do pixel, possibilitando um total de oito cores, tal como o VGA.

Tabela 2.3 – Temporizações Horizontais do modo SVGA.

Descrição	Nº. De Pixéis	Tempo [μ s]
<i>Visible Area</i>	800	16.00
<i>Front Porch</i>	56	1.12
<i>Retrace</i>	120	2.40
<i>Back Porch</i>	64	1.28
Total	1040	20.80

Tabela 2.4 – Temporizações Verticais do modo SVGA.

Descrição	Nº. De Pixéis	Tempo [μ s]
<i>Visible Area</i>	600	12.00
<i>Front Porch</i>	37	0.74
<i>Retrace</i>	6	0.12
<i>Back Porch</i>	23	0.46
Total	666	13.32

3.2.7 - Wii Cam

A *Wii Cam* é uma câmara destinada à detecção de LEDs de infra-vermelhos. Esta câmara está integrada no comando da consola de videojogos *Nintendo® Wii™ (Wiimote)* [21]. Este dispositivo permite a detecção de (até) quatro pontos de emissão de infra-vermelhos, devolvendo as coordenadas de cada um no seu referencial. Para comunicar essas coordenadas, o dispositivo utiliza uma versão modificada do protocolo de comunicação I²C. A utilização desta câmara fora do comando está ainda pouco documentada, por se tratar de uma adaptação de um recurso destinado a ser integrado no comando de uma consola de videojogos, não tendo sido divulgada pelo fabricante qualquer informação relativa ao seu funcionamento. Todos os dados conhecidos sobre este dispositivo foram obtidos através de outros entusiastas ([19] e [20]), com base no método tentativa erro.

Analizando o código disponibilizado pelos dois projectistas referidos, descobriu-se o endereço I²C da câmara, a sua rotina de inicialização e o método de obtenção das coordenadas dos quatro pontos de infra-vermelhos.

A adaptação física da câmara para efectuar a interface com outros controladores que não o *Wiimote*, é também descrita pelos outros projectistas ([19] e [20]), tendo já sido implementada num trabalho realizado na unidade curricular denominada “Sistemas Embutidos e de Tempo Real” (EEC0059), em que esta câmara era integrada com um microprocessador da família 80C51.

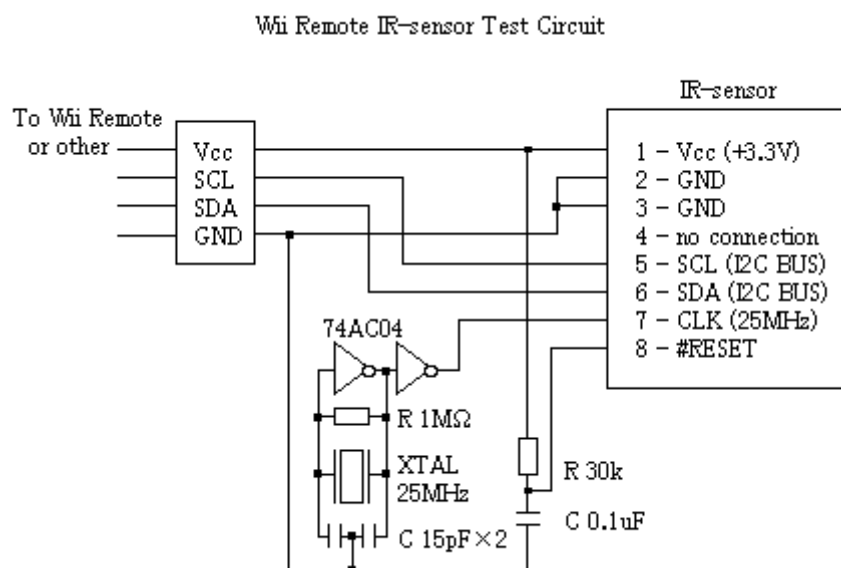


Figura 3.13 - Esquemático da ligação física da *Wii Cam* [18].

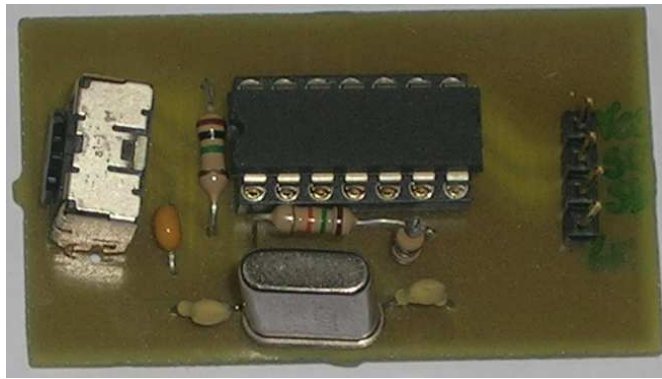


Figura 3.14 - Interface de ligação da Wii Cam utilizada.

Infelizmente, devido à escassez de tempo e pelo facto de esta interface se tratar de um extra que poderia ser adicionado ao trabalho desta dissertação, os módulos de controlo deste dispositivo não foram concluídos a tempo de os apresentar nesta dissertação.

3.3 - Criação dos Módulos

Nesta secção deste capítulo será descrita a estrutura, o funcionamento e as características importantes de cada um dos módulos desenvolvidos.

Todas as entradas de todos módulos com o nome “Clock” deverão ser ligadas ao relógio interno da plataforma de desenvolvimento, através da adição da seguinte linha ao ficheiro “.ucf” do projecto, especificando a localização do recurso e a utilização das linhas destinadas à propagação do sinal de relógio:

```
NET "clock" LOC = "C9" | IOSTANDARD = LVCMOS33;
```

Todas as entradas de todos módulos com o nome “Reset” deverão ser ligadas ao botão de pressão usado para efectuar o reset externo da plataforma de desenvolvimento. Normalmente utiliza-se o botão “South”. A linha de código a adicionar ao projecto deve ser semelhante às utilizadas no módulo dos botões de pressão, que será descrito de seguida, alterando apenas o nome da “NET” e colocar a localização do recurso pretendida.

É possível descarregar todos os módulos desenvolvidos no endereço de Internet destinado à divulgação dos trabalhos desta dissertação [18].

3.3.1 - Botões e LEDs

Neste módulo pretendia-se que fosse efectuado o *unbouncing* dos botões de pressão e botão rotativo da plataforma de desenvolvimento, bem como a ligação de todos os LEDs a um único registo.

Este módulo possui apenas um nível hierárquico, pelo que apenas é necessário o ficheiro “Buttons.v” para usufruir de todas as funcionalidades.

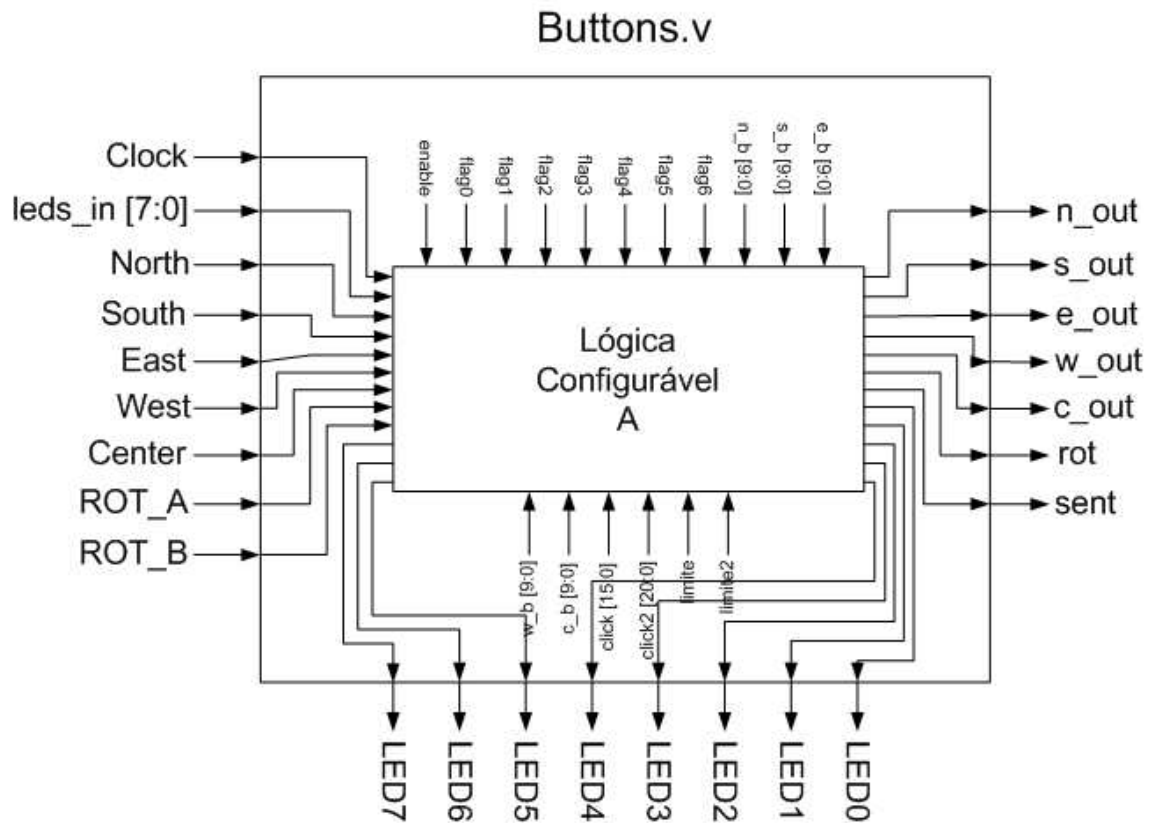


Figura 3.15 - Módulo "Buttons.v".

As entradas "North", "South", "East", "West" e "Center" provêm dos botões de pressão, e devem ser ligadas aos respectivos recursos através da adição das seguintes linhas ao ficheiro ".ucf" do projecto, especificando a localização do recurso, o tipo de ligações (fios) a usar e a ligação de uma resistência de *pull-down* ao recurso:

```
NET "North" LOC = "V4" | IOSTANDARD = LVTTTL | PULLDOWN;
NET "South" LOC = "K17" | IOSTANDARD = LVTTTL | PULLDOWN;
NET "East" LOC = "H13" | IOSTANDARD = LVTTTL | PULLDOWN;
NET "West" LOC = "D18" | IOSTANDARD = LVTTTL | PULLDOWN;
NET "Center" LOC = "V16" | IOSTANDARD = LVTTTL | PULLDOWN;
```

As entradas "ROT_A" e "ROT_B" são os sinais provenientes do botão rotativo, e devem ser ligadas aos respectivos recursos através da adição das seguintes linhas ao ficheiro ".ucf" do projecto, especificando a localização do recurso, o tipo de ligações (fios) a usar e a ligação de uma resistência de *pull-up* ao recurso:

```
NET "ROT_A" LOC = "K18" | IOSTANDARD = LVTTTL | PULLUP ;
NET "ROT_B" LOC = "G18" | IOSTANDARD = LVTTTL | PULLUP ;
```

A entrada "leds_in" especifica uma entrada de oito bits, cujo valor é atribuído pelo utilizador, em que cada um dos bits controla um LED. Por exemplo o LED0 é controlado pelo bit 0 da entrada leds_in.

As saídas "LED0", "LED1", "LED2", "LED3", "LED4", "LED5", "LED6" e "LED7" devem ser ligadas aos LEDs da plataforma de desenvolvimento, para que estes possam ser controlados pela entrada "leds_in". Estes podem ser ligados aos respectivos recursos através da adição das seguintes linhas ao ficheiro ".ucf" do projecto, especificando a localização do recurso, o

tipo de ligações (fios) a usar especificando que as transições devem de ser lentas e o nível da intensidade da corrente:

```
NET "LED7" LOC = "F9" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8;
NET "LED6" LOC = "E9" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8;
NET "LED5" LOC = "D11" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8;
NET "LED4" LOC = "C11" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8;
NET "LED3" LOC = "F11" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8;
NET "LED2" LOC = "E11" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8;
NET "LED1" LOC = "E12" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8;
NET "LED0" LOC = "F12" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8;
```

As saídas “n_out”, “s_out”, “e_out”, “w_out” e “c_out” representam os respectivos botões de pressão após o *unbouncing*.

Os sinais “rot” e “sent” indicam, depois do processamento da informação proveniente do botão rotativo e do respectivo *unbouncing*, se houve alguma rotação, e o sentido da rotação.

No interior do módulo existem diversos registos e parâmetros que são utilizados para realizar o *unbouncing* de todos os botões.

Existe no módulo um divisor de relógio. Este divisor incrementa o registo “click” a cada subida de relógio, e quando este atinge o valor do parâmetro “limite” os registos “n_b”, “s_b”, “e_b”, “w_b”, “c_b” são actualizados, e a contagem reinicia. Estes registos de dez bits guardam o estado dos botões nos últimos dez milissegundos. Quando um destes registos está completamente preenchido com ‘1’, o valor da respectiva saída é colocado a ‘1’, e uma *flag*, individual para cada botão, é activada de forma a garantir que a saída só fica a ‘1’ durante um ciclo de relógio e apenas uma vez por cada pressão do botão. Quando o botão é solto a *flag* e a saída do botão são colocadas a ‘0’

Com este método evitamos as flutuações de nível lógico introduzidas pelo ruído mecânico dos botões e impedimos que uma única pressão do botão crie inúmeras activações na respectiva saída.

Existe ainda no módulo um segundo divisor de relógio, com um maior intervalo de tempo, destinado ao botão rotativo. Este divisor de relógio incrementa o registo “click2” a cada subida de relógio, e quando este atinge o valor do parâmetro “limite2” cria um impulso, com a duração de um ciclo de relógio, no registo “enable” e reinicia a contagem. Sempre que ocorre este impulso é analisado o estado dos sinais provenientes do botão rotativo e, caso tenha havido uma rotação, os registos “rot” e “sent” são actualizados conforme o sentido da rotação. “rot” indica que houve uma rotação e “sent” indica o sentido da rotação (‘0’ para a esquerda e ‘1’ para a direita). Existem ainda duas *flags* para evitar que a mesma rotação seja processada duas ou mais vezes.

Desta forma o ruído de transição e o ruído mecânico do botão rotativo são eliminados, levando a uma utilização fluida deste recurso da plataforma de desenvolvimento.

3.3.2 - LCD

Ao criar este módulo pretendia-se tornar o envio de instruções para o LCD mais simples, de forma a este poder ser facilmente integrado num projecto, e permitir visualizar informação relativamente ao funcionamento da FPGA.

Este módulo possui apenas um nível hierárquico, pelo que apenas é necessário o ficheiro “LCD.v” para usufruir de todas as funcionalidades.

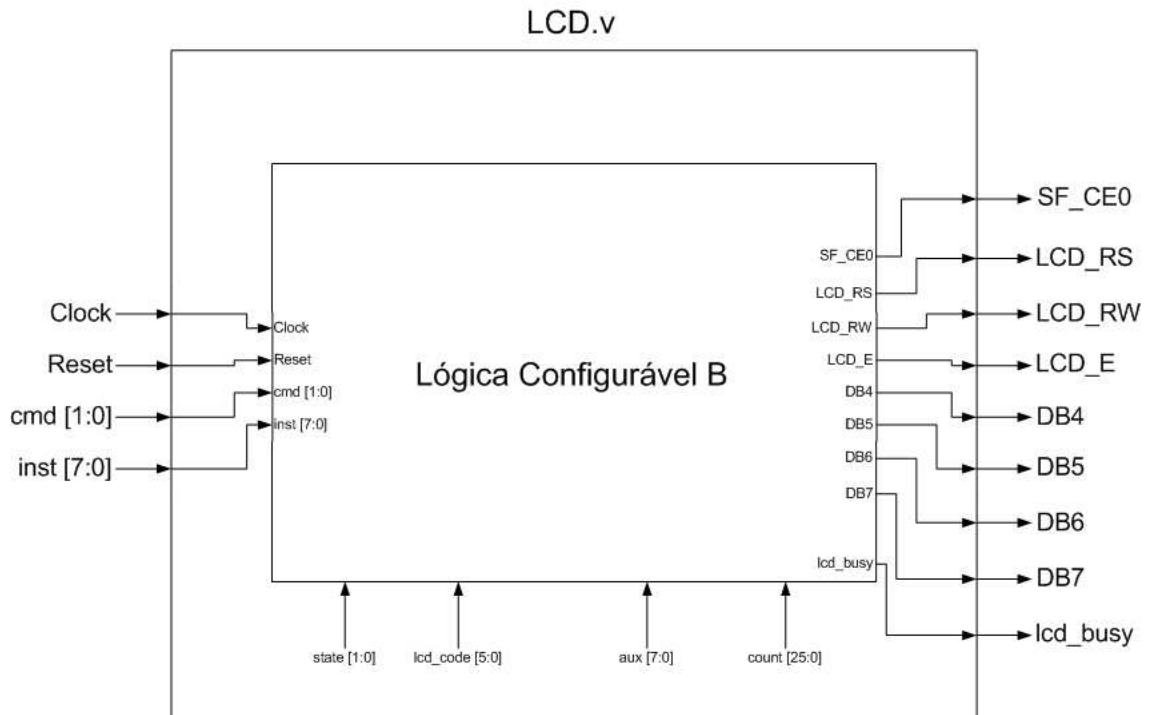


Figura 3.16 - Módulo "LCD.v".

Os sinais "cmd" e "inst" são controlados pelo utilizador e possibilitam um controle total sobre o dispositivo LCD. O registo "cmd" permite 3 opções, manter o LCD em estado *idle* (atribuir a "cmd" o valor 0 ou 3 decimal), escrever um carácter (atribuir 1 decimal a "cmd") e dar um comando ao LCD (atribuir a "cmd" o valor 2 decimal). O registo "inst" especifica a instrução a enviar ao LCD quando "cmd" é igual a 1 ou 2. Quando se pretende escrever um carácter, "inst" deverá ter o seu respectivo código ASCII. Quando se pretende dar um comando ao LCD, "inst" deverá ter o valor do respectivo comando.

Character Display Addresses																Undisplayed Addresses			
1	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	...	27
2	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	...	67
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	...	40

Figura 3.17 - Endereços das posições de memória do LCD [12].

As saídas "SF_CE0", "LCD_RS", "LCD_RW", "LCD_E", "DB4", "DB5", "DB6" e "DB7" representam os pinos de conexão ao LCD da plataforma de desenvolvimento e deverão ser ligados aos respectivos recursos através da adição das seguintes linhas ao ficheiro ".ucf" do projecto, especificando a sua localização, o tipo de ligações (fios) a utilizar, a intensidade de corrente e a velocidade de transição de sinal:

```
NET "LCD_E" LOC = "M18" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW;
NET "LCD_RS" LOC = "L18" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW;
NET "LCD_RW" LOC = "L17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW;
NET "DB4" LOC = "R15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW;
NET "DB5" LOC = "R16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW;
NET "DB6" LOC = "P17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW;
```

NET "DB7" LOC = "M15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW;

NET "SF_CE0" LOC = "D16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW;

O sinal de saída "lcd_busy" serve para indicar quando é que o LCD está a processar a instrução anterior e ainda não está disponível para uma nova instrução. Esta saída é necessária, pois a velocidade de funcionamento do LCD é muito inferior à velocidade da FPGA e é necessário saber quando o LCD está disponível para aceitar novas instruções. Esta é de facto a única inconveniência do LCD, pois é um dispositivo simples de acesso a informação interna da FPGA, pecando apenas na velocidade de actualização.

No interior deste módulo existem quatro registos auxiliares usados no controlo do LCD: "state", "lcdcode", "aux" e "count".

O primeiro é utilizado para controlar a máquina de estados de controlo do LCD. O estado '0' representa a inicialização do LCD e o estado '1' representa o estado *idle* em que o LCD está à espera de novas instruções. O estado '2' está destinado à escrita de caracteres no LCD e o estado '3' à escrita de comandos.

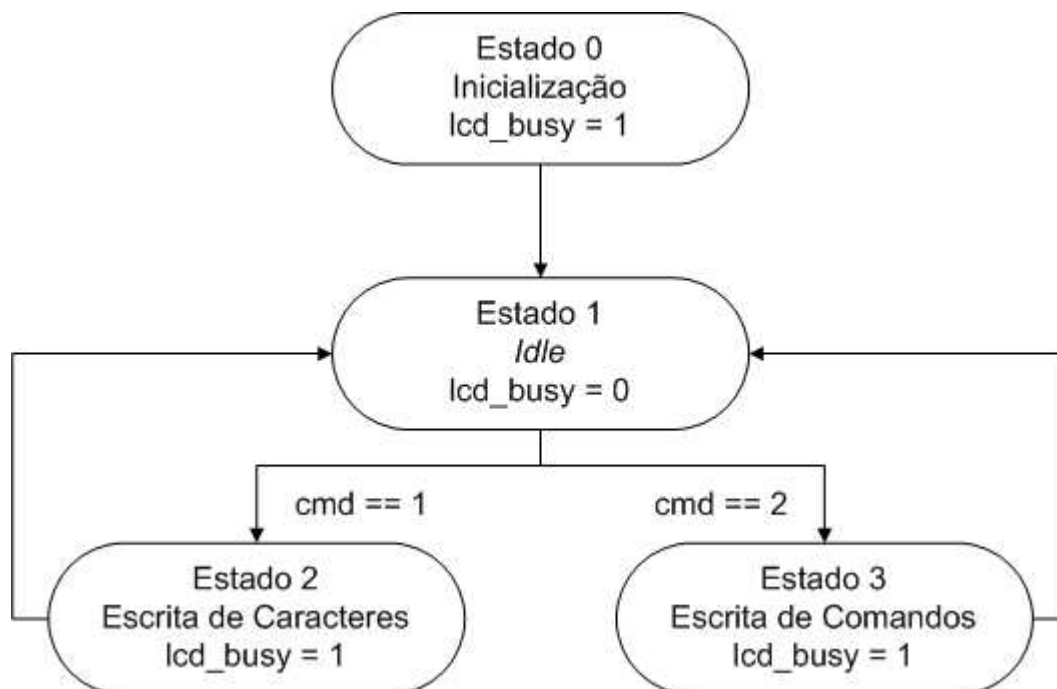


Figura 3.18 - Máquina de estados do módulo "LCD.v".

O registo "lcdcode" é usado para concatenar os seis sinais de controlo usados na escrita de instruções ("LCD_RS", "LCD_RW", "DB7", "DB6", "DB5" e "DB4"), tornando mais simples a atribuição dos respectivos valores aos sinais.

O registo "aux" é utilizado no armazenamento temporário da instrução a ser processada. Sempre que o LCD está no estado "idle" e recebe uma nova instrução, esta é guardada neste registo. Assim, se os sinais de entrada do LCD forem alterados durante o processamento da instrução actual, esta não será afectada.

O registo "count" desempenha três funções: *clock divider*, controlo de várias máquinas de estado e controlo do sinal "LCD_E". Este registo está a ser constantemente incrementado, nos estados '0', '2' e '3' da máquina de estados principal, a cada subida de relógio. Os bits 18 e 19 deste registo são utilizados no controlo de "LCD_E" proporcionando uma *clock rate* de aproximadamente 50Hz ($clock\ rate = clock / 2^{20}$). O conjunto de bits compreendido pelos bits

20 a 25 deste registo funciona como um divisor de relógio, pois só são incrementados uma vez em cada $2^{20} = 1048575$ ciclos de relógio, aproximadamente 21 milissegundos. Funciona também como controlador das máquinas de estado secundárias presentes nos estados '0', '2' e '3' da máquina de estados principal. Estas máquinas de estados secundárias são responsáveis pela actualização dos sinais destinados ao LCD conforme a instrução pretendida. Como este conjunto de bits só é incrementado com um período de aproximadamente 21 milissegundos, todos os intervalos de tempo que é necessário respeitar, demonstrados nas figuras 3.9 e 3.10 deste capítulo, são cumpridos.

3.3.3 - PS/2

Embora estes módulos que implementam o protocolo PS/2 não sejam os módulos presentes no topo da hierarquia de qualquer interface, faz todo o sentido a sua descrição separada devido à utilização destes por dois periféricos: o teclado e o rato.

Como foi visto anteriormente, o protocolo PS/2 implementa uma comunicação bidireccional entre o anfitrião e o periférico. Esta comunicação deve ser arbitrada, de forma a evitar conflitos entre transmissão e recepção e está estabelecido que a transmissão do anfitrião para o periférico é prioritária em relação à recepção.

Tendo em conta estes dados a estrutura a desenvolver é composta por três módulos em dois níveis hierárquicos: um de transmissão (PS2_tx.v), um de recepção (PS2_rx.v) e um de controlo (PS2.v). Os módulos de transmissão e recepção encontram-se no nível hierárquico inferior e são controlados pelo módulo de controlo que é simultaneamente o módulo de topo deste protocolo.

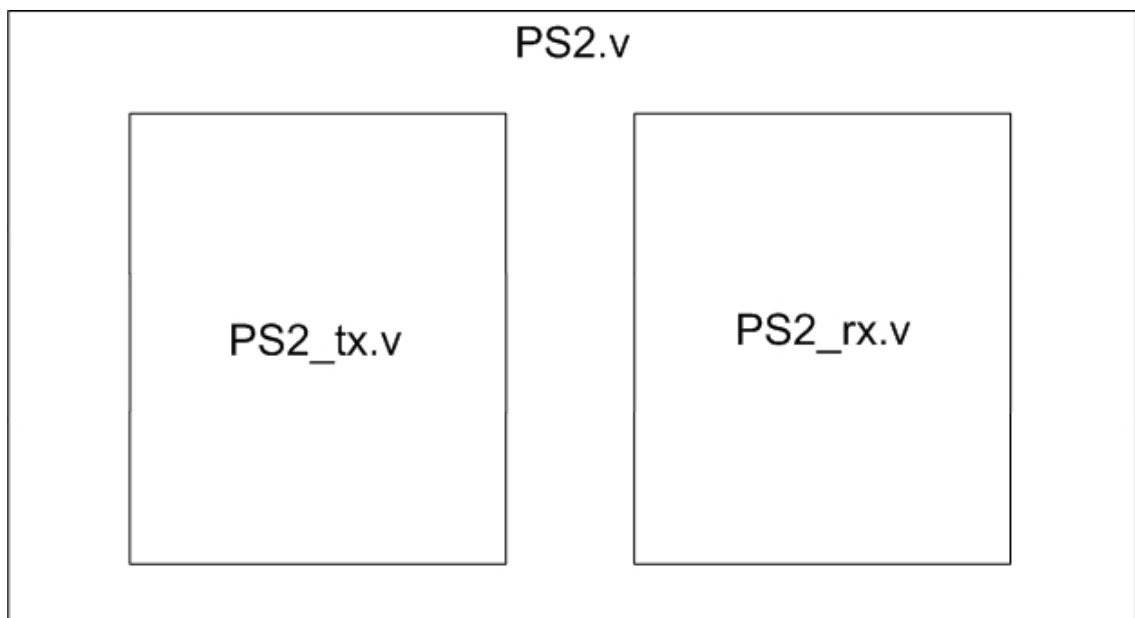


Figura 3.19 - Hierarquia dos módulos que implementam o protocolo PS/2.

O módulo de transmissão deverá receber o byte a transmitir, calcular o bit de paridade adequado e transmitir essa informação para o periférico, transmitindo também as condições de início e fim de transmissão. Para além dos sinais de relógio, *reset* e das linhas de comunicação, deverá ter um sinal de entrada que indica quando deverá transmitir os dados e

dois sinais de saída, um indicando se não está a transmitir e outro para indicar o fim da transmissão.

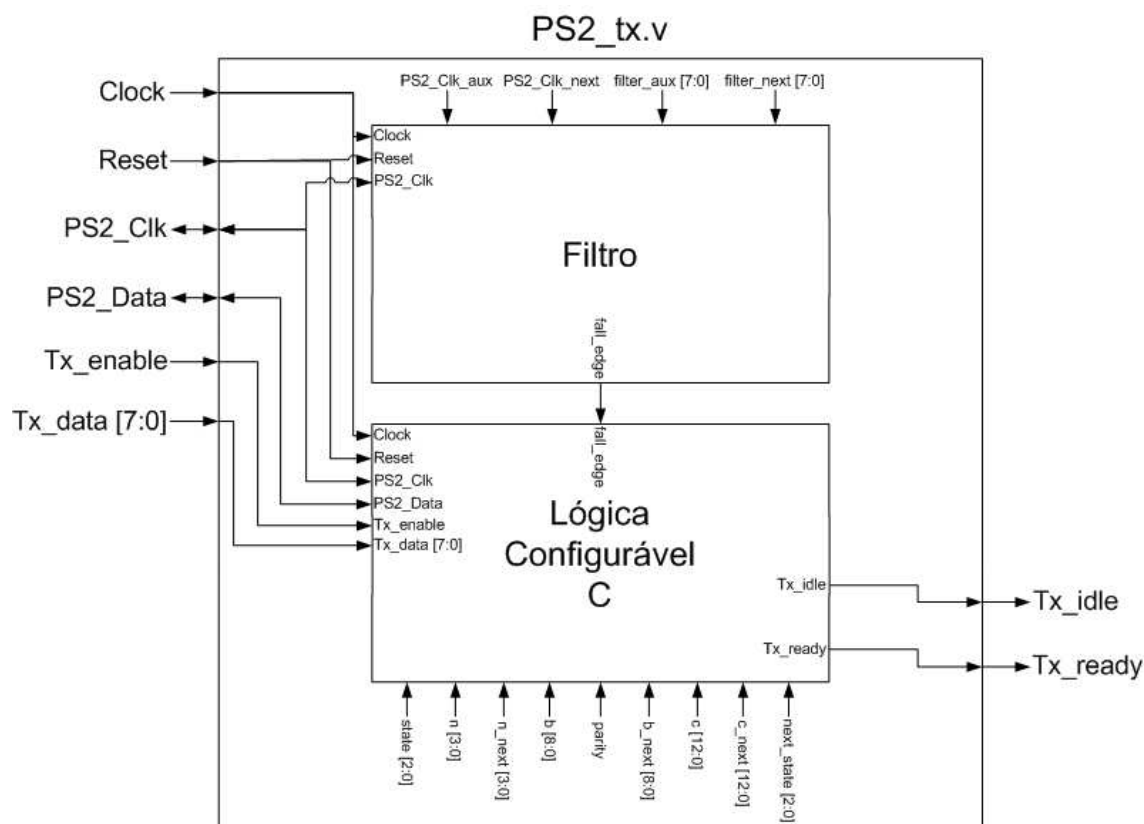


Figura 3.20 - Módulo “PS2_tx.v”.

A entrada “Tx_enable” indica ao módulo quando se deve enviar o byte presente em “Tx_data”.

O sinal bidireccional (*inout*) “PS2_Clk” representa a linha de relógio do barramento PS/2, e o sinal, também bidireccional, “PS2_Data” representa a linha de dados do mesmo barramento. Como se trata de um barramento em que os sinais se encontram em colector comum e são bidireccionais, estes sinais são controlados através de um buffer de três estados. Estes sinais devem ser ligados aos respectivos recursos através da adição das seguintes linhas ao ficheiro “.ucf” do projecto, especificando a sua localização, o tipo de ligações (fios) a utilizar, a intensidade de corrente, e a velocidade de transição de sinal:

```
NET "PS2_Clk" LOC = "G14" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = SLOW;
```

```
NET "PS2_Data" LOC = "G13" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = SLOW;
```

O sinal de saída “Tx_idle” indica, colocando este sinal a ‘1’, quando é que o módulo não se encontra a transmitir, para posteriormente fazer a arbitragem das comunicações no módulo superior de controlo.

O sinal “Tx_ready” indica ao módulo de controlo quando terminou a transmissão de dados.

O fio “parity” calcula automaticamente o bit de paridade dos dados a transmitir.

Os registos internos “PS2_Clk_aux” e “filter_aux” e os fios “PS2_Clk_next”, “filter_next” e “fall-edge” são utilizados para aplicar um filtro ao sinal “PS2_Clk”, que elimina os efeitos negativos de uma transição de sinal lenta ou de *bouncing* do sinal.

O fio “filter_next” armazena os oito últimos estados do sinal “PS2_Clk”, sendo estes dados guardados de forma síncrona com o relógio no registo “filter_aux”.

O fio “PS2_Clk_next” armazena o próximo estado do registo síncrono, “PS2_Clk_aux”, calculado através dos dados presentes em “filter_aux”. Quando “PS2_Clk_next” é igual a ‘0’ e “PS2_Clk_aux” é igual a ‘1’, o que indica uma descida do relógio do barramento de dados, o fio “fall-edge” assume o valor lógico ‘1’. Este fio é posteriormente utilizado na lógica interna do módulo destinada à transmissão de dados.

No interior do módulo existe uma máquina de estados destinada à transmissão de dados. Esta máquina possui cinco estados, definidos em “state” e “next_state” e funciona assincronamente. O estado 0 é o estado de *idle* passando para o estado 1 quando “Tx_enable” é activo, guardando em “b” e “b_next” os dados a transmitir, incluindo o bit de paridade. No estado 1 o sinal “PS2_Clk” é mantido a ‘0’ durante 8191 ciclos de relógio ($2^{13} - 1$), sendo este atraso contabilizado pelos registos “c” e “c_next”, de forma a transmitir uma condição de início de transmissão, passando de seguida ao estado 2. Nesse estado a linha “PS2_Clk” é libertada, assumindo controlo da linha “PS2_data” e colocando-a a ‘0’ até o primeiro “fall_edge”, passando então para o estado 3. O estado 3 é o estado de escrita dos bits, onde cada um dos dez bits (oito bits de dados, um bit de paridade e um bit de fim de transmissão) é escrito a cada “fall_edge”, sendo guardado em “n” e “n_next” o número de bits ainda por enviar. Quando todos os bits tiverem sido enviados passa-se ao estado 4, que existe para colocar “Tx_ready” a ‘1’ e para utilizar mais um ciclo de relógio de forma a actualizar os registos (síncronos com o relógio) “b” e “n”, a partir de “b_next” e “n_next”, regressando ao estado 0. “Tx_idle” tem o valor pré-definido de ‘0’, apenas alterado quando se está no estado 0 da máquina de estados, assumindo o valor lógico ‘1’.

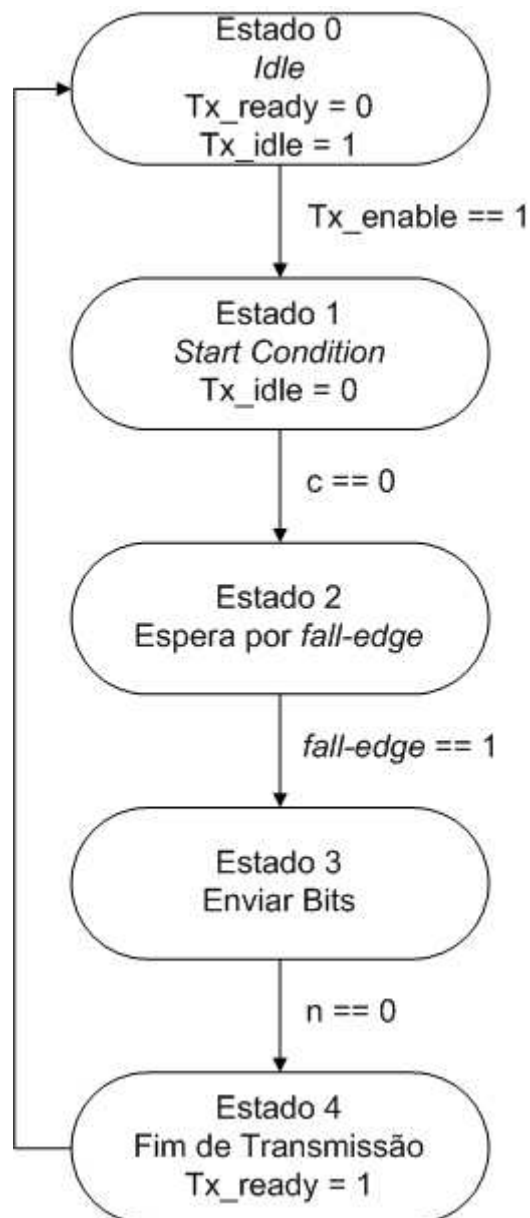


Figura 3.21 - Máquina de Estados do módulo “PS2_tx.v”.

O módulo de recepção deverá guardar os dados recebidos do periférico sempre que a recepção esteja activa. Para além dos sinais de relógio, *reset* e das linhas de comunicação, deverá ter um sinal de entrada para activar e desactivar a transmissão, de forma a arbitrar a comunicação. Deverá ter ainda dois sinais de saída, um indicando a recepção de novos dados, e outro contendo os dados recebidos. Como neste módulo os sinais “PS2_Clk” e “PS2_Data” são apenas utilizados para recepção, estes podem ser considerados sinais unidireccionais de entrada.

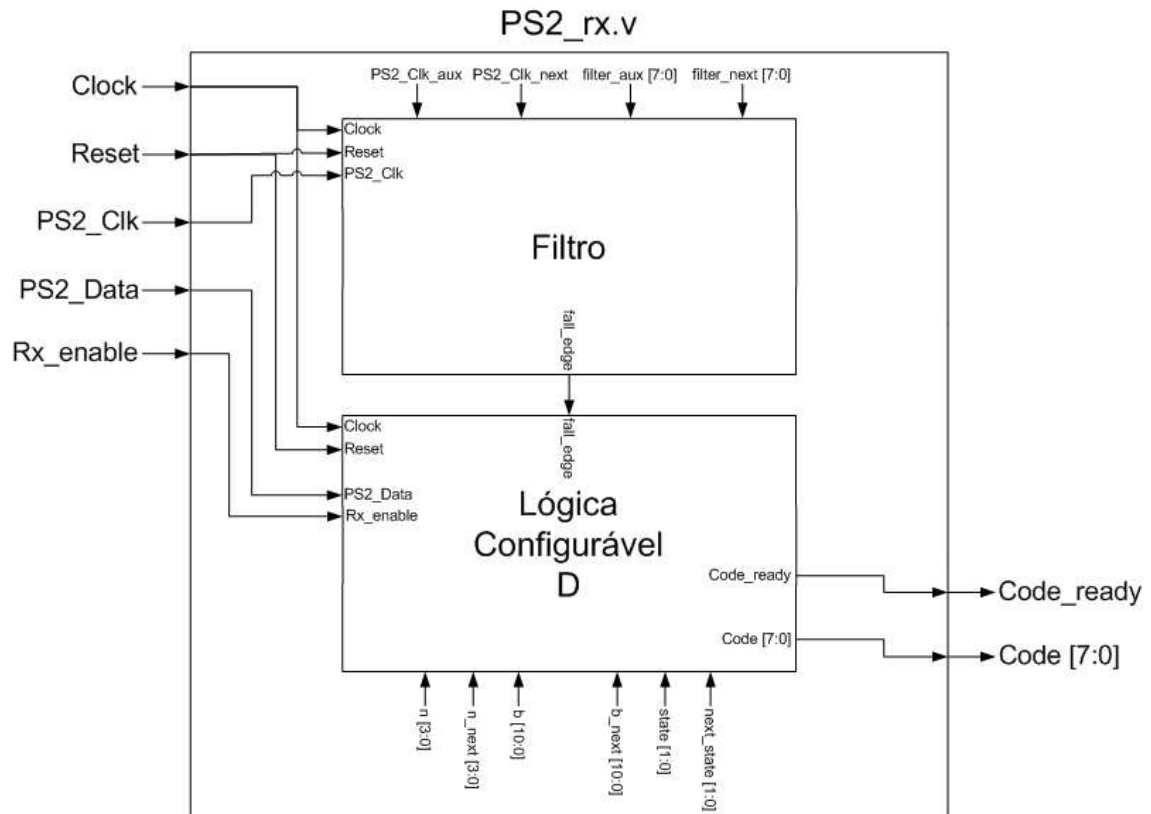


Figura 3.22 - Módulo “PS2_rx.v”.

A entrada “Rx_enable” indica ao módulo quando se deve activar a recepção de dados.

O sinal de entrada “PS2_Clk” representa a linha de relógio do barramento PS/2, e o sinal “PS2_Data” representa a linha de dados do mesmo barramento. Estes sinais devem ser ligados aos respectivos recursos através da adição das seguintes linhas ao ficheiro “.ucf” do projecto, especificando a sua localização, o tipo de ligações (fios) a utilizar, a intensidade de corrente, e a velocidade de transição de sinal:

```
NET "PS2_Clk" LOC = "G14" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = SLOW;
```

```
NET "PS2_Data" LOC = "G13" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = SLOW;
```

O sinal de saída “Code_ready” indica, colocando este sinal a ‘1’, quando é que foi terminada uma recepção, sendo o byte recebido colocado em “Code”.

Neste módulo, tal como no descrito anteriormente, existe um filtro para o sinal “PS2_Clk”. Este filtro é igual ao descrito no módulo precedente, inclusivamente nos registos e fios.

No interior deste módulo existe ainda uma máquina de estados, assíncrona e controlada pelos registos “state” e “next_state”, e possui três estados. O estado 0 é um estado de *idle* e a cada impulso de “fall_edge”, se “Rx_enable” estiver activo, verifica se foi transmitida uma condição de início. Se houver passa para o estado 1 que recebe os oito bits de dados, um bit de paridade e o bit de fim de transmissão. Estes dados são guardados nos registos “b” e “b_next” e o número de bits que ainda falta receber é guardado em “n” e “n_next”. Quando forem recebidos todos os bits, passa ao estado 2, que indica ao módulo superior o fim da recepção e os respectivos dados, regressando ao estado de *idle*.

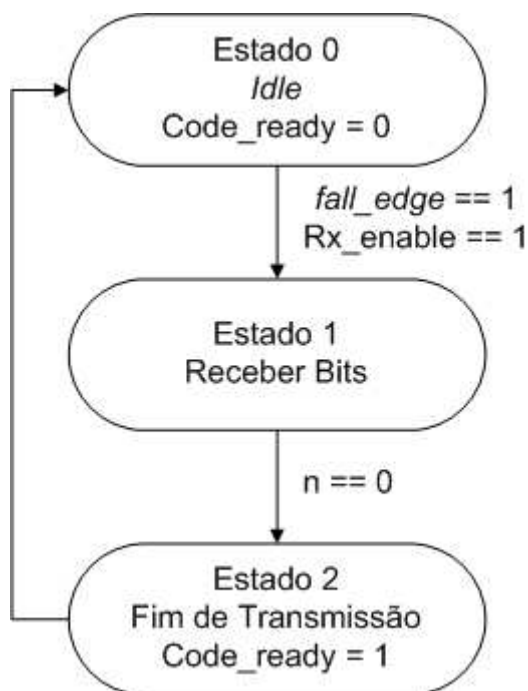


Figura 3.23 - Máquina de estados do módulo “PS2_rx.v”.

O módulo de controlo do protocolo PS/2 deverá conter os dois módulos anteriores e efectuar a gestão das comunicações de forma a evitar conflitos.

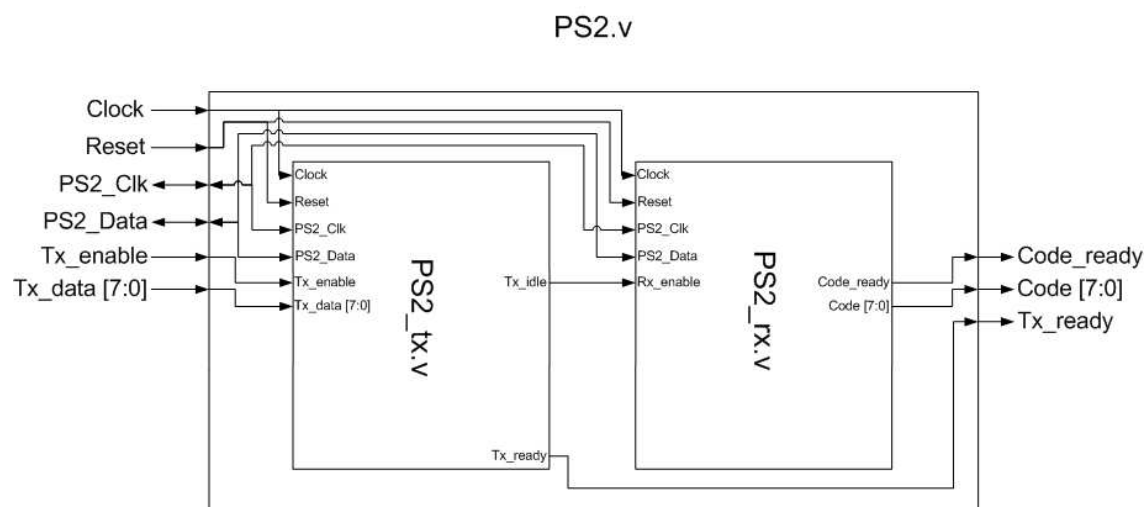


Figura 3.24 - Módulo “PS2.v”.

A entrada “Tx_enable” indica ao módulo quando se deve enviar o byte presente em “Tx_data” e, como a transmissão é prioritária, qualquer recepção deve ser interrompida/proibida.

O sinal bidireccional (*inout*) “PS2_Clk” representa a linha de relógio do barramento PS/2, e o sinal, também bidireccional, “PS2_Data” representa a linha de dados do mesmo barramento. Como se trata de um barramento em que os sinais se encontram em colector comum e são bidireccionais, estes são controlados através de um buffer de três estados. Estes sinais devem ser ligados aos respectivos recursos e aos dois módulos, através da adição das

seguintes linhas ao ficheiro “.ucf” do projecto, especificando a sua localização, o tipo de ligações (fios) a utilizar, a intensidade de corrente e a velocidade de transição de sinal:

```
NET "PS2_Clk" LOC = "G14" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = SLOW;
```

```
NET "PS2_Data" LOC = "G13" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = SLOW;
```

O sinal de saída “Tx_idle” indica, colocando este sinal a ‘1’, quando é que o módulo de transmissão não está ocupado, ligando directamente ao sinal “Rx_enable” do módulo de recepção. Assim, só serão recebidos dados do periférico quando não se estiver a transmitir.

O sinal “Tx_ready” indica quando terminou a transmissão dos dados.

O sinal de saída “Code_ready” indica, colocando este sinal a ‘1’, quando é que foi terminada uma recepção, sendo o byte recebido colocado em “Code”.

Com a junção destes três módulos já é possível receber e enviar dados para o periférico, independentemente do seu tipo, tornando estes módulos o mais flexíveis possível, tal como eram pretendidos.

3.3.4 - Teclado

Para efectuar a interface entre um teclado e a FPGA, para além dos módulos que implementam o protocolo PS/2, é necessário a criação de um módulo que converta os dados provenientes do teclado, designados *make/break codes*, no respectivo código ASCII sempre que aplicável.

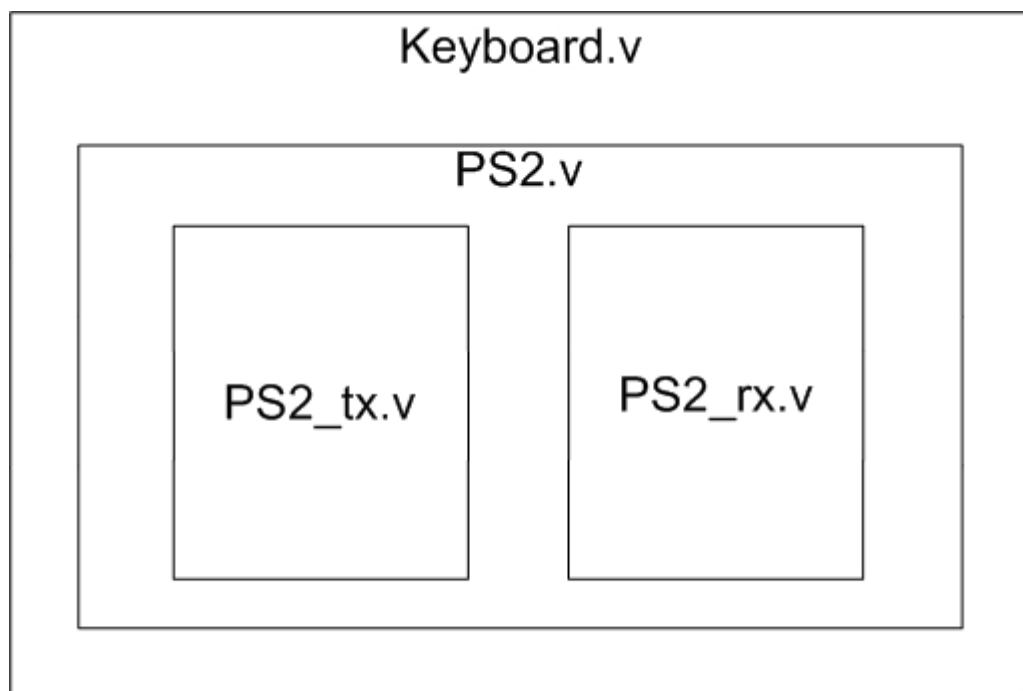


Figura 3.25 - Hierarquia dos módulos que controlam o teclado PS/2.

O *make code* é transmitido quando uma tecla é premida, e o *break code* quando ela é libertada. Estes códigos podem ser compostos por dois ou mesmo três bytes, o que torna necessária uma máquina de estados mais complexa para os processar. Às teclas cujos caracteres não possuam código ASCII, é-lhes atribuído um valor que não represente o código ASCII de nenhum carácter. Assim sendo, este módulo, para além dos sinais de entrada de

relógio, reset e linhas de dados, precisa de duas saídas, uma para indicar o processamento de um novo código e a outra contendo esses mesmos dados.

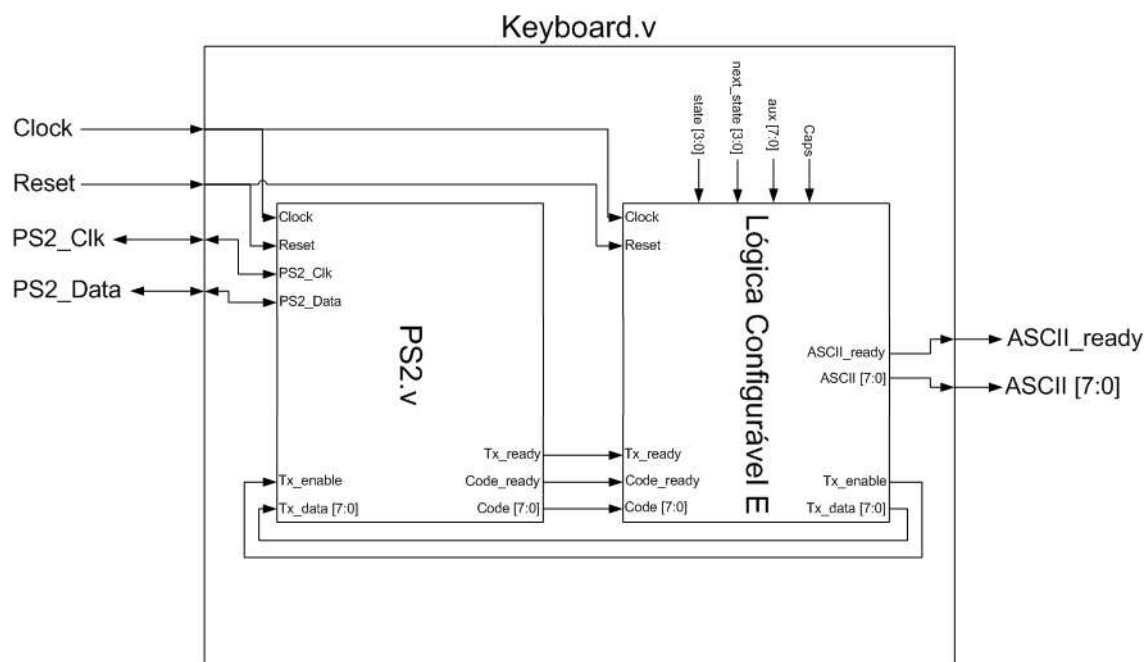


Figura 3.26 - Módulo “Keyboard.v”.

O sinal bidireccional (*inout*) “PS2_Clk” representa a linha de relógio do barramento PS/2, e o sinal, também bidireccional, “PS2_Data” representa a linha de dados do mesmo barramento. Como se trata de um barramento em que os sinais se encontram em colector comum e são bidireccionais, estes sinais são controlados através de um buffer de 3 estados. Estes sinais devem ser ligados aos respectivos recursos, através da adição das seguintes linhas ao ficheiro “.ucf” do projecto, especificando a sua localização, o tipo de ligações (fios) a utilizar, a intensidade de corrente e a velocidade de transição de sinal:

```
NET "PS2_Clk" LOC = "G14" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = SLOW;
```

```
NET "PS2_Data" LOC = "G13" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = SLOW;
```

O sinal de saída “ASCII_ready” indica, colocando este sinal a ‘1’, quando está disponível um novo código, sendo o byte colocado em “ASCII”.

No interior deste módulo, todos os sinais de entrada e saída do módulo “PS2.v” são controlados e processados pela lógica implementada no módulo.

Sempre que é recebido um novo código proveniente do teclado, esse código é convertido no respectivo código ASCII (quando aplicável) do carácter e guardado no registo interno “aux”.

Na máquina de estados do módulo, composta por doze estados e controlada por “state” e “next_state”, é processada toda a restante informação a partir do registo “aux”. No estado 0 a FSM encontra-se à espera de receber o primeiro byte e quando este é recebido passa ao estado 1. Neste estado o byte recebido é processado. Se “aux” for igual a F0h indica que é um *break code* que pode conter 2 ou 3 bytes. Quando “aux” tem o valor de E0h pode-se tratar de um *make* ou *break code* com 2 ou 3 bytes. Nos dois casos anteriores passa-se para o estado 2. Se “aux” for igual a 00h indica que o *CapsLock* foi premido e deve passar ao estado 6 pedindo ao barramento para enviar o byte EDh ao teclado, iniciando a sequência de 2 bytes a

transmitir para acender o LED relativo ao *CapsLock*. Nos restantes valores de “aux”, que indicam o código ASCII de um carácter cujo *make code* é constituído por apenas um byte, o valor de “aux” é copiado para “ASCII” e “ASCII_ready” é colocado a ‘1’, indicando um novo código pronto e regressando ao estado 0.

No estado 2 a máquina de estados espera a recepção do segundo byte do *make/break code* e quando este é recebido, passa para o estado 3. Neste estado se o valor de “aux” for igual a F0h indica que se trata de um *break code* composto por 3 bytes passando ao estado 4. Os restantes valores de “aux” indicam teclas cuja função é representada por um *make code* de 2 bytes, sendo “ASCII_ready” colocado a ‘1’ e o respectivo código da tecla em “ASCII”. No estado 4 é esperado o terceiro byte do *break code* e quando este é recebido passa-se ao estado 5 onde, dependendo do valor de “aux”, é processado o *break code*.

No estado 6 espera-se o fim da transmissão para o teclado e quando esta se completa, passa-se ao estado 7. Neste estado espera-se a recepção de um byte de *acknowledge* do teclado. Quando é recebido um byte passa-se ao estado 8 onde, através do registo “aux” se verifica se é o byte de *acknowledge*. Se for passa-se ao estado 9, pedindo também a transmissão de um byte composto por cinco ‘0’ e o estado dos três LEDs (*Caps*, *Num* e *Scroll Lock*). No estado 9 espera-se o fim da transmissão para o teclado e quando esta se completa, passa-se ao estado 10, onde se espera a recepção de um byte de *acknowledge* do teclado. Quando é recebido um byte passa-se ao estado 11 onde, através do registo “aux” se verifica se é o byte de *acknowledge*, regressa ao estado 0.

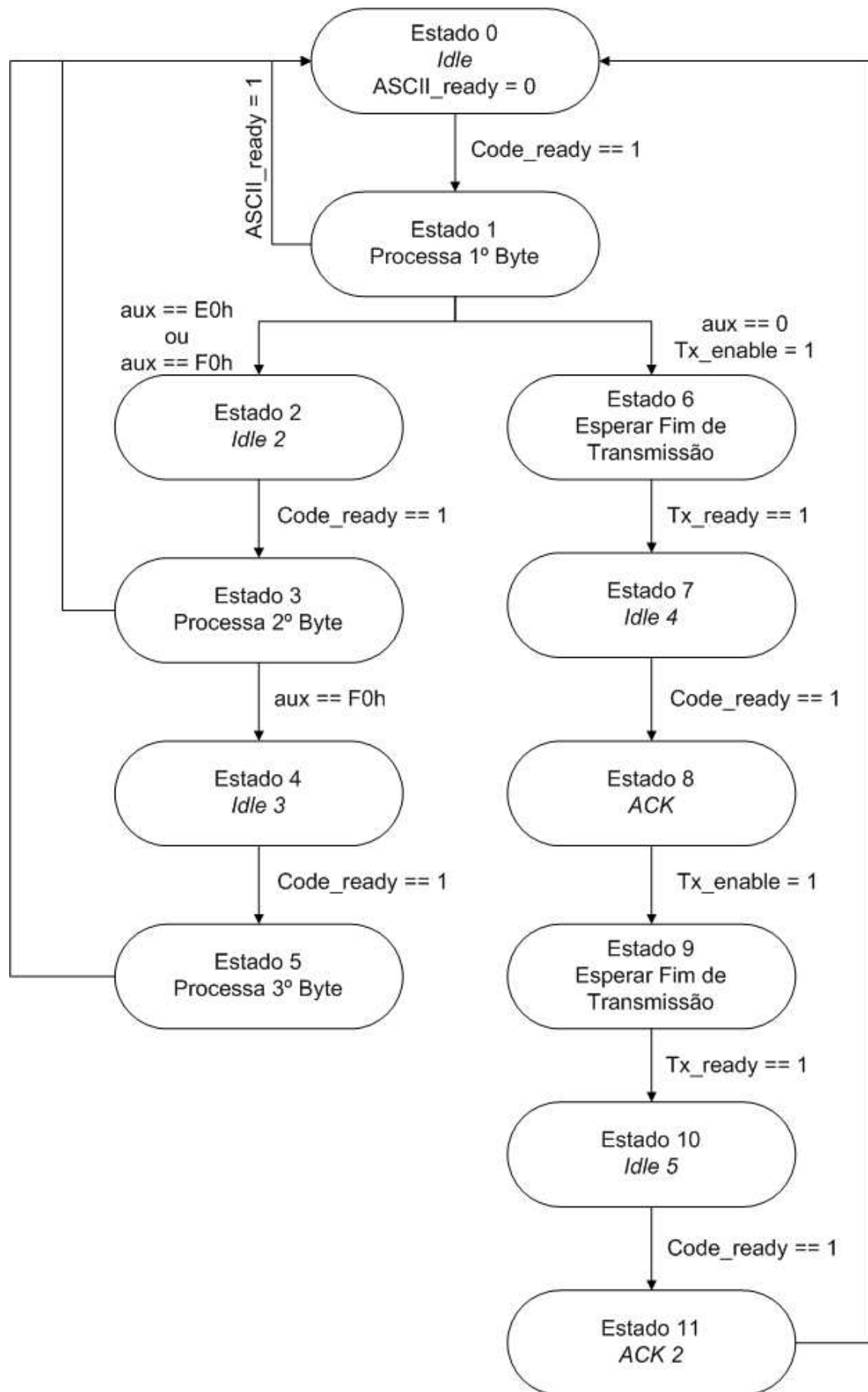


Figura 3.27 - Máquina de estados do módulo "Keyboard.v".

3.3.5 - Rato

Os ratos são dispositivos periféricos destinados a detectar movimento em duas direcções sobre uma superfície, possuindo ainda botões que permitem uma interactividade mais profunda com o sistema que integra. Um rato PS/2 transmite os dados sempre que é detectado movimento, ou uma pressão nos botões. O rato mede a distância do movimento e transmite-a em oito bits em complemento para dois. Transmite também um nono bit indicando o sentido do movimento. Sendo que o rato detecta movimento em duas direcções, são usados dezoito bits só para descrever os movimentos do rato. Para além disso, existem três bits destinados a descrever os estados dos botões. Assim sendo, os dados são transmitidos sobre a forma de 3 tramas do protocolo PS/2, cada uma contendo um bit de início, oito bits de dados, um bit de paridade e um bit de fim. O primeiro byte contém informação relativamente aos botões e a direcção do movimento nos dois eixos (x e y). O segundo byte contém os oito restantes bits da distância de movimento para o eixo x. No terceiro byte estão presentes os restantes oito bits da distância de movimento do eixo y.

Sendo assim, é necessário que o módulo processe os bytes recebidos e os organize de uma forma mais prática em quatro saídas: uma com três bits para os botões, e duas com nove bits para cada um dos eixos e uma saída de um bit que indique que estão prontos novos dados.

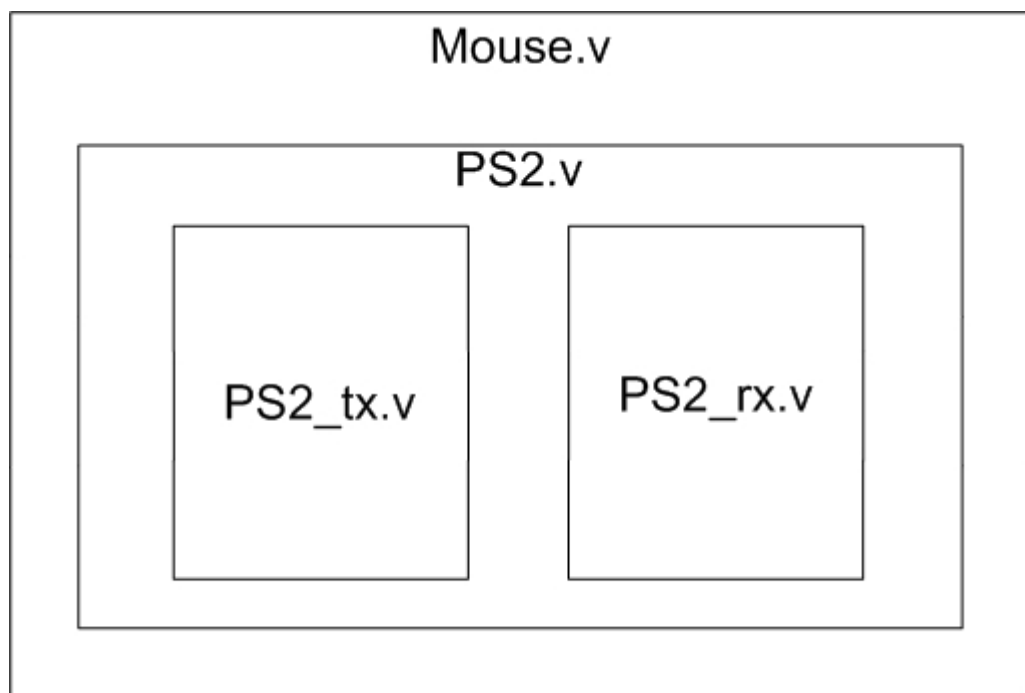


Figura 3.28 - Hierarquia dos módulos que controlam o rato PS/2.

Para além de organizar os dados recebidos, o módulo tem que realizar numa fase inicial a sequência de inicialização do rato PS/2. Estes dispositivos não são *hot-pluggable* e necessitam desta sequência sempre que são ligados. Esta sequência é muito simples e consiste em enviar o byte F4h para o rato e esperar que este envie um byte de *acknowledge* (FEh). Após este procedimento o rato está em *stream mode*, que é o modo de funcionamento comum destes dispositivos. Antes desta sequência de inicialização, e imediatamente após a alimentação ser

ligada o rato efectua um teste diagnóstico e envia o byte AAh para o controlador. No entanto, esta fase pode ser ignorada pois a FPGA ainda não se encontra configurada nesta fase.

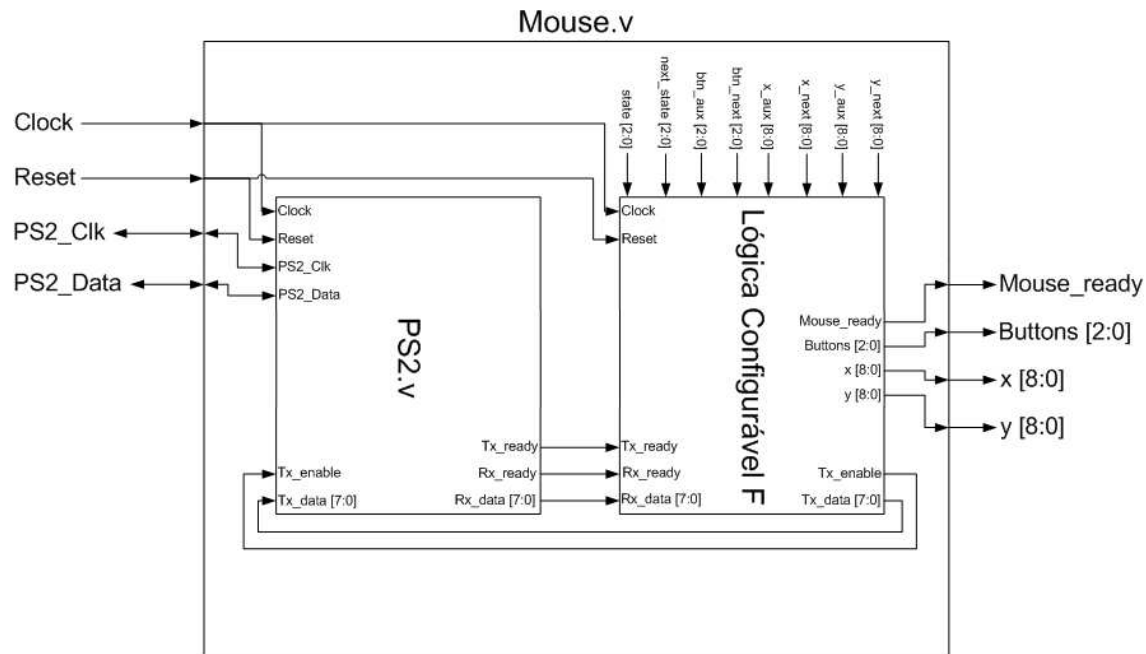


Figura 3.29 - Módulo "Mouse.v".

O sinal bidireccional (*inout*) "PS2_Clk" representa a linha de relógio do barramento PS/2 e o sinal, também bidireccional, "PS2_Data" representa a linha de dados do mesmo barramento. Como se trata de um barramento em que os sinais se encontram em colector comum e são bidireccionais, estes sinais são controlados através de um buffer de três estados. Estes sinais devem de ser ligados aos respectivos recursos, através da adição das seguintes linhas ao ficheiro ".ucf" do projecto, especificando a sua localização, o tipo de ligações (fios) a utilizar, a intensidade de corrente e a velocidade de transição de sinal:

```
NET "PS2_Clk" LOC = "G14" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = SLOW;
```

```
NET "PS2_Data" LOC = "G13" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = SLOW;
```

O sinal de saída "Mouse_ready" indica que estão presentes novos dados nos restantes sinais de saída. O sinal "Buttons" representa o estado dos três botões standard do rato, sendo o bit mais significativo representante do botão central, o menos significativo representante do botão esquerdo e o bit remanescente simboliza o botão direito. As saídas "x" e "y" representam o movimento do rato no respectivo eixo, sendo que o bit mais significativo indica o sentido do movimento e os restantes a amplitude do deslocamento. Quando o bit mais significativo assume o valor lógico '1' significa que o movimento foi realizado ou para a esquerda (no caso da saída x) ou para baixo (no caso da saída y). Naturalmente que, caso o valor lógico deste bit for '0', o movimento ocorreu nos sentidos inversos aos anteriores.

No interior deste módulo existe uma máquina de estados assíncrona, controlada pelos registos "state" e "next_state", com sete estados definidos.

Os primeiros três estados destinam-se à inicialização do periférico. No estado 0 é dada a instrução aos módulos que controlam o barramento PS/2 para enviar o byte F4h, passando-se de seguida para o estado 1, onde se espera a conclusão da transmissão de dados para o periférico. Quando termina o envio dos dados passa-se ao estado 2 onde se espera a recepção

do byte de *acknowledge* (FEh). Após a recepção deste byte, o rato entra no modo *stream* e passa-se ao estado 3. Este é um estado de *idle* onde se espera a recepção do primeiro byte. Quando este é recebido, guarda-se nos bits mais significativos dos registos internos “x_next” e “y_next” os dados relativos ao sentido do movimento e no registo “btn_next” os dados referentes ao estado dos botões passando-se ao estado 4. Estes três registos são assíncronos com o relógio, sendo copiados para os registos internos (síncronos) “x_aux”, “y_aux” e “btn_aux” a cada subida de relógio. Estes últimos estão directamente ligados aos respectivos fios que representam os sinais de saída do módulo. No estado 4 espera-se a recepção do segundo byte, e quando este é recebido copia-se os dados destes para os restantes bits de “x_next” saltando-se para o estado 5 de seguida. Neste estado espera-se a recepção do terceiro e último byte, cujos dados são copiados para os restantes bits de “y_next”. O estado seguinte (6) serve apenas para que se coloque o sinal de saída “Mouse_ready” a ‘1’ e para que os registos se sincronizem, regressando ao estado 3.

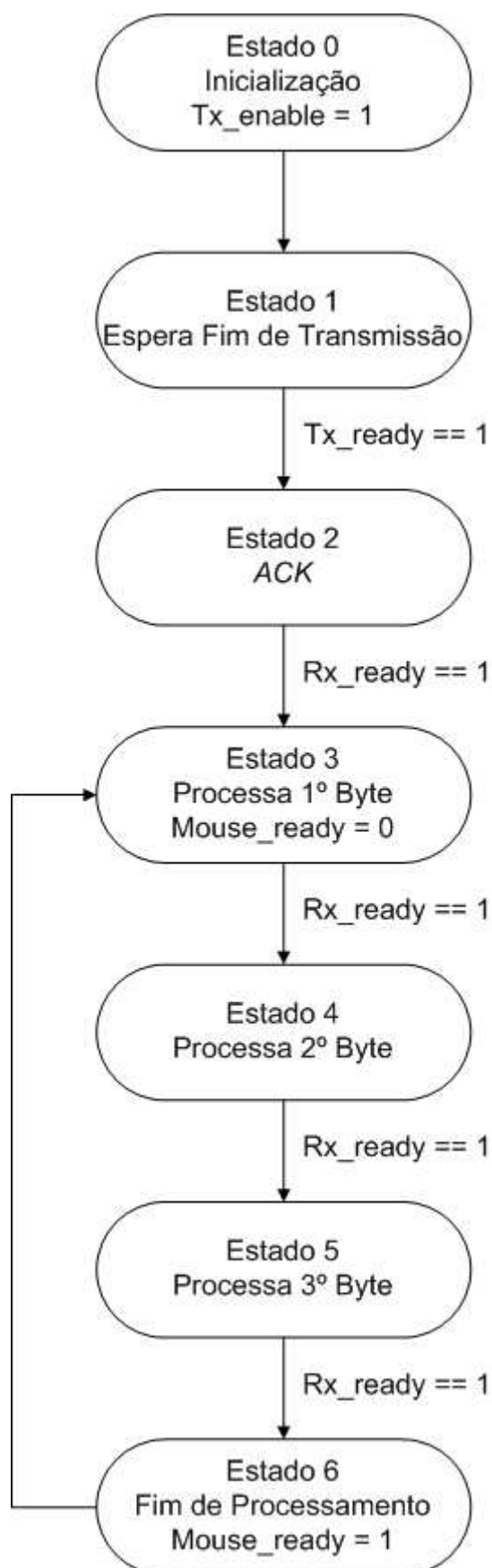


Figura 3.30 - Máquina de estados do módulo "Mouse.v".

3.3.6 - UART

Tal como já foi referido anteriormente o protocolo UART permite a comunicação assíncrona em *full-duplex* entre dois dispositivos. Como complemento ao protocolo de comunicação UART, que apenas define os aspectos relativos à estrutura de dados e de temporização, utiliza-se o protocolo RS-232 que apenas define os aspectos físicos da ligação e transmissão de dados. Como também já foi referido, a transmissão e recepção de dados realizam-se em fios diferentes e ambos sem sinal de relógio associado. Assim sendo a comunicação através destes protocolos foi estruturada em 4 módulos: um de controlo temporal, um de transmissão, um de recepção e um módulo que aglomere os anteriores para que a integração destes dois protocolos seja mais fácil.

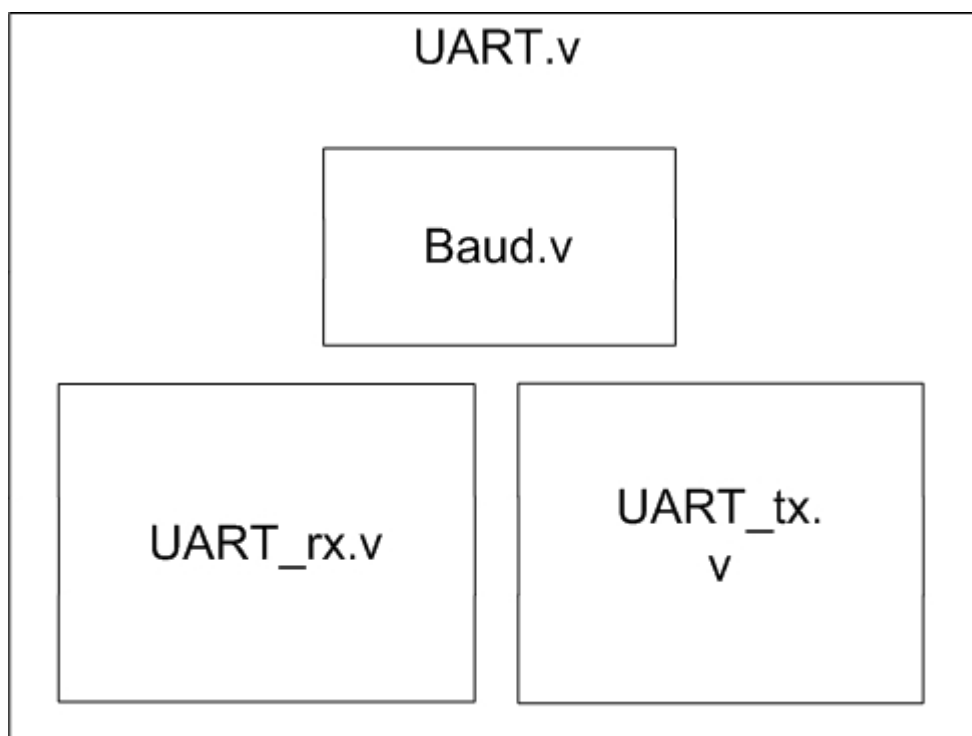


Figura 3.31 - Hierarquia dos módulos que implementam o protocolo UART.

Antes de se poder fazer qualquer transmissão ou recepção torna-se necessária a criação de um módulo que definia as condições temporais de amostragem. Esse módulo deve possuir dois parâmetros temporais já referidos, a *baudrate* e o *oversample*. Através destes dois parâmetros e do sinal de relógio interno, deve gerar um sinal de saída que indique aos módulos de comunicação quando analisar o estado da linha de transmissão ou alterar o seu valor.

O módulo de recepção deve analisar o estado da linha de recepção sempre que o módulo temporal lhe indica e sempre que for iniciada a recepção de novos dados estes devem ser guardados e no final da transmissão colocados na saída deste módulo, juntamente com outro sinal que indique que foi terminada a recepção.

O módulo de transmissão deverá receber os dados a transmitir sempre que lhe for dada a ordem de iniciar uma nova transmissão, iniciar a transmissão de dados através da condição de

início, transmitir os dados e finalizar a transmissão sinalizando o seu término através de um sinal de saída.

O módulo de topo apenas aglomera os três módulos anteriores bem como as suas entradas e saídas, tornando a integração destes num projecto bastante mais fácil.

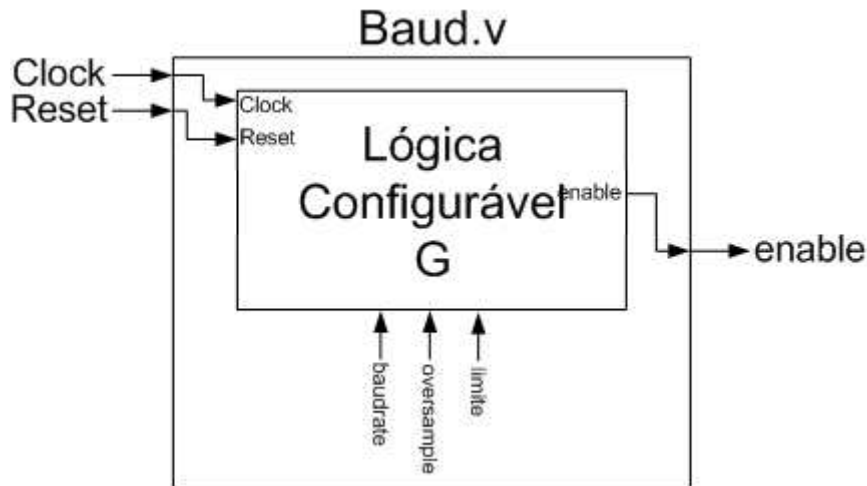


Figura 3.32 - Módulo “Baud.v”.

A saída “enable” deverá ser ligada aos módulos de transmissão e recepção, indicando quando estes devem alterar ou analisar o valor da linha correspondente.

No interior do módulo é calculado no registo “limite” o número de ciclos de relógio entre cada impulso de “enable”. Esse número é calculado através dos parâmetros *oversample*, *baudrate* e também a partir da frequência do relógio interno da plataforma de desenvolvimento (50MHz). Os parâmetros *baudrate* e *oversample* têm os valores pré-definidos de 19200 e 16 respectivamente, podendo ser alterados directamente no módulo ou através da passagem de argumentos para o interior do módulo.

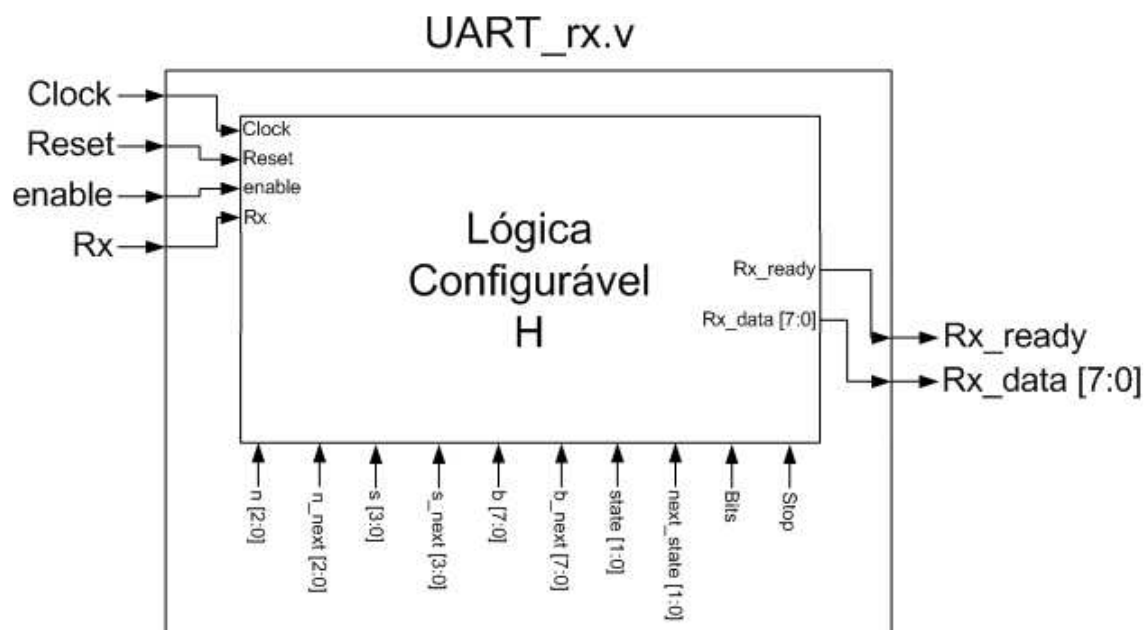


Figura 3.33 - Módulo “UART_rx.v”.

O sinal de entrada “enable” provém do módulo anteriormente referido “Baud.v”.

O sinal de entrada “Rx” representa a linha de dados utilizada para a FPGA receber dados. Este sinal deverá ser ligado aos recursos existentes na plataforma de desenvolvimento através da adição da seguinte linha de código ao ficheiro “.ucf” do projecto, especificando a localização do recurso e o tipo de fios a usar na sua ligação:

```
NET "Rx" LOC = "R7" | IOSTANDARD = LVTTTL ; // Conector Fêmea
```

```
#NET "Rx" LOC = "U8" | IOSTANDARD = LVTTTL ; // Conector Macho
```

No interior deste módulo existem dois parâmetros que especificam o número de bits a receber em cada trama (“Bits”) e o número de “enables” que cada condição de fim dura (“Stop”).

No interior do módulo existe uma máquina de estados assíncrona, controlada por “state” e “next_state”, com 4 estados definidos. Nesta máquina de estados são utilizados os registos, também assíncronos, “s_next”, que conta o número de activações de “enable” durante a condição de início, de fim e em cada bit; “b_next”, que guarda os bits já recebidos, e “n_next”, que contabiliza o número de bits já recebidos. Estes registos são copiados para os registos “s”, “b” e “n” que são síncronos com o relógio.

Na FSM o estado 0 representa um estado de *idle* em que espera que a linha de dados desça para o valor lógico ‘0’, passando para o estado 1. Neste estado é contado o número de activações de “enable” e quando ocorre a oitava, significa que a condição de início está completa, passando-se ao estado 2. Neste estado é efectuada a recepção dos bits de dados, sendo contado para cada bit as dezasseis activações de “enable” necessárias (iguais ao parâmetro *oversample* no módulo “Baud.v”). Quando forem recebidos todos os bits, passa-se ao estado 3 onde se contam o número de activações de “enable” necessárias para completar a condição de fim e quando esta termina, coloca-se a saída “Rx_ready” a ‘1’ sinalizando a recepção de novos dados estando estes presentes em “Rx_data”.

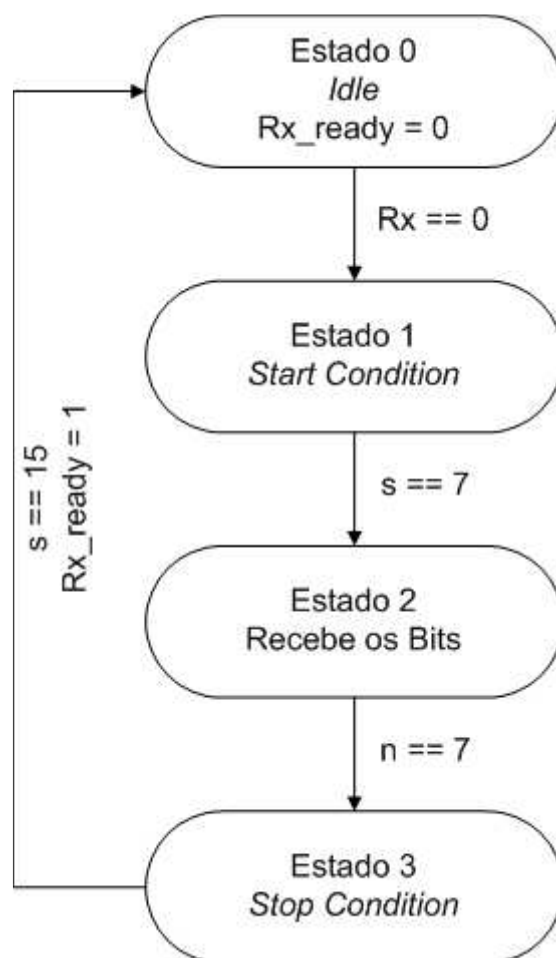


Figura 3.34 - Máquina de estados do módulo “UART_rx.v”.

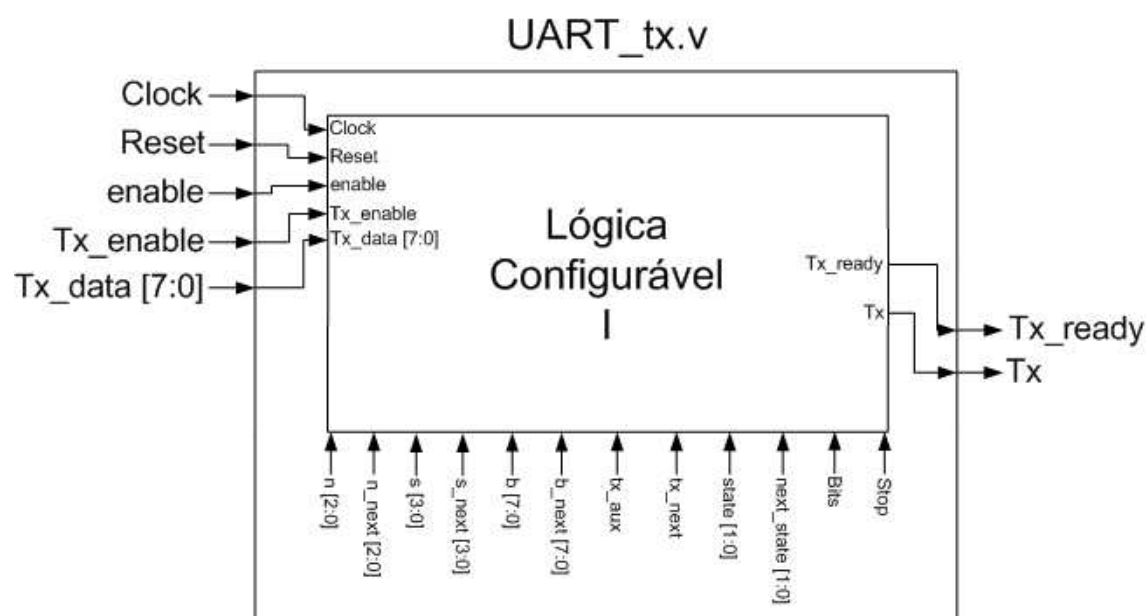


Figura 3.35 - Módulo “UART_tx.v”.

O sinal de entrada “enable” provém do módulo anteriormente referido “Baud.v”.

A entrada “Tx_enable” indica quando se deve iniciar uma nova transmissão, sendo que os dados a transmitir estão presentes na entrada “Tx_data”.

A saída “Tx_ready” indica quando foi concluída uma transmissão e o sinal de saída “Tx” representa a linha de dados utilizada para a FPGA enviar dados. Este sinal deverá ser ligado aos recursos existentes na plataforma de desenvolvimento através da adição da seguinte linha de código ao ficheiro “.ucf” do projecto, especificando a localização do recurso, o tipo de fios a usar na sua ligação, a intensidade da corrente e a velocidade de transição:

```
NET "Tx" LOC = "M14" | IOSTANDARD = LVTTTL | DRIVE = 8 | SLEW = SLOW; // Conector Fêmea
```

```
#NET "Tx" LOC = "M13" | IOSTANDARD = LVTTTL | DRIVE = 8 | SLEW = SLOW; // Conector Macho
```

Tal como no módulo destinado à recepção de dados, existem dois parâmetros internos que especificam o número de bits a receber em cada trama (“Bits”) e o número de “enables” que cada condição de fim dura (“Stop”).

A máquina de estados presente neste módulo é assíncrona, controlada pelos registos “state” e “next_state” e possui quatro estados definidos. Nesta máquina de estados são utilizados os registos, também assíncronos, “s_next”, que conta o número de activações de “enable” durante a condição de início, de fim e em cada bit, “b_next”, que guarda os bits a enviar, “n_next”, que contabiliza o número de bits já enviados e “tx_next” que guarda o bit a ser transmitido. Estes registos são copiados para os registos “s”, “b”, “n” e “tx_aux” que são síncronos com o relógio.

Na FSM o estado 0 representa um estado de *idle* em que se espera que seja dada a ordem de iniciar uma nova transmissão, passando para o estado 1. Neste estado é escrito um ‘0’ na linha de dados contando-se dezasseis activações de “enable” passando-se de seguida ao estado 2. Neste estado é efectuada a transmissão dos bits de dados sendo contado para cada bit as dezasseis activações de “enable” necessárias (iguais ao parâmetro *oversample* no módulo “Baud.v”). Quando forem recebidos todos os bits passa-se ao estado 3 onde se coloca a linha de dados a ‘1’ e conta-se o número de activações de “enable” necessárias para completar a condição de fim e quando esta termina, coloca-se a saída “Tx_ready” a ‘1’ sinalizando o fim da transmissão de dados.

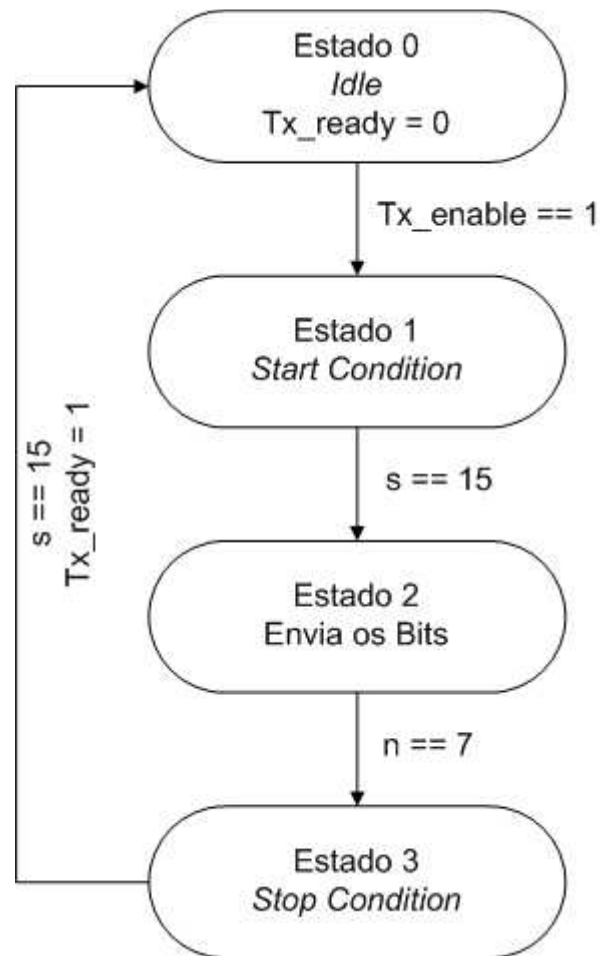


Figura 3.36 - Máquina de estados do módulo “UART_tx.v”.

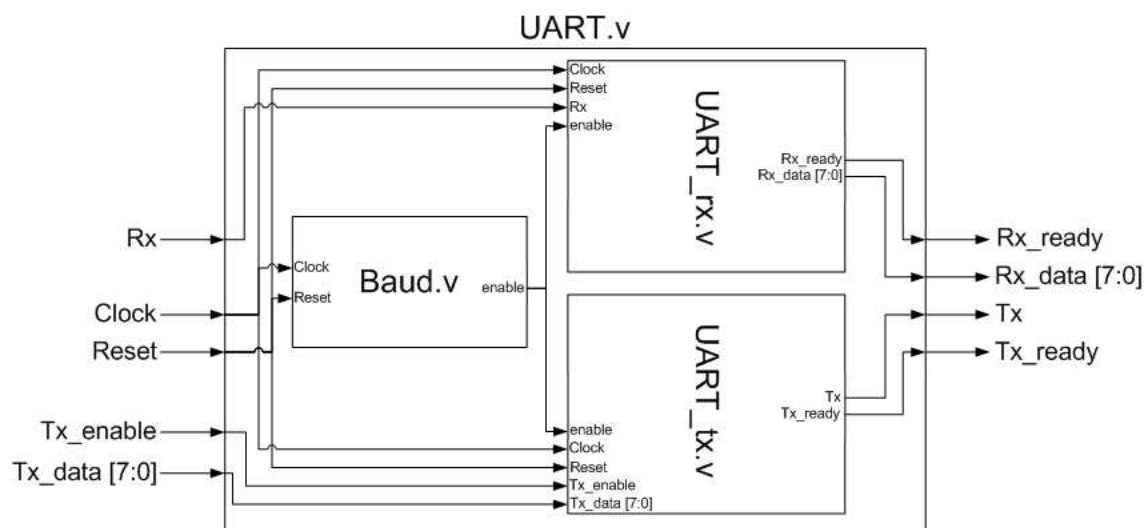


Figura 3.37 - Módulo “UART.v”.

Tal como já foi referido, e pode-se constatar a partir da figura anterior, o módulo “UART.v” é utilizado apenas para aglomerar os três módulos descritos anteriormente, não havendo qualquer tipo de processamento de dados ou lógica configurável associada.

A utilização deste módulo permite ao projectista o controlo total da comunicação série utilizando apenas cinco sinais, para além do sinal de relógio, reset e linhas de dados.

Como a plataforma de desenvolvimento possui duas portas de comunicação série. É possível reutilizar no mesmo projecto estes módulos, alterando apenas o nome das linhas de dados e dos sinais de controlo, para não haver conflitos.

3.3.7 - VGA (Modo Gráfico)

Para se poder utilizar um monitor no modo VGA seria necessário criar um módulo que realizasse todas as operações de sincronismo para que a informação fosse correctamente mostrada no monitor numa resolução de 640 por 400 pixéis, utilizando uma *pixel rate* de 25MHz. Ao adicionar a esse módulo a funcionalidade de indicar a coordenada do pixel actualmente a ser processado e indicando se este é um pixel visível ou não, está construída a base para se poder representar no ecrã do monitor qualquer tipo de informação visual, obviamente limitada no número de cores pelos recursos da FPGA. Assim sendo o módulo desenvolvido apenas actua nos sinais de sincronismo da porta VGA da plataforma de desenvolvimento e nos informa sobre as coordenadas do pixel actual. O controlo dos sinais “R”, “G” e “B” é efectuado externamente, de forma a possibilitar uma forma de utilização ainda mais flexível.

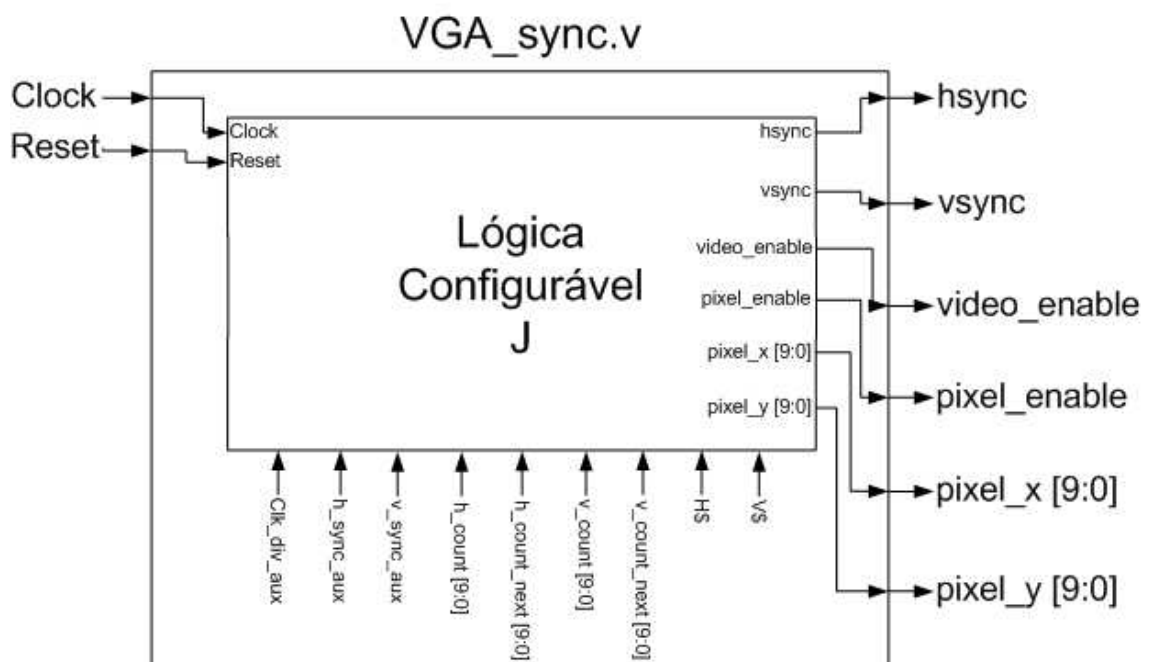


Figura 3.38 - Módulo “VGA_sync.v”.

Os sinais de saída “hsync” e “vsync” são os sinais de sincronismo horizontal e vertical e devem de ser ligados aos pinos respectivos do conector VGA da plataforma de desenvolvimento, através da adição das seguintes linhas no ficheiro “.ucf” do projecto, especificando a localização do recurso, o tipo de fios a utilizar na ligação, a intensidade de corrente e a velocidade de transição de valor lógico:

```
NET "hsync" LOC = "F15" | IOSTANDARD = LVTTTL | DRIVE = 8 | SLEW = FAST;
```

```
NET "vsync" LOC = "F14" | IOSTANDARD = LVTTTL | DRIVE = 8 | SLEW = FAST;
```

O sinal de saída “video_enable” indica se o pixel actual é um pixel de visualização. Se for os sinais “R”, “G” e “B” devem conter o valor respectivo da cor pretendida. Se o pixel actual não for visível, o valor dos sinais anteriormente referidos tem que ser ‘0’.

Como o *pixel rate* é metade da frequência de relógio interno da plataforma de desenvolvimento, é necessário saber quando é que este está activo, utilizando-se para este efeito o sinal “pixel_enable”.

Os sinais “pixel_x” e “pixel_y” indicam a coordenada x e y, respectivamente, do pixel actual, permitindo atribuir uma cor, através dos sinais “R”, “G” e “B” da porta VGA, a cada pixel individualmente.

No interior do módulo estão definidos oito parâmetros, que especificam os parâmetros horizontais e verticais do protocolo VGA. Estes parâmetros têm uma designação de duas letras sendo que a primeira especifica se é um parâmetro horizontal (H) ou vertical (V), e a segunda letra especifica o tipo de parâmetro: *Display* (D), *Front Porch* (F), *Back Porch* (B) e *Retrace* (R). Estes parâmetros são essenciais para o correcto processamento dos sinais de sincronismo.

O registo interno “Clk_div_aux” funciona como um divisor de relógio muito simples, cuja frequência é de 25MHz, o pretendido para o *pixel rate*.

Os registos “v_count” e “v_count_next” são utilizados para efectuar a contagem dos pixels verticais, sendo o seu resultado colocado em “pixel_y”. Estes registos são incrementados a cada subida de “Clk_div_aux” e quando o seu valor é igual a soma de todos os parâmetros verticais menos 1, estes registos são reiniciados.

Os registos “h_count” e “h_count_next” são utilizados para efectuar a contagem dos pixels horizontais, sendo o seu resultado colocado em “pixel_x”. Estes registos são incrementados a cada subida de “Clk_div_aux” e quando o seu valor é igual a soma de todos os parâmetros horizontais menos 1, estes registos são reiniciados.

Os registos “hsync_aux” e “vsync_aux” são utilizados para efectuar o sincronismo com o monitor e o seu resultado é colocado em “hsync” e “vsync”, respectivamente.

O sinal “hsync” assume o valor lógico ‘1’ sempre que o pixel actual tem uma coordenada x entre 656 (*Horizontal Display + Horizontal Back Porch*) e 751 (*Horizontal Display + Horizontal Back Porch + Horizontal Retrace - 1*).

O sinal “vsync” assume o valor lógico ‘1’ sempre que o pixel actual tem uma coordenada y entre 490 (*Vertical Display + Vertical Back Porch*) e 491 (*Vertical Display + Vertical Back Porch + Vertical Retrace - 1*).

3.3.8 - VGA (Modo Texto)

Com a fase de geração dos sinais de sincronismo do protocolo VGA concretizada, surgiram novas possibilidades de criação de módulos que utilizassem esse recurso. Nesse contexto, a ideia de poder ter caracteres imprimidos no ecrã é sem dúvida a que mais utilidade teria no contexto do trabalho realizado durante esta dissertação de mestrado.

Para isso seria necessário recorrer ao módulo anterior para efectuar o sincronismo entre a plataforma de desenvolvimento e o monitor, uma memória do tipo ROM para armazenar os dados relativos à fonte de caracteres, uma memória do tipo RAM para guardar o código ASCII de cada um dos caracteres em todas as posições, bem como um módulo que aglomerasse todos estes recursos e tivesse mecanismos para ler os dados das memórias RAM e ROM, escrever na memória RAM, e determinar se o pixel actual pertence ao carácter ou à cor de fundo.

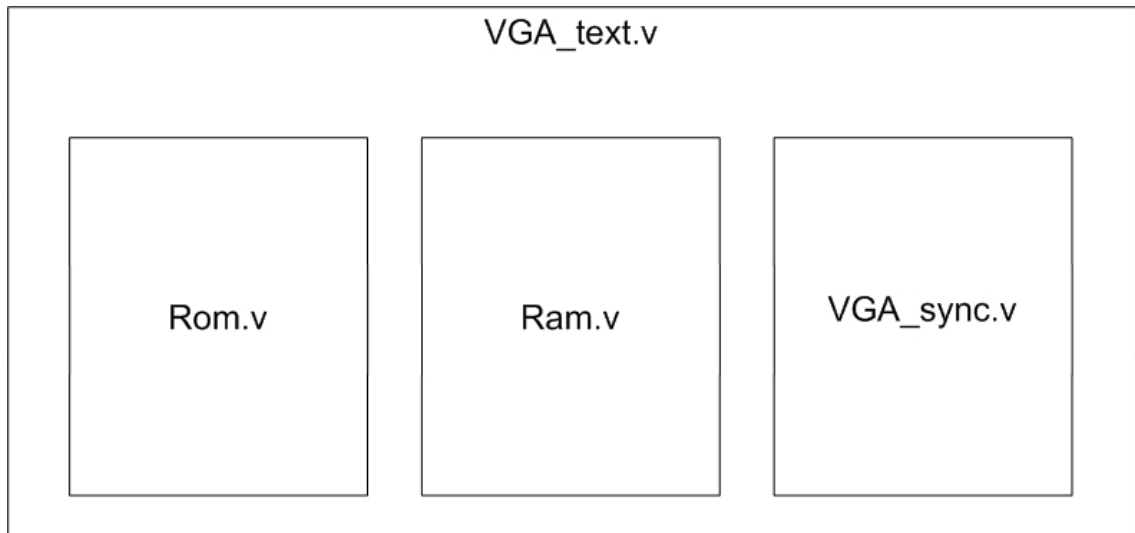


Figura 3.39 - Hierarquia dos módulos que implementam o protocolo VGA em modo de texto.

No trabalho prévio à criação do módulo, foi determinado que se iriam utilizar caracteres com a resolução de 8x8 pixels, dispostos em 80 colunas e 60 linhas. O mapa de bits dos caracteres foi obtido através de uma aplicação, desenvolvida pelo Professor Doutor Hélio Mendes de Sousa Mendonça, orientador dos trabalhos desta dissertação de mestrado, que converte os pixels de uma imagem que contem os caracteres na resolução pretendida, e ordenados pelo seu código ASCII, numa sequência de 1024 palavras com um byte de comprimento. A partir dessa sequência de palavras de oito bits, construiu-se o ficheiro designado “bitmaps.txt” que é copiado para a memória ROM durante a sua inicialização.

Assim sendo, a memória ROM é constituída por 1024 posições com oito bits cada uma, sendo que o endereço para consultar a mesma ocupa dez bits.

A memória RAM foi construída de forma a armazenar o código ASCII de cada um dos caracteres na respectiva posição, necessitando então de pelo menos 4800 posições de oito bits. O valor disponível de posições endereçáveis não menor que 4800 é 8192, sendo que os endereços para ler e escrever na memória ocupam treze bits.

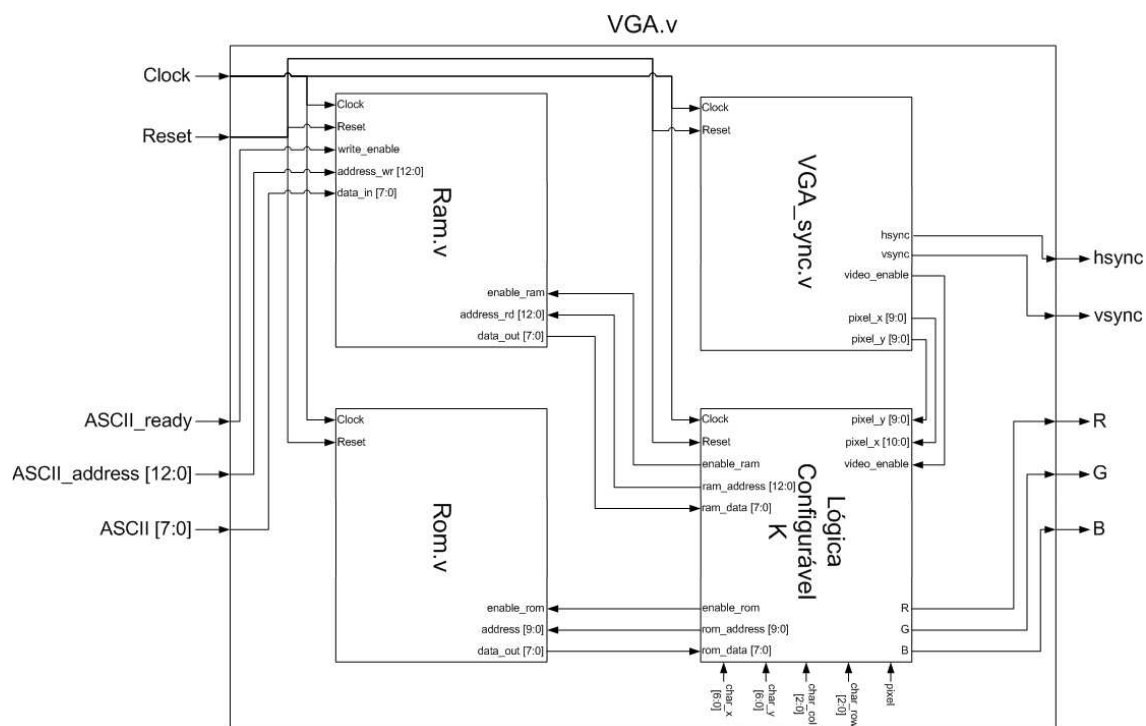


Figura 3.40 - Módulo “VGA.v”.

O módulo de topo desta interface, designado “VGA.v” e representado acima, aglomera todos os recursos necessários para a impressão de caracteres no monitor.

A entrada “ASCII_ready” é utilizada para indicar que se pretende escrever um novo carácter no monitor, indicando o número da célula a que pertence em “ASCII_address” e o respectivo código ASCII do carácter em “ASCII”.

As saídas “hsync” e “vsync” representam os sinais de sincronismo da plataforma de desenvolvimento com o monitor e provêm directamente do módulo “VGA_sync.v” já descrito. Deste módulo também se aproveitam as coordenadas x e y do pixel actual (sinais “pixel_x” e “pixel_y”) bem como o sinal “vídeo_enable” que indica se o pixel actual pertence à área visível do ecrã.

As saídas “R”, “G” e “B” representam recursos da porta VGA que informam sobre a cor do pixel actual e devem ser ligadas utilizando as seguintes linhas no ficheiro “.ucf” do projecto, especificando a sua localização, o tipo de fios a utilizar, a intensidade de corrente e a velocidade de transição:

```
NET "R" LOC = "H14" | IOSTANDARD = LVTTTL | DRIVE = 8 | SLEW = FAST;
```

```
NET "G" LOC = "H15" | IOSTANDARD = LVTTTL | DRIVE = 8 | SLEW = FAST;
```

```
NET "B" LOC = "G15" | IOSTANDARD = LVTTTL | DRIVE = 8 | SLEW = FAST;
```

Com base nestes sinais e sabendo que os pixéis são ordenados em células de oito por oito, é possível separar cada uma das coordenadas do pixel em duas partes: os sete bits mais significativos de cada uma das coordenadas representam o número da coluna ou da linha da célula a que o pixel pertence e os três menos significativos representam o número da coluna ou da linha a que o pixel pertence dentro da célula.

Assim o registo “char_x” abarca os sete bits mais significativos de “pixel_x” e indica a coordenada x da célula actual e o registo “char_y” abarcam os sete bits mais significativos de “pixel_y” e indica a coordenada y da célula actual.

O registo “char_col” reúne os três bits menos significativos de “pixel_x” e indica a coluna a que o pixel actual pertence dentro da célula e o registo “char_row” reúne os três bits menos significativos de “pixel_y” e indica a linha a que o pixel actual pertence dentro da célula.

O endereço de leitura da memória RAM é representado por “ram_address” e é obtido através de “char_x” e “char_y”. Depois de obter os dados relativos ao código ASCII da célula actual é calculado o endereço de leitura da memória ROM, representado por “rom_address”, a partir dos dados recebidos da memória RAM, “ram_data” e de “char_row”. Desta forma obtemos uma linha completa do *bitmap* do carácter, em “rom_data”, sendo estes dados analisados bit a bit utilizando “char_col”. Se o bit actual de “rom_data” estiver a ‘1’ este pixel pertence ao carácter, e os registos “R” e “B” são colocados a ‘0’ e o registo “G” a ‘1’ originando cor verde nos caracteres, se o seu valor for ‘0’ o pixel pertence ao fundo do ecrã, sendo que os registos relativos as cores são colocados todos a ‘0’.

3.3.9 - SVGA (Modo Gráfico)

Como alternativa de utilização ao protocolo VGA surgiu a oportunidade de criar um módulo, que não estava previsto inicialmente, implementando o protocolo SVGA. Este protocolo permite a utilização de uma área de 800 por 600 pixéis, utilizando um *pixel rate* de 50 MHz, aumentando drasticamente a resolução da imagem no monitor.

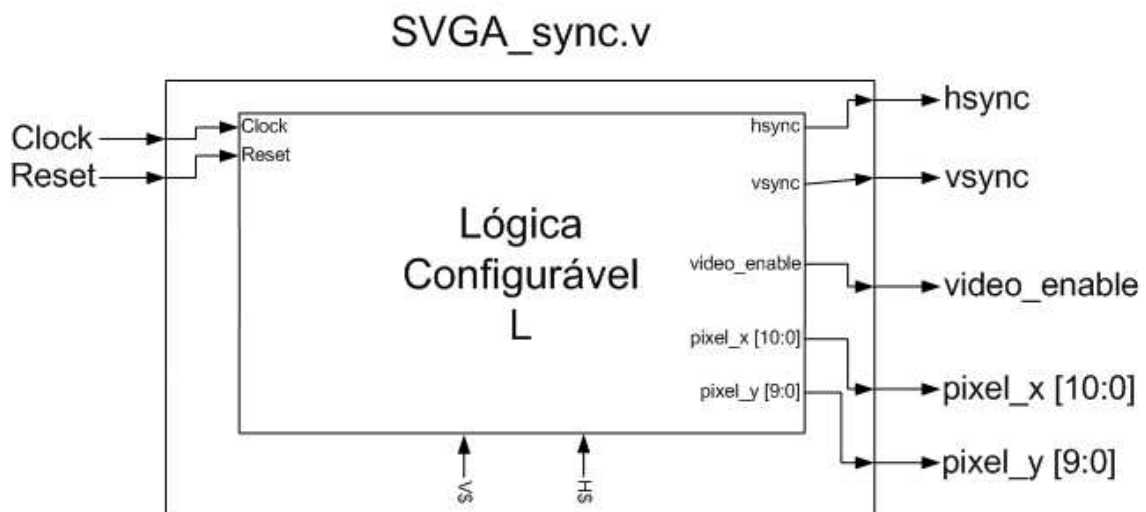


Figura 3.41 - Módulo “SVGA_sync.v”.

Para garantir a maior flexibilidade de utilização possível, tal como no módulo que implementa o sincronismo do protocolo VGA, este módulo não atribui valores aos sinais “R”, “G” e “B” que indicam a cor do pixel actual. A atribuição desses valores é externa ao módulo, tornando a sua reutilização extremamente fácil.

Os sinais de saída “hsync” e “vsync” são os sinais de sincronismo horizontal e vertical e devem de ser ligados aos pinos respectivos do conector VGA da plataforma de desenvolvimento, através da adição das seguintes linhas no ficheiro “..ucf” do projecto, especificando a localização do recurso, o tipo de fios a utilizar na ligação, a intensidade de corrente e a velocidade de transição de valor lógico:

```
NET "hsync" LOC = "F15" | IOSTANDARD = LVTTTL | DRIVE = 8 | SLEW = FAST;
```

NET "vsync" LOC = "F14" | IOSTANDARD = LVTTTL | DRIVE = 8 | SLEW = FAST;

O sinal de saída "video_enable" indica se o pixel actual é um pixel de visualização. Se for, os sinais "R", "G" e "B" devem conter o valor respectivo da cor pretendida. Se o pixel actual não for visível, o valor dos sinais anteriormente referidos tem que ser '0'.

Os sinais "pixel_x" e "pixel_y" indicam a coordenada x e y, respectivamente, do pixel actual, permitindo atribuir uma cor, através dos sinais "R", "G" e "B" da porta VGA, a cada pixel individualmente.

No interior do módulo existem dez parâmetros de sincronismo, divididos equitativamente entre parâmetros horizontais e verticais. Estes parâmetros têm uma designação de duas letras, sendo que a primeira especifica se é um parâmetro horizontal (H) ou vertical (V) e a segunda letra especifica o tipo de parâmetro: *Display* (D), *Front Porch* (F), *Back Porch* (B), *Retrace* (R) e *Total* (T). Estes parâmetros são essenciais para o correcto processamento dos sinais de sincronismo.

O sinal "pixel_x" é incrementado a cada subida de relógio, uma vez que o *pixel rate* é igual ao relógio interno da plataforma de desenvolvimento. Quando o seu valor é igual ao valor do parâmetro HT menos 1, a contagem de "pixel_x" é reiniciada e o sinal "pixel_y" incrementado. Se "pixel_x" estiver na condição anterior e "pixel_y" for igual a VT menos 1, a contagem de "pixel_y" é reiniciada.

O sinal de sincronismo horizontal "hsync" tem o valor lógico '1' quando o valor de "pixel_x" está entre 855 (*Horizontal Display + Horizontal Front Porch - 1*) e 975 (*Horizontal Total - Horizontal Back Porch - 1*), permanecendo no valor lógico '0' nas outras situações.

O sinal de sincronismo vertical "vsync" assume o valor lógico '1' quando "pixel_y" é igual 636 (*Vertical Display + Vertical Front Porch - 1*) e "pixel_x" é igual a 1039 (*Horizontal Total - 1*) regressando ao valor lógico '0' quando "pixel_y" é igual a 642 (*Vertical Total - Vertical Back Porch - 1*) e "pixel_x" é igual a 1039 (*Horizontal Total - 1*).

3.3.10 - SVGA (Modo Texto)

Uma vez construído o módulo que permite a sincronização da plataforma de desenvolvimento com o monitor no modo SVGA, o passo seguinte foi, tal como no caso do protocolo VGA, utilizar esse módulo para permitir a impressão de caracteres.

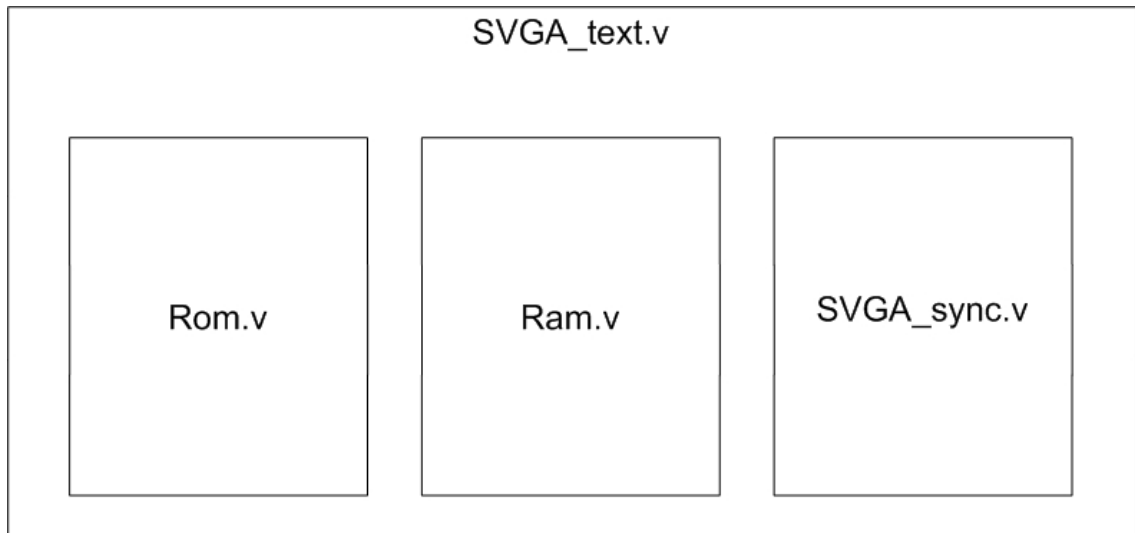


Figura 3.42 - Hierarquia dos módulos que implementam o protocolo SVGA em modo de texto.

Uma vez que os dois módulos que efectuam o sincronismo com o monitor, “VGA_sync.v” e “VGA_sync.v”, funcionam da mesma forma, as adaptações necessárias ao módulo “VGA.v”, que implementava a impressão de texto no monitor utilizando o protocolo VGA, de forma a funcionar utilizando o protocolo SVGA, são muito reduzidas. Essas alterações passam principalmente pelo aumento do número de linhas e colunas de caracteres, devido à maior resolução de pixels permitida pela implementação do protocolo SVGA. Utilizando o mesmo mapa de caracteres, com a mesma resolução por carácter (8x8), podemos dispor de 100 colunas e 75 linhas de caracteres, perfazendo um total de 7500 células. Como os caracteres utilizados são os mesmos e o número de células de caracteres ainda é menor que a capacidade disponibilizada pela memória RAM utilizada no protocolo VGA em modo de texto, as memórias (ROM e RAM), utilizadas anteriormente, foram reaproveitadas para este protocolo.

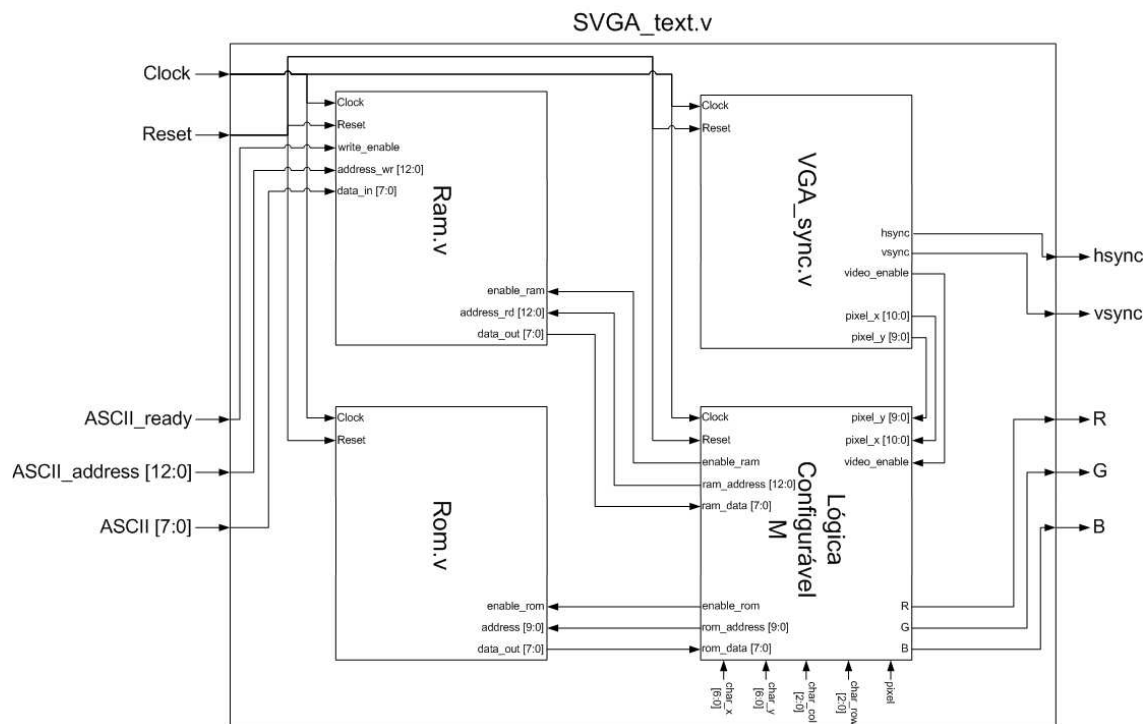


Figura 3.43 - Módulo “SVGA_text.v”.

O módulo desenvolvido funciona da mesma forma que o módulo “VGA.v”. As únicas diferenças provêm do aumento de resolução provocada pelo protocolo SVGA.

A entrada “ASCII_ready” é utilizada para indicar que se pretende escrever um novo carácter no monitor, indicando o número da célula a que pertence em “ASCII_address” e o respectivo código ASCII do carácter em “ASCII”.

As saídas “hsync” e “vsync” representam os sinais de sincronismo da plataforma de desenvolvimento com o monitor e provêm directamente do módulo “SVGA_sync.v” já descrito. Deste módulo também se aproveitam as coordenadas x e y do pixel actual (sinais “pixel_x” e “pixel_y”) bem como o sinal “vídeo_enable” que indica se o pixel actual pertence à área visível do ecrã.

As saídas “R”, “G” e “B” representam recursos da porta VGA que informam sobre a cor do pixel actual e devem ser ligadas utilizando as seguintes linhas no ficheiro “.ucf” do projecto, especificando a sua localização, o tipo de fios a utilizar, a intensidade de corrente e a velocidade de transição:

```
NET "R" LOC = "H14" | IOSTANDARD = LVTTTL | DRIVE = 8 | SLEW = FAST;
```

```
NET "G" LOC = "H15" | IOSTANDARD = LVTTTL | DRIVE = 8 | SLEW = FAST;
```

```
NET "B" LOC = "G15" | IOSTANDARD = LVTTTL | DRIVE = 8 | SLEW = FAST;
```

Com base nestes sinais, e sabendo que os pixéis são ordenados em células de 8 por 8, é possível separar cada uma das coordenadas do pixel em duas partes.

Assim, o registo “char_x” abarca os sete bits mais significativos de “pixel_x” e indica a coordenada x da célula actual e o registo “char_y” abarca os sete bits mais significativos de “pixel_y” e indica a coordenada y da célula actual.

O registo “char_col” reúne os três bits menos significativos de “pixel_x” e indica a coluna a que o pixel actual pertence dentro da célula e o registo “char_row” reúne os três bits menos significativos de “pixel_y” e indica a linha a que o pixel actual pertence dentro da célula.

O endereço de leitura da memória RAM é representado por “ram_address” e é obtido através de “char_x” e “char_y”. Depois de obter os dados relativos ao código ASCII da célula actual é calculado o endereço de leitura da memória ROM, representado por “rom_address”, a partir dos dados recebidos da memória RAM, “ram_data” e de “char_row”. Desta forma obtemos uma linha completa do *bitmap* do carácter, em “rom_data”, sendo estes dados analisados bit a bit utilizando “char_col”. Se o bit actual de “rom_data” estiver a ‘1’ este pixel pertence ao carácter, os registos “R” e “B” são colocados a ‘0’ e o registo “G” a ‘1’, originando cor verde nos caracteres. Se o seu valor for ‘0’ o pixel pertence ao fundo do ecrã, sendo que os registos relativos às cores são colocados todos a ‘0’.

3.3.11 - VGA e SVGA (Modo Gráfico)

Implementados os dois protocolos, faria todo o sentido criar um módulo que pudesse integrar os dois, e permitindo ao utilizador utilizar só um ou comutar entre ambos. Nesse sentido foi projectado um módulo que funciona como um multiplexador para os vários sinais de sincronismo entre a plataforma de desenvolvimento e o monitor.

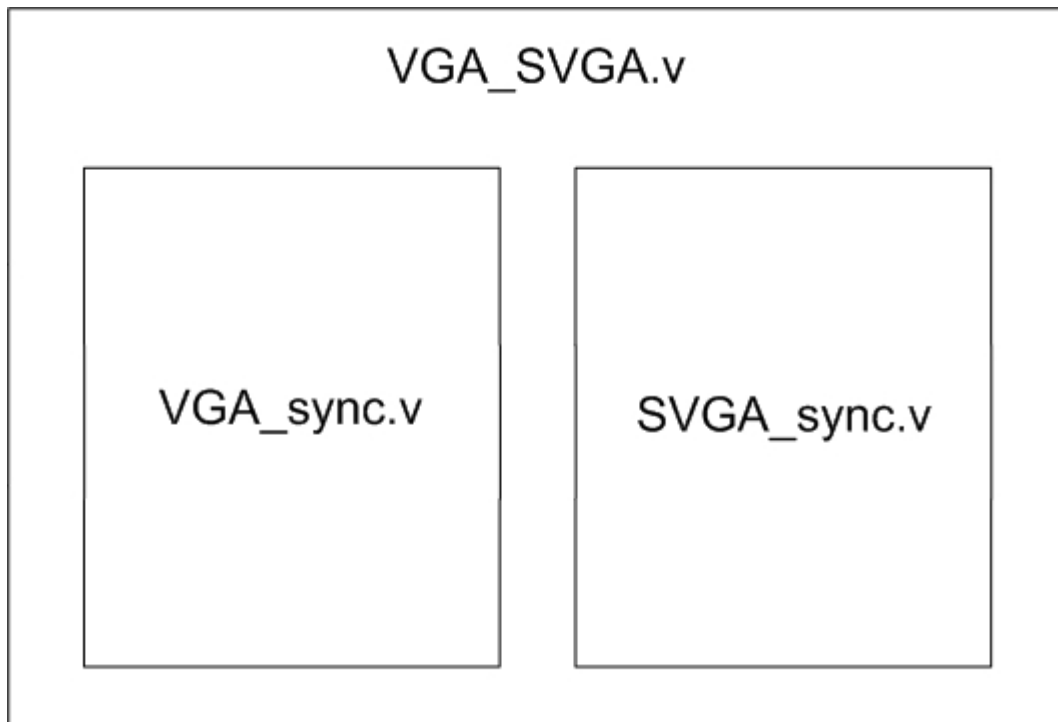


Figura 3.44 - Hierarquia dos módulos que implementam os protocolos VGA e SVGA.

Tal como nos dois módulos de sincronismo descritos anteriormente, este módulo não actua sobre os sinais “R”, “G” e “B”, permitindo uma ainda maior flexibilidade de utilização.

Estes sinais devem de ser processados externamente e permitem a atribuição de uma de oito cores disponíveis ao pixel actualmente a ser processado.

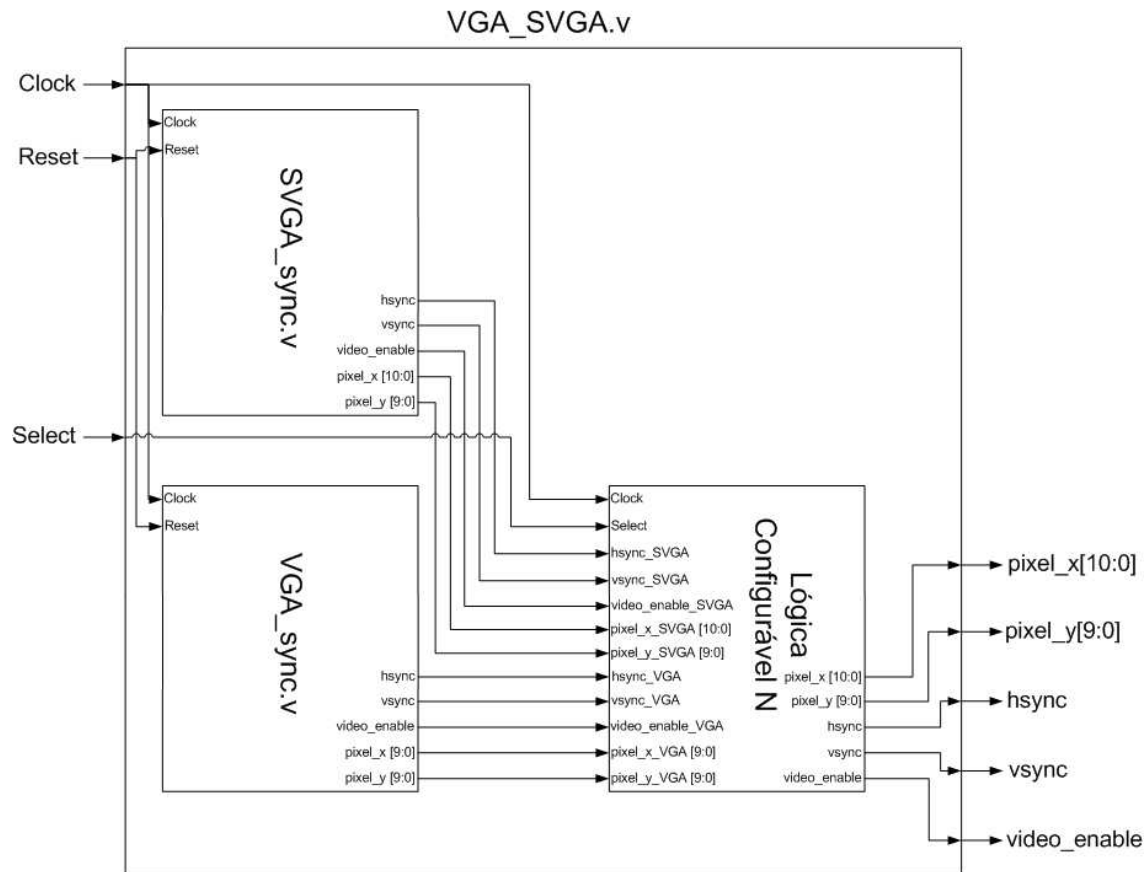


Figura 3.45 - Módulo “VGA_SVGA.v”.

A entrada “Select” representa um sinal de um bit que indica o protocolo a utilizar. Caso se pretenda utilizar o protocolo SVGA o valor lógico desta entrada deverá ser ‘1’. Se o valor de “Select” corresponder ao valor lógico ‘0’, o protocolo implementado é o VGA.

A lógica interna do módulo recebe os sinais de ambos os módulos de sincronismo (“VGA_sync.v” e “SVGA_sync.v”) e, consoante o valor da entrada “Select”, escolhe os sinais do protocolo seleccionado para serem propagados para os sinais de saída.

Os sinais de saída “hsync” e “vsync” são os sinais de sincronismo horizontal e vertical e devem de ser ligados aos pinos respectivos do conector VGA da plataforma de desenvolvimento, através da adição das seguintes linhas no ficheiro “.ucf” do projecto, especificando a localização do recurso, o tipo de fios a utilizar na ligação, a intensidade de corrente e a velocidade de transição de valor lógico:

```
NET "hsync" LOC = "F15" | IOSTANDARD = LVTTTL | DRIVE = 8 | SLEW = FAST;
```

```
NET "vsync" LOC = "F14" | IOSTANDARD = LVTTTL | DRIVE = 8 | SLEW = FAST;
```

O sinal de saída “video_enable” indica se o pixel actual é um pixel de visualização. Se for, os sinais “R”, “G” e “B” devem conter o valor respectivo da cor pretendida. Se o pixel actual não for visível, o valor dos sinais anteriormente referidos têm que ser ‘0’.

Os sinais “pixel_x” e “pixel_y” indicam as coordenadas x e y, respectivamente, do pixel actualmente a ser processado, permitindo atribuir individualmente a cada pixel, uma das oito cores permitidas pela plataforma de desenvolvimento.

3.3.12 - VGA e SVGA (Modo Texto)

Uma vez implementado o módulo anterior, que permite a transição entre o protocolo VGA e SVGA, introduziu-se esse módulo para efectuar o sincronismo entre a plataforma de desenvolvimento e o monitor no módulo “SVGA_text.v”.

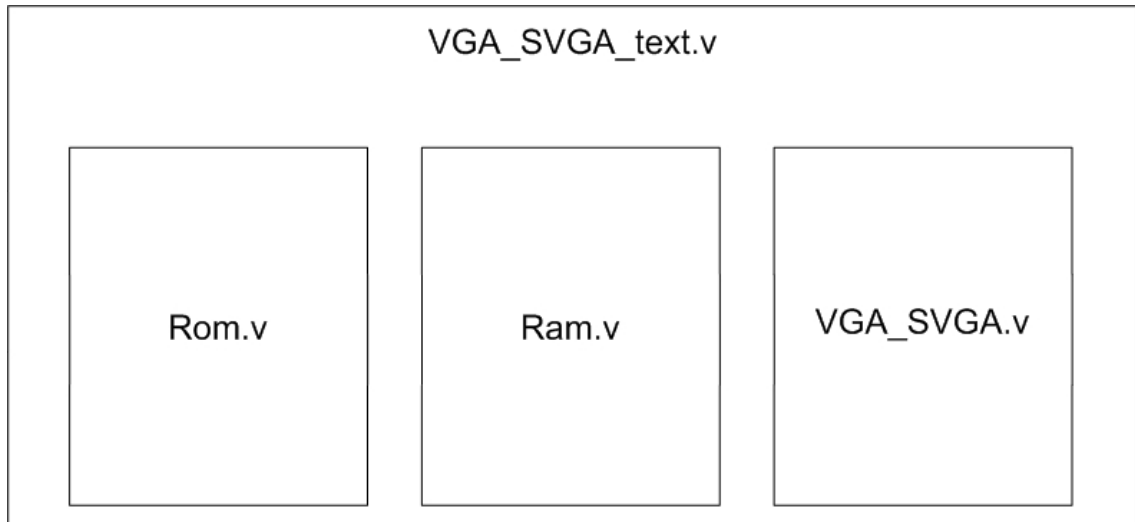


Figura 3.46 - Hierarquia dos módulos que implementam os protocolos VGA e SVGA em modo de texto.

Com algumas pequenas alterações, devido à implementação de ambos os protocolos, surgiu o módulo “VGA_SVGA_text.v” que permite a impressão de caracteres no monitor, utilizando o protocolo VGA ou o protocolo SVGA.

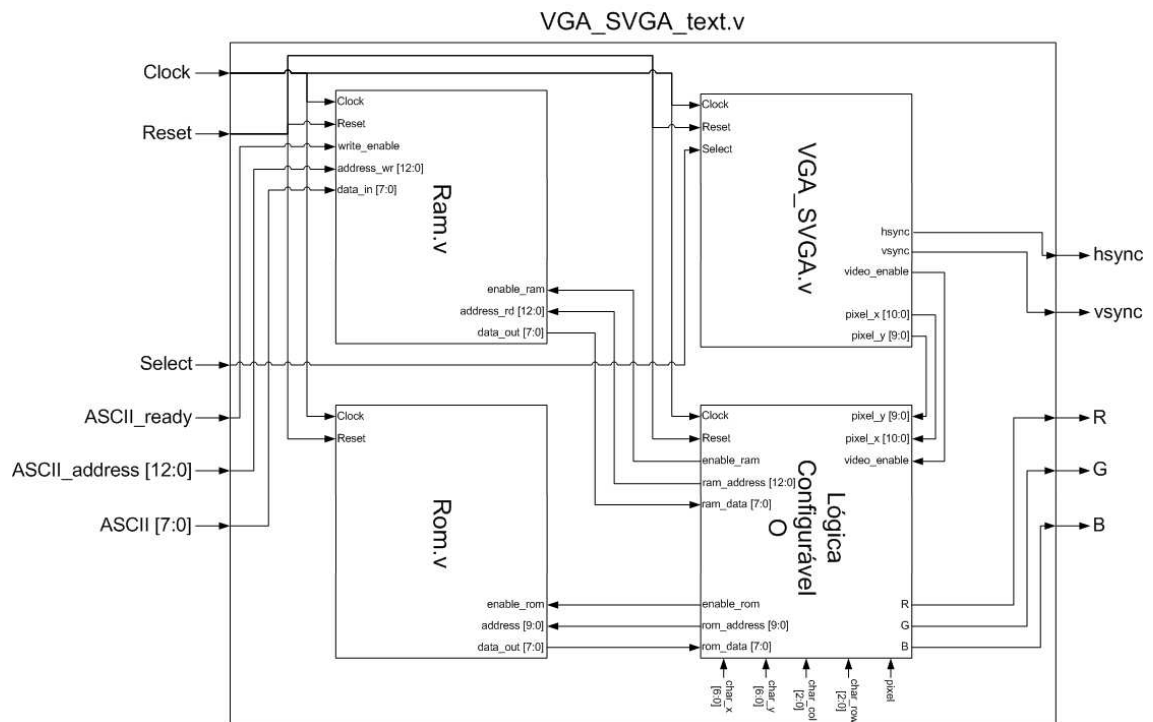


Figura 3.47 - Módulo “VGA_SVGA_text.v”.

As diferenças entre este módulo e o módulo “SVGA_text.v” devem-se às ténues diferenças introduzidas pelo aumento do número de píxeis em SVGA. Assim sendo os sinais “char_x”, “char_y” e “ram_address” assumem valores diferentes entre os dois protocolos, sendo o seu valor definido pela entrada “Select” que selecciona o protocolo a utilizar, da forma descrita no módulo anterior.

Os registos internos “char_x” e “char_y” apenas são afectados no número de bits que lhes são atribuídos, conforme o protocolo utilizado, sendo que o protocolo SVGA implica a utilização de mais um bit em ambos.

O endereço de leitura da memória RAM, “ram_address”, é calculado a partir de “char_x” e “char_y” sendo que a operação de cálculo é diferente para os dois protocolos. No protocolo VGA “ram_address” é igual à coordenada x do carácter mais a coordenada y do mesmo, multiplicada por oitenta (número de colunas em cada linha). No protocolo SVGA, a fórmula é semelhante, somente diferindo na multiplicação, sendo que neste modo existem cem colunas em cada linha.

Ao utilizar este método de cálculo do endereço da memória RAM, que é partilhado pelos dois protocolos, é permitida a reformatação automática do texto quando se alterna entre os dois protocolos disponíveis (VGA e SVGA).

Depois de obter os dados relativos ao código ASCII da célula actual, é calculado o endereço de leitura da memória ROM, representado por “rom_address”, a partir dos dados recebidos da memória RAM, “ram_data” e de “char_row”. Desta forma obtemos uma linha completa do *bitmap* do carácter, em “rom_data”, sendo estes dados analisados bit a bit utilizando “char_col”. Se o bit actual de “rom_data” estiver a ‘1’ este pixel pertence ao carácter e os registos “R” e “B” são colocados a ‘0’ e o registo “G” a ‘1’ originando cor verde nos caracteres, se o seu valor for ‘0’ o pixel pertence ao fundo do ecrã, sendo que os registos relativos às cores são colocados todos a ‘0’.

Os sinais de saída “hsync” e “vsync” são os sinais de sincronismo horizontal e vertical provenientes do módulo “VGA_SVGA.v” e devem de ser ligados aos pinos respectivos do conector VGA da plataforma de desenvolvimento, através da adição das seguintes linhas no ficheiro “.ucf” do projecto, especificando a localização do recurso, o tipo de fios a utilizar na ligação, a intensidade de corrente e a velocidade de transição de valor lógico:

```
NET "hsync" LOC = "F15" | IOSTANDARD = LVTTTL | DRIVE = 8 | SLEW = FAST;
```

```
NET "vsync" LOC = "F14" | IOSTANDARD = LVTTTL | DRIVE = 8 | SLEW = FAST;
```

As saídas “R”, “G” e “B” representam recursos da porta VGA que informam sobre a cor do pixel actual e devem ser ligadas utilizando as seguintes linhas no ficheiro “.ucf” do projecto, especificando a sua localização, o tipo de fios a utilizar, a intensidade de corrente e a velocidade de transição:

```
NET "R" LOC = "H14" | IOSTANDARD = LVTTTL | DRIVE = 8 | SLEW = FAST;
```

```
NET "G" LOC = "H15" | IOSTANDARD = LVTTTL | DRIVE = 8 | SLEW = FAST;
```

```
NET "B" LOC = "G15" | IOSTANDARD = LVTTTL | DRIVE = 8 | SLEW = FAST;
```

3.3.13 - Som

Durante o desenvolvimento das aplicações, que serão descritas posteriormente, utilizadas na validação e verificação dos módulos descritos e para animar as mesmas, tornando-as mais cativantes, foi criado um módulo que permitisse a reprodução de sons, através da utilização

de um pequeno altifalante ligado a um dos portos de expansão da plataforma de desenvolvimento.

O módulo foi projectado de forma a poder reproduzir, durante um intervalo de tempo pré-definido, um de oito sons disponibilizados. No entanto, o módulo foi desenhado para que, através da passagem de parâmetros em Verilog, fosse possível alterar a duração do som, e a frequência do som reproduzido, permitindo uma flexibilidade total da sua utilização, sem perder a sua simplicidade de integração.

Caso se pretenda personalizar a nota a reproduzir, bem como a sua duração, o primeiro parâmetro a introduzir representa a duração do som em ciclos de relógio e o segundo representa o número de ciclos de relógio entre as transições do sinal de saída. Assim sendo, o som número 0 fica personalizado.

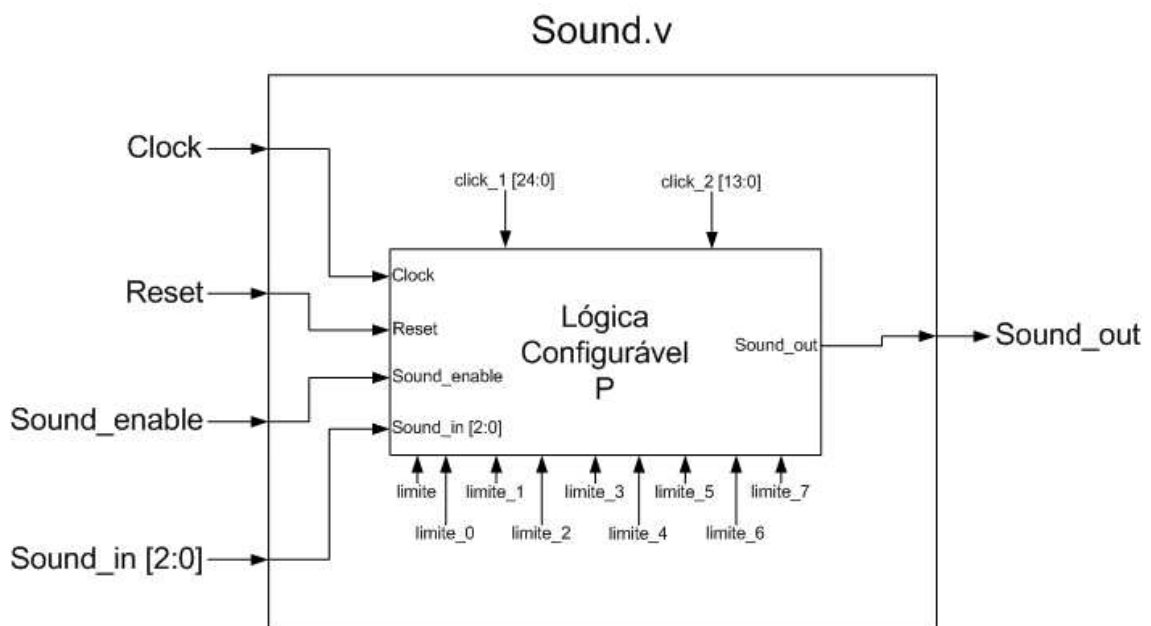


Figura 3.48 - Módulo “Sound.v”.

A entrada “Sound_enable” indica ao módulo quando deve iniciar a reprodução do som indicado na entrada “Sound_in”.

Internamente o módulo possui dois registos, que funcionam como contadores, e nove parâmetros. O parâmetro “limite” especifica a duração do som em ciclos de relógio. Os restantes parâmetros representam o número de ciclos de relógio entre transições de sinal na saída, para cada um dos oito sons.

O registo “click_1” é incrementado a cada subida do relógio interno e regula a duração do som. O registo “click_2” é incrementado a cada subida de relógio, durante o período de reprodução do som, e determina quando é que o sinal de saída deve ser negado.

A saída “Sound_out” é o sinal sonoro produzido e deve ser ligado ao pólo positivo (o pólo negativo deverá ser ligado à massa da plataforma de desenvolvimento) da coluna de som utilizada.

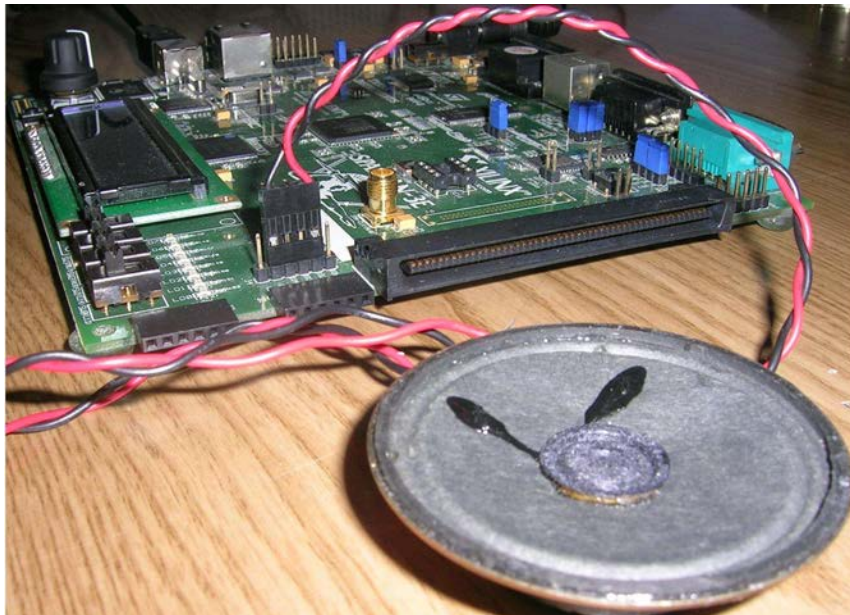


Figura 3.49 - Ligação da coluna à plataforma de desenvolvimento.

No caso do trabalho realizado, a coluna foi ligada ao porto de expansão, denominado “J4”, da plataforma de desenvolvimento, sendo adicionada a seguinte linha ao ficheiro “.ucf” do projecto, especificando a localização do pólo positivo da coluna, o tipo de fios a utilizar na ligação e utilizando uma resistência de *pull-up*.

```
NET "Sound_out" LOC = "C7" | IOSTANDARD = LVTTTL | PULLUP;
```

3.4 - Criação das Aplicações

Nesta secção deste capítulo será descrita a estrutura, o funcionamento e as características importantes das duas aplicações desenvolvidas para efectuar a verificação e validação dos módulos criados. O objectivo destas duas aplicações é aglomerar o maior número de módulos já desenvolvidos e descritos, demonstrando o seu funcionamento, bem como a sua integração num projecto.

Todas as entradas de todos módulos com o nome “Clock” deverão ser ligadas ao relógio interno da plataforma de desenvolvimento, através da adição da seguinte linha ao ficheiro “.ucf” do projecto, especificando a localização do recurso e especificando para utilizar as linhas destinadas à propagação do sinal de relógio:

```
NET "clock" LOC = "C9" | IOSTANDARD = LVCMOS33;
```

Todas as entradas de todos módulos com o nome “Reset” deverão ser ligadas ao botão de pressão usado para efectuar o reset externo à FPGA. Normalmente utiliza-se o botão “South”. A linha de código a adicionar ao projecto deve ser semelhante às utilizadas no módulo dos botões de pressão, que já foi descrito, alterando apenas o nome da “NET” e especificando a localização do recurso pretendida.

3.4.1 - Breakout

O jogo *Breakout*, também conhecido por *Brick Breaker* ou *Arkanoid*, consiste na sustentação de uma bola, usando uma barra controlada pelo jogador, para que esta destrua todos os tijolos presentes na parte superior da área de jogo.

Nesta aplicação era pretendida, no mínimo, a integração de um protocolo que possibilitasse ao jogador controlar a barra e de desenhar os objectos do jogo num monitor usando o protocolo VGA. No entanto mais módulos foram utilizados, tendo então uma aplicação que engloba uma maior parte do trabalho realizado. Os módulos utilizados permitem a utilização de um teclado PS/2 para o jogador controlar a barra, a utilização do monitor em modo gráfico e de texto, simultaneamente, do protocolo VGA, a utilização do LCD da plataforma de desenvolvimento e a reprodução de sons.

Estruturalmente a aplicação foi dividida em duas partes, um módulo que controlasse o movimento de todos os objectos no jogo e outro que era responsável pelo controlo do monitor e do LCD da plataforma de desenvolvimento. Estes dois módulos têm que comunicar entre si, para que o módulo que controla os gráficos saiba as coordenadas da bola e da barra, os tijolos já destruídos, a pontuação e o número de vidas restantes, de forma a imprimir correctamente todos os detalhes do jogo.

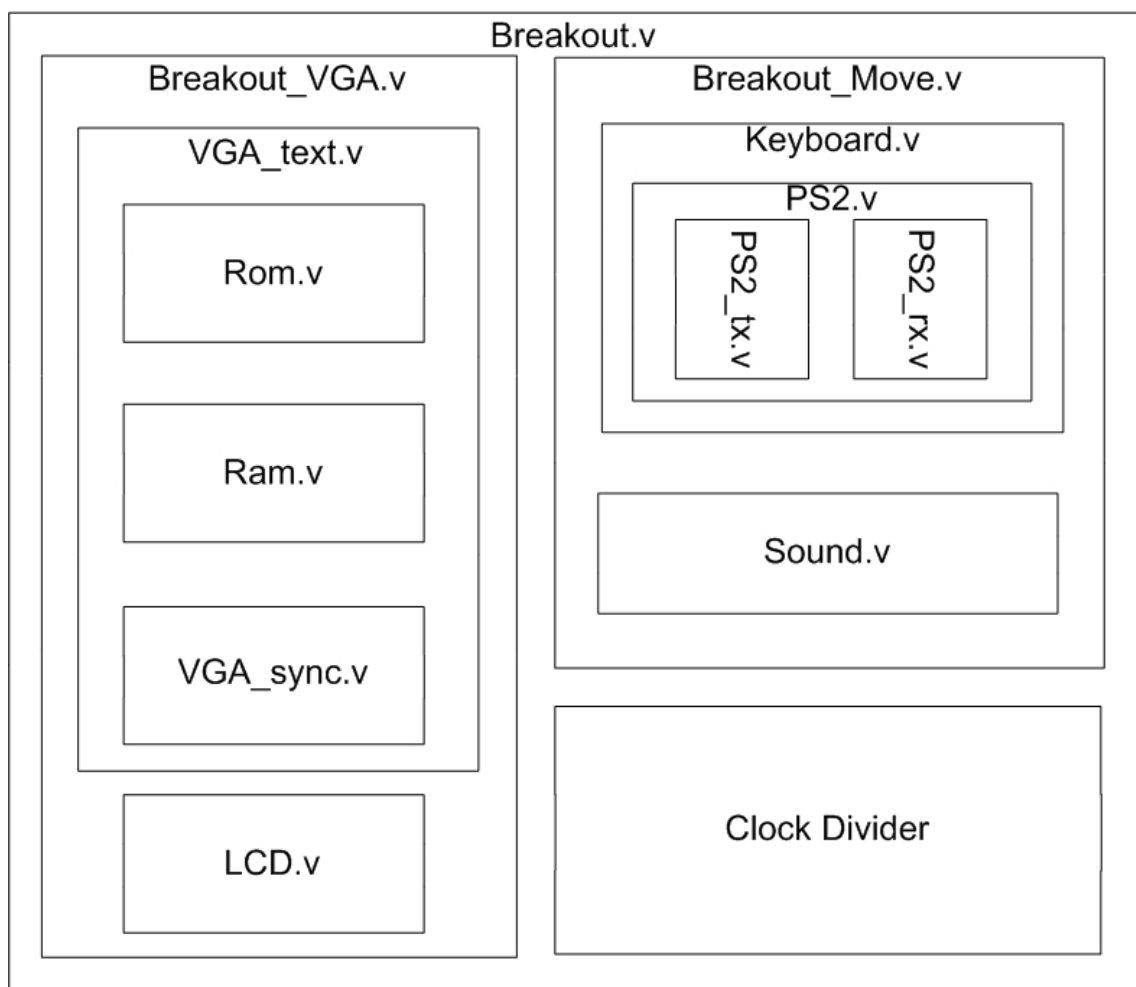


Figura 3.50 - Hierarquia dos módulos que implementam a aplicação "Breakout".

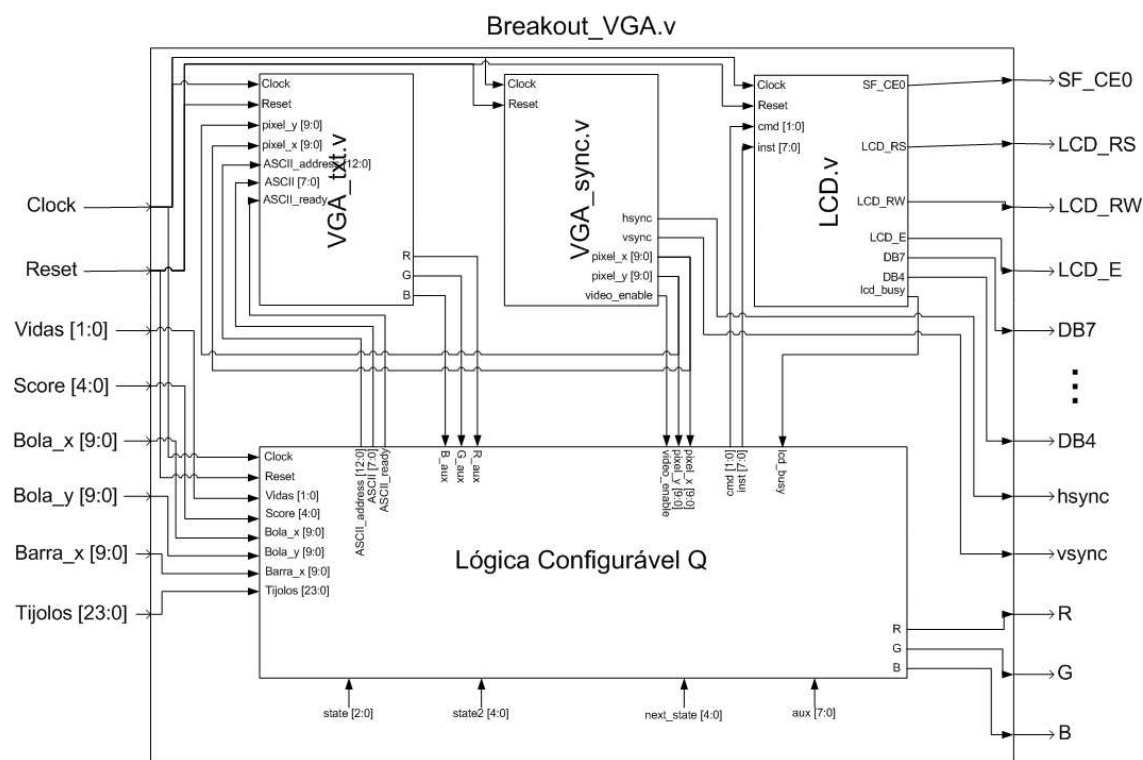


Figura 3.51 - Módulo “Breakout_VGA.v”.

Este é o módulo que controla todos os aspectos gráficos do jogo. Inicialmente, e de forma a integrar o modo gráfico e de texto do protocolo VGA, tiveram que se definir áreas para cada um destes modos. Sendo assim, a área designada para o modo de texto ficou limitada aos extremos da área visível do ecrã, permitindo a impressão de caracteres na primeira e última coluna, bem como na primeira e última linha. Assim, permite-se a utilização do modo de texto do protocolo VGA sem a redução drástica da área de jogo.

Para o utilizador saber os limites da área de jogo foi criado um rectângulo delimitador da mesma, sendo que a linha que o define possui oito pixéis de largura.

Foram também definidos outros parâmetros do jogo, tais como altura e coordenada vertical da barra do jogador e o seu comprimento. Também foram definidos o diâmetro da bola, o comprimento e altura dos tijolos, bem como a sua localização na área de jogo.

Todos os sinais de entrada deste módulo, excepto “Clock” e “Reset”, são provenientes do módulo que controla os movimentos do jogo.

O sinal “Vidas” indica o número de vidas remanescentes ao jogador, na tentativa de destruir todos os tijolos e o sinal “Score” indica a sua pontuação actual.

Os sinais “Bola_x” e “Bola_y” indicam respectivamente as coordenadas x e y da bola, uma vez que este objecto se movimenta no decorrer do jogo.

Durante o jogo a barra do jogador também se movimenta, mas apenas horizontalmente, sendo a sua coordenada x transmitida através do sinal de entrada “Barra_x”.

De forma a este módulo saber quais os tijolos que ainda existem e quais já foram destruídos existe a entrada “Tijolos” de 24 bits, sendo que cada um destes representa o estado de um tijolo.

Os sinais de saída “SF_CEO”, “LCD_RS”, “LCD_RW”, “LCD_E”, “DB7”, “DB6”, “DB5” e “DB4” realizam a interface deste módulo com o LCD da plataforma de desenvolvimento e

devem ser ligados aos respectivos recursos da forma já especificada na secção destinada à descrição do módulo “LCD.v”.

Os sinais “hsync”, “vsync”, “R”, “G” e “B” efectuam a ligação entre o jogo e o conector VGA da plataforma de desenvolvimento, permitindo o sincronismo e a visualização do jogo num monitor utilizando o protocolo VGA. Estes sinais devem de ser ligados aos respectivos recursos da forma descrita na secção destinada à descrição do módulo “VGA_sync.v”.

Internamente neste módulo estão integrados os módulos, já descritos anteriormente, “VGA.v” (responsável pelo modo texto do protocolo VGA, ligeiramente alterado), “VGA_sync.v” (responsável pela sincronização da FPGA com o monitor, e indicação das coordenadas x e y do pixel actualmente a ser processado) e o módulo “LCD.v” (responsável pelo controle do LCD da plataforma de desenvolvimento).

De forma a controlar o módulo “LCD.v” foi criada uma máquina de estados, controlada por “state2” e “next_state”, com 24 estados definidos. Estes estados definidos têm como objectivo escrever no LCD (utilizando os registos “cmd” e “inst”) o número de vidas disponíveis e a pontuação, sendo estes valores actualizados, ciclicamente. O estado 0 é responsável pelo envio do comando de limpar o ecrã do LCD, passando ao estado 1. O estado 1 é um estado muito particular pois é usado sempre que se envia uma instrução ao LCD. Este estado espera pela conclusão dessa operação, indicada pela descida ao valor lógico ‘0’ do sinal “lcd_busy”, e quando isto acontece a FSM passa para o estado guardado em “next_state”. Assim evita-se a replicação sucessiva deste estado, minimizando o número de estados necessários. Os estados 2, 3, 4, 5, 6, 7 e 8 são utilizados para instruir a escrita de cada um dos caracteres presentes de “Score: ” (incluindo o espaço), regressando após cada um desses estados ao estado 1, indicando em “next_state” qual o próximo estado a visitar. O estado 9 é utilizado para mudar o cursor para o primeiro carácter da segunda linha, visitando de seguida o estado 1 e passando para o estado 10 após a operação ter sido completada. Os estados 10, 11, 12, 13, 14, 15 e 16 são utilizados na escrita de cada um dos caracteres de “Vidas: ” (incluindo o espaço), regressando após cada um desses estados ao estado 1, indicando em “next_state” qual o próximo estado a visitar. Todos os estados entre o estado 0 e 16, excluindo o estado 1, só são visitados uma vez após cada inicialização, pois representam a escrita de informação que não será alterada. No estado 17 é ordenada a escrita do número de vidas restantes ao jogador, visitando de seguida o estado 1 e passando de seguida ao estado 18. Neste estado é ordenada a mudança do cursor para o oitavo carácter da primeira linha do LCD, onde será escrito o primeiro dígito da pontuação do jogador, visitando de seguida o estado 1 e passando de seguida ao estado 19. Nos estados 19, 20, 21 e 22 é ordenada a escrita dos 4 dígitos da pontuação do jogador, visitando após cada escrita o estado 1 e indicando em “next_state” o estado seguinte a visitar. Quando terminar a escrita da pontuação, passa-se para o último estado (23) que ordena a mudança do cursor para o oitavo carácter da segunda linha do LCD, regressando ao estado 1, sendo que o estado seguinte a visitar, guardado em “next_state”, será o estado 17 de forma a actualizar continuamente o número de vidas e pontuação do jogador.

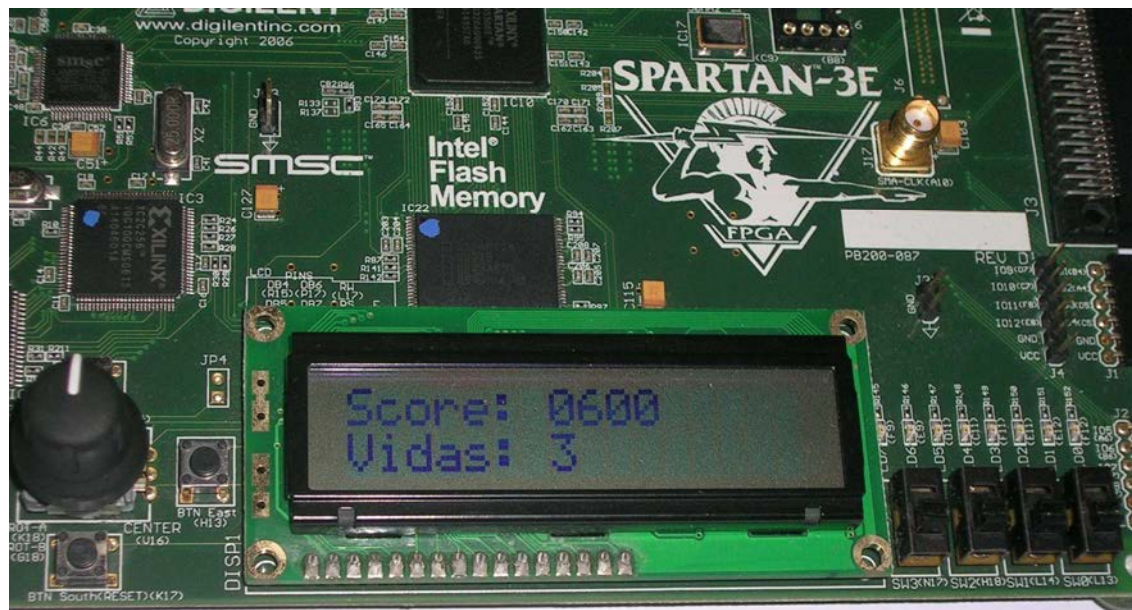


Figura 3.52 - Aspecto do LCD durante o jogo “Breakout”.

Para actualizar a informação relativa ao jogo, disponibilizada pelo modo de texto do protocolo VGA, é utilizada outra máquina de estados, controlada por “state”, com 6 estados definidos. No estado 0 ordena-se a escrita do número de vidas do jogador na respectiva posição da memória RAM do controlador de texto, utilizando os registos “ASCII_ready”, “ASCII_address” e “ASCII”, passando para o estado 1. Neste estado o registo “ASCII_ready” é colocado a ‘0’ e, passando de seguida ao estado 2, onde se ordena a escrita do primeiro dígito da pontuação. De seguida é visitado o estado 3 que tem a mesma função do estado 1, passando de seguida ao estado 4, onde é escrito o segundo dígito da pontuação do jogador. O último estado a ser visitado é o estado 5, que funciona da mesma forma que o estado 3 e 1, regressando ao estado 0, permitindo a actualização constante dos dados. Os dois últimos dígitos da pontuação são constantes e iguais a 0 pelo que já estão previamente guardados na memória RAM do controlador de texto.

O controlador de texto do protocolo VGA utilizado é ligeiramente diferente do descrito anteriormente. As únicas diferenças entre ambos devem-se à inicialização da memória RAM do controlador e da utilização do módulo de sincronismo “VGA_sync.v” externamente ao controlador. Desta forma, o módulo “VGA.v” aqui utilizado possui mais duas entradas para receber as coordenadas x e y do pixel actualmente a ser processado. A memória RAM é inicializada a partir de um ficheiro de texto designado “ram.txt”, que é constituído por 8192 linhas com 8 dígitos binários, de forma a escrever na memória RAM todos os caracteres que não serão alterados durante o jogo. Isto evita a criação de uma máquina de estados de gestão do modo de texto muito maior e com muitos estados que apenas seriam visitados uma vez, após cada inicialização.

A lógica interna do módulo “Breakout_VGA.v” é responsável por definir as áreas destinadas ao modo de texto e modo gráfico do protocolo VGA. Também é responsável pelo desenho de todos os objectos do jogo (bola, barra, tijolos e margens).

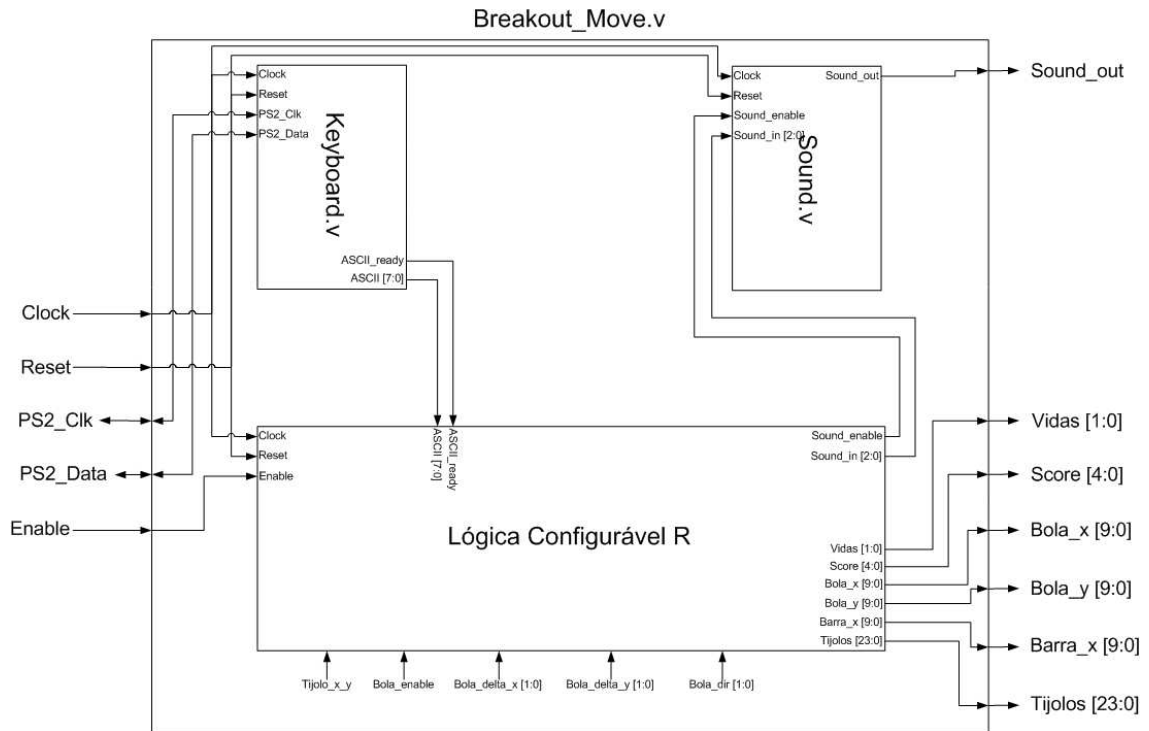


Figura 3.53 - Módulo “Breakout_Move.v”.

Este módulo controla os movimentos dos objectos presentes no jogo, bem como a interacção entre eles.

Internamente no módulo estão integrados os módulos, descritos anteriormente, “Keyboard.v” e “Sound.v”, que são responsáveis pela comunicação com o teclado PS/2 e pela reprodução de som.

As entradas “PS2_Clk” e “PS2_Data” representam as linhas de relógio e de dados do barramento PS/2 e devem de ser ligadas aos respectivos recursos da forma descrita na secção que aborda os módulos que implementam o protocolo PS/2.

A entrada “Enable” está ligada a um divisor de relógio no módulo superior, que serve para abrandar o processamento de dados, tornando a velocidade de funcionamento da aplicação razoável para o utilizador comum.

Todas as saídas deste módulo devem de ser ligadas ao módulo “Breakout_VGA.v”, excepto a saída “Sound_out” que deverá ser ligada ao pólo positivo do altifalante, tal como descrito na secção que abrange o módulo “Sound.v”. Os sinais “Vidas” e “Score” indicam, respectivamente, o número de vidas remanescentes ao jogador e a sua pontuação actual. As saídas “Bola_x” e “Bola_y” indicam as coordenadas x e y da bola e o sinal “Barra_x” a coordenada x da barra do jogador. O estado de cada um dos tijolos (activo ou destruído) é representado pelo sinal “Tijolos” sendo que cada bit representa um objecto.

O controlo do jogo é realizado através de um teclado PS/2 ligado directamente à plataforma de desenvolvimento, utilizando apenas 3 teclas. Para iniciar/pausar o jogo basta premir a tecla de espaço do teclado. Para mover a barra do jogador para a esquerda e para a direita, basta premir as teclas dos caracteres “A” e “D”, respectivamente. Enquanto o jogo estiver em pausa, não é possível mover a barra do jogador, pois isso levaria a situações de batota.

O módulo de som está integrado neste módulo de modo a facilitar a sua integração, uma vez que os sinais sonoros são originados pelas interacções entre objectos. No decorrer do jogo

são reproduzidos 5 sons diferentes, utilizando para esse efeito os sinais “Sound_enable” e “Sound_in”. Um para quando a bola bate nas margens de jogo laterais e superior ou na barra do jogador, outro diferente quando bate na margem inferior, sinalizando a perda de uma vida e um som diferente para cada das três linhas de tijolos.

O registo interno “Bola_enable” indica se o jogo está activo ou em pausa e serve para activar e desactivar todos os movimentos.

O movimento da bola é definido por três registos “Bola_dir”, “Bola_delta_x” e “Bola_delta_y”. O primeiro é composto por dois bits sendo que o mais significativo indica o sentido do movimento da bola no eixo x e o menos significativo indica o sentido do movimento da bola no eixo y. Se o bit mais significativo for igual a ‘0’ a bola está a movimentar-se para a esquerda, se for igual a ‘1’ a bola está a movimentar-se para a direita. Se o bit menos significativo for igual a ‘0’ a bola está a movimentar-se para cima, se for igual a ‘1’ a bola está a movimentar-se para baixo. Os outros dois registos indicam a variação das coordenadas x e y da bola, em pixéis, a cada activação do sinal de entrada “Enable”.

Para controlar o movimento da bola e evitar que ela passe as margens existe uma máquina de estados controlada pelo registo “Bola_dir” que controla o movimento da bola, analisando, conforme o sentido de movimento, a sua posição em relação às margens. Esta máquina de estados apenas controla a interacção da bola com as margens, não controlando a interacção da bola com outros objectos. É também responsável pela actualização do número de vidas e da pontuação, uma vez que quando a bola toca na margem inferior é subtraída uma vida e a pontuação é apagada, fazendo uma reinicialização do jogo caso não haja mais vidas.

A interacção entre a bola e a barra do jogador é feita através da utilização de condições *if/else* possibilitando 3 tipos diferentes de contacto, aumentando o realismo e complexidade do jogo. Quando a bola toca nos extremos da barra o seu movimento será mais rápido no eixo x do que no eixo y (utilizando para isso os registos “Bola_delta_x” e “Bola_delta_y”) e os dois sentidos invertidos. Se a bola tocar entre os extremos e um quarto do seu comprimento total, a velocidade de deslocamento será igual nos dois eixos sendo apenas invertido o sentido de movimento no eixo y. Se a bola tocar no centro da barra, a bola movimentar-se-á mais rapidamente no eixo y do que no eixo x, sendo apenas invertido o sentido de movimento no eixo y.

A interacção entre a bola e cada um dos oito tijolos das três filas é implementado de forma igual, com a alteração de que cada vez que um tijolo é destruído, a pontuação é aumentada em pontos. O estado de cada um dos tijolos está também representado nos registos “Tijolo_y_x” onde y representa a fila de tijolos e x representa a coluna de tijolos. Estes registos são depois concatenados de forma a originar a saída “Tijolos”.

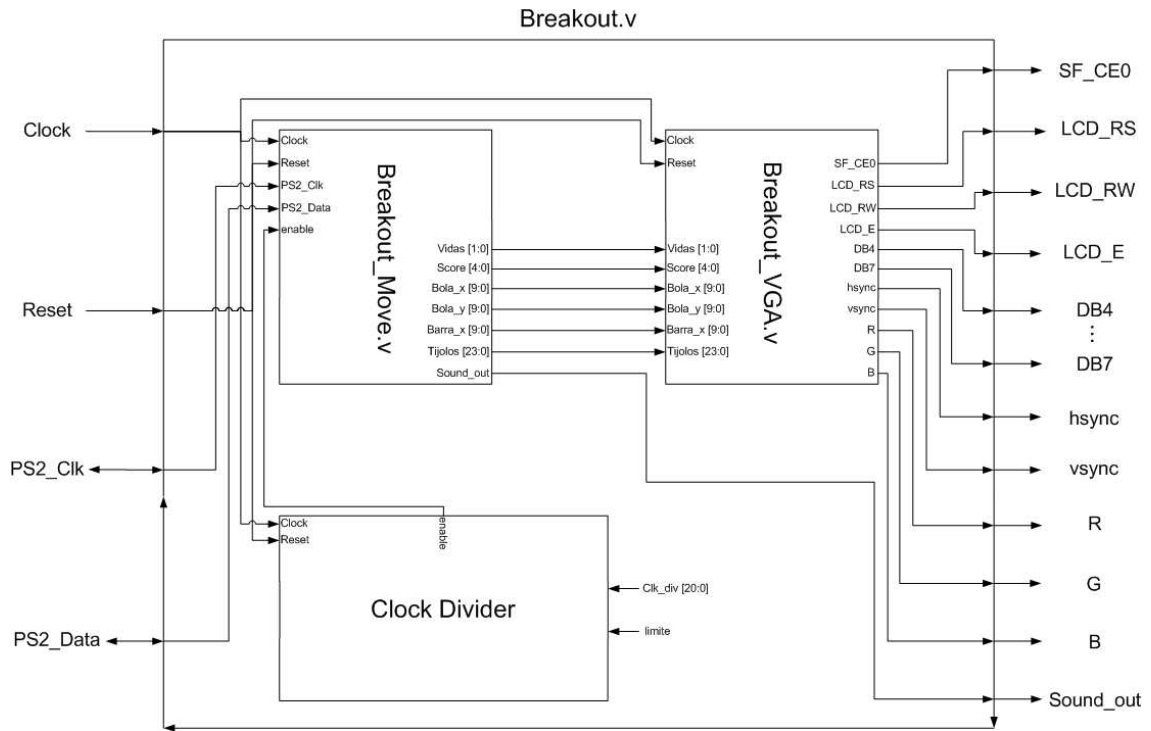


Figura 3.54 - Módulo “Breakout.v”.

O módulo “Breakout.v” efectua as ligações entre os dois módulos inferiores “Breakout_Move.v” e “Breakout_VGA.v”, bem como a ligação dos sinais de saídas destes que tem como destino os recursos da plataforma de desenvolvimento.

O módulo ainda possui um divisor de relógio que controla as activaões de “enable” e regula a dificuldade do jogo. O parâmetro “limite” define o número de ciclos de relógio entre activaões de “enable” e o registo “Clk_div” é incrementado a cada subida de relógio, de forma a contar o número de ciclos de relógio que já passaram desde a última activaão. O valor pré-definido de “limite” indica um milhão de ciclos de relógio entre activaões de “enable”, podendo este valor ser alterado para aumentar ou reduzir a dificuldade do jogo.

Está disponível no manual desta aplicação, que se encontra na página da Internet desta dissertação [18], uma ligação para um vídeo que mostra a aplicação em funcionamento.



Figura 3.55 - Aspecto do monitor VGA durante a aplicação “Breakout”.

3.4.2 - Pong

De forma a mostrar ainda mais capacidades dos módulos desenvolvidos, foi projectada e desenvolvida uma nova aplicação, para dois jogadores em simultâneo, que simula o jogo “Pong”. O objectivo deste jogo é semelhante ao do ténis de mesa, mas neste caso a bola não pode sair da mesa.

O jogo implementa o protocolo VGA (modo de texto e gráfico simultaneamente), o protocolo PS/2 (rato e teclado simultaneamente), utiliza o LCD da plataforma de desenvolvimento, bem como a reprodução de sons.

A estrutura desenvolvida para esta aplicação é muito semelhante à da aplicação anterior, possuindo também um módulo para controlo dos aspectos gráficos e outro para controlo de movimentos.

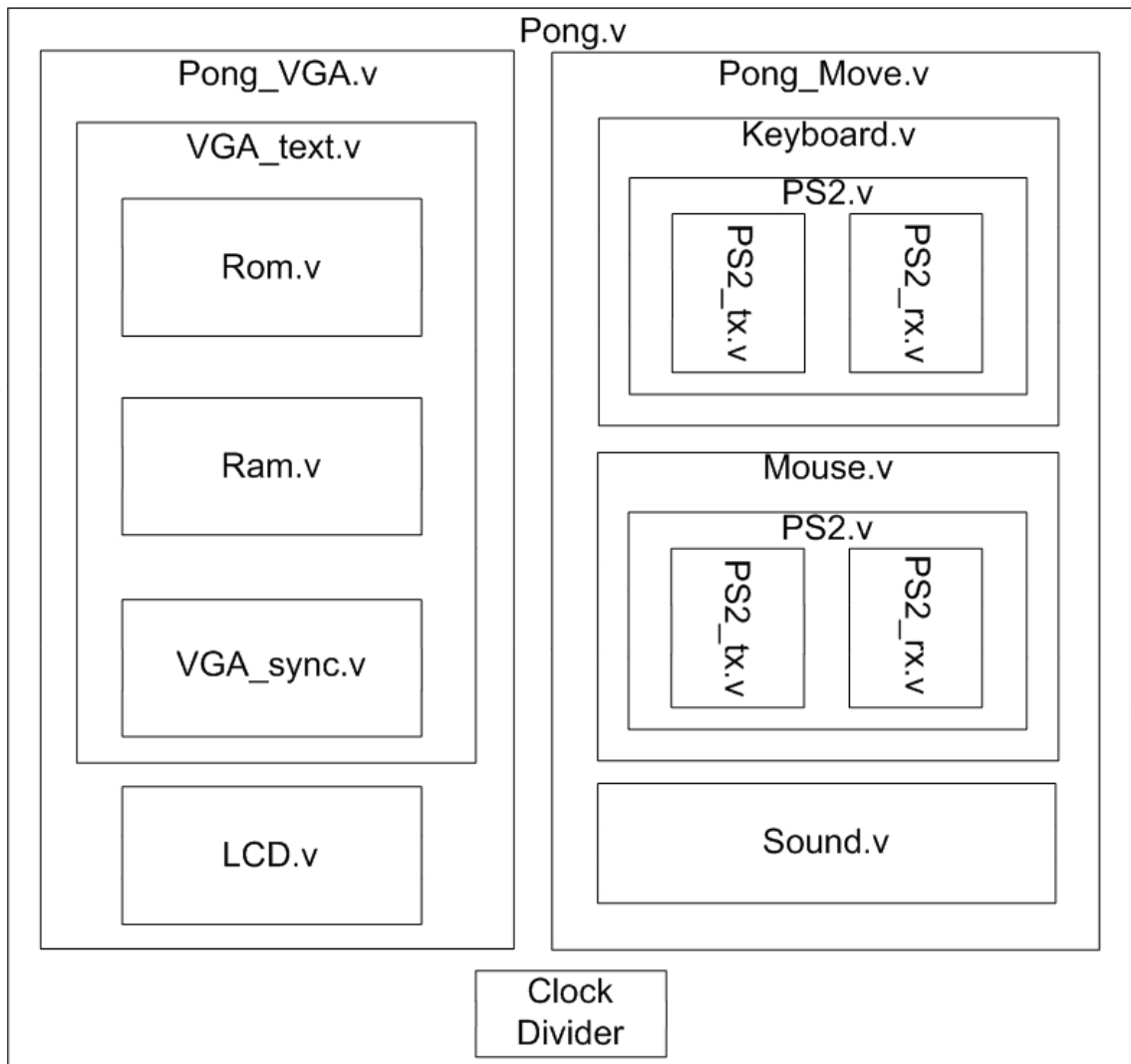


Figura 3.56 - Hierarquia dos módulos que implementam a aplicação “Pong”.

Como o jogo é destinado a duas pessoas, surgiu a ideia de criar um segundo porto PS/2, ligado a uma porta de expansão da plataforma de desenvolvimento. Assim seria possível integrar dois teclados, ou dois ratos, ou um rato e um teclado. Por uma questão de aplicar mais módulos diferentes, foi escolhido utilizar um rato e um teclado, sendo o rato ligado ao novo porto e o teclado ao conector, já existente na plataforma de desenvolvimento.

Ao construir esse novo conector surgiu uma dificuldade, devido às diferenças de voltagem entre a porta de expansão da plataforma de desenvolvimento e as necessárias para o protocolo PS/2. Devido a esse facto foi necessário aplicar um conversor de voltagem de linha, que é utilizado de uma forma mais regular nas comunicações, usando o protocolo I²C.

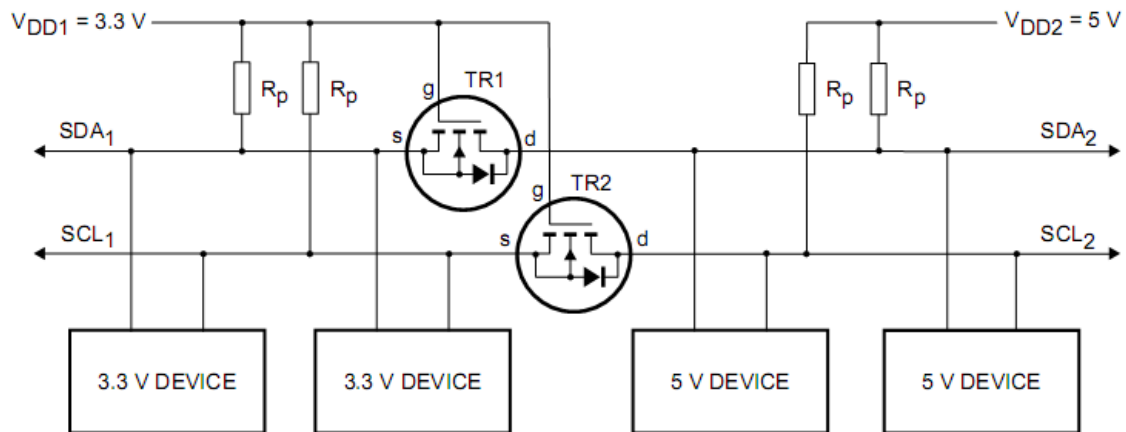


Figura 3.57 - Esquemático do conversor de linha implementado [14].

Para a alimentação de 5 Volts recorreu-se a um transformador, que fornece essa voltagem, para ligar as resistências de pull-up do lado do conector PS/2 e para alimentar o próprio dispositivo PS/2.

O conversor de linha foi construído e ligado à porta de expansão designada “J2”.

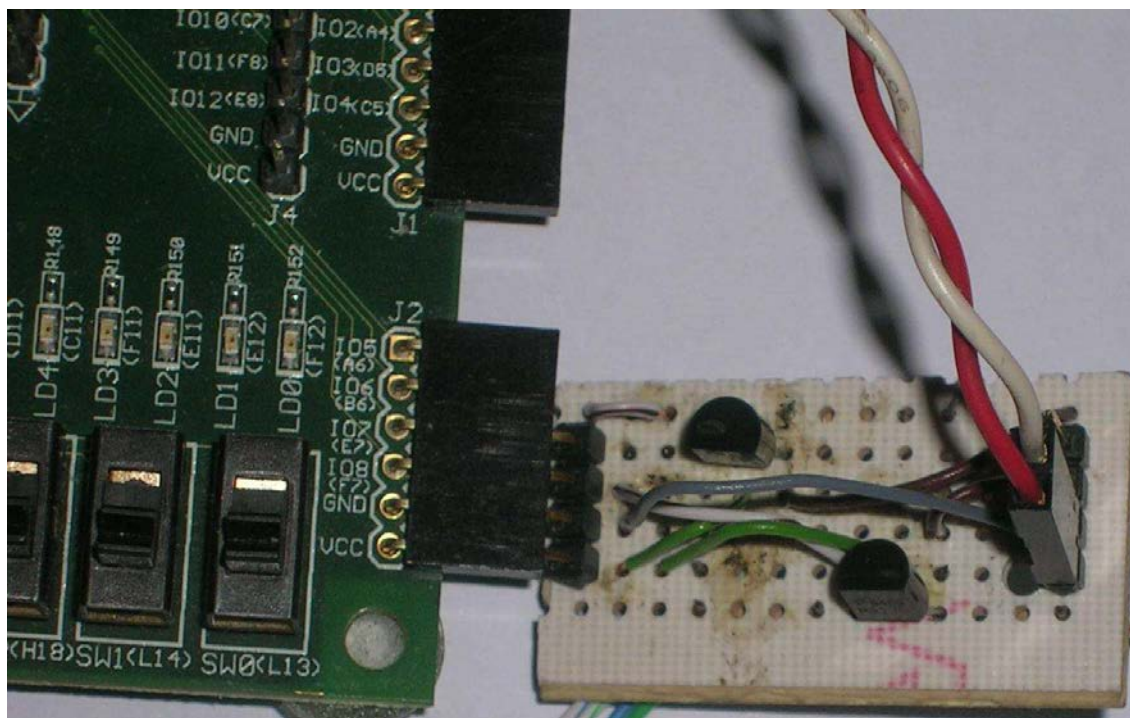


Figura 3.58 - Rversor de linha ligado à plataforma de desenvolvimento.

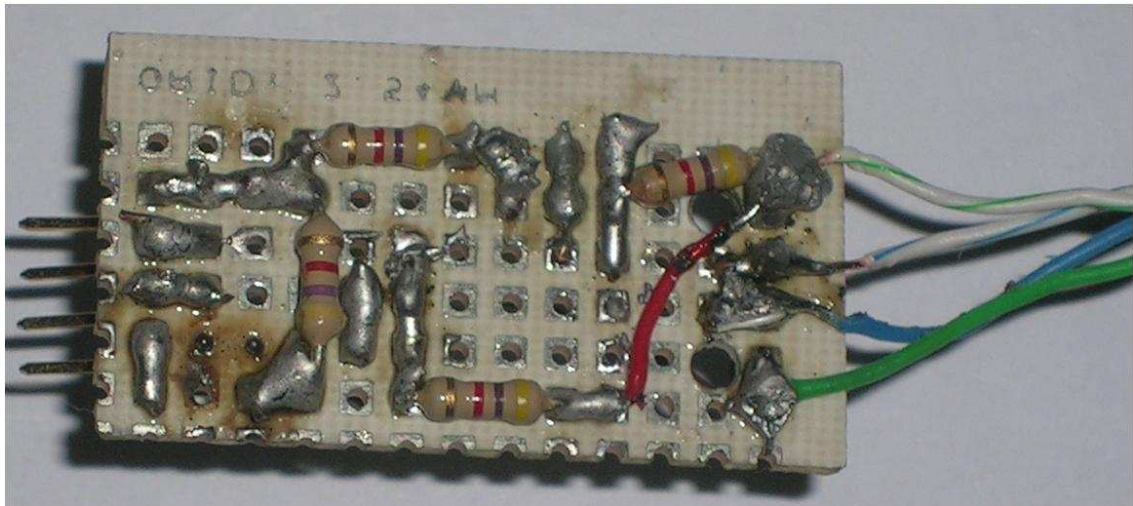


Figura 3.59 - Vista da parte inferior do conversor de linha.

Como este conversor de voltagem de linha funciona de forma igual para os dois sinais (relógio e dados) do protocolo PS/2. O conector PS/2 e os fios a ele ligados definiram as posições das linhas ficando ligado à linha IO7 (E7) da plataforma de desenvolvimento o sinal de dados e à linha IO8 (F7) o sinal de relógio.

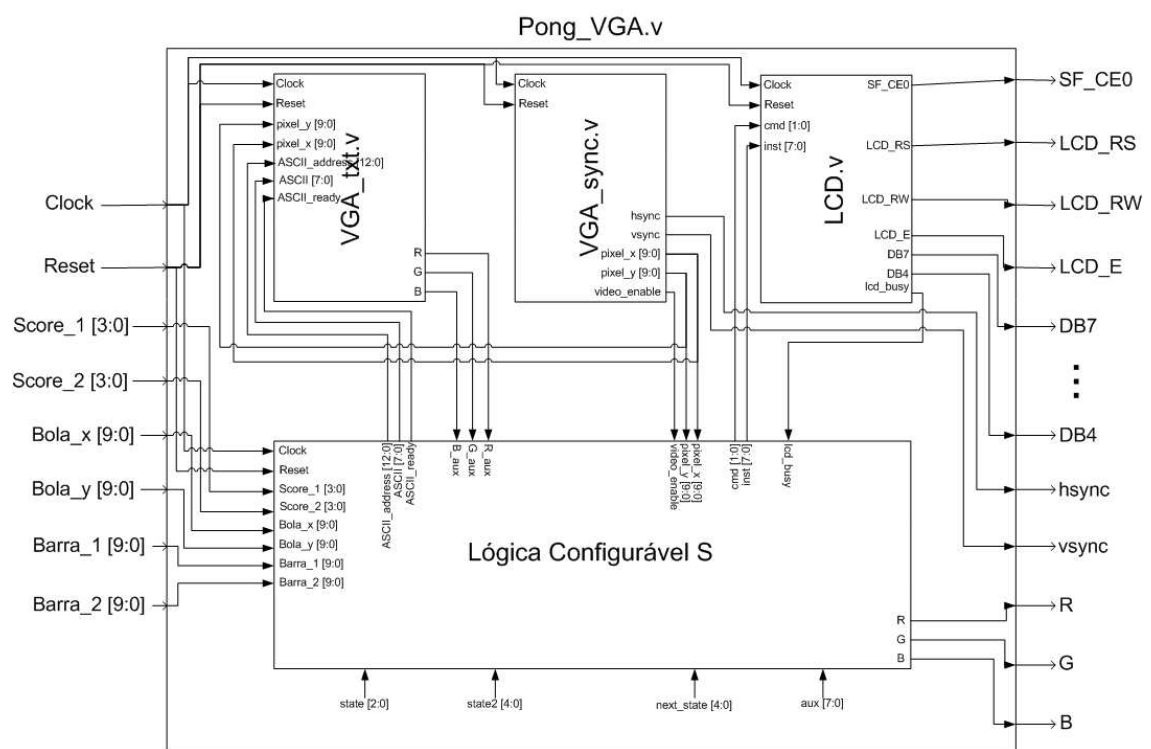


Figura 3.60 - Módulo “Pong_VGA.v”.

Este módulo controla todos os aspectos gráficos do jogo. Inicialmente e de forma a integrar o modo gráfico e de texto do protocolo VGA, tiveram que se definir áreas para cada um destes modos. Sendo assim, a área designada para o modo de texto ficou limitada aos extremos da área visível do ecrã, permitindo a impressão de caracteres na primeira e última

coluna, bem como na primeira e última linha. Assim, permite-se a utilização do modo de texto do protocolo VGA sem a redução drástica da área de jogo.

Para os jogadores saberem os limites da área de jogo foi criado um rectângulo delimitador da mesma, sendo que a linha que o define possui oito pixéis de largura.

Foram também definidos outros parâmetros do jogo, tais como a largura e coordenadas horizontais das barras dos jogadores, o diâmetro da bola, o comprimento e altura dos tijolos, bem como a sua localização na área de jogo.

Todos os sinais de entrada deste módulo, excepto “Clock” e “Reset”, são provenientes do módulo que controla os movimentos do jogo.

O sinal “Score_1” indica a pontuação do jogador 1 e o sinal “Score” indica a pontuação do jogador 2.

Os sinais “Bola_x” e “Bola_y” indicam, respectivamente, as coordenadas x e y da bola, uma vez que este objecto se movimenta no decorrer do jogo.

Durante o jogo as barras dos jogadores também se movimentam, mas apenas verticalmente, sendo as suas coordenadas y transmitidas através dos sinais de entrada “Barra_1” e “Barra_2”.

Os sinais de saída “SF_CE0”, “LCD_RS”, “LCD_RW”, “LCD_E”, “DB7”, “DB6”, “DB5” e “DB4” realizam a interface deste módulo com o LCD da plataforma de desenvolvimento, e devem de ser ligados aos respectivos recursos da forma já especificada na secção destinada à descrição do módulo “LCD.v”.

Os sinais “hsync”, “vsync”, “R”, “G” e “B” efectuem a ligação entre o jogo e o conector VGA da plataforma de desenvolvimento, permitindo o sincronismo e a visualização do jogo num monitor utilizando o protocolo VGA. Estes sinais devem de ser ligados aos respectivos recursos da forma descrita na secção destinada à descrição do módulo “VGA_sync.v”.

Internamente neste módulo estão integrados os módulos, já descritos anteriormente, “VGA.v” (responsável pelo modo texto do protocolo VGA, ligeiramente alterado), “VGA_sync.v” (responsável pela sincronização da FPGA com o monitor, e indicação das coordenadas x e y do pixel actualmente a ser processado) e o módulo “LCD.v” (responsável pelo controle do LCD da plataforma de desenvolvimento).

De forma a controlar o módulo “LCD.v” foi criada uma máquina de estados, controlada por “state2” e “next_state”, com 27 estados definidos. Estes estados definidos têm como objectivo escrever no LCD (utilizando os registos “cmd” e “inst”) a pontuação de ambos os jogadores, sendo estes valores actualizados, ciclicamente. O estado 0 é responsável pelo envio do comando de limpar o ecrã do LCD, passando ao estado 1. O estado 1 é um estado muito particular, pois é usado sempre que se envia uma instrução ao LCD. Este estado espera pela conclusão dessa operação, indicada pela descida ao valor lógico ‘0’ do sinal “lcd_busy”, e quando isto acontece a FSM passa para o estado guardado em “next_state”. Assim, evita-se a replicação sucessiva deste estado, minimizando o número de estados necessários. Os estados 2, 3, 4, 5, 6, 7, 8, 9, 10 e 11 são utilizados para instruir a escrita de cada um dos caracteres presentes em “Player 1: ” (incluindo os espaços), regressando após cada um desses estados ao estado 1, indicando em “next_state” qual o próximo estado a visitar. O estado 12 é utilizado para mudar o cursor para o primeiro carácter da segunda linha, visitando de seguida o estado 1 e passando para o estado 13 após a operação ter sido completada. Os estados 13, 14, 15, 16, 17, 18, 19, 20, 21 e 22 são utilizados na escrita de cada um dos caracteres de “Player 2: ” (incluindo os espaços), regressando após cada um desses estados ao estado 1, indicando em “next_state” qual o próximo estado a visitar. Todos os estados

entre o estado 0 e 22, excluindo o estado 1, só são visitados uma vez após cada inicialização, pois representam a escrita de informação que não será alterada. No estado 23 é instruído ao LCD para colocar o cursor no décimo primeiro carácter da primeira linha, visitando de seguida o estado 1, passando para o estado 24. No estado 24 é ordenada a escrita da pontuação actual do jogador 1. Após nova passagem pelo estado 1, visita-se o estado 25 onde se ordena a mudança do cursor para o décimo primeiro carácter da segunda linha. Volta-se novamente ao estado 1 e de seguida ao estado 26, e último, onde se escreve a pontuação actual do jogador 2, passando ao estado 1 e regressando ao estado 23. Com esta máquina de estados assegura-se que a informação disponível no LCD da plataforma de desenvolvimento está sempre actualizada.

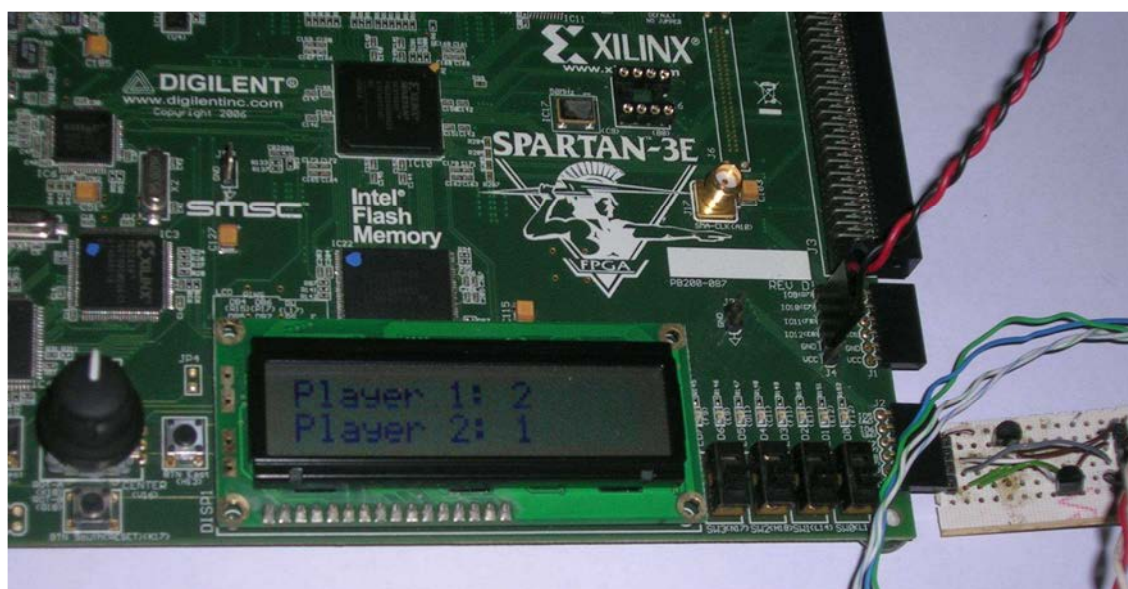


Figura 3.61 - Aspecto do LCD durante o jogo “Pong”.

Para actualizar a informação relativa ao jogo, disponibilizada pelo modo de texto do protocolo VGA, é utilizada outra máquina de estados, controlada por “state”, com 4 estados definidos. No estado 0 ordena-se a escrita da pontuação actual do jogador 1 na respectiva posição da memória RAM do controlador de texto, utilizando os registos “ASCII_ready”, “ASCII_address” e “ASCII”, passando para o estado 1. Neste estado o registo “ASCII_ready” é colocado a ‘0’ e passa-se de seguida ao estado 2 onde se ordena a escrita da pontuação actual do jogador 2. De seguida é visitado o estado 3, que tem a mesma função do estado 1, regressando de seguida ao estado 0.

O controlador de texto do protocolo VGA utilizado é ligeiramente diferente do descrito anteriormente. As únicas diferenças entre ambos devem-se à inicialização da memória RAM do controlador e da utilização do módulo de sincronismo “VGA_sync.v” externamente ao controlador. Desta forma o módulo “VGA.v” aqui utilizado possui mais duas entradas para receber as coordenadas x e y do pixel actualmente a ser processado. A memória RAM é inicializada a partir de um ficheiro de texto designado “ram.txt”, que é constituído por 8192 linhas com 8 dígitos binários, de forma a escrever na memória RAM todos os caracteres que não serão alterados durante o jogo. Isto evita a criação de uma máquina de estados de gestão do modo de texto muito maior, e com muitos estados que apenas seriam visitados uma vez, após cada inicialização.

A lógica interna do módulo “Pong_VGA.v” é responsável por definir as áreas destinadas ao modo de texto e modo gráfico do protocolo VGA. Também é responsável pelo desenho de todos os objectos do jogo (bola, barras e margens).

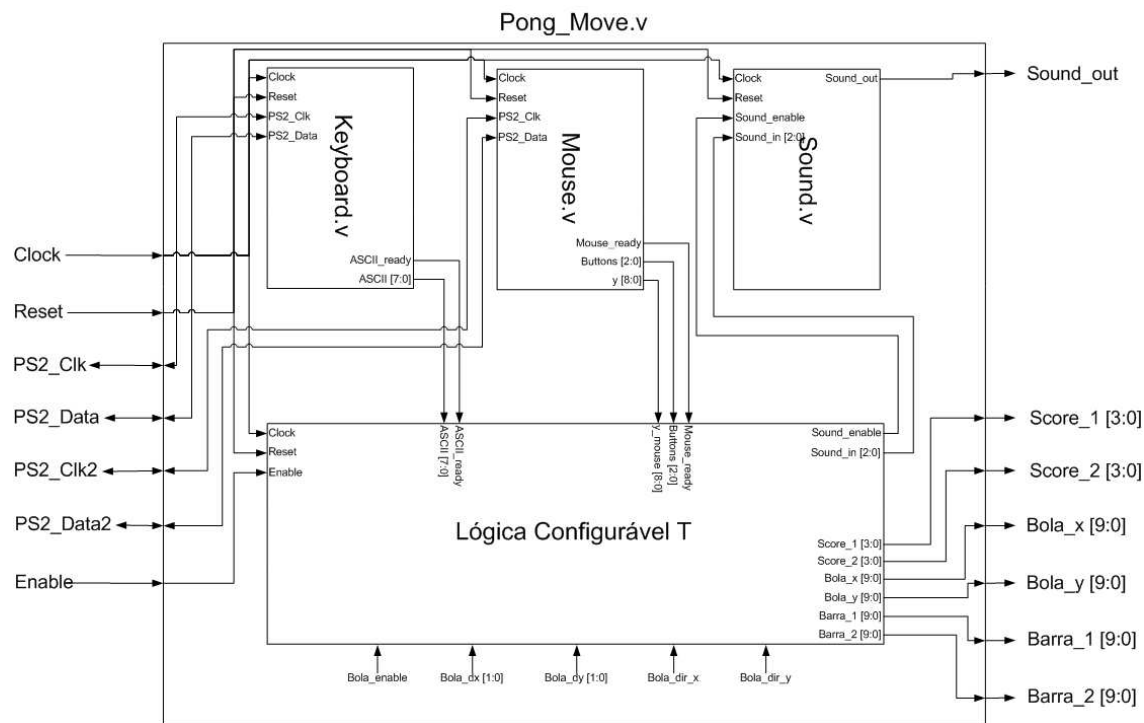


Figura 3.62 - Módulo “Pong_Move.v”.

Este módulo controla os movimentos dos objectos presentes no jogo, bem como a interação entre eles.

Internamente no módulo estão integrados os módulos, descritos anteriormente, “Keyboard.v”, “Mouse.v” e “Sound.v”, que são responsáveis pela comunicação com o teclado e rato PS/2 e pela reprodução de som.

As entradas “PS2_Clk” e “PS2_Data” representam as linhas de relógio e de dados do barramento PS/2, devendo ser ligadas aos respectivos recursos da forma descrita na secção que aborda os módulos que implementam o protocolo PS/2 e ao módulo de controlo do teclado.

As entradas “PS2_Clk2” e “PS2_Data2” representam as linhas de relógio e de dados do barramento PS/2 construído e devem ser ligadas aos respectivos recursos de forma semelhante às ligações anteriores, mudando apenas o nome do fio e a localização do recurso, e ao módulo de controlo do rato.

A entrada “Enable” está ligada a um divisor de relógio no módulo superior, que serve para abrandar o processamento de dados, tornando a velocidade de funcionamento da aplicação razoável para o utilizador comum.

Todas as saídas deste módulo devem de ser ligadas ao módulo “Pong_VGA.v”, excepto a saída “Sound_out” que deverá ser ligada ao pólo positivo do altifalante, tal como descrito na secção que abrange o módulo “Sound.v”. Os sinais “Score_1” e “Score_2” indicam respectivamente a pontuação actual de cada jogador. As saídas “Bola_x” e “Bola_y” indicam as coordenadas x e y da bola e os sinais “Barra_1” e “Barra_2” indicam as coordenadas y das barras dos jogadores.

O controlo da barra do jogador 1 é realizado através de um teclado PS/2 ligado directamente à plataforma de desenvolvimento, utilizando apenas 3 teclas. Para iniciar/pausar o jogo basta premir a tecla de espaço do teclado. Para mover a barra do jogador para cima e para baixo, basta premir as teclas dos caracteres “W” e “S”, respectivamente.

O controlo da barra do jogador 2 é realizado através de um rato PS/2 ligado ao conector PS/2 construído, utilizando apenas um botão e o movimento vertical do rato. Para iniciar/pausar o jogo basta premir o botão direito do rato. Para mover a barra do jogador para cima e para baixo, mover o rato verticalmente no sentido desejado. Enquanto o jogo estiver em pausa, não é possível mover as barras dos jogadores, pois isso levaria a situações de batota.

O módulo de som está integrado neste módulo de modo a facilitar a sua integração, uma vez que os sinais sonoros são originados pelas interacções entre objectos. No decorrer do jogo são reproduzidos dois sons diferentes, utilizando para esse efeito os sinais “Sound_enable” e “Sound_in”. Um para quando a bola bate nas margens de jogo superior e inferior ou nas barras dos jogadores e outro diferente quando bate nas margens laterais, sinalizando a perda de uma vida.

O registo interno “Bola_enable” indica se o jogo está activo ou em pausa e serve para activar e desactivar todos os movimentos.

O movimento da bola é definido por quatro registos “Bola_dir_x”, “Bola_dir_y”, “Bola_delta_x” e “Bola_delta_y”. O primeiro indica o sentido do movimento da bola no eixo x e o segundo indica o sentido do movimento da bola no eixo y. Se o primeiro for igual a ‘0’ a bola está a movimentar-se para a esquerda, se for igual a ‘1’ a bola está a movimentar-se para a direita. Se o segundo for igual a ‘0’ a bola está a movimentar-se para cima, se for igual a ‘1’ a bola está a movimentar-se para baixo. Os outros dois registos indicam a variação das coordenadas x e y da bola, em pixéis, a cada activação do sinal de entrada “Enable”.

Os movimentos da bola são controlados utilizando condições *if/else* não permitindo que a bola ultrapasse a margem delimitadora da área de jogo.

A interacção entre a bola e as barras dos jogadores é feita através da utilização de condições *if/else* possibilitando três tipos diferentes de contacto, aumentando o realismo e complexidade do jogo. Quando a bola toca nos extremos da barra o seu movimento será mais rápido no eixo x do que no eixo y (utilizando para isso os registos “Bola_delta_x” e “Bola_delta_y”) e os dois sentidos invertidos. Se a bola tocar entre os extremos e um quarto do seu comprimento total, a velocidade de deslocamento será igual nos dois eixos sendo apenas invertido o sentido de movimento no eixo y. Se a bola tocar no centro da barra, a bola movimentar-se-á mais rapidamente no eixo y do que no eixo x, sendo apenas invertido o sentido de movimento no eixo y.

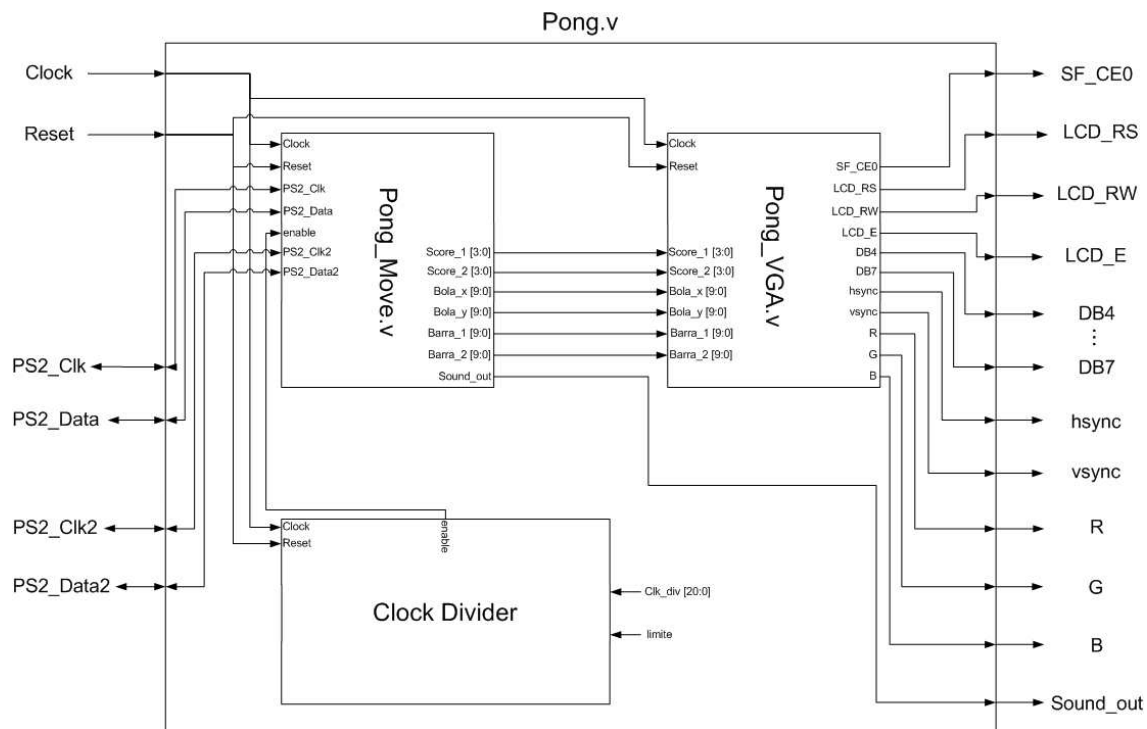


Figura 3.63 - Módulo “Pong.v”.

O módulo “Pong.v” efectua as ligações entre os dois módulos inferiores “Pong_Move.v” e “Pong_VGA.v”, bem como a ligação dos sinais de saídas destes que tem como destino os recursos da plataforma de desenvolvimento.

O módulo possui ainda um divisor de relógio que controla as activações de “enable” e regula a dificuldade do jogo. O parâmetro “limite” define o número de ciclos de relógio entre activações de “enable” e o registo “Clk_div” é incrementado a cada subida de relógio, de forma a contar o número de ciclos de relógio que já passaram desde a última activação. O valor pré-definido de “limite” indica um milhão de ciclos de relógio entre activações de “enable”, podendo este valor ser alterado para aumentar ou reduzir a dificuldade do jogo.

Está disponível no manual desta aplicação, que se encontra na página da Internet desta dissertação [18], uma ligação para um vídeo que mostra a aplicação em funcionamento.



Figura 3.64 - Aspecto do monitor VGA durante a aplicação “Pong”.

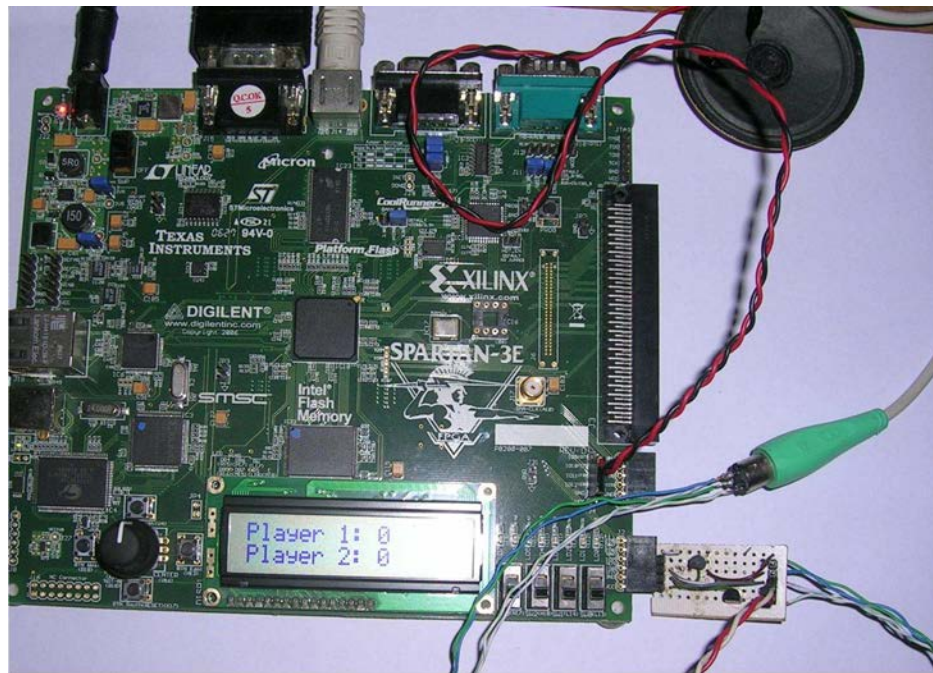


Figura 3.65 - Montagem da plataforma de desenvolvimento para a aplicação “Pong”.

3.5 - Criação dos Manuais

Juntamente com a criação da biblioteca de módulos em Verilog, era requerido também, para o trabalho desta dissertação, a criação de documentação relativa à utilização e constituição dos módulos criados. Estes documentos produzidos têm como objectivo facilitar a integração dos módulos desenvolvidos em projectos em desenvolvimento, fornecendo informação acerca dos sinais de entrada e suas funções, esquemas da organização dos módulos e explicação da construção dos mesmos. Para além dessas informações foram

também disponibilizados pequenos módulos exemplo, que podem servir como teste à inserção desses módulos noutros projectos.

Todos os módulos possuem um manual, estando estes disponíveis para consulta no endereço de Internet destinado à divulgação dos trabalhos desta dissertação [18], sendo também disponibilizados neste documento no anexo A.

A estrutura dos manuais foi definida em conjunto com o orientador desta dissertação, o professor doutor Hélio Mendes de Sousa Mendonça, tendo como base os manuais dos módulos disponibilizados pela *Xilinx*. Os manuais contêm inicialmente um esquemático de ligações, juntamente com uma pequena descrição dos sinais envolvidos. Posteriormente os detalhes sobre a construção e funcionamento dos módulos são divulgados, estando estes disponíveis para serem descarregados no final da página, na secção de “Anexos”, juntamente com um pequeno exemplo de utilização.

Durante a criação dos manuais houve a preocupação de criar um documento que pudesse ser consultado de forma rápida para esclarecer uma dúvida, e ao mesmo tempo disponibilizar claramente todas as informações necessárias, para que mesmo os alunos menos experientes com um pouco de leitura, possam entender perfeitamente o funcionamento destes módulos desenvolvidos.

Capítulo 4

Resultados

Neste capítulo serão apresentados os resultados obtidos com a implementação dos módulos na plataforma de desenvolvimento. Serão analisados os resultados de ocupação de recursos da FPGA, de forma a ter conhecimento do *overhead* de recursos necessários para a implementação de um determinado módulo. Todos os dados aqui obtidos, são referentes aos módulos presentes na página da Internet desta dissertação [18], estando disponíveis para *download*. Estes dados foram retirados da ferramenta de desenvolvimento da *Xilinx*, utilizada no decorrer do trabalho desta dissertação. Todas as restrições temporais em todos os módulos foram cumpridas, embora estas não sejam preocupantes em quase todos os módulos, pois o processamento de dados necessário a cada um dos protocolos implementados são facilmente realizados utilizando a frequência do relógio interno da plataforma de desenvolvimento.

Os dados sobre os recursos disponíveis da FPGA *Spartan-3E* são apresentados na seguinte tabela.

Tabela 4.1 - Recursos disponíveis na FPGA *Spartan-3E*.

Recurso	Disponibilidade
<i>Total Number of Slice Flip-Flops</i>	9312
<i>Total Number of 4 input LUTs</i>	9312
<i>Total Number of Slices</i>	4656
<i>Number of Bounded IOBs</i>	232
<i>Number of BUFGMUXs</i>	24
<i>Number of RAMB16s</i>	20
<i>Number of MULTI18X18SIOs</i>	20

4.1 - Botões e LEDs

O módulo “Buttons.v” destinado a efectuar o unbounding dos botões de pressão e rotativo, bem como tornar a interface com os LEDs mais apelativa, ocupa poucos recursos da FPGA, pois apesar de efectuar uma função muito importante para a utilização destas interfaces, é um módulo de relativa simplicidade.

Tabela 4.2 — Ocupação de recursos do módulo “Buttons.v”.

Recurso	Utilizados	Disponíveis	Percentagem Ocupada
<i>Total Number of Slice Flip-Flops</i>	102	9312	1.09%
<i>Total Number of 4 input LUTs</i>	84	9312	0.90%
<i>Number Used as Logic</i>	49		
<i>Number Used as Route-Thru</i>	35		
<i>Total Number of Occupied Slices</i>	86	4656	1.85%
<i>Number of Bounded IOBs</i>	31	232	13.36%
<i>Number of BUFGMUXs</i>	1	24	4.17%

De referir ainda que todas as *Slices* contém apenas *related logic*. O *fan-out* médio para os sinais, que não o relógio, obtido foi de 2.28.

4.2 - LCD

Para módulo “LCD.v”, destinado ao controlo do LCD da plataforma de desenvolvimento, foi previsto um aumento dos recursos utilizados em relação ao módulo anterior, devido à lógica mais complexa do módulo. No entanto, isto apenas aconteceu parcialmente, pois o número de LUTs utilizadas foi superior, mas o número de *Flip-Flops* utilizados foi menor, o que levou a uma redução no número de *Slices* ocupadas.

Tabela 4.3 – Ocupação de recursos do módulo “LCD.v”.

Recurso	Utilizados	Disponíveis	Percentagem Ocupada
<i>Total Number of Slice Flip-Flops</i>	50	9312	0.54%
<i>Total Number of 4 input LUTs</i>	110	9312	1.18%
<i>Number Used as Logic</i>	85		
<i>Number Used as Route-Thru</i>	25		
<i>Total Number of Occupied Slices</i>	63	4656	1.35%
<i>Number of Bounded IOBs</i>	21	232	9.05%
<i>Number of BUFGMUXs</i>	1	24	4.17%

Novamente, todas as *Slices* contêm apenas *related logic*. O *fan-out* médio para os sinais, que não o relógio, obtido foi de 2.83.

4.3 - Teclado

Os resultados do módulo “Keyboard.v” e os módulos integrados no mesmo são de maior relevância, em relação aos dois anteriores, pois é um módulo de complexidade superior. Integrados neste módulo estão os módulos que permitem a comunicação PS/2.

Tabela 4.4 – Ocupação de recursos dos módulos do teclado PS/2.

Recurso	Utilizados	Disponíveis	Percentagem Ocupada
<i>Total Number of Slice Registers</i>	73	9312	0.78%
<i>Total Number of 4 input LUTs</i>	278	9312	2.99%
<i>Number Used as Logic</i>	277		
<i>Number Used as Route-Thru</i>	1		
<i>Total Number of Occupied Slices</i>	157	4656	3.37%
<i>Number of Bounded IOBs</i>	13	232	5.60%
<i>Number of BUFGMUXs</i>	1	24	4.17%

De referir que nestes módulos 4 *slice registers* são usados como *Latches* sendo os restantes utilizados como *Flip-Flops*.

Também de referir que nove das treze *Bounded IOBs* são *IOB Latches*. Sendo ainda o *fan-out* médio para os sinais de 3.72.

Como podemos constatar, estes módulos apenas ocupam 3% das *Slices* da FPGA, uma percentagem muito reduzida para as vantagens que a ligação a um teclado permite.

4.4 - Rato

Tendo em conta que os módulos de interface com o rato são parcialmente partilhados com os módulos utilizados na interface do teclado (apenas os módulos de topo de hierarquia diferem), seria plausível que os resultados da implementação fossem semelhantes. No entanto o funcionamento mais complexo do teclado, que obriga à descodificação dos *make* e *break codes*, ocupa mais recursos da FPGA do que os módulos destinados ao funcionamento do rato PS/2.

Tabela 4.5 – Ocupação de recursos dos módulos do rato PS/2.

Recurso	Utilizados	Disponíveis	Percentagem Ocupada
<i>Total Number of Slice Flip-Flops</i>	82	9312	0.88%
<i>Total Number of 4 input LUTs</i>	136	9312	1.45%
<i>Number Used as Logic</i>	135		
<i>Number Used as Route-Thru</i>	1		
<i>Total Number of Occupied Slices</i>	80	4656	1.72%
<i>Number of Bounded IOBs</i>	26	232	11.21%
<i>Number of BUFGMUXs</i>	1	24	4.17%

Novamente todas as *Slices* contêm apenas *related logic*. O *fan-out* médio para os sinais obtido foi de 3.94.

4.5 - UART

Para os módulos que controlam a comunicação série através dos protocolos UART e RS-232, o resultado não era facilmente previsível, pois não existe nenhum outro agrupamento de módulos cuja complexidade seja equivalente. O grupo de módulos que mais se aproxima será certamente o dos utilizados na interface com o rato, pois as máquinas de estados de envio e receção de dados são semelhantes em complexidade, sem no entanto haver mecanismos de cálculo de bits de paridade. O módulo de topo de hierarquia do rato também exige maior processamento de dados, o que origina uma maior ocupação de recursos. Assim, previa-se que os recursos ocupados pelos módulos de comunicação série fossem ligeiramente inferiores aos do rato.

Tabela 4.6 — Ocupação de recursos dos módulos de comunicação série.

Recurso	Utilizados	Disponíveis	Percentagem Ocupada
<i>Total Number of Slice Flip-Flops</i>	44	9312	0.47%
<i>Total Number of 4 input LUTs</i>	121	9312	1.30%
<i>Number Used as Logic</i>	114		
<i>Number Used as Route-Thru</i>	7		
<i>Total Number of Occupied Slices</i>	65	4656	1.40%
<i>Number of Bounded IOBs</i>	23	232	9.91%
<i>Number of BUFGMUXs</i>	1	24	4.17%

De referir que todas as *Slices* contêm apenas *related logic*. O *fan-out* médio para os sinais obtido foi de 3.81.

Tal como foi analisado, estes módulos ocupam ligeiramente menos recursos que os módulos utilizados na implementação do rato PS/2, e utilizando apenas 65 *Slices* é possível obter comunicação *full-duplex* assíncrona entre dois dispositivos.

4.6 - VGA (Modo Gráfico)

O módulo “VGA_sync.v”, responsável pelo sincronismo do monitor VGA é um módulo cujo funcionamento é bastante simples, pelo que não necessitará da utilização de muitos recursos no seu funcionamento.

Tabela 4.7 — Ocupação de recursos do módulo “VGA_sync.v”.

Recurso	Utilizados	Disponíveis	Percentagem Ocupada
<i>Total Number of Slice Flip-Flops</i>	23	9312	0.25%
<i>Total Number of 4 input LUTs</i>	56	9312	0.60%
<i>Number Used as Logic</i>	38		
<i>Number Used as Route-Thru</i>	18		
<i>Total Number of Occupied Slices</i>	31	4656	0.67%
<i>Number of Bounded IOBs</i>	26	232	11.21%
<i>Number of BUFGMUXs</i>	1	24	4.17%

De referir que todas as *Slices* contêm apenas *related logic*. O *fan-out* médio para os sinais obtido foi de 3.21.

4.7 - VGA (Modo Texto)

Este conjunto de módulos necessitará certamente de mais recursos do que o anterior, uma vez que o módulo de sincronismo está integrado neste conjunto. O recurso cuja taxa de ocupação é mais preocupante é sem dúvida a memória RAM, pois é um recurso que é utilizado de uma forma massiva, para armazenar o código dos caracteres em cada posição do ecrã.

Tabela 4.8 – Ocupação de recursos dos módulos do modo de texto em VGA.

Recurso	Utilizados	Disponíveis	Percentagem Ocupada
<i>Total Number of Slice Flip-Flops</i>	23	9312	0.25%
<i>Total Number of 4 input LUTs</i>	74	9312	0.79%
<i>Number Used as Logic</i>	50		
<i>Number Used as Route-Thru</i>	24		
<i>Total Number of Occupied Slices</i>	40	4656	0.86%
<i>Number of Bounded IOBs</i>	29	232	12.50%
<i>Number of BUFGMUXs</i>	1	24	4.17%
<i>Number of RAMB16s</i>	5	20	25.00%
<i>Number of MULTI18X18SIOs</i>	1	20	5.00%

Novamente todas as *Slices* contêm apenas *related logic*. O *fan-out* médio para os sinais obtido foi de 2.66.

4.8 - SVGA (Modo Gráfico)

Neste módulo é interessante comparar os resultados com os obtidos utilizando o protocolo VGA, uma vez que o módulo “SVGA_sync.v” funciona de forma semelhante ao módulo utilizado nesse protocolo. Será interessante ver a variação de recursos na introdução de um módulo que permite uma maior resolução.

Tabela 4.9 – Ocupação de recursos do módulo “SVGA_sync.v”.

Recurso	Utilizados	Disponíveis	Percentagem Ocupada
<i>Total Number of Slice Flip-Flops</i>	26 (+3)	9312	0.28%
<i>Total Number of 4 input LUTs</i>	69 (+13)	9312	0.74%
<i>Number Used as Logic</i>	50		
<i>Number Used as Route-Thru</i>	19		
<i>Total Number of Occupied Slices</i>	40 (+9)	4656	0.86%
<i>Number of Bounded IOBs</i>	10	232	4.31%
<i>Number of BUFGMUXs</i>	1	24	4.17%

Como podemos ver na tabela, onde está representada a diferença para o módulo anterior dentro de parêntesis, os recursos necessários para aumentar de uma resolução de 640x480 para 800x600 são muito poucos, ocupando, por exemplo, apenas mais nove *Slices*.

Foi obtido um *fan-out* médio para os sinais de 2.92.

4.9 - SVGA (Modo Texto)

Novamente neste módulo a comparação com o protocolo VGA assume importância, de forma a verificar se o aumento da resolução e do número de caracteres justifica o aumento de recursos ocupados.

Tabela 4.10 – Ocupação de recursos dos módulos do modo de texto em SVGA.

Recurso	Utilizados	Disponíveis	Percentagem Ocupada
<i>Total Number of Slice Flip-Flops</i>	45 (+22)	9312	0.48%
<i>Total Number of 4 input LUTs</i>	101 (+26)	9312	1.08%
<i>Number Used as Logic</i>	63 (+13)		
<i>Number Used as Route-Thru</i>	37 (+13)		
<i>Total Number of Occupied Slices</i>	54 (+14)	4656	1.16%
<i>Number of Bounded IOBs</i>	9	232	3.88%
<i>Number of BUFGMUXs</i>	1	24	4.17%
<i>Number of RAMB16s</i>	5	20	25.00%
<i>Number of MULTI18X18SIOs</i>	1	20	5.00%

Como se pode ver o aumento da utilização de recursos é maior no modo texto do que no modo gráfico, ao passar do protocolo VGA para SVGA.

Foi obtido um *fan-out* médio para os sinais de 2.92.

4.10 - VGA e SVGA (Modo Gráfico)

Para estes módulos foram calculadas estatísticas em três condições diferentes, a primeira a escolha do protocolo a utilizar podia ser alterada durante o funcionamento da plataforma de desenvolvimento. Os outros dois conjuntos de dados estatísticos reflectem a utilização de recursos quando um dos protocolos é definido durante o projecto e não é possível alternar entre ambos durante o funcionamento.

Tabela 4.11 — Ocupação de recursos dos módulos que implementam VGA + SVGA (Gráfico).

Recurso	Utilizados	Disponíveis	Percentagem Ocupada
<i>Total Number of Slice Flip-Flops</i>	49	9312	0.53%
<i>Total Number of 4 input LUTs</i>	126	9312	1.35%
<i>Number Used as Logic</i>	89		
<i>Number Used as Route-Thru</i>	37		
<i>Total Number of Occupied Slices</i>	71	4656	1.52%
<i>Number of Bounded IOBs</i>	11	232	4.74%
<i>Number of BUFGMUXs</i>	1	24	4.17%

Foi obtido um *fan-out* médio para os sinais de 2.90.

Como podemos ver pelos resultados da tabela anterior, se ambos os módulos de sincronismo estiverem disponíveis, os recursos utilizados na sua implementação são semelhantes à soma dos recursos utilizados individualmente pelos módulos de sincronismo.

Tabela 4.12 — Ocupação de recursos dos módulos que implementam VGA + SVGA (Gráfico), estando fixo a VGA.

Recurso	Utilizados	Disponíveis	Percentagem Ocupada
<i>Total Number of Slice Flip-Flops</i>	26	9312	0.28%
<i>Total Number of 4 input LUTs</i>	56	9312	0.60%
<i>Number Used as Logic</i>	38		
<i>Number Used as Route-Thru</i>	18		
<i>Total Number of Occupied Slices</i>	34	4656	0.73%
<i>Number of Bounded IOBs</i>	11	232	4.74%
<i>Number of BUFGMUXs</i>	1	24	4.17%

Foi obtido um *fan-out* médio para os sinais de 2.73.

Ao implementar somente o protocolo VGA neste módulo, obtemos resultados muito semelhantes à utilização de apenas “VGA_sync.v”. O número de IOBs é reduzido, aumentando apenas a utilização de *Slices*, passando de 31 para 34.

Tabela 4.13 – Ocupação de recursos dos módulos que implementam VGA + SVGA (Gráfico), estando fixo a SVGA.

Recurso	Utilizados	Disponíveis	Porcentagem Ocupada
<i>Total Number of Slice Flip-Flops</i>	26	9312	0.28%
<i>Total Number of 4 input LUTs</i>	69	9312	0.74%
<i>Number Used as Logic</i>	50		
<i>Number Used as Route-Thru</i>	19		
<i>Total Number of Occupied Slices</i>	40	4656	0.86%
<i>Number of Bounded IOBs</i>	11	232	4.74%
<i>Number of BUFGMUXs</i>	1	24	4.17%

Foi obtido um *fan-out* médio para os sinais de 2.88.

Ao implementar somente o protocolo SVGA neste módulo, obtemos resultados muito semelhantes à utilização de apenas “SVGA_sync.v”. Apenas o número de IOBs é incrementado de uma unidade.

4.11 - VGA e SVGA (Modo Texto)

Para estes módulos foram também calculadas estatísticas em três condições diferentes, na primeira a escolha do protocolo a utilizar podia ser alterada durante o funcionamento da plataforma de desenvolvimento. Os outros dois conjuntos de dados estatísticos reflectem a utilização de recursos quando um dos protocolos é definido durante o projecto e não é possível alternar entre ambos durante o funcionamento.

Tabela 4.14 — Ocupação de recursos dos módulos que implementam VGA + SVGA (Texto).

Recurso	Utilizados	Disponíveis	Percentagem Ocupada
<i>Total Number of Slice Flip-Flops</i>	67	9312	0.72%
<i>Total Number of 4 input LUTs</i>	176	9312	1.89%
<i>Number Used as Logic</i>	127		
<i>Number Used as Route-Thru</i>	49		
<i>Total Number of Occupied Slices</i>	93	4656	2.00%
<i>Number of Bounded IOBs</i>	10	232	4.31%
<i>Number of BUFGMUXs</i>	1	24	4.17%
<i>Number of RAMB16s</i>	5	20	25.00%
<i>Number of MULTI18X18SIOs</i>	2	20	10.00%

Foi obtido um *fan-out* médio para os sinais de 3.25.

Como podemos ver pelos resultados da tabela anterior, se ambos os protocolos estiverem disponíveis, os recursos utilizados na sua implementação não são semelhantes à soma dos recursos necessários para implementar cada um individualmente. Isto deve-se ao facto de haver uma partilha de recursos e de lógica interna, que são equivalentes para os dois protocolos.

Tabela 4.15 — Ocupação de recursos dos módulos que implementam VGA + SVGA (Texto), estando fixo a VGA.

Recurso	Utilizados	Disponíveis	Percentagem Ocupada
<i>Total Number of Slice Flip-Flops</i>	45	9312	0.48%
<i>Total Number of 4 input LUTs</i>	90	9312	0.97%
<i>Number Used as Logic</i>	54		
<i>Number Used as Route-Thru</i>	36		
<i>Total Number of Occupied Slices</i>	47	4656	1.01%
<i>Number of Bounded IOBs</i>	10	232	4.31%
<i>Number of BUFGMUXs</i>	1	24	4.17%
<i>Number of RAMB16s</i>	5	20	25.00%
<i>Number of MULTI18X18SIOs</i>	1	20	5.00%

Foi obtido um *fan-out* médio para os sinais de 3.39.

OS resultados obtidos demonstram que ao implementar somente o protocolo VGA gastam-se mais recursos do que ao utilizar o projecto original do controlador do modo de texto em VGA, utilizando-se mais 22 *Flip-Flops*, 26 *LUTs* e 7 *Slices*.

Tabela 4.16 – Ocupação de recursos dos módulos que implementam VGA + SVGA (Texto), estando fixo a SVGA.

Recurso	Utilizados	Disponíveis	Percentagem Ocupada
<i>Total Number of Slice Flip-Flops</i>	45	9312	0.48%
<i>Total Number of 4 input LUTs</i>	100	9312	1.07%
<i>Number Used as Logic</i>	63		
<i>Number Used as Route-Thru</i>	37		
<i>Total Number of Occupied Slices</i>	54	4656	1.16%
<i>Number of Bounded IOBs</i>	10	232	4.31%
<i>Number of BUFGMUXs</i>	1	24	4.17%
<i>Number of RAMB16s</i>	5	20	25.00%
<i>Number of MULTI18X18SIOs</i>	1	20	5.00%

Foi obtido um *fan-out* médio para os sinais de 3.33.

Ao implementar somente o protocolo SVGA neste módulo, obtemos resultados muito semelhantes à utilização do protocolo inicial. Apenas o número de IOBs é reduzido substancialmente.

4.12 - Som

O módulo de controlo de sons, apesar de ser um módulo bastante simples, utiliza dois divisores de relógio, o que obriga à reserva de mais recursos, para armazenar os registos utilizados para contar o número de ciclos de relógio, bem como os parâmetros que definem os limites dos divisores de relógio. Sendo assim, espera-se que seja utilizada uma quantidade considerável de recursos da FPGA, tendo em conta a simplicidade do módulo

Tabela 4.17 – Ocupação de recursos do módulo “Sound.v”.

Recurso	Utilizados	Disponíveis	Percentagem Ocupada
<i>Total Number of Slice Flip-Flops</i>	40	9312	0.43%
<i>Total Number of 4 input LUTs</i>	284	9312	3.05%
<i>Number Used as Logic</i>	247		
<i>Number Used as Route-Thru</i>	37		
<i>Total Number of Occupied Slices</i>	145	4656	3.11%
<i>Number of Bounded IOBs</i>	6	232	2.58%
<i>Number of BUFGMUXs</i>	1	24	4.17%

Foi obtido um *fan-out* médio para os sinais de 3.64.

A tabela seguinte contém informação sobre os recursos ocupados por todos os módulos, podendo-se assim efectuar comparações entre ambos mais facilmente.

Tabela 4.18 - Ocupação de recursos dos módulos mais importantes.

Recurso	Botões	LCD	Teclado	Rato	UART	VGA (Gráfico)	VGA (Texto)	SVGA (Gráfico)	SVGA (Texto)
<i>Flip-Flops</i>	102	50	73	82	44	23	23	26	45
<i>LUTs-4</i>	84	110	278	136	121	56	74	69	101
<i>Slices</i>	86	63	157	80	65	31	40	40	54
<i>IOBs</i>	31	21	13	26	23	26	29	10	9
<i>BUFGMUXs</i>	1	1	1	1	1	1	1	1	1

4.13 - Breakout

A aplicação que implementa o jogo *Breakout* reutiliza vários módulos cujos resultados de implementação já foram analisados, pelo que torna-se interessante saber qual a percentagem de recursos que é atribuída a cada um desses módulos.

Tabela 4.19 – Ocupação de recursos dos módulos que implementam o jogo *Breakout*.

Recurso	Utilizados	Disponíveis	Percentagem Ocupada
<i>Total Number of Slice Registers</i>	375	9312	4.03%
<i>Total Number of 4 input LUTs</i>	2196	9312	23.58%
<i>Number Used as Logic</i>	1988		
<i>Number Used as Route-Thru</i>	208		
<i>Total Number of Occupied Slices</i>	1153	4656	24.76%
<i>Number of Bounded IOBs</i>	20	232	8.62%
<i>Number of BUFGMUXs</i>	1	24	4.17%
<i>Number of RAMB16s</i>	5	20	25.00%
<i>Number of MULTI18X18SIOs</i>	1	20	5.00%

Foi obtido um *fan-out* médio para os sinais de 3.34.

De referir que 13 *Slice Registers* são usados como *Latches* sendo os restantes utilizados como *Flip-Flops*.

Analisando os recursos ocupados pelos módulos utilizados nesta aplicação cujas estatísticas de utilização de recursos são conhecidas, obtemos em percentagem a contribuição de cada um dos módulos na aplicação final.

Tabela 4.20 — Percentagem de contribuição de cada um dos módulos na ocupação de recursos do jogo *Breakout*.

Recurso	Teclado	Som	VGA (Texto)	VGA (Gráfico)	LCD	Total
<i>Total Number of Slice Registers</i>	19.5%	10.7%	6.1%	6.1%	13.3%	55.7%
<i>Total Number of 4 input LUTs</i>	12.7%	12.9%	3.4%	2.6%	5%	36.6%
<i>Total Number of Occupied Slices</i>	13.6%	12.6%	3.5%	2.7%	5.5%	37.9%
<i>Number of RAMB16s</i>	0%	0%	100%	0%	0%	100%
<i>Number of MULT18X18SIOs</i>	0%	0%	100%	0%	0%	100%

Os restantes recursos ocupados pela aplicação devem-se à lógica implementada pelos módulos “Breakout_Move.v”, “Breakout_VGA.v” e “Breakout.v”, que são responsáveis pela implementação das funcionalidades do jogo.

4.14 - Pong

A aplicação que implementa o jogo *Pong* também reutiliza vários módulos cujos resultados de implementação já foram analisados, pelo que torna-se interessante saber qual a percentagem de recursos que é atribuída a cada um desses módulos. No entanto, este módulo integra também a interface com o rato PS/2, o que leva a uma maior complexidade do que a aplicação anterior. Contudo, não é necessária a utilização de tantos recursos para definir as interacções entre objectos, uma vez que não existem os 24 tijolos da aplicação anterior.

Tabela 4.21 — Ocupação de recursos dos módulos que implementam o jogo *Pong*.

Recurso	Utilizados	Disponíveis	Percentagem Ocupada
<i>Total Number of Slice Registers</i>	426	9312	4.57%
<i>Total Number of 4 input LUTs</i>	1865	9312	20.03%
<i>Number Used as Logic</i>	1596		
<i>Number Used as Route-Thru</i>	269		
<i>Total Number of Occupied Slices</i>	1003	4656	21.54%
<i>Number of Bounded IOBs</i>	20	232	8.62%
<i>Number of BUFGMUXs</i>	1	24	4.17%
<i>Number of RAMB16s</i>	5	20	25.00%
<i>Number of MULT18X18SIOs</i>	1	20	5.00%

Foi obtido um *fan-out* médio para os sinais de 2.94.

De referir que 13 *Slice Registers* são usados como *Latches* sendo os restantes utilizados como *Flip-Flops*.

Analisando os recursos ocupados pelos módulos utilizados nesta aplicação cujas estatísticas de utilização de recursos são conhecidas, obtemos em percentagem a contribuição de cada um dos módulos na aplicação final.

Tabela 4.22 – Percentagem de contribuição de cada um dos módulos na ocupação de recursos do jogo *Pong*.

Recurso	Teclado	Rato	Som	VGA (Texto)	VGA (Gráfico)	LCD	Total
<i>Total Number of Slice Registers</i>	17.1%	19.2%	9.4%	5.4%	5.4%	11.7%	68.2%
<i>Total Number of 4 input LUTs</i>	14.9%	7.3%	15.2%	4%	3%	5.9%	50.3%
<i>Total Number of Occupied Slices</i>	15.7%	8%	14.5%	4%	3.1%	6.3%	51.6%
<i>Number of RAMB16s</i>	0%	0%	0%	100%	0%	0%	100%
<i>Number of MULT18X18SIOs</i>	0%	0%	0%	100%	0%	0%	100%

Os restantes recursos ocupados pela aplicação devem-se à lógica implementada pelos módulos “Pong_Move.v”, “Pong_VGA.v” e “Pong.v”, que são responsáveis pela implementação das funcionalidades do jogo.

Capítulo 5

Conclusões

Neste capítulo serão apresentadas as conclusões obtidas durante o decorrer do trabalho desta dissertação de mestrado, nomeadamente as conclusões obtidas a partir dos resultados de implementação dos módulos desenvolvidos.

No final deste capítulo apresentar-se-ão algumas ideias de melhoria dos módulos desenvolvidos, bem como a sugestão de criação de novos módulos baseados, ou não, nos que foram desenvolvidos durante esta dissertação.

5.1 - Conclusões Finais

Durante a construção de alguns módulos foi constatado que algumas funções dos respectivos protocolos foram omissas, de forma a permitir uma interface com os periféricos mais simples, tendo em conta o principal objectivo e motivação da criação deste trabalho: a reutilização destes módulos em projectos em que os alunos do Mestrado Integrado em Engenharia Electrotécnica e de Computadores participem. No entanto, todos os módulos desenvolvidos cumprem na totalidade as suas funções, de uma forma bastante robusta.

Outra razão pela qual não foram implementadas algumas funções foi a necessidade de poupar recursos, uma vez que estes módulos podem ser integrados em projectos de grandes dimensões, em que o impacto nos recursos utilizados pela inserção destes módulos deve de ser o menor possível. Desta forma foi possível obter módulos que efectuassem a interface com os periféricos que não introduzem um *overhead* muito elevado ao projecto.

Caso se pretendam módulos mais robustos, para a aplicação em situações comerciais e industriais, esses aspectos omissos devem ser implementados.

Estas omissões nos módulos incluem a não verificação do bit de paridade nas tramas recebidas pelo protocolo PS/2, o respectivo pedido de retransmissão, a não implementação de algumas teclas nos módulos de controlo do teclado e a não existência de mecanismos de controlo de erros na transmissão série.

Apesar destes pormenores, todos os módulos cumprem os seus requisitos mínimos na totalidade.

Durante a análise dos resultados obtidos da implementação dos módulos é possível concluir que o uso de divisores de relógio deve ser evitado, pois utilizam um grande número

de recursos para efectuarem a contagem do número de ciclos e para armazenar o limite dessa contagem. Uma das soluções encontradas para obter o efeito de um divisor de relógio, utilizando apenas de um registo, foi implementado no módulo de controlo do LCD.

Outras conclusões que se podem obter estão relacionadas com o uso e comparação entre os protocolos VGA e SVGA.

Apesar de o protocolo SVGA possibilitar uma maior resolução no ecrã, pode limitar o projecto no qual está inserido, quer pelos recursos de implementação necessários (não são necessários muito mais recursos do que a utilização do protocolo VGA), quer pelo seu *pixel rate* igual à frequência de relógio interno da plataforma de desenvolvimento, que exige que a cada subida de relógio, o valor dos sinais de controlo dos pixéis (“R”, “G” e “B”) possuam o seu respectivo valor. Isto não acontece no protocolo VGA, uma vez que o seu *rate* é metade da frequência de relógio interno da plataforma de desenvolvimento.

No modo de texto do protocolo SVGA os recursos adicionais necessários, em comparação com a utilização do protocolo VGA, são reduzidos, embora a diferença seja superior aos módulos do modo gráfico. No entanto, o recurso mais utilizado em ambos os casos, os blocos de memória RAM, é utilizada de igual forma pelos dois protocolos. Caso não seja utilizado a totalidade do ecrã em modo de texto estes módulos podem ser adaptados de forma a utilizarem menos memória RAM.

Ao analisar as estatísticas relativamente ao módulo que aglomera os dois protocolos no modo gráfico, observamos que na implementação em que se pode alternar em tempo real entre módulos, os recursos ocupados são semelhantes aos utilizados na soma dos dois protocolos. Caso, durante a fase de projecto, for definido que o módulo só deve utilizar um dos protocolos, a ferramenta disponibilizada pela *Xilinx* para implementar projectos na plataforma, corta os módulos não utilizados durante a síntese do projecto, levando a que os recursos ocupados pelo módulo que vai ser utilizado sejam idênticos (iguais em alguns casos) aos recursos utilizados pelos módulos específicos dos protocolos.

Assim conclui-se que o projectista tem vantagem em escolher o módulo que implementa os protocolos VGA e SVGA e, caso pretenda poupar recursos, pode fixar-se apenas num protocolo, não ocupando mais recursos com isso, quando comparado com os módulos que implementam os protocolos em separado.

No módulo que engloba os dois protocolos no modo de texto, os recursos utilizados na implementação dos dois protocolos são semelhantes aos recursos ocupados caso os protocolos VGA e SVGA em modo de texto fossem implementados separadamente. Caso, durante a fase de projecto, seja definido que o módulo só deve usar o protocolo SVGA, os recursos ocupados são muito semelhantes, mesmo iguais em alguns elementos. O mesmo não acontece com o protocolo VGA que, caso durante a fase de projecto seja definido que o módulo só deve usar este protocolo, são utilizados mais recursos do que usando o módulo que implementa somente este protocolo em modo de texto.

Desta forma conclui-se que este módulo que aglomera os protocolos VGA e SVGA no modo de texto compensa, caso se pretenda utilizar os dois protocolos ou caso se pretenda que fique fixo ao protocolo SVGA. Se o projectista quiser usar apenas o protocolo VGA no modo de texto, é recomendável a utilização dos seus módulos específicos, que ocupam menos recursos, e não o que abrange os dois protocolos.

Tendo em conta os resultados de implementação das duas aplicações, *Breakout* e *Pong*, podemos concluir que a percentagem de recursos que os módulos que implementam o protocolo VGA (modo gráfico e texto) é reduzida, em comparação com os outros módulos

integrados. Os módulos que controlam o teclado ocupam uma grande fatia dos recursos ocupados, possuindo a maior percentagem em todos os recursos, excepto na utilização das LUTs em que é superado pelo módulo que controla o som. O módulo de controlo do som também é responsável pela utilização de muitos recursos, nomeadamente LUTs e *Slices*. Isto deve-se à utilização de dois divisores de relógio no módulo. O módulo de controlo de LCD, que usa um método diferente de contagem no tempo, necessita de mais registos, mas de muito menos LUTs e ocupa menos *Slices*. No caso da aplicação *Pong* e comparando os módulos que controlam o rato e o teclado PS/2, o rato ocupa mais registos, enquanto o teclado ocupa duas vezes mais LUTs e *Slices*. Estas diferenças de recursos devem-se exclusivamente aos módulos “Keyboard.v” e “Mouse.v”, uma vez que os módulos inferiores são iguais para ambos. O teclado utiliza mais LUTs, e consequentemente mais *Slices*, provavelmente devido às duas máquinas de estados do módulo “Keyboard.v”, sendo que uma delas contempla muitos estados, pois é utilizada na conversão de *make/break codes* no código ASCII do carácter.

5.2 - Trabalhos Futuros

Como trabalhos futuros, apresentam-se de seguida algumas sugestões:

- O desenvolvimento de outros módulos que permitam utilizar o conector RJ-45, para conectar a plataforma de desenvolvimento através de outros protocolos.
- O desenvolvimento de módulos que implementem o protocolo USB, juntamente com o desenvolvimento de um conector que possa ser ligado a um porto de expansão da plataforma de desenvolvimento. A introdução deste protocolo permitiria a interacção com um elevado número de periféricos, tais como ratos e teclados com conexão USB, Impressoras, *Scanners*, *Webcams*, *Pendrives* e Controladores *Wireless (Wi-Fi e Bluetooth)*.
- O aperfeiçoamento dos módulos aqui desenvolvidos também seria um trabalho interessante, de forma a torná-los o mais robustos possível, eliminando as pequenas falhas que foram referidas anteriormente. Também seria interessante criar um módulo para escrever *strings* no LCD.
- A criação de módulos para efectuar a ligação com os PWMs presentes na plataforma de desenvolvimento, bem como os conversores analógico/digital e digital/analógico, seria interessante e abria caminho para outro tipo de projectos desenvolvidos com esta plataforma.
- A criação de módulos que permitissem a interacção com a câmara de detecção de infra-vermelhos presentes nos comandos da consola *Nintendo® Wii™* seria uma adição de valor à biblioteca criada no decorrer do trabalho desta dissertação de mestrado.

Anexo A

Manuais

Os manuais elaborados para os módulos desenvolvidos serão colocados nesta dissertação de mestrado sobre a forma de anexo. Estes são incluídos pois são parte dos objectivos iniciais do trabalho pretendido para esta dissertação. Estes manuais poderão ser consultados de uma forma visualmente mais apelativa no endereço de Internet destinado ao trabalho desta dissertação de mestrado [18].

A.1 - Botões e LEDs

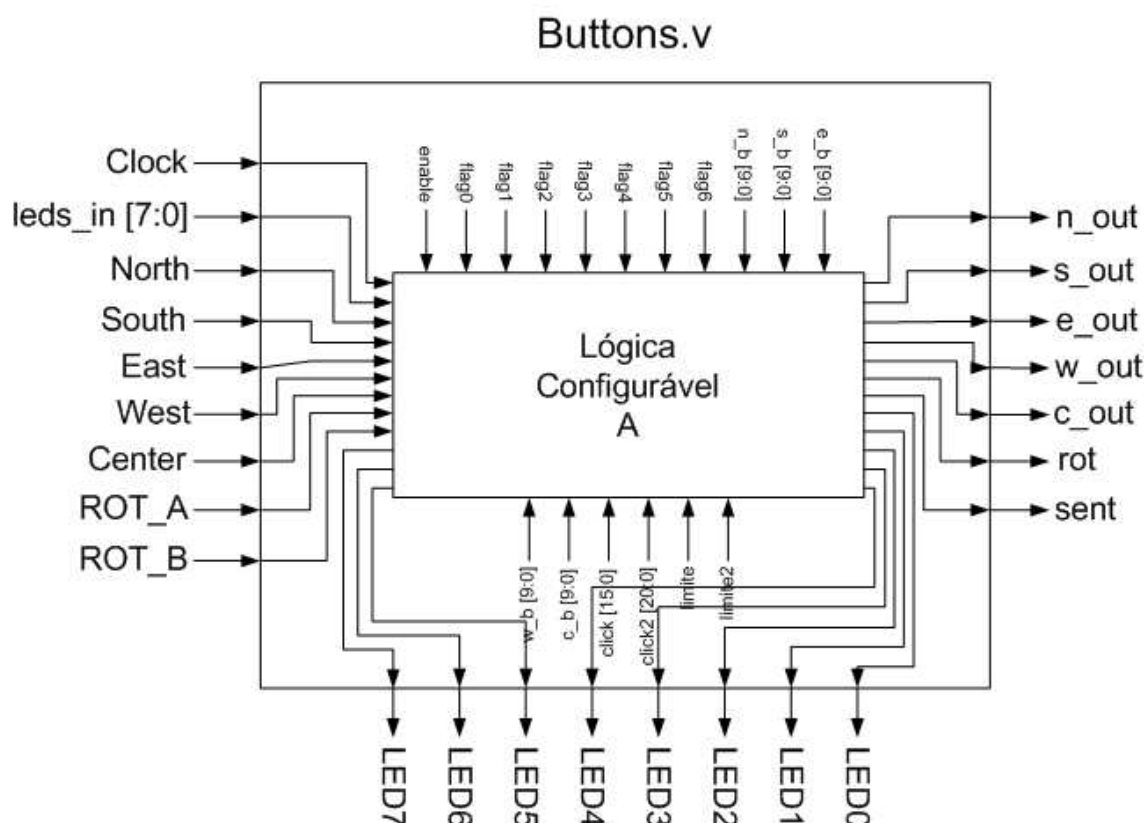


Figura A.1 - Módulo "Buttons.v".

Tabela A.1 - Sinais do módulo "Buttons.v".

Entradas:	
Clock	Relógio interno de 50MHz.
leds_in [7:0]	Sinal que contem o estado pretendido para cada um dos LEDs.
North	Sinal proveniente do botão norte da plataforma de desenvolvimento.
South	Sinal proveniente do botão sul da plataforma de desenvolvimento.
East	Sinal proveniente do botão este da plataforma de desenvolvimento.
West	Sinal proveniente do botão oeste da plataforma de desenvolvimento.
Center	Sinal proveniente do botão centro da plataforma de desenvolvimento.
ROT_A	Sinal proveniente do botão rotativo da plataforma de desenvolvimento.
ROT_B	Sinal proveniente do botão rotativo da plataforma de desenvolvimento.
Saídas:	
LED7 ... LED0	Saída directamente ligada aos LEDs da plataforma de desenvolvimento.
n_out	Sinal "unbounced" do botão norte da plataforma de desenvolvimento.
s_out	Sinal "unbounced" do botão sul da plataforma de desenvolvimento.
e_out	Sinal "unbounced" do botão este da plataforma de desenvolvimento.
w_out	Sinal "unbounced" do botão oeste da plataforma de desenvolvimento.
c_out	Sinal "unbounced" do botão centro da plataforma de desenvolvimento.
rot	Sinal "unbounced" do botão rotativo da plataforma de desenvolvimento, que indica a ocorrência de uma rotação.

sent	Sinal "unbounced" do botão rotativo da plataforma de desenvolvimento, que indica o sentido da rotação.
Registos Internos:	
click [15:0]	Utilizado na contagem para o "unbouncing" dos botões.
click_2 [20:0]	Utilizado na contagem para o "unbouncing" do botão rotativo.
enable	Utilizado para o "unbouncing" dos botões.
flag0 ... flag6	Utilizado para o "unbouncing" dos botões e do botão rotativo.
n_b	"Shift-Register" utilizado para o "unbouncing" do botão norte.
s_b	"Shift-Register" utilizado para o "unbouncing" do botão sul.
e_b	"Shift-Register" utilizado para o "unbouncing" do botão este.
w_b	"Shift-Register" utilizado para o "unbouncing" do botão oeste.
c_b	"Shift-Register" utilizado para o "unbouncing" do botão centro.
Parâmetros Internos:	
limite	Utilizado para o unbouncing dos botões.
limite_2	Utilizado para o unbouncing do botão rotativo.

Design Summary:

Tabela A.2 - Ocupação de recursos do módulo "Buttons.v".

Recurso	Utilizados	Disponíveis	Percentagem Ocupada
<i>Total Number of Slice Flip-Flops</i>	102	9312	1.09%
<i>Total Number of 4 input LUTs</i>	84	9312	0.90%
<i>Number Used as Logic</i>	49		
<i>Number Used as Route-Thru</i>	35		
<i>Total Number of Occupied Slices</i>	86	4656	1.85%
<i>Number of Bounded IOBs</i>	31	232	13.36%
<i>Number of BUFGMUXs</i>	1	24	4.17%

Para a utilização do módulo "Buttons.v" é necessária a ligação deste módulo aos recursos da FPGA que este vai utilizar. Estes recursos são: Relógio de 50MHz, os botões (North, South, East, West e Center) assim como os sinais do botão rotativo (ROT_A e ROT_B) e os oito leds (LED7 ... LED0).

No módulo "s3_Buttons_top.v" é demonstrado um exemplo dessas ligações, assim como um pequeno exemplo de utilização. No ficheiro s3_Buttons_top.ucf encontram-se definidas as localizações destes recursos na FPGA. Caso altere o nome de alguma entrada ou saída no módulo "s3_Buttons_top.v", é também necessário alterar o nome da "net" no ficheiro s3_Buttons_top.ucf, pelo que não se aconselha essa alteração.

O unbouncing dos botões é feito através de um "clock divider" e de um registo de 10 bits. O "clock divider" gera um impulso a cada milissegundo e o registo guarda o valor dos últimos 10 milissegundos. Quando um botão é premido durante 10 milissegundo, é gerado um sinal com a duração de um ciclo de relógio na saída respectiva do botão.

Caso seja necessário alterar o intervalo de unbounding, pode-se fazê-lo alterando o tamanho do registo ou alterando o “clock divider”. Para alterar o “clock divider” apenas é necessário alterar o parâmetro “limite” especificado no início do módulo.

O botão rotativo possui um “clock divider” diferente, pois necessita de um maior intervalo de tempo para unbounding devido ao ruído mecânico de rotação.

Sempre que há uma rotação, o sinal de saída do módulo “Buttons.v” com o nome de “rot” fica durante um período de relógio a ‘1’, e o sinal de saída com o nome “sent” indica o sentido da rotação (‘0’ esquerda, e ‘1’ direita).

Para activar os LEDs utiliza-se a entrada de oito bits “leds” em que o bit mais significativo representa o LED7, e o menos significativo o LED0.

A.2 - LCD

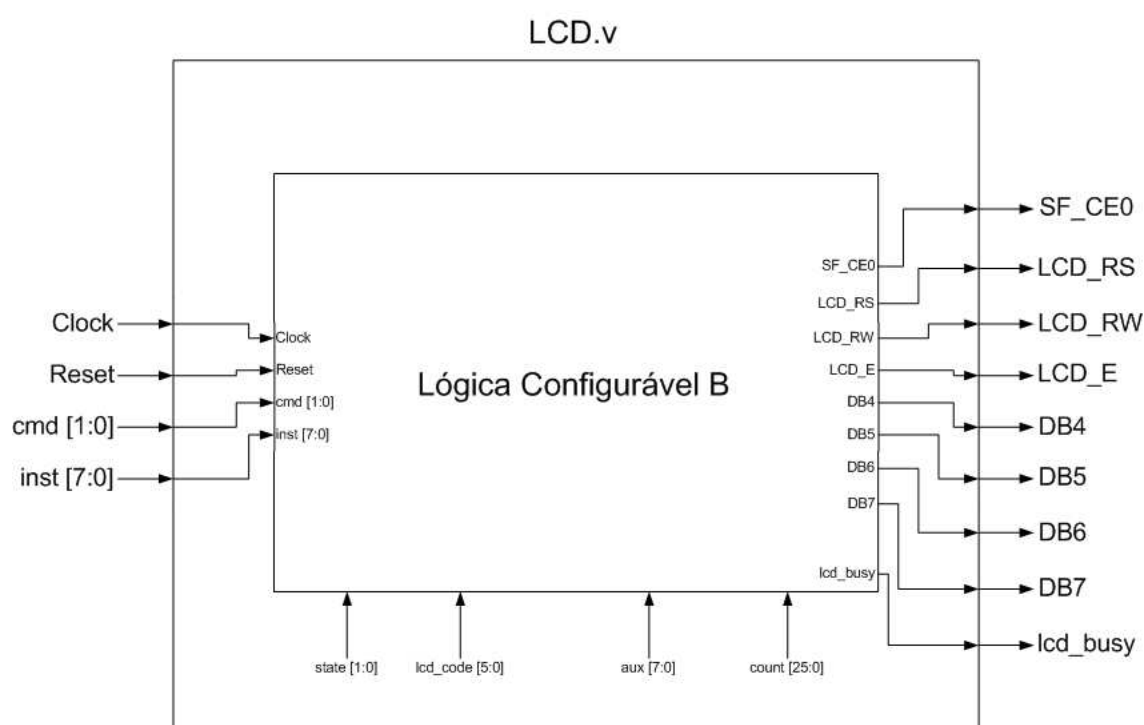


Figura A.2 - Módulo “LCD.v”.

Tabela A.3 - Sinais do módulo “LCD.v”.

Entradas:	
Clock	Relógio interno de 50MHz.
Reset	Sinal externo de reset (activo a '1').
cmd [1:0]	Indica o tipo de comando a ordenar ao LCD.
inst [7:0]	Indica o código da instrução a ser processada pelo LCD.
Saídas:	
SF_CEO	"Chip Enable" do StrataFlash da FPGA (para desactivar o StrataFlash e activar o LCD).
LCD_RS	"Register Select" do LCD.
LCD_RW	"Read/Write" do LCD.
LCD_E	"Enable" do LCD.

DB4 ... DB7	Barramento paralelo de dados do LCD.
lcd_busy	Indica se o LCD da FPGA está ocupado ou livre.
Registos Internos:	
state [1:0]	Usado na FSM de controlo do LCD.
lcd_code [5:0]	Concatenação dos sinais LCD_RS, LCD_RW, DB7, DB6, DB5 e DB4.
aux [7:0]	Utilizado para armazenar a instrução.
count[25:0]	Utilizado como "Clock Divider".

Design Summary:**Tabela A.4 - Ocupação de recursos do módulo "LCD.v".**

Recurso	Utilizados	Disponíveis	Percentagem Ocupada
<i>Total Number of Slice Flip-Flops</i>	50	9312	0.54%
<i>Total Number of 4 input LUTs</i>	110	9312	1.18%
<i>Number Used as Logic</i>	85		
<i>Number Used as Route-Thru</i>	25		
<i>Total Number of Occupied Slices</i>	63	4656	1.35%
<i>Number of Bounded IOBs</i>	21	232	9.05%
<i>Number of BUFGMUXs</i>	1	24	4.17%

Para utilizar o módulo "LCD.v", é necessária a ligação no módulo pretendido a todos os recursos utilizados por este módulo. Esses recursos são: Relógio de 50MHz, um botão de reset, SF_CE0 para desligar o StrataFlash, LCD_RS, LCD_RW, LCD_E, DB4, DB5, DB6 e DB7 que são as linhas de dados e controlo do LCD. Para ligar estes recursos é exemplificada a sua ligação em s3_lcd_top.ucf, juntamente com o módulo "s3_lcd_top.v".

No módulo "s3_lcd_top.v" é demonstrada a ligação do módulo "LCD.v", que possui um pequeno exemplo de utilização deste. Em s3_lcd_top.ucf encontram-se as localizações dos recursos na FPGA, assim como as suas condições de operação. Caso altere o nome das entradas e saídas do módulo s3_lcd_top.v, terá também que alterar o nome da "NET" no ficheiro s3_lcd_top.ucf, pelo que a alteração dos nomes não é aconselhada.

Caso queira inserir o módulo "LCD.v" directamente num projecto já elaborado, pode copiar o conteúdo do ficheiro s3_lcd_top.ucf ao ficheiro ucf do seu projecto, e adicionando as respectivas entradas e saídas no módulo.

Para interagir com o LCD, e depois da ligação do módulo "LCD.v", apenas necessita de utilizar 2 registos (cmd e inst) e uma flag (lcd_busy).

A flag lcd_busy indica se o LCD se encontra a processar um comando ou se está livre para novo comando. Quando lcd_busy = 1, não devem ser enviados comandos para o LCD, pois estes não serão lidos, ou serão lidos incorrectamente. Quando lcd_busy = 0 pode-se enviar o comando ao LCD que este será processado.

O registo cmd especifica o tipo de acção a realizar. Quando o valor de cmd é 0 ou 3 não se actua sobre o LCD, 1 é utilizado para escrever caracteres e 2 para dar um comando ao LCD.

O registo inst especifica a instrução a ser processada pelo LCD. Será um comando quando cmd = 2, e o código ASCII de um carácter quando cmd = 1.

A.3 - Teclado

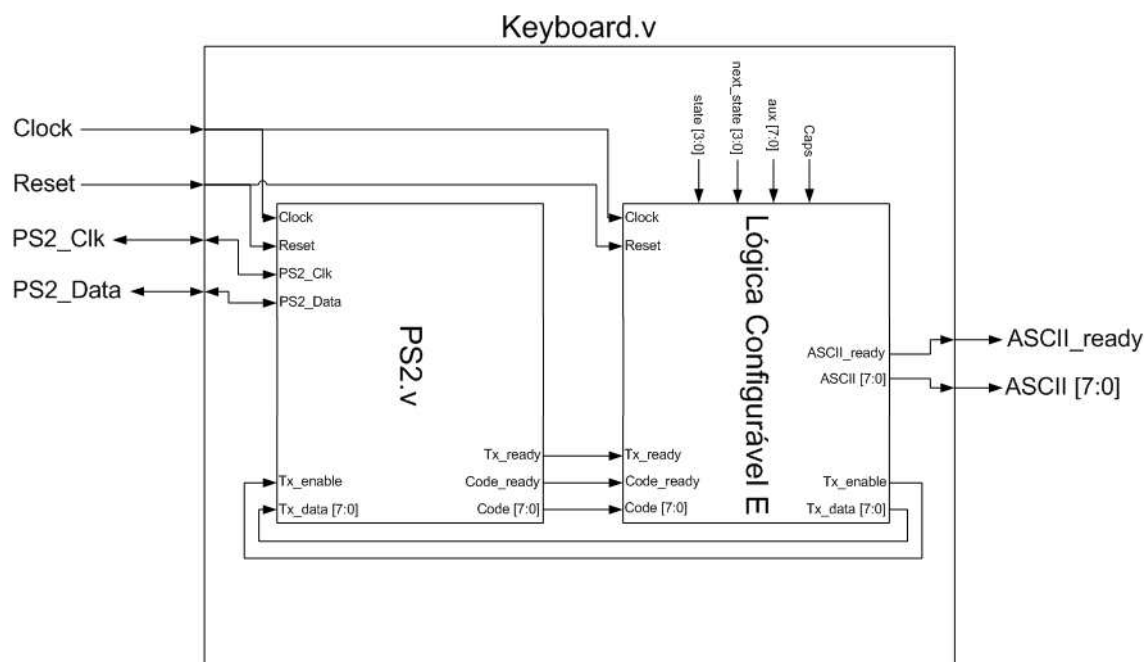


Figura A.3 - Módulo “Keyboard.v”.

Tabela A.5 - Sinais do módulo “Keyboard.v”.

Entradas:	
Clock	Relógio interno de 50MHz.
Reset	Sinal externo de reset (activo a '1').
Bidireccionais:	
PS2_Clk	Linha de relógio do barramento PS2.
PS2_Data	Linha de dados do barramento PS2.
Saídas:	
ASCII_ready	Indica a recepção do código ASCII de uma tecla do teclado PS2.
ASCII [7:0]	Código ASCII da tecla premida no teclado PS2.
Ligações Internas:	
Tx_enable	Activa a transmissão de dados para o teclado PS2.
Tx_data [7:0]	Byte a transmitir para o teclado PS2.
Tx_ready	Indica a conclusão da transmissão para o teclado PS2.
Code_ready	Indica a conclusão de uma recepção do teclado PS2.
Code [7:0]	Byte recebido do teclado PS2.
state [3:0] e next_state [3:0]	Usado na FSM que converte o código da tecla premida no seu código ASCII.
aux [7:0]	Armazena o byte recebido do teclado PS2.
Caps	Indica o estado actual de Caps Lock.

PS2.v

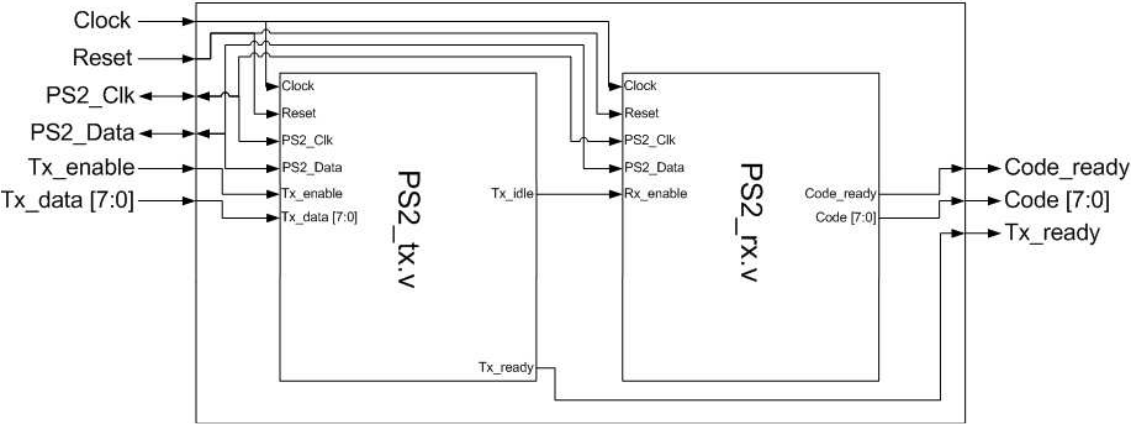


Figura A.4 - Módulo “PS2.v”.

Tabela A.6 - Sinais do módulo “PS2.v”.

Entradas:	
Clock	Relógio interno de 50MHz.
Reset	Sinal externo de reset (activo a '1').
Tx_enable	Activa a transmissão de dados para o teclado PS2.
Tx_data [7:0]	Byte a transmitir para o teclado PS2.
Bidireccionais:	
PS2_Clk	Linha de relógio do barramento PS2.
PS2_Data	Linha de dados do barramento PS2.
Saídas:	
Tx_ready	Indica a conclusão da transmissão para o teclado PS2.
Code_ready	Indica a conclusão de uma recepção do teclado PS2.
Code [7:0]	Byte recebido do teclado PS2.
Ligações Internas:	
Tx_idle/Rx_enable	Gere os conflitos entre transmissão e recepção.

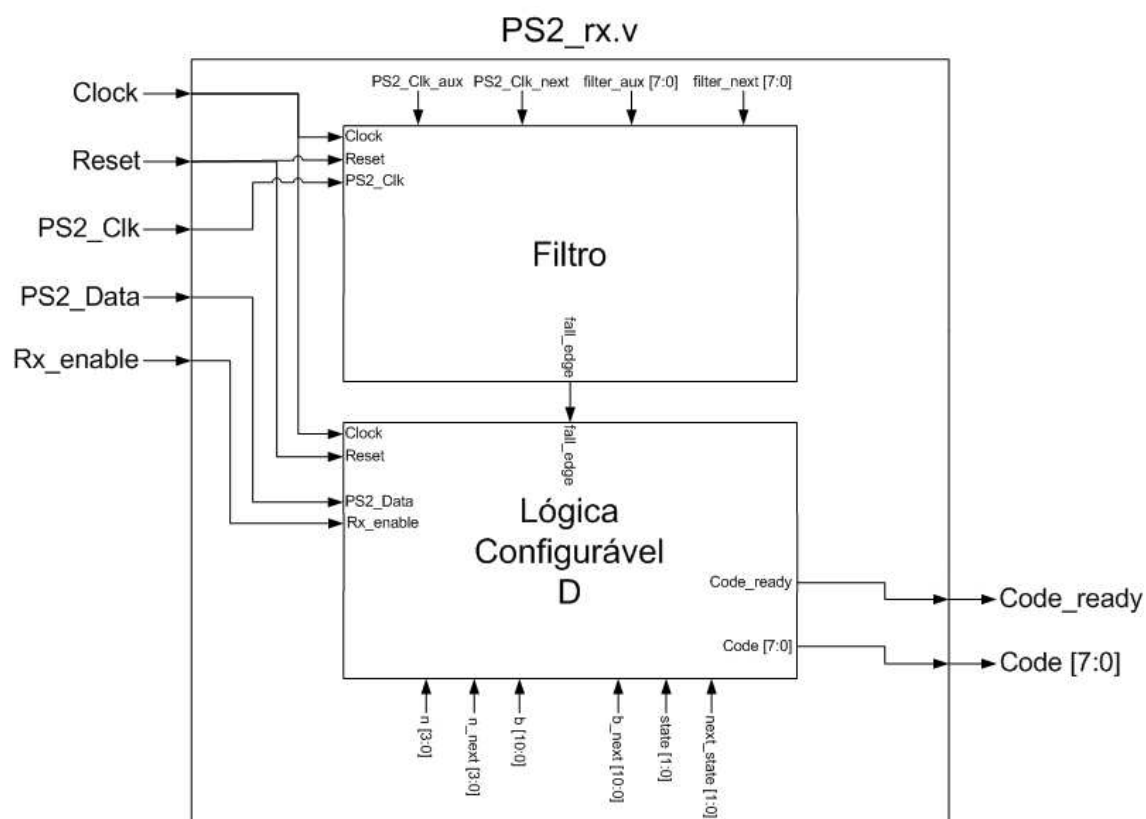


Figura A.5 - Módulo “PS2_rx.v”.

Tabela A.7 - Sinais do módulo “PS2_rx.v”.

Entradas:	
Clock	Relógio interno de 50MHz.
Reset	Sinal externo de reset (activo a '1').
Rx_enable	Activa a recepção de dados do teclado PS2.
Bidireccionais:	
PS2_Clk	Linha de relógio do barramento PS2.
PS2_Data	Linha de dados do barramento PS2.
Saídas:	
Code_ready	Indica a conclusão de uma recepção do teclado PS2.
Code [7:0]	Byte recebido do teclado PS2.
Ligações Internas:	
fall_edge	Indica as transições de '1' para '0' de PS2_Clk.
Registos Internos:	
PS2_Clk_aux e PS2_Clk_next	Usado no filtro aplicado ao sinal PS2_Clk.
filter_aux [7:0] e filter_next [7:0]	Usado no filtro aplicado ao sinal PS2_Clk.
state [1:0] e state_next [1:0]	Usado na FSM usada para receber os dados do teclado PS2.
n [3:0] e n_next [3:0]	Indica o número de bits que ainda falta receber.

b [10:0] e b_next [10:0]	Armazena temporariamente os dados recebidos.
--------------------------	--

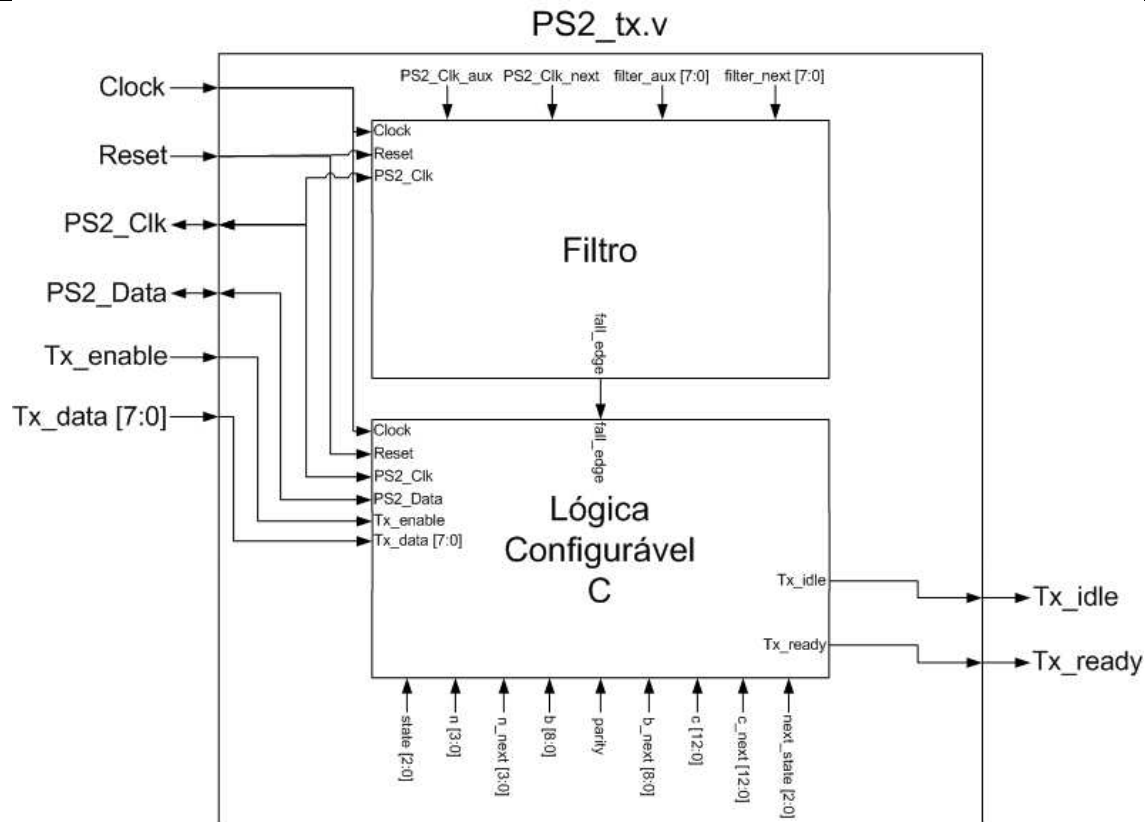


Figura A.6 - Módulo “PS2_tx.v”.

Tabela A.8 - Sinais do modulo “PS_tx.v”.

Entradas:	
Clock	Relógio interno de 50MHz.
Reset	Sinal externo de reset (activo a '1').
Tx_enable	Activa a transmissão de dados para o teclado PS2.
Tx_data [7:0]	Byte a transmitir para o teclado PS2.
Bidireccionais:	
PS2_Clk	Linha de relógio do barramento PS2.
PS2_Data	Linha de dados do barramento PS2.
Saídas:	
Tx_idle	Indica quando o módulo não está a transmitir dados para o teclado PS2.
Tx_ready	Indica a conclusão da transmissão para o teclado PS2.
Ligações Internas:	
fall_edge	Indica as transições de '1' para '0' de PS2_Clk.
Registos Internos:	
PS2_Clk_aux e PS2_Clk_next	Usado no filtro aplicado ao sinal PS2_Clk.
filter_aux [7:0] e filter_next [7:0]	Usado no filtro aplicado ao sinal PS2_Clk.
state [2:0] e	Usado na FSM usada para enviar os dados do teclado PS2.

state_next [2:0]	
n [3:0] e n_next [3:0]	Indica o número de bits que ainda falta enviar.
b [8:0] e b_next [8:0]	Armazena temporariamente os dados a enviar.
c [12:0] e c_next [12:0]	Utilizados como "delay" na "Start Condition".
parity	Usado para calcular o bit de paridade.

Design Summary:

Tabela A.9 - Ocupação de recursos dos módulos do teclado PS/2.

Recurso	Utilizados	Disponíveis	Percentagem Ocupada
<i>Total Number of Slice Registers</i>	73	9312	0.78%
<i>Total Number of 4 input LUTs</i>	278	9312	2.99%
<i>Number Used as Logic</i>	277		
<i>Number Used as Route-Thru</i>	1		
<i>Total Number of Occupied Slices</i>	157	4656	3.37%
<i>Number of Bounded IOBs</i>	13	232	5.60%
<i>Number of BUFGMUXs</i>	1	24	4.17%

Para a utilização do módulo "Keyboard.v" são necessários os módulos "PS2.v", "PS2_rx.v" e "PS2_tx.v", para além da ligação aos seguintes recursos da FPGA: Relógio de 50 MHz, botão/sinal de reset, linhas de dados e relógio do conector PS2.

No módulo "s3_Keyboard_top.v" é demonstrado um pequeno exemplo de utilização deste módulo, que para além dos recursos mencionados anteriormente, utiliza ainda os recursos necessários ao funcionamento do módulo "LCD.v" (para mais informação consultar o manual deste módulo). Todas as ligações aos recursos da FPGA estão especificadas no ficheiro Keyboard.ucf. Caso seja alterado o nome de alguma das entradas ou saídas especificadas no ficheiro ".ucf" é necessário alterar o nome da "NET" nesse ficheiro, pelo que a alteração não é recomendada.

No módulo "PS2_rx.v" são processadas todas as comunicações (recepção apenas) do teclado PS2. Este módulo envia depois o código lido (Code[7:0]) do teclado para o módulo "Keyboard.v", que converte esse código no respectivo código ASCII (ASCII[7:0]). Em ambos os módulos considera-se que existe um novo código quando o respectivo sinal (Code_ready ou ASCII_ready) assume o valor lógico "1" durante um ciclo de relógio.

O sinal que controla a leitura dos bits denomina-se fall_edge e vai a '1' durante um ciclo de relógio sempre que é detectada uma transição de '1' para '0' em PS2_Clk_next (é usado o registo PS2_Clk_aux para detectar essa transição).

A FSM usada para receber os bits possui apenas três estados e opera de forma assíncrona com o relógio. O estado 0 é um estado de "idle", passando para o estado 1 sempre que é detectada uma "Start Condition". No estado 1 são recebidos os onze bits que compõe a trama a receber. Os bits são guardados no registo b_next, sendo que o código se encontra entre o

bit 8 e 1 inclusive. O número de bits que falta receber fica guardado em `n_next`. Quando os bits são todos guardados passa-se ao estado 2 que existe para colocar `Code_ready` a '1' e para utilizar mais um ciclo de relógio de forma a actualizar os registos (síncronos com o relógio) `b` e `n`, a partir de `b_next` e `n_next`, regressando ao estado 0.

No módulo "PS2_tx.v" são processadas todas as comunicações (de envio) do teclado PS2. Após um impulso de `Tx_enable`, os 8 bits de `Tx_data` são enviados juntamente com o bit de paridade (CRC) que é calculado automaticamente. O final da transmissão é assinalado por um impulso com a duração de um ciclo de relógio de `Tx_ready`.

O sinal que controla a escrita dos bits denomina-se `fall_edge` e funciona no mesmo modo que em "PS2_rx.v", sendo que os bits são escritos na descida do relógio, e lidos no teclado na subida do mesmo.

A FSM usada para enviar dados possui cinco estados e funciona assincronamente. O estado 0 é o estado de "idle" passando para o estado 1 quando `Tx_enable` é activo. No estado 1 o sinal `PS2_Clk` é mantido a '0' durante 8191 ciclos de relógio ($2^{13} - 1$) de forma a transmitir uma "Start Condition", passando de seguida ao estado 2. No estado 2 a linha `PS2_Clk` é libertada, assumindo controlo da linha `PS2_data` e colocando-a a '0' até o primeiro "fall_edge" quando passa para o estado 3. O estado 3 é o estado de escrita dos bits, onde cada um dos nove bits é escrito a cada "fall_edge". Quando todos os bits tiverem sido enviados passa-se ao estado 4, que existe para colocar `Tx_ready` a '1' e para utilizar mais um ciclo de relógio de forma a actualizar os registos (síncronos com o relógio) `b` e `n`, a partir de `b_next` e `n_next`, regressando ao estado 0.

O registo `b` e `b_next` guardam os bits a enviar, `n` e `n_next` guarda o número de bits por enviar e `c` e `c_next` são utilizados como "delay" da "Start Condition".

Existe ainda a saída `Tx_idle` deste módulo que está a '1' sempre que não se está a transmitir, servindo como `Rx_enable` no módulo `PS2_rx.v`.

Em ambos os módulos (`PS2_rx.v` e `PS2_tx.v`) existe um filtro para as transições do relógio de PS2, de forma a eliminar os efeitos de uma transição lenta ou de bouncing. Deste modo para o valor de `PS2_Clk_next` alterar são necessários 8 ciclos de relógio em que o valor de `PS2_Clk` seja igual e contrário ao valor actual de `PS2_Clk_next`, caso contrário o valor mantém-se.

O módulo "PS2.v" gere todas as trocas de informação, evitando conflitos entre transmissão e recepção.

No módulo "Keyboard.v" sempre que é detectado um código novo este é processado e, caso seja possível, transformado em código ASCII. No entanto, algumas teclas dos teclados têm "Make Codes" de 2 bytes e "Break Codes" de 3 bytes. Todas estas situações estão contempladas na FSM e permite detectar os "Break Codes" de 3 bytes.

A FSM e a correspondência entre os códigos do teclado e ASCII estão em blocos "always" diferentes para que alterações a serem efectuadas futuramente sejam mais fáceis, como por exemplo a adição de uma nova tecla.

A.4 - Rato

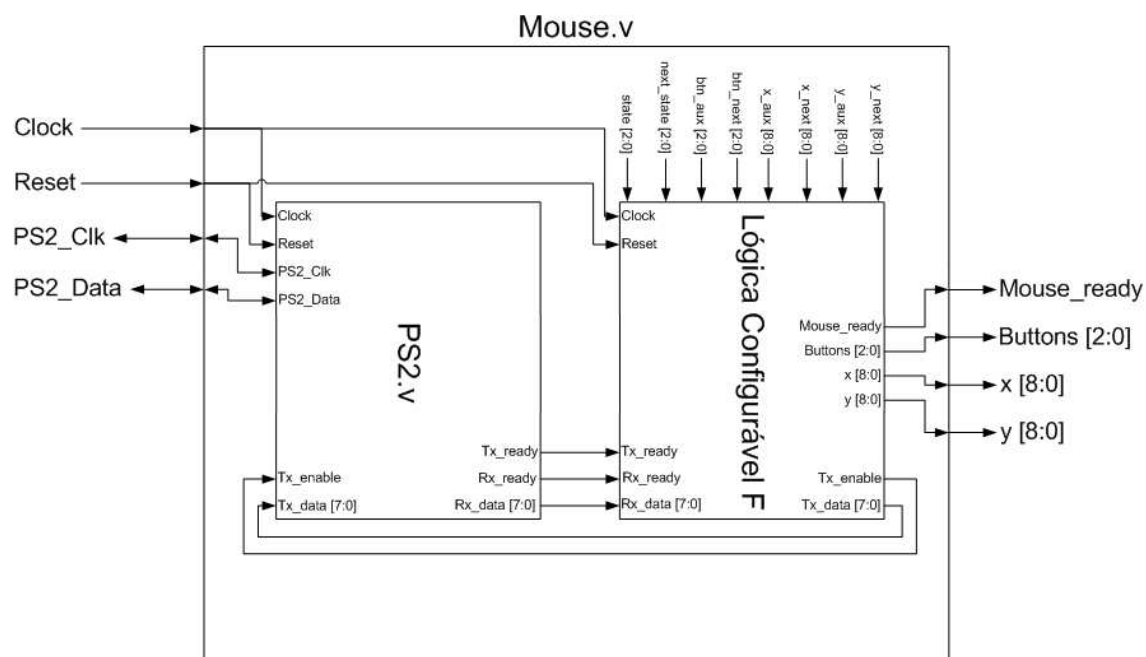


Figura A.7 -Módulo “Mouse.v”.

Tabela A.10 - Sinais do módulo “Mouse.v”.

Entradas:	
Clock	Relógio interno de 50MHz.
Reset	Sinal externo de reset (activo a '1').
Bidireccionais:	
PS2_Clk	Linha de relógio do barramento PS2.
PS2_Data	Linha de dados do barramento PS2.
Saídas:	
Mouse_ready	Indica a recepção de dados do rato PS2.
Buttons [2:0]	Indica o estado dos botões do rato PS2.
x [8:0]	Indica a direcção e amplitude de movimento no eixo x do rato PS2.
y [8:0]	Indica a direcção e amplitude de movimento no eixo y do rato PS2.
Ligações Internas:	
Tx_enable	Activa a transmissão de dados para o rato PS2.
Tx_data [7:0]	Byte a transmitir para o rato PS2.
Tx_ready	Indica a conclusão da transmissão para o rato PS2.
Rx_ready	Indica a conclusão de uma recepção do rato PS2.
Rx_data [7:0]	Byte recebido do rato PS2.
Registos Internos:	
state [2:0] e next_state [2:0]	Usado na FSM que converte os bytes recebidos nas informações de saída.
btn_aux [2:0] e btn_next [2:0]	Armazena temporariamente o estado dos botões do rato PS2.

x_aux [8:0] e x_next [8:0]	Armazena temporariamente a direcção e amplitude de movimento no eixo horizontal do rato PS2.
y_aux [8:0] e y_next [8:0]	Armazena temporariamente direcção e amplitude de movimento no eixo vertical do rato PS2.

PS2.v

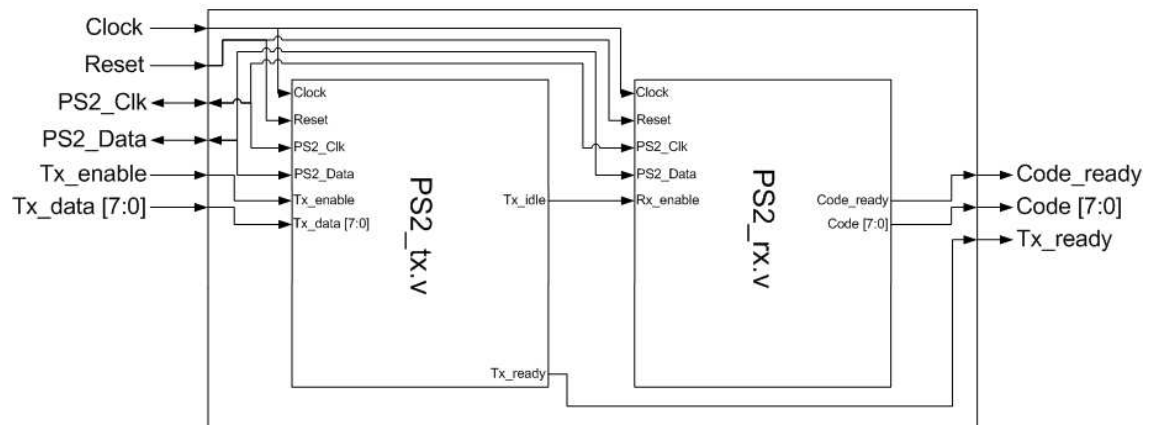


Figura A.8 - Módulo “PS2.v”.

Tabela A.11 - Sinais do módulo “PS2.v”.

Entradas:	
Clock	Relógio interno de 50MHz.
Reset	Sinal externo de reset (activo a '1').
Tx_enable	Activa a transmissão de dados para o teclado PS2.
Tx_data [7:0]	Byte a transmitir para o teclado PS2.
Bidireccionais:	
PS2_Clk	Linha de relógio do barramento PS2.
PS2_Data	Linha de dados do barramento PS2.
Saídas:	
Tx_ready	Indica a conclusão da transmissão para o rato PS2.
Code_ready	Indica a conclusão de uma recepção do rato PS2.
Code [7:0]	Byte recebido do rato PS2.
Ligações Internas:	
Tx_idle/Rx_enable	Gere os conflitos entre transmissão e recepção.

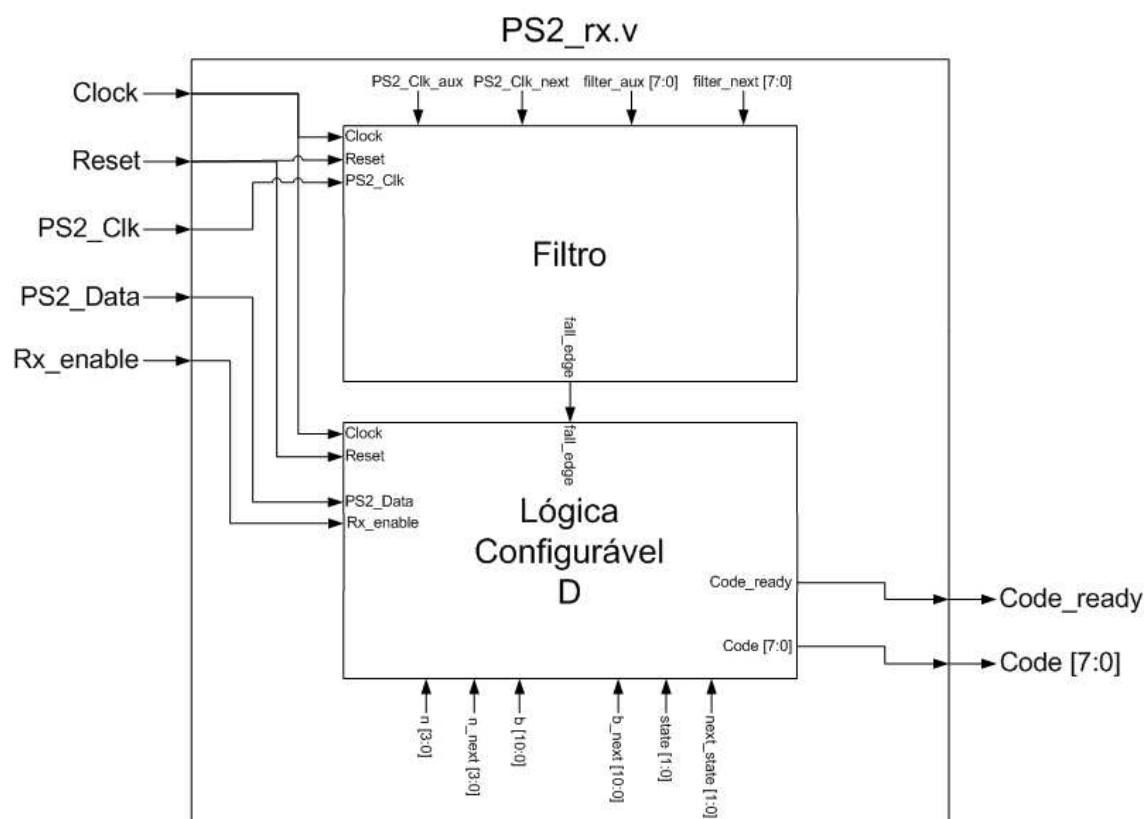


Figura A.9 - Módulo "PS2_rx.v".

Tabela A.12 - Sinais do módulo "PS2_rx.v".

Entradas:	
Clock	Relógio interno de 50MHz.
Reset	Sinal externo de reset (activo a '1').
Rx_enable	Activa a recepção de dados do rato PS2.
Bidireccionais:	
PS2_Clk	Linha de relógio do barramento PS2.
PS2_Data	Linha de dados do barramento PS2.
Saídas:	
Code_ready	Indica a conclusão de uma recepção do rato PS2.
Code [7:0]	Byte recebido do rato PS2.
Ligações Internas:	
fall_edge	Indica as transições de '1' para '0' de PS2_Clk.
Registos Internos:	
PS2_Clk_aux e PS2_Clk_next	Usado no filtro aplicado ao sinal PS2_Clk.
filter_aux [7:0] e filter_next [7:0]	Usado no filtro aplicado ao sinal PS2_Clk.
state [1:0] e state_next [1:0]	Usado na FSM usada para receber os dados do rato PS2.
n [3:0] e n_next [3:0]	Indica o número de bits que ainda falta receber.

b [10:0] e b_next [10:0]	Armazena temporariamente os dados recebidos.
--------------------------	--

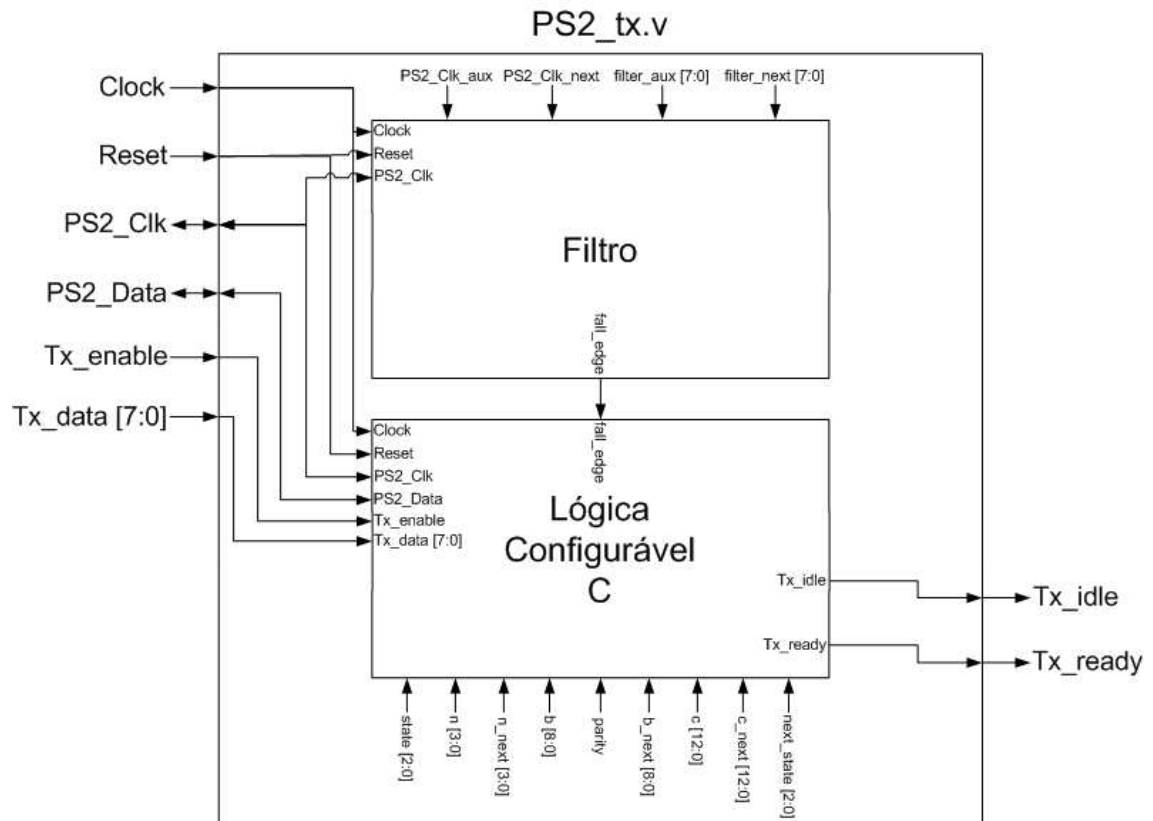


Figura A.10 - Módulo "PS2_tx.v".

Tabela A.13 - Sinais do módulo "PS2_tx.v".

Entradas:	
Clock	Relógio interno de 50MHz.
Reset	Sinal externo de reset (activo a '1').
Tx_enable	Activa a transmissão de dados para o teclado PS2.
Tx_data [7:0]	Byte a transmitir para o rato PS2.
Bidireccionais:	
PS2_Clk	Linha de relógio do barramento PS2.
PS2_Data	Linha de dados do barramento PS2.
Saídas:	
Tx_idle	Indica quando o módulo não está a transmitir dados para o rato PS2.
Tx_ready	Indica a conclusão da transmissão para o rato PS2.
Ligações Internas:	
fall_edge	Indica as transições de '1' para '0' de PS2_Clk.
Registos Internos:	
PS2_Clk_aux e PS2_Clk_next	Usado no filtro aplicado ao sinal PS2_Clk.
filter_aux [7:0] e filter_next [7:0]	Usado no filtro aplicado ao sinal PS2_Clk.

state [2:0] e state_next [2:0]	Usado na FSM usada para enviar os dados do rato PS2.
n [3:0] e n_next [3:0]	Indica o número de bits que ainda falta enviar.
b [8:0] e b_next [8:0]	Armazena temporariamente os dados a enviar.
c [12:0] e c_next [12:0]	Utilizados como "delay" na "Start Condition".
parity	Usado para calcular o bit de paridade.

Design Summary:

Tabela A.14 - Ocupação de recursos dos módulos do rato PS/2.

Recurso	Utilizados	Disponíveis	Percentagem Ocupada
<i>Total Number of Slice Flip-Flops</i>	82	9312	0.88%
<i>Total Number of 4 input LUTs</i>	136	9312	1.45%
<i>Number Used as Logic</i>	135		
<i>Number Used as Route-Thru</i>	1		
<i>Total Number of Occupied Slices</i>	80	4656	1.72%
<i>Number of Bounded IOBs</i>	26	232	11.21%
<i>Number of BUFGMUXs</i>	1	24	4.17%

Para a utilização do módulo "Mouse.v" são necessários os módulos "PS2.v", "PS2_rx.v" e "PS2_tx.v", para além da ligação aos seguintes recursos da FPGA: Relógio de 50 MHz, botão/sinal de reset, linhas de dados e relógio do conector PS2.

Todas as ligações aos recursos da FPGA estão especificadas no ficheiro Keyboard.ucf. Caso seja alterado o nome de alguma das entradas ou saídas especificadas no ficheiro ".ucf" é necessário alterar o nome da "NET" nesse ficheiro, pelo que a alteração não é recomendada.

No módulo "PS2_rx.v" são processadas todas as comunicações (recepção apenas) do rato PS2. Este módulo envia depois o byte lido (Code[7:0]) do rato para o módulo "Mouse.v", que processa todos os dados recebidos.

O sinal que controla a leitura dos bits denomina-se fall_edge e vai a '1' durante um ciclo de relógio sempre que é detectada uma transição de '1' para '0' em PS2_Clk_next (é usado o registo PS2_Clk_aux para detectar essa transição).

A FSM usada para receber os bits possui apenas três estados e opera de forma assíncrona com o relógio. O estado 0 é um estado de "idle", passando para o estado 1 sempre que é detectada uma "Start Condition". No estado 1 são recebidos os onze bits que compõe a trama a receber. Os bits são guardados no registo b_next sendo que o código se encontra entre o bit 8 e 1 inclusive. O número de bits que faltam receber fica guardado em n_next. Quando os bits são todos guardados passa-se ao estado 2 que existe para colocar Code_ready a '1' e para utilizar mais um ciclo de relógio de forma a actualizar os registos (síncronos com o relógio) b e n, a partir de b_next e n_next, regressando ao estado 0.

No módulo “PS2_tx.v” são processadas todas as comunicações (de envio) do rato PS2. Após um impulso de Tx_enable, os 8 bits de Tx_data são enviados juntamente com o bit de paridade (CRC) que é calculado automaticamente. O final da transmissão é assinalado por um impulso com a duração de um ciclo de relógio de Tx_ready.

O sinal que controla a escrita dos bits denomina-se fall_edge e funciona do mesmo modo que em “PS2_rx.v”, sendo que os bits são escritos na descida do relógio, e lidos no teclado na subida do mesmo.

A FSM usada para enviar dados possui cinco estados e funciona assincronamente. O estado 0 é o estado de “idle” passando para o estado 1 quando Tx_enable é activo. No estado 1 o sinal PS2_Clk é mantido a ‘0’ durante 8191 ciclos de relógio ($2^{13} - 1$) de forma a transmitir uma “Start Condition”, passando de seguida ao estado 2. No estado 2 a linha PS2_Clk é libertada, assumindo controlo da linha PS2_data e colocando-a a ‘0’ até ao primeiro “fall_edge” quando passa para o estado 3. O estado 3 é o estado de escrita dos bits, onde cada um dos nove bits é escrito a cada “fall_edge”. Quando todos os bits tiverem sido enviados passa-se ao estado 4, que existe para colocar Tx_ready a ‘1’ e para utilizar mais um ciclo de relógio de forma a actualizar os registos (síncronos com o relógio) b e n, a partir de b_next e n_next, regressando ao estado 0.

Os registos b e b_next guardam os bits a enviar, n e n_next guardam o número de bits por enviar e c e c_next são utilizados como “delay” da “Start Condition”.

Existe ainda a saída Tx_idle deste módulo que está a ‘1’ sempre que não se está a transmitir, servindo como Rx_enable no módulo “PS2_rx.v”.

Em ambos os módulos (“PS2_rx.v” e “PS2_tx.v”) existe um filtro para as transições do relógio de PS2, de forma a eliminar os efeitos de uma transição lenta ou de bouncing. Deste modo, para o valor de PS2_Clk_next alterar são necessários ciclos de relógio em que o valor de PS2_Clk seja igual e contrário ao valor actual de PS2_Clk_next, caso contrário o valor mantém-se.

O módulo “PS2.v” gere todas as trocas de informação, evitando conflitos entre transmissão e recepção.

No módulo “Mouse.v” são processadas todas as informações provenientes do rato, de modo que estas se tornem mais simples de interpretar. Sempre que todas as informações estão prontas, o sinal Mouse_ready emite um impulso com um ciclo de relógio de duração. Os bits de Buttons representam o estado dos botões do rato, x e y indicam o sentido e variação do movimento.

btn_aux e btn_next servem para armazenar o estado dos botões do rato. x_aux, y_aux, x_next e y_next servem para armazenar o sentido e amplitude do movimento.

O bit 0 de Buttons representa o botão da esquerda do rato, o bit 1 representa o botão direito e o bit 2 representa o botão central.

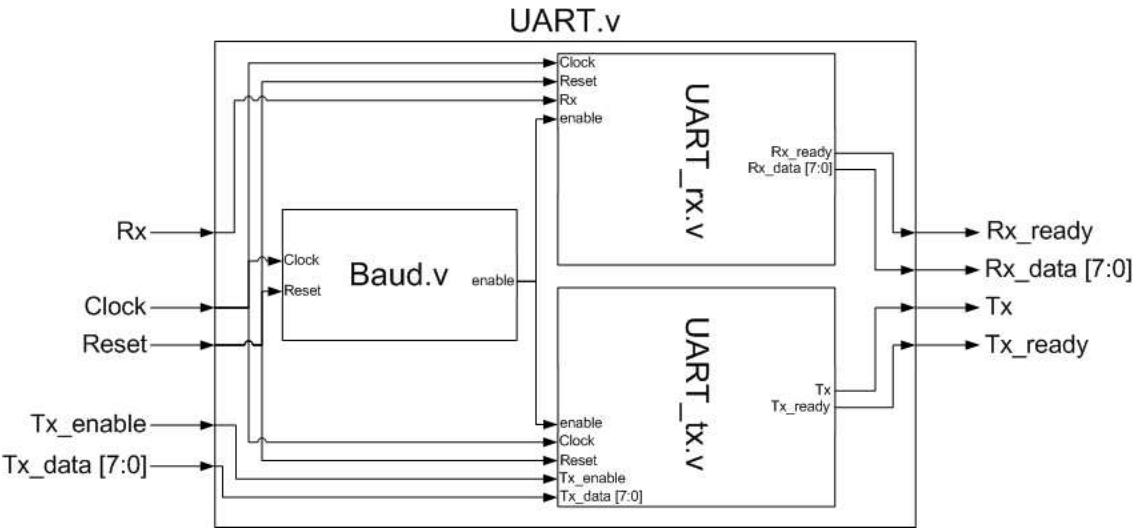
Nos sinais x e y, o bit mais significativo (8) representa o sentido, e os restantes a amplitude do movimento. Se o bit de direcção for igual a ‘1’ representa um movimento para a esquerda ou para baixo conforme o sinal. Se o bit for igual a ‘0’ representa um movimento para a direita ou para cima.

A FSM para processar os dados possui sete estados. Os dois primeiros (0 e 1) só são executados após um “reset” ou na inicialização e transmitem ao rato o comando para iniciar o funcionamento. Os estados 2, 3, 4 e 5 servem para receber e processar os 3 bytes que o rato envia de cada vez que detecta movimento ou botões premidos. O último estado (6) serve para

colocar a *flag* Mouse_ready a '1'e para utilizar mais um ciclo de relógio de forma a actualizar os registos.

Podem encontrar-se um pequeno exemplo de utilização do módulo na página do módulo sobre VGA (Modo Gráfico).

A.5 - UART



Configuração A.11 - Módulo “UART.v”.

Tabela A.15 - Sinais do módulo “UART.v”.

Entradas:	
Clock	Relógio interno de 50MHz.
Reset	Sinal externo de reset (activo a '1').
Tx_enable	Activa a transmissão de dados através da porta série.
Tx_data [7:0]	Byte a transmitir através da porta série.
Rx	Linha de recepção de dados da porta série.
Saídas:	
Tx_ready	Indica a conclusão da transmissão através da porta série.
Rx_ready	Indica a conclusão de uma recepção através da porta série.
Rx_data [7:0]	Byte recebido através da porta série.
Tx	Linha de transmissão de dados da porta série.
Ligações Internas:	
enable	Indica quando se deve analisar o estado das linhas de transmissão de dados.

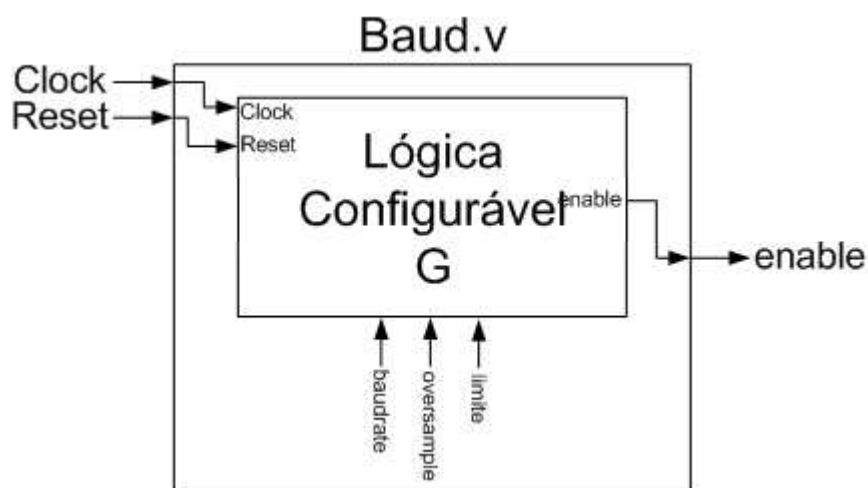


Figura A.12 - Módulo "Baud.v".

Tabela A.16 - Sinais do módulo "Baud.v".

Entradas:	
Clock	Relógio interno de 50MHz.
Reset	Sinal externo de reset (activo a '1').
Saídas:	
enable	Indica quando se deve analisar o estado das linhas de transmissão de dados.
Parâmetros Internos:	
baudrate	Especifica a "Baudrate" a utilizar nas comunicações.
oversample	Especifica o "Oversample" a utilizar nas comunicações.
limite	Calcula, através do baudrate e oversample, quando se deve analisar as linhas de transmissão de dados.

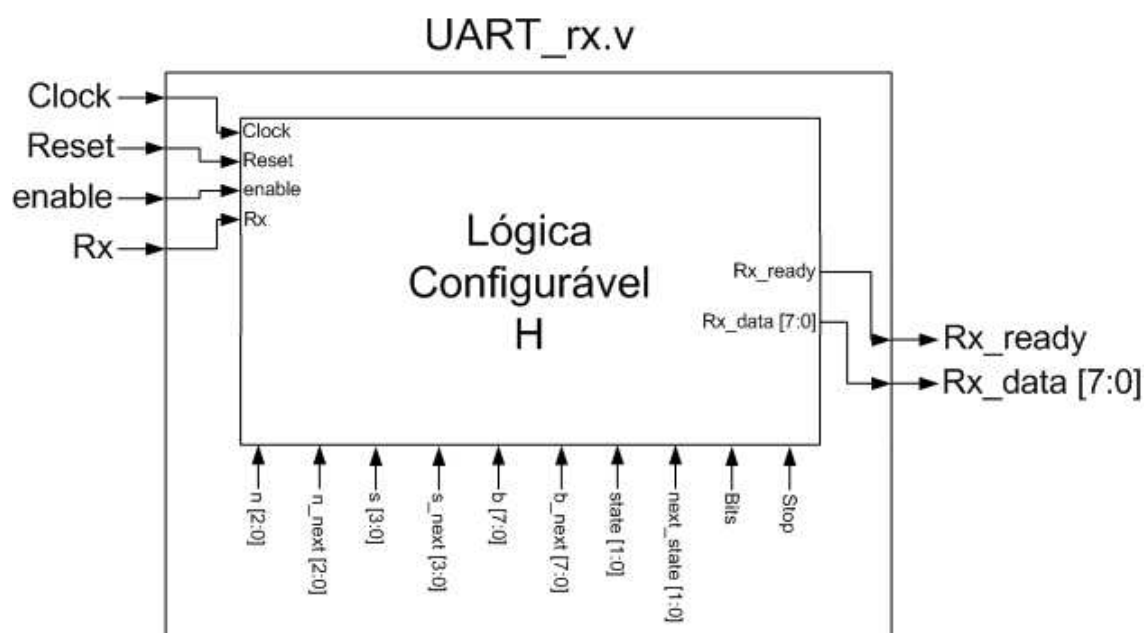


Figura A.13 - Módulo "UART_rx.v".

Tabela A.17 - Sinais do módulo "UART_rx.v".

Entradas:	
Clock	Relógio interno de 50MHz.
Reset	Sinal externo de reset (activo a '1').
enable	Indica quando se deve analisar o estado das linhas de transmissão de dados.
Rx	Linha de recepção de dados da porta série.
Saídas:	
Rx_ready	Indica a conclusão de uma recepção através da porta série.
Rx_data [7:0]	Byte recebido através da porta série.
Parâmetros Internos:	
Bits	Indica o número de bits a receber.
Stop	Indica o número de "enables" necessários para a "Stop Condition".
Registos Internos:	
state [1:0] e state_next [1:0]	Usado na FSM usada para receber os dados através da porta série.
n [2:0] e n_next [2:0]	Indica o número de bits já recebidos.
s [3:0] e s_next [3:0]	Conta o número de "enables" para a "Start Condition", bits recebidos e "Stop Condition".
b [7:0] e b_next [7:0]	Armazena temporariamente os dados recebidos.

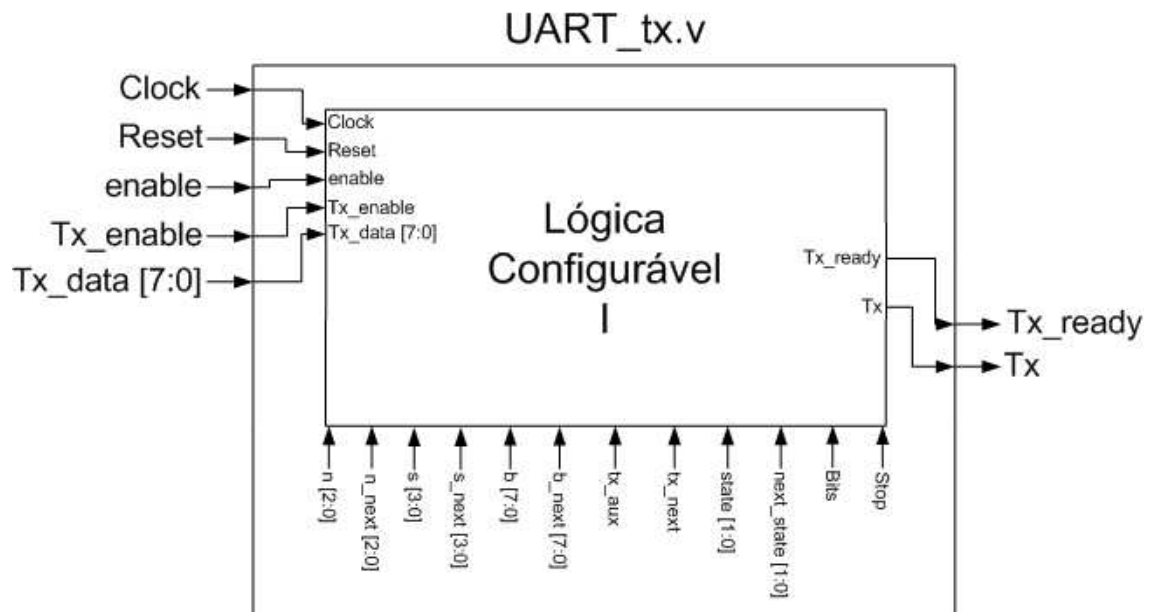


Figura A.14 - Módulo "UART_tx.v".

Tabela A.18 - Sinais do módulo "UART_tx.v".

Entradas:	
Clock	Relógio interno de 50MHz.
Reset	Sinal externo de reset (activo a '1').
enable	Indica quando se deve analisar o estado das linhas de transmissão de dados.
Tx_enable	Activa a transmissão de dados através da porta série.
Tx_data [7:0]	Byte a transmitir através da porta série.
Saídas:	
Tx_ready	Indica a conclusão da transmissão através da porta série.
Tx	Linha de transmissão de dados da porta série.
Parâmetros Internos:	
Bits	Indica o número de bits a receber.
Stop	Indica o número de "enables" necessários para a "Stop Condition".
Registos Internos:	
tx_aux e tx_next	Armazena temporariamente o bit recebido.
state [1:0] e state_next [1:0]	Usado na FSM usada para receber os dados através da porta série.
n [2:0] e n_next [2:0]	Indica o número de bits já recebidos.
s [3:0] e s_next [3:0]	Conta o número de "enables" para a "Start Condition", bits recebidos e "Stop Condition".
b [7:0] e b_next [7:0]	Armazena temporariamente os dados recebidos.

Design Summary:**Tabela A.19 - Ocupação de recursos dos módulos de comunicação série.**

Recurso	Utilizados	Disponíveis	Porcentagem Ocupada
<i>Total Number of Slice Flip-Flops</i>	44	9312	0.47%
<i>Total Number of 4 input LUTs</i>	121	9312	1.30%
<i>Number Used as Logic</i>	114		
<i>Number Used as Route-Thru</i>	7		
<i>Total Number of Occupied Slices</i>	65	4656	1.40%
<i>Number of Bounded IOBs</i>	23	232	9.91%
<i>Number of BUFGMUXs</i>	1	24	4.17%

Para a utilização do módulo “UART.v” são necessários os módulos: “Baud.v”, “UART_tx.v” e “UART_rx.v”, para além da ligação aos seguintes recursos da FPGA: Relógio de 50 MHz, botão/sinal de reset, linhas de transmissão e recepção RS-232.

No módulo “s3_UART_top.v” é demonstrado um pequeno exemplo de utilização deste módulo que, para além dos recursos mencionados anteriormente, utiliza ainda os recursos necessários ao funcionamento dos módulos “LCD.v” e “Keyboard.v” (para mais informação consultar os manuais destes módulos). Neste exemplo, quando é premida uma tecla do teclado ligado à FPGA é impresso o respectivo carácter no LCD e no PC ligado pela UART. Quando é premida uma tecla do PC, é escrito no LCD o respectivo carácter. Todas as ligações aos recursos da FPGA estão especificadas no ficheiro UART.ucf. Caso seja alterado o nome de alguma das entradas ou saídas especificadas no ficheiro “.ucf” é necessário alterar o nome da “NET” nesse ficheiro, pelo que a alteração não é recomendada.

Por predefinição está atribuído no ficheiro “.ucf” que as linhas de RS-232 a utilizar são as do conector fêmea do dispositivo. Caso se queira utilizar o conector macho, é só comentar esta atribuição, e activar as linhas respectivas ao conector macho.

No módulo “Baud.v” é gerado um impulso (“enable”) sempre que se deve amostrar as linhas de transmissão e recepção. Esse impulso é calculado através do relógio e dos parâmetros Baudrate e Oversample, que podem ser alterados e apresentam o valor pré-definido de 19200 e 16, respectivamente.

Quer no módulo “UART_rx.v” quer no módulo “UART_tx.v” é possível alterar o número de bits a receber e o número de enables necessários na “Stop Condition”. Para isso basta alterar os parâmetros “Bits” e “Stop”, que por defeito possuem os valores de 8 e 16 respectivamente, e o tamanho dos registos respectivos.

No módulo “UART_rx.v” são processadas todas as comunicações recebidas através do conector RS-232. Sempre que é recebida toda a informação é emitido um impulso de um ciclo de relógio em Rx_ready e os dados encontram-se presentes em Rx_data.

A FSM utilizada na recepção possui quatro estados. O estado 0 é um estado de “idle” passando ao estado 1 sempre que for detectada uma “Start Condition”. No estado 1 são contados (em s e s_next) o número de enables necessários para a “Start Condition” passando de seguida para o estado 2, onde se guardam os bits recebidos (são guardados em b e b_next, o número de bits já recebidos em n e n_next e o número de enables para cada bit em s e

s_next). Quando todos os bits forem recebidos passa-se ao estado 3 onde se processa a "Stop Condition" e regressa-se ao estado 0.

No módulo "UART_tx.v" são processadas todas as comunicações a transmitir pelo conector RS-232. Para activar a transmissão utiliza-se o sinal Tx_enable e transmite-se os bits presentes em Tx_data. Quando a transmissão estiver completa, isso é assinalado através de Tx_ready.

A FSM utilizada neste módulo é semelhante à utilizada no módulo UART_rx.v, sendo a única diferença no estado 2, que é o estado de transmissão dos bits.

A.6 - VGA (Modo Gráfico)

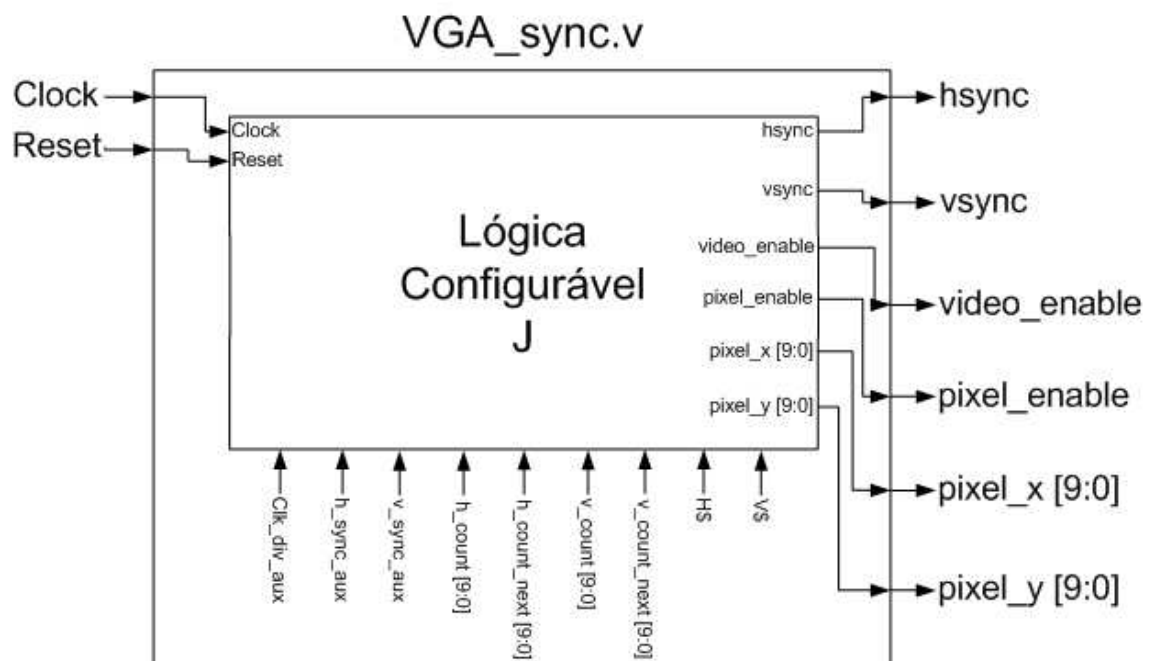


Figura A.15 - Módulo "VGA_sync.v".

Tabela A.20 - Sinais do módulo "VGA_sync.v".

Entradas:	
Clock	Relógio interno de 50MHz.
Reset	Sinal externo de reset (activo a '1').
Saídas:	
hsync	"Horizontal Sync" do conector VGA.
vsync	"Vertical Sync" do conector VGA.
video_enable	Indica se o pixel actual é um pixel visível.
pixel_enable	Indica quando um pixel está activo.
pixel_x [9:0]	Indica a coordenada x do pixel actual.
pixel_y [9:0]	Indica a coordenada y do pixel actual.
Parâmetros Internos:	
H\$ (\$ = D, F, B, R)	Parâmetros horizontais do modo VGA.
V\$ (\$ = D, F, B, R)	Parâmetros horizontais do modo VGA.
Registos Internos:	

Clk_div_aux	Usado no "Clock Divider".
hsync_aux	Usado no sinal hsync.
vsync_aux	Usado no sinal vsync.
h_count [9:0] e h_count_next [9:0]	Usados para determinar quando hsync deve ir a '1'.
v_count [9:0] e v_count_next [9:0]	Usados para determinar quando vsync deve ir a '1'.

Design Summary:

Tabela A.21 - Ocupação de recursos do módulo "VGA_sync.v".

Recurso	Utilizados	Disponíveis	Percentagem Ocupada
<i>Total Number of Slice Flip-Flops</i>	23	9312	0.25%
<i>Total Number of 4 input LUTs</i>	56	9312	0.60%
<i>Number Used as Logic</i>	38		
<i>Number Used as Route-Thru</i>	18		
<i>Total Number of Occupied Slices</i>	31	4656	0.67%
<i>Number of Bounded IOBs</i>	26	232	11.21%
<i>Number of BUFGMUXs</i>	1	24	4.17%

Para a utilização do módulo "VGA_sync.v" é necessária a ligação aos seguintes recursos da FPGA: Relógio de 50 MHz, botão/sinal de reset, sinais hsync, vsync, R, G e B do conector VGA da FPGA.

No módulo "s3_VGA_top.v" é demonstrado um pequeno exemplo de utilização deste módulo, juntamente com o módulo "Mouse.v" (para mais informação consultar o manual deste módulo). Todas as ligações aos recursos da FPGA estão especificadas no ficheiro VGA.ucf. Caso seja alterado o nome de alguma das entradas ou saídas especificadas no ficheiro ".ucf", é necessário alterar o nome da "NET" nesse ficheiro, pelo que a alteração não é recomendada.

O modo de vídeo utilizado neste módulo é baseado na norma VGA 640 x 480, usando uma "pixel rate" de 25MHz. Uma vez que a "pixel rate" é metade da frequência do relógio interno da FPGA, a geração deste sinal torna-se bastante fácil, sendo apenas a inversão do sinal Clk_div_aux sempre que há uma subida do relógio. Este sinal dará também origem ao sinal de saída do módulo, pixel_enable que indica quando está activo o pixel.

Cada pixel possui uma coordenada cartesiana, para que seja fácil identificá-los individualmente. Estas coordenadas estão presentes nas saídas pixel_x e pixel_y. Os valores de coordenadas apenas representam pixéis válidos/visíveis enquanto video_enable for igual a '1'. Ou seja entre 0 e 639 para pixel_x e entre 0 e 479 para pixel_y.

hsync e vsync são processados através dos parâmetros de vídeo definidos pela norma VGA.

O módulo "VGA_sync.v" não atribui valores aos sinais R, G e B. Apenas efectua a sincronização do monitor no modo referido e indica as coordenadas do pixel actual. O módulo

superior, construído conforme os objectivos pretendidos, actua sobre as saídas R, G e B, atribuindo através destes sinais uma das oito cores disponíveis ao respectivo pixel.

A.7 - VGA (Modo Texto)

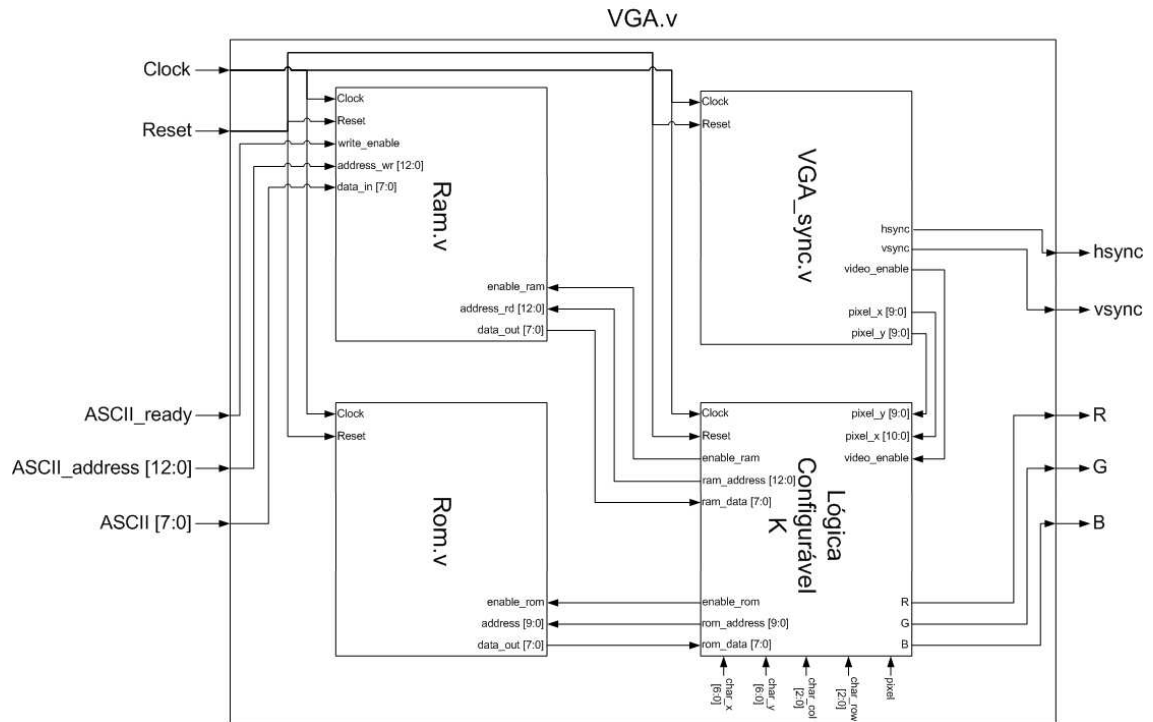


Figura A.16 - Módulo "VGA.v".

Tabela A.22 - Sinais do módulo "VGA.v".

Entradas:	
Clock	Relógio interno de 50MHz.
Reset	Sinal externo de reset (activo a '1').
ASCII_ready	Indica que está pronto a ser escrito um novo carácter na memória RAM do controlador VGA de texto.
ASCII_address [12:0]	Endereço do carácter a escrever na memória RAM do controlador VGA de texto.
ASCII [7:0]	Código ASCII do carácter a escrever na memória RAM do controlador VGA de texto.
Saídas:	
hsync	"Horizontal Sync" do conector VGA.
vsync	"Vertical Sync" do conector VGA.
R	"Red" do conector VGA.
G	"Green" do conector VGA.
B	"Blue" do conector VGA.
Ligações Internas:	
enable_ram	Activa a memória RAM.
enable_rom	Activa a memória ROM.

ram_address [12:0]	Endereço de leitura da memória RAM.
rom_address [9:0]	Endereço de leitura da memória ROM.
ram_data [7:0]	Dados lidos da memória RAM.
rom_data [7:0]	Dados lidos da memória ROM.
video_enable	Indica se o pixel actual é um pixel visível.
pixel_x [9:0]	Indica a coordenada x do pixel actual.
pixel_y [9:0]	Indica a coordenada y do pixel actual.
Registos Internos:	
char_x [6:0]	Coordenada x da grelha de caracteres.
char_y [5:0]	Coordenada y da grelha de caracteres.
char_col [2:0]	Coluna actual dentro da célula do carácter.
char_row [2:0]	Linha actual dentro da célula do carácter.
pixel	Indica se o pixel actual pertence ao carácter ou ao fundo.

Design Summary:

Tabela A.23 - Ocupação de recursos dos módulos do modo de texto em VGA.

Recurso	Utilizados	Disponíveis	Percentagem Ocupada
<i>Total Number of Slice Flip-Flops</i>	23	9312	0.25%
<i>Total Number of 4 input LUTs</i>	74	9312	0.79%
<i>Number Used as Logic</i>	50		
<i>Number Used as Route-Thru</i>	24		
<i>Total Number of Occupied Slices</i>	40	4656	0.86%
<i>Number of Bounded IOBs</i>	29	232	12.50%
<i>Number of BUFGMUXs</i>	1	24	4.17%
<i>Number of RAMB16s</i>	5	20	25.00%
<i>Number of MULTI18X18SIOs</i>	1	20	5.00%

Para a utilização do módulo “VGA.v” são necessários os módulos “VGA_sync.v”, “ram.v” e “rom.v”, para além da ligação aos seguintes recursos da FPGA: Relógio de 50 MHz, botão/sinal de reset, sinais hsync, vsync, R, G e B do conector VGA.

No módulo “s3_VGA_top.v” é demonstrado um pequeno exemplo de utilização deste módulo que, para além dos recursos mencionados anteriormente, utiliza ainda os recursos necessários ao funcionamento do módulo “Keyboard.v” (para mais informação consultar o manual deste módulo). Todas as ligações aos recursos da FPGA estão especificadas no ficheiro VGA.ucf. Caso seja alterado o nome de alguma das entradas ou saídas especificadas no ficheiro “.ucf”, é necessário alterar o nome da “NET” nesse ficheiro, pelo que a alteração não é recomendada.

A sincronização dos sinais de vídeo hsync e vsync é feita pelo módulo “VGA_sync.v” que é descrito em pormenor na secção sobre VGA (Modo Gráfico). Este módulo devolve também as coordenadas actuais do pixel a ser processado, através de pixel_x e pixel_y.

Ao utilizar este módulo estamos a utilizar uma configuração de 640 x 480, sendo que cada carácter tem as dimensões de 8 x 8. Assim temos 80 colunas e 60 linhas de caracteres, originando um total de 4800 caracteres por ecrã.

O módulo “ram.v” implementa uma memória RAM com 8192 posições de 8 bits cada uma, pois esta memória vai armazenar o código ASCII de cada uma das posições do monitor. Para consultar a memória é utilizado o fio ram_address, que indica a coordenada do carácter pretendido, calculada através das coordenadas devolvidas pelo módulo “VGA_sync.v”, recorrendo também aos fios char_x, char_y, char_col e char_row, que representam as coordenadas x e y do carácter e a coluna/linha actual do carácter, respectivamente. O código ASCII é devolvido através de ram_data. Para alterar esta memória, basta utilizar os sinais ASCII, ASCII_ready e ASCII_address. Estes três sinais são entradas do módulo “VGA.v”. O sinal de ASCII_ready funciona como “write enable” da memória, o sinal ASCII_address indica a posição da memória, e ASCII os dados a guardar.

O módulo “rom.v” implementa uma memória ROM com 1024 posições de 8bits que, após a inicialização, contém o *bitmap* dos caracteres a imprimir no monitor. Durante a inicialização o módulo vai buscar os valores a guardar ao ficheiro bitmap.txt. O endereço rom_address de 10 bits funciona da seguinte forma: os 7 bits mais significativos representam o código ASCII do carácter (excluindo o bit mais significativo, uma vez que estamos a usar a tabela ASCII simples e não a estendida), e os 3 bits menos significativos indicam a linha do carácter (cada carácter tem 8 linhas e 8 colunas).

A.8 - SVGA (Modo Gráfico)

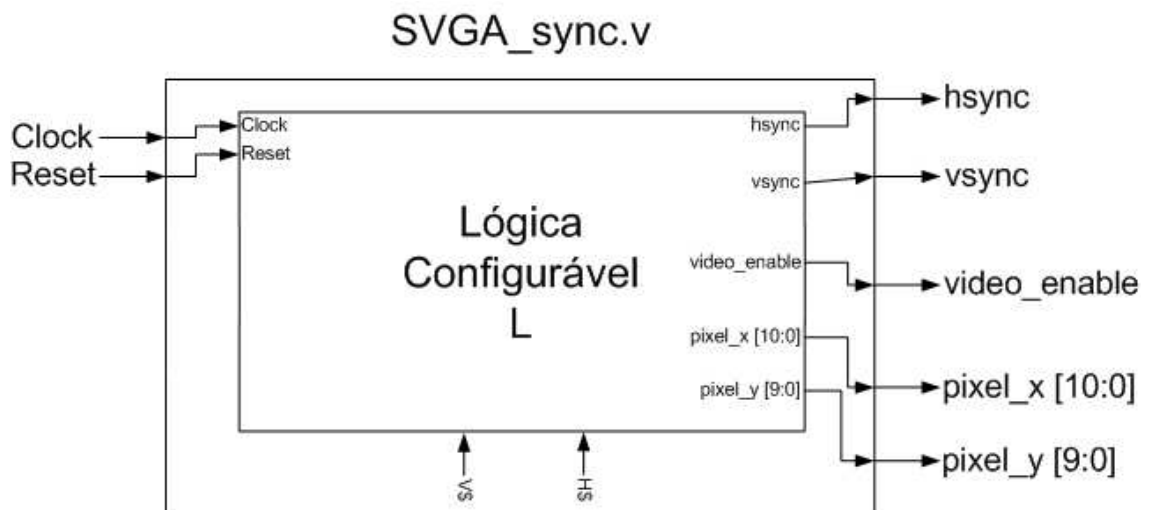


Figura A.17 - Módulo “SVGA_sync.v”.

Tabela A.24 - Sinais do módulo "SVGA_sync.v".

Entradas:	
Clock	Relógio interno de 50MHz.
Reset	Sinal externo de reset (activo a '1').
Saídas:	
hsync	"Horizontal Sync" do conector VGA.
vsync	"Vertical Sync" do conector VGA.
video_enable	Indica se o pixel actual é um pixel visível.
pixel_x [10:0]	Indica a coordenada x do pixel actual.
pixel_y [9:0]	Indica a coordenada y do pixel actual.
Parâmetros Internos:	
H\$ (\$ = D, F, B, R)	Parâmetros horizontais do modo SVGA.
V\$ (\$ = D, F, B, R)	Parâmetros horizontais do modo SVGA.

Design Summary:

Tabela A.25 - Ocupação de recursos do módulo "SVGA_sync.v".

Recurso	Utilizados	Disponíveis	Percentagem Ocupada
<i>Total Number of Slice Flip-Flops</i>	26 (+3)	9312	0.28%
<i>Total Number of 4 input LUTs</i>	69 (+13)	9312	0.74%
<i>Number Used as Logic</i>	50		
<i>Number Used as Route-Thru</i>	19		
<i>Total Number of Occupied Slices</i>	40 (+9)	4656	0.86%
<i>Number of Bounded IOBs</i>	10	232	4.31%
<i>Number of BUFGMUXs</i>	1	24	4.17%

Para a utilização do módulo "SVGA_sync.v" é necessária a ligação aos seguintes recursos da FPGA: Relógio de 50 MHz, botão/sinal de reset, sinais hsync, vsync, R, G e B do conector VGA da FPGA.

No módulo "SVGA_top.v" é demonstrado um pequeno exemplo de utilização deste módulo. Todas as ligações aos recursos da FPGA estão especificadas no ficheiro SVGA.ucf. Caso seja alterado o nome de alguma das entradas ou saídas especificadas no ficheiro ".ucf", é necessário alterar o nome da "NET" nesse ficheiro, pelo que a alteração não é recomendada.

O modo de vídeo utilizado neste módulo é baseado na norma SVGA 800 x 600, usando uma "pixel rate" de 50MHz, uma vez que a "pixel rate" é igual à frequência do relógio interno da FPGA.

Cada pixel possui uma coordenada cartesiana, para que seja fácil identificá-los individualmente. Estas coordenadas estão presentes nas saídas pixel_x e pixel_y. Os valores de coordenadas apenas representam pixels válidos/visíveis enquanto video_enable for igual a '1'. Ou seja entre 0 e 799 para pixel_x e entre 0 e 599 para pixel_y.

hsync e vsync são processados através dos parâmetros de vídeo definidos pela norma VGA.

O módulo “SVGA_sync.v” não atribui valores aos sinais R, G e B. Apenas efectua a sincronização do monitor no modo referido e indica as coordenadas do pixel actual. O módulo superior, construído conforme os objectivos pretendidos, é que actua sobre as saídas R, G e B, atribuindo através destes sinais uma das oito cores disponíveis ao respectivo pixel.

A.9 - SVGA (Modo Texto)

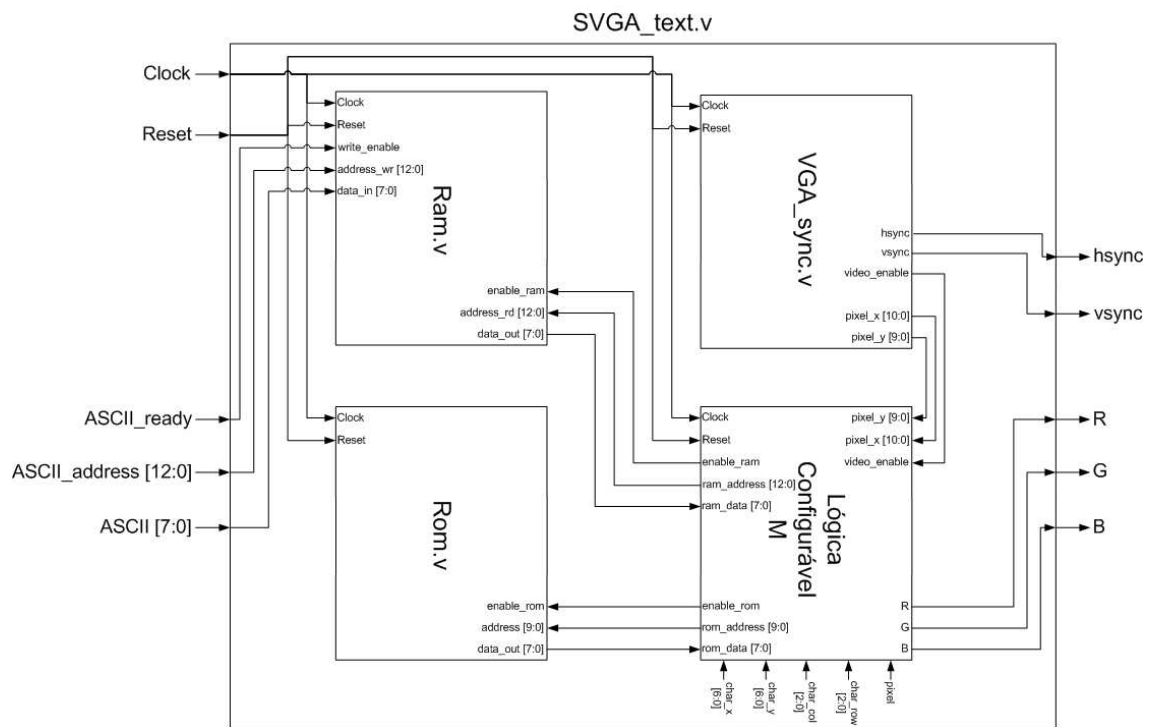


Figura A.18 - Módulo “SVGA_text.v”.

Tabela A.26 - Sinais dos módulos “SVGA_text.v”.

Entradas:	
Clock	Relógio interno de 50MHz.
Reset	Sinal externo de reset (activo a '1').
ASCII_ready	Indica que está pronto a ser escrito um novo carácter na memória RAM do controlador VGA de texto.
ASCII_address [12:0]	Endereço do carácter a escrever na memória RAM do controlador VGA de texto.
ASCII [7:0]	Código ASCII do carácter a escrever na memória RAM do controlador VGA de texto.
Saídas:	
hsync	"Horizontal Sync" do conector VGA.
vsync	"Vertical Sync" do conector VGA.
R	"Red" do conector VGA.
G	"Green" do conector VGA.
B	"Blue" do conector VGA.

Ligações Internas:	
enable_ram	Activa a memória RAM.
enable_rom	Activa a memória ROM.
ram_address [12:0]	Endereço de leitura da memória RAM.
rom_address [9:0]	Endereço de leitura da memória ROM.
ram_data [7:0]	Dados lidos da memória RAM.
rom_data [7:0]	Dados lidos da memória ROM.
video_enable	Indica se o pixel actual é um pixel visível.
pixel_x [10:0]	Indica a coordenada x do pixel actual.
pixel_y [9:0]	Indica a coordenada y do pixel actual.
Registos Internos:	
char_x [6:0]	Coordenada x da grelha de caracteres.
char_y [5:0]	Coordenada y da grelha de caracteres.
char_col [2:0]	Coluna actual dentro da célula do carácter.
char_row [2:0]	Linha actual dentro da célula do carácter.
pixel	Indica se o pixel actual pertence ao carácter ou ao fundo.

Design Summary:

Tabela A.27 - Ocupação de recursos dos módulos do modo de texto em SVGA.

Recurso	Utilizados	Disponíveis	Percentagem Ocupada
<i>Total Number of Slice Flip-Flops</i>	45 (+22)	9312	0.48%
<i>Total Number of 4 input LUTs</i>	101 (+26)	9312	1.08%
<i>Number Used as Logic</i>	63 (+13)		
<i>Number Used as Route-Thru</i>	37 (+13)		
<i>Total Number of Occupied Slices</i>	54 (+14)	4656	1.16%
<i>Number of Bounded IOBs</i>	9	232	3.88%
<i>Number of BUFGMUXs</i>	1	24	4.17%
<i>Number of RAMB16s</i>	5	20	25.00%
<i>Number of MULTI18X18SIOs</i>	1	20	5.00%

Para a utilização do módulo “SVGA_text.v” são necessários os módulos “SVGA_sync.v”, “ram.v” e “rom.v”, para além da ligação aos seguintes recursos da FPGA: Relógio de 50 MHz, botão/sinal de reset, sinais hsync, vsync, R, G e B do conector VGA.

No módulo “SVGA_text_top.v” é demonstrado um pequeno exemplo de utilização deste módulo que, para além dos recursos mencionados anteriormente, utiliza ainda os recursos necessários ao funcionamento do módulo “Keyboard.v” (para mais informação consultar o manual deste módulo). Todas as ligações aos recursos da FPGA estão especificadas no ficheiro VGA.ucf. Caso seja alterado o nome de alguma das entradas ou saídas especificadas no ficheiro “.ucf”, é necessário alterar o nome da “NET” nesse ficheiro, pelo que a alteração não é recomendada.

A sincronização dos sinais de vídeo hsync e vsync é feita pelo módulo “SVGA_sync.v” que é descrito em pormenor na secção sobre SVGA (Modo Gráfico). Este módulo devolve também as coordenadas actuais do pixel a ser processado, através de pixel_x e pixel_y.

Ao utilizar este módulo estamos a utilizar uma configuração de 800 x 600, sendo que cada carácter tem as dimensões de 8 x 8. Assim temos 100 colunas e 75 linhas de caracteres, originando um total de 7500 caracteres por ecrã.

O módulo “ram.v” implementa uma memória RAM com 8192 posições, cada uma com oito bits, pois esta memória vai armazenar o código ASCII de cada uma das posições do monitor. Para consultar a memória é utilizado o fio ram_address, que indica a coordenada do carácter pretendido, calculada através das coordenadas devolvidas pelo módulo SVGA_sync.v, recorrendo também aos fios char_x, char_y, char_col e char_row, que representam a coordenada x/y do carácter e a coluna/linha actual do carácter, respectivamente. O código ASCII é devolvido através de ram_data. Para alterar esta memória basta utilizar os sinais ASCII, ASCII_ready e ASCII_address. Estes três sinais são entradas do módulo “SVGA_text.v”. O sinal de ASCII_ready funciona como “write enable” da memória, o sinal ASCII_address indica a posição da memória e ASCII os dados a guardar.

O módulo “rom.v” implementa uma memória ROM com 1024 posições de 8bits que, após a inicialização, contém o *bitmap* dos caracteres a imprimir no monitor. Durante a inicialização o módulo vai buscar os valores a guardar ao ficheiro bitmap.txt. O endereço rom_address de 10 bits funciona da seguinte forma: os 7 bits mais significativos representam o código ASCII do carácter (excluindo o bit mais significativo, uma vez que estamos a usar a tabela ASCII simples e não a estendida), e os 3 bits menos significativos indicam a linha do carácter (cada carácter tem 8 linhas e 8 colunas).

A.10 - VGA e SVGA (Modo Gráfico)

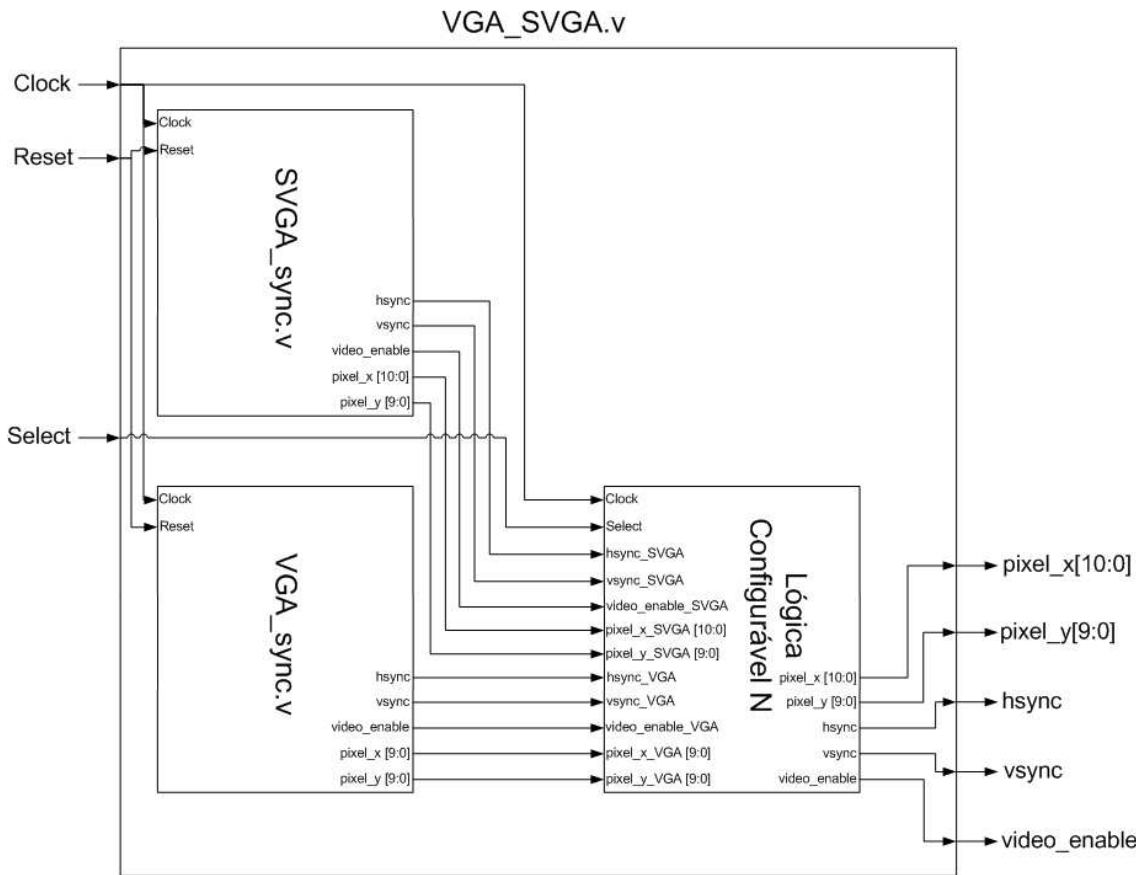


Figura A.19 - Módulo “VGA_SVGA.v”.

Tabela A.28 - Sinais do módulo “VGA_SVGA.v”.

Entradas:	
Clock	Relógio interno de 50MHz.
Reset	Sinal externo de reset (activo a '1').
Select	Sinal de escolha entre VGA e SVGA.
Saídas:	
hsync	"Horizontal Sync" do conector VGA.
vsync	"Vertical Sync" do conector VGA.
video_enable	Indica se o pixel actual é um pixel visível.
pixel_x [10:0]	Indica a coordenada x do pixel actual.
pixel_y [9:0]	Indica a coordenada y do pixel actual.
Ligações Internas:	
hsync_SVGA	"Horizontal Sync" proveniente de SVGA_sync.
vsync_SVGA	"Vertical Sync" proveniente de SVGA_sync.
video_enable_SVGA	"Video_enable" proveniente de SVGA_sync.
pixel_x_SVGA [10:0]	Indica a coordenada x do pixel actual proveniente de SVGA_sync.
pixel_y_SVGA [9:0]	Indica a coordenada y do pixel actual proveniente de SVGA_sync.
hsync_VGA	"Horizontal Sync" proveniente de VGA_sync.

vsync_VGA	"Vertical Sync" proveniente de VGA_sync.
video_enable_VGA	"Video_enable" proveniente de VGA_sync.
pixel_x_VGA [9:0]	Indica a coordenada x do pixel actual proveniente de VGA_sync.
pixel_y_VGA [9:0]	Indica a coordenada y do pixel actual proveniente de VGA_sync.

Design Summary:

Tabela A.29 - Ocupação de recursos dos módulos que implementam VGA + SVGA (Gráfico).

Recurso	Utilizados	Disponíveis	Percentagem Ocupada
<i>Total Number of Slice Flip-Flops</i>	49	9312	0.53%
<i>Total Number of 4 input LUTs</i>	126	9312	1.35%
<i>Number Used as Logic</i>	89		
<i>Number Used as Route-Thru</i>	37		
<i>Total Number of Occupied Slices</i>	71	4656	1.52%
<i>Number of Bounded IOBs</i>	11	232	4.74%
<i>Number of BUFGMUXs</i>	1	24	4.17%

Para a utilização do módulo "VGA_SVGA.v" é necessária a ligação aos seguintes recursos da FPGA: Relógio de 50 MHz, botão/sinal de reset, sinais hsync, vsync, R, G e B do conector VGA da FPGA.

No módulo "VGA_SVGA_top.v" é demonstrado um pequeno exemplo de utilização deste módulo. Todas as ligações aos recursos da FPGA estão especificadas no ficheiro VGA.ucf. Caso seja alterado o nome de alguma das entradas ou saídas especificadas no ficheiro ".ucf", é necessário alterar o nome da "NET" nesse ficheiro, pelo que a alteração não é recomendada.

O módulo "VGA_SVGA.v" utiliza os módulos de sincronismo gráfico do protocolo VGA e SVGA, sendo que o protocolo a ser utilizado no sincronismo com o monitor é definido através da entrada Select. Caso a entrada Select esteja no valor lógico '1' é efectuado o sincronismo com o protocolo SVGA, caso o seu valor seja '0' é efectuado o sincronismo utilizando o protocolo VGA.

O módulo "VGA_SVGA_sync.v" não atribui valores aos sinais R, G e B. Apenas efectua a sincronização do monitor no modo referido e indica as coordenadas do pixel actual. O módulo superior, e construído conforme os objectivos pretendidos é que actua sobre as saídas R, G e B atribuindo, através destes sinais, uma das oito cores disponíveis ao respectivo pixel.

A.11 - VGA e SVGA (Modo texto)

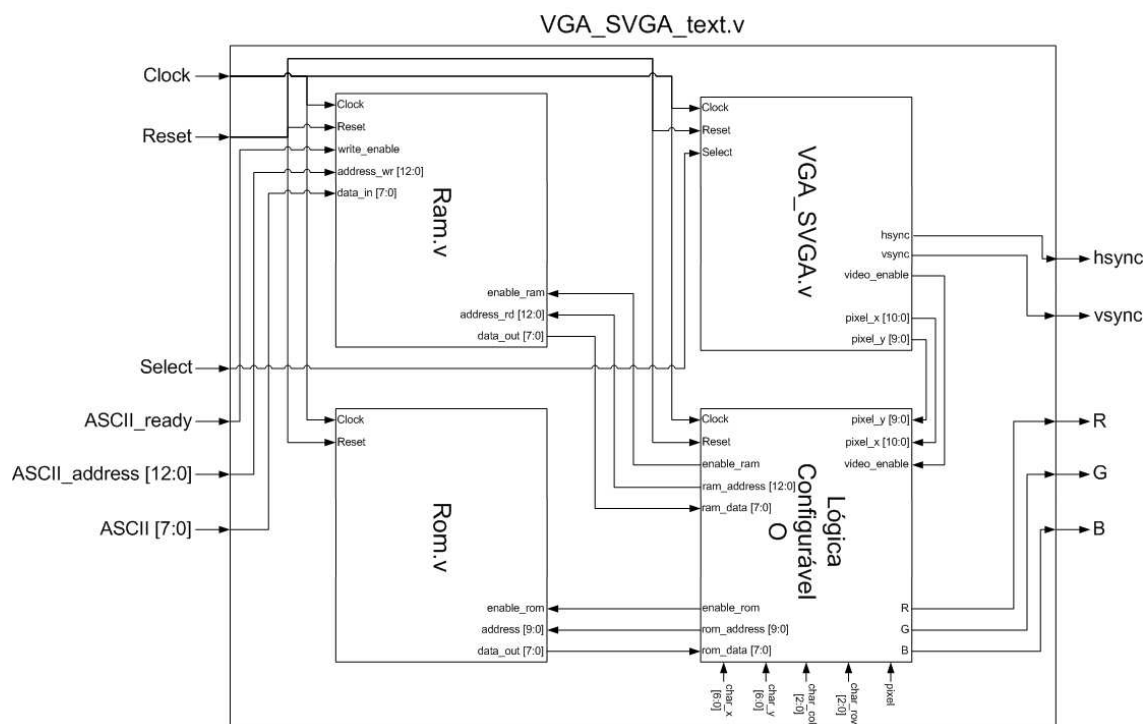


Figura A.20 - Módulo "VGA_SVGA_text.v".

Tabela A.30 - Sinais do módulo "VGA_SVGA_text.v".

Entradas:	
Clock	Relógio interno de 50MHz.
Reset	Sinal externo de reset (activo a '1').
Select	Sinal de escolha entre VGA e SVGA.
ASCII_ready	Indica que está pronto a ser escrito um novo carácter na memória RAM do controlador VGA de texto.
ASCII_address [12:0]	Endereço do carácter a escrever na memória RAM do controlador VGA de texto.
ASCII [7:0]	Código ASCII do carácter a escrever na memória RAM do controlador VGA de texto.
Saídas:	
hsync	"Horizontal Sync" do conector VGA.
vsync	"Vertical Sync" do conector VGA.
R	"Red" do conector VGA.
G	"Green" do conector VGA.
B	"Blue" do conector VGA.
Ligações Internas:	
enable_ram	Activa a memória RAM.
enable_rom	Activa a memória ROM.
ram_address	Endereço de leitura da memória RAM.

[12:0]	
rom_address [9:0]	Endereço de leitura da memória ROM.
ram_data [7:0]	Dados lidos da memória RAM.
rom_data [7:0]	Dados lidos da memória ROM.
video_enable	Indica se o pixel actual é um pixel visível.
pixel_x [10:0]	Indica a coordenada x do pixel actual.
pixel_y [9:0]	Indica a coordenada y do pixel actual.
Registos Internos:	
char_x [6:0]	Coordenada x da grelha de caracteres.
char_y [6:0]	Coordenada y da grelha de caracteres.
char_col [2:0]	Coluna actual dentro da célula do carácter.
char_row [2:0]	Linha actual dentro da célula do carácter.
pixel	Indica se o pixel actual pertence ao carácter ou ao fundo.

Design Summary:

Tabela A.31 - Ocupação de recursos dos módulos que implementam VGA + SVGA (Texto).

Recurso	Utilizados	Disponíveis	Percentagem Ocupada
<i>Total Number of Slice Flip-Flops</i>	67	9312	0.72%
<i>Total Number of 4 input LUTs</i>	176	9312	1.89%
<i>Number Used as Logic</i>	127		
<i>Number Used as Route-Thru</i>	49		
<i>Total Number of Occupied Slices</i>	93	4656	2.00%
<i>Number of Bounded IOBs</i>	10	232	4.31%
<i>Number of BUFGMUXs</i>	1	24	4.17%
<i>Number of RAMB16s</i>	5	20	25.00%
<i>Number of MULT18X18SIOs</i>	2	20	10.00%

Para a utilização do módulo “VGA_SVGA_text.v” são necessários os módulos “VGA_SVGA.v”, “ram.v” e “rom.v”, para além da ligação aos seguintes recursos da FPGA: Relógio de 50 MHz, botão/sinal de reset, sinais hsync, vsync, R, G e B do conector VGA.

No módulo VGA_SVGA_text_top.v é demonstrado um pequeno exemplo de utilização deste módulo que, para além dos recursos mencionados anteriormente, utiliza ainda os recursos necessários ao funcionamento do módulo “Keyboard.v” (para mais informação consultar o manual deste módulo). Todas as ligações aos recursos da FPGA estão especificadas no ficheiro VGA.ucf. Caso seja alterado o nome de alguma das entradas ou saídas especificadas no ficheiro “.ucf”, é necessário alterar o nome da “NET” nesse ficheiro, pelo que a alteração não é recomendada.

Este módulo permite a visualização do modo de texto nos protocolos VGA e SVGA, sendo o protocolo escolhido através do sinal Select. Caso esta entrada esteja no valor lógico '1' é efectuado o sincronismo com o protocolo SVGA, com 100 colunas e 75 linhas de caracteres,

caso o seu valor seja '0' é efectuado o sincronismo utilizando o protocolo VGA, com 80 colunas e 60 linhas de caracteres.

A sincronização dos sinais de vídeo hsync e vsync é feita pelo módulo VGA_SVGA.v que é descrito em pormenor na secção sobre VGA + SVGA (Modo Gráfico). Este módulo devolve também as coordenadas actuais do pixel a ser processado, através de pixel_x e pixel_y.

O módulo “ram.v” implementa uma memória RAM com 8192 posições, cada uma com oito bits, pois esta memória vai armazenar o código ASCII de cada uma das posições do monitor. Para consultar a memória é utilizado o fio ram_address, que indica a coordenada do carácter pretendido, calculada através das coordenadas devolvidas pelo módulo “VGA_SVGA.v”, recorrendo também aos fios char_x, char_y, char_col e char_row, que representam as coordenadas x e y do carácter e a coluna/linha actual do carácter, respectivamente. O código ASCII é devolvido através de ram_data. Para alterar esta memória basta utilizar os sinais ASCII, ASCII_ready e ASCII_address. Estes três sinais são entradas do módulo “VGA_SVGA_text.v”. O sinal de ASCII_ready funciona como “write enable” da memória, o sinal ASCII_address indica a posição da memória e ASCII os dados a guardar.

O módulo “rom.v” implementa uma memória ROM com 1024 posições de 8bits que, após a inicialização, contém o *bitmap* dos caracteres a imprimir no monitor. Durante a inicialização o módulo vai buscar os valores a guardar ao ficheiro bitmap.txt. O endereço rom_address de 10 bits funciona da seguinte forma: os 7 bits mais significativos representam o código ASCII do carácter (excluindo o bit mais significativo, uma vez que estamos a usar a tabela ASCII simples e não a estendida), e os 3 bits menos significativos indicam a linha do carácter (cada carácter tem 8 linhas e 8 colunas).

A.12 - Som

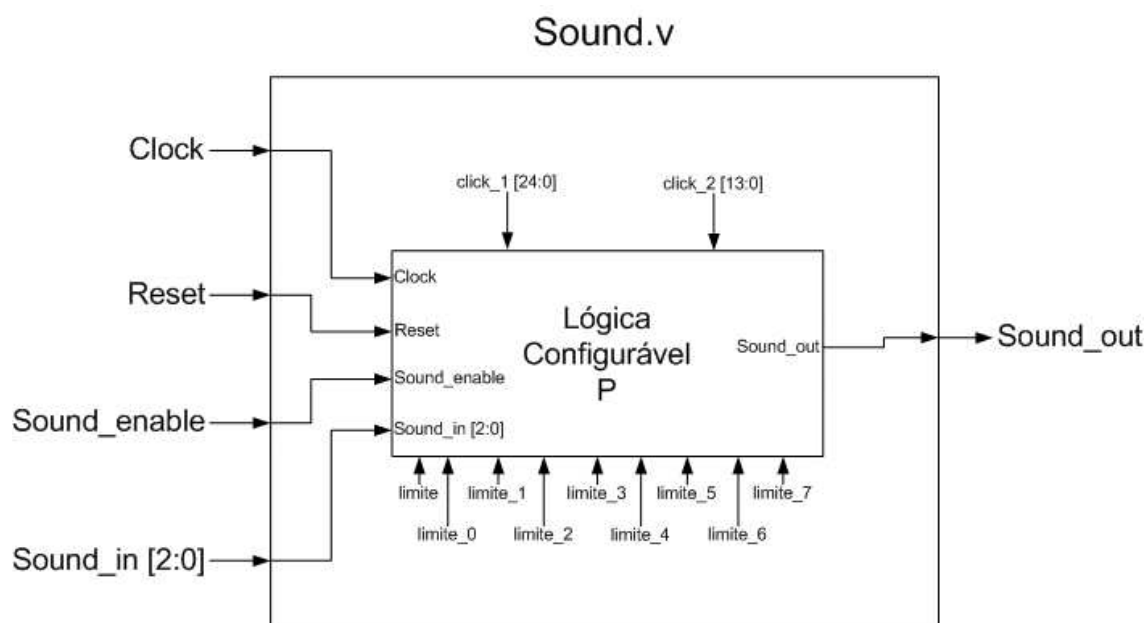


Figura A.21 - Módulo “Sound.v”.

Tabela A.32 - Sinais do módulo "Sound.v".

Entradas:	
Clock	Relógio interno de 50MHz.
Reset	Sinal externo de reset (activo a '1').
Sound_enable	Activa a reprodução de um som.
Sound_in [2:0]	Indica o som a reproduzir, dentro de uma escala pré-definida.
Saídas:	
Sound_out	Saída para a coluna de som.
Parâmetros Internos:	
limite	Especifica (em ciclos de relógio) a duração do som.
limite_x	Especifica (em ciclos de relógio) a frequência do som, em que x corresponde ao valor de "Sound_in".
Registos Internos:	
click_1 [24:0]	Utilizado na contagem para determinar a duração do som.
click_2 [13:0]	Utilizado na contagem para determinar a frequência do som.

Design Summary:

Tabela A.33 - Ocupação de recursos do módulo "Sound.v".

Recurso	Utilizados	Disponíveis	Percentagem Ocupada
<i>Total Number of Slice Flip-Flops</i>	40	9312	0.43%
<i>Total Number of 4 input LUTs</i>	284	9312	3.05%
<i>Number Used as Logic</i>	247		
<i>Number Used as Route-Thru</i>	37		
<i>Total Number of Occupied Slices</i>	145	4656	3.11%
<i>Number of Bounded IOBs</i>	6	232	2.58%
<i>Number of BUFGMUXs</i>	1	24	4.17%

No ficheiro Sound.ucf encontram-se todas as ligações necessárias aos recursos essenciais para o módulo. Caso seja alterado o nome de alguma das entradas ou saídas do módulo Sound.v, é necessário alterar o nome da "NET" no ficheiro ".ucf", pelo que esta alteração de nomes não é recomendada.

Para iniciar um som deve colocar-se (durante um ciclo de relógio) o sinal "Sound_enable" a '1' e no sinal "Sound_in" um valor entre 0 e 7 (inclusive), sendo que o som cujo valor de "Sound_in" é '0' é o mais agudo, e o que tem valor igual a '7' o mais grave.

Caso se pretenda alterar a duração do som, apenas é necessário alterar o parâmetro "limite", que especifica (em ciclos de relógio) a duração do som. Para alterar a frequência dos sons é só alterar o parâmetro "limite_x", onde x é o valor introduzido em "Sound_in".

É ainda possível obter durações e frequências personalizadas através da passagem de parâmetros para o interior do módulo "Sound.v". O primeiro parâmetro especifica a duração (em ciclos de relógio) e o segundo e seguintes a frequência (em ciclos de relógio) dos sons por ordem crescente.

A.13 - Breakout

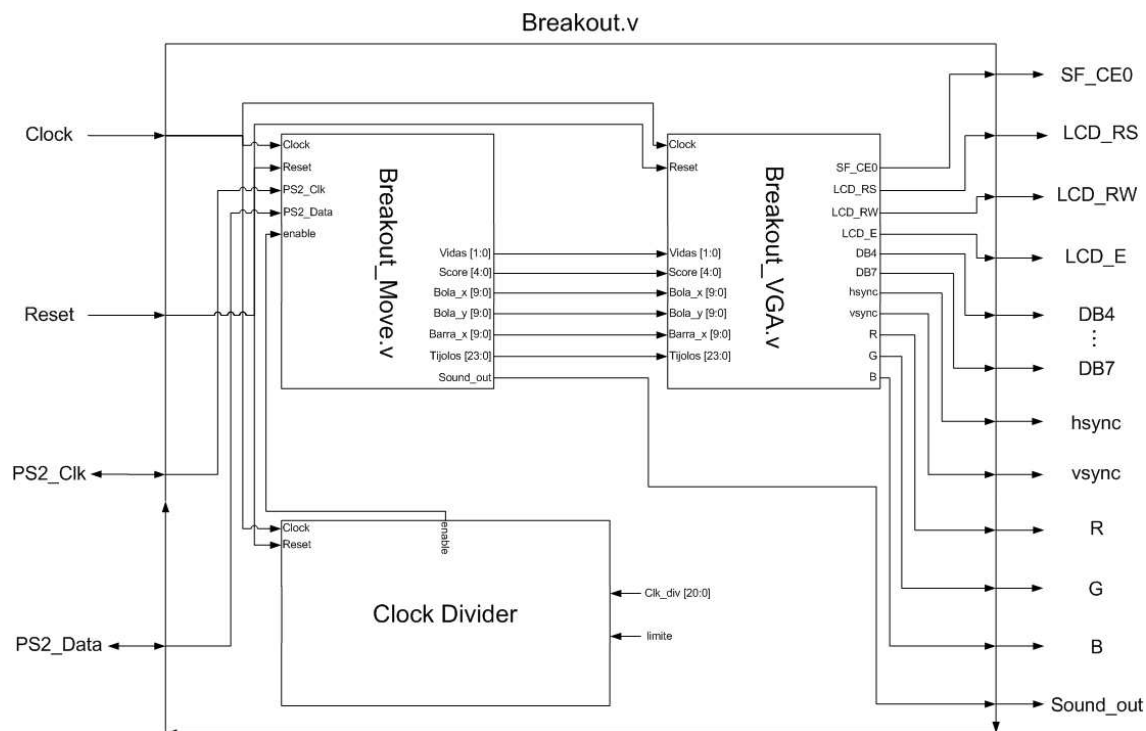


Figura A.22 - Módulo "Breakout.v".

Tabela A.34 - Sinais do módulo "Breakout.v".

Entradas:	
Clock	Relógio interno de 50MHz.
Reset	Sinal externo de reset (activo a '1').
Bidireccionais:	
PS2_Clk	Linha de relógio do barramento PS2.
PS2_Data	Linha de dados do barramento PS2.
Saídas:	
SF_CEO	"Chip Enable" do StrataFlash da FPGA (para desactivar o StrataFlash e activar o LCD).
LCD_RS	"Register Select" do LCD.
LCD_RW	"Read/Write" do LCD.
LCD_E	"Enable" do LCD.
DB4 ... DB7	Barramento paralelo de dados do LCD.
hsync	"Horizontal Sync" do conector VGA.
vsync	"Vertical Sync" do conector VGA.
R	"Red" do conector VGA.
G	"Green" do conector VGA.
B	"Blue" do conector VGA.
Sound_out	Saída para a coluna de som.
Parâmetros Internos:	

limite	Usado como limite do "Clock Divider" que regula a velocidade do jogo.
Registos Internos:	
Clk_div [20:0]	Usado na contagem do "Clock Divider" que regula a velocidade do jogo.
Ligações Internas:	
enable	Regula a velocidade do jogo.
Vidas [1:0]	Indica o número de vidas restantes do jogador.
Score [4:0]	Indica a pontuação actual do jogador.
Bola_x [9:0]	Indica a coordenada x da Bola.
Bola_y [9:0]	Indica a coordenada y da Bola.
Barra_x [9:0]	Indica a coordenada x da Barra.
Tijolos [23:0]	Indica os tijolos "inteiros" e "destruídos".

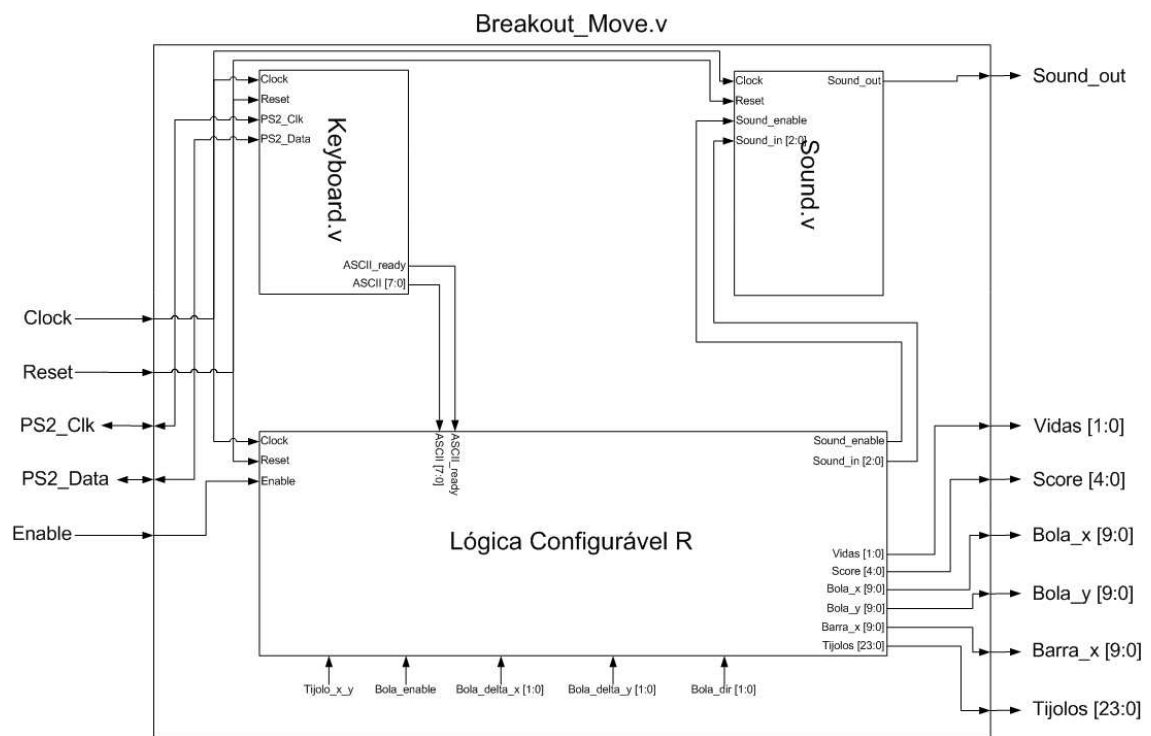


Figura A.23 - Módulo "Breakout_Move.v".

Tabela A.35 - Sinais do módulo "Breakout_Move.v".

Entradas:	
Clock	Relógio interno de 50MHz.
Reset	Sinal externo de reset (activo a '1').
Bidireccionais:	
PS2_Clk	Linha de relógio do barramento PS2.
PS2_Data	Linha de dados do barramento PS2.
Saídas:	
Tijolos [23:0]	Indica os tijolos "inteiros" e "destruídos".
Barra_x [9:0]	Indica a coordenada x da Barra.
Bola_y [9:0]	Indica a coordenada y da Bola.
Bola_x [9:0]	Indica a coordenada x da Bola.

Score [4:0]	Indica a pontuação actual do jogador.
Vidas [1:0]	Indica o número de vidas restantes do jogador.
Sound_out	Saída para a coluna de som.
Registos Internos:	
Bola_enable	Indica se o jogo está activo ou em pausa.
Bola_delta_x [1:0]	Indica a variação (em pixéis) da bola, no eixo x, a cada impulso de "enable" (caso o jogo esteja activo).
Bola_delta_y [1:0]	Indica a variação (em pixéis) da bola, no eixo y, a cada impulso de "enable" (caso o jogo esteja activo).
Bola_dir [1:0]	Indica a o sentido da bola.
Bola_dir[1]	x ('0' -> esquerda, '1' -> direita).
Bola_dir[0]	y ('0' -> cima, '1' -> baixo).
Tijolo_y_x	Indica o estado do tijolo na fila y (1 -> cima, 2 -> meio, 3 -> baixo), coluna x (1 -> esquerda, 8 -> direita).
Ligações Internas:	
ASCII_ready	Indica a recepção do código ASCII de uma tecla do teclado PS2.
ASCII [7:0]	Código ASCII da tecla premida no teclado PS2.
Sound_enable	Activa a reprodução de um som.
Sound_in [2:0]	Indica o som a reproduzir, dentro de uma escala pré-definida.

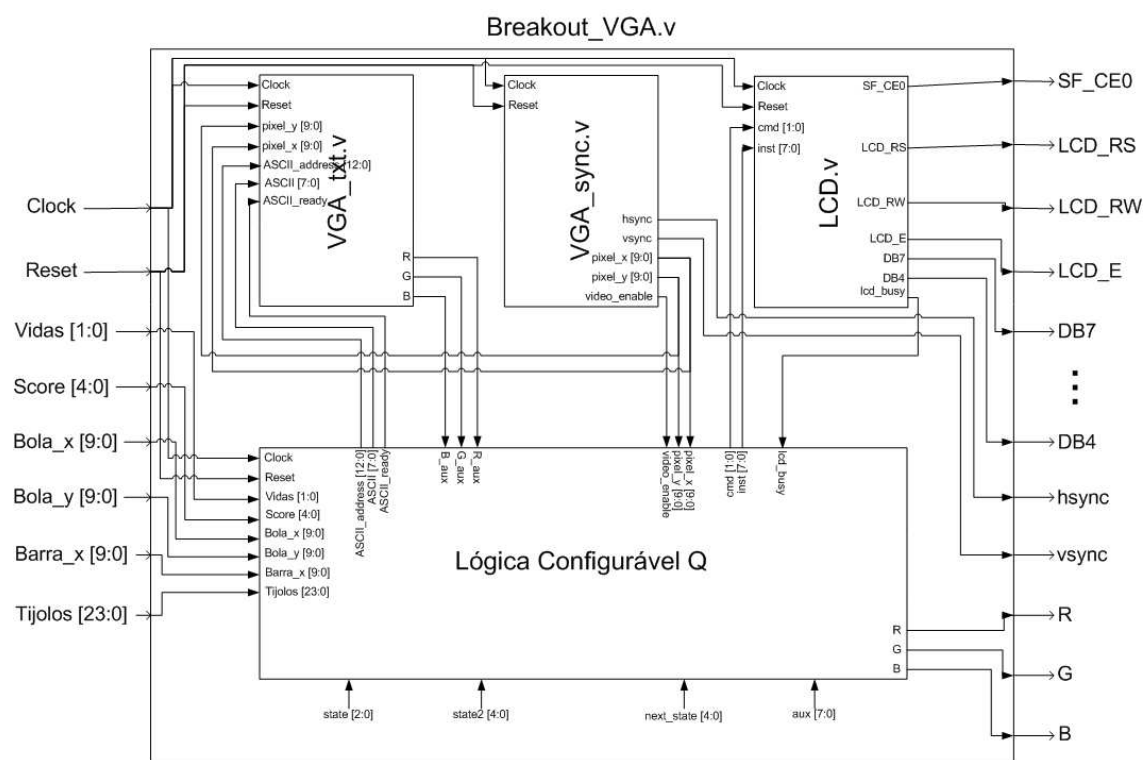


Figura A.24 - Módulo "Breakout_VGA.v".

Tabela A.36 - Sinais do módulo "Breakout_VGA.v".

Entradas:	
Clock	Relógio interno de 50MHz.
Reset	Sinal externo de reset (activo a '1').
Tijolos [23:0]	Indica os tijolos "inteiros" e "destruídos".
Barra_x [9:0]	Indica a coordenada x da Barra.
Bola_y [9:0]	Indica a coordenada y da Bola.
Bola_x [9:0]	Indica a coordenada x da Bola.
Score [4:0]	Indica a pontuação actual do jogador.
Vidas [1:0]	Indica o número de vidas restantes do jogador.
Bidireccionais:	
PS2_Clk	Linha de relógio do barramento PS2.
PS2_Data	Linha de dados do barramento PS2.
Saídas:	
SF_CEO	"Chip Enable" do StrataFlash da FPGA (para desactivar o StrataFlash e activar o LCD).
LCD_RS	"Register Select" do LCD.
LCD_RW	"Read/Write" do LCD.
LCD_E	"Enable" do LCD.
DB4 ... DB7	Barramento paralelo de dados do LCD.
hsync	"Horizontal Sync" do conector VGA.
vsync	"Vertical Sync" do conector VGA.
R	"Red" do conector VGA.
G	"Green" do conector VGA.
B	"Blue" do conector VGA.
Registos Internos:	
state [2:0]	Usado na FSM que escreve na memória RAM do controlador VGA de texto.
state2 [4:0]	Usado na FSM que controla o LCD.
next_state [4:0]	Usado na FSM que controla o LCD.
aux [7:0]	Usado na FSM que controla o LCD.
Ligações Internas:	
video_enable	Indica se o pixel actual é um pixel visível.
pixel_x [9:0]	Indica a coordenada x do pixel actual.
pixel_y [9:0]	Indica a coordenada y do pixel actual.
lcd_busy	Indica se o LCD da FPGA está ocupado ou livre.
cmd [1:0]	Indica o tipo de comando a ordenar ao LCD.
inst [7:0]	Indica o código da instrução a ser processada pelo LCD.
ASCII_ready	Indica que está pronto a ser escrito um novo carácter na memória RAM do controlador VGA de texto.
ASCII_address [12:0]	Endereço do carácter a escrever na memória RAM do controlador VGA de texto.
ASCII [7:0]	Código ASCII do carácter a escrever na memória RAM do controlador VGA de texto.
R_aux	"Red" do controlador VGA de texto.
G_aux	"Green" do controlador VGA de texto.

B_aux	"Blue" do controlador VGA de texto.
-------	-------------------------------------

Módulos Necessários:

Breakout.v
 -Breakout_VGA.v
 --VGA_sync.v
 --VGA_txt.v
 ---ram.v
 ----ram.txt
 ---rom.v
 ----bitmaps.txt
 --LCD.v
 -Breakout_Move.v
 --Keyboard.v
 ---PS2.v
 ----PS2_tx.v
 ----PS2_rx.v
 --Sound.v
 -Breakout.ucf

Design Summary:

Tabela A.37 - Ocupação de recursos dos módulos que implementam o jogo Breakout.

Recurso	Utilizados	Disponíveis	Percentagem Ocupada
<i>Total Number of Slice Registers</i>	375	9312	4.03%
<i>Total Number of 4 input LUTs</i>	2196	9312	23.58%
<i>Number Used as Logic</i>	1988		
<i>Number Used as Route-Thru</i>	208		
<i>Total Number of Occupied Slices</i>	1153	4656	24.76%
<i>Number of Bounded IOBs</i>	20	232	8.62%
<i>Number of BUFGMUXs</i>	1	24	4.17%
<i>Number of RAMB16s</i>	5	20	25.00%
<i>Number of MULTI18X18SIOs</i>	1	20	5.00%

O módulo "Breakout.v" é uma aplicação que utiliza uma parte dos restantes módulos desenvolvidos, servindo assim como uma aplicação exemplo do trabalho efectuado.

Os módulos implementam o conhecido jogo *Breakout*, também conhecido por *Brick Breaker* ou *Arkanoid*.

O utilizador joga utilizando um teclado PS2 ligado ao conector da FPGA, utilizando as teclas 'A' e 'D' para movimentar a barra para a esquerda e direita, respectivamente, e o espaço para iniciar e pausar o jogo.

No ficheiro Breakout.ucf encontram-se todas as ligações necessárias aos recursos essenciais para o jogo. Caso seja alterado o nome de alguma das entradas ou saídas do

módulo “Breakout.v”, é necessário alterar o nome da “NET” no ficheiro “.ucf”, pelo que esta alteração de nomes não é recomendada.

O módulo “Breakout_VGA.v” é responsável por todo o processamento de dados que são disponibilizados ao jogador, desde o processamento do sinal vídeo para o monitor VGA (gráfico e texto) à informação disponibilizada no LCD da FPGA.

Para actualizar a informação disponibilizada no LCD (vidas e pontuação) são usados pela lógica configurável dois registos: cmd e inst.

O módulo utiliza os registos ASCII_ready, ASCII_address e ASCII para actualizar o número de vidas e a pontuação no módulo de texto VGA. As vidas e a pontuação tem endereços específicos na memória RAM do controlador de texto, que estão guardados em parâmetros.

O módulo de texto em VGA tem saídas: R_aux, G_aux e B_aux, que indicam as 3 cores respectivas do pixel, caso o monitor estivesse completamente em modo de visualização de texto, o que não acontece. Como só parte do monitor VGA é utilizado em modo de texto, compete à lógica configurável deste módulo decidir quando utilizar os dados provenientes do controlador de texto.

Para saber que pixel se está a processar é utilizado o módulo “VGA_sync.v”, que indica as coordenadas x e y (registos pixel_x e pixel_y) do pixel actual e se este se trata de um pixel visível ou não (video_enable).

Através das informações obtidas pelas entradas (Bola_x, Bola_y, Barra_x e Tijolos) e das informações relativas ao pixel actual, a lógica configurável é responsável por “desenhar” os objectos no monitor VGA (Bola, Barra e Tijolos “inteiros”).

O módulo “Breakout_Move.v” é responsável pelos movimentos da barra e da bola, indicação da destruição dos tijolos e reprodução de sons. Este módulo processa o movimento dos objectos e interacções entre estes e comunica ao módulo “Breakout_VGA.v” as coordenadas da barra e bola, os tijolos por destruir, número de vidas e pontuação actual.

Sempre que uma tecla é premida a lógica configurável deste módulo é alertada através de ASCII_ready e, caso o código da tecla premida (ASCII) corresponda aos códigos das teclas usadas para jogar o jogo, a lógica configurável actua de forma a responder às instruções do jogador.

Para alterar a posição da barra, é alterada a saída Barra_x. Para alterar a posição da bola, são alteradas as saídas Bola_x e Bola_y. A lógica configurável é ainda responsável por manter a bola e a barra dentro dos limites de jogo e de todas as interacções entre objectos (bola na barra e bola nos Tijolos). Quando um tijolo é “destruído” o seu bit respectivo na saída Tijolos é colocado a '0'.

De referir ainda que a maneira como a bola reage a uma interacção com outro objecto (barra ou tijolos) depende do ponto de contacto entre ambos, variando a direcção e velocidade de deslocamento.

Para produzir um som são utilizados os registos Sound_enable e Sound_in, sendo que o primeiro activa o som indicado pelo segundo.

Na comunicação entre os módulos “Breakout_VGA.v” e “Breakout_Move.v” são usados os seguintes sinais:

Tijolos [23:0]	→ Indica os tijolos “inteiros” e “destruídos”.
Bola_x [9:0]	→ Coordenada x da bola.
Bola_y [9:0]	→ Coordenada y da bola.
Barra_x [9:0]	→ Coordenada x da barra.
Score [4:0]	→ Pontuação actual.

Vidas [1:0] → Número de vidas remanescentes.

A.14 - Pong

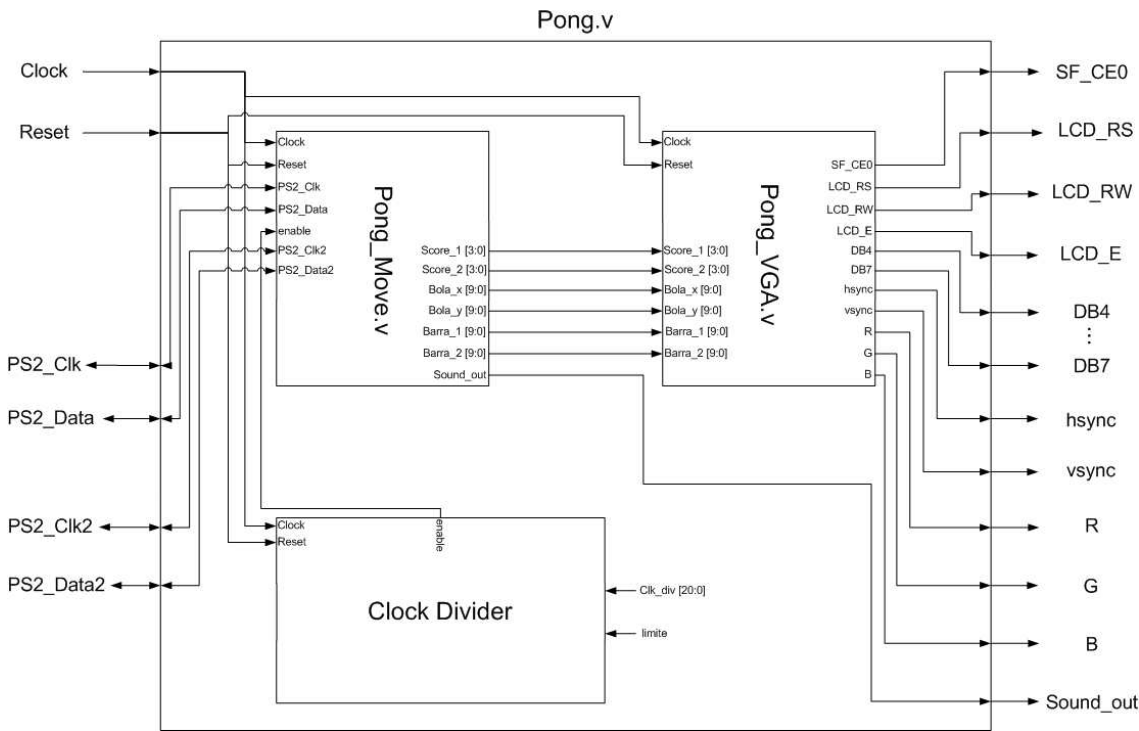


Figura A.25 - Módulo “Pong.v”.

Tabela A.38 - Sinais do módulo “Pong.v”.

Entradas:	
Clock	Relógio interno de 50MHz.
Reset	Sinal externo de reset (activo a '1').
Bidireccionais:	
PS2_Clk	Linha de relógio do barramento PS2 (ligar teclado).
PS2_Data	Linha de dados do barramento PS2 (ligar teclado).
PS2_Clk2	Linha de relógio do barramento PS2 auxiliar (ligar rato).
PS2_Data2	Linha de dados do barramento PS2 auxiliar (ligar rato).
Saídas:	
SF_CEO	"Chip Enable" do StrataFlash da FPGA (para desactivar o StrataFlash e activar o LCD).
LCD_RS	"Register Select" do LCD.
LCD_RW	"Read/Write" do LCD.
LCD_E	"Enable" do LCD.
DB4 ... DB7	Barramento paralelo de dados do LCD.
hsync	"Horizontal Sync" do conector VGA.
vsync	"Vertical Sync" do conector VGA.
R	"Red" do conector VGA.
G	"Green" do conector VGA.

B	"Blue" do conector VGA.
Sound_out	Saída para a coluna de som.
Parâmetros Internos:	
limite	Usado como limite do "Clock Divider" que regula a velocidade do jogo.
Registos Internos:	
Clk_div [20:0]	Usado na contagem do "Clock Divider" que regula a velocidade do jogo.
Ligações Internas:	
Score_1 [3:0]	Indica a pontuação actual do jogador 1 (teclado).
Score_2 [3:0]	Indica a pontuação actual do jogador 2 (rato).
Bola_x [9:0]	Indica a coordenada x da Bola.
Bola_y [9:0]	Indica a coordenada y da Bola.
Barra_1 [9:0]	Indica a coordenada y da Barra do jogador 1 (teclado).
Barra_2 [9:0]	Indica a coordenada y da Barra do jogador 2 (rato).
enable	Regula a velocidade do jogo.

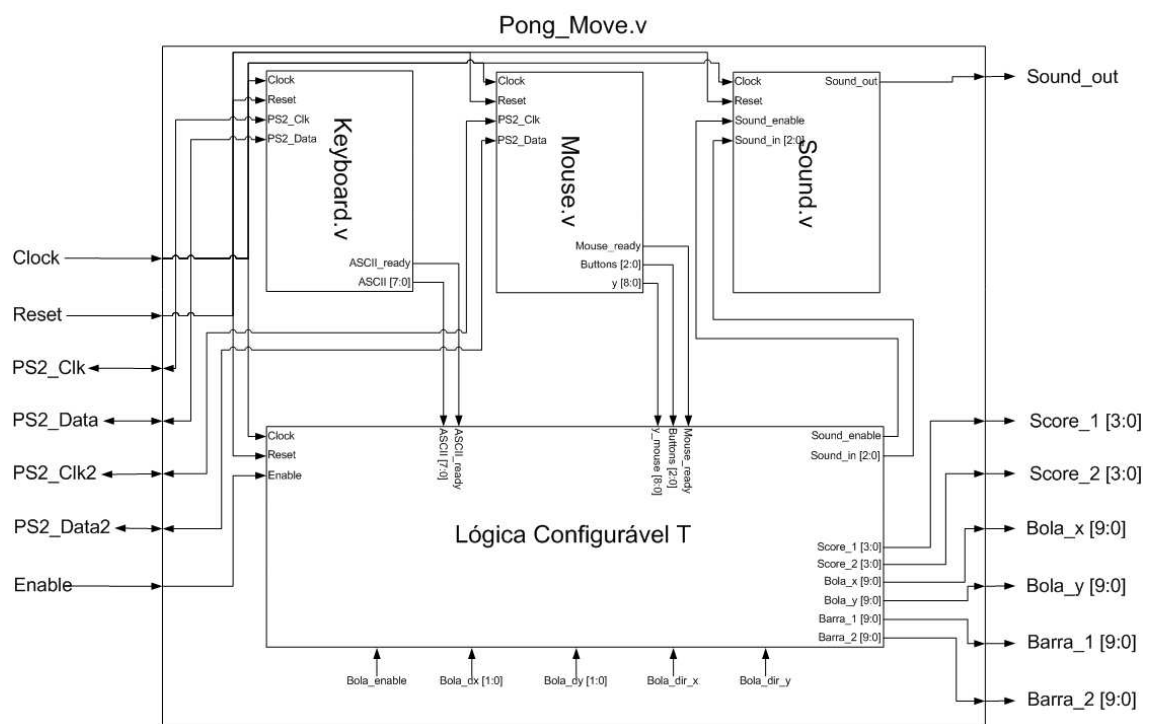


Figura A.26 - Módulo "Pong_Move.v".

Tabela A.39 - Sinais do módulo "Pong_Move.v".

Entradas:	
Clock	Relógio interno de 50MHz.
Reset	Sinal externo de reset (activo a '1').
enable	Regula a velocidade do jogo.
Bidireccionais:	
PS2_Clk	Linha de relógio do barramento PS2 (ligar teclado).
PS2_Data	Linha de dados do barramento PS2 (ligar teclado).
PS2_Clk2	Linha de relógio do barramento PS2 auxiliar (ligar rato).
PS2_Data2	Linha de dados do barramento PS2 auxiliar (ligar rato).

Saídas:	
Score_1 [3:0]	Indica a pontuação actual do jogador 1 (teclado).
Score_2 [3:0]	Indica a pontuação actual do jogador 2 (rato).
Bola_x [9:0]	Indica a coordenada x da Bola.
Bola_y [9:0]	Indica a coordenada y da Bola.
Barra_1 [9:0]	Indica a coordenada y da Barra do jogador 1 (teclado).
Barra_2 [9:0]	Indica a coordenada y da Barra do jogador 2 (rato).
Sound_out	Saída para a coluna de som.
Registos Internos:	
Bola_enable	Indica se o jogo está activo ou em pausa.
Bola_dx [1:0]	Indica a variação (em pixéis) da bola, no eixo x, a cada impulso de "enable" (caso o jogo esteja activo).
Bola_dy [1:0]	Indica a variação (em pixéis) da bola, no eixo y, a cada impulso de "enable" (caso o jogo esteja activo).
Bola_dir_x	Indica o sentido da bola no eixo x ('0' -> esquerda, '1' -> direita).
Bola_dir_y	Indica o sentido da bola no eixo y ('0' -> cima, '1' -> baixo).
Ligações Internas:	
ASCII_ready	Indica a recepção do código ASCII de uma tecla do teclado PS2.
ASCII [7:0]	Código ASCII da tecla premida no teclado PS2.
Mouse_ready	Indica a recepção de dados do rato PS2.
Buttons [2:0]	Indica o estado dos botões do rato PS2.
y_mouse [9:0]	Indica a direcção e amplitude de movimento no eixo vertical do rato PS2.
Sound_enable	Activa a reprodução de um som.
Sound_in [2:0]	Indica o som a reproduzir, dentro de uma escala pré-definida.

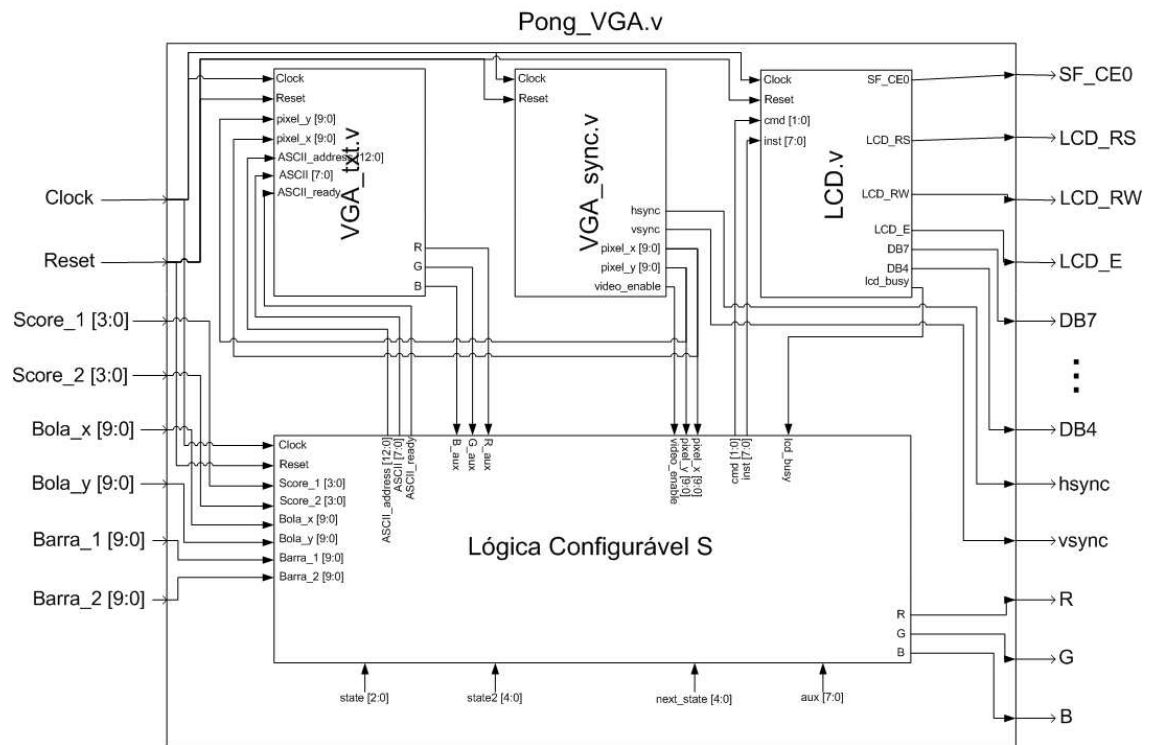


Figura A.27 - Módulo "Pong_VGA.v".

Tabela A.40 - Sinais do módulo "Pong_VGA.v".

Entradas:	
Clock	Relógio interno de 50MHz.
Reset	Sinal externo de reset (activo a '1').
Score_1 [3:0]	Indica a pontuação actual do jogador 1 (teclado).
Score_2 [3:0]	Indica a pontuação actual do jogador 2 (rato).
Bola_x [9:0]	Indica a coordenada x da Bola.
Bola_y [9:0]	Indica a coordenada y da Bola.
Barra_1 [9:0]	Indica a coordenada y da Barra do jogador 1 (teclado).
Barra_2 [9:0]	Indica a coordenada y da Barra do jogador 2 (rato).
Saídas:	
SF_CEO	"Chip Enable" do StrataFlash da FPGA (para desactivar o StrataFlash e activar o LCD).
LCD_RS	"Register Select" do LCD.
LCD_RW	"Read/Write" do LCD.
LCD_E	"Enable" do LCD.
DB4 ... DB7	Barramento paralelo de dados do LCD.
hsync	"Horizontal Sync" do conector VGA.
vsync	"Vertical Sync" do conector VGA.
R	"Red" do conector VGA.
G	"Green" do conector VGA.
B	"Blue" do conector VGA.
Registos Internos:	
state [2:0]	Usado na FSM que escreve na memória RAM do controlador VGA de texto.

state2 [4:0]	Usado na FSM que controla o LCD.
next_state [4:0]	Usado na FSM que controla o LCD.
aux [7:0]	Usado na FSM que controla o LCD.
Ligações Internas:	
video_enable	Indica se o pixel actual é um pixel visível.
pixel_x [9:0]	Indica a coordenada x do pixel actual.
pixel_y [9:0]	Indica a coordenada y do pixel actual.
lcd_busy	Indica se o LCD da FPGA está ocupado ou livre.
cmd [1:0]	Indica o tipo de comando a ordenar ao LCD.
inst [7:0]	Indica o código da instrução a ser processada pelo LCD.
ASCII_ready	Indica que está pronto a ser escrito um novo carácter na memória RAM do controlador VGA de texto.
ASCII_address [12:0]	Endereço do carácter a escrever na memória RAM do controlador VGA de texto.
ASCII [7:0]	Código ASCII do carácter a escrever na memória RAM do controlador VGA de texto.
R_aux	"Red" do controlador VGA de texto.
G_aux	"Green" do controlador VGA de texto.
B_aux	"Blue" do controlador VGA de texto.

Módulos Necessários:

```

Pong.v
-Pong_VGA.v
--VGA_sync.v
--VGA_txt.v
---ram.v
----ram.txt
---rom.v
----bitmaps.txt
--LCD.v
-Pong_Move.v
--Keyboard.v
---PS2.v
----PS2_tx.v
----PS2_rx.v
--Mouse.v
---PS2.v
----PS2_tx.v
----PS2_rx.v
--Sound.v
-Pong.ucf

```

Design Summary:**Tabela A.41** - Ocupação de recursos dos módulos que implementam o jogo Pong.

Recurso	Utilizados	Disponíveis	Percentagem Ocupada
<i>Total Number of Slice Registers</i>	426	9312	4.57%
<i>Total Number of 4 input LUTs</i>	1865	9312	20.03%
<i>Number Used as Logic</i>	1596		
<i>Number Used as Route-Thru</i>	269		
<i>Total Number of Occupied Slices</i>	1003	4656	21.54%
<i>Number of Bounded IOBs</i>	20	232	8.62%
<i>Number of BUFGMUXs</i>	1	24	4.17%
<i>Number of RAMB16s</i>	5	20	25.00%
<i>Number of MULTI18X18SIOs</i>	1	20	5.00%

O modulo Pong.v é uma aplicação que utiliza uma parte dos restantes módulos desenvolvidos, servindo assim como uma aplicação exemplo do trabalho efectuado.

Os módulos implementam o conhecido jogo "Pong".

O jogador 1 utilizando um teclado PS2 ligado ao conector da FPGA, utilizando as teclas 'W' e 'S' para movimentar a barra para a cima e baixo, respectivamente, e o espaço para iniciar e pausar o jogo.

O jogador 2 utilizando um rato PS2 ligado a um conector desenvolvido de forma a ligar a um porto de expansão da FPGA, movimentando o rato verticalmente para movimentar a barra para a cima e baixo, e o botão esquerdo do rato para iniciar e pausar o jogo.

No ficheiro Pong.ucf encontram-se todas as ligações necessárias aos recursos essenciais para o jogo. Caso seja alterado o nome de alguma das entradas ou saídas do módulo Breakout.v, é necessário alterar o nome da "NET" no ficheiro ".ucf", pelo que esta alteração de nomes não é recomendada.

O módulo "Pong_VGA.v" é responsável por todo o processamento de dados que são disponibilizados ao jogador, desde o processamento do sinal vídeo para o monitor VGA (gráfico e texto), à informação disponibilizada no LCD da FPGA.

Para actualizar a informação disponibilizada no LCD (pontuações dos jogadores) são usados pela lógica configurável 2 registos: cmd e inst.

O módulo utiliza os registos ASCII_ready, ASCII_address e ASCII para actualizar a pontuação dos dois jogadores no módulo de texto VGA. As pontuações têm endereços específicos na memória RAM do controlador de texto, que estão guardados em parâmetros.

O módulo de texto em VGA tem três saídas: R_aux, G_aux e B_aux, que indicam as três cores respectivas do pixel, caso o monitor estivesse completamente em modo de visualização de texto, o que não acontece. Como só parte do monitor VGA é utilizado em modo de texto, compete à lógica configurável deste módulo decidir quando utilizar os dados provenientes do controlador de texto.

Para saber que pixel se está a processar é utilizado o módulo “VGA_sync.v” que indica as coordenadas x e y (registos pixel_x e pixel_y) do pixel actual e se este se trata de um pixel visível ou não (video_enable).

Através das informações obtidas pelas entradas (Bola_x, Bola_y, Barra_1 e Barra_2) e das informações relativas ao pixel actual, a lógica configurável é responsável por “desenhar” os objectos no monitor VGA (Bola e Barras).

O módulo “Pong_Move.v” é responsável pelos movimentos das barras e da bola e reprodução de sons. Este módulo processa o movimento dos objectos e interacções entre estes e comunica ao módulo “Pong_VGA.v” as coordenadas das barras e bola e as pontuações.

Sempre que o jogador 1prime uma tecla, a lógica configurável deste módulo é alertada através de ASCII_ready e, caso o código da tecla premida (ASCII) corresponda aos códigos das teclas usadas para jogar o jogo, a lógica configurável actua de forma a responder às instruções do jogador.

Para alterar a posição da barra do jogador 1, é alterada a saída Barra_1.

Sempre que o jogador movimenta o rato ou pressiona um botão, a lógica reconfigurável é alertada através de Mouse_ready, sendo que o estado dos botões é dado por Buttons e a direcção e amplitude de movimento no eixo y dado por y_mouse.

Para alterar a posição da barra do jogador 2, é alterada a saída Barra_2.

Para alterar a posição da bola, são alteradas as saídas Bola_x e Bola_y. A lógica configurável é ainda responsável por manter a bola e a barra dentro dos limites de jogo e de todas as interacções entre objectos (Bola nas Barras).

De referir ainda que a maneira como a bola reage a uma interacção com outro objecto (barras) depende do ponto de contacto entre ambos, variando a direcção e velocidade de deslocamento.

Para produzir um som são utilizados os registos Sound_enable e Sound_in, sendo que o primeiro activa o som indicado pelo segundo.

Na comunicação entre os módulos “Pong_VGA.v” e “Pong_Move.v” são usados os seguintes sinais:

Score_1 [3:0]	→ Pontuação actual do jogador 1.
Score_2 [3:0]	→ Pontuação actual do jogador 2.
Bola_x [9:0]	→ Coordenada x da bola.
Bola_y [9:0]	→ Coordenada y da bola.
Barra_1 [9:0]	→ Coordenada x da barra do jogador 1.
Barra_2 [9:0]	→ Coordenada x da barra do jogador 2.

Referências

- [1] Ciberdúvidas da Língua Portuguesa. Disponível em <http://www.ciberduvidas.com//glossario.php>. Acesso em múltiplas ocasiões
- [2] Dicionário Priberam da Língua Portuguesa. Disponível em <http://www.priberam.pt/DLPO/>. Acesso em múltiplas ocasiões.
- [3] “Dissertação ou Projecto Final: Normas Para o Seu Funcionamento” Faculdade de Engenharia da universidade do Porto. Acesso em múltiplas ocasiões.
- [4] “Spartan-3E LCD Code”. Disponível em <http://www.edaboard.com/ftopic202710.html#852813>. Acesso em múltiplas ocasiões.
- [5] “Spartan-3E FPGA Starter Kit Board Design Examples”. Disponível em http://www.xilinx.com/products/boards/s3estarter/reference_designs.htm. Acesso em múltiplas ocasiões.
- [6] Pong P. Chu, “FPGA prototyping by VHDL examples: Xilinx Spartan-3 Version”. Disponível parcialmente em <http://books.google.pt/books?id=mwUV7ZK9l9gC>. Acesso em múltiplas ocasiões.
- [7] Adam Chapweske, “The PS/2 Mouse/Keyboard Protocol”. Disponível em <http://www.computer-engineering.org/ps2protocol/>. Acesso em múltiplas ocasiões.
- [8] Adam Chapweske, “The PS/2 Keyboard Interface”. Disponível em <http://www.computer-engineering.org/ps2keyboard/>. Acesso em múltiplas ocasiões.
- [9] Adam Chapweske, “The PS/2 Mouse Interface”. Disponível em <http://www.computer-engineering.org/ps2mouse/>. Acesso em múltiplas ocasiões.
- [10] J. D. Neal, “Hardware Level VGA and SVGA Video Programming Information Page”. Disponível em <http://www.osdever.net/FreeVGA/vga/vga.htm>. Acesso em múltiplas ocasiões.
- [11] “Spartan 3E Schematic”. Disponível em http://www.digilentinc.com/Data/Products/S3EBOARD/S3E%20Starter_sch.pdf. Acesso em múltiplas ocasiões.
- [12] “Spartan-3E Starter Kit Board User Guide”. Disponível em http://www.digilentinc.com/Data/Products/S3EBOARD/S3EStarter_ug230.pdf. Acesso em múltiplas ocasiões.
- [13] “Initial Design for Spartan-3E Starter Kit (LCD Display Control)”. Disponível em http://www.xilinx.com/products/boards/s3estarter/files/s3esk_startup.pdf. Acesso em múltiplas ocasiões.

- [14] “*The I2C Bus Specifications: Version 2.1*”. Disponível em http://www.nxp.com/acrobat_download2/literature/9398/39340011.pdf. Acesso em múltiplas ocasiões.
- [15] “*Field Programmable Gate Array*”. Disponível em <http://en.wikipedia.org/wiki/FPGA>. Acesso em 04 de Junho de 2010.
- [16] “*The Spartan™-3 Platform FPGA Family: The World’s Lowest-Cost FPGAs*”. Disponível em http://www.xilinx.com/publications/prod_mktg/pn0010721-4.pdf. Acesso em múltiplas ocasiões.
- [17] Imagem da plataforma de desenvolvimento. Disponível em http://rawski.zpt.tele.pw.edu.pl/pl/files/Spartan_3E_Starter.jpg. Acesso em 04 de Junho de 2010.
- [18] Página da Internet destinada à divulgação do trabalho desta dissertação. Disponível em <http://sites.google.com/site/ee05177/>.
- [19] “*Wii Remote IR Camera Hack With Arduino Interface*”. Disponível em <http://www.instructables.com/id/Wii-Remote-IR-Camera-Hack/>. Acesso em 06 de Junho de 2010.
- [20] “*Hacking The Wiimote IR Camera*”. Disponível em http://blog.makezine.com/archive/2008/11/hacking_the_wiimote_ir_ca.html?CMP=OTC-0D6B48984890. Acesso em 06 de Junho de 2010.
- [21] “*Wiimote Project Wiki*”. Disponível em <http://wiki.wiimoteproject.com/>. Acesso em 06 de Junho de 2010.