

**FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO**

# **Geração Automática de Autómatos Celulares para FPGA**

**André Mendonça Domingues da Costa Lima**



Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador: Prof. João Canas Ferreira

10 de setembro de 2012



# Resumo

Após o conceito ter sido introduzido por John von Neumann nos anos 40, devido ao seu poder computacional, as arquiteturas baseadas em autómatos celulares têm sido extensivamente exploradas pela comunidade científica para o estudo de características, parâmetros e simulação do comportamento de sistemas dinâmicos complexos, fenômenos naturais ou artificiais e para processamento computacional. Para sistemas de elevado grau de complexidade, as implementações de autómatos celulares em software têm tipicamente um elevado custo computacional e levam a simulações longas. Por isso, procuram-se soluções de hardware dedicado ou reconfigurável que permitam acelerar significativamente o processo de simulação. Dado o seu paralelismo espacial inerente, localidade e natureza discreta, os autómatos celulares mapeiam-se naturalmente na arquitetura regular de uma FPGA organizada em blocos lógicos configuráveis.

Nesta dissertação, propõe-se a implementação de um sistema de geração automática de autómatos celulares em FPGA, cujas características e regras são especificadas pelo utilizador do sistema a partir de uma aplicação de software de suporte que permite também controlar a operação, inicializar e ler o estado do autômato celular.

As implementações foram feitas numa FPGA, de modelo XC6SLX45, da família Spartan6 da Xilinx, a trabalhar a uma frequência de 66 MHz. Os resultados finais indicaram que é possível obter um *speed-up* de 181, para uma implementação do Jogo da Vida de John Conway, em comparação com simulações em software levadas a cabo numa máquina Intel Core 2 Quad Q9400 a 2,66 GHz e aproximadamente 4 GB de memória RAM. Em hardware, as dimensões máximas geradas foram de 56x56, 72x72, 72x72 e 40x40 células para o Jogo da Vida, *lattice gases* e para versões simplificadas de um modelo de incêndio florestal e do autômato celular de Greenberg-Hastings, respetivamente.



# Abstract

After the concept was introduced by John von Neumann in the forties, owing to its computational power, cellular automata based architectures have been extensively explored by the scientific community to study the characteristics, parameters and behaviour through simulation of dynamic complex systems, natural and artificial phenomena, and for computational processing. Concerning the systems of high degree of complexity, cellular automata software implementations usually have a high computational cost and lead to long simulations. So, dedicated or reconfigurable hardware solutions are sought in order to accelerate significantly the simulation process. Because of its inherent spacial parallelism, locality and discrete nature, cellular automata are naturally mapped onto the regular architecture of a FPGA organized in configurable logic blocks.

In this dissertation, it is proposed to implement a system to automatically generate cellular automata architectures on FPGA, the characteristics and rules of which are specified by the system user through a support software application that also allows to control the operation, initialize and read the state of the cellular automata.

The implementations were done on Xilinx's Spartan6 family FPGA, model XC6SLX45, operating at a frequency of 66 MHz. The final results showed that it is possible to achieve a speed-up of 181, for an implementation of John Conway's Game of Life, when compared to software simulations performed on a Intel Core 2 Quad Q9400 machine running at 2,66 GHz with approximately 4 GB of RAM. In hardware, the maximal generated dimensions were 56x56, 72x72, 72x72, 40x40 cells for the implementation of the Game of Life, lattice gases automata and simplified versions of the forest-fire model and Greenberg-Hastings cellular automata, respectively.



# Agradecimentos

Quero agradecer a todos os meus amigos e colegas de laboratório, ao Hugo Marques, à Carla Brito, ao Hélder Campos, ao Nuno Paulino, ao José Sá e ao João Teixeira por todo o apoio prestado ao longo desta etapa, que foi essencial para o sucesso deste trabalho.

Aos meus pais, pois sem eles nada disto teria sido alguma vez concretizado e ao meu avô, que infelizmente já não poderei partilhar este momento com ele, por me ter incentivado indiretamente com as suas fantásticas criações a perseguir a área da engenharia.

Por fim, gostaria de agradecer também a todos os professores que contribuíram para a minha formação, ao longo destes anos durante o meu percurso académico, com especial destaque para os professores José Carlos Alves e João Canas Ferreira pelo seu importante e indispensável contributo na formação na área da eletrónica e sistemas digitais. A este último, como orientador da minha dissertação, agradeço por todo o apoio dado, disponibilidade e, em especial, a sua paciência e compreensão.

André



*“Young man, in mathematics you don’t understand things.  
You just get used to them.”*

John von Neumann



# Conteúdo

|          |                                                                                 |           |
|----------|---------------------------------------------------------------------------------|-----------|
| <b>1</b> | <b>Introdução</b>                                                               | <b>1</b>  |
| 1.1      | Objetivos e motivação . . . . .                                                 | 2         |
| 1.2      | Estrutura do documento . . . . .                                                | 3         |
| <b>2</b> | <b>Revisão Bibliográfica</b>                                                    | <b>5</b>  |
| 2.1      | Autómatos celulares . . . . .                                                   | 5         |
| 2.1.1    | Vizinhança de von Neumann . . . . .                                             | 6         |
| 2.1.2    | Vizinhança de Moore . . . . .                                                   | 8         |
| 2.1.3    | Autómatos unidimensionais (1D) . . . . .                                        | 9         |
| 2.1.4    | Autómatos bidimensionais (2D) . . . . .                                         | 10        |
| 2.2      | <i>Field-Gate Programmable Array (FPGA)</i> . . . . .                           | 13        |
| 2.2.1    | Xilinx Spartan6 FPGA . . . . .                                                  | 15        |
| 2.2.2    | Fluxo de projeto . . . . .                                                      | 17        |
| 2.3      | Aplicações de autómatos celulares em FPGA . . . . .                             | 21        |
| 2.3.1    | Autômato celular de Greenberg-Hastings . . . . .                                | 21        |
| 2.3.2    | Autômato celular para <i>lattice gases</i> . . . . .                            | 24        |
| 2.4      | Máquinas de Autómatos Celulares (CAM) . . . . .                                 | 25        |
| 2.5      | Aplicação de autómatos celulares em software . . . . .                          | 26        |
| 2.5.1    | XLife . . . . .                                                                 | 26        |
| 2.5.2    | Golly . . . . .                                                                 | 27        |
| 2.5.3    | MCell . . . . .                                                                 | 28        |
| <b>3</b> | <b>Especificação dos Sistemas de Controlo de Software e Hardware</b>            | <b>31</b> |
| 3.1      | Visão geral . . . . .                                                           | 32        |
| 3.2      | Casos de utilização . . . . .                                                   | 32        |
| 3.3      | Cenário típico de utilização . . . . .                                          | 46        |
| 3.4      | Arquitetura de alto nível . . . . .                                             | 47        |
| 3.5      | Resumo de funcionalidades . . . . .                                             | 49        |
| 3.6      | Comparação das capacidades do SCH e SCS com as de sistemas existentes . . . . . | 50        |
| 3.6.1    | Componente de hardware . . . . .                                                | 51        |
| 3.6.2    | Componente de software . . . . .                                                | 51        |
| <b>4</b> | <b>Arquitetura do Sistema de Controlo de Hardware</b>                           | <b>53</b> |
| 4.1      | Arquitetura detalhada . . . . .                                                 | 53        |
| 4.2      | Estrutura lógica do Autômato Celular . . . . .                                  | 54        |
| 4.2.1    | Arquitetura Celular . . . . .                                                   | 58        |
| 4.3      | Controlo de fronteira e transferência de dados . . . . .                        | 65        |
| 4.4      | Máquina de estados . . . . .                                                    | 69        |

|          |                                                                            |            |
|----------|----------------------------------------------------------------------------|------------|
| 4.4.1    | Registo de comandos e respostas no banco de registos de controlo . . . . . | 71         |
| 4.5      | Implementação em MicroBlaze . . . . .                                      | 73         |
| 4.5.1    | Algoritmo para a reprodução sonora do estado do AC . . . . .               | 80         |
| 4.6      | Soluções de arquitetura alternativas e considerações finais . . . . .      | 82         |
| <b>5</b> | <b>Arquitetura do Sistema de Controlo de Software</b>                      | <b>87</b>  |
| 5.1      | Interface gráfica, funcionalidades e ambientes de trabalho . . . . .       | 88         |
| 5.2      | Gestão de ficheiros e diretório de trabalho . . . . .                      | 92         |
| 5.3      | Especificação do autómato celular . . . . .                                | 96         |
| 5.3.1    | Especificação das características . . . . .                                | 96         |
| 5.3.2    | Especificação da arquitetura celular . . . . .                             | 97         |
| 5.4      | Gestão de paletes de cor e ficheiros mapeadores . . . . .                  | 100        |
| 5.5      | Gestão de gráficos . . . . .                                               | 104        |
| 5.6      | Gestão de <i>bitstreams</i> e implementação em hardware . . . . .          | 107        |
| 5.6.1    | Fluxo de projeto para FPGA . . . . .                                       | 110        |
| 5.7      | Comunicação com hardware e processamento de dados . . . . .                | 112        |
| <b>6</b> | <b>Resultados</b>                                                          | <b>117</b> |
| 6.1      | Resultados gerais . . . . .                                                | 117        |
| 6.2      | Desempenho e utilização de recursos . . . . .                              | 120        |
| 6.3      | <i>Benchmarking</i> . . . . .                                              | 124        |
| <b>7</b> | <b>Conclusões e trabalho futuro</b>                                        | <b>133</b> |
| 7.1      | Resumo do trabalho . . . . .                                               | 133        |
| 7.2      | Mapeamento da arquitetura do sistema de hardware . . . . .                 | 134        |
| 7.3      | Trabalho futuro . . . . .                                                  | 135        |
| 7.3.1    | Propostas para o sistema de hardware . . . . .                             | 135        |
| 7.3.2    | Propostas para o sistema de software . . . . .                             | 136        |
| 7.4      | Considerações finais . . . . .                                             | 138        |
|          | <b>Referências</b>                                                         | <b>139</b> |

# Lista de Figuras

|      |                                                                                      |     |
|------|--------------------------------------------------------------------------------------|-----|
| 2.1  | Distância de Manhattan . . . . .                                                     | 7   |
| 2.2  | Vizinhança de von Neumann em função de $r$ . . . . .                                 | 7   |
| 2.3  | Distância de Chebyshev . . . . .                                                     | 8   |
| 2.4  | Vizinhança de Moore em função de $r$ . . . . .                                       | 9   |
| 2.5  | Evolução da regra 30 (fonte: [1]) e regra 110 (fonte: [2]) ao fim 16 e 250 iterações | 10  |
| 2.6  | Arquitetura genérica de uma FPGA (segundo [3]) . . . . .                             | 14  |
| 2.7  | Modos de interligação de CLBs e IOBs (segundo [3]) . . . . .                         | 16  |
| 2.8  | Fluxo típico de projeto para FPGA (alto nível) (segundo [4]) . . . . .               | 19  |
| 2.9  | Fluxo típico de projeto para FPGA (detalhado) (segundo [4]) . . . . .                | 19  |
| 2.10 | Interface gráfica do XLife . . . . .                                                 | 27  |
| 2.11 | Interface gráfica do Golly . . . . .                                                 | 28  |
| 2.12 | Interface gráfica do MCell . . . . .                                                 | 29  |
| 3.1  | Vista geral dos sistemas e interação . . . . .                                       | 33  |
| 3.2  | Diagrama de casos de utilização . . . . .                                            | 34  |
| 3.3  | Fluxo do cenário tipo de utilização . . . . .                                        | 47  |
| 3.4  | Diagrama de arquitetura de alto nível . . . . .                                      | 49  |
| 4.1  | Diagrama de arquitetura de hardware detalhada . . . . .                              | 55  |
| 4.2  | Lógica do AC: geração da matriz . . . . .                                            | 56  |
| 4.3  | Lógica do AC: sinais de controlo e barramentos de dados . . . . .                    | 57  |
| 4.4  | Arquitetura celular . . . . .                                                        | 61  |
| 4.5  | Arquitetura dos <i>shift-registers</i> PISO e SIPO . . . . .                         | 66  |
| 4.6  | Arquitetura do módulo de controlo de fronteira e transferência de dados. . . . .     | 68  |
| 4.7  | Diagrama de estados . . . . .                                                        | 70  |
| 4.8  | Circuito para aplicar um impulso com duração exata de um ciclo de relógio. . . . .   | 72  |
| 5.1  | Janela principal da interface gráfica da aplicação . . . . .                         | 88  |
| 5.2  | Conteúdo dos menus da barra de ferramentas . . . . .                                 | 91  |
| 5.3  | Árvore do diretório de trabalho . . . . .                                            | 93  |
| 5.4  | Conteúdo do diretório “verilog” e dos seus subdiretórios . . . . .                   | 94  |
| 5.5  | Gestor de arquitetura celular . . . . .                                              | 98  |
| 5.6  | Acesso ao menu do gestor de paletes de cor . . . . .                                 | 102 |
| 5.7  | Ferramentas de manipulação de paletes de cor e mapeamentos . . . . .                 | 102 |
| 5.8  | Representação gráfica do estado um AC com paletes de cor distintas . . . . .         | 104 |
| 5.9  | Deslocamento de uma célula da matriz do AC no painel gráfico . . . . .               | 107 |
| 5.10 | <i>Zoom out</i> sobre um AC de 64x64 células de dois estados por célula . . . . .    | 108 |
| 5.11 | Janela do gestor de <i>bitstreams</i> . . . . .                                      | 109 |
| 5.12 | Interface gráfica da aplicação após estabelecida a comunicação série . . . . .       | 113 |



# Lista de Tabelas

|     |                                                                                                                                                                                          |     |
|-----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 2.1 | Recursos da FPGA Spartan6 [5]                                                                                                                                                            | 17  |
| 2.2 | Resultados de síntese para um AC de 512x512 células de Greenberg-Hastings                                                                                                                | 23  |
| 2.3 | Número de atualizações de estado das células por segundo para um módulo SPACE em comparação com outras plataformas.                                                                      | 25  |
| 3.1 | Descrição de casos de utilização de especificação                                                                                                                                        | 37  |
| 3.2 | Descrição de casos de utilização de implementação                                                                                                                                        | 39  |
| 3.3 | Descrição de casos de utilização de configuração                                                                                                                                         | 41  |
| 3.4 | Descrição de casos de utilização de comunicação                                                                                                                                          | 42  |
| 3.5 | Descrição de casos de utilização de execução                                                                                                                                             | 44  |
| 3.6 | Descrição de casos de utilização de visualização                                                                                                                                         | 46  |
| 4.1 | Interligações internas ( $x$ ) e de fronteira ( $y$ ) das células                                                                                                                        | 55  |
| 4.2 | Largura dos barramentos de entrada e saída em número de interligações com as células. Multiplicando as expressões por $b$ , obtém-se a largura em número total de bits, com $b \geq 1$ . | 59  |
| 4.3 | Modos de operação da célula e sua descrição                                                                                                                                              | 59  |
| 4.4 | Largura dos barramentos de entrada e saída de uma célula em número de bits, com $b \geq 1$ .                                                                                             | 60  |
| 4.5 | Organização do barramento de entrada por indexação em $i$ de $b$ bits                                                                                                                    | 60  |
| 4.6 | Organização do banco de registos de controlo                                                                                                                                             | 71  |
| 4.7 | Comandos suportados pelo protocolo implementado para a comunicação série.                                                                                                                | 76  |
| 5.1 | Identificadores e barramentos para referenciar as células da vizinhança.                                                                                                                 | 97  |
| 5.2 | Descrição dos ficheiros incluídos num subdiretório de especificação de um AC                                                                                                             | 109 |
| 6.1 | Tempo necessário para ler e carregar a matriz do AC em ciclos de relógio com $b = 1$ .                                                                                                   | 118 |
| 6.2 | Tempo necessário, em milisegundos, para transmitir os bits de estado das células do AC, em função de $c$ , $r$ com $B = 460800$ e $b = 1$ .                                              | 120 |
| 6.3 | Recursos disponíveis na FPGA Spartan6                                                                                                                                                    | 121 |
| 6.4 | Recursos ocupados pelos núcleos do sistema de hardware                                                                                                                                   | 121 |
| 6.5 | Recursos ocupados e desempenho do AC do Jogo da Vida em função das suas dimensões $r \times c$                                                                                           | 127 |
| 6.6 | Recursos ocupados e desempenho do AC do modelo de incêndio florestal em função das suas dimensões $r \times c$                                                                           | 128 |
| 6.7 | Recursos ocupados e desempenho do AC de <i>lattice gases</i> em função das suas dimensões $r \times c$                                                                                   | 129 |
| 6.8 | Recursos ocupados e desempenho do AC de Greenberg-Hastings em função das suas dimensões $r \times c$                                                                                     | 130 |

|      |                                                                                                                                                                                                 |     |
|------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 6.9  | Recursos ocupados e desempenho dos ACs unidimensionais de Wolfram, regra 30 e 110, em função da sua largura $L$ . . . . .                                                                       | 131 |
| 6.10 | Tempos efetivos de simulação $T_{SW}$ , em segundos, de um AC do Jogo da Vida no XLife em função das suas dimensões. . . . .                                                                    | 132 |
| 6.11 | Tempos efetivos de simulação $T_{HW}$ , em segundos, de um AC do Jogo da Vida na FPGA em função das suas dimensões. . . . .                                                                     | 132 |
| 6.12 | Ganhos em velocidade de simulação ( <i>speed-up</i> ) $T_{SW}/T_{HW}$ aproximados de um AC do Jogo da Vida em função das suas dimensões. . . . .                                                | 132 |
| 6.13 | Tempos totais $T_A$ , em segundos, desde o início de uma simulação em hardware até obter o resultado na aplicação de software, para um AC do Jogo da Vida em função das suas dimensões. . . . . | 132 |



# Abreviaturas e Símbolos

|       |                                                                   |
|-------|-------------------------------------------------------------------|
| AC'97 | Audio Codec '97                                                   |
| AC    | Autómato Celular                                                  |
| API   | Application Programming Interface                                 |
| ASCII | American Standard Code for Information Interchange                |
| BMM   | Block RAM Memory Map                                              |
| BRAM  | Block RAM                                                         |
| CMOS  | Complementary Metal-Oxide-Semiconductor                           |
| CPU   | Central Processing Unit                                           |
| DCM   | Digital Clock Manager                                             |
| ELF   | Executable and Linkable Format                                    |
| FIFO  | First In First Out                                                |
| FPGA  | Field-Programmable Gate Array                                     |
| FSM   | Finite State Machine                                              |
| HDL   | Hardware Description Language                                     |
| LMB   | Local Memory Bus                                                  |
| LUT   | Look-Up Table                                                     |
| IP    | Intellectual Property                                             |
| ISE   | Integrated Software Environment                                   |
| JTAG  | Joint Test Action Group                                           |
| LFSR  | Linear Feedback Shift-Register                                    |
| MHS   | Microprocessor Hardware Specification                             |
| NCD   | Native Circuit Description                                        |
| NCF   | Netlist Constraints File                                          |
| NGC   | Native Generic Circuit                                            |
| PCM   | Pulse Code Modulation                                             |
| PISO  | Parallel In Serial Out                                            |
| PLB   | Processor Local Bus                                               |
| RAM   | Random-Access Memory                                              |
| RGB   | Red, Green, Blue                                                  |
| ROM   | Read-Only Memory                                                  |
| SCH   | Sistema de Controlo de Hardware                                   |
| SCS   | Sistema de Controlo de Software                                   |
| SDF   | Standard Delay Format                                             |
| SDK   | Software Development Kit                                          |
| SIPO  | Serial In Parallel Out                                            |
| SR    | Shift-register                                                    |
| SRAM  | Static Random-Acess Memory                                        |
| VHDL  | Very-high-speed integrated circuits Hardware Description Language |
| VLSI  | Very Large Scale Integration                                      |
| UART  | Universal Asynchronous Receiver Transmitter                       |
| UCF   | User Constraints File                                             |
| USB   | Universal Serial Bus                                              |
| TRACE | Timing Reporter And Circuit Evaluator                             |
| XPS   | Xilinx Plataform Studio                                           |
| XST   | Xilinx Synthesis Technology                                       |

# Capítulo 1

## Introdução

Desde à vários anos, a aplicação de modelos baseados em autómatos celulares (ACs) têm sido extensivamente explorada pela comunidade científica para o estudo de características e simulação do comportamento de sistemas dinâmicos complexos, fenómenos naturais ou artificiais, ao longo do tempo, e processamento computacional para aplicações específicas, como por exemplo em planeamento urbano [6, 7], simulação de tráfego [8, 9], geração de sequências pseudo-aleatórias [10, 11, 12], redes complexas [13], biologia [14], modelação do coração humano [15], física [16, 17], processamento de imagem [18].

Os ACs são modelos matemáticos e sistemas dinâmicos, cuja evolução temporal e espacial é discreta. Estes consistem num elevado número de células, ligadas entre si com conectividade local, com funcionalidade idêntica organizadas numa matriz regular ou irregular com determinadas dimensões. Geralmente, os ACs mais comuns são unidimensionais ou bidimensionais embora, na prática, também se usem modelos tridimensionais. No primeiro caso representam-se por um vetor ou anel de células e no segundo por um plano ou um toróide. Se a condição fronteira for contínua, os extremos do vetor e do plano de células são fechados entre si formando um anel e um toróide. Como a evolução temporal é discreta, a cada iteração, cada célula atualiza o seu estado, em sincronismo com as restantes, em função do seu estado atual e do estado das células vizinhas circundantes, e não necessariamente adjacentes, e o AC evolui a partir de uma condição inicial autonomamente. O estado de cada célula é representado por um número fixo de bits, inteiro para ACs discretos e real para ACs contínuos, e a função ou a regra que descreve a transição de estado é descrita por operações aritméticas e/ou lógicas entre os bits de estado. Contudo, a regra de transição de estado pode depender de outros fatores, como por exemplo uma ou mais probabilidades, que dependem dos cenários de teste e aplicação em causa, o que torna um AC não homogêneo ou não uniforme e probabilístico em vez de determinístico, ou seja, a funcionalidade das células não é idêntica.

A necessidade de criar ambientes de simulação de elevado grau de complexidade que descrevam minuciosamente o comportamento de um sistema, e criar algoritmos para processamento de dados que realizem milhares de cálculos por segundo, têm, na prática, um elevado custo computacional cujo tempo de execução em aplicações de software pode ser dispendioso, mesmo tirando

proveito das arquiteturas *multi-core* para otimização e divisão da execução de tarefas em *threads* distintos. Em alternativa, procuram-se soluções de hardware que permitam não apenas reduzir o tempo de execução mas também a complexidade de implementação. Os sistemas baseados em reconfiguração, as FPGAs (*Field-Programmable Gate Array*), são possíveis candidatos como plataformas de desenvolvimento de hardware customizável e paralelismo inerente, para acelerar sistemas dedicados cujas arquiteturas são baseadas em modelos de ACs, uma vez que proporcionam ganhos em desempenho de excelência quando comparados com implementações em software otimizado, que emulam, por vezes, de forma sequencial, o processamento paralelo inerente aos sistemas baseados em ACs que contribuí consideravelmente para o aumento do tempo de execução.

Em contraste com a arquitetura tradicional de processamento sequencial, introduzida por John von Neumann nos anos 50, como modelo para a computação digital moderna, mais recentemente surgiu a motivação para investigar arquiteturas alternativas para o processamento em paralelo. As arquiteturas baseadas em ACs digitais são também uma possível abordagem a considerar [19]. A rápida evolução da tecnologia de circuitos integrados e o aparecimento de arquiteturas *multi-core*, derivado das restrições impostas pela tecnologia de semicondutores, veio a contribuir para a esta motivação uma vez que a tecnologia CMOS (*Complementary Metal-Oxide-Semiconductor*) tende para atingir os seus limites físicos na redução do tamanho de componentes em circuitos VLSI (*Very Large Scale Integration*).

O paralelismo inerente dos ACs, a sua natureza discreta, a arquitetura e poder computacional faz com estes tenham o potencial necessário para realizar operações complexas com um elevado grau de eficiência, robustez e simplicidade. Os ACs podem ser usados para processamento em paralelo de alta velocidade em simulações ou aplicações de elevado custo computacional. Assim, os sistemas baseados em ACs são candidatos ideais para implementação em FPGA, uma vez que a arquitetura destes dispositivos proporciona paralelismo espacial, os blocos que a constituem são reconfiguráveis e as suas interligações são redirecionáveis, o estado dos registos internos podem ser atualizados em paralelo a partir de funções lógicas implementadas em LUTs (*Look-Up Tables*) e sua estrutura arquitetural é idêntica a uma matriz regular. Desta forma, a funcionalidade de cada célula pode ser mapeada em LUTs, o seu estado atual armazenado em registos (*flip-flops*) ou em *block* RAM (BRAM) e a cada ciclo de relógio o estado de cada célula ou de um grupo de células pode ser atualizado em paralelo.

## 1.1 Objetivos e motivação

O trabalho consiste em desenvolver um sistema numa plataforma de desenvolvimento de hardware, uma FPGA Spartan6 da Xilinx, que implemente uma arquitetura parametrizável para a geração automática de ACs, cujas características e regras são especificadas pelo utilizador externo para uma aplicação em concreto.

A interação entre o utilizador e o autómato celular é feita por meio de uma aplicação de software de ambiente gráfico, um *framework* de trabalho e *front-end* do utilizador, que permite especificar as características e regras de um AC, controlar a sua operação, inicializá-lo, simulá-lo,

ler e visualizar graficamente o seu estado ao fim de  $n$  iterações. A especificação completa de um AC é integrada nos *templates* pré-construídos que descrevem a arquitetura dos módulos do sistema de hardware, cujas características são parâmetros de configuração desses mesmos módulos.

A aplicação de software implementa o *design flow* típico para FPGA, recorrendo às ferramentas disponibilizadas pela Xilinx, desde a síntese dos circuitos à sua implementação no dispositivo alvo e, por fim, a geração do ficheiro de *bitstream* que descreve a lógica da FPGA e das suas interligações internas da arquitetura final do sistema de hardware, para uma determinada especificação de um AC. O número de especificações de ACs distintos é ilimitado, desde que sejam cumpridas as restrições impostas pela arquitetura do sistema de hardware quanto à especificação das características e regras, e os ficheiros de *bitstream* gerados são guardados para uso futuro.

A arquitetura do sistema de hardware deve ser o mais flexível possível, tanto na especificação das características como das regras que permitem calcular o estado seguinte de cada célula, de modo a abranger e suportar cenários de operação distintos. Desta forma, as dimensões da matriz do AC, o número de bits de estado por célula, o tipo de vizinhança a considerar em torno de uma célula e a condição fronteira são características passíveis de configuração por parte do utilizador. A função de transição de estado de cada célula deve também ser específica pelo utilizador, a partir de operações aritméticas e lógicas entre os bits de estado das células de uma vizinhança, de forma a descrever o comportamento do AC ao longo do tempo.

Cada especificação consiste num projeto de um AC final, cuja arquitetura de hardware é dedicada na sua totalidade a essa mesma especificação, isto é, o objetivo é fixar logo de início as características e regras desejadas para o AC, pois estas não devem ser modificadas em *run-time*. Assim, para, por exemplo, modificar as dimensões de uma especificação de um AC já existente, deve-se gerar o novo ficheiro de *bitstream* com as modificações desejadas. Embora o processo de geração do ficheiro de *bitstream* seja demorado e quanto maior a complexidade do AC, pior é o tempo que decorre desde a síntese até a uma implementação final bem sucedida, o que é a desvantagem desta abordagem, por outro lado, o *bitstream* associado a uma especificação de um AC só tem de ser gerado apenas uma vez. Além disso, uma arquitetura dedicada tem a vantagem de eliminar *overheads* que estariam presentes numa arquitetura genérica que fosse configurável em *run-time*. Nesta situação, a complexidade dos circuitos aumenta e, desta forma, a percentagem de área ocupada e recursos utilizados, para permitir a configuração das características do AC em *run-time*, aumenta também. Uma percentagem da lógica necessária para esse efeito é, por sua vez, desaproveitada em lógica de configuração e controlo extra, tendo um impacto negativo sobre as dimensões máximas do AC possíveis de gerar e, possivelmente, na frequência de operação do sistema.

## 1.2 Estrutura do documento

O restante conteúdo do documento está organizado da seguinte forma. No capítulo 2, é feita uma revisão da literatura no que diz respeito ao conceito de AC, à sua origem e características principais, explica-se o que é um dispositivo baseado em reconfiguração e a sua arquitetura típica,

nomeadamente as FPGAs da Xilinx, descrevem-se sinteticamente as características do modelo da Spartan6 usado para o desenvolvimento deste trabalho, explica-se detalhadamente o fluxo de projeto em FPGA e ilustram-se aplicações e implementações já existentes de ACs em FPGA e software. No capítulo 3, é feita a especificação dos sistemas de controlo de hardware e software, mostram-se quais são as suas funcionalidades, o grau de liberdade na especificação das características e regras do AC, o cenário típico de utilização e como é que estes interagem entre si e com o utilizador externo. No capítulo 4, é feita a especificação dos módulos da arquitetura do sistema de controlo de hardware. Ilustram-se os modos de operação do AC, os processos de inicialização e leitura, como é realizado o armazenamento e transferência de dados, qual é a arquitetura típica de cada célula, explica-se como é feito o controlo global do sistema e como é que a comunicação com a aplicação de software e processamento de dados transferidos se processa. No capítulo 5, especifica-se a arquitetura do sistema de controlo de software. São ilustradas as funcionalidades associadas aos elementos gráficos e menus da aplicação de software, explica-se como são geridos e manipulados os ficheiros do diretório do trabalho e a sua organização, qual o procedimento para especificar um AC e como é que essa especificação é integrada nos módulos que descrevem a arquitetura de hardware, como é que o utilizador interage com o AC e como é que este se representa graficamente, como é que são geridos os ficheiros de *bitstream* e como é que o *design flow* se integra na aplicação, e, por fim, explica-se como é que os dados recebidos e enviados, de e para a FPGA, são processamentos pela aplicação e mapeados numa representação gráfica. Destaca-se também o processo de simulação de um determinado AC a partir de um ficheiro de *bitstream* já existente. No capítulo 6, são apresentando os resultados globais obtidos em termos de desempenho e recursos ocupados da FPGA para diferentes tipos de especificações de ACs baseados em aplicações reais. Faz-se ainda uma comparação com aplicações de simulação em software. O capítulo 7 fecha esta dissertação com um resumo do trabalho realizado, considerações finais e propostas para futuras modificações a fazer ao sistema como um todo.

## Capítulo 2

# Revisão Bibliográfica

Neste capítulo é feito o levantamento do estado da arte no que diz respeito à origem, conceito, características e propriedades dos ACs e realçam-se ainda alguns modelos de ACs que foram mais popularizados e estudados com maior detalhe. São apresentadas também algumas implementações já existentes de ACs em FPGA.

Ilustra-se também a arquitetura genérica de uma FPGA e referem-se as características principais da FPGA Spartan6 da Xilinx, descreve-se detalhadamente o fluxo de projeto típico e passos intermédios para o desenvolvimento de circuitos digitais em arquiteturas baseadas em FPGA.

### 2.1 Autómatos celulares

Os ACs têm sido alvo de estudo já há mais de meio século. O conceito foi introduzido no início dos anos 50 por John von Neumann, alguns anos depois da sua proposta da arquitetura de computadores tradicional que se conhece hoje em dia, quando este estudava a hipótese de modelar a simular sistemas autónomos capazes de se auto-reproduzir [20], após sugestão do seu colega Stanislaw Ulam para encarar o problema com que se deparava recorrendo a uma abstração matemática do mesmo.

Os ACs são sistemas autónomos que evoluem de forma automática a partir de uma condição inicial e dinâmicos porque evoluem ao longo do tempo segundo um conjunto de regras bem definidas. Um AC consiste numa coleção de células ligadas entre si, formando uma estrutura regular ou irregular de tamanho variável, na prática, de uma a três dimensões. Para uma dimensão considera-se um vetor de células, para duas um plano ou matriz e para três, por exemplo, um cubo ou um paralelepípedo. A cada célula é atribuída um estado que pode ser uma quantidade discreta ou real de dois ou mais estados, sendo, no primeiro caso, para ACs discretos e no segundo para ACs contínuos. Os ACs mais simples têm apenas dois estados por célula, 1 ou 0, e designam-se por ACs unidimensionais. Na literatura, por vezes, também se referem como elementares e por se caracterizarem por apenas dois estados distintos, também é-lhes atribuída a designação de binários. O tempo evolui de forma discreta e, em cada iteração, o próximo estado de uma célula é calculado em sincronismo e em paralelo com as restantes e define-se, matematicamente, como uma função

do seu estado atual e dos estados das células de uma vizinhança local. Considera-se como uma vizinhança local o conjunto de células em torno de uma célula central, adjacentes (imediatas) ou não, cujo número de células depende do tipo de vizinhança definida e da estrutura de cada célula. Tipicamente, as células são representadas numa grelha em que cada elemento é um quadrado para uma e duas dimensões e um cubo para três dimensões, o que implica que a vizinhança de células adjacentes pode ser até 3, 9 e 25, respetivamente, incluindo a célula central. Contudo, vizinhanças, por exemplo, triangulares ou hexagonais também são uma possibilidade o que reduz o número de células vizinhas adjacentes para 3 e 6, respetivamente. A função de transição de estado pode ser descrita por uma LUT, em que cada bit de estado resulta de uma combinação dos bits de estado correspondentes das células vizinhas que define a regra de transição de estados local a cada célula. Esta regra diz-se determinística se esta não se modificar ao longo do tempo, ou probabilística se depender de um fator de natureza aleatória; diz-se também como sendo linear se o estado seguinte depender de uma combinação de operações lineares dos estados atuais das células da vizinhança considerada e da célula central. Assim, um AC designa-se como sendo uniforme ou homogêneo se todas as células obedecerem à mesma regra de transição de estado, caso contrário designam-se por não uniformes ou heterogêneos. Uma vez que as dimensões de um AC não se podem expandir até ao infinito pois a sua representação seria impraticável, especialmente em aplicações de software ou hardware que implementem estes modelos, há que definir condições fronteira nos limites das estruturas que representam um AC. No caso dos ACs unidimensionais, diz-se ter uma fronteira nula se os extremos do vetor são representados por células virtuais de estado nulo ou uma fronteira contínua, ou periódica, se as células dos extremos do vetor se encontrarem ligadas uma à outra formando um anel de células. Para os ACs bidimensionais, com uma condição fronteira do tipo contínua, o plano é fechado entre si num toróide.

### 2.1.1 Vizinhança de von Neumann

A vizinhança de von Neumann caracteriza-se por uma conectividade de 4 células ortogonalmente adjacentes a uma célula central, de distância  $r = 1$ , num AC bidimensional [21]. Embora, geralmente, em ACs bidimensionais, a vizinhança de von Neumann considere apenas as células referidas, o conceito pode ser expandido ao alargar a vizinhança para um conjunto de células a uma distância  $r > 1$  da célula central. A distância de Manhattan ou *city-block* define o conjunto de células pertencentes à vizinhança de von Neumann quando  $r > 1$ .

Na figura 2.1, ilustra-se a distância de Manhattan  $d_m$  entre duas células num plano de coordenadas cartesianas  $(i, j)$ , que é definida pela soma da diferença absoluta entre as coordenadas de cada célula. Como  $C_0 = (C_{0i} = 5, C_{0j} = 0)$  e  $C_1 = (C_{1i} = 0, C_{1j} = 5)$ , então  $d_m(C_1, C_2) = |C_{0i} - C_{1i}| + |C_{0j} - C_{1j}| = 10$ . A distância  $d_m$  define a soma do comprimento de  $n_s$  segmentos projetados em  $i$  e  $j$  necessários para ligar as duas células. Se a distância entre células ortogonalmente adjacentes for unitária, tem-se que  $d_m = n_s$ , como apresentado na figura pelos segmentos a vermelho, que é equivalente à distância mínima de Manhattan. O caminho estabelecido pelos segmentos a verde e azul também definem a distância de Manhattan, porque estabelecem da mesma forma o caminho mais curto entre as duas células. Da mesma forma, o caminho alternativo representado a

rosa também é válido, já que a distância de Manhattan percorrida entre as células de coordenadas  $(1, 1)$  e de  $(0, 3)$  é a mesma que pelo caminho a verde.

A figura 2.2, ilustra a vizinhança tradicional de von Neumann, representada a vermelho, em torno de uma célula central ( $r = 1$ ), representada a preto, e quando  $r = 2$  e  $r = 3$  a vizinhança é alargada e são incluídas as células a azul e verde, respetivamente, dando à vizinhança uma aparência um diamante. Na figura, a distância  $d_m \Leftrightarrow r$  está marcada nas células e o número de células da vizinhança de von Neumann em função do raio de alcance  $r$  é dado por  $N = 2 \cdot r \cdot (r + 1) + 1$ . Assim, para  $r = \{1, 2, 3\}$  tem-se que  $N = \{5, 13, 25\}$ , respetivamente, em que a célula central é contabilizada. Para ACs tridimensionais com estrutura celular é cúbica, por exemplo, quando  $r = 1$ , tem-se que  $N = 7$  e a forma de diamante estende-se para o espaço tridimensional dando assim a aparência de um octaedro.

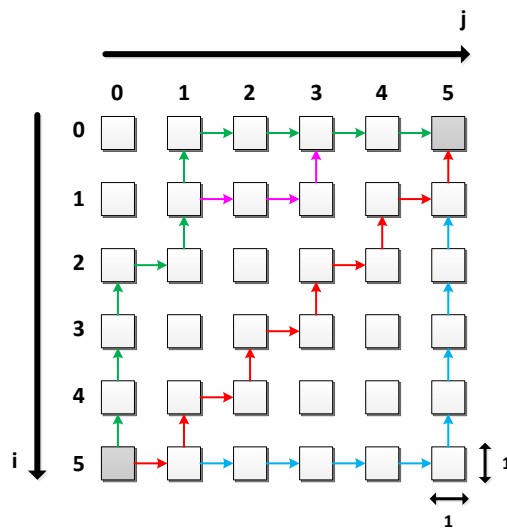


Figura 2.1: Distância de Manhattan

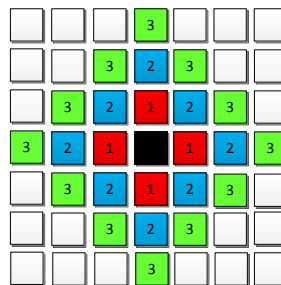


Figura 2.2: Vizinhança de von Neumann em função de  $r$

### 2.1.2 Vizinha de Moore

A vizinhança de Moore é uma extensão da vizinha de von Neumann na medida em que além de se considerar as células ortogonalmente adjacentes em torno de uma célula central, consideram-se também as células diagonalmente adjacentes, estabelecendo assim conectividade de 8 de células com distância  $r = 1$  da célula central [22]. Da mesma forma, pode-se expandir o conceito ao alargar a vizinhança para o conjunto de células tal abrangidas quando  $r > 1$ . A distância de Chebyshev define o conjunto de células que pertencem à vizinhança de Moore quando  $r > 1$ .

Na figura 2.3, ilustra a distância de Chebyshev  $d_c$  entre duas células num plano de coordenadas cartesianas  $(i, j)$ , que é definida como a maior diferença absoluta entre as abcissas e as ordenadas das suas coordenadas. Como  $C_0 = (C_{0i} = 5, C_{0j} = 0)$  e  $C_1 = (C_{1i} = 1, C_{1j} = 2)$ , então  $d_c(C_0, C_1) = \max(|C_{0i} - C_{1i}|, |C_{0j} - C_{1j}|) = 4$ . A distância  $d_c$ , com distância entre células unitária, define a distância de Chebyshev, que é mínima, e o número de células que é necessário percorrer para ir desde a célula  $C_0$  até à  $C_1$ , como representado na figura. Repare-se que o caminho estabelecido pelos segmentos a verde não define a distância de Chebyshev, uma vez que a distância entre as células de coordenadas  $(5, 0)$  e a de  $(4, 1)$  pode se coberta com uma deslocação única na diagonal, representada a vermelho, em vez de duas deslocações, uma deslocação horizontal e outra vertical.

A figura 2.4, ilustra a vizinhança tradicional de Moore, representada a vermelho, em torno de uma célula central ( $r = 1$ ), representada a preto, e quando  $r = 2$  e  $r = 3$  a vizinhança é alargada e são incluídas as células a azul e verde, respetivamente, dando à vizinhança uma aparência um quadrado. Na figura, a distância  $d_c \Leftrightarrow r$  está marcada nas células e o número de células da vizinhança de Moore em função do raio de alcance  $r$  é dado por  $M = (2 \cdot r + 1)^2$ . Assim, para  $r = \{1, 2, 3\}$  tem-se que  $N = \{9, 25, 49\}$ , respetivamente, em que a célula central é contabilizada. Para ACs tridimensionais com estrutura celular é cúbica, por exemplo, quando  $r = 1$ , tem-se que  $N = 27$  e a forma de quadrado estende-se para o espaço tridimensional dando assim a aparência de um cubo.

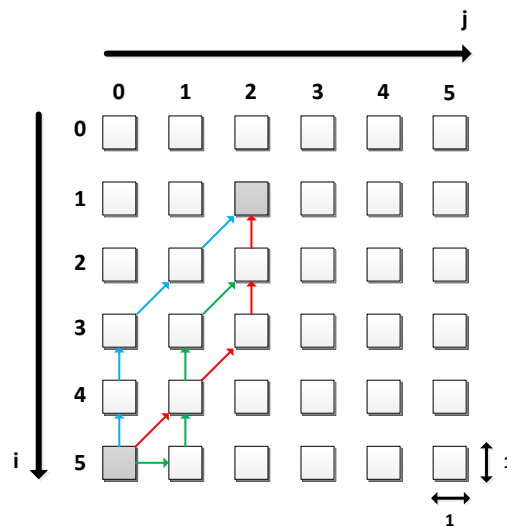


Figura 2.3: Distância de Chebyshev

Figura 2.4: Vizinhança de Moore em função de  $r$ 

### 2.1.3 Autômatos unidimensionais (1D)

Stephen Wolfram iniciou a investigação em ACs nos anos 80, estudou e escreveu durante vários anos, em particular, sobre ACs unidimensionais [23]. Mais recentemente, escreveu extensivamente sobre ACs e a sua aplicação em vários domínios da ciência [24]. Wolfram caracteriza os ACs como sendo um modelo matemático que, a partir de um conjunto de regras simples, são capazes de exibir vários tipo de comportamento que permitem assim estudar e modelar a dinâmica dos mais variados sistemas complexos. Wolfram observou que dependendo da condição inicial imposta e das regras aplicadas, os ACs exibem padrões e comportamentos distintos mas, por vezes, com formas semelhantes; este classificou os ACs em quatro classes distintas [25]:

- Classe 1 — Evolução para um estado de homogeneidade em que todas as células tomam e fixam o mesmo valor; atinge-se um estado de regime estacionário.
- Classe 2 — Evolução para um conjunto de formas que evoluem ciclicamente, estabelecendo um padrão periódico.
- Classe 3 — Evolução para um comportamento aperiódico em que as formas produzidas, em cada iteração, variam ao longo do tempo e do espaço; comportamento imprevisível e caótico.
- Classe 4 — Evolução para conjuntos de estruturas complexas, por vezes propagando-se;

Wolfram classificou os ACs unidimensionais convenientemente de forma a referir a regra do AC a partir da sua representação decimal equivalente [26, 27]. Sendo os ACs unidimensionais lineares, a vizinhança em torno de uma célula central estende-se para a sua esquerda e para a sua direita. No caso mais simples, consideram-se apenas as células imediatamente à esquerda e à direita e o estado de cada uma é binário (1 ou 0, preto e branco respetivamente). Considere-se que  $c_i(t)$  é o estado da célula de índice  $i$  no instante de tempo  $t$ . Desta forma, o estado das células da vizinhança imediata é dado por  $c_{i-1}(t)$  e  $c_{i+1}(t)$ . No instante de tempo  $t + 1$ ,  $c_i(t + 1)$  é calculado em função de  $c_i(t)$ ,  $c_{i-1}(t)$  e  $c_{i+1}(t)$ . Na figura 2.5, ilustram-se as evoluções temporais, ao longo da segunda dimensão, dos ACs unidimensionais de regra 30 e 110 em que cada linha representa um instante de tempo  $t$ . Cada regra define-se pela saída das  $2^3 = 8$  combinações possíveis do estado

de uma vizinhança de três células no instante  $t$  e, no instante  $t + 1$ , o estado da célula central é calculado em função da combinação anterior. O conjunto dos 8 bits de saída define um número entre 0 e 255 que representa a regra em questão. Assim,  $c_i(t + 1) = f(c_{i-1}(t), c_i(t), c_{i+1}(t))$ . As regras 30 e 110 tem especial interesse uma vez que a primeira é usada como um gerador de números aleatórios, tomando a coluna central da representação, cuja condição inicial é colocar a primeira célula dessa mesma coluna com estado 1, e interpretando os estados das células como uma representação binária a partir da leitura de bits sucessivos. A segunda porque exibe, tal como uma máquina de Turing, um carácter de universalidade [28].

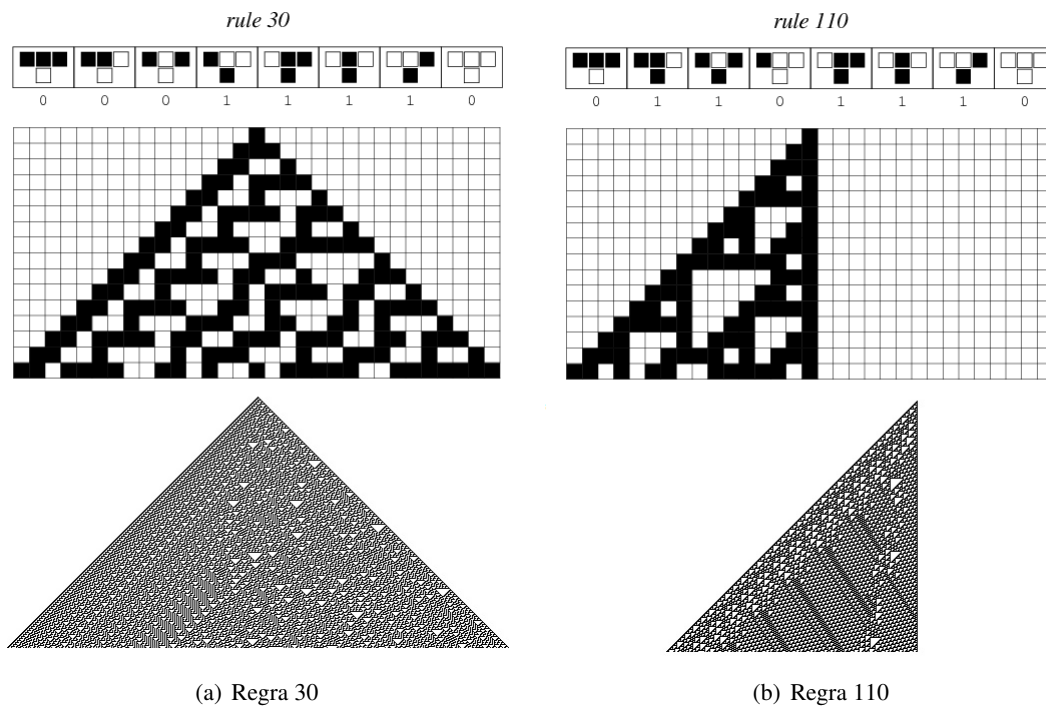


Figura 2.5: Evolução da regra 30 (fonte: [1]) e regra 110 (fonte: [2]) ao fim 16 e 250 iterações

## 2.1.4 Autómatos bidimensionais (2D)

Nesta secção apresentam-se alguns modelos de ACs bidimensionais.

### 2.1.4.1 O Jogo da Vida

O “Jogo da Vida” [29], inventado pelo matemático inglês John Horton Conway, é um AC bidimensional totalístico [30] e determinístico que foi introduzido e popularizado, nos anos 70, na secção de jogos matemáticos de Martin Gardner na revista científica *Scientific American*. Cada célula encontra-se em um de dois estados (0 ou 1); o estado 0 indica que a célula está morta e o estado 1 que célula está viva. O estado de cada célula, para cada nova geração, define-se em

função do estado atual das 8 células adjacentes a uma célula central, pelo que o tipo de vizinhança é de Moore. A transição de estado ocorre de acordo com as seguintes regras:

- Morte — uma célula viva morre na próxima geração se na sua vizinhança existem mais de três células vivas e menos de duas.
- Sobrevivência — uma célula sobrevive, isto é, o seu estado mantém-se a 1, se na sua vizinhança existem exatamente duas ou três células vivas.
- Nascimento — uma célula nasce, isto é, ocorre uma transição de estado  $0 \rightarrow 1$ , apenas se na sua vizinhança existem exatamente três células vivas.

Este AC específico exhibe uma vasta variedade de formas complexas e comportamentos consoante a condição inicial que lhe é aplicada. As células podem simplesmente extinguir-se, exibir padrões periódicos que se propagam ou não no espaço ao longo do tempo, exibir padrões irregulares, atingir um estado estacionário em que todo o sistema não evolui mais no tempo e até exibir formas que se auto-reproduzem. Há muitos tipos de padrões que ocorrem no Jogo da Vida, como por exemplo formas de vida estáticas, padrões que se repetem, designados por osciladores, e percorrem ciclicamente um conjunto de configurações distintas de células e padrões que se deslocam ao longo da matriz de células, designados por naves espaciais. À semelhança do AC unidimensional de regra 110, o Jogo da Vida é um AC de carácter universal.

Existem uma variante do Jogo da Vida intitulada de Gerações cujas regras são idênticas, mas além dos dois estados possíveis existem outros tantos, um número arbitrário, que representam o histórico da evolução no tempo, ou seja, a idade das células. As células morrem quando atingem a idade máxima possível, estabelecida pelo estado de valor mais alto, independentemente do estado da vizinhança. Contudo, as células de estado  $s \geq 2$  não podem dar vida as novas células, em que  $s = 0$  indica a ausência de vida e  $s = 1$  uma nova forma de vida, acabada de nascer, que pode dar vida a novas células. Quando uma célula sobrevive entre iterações consecutivas, o seu estado é incrementado e a célula envelhece.

#### 2.1.4.2 Modelo de incêndios florestais

Um modelo de incêndio florestal e crescimento pode ser descrito por um AC bidimensional probabilístico. Cada célula pode encontrar-se em um de três estados: vazio, ocupado por uma árvore ou a arder. A vizinhança considerada é de von Neumann e o comportamento do modelo é descrito pelas seguintes regras [31]:

1. Uma célula que se encontra a arder torna-se numa célula vazia.
2. Uma árvore arde se pelo menos um dos seus vizinhos se encontra a arder.
3. Uma árvore pega fogo com probabilidade  $f$  mesmo que nenhum vizinho se encontre a arder (combustão instantânea).

4. Numa célula vazia nasce uma árvore com probabilidade  $p$ .

A dinâmica do modelo é controlada pela relação  $p/f$ , que, no fundo, se traduz quão rápido nascem novas árvores após o início de um incêndio. Assim, o ponto de início de um incêndio espalha-se e vai queimando árvores em seu redor; atrás da linha de fogo, nascem novas árvores. Para certos valores de  $p$  e  $f$ , ao fim de algum tempo, observam-se aglomerados de árvores queimadas e a nascer, tipicamente quando  $f \ll p$ , caso contrário observam-se distribuições aleatórias de células com árvores a arder ou não e células vazias.

#### 2.1.4.3 Sementes

Sementes é o nome de um AC bidimensional determinístico, semelhante ao Jogo da Vida, inventado por Brian Silverman. Cada célula tem da mesma forma dois estados distintos e a mesma vizinhança é considerada, contudo as regras modificam-se ligeiramente:

- Uma célula viva morre independentemente.
- Uma célula nasce se tiver exatamente duas células vizinhas vivas.

#### 2.1.4.4 O cérebro de Brian

O cérebro de Brian é um AC bidimensional determinístico, semelhante ao Jogo da Vida, também inventado por Brian Silverman. Este AC em vez de dois tem três estados por célula que indicam que uma célula pode estar morta, a morrer ou viva. A vizinhança considerada é de Moore. As regras são as seguintes:

- Uma célula viva entra no estado “a morrer” independentemente.
- Uma célula nasce apenas se tiver exatamente duas células vizinhas vivas.
- Uma célula no estado “a morrer”, morre na iteração seguinte.

#### 2.1.4.5 Wireworld

O *Wireworld* é um AC bidimensional determinístico de quatro estados por célula introduzido por Brian Silverman em 1987 [32]. Dos ACs bidimensionais este é particularmente interessante devido à sua aplicação prática. As regras deste AC são tais que permitem simular o comportamento de elementos lógicos digitais, como por exemplo, portas lógicas do tipo E, OU, OU exclusivo, inversores, flip-flops e entre outros. De forma geral, as regras descrevem como é que o fluxo de corrente ao longo de condutores é realizado. A vizinhança considerada é de Moore e as regras são as seguintes:

1. Para células de estado igual a zero, indica-se a presença de um elemento não condutor.
2. Para células de estado igual a um, considera-se que é a cabeça de um eletrão; na iteração seguinte esta célula passa a ser a cauda do eletrão respetivo.

3. Para células de estado igual a dois, considera-se que é a cauda de um eletrão; na iteração seguinte a cauda propaga-se para as células onde se encontram as cabeças respetivas deixando para trás o condutor livre para a passagem de outro eletrão.
4. Para células de estado igual a três, indica-se a presença de um condutor que é o meio por onde os eletrões se deslocam; uma célula de condutor é ocupada por uma cabeça de um eletrão caso na sua vizinhança esteja exatamente uma ou duas cabeças de eletrão.

Repare-se que cada eletrão é representado por dois estados distintos (um e dois) e ocorrem sempre par a par, pelo que não é possível uma célula de estado um e dois encontrarem-se isoladas (exceto quando um eletrão dirige para um circuito aberto), condição necessária para estabelecer o sentido e direção do movimento do eletrão. A presença ou ausência de um eletrão num condutor indica a passagem de bit 1 ou 0, respetivamente. A cada iteração, um eletrão desloca-se de uma distância equivalente a uma célula.

A forma como se define um elemento lógico depende única e exclusivamente de como os condutores estão organizados e distribuídos, pelo que as suas formas podem limitar a propagação de sinal ao longo dos mesmos. A regra número quatro indica como é que a propagação de sinal pode ser limitada. Se dois condutores se cruzarem entre si na perpendicular, estabelecendo um curto-circuito, um eletrão que viaje, por exemplo, na direção horizontal, ao alcançar a interseção este divide-se em dois eletrões que se deslocam em sentidos opostos agora na direção vertical. Na direção horizontal, para onde o eletrão se deslocava, agora, nada acontece. Por outro lado, quando dois eletrões se deslocam ao longo de um condutor em sentidos contrários, sem qualquer caminho alternativo, estes anulam-se um ao outro.

## 2.2 *Field-Gate Programmable Array (FPGA)*

Uma FPGA é um dispositivo de hardware e um caso particular de uma família de circuitos integrados projetada para ser reconfigurável pelo utilizador, o que significa que pode ser reprogramada um número infinito de vezes. Reconfigurar uma FPGA entende-se por modificar o mapeamento da lógica interna para obter uma determinada funcionalidade. Os circuitos que exprimem a funcionalidade pretendida podem ser descritos por meio de um esquemático onde se interligam componentes lógicos (portas lógicas, *flip-flops*, ...) ou por uma linguagem de descrição de hardware (HDL), tipicamente Verilog HDL ou VHDL (*Very-high-speed integrated circuits Hardware Description Language*), que permite descrever a sua estrutura ou comportamento. As estruturas de código suportadas por estas linguagens são posteriormente traduzidas e mapeadas em circuitos efetivos na FPGA. Para isso, utilizam-se ferramentas dedicadas para o efeito que permitem realizar a tradução, mapeamento e implementação e gerar um ficheiro binário, designado por *bitstream*, que contenha a configuração da lógica interna da FPGA para uma determinada arquitetura especificada. Técnicas especiais permitem também modificar parcialmente o mapeamento interno da lógica de hardware e em *run-time*; esta técnica designa-se por reconfiguração parcial e dinâmica, respetivamente [3].

A arquitetura genérica de uma FPGA consiste em três blocos principais: Blocos Lógicos Configuráveis (CLB), Blocos de Entrada e Saída (IOB) e lógica de comunicação e interligação [3]. A arquitetura assemelha-se a uma matriz regular ou irregular, em que cada posição da mesma é um CLB e entre cada um deles existem linhas disponíveis para a sua interligação. Na figura 2.6 apresenta-se um diagrama de blocos da arquitetura de uma FPGA.

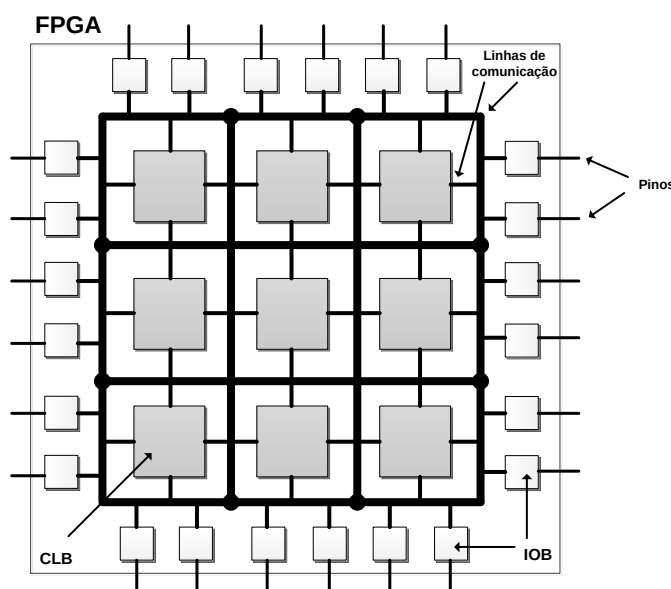


Figura 2.6: Arquitetura genérica de uma FPGA (segundo [3])

Um CLB é o bloco elementar da arquitetura de uma FPGA. Estes blocos são capazes de desempenhar funções lógicas específicas, de acordo com a sua configuração, através de LUTs. As LUTs armazenam num *flip-flop*, uma memória de 1 bit associada a cada LUT, o resultado de uma função para todas as combinações possíveis aplicáveis à sua entrada. Durante o processo de configuração da FPGA, as memórias das LUTs são escritas para implementar a função pretendida e a lógica de interligação em seu torno é configurada de modo a encaminhar os sinais de entrada e saída deste bloco. Entende-se por lógica de interligação conjuntos de portas lógicas, multiplexadores e *latches*. A arquitetura interna de um CLB, e termos do número de elementos lógicos, varia com o fabricante.

Os IOBs são responsáveis por fazer a interligação entre os sinais de entrada e saída e os pinos da FPGA. Estes blocos configuram a direção da comunicação dos pinos, de forma independente, e tem um bloco próprio de memória interna para configuração associada aos níveis de tensão do pino correspondente.

Para fazer as interligações entre os CLBs e os IOBs, uma FPGA dispõe essencialmente de dois modos distintos: interligação direta ou segmentada. Em interligação direta (ver figura 2.7(a)), os grupos de ligações são feitas ao longo de toda a área da FPGA e cruzam-se entre CLBs que ligam as suas saídas e entradas às linhas adequadas para a transmissão de dados. Por outro lado, em

interligação segmentada (ver figura 2.7(b)), nos pontos de cruzamento entre as linhas existem matrizes de comutação programáveis que permitem estabelecer os nós de interligação.

A configuração da lógica interna da FPGA é feita por uma ferramenta específica, uma aplicação de software, que permite descarregar o ficheiro de configuração, o *bitstream*, na memória SRAM (*Static Random-Access Memory*) do dispositivo. Sendo esta memória volátil, quando a alimentação do dispositivo é desligada e ligada de novo, a configuração anterior é perdida e deve ser de novo carregada. Contudo, algumas plataformas de desenvolvimento disponibilizam uma memória não-volátil ROM (*Read-Only Memory*) ou *flash*, onde o ficheiro de *bitstream* pode ser carregado e, quando a alimentação é ligada, o seu conteúdo é copiado para a memória SRAM configurando automaticamente a FPGA. Caso o ficheiro de *bitstream* seja parcial, apenas uma gama específica de endereços da SRAM é escrita.

Além dos três blocos principais referidos, as FPGAs mais recentes tem recursos adicionais, como por exemplo memórias dedicadas (*block RAM*), multiplicadores, processadores, processadores de sinal digital, etc. A vantagem de usar memória dedicada em vez de distribuída, é que a segunda consome recursos associados a lógica combinacional, ou seja, lógica reconfigurável (LUTs). Para aplicações em que é necessário armazenar um elevado volume de dados, é conveniente usar BRAM em vez de memória distribuída e salvaguardar lógica reconfigurável para outras funções. Os processadores podem ser do tipo *hard-core* ou *soft-core*; o primeiro equivale a um *chip* dedicado, como por exemplo o PowerPC, e o segundo equivale a processadores cuja arquitetura é construída diretamente a partir da lógica interna da FPGA, dedicando parte dos seus recursos para esse propósito, como por exemplo o MicroBlaze. Enquanto que um processador *hard-core* é uma entidade física, isto é, um circuito integrado, um *soft-core*, por outro lado, pode ser instanciado mais que uma vez na FPGA. Dependendo da arquitetura do projeto de hardware, os que incluem processadores designam-se por sistema embebidos e o projeto é dividido numa componente de hardware e software, sendo a segunda de desenvolvimento do(s) programa(s) que o(s) processador(es) executa(m).

### 2.2.1 Xilinx Spartan6 FPGA

Nesta secção, apresentam-se as características gerais da família de FPGAs Spartan6.

#### 2.2.1.1 CLBs, Slices e LUTs

Nesta família de FPGAs, cada CLB contém um par de *slices*, alinhados lado a lado em duas colunas distintas. Estes podem ser de três tipos: SLICEM, SLICEL e SLICEX [33].

Na Spartan6, cada *slice* contém quatro LUTs para lógica combinacional, oito *flip-flops* para lógica sequencial e lógica adicional (portas lógicas, multiplexadores, latches). As SLICEMs correspondem a 25% do número total de *slices* disponíveis. Cada uma das quatro LUTs de um SLICEM pode ser configurada como uma LUT de 6 bits de entrada e 1 bit de saída que é armazenado numa memória de 1 bit (*flip-flop*) ou configurada em duas LUTs de 5 bits de entrada cada, partilhando o mesmo barramento de endereço, e o sexto bit é usado para multiplexar as saídas independentes de

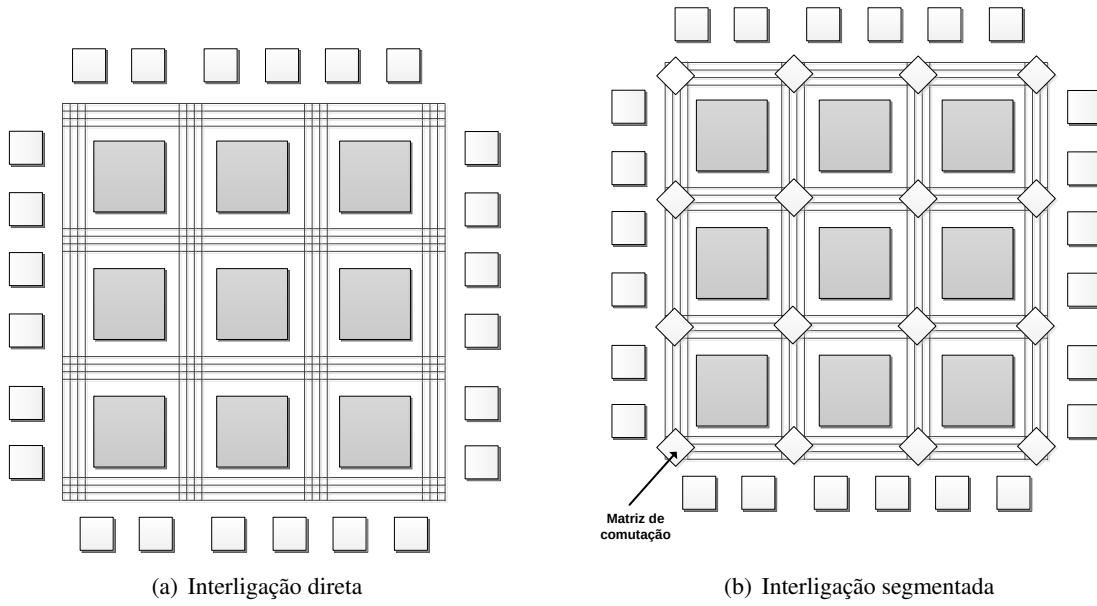


Figura 2.7: Modos de interligação de CLBs e IOBs (segundo [3])

ambas as LUTs. Estas LUTs podem ainda ser configuradas como memória RAM distribuída com 64 bits por endereço ou  $2 \times 32$  bits por LUT, como um *shift-register* de 32 bits ou dois de 16 bits cada, com comprimento endereçável.

As SLICELs correspondem, também, a 25% do número total de *slices* disponíveis. Estas são idênticas às SLICEMs, contudo não suportam a funcionalidade de memória distribuída e *shift-register*. Metade do número total de *slices* disponíveis correspondem a SLICEXs que são idênticas a SLICELs, mas não suportam propagação do sinal de *carry* em operações aritméticas nem multiplexadores largos, com um elevado número de entradas.

### 2.2.1.2 Block RAM

As BRAMs, células de memória dedicada, podem ser usadas de várias formas consoante as necessidades. A família Spartan6 oferece entre 12 a 268 BRAMs de acesso dual, cada uma com capacidade de 18 Kb [34]. Cada BRAM tem duas portas de acesso e endereçamento independentes, de largura configurável, que partilham os dados armazenados. Ainda assim, uma BRAM pode ser configurada para apenas acesso simples (uma porta) e como duas BRAMs de 9 Kb independentes. A relação entre a largura das portas de acesso  $r_W = W_{P_A}/W_{P_B}$  não é necessariamente unitária, o que significa que cada porta pode ter uma largura diferente da outra, desde que  $n = \log_2(r_W)$  com  $n \in \mathbb{N}_0$ . A profundidade total de uma memória, ou o número de endereços, deve ser  $P = 2^m$  bits, com  $m \in \mathbb{N}_0^+$ , em que  $m$  é o número de bits de endereçamento. A capacidade total  $C$  da memória é diretamente proporcional à largura da porta mais estreita,  $W_{min} = \min(W_{P_A}, W_{P_B})$ , o que significa

que  $C = W_{min} \cdot P$  bits. Assim, a profundidade associada à porta mais larga é:

$$P' = \begin{cases} \frac{C}{W_{min}} \cdot \frac{1}{r_W} & n > 0 \\ \frac{C}{W_{min}} \cdot r_W & n \leq 0 \end{cases}$$

Todos os acessos à BRAM, para leitura e escrita de dados, são controlados pelo sinal de relógio pelo que num único ciclo de relógio é possível ler e/ou escrever o número de bits equivalente à largura portas.

### 2.2.1.3 Sumário de recursos disponíveis

Na tabela 2.1, mostram-se os recursos que a FPGA Spartan6 oferece.

| Modelo   | Células lógicas | CLBs          |                   |                      | BRAMs |          |
|----------|-----------------|---------------|-------------------|----------------------|-------|----------|
|          |                 | <i>Slices</i> | <i>Flip-flops</i> | RAM distribuída (Kb) | 18 Kb | Max (Kb) |
| XC6SLX45 | 43661           | 6882          | 54576             | 401                  | 116   | 2088     |

Tabela 2.1: Recursos da FPGA Spartan6 [5]

Embora não se encontre na tabela, a arquitetura da Spartan6 também tem 58 *slices* DSP48A1, dedicados para aplicações de processamento digital de sinal, em que cada um tem um multiplicador de 18x18 bits de complemento para dois, um somador e um acumulador de 48 bits. O acumulador pode ser usado como um contador síncrono positivo e negativo e o multiplicador consegue realizar *barrel shifting*.

### 2.2.2 Fluxo de projeto

O fluxo típico de projeto para FPGA consiste no processo de criar, implementar, verificar e descarregar uma especificação no hardware. O fluxo completo descreve um processo iterativo em que se faz a verificação dos resultados obtidos em simulação comportamental e temporal e, posteriormente, a verificação direta no circuito físico. Uma especificação inicial é validada assim que se demonstre que a funcionalidade pretendida está correta (ver figura 2.8). O fluxo consiste então em três passos [4]:

**Especificação do projeto e síntese** Uma especificação de arquitetura pode ser feita de duas maneiras distintas e/ou interlaçar ambas. A primeira é com recurso a um editor de esquemáticos que permite especificar diretamente a estrutura do circuito com portas e elementos lógicos. A segunda é a partir de uma Linguagem de Descrição de Hardware (HDL), que permite fazer uma descrição textual dos circuitos não apenas em estrutura mas também em comportamento. Estas linguagens suportam um conjunto de regras, instruções e operadores que permitem organizar uma especificação em módulos distintos, com entradas e saídas, e descrever o seu comportamento a partir de estruturas de código que se traduzem em elementos lógicos de circuitos digitais, como *flip-flops*, multiplexadores, portas lógicas, somadores,

multiplexadores, LUTs, etc. A ferramenta da Xilinx designada por XST (*Xilinx Synthesis Technology*), permite traduzir uma descrição textual de comportamento para uma descrição estrutural, isto é, uma *netlist*; a este passo dá-se o nome de síntese e traduz um ou mais ficheiros HDL para um único ficheiro NGC (*Native Generic Circuit*).

**Implementação** Esta fase recebe um ficheiro NGC gerado no processo de síntese, cujo formato descreve uma estrutura lógica da especificação, e é convertido num formato de representação física dos circuitos da mesma para uma arquitetura de FPGA alvo. O formato de saída é designado por NCD (*Native Circuit Description*) que contém a informação associada ao mapeamento da lógica da especificação aos componentes físicos da FPGA. Este ficheiro pode ser posteriormente usado para gerar o ficheiro de configuração da FPGA, um ficheiro binário designado por *bitstream*, cujo conteúdo define a configuração da lógica interna da FPGA e encaminhamento de interligações. O *bitstream* é descarregado na FPGA a partir da ferramenta da Xilinx específica para o efeito, o iMPACT. A ligação entre um PC e a FPGA, dependendo da plataforma de desenvolvimento, pode ser feita através de um cabo USB (*Universal Serial Bus*), série ou paralelo para uma interface JTAG (*Joint Test Action Group*).

**Verificação** A verificação é feita a nível funcional e temporal, a partir da construção de uma bancada de teste (*testbench*), de modo a assegurar que a funcionalidade pretendida é a correta e que não existem violações temporais. Esta pode ser feita com um simulador RTL, aplicando estímulos nas entradas e ver o comportamento das saídas, ou então fazer *debugging* diretamente na FPGA a partir, também, da interface JTAG. A Xilinx providencia um simulador RTL, o ISim, com uma interface gráfica onde se pode visualizar as formas de onda.

Na figura 2.8, apresenta-se um diagrama de alto nível do fluxo típico de projeto para FPGA. A geração de informação atrasos corresponde à criação do ficheiro SDF (*Standard Delay Format*), durante a fase de implementação, que especifica os atrasos nos elementos de arquitetura de hardware e das suas interligações. Esta informação é usada para a simulação temporal dos circuitos para a verificação do cumprimento das restrições temporais [35].

A análise temporal estática ocorre depois da fase de implementação (mapeamento, colocação e encaminhamento) e antes da simulação temporal, e tem como objetivo analisar os atrasos induzidos nas interligações dos circuitos e concluir se respeitam ou não as restrições temporais especificadas. Esta análise não inclui a aplicação de vetores de estímulos nas entradas, daí ser simplesmente estática, avaliando apenas o impacto a nível temporal de propagar um ou mais bits por um caminho específico. O caminho que introduz o maior atraso no circuito designa-se por caminho crítico, e é o que limita a frequência máxima de operação do sistema.

A simulação funcional ocorre depois do processo de síntese e, ao contrário da simulação temporal, as restrições temporais não são tidas em conta. Desta forma, o comportamento do circuito pode ser avaliado como se operasse num cenário ideal, onde não fossem induzidos atrasos de propagação nas interligações no circuito real. Esta simulação é particularmente útil para detetar erros de projeto numa fase inicial, associados à funcionalidade, que normalmente acontecem.

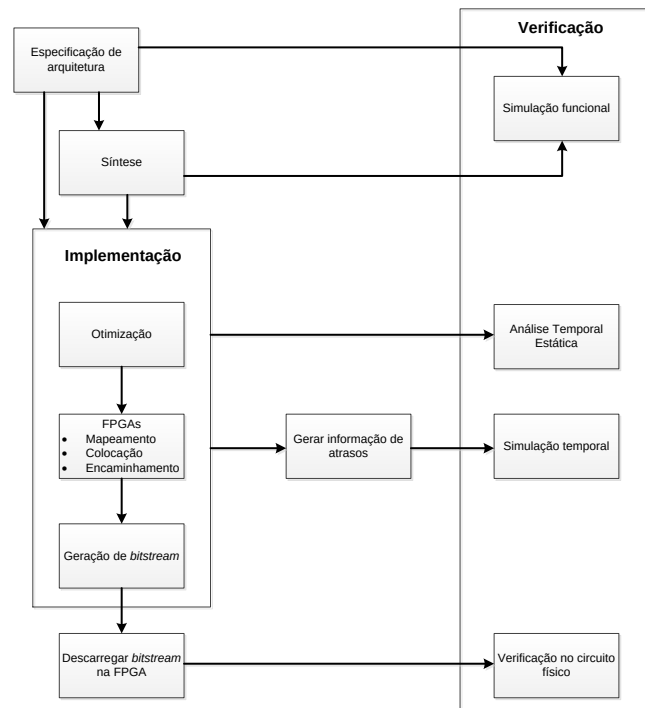


Figura 2.8: Fluxo típico de projeto para FPGA (alto nível) (segundo [4])

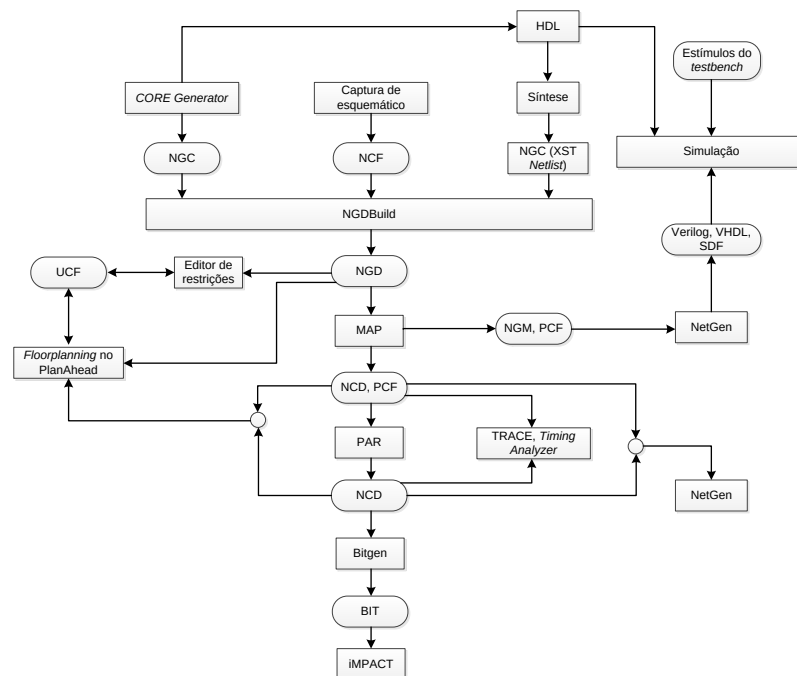


Figura 2.9: Fluxo típico de projeto para FPGA (detalhado) (segundo [4])

Na figura 2.9, ilustra-se um diagrama do fluxo típico de projeto para FPGA detalhado. A especificação do projeto da arquitetura de hardware pode ser feita a partir de uma descrição textual de estrutura e/ou comportamento, uma linguagem de descrição de hardware (HDL), ou a partir da captura de um esquemático que descreve a estrutura do circuito a partir de elementos lógicos. A ferramenta da Xilinx designada por *CORE Generator* disponibiliza módulos pré-construídos parametrizáveis e otimizados para FPGAs da Xilinx, que podem ser incluídos no projeto. A livreria da ferramenta inclui módulos para implementar FIFOs (First In First Out), instanciar BRAMs, etc. Durante o processo de síntese são construídas as *netlists* associadas ao HDL desenvolvido e ao HDL que descreve os módulos especificados pelo *CORE Generator*. Se o projeto incluir algum esquemático, é incluído o respetivo ficheiro NCF (*Netlist Constraints File*) que contém as restrições lógicas associadas ao esquemático. A ferramenta NGDBuild recebe posteriormente os ficheiros NGC, NCF e UCF (*User Constraints File*), sendo que o último contém restrições temporais e de *layout* que afetam como é que a lógica especificada no projeto é implementada na FPGA alvo. NGDBuild lê as *netlists* e as restrições associadas e produz um ficheiro no formato NGD (*Native Generic Description*) que contém a descrição lógica do projeto em termos de elementos lógicos (*flip-flops*, multiplexadores, portas lógicas, ...) e primitivas de baixo nível.

A fase de implementação inicia-se com o mapeamento da lógica do projeto para uma FPGA específica e é concluída quando o projeto a nível físico é encaminhado com sucesso e o respetivo ficheiro de *bitstream* gerado. Realizam-se então três processos distintos e pela seguinte sequência: mapeamento, colocação e encaminhamento e a geração do ficheiro de *bitstream*.

O mapeamento é realizado pela ferramenta MAP que recebe um ficheiro no formato NGD e ficheiros NMC (*Macro Library File*), caso sejam instanciadas *macros* no ficheiro NGD, que contém a descrição física de *hard-macros*, e mapeia a lógica por este descrita em componentes da FPGA alvo (células lógicas, células de E/S e outros). É produzido um ficheiro NCD (*Native Circuit Description*) cujo conteúdo é uma representação física do projeto mapeado em componentes da FPGA alvo, um ficheiro PCF (*Physical Constraints File*) que contém restrições fornecidas na fase de especificação da arquitetura de projeto exprimidas em termos dos elementos físicos mapeados e um ficheiro NGM que contém a informação relativa ao mapeamento físico do projeto produzida por MAP. Este último é usado pela ferramenta Netgen para gerar o ficheiro SDF para as simulações temporais.

A colocação e encaminhamento são realizados pela ferramenta PAR (*Place and Route*). A ferramenta recebe um ficheiro NCD mapeado e PCF, aloca (coloca) componentes físicos ao mapeamento do ficheiro NCD e estabelece as interligações (encaminha) entre esses componentes em função das restrições físicas do ficheiro PCF. O ficheiro de saída é também do formato NCD colocado e encaminhado. A fase de colocação depende fatores como o comprimento das ligações e dos recursos disponíveis para encaminhamento. Já na fase de encaminhamento, o encaminhador executa um procedimento iterativo e convergente de forma a que as restrições temporais sejam cumpridas.

Concluída a fase de colocação e encaminhamento, a ferramenta TRACE (*Timing Reporter And Circuit Evaluator*) permite realizar a análise temporal estática do projeto especificado. A

ferramenta recebe o ficheiro NCD mapeado, colocado ou colocado e encaminhado juntamente com a opção de incluir o ficheiro PCF para indicar restrições físicas impostas pelo utilizador, como por exemplo restrições temporais para pinos específicos da FPGA, atraso máximo permitido numa determinada interligação e outros. São produzidos dois ficheiros cujo conteúdo é um relatório da análise temporal efetuada. O primeiro é um ficheiro TWR que é o relatório textual produzido por omissão, e o segundo um ficheiro TWX que é o relatório em formato XML (*eXtensible Markup Language*) que pode ser exportado e visualizado pela ferramenta *Timing Analyzer* de interface gráfica.

Por fim, a ferramenta Bitgen permite gerar o ficheiro de *bitstream*. Bitgen recebe um ficheiro NCD já encaminhado e produz um ficheiro binário de extensão *.bit* que contém a configuração da lógica interna da FPGA e das suas interligações. O ficheiro pode ser descarregado na memória SRAM da FPGA a partir do iMPACT, a ferramenta da Xilinx para configurar FPGAs.

A etapa de *floorplanning* não é abordada neste trabalho pelo que não é aqui descrita.

Existe ainda outra ferramenta particularmente útil, designada por XFLOW, que permite automatizar os processos de síntese, implementação e simulação do fluxo de projeto através de *scripts*. Estes *scripts* indicam quais as fases do fluxo de projeto que devem ser realizadas e são completamente customizáveis. O XFLOW tem a vantagem de não ser necessário ter muito conhecimento sobre a utilização de cada ferramenta em específico, e de se poder percorrer todas as fases do fluxo de projeto com uma única invocação à ferramenta, em vez da invocação direta e tradicional a XST, NGDBuild, PAR, TRACE (opcional) e Bitgen.

O fluxo de projeto associado a um sistema embebido, de hardware e um ou mais processadores, onde há a necessidade de incorporar alguns detalhes adicionais ao fluxo de projeto base, é descrito na subsecção 5.6.1 uma vez que se relaciona diretamente com o desenvolvimento do trabalho.

## 2.3 Aplicações de autómatos celulares em FPGA

Nesta secção, apresentam-se algumas implementações existentes de modelos de ACs em FPGA para aplicações específicas.

### 2.3.1 Autómato celular de Greenberg-Hastings

Vlassopoulos *et al.* [36], descrevem uma arquitetura de hardware para FPGA de um AC bidimensional probabilístico para o modelo de Greenberg-Hastings, que simula a propagação de ondas em sistemas de reação-difusão, para estudar o seu comportamento. Estes salientam que a principal motivação para usar sistemas baseados em FPGA, deve-se ao facto da sua arquitetura regular e bidimensional providenciar um meio ideal para o mapeamento da estrutura de um AC. Além disso, o paralelismo inerente da arquitetura de uma FPGA e do modelo de um AC, faz com que joguei a favor no que diz respeito ao desempenho face a aplicações de software. O ganho obtido em desempenho é viável para compensar a necessidade de realizar um elevado número de simulações e iterações por simulação, para obter um conjunto razoável de dados estatísticos do

comportamento do modelo, e também pela possibilidade de introduzir mais estados por célula, aumentando a complexidade mas tornando o modelo mais robusto.

Cada célula do AC pode estar em um estado de  $\{0, \dots, M\}$ , onde o estado 0 é designado por estado neutro, o estado M por estado de excitação e os restantes por estados refratários. A cada instante  $t$ , o estado de uma célula evolui em função do seu próprio estado e do estado das células vizinhas, em que a vizinhança pode ser do tipo de von Neumann ou Moore. As regras que descrevem a evolução do AC de Greenberg-Hastings são as seguintes:

- Uma célula no estado neutro entra num estado de excitação, com uma probabilidade  $p$ , se pelo menos uma das células vizinhas também está no estado de excitação.
- Uma célula num estado refratário, isto é,  $\{1, \dots, M - 1\}$ , decremente o seu estado sucessivamente até atingir o estado neutro, independentemente do estado das células vizinhas.
- Uma célula no estado neutro mantém-se nesse mesmo estado, se todas as células vizinhas também se encontrarem no estado neutro ou num refratário.

O objetivo é semelhante ao do trabalho proposto nesta dissertação, uma vez que também se pretende implementar uma ferramenta em software, que permita interagir com o hardware de forma a atribuir as condições iniciais, iniciar uma simulação, ler de volta dos resultados para análise e representar graficamente o estado do AC. Da mesma forma, os parâmetros do AC são configuráveis com um certo grau de liberdade, mas a regra de evolução do estado das células é fixa, pois aqui trata-se de uma aplicação específica. O número de iterações é também especificado em software e a simulação termina assim que esse número seja atingido, ou quando todas as células se encontram no estado neutro, que se refere à condição absoluta do fim da simulação. A interface entre software e o hardware é feita por meio de uma memória que armazena os parâmetros de simulação, para permitir ler, inicializar e ajustar o estado de cada célula e modificar a probabilidade  $p$ . Desta forma, a FPGA é vista como um periférico cuja configuração e os estados do AC são mapeados em memória e acessíveis por endereçamento direto.

A arquitetura proposta consiste no agrupamento das células do AC em blocos distintos, em que cada um contém um subconjunto de células em torno de uma BRAM, de onde são lidos e armazenados os estados das células do bloco. Os blocos de subconjuntos de células encontram-se ligados entre si para permitir transferir os estados das células entre blocos. Por sua vez, os blocos encontram-se organizados em grupos cujo processamento interno é realizado em paralelo, mas, entre grupos, o processamento é sequencial, uma vez que é preciso comunicar a cada grupo o resultado do processamento do grupo anterior, isto é, os estados das células de fronteira do grupo. Salienta-se que esta abordagem explora razoavelmente bem a arquitetura da FPGA para o mapeamento da estrutura regular bidimensional de um AC e, além disso, como os blocos de memória estão regularmente distribuídos na FPGA entre lógica reconfigurável, a organização das células em torno destes é viável e permite otimizar a utilização de recursos na FPGA. A arquitetura de cada célula consiste em lógica combinacional que implementa as regras para o cálculo do estado seguinte e implementa um LFSR (*Linear Feedback Shift-Register*), como gerador de números aleatórios,

para determinar, em função de  $p$ , se a célula deve entrar no estado de excitação, caso esta e as células vizinhas estejam no estado neutro.

O funcionamento do sistema é descrito como sendo sequencial, à medida que se vai iterando ao longo de todos os grupos. Dentro de um grupo o estado de cada célula é determinado em paralelo, o resultado do processamento (células que não se encontram na fronteira do grupo) é armazenado na BRAM do bloco associado, e o estado das células da fronteira do grupo são armazenados nas memórias de fronteira. As memórias que guardam os estados das células de fronteira de cada grupo armazenam os estados entre dois instantes temporais consecutivos. Desta forma, uma iteração consiste em dois passos distintos. Na iteração de instante  $t$ , o primeiro passo consiste em determinar o estados das células de fronteira e armazená-los na memória de fronteira e o segundo passo consiste em ler o estado armazenado das células de fronteira, do instante  $t - 1$ , e determinar o estado seguinte das células restantes e armazená-los nas BRAMs de bloco. O processo repete-se para  $t + 1$ , os bancos de memória armazenam o estados das células de fronteira do instante  $t$  e  $t + 1$ , respetivamente, são trocados de forma a que cada grupo leia a informação de estado atualizada em  $t + 1$  e assim sucessivamente.

Por fim, o controlo do sistema é assegurado por um módulo de controlo global, que gera endereços de memória e sinais de controlo para determinar, por exemplo, as condições de fim de simulação e até dar a possibilidade de realizar apenas uma iteração do AC.

A tabela 2.2, mostra os resultados obtidos, apresentados no artigo, no relatório de síntese para um AC de 512x512 células. A FPGA utilizada foi uma Virtex 4 da Xilinx (XC4VLX100-10FF1513) que disponibiliza 49152 *slices* para lógica reconfigurável e sequencial e 240 BRAMs.

| Módulo   | <i>Slices</i>            | BRAMs |
|----------|--------------------------|-------|
| Célula   | 34 ( $< 1\%$ )           | 0     |
| Bloco    | 234 ( $< 1\%$ )          | 1     |
| Grupo    | 12743 ( $\approx 25\%$ ) | 64    |
| Ambiente | 13143 ( $\approx 26\%$ ) | 136   |

Tabela 2.2: Resultados de síntese para um AC de 512x512 células de Greenberg-Hastings

Na tabela o ambiente corresponde a todos os grupos de blocos que constituem a plataforma de simulação do AC. No que diz respeito a simulação em software, estas foram realizadas para  $10^4$  iterações para um AC de 256x256 células numa máquina com um processador Intel(R) Core(TM) 2 Quad CPU (*Central Processing Unit*), Q9950 com uma frequência de operação de 2.83 Ghz. Dado que cada iteração é processada em 512 ciclos de relógio e a FPGA opera a uma frequência de 100 Mhz, então uma iteração é completada em  $5120\eta s \approx 5.2\mu s$ . Assim, o tempo necessário para realizar uma simulação de  $10^4$  iterações é dado por  $5.2 \cdot 10^4 \mu s = 52ms$ . O ganho em velocidade de execução obtido (*speed-up*) foi aproximadamente de 1650, uma vez que o tempo médio total de execução no CPU foi de  $86.5 \cdot 10^3 ms$ .

### 2.3.2 Autômato celular para *lattice gases*

Shaw *et al.* [37], descrevem a implementação de uma arquitetura de hardware baseada em FPGA de um AC bidimensional para um modelo de *lattice gases*. Embora o trabalho relatado descreva uma implementação numa plataforma de hardware já ultrapassada (1995), que consiste num conjunto de FPGAs de baixa capacidade ligadas entre si para elevar o poder computacional limitado pela tecnologia de FPGA da mesma altura, as regras que regem a evolução do AC implementado para o modelo de *lattice gases* são particularmente interessantes. Esta plataforma ou máquina designa-se por SPACE (*Scalable Parallel Architecture for Concurrency Experiments*) e é um computador baseado em *chips* CAL1024, desenvolvidos pela Algotronix no início dos anos 90, que se tratam de FPGAs.

Os *lattice gases* são modelos que permitem emular a dinâmica de fluídos ou gases através de um sistema digital. Estes podem ser caracterizados através da interação de elementos discretos, isto é, colisões de partículas, que pode ser descrita por regras locais a cada célula de um AC.

Para emular o comportamento do movimento de partículas, os autores consideram apenas quatro direções possíveis, em que se estabelece assim uma vizinhança do tipo de von Neumann. Considera-se também que as partículas movem-se com velocidade unitária e aplica-se a lei de conservação de momento durante uma colisão. Desta forma, para cada célula do AC, representando o momento de entrada e saída por  $(v_x, v_y)$  tem-se 9 combinações distintas possíveis  $(-1, -1), (-1, 0) \dots (1, 1)$ . O par  $(v_x, v_y)$  refere-se à soma dos momentos de um conjunto de partículas, que se deslocam na direção de  $x$  e  $y$ , e não apenas de uma. Assim sendo, quando a soma dos momentos é zero (considerando a massa das partículas como unitária também), e há um choque frontal de partículas, a sua direção de deslocamento é alterada em 90 graus em relação à direção de origem. Quando o momento total é diferente de zero em qualquer uma das direções  $x$  e  $y$ , em caso de colisão as partículas passam uma pela outra e mantêm o seu rumo.

Cada célula do AC tem quatro bits de estado e em que cada iteração é capturado o fluxo de partículas provenientes das quatro direções distintas (Norte, Sul, Este e Oeste). A lógica de colisão implementada em cada célula, determina quais as saídas da mesma que devem transportar uma partícula para as células vizinhas na iteração seguinte. Considere-se o seguinte: quando uma partícula se desloca na direção negativa de  $x$ , ou seja, quando viaja de Oeste para Este, esta mantém o seu rumo e da célula sai uma partícula para a célula vizinha a Oeste. Além disso, se houver numa colisão entre duas partículas que se deslocam em sentidos contrários na direção de  $y$  (Norte, Sul), haverá também uma partícula que sai da célula em direção à célula Oeste vizinha, devido à alteração na direção em 90 graus. Desta forma, define-se pela expressão lógica  $O_{out} = c \oplus E_{in}$  quando é que uma partícula sai de uma célula no sentido Oeste. Na expressão cada fator é de 1 bit e  $O_{out}$  indica que uma partícula viaja ou não para o sentido Oeste,  $c$  indica se há ou não uma colisão entre duas partículas na direção de  $y$  ou  $x$  e  $E_{in}$  indica se uma partícula chega ou não à célula deslocando-se no sentido Este-Oeste. A expressão generaliza-se para as restantes três saídas de uma célula:

$$\begin{aligned}
N_{out} &= c \oplus S_{in} \\
S_{out} &= c \oplus N_{in} \\
E_{out} &= c \oplus O_{in}
\end{aligned}$$

Os sinais  $c_{EO}$  e  $c_{NS}$ , que são colisões entre duas partículas que se deslocam na direção de  $x$  (Este, Oeste) e na direção de  $y$  (Norte, Sul), em sentidos contrários, respetivamente, definem-se pelas seguintes expressões lógicas:

$$\begin{aligned}
c_{EO} &= (E_{in} \bullet \neg S_{in}) \bullet (O_{in} \bullet \neg N_{in}) \\
c_{NS} &= (\neg E_{in} \bullet S_{in}) \bullet (\neg O_{in} \bullet N_{in})
\end{aligned}$$

Uma colisão  $c$  qualquer entre duas partículas que se deslocam em sentidos opostos ocorre quando  $c_{EO} = 1$  ou  $c_{NS} = 1$ . Assim, tem-se a seguinte expressão lógica:

$$c = c_{EO} + c_{NS}$$

A experiência realizada baseia-se no estudo da evolução de *lattice gases*, usando como aplicação específica a propagação de ondas sonoras de frequência de áudio. As regras e as expressões atrás apresentadas descrevem na sua totalidade o comportamento das células do AC bidimensional para esta experiência. Contudo, são introduzidos circuitos e lógica adicional para controlar a operação do sistema perante condições específicas. Aqui não é descrita o restante da operação, circuitos e lógica adicional necessária para a experiência realizada, uma vez que no âmbito desta dissertação não tem qualquer impacto. Ainda assim, apresentam-se os resultados obtidos na tabela 2.3, em termos do número de atualizações de estado das células por segundo, em comparação com outras plataformas de arquitetura distinta:

| Máquina de simulação                                    | Atualizações de estado por segundo |
|---------------------------------------------------------|------------------------------------|
| Cray X-MP/146                                           | $5 \cdot 10^8$                     |
| Connection Machine 2                                    | $7,5 \cdot 10^8$                   |
| Dois módulos CAM-8 ( <i>Cellular Automata Machine</i> ) | $6,6 \cdot 10^7$                   |
| Um módulo SPACE                                         | $3 \cdot 10^7$                     |
| Um módulo SPACE com memória externa                     | $6 \cdot 10^8$ (estimado)          |

Tabela 2.3: Número de atualizações de estado das células por segundo para um módulo SPACE em comparação com outras plataformas.

A última entrada da tabela refere-se a uma expansão do módulo SPACE, por instalação de memórias externas para leitura e escrita de dados no sistema para acelerar as simulações.

## 2.4 Máquinas de Autômatos Celulares (CAM)

*Cellular Automata Machines* (CAM), ou Máquinas de Autômatos Celulares, é uma arquitetura de computador otimizada para a simulação de ACs a grande escala proposta por Norman Margo-

lus e Tommaso Toffoli nos anos 80. A proposta consiste na subdivisão de uma matriz de células em módulos distintos, que contêm um subconjunto de células da matriz e um ou mais processadores. Em cada módulo, o estado seguinte de cada célula é determinado sequencialmente, por atribuição de *slots* temporais a um único processador, ou em paralelo para mais que um processador. Daqui nasceu o CAM-6, um módulo de hardware dedicado que implementa esta arquitetura comercializado como uma expansão compatível para PC.

## 2.5 Aplicação de autómatos celulares em software

Nesta secção, apresentam-se algumas aplicações de software existentes para simulação de ACs.

### 2.5.1 XLife

O Xlife [38], é um projeto *open-source* iniciado em 1989 e em constante atualização até à data (Vladimir Lidovski, 2011-2012). É uma aplicação de software de interface gráfica desenvolvido para o *X Window System*, ou X11, um gestor de janelas para Unix, que permite simular, essencialmente, modelos de ACs baseados no Jogo da Vida de John Conway, mas também tem suporte para outros tipos de AC (bidimensionais), como por exemplo, o modelo designado por *Iterated Prisoner's Dilemma*, que é um AC que permite estudar a evolução de comportamento cooperativo, *Brian's Brain* que é semelhante ao jogo da vida e *WireWorld* que implementa o comportamento de elementos lógicos digitais. A aplicação suporta um diverso conjunto de funcionalidades entre as quais se destacam as seguintes:

- Interface gráfica para navegação fácil; indicação do número de células, coordenadas, cores para identificação de estado, deslocamento da matriz.
- Especificação do número máximo de iterações a simular; simulação passo-a-passo.
- Até 64 bits de estado por célula.
- Retenção automática do histórico de estado do AC.
- Inicialização automática de estado através de *macros* especificadas num formato estruturado.
- Manipulação e transformação de padrões de estado.
- Configuração do universo (dimensões e fronteiras).
- *Benchmarking*.
- Diferentes velocidades de simulação.

Na figura 2.10, ilustra-se a interface gráfica da aplicação. É apresentado um AC bidimensional de 64x64 células com condição fronteira toroidal (contínua) com dois estado por célula (Jogo da Vida).



Figura 2.10: Interface gráfica do XLife

### 2.5.2 Golly

Golly [39] é um projeto *open-source* e uma aplicação de software multiplataforma para simulação do Jogo da Vida de John Conway e outros ACs, desenvolvida por Andrew Trevorrow e Tom Rokicki. A aplicação tem uma interface gráfica que permite ao utilizador interagir de forma rápida e intuitiva com as suas funcionalidades das quais se destacam as seguintes:

- Suporte até 256 estados por célula.
- Configuração do universo (dimensões e fronteiras).
- Suporte de múltiplos algoritmos para a evolução de um AC.
- Suporte para vários tipos de AC, como por exemplo, o *Wireworld*, ACs unidimensionais de Wolfram, o AC de 29 estados por célula de John von Neumann, Gerações e entre outros.
- Suporte para leitura de formatos de imagem BMP, PNG (*Portable Network Graphics*), GIF (*Graphics Interchange Format*) e TIFF (*Tagged Image File Format*) para inicializar um AC.
- Suporte para simulação passo-a-passo e especificação do número de iterações pretendidas.
- Suporte para diferentes velocidades de simulação.
- Permite gravar e reproduzir simulações.

- Aceita *scripts* em Python e Perl para especificar regras e padrões de inicialização.
- Possibilidade de extrair ficheiros compactados diretamente e carregar o respetivo conteúdo para a aplicação (padrões de inicialização, *scripts* e regras).
- Possibilidade de ler ficheiros cujo formato é proveniente de outras aplicações de simulação de ACs, como por exemplo o MCell, e ficheiros RLE (*Run Length Encoded Bitmap*).

Na figura 2.11, ilustra-se a interface gráfica da aplicação. No menu da esquerda é possível seleccionar qualquer padrão de inicialização já existente do AC respetivo, guardados no diretório de trabalho da aplicação, e carregá-lo para a janela da direita de edição e visualização do AC.

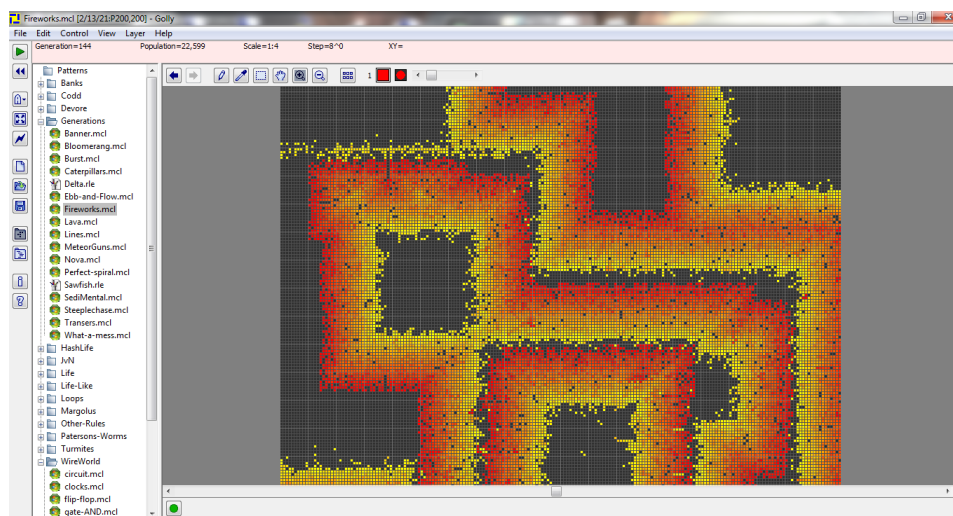


Figura 2.11: Interface gráfica do Golly

### 2.5.3 MCell

O MCell [40], ou *Mirek's Celebration*, é uma aplicação de software desenvolvida por Mirek Wojtowics para Windows e publicada em Abril de 1999. A aplicação suporta uma vasta variedade de ACs, unidimensionais e bidimensionais, e tem também uma interface gráfica para o utilizador interagir com as suas funcionalidades. Destacam-se as seguintes:

- Suporte para ACs unidimensionais e bidimensionais e até 256 estados por célula.
- Suporte de 14 famílias distintas de ACs, entre as quais o Jogo da Vida, Gerações, ACs unidimensionais de Wolfram, *General Binary*, *Neumann binary*, Regras Especiais e entre outras.
- Biblioteca vasta de regras predefinidas para cada família.
- Suporte para diferentes tipos de vizinhanças (Moore 8 e 9, von Neumann 4 e 5, Hexagonal e entre outros).

- Interface gráfica intuitiva e simples de utilizar para manipulação de ACs.
- Suporte para vários formatos de ficheiros para leitura e escrita (versões anteriores do MCell, RLE e outras aplicações de simulação de ACs).
- Suporte para customização de regras para ACs a partir de ficheiros DLL (*Dynamic-Link Library*).
- Explorador de regras de ACs, a partir de um menu dedicado, da biblioteca armazenada no diretório de trabalho da aplicação.
- Análise de dados estatísticos e outros.
- Manipulação de padrões de um AC e edição do seu estado enquanto uma simulação decorre.

Na figura 2.12, ilustra-se a interface gráfica da aplicação.

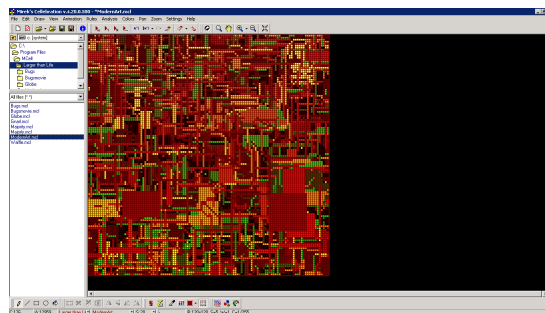


Figura 2.12: Interface gráfica do MCell



## Capítulo 3

# Especificação dos Sistemas de Controlo de Software e Hardware

Neste capítulo introduzem-se aspetos relativos à funcionalidade que os Sistema de Controlo de Software (SCS) e o Sistema de Controlo de Hardware (SCH) proporcionam. O SCS representa o *framework*, uma aplicação de software implementada em Java, e posiciona-se como o *front-end* para o utilizador, por meio de uma interface gráfica, que permite especificar parâmetros de *templates* de módulos de hardware descritos em Verilog HDL, isto é, parâmetros que definem a especificação da arquitetura de hardware, controlar o modo de operação do SCH especificando parâmetros de controlo adicionais e enviar e receber dados processados do SCH para posterior análise. O SCH representa uma plataforma de desenvolvimento de hardware, uma FPGA Spartan6 da Xilinx, cujos circuitos são descritos pelos módulos de hardware pré-construídos parametrizados à medida das necessidades do utilizador.

Na secção 3.1 é feita uma breve introdução ao *front-end* e ao hardware, ilustrando a sua interação com o utilizador com apoio de um diagrama. Para ilustrar, mais detalhadamente, a interação entre o utilizador, o *front-end* e o hardware, recorre-se a um diagrama de casos de utilização para descrever as funcionalidades e limitações dos sistemas, na secção 3.2. Na secção 3.3, com recurso a um diagrama de interação, ilustra-se a sequência de passos necessários a realizar ou o cenário típico, de forma intuitiva, desde a especificação de parâmetros dos módulos de hardware até ao arranque de simulações na plataforma de hardware e extração dos respetivos resultados. É feita também, na secção 3.4, uma divisão do SCS e do SCH com um diagrama de blocos, apresentando uma vista de alto nível de ambos os sistemas e dos seus módulos de organização de arquitetura constituintes. Por fim, em jeito de conclusão, na secção 3.5, é feito um resumo de todas as funcionalidades e limitações inerentes ao SCH e SCS. A especificação da arquitetura do SCH é feita com detalhe no capítulo 4 e da arquitetura do SCS é feita no capítulo 5.

### 3.1 Visão geral

Na figura 3.1, é apresentada uma vista geral do funcionamento dos sistemas de controlo de hardware e software, é ilustrada a sua interação mútua e a interação do conjunto formado pelos dois sistemas com o utilizador externo. O utilizador fornece uma especificação para a parametrização arquitetura dos módulos de hardware que permite caracterizar esses módulos e modelá-los, constituindo assim uma arquitetura fechada de hardware para as suas necessidades. A especificação envolve definir valores de parâmetros como as dimensões da matriz do autómato celular (AC) e, conseqüentemente o tipo do AC, o número de bits por célula para a codificação de estado, a condição fronteira, o tipo de vizinhança a suportar e a arquitetura de cada célula, isto é, a regra de transição de estado. Estes parâmetros caracterizam o modelo do AC na sua totalidade e o seu modo de operação.

Fornecida uma nova especificação, são iniciados os procedimentos necessários para obter um ficheiro de *bitstream* válido à saída, para ser descarregado na FPGA. O processo que envolve os diversos passos designa-se por *FPGA Design Flow* que, simplificando, se encarrega de pegar nos ficheiros que descrevem os circuitos de hardware dos diversos módulos que constituem a sua arquitetura e mapear essas descrições numa área física com as interligações necessárias entre os módulos, respeitando todas as restrições impostas, não apenas temporais mas também arquiteturais.

Do lado da aplicação de software, existe uma representação visual do AC num mapa de cores, que permite ao utilizador interagir com ela impondo as condições que pretender de acordo com a especificação previamente fornecida, isto é, os estados das células do AC e também visualizar os resultados obtidos pós-processamento, depois de  $n$  iterações. Impostas as condições, a matriz pode ser enviada para a FPGA, em que o meio de comunicação é do tipo série (RS-232), após uma fase de processamento do conteúdo das células em que consiste organizar os bits de estado em múltiplos bytes para a transferência. Por sua vez, do lado da FPGA, assim que a informação de estado das células é recebida, é realizado, também, processamento ao nível dos bytes recebidos, agrupando-os e armazenando-os. Os dados armazenados são lidos e processados pelo bloco de hardware, que contém a lógica associada à matriz do AC e respetivo controlo, e o resultado final é armazenado novamente para posterior leitura. Aquando da leitura do resultado, é realizado o processo inverso na fase de processamento, em que a informação de estado das células é agrupada em bytes e transferida para a aplicação de software. Por fim, os dados recebidos são novamente processados de forma a serem representáveis graficamente ao utilizador, isto é, é feita uma separação dos bits de estado das células e estes são representados de acordo com um mapa de cor-estado.

### 3.2 Casos de utilização

Nesta secção, apresenta-se o diagrama de casos de utilização, na figura 3.2, para ilustrar as funcionalidades de interação do utilizador com a aplicação de software e desta com a plataforma de hardware. Para cada caso de utilização, é feita uma breve descrição do mesmo, são indicados

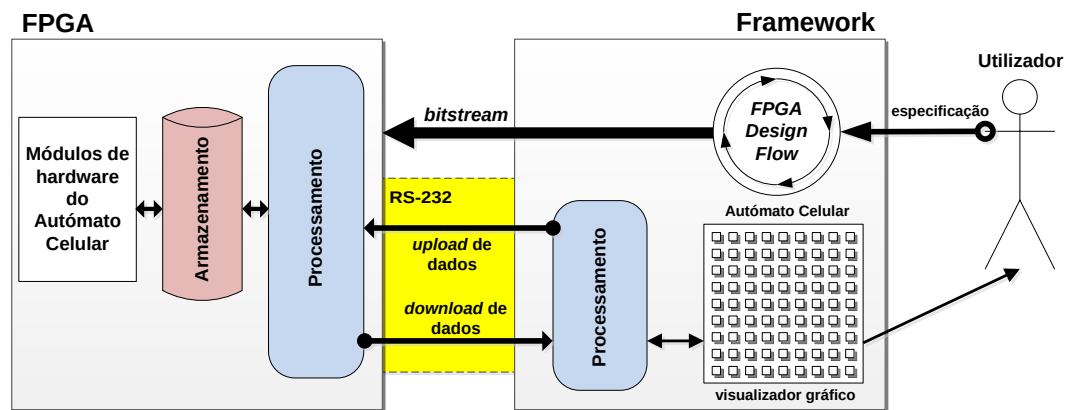


Figura 3.1: Vista geral dos sistemas e interação

os casos de utilização precedentes e são indicadas as funcionalidades e limitações de hardware ou software que serão posteriormente detalhadas nos capítulos respetivos: para hardware no capítulo 4 e para software no capítulo 5. Os casos de utilização estão divididos em cinco categorias distintas: especificação, implementação, configuração, comunicação, execução, e visualização.

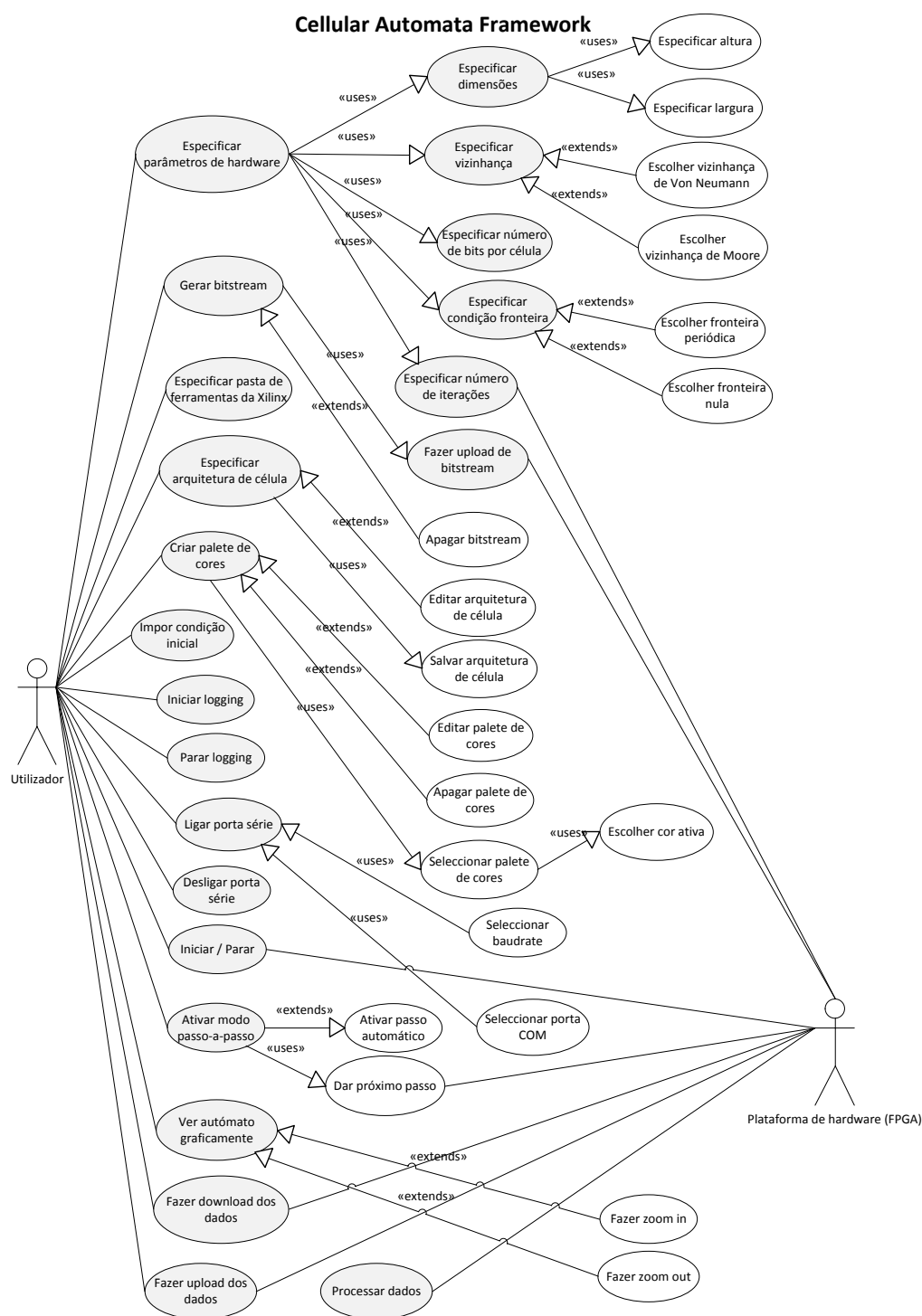


Figura 3.2: Diagrama de casos de utilização

A categoria de casos de utilização de especificação descritos na tabela 3.1 refere-se à especificação da arquitetura de hardware no que diz respeito aos parâmetros de configuração e funcionamento do AC: dimensões, tipo de vizinhança, condição fronteira, número de bits por célula, número de iterações a realizar e a arquitetura celular.

| Caso de utilização                 | Código | Precedência | Descrição                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|------------------------------------|--------|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Especificar parâmetros de hardware | CU-01  | -           | Permite ao utilizador parametrizar os módulos de hardware pré-construídos consoante as suas necessidades. Estes parâmetros definem assim a arquitetura do SCH. Os parâmetros de seleção obrigatórios são as dimensões, a altura e a largura da matriz de células do autómato celular, o tipo de vizinhança a considerar que pode ser do tipo von Neumann ou Moore, o número de bits por célula para a codificação de estado e a condição fronteira, que pode ser periódica (contínua) ou nula. Após o preenchimento e validação destes parâmetros, o utilizador pode gerar o ficheiro de <i>bitstream</i> para especificação do sistema fornecida. |
| Especificar dimensões              | CU-02  | -           | O utilizador deve especificar as dimensões desejadas para a matriz do autómato celular, a sua altura (número de linhas) e a sua largura (número de colunas), de tal forma a serem corretamente validadas. Uma matriz de altura unitária representa um autómato unidimensional. Por questões arquiteturais detalhadas no capítulo 3, a largura do autómato deve ser sempre um múltiplo de oito e, naturalmente, diferente de zero.                                                                                                                                                                                                                  |

| <b>Caso de utilização</b>             | <b>Código</b> | <b>Precedência</b> | <b>Descrição</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|---------------------------------------|---------------|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Especificar vizinhança                | CU-03         | CU-02              | O utilizador deve especificar um de dois tipos de vizinhança a considerar pela matriz do autómato celular, vizinhança de von Neumann ou de Moore, de tal forma a serem corretamente validadas. A seleção do tipo de vizinhança depende do tipo de autómato celular, pelo que um autómato celular unidimensional não pode ser caracterizado por uma vizinhança do tipo de Moore. Embora num autómato celular unidimensional se considere, geralmente, apenas vizinhança de duas células, por questões arquiteturais explicadas no capítulo 4, este é caracterizado por uma vizinhança de Moore. Um autómato bidimensional, nesta arquitetura, pode ser caracterizado por uma vizinhança de Moore ou de von Neumann. |
| Especificar número de bits por célula | CU-04         | -                  | O utilizador deve especificar o número de bits por célula para a codificação de estado, de tal forma a ser corretamente validado.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Especificar condição fronteira        | CU-05         | -                  | O utilizador deve selecionar o tipo de condição fronteira que deseja para a sua especificação de entre duas: fronteira nula ou periódica (contínua).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| Especificar o número de iterações     | CU-06         | CU-19              | O utilizador deve especificar o número de iterações a realizar até um máximo de 4.294.967.295 (32 bits). O valor máximo referido não é imposto por uma limitação arquitetural e é facilmente expansível. Para este valor ser definido em hardware, a conexão por porta série deve ser efetuada previamente.                                                                                                                                                                                                                                                                                                                                                                                                        |

| <b>Caso de utilização</b>          | <b>Código</b> | <b>Precedência</b> | <b>Descrição</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|------------------------------------|---------------|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Especificar arquitetura de células | CU-11         | -                  | Permite ao utilizador descrever a arquitetura das células do autómato celular a partir operações lógicas e aritméticas entre os bits de estado das células da vizinhança que, no fundo, se traduz na regra ou função de transição de estado do AC. A especificação da arquitetura das células é feita textualmente com a linguagem de descrição de hardware Verilog HDL, pois é integrada diretamente nos ficheiros que descrevem o circuito da arquitetura das mesmas e, por uma questão de simplicidade, as referências para a vizinhança de uma dada célula são representadas por uma linguagem própria, e posteriormente traduzidas para o sinal de referência no módulo de hardware. No capítulo 5 é descrita a sintaxe desta linguagem, os graus de liberdade para a referência de células vizinhas, limitações e o grau de deteção de erros sintáticos. Após a especificação da arquitetura de células, o utilizador pode salvá-la. |
| Editar arquitetura de células      | CU-12         | CU-13              | A partir dos conjunto das arquiteturas de células existentes, o utilizador pode alterá-la, adaptá-la às suas necessidades e salvá-la.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| Salvar arquitetura de células      | CU-13         | CU-02, CU-11       | Após especificar ou editar a arquitetura de células, o utilizador pode salvá-la. Ao salvar a arquitetura de células descrita, esta pode ser aplicada a uma dada linha do AC, ou a partir da linha da matriz de referência, isto é, a linha localizada no topo da matriz, aplicá-la às restantes. Esta possibilidade representa uma funcionalidade da arquitetura de hardware justificada e especificada no capítulo 4. Para salvar uma arquitetura de células, deverá existir uma especificação das dimensões do AC.                                                                                                                                                                                                                                                                                                                                                                                                                       |

Tabela 3.1: Descrição de casos de utilização de especificação

A segunda categoria refere-se aos casos de utilização de implementação, descritos na tabela 3.2, para manipular ficheiros de configuração e programação da FPGA (*bitstreams*). A partir de uma especificação de arquitetura de hardware, a aplicação permite gerar o ficheiro de *bitstream* associado. Os ficheiros de *bitstream* existentes podem ser descarregados na FPGA ou apagados.

| Caso de utilização                      | Código | Precedência         | Descrição                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-----------------------------------------|--------|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Gerar <i>bitstream</i>                  | CU-07  | CU-02, CU-03, CU-04 | Quando existe uma especificação da arquitetura do hardware validada, o utilizador pode gerar o ficheiro de <i>bitstream</i> correspondente e, em caso de sucesso, descarregá-lo posteriormente para a plataforma de hardware. A validação da especificação refere-se às dimensões da matriz do autómato celular, o tipo de vizinhança que o caracteriza e o número de bits por célula para a codificação de estado. Embora o caso de utilização CU-05 constitua um parâmetro de especificação da arquitetura de hardware, é, de facto, uma opção do utilizador seleccionar a condição fronteira, pelo que se nada fizer, por omissão, é seleccionada uma condição fronteira do tipo periódica. A geração de <i>bitstream</i> é sempre baseada na última alteração feita aos ficheiros que constituem os módulos da arquitetura de hardware. Para gerar um ficheiro de <i>bitstream</i> , o utilizador deverá dar-lhe um nome à sua escolha e uma descrição. Caso já exista um ficheiro de <i>bitstream</i> com o nome especificado, o utilizador deverá fornecer outro. |
| Fazer <i>upload</i> de <i>bitstream</i> | CU-08  | CU-07               | Após a geração de um ficheiro de <i>bitstream</i> e caso já existam outros ficheiros de <i>bitstream</i> , o utilizador pode descarregar qualquer um deles para a plataforma de hardware. Os parâmetros de arquitetura de hardware associada ao <i>bitstream</i> seleccionado, são carregados automaticamente para o software como uma nova configuração pronta a ser usada para simulações.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

| Caso de utilização      | Código | Precedência | Descrição                                                                                                                                                                                              |
|-------------------------|--------|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Apagar <i>bitstream</i> | CU-09  | CU-07       | Permite ao utilizador eliminar um ficheiro de <i>bitstream</i> de especificação de arquitetura de hardware associada. Para o fazer, deverão existir ficheiros de <i>bitstream</i> gerados previamente. |

Tabela 3.2: Descrição de casos de utilização de implementação

A categoria de configuração refere-se aos casos de utilização de configuração, descritos na tabela 3.3, da aplicação de software no que diz respeito à manipulação de paletes de cor, indicação do caminho para as ferramentas da Xilinx e *logging*.

| Caso de utilização                         | Código | Precedência | Descrição                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------------------------------|--------|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Especificar pasta de ferramentas da Xilinx | CU-10  | -           | Consoante o sistema operativo em que o utilizador se encontra a executar o software, por omissão, a pasta de ferramentas da Xilinx é a pasta associada por omissão durante a instalação das mesmas. Caso o utilizador tenha as ferramentas instaladas noutra via, deverá especificá-lo, caso contrário durante o processo de geração ou descarga de um ficheiro de <i>bitstream</i> , será produzido um erro indicando a falta de presença das ferramentas.                                                                                                                                                                                                                                                            |
| Criar palette de cores                     | CU-14  | -           | Permite ao utilizador criar uma nova palette de cores para a representação gráfica dos estados do autómato celular. O utilizador deve especificar qual o número de cores, e, por sua vez, o número de estados a codificar com a palette de cores em questão. As cores associadas a cada estado são geradas aleatoriamente a partir da codificação em RGB. Quando a aplicação de software se inicia, é verificada a existência da palette de nome <i>default</i> , a palette por omissão, a carregar no arranque da aplicação uma vez que é necessário que exista sempre uma palette de cores ativa. Se a palette de cores com o nome referido não existir, esta é criada e carregada para a aplicação automaticamente. |

| <b>Caso de utilização</b>  | <b>Código</b> | <b>Precedência</b> | <b>Descrição</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|----------------------------|---------------|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Editar paleta de cores     | CU-15         | CU-14              | Permite ao utilizador editar o mapeamento de cor-estado da célula, de uma paleta de cores existente, a partir da seleção de cores da codificação em RGB.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| Apagar paleta de cores     | CU-16         | CU-14              | Permite ao utilizador apagar uma paleta de cores existente. Só não o poderá fazer se a paleta em questão é a única paleta existente ou se se encontrar ativa.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Selecionar paleta de cores | CU-17         | CU-14              | De entre um conjunto de paletas de cores existentes, o utilizador pode selecionar outra à sua escolha. Caso a paleta pretendida para a seleção ativa não possua um mapeamento de cor-estado de célula suficiente para representar a codificação de estados do autómato celular corrente, o pedido de seleção é rejeitado emitindo uma mensagem de erro em concordância. É possível também selecionar diferentes paletas de cor para linha distintas da matriz do AC.                                                                                                                                                                              |
| Escolher cor ativa         | CU-18         | CU-17              | Dada a paleta de cores ativa, o utilizador pode escolher qualquer uma das cores disponíveis, com a codificação de estado associada representada, para caracterizar o estado das células do seu autómato celular.                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Iniciar <i>logging</i>     | CU-20         | -                  | É permitido ao utilizador guardar continuamente um registo das mensagens de saída da aplicação ( <i>log</i> ), num ficheiro de texto simples, para posterior análise. Ao iniciar o registo de mensagens, o utilizador deve indicar o nome que pretende para esse mesmo registo. Caso um ficheiro com o nome idêntico já exista, é solicitado ao utilizador que confirme se deseja substituir o ficheiro existente, perdendo o conteúdo que nele existia previamente. Caso não seja possível criar o ficheiro para este registo, é gerado um erro, o ficheiro não é criado e, consequentemente, a gravação das mensagens de saída não é realizada. |

| Caso de utilização   | Código | Precedência | Descrição                                                                                                                    |
|----------------------|--------|-------------|------------------------------------------------------------------------------------------------------------------------------|
| Parar <i>logging</i> | CU-21  | CU-20       | Após o início da gravação contínua de um registo das mensagens de saída, o utilizador pode parar a mesma assim que entender. |

Tabela 3.3: Descrição de casos de utilização de configuração

Para estabelecer uma comunicação entre a aplicação de software e a plataforma de hardware, os casos de utilização de comunicação descrevem, na tabela 3.4, as funcionalidades necessárias para o efeito.

| Caso de utilização | Código | Precedência         | Descrição                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------------------|--------|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Ligar porta série  | CU-22  | CU-01, CU-20, CU-21 | A comunicação para a transferência de dados de e para o hardware é realizada por porta série. Assim, o utilizador pode ligar (abrir) a porta série assim que seleccionar uma das portas existentes (CU-24) e a velocidade de transmissão (CU-23). Caso a porta série seleccionada deixe de estar ativa e o utilizador ainda assim efetue a ligação, é gerado um erro. Qualquer outro tipo de erro que ocorra durante o processo de ligação (abertura) da porta série, também é reportado. Para efetuar a ligação, deverá já existir uma especificação da arquitetura de hardware pelo que os parâmetros associados são usados como valor de referência para cálculos a realizar aquando do envio e receção de dados. |

| Caso de utilização         | Código | Precedência | Descrição                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|----------------------------|--------|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Selecionar <i>baudrate</i> | CU-23  | -           | Permite ao utilizador seleccionar a velocidade de transmissão da comunicação série. Embora os módulos da arquitetura de hardware que se encarregam da comunicação série estejam configurados para um <i>baudrate</i> de 115200 bits por segundo, os valores, pré-definidos na aplicação de software, admissíveis são: 9600, 14400, 19200, 38400, 57600, 115200 bits por segundo. Na aplicação de software, por omissão, a <i>baudrate</i> é de 115200 bits por segundo. O SCH, por omissão, também especifica uma <i>baudrate</i> de 115200 bits por segundo e não é configurável. |
| Selecionar porta COM       | CU-24  | -           | Permite ao utilizador seleccionar a porta COM, de um conjunto de portas existentes, pela qual pretende iniciar uma nova comunicação. Se a porta série ficar apenas disponível depois da aplicação de software tiver sido iniciada, isto é, se a plataforma de hardware for ligada depois do arranque da aplicação, assim como o cabo para a transmissão de dados, a aplicação deve ser reiniciada de modo a ser possível seleccionar a nova porta.                                                                                                                                 |
| Desligar porta série       | CU-25  | CU-22       | Permite ao utilizador terminar (fechar) uma ligação ativa da porta série. Caso algum erro ocorra, originando, por exemplo, o impedimento para terminar a ligação, é gerado um erro. Qualquer outro tipo de erro é também reportado.                                                                                                                                                                                                                                                                                                                                                |

Tabela 3.4: Descrição de casos de utilização de comunicação

Depois de fornecer uma especificação para a arquitetura de hardware e gerar o ficheiro de *respetivo* para implementação na FPGA, os casos de utilização de execução descritos na tabela 3.5, mostram as funcionalidades que a aplicação de software oferece para interagir e comunicar com a plataforma de hardware, sobre um protocolo de comunicação específico, controlando assim a inicialização e execução do AC.

| <b>Caso de utilização</b> | <b>Código</b> | <b>Precedência</b> | <b>Descrição</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|---------------------------|---------------|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Impor condição inicial    | CU-19         | CU-02              | Ao utilizador é permitido alterar os estados das células do autómato celular, impondo assim a condição inicial desejada. Deverá existir uma especificação dos parâmetros do autómato celular, no que diz respeito à altura e largura da matriz, para que seja permitido alterar os estados das células.                                                                                                                                                                                                                                                                        |
| Iniciar / Parar           | CU-26         | CU-22, CU-27       | É possível ao utilizador iniciar ou parar a simulação do autómato celular em hardware. Para tal, deve existir uma ligação pela porta série ativa para tornar-se possível transmitir o comando de arranque ou paragem para a plataforma de hardware. Embora a opção de paragem esteja disponível para o utilizador, esta só está de facto ativa quando o modo passo-a-passo (CU-27) está ativado. Quando desativado, a paragem do sistema é automática de acordo com o protocolo implementado. Mais detalhes sobre o protocolo de transmissão de dados são dados no capítulo 4. |
| Ativar modo passo-a-passo | CU-27         | CU-26              | O modo passo-a-passo é uma das funcionalidades da arquitetura de hardware que permite ao utilizador iterar o autómato celular apenas uma vez. Esta opção deve ser ativada antes de iniciar uma nova simulação, ou seja, quando o sistema está em estado de espera.                                                                                                                                                                                                                                                                                                             |
| Ativar passo automático   | CU-28         | CU-27              | O passo automático é uma funcionalidade de software que, quando o sistema se encontra em modo passo-a-passo, permite enviar automaticamente o comando correspondente para a plataforma de hardware para processar a próxima iteração.                                                                                                                                                                                                                                                                                                                                          |
| Dar próximo passo         | CU-29         | CU-27              | Quando o sistema se encontra em modo passo-a-passo, é permitido ao utilizador avançar para a iteração seguinte do autómato celular enviando o pedido para o comando correspondente para a plataforma de hardware.                                                                                                                                                                                                                                                                                                                                                              |

| Caso de utilização              | Código | Precedência  | Descrição                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|---------------------------------|--------|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Fazer <i>download</i> dos dados | CU-33  | CU-22, CU-26 | De modo a descarregar os dados processados na plataforma de hardware, em primeiro lugar deverá existir uma ligação ativa da porta série e, em segundo lugar, uma simulação tem que ser iniciada. O <i>download</i> dos dados processados é feito sempre no fim de cada simulação ou, se o modo passo-a-passo (CU-27) estiver ativo, no fim de cada iteração, de acordo com o protocolo especificado no capítulo 5. Os dados recebidos, isto é, os estados das células do AC, aquando da sua receção, são representados graficamente após o seu processamento. O <i>download</i> dos dados não se refere apenas aos estados das células após várias iterações, mas também à possibilidade de reaver o estado inicial previamente carregado para a plataforma de hardware. |
| Fazer <i>upload</i> dos dados   | CU-34  | CU-19, CU-22 | De modo a carregar novos dados para a plataforma de hardware, deverá existir, em primeiro lugar, uma ligação ativa da porta série e, em segundo lugar, embora não seja estritamente necessário, a imposição de um estado inicial da matriz do AC. Por omissão, a matriz do AC na aplicação de software é inicializada a zeros quando especificada a sua dimensão.                                                                                                                                                                                                                                                                                                                                                                                                        |
| Processar dados                 | CU-35  | CU-34, CU-26 | Processar os dados refere-se à simulação do AC ao longo de $n$ iterações na plataforma de hardware e transmitir o resultado de volta para a aplicação de software. Em primeiro lugar, a simulação deverá ser iniciada para o cálculo das diversas iterações e, embora a matriz do AC em hardware se encontre preenchida com zeros quando este é ligado, deve-se carregar o estado desejado caso contrário a simulação será realizada com células inicializadas com o estado nulo.                                                                                                                                                                                                                                                                                        |

Tabela 3.5: Descrição de casos de utilização de execução

A última categoria refere-se aos casos de utilização de visualização que descrevem, na tabela 3.6, as funcionalidades associadas à representação gráfica dos ACs na aplicação de software.

| Caso de utilização        | Código | Precedência  | Descrição                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|---------------------------|--------|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Ver autómato graficamente | CU-30  | CU-02        | Uma das vantagens de uma aplicação com uma interface gráfica, neste tipo de sistema e aplicação em concreto, é o facto de ser possível visualizar facilmente os resultados de uma forma gráfica e intuitiva. Para qualquer especificação de um autómato celular, é possível visualizar a sua representação gráfica a partir de um mapa de cores que representam o estado associado a cada célula. Toda a interação com o autómato é realizada nesta vista gráfica: a imposição do estado inicial, por seleção das cores que codificam os estados na paleta de cores ativa, e a representação dos estados seguintes ao longo das diversas iterações. Esta vista encontra-se ativa assim que existir uma especificação dos parâmetros da arquitetura de hardware, em concreto as dimensões da matriz do autómato celular. |
| Fazer <i>zoom in</i>      | CU-31  | CU-02, CU-32 | Permite ao utilizador visualizar a representação gráfica do AC numa janela de 30x30 células. O <i>zoom in</i> é sempre passível de ser efetuado, pelo que a dimensão da matriz do CA adapta-se à área de visualização. Deverá existir uma especificação dos parâmetros da arquitetura de hardware para ser possível esta interação com a janela de visualização, em concreto as dimensões da matriz do autómato celular.                                                                                                                                                                                                                                                                                                                                                                                                |

| Caso de utilização    | Código | Precedência  | Descrição                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|-----------------------|--------|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Fazer <i>zoom out</i> | CU-32  | CU-02, CU-31 | Permite ao utilizador visualizar a representação gráfica do AC até ao limite de 200x200 células. A razão pelo qual foi imposto este limite deve-se à área disponível na janela da aplicação. Se o tamanho da matriz do CA exceder as dimensões máximas de 200x200, o <i>zoom out</i> não será possível realizar-se e será emitido um aviso. A dependência mútua entre este caso de utilização (CU-32) e o caso de utilização CU-31, refere-se ao facto de só serem permitidos dois níveis de <i>zoom</i> . Deverá, também, existir uma especificação dos parâmetros da arquitetura de hardware, em concreto as dimensões da matriz do autómato celular, para ser possível esta interação com a janela de visualização. |

Tabela 3.6: Descrição de casos de utilização de visualização

### 3.3 Cenário típico de utilização

O diagrama de interação da figura 3.3, ilustra o cenário típico de utilização do *framework* desde a especificação dos parâmetros da arquitetura de hardware até à extração dos resultados, isto é, o estado do AC ao fim de  $n$  iterações. O fluxo associado a este cenário é representado por um diagrama de atividade que se assemelha a um fluxograma, mostrando a sequência de passos necessários deste cenário de utilização. A comunicação entre a aplicação de software e a FPGA é representada num diagrama de sequência que mostra a troca de dados e comandos entre as duas plataformas. Este diagrama permite também dar uma noção temporal da sequência de passos apresentados.

O cenário típico de utilização consiste em fornecer uma especificação para o AC a gerar cujos parâmetros são as suas dimensões, o tipo de condição fronteira, o tipo de vizinhança, o número de bits por célula e a regra ou função de transição de estado. Posteriormente, é feita a validação da especificação fornecida que consiste em avaliar se os parâmetros fornecidos estão de acordo com as restrições aplicáveis à arquitetura de hardware, nomeadamente as dimensões do AC. Se o processo de validação não retornar um erro, o utilizador pode gerar o ficheiro de *bitstream* associado, que é guardado numa pasta de ficheiros de *bitstream* e descarregá-lo na FPGA caso este seja gerado corretamente. Para realizar simulações no AC gerado e após o ficheiro de *bitstream* ter sido descarregado com sucesso na FPGA, o utilizador deve iniciar a comunicação série com a

plataforma de desenvolvimento de hardware. Estando uma comunicação ativa com a plataforma de desenvolvimento de hardware, o utilizador deve definir o estado inicial do AC, a partir do visualizador gráfico na aplicação de software, e enviá-lo. De seguida o utilizador deve especificar o número de iterações que pretende que o AC realize. Por fim, a simulação pode ser iniciada e após a receção do estado final do AC ao fim do número de iterações especificado, o utilizador pode analisá-lo a partir do visualizador gráfico. O estado final recebido pode ser editado no visualizar gráfico e, caso o utilizador pretenda, pode ser enviado como um novo estado inicial.

Observe-se que caso já existam especificações de hardware, o utilizador pode seleccionar o *bitstream* correspondente e descarregá-lo na FPGA. O *framework* permite também ao utilizador, a partir da mesma especificação de hardware, fazer simulações de um AC as vezes que entender, indicando o número de iterações desejadas e a nova condição inicial.

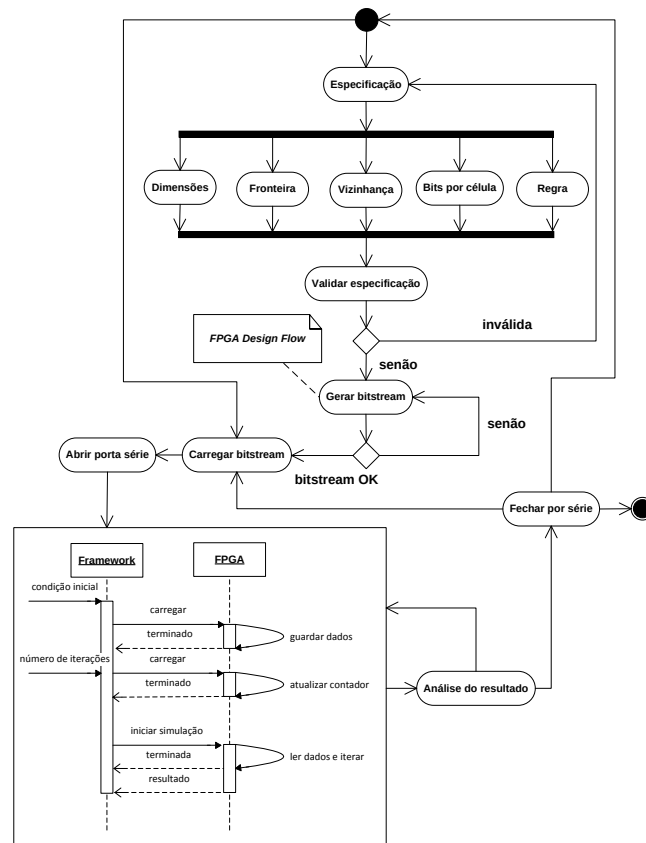


Figura 3.3: Fluxo do cenário tipo de utilização

### 3.4 Arquitetura de alto nível

O diagrama de arquitetura de alto nível, apresentado na figura 3.4, mostra a divisão da arquitetura de ambos os sistemas em módulos de alto nível, que representam conjuntos de funcionalidades que os sistemas oferecem. A constituição interna de cada um destes módulos é detalhada

nos dois capítulos posteriores: no capítulo 4 no que diz respeito ao hardware e no capítulo 5 para o software.

Do lado da FPGA, a arquitetura encontra-se dividida em dois grandes conjuntos. A interface com a aplicação de software é realizada através de um processador *soft-core*, o MicroBlaze, que se encarrega de enviar e receber comandos e dados do hardware para o software e vice-versa. O processador encarrega-se de interpretar os comandos recebidos e responder em concordância, sejam comandos vindos da aplicação ou dos módulos de controlo do hardware, e de processar os dados que circulam nos dois sentidos, tanto para leitura como escrita, organizando-os adequadamente para a transmissão ou receção. Os restantes módulos da arquitetura de hardware encontram-se organizados em memória para armazenamento de dados, e controlo que permite a leitura e escrita na memória bem como o controlo necessário para assegurar a correta operação do autómato celular e, por fim, controlo para permitir carregar e ler a matriz do autómato. O último módulo constitui toda a lógica interna da matriz do autómato, isto é, a arquitetura das células, os barramentos que as interligam e barramentos de entrada e saída para o tratamento de condições fronteira.

No que diz respeito ao *framework*, cada módulo da arquitetura de alto nível representa um conjunto de classes e funções que tratam de realizar as operações necessárias para concretizar as funcionalidades do sistema. Para a especificação de parâmetros da arquitetura de hardware, o *framework* trata de editar os ficheiros necessários que descrevem a arquitetura dos módulos de hardware, ou recorre a *templates* pré-construídos para a geração de novos ficheiros, acedendo à pasta onde o projeto se encontra, o diretório de trabalho, recorrendo às funções de leitura e escrita do gestor de ficheiros. O mesmo se aplica à gestão de paletes de cor, gestão de ficheiros de *bitstream* e respetivos ficheiros de configuração de software associados, geração de *scripts* e ao gestor de *logs*. Todos estes ficheiros se encontram armazenados no diretório de trabalho e os acessos a estes são realizados a partir do *framework*. Os *scripts* são criados aquando da geração ou *upload* de um ficheiro de *bitstream* para a FPGA, e executados a partir dos comandos necessários que são lançados em linha de comando em *background* na aplicação para iniciar estes dois processos. Para o processo de geração de um ficheiro de *bitstream*, é gerado um *script* que chama as ferramentas da Xilinx necessárias para realizar todos os passos do *design flow*, enquanto que no processo de fazer *upload* de um ficheiro de *bitstream* existente para a FPGA, é gerado um *script* que chama a ferramenta da Xilinx de programação e configuração de FPGAs, o iMPACT, que permite selecionar um ficheiro de *bitstream* existente e descarregá-lo. A saída da linha de comandos e o estado do hardware e da comunicação série são apresentados textualmente numa janela embebida na aplicação, cujo conteúdo pode ser gravado a qualquer momento. Por fim, o gestor de visualização gráfica contempla um conjunto de funções que permite a leitura e escrita na estrutura de dados que contém informação relativa a todas as células do autómato (estado, paleta de cor associada e posição no visualizador), e para representação das células no visualizador. A informação sobre a mapeamento de cor-estado associada a cada célula é consultada a partir de funções do gestor de paletes de cor.

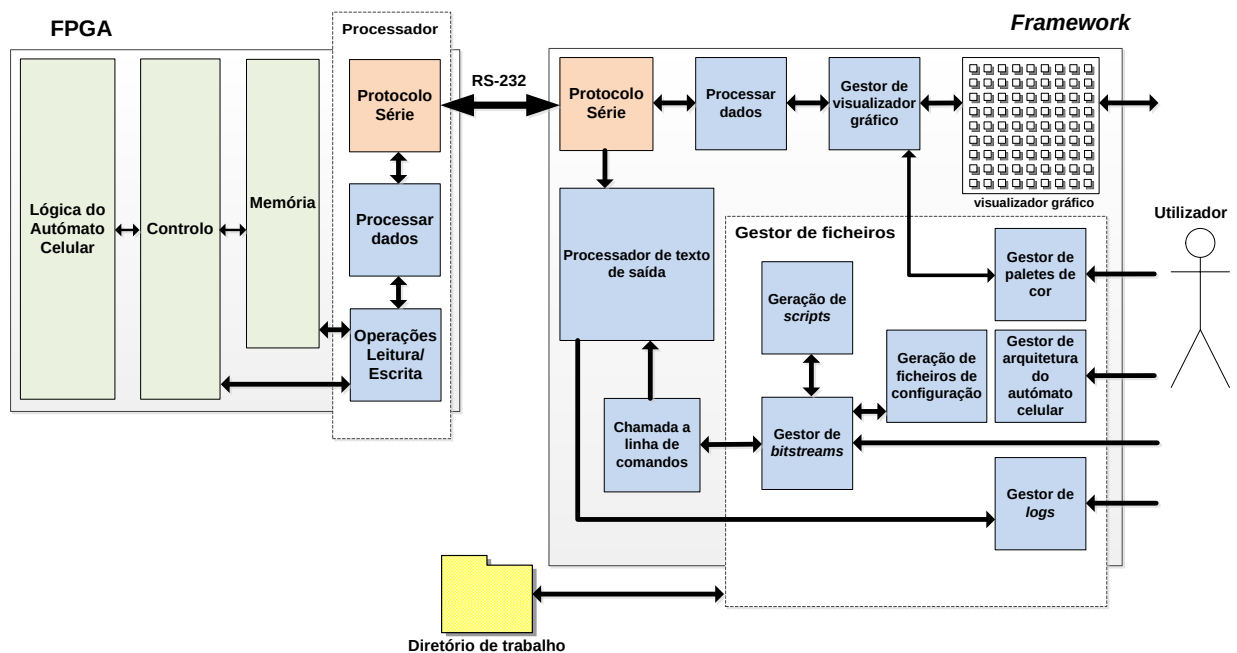


Figura 3.4: Diagrama de arquitetura de alto nível

### 3.5 Resumo de funcionalidades

Nas secções anteriores apresentaram-se as funcionalidades da perspectiva do utilizador externo e aspetos genéricos da interação de software com o hardware. Foram descritas também interações a nível interno do hardware e do software entre os diversos módulos que constituem a arquitetura de ambos os sistemas.

Para relembrar, é apresentada uma lista que compila o conjunto de funcionalidades do ponto de vista do utilizador externo:

**Dimensões do AC** A especificação das dimensões pretendidas para o AC por parte do utilizador são limitadas apenas na largura do autômato, isto é, o número de colunas da matriz deve ser sempre um múltiplo de oito.

**Tipo de AC** A arquitetura de hardware suporta ACs do tipo unidimensional (vetor) e bidimensional (matriz). Note-se que o tipo de AC está estritamente relacionado com as suas dimensões, pelo que um AC com apenas uma linha, considera-se como um AC unidimensional.

**Bits para codificação de estado** Cada célula da matriz do AC pode ter vários bits para a codificação de estado, em que o mínimo é 1 bit (2 estados).

**Vizinhança suportada** A vizinhança suportada para a arquitetura de hardware desenvolvida é do tipo de Moore ou von Neumann com  $r = 1$  (ver subsecções 2.1.1 e 2.1.2).

**Condição fronteira** As condições fronteira suportadas são tipo contínua ou periódica ou nula.

**Arquitetura das células** A arquitetura das células do AC traduz-se na regra de transição das mesmas e pode ser descrita por operações lógicas e aritméticas entre os bits de estado das células da vizinhança. O utilizador pode especificar um AC homogéneo, isto é, em que a arquitetura das células é idêntica ou especificar arquiteturas diferentes para cada linha de células da matriz do AC. Assim, o utilizador, se pretender, pode simular diversos ACs unidimensionais numa configuração bidimensional. A especificação da arquitetura das células é feita textualmente e implica ao utilizador ter conhecimentos de Verilog HDL.

**Estado inicial do AC** A partir do *framework*, o utilizador tem à sua disposição um meio gráfico, com mapeamento de cor-estado para cada célula, para a inicialização do AC. Posteriormente, o estado inicial pode ser carregado para o hardware.

**Leitura do AC** Ao fim de  $n$  iterações, o estado final do AC pode ser lido do hardware e representado da mesma forma como este é inicializado. Assim, o resultado obtido pode servir de novo estado inicial. Da mesma forma, o estado inicial previamente carregado para memória pode ser também reavido.

**Reutilização de especificações prévias** Após a geração de um ficheiro de *bitstream* para uma determinada especificação da arquitetura do hardware, este é guardado num diretório dedicado e pode ser reutilizado sempre que se pretender. Ao carregar o *bitstream* para a FPGA, o *framework* é configurado automaticamente com os parâmetros da arquitetura da especificação em causa.

**Modos de simulação** Além de especificar o número de iterações desejadas para a simulação do AC, o utilizador pode selecionar um de dois modos de simulação: contínuo ou passo-a-passo. Na simulação contínua, o AC itera  $n$  vezes e o seu estado final é devolvido para o *framework*, ou seja, o utilizador só tem acesso a uma representação visual do estado do AC na iteração  $n$ . Por outro lado, na simulação passo-a-passo, o utilizador pode visualizar graficamente estados consecutivos do AC, iteração a iteração.

**Múltiplas simulações** Para uma dada especificação, o AC pode ser simulado um número infinito de vezes. Dado um estado inicial e iterando o AC  $n$  vezes, é possível, sendo agora o estado inicial a iteração  $n$  da simulação prévia, iterar o AC mais  $m$  vezes.

As dimensões, o tipo, o número de bits para a codificação de estado, a vizinhança, a condição fronteira e a arquitetura das células definem a complexidade do AC.

### 3.6 Comparação das capacidades do SCH e SCS com as de sistemas existentes

No capítulo 2 apresentaram-se alguns sistemas de simulação de ACs, tanto em hardware, nomeadamente em FPGA, com em software. Nesta secção estabelecem-se algumas comparações entre as funcionalidades do SCH e do SCS e as desses sistemas.

### 3.6.1 Componente de hardware

Os sistemas de hardware apresentados no capítulo 2, destacam-se pelas suas implementações dedicadas a aplicações específicas. Embora algumas características do AC possam ser configuradas, como o tipo de vizinhança a considerar, dimensões e o número de bits de estado por célula, a regra de transição de estado não é passível de ser modificada, o que é justificável pois tratam-se de sistemas para aplicações específicas. Por outro lado, a arquitetura de cada célula introduz fatores de natureza aleatória, algo que não é possível fazer no sistema proposto uma vez que a especificação da regra de transição de estado só suporta operações lógicas e aritméticas entre bits de estado. Desta forma, não existem circuitos adicionais para oferecer maior complexidade para a regra de transição de estado ao ponto de suportar funções de variável aleatória. Contudo, é possível especificar uma regra de transição de estado distinta para cada linha da matriz do AC, o que não acontece nos outros sistemas de hardware.

No sistema proposto, com exceção do suporte à especificação de regras de transição de estado simples, há suporte para qualquer número de bits de estado de cada célula, dois tipos de condição fronteira e vizinhança, o número de linhas da matriz pode ser qualquer, sendo o mínimo uma linha, o número de colunas deve ser um múltiplo de 8 e o AC pode ser unidimensional ou bidimensional. Nos sistemas propostos no capítulo 2, não há qualquer indicação sobre os limites da especificação das dimensões do AC, mas há suporte para pelo menos dois tipos de vizinhança, a condição fronteira é única, isto é, do tipo contínua e o AC é apenas bidimensional.

Quanto à inicialização e leitura do estado do AC, o processo é semelhante. O AC é inicializado e o seu estado lido a partir da memória em hardware e os dados transferidos para um computador por um canal de transmissão físico qualquer.

### 3.6.2 Componente de software

A aplicação de software, para o sistema proposto, trata-se sobretudo de suporte para o sistema de hardware para controlar a operação, inicializar, ler e visualizar o estado do AC. Assim sendo, é natural que as aplicações de software dedicadas à simulação de ACs sejam bastante mais ricas em funcionalidades disponíveis do que a aplicação de software proposta. Como foi visto na secção 2.5, estas aplicações suportam diversos formatos de ficheiros para definir regras e padrões de inicialização, têm todas uma interface gráfica para manipular o AC, suportam vários tipos de ACs cuja configuração de características é diversa, têm ricas bibliotecas com pré-configurações para os ACs, aceitam *scripts* escritos pelo utilizador, recolhem dados estatísticos das simulações, permitem importar e exportar o estado do AC como imagem de diversos formatos, etc. No fundo, estas aplicações têm um número de funcionalidades bastante superior à aplicação proposta e implementada. Apenas não fazem gestão de ficheiros de *bitstream*, uma vez que não interagem com nenhuma plataforma de hardware, neste caso uma FPGA.



## Capítulo 4

# Arquitetura do Sistema de Controlo de Hardware

No capítulo 3, foi feita a introdução ao SCH, implementado numa FPGA Spartan6 da Xilinx, e ao SCS, o *framework* de simulação de ACs e *front-end* do utilizador externo, implementado em Java. Foram apresentados os casos de utilização, o cenário típico de utilização e o fluxo de atividades associado e, por fim, um diagrama de arquitetura de alto nível que ilustra a interação interna entre os diversos módulos de cada sistema e o acesso do utilizador externo a estes.

Neste capítulo, é feita a especificação da arquitetura do Sistema de Controlo de Hardware, descrevendo detalhadamente a arquitetura que constitui cada módulo do sistema, o seu funcionamento e interação com outros módulos. Na secção 4.1, apresenta-se um diagrama de arquitetura interna do hardware detalhada, expandindo o diagrama apresentado na figura 3.4, ao nível do módulo de hardware descrito em Verilog HDL descrevendo a forma como estes interagem e o seu funcionamento. Na secção 4.2, é descrito o funcionamento e arquitetura do módulo que constitui toda a lógica do AC incluindo a arquitetura celular. Os módulos responsáveis pelo controlo de condições fronteira e transferência de dados são detalhados na secção 4.3. A máquina de estados que controla toda a operação dos módulos de hardware é apresentada na secção 4.4. Na 4.5 descreve-se a implementação realizada no processador *soft-core* Microblaze no que diz respeito à organização dos dados para as operações de escrita e leitura da memória, ao tipo de comandos suportados pelo protocolo implementado e a sua especificação e ao tratamento de dados recebidos e dados a enviar. Descreve-se ainda o algoritmo implementado para a reprodução sonora de cada estado do AC e a geração de forma de onda à saída. Por fim, na secção 4.6, discutem-se soluções de arquitetura alternativas para o sistema de controlo de hardware.

### 4.1 Arquitetura detalhada

O domínio do hardware é constituído por diversos núcleos (IP) e um processador *soft-core*, Microblaze. O Microblaze atua como mestre do barramento PLB (*Processor Local Bus*) que é um barramento partilhado de 32-bits que permite ao processador interagir com os restantes núcleos

(escravos) do sistema de hardware. Os núcleos relevantes para o funcionamento do sistema são apresentados na figura 4.1. Os módulos de hardware implementados são apresentados na figura à esquerda do PLB que é o núcleo do AC. Os núcleos UART e o controlador do codec de áudio AC'97 são usados para comunicação série e reprodução sonora, respetivamente. Este último é apenas usado para efeitos de demonstração de uma aplicação prática de um AC, reproduzindo sonoramente o seu estado. O núcleo de hardware UART implementa um controlador para a comunicação série (RS-232) e é já fornecido pelas ferramentas das Xilinx. O *chip* do codec de áudio vem integrado na plataforma de desenvolvimento da Digilent utilizada, a Atlys, e o controlador é fornecido pela mesma.

Na figura 3.4 mostra-se que o núcleo do AC está dividido em três blocos fundamentais: memória, controlo e lógica do AC. O bloco de lógica do AC refere-se ao encapsulamento de todas as células de constituem o AC, sinais de controlo e barramentos para a escrita e leitura de dados da matriz. O bloco de memória é dividido em duas BRAMs (*Block RAM*), sendo uma de entrada onde os dados a carregar no AC são escritos e a outra de saída onde o estado da iteração  $i$  do AC, ambas as operações asseguradas pelo tanto pelo MicroBlaze como pelo núcleo do AC. Note-se que a BRAM de entrada pode ser também lida pelo MicroBlaze sendo que esta possibilidade refere-se à recuperação de um estado inicial carregado previamente. O bloco de controlo refere-se à máquina de estados (FSM) que controla globalmente os módulos do AC, aos *shift-registers* (SR) SIPO (*Serial In Parallel Out*) e PISO (*Parallel In Serial Out*) responsáveis pela organização dos dados para leitura e escrita das BRAMs, respetivamente, ao módulo de controlo de fronteira que controla os barramentos da lógica do AC e a leitura de dados de SIPO SR e escrita de dados em PISO SR. Um banco de registos de controlo é partilhado entre a máquina de estados e o MicroBlaze onde são registados pedidos por parte do MicroBlaze e, por sua vez, pedidos vindos da aplicação de software, por parte de um utilizador externo, e respostas por parte da máquina de estados que são interpretadas pelo Microblaze. Os sinais registados neste banco de registos são de extrema importância, pelo que definem a sequência e controlo da operação do sistema.

## 4.2 Estrutura lógica do Autómato Celular

O módulo de lógica do AC contém todas as células geradas e as suas interligações para uma determinada especificação de arquitetura. Para gerar a matriz é necessário, para além das suas dimensões, ter em atenção as interligações entre as células porque o seu número depende se o tipo de vizinhança é de Moore ou von Neumann. O facto de o tipo de vizinhança poder ser de um entre dois tipos, é particularmente importante na geração das células que se posicionam na periferia ou fronteira da matriz. Isto deve-se ao facto de estas células terem um número de entradas, isto é, interligações com a vizinhança por onde se propagam os estados das células vizinhas diferentes das das células posicionadas no interior da matriz, na medida em que parte destas interligações encontram-se viradas para a fronteira da matriz, pelo que devem ser tratadas como um caso particular. Assim, pode-se caracterizar uma célula pelos seus tipos de interligações

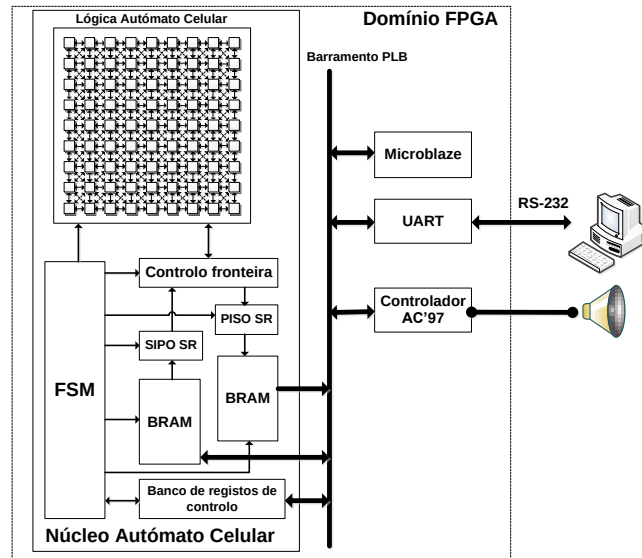


Figura 4.1: Diagrama de arquitetura de hardware detalhada

e número: interligações internas,  $x$ , e interligações de fronteira,  $y$ , com  $x + y = 8$  para vizinhança do tipo de Moore e  $x + y = 4$  para uma vizinhança do tipo de von Neumann.

Na figura 4.2 mostra-se uma representação genérica de uma matriz de em AC, com o número de colunas igual a  $c = n - 1$  e o número de linhas igual a  $r = m - 1$ . A vermelho representam-se as células da fronteira e a cinzento as células interiores. Embora as células amarelas também sejam células de fronteira, são células de canto e o seu número de interligações de fronteira é distinto do das restantes, a vermelho, pelo que representam um segundo caso particular. A azul e a verde estão representadas as interligações de fronteira para uma vizinhança de Moore. Na tabela 4.1 são apresentados os resultados por análise da figura 4.2. Note-se que o que foi descrito também é válido para ACs unidimensionais, isto é, com  $m = 1$ .

| Células   | Coordenadas                                                                                                      | Tipo de vizinhança |                |
|-----------|------------------------------------------------------------------------------------------------------------------|--------------------|----------------|
|           |                                                                                                                  | Moore              | von Neumann    |
| Interna   | $(1, 1) \dots (i, j)$                                                                                            | $x = 8; y = 0$     | $x = 4; y = 0$ |
| Canto     | $(0, 0)$<br>$(m - 1, 0)$<br>$(0, n - 1)$<br>$(m - 1, n - 1)$                                                     | $x = 3; y = 5$     | $x = y = 2$    |
| Fronteira | $(0, 1) \dots (0, j)$<br>$(1, 0) \dots (i, 0)$<br>$(m - 1, 1) \dots (m - 1, j)$<br>$(1, n - 1) \dots (i, n - 1)$ | $x = 5; y = 3$     | $x = 3; y = 1$ |

Tabela 4.1: Interligações internas ( $x$ ) e de fronteira ( $y$ ) das células

Além das dimensões e do tipo de vizinhança, a lógica do AC é também caracterizada pelo

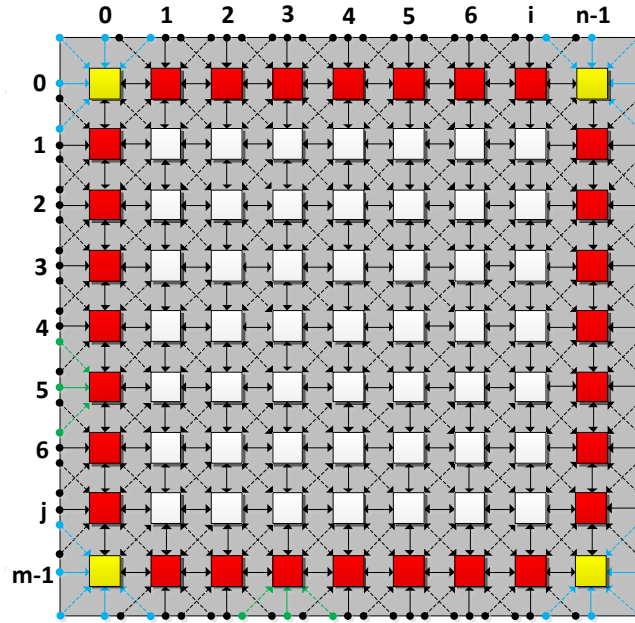


Figura 4.2: Lógica do AC: geração da matriz

número de bits por célula,  $b \geq 1$ . As interligações internas e as de fronteira são barramentos de largura  $b$  que quanto mais largos forem, maior a complexidade para interligação. O módulo de lógica do AC é caracterizado por dois barramentos de dados, um de entrada (BE) e outro de saída (BS), cujas larguras dependem de  $b$ , das dimensões do AC,  $c$  e  $r$ , do tipo de vizinhança e do tipo de AC. Estes barramentos são responsáveis pela circulação de dados entre fronteiras e acesso à lógica interna do AC. O BE é sempre mais largo que o BS, seja qual for a configuração do AC, uma vez que cada célula tem uma e só uma saída que é propagada para as células vizinhas e um número de entradas variável, conforme o tipo de vizinhança. Ambos os barramentos são organizados numa sequência de bits que representam a informação de estado das interligações de fronteira das células posicionadas nas fronteiras e cantos. A informação é organizada sequencialmente, na medida em que os bits de estado são posicionados nos barramentos de tal forma a corresponder à sequência de células adjacentes ao longo das fronteiras. Desta forma, o acesso é facilitado e os barramentos podem ser indexados  $b$  a  $b$  bits conduzindo cada grupo de  $b$  bits para as células correspondentes. Os índices dos barramentos e o posicionamento dos grupos de  $b$  bits nos mesmos iniciam-se a partir da célula de coordenada  $(0,0)$ , no sentido dos ponteiros do relógio, até à célula de coordenada  $(1,0)$  referenciando todas as interligações de fronteira ao longo das células. O excerto de código da listagem 4.1 ilustra como é realizada a indexação do barramento de entrada.

Listing 4.1: Código para indexação do barramento de entrada da lógica do AC

```

1 generate
2   genvar a;
3   for (a = 0; a < N_INPUTS; a = a + 1)

```



o tipo de vizinhança de Moore. Os ACs unidimensionais, sendo apenas um vetor de células numa estrutura linear com evolução temporal representada numa matriz em que cada linha é uma nova iteração, o seu tipo de vizinhança é também linear na medida em que só são considerados os vizinhos à esquerda e à direita de uma determinada célula. Nesta arquitetura, um AC unidimensional suporta apenas vizinhança do tipo de von Neumann, que é bidimensional, dadas as características de inicialização e leitura do estado do AC, uma vez que é necessário que as células de um AC unidimensional têm que ter interligações de fronteira para o acesso do barramento de entrada à lógica interna do AC. Sendo os ACs unidimensionais caracterizados por uma vizinhança de von Neumann nesta arquitetura, o vizinho fictício a Sul de uma célula tem sempre os seus bits de estado a zero e pode ser ignorado o vizinho a Norte, que faz parte do barramento de entrada da lógica do AC, uma vez que é usado apenas para a inicialização do AC. Estas questões são detalhadas na secção 4.3 e subsecção 4.2.1.

As expressões apresentadas na tabela 4.2 resultam de uma análise das figuras 4.2 e 4.3 em que a tracejado representam-se as interligações para uma vizinhança do tipo de Moore, sendo as restantes para uma vizinhança do tipo de von Neumann. A expressão para o BE de um AC bidimensional com uma vizinhança do tipo de von Neumann tem dois fatores distintos. O primeiro fator,  $2 \cdot c$ , representa o número de colunas da fronteira Norte e Sul da matriz e o segundo fator,  $2 \cdot r$ , da mesma forma, representa o número de linhas da fronteira Este e Oeste. Como uma célula de canto é contabilizada duas vezes em cada fator, significa que estas células possuem duas interligações. Por outro lado, para uma vizinhança de Moore, a expressão apresenta três fatores distintos. O primeiro fator,  $(2 \cdot 3) \cdot (c - 2)$ , representa o número de células das fronteiras Norte e Sul, com exceção das células de canto dada pela parcela  $-2$ . A constante  $(6 \cdot 3)$ , representa o número de interligações de fronteira para as células referidas, uma vez que cada célula de fronteira, numa vizinhança do tipo de Moore, tem três interligações. O segundo fator tem o mesmo significado para as fronteiras Este e Oeste e o terceiro fator,  $(4 \cdot 5)$ , representa o número de interligações (5) das células de canto (4). Para o caso de um AC unidimensional, a expressão para a largura do BE é caracterizada por duas parcelas em que a primeira, a constante 2, representa as duas entradas das células de canto, uma da fronteira Este e outra da Oeste. A segunda parcela,  $c$ , representa o número de entradas, uma por célula, ao longo do vector horizontal do AC na fronteira Norte. No caso do BS, o número de interligações é dado por  $c$  que é exatamente igual número de colunas do AC.

O módulo de lógica do AC é controlado por apenas dois sinais de controlo, *rule* e *shift*, que é comum a todas as células do AC. As suas funções são detalhadas na subsecção 4.2.1.

### 4.2.1 Arquitetura Celular

A arquitetura de cada célula do módulo de lógica do AC é constituída por um circuito de lógica combinacional que define o modo de operação da célula e a sua regra de transição de estado, e outro sequencial onde o estado atual da célula é armazenado em registos. Desta forma, a cada ciclo de relógio é calculada uma nova iteração do AC. O módulo que descreve a arquitetura da célula é único e é replicado para todas as células do AC no seu módulo de lógica. Este módulo

| Barramento | Tipo de AC | Tipo de vizinhança                      |                                                                     |
|------------|------------|-----------------------------------------|---------------------------------------------------------------------|
|            |            | von Neumann                             | Moore                                                               |
| Entrada    | 1D         | $2 + c$                                 | -                                                                   |
|            | 2D         | $2 \cdot c + 2 \cdot r$                 | $(2 \cdot 3) \cdot (c - 2) + (2 \cdot 3) \cdot (r - 2) + 4 \cdot 5$ |
| Saída      | 1D         | $c$                                     | -                                                                   |
|            | 2D         | $4 + 2 \cdot (c - 2) + 2 \cdot (r - 2)$ | $4 + 2 \cdot (c - 2) + 2 \cdot (r - 2)$                             |

Tabela 4.2: Largura dos barramentos de entrada e saída em número de interligações com as células. Multiplicando as expressões por  $b$ , obtém-se a largura em número total de bits, com  $b \geq 1$ .

é também caracterizado por um barramento de entrada (BE) e um barramento de saída (BS), que corresponde às interligações com a sua vizinhança imediata, seja do tipo de Moore ou von Neumann, e ao estado atual da célula, respetivamente. A largura do BE é variável, tal como no módulo de lógica do AC, consoante o tipo de vizinhança considerada e está também organizado em grupos de  $b$  bits que correspondem aos bits de estado propagados para a célula pelas interligações da sua vizinhança. De igual modo, este barramento é também indexado para fácil acesso e identificação do vizinho de origem. O modo de operação de cada célula é controlado por dois sinais de um bit, *shift* e *rule*, e tem três modos de operação. Na tabela 4.3, apresenta-se a tabela de verdade e descrição associada ao modo de operação correspondente e na tabela 4.4 mostram-se as expressões que permitem calcular as larguras do BE e do BS. Designe-se por  $v_{Moore}$  uma vizinhança do tipo de Moore e por  $v_{Neumann}$  uma vizinhança do tipo de von Neumann, o número de células a considerar. Relembre-se que como  $r = 1$ ,  $v_{Neumann} = 5$  e  $v_{Moore} = 9$ .

| Sinal        |             | Operação realizada                                                                                                                                                                      |
|--------------|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>shift</i> | <i>rule</i> |                                                                                                                                                                                         |
| 0            | 0           | A célula mantém o seu estado atual.                                                                                                                                                     |
| 0            | 1           | Aplicando a regra de transição de estado descrita na arquitetura da célula e a partir do conteúdo presente no BE, no próximo ciclo de relógio é determinado o próximo estado da célula. |
| 1            | 0           | O estado seguinte da célula, no próximo ciclo de relógio, é igual ao estado anterior do vizinho imediatamente a Norte.                                                                  |
| 1            | 1           | Embora esta situação não aconteça porque a máquina de estados que controla o sistema não o permita, o sinal <i>rule</i> tem prioridade sobre o sinal <i>shift</i> .                     |

Tabela 4.3: Modos de operação da célula e sua descrição

A parcela  $-1$  representa o facto da célula que se está a considerar não contribuir para a caracterização da largura do BE, uma vez que este só transporta dados das interligações da vizinhança da célula e não da própria célula. Na tabela 4.5 mostra-se como está organizado o BE em função do tipo de vizinhança considerada. Para facilitar a referência à interligação dos vizinhos imediatos, usa-se a notação de pontos cardeais e colaterais, em relação a uma célula qualquer do AC, para representar os bits de estado das células vizinhas. A listagem 4.2, mostra o código usado no módulo que descreve a arquitetura da célula para a indexação do BE.

| Barramento | Tipo de AC | Tipo de vizinhança           |                            |
|------------|------------|------------------------------|----------------------------|
|            |            | von Neumann                  | Moore                      |
| Entrada    | 1D         | $(v_{Neumann} - 1) \times b$ | -                          |
|            | 2D         | $(v_{Neumann} - 1) \times b$ | $(v_{Moore} - 1) \times b$ |
| Saída      | 1D         | $b$                          | -                          |
|            | 2D         | $b$                          | $b$                        |

Tabela 4.4: Largura dos barramentos de entrada e saída de uma célula em número de bits, com  $b \geq 1$ .

Listing 4.2: Código para indexação do barramento de entrada da célula

```

1 generate
2   genvar a;
3   for (a = 0; a < NEIGHBOURHOOD; a = a + 1)
4     begin: gen_iboundary
5       wire [BITS_PER_CELL-1:0] input_wires;
6       assign input_wires =
7         cell_in [(a+1)*BITS_PER_CELL-1:a*BITS_PER_CELL];
8     end
9 endgenerate

```

| Tipo de vizinhança | Índice do barramento ( $i$ ) |       |       |       |          |          |          |          |
|--------------------|------------------------------|-------|-------|-------|----------|----------|----------|----------|
|                    | 0                            | 1     | 2     | 3     | 4        | 5        | 6        | 7        |
| Moore              | $v_S$                        | $v_E$ | $v_N$ | $v_O$ | $v_{SE}$ | $v_{SO}$ | $v_{NE}$ | $v_{NO}$ |
| von Neumann        | $v_S$                        | $v_E$ | $v_N$ | $v_O$ |          |          |          |          |

Tabela 4.5: Organização do barramento de entrada por indexação em  $i$  de  $b$  bits

Na listagem 4.2, na linha de código 7 o sinal *cell\_in* representa o BE e NEIGHBOURHOOD, na linha de código 3, é a igual a  $v_{Neumann} - 1$  ou  $v_{Moore} - 1$ , conforme o caso. O barramento de entrada *gen\_iboundary[i].input\_wires*, da mesma forma que no módulo de lógica do AC, é indexado em  $i$  e o sinal resultante é um barramento de interligação de células de BITS\_PER\_CELL bits. Por exemplo, *gen\_iboundary[0].input\_wires* é igual a  $v_S$ .

Na figura 4.4 apresenta-se um diagrama de blocos da arquitetura de uma célula. Os sinais de controlo da célula *shift* e *rule* selecionam uma das entrada de dois multiplexadores que é propagada para o registo de estado da célula de acordo com a tabela de verdade indicada na tabela 4.3. A arquitetura das células é também caracterizada pelo facto de a regra de transição de estado não ter de ser necessariamente a mesma entre todas. Quando o módulo de arquitetura da célula é instanciado do módulo de lógica do AC, é passado automaticamente a este um parâmetro numérico que designa a linha da matriz do AC e que seleciona de entre um conjunto regras, a regra referenciada pelo valor do parâmetro. No fundo, considera-se que existe um banco de regras que

é um conjunto de ficheiros que além de descreverem o modo comum de operação da célula e a sua arquitetura base, contém a função que descreve como é calculado o estado seguinte da célula. O ficheiro que integra o corpo da arquitetura da célula é seleccionado pelo parâmetro numérico passado ao módulo a partir de uma estrutura do tipo *case*. Embora nesta arquitetura uma linha de células tem sempre a mesma regra de transição de estados, pois o módulo de lógica do AC assim o implementa e porque em termos práticos é mais relevante, é possível passar manualmente o parâmetro para seleccionar qualquer ficheiro desde que este exista. O processo de selecção do ficheiro que integra o corpo do módulo da arquitetura da célula é feito aquando da síntese lógica. A listagem 4.3 mostra o conteúdo do módulo de uma célula, o seu cabeçalho e o corpo. O excerto de código da listagem 4.2 também é integrado no módulo mas por uma questão de simplicidade já não é representado.

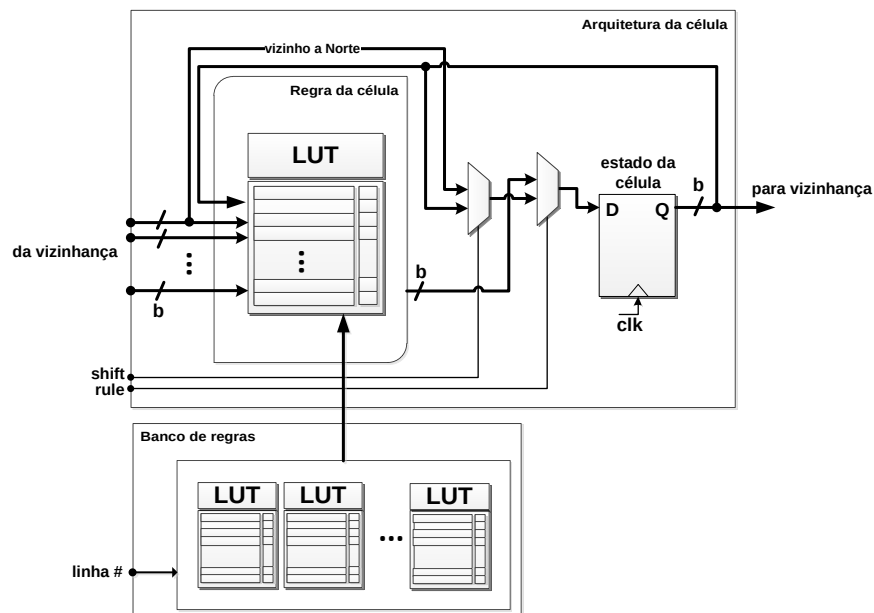


Figura 4.4: Arquitetura celular

Listing 4.3: Estrutura do módulo de arquitetura celular

```

1 module ca_cell(clk, rst, cell_in, shift, rule, cell_out);
2
3 'include "includes/parameter.vh"
4
5 parameter RULE = 0;
6
7 input    clk,
8 input    rst,
9 input    shift,

```

```

10 input    rule;
11 input    [NEIGHBOURHOOD*BITS_PER_CELL-1:0] cell_in;
12 output   [BITS_PER_CELL-1:0] cell_out;
13 reg      [BITS_PER_CELL-1:0] cell_out;
14
15 generate
16     case (RULE)
17
18         8'd0:
19             begin: row_0
20                 'include "includes/ca_rows/row_0.v"
21             end
22
23         8'd1:
24             begin: row_1
25                 'include "includes/ca_rows/row_1.v"
26             end
27
28         8'd2:
29             begin: row_2
30                 'include "includes/ca_rows/row_2.v"
31             end
32
33         ...
34
35         8'd255:
36             begin: row_255
37                 'include "includes/ca_rows/row_255.v"
38             end
39
40         default:
41             begin: default_rule
42                 'include "includes/ca_rows/row_0.v"
43             end
44
45     endcase
46 endgenerate

```

Na listagem 4.3, os barramentos *cell\_in* e *cell\_out* correspondem ao BE e BS, respetivamente. Na linha de código 3, o conteúdo do ficheiro incluído é apresentado na listagem 4.4 e refere-se a parâmetros partilhados pelos módulos da arquitetura de hardware para parametrização externa

das dimensões do AC, tipo de vizinhança e o número de bits de estado por célula e caracterização da arquitetura global de hardware. O parâmetro passado pelo módulo de lógica do AC designa-se por RULE e varia entre 0, que corresponde à linha zero da matriz do AC, até, neste momento, 255 embora seja possível acrescentar mais referências na estrutura *case*. Note-se que se o AC for homogéneo, isto é, se as células tem todas a mesma regra de transição de estado, o conteúdo dos ficheiros é idêntico. Nesta situação a estrutura *case* não seria necessária mas como se pretende garantir a funcionalidade de gerar ACs unidimensionais numa estrutura bidimensional há a necessidade de especificar regras de transição de estado diferentes para cada linha da matriz do AC. Mesmo que o AC não seja homogéneo, consoante o tipo de vizinhança, é ainda possível referenciar células vizinhas para o cálculo do próximo estado que pertençam a essa mesma vizinhança. Contudo, é interesse do utilizador referenciar, para cada célula de uma linha da matriz do AC, apenas as células vizinhas localizadas a Este e Oeste pois, na prática, para um AC unidimensional só haja interesse nessa vizinhança.

A listagem 4.5 mostra como é realizada a instância de módulos de arquitetura celular no módulo de lógica do AC, com a passagem do parâmetro para selecionar o ficheiro que descreve a regra de transição de estado e a arquitetura comum da célula e a listagem 4.6 mostra o conteúdo de um dos ficheiros de inclusão que se trata de uma descrição textual do diagrama da figura 4.4.

Listing 4.4: Parâmetros da arquitetura de hardware

```
1 parameter CA_HEIGHT = 32;
2 parameter CA_WIDTH = 32;
3 parameter BITS_PER_CELL = 1;
4 parameter NEIGHBOURHOOD = 4;
```

Listing 4.5: Instância de células com passagem de parâmetro de seleção do ficheiro de inclusão.

[illegible]

```

17             cdata[i+1][j-1],    // south-west
18             cdata[i+1][j+1],    // south-east
19             cdata[i][j-1],      // west
20             cdata[i-1][j],      // north
21             cdata[i][j+1],      // east
22             cdata[i+1][j]      // south
23         }
24     ),
25     .shift(cell_shift),
26     .rule(cell_rule),
27     .cell_out(cdata[i][j])
28 );
29     end
30 end
31 end
32 endgenerate

```

Listing 4.6: Conteúdo de um dos ficheiros de inclusão de arquitetura celular

```

1  // BEGIN ASSIGN
2  wire[3:0] bit_count;
3
4  assign bit_count = gen_iboundary[0].input_wires +
5      gen_iboundary[1].input_wires +
6      gen_iboundary[2].input_wires +
7      gen_iboundary[3].input_wires +
8      gen_iboundary[4].input_wires +
9      gen_iboundary[5].input_wires +
10     gen_iboundary[6].input_wires +
11     gen_iboundary[7].input_wires;
12 // END ASSIGN
13
14 always @(posedge clk)
15 begin
16     if (rst)
17     begin
18         cell_out <= 0;
19     end
20     else
21     begin
22         if (shift)

```

```

23     begin
24         cell_out <= gen_iboundary[2].input_wires;
25     end
26     else
27         begin
28             if (rule)
29                 begin
30                     // BEGIN RULE
31                     cell_out <= (bit_count == 2 & cell_out) | bit_count == 3;
32                     // END RULE
33                 end
34             end
35         end
36     end

```

O código apresentado na listagem 4.5 refere-se à geração das células internas da matriz de um AC bidimensional, para uma vizinhança do tipo de Moore. O sinal *cdata* é o barramento de interligações de células de dimensão idêntica ao AC especificado, em que *cdata[i][j]* representa o estado atual da célula de coordenadas  $(i, j)$ . A listagem 4.6 apresenta a arquitetura de uma célula para um AC bidimensional com uma regra de transição de estados, a título de exemplo, equivalente à do “Jogo da Vida”.

### 4.3 Controlo de fronteira e transferência de dados

Na secção 4.2 foi detalhada a arquitetura do módulo de lógica do AC e a arquitetura celular. Nesta secção, explica-se como é que o módulo de controlo de fronteira e transferência de dados atua sobre os barramentos de entrada e saída do módulo de lógica do AC.

O módulo para controlo de fronteira e transferência de dados é constituído inteiramente por lógica combinacional que permite redirecionar barramentos de dados. O redirecionamento dos barramentos é controlado pela máquina de estados (FSM), que controla os módulos restantes do núcleo do AC, e assegurado por um conjunto de multiplexadores em dois andares distintos (em série) em que cada andar é controlado por um sinal comum de um bit para a seleção de um barramento de dados. Estes barramentos interligam os módulos de lógica do AC, para controlo dos barramentos de fronteira de entrada e saída, e os *shift-registers* SIPO, para a serialização de dados provenientes de memória e inicialização do AC, e PISO, para a deserialização de dados provenientes do AC para escrita em memória e leitura do AC. Na arquitetura detalhada do sistema na figura 4.1 é possível observar, de forma genérica, as interligações referidas.

A arquitetura dos *shift-registers* (SR) é apresentada na figura 4.5. As memórias a que os SRs acedem para leitura e escrita de dados são BRAMs de porta dual em que uma porta tem largura de 8 bits a outra de 32 bits e sua profundidade é de 4096 bytes, o número de bytes endereçáveis pela porta de 8 bits. As portas de 8 bits são usadas pelos SRs para ler da memória de entrada e para

escrever na memória de saída. Por outro lado, as portas de 32 bits são usadas pelo barramento PLB para acesso por parte do processador MicroBlaze, da mesma forma, para inicializar ou ler o AC. Embora possa ser configurado para 64 bits, o barramento PLB, por omissão, é de 32 bits e é a largura que o barramento utiliza na arquitetura de hardware implementada.

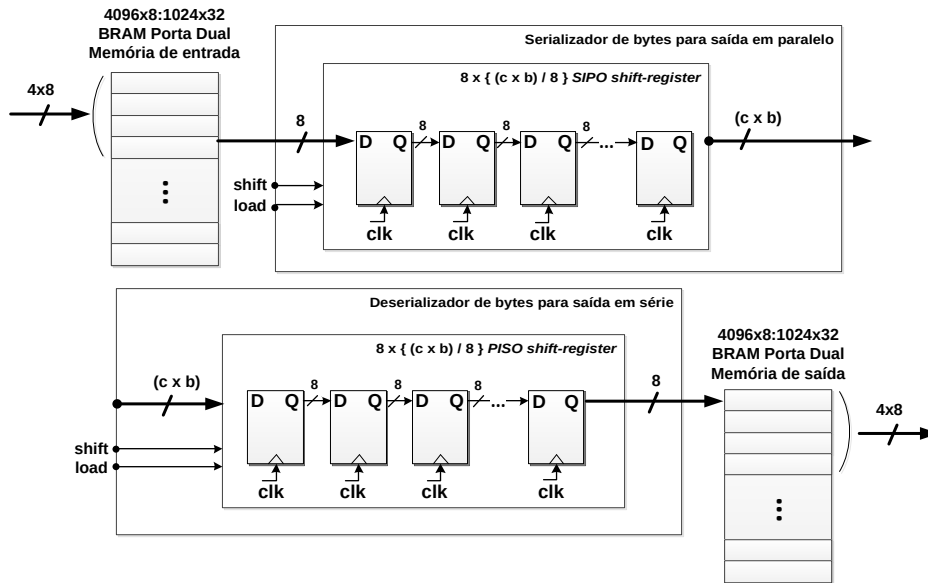


Figura 4.5: Arquitetura dos *shift-registers* PISO e SIPO

Cada andar dos SRs tem 8 bits ou 1 byte de profundidade para permitir ler e escrever da memória na porta respetiva. O número de andares de cada SR é dado por  $(c \times b)/8$  e o número total de bits armazenado no SR é dado por  $(c \times b)$ , em que  $c$  é o número de colunas da matriz do AC e  $b$  o número de bits de estado por célula. Assim, conclui-se rapidamente que  $c \times b$  deverá ser um múltiplo inteiro de 8, pois o número de andares do SR tem de ser um número inteiro. Mais concretamente,  $c$ , o número de colunas, ou a largura do AC, deverá ser sempre um múltiplo de 8, significando assim que a largura do AC mais pequeno deverá ser sempre de pelo menos  $(8 \times 1)$  bits e 8 células. Embora, nesta arquitetura, só seja então permitido gerar ACs com um número de colunas múltiplo de 8, o número de linhas não está restrito a esta imposição uma vez que o AC é lido linha a linha como será explicado mais à frente nesta secção. A razão pelo os dados em memória são lidos e escritos byte a byte, deve-se ao facto de reduzir a complexidade de implementação da arquitetura de hardware, uma vez que se pretende gerar ACs com dimensões variadas e com um número de bits por célula também variável, tornando a arquitetura o mais flexível possível. Se se usasse uma porta de acesso a dados de maior largura e se se considerasse qualquer dimensão para o número de colunas da matriz do AC, a complexidade da arquitetura de hardware aumentava uma vez que obrigaria a tratar à parte bits de estado excedentes, isto é, quando  $(c \times b)$  não é um múltiplo de  $w$  bits da largura da porta de acesso à memória,  $W = (c \times b)/w$ , o número de conjuntos de  $w$  bits a escrever ou ler da memória, não seria um número inteiro. Desta forma, como não é possível considerar frações de bits, haveriam mais  $W' = (c \times b) - W \times w$

conjuntos de  $w$  bits, sendo  $W'$  o resto da divisão inteira  $W$ , que deveriam ser enviados ou recebidos. Assim, deveriam ser enviados ou recebidos  $b_w = w \cdot (W + W')$  bits para transportar  $(c \times b)$  bits de estado e  $b_w - (c \times b)$  são bits que não têm qualquer significado. Se  $w \neq 8$ , numa comunicação orientada ao byte, os  $b_w$  de dados deveriam ser distribuídos em conjuntos de 8 bits adotando um processo semelhante ao descrito atrás, o que complicaria ainda mais o processamento dos dados.

Uma vez que o AC é lido linha a linha, o número de bits de cada linha é também  $(c \times b)$ . O SIPO SR armazena uma linha a carregar no AC e tem uma saída de largura de  $(c \times b)$  para o efeito. Após serializar da memória cada  $(c \times b)/8$  bytes, uma nova linha pode ser carregada para o AC em paralelo. Da mesma forma, o PISO SR armazena em paralelo uma linha proveniente do AC e escreve, byte a byte, o seu conteúdo na memória de saída. Cada um dos SRs são controlados por dois sinais independentes, *shift* e *load*, que permitem deslocar o SR de um byte da esquerda para a direita e carregar um novo byte para o andar mais à esquerda, no caso de SIPO SR, e carregar uma nova linha do AC no caso de PISO SR, respetivamente.

Na figura 4.6 é apresentado um diagrama de blocos da arquitetura do módulo de controlo de fronteira e transferência de dados em conjunto com a FSM, SRs e BRAMs para ilustrar facilmente a interação entre os diversos módulos. Consoante o modo de operação em que o sistema se encontra, os barramentos são redirecionados pela FSM atuando nos sinais de seleção dos multiplexadores. Repare-se que o barramento a roxo, do ponto de vista do módulo de lógica do AC, corresponde a uma fração do seu barramento de entrada, ou seja, às interligações de fronteira das células de fronteira e canto a Norte. Da mesma forma, o barramento a azul, corresponde a uma fração do barramento de saída, mais concretamente, às interligações de fronteira das células de fronteira e canto a Sul. Os barramentos a vermelho e rosa têm o mesmo significado para as fronteiras Este e Oeste. No caso destes dois últimos, as interligações de fronteira dos barramentos de saída são diretamente redirecionadas para as interligações de fronteira do barramento de entrada do módulo de lógica do AC, tendo o cuidado de referenciar corretamente as interligações, para o tipo de vizinhança configurada, ao longo das fronteiras na ligação dos barramentos.

Quando o sistema se encontra no modo de inicialização do AC, a memória de entrada é lida byte a byte. Sempre que SIPO SR carrega e desloca um número de bytes equivalente à largura do AC, isto é,  $(c \times b)/8$  bytes, os barramentos a verde e castanho são selecionados nos multiplexadores, o sinal de *shift* do módulo de lógica do AC é colocado a 1 e, no ciclo de relógio seguinte, todas as células do AC adquirem o estado do vizinho a Norte, ou seja, o AC é deslocado de cima para baixo em uma linha, como é visível na figura. A linha do AC da fronteira Sul é perdida uma vez que o primeiro multiplexador seleciona o barramento verde. Após carregada uma linha no AC, o processo de ler mais bytes da memória, carregar e deslocar o SIPO SR com  $(c \times b)/8$  bytes repete-se até que sejam carregadas  $r$  linhas no AC, isto é, o número de total linhas do AC.

No modo de leitura do estado atual do AC, são selecionados os barramentos a azul e castanho nos multiplexadores. Quando o sinal *shift* do módulo de lógica do AC é colocado a 1, a matriz do AC é deslocada também de cima para baixo de uma linha. A linha que estava presente na fronteira Sul do AC é deslocada para a fronteira Norte e carregada, no mesmo ciclo de relógio com o sinal *load* a 1, para o PISO SR. A linha do AC é escrita em memória byte a byte até o

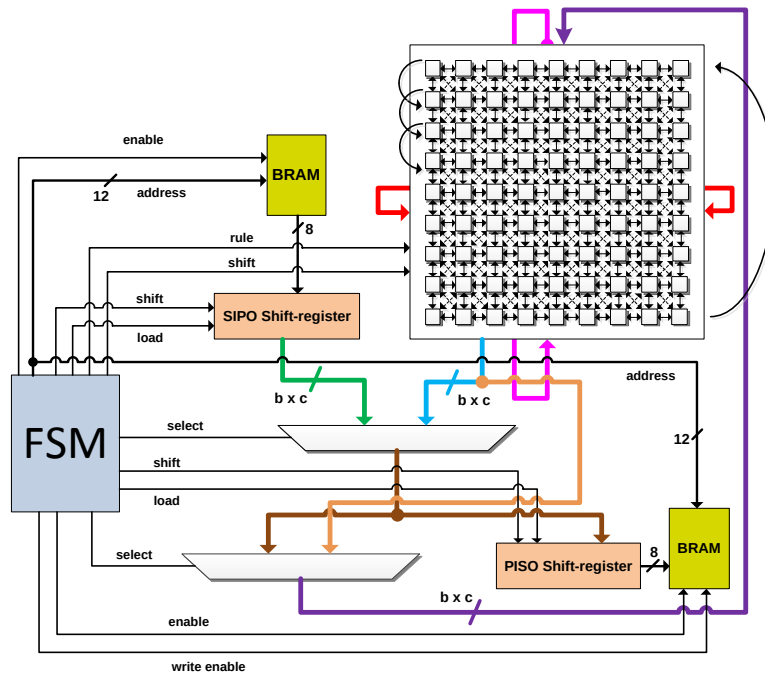


Figura 4.6: Arquitetura do módulo de controlo de fronteira e transferência de dados.

SR ter sido deslocado de  $(c \times b)/8$  bytes. Assim que a linha tiver sido descarregada, a matriz do AC é deslocada novamente e uma nova linha é adquirida. Neste modo, o AC encontra-se fechado sobre si mesmo num cilindro em que as suas linhas são deslocadas e repostas na matriz. Após deslocadas  $r$  linhas, o AC encontra-se exatamente com o mesmo estado antes de ter sido iniciado o processo da sua leitura e, na memória de saída, encontram-se armazenados os estados de todas as células do AC.

Por fim, quando o AC se encontra a iterar, há que distinguir se o tipo de fronteira é contínua (periódica) ou nula. Na figura 4.6, os barramentos a vermelho e laranja/roxo representam a interligação da fronteira Este e Oeste e da fronteira Norte e Sul, respetivamente, e a configuração apresentada caracteriza o AC por uma fronteira do tipo contínua, pelo que o barramento laranja é seleccionado no segundo multiplexador. Sendo a fronteira do tipo contínua, o AC é fechado nas fronteiras Norte-Sul e Este-Oeste pelo que se torna num toroide. Note-se que os barramentos laranja e roxo só interligam as fronteiras Norte e Sul no sentido Sul-Norte. A interligação no sentido Norte-Sul é direta e é feita pelo barramento a rosa. Repare-se também que se a arquitetura apenas suportasse uma fronteira do tipo contínua, não fazia qualquer sentido o segundo multiplexador existir já que o barramento castanho poderia ser ligado diretamente a PISO SR e à fronteira Norte do AC. Como existe a possibilidade de especificar uma arquitetura de um AC com fronteira do tipo nula, nesta situação, os barramentos laranja, rosa e vermelho estariam ligados à massa. Assim, o barramento de saída do módulo de lógica do AC seria apenas utilizado para a inicialização e leitura do AC e o barramento de entrada estaria ligado à massa, pelo que as células de canto e fronteira, nas suas interligações de fronteira, veem vizinhos fictícios com os bits de estado a zero,

caracterizando-se assim uma fronteira do tipo nula.

## 4.4 Máquina de estados

Na secção 4.3 explicou-se detalhadamente como é que o módulo de controlo de fronteira e transferência de dados funciona nos três modos de operação do sistema: inicialização, leitura de estado e iteração do AC. Nesta secção, mostra-se como é que é realizado o controlo dos módulos de hardware a partir da máquina de estados responsável pela operação global dos módulos do núcleo do AC.

Na figura 4.7 apresenta-se um diagrama de estados associado ao fluxo de operação do sistema. As etiquetas de fluxo a azul representam comandos vindos do banco de registos de controlo e, por sua vez, do processador MicroBlaze. Por outro lado, as etiquetas de fluxo a vermelho representam respostas, sinais de um bit, da máquina de estados, à conclusão de uma determinada operação, sendo essa resposta sinalizada no banco de registos de controlo para posterior leitura do MicroBlaze (ver interação no diagrama de arquitetura detalhada da figura 4.1). O processo de registo de comandos e respostas no banco de registos de controlo é explicado na subsecção 4.4.1.

Quando o sistema de hardware se inicia, este é colocado num estado de espera (AC PARADO). Assim que no banco de registos de controlo seja sinalizado o bit de arranque da operação do AC ou o bit de inicialização do AC, o sistema troca o seu estado para AC INICIADO ou SIPO SR, respetivamente. Se o comando recebido for para inicializar o AC e, mesmo que a memória de entrada não tenha sido carregada, uma vez que todos os endereços são inicializados a zero, o sistema inicia o processo de leitura da memória, byte a byte, e vai carregando a matriz do AC linha a linha. Sempre que SIPO SR tiver o número de bytes equivalentes a uma linha do AC, esta é carregada e, até que todas as linhas tenham sido carregadas na matriz, o sistema continua a ler bytes da memória e carregar novas linhas. Assim que todas as linhas tiverem sido carregadas, o sistema regressa ao estado inicial AC PARADO.

A partir do estado inicial AC PARADO, se o comando recebido for para iniciar a operação do AC o sistema transita para o estado AC INICIADO. Daqui, há a necessidade de verificar a veracidade de duas condições. Se o bit de controlo que sinaliza que o sistema deve iterar passo-a-passo estiver a 1, o sistema itera o AC uma vez e transita para o estado CARREGAR PISO SR para dar início à leitura do estado do AC da última iteração efetuada. Se o sistema não se encontrar no modo passo-a-passo, este continua a iterar até que o número máximo de iterações impostas ao sistema, registado no banco de registos de controlo, seja atingido e posteriormente transita também para o estado CARREGAR PISO SR para ler o estado do AC da última iteração feita. A matriz do AC é lida linha a linha, carregada no SR, de entrada paralela e saída em série, que vai deslocando e escrevendo na memória de saída os bytes que transportam a informação de estado das células do AC. Sempre que uma nova linha da matriz do AC está disponível em memória, é colocado o bit de controlo a 1 que corresponde à sinalização de uma nova linha disponível em memória. Esta linha é processada pelo MicroBlaze e assim que o processamento tiver sido terminado, o processador envia o comando de pedido de uma nova linha, colocando o bit correspondente a 1 no banco de

registos de controlo. Assim que a máquina de estados receber a sinalização do comando pedido, uma nova linha da matriz do AC é descarregada em memória e o processo repete-se até que todas as linhas tenham sido descarregadas em memória. Nesta situação, a matriz do AC está toda disponível em memória. Repare-se agora que o estado ESPERAR COMANDO ITERAÇÃO só é atingido se o sistema tiver sido colocado previamente no modo passo-a-passo, caso contrário após a transição do sistema para o estado AC INICIADO, o número máximo de iterações acaba por ser atingido e o bit de controlo que sinaliza esta situação é colocado a 1. Este sinal, além de ser registado no banco de controlo de registos é também avaliado quando o AC é descarregado em memória. Se o número máximo de iterações não tiver sido atingido, o sistema transita para o estado ESPERAR COMANDO ITERAÇÃO e encontra-se no modo passo-a-passo. O sistema regressa ao estado AC INICIADO assim que receber que o bit de controlo do comando respetivo for a 1, e é calculada a próxima iteração. Enquanto o sistema se encontra no estado ESPERAR COMANDO ITERAÇÃO, é possível abortar a execução do AC caso o processador envie o comando de paragem do sistema, regressando ao estado inicial AC PARADO. Caso o sistema se encontre no modo passo-a-passo, se o processador o solicitar, este modo pode ser desativado e o respetivo bit de controlo é colocado a 0. Nesta situação, quando o sistema regressa ao estado AC INICIADO, o AC itera até atingir o número máximo de iterações imposto, não sendo possível regressar ao modo passo-a-passo enquanto o sistema não regressar ao estado AC PARADO.

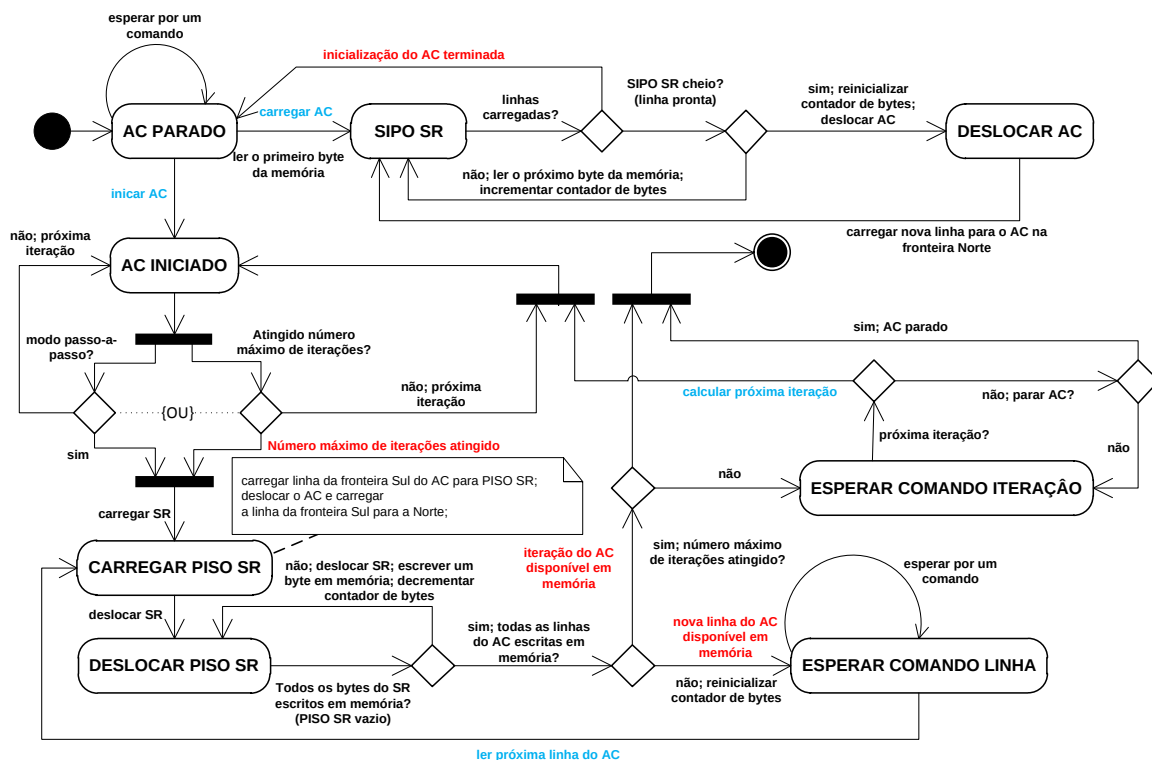


Figura 4.7: Diagrama de estados

#### 4.4.1 Registo de comandos e respostas no banco de registos de controlo

Os comandos são enviados pelo processador MicroBlaze e as respostas enviadas pela máquina de estados, e ambos registados num banco de registos de controlo comum no módulo que realiza a interface entre o barramento PLB e núcleo de hardware do AC. Este banco de registos é partilhado entre o barramento PLB, para acesso por parte do processador MicroBlaze, e pela máquina de estados do sistema. Além de sinais de controlo de comando e resposta, são também armazenados nestes registos as dimensões e o número de bits por célula do AC, para a realização de cálculos aquando da leitura e escrita de bytes em memória e transferência de dados para o PC, e o número de iterações que o AC deve realizar. São usados três registos de 32 bits, uma vez que o barramento PLB também é de 32 bits, e a sua organização é apresentada na tabela 4.6.

| Registo | Direção    |            | Bits    | Descrição                               |
|---------|------------|------------|---------|-----------------------------------------|
|         | Leitura    | Escrita    |         |                                         |
| 0       | FSM        | MicroBlaze | [0:31]  | Número de iterações a realizar pelo AC. |
| 1       | FSM        | MicroBlaze | [0:25]  | Sem uso.                                |
|         |            |            | 26      | Activar/Desactivar modo passo-a-passo.  |
|         |            |            | 27      | Calcular próxima iteração do AC.        |
|         |            |            | 28      | Ler próxima linha do AC.                |
|         |            |            | 29      | Parar AC.                               |
|         |            |            | 30      | Iniciar AC.                             |
|         |            |            | 31      | Inicializar AC.                         |
| 2       | FSM        | Microblaze | [0:7]   | Número de colunas do AC.                |
|         |            |            | [8:15]  | Número de linhas do AC.                 |
|         |            |            | [16:23] | Número de bits por célula.              |
|         |            |            | [24:27] | Sem uso.                                |
|         | Microblaze | FSM        | 28      | Nova linha do AC disponível em memória. |
|         |            |            | 29      | Número máximo de iterações atingido.    |
|         |            |            | 30      | Iteração do AC disponível em memória.   |
|         |            |            | 31      | Inicialização do AC terminada.          |

Tabela 4.6: Organização do banco de registos de controlo

Os bits 28, 29, 30 e 31 do registo dois assinalam as possíveis respostas da máquina de estados no fluxo de operação do sistema. Estes bits são colocados a 1, para sinalizar a resposta respetiva, assim que a máquina de estados coloca as saídas ligadas a estes bits de controlo a 1. Enquanto o processador MicroBlaze não efetuar uma leitura do estado do registo dois de controlo, o seu estado é mantido mesmo que a máquina de estados coloque as saídas correspondentes a 0. Desta forma, mesmo que o barramento PLB esteja ocupado, a efetuar escritas ou leituras noutra núcleo de hardware, e a leitura do registo só possa ser efetuada mais tarde, a(s) resposta(s) da máquina de estados mantêm-se registadas pelo que não há perda de informação de controlo. Após a leitura do registo dois, os quatro bits de controlo são colocados a 0.

Os comandos vindos do processador MicroBlaze são registados no registo um e correspondem aos bits de controlo 27 a 31. Assim que um destes bits de controlo seja colocado a 1, a máquina

de estados processa o pedido feito e realiza a operação descrita. Estes sinais tem duração exata de um ciclo de relógio para evitar repetir a operação de um dado comando mais que uma vez. Se apenas fosse usado o registo um com os bits de controlo especificados ligados diretamente às entradas da máquina de estados correspondentes a estes comandos, poderia acontecer que se o barramento PLB estivesse ocupado durante vários ciclos de relógio, o número de ciclos de relógio que decorrem, por exemplo, entre um comando de inicialização do CA e o regresso do sistema ao estado inicial, poderia fazer com que o bit de controlo de inicialização do AC não tivesse sido repostado a 0 pelo processador e o AC seria novamente inicializado. Da mesma forma, poderia acontecer que quando fosse dado o comando para o pedido de leitura de uma nova linha da matriz do AC e o sistema regressando de novo a este estado, devendo ficar à espera de um novo comando para ler a linha seguinte da matriz do AC, como o bit de controlo não tinha sido repostado a 0, faria com que uma nova linha fosse lida sem o próximo comando ter sido enviado pelo processador. Nestas situações, poderiam ocorrer problemas de sincronismo entre a máquina de estados e, consequentemente, entre a operação do AC e o processador. Embora uma situação deste género não tenha sido comprovada, foram tomadas, desde início, as precauções necessárias. O tempo mínimo para sinalizar um comando vindo do processador à máquina de estados é o período equivalente a um ciclo de relógio. Assim, quando um bit de controlo do registo um é colocado a 1, pretende-se dar um impulso, com a duração de um ciclo de relógio, à máquina de estados. Para este efeito, usou-se o circuito auxiliar apresentado na figura 4.8 para cada bit de controlo do registo um com exceção do bit 26 que ativa ou desativa o modo passo-a-passo, pois pretende-se que o sistema esteja continuamente neste modo entre múltiplas iterações do AC, pelo que não deve ser aplicado um impulso de duração de um ciclo neste bit de comando.

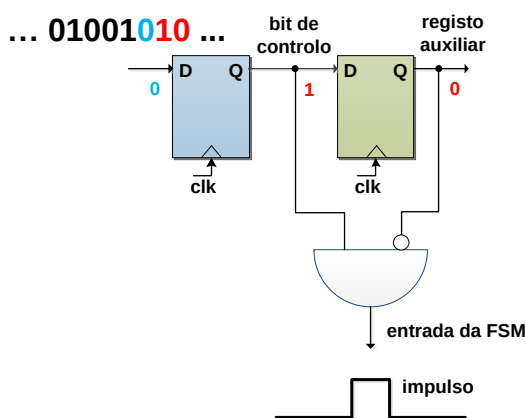


Figura 4.8: Circuito para aplicar um impulso com duração exata de um ciclo de relógio.

Na figura 4.8, o registo a azul corresponde a um dos bits de controlo do registo um da tabela 4.6 e o registo a verde é um registo auxiliar. Repare-se na sequência de bits apresentados à entrada do registo de controlo. Sabendo que quando o sistema de hardware arranca estes registos são inicializados a zero e entra num estado inicial de espera até que um comando seja recebido;

suponha-se que o bit 1 a vermelho representa um comando para dar início ao processo de inicialização do AC. No ciclo de relógio de instante  $t$  o bit do registo de controlo é colocado a 1 e, nesse mesmo instante, a máquina de estados recebe de imediato a ordem para a inicialização do AC. No ciclo de relógio  $t + 1$ , a máquina de estados transita para o estado de leitura da memória de entrada e mesmo que o bit do registo de controlo não seja repostado a 0, a porta lógica que recebe como entradas o bit de controlo e o bit da saída do registo auxiliar negado, faz com que no ciclo de relógio  $t$  a saída da porta lógica esteja a 1 e no ciclo de relógio  $t + 1$  a sua saída já esteja a zero, aplicando um impulso com a duração exata de um ciclo de relógio. Note-se que para aplicar um novo impulso, após o bit do registo de controlo ser colocado a 1 este deve ser colocado a 0. Se o bit do registo de controlo se mantivesse, após a transição, a 1, a saída da porta lógica estaria também sempre a 1 nos ciclos de relógio seguintes e nunca mais seria possível gerar um novo impulso. Significa assim que a sequência de entrada no circuito deve ser sempre, em primeiro lugar, uma transição de  $0 \rightarrow 1$  e, em segundo lugar, uma transição de  $1 \rightarrow 0$ .

## 4.5 Implementação em MicroBlaze

Na secção 4.4 explicou-se como foi implementada a máquina de estados de controlo do sistema de hardware, mostrou-se o fluxo de transição de estados e a interação realizada, do ponto de vista da máquina de estados, com o banco de registos de controlo e, por sua vez, com o processador MicroBlaze. Nesta secção, discute-se a implementação realizada no processador: leitura e escrita das memórias, envio e transferência de dados por comunicação série especificando o protocolo criado e os comandos suportados para a comunicação entre a FPGA e a aplicação de software no PC (*framework*) e o processamento realizado sobre os bits de informação de estado das células.

A implementação realizada no processador MicroBlaze foi feita com a linguagem de programação C. Esta implementação tem o suporte para o interpretador de comandos vindos da aplicação de software do PC, dados por um utilizador externo que especifica a arquitetura de hardware, que se traduz num protocolo de transferência e envio de dados sobre uma comunicação em série. O núcleo de hardware utilizado para a comunicação em série implementa um controlador UART. Este núcleo encontra-se também ligado ao barramento PLB e é controlado pelo MicroBlaze. Além das BRAM acedidas pelo MicroBlaze como memórias de dados e instruções, é utilizado um núcleo de hardware, também ligado ao barramento PLB e controlado pelo MicroBlaze, que implementa um controlador para o codec de áudio AC'97. A saída de áudio disponível é usada para efeitos de demonstração de uma aplicação prática dos ACs, reproduzindo uma melodia associada a um determinado estado ao AC. Na subsecção 4.5.1 é explicado o algoritmo implementado que permite reproduzir o estado de um AC sonoramente em função da informação de estado das células.

O protocolo implementado para a leitura e receção de dados é orientado ao byte e, dada a sua simplicidade, não são suportados quaisquer tipo de mecanismos de deteção e recuperação de erros. Dado que não havia a necessidade de ter uma comunicação 100% fiável e de alto débito, optou-se por implementar um protocolo o mais simples possível sobre um meio físico de comunicação simples também. O mecanismo base do protocolo assenta na receção de um comando enviado

pela aplicação de software no PC (1 byte, ASCII), interpretá-lo e, caso seja um comando válido, processar o pedido correspondente. Após a validação, são enviados ou recebidos mais  $n$  bytes, uma contagem conhecida à *priori* consoante o comando recebido e, no fim, o processador envia um último byte de confirmação a informar que todos os bytes foram recebidos ou enviados. Se, por algum motivo de falha de comunicação, no caso da receção, não sejam recebidos todos os  $n$  bytes esperados, o processador fica à espera até que os bytes que faltam para concluir o pedido feito sejam recebidos.

O número de bytes,  $n$ , a receber é uma função das dimensões do AC especificado nos processos de inicialização e leitura do AC. Relembrando o que foi explicado na secção 4.3, sendo  $c$  o número de colunas,  $r$  o número de linhas do AC e  $b$  o número de bits por célula, o número de colunas  $c$ , ou a largura do AC, deve ser sempre um múltiplo de 8. Sabendo que a expressão  $(c \times b)$  é igual ao número de bits de estado de cada linha do AC,  $(c \times b)/8$  é o número de bytes. Assim, a expressão  $n = (c \times b \times r)/8$  traduz-se no número de bytes a transmitir ou a receber. O facto de o AC ser escrito e lido da memória byte a byte, do lado do núcleo do AC, em conjunto com a utilização de uma comunicação série orientada ao byte não é coincidência mas sim uma conveniência que foi tida em conta aquando da especificação da arquitetura de hardware, da mesma forma em que o conteúdo de cada byte transporta os bits de estado de cada célula, agrupados de  $b$  em  $b$  bits em série, de tal forma que cada  $b$  bits correspondam exactamente aos bits de estado de células adjacentes na matriz do AC. Assim, se colocássemos todos os bytes armazenados de memória em série, a partir do endereço 0 sendo este byte o menos significativo, é possível ler o conteúdo de células distintas  $b$  a  $b$  bits.

A lista de comandos suportados pelo protocolo implementado é apresentada na tabela 4.7. A maior parte destes comandos tem interação com o banco de registos de controlo, colocando a 1 o bit de controlo correspondente, e de seguida a 0, ao comando recebido. Todas as alterações realizadas ao estado dos registos são indicadas na tabela (consultar tabela 4.6 para ver o significado dos bits de cada registo de controlo).

| Identificador ASCII | Bits de controlo    | $n$ bytes seguintes | Descrição                                                        |
|---------------------|---------------------|---------------------|------------------------------------------------------------------|
| R                   | Bit 31 do Registo 1 | 0                   | Inicializar a matriz do AC com o conteúdo da memória de entrada. |

| Identificador ASCII | Bits de controlo         | $n$ bytes seguintes       | Descrição                                                                                                                                                                                                                                                                                                                                                      |
|---------------------|--------------------------|---------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| U                   | Bit 31 do Registo 1      | $(c \times b \times r)/8$ | Receber $n$ bytes relativos à informação de estado das células do AC, escrever na memória o conteúdo recebido e dar o comando de inicialização da matriz do AC a partir da leitura da memória de entrada. Assim que a máquina de estados de controlo do sistema de hardware responde que a inicialização do AC foi concluída, é enviado o byte de confirmação. |
| B                   | Bit 30 do Registo 1      | 0                         | Dar início à operação do AC.                                                                                                                                                                                                                                                                                                                                   |
| G                   | Bits [0:31] do Registo 0 | 4                         | Receber 4 bytes que correspondem aos 32 bits que constituem o número total de iterações que o AC deve realizar e escrevê-los no registo indicado.                                                                                                                                                                                                              |
| A                   | Bit 26 do Registo 1      | 1                         | Receber 1 byte que indica se o modo passo-a-passo é ativado ou desativado. O byte recebido é igual a 1 se é ativado e igual a 0 se é desativado. Esta informação é guardada numa variável no código C para tomada de decisões.                                                                                                                                 |
| L                   | -                        | $(c \times b \times r)/8$ | Enviar $n$ bytes relativos à informação de estado das células presente na memória de entrada (recuperação de um estado previamente carregado).                                                                                                                                                                                                                 |

| Identificador ASCII | Bits de controlo    | $n$ bytes seguintes | Descrição                                                                                                                                                                                                                                                             |
|---------------------|---------------------|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Z                   | -                   | 0                   | Enviar apenas um byte de confirmação (código ASCII da letra O) para informar a aplicação de software que o processador se encontra a correr.                                                                                                                          |
| D                   | -                   | 1                   | Receber 1 byte que indica se o modo de demonstração de reprodução sonora do estado do AC é ativado ou desativado. O byte recebido é igual a 1 se é ativado e igual a 0 se é desativado. Esta informação é guardada numa variável no código C para tomada de decisões. |
| N                   | Bit 27 do Registo 1 | 0                   | Se o sistema estiver no modo passo-a-passo e o AC já tiver sido iniciado, a receção deste comando indica ao processador para sinalizar a máquina de estados que deve ser calculado o próximo estado do AC, isto é, realizar mais uma iteração.                        |
| S                   | Bit 29 do Registo 1 | 0                   | Se o sistema estiver no modo passo-a-passo e o AC já tiver sido iniciado, a receção deste comando indica ao processador para sinalizar a máquina de estados que o AC deve ser parado.                                                                                 |

Tabela 4.7: Comandos suportados pelo protocolo implementado para a comunicação série.

De modo a sincronizar o MicroBlaze com a máquina de estados de controlo do sistema de hardware, é implementada, também, no processador uma máquina de estados de apenas dois estados: um estado de repouso e um estado de execução do AC. Quando o MicroBlaze está no estado de repouso, dos comandos suportados pelo protocolo apresentados na tabela 4.7, apenas os comandos de identificador **R**, **U**, **B**, **G**, **A**, **L**, **Z** e **D** são admitidos. Os restantes comandos, de identificadores **N** e **S**, só são válidos depois de o AC ser iniciado e se o sistema se encontrar no modo passo-a-passo.

Quando o processador arranca, ou seja, quando o sistema de hardware é iniciado, as características do AC são lidas do registo do controlo 2 (ver tabela 4.6) e guardadas em variáveis no código C para realizar cálculos, e transita-se para um estado de repouso. Quando é recebido o comando de identificador **U**, é executado o excerto de código apresentado na listagem 4.7. Cada byte recebido é agrupado numa variável de 32 bits, *mem\_data*, que representa o conteúdo a ser escrito em memória. Assim que são recebidos 4 bytes, *mem\_data* é escrito em memória. Este processo repete-se até que o ciclo *while* termine, ou seja, que os  $n = (c \times b \times r)/8$  bytes sejam recebidos. Após o fim do ciclo, na linha 16, é realizada uma última escrita em memória para o caso em que  $n$  não é um múltiplo de 4, pois nessa situação a condição da linha 9 não é verificada para os últimos 3 ou menos bytes. Por outro lado, quando o comando de identificador **L** é recebido, é realizado o processo inverso, isto é, são lidos da memória 32 bits de dados que são enviados pela porta série em blocos de 8 bits de cada vez. Assim que são enviados 4 bytes, são lidos da memória os 4 bytes seguintes e assim sucessivamente. O ciclo *while* de envio termina assim que são enviados, também,  $n = (c \times b \times r)/8$  bytes. O excerto de código, para o processo de envio, é apresentado na listagem 4.8.

Listing 4.7: Código para receção da informação de estado das células e escrita em memória.

```

1 mem32_addr = 0;
2 byte_count = 0;
3 mem_data = 0;
4 count = ca_bytes_count;
5 while (count--)
6 {
7     serial_recv(&serial_ptr, &serial_data, 1);
8     mem_data |= serial_data << (8 * byte_count++);
9     if (byte_count == 4)
10    {
11        byte_count = 0;
12        write_input_mem(mem_data, mem32_addr++);
13        mem_data = 0;
14    }
15 }
16 write_input_mem(mem_data, mem32_addr);

```

Listing 4.8: Código para envio da informação de estado das células e leitura da memória.

```

1 mem32_addr = 0;
2 byte_count = 0;
3 mem_data = 0;
4 count = ca_bytes_count;
5 while (count--)
6 {
7     serial_data = (read_input_mem(mem32_addr) >> 8 * byte_count++)
8         & 0x000000FF;
9     if (byte_count == 4)
10    {
11        mem32_addr++;
12        byte_count = 0;
13    }
14    serial_send(&serial_ptr, &serial_data, 1);
15    while (XUartLite_IsSending(&serial_ptr)) { ; }
16 }

```

Relembre-se da secção 4.3 que o AC é lido linha a linha, deslocando-as na matriz de cima para baixo, e escritas byte a byte na memória de saída. Da mesma forma, no processo de inicialização do AC, a memória de entrada é lida byte a byte, constrói-se a nova linha no SR e esta é carregada no topo da matriz, deslocando-a também de cima para baixo e assim sucessivamente. Isto significa que, tanto para a inicialização como para a leitura, os dados em memória estão organizados de tal forma que o byte localizado no endereço 0, considerando endereçamento ao byte, corresponde a bits de estado de células localizadas no canto inferior direito da matriz do AC e os bytes nos endereços seguintes correspondem a bits de estados de células localizadas na direção do canto inferior esquerdo da matriz do AC. Considere-se que a linha do topo da matriz e que a coluna mais à esquerda é de índice 0 e que a matriz tem  $r$  linhas,  $c$  colunas e  $b$  bits por célula. Sabendo que cada linha tem  $r_b = (c \times b)/8$  bytes, os primeiros  $r_b$  bytes da memória, a começar no endereço 0, correspondem aos bits de estado da linha de índice  $r - 1$ . Os  $r_b$  bytes seguintes, a partir do endereço de memória  $r_b$ , correspondem aos bits de estado da linha de índice  $r - 2$ . Os  $r_b$  bytes localizados no endereço  $2 \cdot r_b$  correspondem à linha de índice  $r - 3$  e assim sucessivamente. Quer assim dizer que os dados estão organizados, do ponto de vista das memórias, de tal forma a que a matriz é lida em memória de baixo para cima e da esquerda para a direita, permitindo a inicialização e leitura da matriz do AC como é explicado na secção 4.3. Além disso, os bytes devem ser recebidos nesta mesma sequência. Repare-se na linha 8 da listagem 4.7. Cada byte recebido é colocado em *mem\_data* da direita para a esquerda e o mesmo se aplica no processo de leitura, de um bloco de 32 bits cada 8 são enviados dos bits menos para os mais significativos, ou seja, da direita para a esquerda (ver linha 7 da listagem 4.8) mantendo assim a sequência necessária para a inicialização e leitura do AC.

Quando o AC é iniciado, o processador transita para o estado de execução. Neste estado, os bits [28:30] (o bit 31 só é relevante quando é dado o comando de inicialização do AC) do registo de controlo 2 (ver tabela 4.6) são vigiados e enquanto não houver uma resposta por parte da máquina de estados, o processador é retido num ciclo à espera de uma. Como o AC é lido linha a linha, o bit 28 é o primeiro a ser colocado a 1, indicando que uma linha do AC encontra-se na memória de saída. O conteúdo dessa linha é lido da memória e armazenado num *buffer* de bytes sobre qual pode ser realizado qualquer tipo de processamento. Neste caso, como se pretende, para efeitos de demonstração, reproduzir sonoramente o estado atual do AC, é chamada uma função que implementa um algoritmo que realiza cálculos sobre os bits de estado das células dessa linha, lendo a informação armazenada no *buffer*, que determina um índice para um vetor de frequências, que se traduz numa nota musical, e que é armazenado num vetor de índices. Este algoritmo é explicado detalhadamente na subsecção 4.5.1. Após o processamento, o *buffer* é enviado, byte a byte, pela porta série para a aplicação de software. O processador coloca a 1 o bit 28 do registo de controlo 1, e de seguida a 0, e regressa ao ciclo de espera de observação dos bits de controlo do registo 2. O processo repete-se até que todas as linhas sejam lidas, processadas e enviadas. O fim de uma iteração do AC traduz-se na leitura de todas as linhas da matriz, e ao envio de  $(c \times b)/8$  bytes, sinalizando a máquina de estados tal situação. Nesta situação, se o modo de demonstração de reprodução sonora estiver ativo, o vetor de índices é lido e cada índice é usado para consultar um vetor de frequências para gerar uma onda, com a frequência obtida, na saída. Se a máquina de estados tiver sinalizado também que o número máximo de iterações foi atingido, é enviado para a aplicação de software um byte de finalização de identificador **E** e o processador regressa ao estado inicial de repouso, caso contrário é enviado o byte de identificador **M**. Nesta situação, se o sistema estiver no modo passo-a-passo, o processador é retido num ciclo em que espera por um comando para prosseguir a execução do AC, isto é, calcular a próxima iteração ou então um comando de paragem, caso contrário o bit 27 é colocado a 1, e de seguida a 0, e a próxima iteração é calculada de imediato.

A listagem 4.9 apresenta o código que é executado aquando da leitura de uma linha da matriz do AC da memória de saída.

Listing 4.9: Código para a leitura de uma linha da matriz do AC da memória de saída.

```

1
2 count = 0;
3 ca_row_bytes_count = (ca_width * bits_per_cell) >> 3;
4
5 while (count < ca_row_bytes_count)
6 {
7     ca_row_data[count] = read_output_mem8(mem8_addr + endianness_ctrl);
8     count++;
9     endianness_ctrl--;
10    if (endianness_ctrl == -1)

```

```

11  {
12      endianness_ctrl = 3;
13      mem8_addr += 4;
14  }
15  }

```

O ciclo *while* é executado até que todos os  $(c \times b)/8$  bytes de uma linha tenham sido lidos e uma vez que as linhas vão sendo escritas em memória, isto é, não se encontra em memória apenas uma linha (exceto se o AC for unidimensional), o contador de endereço de memória *mem8\_addr* não é reinicializado entre execuções do ciclo *while*. Em cada passagem do ciclo *while*, é lido um byte da memória de saída e guardado no *buffer* designado por *ca\_row\_data*, que, ao fim da última passagem pelo ciclo, contém os bytes correspondentes a uma linha do AC. Uma vez que a arquitetura do MicroBlaze é do tipo *big endian*, é necessário fazer uma compensação ao incremento de endereço. Desta forma, a variável *endianness\_ctrl*, inicializada a 3 e o seu valor é mantido entre execuções consecutivas do ciclo *while*, é o fator de compensação do contador de endereço, que o aponta para o quarto byte de um conjunto de 32 bits da memória e decrementa-o sucessivamente. O fator de compensação é reinicializado quando se verifica a condição da linha de código 10 e avança-se para os 4 bytes seguintes da memória. O contador de endereço *mem8\_addr* é reinicializado aquando do início de uma nova simulação do AC.

#### 4.5.1 Algoritmo para a reprodução sonora do estado do AC

O algoritmo para a reprodução sonora de um determinado estado do AC assenta no cálculo da densidade de bits de estado das células de uma linha da matriz do AC, isto é, o determinar a razão  $D_b$  entre o número de bits de estado com o valor lógico 1 e o número total de bits presentes numa linha. A razão obtida é convertida num valor entre 0 e 87, um índice para indexar um vetor de 88 frequências que correspondem às 88 teclas de um piano típico na escala igualmente temperada de 12 tons por oitava. Assim, quanto maior for  $D_b$  mais aguda é a nota que se toca. Cada linha da matriz do AC corresponde a uma nota a tocar, pelo que o vetor de índices tem comprimento  $r$ , o número total de linhas da matriz do AC. Por fim, para cada frequência, com uma duração fixa de 500 ms, é gerada uma onda quadrada.

Depois de ser efetuada a leitura de uma linha da matriz do AC para um *buffer* auxiliar, o código apresentado na listagem 4.10 é executado.

Listing 4.10: Código da chamada à função que determina a frequência da nota

```

1  if (music_demo == 1)
2  {
3      ca_row_frequency_index[frequency_index++] =
4          get_frequency_index(ca_row_data, ca_row_bytes_count);
5  }

```

A função *get\_frequency\_index*, que recebe como parâmetros o *buffer* cujo conteúdo são os bits de estado das células de uma linha da matriz do AC e o seu comprimento, e determina o índice do vetor de frequências, em que cada índice é guardado no vetor *ca\_row\_frequency\_index*. A listagem 4.11 mostra o código que calcula o índice a partir da densidade de bits de uma linha da matriz do AC.

Listing 4.11: Código da função que implementa o algoritmo para determinar a frequência da nota

```

1  unsigned int get_frequency_index(unsigned char * ca_row ,
2      unsigned int size)
3  {
4      unsigned int i;
5      unsigned int bit_count = 0;
6      unsigned int byte_shift = 7;
7      unsigned int index;
8
9      for (i = 0; i < size; i++)
10     {
11         while (byte_shift)
12         {
13             bit_count += (ca_row[i] >> byte_shift) & 0x01;
14             byte_shift--;
15         }
16         byte_shift = 7;
17     }
18
19     index = (bit_count * 1.75 * PIANO_KEYS) / (size * 8);
20
21     if (index >= PIANO_KEYS)
22         return PIANO_KEYS - 1;
23     else
24         return index;
25 }
```

Na linha de código 19 é determinada a densidade de bits  $D_b$  e a *macro* PIANO\_KEYS é o número de frequências total a utilizar que é igual a 88, que permite converter  $D_b$  num valor entre 0 e 88. A constante 1.75 é um fator de ganho necessário para compensar de forma relativa a não linearidade da escala igualmente temperada, pois se  $D_b = 50\%$  não corresponde a metade da frequência máxima da escala (4186.01 Hz), mas sim a uma frequência inferior. Esta compensação é relevante quando se geram ACs cujas regras de transição de estado fazem com que o estados das células se anulem rapidamente ao fim de algumas iterações, ou com que os estados não nulos sejam bastante dispersos na matriz. Dado o ganho aplicado, é necessário fazer a verificação da

linha de código 21 para garantir que não sejam calculados índices iguais ou superior a 88. A função *get\_frequency\_index* pode ser facilmente modificada para implementar qualquer outro tipo de algoritmo. Conclui-se rapidamente que este algoritmo é transparente em relação à posição horizontal das células ao longo de uma linha, isto é, o facto de uma célula estar posicionada numa coluna mais à esquerda ou mais à direita é indiferente pelo que  $D_b$  mantém-se igual. A aplicação de pesos nos estados das células em função da sua posição horizontal dado o índice da coluna da matriz do AC de uma determinada célula, pode ser uma opção a considerar para implementar um algoritmo de maior complexidade.

Para gerar uma onda quadrada na saída, é necessário calcular o número de *frames* de audio a enviar para o codec de forma a gerar a onda com duração de 500 ms e o de quantas em quantas *frames* é que a saída deve ser comutada. A listagem 4.12 mostra o código que permite gerar a onda quadrada. Sabendo que a frequência de uma *frame* é de 48kHz, o número de *frames* a enviar, *frames\_to\_send*, é dado por  $\frac{48 \cdot 10^3}{1 \cdot 10^3} \times 500 = 24000$ . O PCM é colocado com o valor dado pela variável *pcm\_data* e a saída deve ser comutada em cada  $\frac{48 \cdot 10^3}{n \cdot f}$  *frames*, em que  $f$  é a frequência do vetor de frequências *piano\_frequencies* indexado pelo vetor de índices *ca\_row\_frequency\_index* e  $n$  o número de níveis da onda. Uma vez se está a gerar uma onda quadrada, toma-se  $n = 2$  para comutar a saída à frequência  $2 \cdot f$ , ou então a metade do período da onda. O código é facilmente adaptado para gerar uma onda de maior complexidade, como por exemplo uma onda em dente de serra, que permita tirar proveito da resolução de 20 bits do PCM.

Listing 4.12: Código para gerar uma onda quadrada

```

1 count_skip_frames = 0; pcm_data = 0x0001FFFF; i = 0; j = 0;
2 while (j < ca_height) {
3     frames_to_skip = sample_rate /
4         ( 2 * piano_frequencies[ ca_row_frequency_index[j] ] ) ;
5     while (i < frames_to_send) {
6         AC97_Wait_For_New_Frame(XPAR_AC97_IF_0_BASEADDR);
7         send_audio_frame(pcm_data , BOTH_CHANNELS);
8         count_skip_frames++;
9         if (count_skip_frames >= frames_to_skip) {
10             pcm_data = ~pcm_data; count_skip_frames = 0;
11         }
12         i++; }
13     i = 0; j++; }
```

## 4.6 Soluções de arquitetura alternativas e considerações finais

Ao longo deste capítulo foi descrita a arquitetura de cada módulo do sistema de controlo de hardware, o seu funcionamento e a sua interação com os outros módulos. As soluções apresentadas para a arquitetura de cada célula, interação intercelular (tipo de vizinhança), os processos

de inicialização e leitura do estado do AC, controlo de condições fronteira e transferência de dados para a aplicação de software no PC por um canal físico de comunicação, traduzem-se na arquitetura global do sistema de controlo de hardware que se encontra atualmente implementada e 100% funcional com todas as características que tem vindo a ser detalhadas. Contudo, ao longo do desenvolvimento deste trabalho foram estudadas soluções de arquitetura alternativas para a implementação dos subsistemas de hardware que concretizem as funcionalidades impostas pelos requisitos do sistema.

A arquitetura de hardware implementada permite gerar ACs unidimensionais e bidimensionais em que em cada ciclo de relógio é calculado o estado seguinte de cada célula. Para este tipo de ACs, houve a preocupação da possibilidade de especificar, uma de entre pelo menos duas hipóteses, características essenciais de modelos de ACs genéricos, isto é, o tipo de vizinhança e a condição fronteira, de forma a oferecer flexibilidade na criação de especificações dedicadas a aplicações distintas. No caso dos ACs bidimensionais, os tipos de vizinhança de Moore, de conectividade 8, e de von Neumann, de conectividade 4, são os mais abrangentes e permitem referenciar a vizinhança circundante, a uma distância  $r$ , em torno de uma célula (ver subsecções 2.1.2 e 2.1.1, respetivamente). Na arquitetura desenvolvida,  $r = 1$  pelo que a complexidade é a menor possível e tem-se apenas acesso à vizinhança imediata de uma célula, isto é, as células adjacentes. Para os ACs unidimensionais, a conectividade é apenas de 2 e usou-se também  $r = 1$ . A condição fronteira permite definir o comportamento das células que se encontra nos extremos de uma matriz pelo que também é uma característica de um AC genérico. Assim, procurou-se suportar uma condição fronteira que interligasse as células dos extremos umas com as outras, fechando a matriz do AC num toróide, no caso de um AC bidimensional e num anel no caso de um AC unidimensional, ou um cilindro dada a sua representação temporal em duas dimensões. A outra opção tida em conta era a possibilidade de gerar células fictícias de estado nulo ao longo das fronteiras da matriz. Desta forma, é possível criar, além da seleção de um entre dois tipos de vizinhança, dois cenários distintos para o comportamento das células do extremo da matriz. A especificação do número de bits por célula permite abordar ACs mais complexos com a resolução necessária para a sua simulação. Estas foram as características e possibilidades de especificação consideradas aquando da implementação do sistema, de modo a permitir gerar diversos ACs e abranger diversas áreas de aplicação. O grau de complexidade introduzido no AC não depende de qualquer aspeto da arquitetura de hardware mas sim da especificação fornecida, mas depende do número de bits por célula utilizados e da complexidade da regra de transição de estado. Assim, o grau máximo de complexidade é imposto pelo limite de recursos que a FPGA disponibiliza para gerar a arquitetura de hardware para uma determinada especificação.

Inicialmente, considerou-se uma arquitetura em que o estado seguinte é determinado ao fim de  $n$  ciclos de relógio ao contrário do que acontece com a arquitetura atual. A ideia seria de especificar pequenos núcleos de processamento idênticos, a funcionar em paralelo, em que cada um contém um subconjunto de células do AC completo. Na presente arquitetura, cada célula representa uma pequena máquina de estados com um circuito combinatório e outro sequencial para determinar o estado seguinte da célula e guardá-lo em registos, respetivamente. Para cada núcleo

de processamento, em vez de colocar a lógica que define a regra de transição de estado em cada célula, há um módulo de lógica dedicado que é multiplexado entre o subconjunto de células do núcleo que receberia o estado de cada célula e dos seus vizinhos e na saída colocaria o estado seguinte da célula respetiva, guardando-o em registos. Após o cálculo do estado seguinte, este é escrito numa memória dedicada pertencente ao núcleo, onde ficam guardados os estados de todas as células do subconjunto do núcleo. A memória é dividida em dois segmentos distintos, um para leitura e outro para escrita, de forma a se poder realizar simultaneamente leituras e escritas evitando colisões e perdas de dados. O segmento de escrita diz respeito ao cálculo do estado seguinte e o segmento de leitura ao estado atual, pelo que os seus papéis são depois invertidos. Os núcleos de processamento são replicados na FPGA e interligados uns aos outros de forma a que as células possam partilhar os seus estados com os vizinhos. Com uma arquitetura deste género, à partida, tira-se mais proveito dos recursos da FPGA, uma vez os estados das células são guardados em memória BRAM e não em registos e a lógica que define a regra de transição de estado é partilhada por um subconjunto de células e não está presente em cada módulo da própria célula, com o custo da latência de  $n$  ciclos de relógio para determinar a iteração seguinte do AC pois cada leitura e escrita efetuada em memória demora 1 ciclo de relógio. Além do controlo necessário para propagar os estados das células de uns núcleos de processamento para os outros, leituras e escritas em memória para salvaguardar os estados atuais das células e o estado seguinte, é também necessário controlo adicional para o tratamento das condições fronteira. Na arquitetura que foi proposta, as fronteiras são ligadas diretamente umas às outras por meio de dois barramentos que transportam os bits de estado das células posicionadas ao longo das fronteiras em cada sentido. A proposta alternativa é de ter uma memória auxiliar, partilhada pelos núcleos de processamento que se encontram na fronteira, em que estejam guardados os estados das células da fronteira para serem lidos, na iteração  $t$ , para o cálculo do estado seguinte. Na iteração  $t - 1$  a memória auxiliar é escrita com o estado seguinte das células da fronteira, que se encontram armazenados na memória local do núcleo de processamento, para que na iteração  $t$  já esteja disponível o estado seguinte. Da mesma forma que as memórias dos núcleos, para evitar perda de dados entre leituras e escritas, pode-se usar segmentos distintos de memória alternadamente para a escrita e leitura. Cada segmento de memória corresponde ao acesso de diferentes núcleos da fronteira. Apesar desta arquitetura alternativa parecer atrativa na medida em que se pode poupar e aproveitar melhor os recursos da FPGA, a complexidade de implementação para gerar ACs de dimensões diversas é elevada. De qualquer forma, do ponto de vista do tempo de execução para realizar um número elevado de iterações, esta solução, dada a latência imposta pelos acessos a memória, não é a mais viável, embora, do ponto de vista das dimensões do AC, dada a configuração dos núcleos de processamento, seja possível gerar ACs de maior dimensão do que na arquitetura que foi proposta.

O processo de inicializar e ler o estado do AC foi estudado cuidadosamente, tendo em conta que os dados deveriam ser recebidos e enviados por comunicação série RS-232. A introdução do MicroBlaze no sistema foi apenas uma opção de conveniência, uma vez que a complexidade de implementar o protocolo de transmissão de dados especificado é menor utilizando a linguagem de programação C do que Verilog HDL. Além disso, se o MicroBlaze não fosse incluído seria

necessário escrever o módulo do controlador UART pois o módulo fornecido tem apenas interface com o barramento PLB, mas as conversões de dados realizadas devido ao facto de se usar portas de acesso a memória com larguras diferentes seriam evitadas. Posteriormente, com a decisão de incluir uma demonstração de uma aplicação prática dos ACs reproduzindo sonoramente o seu estado, o MicroBlaze releva-se extremamente útil para realizar processamento adicional sobre os bits de estado das células. Da mesma forma, o controlador do AC'97 teria que ser adaptado pois a sua interface também é com o barramento PLB. Para os *shift-registers* (SR) que acedem às memórias para leitura e escrita, considerou-se inicialmente que o seu comprimento seria fixo mas endereçável, isto é, o número de andares dos SRs a usar seria uma fração do total de andares disponíveis. Contudo, uma vez que a especificação de um AC consiste numa arquitetura dedicada para a simulação desse mesmo AC, procurou-se uma solução de forma a que, tal como se gera as dimensões exatas pretendidas do AC e não um AC de dimensão fixa em que se usa apenas uma fração da sua dimensão total, o número de andares dos SRs fosse o número de andares necessários para a leitura de uma linha da matriz do AC e que a sua arquitetura permitisse ao mesmo tempo escrever e ler dados das memórias sem qualquer módulo de suporte adicional. Dada a arquitetura de cada célula, do estilo máquina de estados, e para reduzir a sua complexidade o máximo possível, optou-se, para a leitura e inicialização da matriz do AC, por deslocar a matriz linha a linha de cima para baixo lendo o estado de várias células em paralelo, com controlo adicional para que cada linha que fosse lida pudesse ser facilmente reposta no AC sem alterar o seu estado após a leitura de todas as linhas. Considerou-se, em vez da solução proposta, em interligar as fronteiras da matriz de modo em que todas as células ficassem ligadas em série e o AC seria deslocado e lido célula a célula. Esta alternativa foi abandonada pois introduz lógica complexa no controlo dos barramentos das fronteiras, além da lógica necessária para aplicar a condição fronteira. Além disso, em linhas alternadas da matriz, as células teriam que receber o estado do vizinho da direita e o da esquerda pelo que seria necessário contabilizar com o índice par e ímpar da linha onde as células se encontram. Esta solução introduz também uma maior latência do que a solução atual uma vez que, considerando  $c$  o número de colunas da matriz e  $b$  o número de bits por célula, por cada ciclo de relógio são disponibilizados apenas  $b$  bits de estado, ou seja, o equivalente a uma célula contra  $(c \times b)$  bits de estado. Seria necessário também um mecanismo semelhante ao que foi implementado com os SRs para a leitura e escrita de dados em memória. A decisão de usar dois blocos distintos de memória deve-se apenas ao facto de separar o que são dados de inicialização e dados de leitura e também para evitar o acesso a segmentos distintos de memória, no mesmo espaço de endereçamento, desnecessariamente.

Embora a possibilidade de desenvolver uma arquitetura de hardware capaz de gerar ACs de três dimensões não tivesse sido posta de parte, dada a sua complexidade, optou-se apenas pelos ACs unidimensionais e bidimensionais. Ainda assim, procurou-se desenvolver uma arquitetura de hardware o mais flexível possível de maneira a sua expansão não tivesse um custo de implementação adicional demasiado elevado. Desta forma, o passo a dar de uma arquitetura que gere ACs bidimensionais para uma que gere ACs tridimensionais não é demasiado grande.

A expansão para a geração de ACs tridimensionais consiste, tendo em conta a solução de

arquitetura atual, em avaliar quais os recursos adicionais necessários para o efeito. A expansão passa pela replicação do módulo de lógica do AC para a terceira dimensão e alargamento dos barramentos de forma a interligar os módulos de lógica adicionais: suporte das condições fronteira e da vizinhança das células ao longo da terceira dimensão. Para um AC de três dimensões, em torno de uma célula central, o número de células vizinhas é de  $v_{Neumann} = 6$  e  $v_{Moore} = 26$  para uma vizinhança do tipo de von Neumann e Moore, respetivamente, com  $r = 1$  (ver subsecções 2.1.2 e 2.1.1, respetivamente). Considerando agora uma estrutura tridimensional como um cubo ou um paralelepípedo, a expansão direta para a leitura e inicialização do AC seria ler os bits de estado de um conjunto de células situadas no mesmo plano, deslocando a estrutura plano a plano de cima para baixo. Contudo, a leitura de um plano de células obriga a replicar a lógica de controlo de transferência de dados, isto é, os multiplexadores, os barramentos e os SRs. Embora o único cuidado a ter seria reservar segmentos de memória para cada bloco bidimensional ao longo da terceira dimensão, quanto maior fosse a profundidade do AC maior o número de recursos utilizados para a transferência de dados. Apesar da replicação de recursos permitir ler  $p$  linhas em paralelo do AC, em que  $p$  é sua profundidade, totalizando  $(c \times p \times b)$  bits de estado num único ciclo de relógio, não é uma solução viável dada a elevada utilização de recursos da FPGA. Uma melhor solução passa por multiplexar os barramentos de saída e demultiplexar os barramentos de entrada, da fronteira sul e norte do AC, respetivamente, de cada bloco bidimensional de células ao longo da terceira dimensão. Assim, não há a necessidade de replicar a lógica de controlo de transferência de dados e a percentagem de recursos ocupados é menor.

## Capítulo 5

# Arquitetura do Sistema de Controlo de Software

No capítulo 4, foi feita a especificação da arquitetura do Sistema de Controlo de Hardware (SCH) descrevendo ao pormenor a arquitetura interna de cada módulo de hardware, a interação entre o processador MicroBlaze e o núcleo do AC através do registo de comandos e respostas, os processos de inicialização e leitura do AC e o protocolo de comunicação que permite transferir dados entre a plataforma de hardware e a aplicação de software no PC.

Neste capítulo, é feita a especificação da arquitetura do Sistema de Controlo de Software (SCS) onde se descreve com detalhe a implementação de software realizada para cada componente da aplicação e a sua funcionalidade. Recorde-se a figura 3.4, do capítulo 3, que apresenta um diagrama de arquitetura de alto nível que ilustra a divisão do SCS nos seus componentes constituintes, a interação entre eles e a interação direta do utilizador externo com estes. Na secção 5.1, é apresentada a interface gráfica da aplicação de software, aloca-se os elementos gráficos (menus, botões, ...) às funcionalidades que a aplicação oferece e descreve-se ambientes de trabalho da aplicação. São descritos também aspetos gerais de configuração e validação realizados pela aplicação. A secção 5.2 explica como é dividido o diretório de trabalho e sua organização, sobre o qual a aplicação realiza operações de leitura e escrita de ficheiros, descrevem-se as funções que o gestor de ficheiros implementa, o tipo e quais os ficheiros usados para configuração da aplicação e implementação em hardware. Na secção 5.3, descreve-se como é realizada a parametrização dos ficheiros dos módulos de hardware para a especificação de um AC, como é feita a especificação da arquitetura celular, indicando a regra ou função de transição de estado da célula e como esta especificação é integrada nos ficheiros de descrição dos módulos de hardware. A secção 5.4 descreve como é que são geridas as paletes de cor pela aplicação, que mapeiam os estados das células em cores distintas para representação gráfica, e o conteúdo dos ficheiros mapeadores. O visualizar gráfico do AC é descrito na secção 5.5, onde se explica como é que o utilizador interage com o AC, as funcionalidades gráficas disponíveis para o manuseamento do visualizador e como é que é armazenada e qual a informação relativa a cada célula para representação gráfica. Na secção 5.6, descreve-se como é que o utilizador pode gerir ficheiros de *bitstream*: gerar um ficheiro de

*bitstream* a partir de uma especificação de um AC existente, programar a FPGA a partir de um ficheiro de *bitstream* existente, quais os *scripts* criados, o seu conteúdo e execução, que chamam as ferramentas necessárias da Xilinx para os processos programação da FPGA e geração de um ficheiro de *bitstream*, de acordo com o *design flow*, e os ficheiros de configuração de software criados e associados a uma especificação de um AC. Por fim, a secção 5.7 explica como é realizada a comunicação com a plataforma de hardware para inicialização, simulação e leitura do AC. São revistos aspetos das funcionalidades associadas à arquitetura de hardware implementada e ao protocolo de comunicação especificado no capítulo 4 e explica-se como é realizado o processamento dos bits de estado das células para o envio e receção, a partir de e para uma representação gráfica do AC, respetivamente.

## 5.1 Interface gráfica, funcionalidades e ambientes de trabalho

A aplicação de software desenvolvida de suporte à plataforma de hardware tem uma interface gráfica, de modo a o utilizador poder interagir de forma intuitiva para configurar e aceder às funcionalidades disponíveis. O utilizador manipula diretamente a aplicação e obtém informação a partir dela através dos seus elementos gráficos que respondem a determinadas ações ou eventos e executam as funções por eles implementadas. Na figura 5.1 mostra-se a janela principal da interface gráfica do utilizador da aplicação, onde estão representadas seis áreas distintas, numeradas de um a seis, de agrupamento de elementos gráficos.

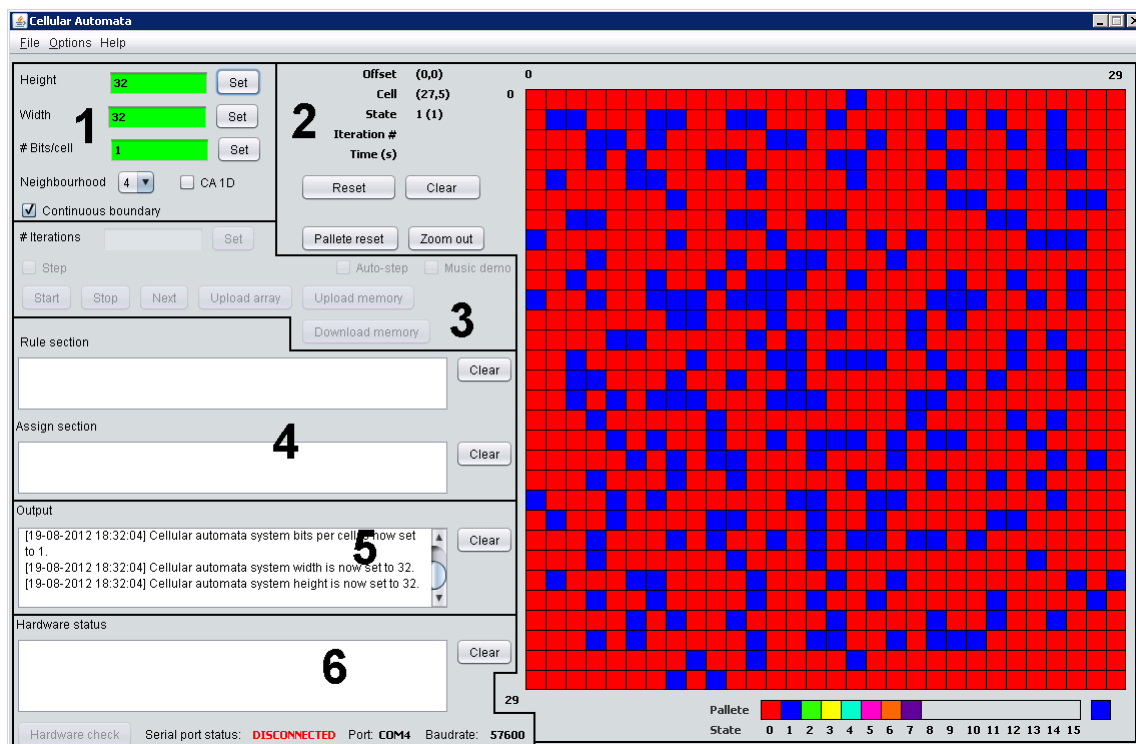


Figura 5.1: Janela principal da interface gráfica da aplicação

Na lista abaixo é descrita a alocação de funcionalidades às diferentes áreas representadas na figura. A lista é também numerada de um a seis para estabelecer a correspondência exata com a figura.

1. Esta área contém os elementos gráficos destinados à especificação das características de um AC. As dimensões do AC e o número de bits por célula são especificados ao introduzir os valores em três campos de texto. Os três botões intitulados de *set* fazem a validação dos valores introduzidos. O processo de validação verifica se os valores introduzidos são puramente numéricos e não nulos e se estão de acordo com as restrições da arquitetura de hardware, nomeadamente a largura do AC que deve ser um múltiplo de oito. Se os valores introduzidos forem corretamente validados, o fundo do campo de texto fica a verde caso contrário é lançado um erro numa janela diferente e o fundo fica a amarelo. O tipo de vizinhança do AC é especificado por uma caixa de seleção em que se pode selecionar uma vizinhança do tipo de Moore ou von Neumann, em que os valores da caixa de seleção são os números oito e quatro (células), respetivamente. Para ACs unidimensionais o único valor presente na caixa de seleção é o número dois (células). Existem também duas caixas de verificação que permitem definir a condição de fronteira, se é contínua ou nula, e indicar se o AC é unidimensional ou bidimensional. Se o AC for unidimensional, o campo de texto correspondente à altura do AC é colocado com o valor um e validado como correto de imediato, e a caixa de seleção do tipo de vizinhança é modificada incluindo apenas uma opção. Quando são especificadas as dimensões do AC, a sua representação gráfica surge no painel gráfico quadrado presente na área dois. Recorde-se a especificação da arquitetura de hardware do capítulo 4 que, apesar do modelo de AC unidimensional considerar apenas como vizinhança as células à esquerda e à direita de uma célula central, os ACs unidimensionais suportam uma vizinhança do tipo de von Neumann.
2. Área que contém o painel de visualização gráfica para inicialização e leitura do AC, a paleta de cores ativa com o mapeamento cor-estado da célula indicado, posicionada abaixo do visualizador gráfico e a cor selecionada à sua direita, indicação das coordenadas da célula atual e do seu estado na base decimal e binária, os limites do visualizador por etiquetas que indicam as coordenadas das células dos seus extremos e o *offset* de deslocamento da matriz do visualizador, em coordenadas, em relação à sua origem. Os botões intitulados de *reset* e *pallette reset* permitem colocar todas as células com o estado nulo e atribui-las a paleta de cores ativa, respetivamente. O botão intitulado de *clear* limpa o visualizador e ao mesmo tempo a especificação das dimensões do AC presentes na área um. O botão de *zoom out* fala por si mesmo.
3. Nesta área encontram-se os elementos gráficos que implementam o processo de inicialização, simulação e leitura do AC e ficam ativos após estabelecida uma comunicação com a plataforma de hardware. O número de iterações é especificado num campo de texto e, da mesma forma que as dimensões do AC e o número de bits de estado por células, o botão intitulado de *set* inicia o processo de validação idêntico, mais a verificação se o número de

iterações introduzido cabe em 32 bits, e caso seja um número válido este é enviado para a FPGA. Os restantes elementos gráficos, seis botões e três caixas de verificação, implementam as funcionalidades especificadas pelo protocolo de comunicação de dados do capítulo 4, secção 4.5. Relembrando, os botões intitulados de *upload memory* e *download memory* permitem inicializar a ler a memória de entrada, a partir de e para as células do visualizador gráfico do AC. O botão intitulado de *upload array*, além de inicializar a memória de entrada, carrega o conteúdo dela para a matriz do módulo de lógica do AC. As funcionalidades dos botões intitulados de *start* e *stop* são óbvias e a do botão *next* permite avançar para a próxima iteração no modo passo-a-passo. A caixa de verificação intitulada de *step* ativa/desativa o modo passo-a-passo e *auto-step* permite avançar para a próxima iteração automaticamente com um temporizador em software. Por fim, a caixa de verificação intitulada de *music demo* ativa/desativa o modo de demonstração para a reprodução sonora do estado do AC.

4. As caixas de texto desta área permitem descrever a arquitetura das células com a linguagem de descrição de hardware Verilog HDL e uma linguagem que permite, por uma questão de simplicidade, referenciar a vizinhança em torno de uma célula com identificadores específicos e argumentos, evitando assim a referência completa dos sinais que representam os barramentos de interligação com a vizinhança. A caixa de cima permite descrever a função para a atualização de estado das células (parte sequencial) e a de baixo, opcionalmente, para descrever circuitos combinatórios auxiliares. Os botões intitulados de *clear* limpam o conteúdo das caixas de texto respetivas. Mais detalhes sobre especificação da arquitetura celular e a linguagem de referência de vizinhança encontram-se na secção 5.3.
5. A caixa de texto presente nesta área é apenas um registo de mensagens de informação e de erro produzidas pela aplicação para orientação do utilizador.
6. A última área contém também uma caixa de texto para registo de mensagens produzidas aquando da transferência e receção de dados: os bytes transferidos, confirmações de receção por parte da FPGA e *timeouts*. Abaixo do registo de mensagens, é dada informação sobre o estado e parâmetros da comunicação série. O botão intitulado de hardware *check* permite enviar um comando para confirmar a atividade do mesmo.

Ainda na figura 5.1, na barra de ferramentas, no topo da janela da aplicação, estão presentes três menus diferentes: ficheiro, opções e ajuda. O conteúdo de cada menu é apresentado na figura 5.2.

O menu de ficheiro permite abrir e guardar ficheiros de descrição de arquitetura celular e guardar ficheiros de *log* do registo de mensagens da aplicação, isto é, guardar apenas o seu conteúdo no preciso momento. As opções de abrir e guardar uma descrição de arquitetura celular reencaminham o utilizador para o gestor de arquitetura de células. A partir deste menu é também possível fechar a aplicação.

O segundo menu, de opções, dá acesso ao utilizador às seguintes funcionalidades:

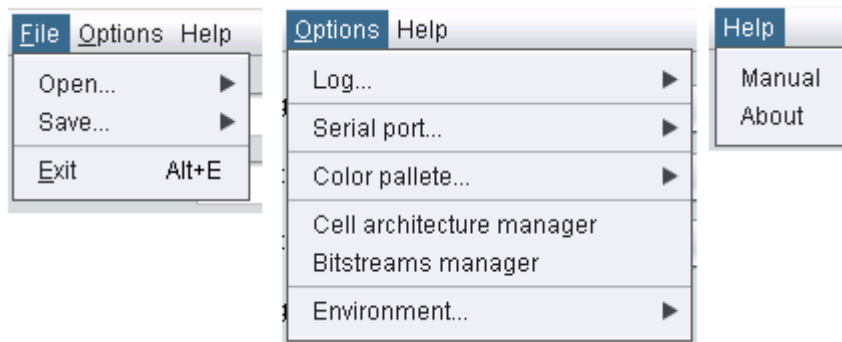


Figura 5.2: Conteúdo dos menus da barra de ferramentas

**Logs** O utilizador pode gravar o conteúdo que vai sendo escrito no registo de mensagens da aplicação para um ficheiro de texto simples criando um *log*. A gravação é contínua a partir do momento em que o utilizador dá a ordem de início de gravação até que este dê a ordem para terminá-la, ficando posteriormente o ficheiro disponível no diretório de *logs*.

**Porta série** Menu de configuração da comunicação série: escolher a porta COM presente no sistema, a velocidade de transmissão e iniciar/terminar a ligação.

**Gestor de paletes de cor** Este menu dá acesso ao gestor de paletes de cor que permite editar mapeamentos de cor-estado de célula existentes, criar e apagar paletes de cor e seleccionar uma paleta de cor existente para paleta ativa.

**Gestor de arquitetura de células** Menu de acesso ao gestor de arquitetura de células que permite abrir ou guardar descrições de arquitetura celular para conjuntos de células.

**Gestor de *bitstreams*** Menu de acesso ao gestor de *bitstreams* que permite gerar o ficheiro de *bitstream* para uma especificação corrente de um AC, programar a FPGA com um ficheiro de *bitstream* existente ou apagá-lo. Cada ficheiro de *bitstream* é acompanhado por um ficheiro de texto que descreve as características do AC que implementa. O caminho para o diretório de ferramentas de desenvolvimento da Xilinx é aqui indicado.

**Ambiente** Este menu permite seleccionar um de dois ambientes de trabalho: apenas simulação ou desenvolvimento e simulação. O ambiente de apenas simulação permite pegar em ficheiros de *bitstream* já existentes, programar a FPGA e simular o AC implementado. Por outro lado, o ambiente de desenvolvimento e simulação além de dar acesso às funcionalidades do ambiente de apenas de simulação, é possível também criar novas especificações de ACs e gerar o respetivo ficheiro de *bitstream*. Desta forma, é permitido utilizar a aplicação mesmo que o projeto de arquitetura de hardware não esteja presente no diretório de trabalho.

Por fim, o menu de ajuda dá acesso ao manual da aplicação e a notas sobre a mesma.

Quando a aplicação é iniciada, verifica-se, em primeiro lugar, se existe o ficheiro de paleta de cores de omissão no diretório respetivo. Se o ficheiro não existir, é criada uma nova paleta de cores,

com o respetivo mapeamento de cor-estado, com cores geradas aleatoriamente em codificação RGB. A nova paleta de cores é assumida como a paleta de omissão e é carregada para a aplicação, sendo guardada no diretório de paletes de cor. Se o diretório de paletes de cor não existir no diretório de trabalho, este é criado em primeiro lugar e se a aplicação não conseguir criar a paleta de cores por omissão, esta devolve um erro e termina. Esta configuração e verificação inicial são necessárias uma vez que tem que existir sempre uma paleta de cores ativa na aplicação.

Em segundo lugar, verifica-se se o diretório de projeto de hardware está presente no diretório de trabalho. Se estiver, a aplicação entra no ambiente de desenvolvimento e simulação, caso contrário entra no ambiente de apenas simulação. Se a pasta do projeto de hardware for colocada no diretório de trabalho, com o nome por omissão, o utilizador pode posteriormente aceder ao ambiente de desenvolvimento e simulação apenas se o caminho para o diretório das ferramentas de desenvolvimento da Xilinx for correto, isto é, se o diretório existir.

Em terceiro lugar, verifica-se se as ferramentas de desenvolvimento da Xilinx estão instaladas no diretório por omissão. Se as ferramentas forem encontradas, a aplicação entra no ambiente de desenvolvimento e simulação, caso contrário a aplicação entra no ambiente de apenas simulação e o utilizador deverá aceder ao gestor de *bitstreams* e especificar o caminho alternativo para o diretório onde as ferramentas se encontram instaladas. Assim que o caminho para o diretório for indicado e estiver correto, é permitido entrar no ambiente de desenvolvimento e simulação apenas se a pasta do projeto de hardware existir no diretório de trabalho.

Note-se a dependência mútua entre a segunda e a terceira verificação para o acesso ao ambiente de desenvolvimento e simulação.

A aplicação não se encontra 100% preparada para executar em ambiente Unix uma vez que foi desenvolvida em ambiente Windows. Apesar de ter sido desenvolvida em Java, cujas aplicações são multi-plataforma, é necessário ter um cuidado especial quanto ao acesso a diretórios, no que diz respeito à indicação de caminhos para os mesmos, à execução de *scripts* em linha de comandos ou *shell*, em Windows e Unix respetivamente, em plano de fundo, uma vez que as extensões dos ficheiros são diferentes. Contudo, as modificações a fazer para suportar a execução em ambos os sistemas operativos não são extensivas.

## 5.2 Gestão de ficheiros e diretório de trabalho

A raiz do diretório de trabalho é constituída por cinco diretórios e o ficheiro executável da aplicação Java com a extensão *jar*. Na figura 5.3, é apresentada a árvore do diretório de trabalho em que a sua raiz é o diretório designado por “cellular-automata”. Os cinco diretórios imediatamente abaixo da raiz traduzem-se na organização de alto nível do diretório de trabalho.

O diretório “bitstreams” contém subdiretórios em que cada um corresponde a uma especificação de um AC distinto. Cada subdiretório contém um ficheiro de *bitstream* de extensão *bit*, um ficheiro de configuração de software que permite carregar a especificação do AC para a aplicação, de extensão *cfg*, um ficheiro de informação para o utilizador que é uma descrição textual das características do AC com extensão *info*, um ficheiro de extensão *bat*, um ficheiro *batch*, que

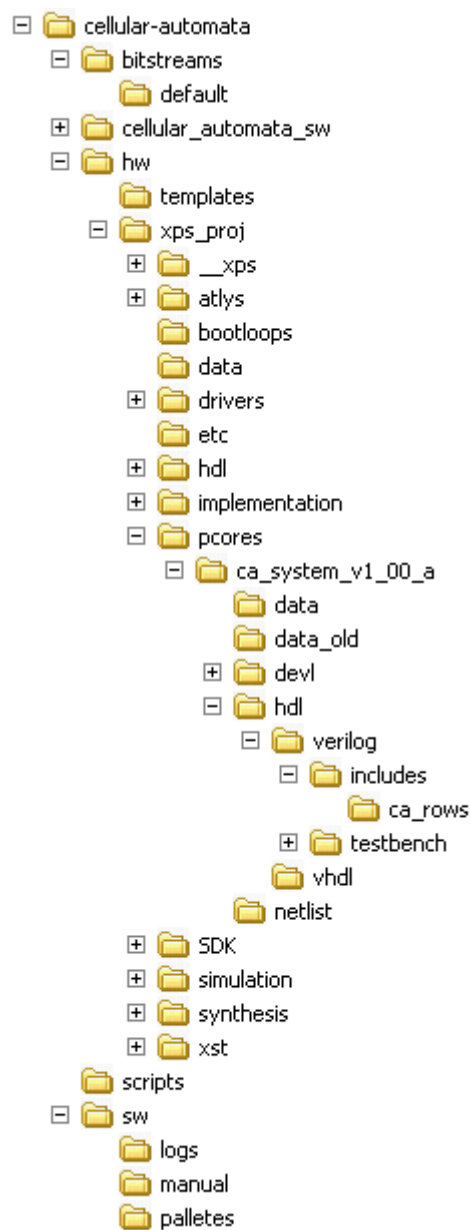


Figura 5.3: Árvore do diretório de trabalho

é um *script* executado em linha de comandos que permite chamar a ferramenta de programação de FPGAs da Xilinx, o iMPACT, e um ficheiro de configuração do iMPACT, de extensão *cmd*, que fornece uma sequência de comandos à ferramenta para realizar os passos de configuração de comunicação e programação da FPGA. Todos estes ficheiros tem o mesmo nome que é igual ao nome do subdiretório de *bitstream*. Repare-se que os ficheiros *batch* só são executáveis em ambiente Windows em linha de comandos. Para ambiente Unix, os *scripts* devem ter a extensão *sh* de *shell*.

O projeto de software encontra-se no diretório “cellular\_automata\_sw” e contém todos os ficheiros necessários ao desenvolvimento da aplicação. O projeto é incluído no diretório de trabalho apenas para alterações futuras.

Dentro do diretório designado por “hw” existem dois subdiretórios: o diretório “templates” e “xps\_proj”. O primeiro subdiretório tem dois ficheiros um de extensão *v* e outro *vh*, ambos escritos em Verilog HDL, em que o segundo ficheiro é um ficheiro de cabeçalho. O ficheiro “cell.v” é um template da arquitetura celular que é usado para integrar a especificação do utilizador para as células do AC e o ficheiro “parameter.vh” é um *template* que integra os parâmetros que são incluídos nos módulos da arquitetura de hardware para especificar as características do AC: dimensões, tipo de vizinhança e número de bits de estado por célula. O segundo subdiretório, “xps\_proj”, contém todo o projeto de hardware que integra os módulos do núcleo do AC com os outros núcleos IP que são ligados ao barramento PLB e o processador MicroBlaze. O projeto completo pode ser aberto pela ferramenta de Xilinx de desenvolvimento de sistemas embebidos, o *Xilinx Project Studio* (XPS), e o projeto de software para o processador MicroBlaze pode ser aberto na ferramenta de desenvolvimento de software da Xilinx, o *Software Development Kit* (SDK), que se encontra no subdiretório designado por “SDK”. Para o núcleo do AC existe um projeto independente na ferramenta da Xilinx para o desenvolvimento de RTL e síntese de circuitos, o *Integrated Software Environment* (ISE), que se encontra no subdiretório “dev1” do diretório “pcores” onde se encontram os núcleos IP desenvolvidos. Embora todos os subdiretórios do diretório de projeto de hardware sejam necessários, para a aplicação de software só é relevante o subdiretório designado por “verilog”, pois são nos ficheiros deste diretório que a aplicação faz as alterações necessárias para a especificação fornecida pelo utilizador. Este diretório contém os ficheiros que descrevem os módulos em Verilog HDL da arquitetura de hardware e ficheiros de inclusão. O seu conteúdo e o conteúdo dos seus subdiretórios é apresentado na figura 5.4.

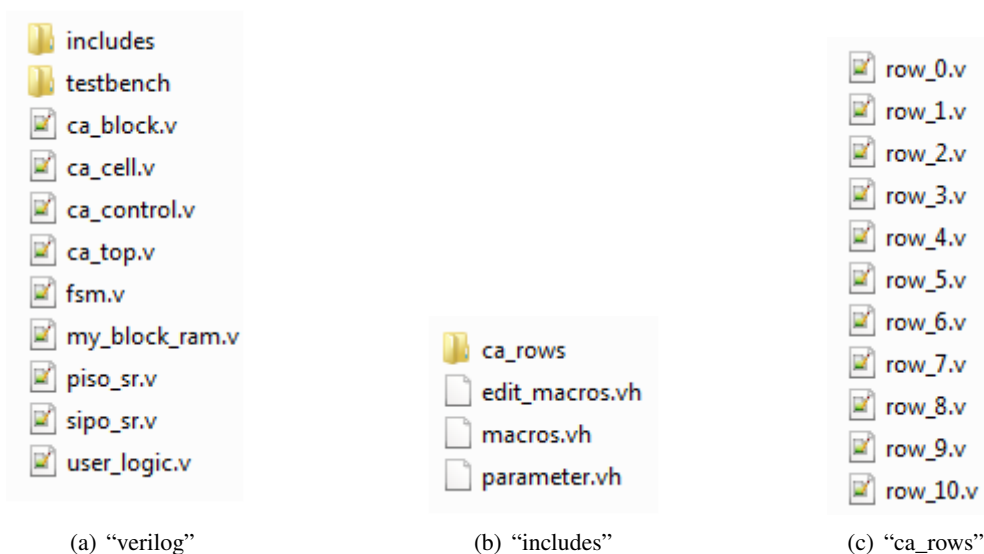


Figura 5.4: Conteúdo do diretório “verilog” e dos seus subdiretórios

No subdiretório “includes”, o ficheiro “edit\_macros.vh” define a condição fronteira do AC especificado e ficheiro “parameter.vh” define os parâmetros de caracterização do AC que é construído a partir do *template*, com o mesmo nome, do diretório “templates” já referido. O subdiretório “ca\_rows” contém os 256 ficheiros (na figura só são visíveis 10 ficheiros por uma questão de simplicidade) que descrevem a arquitetura de cada linha de células da matriz do AC, sendo o sufixo do nome do ficheiro o índice da linha. Cada um destes ficheiros é incluído uma vez no ficheiro “ca\_cell.v” que se encontra no diretório “verilog”, por cada instância feita do módulo de arquitetura celular no ficheiro “ca\_block.v”, que descreve a lógica bloco do AC, tal como foi descrito na subsecção 4.2.1 do capítulo 4. No diretório “verilog”, o ficheiro “ca\_top.v” é o módulo que instancia todos os módulos do núcleo do AC que, por sua vez, é instanciado no ficheiro “user\_logic.v” que é o módulo que descreve também a interface com o barramento PLB para acesso a memórias e ao banco de registos de controlo.

O diretório de alto nível designado por “scripts”, contém, no fundo, os ficheiros de *templates* para os *scripts* necessários para gerar ficheiros de *bitstream* e programar a FPGA, que são lidos pela aplicação e modificados para indicar caminhos para outros diretórios, ficheiros, outros *scripts* e ficheiros do tipo *makefile* que implementam a chamada às ferramentas da Xilinx na sequência do *design flow*.

Por fim, o diretório de alto nível designado por “sw” contém os subdiretórios e ficheiros que permitem configurar a aplicação de software e os que são criados como saída de resultados e registos de informação. O subdiretório que diz respeito à configuração da aplicação refere-se àquele que é designado por “palletes”. Este subdiretório contém os ficheiros, de extensão *.palette*, que mapeiam os estados das células numa paleta de cores codificada em RGB e que são usados pela aplicação para representar através de um mapa de cores o estado de um AC. No subdiretório designado por “logs” são guardados os registos de informação produzidos pela aplicação num formato de texto simples, de extensão *.txt*, e no subdiretório “manual” encontram-se os manuais de utilização da aplicação e especificação de ACs.

Todas as operações realizadas sobre ficheiros pela aplicação estão a cargo do gestor de ficheiros por ela implementada. O gestor de ficheiros, implementado sobre a API de Java para manipulação de ficheiros, implementa um conjunto de métodos que permitem simplificar o acesso a ficheiros e diretórios, operações de leitura e escrita, criar, remover e copiar ficheiros ou diretórios e tratamento de erros. Outras operações incluem obter a lista de ficheiros e subdiretórios de um caminho para um diretório especificado, ler o conteúdo de um ficheiro, filtrar o nome de um ficheiro para obter a sua extensão, procurar um termo no conteúdo de um ficheiro, entre outras. Uma vez que, aquando da especificação de um AC, os ficheiros dos módulos Verilog da arquitetura de hardware são modificados, as operações realizadas para a manipulação destes ficheiros dependem dos métodos implementados pelo gestor de ficheiros. Na secção 5.3 são descritas estas operações, são indicados os ficheiros que sofrem alterações e o conteúdo que é modificado, que são implementadas pelo gestor de arquitetura do AC.

## 5.3 Especificação do autómato celular

A especificação de um AC consiste em dois passos. O primeiro passo é a especificação das suas características, isto é, dimensões, número de bits por célula, tipo de vizinhança e condição fronteira e o segundo é a especificação da arquitetura das células, ou seja, a regra de transição de estado. A subsecção 5.3.1 descreve como é realizada a especificação das características de um AC e indica quais e como é modificado o conteúdo dos ficheiros dos módulos que descrevem a arquitetura do AC. A subsecção 5.3.2 descreve como é que o utilizador pode especificar a arquitetura das células a partir de uma linguagem simples criada para referenciar as células vizinhas, descreve as características desta linguagem, indica quais os ficheiros que descrevem a arquitetura celular, qual é o seu conteúdo e como é que este é modificado. Introduce-se também o gestor de arquitetura celular.

### 5.3.1 Especificação das características

Atente-se na figura 5.1 e na área de interface gráfica marcada com o número um. Quando o utilizador especifica as dimensões do AC e o número de bits por célula e valida os valores introduzidos, se estes estiverem de acordo com uma especificação válida, o ficheiro de inclusão “parameter.vh” do subdiretório “includes” é modificado (ver figura 5.4). Na verdade, o seu conteúdo é reescrito em função da especificação fornecida a partir do conteúdo do ficheiro de *template* “parameter.vh” do subdiretório “templates” (ver figura 5.3) por substituição de parâmetros do mesmo. O conteúdo do *template* é apresentado na listagem 5.1.

Listing 5.1: Conteúdo do ficheiro de template “parameter.vh”

```
parameter CA_HEIGHT = $H;
parameter CA_WIDTH = $W;
parameter BITS_PER_CELL = $B;
parameter NEIGHBOURHOOD = $N;
```

Os parâmetros **\$H**, **\$W**, **\$B** e **\$N** são substituídos pela altura, a largura, o número de bits por célula e o número de células vizinhas a considerar, respetivamente. Note-se que o identificador **\$N** é sempre substituído por quatro ou oito, apesar de na caixa de seleção do tipo de vizinhança, para ACs unidimensionais, a única opção disponível é o número dois.

Na seleção da condição fronteira, o ficheiro “edit\_macros.vh” do subdiretório “includes” é modificado. Uma vez que este ficheiro não necessita de um *template*, o seu conteúdo, apresentado na listagem 5.2, é alterado diretamente.

Listing 5.2: Conteúdo do ficheiro que define a condição fronteira

```
‘define CONTINUOUS_BOUNDARY
```

A *macro*, se estiver definida no ficheiro, permite indicar à ferramenta de síntese que a fronteira é do tipo contínua, caso contrário a fronteira é nula. A caixa de verificação da interface gráfica

permite indicar se esta *macro* deve ser definida ou não e é incluída no ficheiro “ca\_control.v”, o módulo de controlo de condições fronteira, que se encontra no subdiretório “verilog”.

### 5.3.2 Especificação da arquitetura celular

Atente-se, novamente, na figura 5.1 mas na área de interface gráfica marcada com o número quatro. Nas caixas de texto disponibilizadas, o utilizador escreve a sua descrição para a arquitetura das células de uma linha da matriz do AC. A caixa de baixo destina-se à descrição de lógica combinacional auxiliar, sendo opcional, e a caixa de baixo à descrição da função de transição de estado da célula, que corresponde à parte sequencial. Para isso, o utilizador dispõe de uma linguagem que usa identificadores que permitem simplificar a referência dos sinais que constituem os barramentos de entrada do módulo da célula, isto é, os barramentos que interligam a célula com as suas vizinhas e propagam os bits de estado. Esta linguagem é usada juntamente com Verilog HDL. Na tabela 5.1, apresentam-se os identificadores para referenciar as células da vizinhança, qual a célula que referenciam e o barramento correspondente. Por uma questão de simplicidade, é usada a notação de pontos cardeais e colaterais para indicar a localização das células vizinhas em torno de uma célula central.

| Célula vizinha a | Tipo de vizinhança |       | Barramento                   |
|------------------|--------------------|-------|------------------------------|
|                  | von Neumann        | Moore |                              |
| Norte            | .@n                | .@n   | gen_iboundary[2].input_wires |
| Sul              | .@s                | .@s   | gen_iboundary[0].input_wires |
| Este             | .@e                | .@e   | gen_iboundary[1].input_wires |
| Oeste            | .@w                | .@w   | gen_iboundary[3].input_wires |
| Nordeste         | -                  | .@ne  | gen_iboundary[6].input_wires |
| Sudeste          | -                  | .@se  | gen_iboundary[4].input_wires |
| Sudoeste         | -                  | .@se  | gen_iboundary[5].input_wires |
| Noroeste         | -                  | .@no  | gen_iboundary[7].input_wires |
| Centro           | .@co               | .@co  | cell_out                     |

Tabela 5.1: Identificadores e barramentos para referenciar as células da vizinhança.

A última entrada da tabela indica como referenciar a própria célula e o prefixo .@ indica a presença de um identificador. Cada barramento `gen_iboundary[j].input_wires` é, de facto, uma fração do barramento de entrada do módulo de arquitetura de célula por onde são transportados  $b$  bits de estado.

O propósito desta linguagem é de implementar um simples mecanismo de substituição de texto, isto é, substituir os identificadores pelo barramento correspondente, pelo que as suas características, listadas abaixo, são bastante limitadas:

- Uma vez que cada célula pode ter mais que um bit de estado, os identificadores suportam argumentos para referenciar o estado de uma célula vizinha ao nível do bit. Por exemplo, o identificador .@n(0) permite referenciar o bit de estado zero, o menos significativo, da célula

localizada a norte de uma célula central. Da mesma forma, `gen_iboundary[2].input_wires[0]` tem o mesmo significado.

- Não é suportada qualquer tipo de verificação caso sejam usados identificadores que não existam na vizinhança especificada, embora seja verificado se um identificador fornecido não existe de todo. Nesta situação, é emitido um erro.
- Não é suportada qualquer tipo de verificação caso sejam passados em argumento bits de estado que não existam na vizinhança especificada.
- Na passagem de um argumento, se o segundo parêntesis não for encontrado é emitido um erro.

O utilizador ao descrever a arquitetura das células deve ter o cuidado de contornar as situações indicadas na lista acima, uma vez que se a substituição dos identificadores pelos barramentos não for feita corretamente, o processo de síntese do circuito emite um erro. Se o utilizador pretender, pode inserir as referências diretas dos barramentos em vez dos identificadores para os mesmos.

Após feita a especificação da arquitetura celular, o utilizador deve abrir o gestor de arquitetura celular que é apresentado na figura 5.5. O gestor está dividido em três conjuntos de funcionalidades: permite salvar uma especificação de arquitetura celular para uma linha de células da matriz do AC à sua escolha, permite abrir uma especificação de arquitetura existente para edição e permite gerar arquiteturas de célula para todas as linhas da matriz a partir da linha do índice de referência, ou seja, a linha de índice zero que se encontra no topo da matriz.

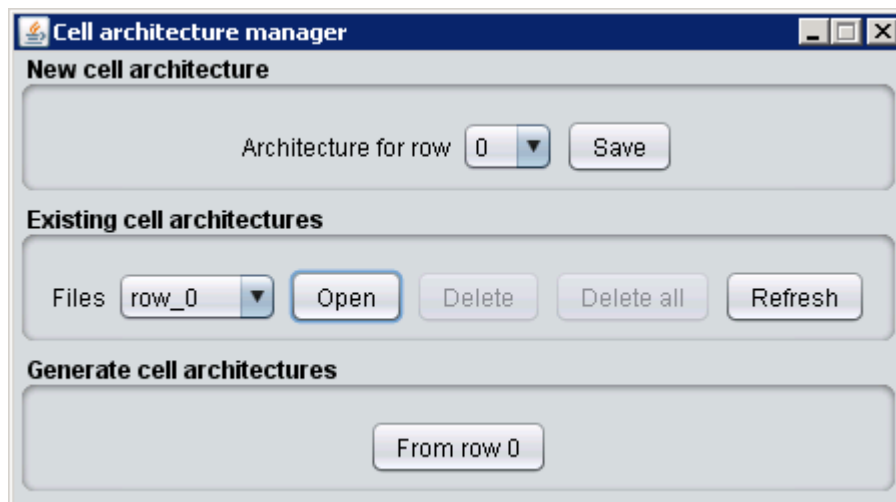


Figura 5.5: Gestor de arquitetura celular

Para salvar uma especificação de arquitetura celular, o utilizador deve, em primeiro lugar, especificar a altura da matriz do AC para habilitar no gestor a seleção da linha da matriz a que pretende atribuir a especificação de arquitetura feita e, em segundo lugar, seleccionar então a linha

da matriz e premir o botão para salvar. Quando uma especificação de arquitetura celular é salva são realizados os seguintes passos:

1. O código introduzido tanto para a lógica combinatória como para a parte sequencial é pesquisado por identificadores de referência à vizinhança de uma célula e estes são substituídos pelos nomes dos barramentos respetivos.
2. O ficheiro de *template* “cell.v” que se encontra no subdiretório designado por “templates” (ver figura 5.3), cujo conteúdo é apresentado na listagem 5.3, é lido e os seus parâmetros são substituídos pelos segmentos de código escritos pelo utilizador, o que significa que este não tem acesso a qualquer outra parte do *template* evitando assim danos ao conteúdo do mesmo.
3. Os parâmetros **\$ASSIGN** e **\$RULE** são substituídos pelos segmentos de código que correspondem à lógica combinacional e parte sequencial, respetivamente, e são delimitados por duas linhas de texto comentado, uma no início e outra no fim, que servem como referência para a localização dos segmentos de código aquando da leitura de um ficheiro de arquitetura celular (ver listagem 4.6, da subsecção 4.2.1, como exemplo de um módulo de arquitetura celular resultante).
4. O conteúdo do ficheiro de *template* já parametrizado é copiado para o ficheiro que especifica a arquitetura celular, localizado no subdiretório “ca\_rows” (ver figura 5.4), das células da linha da matriz selecionada. Se o ficheiro não existir, este é criado.

Listing 5.3: Conteúdo do ficheiro de *template* de arquitetura celular

```
$ASSIGN
always @(posedge clk)
begin
    if (rst)
        cell_out <= 0;
    else
        begin
            if (shift)
                cell_out <= gen_iboundary[2].input_wires;
            else
                begin
                    if (rule)
                        begin
                            $RULE
                        end
                    end
                end
            end
        end
    end
```

**end**

Para abrir uma especificação de arquitetura celular existente, o utilizador deve seleccionar o ficheiro a partir da caixa de seleção do gestor de arquitetura celular. O nome dos ficheiros é sempre da forma “row\_*i.v*”, em que *i* é um sufixo numérico que indica o índice da linha da matriz do AC. Os segmentos de código associados à lógica combinatória e à parte sequencial são extraídos do ficheiro e carregados para as caixas de edição de texto respetivas. A referência dos barramentos é convertida agora para os identificadores a eles associados. Se as referências dos barramentos não estiverem corretas, é devolvido um erro e o segmento de código respetivo não é carregado.

A última funcionalidade do gestor de arquitetura celular é de copiar o conteúdo do ficheiro “row\_0.v” para os restantes, isto é, replicar a arquitetura das células da linha da matriz de índice zero para todas as outras. Esta funcionalidade é útil para gerar ACs homogéneos, em que as células tem a mesma regra de transição de estado, evitando assim especificar a mesma arquitetura o mesmo número de vezes que o número de linhas da matriz. Se o ficheiro não existir nem tiver sido especificada a altura da matriz do AC, é devolvido um erro.

## 5.4 Gestão de paletes de cor e ficheiros mapeadores

A representação gráfica dos estados das células de um AC é feita a partir de mapeamentos das células de estado em cores distintas. Estes mapeamentos estão guardados em ficheiros, no diretório de alto nível designado por “palletes” (ver figura 5.3), e estabelecem a correspondência entre o código RGB de uma cor e o valor numérico do estado de uma célula. Um exemplo do conteúdo destes ficheiros é mostrado na listagem 5.4

Listing 5.4: Exemplo do conteúdo de um ficheiro de mapeamento cor-estado de célula

```
0=255,0,0
1=0,0,255
2=51,255,0
3=255,255,0
4=0,255,204
5=255,0,204
6=255,102,0
7=102,0,153
```

Os mapeamentos estabelecem-se como tabelas de dispersão em que, para cada entrada da mesma, cada valor à esquerda da igualdade, no conteúdo do ficheiro, representa uma chave, o valor numérico associado ao estado de uma célula, e o valor à direita da igualdade representa o código RGB associado a essa chave. Esta tabela tem uma relação de um para um e atribui uma paleta de cores a um grupo de estados. O número de entradas da tabela não é limitado, contudo a aplicação só reconhece até a um máximo de dezasseis chaves (de zero a quinze), pois foi o valor máximo estipulado para uma paleta de cores ativa.

Quando a aplicação é iniciada, o diretório “palletes” é pesquisado e o conteúdo de todos os ficheiros lá presentes é lido. Se não existirem ficheiros mapeadores, a aplicação cria um novo automaticamente e define-o como o mapeamento por omissão, caso contrário se os ficheiros tiveram a extensão correta (*.palette*) e o seu conteúdo estiver no formato apresentado no exemplo da listagem 5.4, a aplicação cria um vetor de objetos cujo tipo é implementado pela classe apresentada na listagem 5.5, sendo cada elemento do vetor um objeto da classe. Este vetor define o banco de paletes de cor disponíveis carregado a partir dos ficheiros mapeadores do diretório para a aplicação. Ao carregar todas as paletes de cor disponíveis para a aplicação, mesmo que os ficheiros mapeadores sejam apagados enquanto esta se encontra a executar, não origina qualquer conflito, e todas as alterações feitas refletem-se primeiro internamente e só depois nos ficheiros mapeadores. Se algum ficheiro tiver sido apagado enquanto a aplicação se encontrar a executar, aquando da modificação da palette correspondente o ficheiro mapeador é novamente criado.

Listing 5.5: Classe que implementa um mapeamento cor-estado de célula

```
public static class Palette {  
  
    HashMap colorMapping;  
    int ID;  
    String name;  
    boolean active;  
  
    (...)  
}
```

A classe implementa uma tabela de dispersão para o mapeamento cor-estado, um número interno de série, o nome do ficheiro de origem e uma variável de dois estados que indica se a palette de cores correspondente é a palette ativa. O número de série é usado para atribuir às células de um AC a sua palette de cores e para manipulação interna, e o nome é usado como identificador visual para o utilizador apagar, editar o mapeamento associado à palette de cores ou criar novas paletes de cor.

Depois da inicialização do banco de paletes de cor e de carregada para a aplicação a palette de cores ativa, o gestor de paletes de cor é iniciado e o utilizador pode aceder às suas funcionalidades. O acesso ao gestor é feito pelo menu de opções situado na barra de ferramentas da aplicação. Na figura 5.6 ilustra-se o acesso ao menu do gestor que tem quatros itens de menu e um submenu disponíveis. De cima para baixo, o primeiro item permite aceder às funcionalidades de criar ou remover paletes de cor, o segundo permite atribuir uma palette de cores a uma linha da matriz do AC e o terceiro permite editar paletes de cor e o mapeamento cor-estado associado. Para seleccionar outra palette como palette ativa, ao submenu são mostradas as paletes de cor atualmente carregadas na aplicação. Na figura 5.7 mostram-se as janelas que são abertas ao aceder a cada um dos primeiros três itens do menu do gestor.

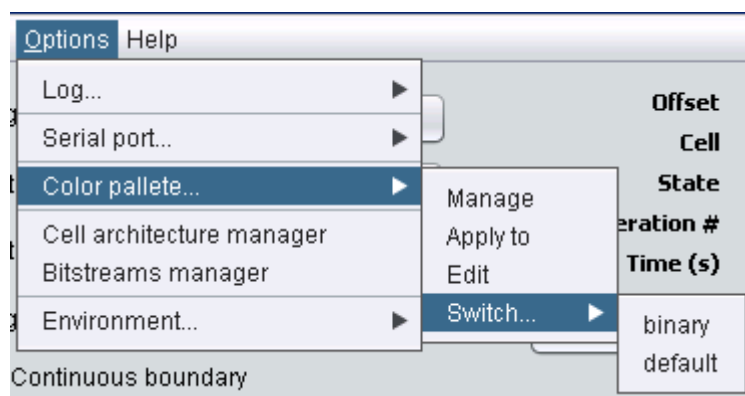
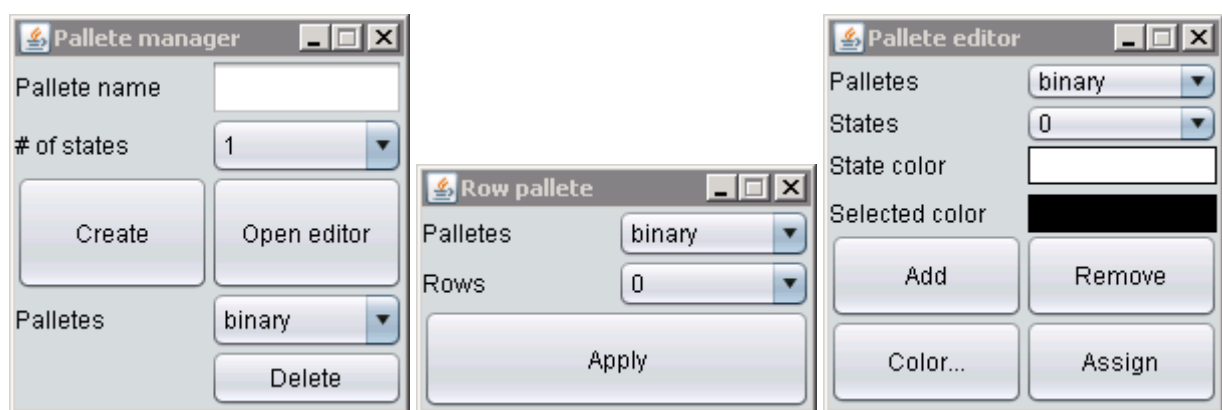


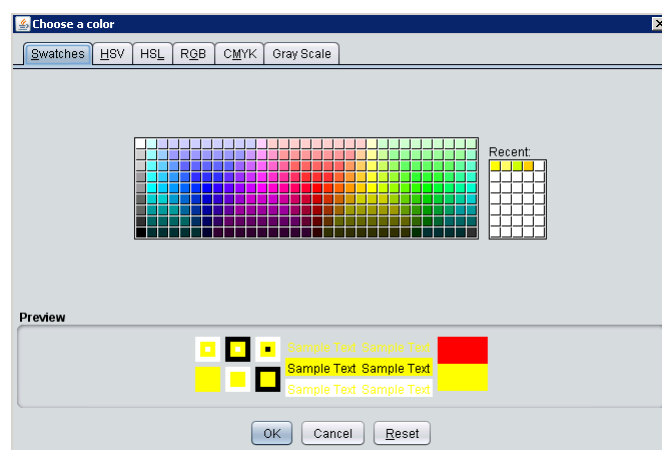
Figura 5.6: Acesso ao menu do gestor de paletes de cor



(a) Criar/remover

(b) Atribuir

(c) Editar



(d) Paleta de seleção de cor

Figura 5.7: Ferramentas de manipulação de paletes de cor e mapeamentos

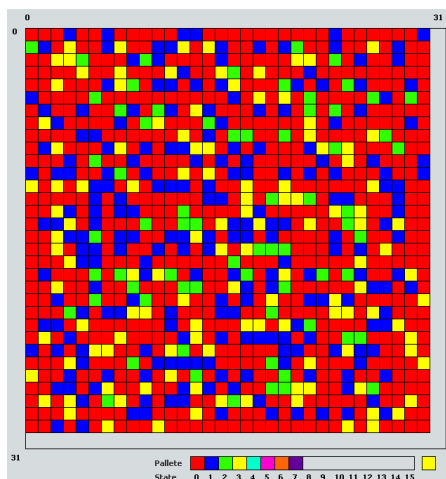
Na janela da figura 5.7(a), para criar uma nova paleta deve ser fornecido um nome e o número de estado de células a codificar. As cores da paleta são geradas aleatoriamente e o ficheiro mapea-

dor pode ser posteriormente editado. Se o nome fornecido já existe, a aplicação emite um aviso e pergunta ao utilizador se deseja substituir o mapeamento existente. Em caso afirmativo, a paleta de cores e o mapeamento associado são modificados internamente e o conteúdo do ficheiro mapeador é reescrito, caso contrário a nova paleta de cores é adicionada ao banco interno de paletes de cor da aplicação e o ficheiro mapeador é criado. As paletes de cor existentes são apresentadas numa caixa de seleção e podem ser removidas tanto da aplicação como o ficheiro mapeador associado do diretório de trabalho. Contudo, remover uma paleta de cores só é possível se esta não estiver a ser utilizada, isto é, se não for a paleta de cores ativa na aplicação nem se estiver aplicada a uma ou mais linhas de células da matriz do AC. Ao remover uma paleta de cores, a sua entrada no banco interno de paletes de cor é apagada e, de seguida, o ficheiro mapeador no diretório de trabalho removido.

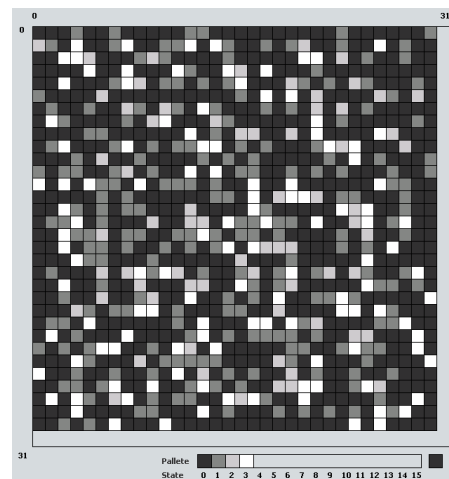
Na figura 5.7(b), mostra-se a janela da ferramenta para atribuir uma paleta de cores a uma dada linha da matriz do AC. Esta operação atribui às células da linha selecionada o número de série interno da paleta de cores escolhida, e é bem sucedida apenas se o ficheiro mapeador tiver entradas suficientes para codificar os estados das células da linha, caso contrário a atribuição da paleta de cores é rejeitada emitindo um erro. Da mesma forma, quando se troca a paleta de cores ativa o mesmo princípio é aplicado com a única diferença de que os estados de todas as células da matriz do AC são avaliados.

A última ferramenta do gestor de paletes de cor é apresentada na figura 5.7(c). O editor de paletes de cor permite editar uma paleta de cores existente e o mapeamento cor-estado associado. O utilizador pode remover ou adicionar entradas no mapeamento e codificar os estados com a cor que pretender a partir do menu de seleção de cores apresentado na figura 5.7(d). Além de cores já predefinidas, o utilizador pode especificar a cor que pretender em RGB ou até selecionar uma cor numa escala de cinzentos, acedendo ao separador respetivo. A remoção e adição de entradas nos mapeamentos é sempre feita sequencialmente uma vez que as entradas dos ficheiros mapeadores estão organizadas por ordem crescente, ou seja, ao adicionar uma nova entrada o estado seguinte é o estado da última entrada somado de um e remove-se sempre a última entrada que é a que tem o valor numérico de estado maior. Ao remover a última entrada num mapeamento cuja paleta de cores é a que está ativa ou atribuída a uma linha da matriz do AC, se alguma das células da matriz do AC em causa tiver o estado de valor numérico mais alto, é emitido um aviso e perguntado ao utilizador se pretende saturar o estado destas células para o estado imediatamente abaixo. Se a resposta for negativa, a entrada do mapeamento não é removida. No caso da alteração ser feita à paleta de cores ativa, o número de cores disponíveis ao utilizador para a atribuição de estados às células é atualizado e reduzido em um.

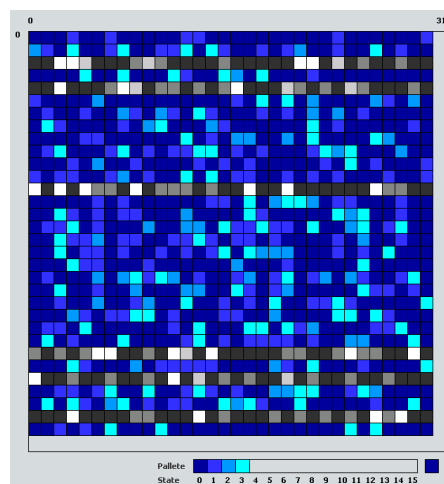
Na figura 5.8, apresentam-se resultados da implementação das funcionalidades que o gestor de paletes de cor oferece e o aspeto final. Note-se que são apenas exemplos de aplicação do uso das paletes de cor para um AC qualquer de quatro estados com dimensão de 32x32 células. O estado do AC nas figuras 5.8(a), 5.8(b) e 5.8(c) é idêntico.



(a) Paleta de cores primárias



(b) Paleta de cores numa escala de cinzentos



(c) Paleta de cores em tons de azul e cinzentos

Figura 5.8: Representação gráfica do estado um AC com paletes de cor distintas

## 5.5 Gestão de gráficos

O visualizador gráfico trata-se de um painel que é usado pelos métodos da API de Java de gráficos 2D com um moldura para desenhar as células de um AC. A representação das células é feita por quadrados com 20 píxeis de lado e o painel do visualizador é também um quadrado com 600 píxeis de lado. Assim, na área do painel é possível mostrar de uma vez só 30x30 células. As dimensões adotadas advêm da opção de proporcionar uma vista imediata de um AC na janela principal da aplicação e da limitação imposta da área da mesma, de forma a conseguir integrar o maior número de funcionalidades no mesmo espaço disponível.

A representação gráfica de um AC é visível assim que o utilizador especifica as suas dimensões e quando são reajustadas, o número de células no painel é também reajustado removendo ou adicionando células. Por especificação ou reajustamento, todas as novas células que são incluídas

na matriz do AC são inicializadas com o estado zero. A informação relativa a cada célula é guardada num vetor criado pela aplicação cujo tipo de dados são objetos da classe apresentada na listagem 5.6.

Listing 5.6: Classe que implementa uma célula de um AC

```
public static class Cell {

    Point2D pos;
    Point2D initPos;
    Point2D coords;
    int state;
    int palleteID;

    (...)
}
```

A classe Point2D implementa um ponto em coordenadas cartesianas  $(x,y)$ . Os objetos *initPos*, *pos* e *coords*, guardam a informação relativa às coordenadas iniciais em píxeis da posição da célula no painel, que correspondem a um *offset* nulo em relação à sua posição inicial, guardam as coordenadas em píxeis da posição atual da célula no painel, pelo que a diferença entre estas e às coordenadas iniciais indicam o *offset* de posição e guardam as coordenadas da célula  $(i,j)$  da linha e coluna da matriz onde se encontra, respetivamente. Cada célula também mantém a sua informação de estado e o número de série interno associado à paleta de cor que lhe está atribuída.

Quando é feita uma especificação, ajuste das dimensões do AC ou atribuição de uma nova paleta de cores, o conteúdo do painel do visualizador gráfico deve ser alterado. Para isso, a API de Java de gráficos implementa o método *paintComponent()* que permite redesenhar o conteúdo do painel, e a sua invocação é feita indiretamente ao invocar o método *repaint()* sempre que é necessário fazer alterações aos gráficos. O método *paintComponent()* deve ser reescrito (*override*), isto é, substituído pelo o método original mas implementando a mesma a funcionalidade imposta pela super classe desse método. Para redesenhar os gráficos, o método *paintComponent()* lê o conteúdo do vetor de células e, uma a uma, invoca os métodos para desenhar um quadrado cujas coordenadas em píxeis são dadas pelo objeto de classe *pos*. O número de série de paleta de cor é usado para consultar o banco de paletes de cor e obter a cor que mapeia o estado atual da célula, preenchendo assim o quadrado desenhado.

A gestão de gráficos implementa funcionalidades que permitem ao utilizador interagir com o painel do visualizador fazendo uso do rato, para melhorar a navegação, detetando as suas ações e desplotando eventos para as processar. Na lista abaixo são explicadas estas funcionalidades.

**Leitura de coordenadas e estado** Quando o utilizador passa o cursor do rato por cima da matriz de células do AC, é possível obter de imediato as coordenadas  $(i,j)$  da matriz e o estado da célula para a qual o cursor aponta.

**Deslocamento da matriz do AC** Uma vez que na área visível do painel de gráficos só cabem 30x30 células, para ACs de dimensões superiores, o utilizador, com o botão esquerdo do rato premido, pode deslocar a matriz para visualizar as células fora da área visível. Enquanto a matriz é deslocada, de forma a orientar o utilizador, no exterior e nos extremos do painel são indicadas as coordenadas das células que se encontram nos cantos da área visível. O método que implementa esta funcionalidade implementa os cálculos necessários para determinar as coordenadas em píxeis da nova posição de cada célula. O deslocamento corresponde à diferença entre as coordenadas atuais do cursor e a coordenadas quando o botão do rato foi clicado. A partir daí, o deslocamento, positivo ou negativo, é somado às coordenadas atuais da célula. Note-se que a origem dos eixos é no canto superior esquerdo, sendo que  $x$  cresce para a direita e  $y$  para baixo e o deslocamento é limitado em função das dimensões em píxeis da matriz para que a sua representação gráfica se mantenha dentro dos limites do painel. O *offset* máximo de deslocamento  $(i_{os}^{max}, j_{os}^{max})$  indica o número de píxeis a que uma célula pode ser deslocada em  $x$  e  $y$  em relação às coordenadas iniciais  $(i_0, j_0)$  e é dado por  $(l * c) - L$  e  $(l * r) - L$ , respetivamente, em que  $l$  é o lado do quadrado de uma célula e  $L$  o lado do painel, ambos em píxeis, e  $c$  é o número de colunas e  $r$  o número de linhas. Sendo  $l = 20$  e  $L = 600$ , não é permitido deslocar as células da matriz em  $x$  se  $(l * c) \leq L$  e em  $y$  se  $(l * r) \leq L$ , pelo que o *offset* máximo de deslocamento é igual a zero com  $r \leq 30$  e  $c \leq 30$ . Assim, o deslocamento em  $x$  ou  $y$  só ocorre se as novas coordenadas da célula  $(i, j)$  satisfizerem a condição  $(i_0, j_0) - (i_{os}^{max}, j_{os}^{max}) < (i, j) < (i_0, j_0)$ , em que a primeira condição refere-se à limitação do deslocamento negativo e a segunda à do deslocamento positivo. A figura 5.9 ilustra o que foi explicado.

**Atribuição de estado** A partir da paleta de cores ativa e consultando o mapeamento cor-estado respetivo, o utilizador pode selecionar a cor e “pintar” o quadrado correspondente à célula cujo estado quer alterar. A cor é selecionada clicando com o botão esquerdo do rato na paleta de cores ativa e, para “pintar” um quadrado, clicar com o botão direito do rato em cima do mesmo no painel. Internamente, de facto, não se troca a cor do quadrado mas sim o estado da célula que ele representa uma vez que uma célula não retém a informação de cor. Ao trocar o estado e na invocação do método *paintComponent()* pelo Java, a partir do número de série interno da paleta de cor atribuída à célula, consulta-se o mapeamento estado-cor e obtém-se a nova cor para o preenchimento do quadrado.

**“Pintura” rápida** O clique no botão do meio do rato ativa/desativa o modo de “pintura” rápida. Deslocando apenas o cursor por cima da matriz de células, sem clicar com o botão direito do rato, o estado das células é modificado de imediato e a cor dos quadrados associados é trocada para a cor selecionada da paleta ativa.

**Zoom in e zoom out** Ao deslocar para cima ou para baixo o botão de deslocamento do rato, é possível fazer *zoom in* e *zoom out*, respetivamente, de forma a visualizar o AC todo na área disponível do painel gráfico. Da mesma forma que no deslocamento da matriz, o *zoom out* só é permitido, e só faz sentido, se o AC tiver dimensões superiores a 30x30 células. Esta

condição é acrescida de outra que limita a redução do lado  $l$  dos quadrados das células, de forma a não tornar ilegível o conteúdo do AC, que depende do lado  $L$  do painel gráfico e do número de colunas  $c$  e linhas  $r$  da matriz. Sendo  $l_z$  o limite mínimo para o lado do quadrado de uma célula em *zoom out*, este só é permitido se se verificar que  $l_x = L/c \leq l_z \wedge l_y = L/r \leq l_z$ . Os quocientes  $L/c$  e  $L/r$  são sempre arredondados por defeito às unidades o que leva a que, se não forem inteiros, a área do painel gráfico não é ocupada na sua totalidade, uma vez que se pretende manter a proporção de um quadrado, isto é,  $l_x/l_y = 1$ , então  $l = \min(l_x, l_y)$ . Na aplicação, tem-se  $l_z = 3$  pelo que  $c_{max} = r_{max} = 200$ , o que significa que só é possível fazer *zoom out* em ACs com dimensões iguais ou inferiores a 200x200 células. Na figura 5.10 ilustra-se o *zoom out* de um AC com 64x64 células com dois estados por célula.

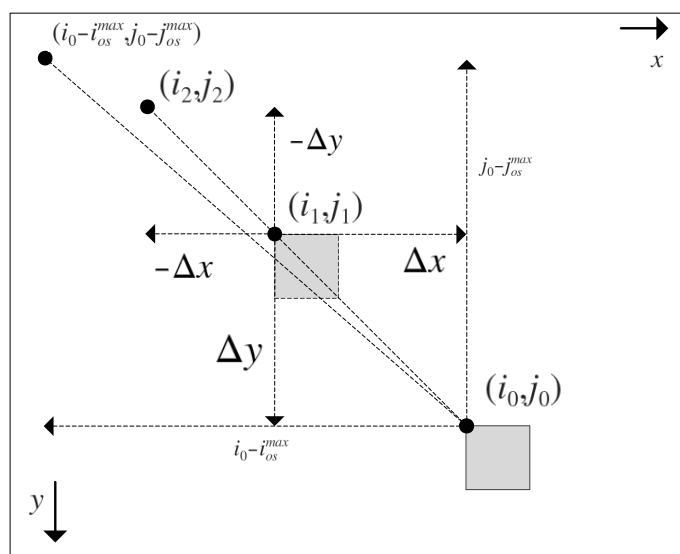


Figura 5.9: Deslocamento de uma célula da matriz do AC no painel gráfico

## 5.6 Gestão de *bitstreams* e implementação em hardware

O gestor de *bitstreams* permite ao utilizador programar a FPGA com um *bitstream* gerado previamente que especifica a arquitetura de um determinado AC, gerar novos ficheiros de *bitstream*, a partir da especificação das características e arquiteturas celulares do AC conforme explicado na secção 5.3 e especificar o caminho para o diretório de ferramentas da Xilinx. O processo de geração e programação de ficheiros de *bitstream* é concretizado por execução de *scripts* em linha de comandos, que realizam as chamadas às ferramentas da Xilinx necessárias para cada um dos processos. O processo de geração de um ficheiro de *bitstream* segue a sequência de passos indicados pelo “FPGA Design Flow” descritos sinteticamente na subsecção 5.6.1. Os *scripts* precedem

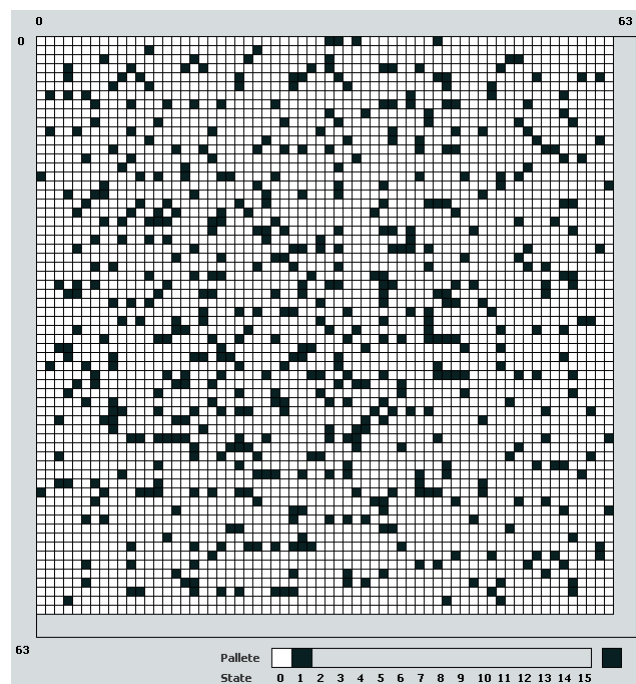


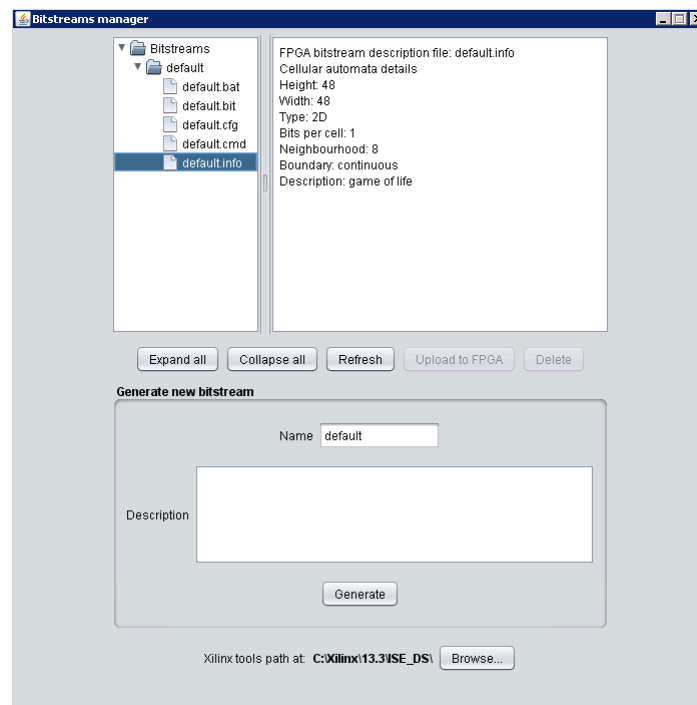
Figura 5.10: *Zoom out* sobre um AC de 64x64 células de dois estados por célula

a qualquer chamada às ferramentas com a execução de um *script* secundário, que já é fornecido pela Xilinx, que permite definir as variáveis de ambiente que indicam o caminho completo para a localização das ferramentas, o que, desta forma, torna possível a chamada direta às mesmas sem a necessidade de navegar internamente até cada um dos diretórios onde se encontram os ficheiros executáveis das ferramentas. O *script* secundário encontra-se no diretório de instalação das ferramentas da Xilinx e designa-se por “settings32.bat” ou “settings64.bat” consoante a plataforma. Se a aplicação não encontrar nenhum destes ficheiros no caminho especificado para as ferramentas, não é possível gerar nem programar um ficheiro de *bitstream*.

Na figura 5.11 mostra-se a janela do gestor de *bitstreams*. A árvore de diretórios existente mapeia diretamente o conteúdo do diretório “bitstreams” do diretório de trabalho (ver figura 5.3). Este diretório é por sua vez organizado em subdiretórios em que cada um diz respeito a uma especificação de um AC diferente. Cada subdiretório contém cinco ficheiros distintos cuja descrição encontra-se na tabela 5.2.

Quando o utilizador selecciona na árvore de subdiretórios o ficheiro de extensão *.info*, o seu conteúdo é mostrado na área de texto à direita da árvore. O conteúdo dos ficheiros restantes é escondido uma vez que não tem qualquer importância para o utilizador comum.

Para programar um ficheiro de *bitstream* existente na FPGA, o utilizador deve escolher o subdiretório de especificação que pretende carregar, seleccionar o ficheiro de *bitstream* associado e premir o botão intitulado de “Upload to FPGA”. A partir desse momento, a aplicação executa o *script*, de extensão *.bat* que chama a ferramenta iMPACT, em linha de comandos e em plano de fundo. O ficheiro de extensão *.cmd* é lido e o iMPACT vai produzindo mensagens à medida

Figura 5.11: Janela do gestor de *bitstreams*

| Extensão     | Descrição                                                                                                                                                                                                                                                                                                                                                                                          |
|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>.cmd</i>  | Ficheiro de configuração da ferramenta da Xilinx para programar FPGAs, o iMPACT, que contém os comandos para configurar a ligação com uma FPGA e programar um ficheiro de <i>bitstream</i> existente. Este ficheiro recebe como parâmetro o caminho para a localização de um ficheiro de <i>bitstream</i> .                                                                                        |
| <i>.bat</i>  | Ficheiro <i>batch</i> que executa o <i>script</i> que faz a chamada ao iMPACT que recebe como parâmetro o ficheiro de extensão <i>.cmd</i> .                                                                                                                                                                                                                                                       |
| <i>.info</i> | Ficheiro que contém a informação associada à especificação do AC, as suas características e descrição, num formato legível para o utilizador.                                                                                                                                                                                                                                                      |
| <i>.cfg</i>  | Ficheiro que contém a informação de configuração da aplicação que é lido e carregado para a mesma aquando da programação do ficheiro de <i>bitstream</i> na FPGA. O seu conteúdo é semelhante ao do ficheiro de extensão <i>.info</i> , mas o seu formato é idêntico ao de uma tabela de dispersão, cujas chaves são as características do AC, que facilitam a sua leitura por parte da aplicação. |
| <i>.bit</i>  | Ficheiro de <i>bitstream</i> que contém as definições da lógica interna da FPGA e das suas interligações de especificação de arquitetura de hardware e informação específica associada à FPGA em causa.                                                                                                                                                                                            |

Tabela 5.2: Descrição dos ficheiros incluídos num subdiretório de especificação de um AC

que a configuração da ferramenta é realizada e o ficheiro de *bitstream* programado. No final do processo de programação é devolvida uma mensagem de erro ou sucesso em conformidade com o resultado. As mensagens devolvidas pela linha de comandos são processadas pela aplicação e

redirecionadas para a caixa de texto na interface gráfica onde também são registadas as mensagens de saída da aplicação, podendo o utilizador, se pretender, guardar um ficheiro de *log*. (ver figura 5.1, área marcada com o número cinco). Depois de programar a FPGA, o utilizador pode dar início às simulações do AC estabelecendo a comunicação série com a plataforma de hardware.

Após feita a especificação completa de um AC, o utilizador deve abrir o gestor de *bitstreams*, dar o nome desejado ao ficheiro de *bitstream* e uma descrição da especificação. Se o nome fornecido já existir, é devolvido um erro e o utilizador deve dar um nome diferente. Ao premir o botão intitulado de “Generate” o processo de geração do ficheiro de *bitstream* é iniciado. Agora, a aplicação lê do diretório “scripts” do diretório de trabalho o *script* que realiza a chamada às ferramentas da Xilinx na sequência dos passos especificados pelo *FPGA Design Flow*, o ficheiro “generate.bat”, e executa-o na linha de comandos em plano de fundo. As mensagens produzidas pelas ferramentas, da mesma forma que no processo de programação da FPGA, são reencaminhadas e registadas na caixa de texto de saída da interface gráfica. Quando o processo de geração do ficheiro de *bitstream* termina, se for bem sucedido, são criados os restantes quatro ficheiros (ver tabela 5.2) que constituem o conteúdo do subdiretório de especificação de um AC. Em primeiro lugar, o subdiretório é criado no diretório “bitstreams” cujo nome é o que o utilizador forneceu previamente. Os ficheiros de extensões *.info* e *.cfg* são criados no novo subdiretório a partir da especificação das características do AC existentes na aplicação. Os ficheiros de extensões *.bat* e *.cmd* localizam-se no diretório “scripts” do diretório de trabalho como *templates* de nome “program.bat” e “impact.cmd”, respetivamente, e o seu conteúdo é lido, modificado ao introduzir os parâmetros a eles associados (ver novamente a tabela 5.2) e escrito em novos ficheiros no subdiretório de *bitstream* criado com as extensões respetivas. Por fim, o ficheiro de *bitstream* gerado “system.bit”, que se encontra no subdiretório “implementation” do diretório do projeto de hardware “xps\_proj” (ver figura 5.3), uma vez que os ficheiros relacionados com a arquitetura de hardware estão aí localizados, é copiado para o novo subdiretório de *bitstream* e renomeado para o nome que o utilizador forneceu e a árvore de subdiretórios do gestor de *bitstreams* é atualizada automaticamente. Todos os ficheiros associados a uma especificação de um AC, ficam com o mesmo nome do subdiretório de *bitstream* correspondente.

### 5.6.1 Fluxo de projeto para FPGA

O projeto de arquitetura de hardware completo que integra os núcleos IP de hardware com o processador MicroBlaze, foi desenvolvido no XPS que é um software da Xilinx para o desenvolvimento de sistemas embebidos. O XPS cria automaticamente um ficheiro do tipo *makefile* para o projeto que, dependendo do parâmetro que lhe é passado, permite concretizar um ou mais passos associados ao fluxo de projeto típico de FPGA e gerar os ficheiros de saída correspondentes que descrevem o projeto de hardware. São feitas três chamadas ao ficheiro *makefile* que são integradas no *script* criado para gerar o ficheiro de *bitstream*. As operações realizadas por cada chamada são as seguintes:

1. Remover todos os ficheiros previamente gerados para limpar o diretório do projeto de hardware.
2. Correr as ferramentas para síntese, implementação e geração do *bitstream*.
3. Inicializar o conteúdo da memória de instruções, armazenada em BRAMs, do processador MicroBlaze no *bitstream*.

O primeiro passo do fluxo de projeto é a síntese lógica que converte os ficheiros que descrevem o comportamento dos circuitos (HDL) numa *netlist* estrutural, um ficheiro NGC (*Native Generic Circuit*). A ferramenta Platgen (*Platform Generation*) analisa os ficheiros HDL que descrevem os núcleos IP que constituem o projeto do sistema embebido, neste caso o núcleo dos controladores UART e do codec de áudio AC'97, e sintetiza cada um deles usando a ferramenta XST (*Xilinx Synthesis Technology*). São também gerados os ficheiros HDL de alto nível, os *wrappers* ou envelopes, de cada núcleo IP do sistema de hardware e o ficheiro de alto nível do projeto de hardware completo, que interliga todos os núcleos IP, barramento PLB, processador MicroBlaze e inclui os pinos de entrada e saída da FPGA. Para este projeto, é usado um pino para entrada do sinal de relógio e outro para o sinal de *reset*. Para o controlador UART é usado um pino de entrada para receção (Rx) e um de saída para envio (Tx). Por fim, para o controlador do AC'97 são usados dois pinos como entrada, um para o sinal de relógio (BITCLK) e outro para a entrada de dados em série (SDATA\_IN), e três pinos como saída, um para o sinal de *reset* (RESET\_N), um para a saída de dados em série (SDATA\_OUT) e um sinal de sincronismo (SYNC). Estes cinco sinais, duas entradas e três saídas, fazem a parte da interface série digital, o *AC-Link*, que permite ao controlador comunicar com o *chip* do codec de áudio e vice-versa.

O segundo passo é a implementação do projeto que converte a *netlist* NGC gerada durante o processo de síntese num ficheiro NCD (*Native Circuit Description*), que é uma representação física do projeto de hardware mapeado em componentes da arquitetura da FPGA alvo, neste caso a Spartan6. Posteriormente, a ferramenta PAR (*Place and Route*) produz um ficheiro NCD com os componentes necessários alocados e o encaminhamento de interligações da FPGA para a especificação do projeto de hardware.

A geração do ficheiro de *bitstream* é o terceiro passo do fluxo de projeto. A ferramenta Bitgen recebe um ficheiro NCD já com a alocação de componentes e encaminhamento de interligações, e produz o ficheiro de configuração ou programação da FPGA que descreve a lógica interna da mesma para o projeto de hardware especificado. Uma vez que o projeto implementa um processador *soft-core* na FPGA, é necessário inicializar o conteúdo da sua memória de instruções no *bitstream*. A ferramenta Bitinit recebe três ficheiros distintos: um ficheiro ELF (*Executable and Linkable Format*) que contém as instruções a executar no processador, um ficheiro MHS (*Microprocessor Hardware Specification*) que define as configurações do sistema embebido: a arquitetura do barramento, os periféricos, o processador, interligações entre componentes e o seu espaço de endereços e, por fim, um ficheiro BMM (*Block RAM Memory Map*) que contém o mapa de endereços e a localização da memória de instruções do processador. A ferramenta Bitinit, por sua vez,

invoca a ferramenta Data2MEM que permite inicializar o conteúdo de BRAMs da FPGA, onde a memória de instruções do processador é alocada.

## 5.7 Comunicação com hardware e processamento de dados

A comunicação entre a aplicação de software e a plataforma de hardware é do tipo série da variante RS-232. Para inicializar, simular e ler o estado do AC, o meio de comunicação permite enviar comandos da aplicação e enviar/receber a informação dos bits de estado das células ao fim de  $n$  iterações. O protocolo que implementa as regras de comunicação entre a aplicação e o hardware está especificado no capítulo 4, na secção 4.5. Desta forma, nesta secção discutem-se as soluções de processamento e organização dos dados para o envio e receção, de e para o painel gráfico da representação visual do AC.

Na figura 5.12 apresenta-se a interface gráfica após o estabelecimento da comunicação série. Em comparação com a figura 5.1, repare-se que não é possível alterar as opções de especificação das características dos AC (área um do conjunto de elementos gráficos) mas o acesso às funcionalidades de simulação estão agora disponíveis (área três do conjunto de elementos gráficos). Da mesma forma, estabelecida a comunicação série, não é permitido ao utilizador especificar novas arquiteturas celulares, nem gerar ficheiros de *bitstream* e nem programar a FPGA, pelo que para voltar a ter acesso a estas funcionalidades a comunicação série deve ser desligada em primeiro lugar. Após estabelecida a comunicação, o utilizador pode especificar o número máximo de iterações, ativar/desativar o modo passo-a-passo e/ou a reprodução sonora do estado do AC e, por fim, iniciar a simulação e analisar os resultados.

No capítulo 4 de especificação de arquitetura de hardware, explicou-se como é que os bits de estado das células do AC estão organizados em memória. Cada byte em memória contém bits de estado de uma até a oito células, conforme a especificação fornecida para número de bits de estado, o que significa que para um bit de estado por célula um byte transporta o estado de oito células, para dois transporta o estado de quatro, para quatro transporta o estado de duas e para oito transporta o estado de uma. De facto, a razão  $8/b$ , em que  $b$  é o número de bits de estado por célula, indicaria o número de células que um byte transporta. Contudo, esta razão só é válida quando  $b$  é um divisor de 8 pelo que só seriam suportados 1, 2, 4 e 8 bits de estado por célula. Relembre-se que a arquitetura de hardware especifica que a largura da matriz do AC deve ser um múltiplo de 8 mas não especifica qualquer limitação quanto ao número de bits por célula. Desta forma, se  $b$  não for um divisor de 8, acontece que os bits de estado de uma célula podem encontrar-se em bytes distintos, o que leva a realizar algum processamento adicional na aplicação para organizar os dados devidamente tanto para o envio como para a receção. Do lado do hardware,  $b$  não tem qualquer impacto no processamento, uma vez que é orientado ao byte e  $c$ , o número de colunas da matriz do AC, é múltiplo de 8 o que leva a que  $(c \times b)$ , o número de bits de uma linha, seja necessariamente um múltiplo de 8 e  $(c \times b)/8$ , o número de bytes a transmitir/receber, seja um número inteiro apesar de os bits de estado das células poderem encontrar-se divididos entre dois bytes distintos. Relembre-se também que os dados em memória estão numa sequência invertida,

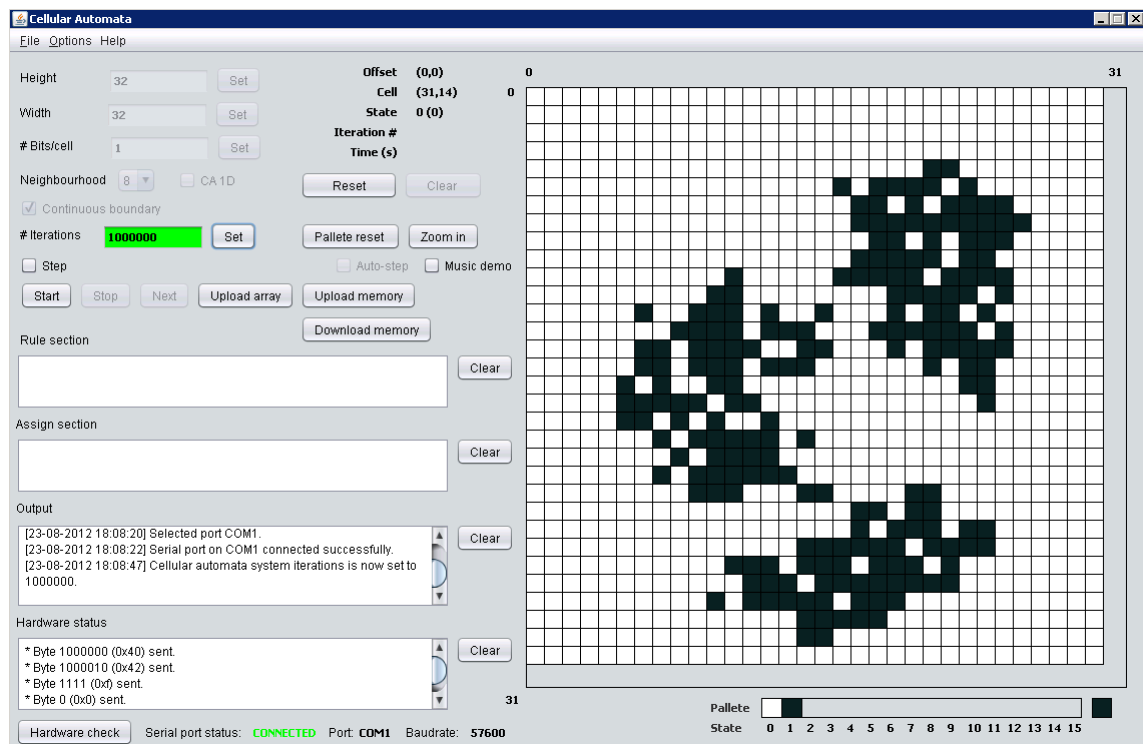


Figura 5.12: Interface gráfica da aplicação após estabelecida a comunicação série

peço que à medida que se avança na leitura ou escrita das memórias, a partir do endereço zero, lê-se o conteúdo da matriz do AC de baixo para cima e da direita para esquerda, isto é, dos bits menos para os mais significativos.

A transmissão/recepção é feita byte a byte e o processamento é realizado ao nível do bit. Designem-se por  $b = b_{n-1} \cdots b_0$ , em que  $n \geq 1$ , apontadores para os  $n - 1$  bits de estado de uma célula em que  $b_i$  é o bit de ordem  $i$  e por  $s = s_7 \cdots s_0$  apontadores para os bits do byte a enviar/receber, em que  $s_j$  é o bit de ordem  $j$  e por  $e(r, c)$  a célula atual cujos bits de estado recebidos são atribuídos ou enviados. Designem-se também por  $C$  e  $R$  constantes que designam o número de colunas e linhas da matriz do AC, respetivamente. Note-se que as coordenadas  $(0, 0)$  localizam-se no canto superior esquerdo da matriz.

O algoritmo 1 implementa o procedimento para filtrar os bits de estado de célula de cada byte recebido e atribuí-los às células respetivas. Para cada bit do byte recebido, avaliado em cada passagem do ciclo **while** da linha 10, é aplicada uma máscara de forma a obter apenas o bit de ordem  $j$ . O resultado dado por  $d$  dá o número de deslocamentos ao nível do bit a efetuar em valor absoluto, uma vez que  $i > j$  quando  $n > 8$ , para atribuir o bit  $s_j$  ao bit de ordem  $i$ ,  $b_i$ , da variável temporária  $b$  que guarda do estado da célula  $e(r, c)$  enquanto os seus bits de estado estão a ser processados. As condições entre a linha 13 e a linha 19 determinam o bit  $b_i$  a que o bit  $s_j$  do byte mascarado deve ser atribuído. A atribuição é feita com uma operação lógica do tipo OU entre  $b$  e o byte  $s$  mascarado. Uma vez que os apontadores  $s_j$  e  $b_j$  podem não estar alinhados, há que deslocar o bit  $s_j$  de  $d$  unidades para a esquerda ou para a direita consoante o avanço dos

apontadores. O desalinhamento ocorre logo no primeiro byte se  $n < 8$  ou a partir do segundo byte se  $n > 8$ . Como  $n$  indica o número de bits de estado de cada célula, assim que  $i = n$ , condição verificada na linha 21, os  $n$  bits de estado em  $b$  podem ser atribuídos à célula atual, uma vez que já foram processados. A variável  $b$  é repostada a zero e o apontador de ordem  $i$  é reinicializado para receber os bits de estado da célula seguinte. A sequência inversa da chegada dos dados faz com que se inicie a atribuição dos bits de estado à célula  $e(r = R - 1, c = C - 1)$ , recuando até à célula da origem  $e(0, 0)$ . A condição da linha 24 verifica se os bits de estado de uma linha completa do AC já foram recebidos e, se for verdade, o índice de coluna  $c$  é repostado na coluna mais à direita e  $r$  é decrementado em uma linha, caso contrário apenas se recua uma célula na matriz na mesma linha dada pelo índice  $r$ . O processo de leitura e atribuição dos bits de estado às células do AC de software termina quando o índice de linha  $r$  for decrementado de  $0 \rightarrow -1$  que significa que todas as células do AC de software foram atualizadas. O painel do visualizar gráfico é atualizado com o novo estado do AC no fim da receção.

O processo de inicialização do estado do AC em hardware segue a mesma linha de orientação, pelo que a aplicação implementa o algoritmo 2 que realiza a operação inversa. Desta vez, os bits de estado das células são processados e o byte a enviar vai sendo construído bit a bit. Aos bits de estado  $e(r, c)_i$  é aplicada uma máscara para obter o bit de ordem  $i$  e atribuí-lo ao bit  $s_j$  do byte a transmitir. Quando  $i = n$  avança-se para a célula seguinte, da mesma forma que no algoritmo 1 de receção, e ao fim de  $j = 8$  bits de estado processados, o byte  $b$  é enviado. As chamadas ao algoritmo terminam assim que  $r = -1$ , também. É de salientar que para ambos os algoritmos funcionarem corretamente, as variáveis  $r$ ,  $c$  e  $i$  são estáticas o que significa que só são inicializadas uma única vez e mantêm o seu valor entre chamadas consecutivas ao processamento.

**Algorithm 1** Atribuição dos bits de estado às células respectivas

---

```

1: procedure RECEBER( $s, n$ )
2:    $b \leftarrow 0$ 
3:    $j \leftarrow 0$ 
4:    $i \leftarrow 0$ 
5:    $r \leftarrow R - 1$ 
6:    $c \leftarrow C - 1$ 
7:   if  $r < 0$  then
8:     return
9:   end if
10:  while  $j < 8$  do
11:     $d \leftarrow |j - i|$ 
12:     $s_j \leftarrow s \wedge (1 \ll j)$ 
13:    if  $j > i$  then
14:       $b_i \leftarrow (s_j \gg d)$ 
15:    else if  $j < i$  then
16:       $b_i \leftarrow (s_j \ll d)$ 
17:    else
18:       $b_i \leftarrow s_j$ 
19:    end if
20:     $i \leftarrow i + 1$ 
21:    if  $i = n$  then
22:       $e(r, c) \leftarrow b$ 
23:       $c \leftarrow c - 1$ 
24:      if  $c < 0$  then
25:         $c \leftarrow C - 1$ 
26:         $r \leftarrow r - 1$ 
27:      end if
28:       $b \leftarrow 0$ 
29:       $i \leftarrow 0$ 
30:    end if
31:     $j \leftarrow j + 1$ 
32:  end while
33: end procedure

```

---

▷ variável estática

▷ variável estática

▷ variável estática

▷ todas as linhas recebidas

▷ máscara de 1 bit em  $s_j$ 

▷ se for verdade, uma linha foi recebida

▷ ir para a coluna mais à direita ...

▷ ... e ir para a linha anterior

---

**Algorithm 2** Envio dos bits de estado das células do AC da aplicação
 

---

```

1: procedure ENVIAR( $c$ )
2:    $j \leftarrow 0$ 
3:    $i \leftarrow 0$  ▷ variável estática
4:    $r \leftarrow R - 1$  ▷ variável estática
5:    $c \leftarrow C - 1$  ▷ variável estática
6:   if  $r < 0$  then ▷ todas as linhas enviadas
7:     return
8:   end if
9:   while  $j < 8$  do
10:     $e(r, c)_i \leftarrow e(r, c) \wedge (1 \ll i)$  ▷ máscara de 1 bit em  $c(r, c)_i$ 
11:     $s_j \leftarrow e(r, c)_i$ 
12:     $i \leftarrow i + 1$ 
13:    if  $i = n$  then
14:       $c \leftarrow c - 1$ 
15:      if  $c < 0$  then ▷ se for verdade, uma linha foi recebida
16:         $c \leftarrow C - 1$  ▷ ir para a coluna mais à direita ...
17:         $r \leftarrow r - 1$  ▷ ... e ir para a linha anterior
18:      end if
19:       $i \leftarrow 0$ 
20:    end if
21:     $j \leftarrow j + 1$ 
22:  end while
23:  enviar  $b$ 
24: end procedure

```

---

## Capítulo 6

# Resultados

Neste capítulo, apresentam-se os resultados obtidos em termos de desempenho e recursos da FPGA ocupados para diferentes modelos de ACs. Na secção 6.1, mostram-se as expressões que definem o tempo necessário para carregar a matriz do AC a partir da memória de entrada e lê-la para a memória de saída, o tempo necessário para transmitir ou receber toda a informação de estado das células do AC e o número de iterações que o AC consegue realizar por segundo. Na secção 6.2, apresentam-se os resultados em termos de recursos ocupados e desempenho no que diz respeito a diversas especificações de ACs (dimensões, vizinhança, número de bits por célula, condição fronteira). Por fim, na secção 6.3, faz-se a comparação direta do desempenho de um AC do Jogo da Vida implementado em hardware com uma aplicação de software que realize a mesma simulação e nas mesmas condições.

### 6.1 Resultados gerais

No capítulo 4 foi feita a especificação da arquitetura do sistema de hardware, onde de foi descrito o funcionamento dos circuitos que permitem ler a informação de estado das células do AC e carregá-las para a matriz a partir da memória de entrada e o processo inverso, isto é, ler o estado do AC e carregá-lo na memória de saída. Relembre-se que em cada uma das operações são usados dois *shift-registers* (SR) para a conversão da organização de dados. O número de registos (*flip-flops*) que cada SR tem é equivalente do número total de bits de uma linha da matriz e a cada ciclo de relógio o SR é deslocado de 8 bits de uma só vez. Designe-se por  $c$  e  $r$  o número de colunas e linhas da matriz do AC, respetivamente, e por  $b$  o número de bits de estado por célula. O número total de bytes em memória para carregar ou ler o AC é dado por  $(c \times r \times b)/8$  e o comprimento total de cada SR é de  $(c \times b)$  bits com  $(c \times b)/8$  andares de 8 bits cada. Como cada SR é carregado, para ler da memória de entrada, e descarregado, para escrever na memória de saída, byte a byte, são necessários  $(c \times b)/8$  ciclos de relógio para cada operação que se repete  $r$  vezes, uma vez que é carregada uma linha na matriz do AC por cada ciclo de relógio. Os multiplexadores auxiliares

são pura lógica combinacional pelo que não são contabilizados. Assim, o número de ciclos de relógio  $C_{in}$  para carregar a matriz do AC a partir da leitura da memória de entrada é dado por

$$C_{in} = 1 + \frac{c \times b}{8} \cdot (r + 1) \quad (6.1)$$

e  $C_{out}$ , o número de ciclos de relógio para ler a matriz do AC e escrever a memória de saída, é dado por:

$$C_{out} = r \cdot \left( 1 + \frac{c \times b}{8} \right) \quad (6.2)$$

Na equação 6.1, o primeiro termo indica que é necessário um ciclo de relógio para ler um byte da memória e só é contabilizado uma vez, quando o SR é carregado pela primeira vez durante o processo de inicialização do AC. Apesar de o número de andares do SR ser igual a  $(c \times b)/8$ , cada byte lido da memória é guardado num registo auxiliar e que, de facto, é também contabilizado como um andar adicional do SR e por cada linha do AC que é carregada, no mesmo ciclo de relógio, o byte presente neste registo auxiliar é carregado para o primeiro andar efetivo do SR, o que justifica a contabilização única do primeiro termo da equação. O processo de deslocamento dos andares do SR ocorre  $r$  vezes, para cada linha da matriz do AC, após o primeiro carregamento dos andares do SR. As últimas duas condições são dadas pelo segundo termo.

O processo de leitura do AC, por outro lado, é mais demorado apesar de não haver um ciclo de relógio de latência ao escrever dados na memória de saída. Uma vez que cada linha da matriz do AC é carregada em paralelo no SR, é necessário esperar que os seus bytes sejam todos escritos em memória para poder ler a linha seguinte sem ocorrer perda de dados. Assim, a equação 6.2 indica que são necessários  $r$  ciclos de relógio para ler todas as  $r$  linhas e, por cada uma, são necessários  $(c \times b)/8$  ciclos de relógio para escrever os bytes do SR em memória.

Na tabela 6.1, apresentam-se os resultados, por aplicação destas expressões, de algumas configurações de ACs.

| $r \times c$ | $C_{in}$ | $C_{out}$ | $C_{in}/C_{out}$ |
|--------------|----------|-----------|------------------|
| 8x8          | 10       | 16        | 62,50%           |
| 16x16        | 35       | 48        | 72,92%           |
| 24x24        | 76       | 96        | 79,17%           |
| 32x32        | 133      | 160       | 83,13%           |
| 40x40        | 206      | 240       | 85,83%           |
| 48x48        | 295      | 336       | 87,80%           |
| 56x56        | 400      | 448       | 89,29%           |
| 64x64        | 521      | 576       | 90,45%           |
| 72x72        | 658      | 720       | 91,39%           |
| 80x80        | 811      | 880       | 92,16%           |
| 88x88        | 980      | 1056      | 92,80%           |
| 96x96        | 1165     | 1248      | 93,35%           |

Tabela 6.1: Tempo necessário para ler e carregar a matriz do AC em ciclos de relógio com  $b = 1$ .

Repare-se na tabela 6.1 que quando as dimensões do AC aumentam, o tempo de leitura e carregamento tendem a aproximar-se. De facto quando  $r, c \rightarrow +\infty$  tem-se

$$C = C_{in} = C_{out} \approx \frac{r \times c \times b}{8} \quad (6.3)$$

e com  $r = c \wedge b = 1$ , o número de ciclos de relógio cresce quadraticamente com  $x^2/8$ . O facto de  $C_{in}$  e  $C_{out}$  se igualarem no infinito, deve-se a que para grandes dimensões do AC, o tempo de espera para a leitura de uma linha tende a ser o mesmo que o tempo necessário para o carregamento do andares do SR a partir da memória de entrada. Além disso, o byte que é carregado no SR ao mesmo tempo que uma linha é carregada no AC, que se traduz no ganho de um ciclo de relógio, deixa de ter qualquer relevância para o tempo total. Por outro lado, se se fixar  $r$ , com  $r \neq c$ , o crescimento é linear com  $c$ .

As memórias de entrada e saída tem capacidade para armazenar 4096 bytes, isto é,  $4096 \times 8 = 32768$  bits de estado de célula. Se  $b = 1$ , é possível gerar 32768 células e se a matriz de o AC for quadrada, então tem-se que  $c_{max} = r_{max} = \sqrt{32768} \approx 181$ . Para as especificações de ACs que foram implementadas (ver secção 6.2), nunca foi possível atingir estas dimensões pois a percentagem de recursos ocupados da FPGA ultrapassava os 100% para dimensões inferiores. Apesar de não ser um dado adquirido, isto é, de não se ter confirmado, dado o número elevado de combinações possíveis para regras de transição de estado, se é possível ou não atingir as dimensões referidas, crê-se que, para um AC bidimensional, mesmo implementando a regra mais simples de todas, com uma vizinhança de von Neumann e um bit de estado por célula, não é possível atingir tais dimensões dados os recursos necessários para gerar a arquitetura de hardware especificada.

O envio e receção de dados é feito por comunicação série, variante RS-232, com uma velocidade de transmissão de  $B = 460800$  bps (bits por segundo). A *baudrate*  $B$  é limitada ao valor referido porque além de ser um dos valores permitidos para a configuração do controlador UART, sendo o valor imediatamente acima de 921600 bps, é também a velocidade máxima que este tolera. Desta forma, o tempo de transmissão total  $T_{B_0}$ , em segundos, vem em função de  $r$ ,  $c$  e  $b$  e é dado por

$$T_{B_0} = \frac{c \times r \times b}{B} \quad (6.4)$$

É claro que a equação 6.4 só indica o tempo total efetivo de transmitir bits pelo canal. Do lado do processador do MicroBlaze e da aplicação de software em Java, existem *overheads* de processamento adicionais  $\Delta t = \Delta t_{MB} + \Delta t_{SW}$ , respetivamente, para organizar os dados em bytes para o envio ou receção e fazer leituras e escritas em memória (equação 6.5).

$$T_B = T_{B_0} + \Delta t \quad (6.5)$$

Contudo, quando  $T_{B_0} \gg \Delta t$ , a influência destes *overheads* torna-se insignificante estando o *bottleneck* localizado na transmissão pelo canal. Tem-se também  $\Delta t_{MB} < \Delta t_{SW}$ , uma vez que na aplicação de software é preciso aceder a um vetor de comprimento  $r \times c$  tanto como para leitura e

escrita e realizar operações sobre os bits de estado das células, sendo este processo mais lento do que ler e escrever em memória por parte do processador MicroBlaze em hardware, uma vez que não é necessário realizar processamento adicional sobre os bytes recebidos e enviados. O núcleo de hardware do controlador UART tem um *buffer* (FIFO) de entrada e saída de 16 bytes cada e verificou-se que a transmissão do hardware para a aplicação de software é mais lenta do que no sentido inverso, na medida em que o *buffer* de saída enche rapidamente, se se enviar vários bytes seguidos, e a aplicação Java não consegue processar à mesma cadência. Desta forma, optou-se por colocar em fila de espera um byte de cada vez, e só quando a transmissão desse byte é concluída é que se coloca o seguinte. Crê-se que a limitação se encontre na aplicação de software e na API que implementa o acesso ao hardware do RS-232, uma vez que o controlador USB-UART para a FPGA suporta velocidades até 12 Mbps e tem um *buffer* (FIFO) de entrada e saída de 384 e 128 bytes, respetivamente.

Na tabela 6.2, apresentam-se os tempos de transmissão, por aplicação da equação 6.4, de algumas configurações de ACs.

| $r \times c$ | T (ms) |
|--------------|--------|
| 8x8          | 0,14   |
| 16x16        | 0,56   |
| 24x24        | 1,25   |
| 32x32        | 2,22   |
| 40x40        | 3,47   |
| 48x48        | 5,00   |
| 56x56        | 6,81   |
| 64x64        | 8,89   |
| 72x72        | 11,25  |
| 80x80        | 13,89  |
| 88x88        | 16,81  |
| 96x96        | 20,00  |

Tabela 6.2: Tempo necessário, em milissegundos, para transmitir os bits de estado das células do AC, em função de  $c$ ,  $r$  com  $B = 460800$  e  $b = 1$ .

Por fim, refere-se que na arquitetura de hardware proposta é possível calcular uma nova iteração do AC em apenas um ciclo de relógio. Assim sendo, se a frequência do sinal de relógio for  $f_{CLK}$ , então, por segundo, são realizadas também  $f_{CLK}$  iterações. Então, para  $n$  iterações, o tempo efetivo de simulação  $T_{HW}$  em hardware, em segundos, é dado por

$$T_{HW} = \frac{n}{f_{CLK}} \quad (6.6)$$

## 6.2 Desempenho e utilização de recursos

Na tabela 6.4, apresentam-se os recursos ocupados, e a sua percentagem face aos recursos disponíveis (ver tabela 6.3), na FPGA pelos núcleos que constituem o sistema de hardware que

não sofrem alterações para especificações de ACs distintas.

| Recursos FPGA Spartan6 |       |       |
|------------------------|-------|-------|
| <i>Flip-flops</i>      | LUTs  | BRAMs |
| 27288                  | 54576 | 116   |

Tabela 6.3: Recursos disponíveis na FPGA Spartan6

| Núcleo          | Descrição                                                                                | <i>Flip-flops</i> |        | LUTs |        | BRAMs |         |
|-----------------|------------------------------------------------------------------------------------------|-------------------|--------|------|--------|-------|---------|
|                 |                                                                                          | #                 | %      | #    | %      | #     | %       |
| xps_timer       | Temporizador e contador                                                                  | 364               | 0,667% | 343  | 1,257% | 0     | 0%      |
| debug_module    | Módulo de <i>debug</i> do processador                                                    | 123               | 0,225% | 126  | 0,462% | 0     | 0%      |
| proc_sys_reset  | Módulo de <i>reset</i> do processador e barramento PLB e LMB ( <i>Local Memory Bus</i> ) | 69                | 0,126% | 54   | 0,198% | 0     | 0%      |
| clock_generator | Gerador de sinais de relógio ou DCM ( <i>Digital Clock Manager</i> )                     | 0                 | 0%     | 1    | 0,004% | 0     | 0%      |
| rs232_uart_lite | Controlador UART                                                                         | 147               | 0,269% | 145  | 0,531% | 0     | 0%      |
| lmb_bram        | Memória BRAM de instruções e dados do processador                                        | 0                 | 0%     | 0    | 0%     | 32    | 27,586% |
| ilmb_cntlr      | Controlador do LMB para memória de instruções                                            | 2                 | 0,004% | 6    | 0,022% | 0     | 0%      |
| dlmb_cntlr      | Controlador do LMB para memória de dados                                                 | 2                 | 0,004% | 6    | 0,022% | 0     | 0%      |
| dlmb            | Barramento de dados LMB                                                                  | 1                 | 0,002% | 0    | 0%     | 0     | 0%      |
| ilmb            | Barramento de instruções LMB                                                             | 1                 | 0,002% | 0    | 0%     | 0     | 0%      |
| mb_plb          | Barramento PLB e interface entre o processador e os núcleos de hardware restantes        | 155               | 0,284% | 373  | 1,367% | 0     | 0%      |
| microblaze      | Processador <i>soft-core</i> MicroBlaze                                                  | 1665              | 3,051% | 1595 | 5,845% | 0     | 0%      |

Tabela 6.4: Recursos ocupados pelos núcleos do sistema de hardware

Nos capítulos 3 e 4 onde é feita a especificação geral do sistema e do sistema de hardware, respetivamente, só foram apresentados os núcleos de hardware que interagem diretamente com o núcleo de lógica e controlo do AC. Contudo, existem outros núcleos necessários para o funcionamento do sistema de hardware como um todo e são apresentados na tabela 6.4. As percentagens de recursos ocupados apresentadas na tabela, são valores estimados pela ferramenta de síntese XST antes do passo de implementação do sistema. Os valores reais, para cada núcleo, não são fornecidos nos relatórios produzidos pelas ferramentas nos passos e processos intermédios do *design flow*. Dos resultados apresentados, destaca-se o seguinte:

- O processador MicroBlaze utiliza 32 BRAMs para memória de dados e instruções, o que perfaz um total de 64K bytes reservados para o efeito. Para  $64K \approx 65535$  bytes, a configuração permitida é  $16384$  bits de profundidade por  $32$  bits de largura de cada porta de acesso. Uma vez que na Spartan6 cada BRAM é de  $18K \approx 16384$  bits, são necessárias  $(64K\text{Bytes} \times 8)/16384 = 32$  BRAMs. Uma vez que os recursos de memória não são críticos para a arquitetura do sistema de hardware, e há bastante memória disponível ( $116 \times 16384 \approx 1900K\text{bits}$ ), reservou-se espaço por excesso para as instruções e dados.
- O sistema de hardware utiliza na sua totalidade  $32 + 4 = 36$  BRAMs. As 4 BRAMs adicionais correspondem às memória de entrada (2) e saída (2) do módulo de lógica e controlo do AC. Relembre-se que cada uma dessas memórias tem 4096 bytes de profundidade, então  $(4096 \times 8)/16384 = 2$ .
- O MicroBlaze ainda consome uma fatia significativa (5,845%) das LUTs disponíveis da FPGA. Contudo, as potencialidades da linguagem C e o poder de processamento que este oferece, tira-se maior partido para realizar operações cujo processamento é mais dispendioso e a complexidade de implementação em hardware mais elevada.
- O controlador UART é já fornecido pela Xilinx e implementa uma interface com o barramento PLB. É uma versão mais leve do controlador padrão mas suficiente para o sistema implementado e para a funcionalidade pretendida, isto é, uma comunicação de baixo débito. Além disso, a percentagem de recursos ocupados não é significativa.

Nas tabelas 6.5, 6.6, 6.7, 6.8 e 6.9 apresentam-se os resultados obtidos em utilização de recursos da FPGA, em LUTs e *flip-flops* (FF) e desempenho para várias dimensões dos ACs bidimensionais do Jogo da Vida (ver secção 2.1.4.1), do modelo de incêndio forestal (ver secção 2.1.4.2), *lattice gases* (ver secção 2.3.2) e Greenberg-Hastings (ver secção 2.3.1). O última tabela mostra os resultados obtidos para dois ACs unidimensionais de Wolfram, as regras 30 e 110 (ver secção 2.1.3) em função da sua largura. Todas as tabelas apresentam os resultados do relatório produzido pela ferramenta de síntese (XST), que são valores aproximados antes do passo de implementação, e os resultados no que diz respeito à ocupação de recursos efetiva do sistema de hardware, dados obtidos do relatório produzido pela ferramenta PAR (*Place & Route*), isto é, após implementação, e a frequência de operação efetiva a que o sistema consegue operar, incluindo

todos os núcleos de hardware, que é determinada por análise estática temporal, pela ferramenta TRACE, de todos os caminhos dos circuitos e é dada pelo inverso do tempo de propagação do caminho crítico. A frequência de operação apresentada associada ao núcleo de lógica e controlo do AC, é a frequência máxima estimada para a operação isolada do mesmo. Não são apresentados os dados de ocupação de recursos efetiva apenas do núcleo de lógica e controlo do AC, uma vez que os relatórios não indicam tal informação.

Na tabela 6.5, as siglas FC e FN significam condição Fronteira Contínua e Fronteira Nula, respetivamente. Quando não indicado, os resultados referem-se sempre a uma condição fronteira do tipo contínua. Para dimensões superiores a 40x40, no passo de implementação a ferramenta PAR, mais concretamente na fase de encaminhamento, o encaminhador não conseguiu determinar uma solução satisfatória para a implementação de uma condição fronteira do tipo nula e para dimensões superiores a 56x56 não conseguiu independentemente da condição. Em ambas as situações, a percentagem de recursos ocupados excedia os 100% ou as restrições temporais não era satisfeitas. Observe-se que para as dimensões de 32x32 e 40x40, o número de *flip-flops* estimado e real é idêntico para condições fronteira distintas. Isto deve-se ao facto de o número de bits de estado de cada célula ser igual, isto é, o número de *flip-flops* por célula. Contudo, repare-se, que o número de LUTs utilizadas é distinto, sendo menor para uma condição fronteira do tipo nula. Uma vez que quando se tem uma condição fronteira do tipo contínua as células que se encontram na periferia da matriz do AC estão diretamente interligadas, é natural que seja necessário introduzir mais lógica para redirecionar os barramentos do que quando são simplesmente ligados à massa. Para dimensões iguais a 56x56, repare-se que a ferramenta de síntese estimou que a percentagem de recursos ocupados excedia os 100%, mas durante o passo de implementação, a ferramenta PAR conseguiu encontrar uma solução que satisfizesse as restrições associadas à arquitetura do sistema. À medida que as dimensões especificadas tomam valores maiores, a complexidade do AC aumenta e, consequentemente, faz sentido que a percentagem de recursos necessários para gerar o AC seja maior também, uma vez que a lógica de cada célula é replicada o número de vezes igual às dimensões do AC. No que diz respeito à frequência efetiva de operação do sistema, à medida que as dimensões do AC aumentam, os valores obtidos não crescem nem decrescem monotonicamente. Por exemplo, para dimensões de 16x16 tem-se aproximadamente 100 Mhz e para 8x8 e 24x24 tem-se frequências inferiores a esta. Não é trivial explicar este comportamento, uma vez que depende do processo realizado pelas ferramentas no passo de implementação, para determinar a melhor solução de arquitetura, o seu mapeamento na FPGA e cumprindo as restrições temporais impostas, isto é, como é que a ocupação e distribuição dos recursos e as interligações entre os elementos da FPGA devem ser feitas. De qualquer forma, o sistema de hardware trabalha a uma frequência de  $f_{CLK} \approx 66\text{Mhz}$ , que é imposta pelo núcleo de hardware que gera este sinal de relógio a partir do oscilador externo de 100 Mhz, e é a frequência de operação do barramento PLB que interliga todos os núcleos do sistema.

Os resultados apresentados para o AC do modelo de incêndio florestal (tabela 6.6) e Greenberg-Hastings (tabela 6.8), refletem a implementação de versões simplificadas destes modelos. Relembre-se que ambos os modelos introduzem fatores de natureza aleatória para determinar o estado se-

guinte de uma célula. Contudo, a arquitetura geral do núcleo de lógica e controlo do AC e a arquitetura das próprias células, não suportam a especificação de condições de carácter aleatório. Assim sendo, para estas implementações, o(s) fator(es) aleatório(s) não existe(m) o que se traduz numa aproximação grosseira ao comportamento descrito pelos modelos originais. A diferença entre os resultados obtidos entre cada uma destas quatro implementações (Jogo da Vida, modelo de incêndio florestal, *lattice gases* e Greenberg-Hastings), deve-se à complexidade da regra de transição de estado implementada em cada célula, para cada modelo, que permite gerar ACs de maiores dimensões se a complexidade for reduzida e de menores dimensões se a complexidade for elevada. A eficiência com que é implementada a regra também reflete-se na percentagem de recursos utilizados, no que diz respeito a LUTs e lógica adicional.

Para o AC de Greenberg-Hastings, relembrando o que foi descrito na secção 2.3.1, a arquitetura proposta pelos autores permitiu gerar ACs com 512x512 células. Contudo, devido à latência de acesso a memória para leitura e escrita, são necessários 512 ciclos de relógio para calcular uma iteração que se traduz-se num paralelismo efetivo de 128 células por ciclo para dimensões de 256x256 células, sendo a frequência de operação de 100 MHz. Na arquitetura proposta neste trabalho o paralelismo efetivo é de 100%, uma vez que é possível atualizar o estado de todas as células num único ciclo de relógio. Além disso, a frequência de operação aqui é de 66 MHz e ainda assim se obtém um ganho em desempenho já que  $\frac{512 \cdot n}{100 \cdot 10^6} > \frac{n}{66 \cdot 10^6}$ , sendo  $n$  o número de iterações. De facto, para o desempenho ser idêntico, a frequência do sinal de relógio tinha de ser 512 vezes mais rápida que os 66 MHz, ou seja, 33,792 GHz. Em contrapartida, as dimensões passíveis de serem geradas são bastante inferiores.

Os resultados obtidos para a regra 30 e 110 e Wolfram são apresentados na tabela 6.9. Para o comprimento do AC idêntico, a percentagem da ocupação de recursos estimada é a mesma. Após implementação (*Place & Route*), o número efetivo de *flip-flops* mantém-se igual, e, apesar do número efetivo de LUTs ser diferente, a diferença não é muito significativa. Em ACs unidimensionais, a regra de transição de estado pode ser descrita por uma LUT de três entradas, que corresponde às oito combinações possíveis de estado das duas células vizinhas e da célula central. Desta forma, até seria expectável que o número de LUTs fosse idêntico para o mesmo comprimento mas com regras diferentes. Contudo, uma vez que a percentagem de recursos ocupados pelos outros núcleos não se altera entre especificações de ACs diferentes, a diferença pode-se encontrar na forma como as ferramentas mapeiam a regra de cada célula nas LUTs da FPGA para estas duas regras em particular. Embora não se tenham realizado implementações destas regras para larguras maiores ( $L > 256$ ), é evidente, pela percentagem de recursos ocupados quando  $L = 256$ , que é certamente possível gerar ACs unidimensionais com um maior número de células.

### 6.3 Benchmarking

Para ter uma medida do desempenho do hardware para a simulação de ACs, é feita uma comparação ao nível do tempo de simulação, para realizar  $n$  iterações do AC, com uma aplicação de

software nas mesmas condições, isto é, capaz de simular o AC exatamente com as mesmas características. Obtém-se também uma medida do ganho de velocidade de simulação, ou *speed-up*, em hardware face a software. Por fim, mostra-se o tempo total que decorre desde o arranque de uma simulação, o envio do comando da aplicação de software para a FPGA, mais a realização de  $n$  iterações e até obter o estado final do AC novamente na aplicação, por receção dos bytes que contêm a informação de estado das células.

A aplicação de software escolhida para simular ACs é o XLife (ver subsecção 2.5.1), que permite obter o tempo de simulação ao fim de  $n$  iterações. Foi escolhido o AC do Jogo da Vida e as condições de simulação e as características do AC são as seguintes:

- As dimensões do AC utilizadas são 32x32, 40x40 e 56x56.
- Para cada especificação das dimensões, são realizadas  $n$  iterações tanto para condições fronteira contínua e nula e o AC é inicializado com o mesmo padrão.
- O tipo de vizinhança considerada é de Moore.
- O AC é iterado  $10^5$ ,  $10^6$ ,  $10^7$ ,  $10^8$  e  $10^9$  vezes.

Os tempos obtidos para o XLife, dados por  $T_{SW}$ , são apresentados na tabela 6.10 e aplicação foi executada numa máquina de processador Intel Core 2 Quad Q9400 a 2.66 Ghz e compilada com o fator de otimização mais elevado. Há duas observações a fazer quanto a estes resultados. A primeira, refere-se ao facto de que ao multiplicar sucessivamente o número de iterações por um fator de 10, o tempo de simulação tem um crescimento aproximadamente linear para as mesmas dimensões. A segunda, refere-se à diferença entre os tempos para dimensões idênticas mas com condições fronteira distintas. Repare-se que à medida que as dimensões e o número de iterações aumentam, a diferença referida também se vai intensificando. O facto de uma condição fronteira nula causar um maior atraso de simulação, pode dever-se à eventual existência de um *buffer* de células fictícias ou virtuais para definir o estado nulo ao longo das fronteiras. Esta situação traduz-se no alargamento da matriz do AC em uma célula para o seu exterior e acrescentar mais quatro células de canto, também virtuais. O acesso a este *buffer* adicionar poderia justificar o atraso adicional, que se vai acumulando ao longo das diversas iterações. O atraso pode também ser influenciado pela carga imposta ao processador no instante em que a simulação decorre, pelo que os tempos apresentados não são constantes em simulações, nas mesmas condições, consecutivas. Trata-se apenas de uma aproximação.

No que diz respeito ao tempo de simulação na FPGA, os resultados são apresentados na tabela 6.11. A equação 6.6, indica que o tempo de simulação efetivo  $T_{HW}$ , isto é, o tempo que o AC está efetivamente a iterar, é invariante com as dimensões e condição fronteira. De facto, se a cada ciclo de relógio é feita uma iteração, o tempo de simulação  $T_{HW}$  só depende do número total de iterações. O resultados apresentados referem-se à frequência de operação do sistema de  $f_{CLK} \approx 66\text{Mhz}$ .

Na tabela 6.12, mostra-se a relação entre os tempos de simulação  $T_{SW}/T_{HW}$ , isto é, o ganho obtido em desempenho, ou *speed-up*, na simulação em FPGA. Como o tempo de simulação na

FPGA não depende da condição fronteira nem das dimensões do AC, à medida de estas aumentam e o número de iterações também, o ganho obtido vem significativamente melhor. A diferença é notável a partir de  $n \geq 10.000.000$  iterações. No pior caso, para dimensões de 56x56 células, a diferença é abismal e é de 15 segundos para aproximadamente 45 minutos com um *speed-up* de 181. Em termos práticos, este é um resultado relevante e uma potencialidade a favor das simulações em FPGA ou hardware em geral face a aplicações de software. Contudo, uma vez não tendo conhecimento de como a aplicação está implementada, isto é, se esta tira ou não partido das arquiteturas de processadores *multi-core*, por utilização de *threads* para divisão de tarefas e execução em paralelo, é possível que os ganhos obtidos pudessem vir mais reduzidos face a aplicações que exploram o paralelismo. Note-se que a arquitetura de um AC proporciona paralelismo inerente e ao simular o seu comportamento de forma sequencial, o processamento torna-se bastante dispendioso pois é necessário recorrer a duplo *buffering* para não se perder informação.

Por fim, na tabela 6.13, apresentam-se os tempos que decorrem desde a ordem de início de simulação, a partir do comando que é enviado da aplicação de software para a FPGA, mais a realização de  $n$  iterações do AC até à visualização gráfica dos resultados, isto é, o estado final do AC que é transmitido por meio da comunicação série de volta para a aplicação. Assume-se que a matriz do AC na FPGA encontra-se inicializada, embora não influencie os resultados apresentados, uma vez que o tempo de simulação não depende do padrão de inicialização. Note-se que a estes tempos, além de virem em função de  $T_{HW}$  que é definido pelo tempo base ou efetivo de simulação, é-lhes acrescido o *overhead* de transmissão, acesso à memória de saída por parte do processador MicroBlaze e extração dos bits de estado dos bytes recebidos pela aplicação de software para a representação gráfica do AC. Assim, tem-se que

$$T_A = T_{HW} + T_{B_0} + \Delta t_{MB} + \Delta t_{SW} + C_{out} / f_{CLK} \quad (6.7)$$

em que  $\Delta t_{MB}$  e  $\Delta t_{SW}$  são os *overheads* de acesso a memória e processamento por parte do MicroBlaze e da aplicação de software, respetivamente,  $T_{B_0}$  é o tempo efetivo de transmissão pelo canal de comunicação série,  $C_{out}$  é o número de ciclos de relógio necessários para ler o estado das células da matriz do AC e escrever o seu conteúdo na memória de saída e  $f_{CLK}$  a frequência de operação do sistema. Comparando estes tempos com os tempos da tabela 6.11, observa-se que a diferença é apenas de algumas décimas. Contudo, à medida que as dimensões do AC aumentam, todos os termos de  $T_A$  aumentam também, com exceção de  $T_{HW}$ , uma vez que é necessário ler da memória de saída uma quantidade superior de bytes, é necessário processar mais bytes do lado da aplicação, o volume de bits a transmitir é maior e são necessários mais ciclos de relógio para ler a matriz do AC.

| Jogo da Vida |            |         |         |           |         |              |              |              |              |              |               |                |
|--------------|------------|---------|---------|-----------|---------|--------------|--------------|--------------|--------------|--------------|---------------|----------------|
|              | Ferramenta | Recurso | Unidade | Dimensões |         |              |              |              |              |              |               |                |
|              |            |         |         | 8x8       | 16x16   | 24x24        | 32x32        |              | 40x40        |              | 48x48         | 56x56          |
| Núcleo AC    | XST        | LUT     | #       | 957       | 2915    | 6143         | 10661        | 9868         | 16456        | 15503        | 23489         | 30499          |
|              |            |         | %       | 3,507     | 10,682  | 22,512       | 39,068       | 36,162       | 60,305       | 56,813       | 86,078        | <b>111,767</b> |
|              |            | FF      | #       | 373       | 585     | 925          | <b>1382</b>  | <b>1382</b>  | <b>1984</b>  | <b>1984</b>  | 2693          | 3542           |
|              |            |         | %       | 0,683     | 1,072   | 1,695        | <b>2,532</b> | <b>2,532</b> | <b>3,635</b> | <b>3,635</b> | 4,934         | 6,490          |
| Sistema HW   | XST        | Freq.   | Mhz     | 176,585   | 177,525 | 177,999      | 154,345      | 154,345      | 175,070      | 174,551      | 111,591       | 102,501        |
|              |            | LUT     | #       | 3606      | 5564    | 8792         | 13310        | 12517        | 19105        | 18152        | 26138         | 33148          |
|              |            |         | %       | 13,215    | 20,390  | 32,219       | 48,776       | 45,870       | 70,012       | 66,520       | 95,786        | <b>121,475</b> |
|              |            | FF      | #       | 2902      | 3114    | 3454         | <b>3911</b>  | <b>3911</b>  | <b>4513</b>  | <b>4513</b>  | 5222          | 6071           |
|              | %          |         | 5,317   | 5,706     | 6,329   | <b>7,166</b> | <b>7,166</b> | <b>8,269</b> | <b>8,269</b> | 9,568        | 11,124        |                |
|              | LUT        | #       | 3035    | 4945      | 8153    | 10114        | 9467         | 18449        | 17233        | 19675        | 24818         |                |
|              |            | %       | 11,122  | 18,122    | 29,878  | 37,064       | 34,693       | 67,608       | 63,152       | 72,101       | <b>90,948</b> |                |
|              | PAR        | FF      | #       | 2074      | 2286    | 2626         | <b>3083</b>  | <b>3083</b>  | <b>3685</b>  | <b>3685</b>  | 4394          | 5243           |
| %            |            | 3,800   | 4,189   | 4,812     | 5,649   | 5,649        | 6,752        | 6,752        | 8,051        | 9,607        |               |                |
|              | Freq.      | Mhz     | 78,598  | 100,311   | 74,091  | 85,727       | 75,267       | 72,770       | 72,401       | 70,136       | 69,305        |                |

Tabela 6.5: Recursos ocupados e desempenho do AC do Jogo da Vida em função das suas dimensões  $r \times c$

| Incêndio Florestal |            |         |         |           |         |         |         |         |         |         |        |
|--------------------|------------|---------|---------|-----------|---------|---------|---------|---------|---------|---------|--------|
|                    | Ferramenta | Recurso | Unidade | Dimensões |         |         |         |         |         |         |        |
|                    |            |         |         | 8x8       | 16x16   | 24x24   | 32x32   | 40x40   | 48x48   | 56x56   | 64x64  |
| Núcleo AC          | XST        | LUT     | #       | 701       | 1398    | 2704    | 4543    | 6883    | 9725    | 13126   | 16987  |
|                    |            |         | %       | 2,569     | 5,123   | 9,909   | 16,648  | 25,224  | 35,638  | 48,102  | 62,251 |
|                    |            | FF      | #       | 467       | 876     | 1554    | 2475    | 3662    | 5108    | 6794    | 8748   |
|                    |            |         | %       | 0,856     | 1,605   | 2,847   | 4,535   | 6,710   | 9,359   | 12,449  | 16,029 |
| Sistema HW         | XST        | Freq.   | Mhz     | 181,810   | 166,251 | 174,459 | 162,075 | 162,238 | 168,256 | 108,205 | 96,768 |
|                    |            | LUT     | #       | 3350      | 4047    | 5353    | 7192    | 9532    | 12374   | 15775   | 19636  |
|                    |            |         | %       | 12,276    | 14,831  | 19,617  | 26,356  | 34,931  | 45,346  | 57,809  | 71,958 |
|                    |            | FF      | #       | 2996      | 3405    | 4083    | 5004    | 6191    | 7637    | 9323    | 11277  |
|                    |            |         | %       | 5,490     | 6,239   | 7,481   | 9,169   | 11,344  | 13,993  | 17,083  | 20,663 |
|                    |            | LUT     | #       | 2744      | 3383    | 4486    | 6116    | 8211    | 10748   | 13679   | 17122  |
|                    |            |         | %       | 10,056    | 12,397  | 16,439  | 22,413  | 30,090  | 39,387  | 50,128  | 62,746 |
|                    | PAR        | FF      | #       | 2168      | 2577    | 3259    | 4176    | 5363    | 6809    | 8495    | 10449  |
|                    |            |         | %       | 3,972     | 4,722   | 5,971   | 7,652   | 9,827   | 12,476  | 15,565  | 19,146 |
|                    |            | Freq.   | Mhz     | 90,261    | 71,078  | 70,423  | 71,205  | 70,977  | 87,428  | 69,147  | 69,339 |
|                    |            |         |         |           |         |         |         |         |         |         |        |
|                    |            |         |         |           |         |         |         |         |         |         |        |

Tabela 6.6: Recursos ocupados e desempenho do AC do modelo de incêndio florestal em função das suas dimensões  $r \times c$

| Lattice Gases |            |            |         |           |         |         |         |         |         |        |        |        |
|---------------|------------|------------|---------|-----------|---------|---------|---------|---------|---------|--------|--------|--------|
|               | Ferramenta | Recurso    | Unidade | Dimensões |         |         |         |         |         |        |        |        |
|               |            |            |         | 8x8       | 16x16   | 24x24   | 32x32   | 40x40   | 48x48   | 56x56  | 64x64  | 72x72  |
| Núcleo AC     | XST        | LUT        | #       | 633       | 1485    | 2841    | 4720    | 7112    | 9954    | 13365  | 17252  | 21700  |
|               |            |            | %       | 2,320     | 5,442   | 10,411  | 17,297  | 26,063  | 36,478  | 48,978 | 63,222 | 79,522 |
|               |            | FF         | #       | 620       | 1449    | 2792    | 4648    | 7023    | 9898    | 13288  | 17193  | 21610  |
|               |            |            | %       | 1,136     | 2,655   | 5,116   | 8,517   | 12,868  | 18,136  | 24,348 | 31,503 | 39,596 |
|               |            | Freq.      | Mhz     | 174,109   | 160,380 | 161,762 | 167,112 | 167,025 | 147,452 | 78,183 | 66,854 | 57,424 |
|               |            | Sistema HW | XST     | LUT       | #       | 3282    | 4134    | 5490    | 7369    | 9761   | 12603  | 16014  |
| %             | 12,027     |            |         |           | 15,150  | 20,119  | 27,005  | 35,770  | 46,185  | 58,685 | 72,929 | 88,230 |
| FF            | #          |            |         | 3149      | 3978    | 5321    | 7177    | 9552    | 12427   | 15817  | 19722  | 24139  |
|               | %          |            |         | 5,770     | 7,289   | 9,750   | 13,150  | 17,502  | 22,770  | 28,982 | 36,137 | 44,230 |
| Sistema HW    | PAR        | LUT        | #       | 2671      | 3580    | 4859    | 6758    | 9142    | 11993   | 15394  | 19295  | 23870  |
|               |            |            | %       | 9,788     | 13,119  | 17,806  | 24,765  | 33,502  | 43,950  | 56,413 | 70,709 | 87,474 |
|               |            | FF         | #       | 2321      | 3150    | 4493    | 6349    | 8724    | 11599   | 14989  | 18894  | 23311  |
|               |            |            | %       | 4,253     | 5,772   | 8,233   | 11,633  | 15,985  | 21,253  | 27,464 | 34,620 | 42,713 |
|               |            | Freq.      | Mhz     | 88,324    | 80,834  | 75,019  | 73,062  | 80,141  | 76,781  | 76,652 | 69,382 | 77,172 |

Tabela 6.7: Recursos ocupados e desempenho do AC de *lattice gases* em função das suas dimensões  $r \times c$

| Greenberg-Hastings |            |            |         |           |         |         |         |         |
|--------------------|------------|------------|---------|-----------|---------|---------|---------|---------|
|                    | Ferramenta | Recurso    | Unidade | Dimensões |         |         |         |         |
|                    |            |            |         | 8x8       | 16x16   | 24x24   | 32x32   | 40x40   |
| Núcleo AC          | XST        | LUT        | #       | 1273      | 4033    | 8601    | 15127   | 23554   |
|                    |            |            | %       | 4,665     | 14,779  | 31,519  | 55,435  | 86,316  |
|                    |            | FF         | #       | 624       | 1458    | 2802    | 4665    | 7016    |
|                    |            |            | %       | 1,143     | 2,672   | 5,134   | 8,548   | 12,855  |
|                    |            | Freq.      | Mhz     | 179,791   | 164,004 | 159,854 | 157,406 | 132,469 |
|                    |            | Sistema HW | XST     | LUT       | #       | 3922    | 6682    | 11250   |
| %                  | 14,373     |            |         |           | 24,487  | 41,227  | 65,142  | 96,024  |
| FF                 | #          |            |         | 3153      | 3987    | 5331    | 7194    | 9545    |
|                    | %          |            |         | 5,777     | 7,305   | 9,768   | 13,182  | 17,489  |
| LUT                | #          |            |         | 3308      | 6056    | 10558   | 17071   | 23854   |
|                    | %          |            |         | 12,123    | 22,193  | 38,691  | 62,559  | 87,416  |
| PAR                | FF         | #          | 2325    | 3159      | 4503    | 6366    | 8717    |         |
|                    |            | %          | 4,260   | 5,788     | 8,251   | 11,664  | 15,972  |         |
|                    |            | Freq.      | Mhz     | 73,768    | 68,213  | 71,383  | 68,729  | 69,199  |

Tabela 6.8: Recursos ocupados e desempenho do AC de Greenberg-Hastings em função das suas dimensões  $r \times c$

| ACs unidimensionais de Wolfram |            |         |         |         |         |         |         |         |         |
|--------------------------------|------------|---------|---------|---------|---------|---------|---------|---------|---------|
|                                | Ferramenta | Recurso | Unidade | L = 64  |         | L = 128 |         | L = 256 |         |
|                                |            |         |         | R30     | R110    | R30     | R110    | R30     | R110    |
| Núcleo AC                      | XST        | LUT     | #       | 473     | 473     | 732     | 732     | 1117    | 1117    |
|                                |            |         | %       | 1,733   | 1,733   | 2,682   | 2,682   | 4,093   | 4,093   |
|                                |            | FF      | #       | 481     | 481     | 674     | 674     | 1057    | 1057    |
|                                |            |         | %       | 0,881   | 0,881   | 1,235   | 1,235   | 1,937   | 1,937   |
| Sistema HW                     | XST        | Freq.   | Mhz     | 174,703 | 174,703 | 173,732 | 173,732 | 170,155 | 170,155 |
|                                |            | LUT     | #       | 3118    | 3118    | 3377    | 3377    | 3762    | 3762    |
|                                |            |         | %       | 11,426  | 11,426  | 12,375  | 12,375  | 13,786  | 13,786  |
|                                |            | FF      | #       | 3006    | 3006    | 3199    | 3199    | 3582    | 3582    |
|                                |            |         | %       | 5,508   | 5,508   | 5,862   | 5,862   | 6,563   | 6,563   |
|                                | PAR        | LUT     | #       | 2497    | 2529    | 2677    | 2673    | 2972    | 2972    |
|                                |            |         | %       | 9,151   | 9,268   | 9,810   | 9,796   | 10,891  | 10,891  |
|                                |            | FF      | #       | 2178    | 2178    | 2371    | 2371    | 2754    | 2754    |
|                                |            |         | %       | 3,991   | 3,991   | 4,344   | 4,344   | 5,046   | 5,046   |
|                                |            | Freq.   | Mhz     | 91,743  | 84,832  | 83,724  | 87,078  | 79,936  | 79,936  |

Tabela 6.9: Recursos ocupados e desempenho dos ACs unidimensionais de Wolfram, regra 30 e 110, em função da sua largura L

| Dimensões | Fronteira | Iterações |        |        |         |          |
|-----------|-----------|-----------|--------|--------|---------|----------|
|           |           | $10^5$    | $10^6$ | $10^7$ | $10^8$  | $10^9$   |
| 32x32     | Contínua  | 0,063     | 0,504  | 5,984  | 59,172  | 590,703  |
|           | Nula      | 0,063     | 0,641  | 6,438  | 64,912  | 649,375  |
| 40x40     | Contínua  | 0,109     | 1,125  | 11,063 | 110,837 | 1103,500 |
|           | Nula      | 0,109     | 1,125  | 11,063 | 114,119 | 1114,375 |
| 56x56     | Contínua  | 0,266     | 2,500  | 24,940 | 251,802 | 2490,931 |
|           | Nula      | 0,261     | 2,734  | 27,141 | 271,203 | 2721,563 |

Tabela 6.10: Tempos efetivos de simulação  $T_{SW}$ , em segundos, de um AC do Jogo da Vida no XLife em função das suas dimensões.

| Dimensões | Iterações |        |        |        |        |
|-----------|-----------|--------|--------|--------|--------|
|           | $10^5$    | $10^6$ | $10^7$ | $10^8$ | $10^9$ |
| 32x32     | 0,0015    | 0,015  | 0,15   | 1,50   | 15,00  |
| 40x40     | 0,0015    | 0,015  | 0,15   | 1,50   | 15,00  |
| 56x56     | 0,0015    | 0,015  | 0,15   | 1,50   | 15,00  |

Tabela 6.11: Tempos efetivos de simulação  $T_{HW}$ , em segundos, de um AC do Jogo da Vida na FPGA em função das suas dimensões.

| Dimensões | Fronteira | Iterações |        |        |        |        |
|-----------|-----------|-----------|--------|--------|--------|--------|
|           |           | $10^5$    | $10^6$ | $10^7$ | $10^8$ | $10^9$ |
| 32x32     | Contínua  | 42        | 40     | 40     | 39     | 39     |
|           | Nula      | 42        | 43     | 43     | 43     | 43     |
| 40x40     | Contínua  | 73        | 75     | 74     | 74     | 74     |
|           | Nula      | 73        | 75     | 74     | 76     | 74     |
| 56x56     | Contínua  | 177       | 167    | 166    | 168    | 166    |
|           | Nula      | 187       | 182    | 181    | 181    | 181    |

Tabela 6.12: Ganhos em velocidade de simulação (*speed-up*)  $T_{SW}/T_{HW}$  aproximados de um AC do Jogo da Vida em função das suas dimensões.

| Dimensões | Iterações |        |        |        |        |
|-----------|-----------|--------|--------|--------|--------|
|           | $10^5$    | $10^6$ | $10^7$ | $10^8$ | $10^9$ |
| 32x32     | 0,032     | 0,045  | 0,182  | 1,531  | 15,030 |
| 40x40     | 0,060     | 0,073  | 0,205  | 1,556  | 15,056 |
| 56x56     | 0,164     | 0,176  | 0,311  | 1,661  | 15,159 |

Tabela 6.13: Tempos totais  $T_A$ , em segundos, desde o início de uma simulação em hardware até obter o resultado na aplicação de software, para um AC do Jogo da Vida em função das suas dimensões.

## Capítulo 7

# Conclusões e trabalho futuro

Neste capítulo, apresenta-se um breve resumo do trabalho realizado na secção 7.1, discute-se o problema do mapeamento da arquitetura do sistema de hardware em FPGA na secção 7.2 e são expostas algumas ideias para melhorar a arquitetura do sistema de hardware e a aplicação de software na secção 7.3.

### 7.1 Resumo do trabalho

O trabalho consistiu na especificação e projeto de dois tipos de sistemas que se complementam: um sistema de hardware e outro de software.

O sistema de hardware trata-se de uma arquitetura para gerar sistemas baseados em ACs em FPGA a partir de *templates* pré-construídos, que descrevem os circuitos da arquitetura base, e parametrizáveis externamente pelo utilizador do sistema para especificar as características e regras do AC pretendido. O controlo da operação, inicialização e leitura do AC ficam também a cargo do utilizador externo, enviando os comandos respetivos a partir do sistema de software. A comunicação entre os dois sistemas é do tipo série e é feita por um protocolo especificado para o efeito conhecido por ambos. O processador *soft-core* MicroBlaze processa os pedidos externos e aciona os sinais de controlo internos respetivos do núcleo de lógica e controlo do AC.

O sistema de software é uma aplicação de software em Java que corre num PC e de suporte ao sistema de hardware, que permite controlar totalmente a operação de uma especificação de um AC, ler e inicializar o seu estado a partir de uma representação gráfica que mapeia os estados das células em cores para fácil leitura e identificação. Todas as especificações de ACs realizadas ficam guardadas no diretório de trabalho, cujo formato dos ficheiros é binário (*bitstream*), e o utilizador pode programar qualquer *bitstream* na FPGA assim que entender.

A lista abaixo mostra a sequência de etapas para a especificação completa de ambos os sistemas.

1. Especificação da arquitetura celular.
2. Especificação da arquitetura do módulo de interligação das células.

3. Especificação da arquitetura do módulo de controlo de fronteira e transferência de dados.
4. Desenvolvimento do módulo de controlo global do sistema (máquina de estados).
5. Desenvolvimento do módulo de interface entre o MicroBlaze e o módulo de lógica e controlo do AC.
6. Especificação do protocolo de comunicação entre o sistema de hardware e software.
7. Desenvolvimento do código C para o MicroBlaze.
8. Desenvolvimento da aplicação de software em Java.
9. Integração vertical dos dois sistemas.
10. Desenvolvimento do modo de demonstração: reprodução sonora do estado do AC.

## 7.2 Mapeamento da arquitetura do sistema de hardware

A forma como é feito o mapeamento da arquitetura do sistema de hardware e a distribuição e alocação de recursos da FPGA para a lógica das células do AC, conduz à redução do desempenho global do sistema e pior aproveitamento dos recursos disponíveis para a implementação.

Embora o paralelismo efetivo do sistema seja máximo, uma vez que é possível atualizar o estado das células em apenas um ciclo de relógio, a proximidade espacial física das células depois da implementação do sistema não é uniforme e, na prática, o mapeamento do AC na FPGA não se traduz num mapeamento direto, isto é, não se tem de facto uma arquitetura puramente baseada em ACs ao nível físico. O grau de compactação de uma solução de arquitetura de hardware para uma especificação de um AC é tanto mais alto quanto mais próximas estão as células do AC fisicamente.

Sabendo que um sistema baseado em ACs caracteriza-se por interações localizadas entre células de uma dada vizinhança, quando é realizada a implementação do sistema por parte das ferramentas, nada nos garante que a proximidade física entre as células seja uniforme ao longo da FPGA uma vez que as ferramentas otimizam as implementações e fazem a distribuição de recursos de forma a satisfazer todas as restrições do sistema. Desta forma, a distância entre as células não é uniforme e as suas interligações têm comprimentos distintos, uns mais longos que outros, o que pode influenciar negativamente os tempos de propagação e diminuir o desempenho do sistema, isto é, baixar a frequência de operação. Relembre-se da secção 2.2, que a arquitetura de uma FPGA está dividida em vários blocos de lógica configurável (CLB) que podem ser interligados conforme as necessidades através de linhas de comunicação que se situam entre estes. Para garantir a proximidade espacial física das células aquando do mapeamento no dispositivo, um conjunto de células e a lógica associada devem ser mapeados diretamente num CLB e dentro destes na LUTs e *flip-flops* disponíveis. Se o procedimento se repetir para todos os CLBs da FPGA, a da distância física entre células é idêntica, ou parcialmente idêntica, e a uniformidade é máxima o que se

traduz também em desempenho máximo, isto é, a frequência máxima não seria limitada pela distribuição e mapeamento das células nos CLBs, mas sim pelas interligações entre estes. Por outro lado, a distribuição não uniforme origina a um mau aproveitamento dos recursos, na medida em que os *flip-flops* de um CLB podem conter os bits de estado de células que não se encontram na mesma vizinhança, e podem haver recursos do mesmo CLB que nem sequer são utilizados. Esta situação pode justificar os resultados obtidos no que diz respeito às dimensões máximas geradas para os ACs especificados. Os recursos de memória BRAM na FPGA podem também ser melhor aproveitados para o tratamento de condições fronteira, de forma a diminuir os caminhos de lógica combinacional, isto é, os barramentos que realizam a interligação direta entre as fronteiras da matriz do AC.

Para tirar o proveito máximo da arquitetura regular da FPGA, o mapeamento direto das células do AC nos CLBs da FPGA deve ser feito manualmente ou indicando às ferramentas de síntese e implementação onde é que cada célula deve ser colocada.

## 7.3 Trabalho futuro

Nesta secção apresentam-se propostas de trabalho a realizar no futuro, tanto para o sistema de hardware como para o sistema de software.

### 7.3.1 Propostas para o sistema de hardware

A arquitetura proposta para o sistema de hardware é uma solução que se encontra funcional. Contudo, deixam-se aqui algumas propostas para a expansão desta arquitetura que são apresentadas na lista abaixo.

**Mapeamento das células orientado à arquitetura da FPGA** Em iterações futuras, seria de procurar resolver os problemas de distribuição e alocação de recursos para a lógica das células do AC e das suas interligações na arquitetura de uma FPGA, de forma a garantir a proximidade física entre as células (maior compactação) e ao mesmo tempo procurar manter o elevado grau de paralelismo no cálculo do estado seguinte de cada célula.

**Suporte para qualquer número de células por linha** A única limitação da arquitetura do sistema de hardware, no que diz respeito à especificação de dimensões do AC, refere-se à largura da matriz do mesmo que deve ser sempre um múltiplo de 8 cuja restrição é de origem conceptual. Seria interessante desenvolver uma solução que permitisse especificar a largura da mesma forma que a altura da matriz, isto é, para qualquer valor inteiro superior a zero.

**Regras de transição mais complexas** Neste momento, a arquitetura celular só suporta a especificação da regra de transição baseada em operações aritméticas e lógicas entre os bits de estado. Para algumas aplicações, como foi visto, é necessário criar outros tipos de cenários, como por exemplo incluir fatores de natureza aleatória. A implementação de um LFSR,

que é um *shift-register* cuja entrada é uma função linear do estado anterior, permite gerar sequências pseudo-aleatórias e, por sua vez, modelar o comportamento de uma variável aleatória. Geralmente, os ACs são determinísticos ou probabilísticos e com um LFSR é possível assegurar a segunda característica [36] (ver subsecção 2.3.1).

**Leitura e inicialização do estado do AC por *bitstream*** Um ficheiro de *bitstream* contém a informação relativa à configuração a realizar numa FPGA e a inicialização das memórias BRAM e registos internos. Desta forma, a inicialização do estado do AC poderia ser feita diretamente no *bitstream* e, inversamente, o *bitstream* poderia ser lido da FPGA, num dado instante de tempo, e os valores atuais dos registos e dos endereços de memória poderiam ser obtidos. A principal vantagem deste método é que se poderia descartar todos os módulos de hardware relativos ao controlo e transmissão de dados, incluindo o MicroBlaze. Os recursos ocupados por estes módulos poderiam assim ser aproveitados para gerar ACs de dimensão superior. Por outro lado, sempre que se quisesse impor uma nova condição inicial ao AC, seria necessário gerar um novo ficheiro de *bitstream*. Seria necessário também estudar e entender o formato de um ficheiro *bitstream* para saber a localização exata dos *flip-flops* e posições de memória no ficheiro. Técnicas de reconfiguração dinâmica poderiam ser usadas para a reinicialização do estado inicial do AC a partir de ficheiros de *bitstream* parciais que modificariam apenas o estado dos *flip-flops* e posições de memória, deixando a lógica restante intacta. Esta solução teria que ser analisada com maior detalhe para avaliar a sua viabilidade.

**Suporte para vizinhanças maiores** As vizinhanças de Moore e von Neumann são as mais comuns em arquiteturas baseadas em ACs. Relembrando o que foi descrito nas subsecções 2.1.2 e 2.1.1, estas vizinhanças podem ser expandidas tomando  $r = \{1, 2, 3, \dots, n\} \in \mathbb{N}_+$ . Para algumas aplicações poderia ser interessante considerar uma área de interação local mais alargada.

**Expansão da arquitetura para ACs tridimensionais** Na secção 4.6 foi deixada uma referência à expansão da arquitetura para ACs tridimensionais. Como foi referido, a solução mais direta seria replicar o módulo que contém as células do AC  $l$  vezes, em que  $l$  indica a profundidade da terceira dimensão. Para inicializar e ler o estado de todas as  $l$  camadas, os módulos de hardware a cargo desta funcionalidade teriam que ser adaptados de forma a, por exemplo, multiplexar o acesso à terceira dimensão, atribuindo assim *slots* temporais para o acesso (leitura e inicialização) de todas as camadas.

### 7.3.2 Propostas para o sistema de software

Apesar da aplicação de software desenvolvida proporcionar as funcionalidades necessárias para controlar na totalidade a operação e permitir a inicialização, leitura e especificação de um AC, propõem-se aqui algumas funcionalidades adicionais e aspetos gerais de melhoria a implementar no futuro. A lista abaixo descreve estas propostas.

**Exportar/importar o estado de um AC** Neste momento, a inicialização de um AC tem que ser feita manualmente selecionado o estado desejado para cada célula. Da mesma forma, não é possível exportar o estado final de um AC, ao fim de  $n$  iterações, e guardá-lo no diretório de trabalho. Desta forma, propõe-se a exportação do estado de um AC em dois formatos distintos: imagem e ASCII. O segundo serviria também para importar o padrão de inicialização de um AC e atribuir a cada célula o estado especificado no ficheiro.

**Abstração do utilizador a Verilog HDL** A especificação da regra de transição de estado, neste momento, obriga ao utilizador ter conhecimentos mínimos de Verilog HDL. Assim sendo, propõe-se criar uma linguagem específica cuja sintaxe seja simples e intuitiva o suficiente que permita, facilmente, a um utilizador comum, isto é, sem quaisquer conhecimentos de projeto e linguagens de descrição de hardware, especificar a regra de transição de estado abstraindo-o de pormenores mais técnicos.

**Melhoria geral da interface gráfica** O desenvolvimento desta aplicação implicou estudar e relembrar alguns aspetos de linguagens orientadas a objetos e aprender a trabalhar com interfaces gráficas no geral, uma vez que a experiência de trabalho com tais aplicações era praticamente inexistente. Desta forma, o aspeto e organização final da interface gráfica e dos seus componentes não é o melhor. Propõe-se então uma melhor separação das funcionalidades que a aplicação oferece no que diz respeito à distribuição destas por menus, janelas secundárias e agrupamento de elementos gráficos, o que permite melhorar o aspeto no geral, organização e acessibilidade.

**Recolha de dados para estatística** Para algumas aplicações de ACs para o estudo de certos modelos, é interessante recolher dados sobre a evolução do AC ao longo do tempo para análise posterior. A forma como as estatísticas são realizadas, depende essencialmente da aplicação em questão.

**Melhoria da robustez de desempenho** A aplicação ainda apresenta alguns problemas no que diz respeito ao tratamento de erros que ocorrem em determinadas situações, por isso é necessário identificar estes problemas e corrigi-los. O código desenvolvido não está de todo otimizado, pelo que seria necessário fazer uma revisão geral para melhorar o desempenho global da aplicação.

**Melhoria do visualizador do AC e paletes de cor** O visualizador gráfico do AC está, neste momento, limitado à visualização de ACs de dimensões de 200x200 células, que é o máximo tolerado pela ferramenta de *zoom*. Além disso, uma vez que a evolução temporal dos ACs unidimensionais é representada na vertical, com a ferramenta de *zoom* só são representáveis 200 iterações do AC na mesma janela. Desta forma, para permitir representar em simultâneo um maior número de células para ACs bidimensionais e um maior número de iterações para ACs unidimensionais, deve existir uma janela secundária, de maior área visível que a deste momento, para que os estados das células continuem a ser perceptíveis e legíveis quando se faz o *zoom out* ao visualizador. É necessário também alargar a gama de cores

atual permitidas pela paleta de cores ativa. Provavelmente, mover também as ferramentas associadas à manipulação do estado das células para um menu independente ou janela distinta.

## 7.4 Considerações finais

Os sistemas de hardware e software foram implementados com sucesso, encontram-se 100% funcionais, tendo sido atingidos os objetivos iniciais propostos. Ainda assim, verificou-se que para melhorar o desempenho global do sistema de hardware, há a necessidade de procurar soluções alternativas que permitam resolver os problemas de distribuição e alocação de recursos da FPGA. No que diz respeito à aplicação de software, foram deixadas algumas propostas de trabalho para melhorar a interação com o utilizador, o aspeto e organização geral da interface gráfica e funcionalidades adicionais para enriquecer a aplicação.

Há ainda três aspetos de interesse a considerar. O primeiro refere-se à utilização do Micro-Blaze para correr uma série de simulações consecutivamente e automaticamente. No final, após a realização de todas as simulações, os resultados obtidos, isto é, o estado final do AC para  $n$  iterações para cada simulação, são enviados de volta para a aplicação de software para análise. Em vez de enviar uma nova condição inicial sempre que se pretende realizar uma nova simulação, a ideia seria carregar um conjunto condições iniciais designado pelo vetor de testes para simulação. O segundo aspeto, o mais relevante, refere-se à expansão do sistema de hardware para uma arquitetura multi-FPGA. Embora seja necessário, naturalmente, realizar uma análise detalhada de uma solução neste sentido para avaliar as restrições conceptuais, de operação e, particularmente, a forma como é realizada a transferência de dados entre FPGAs, a vantagem clara, à partida, é a possibilidade de gerar ACs de dimensões maiores. Para um sistema deste género, cada FPGA pode ter um mapeamento de memória e processadores independentes, o que facilita o acoplamento e desacoplamento de FPGAs, que são vistas como periféricos de expansão do sistema principal. O terceiro aspeto refere-se à implementação do sistema de hardware em tecnologia de FPGA mais recente, poderosa e de maior capacidade, como por exemplo a Virtex6. Com a resolução dos problemas de alocação e distribuição de recursos para uma utilização mais eficiente dos recursos disponíveis, a implementação numa Virtex6 é certamente uma mais valia.

# Referências

- [1] Eric W. Weisstein. Elementary Cellular Automaton. From MathWorld—A Wolfram Web Resource. <http://mathworld.wolfram.com/Rule30.html>.
- [2] Eric W. Weisstein. Elementary Cellular Automaton. From MathWorld—A Wolfram Web Resource. <http://mathworld.wolfram.com/Rule110.html>.
- [3] FPGA technology and dynamic reconfiguration. Em *Reconfigurable System Design and Verification*, páginas 15–44. CRC Press, Fevereiro 2009. 0. URL: <http://dx.doi.org/10.1201/9781420062670.ch2>.
- [4] Inc. Xilinx. *Command Line Tools User Guide*. Xilinx, Inc., Dezembro 2009.
- [5] Inc. Xilinx. *Spartan-6 Family Overview*. Xilinx, Inc., Outubro 2011.
- [6] José I. Barredo, Marjo Kasanko, Niall McCormick, e Carlo Lavalle. Modelling dynamic spatial processes: simulation of urban future scenarios through cellular automata. *Landscape and Urban Planning*, 64(3):145–160, 2003. URL: <http://www.sciencedirect.com/science/article/pii/S0169204602002189>, doi:10.1016/S0169-2046(02)00218-9.
- [7] Robert M. e Itami. Simulating spatial dynamics: cellular automata theory. *Landscape and Urban Planning*, 30(1–2):27–47, 1994. URL: <http://www.sciencedirect.com/science/article/pii/0169204694900655>, doi:10.1016/0169-2046(94)90065-5.
- [8] Bastien Chopard, Pierre antoine Quelo, e Pascal O. Luthi. Traffic models of a 2D road network, 1995.
- [9] B Chopard, P O Luthi, e P-A Quelo. Cellular automata model of car traffic in a two-dimensional street network. *Journal of Physics A: Mathematical and General*, 29(10):2325, 1996. URL: <http://stacks.iop.org/0305-4470/29/i=10/a=012>.
- [10] Sheng-Wei Guan e S.K. Tan. Pseudorandom number generation with self-programmable cellular automata. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(7):1095 – 1101, july 2004. doi:10.1109/TCAD.2004.829808.
- [11] Sastra Wijaya, Syn Kiat Tan, e Sheng-Wei Guan. Permutation and sampling with maximum length CA or pseudorandom number generation. *Applied Mathematics and Computation*, 185(1):312–321, 2007. URL: <http://www.sciencedirect.com/science/article/pii/S0096300306008629>, doi:10.1016/j.amc.2006.06.099.
- [12] Sheng-Wei Guan, Shu Zhang, e M.T. Quieta. 2-D CA variation with asymmetric neighborhood for pseudorandom number generation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(3):378–388, march 2004. doi:10.1109/TCAD.2004.823344.

- [13] Yoshihiko Kayama. Complex networks derived from cellular automata. *Network*, página 10, 2010. URL: <http://arxiv.org/abs/1009.4509>.
- [14] Ingo Kusch e Mario Markus. Mollusc Shell Pigmentation: Cellular Automaton Simulations and Evidence for Undecidability. *Journal of Theoretical Biology*, 178(3):333–340, 1996. URL: <http://www.sciencedirect.com/science/article/pii/S0022519396900296>, doi:10.1006/jtbi.1996.0029.
- [15] Danuta Makowiec. Cellular automata model of cardiac pacemaker. *Acta Physica Polonica B*, 39(5):1067 – 1085, 2008. URL: <http://search.ebscohost.com/login.aspx?direct=true&db=a9h&AN=32507628&site=ehost-live>.
- [16] Franco Bagnoli. From newton to cellular automata. *System*, páginas 1–21, 2009. URL: <http://arxiv.org/abs/0912.2056>.
- [17] G.E. Bridges e N.R.S. Simons. Extremely low-precision integer cellular array algorithm for computational electromagnetics. *Microwave and Guided Wave Letters, IEEE*, 9(1):1 –3, jan 1999. doi:10.1109/75.752105.
- [18] P.L. Rosin. Training cellular automata for image processing. *IEEE Transactions on Image Processing*, 15(7):2076–2087, july 2006. doi:10.1109/TIP.2006.877040.
- [19] V. Zhirnov, R. Cavin, G. Leeming, e K. Galatsis. An Assessment of Integrated Digital Cellular Automata Architectures. *Computer*, 41(1):38 –44, jan. 2008. doi:10.1109/MC.2008.4.
- [20] John Von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Champaign, IL, USA, 1966.
- [21] Eric W. Weisstein. von Neumann Neighborhood. From MathWorld—A Wolfram Web Resource. <http://mathworld.wolfram.com/vonNeumannNeighborhood.html>.
- [22] Eric W. Weisstein. Moore Neighborhood. From MathWorld—A Wolfram Web Resource. <http://mathworld.wolfram.com/MooreNeighborhood.html>.
- [23] Stephen Wolfram. *Cellular Automata And Complexity: Collected Papers*. Westview Press, 1994.
- [24] Stephen Wolfram. *A New Kind of Science*. Wolfram Media, 2002.
- [25] Stephen Wolfram. Universality and complexity in cellular automata. *Physica D: Nonlinear Phenomena*, 10(1–2):1 – 35, 1984. URL: <http://www.sciencedirect.com/science/article/pii/0167278984902458>, doi:10.1016/0167-2789(84)90245-8.
- [26] Eric W. Weisstein. Elementary Cellular Automaton. From MathWorld—A Wolfram Web Resource. <http://mathworld.wolfram.com/ElementaryCellularAutomaton.html>.
- [27] Stephen Wolfram. Statistical mechanics of cellular automata. *Rev. Mod. Phys.*, 55:601–644, Jul 1983. URL: <http://link.aps.org/doi/10.1103/RevModPhys.55.601>, doi:10.1103/RevModPhys.55.601.
- [28] Eric W. Weisstein. Universality. From MathWorld—A Wolfram Web Resource. <http://mathworld.wolfram.com/Universality.html>.
- [29] Eric W. Weisstein. Game of life. From MathWorld—A Wolfram Web Resource. <http://mathworld.wolfram.com/GameofLife.html>.

- [30] Eric W. Weisstein. Totallistic celular automaton. From MathWorld—A Wolfram Web Resource. <http://mathworld.wolfram.com/TotalisticCellularAutomaton.html>.
- [31] B. Drossel e F. Schwabl. Self-organized critical forest-fire model. *Phys. Rev. Lett.*, 69:1629–1632, Sep 1992. URL: <http://link.aps.org/doi/10.1103/PhysRevLett.69.1629>, doi:10.1103/PhysRevLett.69.1629.
- [32] Ed Jr. Pegg. Wireworld. From MathWorld—A Wolfram Web Resource, created by Eric W. Weisstein. <http://mathworld.wolfram.com/WireWorld.html>.
- [33] Inc. Xilinx. *Spartan-6 FPGA Configurable Logic Block*. Xilinx, Inc., Fevereiro 2010.
- [34] Inc. Xilinx. *Spartan-6 FPGA Block RAM Resources*. Xilinx, Inc., Julho 2011.
- [35] Inc. Xilinx. *Timing Analyzer Guide*. Xilinx, Inc.
- [36] Nikolaos Vlassopoulos, Nazim Fates, Hugues Berry, e Bernard Girau. An FPGA design for the stochastic Greenberg-Hastings cellular automata. Em *2010 International Conference on High Performance Computing and Simulation (HPCS)*, páginas 565 –574, 28 2010-july 2 2010. doi:10.1109/HPCS.2010.5547071.
- [37] Paul Shaw, Paul Cockshott, e Peter Barrie. Implementation of lattice gases using FPGAs. *The Journal of VLSI Signal Processing*, 12:51–66, 1996. 10.1007/BF00936946. URL: <http://dx.doi.org/10.1007/BF00936946>.
- [38] Jon Bennett, Chuck Silvers, Paul Callahan, Eric S. Raymond, e Vladimir Lidovski (2011-2012). XLife. <http://litwr2.atSPACE.eu/xlife.php>.
- [39] Andrew Trevorrow, Tom Rokicki, Tim Hutton, Dave Greene, Jason Summers, Maks Verver, e Robert Munafo. Golly. <http://golly.sourceforge.net/>.
- [40] Mirek Wojtowicz. MCell. <http://mirekw.com/ca/index.html>.