instituto superior de engenharia de lisboa

Roulette Game

6º Relatório do Projeto *Roulette Game*

Trabalho realizado por:

Gustavo Costa | Nº 52808

Ian Frunze | Nº 52867

Rafael Pereira | Nº 52880

Turma: **LEIC24D**

Docentes: **David Velez e Rui Duarte**

Licenciatura em Engenharia Informática e de Computadores

Laboratório de Informática e de Computadores (LIC)

2024 / 2025 - Semestre de Verão

Conteúdo

Lista de Figuras

1 Introdução

O projeto *Jogo da Roleta* consiste no desenvolvimento de um sistema de jogo de roleta eletrónico que integra estreitamente hardware digital programável e *software* em alto nível. Foi realizado no contexto pedagógico da unidade curricular *Laboratório de Informática e Computadores*, tendo como objetivos educacionais proporcionar experiência na *co-design* de sistemas híbridos (*hardware*/*software*), design de *firmware* para FPGAs, implementação de protocolos de comunicação série e estruturação de aplicações concorrentes baseadas em máquinas de estados. Em termos tecnológicos, o projeto conjuga módulos de *hardware* em FPGA (implementados em VHDL) com um programa de controlo em *Kotlin* executado num PC, comunicando através de interfaces série dedicadas. Desta forma, explora-se a interação entre camadas de abstração distintas, desde o nível físico de sinais digitais até à lógica de aplicação em *software*.

No funcionamento geral do sistema, o jogador insere "moedas" num moedeiro virtual para obter créditos, visualizando o saldo num ecrã LCD. Com créditos disponíveis, pode iniciar uma nova sessão de jogo. A cada sessão, o jogador realiza apostas num ou mais números/letras da roleta (`0-9`, `A-D`) através de um teclado matricial 4×4. Cada aposta consome um crédito do saldo. Após efetuar as apostas desejadas, o jogador finaliza a fase de apostas, momento em que o sistema "roda" a roleta e determina aleatoriamente um resultado. O número ou letra sorteado é indicado num conjunto de mostradores de 7 segmentos (denominado *Roulette Display*) e também apresentado no LCD juntamente com o resultado monetário da rodada (créditos ganhos ou perdidos). O sistema então atualiza o saldo de créditos do jogador com base nas apostas ganhas.

Adicionalmente, o sistema dispõe de um `modo de manutenção - M`, acionado por um operador para consultar estatísticas de uso (contagem de moedas inseridas, número de jogos realizados, frequência de ocorrência de cada número sorteado, etc.) ou efetuar operações administrativas (por exemplo, reposição de valores ou desligar a máquina). Todos os módulos de *hardware* comunicam com o *software* de controlo de forma assíncrona e em série, garantindo uma arquitetura modular e escalável. Em seguida, nas secções seguintes, detalham-se os objetivos específicos e especificações, a arquitetura em camadas do sistema, o comportamento por máquina de estados e as funcionalidades do modo de manutenção, bem como os procedimentos de teste, validação e conclusões do projeto.

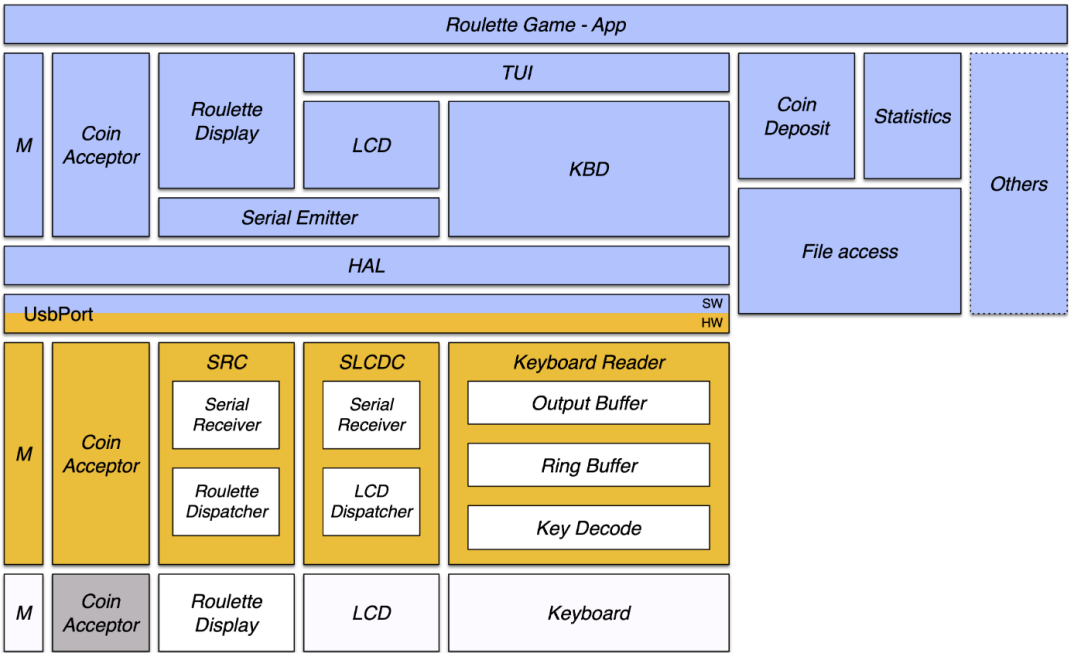

Figura 1: Arquitetura global do projeto

2 Objetivos e Especificações

O objetivo geral do projeto é implementar um jogo de roleta eletrónico totalmente funcional e fiável, integrando componentes de *hardware* dedicados e *software* de controlo, de forma a simular a experiência de uma máquina de apostas real. Este objetivo desdobra-se nos seguintes objetivos específicos e requisitos funcionais:

- **Leitura fiável do teclado:** Detetar, correta e instantaneamente, qualquer uma das 16 teclas do teclado matricial 4×4, sem ambiguidades ou falhas na identificação, mesmo sob utilização intensa (várias teclas pressionadas rapidamente ou quase simultâneas). Deve ser aplicado *debouncing* e lógica robusta de varrimento para evitar leituras instáveis.
- **Gestão de créditos e apostas:** Permitir a inserção de "moedas" de diferentes valores (ex. moedas virtuais que valem 2 ou 4 créditos) através do moedeiro, atualizando o saldo de créditos do jogador. Cada aposta realizada pelo jogador deve consumir exatamente 1 crédito do saldo e ser registada associada ao número/letra escolhido. O sistema deve impor um limite máximo de apostas por jogo (por defeito, 9 apostas por cada dígito) para controlar a duração de cada rodada.
- **Regras do jogo e pagamento:** Implementar a lógica de jogo em conformidade com as regras definidas: o jogador inicia a roleta quando tiver pelo menos 1 crédito, efetua uma ou mais apostas (cada uma num único número ou letra), e então finaliza as apostas para dar início ao sorteio. No sorteio, a roleta deve gerar aleatoriamente um resultado entre as 14 possibilidades (`0-9`, `A-D`). Se o jogador tiver apostado na opção sorteada, ganha um prémio proporcional (no mínimo, recupera 2 créditos por cada aposta correta, correspondendo a duplicar o custo da aposta, isto é, um lucro líquido de `+1` crédito por aposta vencedora). Apostas em opções não sorteadas resultam em perda dos créditos apostados. O sistema deve calcular automaticamente os créditos ganhos no final de cada rodada e adicioná-los ao saldo (não podendo esse acréscimo ser negativo – perdas já foram debitadas no ato da aposta).
- **Feedback ao utilizador:** Apresentar claramente as informações do jogo ao jogador em tempo real. A interface de utilizador inclui um ecrã LCD de 2 linhas/16 colunas para mensagens e indicações textuais (saldo atual, instruções, resultados, avisos) e um conjunto de 6 *displays* de 7 segmentos (*Roulette Display*) para efeitos visuais do sorteio (animação de rotação e indicação do número sorteado). Por exemplo, após inserir moedas, o LCD mostra o saldo em créditos; durante as apostas, o LCD pode listar as opções apostadas ou contadores de apostas, e após o sorteio exibe o resultado e o montante ganho. Os mostradores de 7 segmentos, por sua vez, simulam a roleta girando (acendendo sequencialmente padrões numéricos) e finalmente exibem de forma destacada o número vencedor.
- **Interface de controlo e teclas:** Definir claramente as funções de cada tecla do sistema:
  - Tecla `*` (asterisco): usada para iniciar um novo jogo quando há créditos (passar do modo inativo para a fase de apostas) e também para confirmar ações especiais. Durante a fase de apostas, o jogador pressiona `*` para finalizar as apostas e dar início ao sorteio da roleta.
  - Tecla `#` (cardinal): usada principalmente como comando de retrocesso ou cancelamento. Se for pressionada durante a introdução de apostas, cancela a última aposta inserida (permitindo corrigir enganos removendo a última seleção e recuperando o crédito apostado). No modo de manutenção, a tecla `#` serve para sair/voltar atrás nos *menus*.
  - Teclas numéricas `0-9` e letras `A-D`: representam as possíveis apostas na roleta. Cada uma corresponde a uma opção (digito `0-9` ou letra `A, B, C, D`) que o jogador pode apostar. Essas teclas, quando pressionadas durante a fase de apostas, registam uma aposta naquela opção (consumindo crédito). Caso uma opção seja apostada múltiplas vezes, o sistema incrementa o número de apostas nessa opção (até ao limite definido). Importa notar que no modo normal de jogo, as teclas `A`, `B`, `C`, `D` funcionam exatamente como opções de aposta adicionais (equivalentes a "`10, 11, 12, 13`" em hexadecimal).

  - Teclas de manutenção: no *modo de manutenção*, as funções de algumas teclas são alteradas. Em particular, `A` e `B` passam a atuar como teclas de navegação (*scroll*) para cima e para baixo nos menus de manutenção (indicadas no ecrã por setas ao lado de "A" e "B"). A tecla confirma ações (por exemplo, *reset* de estatísticas ou confirmação de desligar sistema) neste modo, e `#` sai do *menu* atual de manutenção. As teclas `C` e `D`, quando no *menu* principal de manutenção, correspondem respetivamente às opções de consultar estatísticas de números sorteados e de desligar o sistema. (Fora do modo de manutenção, as teclas `C` e `D` apenas servem como opções de aposta normais; o acesso ao modo de manutenção é protegido e descrito adiante.)

- **Modo bónus de apostas:** Introduzir um sub-período extra de apostas (*bónus*) após o jogador sinalizar fim das apostas normais. O sistema deverá conceder uma janela de tempo curta (ex. 5 segundos) para o jogador inserir algumas apostas adicionais de última hora, até um limite fixo (por exemplo até 3 apostas bónus). Este "tempo bónus" é indicado por um temporizador em contagem decrescente no LCD, e findo esse prazo (ou atingido o limite de apostas bónus), a roleta inicia automaticamente o sorteio. Este mecanismo adiciona dinamismo ao jogo e foi especificado como um requisito funcional extra.

- **Modo de manutenção:** Disponibilizar funcionalidades de manutenção e diagnóstico acessíveis apenas a operadores autorizados. No modo de manutenção, o sistema deve mostrar *menus* no LCD que permitam consultar:

  - Contadores acumulados de moedas inseridas (por tipo de moeda) e número total de jogos realizados.
  - Estatísticas de ocorrências de resultados da roleta (frequência de cada número/letra sorteado e créditos ganhos associados a cada um).

  Além da consulta, o operador deve poder reiniciar/resetar estes contadores/estatísticas para zero através de comandos explícitos no menu (com confirmação para evitar acionamento acidental), por exemplo pressionando `*` numa opção de "*reset*". Por fim, o modo de manutenção deve oferecer a opção de desligar com segurança o sistema (`D`) — que, no contexto de um protótipo, termina a execução do programa de controlo. O acesso ao modo de manutenção é protegido por um interruptor físico (*maintenance switch*) no *hardware*: apenas ativando esse interruptor o sistema entra neste modo, suspendendo temporariamente o funcionamento normal do jogo.

Em síntese, o sistema deve satisfazer os requisitos acima garantindo confiabilidade e integridade: toda tecla premida deve ser corretamente registada (sem perdas), toda mensagem enviada aos periféricos (LCD, *display* da roleta) deve ser entregue corretamente (uso de protocolos com *handshake* e bits de paridade para deteção de erros), e o fluxo das fases do jogo deve obedecer às regras definidas, proporcionando uma experiência contínua ao utilizador. O desempenho em termos de latência deve ser tal que o jogador não perceba atrasos indevidos na resposta às suas ações (por exemplo, a resposta ao pressionar teclas ou à atualização de saldo deve ser praticamente imediata). A seguir, é apresentada a arquitetura em camadas do sistema, com descrição detalhada dos módulos de *hardware* desenvolvidos e da interação destes com o *software* de controlo.

3 Introdução à Arquitetura do Sistema

O sistema foi projetado em três camadas principais – *hardware*, *abstração* e *aplicação* – de forma a modularizar responsabilidades e facilitar a integração de componentes heterogéneos. A Figura?? ilustra estas camadas e os principais módulos em cada uma, bem como as interações entre elas (através de interfaces série e chamadas de função).

3.1 Camada de *Hardware* (FPGA)

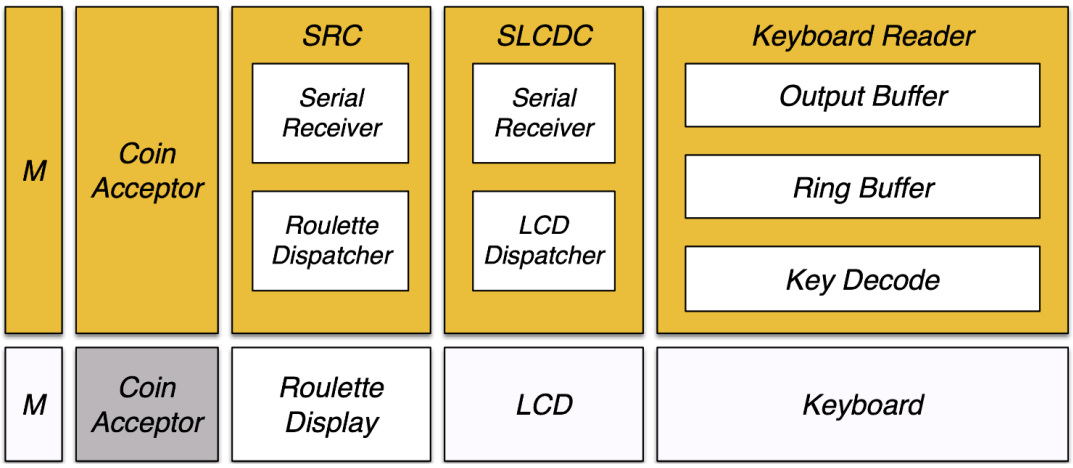

Figura 2: Visão geral da Camada de Hardware

3.1.1 Keyboard Reader (KBD)

O módulo *Keyboard Reader* é responsável pela leitura contínua de um teclado matricial 4×4, identificando de forma única cada tecla pressionada. Internamente, é composto por três sub-blocos principais: `KeyDecode`, `RingBuffer` e `OutputBuffer`.

O bloco `KeyDecode` realiza o varrimento cíclico das colunas do teclado, ativando uma coluna de cada vez e lendo as linhas (`LIN`) para detetar teclas pressionadas. Quando uma tecla é identificada, gera um código digital de 4 bits (`0x0` a `0xF`) e ativa o sinal `key_pressed`.

Este código é então armazenado no `RingBuffer`, que funciona como uma fila `FIFO`, permitindo reter múltiplas teclas premidas em rápida sucessão caso o sistema de controlo ainda não as consiga processar, garantindo assim a não perda de dados.

Por fim, o `OutputBuffer` disponibiliza os códigos ao *software* de controlo através de um protocolo de *handshake* com os sinais `Dval` (dado válido) e `ACK`. O sinal `Dval` indica que há um novo código disponível na saída `D`, sendo este lido pelo *software* e confirmado com `ACK`, permitindo o avanço para o próximo dado armazenado.

Este sistema assegura uma leitura fiável e desacoplada das teclas, que tolera variações de velocidade de escrita e evita perdas de informação.

3.1.2 Serial LCD Controller (SLCDC)

O módulo *Serial LCD Controller* (SLCDC) faz a interface entre o *software* e o LCD alfanumérico (16×2) da *Expansion Board*, convertendo comandos em série, para ações compatíveis com o controlador paralelo do LCD, que exige sinais e temporizações rigorosas.

A comunicação com o software é feita através de duas linhas: `LCDsel` (ativa a transmissão, *active-low*) e `SCLK` (*clock*), usando tramas de 6 bits: 1 bit de tipo (`RS`), 4 bits de dados (*nibble*) e 1 bit de paridade ímpar para deteção de erros.

O SLCDC é constituído por dois blocos principais:

- `Serial Receiver` – Recebe e reconstrói a trama bit a bit, sincronizado com `SCLK`. Valida a paridade ímpar e, se correta, entrega os 5 bits ao bloco seguinte com um sinal `data_valid`. Tramas com paridade incorreta são descartadas.
- `LCD Dispatcher` – Interpreta o bit `RS` para distinguir comandos (`RS=0`) de dados (`RS=1`). Envia o *nibble* para o LCD em modo de 4 bits, controla os sinais `RS` e `E`, e gere as temporizações necessárias entre operações, sinalizando `busy` enquanto o LCD estiver ocupado.

Este módulo abstrai a complexidade do controlo do LCD, permitindo que o *software* escreva no visor através de simples tramas seriais. O uso de paridade melhora a fiabilidade da transmissão, e a interface série reduz o número de fios necessários. O SLCDC permite apresentar informações úteis ao jogador, como créditos, apostas, resultados e mensagens de estado, e ainda executar comandos de controlo do ecrã como limpeza, posicionamento do cursor e criação de caracteres personalizados.

3.1.3 Serial Roulette Controller (SRC)

O módulo *Serial Roulette Controller* (SRC) faz a interface com o *Roulette Display*, composto por *displays* de 7 segmentos e, opcionalmente, LEDs de animação. Tal como o SLCDC, recebe comandos em série do *software* e executa as ações correspondentes no *hardware*.

A comunicação utiliza tramas de 9 bits: 3 bits de comando (`cmd`), 5 bits de dados e 1 bit de paridade ímpar. Os comandos abrangem a atualização de um dos 6 dígitos individuais (`cmd = 000` a `101`), controlo global (`cmd = 110`, para ações como atualização simultânea de todos os dígitos ou animações), e controlo de energia (`cmd = 111` com o bit de dados `LSB = 0` para ligar ou 1 para desligar os segmentos).

O módulo é composto por dois blocos principais:

- `Serial Receiver` – Recebe bit a bit os 9 bits da trama, sincronizando com `SCLK`. Após a receção, valida a paridade ímpar e, se correta, disponibiliza os 8 bits (`cmd` + dados) paralelamente com um sinal `data_valid`. Tramas inválidas são descartadas.
- `Roulette Dispatcher` – Interpreta os comandos e executa as ações no *hardware* dos *displays*. Pode atualizar dígitos individualmente, ligar/desligar todos os segmentos ou realizar atualizações em bloco. Também está preparado para processar comandos de animação, embora na implementação atual essa funcionalidade seja tratada no *software*.

O SRC isola a complexidade do controlo dos segmentos e LEDs, oferecendo uma interface simples e fiável ao *software*. A verificação por paridade reduz o risco de comandos corrompidos afetarem os *displays*. Este módulo permite apresentar os resultados da roleta de forma clara e animada, melhorando a experiência do utilizador.

3.1.4 Coin Acceptor

Módulo do *hardware* responsável por detetar a introdução de moedas e converter isso em créditos. No protótipo, o moedeiro é simulado por um sinal de entrada digital que indica presença de moeda e por sinais que codificam o tipo de moeda. O `Coin Acceptor` monitora continuamente um pino de "moeda inserida" e quando este sinal ativa, o módulo lê o identificador da moeda (por exemplo, um bit ou conjunto de bits que distinguem dois tipos: moeda de 2 créditos ou de 4 créditos). Imediatamente, aciona-se um sinal de `ACCEPT` (saída) para indicar ao utilizador que a moeda foi contabilizada (no protótipo, poderia acender um LED ou gerar um som) e o módulo fica temporariamente bloqueado até a moeda ser "retirada" (no caso real seria o tempo de a moeda cair no cofre, aqui simulamos aguardando o sinal de inserção voltar a 0). Depois disso, o `Coin Acceptor` desativa `ACCEPT` e fica pronto para a próxima moeda. Este módulo não comunica via protocolo série como os anteriores, mas sim através de leituras diretas de bits de I/O pela camada de abstração (`HAL`). O *software* consulta periodicamente o estado do `Coin Acceptor` para verificar se há nova moeda e, em caso afirmativo, incrementa o saldo de créditos conforme o tipo. Dois contadores internos (um para cada tipo de moeda) registam o total de moedas acumuladas de cada tipo, para fins de estatística no modo manutenção.

Todos os módulos de *hardware* acima foram implementados em VHDL e sintetizados na FPGA, comunicando com o meio exterior (teclado, LCD, *displays*, entradas de moeda) através de pinos físicos da placa `DE10-Lite`. A separação em módulos distintos promoveu a clarificação de funcionalidades: cada módulo possui responsabilidade bem definida e pôde ser desenvolvido e testado isoladamente. Além disso, a comunicação entre eles e com o *software* segue interfaces específicas (protocolos série ou sinais de controle), o que facilitou a integração. Os módulos `Keyboard Reader`, `SLCDC` e `SRC` apresentam arquiteturas semelhantes no sentido de incorporarem um `SerialReceiver` (nos dois últimos casos) e um bloco de despacho, bem como sinais de *handshake*, reforçando a consistência do desenho de sistema.

3.2 Camada de Abstração (Drivers de *Software*)

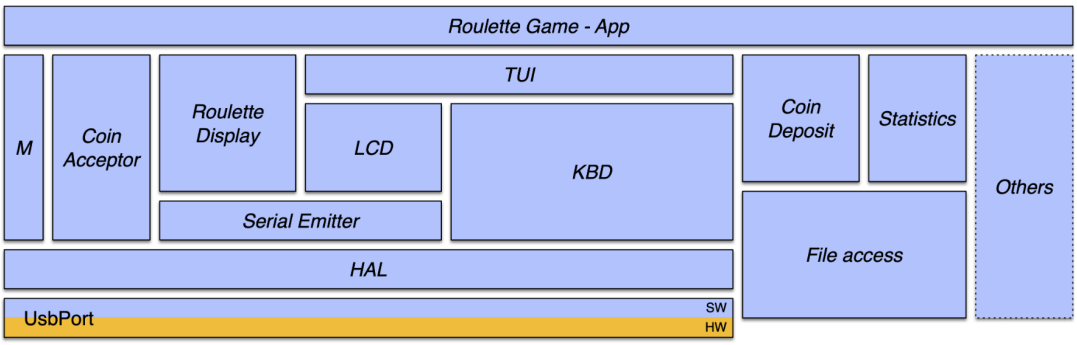

Figura 3: Visão geral da camada de Software

Esta camada, implementada em *Kotlin*, fornece uma interface de alto nível entre a aplicação e o *hardware*, abstraindo os detalhes de baixo nível como endereços de I/O e temporizações. Os seus principais componentes são descritos nas secções seguintes.

3.2.1 HAL (*Hardware Abstraction Layer*)

A `HAL` é uma biblioteca que permite o acesso direto aos registos de entrada/saída do hardware. Fornece métodos como `HAL.isBit(x)` para ler o estado de um bit, ou `HAL.setBits(y)` e `HAL.clrBits(y)` para definir ou limpar saídas digitais. Internamente, a `HAL` conhece a forma como a FPGA está mapeada (por exemplo, via memória ou USB/UART), permitindo à aplicação interagir com o *hardware* sem conhecer esses detalhes.

3.2.2 Serial Emitter

O `SerialEmitter` trata do envio de tramas seriais para módulos como o SLCDC e o SRC. Define destinos como `Destination.LCD` ou `Destination.ROULETTE`, e disponibiliza o método `send(destino, valor, tamanhoBits)`. Automatiza a ativação das linhas de seleção, o envio bit-a-bit sincronizado com `SCLK` e o cálculo/inclusão do bit de paridade. Desta forma, a aplicação pode enviar comandos sem se preocupar com temporizações ou protocolos de envio.

3.2.3 KBD (Driver de Teclado)

O módulo `KBD` lê as teclas fornecidas pelo *Keyboard Reader*. Consulta periodicamente o sinal `Dval` e, quando ativo, lê o código presente nas linhas `D`, mapeando-o para caracteres (`0-9`, `A-D`, * ou #). Após a leitura, envia o sinal `ACK` para libertar o próximo código. Este processo pode ser feito por *polling* ou com uma função não bloqueante que devolve `NONE` se não houver tecla disponível.

3.2.4 Utilitários Adicionais

Incluem-se nesta categoria classes como a `TUI` (Textual User Interface), que facilita operações no LCD, tais como `TUI.write()` ou `TUI.clear()`. Estes utilitários também permitem a criação e uso de símbolos customizados, como setas ou ícones de moeda. Internamente, utilizam o `SerialEmitter` e a `HAL` para concretizar as ações no *hardware*.

3.3 Resumo

Em conjunto, estes *drivers* encapsulam a complexidade da comunicação com o *hardware*, promovendo modularidade, reutilização e facilidade de manutenção. Mudanças no *hardware* requerem apenas ajustes nesta camada, sem afetar a lógica do jogo.

4 Máquina de Estados Principal

Esta secção descreve a lógica de controlo central que gere o fluxo de jogo desde a inserção de créditos até à atualização de estatísticas, suportada por uma máquina de estados finitos (ASM) residente na função `run()`.

Tal como podemos observar na Figura 4, a ASM principal contém três estados:

- **Lobby** – Apresenta saldo, aceita moedas, permite entrar em manutenção e, quando o jogador prime `*`, avança para a recolha de apostas.
- **Game** – Executa animações, sorteia o número vencedor, calcula prémios e atualiza o saldo.
- **UpdateStats** – Regista o resultado e incrementa contadores antes de regressar ao **Lobby**.

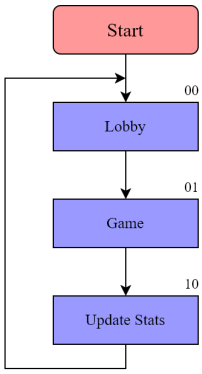

Figura 4: Diagrama ASM do ciclo principal realizado pela função `run()`.

4.1 Estado Lobby

Este é o estado quiescente da FSM, no qual o sistema aguarda por interação do utilizador. O LCD exibe uma mensagem de boas-vindas, como `"Roulette Game - Insert coin"`, e o sistema permanece em espera até serem detetados créditos válidos. Esta deteção ocorre ciclicamente através da função `updateCreds()`, que consulta o módulo `CoinAcceptor`.

Figura 6: Simulação no estado Lobby

Quando uma moeda é inserida:

- O saldo `CREDS` é incrementado;
- Uma animação de moeda personalizada é exibida no LCD (aproveitando os caracteres definidos com `LCD.defineCustomChar()`);
- O novo saldo é mostrado no canto superior direito do visor.

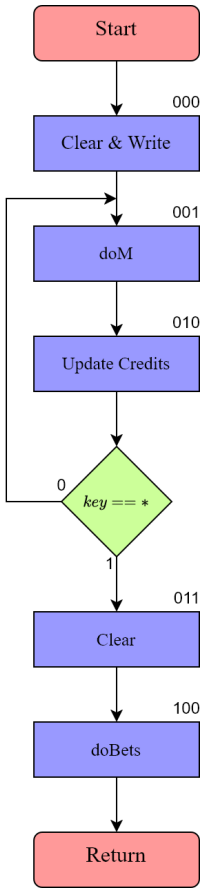

Figura 5: Diagrama ASM do Lobby

Caso `CREDS > 0`, o jogador pode iniciar o jogo pressionando a tecla `*`. A ASM verifica este evento e transita para o estado de *Apostas*. Se `*` for pressionada sem saldo, a ASM ignora a ação ou apresenta a mensagem `"No Credits"`.

Alternativamente, se o operador ativar o modo de manutenção (via interruptor físico), a ASM do jogo pausa e transita para a ASM de manutenção, mantendo-se congelada até esta terminar.

4.2 Estado Lobby (Fase de Apostas)

Após a entrada no *lobby*, o LCD apresenta as instruções de aposta e as opções válidas (`0-9`, `A-D`). O jogador pode colocar múltiplas apostas, cada uma correspondendo a um débito de 1 crédito.

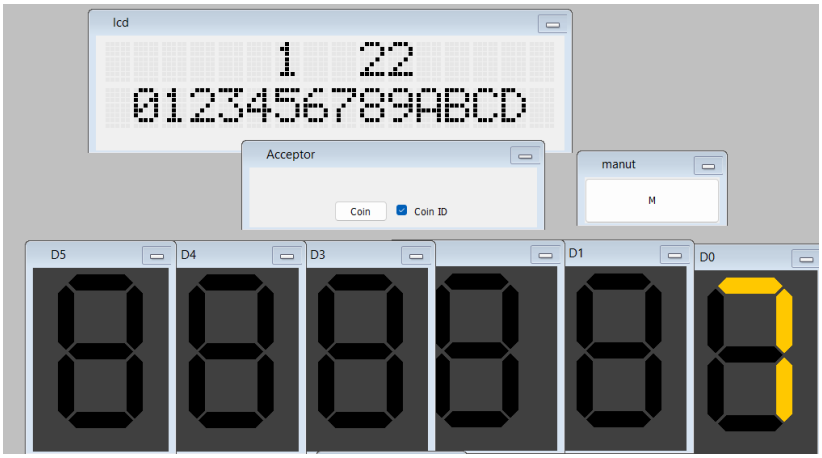

Figura 7: Simulação no estado Lobby (apostas)

Cada tecla válida captada pelo `KeyboardReader`:

- Atualiza o mapa `BETS` com a contagem de fichas por opção;
- Deduz 1 crédito de `CREDS`;
- Atualiza visualmente a linha de apostas no LCD;
- Verifica, via `canUpdateBets()`, se não excede o limite de fichas.

A tecla `#` atua como *undo*, removendo a última aposta e devolvendo o crédito correspondente. O LCD é imediatamente atualizado.

O jogador termina a fase premindo novamente `*`. A função `canStartGame()` valida se há pelo menos uma aposta; se sim, a ASM transita para o subestado de bónus. Caso contrário, é exibido um aviso e o sistema permanece no *lobby*.

4.3 Subestado de Apostas Bónus

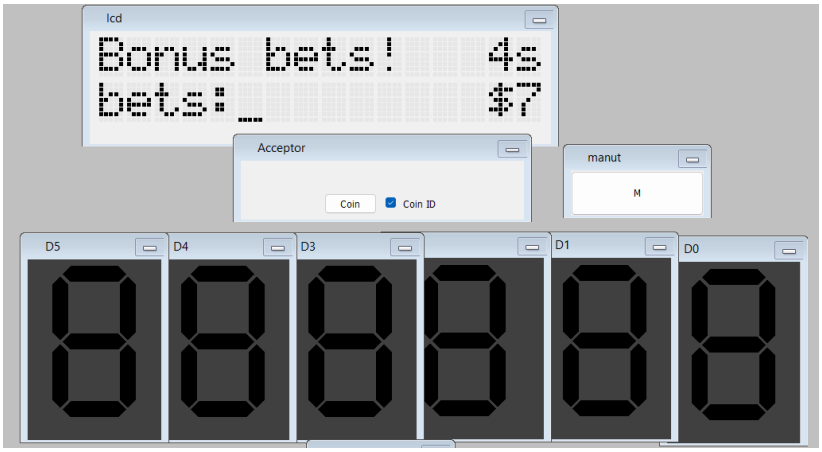

Figura 8: Simulação no estado Bonus

Este subestado inicia-se após o jogador finalizar as apostas regulares. Durante um intervalo de `TIMEBONUS` segundos (normalmente 5), o jogador pode inserir até `BONUSBETS` apostas adicionais.

O LCD apresenta:

- Uma mensagem `"Bonus bets!"` com temporizador regressivo;
- Uma lista das apostas bónus inseridas (ex: `7,C`).

Cada nova aposta:

- É adicionada ao conjunto principal `BETS`;
- Deduz 1 crédito de `CREDS`;
- É exibida no LCD;
- Pode ser desfeita com a tecla `#`.

Quando o tempo termina ou o número máximo de apostas bónus é atingido, o subestado é encerrado com `doBonusEndAnimation()`, que exibe mensagens temáticas no LCD dependendo da participação do jogador (ex: `"Meh.."`, `"OH YEAH!"`).

Após a conclusão da fase de apostas (regulares e bónus), a ASM transita para o estado de *Sorteio*, que combina dois momentos fundamentais do jogo: a simulação visual do giro da roleta e a apresentação do resultado, acompanhada do cálculo e entrega de prémios.

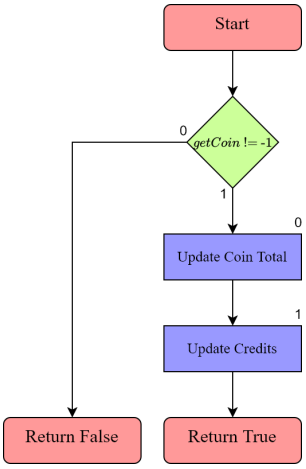

Figura 9: Diagrama ASM da função updateCreds.

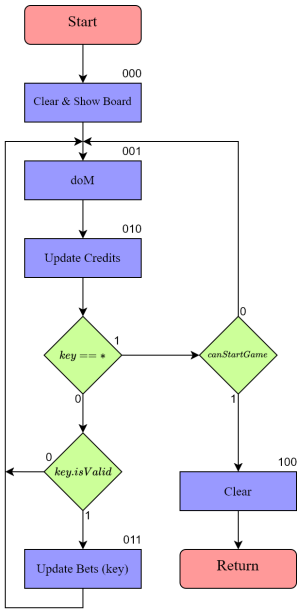

Figura 10: Diagrama ASM da função doBets.

4.4 Estado Game

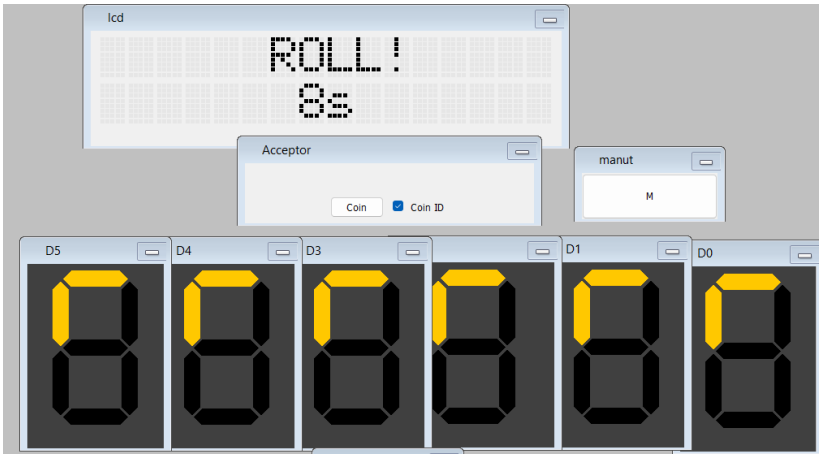

Figura 11: Simulação no estado Roll

Início do Sorteio e *Feedback* Visual O LCD é atualizado para assinalar o início do sorteio. A primeira linha exibe `"ROLL!"`, enquanto a segunda mostra uma contagem decrescente indicando o tempo restante até ao resultado. Esta duração é pseudoaleatória, gerada com:

```
timeRoll = (3..9).random()
```

Durante este período, o valor é decrementado a cada segundo e o LCD é atualizado para mostrar `"Xs"`, mantendo o utilizador informado de que o sorteio está em curso.

Animação da Roleta nos *Displays* Enquanto o temporizador decorre, os seis mostradores de sete segmentos apresentam animações contínuas através do módulo `RouletteDi` `animationA()` é executada em ciclos rápidos, iluminando padrões sequenciais nos dígitos. O uma rotação de roleta, com transições entre caracteres hexadecimais (`0-F`), movimentos de pontos de deslocamento de símbolos entre os *displays*. Esta animação decorre até ao fim do tempo de realismo da simulação.

Determinação do Resultado Sorteado Quando `timeRoll` atinge zero, a ASM interrompe a animação e procede à seleção do número vencedor. A escolha é feita aleatoriamente com:

```
sorted = validBets.random()
```

Este valor corresponde a uma das 14 posições possíveis (caracteres `0-9, A-D`) e é atribuído à variável global `SORTED`. A transição para a apresentação do resultado ocorre imediatamente.

Exibição do Resultado ao Jogador O LCD é limpo e reconfigurado para mostrar o número sorteado e os créditos ganhos. A primeira linha exibe:

```
"Sorted:  X"
```

A segunda apresenta:

```
"Won:  $Y"
```

Os textos são centrados e apresentados com `TUI.writeCenterLine()` ou equivalente. Simultaneamente, os mostradores de 7 segmentos exibem o mesmo caractere `sorted` em todos os dígitos. Durante cerca de 3 segundos, é aplicada uma animação de *piscar*, alternando entre mostrar o valor e apagar o *display* em intervalos de `100 ms`, produzindo um efeito visual atrativo.

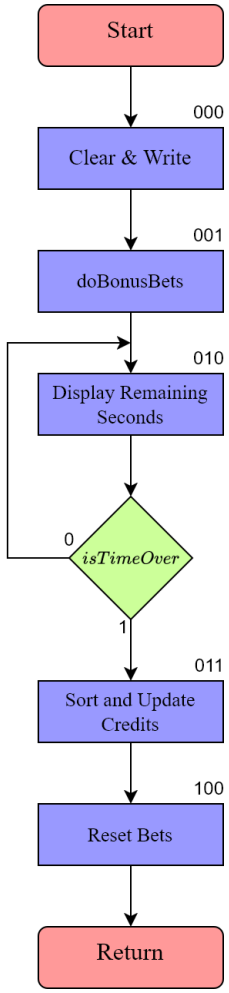

Figura 12: Diagrama ASM da função Game.

Cálculo de Créditos Ganhos A função `updateStats()` calcula os ganhos do jogador. Para tal:

- Cada ficha apostada na casa correta resulta em `COST * 2` créditos ganhos (lucro +1);
- Fichas em casas erradas resultam numa perda do crédito já descontado;
- O valor total de `wonCredits` é limitado inferiormente a zero (nunca há perda adicional).

Exemplo: Se o jogador apostou cinco créditos, com um acerto:

```
2 - 4 = -2 -> ajustado para 0
```

Se acertou duas fichas em cheio, sem erros:

```
2 * 2 = 4 créditos
```

O saldo `CREDS` é atualizado com o valor `wonCredits` e os totais de apostas são resetados para a próxima ronda.

Atualização de Estatísticas e Persistência Após o pagamento, o sistema:

- Incrementa `TOTAL_GAMES`;
- Atualiza os contadores da opção sorteada em `Statistics`;
- Guarda os dados persistentes com `writeAllStats()`.

Os valores `SORTED` e `WON` são retidos para referência em menus de manutenção. Após uma breve pausa, a ASM regressa ao estado `Idle`, com o LCD novamente a indicar o saldo atual e instruções para começar nova partida.

Robustez e Encadeamento de Estados Este estado encerra de forma fiável o ciclo de jogo. Todas as variáveis são reiniciadas de forma limpa e não há transições prematuras. A sincronização entre animações, sorteio, exibição e registo é coordenada para garantir uma experiência coesa. A ASM está preparada para interrupções por modo de manutenção, sendo possível interromper o jogo no estado `Idle` sem perdas de integridade.

5 Modo de Manutenção

O modo de manutenção é um estado especial ativado por um interruptor físico ligado ao FPGA, cuja deteção é feita via `HAL`. Quando o sinal é reconhecido (`M.inM = true`), o sistema suspende temporariamente a FSM principal e transita para uma FSM secundária dedicada à gestão técnica e administrativa. Esta transição só ocorre se o sistema estiver em `Idle` ou em fase de apostas, evitando interferência com sorteios em curso.

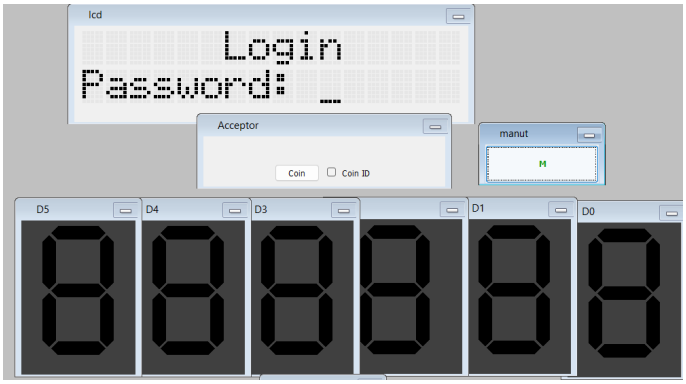

Figura 16: Simulação no menu de Manutenção

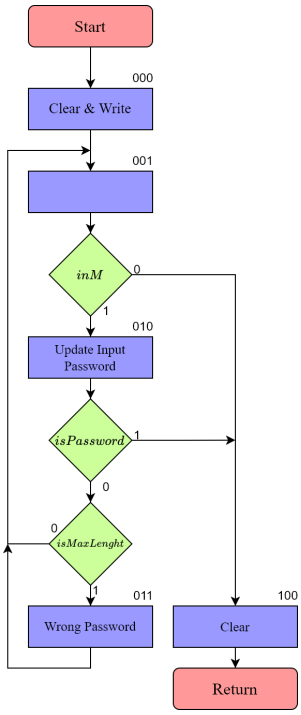

Figura 13: Diagrama ASM da função LogIn.

5.1 Menu Principal

O LCD alterna entre duas telas principais com as opções disponíveis:

- Primeira tela: `A: Counters` e `C: Stats`;
- Segunda tela: `*: Test Game` e `D: Shutdown`.

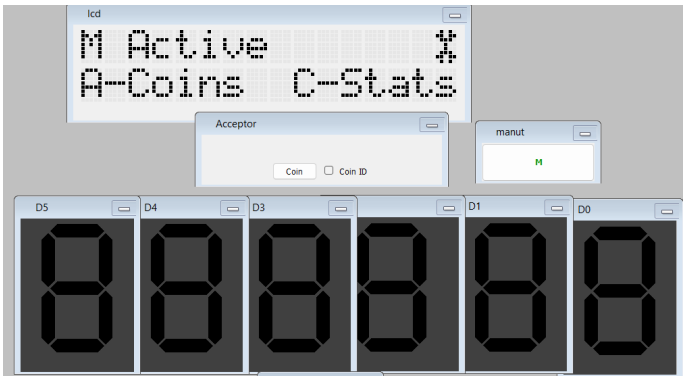

Figura 17: Simulação no menu de Manutenção

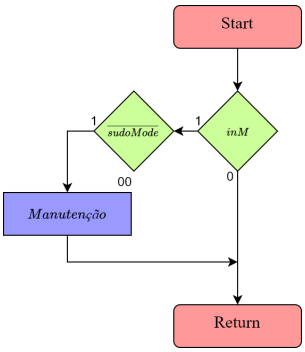

Figura 14: Diagrama ASM da função doM.

A alternância é controlada por um temporizador interno e ilustrada com ícones. As teclas de controlo correspondem a cada ação, sendo a tecla `#` inativa nesta fase (a saída é feita desligando o interruptor físico).

5.2 Página de Contadores

Selecionando `A`, são apresentadas as contagens de moedas de 2 e 4 créditos, o total de moedas e o número de jogos. Esta informação é listada com suporte a navegação vertical (via `A`/`B`), utilizando a classe `Page`. O operador pode:

- Navegar entre linhas;

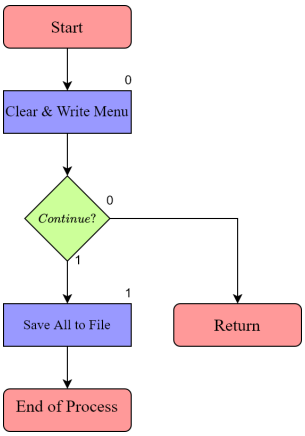

Figura 15: Diagrama ASM da função shutdown.

- Sair com `#`;
- Resetar os valores com `*`, após confirmação via `TUI.confirmMenu()`.

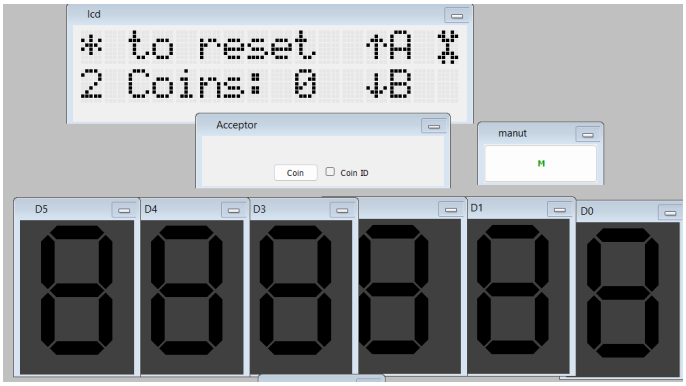

Figura 21: Simulação no menu de Manutenção

Se confirmado, executam-se `CoinDeposit.resetTotal()` e `Statistics.resetGames()`, seguidos de um redesenho dos valores no LCD.

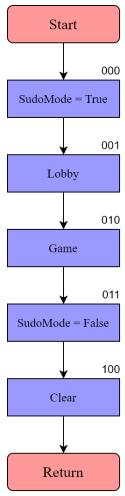

Figura 18: Diagrama ASM do jogo simulado

5.3 Página de Estatísticas

Selecionando `C`, apresenta-se uma lista com todos os 16 símbolos da roleta (`0-9`, `A-D`), acompanhados da contagem de vezes que saíram e créditos ganhos. Também aqui se usa `Page` com *scroll* e as seguintes ações:

- `A`/`B` para navegar;
- `#` para sair;
- `*` para reset total com confirmação.

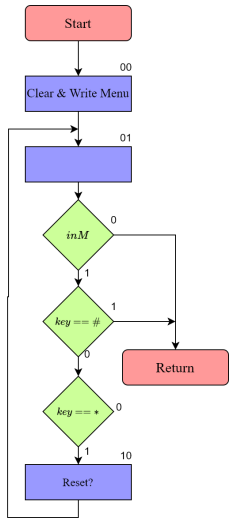

Figura 19: Diagrama ASM da função coinPage e statsPage.

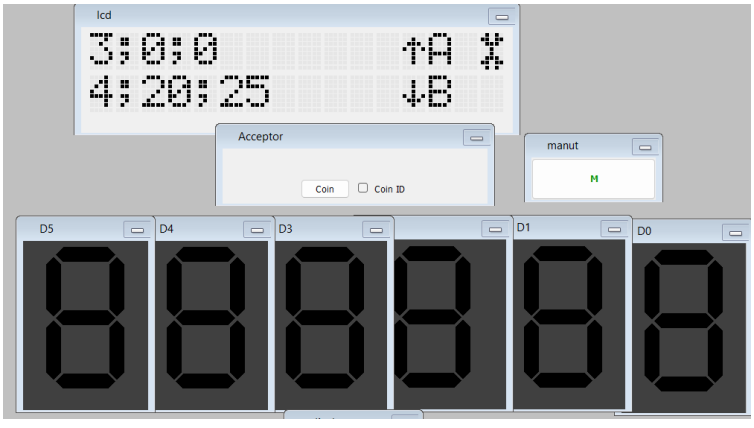

Figura 22: Simulação no menu de Manutenção

As estatísticas são resetadas via `Statistics.resetAll()`, e os valores exibidos no LCD são atualizados imediatamente.

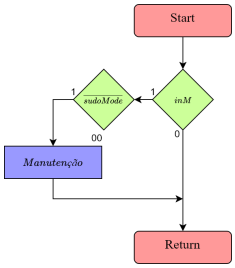

Figura 20: Diagrama ASM da função doM.

5.4 Desligamento do Sistema

Ao pressionar `D`, o sistema solicita confirmação para desligar. Se o operador confirmar com `*`, executa-se:

- Exibição de "`Shutting down...`" no LCD;
- Escrita de todos os dados com `writeAllStats()`;
- Limpeza do LCD e encerramento com `exitProcess(0)`.

Se o operador recusar com `#`, volta ao menu principal.

5.5 Jogo Simulado

A opção `*` ativa o `sudoMode` para executar uma ronda completa sem afetar saldo nem estatísticas. A função `runMockGame()` simula apostas e sorteio, e no final restaura o estado original. Nenhuma alteração permanente é feita ao sistema.

5.6 Saída do Modo de Manutenção

Desligando o interruptor de manutenção, o sistema limpa o LCD, restaura a ASM principal no estado anterior (normalmente `Idle`) e aplica todas as alterações feitas nos contadores e estatísticas, conforme o que foi executado.

5.7 Resumo

O modo de manutenção garante:

- Consulta clara e segura a dados acumulados;
- *Reset* seletivo de contadores e estatísticas;
- Execução de testes funcionais isolados;
- Desligamento controlado do sistema.

A navegação é intuitiva e protegida por confirmações, com *prompts* claros no LCD. Toda a lógica foi testada para garantir robustez, isolamento completo da ASM principal e segurança dos dados críticos.

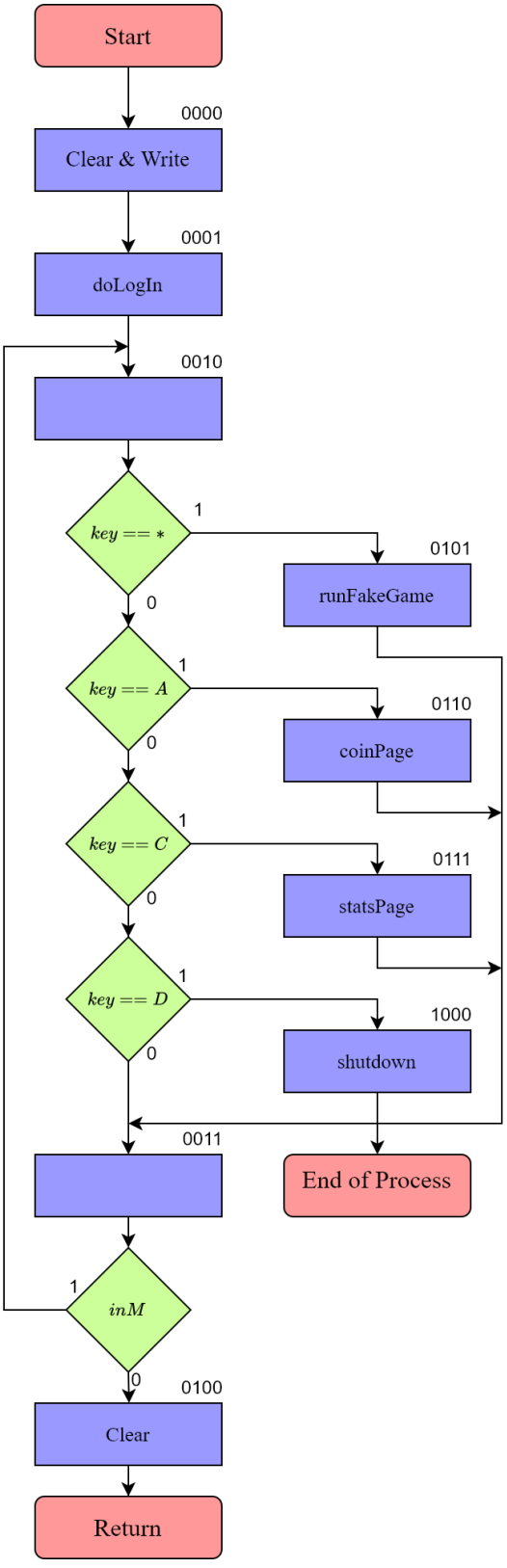

Figura 23: Diagrama ASM do estado de manutenção

6 Conclusão

O projeto *Jogo da Roleta* resultou na construção de um sistema híbrido *hardware–software* plenamente funcional, atingindo todos os objetivos propostos tanto do ponto de vista pedagógico como técnico. Através da integração entre módulos digitais em FPGA — como o `KeyboardReader`, `SLCDC`, `SRC` e `CoinAcceptor` — e uma aplicação de controlo desenvolvida em *Kotlin*, foi demonstrada com sucesso a viabilidade de coordenação eficiente entre domínios distintos.

A arquitetura modular em três camadas revelou-se eficaz, permitindo que cada componente fosse desenvolvido, simulado e testado de forma isolada. Posteriormente, a integração decorreu sem grandes dificuldades, graças à definição clara de interfaces e protocolos de comunicação. A ASM implementada no *software* assegurou um controlo preciso do fluxo de jogo, abrangendo todos os estados e subestados — incluindo os modos bónus e manutenção — com consistência e sem sobreposição de eventos.

6.1 Resultados Destacados

Integração e funcionalidade completa. O sistema final permite ao utilizador participar repetidamente em rondas do jogo, com uma sequência natural: inserir créditos, colocar apostas, assistir ao sorteio animado e consultar os ganhos obtidos. Todas as interações são devidamente refletidas nos dispositivos de saída (LCD, mostradores, saldo interno), e o comportamento do sistema mantém-se previsível e coerente com as regras do jogo. Esta fluidez operacional evidencia o cumprimento integral dos objetivos funcionais definidos no início do projeto.

Robustez e fiabilidade. O projeto implementou estratégias sólidas de tolerância a erros e falhas. O módulo de teclado, com buffer em anel (`RingBuffer`), assegura que nenhuma entrada é perdida mesmo sob digitação rápida. Os canais de comunicação serial, com bit de paridade e sinalização explícita (`data_valid`, `ACK`), evitam corrupções ou comandos perdidos. A sincronização entre domínios de relógio (PC e FPGA) foi assegurada com *handshakes* apropriados. Durante a campanha de testes, o sistema demonstrou estabilidade em todas as condições: não ocorreram *crashes*, *deadlocks* ou comportamentos indeterminados, mesmo quando foram forçadas sequências de eventos invulgares (ex. entrada indevida no modo de manutenção, apostas com saldo nulo). A deteção e bloqueio dessas ações incorretas foram bem tratadas, provando a maturidade da ASM e dos mecanismos de verificação implementados.

Aprendizagem e competências adquiridas. O desenvolvimento deste projeto permitiu aos autores explorar e aplicar um leque alargado de competências: conceção de controladores digitais em `VHDL` com ASMs internas; desenvolvimento de protocolos seriais personalizados; controlo concorrente em software de alto nível; construção de interfaces homem-máquina interativas; manipulação de estruturas de dados e persistência em ficheiros; e integração global de um sistema embebido modular. Este contacto direto com um projeto completo, desde o planeamento até à validação, reforçou a importância de boas práticas como testes unitários, simulações antes da síntese, documentação clara e divisão de responsabilidades por módulos.

Em síntese, o *Jogo da Roleta* alcançou com sucesso os objetivos delineados, demonstrando que é possível construir uma aplicação lúdica e educativa com integração fluida entre *hardware* digital e *software* de aplicação. Este projeto serviu como base sólida para consolidar aprendizagens essenciais no domínio dos sistemas digitais embebidos.