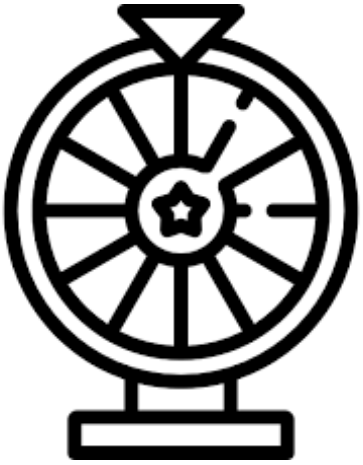
ISEL

Departamento de Engenharia Eletrónica e Telecomunicações e de Computadores

Licenciatura em Engenharia Informática e de Computadores



JOGO DA ROLETA

(*Roulette Game*)

Projeto

de

Laboratório de Informática e Computadores

2024 / 2025 verão

publicado: 17 de fevereiro de 2025

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

1 Descrição

Pretende-se implementar o jogo da Roleta (*Roulette Game*), no qual a roleta compreende números entre 0 e 9, e alfabetos entre A e D, um jogador realiza apostas premindo as teclas de um teclado correspondentes aos números/alfabetos em que pretende apostar. Por cada aposta é debitado um crédito ao saldo acumulado do jogador, podendo o jogador apostar mais do que um crédito num mesmo número. Os créditos são obtidos pela introdução de moedas no moedeiro, este só aceita dois tipos de moeda, que correspondem a dois (2) e quatro (4) créditos.
O sistema que implementa o jogo será constituído por: um PC (*Control*); um teclado de 16 teclas; um moedeiro (*Coin Acceptor*); um mostrador *Liquid Cristal Display* (*LCD*) de duas linhas com 16 caracteres; um mostrador da roleta (*Roulette Display*) e uma chave de manutenção designada por *M*, para colocação do sistema em modo de Manutenção. Na Figura 1 apresenta-se o diagrama de blocos do jogo da Roleta.

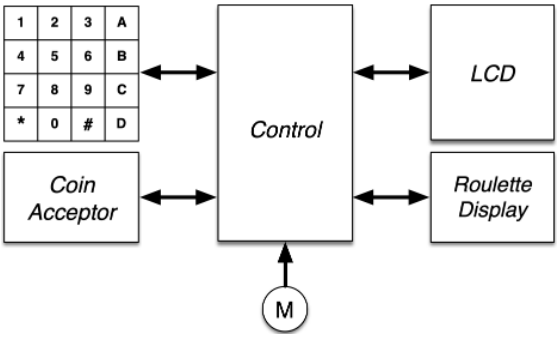


Figura 1: Diagrama de blocos do jogo da Roleta (*Roulette Game*)

Sobre o sistema proposto podem realizar-se as seguintes ações em modo de Jogo:

- **Jogo** – Inicia-se o jogo premindo a tecla '∗', e existam créditos disponíveis. Utilizando as teclas numéricas (0-9) e alfabéticas (A-D) realizam-se as apostas, retirando-se um crédito ao saldo do jogador por cada aposta realizada. O jogador termina as apostas premindo a tecla '#', o que dá início ao sorteio. Durante um tempo aleatório, o sistema simula o girar da Roleta no *Roulette Display*, permitindo ainda realizar apostas até 5 segundos antes desta parar. Ao parar a Roleta, o número sorteado e os créditos obtidos na jogada são apresentados no *Roulette Display*. Os créditos obtidos são acumulados após 5 segundos ao saldo do jogador, também apresentados no *LCD*.

No modo Manutenção podem realizar-se as seguintes ações sobre o sistema:

- **Teste** – Premindo a tecla '∗' inicia-se um jogo sem créditos e sem contabilizar os números sorteados.
- **Consultar os contadores de moedas e jogos** – Carregando na tecla 'A' permite-se a listagem dos contadores de moedas e jogos realizados.
- **Iniciar os contadores de moedas e jogos** – Premindo a tecla 'A' e em seguida a tecla '∗', o sistema de gestão coloca os contadores de moedas e jogos a zero, iniciando um novo ciclo de contagem.

- **Consultar a lista de números sorteados** – Carregando na tecla 'C' permite-se a listagem dos números sorteados.
- **Iniciar a lista de números sorteados** – Premindo a tecla 'C' e em seguida a tecla '∗', o sistema inicia um novo ciclo de estatística de números sorteados.
- **Desligar** – Premindo a tecla 'D' permite desligar o sistema, que encerra apenas após a confirmação do utilizador, ou seja, o programa termina e as estruturas de dados, contendo a informação dos contadores e da Lista de Números Sorteados, são armazenadas de forma persistente em dois ficheiros de texto, por linha e com os campos de dados separados por ";". O primeiro ficheiro deverá conter o número de jogos realizados e o número de moedas guardadas no cofre do moedeiro. O segundo ficheiro deverá conter a Lista de Números Sorteados, que contém o número de saídas e os prémios atribuídos por cada número. Os dois ficheiros devem ser carregados para o sistema no seu processo de arranque.

2 Arquitetura do Sistema

O sistema será implementado numa solução híbrida de hardware e software, como apresentado no diagrama de blocos da Figura 2. A arquitetura proposta é constituída por cinco módulos principais: *i*) um leitor de teclado, designado por *Keyboard Reader*; *ii*) um módulo de interface com o *LCD*, designado por *Serial LCD Controller* (*SLCDC*); *iii*) um módulo de interface com o mostrador da roleta (*Roulette Display*), designado por *Serial Roulette Controller* (*SRC*); *iv*) um moedeiro, designado por *Coin Acceptor*; e *v*) um módulo de controlo, designado por *Control*. Os módulos *i*), *ii*) e *iii*) deverão ser implementados em hardware, o moedeiro (*iv*) será simulado utilizando um interruptor e um *LED*, enquanto o módulo de controlo (*v*) deverá ser implementado em software, descrito em linguagem *Kotlin* e executado num PC.

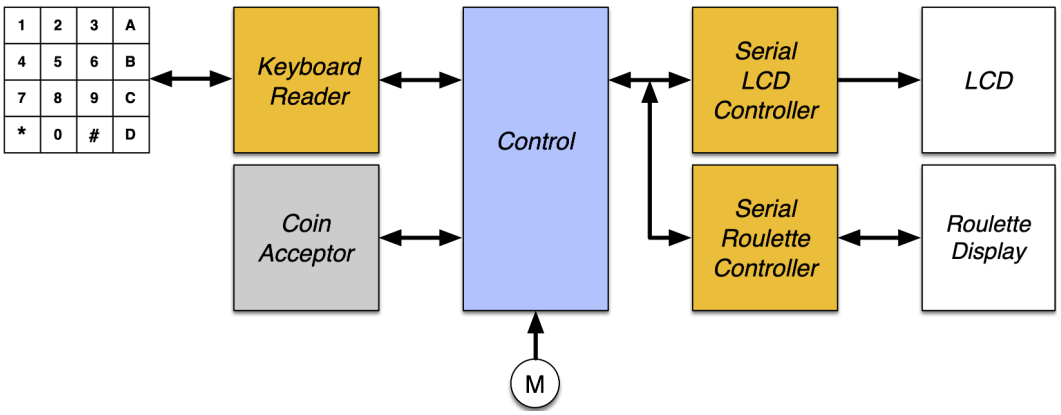


Figura 2: Arquitetura do sistema que implementa o jogo da Roleta (*Roulette Game*)

O módulo *Keyboard Reader* é responsável pela descodificação do teclado matricial de 16 teclas, determinando qual a tecla pressionada e disponibilizando o seu código, com quatro bits, ao módulo *Control*. Caso este não esteja disponível para o receber imediatamente, o código da tecla é armazenado internamente, até ao limite de dezoito (18) códigos. O módulo *Control* processa os dados e envia a informação a apresentar no *LCD* através do módulo *SLCDC*. O *Roulette Display* é atuado pelo módulo *Control*, através do módulo *SRC*. Ppor forma a minimizar o número de interligações, a comunicação entre o módulo *Control* e os módulos *SLCDC* e *SRC* é realizada através de um protocolo série.

2.1 *Keyboard Reader*

O módulo *Keyboard Reader* é constituído por três blocos principais: *i*) o descodificador de teclado (*Key Decode*); *ii*) o bloco de armazenamento (designado por *Ring Buffer*); e *iii*) o bloco de entrega ao consumidor (designado por *Output Buffer*). Neste caso o módulo *Control*, implementado em software, é a entidade consumidora.

2.1.1 *Key Decode*

O bloco *Key Decode* deverá implementar um descodificador de um teclado matricial 4×4 por hardware, sendo constituído por três sub-blocos: *i*) um teclado matricial de 4×4; *ii*) o bloco *Key*

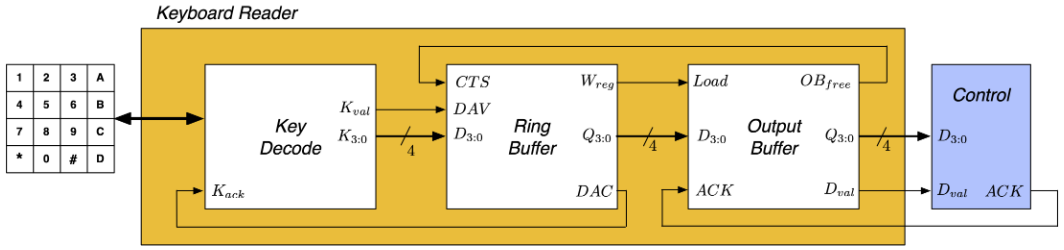
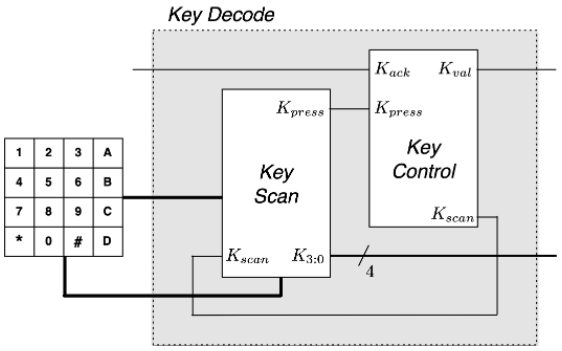
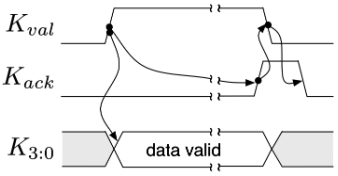


Figura 3: Diagrama de blocos do *Keyboard Reader*

Scan, responsável pelo varrimento do teclado; e *iii*) o bloco *Key Control*, que realiza o controlo do varrimento e o controlo de fluxo, conforme o diagrama de blocos representado na Figura 4a. O controlo de fluxo de saída do bloco *Key Decode* (para o bloco *Ring Buffer*) define que o sinal K_{val} é ativado quando é detetada a pressão de uma tecla, sendo também disponibilizado o código dessa tecla no barramento $K_{3:0}$. Apenas é iniciado um novo ciclo de varrimento ao teclado quando o sinal K_{ack} for ativado e a tecla premida for libertada. O diagrama temporal do controlo de fluxo está representado na Figura 4b.



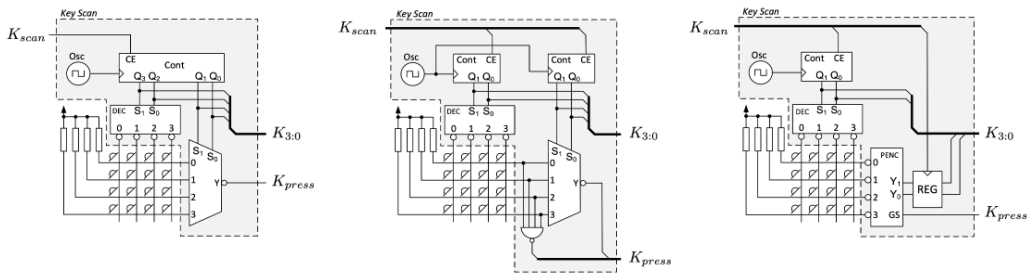
(a) Diagrama de blocos



(b) Diagrama temporal

Figura 4: Bloco *Key Decode*

O bloco *Key Scan* deverá ser implementado de acordo com um dos diagramas de blocos representados na Figura 5, enquanto o desenvolvimento e a implementação do bloco *Key Control* ficam como objeto de análise e estudo, devendo a sua arquitetura ser proposta pelos alunos.



(a) versão I **(b)** versão II **(c)** versão III

Figura 5: Diagrama de blocos do bloco *Key Scan*

2.1.2 *Ring Buffer*

O bloco *Ring Buffer* a desenvolver deverá ser uma estrutura de dados para armazenamento de teclas com disciplina $FIFO$ (*First In First Out*), com capacidade de armazenar até dezasseis (16) palavras de quatro (4) bits. A escrita de dados no *Ring Buffer* inicia-se com a ativação do sinal DAV (*Data Available*) pelo sistema produtor, neste caso pelo *Key Decode*, indicando que tem dados para serem armazenados. Logo que tenha disponibilidade para armazenar informação, o *Ring Buffer* escreve os dados $D_{3:0}$ em memória. Concluída a escrita em memória ativa o sinal DAC (*Data Accepted*) para informar o sistema produtor que os dados foram aceites. O sistema produtor mantém o sinal DAV ativo até que DAC seja ativado. O *Ring Buffer* só desativa DAC depois de DAV ter sido desativado. A implementação do *Ring Buffer* deverá ser baseada numa memória RAM (*Random Access Memory*). O endereço de escrita/leitura, selecionado por $put\overline{get}$, deverá ser definido pelo bloco *Memory Address Control* (MAC) composto por dois registos, que contêm o endereço de escrita e leitura, designados por $putIndex$ e $getIndex$ respetivamente. O MAC suporta assim ações de $incPut$ e $incGet$, gerando informação se a estrutura de dados está cheia (*Full*) ou se está vazia (*Empty*). O bloco *Ring Buffer* procede à entrega de dados à entidade consumidora, sempre que esta indique que está disponível para receber, através do sinal *Clear To Send* (CTS). Na Figura 6 é apresentado o diagrama de blocos para uma estrutura do bloco *Ring Buffer*.

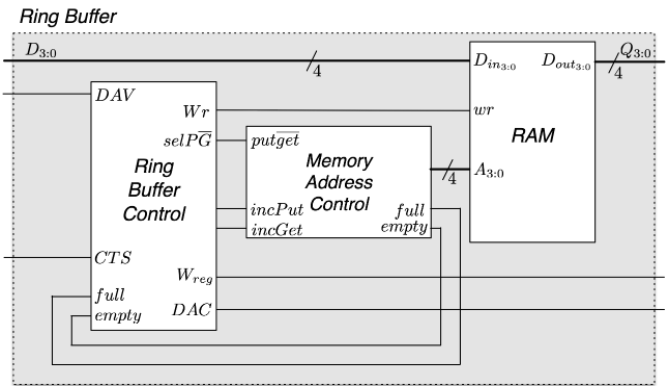


Figura 6: Diagrama de blocos do *Ring Buffer*

2.1.3 *Output Buffer*

O bloco *Output Buffer* do *Keyboard Reader* é responsável pela interação com o sistema consumidor, neste caso o módulo *Control*. O *Output Buffer* indica que está disponível para armazenar dados através do sinal OB_{free}. Assim, nesta situação o sistema produtor pode ativar o sinal *Load* para registar os dados. O *Control* quando pretende ler dados do *Output Buffer*, aguarda que o sinal D_{val} fique ativo, recolhe os dados e ativa o sinal ACK indicando que estes já foram consumidos. O *Output Buffer*, logo que o sinal ACK seja ativado, deve invalidar os dados baixando o sinal D_{val} e sinalizar que está novamente disponível para entregar dados ao sistema consumidor, ativando o sinal OB_{free}. Na Figura 7, é apresentado o diagrama de blocos do *Output Buffer*. Sempre que o bloco emissor *Ring Buffer* tenha dados disponíveis e o bloco de entrega *Output Buffer* esteja disponível (OB_{free} ativo), o *Ring Buffer* realiza uma leitura da memória e entrega

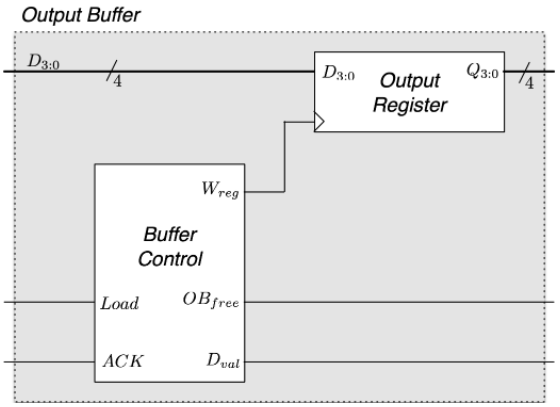
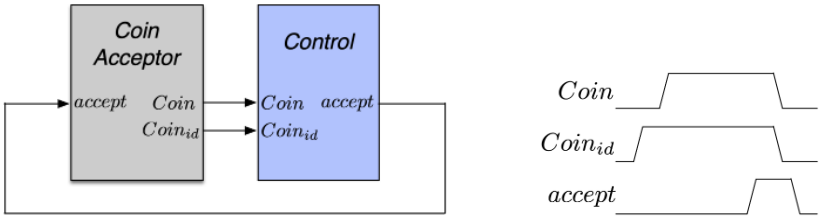


Figura 7: Diagrama de blocos do *Output Buffer*

os dados ao *Output Buffer* ativando o sinal W_{reg}. O *Output Buffer* indica que já registou os dados desativando o sinal OB_{free}.

2.2 *Coin Acceptor*

O módulo *Coin Acceptor* implementa a interface com o moedeiro, sinalizando ao módulo *Control* que o moedeiro recebeu uma moeda através da ativação do sinal *Coin*, identificando o tipo de moeda pelo sinal $Coin_{id}$. A entidade consumidora informa o *Coin Acceptor* que já contabilizou a moeda ativando o sinal *accept*, conforme apresentado no diagrama temporal da Figura 8b.



(a) Diagrama de blocos **(b)** Diagrama temporal

Figura 8: Bloco *Coin Acceptor*

2.3 *Serial LCD Controller*

O módulo *Serial LCD Controller* (*SLCDC*) implementa a interface com o *LCD*, fazendo a receção em série da informação enviada pelo módulo de controlo e entregando-a posteriormente ao *LCD*, conforme representado na Figura 9.

O módulo *SLCDC* recebe em série uma mensagem constituída por cinco (5) bits de informação e um (1) bit de paridade. A comunicação com este módulo realiza-se segundo o protocolo ilustrado na Figura 10, em que o bit RS é o primeiro bit de informação e indica se a mensagem é de controlo ou dados. Os seguintes quatro (4) bits contêm os dados a entregar ao *LCD*. O último bit contém a informação de paridade ímpar, utilizada para detetar erros de transmissão.

O emissor, realizado em software, quando pretende enviar uma trama para o módulo *SLCDC* promove uma condição de início de trama (*Start*), que corresponde a uma transição descendente

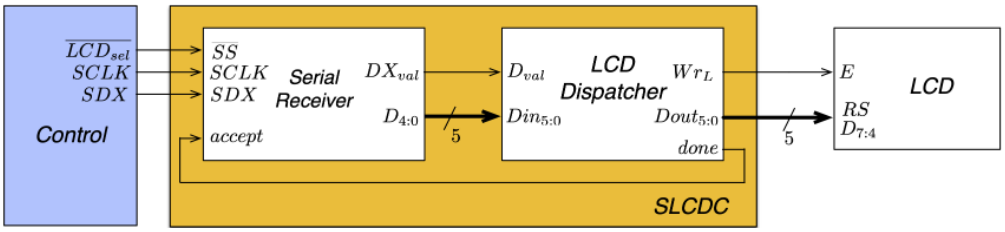


Figura 9: Diagrama de blocos do módulo *Serial LCD Controller* (*SLCDC*)

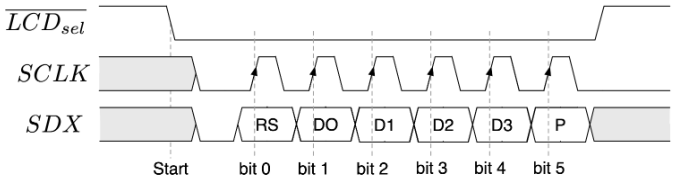


Figura 10: Protocolo de comunicação com o módulo *Serial LCD Controller* (*SLCDC*)

na linha ($\overline{LCD_{sel}}$). Após a condição de início, o módulo *SLCDC* armazena os bits de dados da trama nas transições ascendentes do sinal $SCLK$.

2.3.1 *Serial Receiver*

O bloco *Serial Receiver* do módulo *SLCDC* é constituído por quatro blocos principais: *i*) um bloco de controlo; *ii*) um bloco conversor série paralelo; *iii*) um contador de bits recebidos; e *iv*) um bloco de validação de paridade, designados por *Serial Control*, *Shift Register*, *Counter* e *Parity Check* respetivamente. O bloco *Serial Receiver* deverá ser implementado com base no diagrama de blocos apresentado na Figura 11.

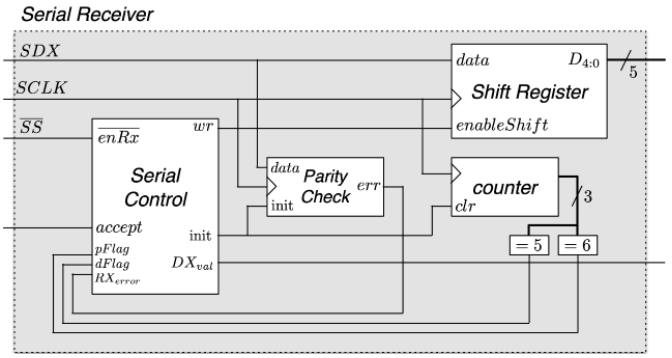


Figura 11: Diagrama de blocos do bloco *Serial Receiver*

2.3.2 *LCD Dispatcher*

O bloco *LCD Dispatcher* é responsável pela entrega das tramas válidas recebidas pelo bloco *Serial Receiver* ao *LCD*, através da ativação do sinal Wr_L. A receção de uma trama válida é sinalizada pela ativação do sinal D_{val}. O processamento das tramas recebidas pelo *LCD* respeita os comandos definidos pelo fabricante, não sendo necessário esperar pela sua execução para

libertar o canal de receção série. Assim, o bloco *LCD Dispatcher* pode ativar, prontamente, o sinal *done* para notificar o bloco *Serial Receiver* que a trama já foi processada.

2.4 *Serial Roulette Controller*

O módulo *Serial Roulette Controller* (*SRC*) implementa a interface com o mostrador da roleta (*Roulette Display*), realizando a receção em série da informação enviada pelo módulo de controlo e entregando-a posteriormente ao mostrador de pontuação, conforme representado na Figura 12. O módulo *SRC* recebe em série uma mensagem composta por oito (8) bits de informação e um

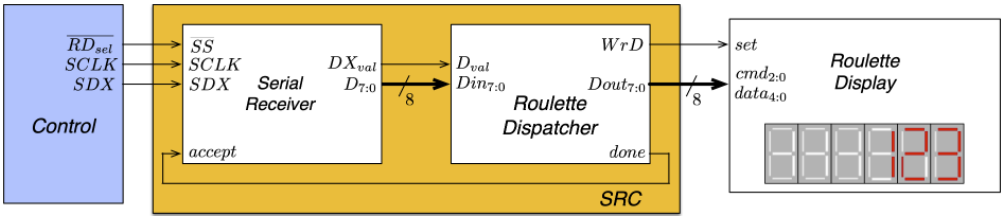


Figura 12: Diagrama de blocos do módulo *Serial Roulette Controller* (*SRC*)

(1) bit de paridade ímpar, segundo o protocolo de comunicação ilustrado na Figura 13. Os três (3) primeiros bits de informação, indicam o comando a realizar no mostrador da roleta, segundo a Tabela 1. Os restantes cinco (5) bits identificam o campo de dados. Tal como acontece com o *SLCDC*, o canal de receção série pode ser libertado após a receção da trama recebida pelo *Roulette Display*, não sendo necessário esperar pela sua execução do comando correspondente. Assim, o bloco *Roulette Dispatcher* pode ativar, prontamente, o sinal *done* para informar o bloco *Serial Receiver* que a trama já foi processada.

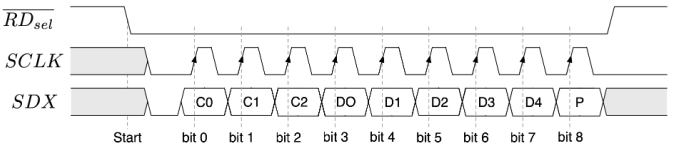


Figura 13: Protocolo de comunicação com o módulo *Serial Roulette Controller* (*SRC*)

Tabela 1: Operações no *Roulette Display*

Cmd 2 1 0	*data* 4 3 2 1 0	*Function*
0 0 0	$d_4\ d_3\ d_2\ d_1\ d_0$	*update digit* 0
0 0 1	$d_4\ d_3\ d_2\ d_1\ d_0$	*update digit* 1
...	...	...
1 0 1	$d_4\ d_3\ d_2\ d_1\ d_0$	*update digit* 5
1 1 0	$-\ -\ -\ -\ -$	*update display*
1 1 1	$-\ -\ -\ -\ 0$	*display on*
1 1 1	$-\ -\ -\ -\ 1$	*display off*

2.4.1 *Serial Receiver*

O bloco *Serial Receiver* do módulo *SRC* deve ser implementado adotando, com as devidas adaptações, uma arquitetura similar à do bloco *Serial Receiver* do módulo *SLCDC*, neste caso adaptando a estrutura para a receção de oito (8) bits de informação em vez de cinco (5) bits.

2.4.2 *Roulette Dispatcher*

Após a receção de uma trama válida (proveniente do bloco *Serial Receiver*), o bloco *Roulette Dispatcher*, deverá proceder à atuação do comando recebido sobre o mostrador da roleta.

2.5 *Control*

A implementação do módulo *Control* deverá ser realizada em software, usando a linguagem *Kotlin* e seguindo a arquitetura lógica apresentada na Figura 14. As assinaturas das principais funções e objetos a desenvolver são apresentadas nas próximas secções. As restantes são objeto de análise e decisão livre.

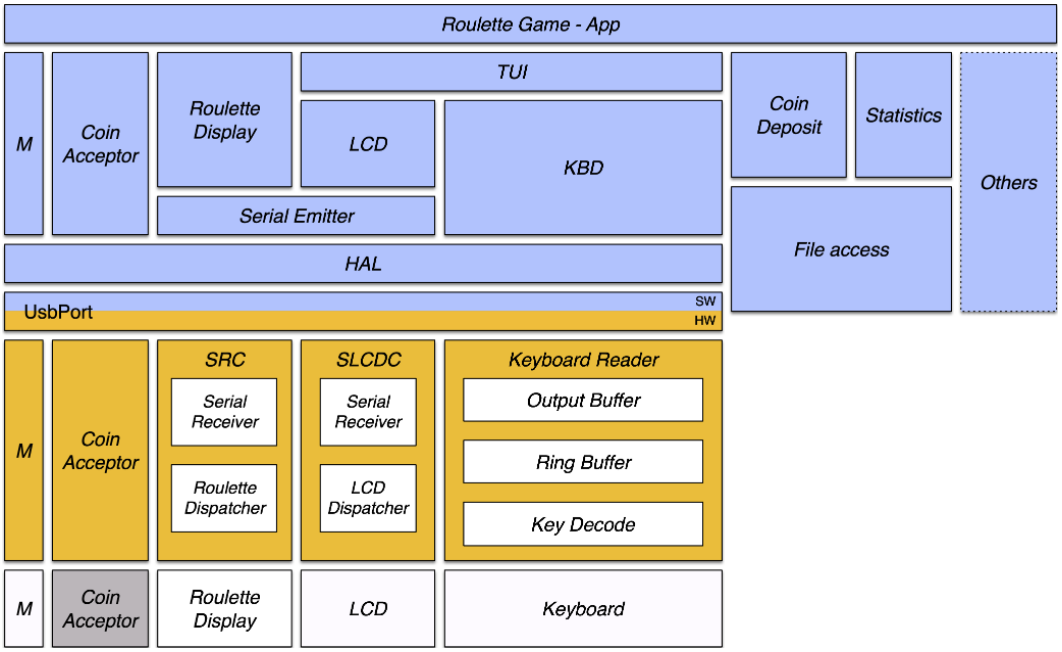


Figura 14: Diagrama lógico do Jogo da Roleta (*Roulette Game*)

Algoritmo 1: HAL - Hardware Abstract Layer

```
object HAL {
// Inicia o objeto
fun init() ...

// Retorna 'true' se o bit definido pela mask esta com o valor logico '1' no UsbPort
fun isBit(mask: Int): Boolean ...

// Retorna os valores dos bits representados por mask presentes no UsbPort
fun readBits(mask: Int): Int ...

// Escreve nos bits representados por mask os valores dos bits correspondentes em value
fun writeBits(mask: Int, value: Int) ...

// Coloca os bits representados por mask no valor logico '1'
fun setBits(mask: Int) ...

// Coloca os bits representados por mask no valor logico '0'
fun clrBits(mask: Int) ...
}
```

Algoritmo 2: KBD - Keyboard

```
// Ler teclas. Funcoes retornam '0'..'9','A'..'D','#','*' ou NONE.
object KBD {
const val NONE = 0;

// Inicia a classe
fun init() ...

// Retorna de imediato a tecla premida ou NONE se nao ha tecla premida.
fun getKey(): Char ...

// Retorna a tecla premida, caso ocorra antes do 'timeout' (em milissegundos),
// ou NONE caso contrario.
fun waitKey(timeout: Long): Char ...
}
```

Algoritmo 3: LCD

```
// Escreve no LCD usando a interface a 4 bits.
object LCD {

// Dimensao do display.
private const val LINES = 2, COLS = 16
// Define se a interface e Serie ou Paralela
private const val SERIAL_INTERFACE = false

// Escreve um byte de comando/dados no LCD em paralelo
private fun writeNibbleParallel(rs: Boolean, data: Int) ...

// Escreve um byte de comando/dados no LCD em serie
private fun writeNibbleSerial(rs: Boolean, data: Int) ...

// Escreve um nibble de comando/dados no LCD
private fun writeNibble(rs: Boolean, data: Int) ...

// Escreve um byte de comando/dados no LCD
private fun writeByte(rs: Boolean, data: Int) ...

// Escreve um comando no LCD
private fun writeCMD(data: Int) ...
```

```
// Escreve um dado no LCD
private fun writeDATA(data: Int) ...

// Envia a sequencia de iniciacao para comunicacao a 4 bits.
fun init() ...

// Escreve um carater na posicao corrente.
fun write(c: Char) ...

// Escreve uma string na posicao corrente.
fun write(text: String) ...

// Envia comando para posicionar cursor ('line':0..LINES-1 , 'column':0..COLS-1)
fun cursor(line: Int, column: Int) ...

// Envia comando para limpar o ecra e posicionar o cursor em (0,0)
fun clear() ...
}
```

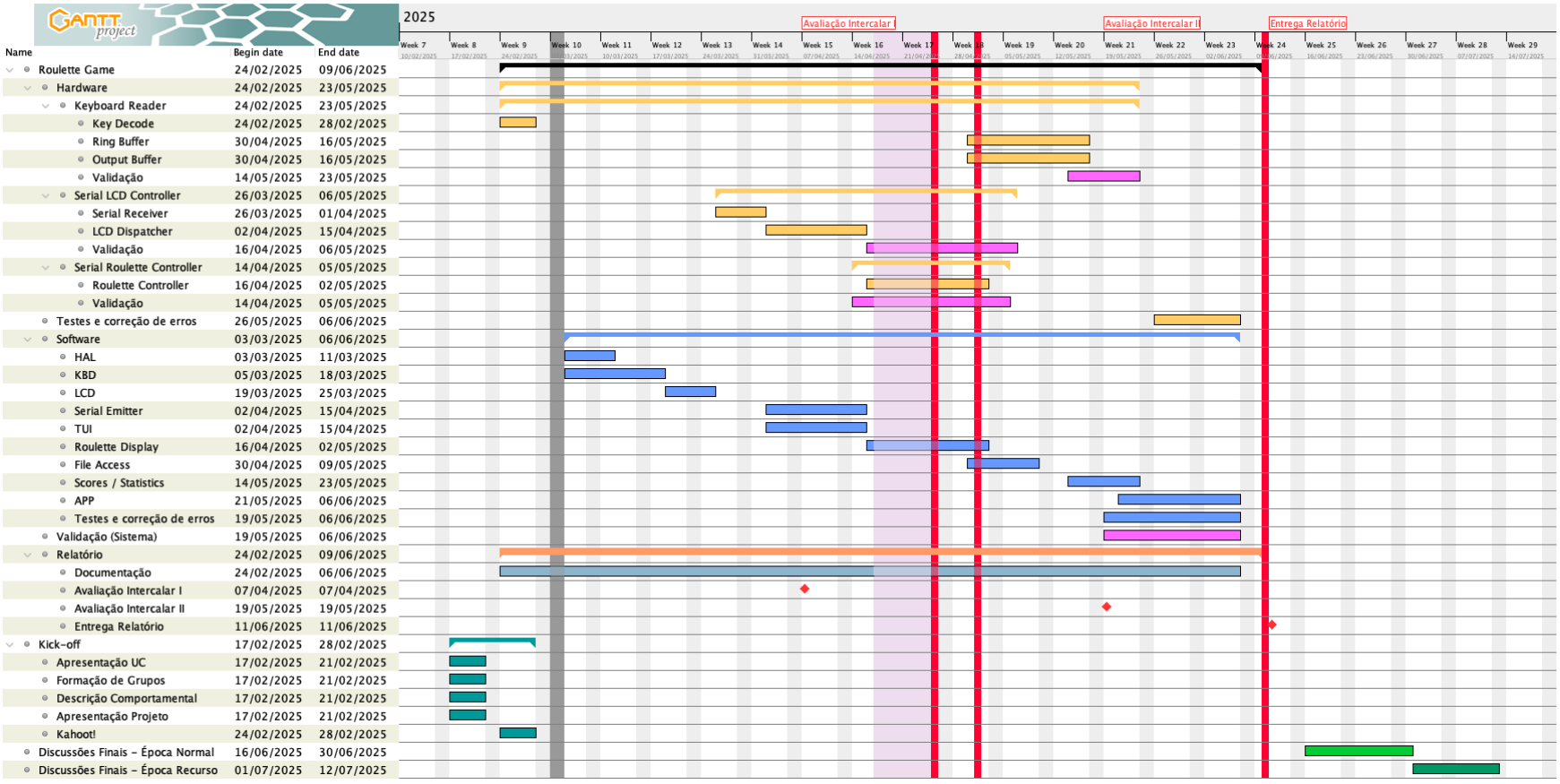
Algoritmo 4: Serial Emitter

```
// Envia tramas para os diferentes modulos Serial Receiver.
object SerialEmitter {

enum class Destination {LCD, ROULETTE}

// Inicia a classe
fun init() ...

// Envia uma trama para o SerialReceiver
// identificado o destino em 'addr',
// os bits de dados em 'data'
// e em 'size' o numero de bits a enviar.
fun send(addr: Destination, data: Int, size : Int) ...
}
```

Algoritmo 5: Roulette Display

```
// Controla o mostrador de pontuacao.
object RouletteDisplay {

// Inicia a classe, estabelecendo os valores iniciais.
fun init() ...

// Realiza a animacaoo do sorteio
fun animation()

// Envia comando para atualizar o valor do mostrador da roleta
fun setValue(value: Int) ...

// Envia comando para desativar/ativar a visualizacao do mostrador da roleta
fun off(value: Boolean) ... }
```

3 Calendarização



Name	Begin date	End date
Roulette Game	24/02/2025	09/06/2025
Hardware	24/02/2025	23/05/2025
Keyboard Reader	24/02/2025	23/05/2025
Key Decode	24/02/2025	28/02/2025
Ring Buffer	30/04/2025	16/05/2025
Output Buffer	30/04/2025	16/05/2025
Validação	14/05/2025	23/05/2025
Serial LCD Controller	26/03/2025	06/05/2025
Serial Receiver	26/03/2025	01/04/2025
LCD Dispatcher	02/04/2025	15/04/2025
Validação	16/04/2025	06/05/2025
Serial Roulette Controller	14/04/2025	05/05/2025
Roulette Controller	16/04/2025	02/05/2025
Validação	14/04/2025	05/05/2025
Testes e correção de erros	26/05/2025	06/06/2025
Software	03/03/2025	06/06/2025
HAL	03/03/2025	11/03/2025
KBD	05/03/2025	18/03/2025
LCD	19/03/2025	25/03/2025
Serial Emitter	02/04/2025	15/04/2025
TUI	02/04/2025	15/04/2025
Roulette Display	16/04/2025	02/05/2025
File Access	30/04/2025	09/05/2025
Scores / Statistics	14/05/2025	23/05/2025
APP	21/05/2025	06/06/2025
Testes e correção de erros	19/05/2025	06/06/2025
Validação (Sistema)	19/05/2025	06/06/2025
Relatório	24/02/2025	09/06/2025
Documentação	24/02/2025	06/06/2025
Avaliação Intercalar I	07/04/2025	07/04/2025
Avaliação Intercalar II	19/05/2025	19/05/2025
Entrega Relatório	11/06/2025	11/06/2025
Kick-off	17/02/2025	28/02/2025
Apresentação UC	17/02/2025	21/02/2025
Formação de Grupos	17/02/2025	21/02/2025
Descrição Comportamental	17/02/2025	21/02/2025
Apresentação Projeto	17/02/2025	21/02/2025
Kahoot!	24/02/2025	28/02/2025
Discussões Finais – Época Normal	16/06/2025	30/06/2025
Discussões Finais – Época Recurso	01/07/2025	12/07/2025

Figura 15: Diagrama de *Gantt* relativo à calendarização do projeto)