ISEL
ADEETC
Área Departamental de
Engenharia Electrónica e
de Telecomunicações e
de Computadores

Licenciatura em Engenharia Informática e de Computadores

Jogo da Roleta
(*Roulette Game*)

Projeto
de
Laboratório de Informática e Computadores
2020 / 2021 inverno

1 Descrição

Pretende-se implementar o jogo da Roleta (*Roulette Game*), no qual a roleta compreende números entre 0 e 9, um jogador realiza apostas premindo as teclas de um teclado correspondentes aos números 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 moedas de 1.00€, que corresponde a dois créditos.
O sistema que implementa o jogo será constituído por: um PC (*Control*); um teclado de 12 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.

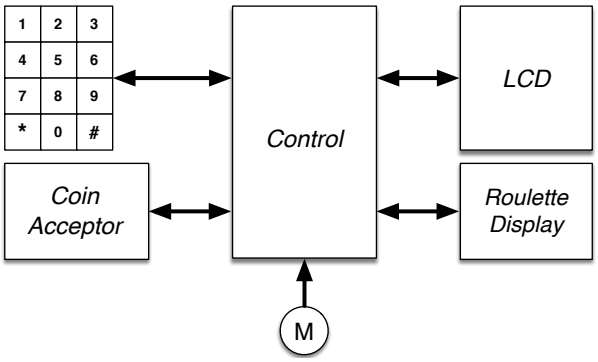


Figura 1 – Diagrama de blocos do jogo da Roleta (*Roulette Game*)

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

- **Jogo –** O jogo inicia-se quando é premida a tecla '*' e existem créditos disponíveis. Utilizando as teclas numéricas (0-9) 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 é apresentado no *Roulette Display* e os créditos obtidos na jogada são apresentados no LCD. Os créditos obtidos são acumulados após 5 segundos ao saldo do jogador, também apresentado 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 '#' permite-se a listagem dos contadores de moedas e jogos realizados.
- **Iniciar os contadores de moedas e jogos –** Premindo a tecla '#' 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 '0' permite-se a listagem dos números sorteados.
- **Iniciar a lista de números sorteados –** Premindo a tecla '0' e em seguida a tecla '*', o sistema inicia um novo ciclo de estatística de números sorteados.
- **Desligar –** 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 quatro módulos principais: *i*) um leitor de teclado, designado por *Keyboard Reader*; *ii*) um módulo de interface com o *LCD* e com o mostrador da roleta, designado por *Serial Output Controller* (*SOC*); *iii*) um moedeiro, designado por *Coin Acceptor*; e *iv*) um módulo de controlo, designado por *Control*. Os módulos *i*) e *ii*) deverão ser implementados em *hardware,* o moedeiro será simulado utilizando um interruptor e um LED, enquanto o módulo de controlo deverá ser implementado em *software* escrito usando linguagem Java e que será executado num PC.

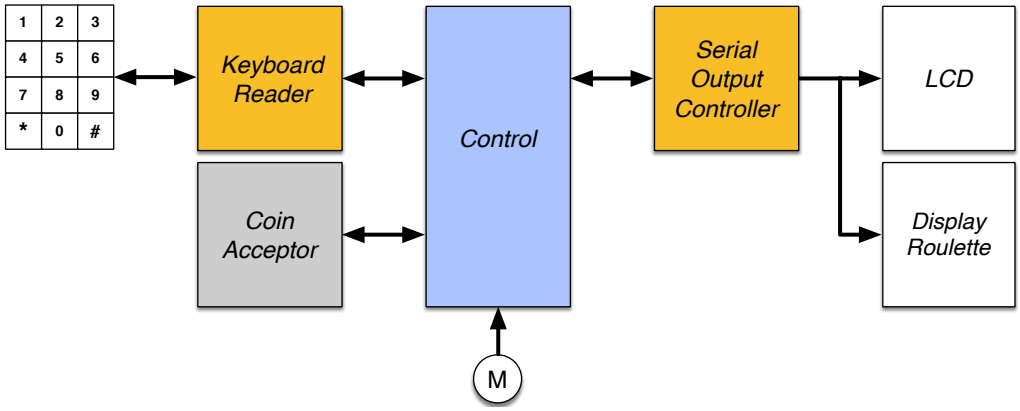


Figura 2 – Arquitetura do sistema que implementa o jogo da Roleta (*Roulette Game*)

O módulo *Keyboard Reader* é responsável pela descodificação do teclado matricial de 12 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 dois códigos. O módulo *Control* processa os dados e envia a informação a apresentar no *LCD* através do módulo *SOC*. O *Roulette Display* é atuado pelo módulo *Control*, através do módulo *SOC*. Por razões de ordem física, e por forma a minimizar o número de fios de interligação, a comunicação entre o módulo *Control* e o módulo *SOC* é realizada através de um protocolo série.

2.1 Keyboard Reader

O módulo *Keyboard Reader* é constituído por dois blocos principais: *i*) o descodificador de teclado *Key Decode*; e *ii*) o bloco de armazenamento e de entrega ao consumidor, designado por *Key Buffer*, conforme ilustrado na Figura 3. Neste caso o módulo de controlo, implementado em *software*, é a entidade consumidora.

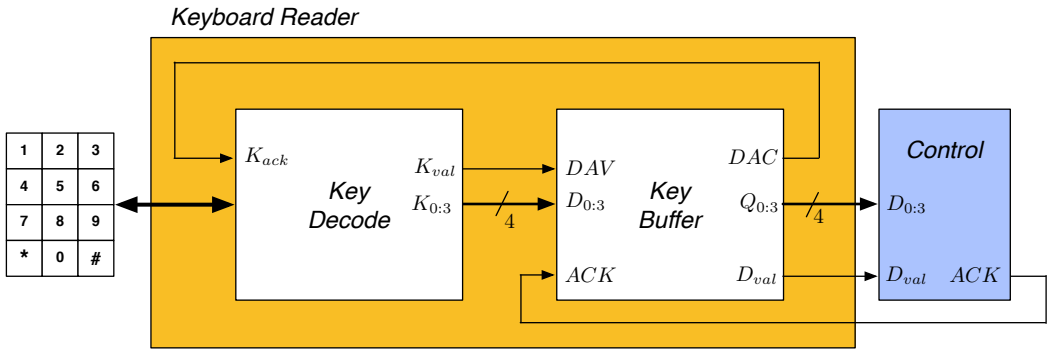
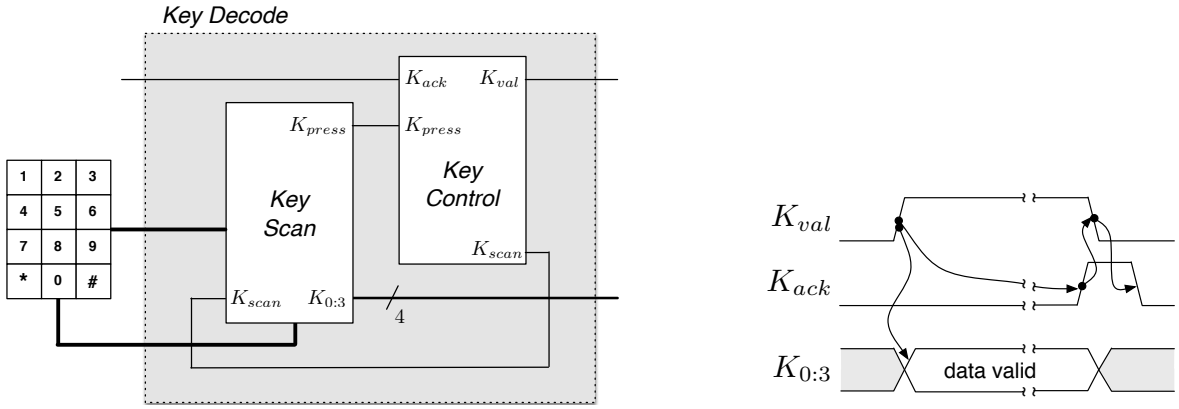


Figura 3 – Diagrama de blocos do módulo *Keyboard Reader*

2.1.1 Key Decode

O bloco *Key Decode* deverá implementar um descodificador de um teclado matricial 4x3 por *hardware*, sendo constituído por três sub-blocos: *i)* um teclado matricial de 4x3; *ii)* o bloco *Key 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.

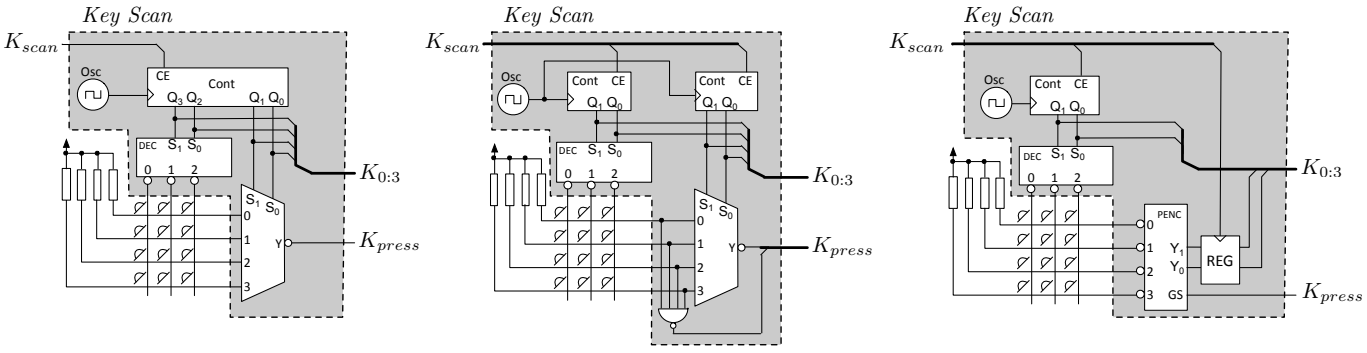
O controlo de fluxo de saída do bloco *Key Decode* (para o módulo *Key Buffer*), define que o sinal K_{val} deve ser ativado quando uma tecla é pressionada, sendo também disponibilizando o código dessa tecla no barramento $K_{0:3}$. Apenas é iniciado um novo ciclo de varrimento ao teclado quando o sinal 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 Key Buffer

O bloco *Key Buffer* a desenvolver corresponderá a uma estrutura de armazenamento de dados, com capacidade para armazenar uma palavra de quatro bits. A escrita de dados no bloco *Key Buffer*, cujo diagrama de blocos é apresentado na Figura 6, inicia-se com a ativação do sinal *DAV* (*Data Available*) pelo sistema produtor, neste caso pelo bloco *Key Decode*, indicando que tem dados ($D_{0:3}$) para serem armazenados. Logo que tenha disponibilidade para armazenar esta informação, o bloco *Key Buffer* regista os dados 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 o sinal *DAC* seja ativado. O bloco *Key Buffer* só desativa o sinal *DAC* após o sinal *DAV* ter sido desativado.
A implementação do bloco *Key Buffer* deverá ser baseada numa máquina de controlo (*Key Buffer Control*) e num registo (*Output Register*).

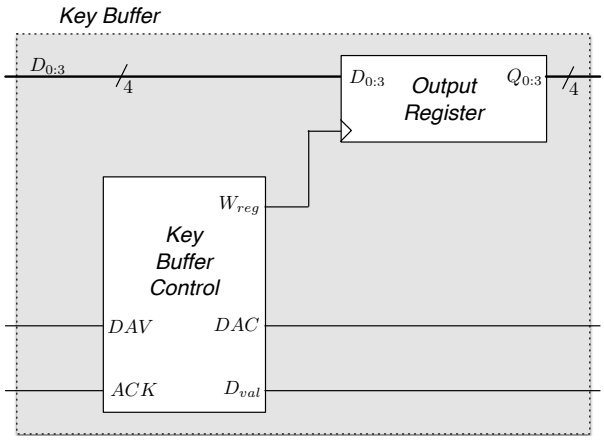
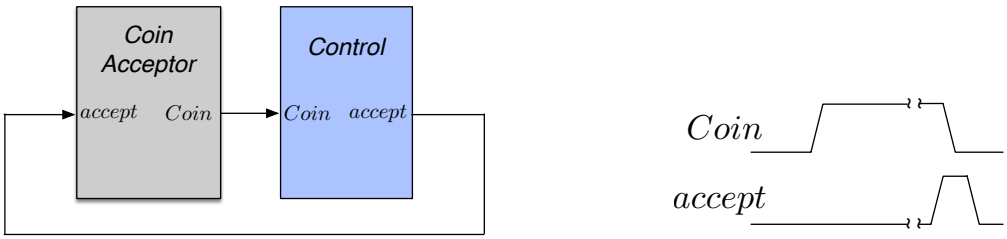


Figura 6 – Diagrama de blocos do bloco *Key Buffer*

O sub-bloco *Key Buffer Control* do bloco *Key Buffer* também é responsável pela interação com o sistema consumidor, neste caso o módulo *Control*. Quando pretende ler dados do bloco *Key Buffer*, o módulo *Control* aguarda que o sinal D_{val} fique ativo, recolhe os dados e ativa o sinal *ACK* para indicar que estes já foram consumidos. Logo que o sinal *ACK* fique ativo, o módulo *Key Buffer Control* deve invalidar os dados armazenados no *Output Register* baixando o sinal D_{val}. Para que uma nova palavra possa ser armazenada será necessário que o módulo *Control* tenha desativado o sinal *ACK*.

2.2 Coin Acceptor

O módulo *Coin Acceptor* simula a interface com o moedeiro, sinalizando ao módulo *Control* que o moedeiro recebeu uma moeda através da ativação do sinal *Coin*. A entidade consumidora informa o *Coin Acceptor* que já contabilizou a moeda ativando o sinal *accept*, conforme apresentado no diagrama temporal da Figura 7. O sinal *Coin* do moedeiro deverá ser gerado através de um interruptor e o sinal *accept* deverá ser visualizado num LED.



a) Diagrama de blocos b) Diagrama temporal

Figura 7 – Módulo *Coin Acceptor*

2.3 Serial Output Controller

O módulo *Serial Output Controller* (*SOC*) implementa a interface com o *LCD* e com o mostrador da roleta, fazendo a receção da informação enviada pelo módulo de controlo e entregando-a posteriormente ao destinatário, conforme representado na Figura 8.

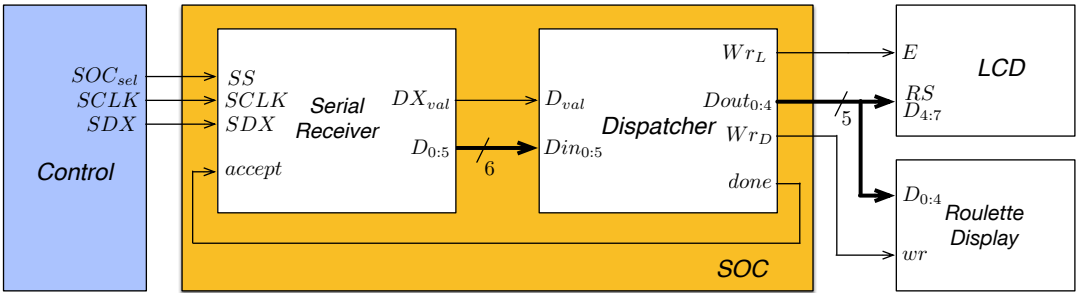


Figura 8 – Diagrama de blocos do módulo *Serial Output Controller*

O módulo *SOC* recebe, em série, uma mensagem constituída por 6 bits de informação e um bit de paridade, utilizado para detetar erros de transmissão. A comunicação com este módulo realiza-se segundo o protocolo ilustrado na Figura 9, em que o bit *LnD* identifica o destinatário da mensagem e o último bit (P) contém a informação de paridade ímpar. Nas mensagens para o LCD, ilustradas na Figura 10, o bit *RS* é o primeiro bit de informação e indica se a mensagem é de controlo ou dados. Os seguintes 4 bits contêm os dados a entregar ao *LCD*. Nas mensagens para o *Roulette Display*, ilustradas na Figura 11, os 5 bits constituem os dados a entregar ao dispositivo.

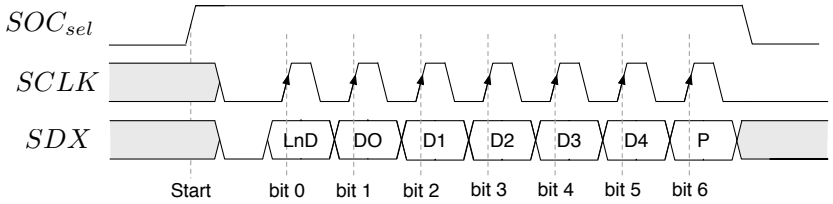


Figura 9 – Protocolo de comunicação com o módulo *Serial Output Controller*

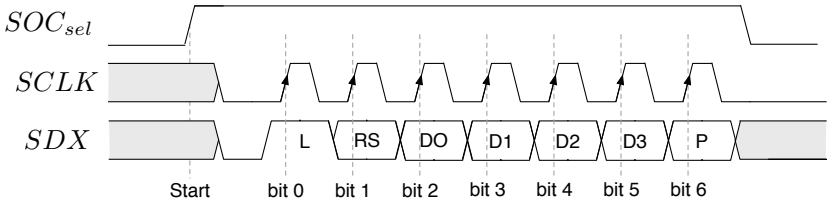


Figura 10 – Trama para o *LCD*

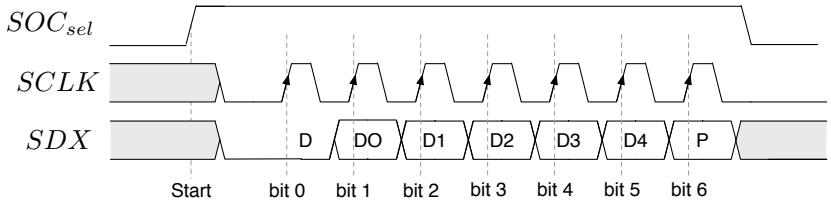


Figura 11 – Trama para o *Roulette Display*

O emissor, realizado em *software,* quando pretende enviar uma trama para o módulo *SOC* promove uma condição de início de trama (*Start*), que corresponde a uma transição ascendente na linha SOC_{sel}. Após a condição de início, o módulo *SOC* é responsável por armazenar os bits de dados da trama nas transições ascendentes do sinal *SCLK*.

2.3.1 Serial Receiver

O bloco *Serial Receiver* do módulo *SOC* 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 12.

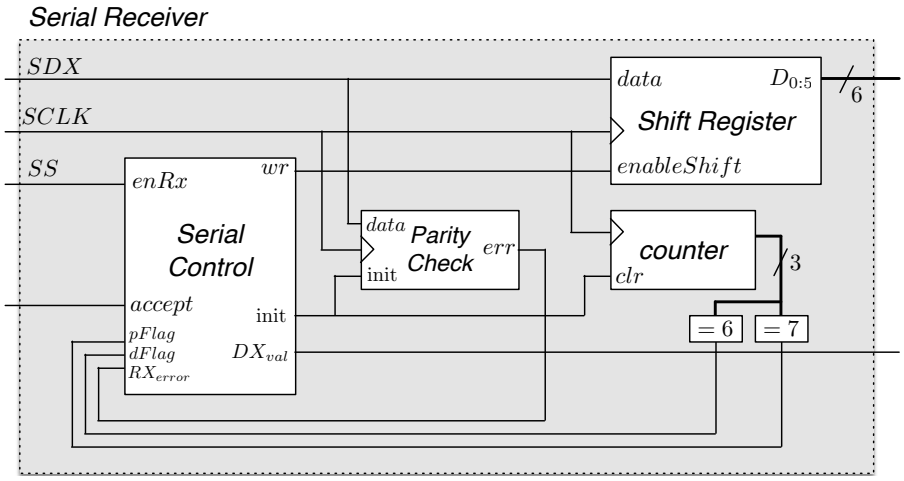


Figura 12 – Diagrama de blocos do bloco *Serial Receiver*

2.3.2 Dispatcher

O bloco *Dispatcher* é responsável pela entrega ao *LCD* e ao *Roulette Display* das tramas válidas recebidas pelo bloco *Serial Receiver*. A receção de uma nova trama válida é sinalizada pela ativação do sinal D_{val} e a sua entrega ao periférico alvo é feita através da ativação dos sinais Wr_L (para o LCD) e Wr_D (para o *Roulette Display*).
O processamento das tramas recebidas pelo *SOC,* para o *LCD* ou para o *Roulette Display,* deverá respeitar a especificação definida pelo fabricante de cada periférico e possibilitar a libertação do canal de receção série o mais rapidamente possível.

2.4 *Control*

A implementação do módulo *Control* deverá ser realizada em *software* usando a linguagem Java e seguindo a arquitetura lógica apresentada na Figura 13.

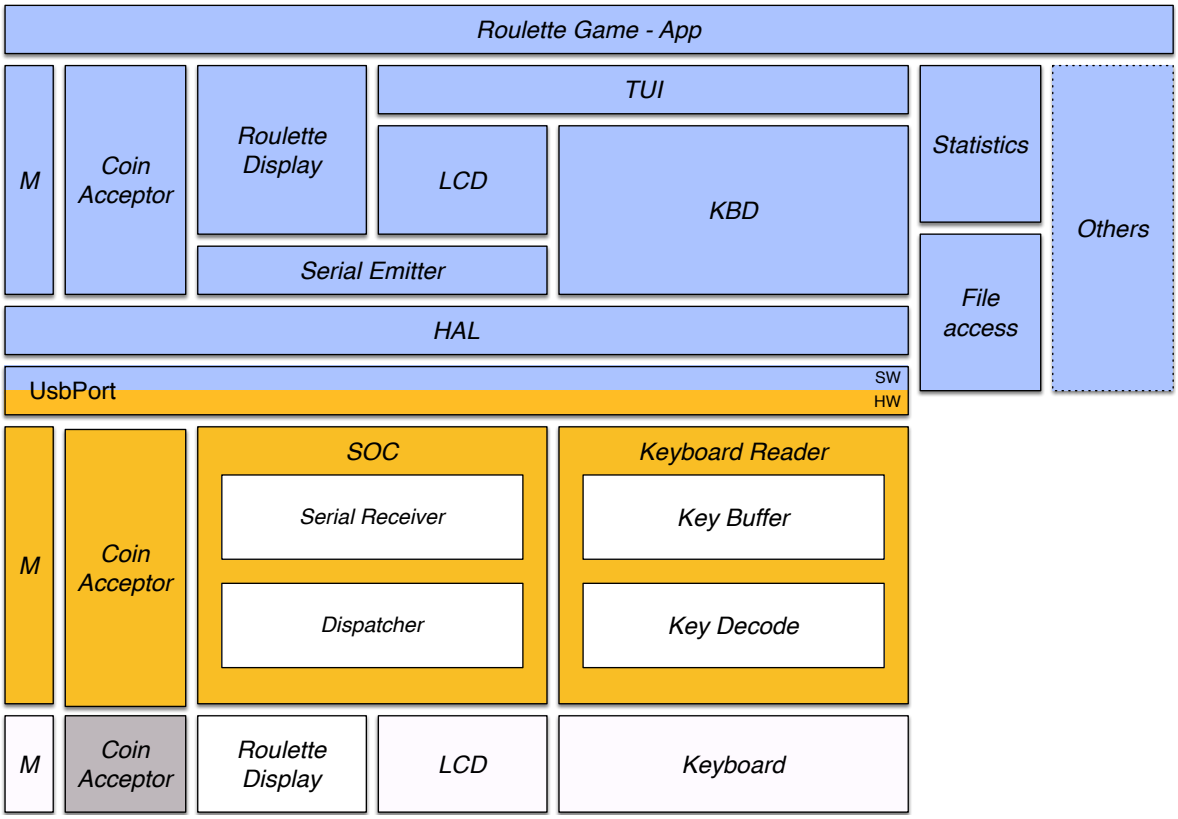


Figura 13 – Diagrama lógico do Jogo da Roleta (*Roulette Game*)

As assinaturas das principais classes a desenvolver são apresentadas nas próximas secções. As restantes são objeto de análise e decisão livre.

2.4.1 *Classe HAL*

```
public class HAL {          // Virtualiza o acesso ao sistema UsbPort
   // Inicia a classe
   public static void init() ...
   // Retorna true se o bit tiver o valor lógico '1'
   public static boolean isBit(int mask) ...
   // Retorna os valores dos bits representados por mask presentes no UsbPort
   public static int readBits(int mask) ...
   // Escreve nos bits representados por mask o valor de value
   public static void writeBits(int mask, int value) ...
   // Coloca os bits representados por mask no valor lógico '1'
   public static void setBits(int mask) ...
   // Coloca os bits representados por mask no valor lógico '0'
   public static void clrBits(int mask) ...
}
```

2.4.2 *Classe KBD*

```
public class KBD {          // Ler teclas. Métodos retornam '0'..'9','#','*' ou NONE.
   public static final char NONE = 0;
   // Inicia a classe
   public static void init() ...
```

```
   // Retorna de imediato a tecla premida ou NONE se não há tecla premida.
   public static char getKey() ...
   // Retorna quando a tecla é premida ou NONE após decorrido 'timeout' milisegundos.
   public static char waitKey(long timeout) ...
}
```

2.4.3 *Classe SerialEmitter*

```
public class SerialEmitter { // Envia tramas para o módulo Serial Receiver.
   public static enum Destination {RDisplay,LCD};
   // Inicia a classe
   public static void init() ...
   // Envia uma trama para o Serial Receiver identificando o destino em addr e os bits de dados em
   'data'.
   public static void send(Destination addr, int data) ...
}
```

2.4.4 *Classe LCD*

```
public class LCD {           // Escreve no LCD usando a interface a 4 bits.
   private static final int LINES = 2, COLS = 16; // Dimensão do display.
   // Escreve um nibble de comando/dados no LCD
   private static void writeNibble(boolean rs, int data) ...
   // Escreve um byte de comando/dados no LCD
   private static void writeByte(boolean rs, int data) ...
   // Escreve um comando no LCD
   private static void writeCMD(int data) ...
   // Escreve um dado no LCD
   private static void writeDATA(int data) ...
   // Envia a sequência de iniciação para comunicação a 4 bits.
   public static void init() ...
   // Escreve um caráter na posição corrente.
   public static void write(char c) ...
   // Escreve uma string na posição corrente.
   public static void write(String txt) ...
   // Envia comando para posicionar cursor ('lin':0..LINES-1 , 'col':0..COLS-1)
   public static void cursor(int lin, int col) ...
}
```

2.4.5 *Classe RouletteDisplay*

```
public class RouletteDisplay {      // Controla o Roulette Display.
   // Inicia a classe, estabelecendo os valores iniciais.
   public static void init() ...
   // Envia comando para apresentar o número sorteado
   public static void showNumber(int number) ...
   // Envia comando de animação
   public static void animation() ...
}
```

3 Calendarização do projeto

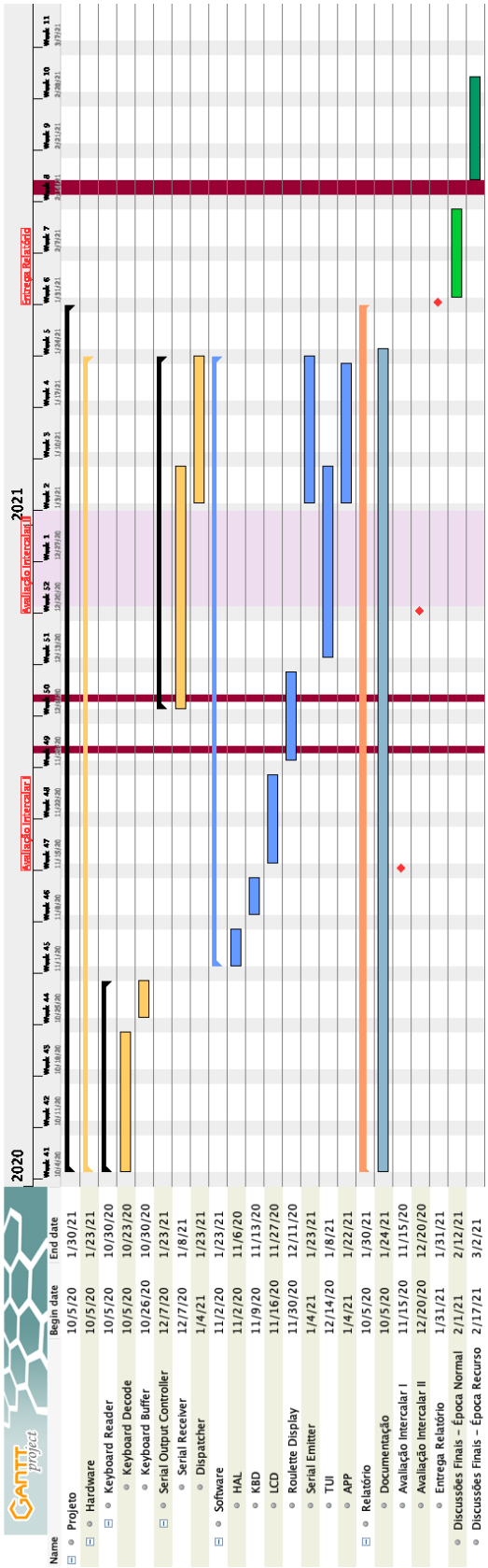


Figura 14 – Diagrama de Gantt relativo à calendarização do projeto