ISEL
Departamento de Engenharia Eletrónica e Telecomunicações e de Computadores

Licenciatura em Engenharia Informática e de Computadores

Jogo Invasores Espaciais
(*Space Invaders Game*)

Projeto
de
Laboratório de Informática e Computadores
2023 / 2024
verão

publicado: 21 de fevereiro de 2024

atualizado: 20 de março de 2024

1 Descrição

Pretende-se implementar o jogo Invasores Espaciais (*Space Invaders Game*) utilizando um PC e periféricos para interação com o jogador. Neste jogo, os invasores espaciais são representados por números entre 0 e 9, e a nave espacial realiza mira sobre o primeiro invasor da fila eliminando-o, se no momento do disparo os números da mira e do invasor coincidirem. O jogo termina quando os invasores espaciais atingirem a nave espacial. Para se iniciar um jogo é necessário um crédito, obtido pela introdução de moedas. O sistema só aceita moedas de 1,00€, que correspondem a dois créditos.
O sistema de jogo é constituído por: um teclado de 12 teclas; um moedeiro (*Coin Acceptor*); um mostrador *Liquid Cristal Display* (*LCD*) de duas linhas com 16 caracteres; um mostrador de pontuação (*Score Display*) e uma chave de manutenção designada por *M*, para colocação do sistema em modo de Manutenção. O diagrama de blocos do jogo Invasores Espaciais é apresentado na Figura 1.

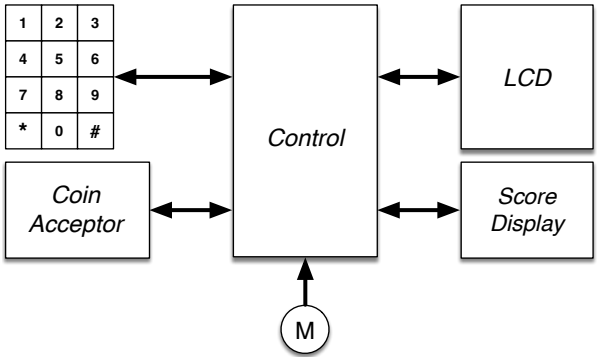


Figura 1 – Diagrama de blocos do jogo Invasores Espaciais (*Space Invaders Game*)

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

- **Jogo –** O jogo inicia-se quando for premida a tecla '#' e existirem créditos disponíveis. Os Invasores Espaciais aparecem do lado direito do *LCD*, em ambas as linhas. Ao premir a tecla '*' a mira do canhão da nave permuta de linha, utilizando as teclas numéricas (0-9) efetua-se a mira sobre o invasor sendo este eliminado após a realização do disparo que é executado quando for premida a tecla '#'. O jogo termina quando os invasores atingirem a nave espacial. A pontuação final é determinada pelo acumular dos pontos realizados durante o jogo, estes são obtidos através da eliminação dos invasores.
- **Visualização da Lista de Pontuações –** Esta ação é realizada sempre que o sistema está modo de espera de início de um novo jogo e após a apresentação, por 10 segundos da mensagem de identificação do jogo.

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

- **Teste** – Permite realizar um jogo, sem créditos e sem a pontuação do jogo ser contabilizada para a Lista de Pontuações.
- **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.
- **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 Pontuações, 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 Pontuações, que compreende as 20 melhores pontuações e o respetivo nome do jogador. 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 de pontuação (*Score Display*), designado por *Serial Score Controller* (*SSC*); *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 deverá ser simulado, enquanto o módulo de controlo deverá ser implementado em *software* a executar num PC usando linguagem *Kotlin*.

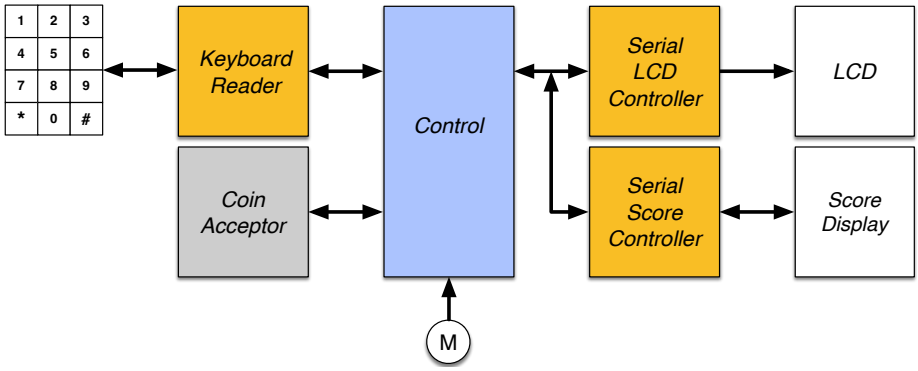


Figura 2 – Arquitetura do sistema que implementa o jogo Invasores Espaciais (*Space Invaders Game*)

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 até ao limite de dez códigos. O módulo *Control* processa os dados e envia a informação a apresentar no *LCD* através do módulo *SLCDC*. O mostrador de pontuação é atuado pelo módulo *Control*, através do módulo *SSC*. Por razões de ordem física, e por forma a minimizar o número de interligações, a comunicação entre o módulo *Control* e os módulos *SLCDC e SSC* é realizada através de um protocolo série síncrono.

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.

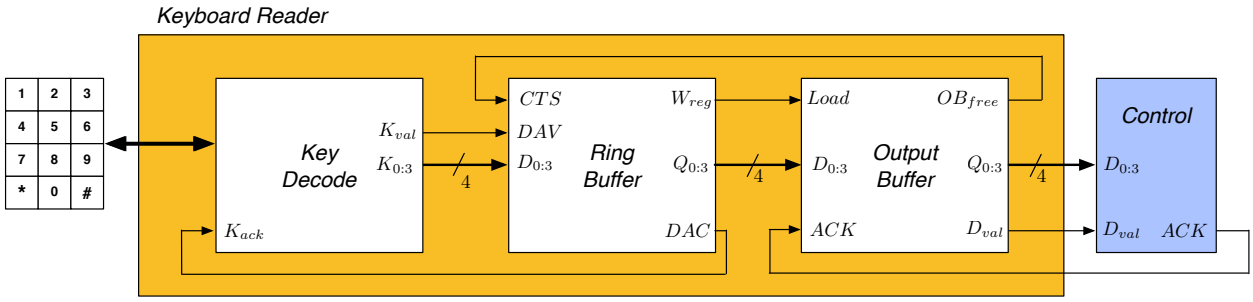
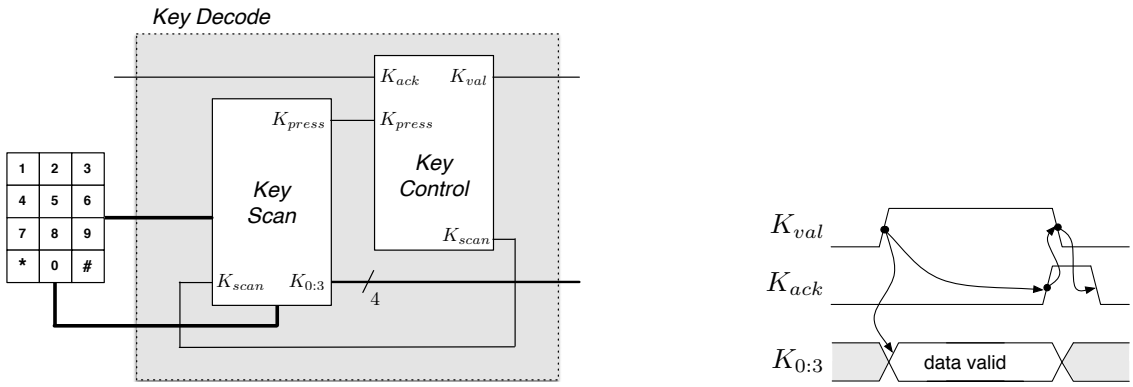


Figura 3 – Diagrama de blocos do *Keyboard Reader*

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.

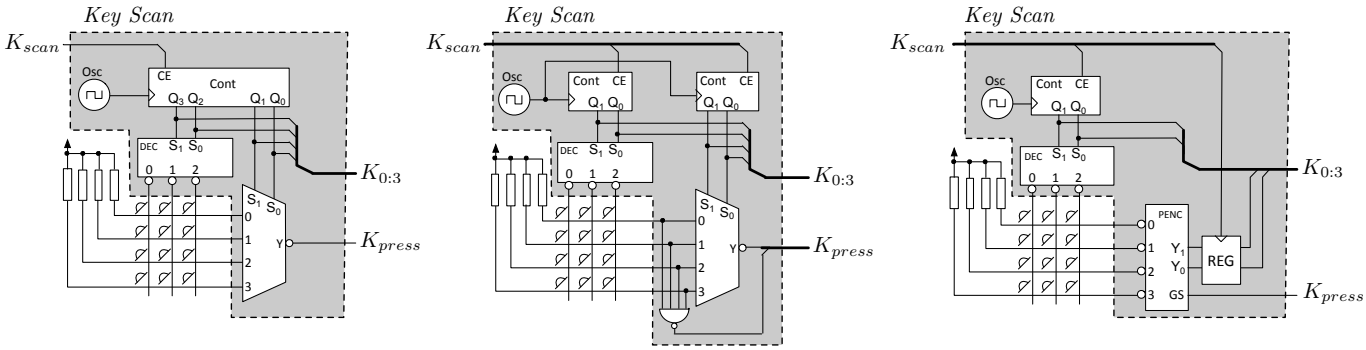
O controlo de fluxo de saída do bloco *Key Decode* (para o bloco *Ring Buffer*) define que o sinal K_{val} é ativado quando é detetada a pressão de uma tecla, sendo também disponibilizado o código dessa tecla no barramento $K_{0:3}$. Apenas é iniciado um novo ciclo de varrimento ao teclado quando o sinal 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 *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 oito palavras de quatro 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 $D_{0:3}$ 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*.

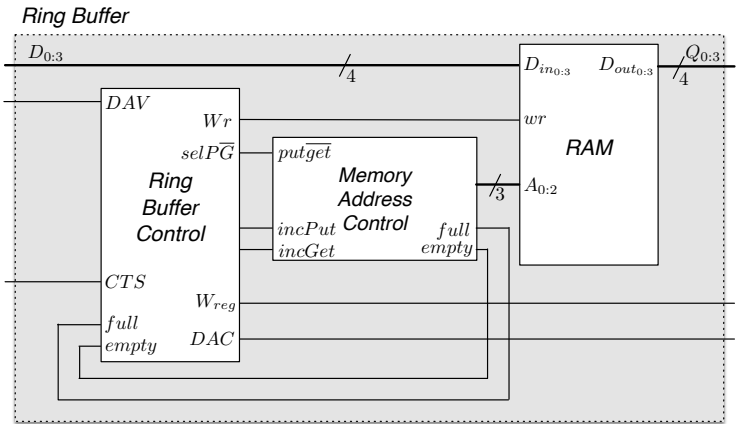


Figura 6 – Diagrama de blocos do bloco *Ring Buffer*

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*.

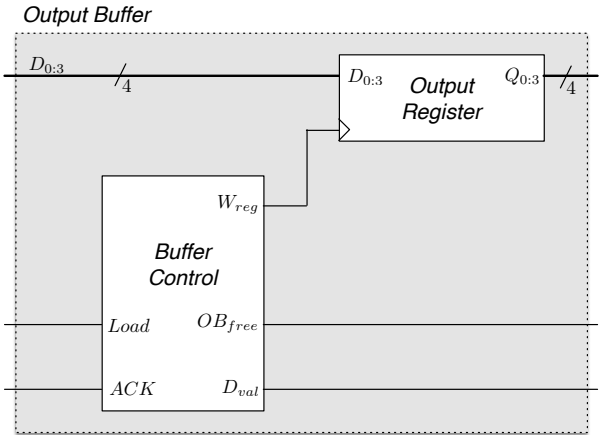
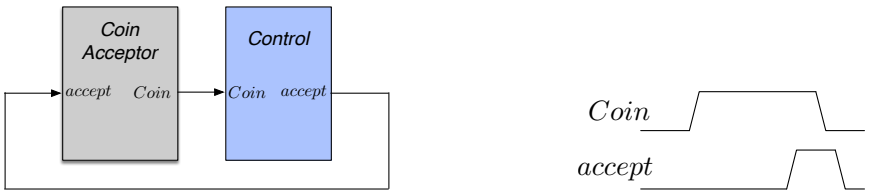


Figura 7 – Diagrama de blocos do bloco *Output Buffer*

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 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*. A entidade consumidora informa o *Coin Acceptor* que já contabilizou a moeda ativando o sinal *accept*, conforme apresentado no diagrama temporal da Figura 8.



a) Diagrama de blocos b) Diagrama temporal

Figura 8 – Módulo *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.

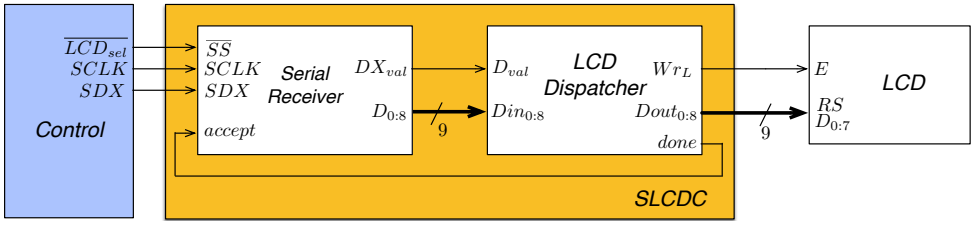


Figura 9 – Diagrama de blocos do módulo *Serial LCD Controller*

O módulo *SLCDC* recebe em série uma mensagem constituída por nove (9) 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 oito (8) bits contêm os dados a entregar ao *LCD*. O último bit contém a informação de paridade par, utilizada para detetar erros de transmissão.

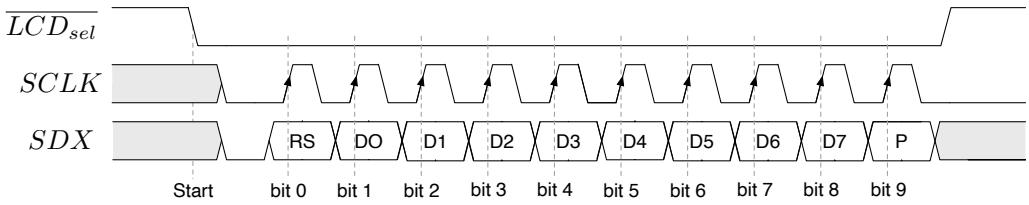


Figura 10 – Protocolo de comunicação com o módulo *Serial LCD Controller*

O emissor, realizado em *software,* quando pretende enviar uma trama para o módulo S*LCDC* promove uma condição de início de trama (*Start*), que corresponde a uma transição descendente na linha $\overline{LCD_{sel}}$. Após a condição de início, o módulo S*LCDC* armazena os bits de dados da trama nas transições ascendentes do sinal *SCLK*.

2.3.1 Serial Receiver

O bloco *Serial Receiver* do módulo S*LCDC* 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.

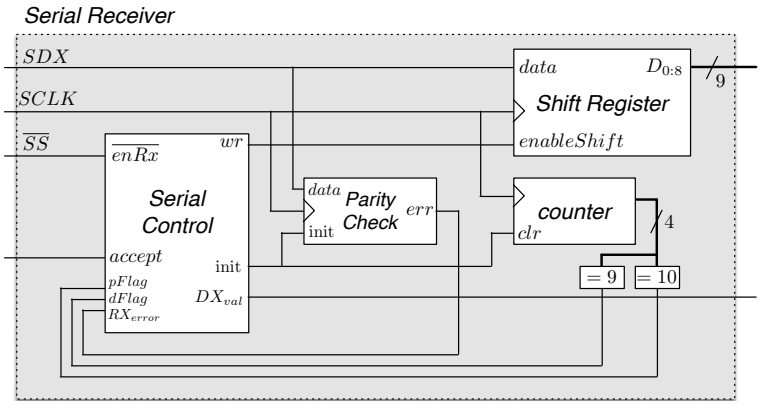


Figura 11 – Diagrama de blocos do bloco *Serial Receiver*

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 Score Controller

O módulo *Serial Score Controller* (*SSC*) implementa a interface com o mostrador de pontuação (*Score 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.

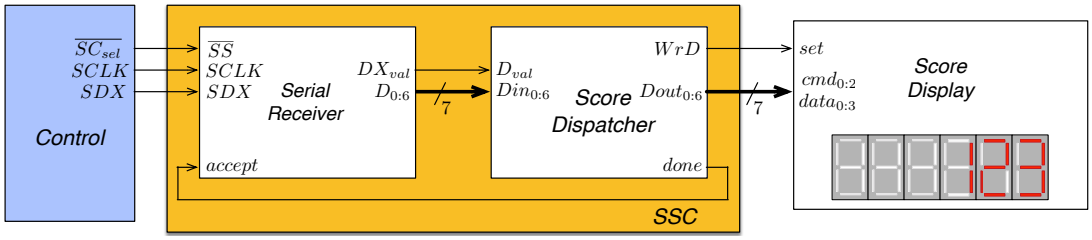


Figura 12 – Diagrama de blocos do módulo *Serial Score Controller*

O módulo *SSC* recebe em série uma mensagem composta por sete (7) bits de informação e um (1) bit de paridade par, segundo o protocolo de comunicação ilustrado na Figura 13. Os três primeiros bits de informação, indicam o comando a realizar no mostrador de pontuação, segundo a Tabela 1. Os restantes quatro 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 *Score Display*, não sendo necessário esperar pela sua execução do comando correspondente. Assim, o bloco *Score Dispatcher* pode ativar, prontamente, o sinal *done* para informar o bloco *Serial Receiver* que a trama já foi processada.

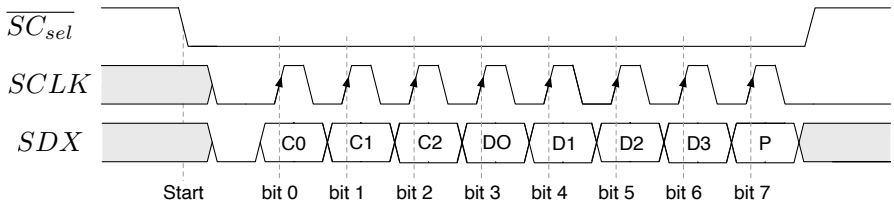


Figura 13 – Protocolo de comunicação do módulo *Serial Score Controller*

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

Tabela 1 – Comandos do módulo *Score Display*

2.4.1 Serial Receiver

O bloco *Serial Receiver* do módulo *SSC* 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 sete (7) bits de informação em vez de nove (9) bits.

2.4.2 Score Dispatcher

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

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

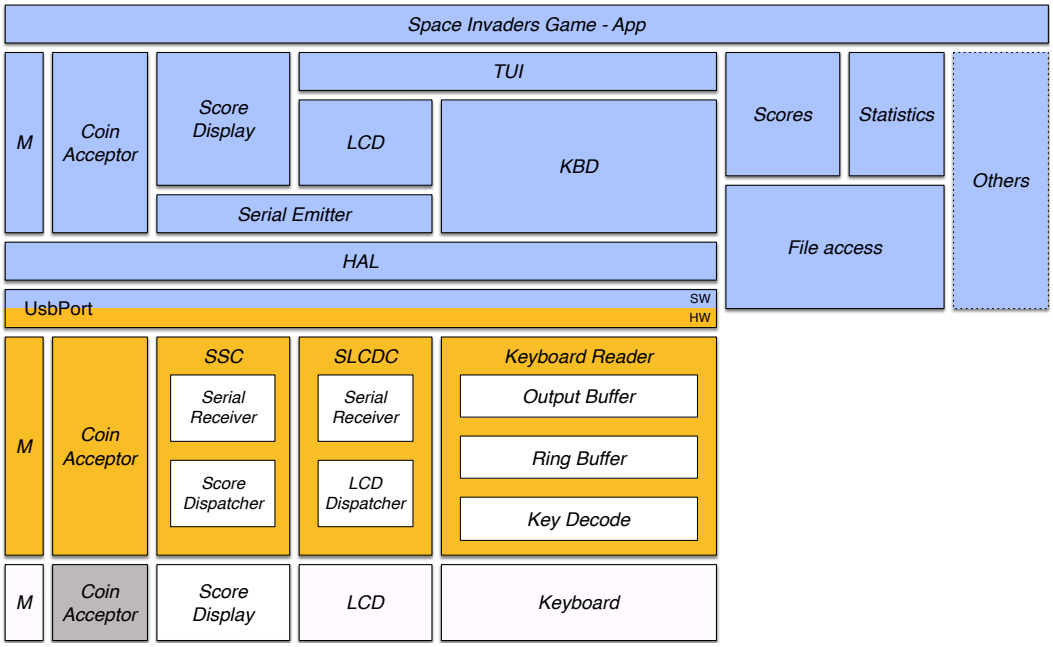


Figura 14 – Diagrama lógico do Jogo Invasores Espaciais (*Space Invaders Game*)

2.5.1 HAL

```
object HAL {   // Virtualiza o acesso ao sistema UsbPort
   // Inicia a classe
   fun init() ...
   // Retorna true se o bit tiver o valor lógico '1'
   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 lógico '1'
   fun setBits(mask: Int) ...
   // Coloca os bits representados por mask no valor lógico '0'
   fun clrBits(mask: Int) ...
}
```

2.5.2 KBD

```
object KBD {   // Ler teclas. Métodos retornam '0'..'9','#','*' ou NONE.
   const val NONE = 0;
   // Inicia a classe
   fun init() ...
   // Retorna de imediato a tecla premida ou NONE se não há tecla premida.
```

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

2.5.3 LCD

```
object LCD {   // Escreve no LCD usando a interface a 8 bits.
    private const val LINES = 2, COLS = 16 // Dimensão do display.
    private const val SERIAL_INTERFACE = false // Define se a interface é Série ou Paralela
    // Escreve um byte de comando/dados no LCD em paralelo
    private fun writeByteParallel(rs: Boolean, data: Int) ...
    // Escreve um byte de comando/dados no LCD em série
    private fun writeByteSerial(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 sequência de iniciação para comunicação a 8 bits.
    fun init() ...
    // Escreve um caráter na posição corrente.
    fun write(c: Char) ...
    // Escreve uma string na posição 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 ecrã e posicionar o cursor em (0,0)
    fun clear() ...
}
```

2.5.4 SerialEmitter

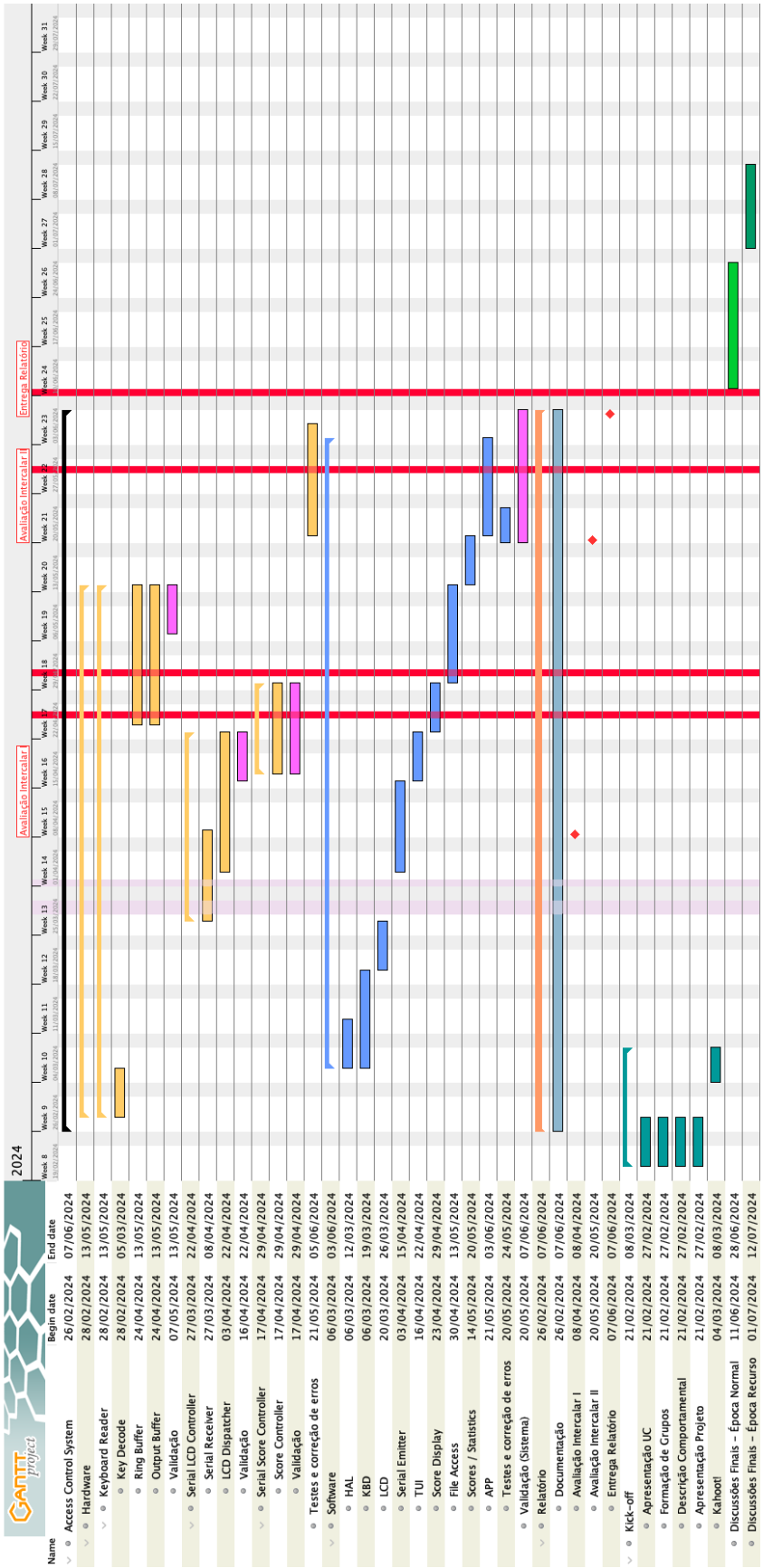
```
object SerialEmitter {        // Envia tramas para os diferentes módulos Serial Receiver.
    enum class Destination {LCD, SCORE}
    // Inicia a classe
    fun init() ...
    // Envia uma trama para o SerialReceiver identificado o destino em addr,os bits de dados em
    'data' e em size o número de bits a enviar.
    fun send(addr: Destination, data: Int, size : Int) ...
}
```

2.5.5 ScoreDisplay

```
object ScoreDisplay {     // Controla o mostrador de pontuação.
    // Inicia a classe, estabelecendo os valores iniciais.
    fun init() ...
    // Envia comando para atualizar o valor do mostrador de pontuação
    fun setScore(value: Int) ...
    // Envia comando para desativar/ativar a visualização do mostrador de pontuação
    fun off(value: Boolean) ...

}
```

3 Calendarização do projeto



Name	Begin date	End date
Access Control System	26/02/2024	07/06/2024
Hardware	28/02/2024	13/05/2024
Keyboard Reader	28/02/2024	13/05/2024
Key Decode	28/02/2024	05/03/2024
Ring Buffer	24/04/2024	13/05/2024
Output Buffer	24/04/2024	13/05/2024
Validação	07/05/2024	13/05/2024
Serial LCD Controller	27/03/2024	22/04/2024
Serial Receiver	27/03/2024	08/04/2024
LCD Dispatcher	03/04/2024	22/04/2024
Validação	16/04/2024	22/04/2024
Serial Score Controller	17/04/2024	29/04/2024
Score Controller	17/04/2024	29/04/2024
Validação	17/04/2024	29/04/2024
Testes e correção de erros	21/05/2024	05/06/2024
Software	06/03/2024	03/06/2024
HAL	06/03/2024	12/03/2024
KBD	06/03/2024	19/03/2024
LCD	20/03/2024	26/03/2024
Serial Emitter	03/04/2024	15/04/2024
TUI	16/04/2024	22/04/2024
Score Display	23/04/2024	29/04/2024
File Access	30/04/2024	13/05/2024
Scores / Statistics	14/05/2024	20/05/2024
APP	21/05/2024	03/06/2024
Testes e correção de erros	20/05/2024	24/05/2024
Validação (Sistema)	20/05/2024	07/06/2024
Relatório	26/02/2024	07/06/2024
Documentação	26/02/2024	07/06/2024
Avaliação Intercalar I	08/04/2024	08/04/2024
Avaliação Intercalar II	20/05/2024	20/05/2024
Entrega Relatório	07/06/2024	07/06/2024
Kick-off	21/02/2024	08/03/2024
Apresentação UC	21/02/2024	27/02/2024
Formação de Grupos	21/02/2024	27/02/2024
Descrição Comportamental	21/02/2024	27/02/2024
Apresentação Projeto	21/02/2024	27/02/2024
Kahoot!	04/03/2024	08/03/2024
Discussões Finais – Época Normal	11/06/2024	28/06/2024
Discussões Finais – Época Recurso	01/07/2024	12/07/2024

Figura 15 – Diagrama de *Gantt* relativo à calendarização do projeto