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

Licenciatura em Engenharia Informática e de Computadores

Jogo Invasores Espaciais
(*Space Invaders Game*)

Projeto
de
Laboratório de Informática e Computadores
2019 / 2020 verão

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 gerador de sons (*Sound Generator*) 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.

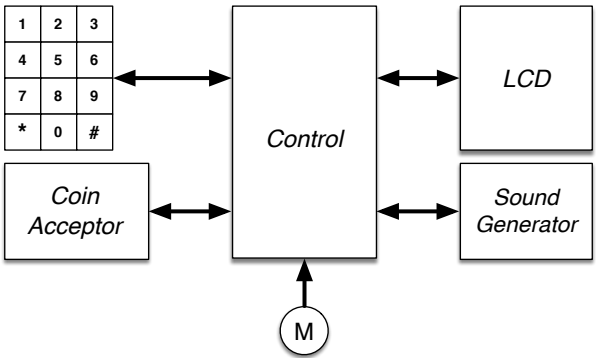


Figura 1 – Diagrama de blocos do jogo Invasores Espaciais (*Space Invaders Game*)

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*. 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 gerador de sons (*Sound Generator*), designado por *Serial Sound 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 Java.

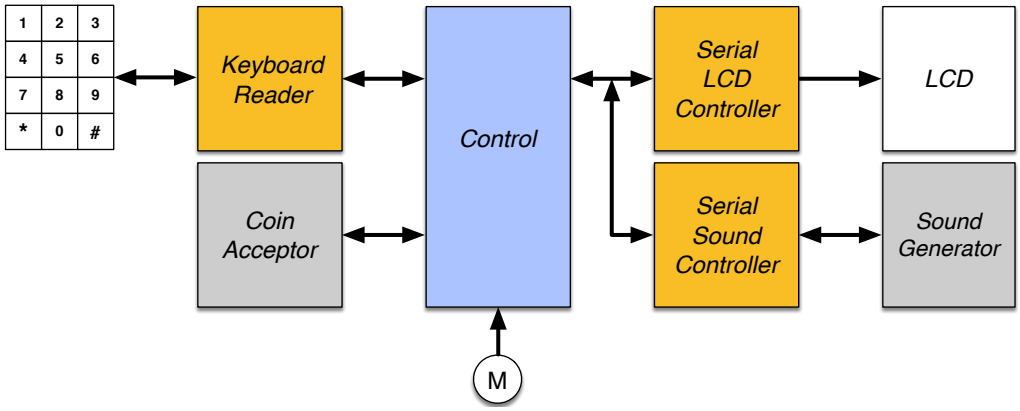


Figura 2 – Arquitetura do sistema que implementa o jogo Invasores Espaciais (*Space Invaders Game*)

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 dois códigos. O módulo *Control* processa os dados e envia a informação a apresentar no *LCD* através do módulo *SLCDC*. O gerador de sons é atuado pelo módulo *Control*, através do módulo *SSC*. Por razões de ordem física, e por forma a minimizar o número de fios de interligação, 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 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.

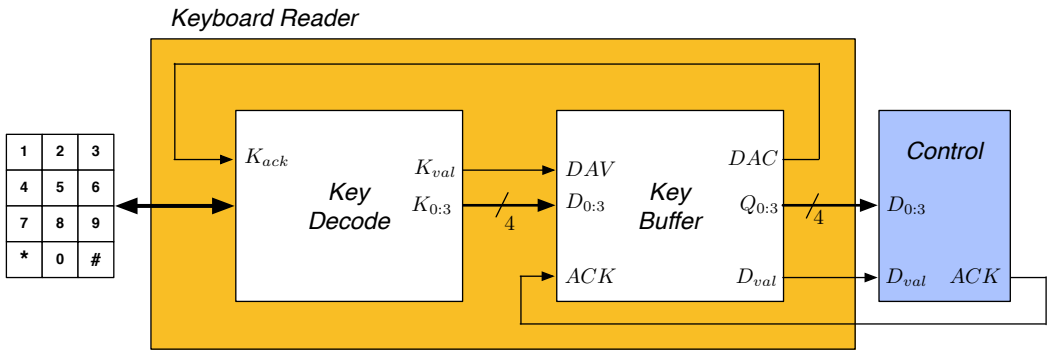
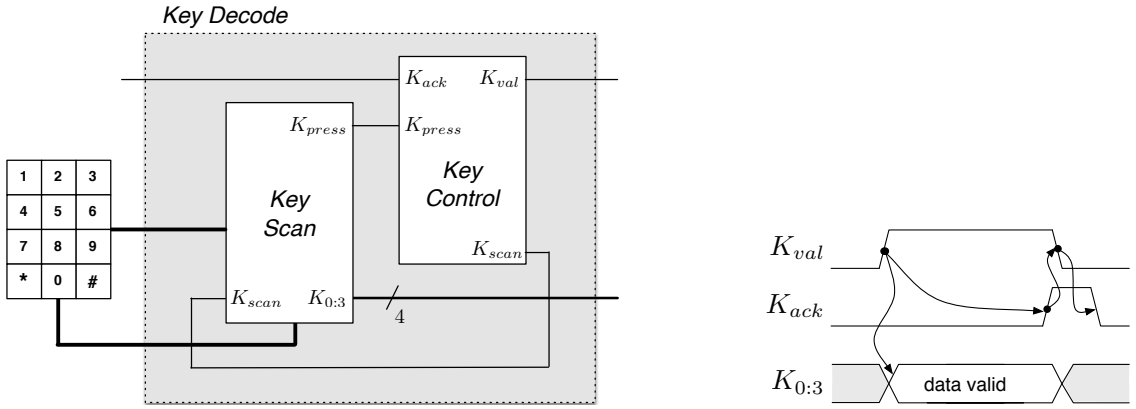


Figura 3 – Diagrama de blocos do módulo *Keyboard Reader*

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.

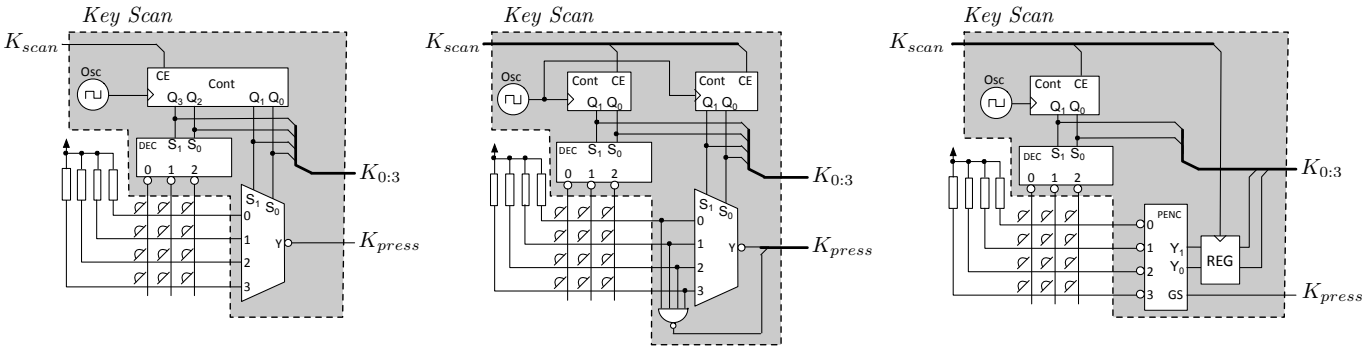
O controlo de fluxo de saída do bloco *Key Decode* (para o módulo *Key Buffer*), define que o sinal K_{val} é ativado quando é detectada a pressão de uma tecla, sendo também disponibilizando o código dessa tecla no barramento $K_{0:3}$. Apenas é iniciado um novo ciclo de varrimento ao teclado quando 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 para serem armazenados. Logo que tenha disponibilidade para armazenar informação, o bloco *Key Buffer* regista 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 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*).

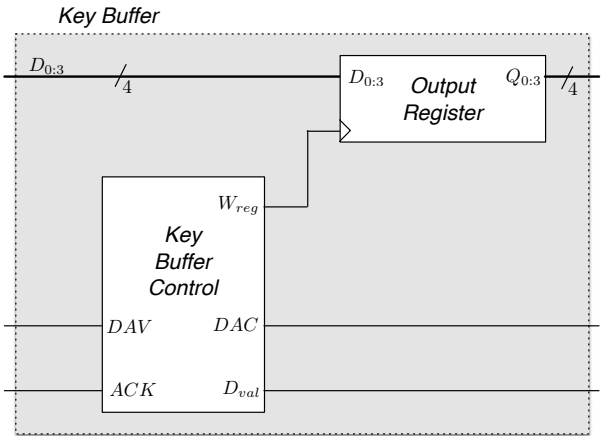
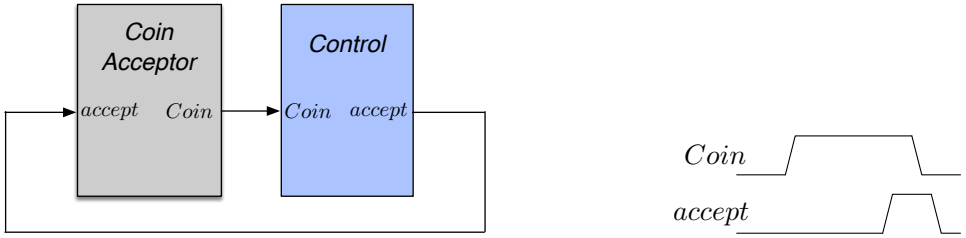


Figura 6 – Diagrama de blocos do bloco *Key Buffer*

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 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* 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 7.



a) Diagrama de blocos b) Diagrama temporal

Figura 7 – 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 8.

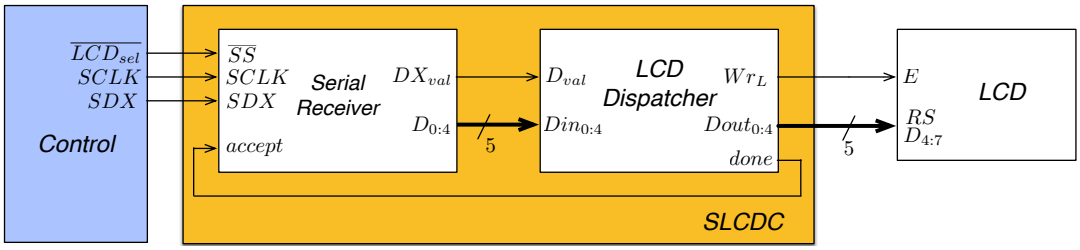


Figura 8 – Diagrama de blocos do módulo *Serial LCD Controller*

O módulo *SLCDC* recebe em série uma mensagem constituída por cinco bits de informação e um bit de paridade. A comunicação com este módulo realiza-se segundo o protocolo ilustrado na Figura 9, em que 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*. O último bit contém a informação de paridade ímpar, utilizada para detetar erros de transmissão.

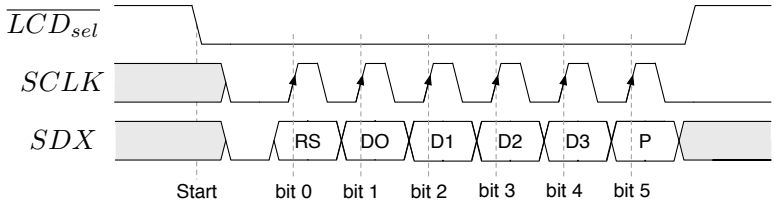


Figura 9 – Protocolo de comunicação com o módulo *Serial LCD Controller*

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 10.

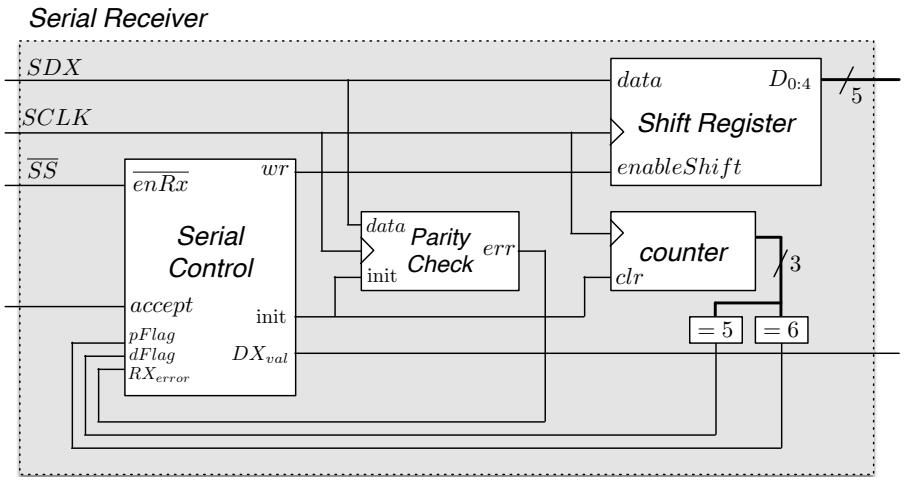


Figura 10 – Diagrama de blocos do bloco *Serial Receiver*

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 Sound Controller

O módulo *Serial Sound Controller* (*SSC*) implementa a interface com o gerador de sons, realizando a receção em série da informação enviada pelo módulo de controlo e entregando-a posteriormente ao gerador de sons, conforme representado na Figura 11.

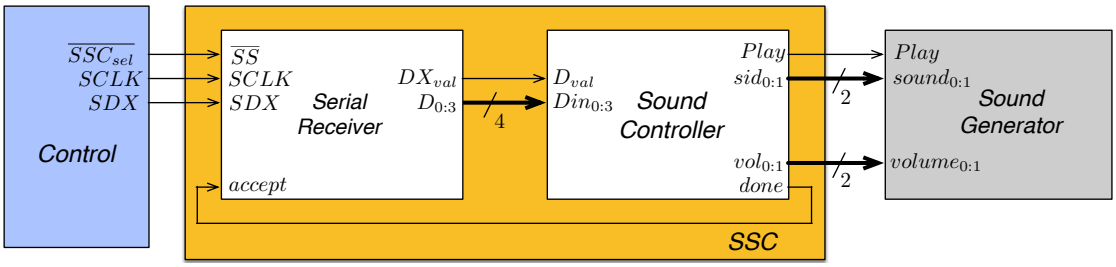


Figura 11 – Diagrama de blocos do módulo *Serial Sound Generator Controller*

O módulo *SSC* recebe em série uma mensagem composta por quatro bits de informação e um bit de paridade ímpar, segundo o protocolo de comunicação ilustrado na Figura 12. Os dois primeiros bits de informação, indicam o comando a realizar no gerador de sons, segundo a Tabela 1. Os restantes dois bits identificam o som a reproduzir ou o valor do volume. Tal como acontece com o *SLCDC*, o canal de receção série pode ser libertado após a receção da trama recebida pelo *Sound Generator*, não sendo necessário esperar pela sua execução do comando correspondente. Assim, o bloco *Sound Controller* pode ativar, prontamente, o sinal *done* para informar o bloco *Serial Receiver* que a trama já foi processada.

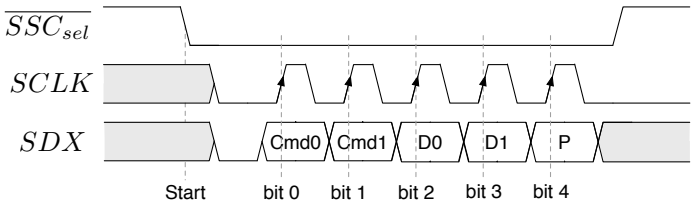


Figura 12 – Protocolo de comunicação do módulo *Serial Sound Controller*

Cmd 1 0	*data* 1 0	*Function*
0 0	* *	*stop*
0 1	* *	*play*
1 0	$s_1\ s_0$	*set sound*
1 1	$v_1\ v_0$	*set volume*

Tabela 1 – Comandos do módulo *Sound Generator*

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 quatro bits de informação em vez de cinco bits.

2.4.2 Sound Controller

Após a receção de uma trama válida (proveniente do bloco *Serial Receiver*), o bloco *Sound Controller*, deverá proceder à atuação do comando recebido sobre o gerador de sons. Salienta-se que para reproduzir um som, o gerador de sons necessita de ter presente nas entradas *volume* e *sound* o volume e o identificador do som, respetivamente.

2.5 Control

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

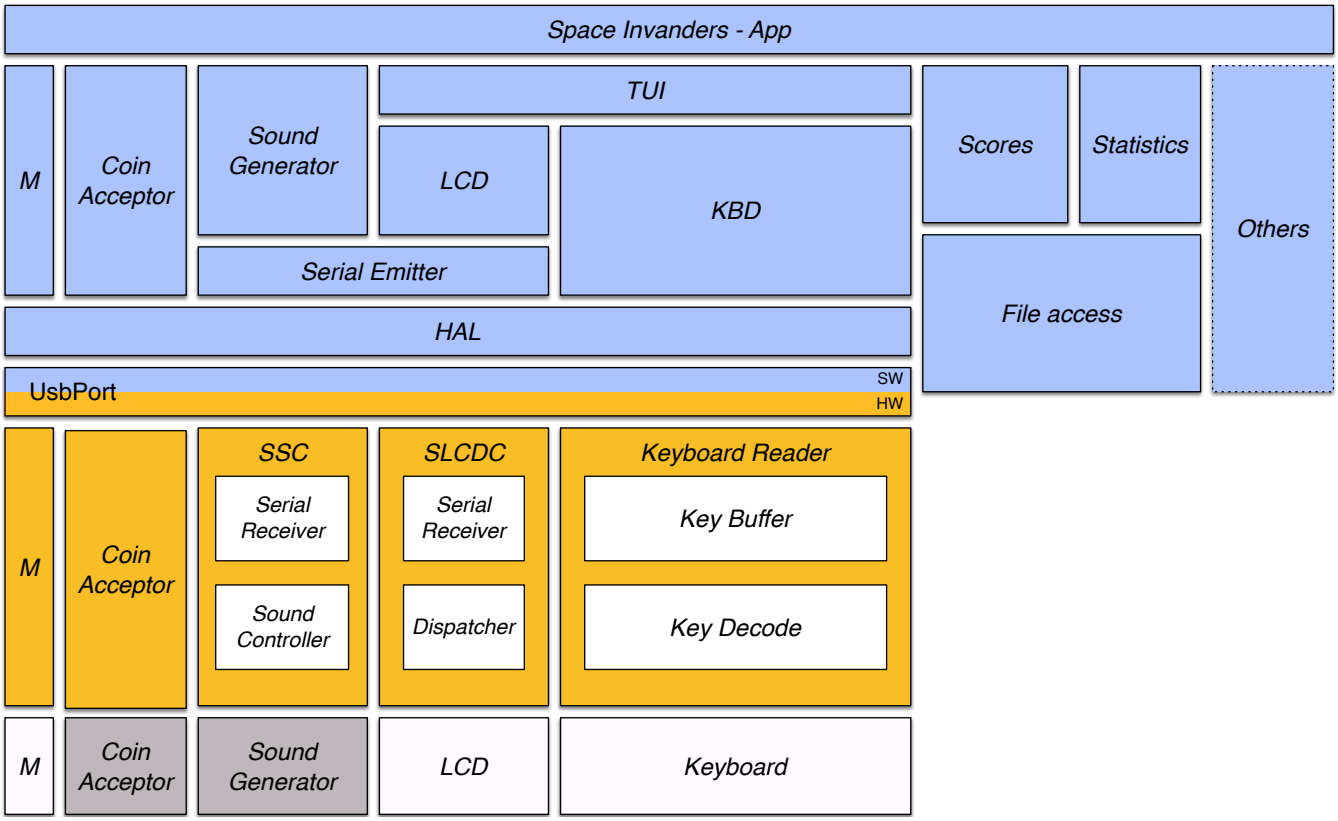


Figura 13 – Diagrama lógico do Jogo Invasores Espaciais (*Space Invaders Game*)

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

2.5.1 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) ...
}
```

2.5.2 Classe KBD

```
public class KBD {           // Ler teclas. Métodos retornam '0'..'9','A'..'F' 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 for premida ou NONE após decorrido 'timeout' milisegundos.
   public static char waitKey(long timeout) ...
}
```

2.5.3 Classe LCD

```
public class LCD {    // Escreve no LCD usando a interface a 4 bits.
   public static final int LINES = 2, COLS = 16; // Dimensão do display.
   // Define se a interface com o LCD é série ou paralela
   private static final boolean SERIAL_INTERFACE = false;
   // Escreve um nibble de comando/dados no LCD em paralelo
   private static void writeNibbleParallel(boolean rs, int data) ...
   // Escreve um nibble de comando/dados no LCD em série
   private static void writeNibbleSerial(boolean rs, int data) ...
   // 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) ...
   // Envia comando para limpar o ecrã e posicionar o cursor em (0,0)
   public static void clear() ...
}
```

2.5.4 Classe SerialEmitter

```
public class SerialEmitter { // Envia tramas para os diferentes módulos Serial Receiver.
   public static enum Destination {SLCD,SSC};
   private static int DESTINATION_MASKS[] = {SLCDC_SELECT_MASK, SSC_SELECT_MASK};
   // Inicia a classe
   public static void init() ...
   // Envia uma trama para o SerialReceiver identificado por addr, com a dimensão de size e os
   bits de 'data'.
   public static void send(Destination addr, int size, int data) ...
}
```

2.5.5 Classe SoundGenerator

```
public class SoundGenerator {       // Controla o Sound Generator.
   // Inicia a classe, estabelecendo os valores iniciais.
   public static void init() ...
   // Envia comando para reproduzir um som, com a identificação deste
   public static void play(int sound) ...
   // Envia comando para parar o som
   public static void stop() ...
   // Envia comando para definir o volume do som
   public static void setVolume(int volume) ...
}
```

3 Calendarização do projeto

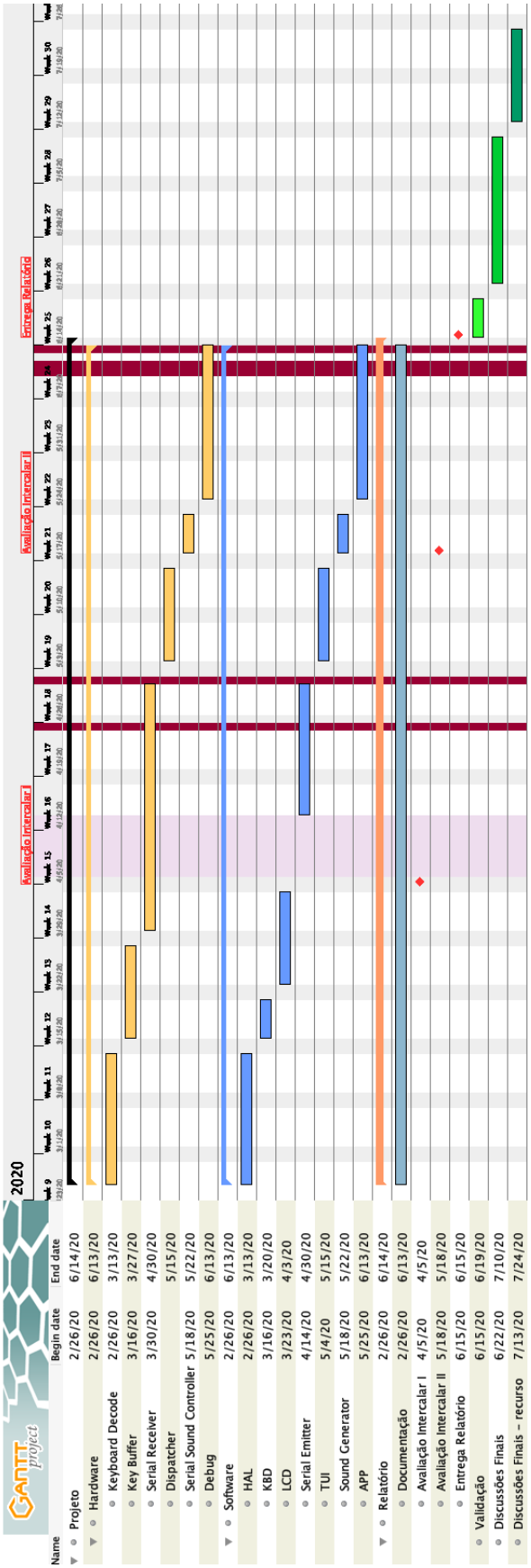


Figura 14 – Diagrama de *Gantt* relativo à calendarização do projeto