Projeto final EA871 - Genius

Thiago M. Pavão – 247381, Vinícius Esperança Mantovani – 247395

1 Introdução

A proposta de projeto é implementar o jogo *Simon*, conhecido pelo produto físico da Figura 1, outro nome do jogo é *Genius* é este que adotamos para o nosso. O objetivo do jogo é acertar a sequência de cores mostradas. A sequência incrementa seu tamanho em um quando o jogador consegue inserí-la com sucesso, começando em 1 de tamanho e aumentando até que a memória do jogador falhe.

Figura 1: Jogo Simon, figura retirada de [**1**]

O jogo conhecido tem por padrão 4 botões/cores, que podem acender e formar a sequêcia, nossa ideia é fornecer a opção para o usuário de selecionar o mapa de jogo, podendo selecionar entre o clássico (4 posições), 6 posições ou até mesmo 9, para os mais ousados. A escolha do mapa é feita logo antes do começo da partida.

2 Periféricos e lógica de controle

2.1 Sensor IR - TPM

A interação do usuário com o jogo é feita via controle remoto de televisão, o sinal é detectado com o sensor da Figura 2, com o nosso controle, descobrimos que o sinal segue o protocolo de transmissão NEC [**2**], que se baseia na duração de uma sequencia de pulsos de luz infra-vermelha e do tempo entre os pulsos para transmitir uma mensagem de forma sequencial. Cada transmissão leva em média 67.5 ms.

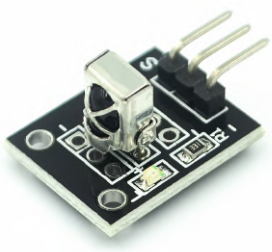

Figura 2: Sensor receptor de infra-vermelho, utilizado para ler o sinal do controle remoto

Descobrimos o protocolo de comunicação do nosso controle remoto conectando o pino de sinal do sensor IR no analisador lógico e pressionando um dos botões do controle, a forma de onda observada foi a mostrada na Figura 3. O sinal funciona da seguinte forma, como lido na descrição do protocolo NEC: O controle envia pulsos de infra-vermelho (IR) de somente duas durações, 9 ms e 562.5 μs. Para indicar o início de um sinal o controle envia um pulso de 9 ms e após 4.5 ms de espera se inicia a transmissão dos dados. Sempre que há sinal do controle a saída vai para nível lógico baixo (LOW) e quando não há sinal ele volta para alto, isto é visível na Figura 3.

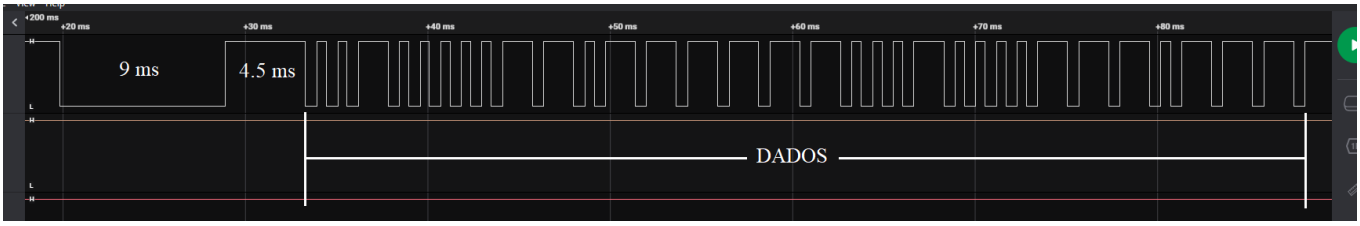

Figura 3: Sinal do sensor, lido com o analisador lógico ao pressionar um dos botões dos controle remoto.

A transferência dos dados é feita apenas com o pulso de menor duração, 562.5 μs. O que indica se um bit é 0 ou 1 é o tempo até que ocorra o próximo pulso, se o intervalo até a próxima recepção for também 562.5 μs, o bit é um 0, caso seja 1.6875 ms o bit é 1. Isto é ilustrado na Figura 4, lembrando que há recepção de sinal IR do controle remoto quando o sinal é baixo.

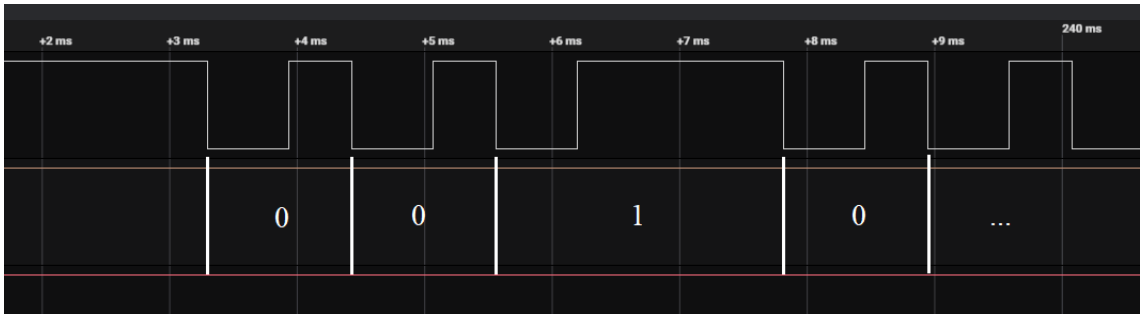

Figura 4: Ilustração da diferenciação de um bit 0 e um bit 1 no sinal do sensor IR.

Ao todo, são transmitidos 32 bits, formando 4 bytes na sequência $A\overline{A}B\overline{B}$. No total, a informação real transmitida é de 16 bits (os bytes AB) e os bytes invertidos servem para possibilitar a detecção de erros na transmissão, ao fim dela é possível checar se $A + \overline{A} = B + \overline{B} = 0xFF$, para detectar possíveis erros na transmissão. Isto define qual o sinal que deve ser decodificado, mas como fazer isso?

Foi utilizado um canal do módulo TPM em *Input Capture* sensível à qualquer borda (subida ou descida). Para ler um sinal com esta configuração basta utilizar o TPM para capturar a duração de cada pulso e intervalo entre pulsos enviado pelo controle remoto e comparar com os valores esperados, por exemplo a sequencia 9 ms, 4.5 ms, 562.5 s, ... seria considerada válida e os dois bytes de informação real seriam extraídos. A configuração do TPM foi feita com o menor período possível (melhor resolução) que fosse maior do que o maior tempo que se deseja ler no sinal (9 ms), para que fosse possível ler qualquer duração sem se preucupar com mais de um overflow do TPM, para satisfazer esta condição, configuramos o TPM com a função *TPM_config_especifica* para um período de aproximadamente 12.5 ms. No reconhecimento das durações de tempo, consideramos uma margem de 150 μs, já que certamente nenhum tempo será reconhecido de forma exata.

Para implementar este funcionamento foi utilizada uma estrutura na máquina de estados que se repete sempre que há a necessidade de receber entrada do usuário pelo controle remoto, os estados são *ESPERA*, *LEITURA* e *INTERPRETA*. No primeiro deles apenas espera-se alguma borda no sensor IR, nada é executado até que isto aconteça. A partir da primeira borda (detectada por interrupção), espera-se que várias bordas ocorram (vide Figura 3) então troca-se para o estado *LEITURA*, neste estado cada borda no sensor causa uma interrupção que salva o número de clocks do TPM desde a última borda, comparando-se o valor do campo *VAL* do canal na primeira e segunda borda. Estes valores são salvos em um buffer circular, a partir deles é possível obter as durações de cada pulso e intervalo entre pulsos apenas dividindo-os pelo valor máximo e multiplicando pelo período configurado.

Neste mesmo estado, enquanto as durações são inseridas no buffer circular sempre que ocorre uma interupção, o fluxo normal de programa chama a função de decodificação: *IR_Leitura*, que retira os valores do buffer e compara com os valores esperados, decodificando a mensagem. Se em algum momento um tamanho de pulso inesperado é detectado ou ao fim da leitura a verificação de soma não resultar em $0xff$ a função retorna um código de erro. Com o fim da execução da função, em caso de erro o estado retorna *ESPERA*, sem passar por algo que avise o usuário, já que é possível que o sinal lido não fosse sequer do controle remoto, mas alguma outra fonte intrusa. Se não ocorreu erro, o estado segue para *INTERPRETA*, onde o valor de 16 bits pode ser lido e comparado com valores previamente adquiridos para descobrir qual foi o botão pressinado e realizar a ação correspondente em cada caso.

Por fim, vale ressaltar dois pontos: Na transição de *ESPERA* para *LEITURA*, o buffer circular é reiniciado, com a chamada de *BC_reset*, também nesta transição é ativada a interrupção por overflow do TPM, isto é feito para que o sistema identifique que o sinal foi problemático sem que ocorra um pulso de duração inesperada (que já é um problema detectado), mas em uma situação em que o sinal estava correto, ou seja, a sequência de duração dos pulsos era compatível até o momemnto, e foi derrepente interrompido. O código no fluxo normal de execução apenas espera que tenham items no buffer circular e trata-os um de cada vez, para detectar portanto que o sinal foi interrompido é necessária uma lógica separada. Isto foi feito com uma flag chamada *seguranca* o sistema escreve 1 nela sempre que ocorre overflow, e 0 sempre que uma borda no sensor é detectada. Antes de escrever 1 no caso de overflow verifica-se se a flag já estava configurada, pois neste caso é possível concluir que houveram dois overflows desde a última borda no sensor e portanto o pulso atual teria pelo menos a duração igual ao período do TPM, de 12.5 ms. Como não existem pulsos deste tamanho previstos no protocolo, quando este tipo de situação é detectada insere-se o valor $0xff$ no buffer circular, este valor será então em algum momento lido pela função de decodificação, que acusará o erro, causando na transição para o estado de *ESPERA*.

Esta última situação parece improvável de acontecer mas é importante ter sistemas de prevenção, caso elas venham a surgir. A lógica descrita e a flag *seguranca* se encontram em *ISR.c*.

2.2 Matriz de LEDs - SPI

A visualização do jogo será feita por uma matriz de 8x8 LEDs (Figura 5), controlada por MAX7219. A comunicação com o periférico é feita por intermédio do módulo SPI da placa que utilizamos no laboratório. Esta comunicação é serial e síncrona, a comunicação com MAX7219 resulta em escrita nos seus registradores internos, os dados são enviados em pacotes de 16 bits, contendo o endereço do registrador e o valor que se deseja escrever nele. Inicialmente deve ser feita uma incialização, escrevendo em determinados registradores, e posteriormente, para controlar os LEDs basta escrever em 8 registradores, cada um controlando uma linha da matriz de LEDS. A especificação da comunicação em detalhes será feita seguindo o datasheet do MAX7219 [**3**].

Para entendermos o funcionamento do módulo SPI, inicialmente, estudamos o manual de referência da placa e, em seguida, procuramos entender o funcionamento de cada registrador do módulo e seu funcionamento. Desse modo, pudemos introduzir a lógica de funcionamento, controle e inicialização do módulo em nosso código. Para tanto, tentamos, de inicio, acender a matriz de LEDs completamente. Fizemos isso inicializando os pinos PTE1, PTE2 e PTE4 como, respectivamente, SPI1_MOSI, SPI1_SCK, SPI1_PCS0. Além disso, inicializamos o SPI por meio dos registradores SPI1_C1, SPI1_C2 e, SPI1_BR. O primeiro é responsável por habilitar o módulo, configurá-lo como mestre, setar a polaridade de seu clock como ativo-alto e, em conjunto com o campo MODFEN de SPI1_C2, SPI1_C1 configura ainda o funcionamento do pino SS/ como slave select input. Já o registrador SPI1_C2 é usado para seleção do modo slave select output, determinar o uso de I/O de SPI como output e, garantir que o clock do módulo pare quando MCU entra no modo de espera.

Após essa inicialização, trabalhamos para a ativação de todos os LEDs da matriz, para testá-la. Seguindo o datasheet, escrevemos o valor 1 no endereço 0xX9. Ou seja, enviamos, por meio do registrador SPI1_D, os 8 bits referentes ao endereço 0xX9 e, em seguida, os 8 bits referentes ao dado, 0x1.

Então, logo que conseguimos inicializar corretamente o módulo SPI e testar a matriz de LEDs, partimos para a inicialização do LED e a generalização da escrita em registradores da matriz. Para tal inicialização, precisamos escrever 8 bits de endereço de registrador seguido dos 8 bits de dados a serem passados para o registrador. Para tanto, criamos uma função que escreve 16 bits, dados dois parâmetros de 8 bits, referentes ao endereço do registrador e aos dados a serem passados. Assim, definimos o brilho, o modo de operação e outras configurações importantes para o funcionamento da matriz, usando a função citada anteriormente.

Ademais, para controle da matriz, escrevemos funções usadas para manipulação de mais alto nível do display.

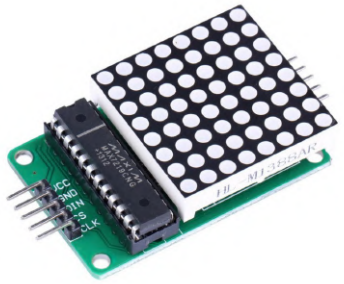

Figura 5: Matriz de LEDs, controlada por MAX7219

2.3 Buzzer - TPM

Uma parte importante do jogo é o som, emitido sempre que uma posição é acesa, tanto quando a sequência está sendo apresentada, como quando o usuário está inserindo. Para isto, será utilizado um buzzer passivo, não identico à mas do mesmo funcionamento que o mostrado na Figura 6

Para configurarmos o Buzzer, determinamos funções que levam em conta a frequência com que queremos o som ou a posição da matriz cuja frequência correspondente será soada. Para isso, utilizamos o TPM no modo de operação PWM, de forma que, na função dependente da frequência passada de parâmetro, o TPM é configurado com máximo do contador igual ao clock dividido pela frequência. Enquanto que o canal configurado com modo PWM, tem valor de CnV configurado como metade do valor definido como máximo do contador do TPM. Assim, os pulsos de PWM foram usados como pulsos para acionamento do buzzer, definindo a frequência do som de acordo com a frequência dos pulsos, determinada inicialmente pelo parâmetro passado para a função feita.

Além disso, fizemos funções de desligar o buzzer e, algumas outras para tocar músicas para início, acerto de sequência, erro de sequência...

Vale notar que o TPM utilizado para este controle é diferente do utilizado para o sensor IR, já que são aplicações bem diferentes que não poderiam se misturar.

Figura 6: Buzzer análogo ao que será utilizado no projeto.

2.4 PIT

Usamos, também, o módulo PIT, inicializado com período igual a aproximadamente 0,9 segundos. Usamos o mesmo período usado em roteiro de atividade anterior. Isso porque, embora pudessemos projetar o período de acordo com LDVAL e OUTDIV4, o tempo não deveria ter tanto rigor e teria de ser testado, logo, começamos a testar com o valor dado e, concluimos que o intervalo seria bom para aquilo que se presta.

Esse módulo foi usado para alternar entre o estado ESPERA INICIO e o estado ATUALIZA INICIO, de forma que, a cada overflow no PIT (de 0,9 em 0,9 segundos), ocorre a transição do estdo ESPERA INICIO para o ATUALIZA INICIO, no qual a letra mostrada no display é alterada para a próxima no vetor de char (string) "GENIUS "(nome do jogo mais um espaço). O caractér espaço no vetor é utilizado para manter a matriz sem LEDs acesos durante um tempo entre uma escrita da palavra GENIUS e outra.

3 Máquina de Estados

A máquina de estados geral da aplicação consiste de 5 macro estados: *APRESENTACAO*: Música e letras no display, sequência de inicialização e apresentação do jogo. *INÍCIO*: Aplicação em stand-by até que ocorra interaçãco do usuário, periódicamente a letra na tela muda, formando a palavra "GENIUS" em loop. *SELEÇÃO DE MAPA*: Seleção do mapa (dificuldade) do jogo, dentre as três opções: 4, 6 e 9. *JOGO*: Jogatina em si, em que a sequência é mostrada e deve ser repetida pelo jogador. *RESULTADO*: Mostra para o jogador o número máximo de posições memorizadas por ele, ou seja, o número de posições da última sequência que foi inserida corretamente por completo. Estes estados podem ser vistos na Figura 7.

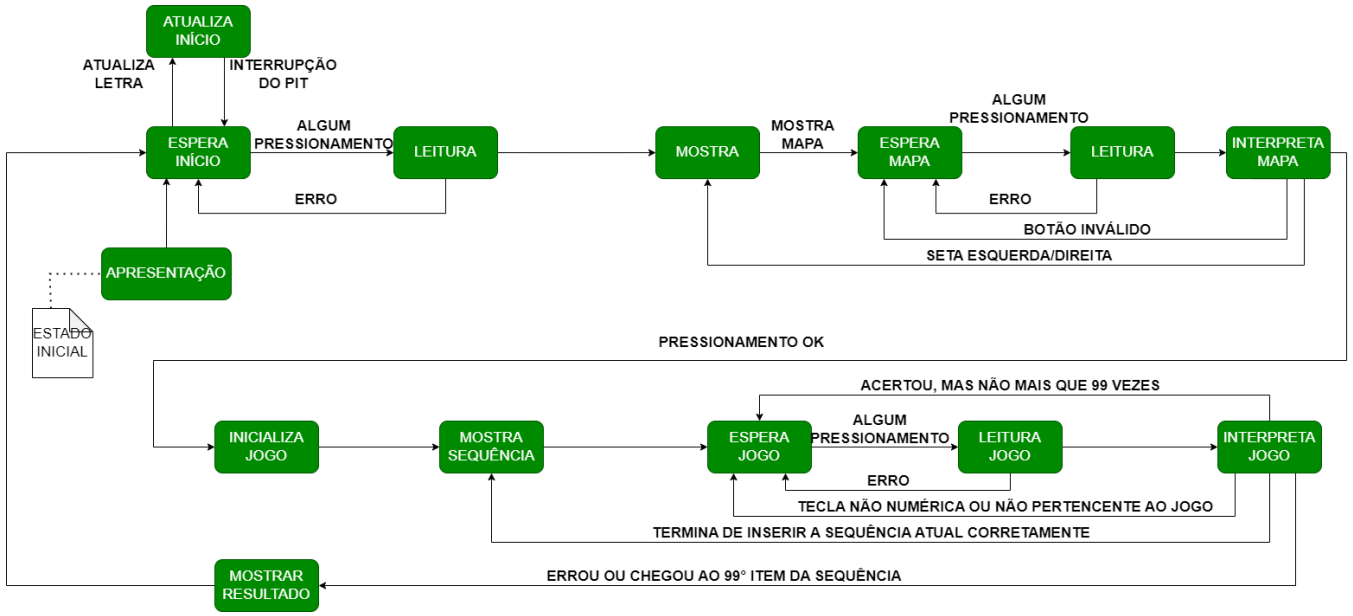

Figura 7: Máquina de estados completa, com os micro estados.

Estes macro-estados não são estados reais da máquina de estados da aplicação, apenas abstrações de blocos maiores da execução.

3.1 *APRESENTAÇÃO*

Este estado é responsável por exibir na tela o nome do jogo: *GENIUS*, como se o jogo estivesse se apresentando ao jogador. Enquanto o nome é exibido um trecho de uma música toca no buzzer. Este é o estado inicial da aplicação e só ocorre quando o sistema inicializa. Em seguida temos a tela de INÍCIO

3.2 *INÍCIO*

Este macro estado é responsável por mostrar na tela o nome do jogo em loop, isto é feito na *main*, piscando letra por letra na matriz de LEDs sequencialmente. Isto continua até que um pressionamento no controle remoto seja reconhecido, como explicado na seção 2.1 isto é feito com uma estrutura padrão de estados para o reconhecimento, aqui não foi necessário o estado de *INTERPETAÇÃO*, pois o pressionamento de qualquer tecla faz com que o estado siga para a seleção de mapa.

Temos portanto os estados *ESPERA_INICIO* e *LEITURA_INICIO*, mas mais um é necessário. Como deseja-se atualizar a tela periódicamente criamos o estado *ATUALIZA_INICIO*, que é responsável por atualizar na tela a letra que está aparecendo, e em seguida voltar para *ESPERA_INICIO*. O chaveamento para este novo estado é feito sempre que ocorre uma interrupção de PIT, que é ativado antes de entrar

no estado de *ESPERA* pela primeira vez, (vindo de *APRESENTACAO* ou de *RESULTADO*) e desativado quando não é mais preciso, no chaveamento para seleção de mapa.

3.3 *SELEÇÃO DE MAPA*

Aqui, o usuário deve escolher o mapa em que deseja jogar, na realiadade, o mapa será sempre uma grade 3 por 3, com 9 espaços, mas a seleção permite que o usuário escolha quais quadrados podem acender, e assim fazer parte da sequência. Cada quadrado do mapa é associado à um número, correspondendo à um numero no teclado numérico do controle remoto, conforme especificado pela Figura 8, portanto cada opção de mapa consiste em um vetor com alguns números, contendo as posições válidas para aquela opção. Isto é utilizado para desenhar o mapa no momento da seleção, enquanto alguma variável armazena o mapa atualmente selecionado.

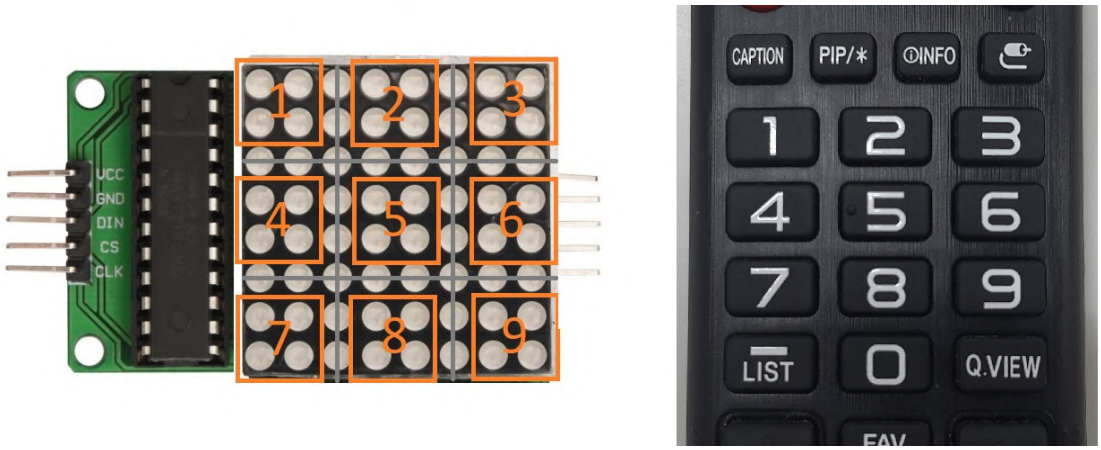

Figura 8: Identificação das posições do tabuleiro.

O usuário pode trocar o mapa selecionado utilizando as setas do controle remoto, quando satisfeito com a decisão, basta pressionar o botão OK para confirmar e começar a jogar. Este funcionamento em geral envolve os estados: *MOSTRA_MAPA*, *ESPERA_MAPA*, *LEITURA_MAPA* e *INTERPRETA_MAPA*, esses estados e suas conexões podem ser vistas na Figura 7.

O estado *MOSTRA* é responsável por desenhar na matriz de LED o mapa selecionado, assim que isto é feito, o estado muda para *ESPERA*, e o processo de leitura padrão é feito. No estado *INTERPRETA*, a tecla é processada e a ação é feita de acordo:

- Se a tecla da esquerda ou direita for pressionada, a variável que armazena o índice do mapa selecionado é atualizada e o estado volta para *MOSTRA*, para atualizar a tela.
- No caso de uma tecla *OK*, o usuário indica que selecionou o mapa e o fluxo da máquina segue para o *JOGO*, mais especificamente, para o estado *INICIALIZA_JOGO*.
- Se qualquer outra tecla foi apertada, o buzzer emite um som de erro, para indicar que a tecla pressionada não tem ação e a máquina segue para o estado de *ESPERA*, para que o usuário possa tentar novamente.

3.4 *JOGO*

Algumas informações são armazenadas para controlar o jogo: a sequência é armazenada em um vetor, sendo que ela será formada por números indicando as posições seguindo a numeração dada na Figura 8, o tamanho atual da sequência é salvo em outra variável e a posição do usuário (para o momento em que ele estiver inserindo a sequência) em outra.

Primeiramente, é feita a inicialização do jogo, que consiste em gerar a primeira posição no vetor da sequência e registrar que o tamanho atual é 1. Isto é feito apenas uma vez por jogo. As posições são geradas da seguinte forma: Existe um vetor que armazena o número de casas que podem acender em cada mapa, e outro que armazena quais são essas posições, para cada mapa. Portanto, para gerar uma posição válida basta ler o valor do contador TPM (que garente aleatóriedade devido à alta interação com o usuário),

em seguida pega-se o resto da divisão deste valor pela quantidade de posições válidas do mapa que foi selecionado, gerando então um valor que pode ser utilizado como índice no vetor de posições válidas do mapa selecionado. A lógica é feita de forma genérica suficiente para funcionar de forma equivalente para qualquer um dos mapas selecionados, sem a necessidade de verificação manual.

Após a inicialização o jogo segue para o estado *MOSTRA_SEQUENCIA*, este estado é responsável por mostrar para o usuário toda a sequência armazenada, piscando e ativando o buzzer para cada uma delas em sequência. Cada uma fica acesa por meio segundo e o intervalo entre elas é de pouco menos que isso. Ao fim deste processo, é a vez do usuário de inserir a sequência. A váriavel que armazena a posição do usuário na sequência é reiniciada em zero (já que ele sempre insere ela desde a primeira posição novamente), e então novamente ocorre a estrutura de leitura do controle remoto na máquina de estados, o estado vai para *ESPERA_JOGO* até que algo seja detectado e quando uma tecla é identificada com sucesso a máquina transita para *INTERPRETA_JOGO*. Neste momento, temos novamente que a ação a ser realizada depende da tecla pressionada, de forma simples temos:

- Se o número correto foi pressionado, ou seja, se a posição atual do usuário na sequência corresponde com a tecla pressionada, mas esta posição ainda não é a última da sequência: A posição equivalente na matriz de LEDs acende e o buzzer toca, indicando que o pressionamento foi registrado, a variável de posição do usuário é incrementada e o estado volta para *ESPERA*, para que ele insira a próxima posição da sequência.
- Se o número correto foi pressionado e a posição era a última, um som de sucesso é tocado e mais um item é gerado na sequência, que é novamente mostrada por completo transitando-se para o estado *MOSTRA*.
- Por fim, se o número inserido não era o próximo da sequência o jogo termina, uma musica de derrota é soada pelo buzzer e o estado transita para *RESULTADO*.

Alguns casos especiais são os que se seguem. Se a tecla pressionada não for um número (por exemplo as setas ou a tecla OK), ou se for um dos números de 1 a 9 mas não for uma posição válida para o mapa selecionado, o usuário não é prejudicado, apenas é feito um som de erro com o buzzer para indicar que a tecla pressionada não faz sentido e ajudar o jogador a entender o problema. O outro problema surge do fato que o vetor que armazena a sequência é alocado estáticamente, e seu tamanho é definido pelo valor da macro *MAX_SEQUENCIA* em *main.c* que tem como valor 99. Se de alguma forma um usuário conseguir chegar neste tamanho de sequência ocorre um problema, pois não é possível continuar crescendo com ela, portanto para que não ocorram erros inesperados o estado é automaticamente transitado para *RESULTADO*, onde o jogador verá que conseguiu "vencer" o jogo, se isto foi feito sem trapaça é algo que não cabe ao jogo ou a nós decidir, mas o limite existe e o sistema está protegido.

3.5 *RESULTADO*

Este é o macro estado mais simples, ele é responsável por mostrar na tela o número de posições da última sequência inserida corretamente por completo pelo usuário, ou seja, sua pontuação no jogo. O número é mostrado digito a digito, piscando os caracteres na matriz de LEDs. Para isso, é necessário apenas um micro estado: *MOSTRA_RESULTADO*. As pontuações são sempre impressas com dois dígitos, como forma de padronização do valor e estilo.

4 Manual de usuário

4.1 Montagem

Nesta seção, explicamos as conexões de cada periférico:

1. Buzzer conectado ao ground e ao pino PTE22;
2. Matriz de LEDs conectada ao ground, ao VCC de 5V e, aos pinos PTE1 (DIN), PTE2 (CLK) e PTE4 (CS);
3. Sensor Infra conectado a ground, VCC de 3.3V e, ao pino PTE20.

Seguindo as posições amostradas na Figura 9

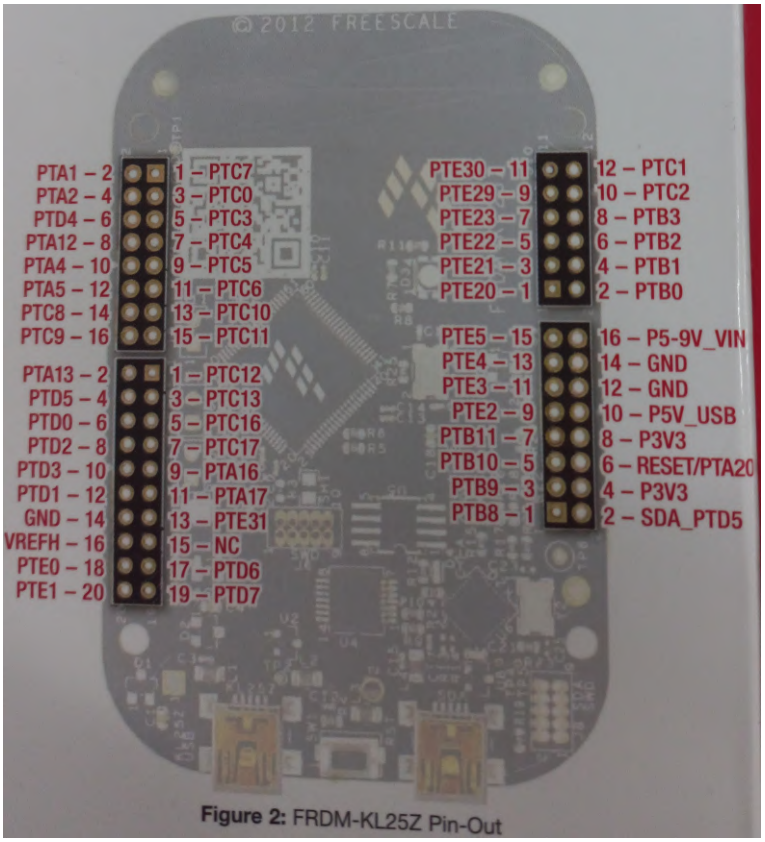

Figura 9: Portas presentes na placa

4.2 Uso

1. O usuário verá a palavra GENIUS sendo escrita letra a letra na matriz e, deverá pressionar qualquer tecla do controle para prosseguir para a escolha de mapa.

2. A primeira opção de mapa aparecerá na matriz de LEDs e, deve-se pressionar a seta para a direita ou para a esquerda para alterar o mapa (número de posições possíveis de serem acesas na sequência) ou o botão OK para confirmar o mapa mostrado. As opções são: teclas 2, 4, 6, 8 (quatro teclas); teclas 1, 2, 3, 4, 5, 6 (seis teclas); ou, para aqueles que gostarem de um desafio, teclas 1 a 9 (todas as 9 teclas numéricas do controle).

3. Agora, o usuário começará a ver as sequências sendo reproduzidas de maneira crescente, ou seja, a primeira terá apenas uma posição acesa, a segunda terá duas... Então, após a apresentação da sequência aleatória, o jogador deverá reproduzí-la por meios das teclas numéricas respectivas a cada posição na matriz.

4.3 Protótipo

Na Figura 10 é possivel ver o protótipo montado em um acrílico para fixar os componentes, e o controle remoto que foi utilizado para o desenvolvimento. Não estudamos a fundo mas teoricamente outros controles que seguem o mesmo protocolo de comunicação também devem funcionar com a aplicação. Também há um vídeo demo, no link `https://www.youtube.com/shorts/t-NTu1HTvTg` .

Figura 10: Setup do jogo, com o controle remoto de entrada, matriz de LEDs, Sensor IR, buzzer e por fim a placa controladora.

Referências

[**1**] Página da wikipédia do jogo Simon
`https://en.wikipedia.org/wiki/Simon_(game)`

[**2**] Procolo de transmissão NEC
`https://techdocs.altium.com/display/FPGA/NEC+Infrared+Transmission+Protocol`

[**3**] Datasheet do MAX7219
`https://www.analog.com/media/en/technical-documentation/data-sheets/max7219-max7221.pdf`