CAPÍTULO 7

Displays de LED

Até aqui, você trabalhou com LEDs individuais de 5 mm. LEDs também podem ser comprados em pacotes, ou como um display de matriz de pontos, sendo que a opção mais popular é uma matriz de 8 x 8 LEDs, ou de 64 LEDs no total. Você também pode obter displays de matriz de pontos bicolores (por exemplo, vermelho e verde) ou até mesmo um display de matriz de pontos RGB, capaz de reproduzir qualquer cor, e com um total de 192 LEDs em um único pacote. Neste capítulo, você trabalhará com um display de matriz de pontos usual, 8 x 8 e de cor única, e aprenderá como exibir imagens e texto. Iniciaremos com uma simples demonstração de como criar uma imagem animada em um display 8 x 8, e depois avançaremos para projetos mais complexos. Nesse processo, você aprenderá um conceito muito importante: a multiplexação.

Projeto 19 – Display de matriz de pontos LED – Animação básica

Neste projeto, você utilizará novamente dois registradores de deslocamento, que nesse caso estarão conectados às linhas e colunas do display de matriz de pontos. Depois, você reproduzirá um objeto simples, ou sprite, no display e também o animará. O objetivo principal deste projeto é mostrar-lhe como funciona um display de matriz de pontos e apresentar o conceito da multiplexação, pois essa é uma habilidade valiosíssima.

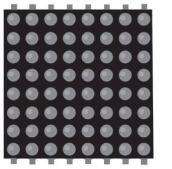
Componentes necessários

Você necessitará de dois registradores de deslocamento (74HC595) e de oito resistores limitadores de corrente. Também deverá obter um display de matriz de pontos de ânodo comum (C+), assim como suas especificações para que saiba quais pinos devem ser conectados às linhas e colunas.

2 CIs registradores de deslocamento 74HC595

8 resistores limitadores de corrente

Display de matriz de pontos 8 x 8 (C+)



Conectando os componentes

Analise o diagrama cuidadosamente. É importante que você não conecte o Arduino ao cabo USB ou à força até que o circuito esteja completo; do contrário, você pode danificar os registradores de deslocamento, ou o display de matriz de pontos. Este é um exercício de fiação complicada, por isso tenha cuidado. Certifique-se de conectar os componentes com calma e de forma metódica.

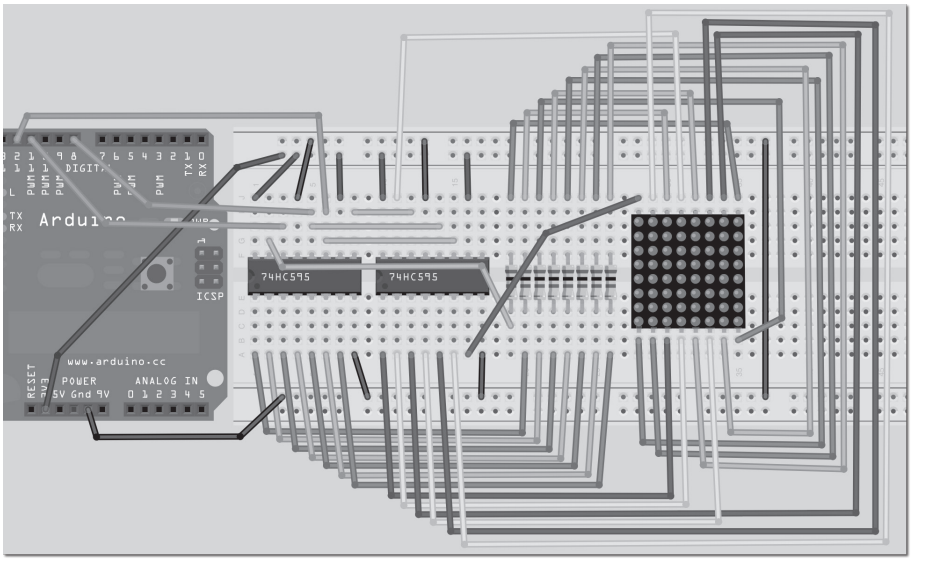


Figura 7.1 – Circuito para o Projeto 19 – Display de matriz de pontos LED – Animação básica (consulte o site da Novatec para versão colorida).

O diagrama da fiação na figura 7.1 é relevante à unidade de matriz de pontos específica utilizada na criação deste projeto, uma miniunidade 8 x 8 de display de matriz de pontos vermelhos. Entretanto, seu display pode ter (e provavelmente terá) pinos

diferentes dos utilizados nesse caso. É *necessário* que você leia o datasheet da unidade que comprou, para garantir que os pinos de saída do registrador de deslocamento sejam conectados corretamente aos pinos do display de pontos. Para um bom tutorial em PDF sobre como ler um datasheet, acesse *www.egr.msu.edu/classes/ece480/goodman/read_datasheet.pdf*.

Para facilitar, a tabela 7.1 mostra quais pinos devem ser conectados no registrador de deslocamento e no display de matriz de pontos. Ajuste o circuito de acordo com o tipo de display que você adquiriu.

Tabela 7.1 – Pinos necessários para o display de matriz de pontos

	Registrador de deslocamento 1	Registrador de deslocamento 2
Linha 1	Pino 15	
Linha 2	Pino 1	
Linha 3	Pino 2	
Linha 4	Pino 3	
Linha 5	Pino 4	
Linha 6	Pino 5	
Linha 7	Pino 6	
Linha 8	Pino 7	
Coluna 1		Pino 15
Coluna 2		Pino 1
Coluna 3		Pino 2
Coluna 4		Pino 3
Coluna 5		Pino 4
Coluna 6		Pino 5
Coluna 7		Pino 6
Coluna 8		Pino 7

O diagrama esquemático para o display de matriz de pontos utilizado neste projeto pode ser visto na figura 7.2. Como você pode perceber, as linhas e colunas (ânodos e cátodos) não estão ordenadas logicamente. Utilizando a tabela 7.1 e o diagrama da figura 7.2, você pode ver que o pino 15 no registrador de deslocamento 1 deve ser conectado à linha 1 no display e, portanto, vai (por meio de um resistor) ao pino 9 do display. O pino 1 no registrador de deslocamento deve ir para a linha 2 e, portanto, vai para o pino 14 do display, e assim por diante.

Será necessário que você leia o datasheet do display que adquiriu, e que realize um processo semelhante ao que mostramos para verificar quais pinos do registrador de deslocamento devem ser conectados aos pinos do display LED.

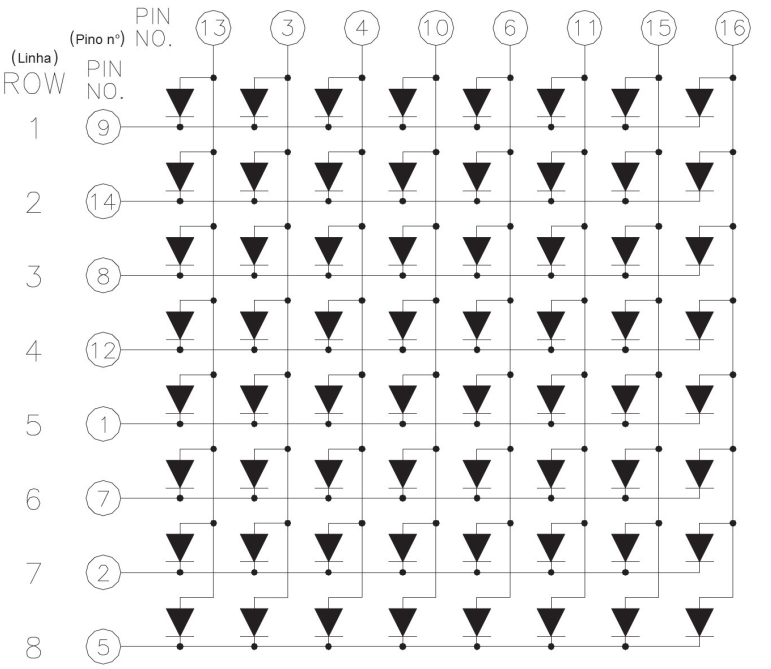


Figura 7.2 – Diagrama esquemático típico para um display de matriz de pontos LED 8 x 8.

Digite o código

Assim que você tiver confirmado que sua fiação está correta, digite o código da listagem 7.1 e faça seu upload para o Arduino. Você também terá de fazer o download da biblioteca TimerOne, que pode ser encontrada no site do Arduino, em *www.arduino.cc/playground/Code/Timer1*. Depois do download da biblioteca, descompacte-a e coloque a pasta `TimerOne` inteira na pasta `hardware/libraries`, dentro da instalação do Arduino. Esse é um exemplo de uma biblioteca externa. O IDE do Arduino vem pré-carregado com muitas bibliotecas, como Ethernet, LiquidCrystal, Servo etc. A biblioteca TimerOne é uma biblioteca externa, e basta fazer seu download e sua instalação para que ela funcione (você terá de reiniciar seu IDE antes que ela seja reconhecida).

Uma biblioteca é simplesmente uma coleção de código escrito por outra pessoa, oferecendo uma funcionalidade que, do contrário, você teria de criar do zero. Tal prática representa o princípio da reutilização de código, e ajuda a acelerar seu processo de desenvolvimento. Afinal, não há nada a ser ganho com a reinvenção da roda. Se alguém já criou um trecho de código que realiza uma tarefa da qual você necessita, e esse código é de domínio público, vá em frente e utilize-o.

Assim que o código tiver sido executado, você verá um coração no display. A cada meio segundo, aproximadamente, o display inverterá para oferecer um efeito de animação básica à imagem.

Listagem 7.1 – Código para o projeto 19

```
// Projeto 19
#include <TimerOne.h>

int latchPin = 8;    // Pino conectado ao pino 12 do 74HC595 (Latch)
int clockPin = 12;   // Pino conectado ao pino 11 do 74HC595 (Clock)
int dataPin = 11;    // Pino conectado ao pino 14 do 74HC595 (Data)

byte led[8];         // array de bytes com 8 elementos para armazenar o sprite

void setup() {
  pinMode(latchPin, OUTPUT);   // define os 3 pinos digitais como saída
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
  led[0] = B11111111;    // insere a representação binária da imagem
  led[1] = B10000001;    // no array
  led[2] = B10111101;
  led[3] = B10100101;
  led[4] = B10100101;
  led[5] = B10111101;
  led[6] = B10000001;
  led[7] = B11111111;
  // define um timer com duração de 10000 microssegundos (1/100 de um segundo)
  Timer1.initialize(10000);
  // anexa a função screenUpdate ao timer de interrupção
  Timer1.attachInterrupt(screenUpdate);
}

void loop() {
  for (int i=0; i<8; i++) {
    led[i]= ~led[i];  // inverte cada linha da imagem binária
  }
  delay(500);
}

void screenUpdate() {             // função para exibir a imagem
  byte row = B10000000;           // linha 1
  for (byte k = 0; k < 9; k++) {
    digitalWrite(latchPin, LOW);   // abre o latch, deixando-o pronto para receber dados
  shiftIt(~led[k]);            // envia o array de LEDs (invertido) para os chips
  shiftIt(row);                // envia o número binário da linha para os chips

  // Fecha o latch, enviando os dados no registrador para o display de matriz
  digitalWrite(latchPin, HIGH);
  row = row >> 1;    // deslocamento para a direita
  }
}
```

```
void shiftIt(byte dataOut) {      // Desloca 8 bits, com o menos significativo deslocado
                                  // primeiro, durante o extremo ascendente do clock

  boolean pinState;
  digitalWrite(dataPin, LOW);    // libera o registrador de deslocamento, deixando-o pronto
                                  // para enviar dados

  for (int i=0; i<8; i++)  {     // para cada bit em dataOut, envie um bit
    digitalWrite(clockPin, LOW); // define clockPin como LOW, antes de enviar o bit

    // se o valor de dataOut e (E lógico) uma máscara de bits
    // forem verdadeiros, defina pinState como 1 (HIGH)
    if ( dataOut & (1<<i) ) {
      pinState = HIGH;
    }
    else {
      pinState = LOW;
    }
    // define dataPin como HIGH ou LOW, dependendo de pinState
    digitalWrite(dataPin, pinState);
    digitalWrite(clockPin, HIGH);    // envia o bit durante o extremo ascendente do clock
    digitalWrite(dataPin, LOW);
  }
digitalWrite(clockPin, LOW); // interrompe o deslocamento
}
```

Projeto 19 – Display de matriz de pontos LED – Animação básica – Análise do hardware

Para este projeto, veremos como funciona o hardware, antes do código. Isso fará com que seja mais fácil compreender o código posteriormente.

Você aprendeu como utilizar o 74HC595 nos projetos anteriores. A única adição ao circuito, dessa vez, é uma unidade de display 8 x 8 de pontos LED.

Unidades de matriz de pontos tipicamente vêm no formato de uma matriz de LEDs de 5 x 7 ou 8 x 8. Os LEDs são conectados à matriz de forma que o ânodo ou o cátodo de cada LED seja comum em cada linha. Em outras palavras, em uma unidade de matriz de pontos LED habitual, cada linha de LEDs terá todos os seus ânodos conectados. Os cátodos de cada coluna também estarão todos conectados. A razão disso vai ficar aparente muito em breve.

Uma típica unidade de matriz de pontos 8 x 8 colorida terá 16 pinos, oito para as linhas e oito para as colunas. Você também pode obter unidades bicolores (por exemplo, nas cores vermelho e verde), assim como unidades RGB — aquelas utilizadas

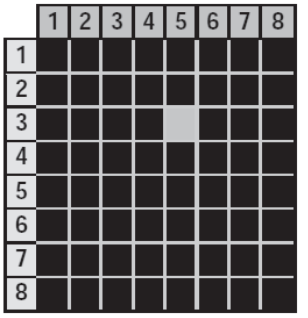
em grandes telões. Unidades bicolores ou tricolores (RGB) têm dois ou três LEDs em cada pixel do array. São LEDs muito pequenos, e que ficam posicionados bem próximos uns dos outros.

Ao acender combinações diferentes de vermelho, verde ou azul em cada pixel, além de variar seu brilho, pode-se obter qualquer cor.

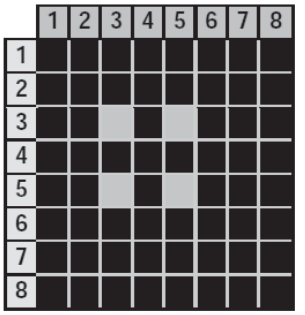
O motivo de as linhas e colunas serem todas conectadas é para minimizar o número de pinos necessários. Se não o fizéssemos, uma única unidade de matriz de pontos 8 x 8 colorida necessitaria de 65 pinos, um para cada LED e um conector de ânodo ou cátodo comum. Ligando a fiação das linhas e colunas, apenas 16 pinos são necessários.

Entretanto, isso representa um problema se você deseja que um LED específico acenda em certa posição. Caso, por exemplo, você tivesse uma unidade de ânodo comum e quisesse acender o LED na posição X, Y, de valores 5, 3 (quinta coluna, terceira linha), você aplicaria uma corrente à terceira linha e o terra ao pino da quinta coluna.

O LED na quinta coluna e na terceira linha acenderia.



Agora, imagine que você deseja acender também o LED na coluna 3, linha 5. Então, você aplicaria uma corrente à quinta linha e o terra ao pino da terceira coluna. O LED correspondente agora seria iluminado. Mas espere... os LEDs da coluna 3, linha 3 e coluna 5, linha 5 também acenderam.



Isso ocorre porque você está aplicando energia às linhas 3 e 5, e o terra às colunas 3 e 5. Você não pode apagar os LEDs indesejados sem apagar também aqueles que devem estar acesos. Parece não haver forma de acender apenas os LEDs necessários, com a fiação que utilizamos para linhas e colunas. A única maneira de isso dar certo seria se tivéssemos uma pinagem separada para cada LED, o que faria o número de pinos pular de 16 para 65. Uma unidade de matriz de pontos de 65 pinos teria uma fiação muito complexa e seria muito difícil de controlar, pois você necessitaria de um microcontrolador de ao menos 64 saídas digitais.

Haveria uma solução para esse problema? Sim, ela existe, e é o que chamamos de *multiplexação* (*multiplexing*, ou *muxing*).

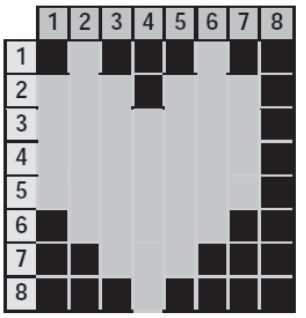
Multiplexação

A multiplexação é a técnica de acender uma linha do display de cada vez. Selecionando a coluna que contém a linha, que, por sua vez, contém o LED que você deseja acender, e ligando a alimentação para essa linha (ou da forma oposta, para displays comuns de cátodo), os LEDs escolhidos nessa linha serão iluminados. Essa linha será, então, apagada, e a próxima acesa, novamente com as colunas apropriadas escolhidas, e fazendo com que os LEDs da segunda linha agora sejam iluminados. Repita esse processo para cada linha até que você atinja a base e, então, reinicie do topo.

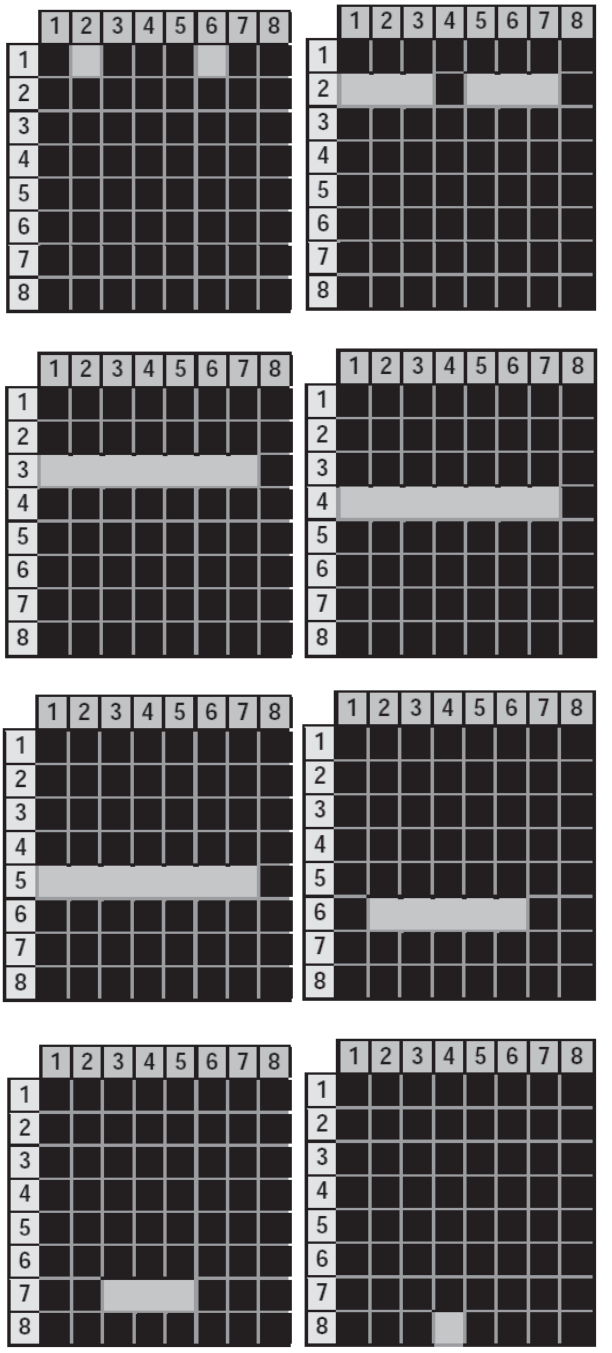
Se isso for feito a uma velocidade suficiente (mais de 100 Hz, ou cem vezes por segundo), o fenômeno de *persistência da visão* (em que uma pós-imagem permanece na retina por, aproximadamente, 1/25 de um segundo) fará com que o display pareça acender por inteiro, mesmo que cada linha acenda e apague em sequência.

Utilizando essa técnica, você soluciona o problema da exibição de LEDs individuais, sem que outros LEDs na mesma coluna também acendam.

Por exemplo, vamos supor que você queira exibir a imagem a seguir em seu display:



Cada linha seria acesa da seguinte maneira:



Descendo pelas linhas, iluminando os LEDs respectivos em cada coluna, e fazendo isso muito rapidamente (a mais de 100 Hz), o olho humano reconhecerá a imagem por inteiro, e um coração poderá ser reconhecido no padrão dos LEDs.

Você está utilizando essa técnica de multiplexação no código do projeto. É dessa forma que você exibe a animação do coração, sem acender LEDs desnecessários.

Projeto 19 – Display de matriz de pontos LED – Animação básica – Análise do código

O código para este projeto utiliza um recurso do chip ATmega, conhecido como Hardware Timer: essencialmente, um timer que pode ser utilizado para disparar um evento. Em seu caso, você está definindo a ISR (Interrupt Service Routine, ou rotina de serviço de interrupção) para que dispare a cada 10 mil microssegundos, o equivalente a um centésimo de segundo.

Neste código, você utiliza uma biblioteca que facilita o uso de interrupções, a TimerOne, e que torna muito fácil criar uma ISR. Você simplesmente diz à função qual é o intervalo (nesse caso, 10 mil microssegundos), e passa o nome da função que deseja ativar cada vez que a interrupção for disparada (nesse caso, a função `screenUpdate()`)

TimerOne é uma biblioteca externa, por isso você tem de incluí-la em seu código. Isso pode ser feito com facilidade utilizando a diretiva `#include`:

```
#include <TimerOne.h>
```

Na sequência, são declarados os pinos utilizados para interfacear os registradores de deslocamento:

```
int latchPin = 8;    // Pino conectado ao pino 12 do 74HC595 (Latch)
int clockPin = 12;   // Pino conectado ao pino 11 do 74HC595 (Clock)
int dataPin = 11;    // Pino conectado ao pino 14 do 74HC595 (Data)
```

Depois, você cria um array de tipo byte, com oito elementos. O array `led[8]` será utilizado para armazenar a imagem a ser exibida no display de matriz de pontos:

```
byte led[8];    // array de bytes com 8 elementos para armazenar o sprite
```

Na rotina de setup, você define os pinos do latch, do clock e de dados como saídas:

```
void setup() {
  pinMode(latchPin, OUTPUT);   // define os 3 pinos digitais como saída
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
```

Assim que os pinos tiverem sido definidos como saída, o array `led` será carregado com as imagens binárias de 8 bits que serão exibidas em cada linha do display de matriz de pontos 8 x 8:

```
  led[0] = B11111111;    // insere a representação binária da imagem
  led[1] = B10000001;    // no array
  led[2] = B10111101;
  led[3] = B10100101;
  led[4] = B10100101;
```

```
led[5] = B10111101;
led[6] = B10000001;
led[7] = B11111111;
```

Analisando o array anterior, você pode imaginar a imagem que será exibida, que é uma caixa dentro de uma caixa. Os 1s indicam os LEDs que acenderão, e os 0s indicam os LEDs que ficarão apagados. Você pode, evidentemente, ajustar os 1s e 0s como quiser, e criar um sprite 8 x 8 de sua escolha.

Depois disso, utilizamos a função `Timer1`. Primeiramente, ela tem de ser inicializada com a frequência na qual será ativada. Nesse caso, você define o período como 10 mil microssegundos, ou um centésimo de segundo. Assim que a interrupção tiver sido inicializada, você deve anexar a ela uma função que será executada sempre que o período de tempo for atingido. Esta é a função `screenUpdate()`, que disparará a cada centésimo de segundo:

```
// define um timer com duração de 10000 microssegundos (1/100 de segundo)
Timer1.initialize(10000);
// anexa a função screenUpdate ao timer de interrupção
Timer1.attachInterrupt(screenUpdate);
```

No loop principal, um loop `for` percorre cada um dos oito elementos do array `led` e inverte o conteúdo, utilizando o operador bit a bit `~`, ou NÃO. Isso simplesmente transforma a imagem binária em um negativo dela mesma, transformando todos os 1s em 0s e todos os 0s em 1s. Então, esperamos 500 milissegundos antes de repetir o processo.

```
for (int i=0; i<8; i++) {
  led[i]= ~led[i];  // inverte cada linha da imagem binária
}
delay(500);
```

Agora temos a função `screenUpdate()`, ativada pela interrupção a cada centésimo de segundo. Toda essa rotina é muito importante, pois garante que os LEDs do array na matriz de pontos acendam corretamente e exibam a imagem que você deseja representar. Trata-se de uma função muito simples, mas eficiente.

```
void screenUpdate() {             // função para exibir a imagem
  byte row = B10000000;          // linha 1
  for (byte k = 0; k < 9; k++) {
    digitalWrite(latchPin, LOW);   // abre o latch, deixando-o pronto para receber dados
  shiftIt(~led[k] );           // envia o array de LEDs (invertido) para os chips
  shiftIt(row );               // envia o número binário da linha para os chips

  // Fecha o latch, enviando os dados no registrador para o display de matriz
  digitalWrite(latchPin, HIGH);
  row = row >> 1; // deslocamento para a direita
  }
}
```

Um byte, `row`, é declarado e inicializado com o valor B10000000:

```
byte row = B10000000;    // linha 1
```

Você, agora, percorre o array `led`, e envia esses dados para o registrador de deslocamento (processado com o operador bit a bit NÃO (~), para garantir que as colunas que você deseja exibir estejam apagadas, ou ligadas ao terra), seguidos pela linha:

```
for (byte k = 0; k < 9; k++) {
  digitalWrite(latchPin, LOW); // abre o latch, deixando-o pronto para receber dados
  shiftIt(~led[k] );           // envia o array de LEDs (invertido) para os chips
  shiftIt(row );               // envia o número binário da linha para os chips
```

Assim que você tiver deslocado os oito bits da linha atual, o valor em `row` será deslocado um bit para a direita, de forma que a próxima linha seja exibida (por exemplo, a linha com o 1 é exibida). Você aprendeu sobre o comando `bitshift` no capítulo 6.

```
row = row >> 1;   // deslocamento para a direita
```

Lembre-se de que vimos, na análise do hardware, que a rotina de multiplexação está exibindo apenas uma linha de cada vez e, depois, apagando essa linha e exibindo a linha seguinte. Isso é feito a 100 Hz, rápido demais para que o olho humano perceba uma cintilação.

Por fim, você tem a função `shiftIt`, a mesma dos projetos anteriores com base em registradores de deslocamento, a qual envia os dados para os chips do 74HC595:

```
void shiftIt(byte dataOut)
```

Portanto, o conceito básico deste projeto é uma rotina de interrupção executada a cada centésimo de segundo. Nessa rotina, você apenas analisa o conteúdo de um array de buffer de tela (nesse caso, `led[]`), exibindo-o na unidade de matriz de pontos, uma linha de cada vez. Isso é feito tão rapidamente que, para o olho humano, tudo parece acontecer ao mesmo tempo.

O loop principal do programa está simplesmente alterando o conteúdo do array do buffer de tela, permitindo que a ISR faça o trabalho necessário.

A animação neste projeto é muito simples, mas manipulando os 1s e 0s no buffer, você pode representar o que quiser na unidade de matriz de pontos, desde formas diferentes até textos de rolagem horizontal. No próximo projeto, vamos experimentar uma variação do que fizemos, na qual você criará um sprite animado com rolagem horizontal.

Projeto 20 – Display de matriz de pontos LED – Sprite com rolagem horizontal

Neste projeto, você utilizará o mesmo circuito do projeto anterior, mas com uma pequena variação no código para criar uma animação de vários quadros (frames), que também se movimentará da direita para a esquerda. Nesse processo, você será

apresentado ao conceito de arrays multidimensionais, e também aprenderá um pequeno truque para realizar a rotação de bits (ou deslocamento circular). Para iniciar, você utilizará exatamente o mesmo circuito do projeto 19.

Digite o código

Digite e faça o upload do código da listagem 7.2.

Listagem 7.2 – Código para o projeto 20

```
// Projeto 20
#include <TimerOne.h>

int latchPin = 8;    // Pino conectado ao pino 12 do 74HC595 (Latch)
int clockPin = 12;   // Pino conectado ao pino 11 do 74HC595 (Clock)
int dataPin = 11;    // Pino conectado ao pino 14 do 74HC595 (Data)
byte frame = 0;      // variável para armazenar o quadro atual sendo exibido

byte led[8][8] ={ {0, 56, 92, 158, 158, 130, 68, 56},    // 8 quadros de uma animação
           {0, 56, 124, 186, 146, 130, 68, 56},
           {0, 56, 116, 242, 242, 130, 68, 56},
           {0, 56, 68, 226, 242, 226, 68, 56},
           {0, 56, 68, 130, 242, 242, 116, 56},
           {0, 56, 68, 130, 146, 186, 124, 56},
           {0, 56, 68, 130, 158, 158, 92, 56},
           {0, 56, 68, 142, 158, 142, 68, 56} };

void setup() {
  pinMode(latchPin, OUTPUT);   // define os 3 pinos digitais como saídas
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);

  Timer1.initialize(10000);  // define um timer com duração de 10000 microssegundos
  Timer1.attachInterrupt(screenUpdate);    // anexa a função screenUpdate
}

void loop() {
  for (int i=0; i<8; i++) {  // faz um loop, percorrendo todos os 8 quadros da animação
    for (int j=0; j<8; j++) {  // faz um loop pelas 8 linhas de cada quadro
      led[i][j] = led[i][j] << 1 | led[i][j] >> 7; // rotação bit a bit
    }
  }
  frame++;               // vai para o próximo quadro da animação
  if (frame>7) { frame =0; }     // certifica-se de retornar ao frame 0, depois de passar do 7
  delay(100);              // espera um pouco entre cada frame
}
```

```
void screenUpdate() {            // função para exibir a imagem
  byte row = B10000000;         // linha 1
  for (byte k = 0; k < 9; k++) {
    digitalWrite(latchPin, LOW); // abre o latch, deixando-o pronto para receber dados

    shiftIt(~led[frame][k]); // envia o array de LEDs (invertido) para os chips
    shiftIt(row);        // envia o número binário da linha para os chips

    // Fecha o latch, enviando os dados nos registradores para a matriz de pontos
    digitalWrite(latchPin, HIGH);
    row = row >> 1; // deslocamento para a direita
  }
}

void shiftIt(byte dataOut) {
  // Desloca 8 bits, com o menos significativo deslocado primeiro, durante o extremo ascendente
  // do clock
  boolean pinState;

  // libera o registrador de deslocamento, deixando-o pronto para enviar dados
  digitalWrite(dataPin, LOW);

  // para cada bit em dataOut, envie um bit
  for (int i=0; i<8; i++)  {
    // define clockPin como LOW, antes de enviar o bit
    digitalWrite(clockPin, LOW);

    // se o valor de dataOut e (E lógico) uma máscara de bits
    // forem verdadeiros, define pinState como 1 (HIGH)
    if ( dataOut & (1<<i) ) {
      pinState = HIGH;
    }
    else {
      pinState = LOW;
    }

    // define dataPin como HIGH ou LOW, dependendo do pinState
    digitalWrite(dataPin, pinState);
    // envia o bit durante o extremo ascendente do clock
    digitalWrite(clockPin, HIGH);
    digitalWrite(dataPin, LOW);
  }

digitalWrite(clockPin, LOW);   // interrompe o deslocamento
}
```

Quando você executar o projeto 20, poderá ver uma roda animada, rolando horizontalmente. O hardware não sofreu alterações, por isso não é necessário discuti-lo. Vamos descobrir como funciona o código.

Projeto 20 – Display de matriz de pontos LED – Sprite com rolagem horizontal – Análise do código

Novamente, você carrega a biblioteca TimerOne e define os três pinos que controlam os registradores de deslocamento:

```
#include <TimerOne.h>

int latchPin = 8;    // Pino conectado ao pino 12 do 74HC595 (Latch)
int clockPin = 12;   // Pino conectado ao pino 11 do 74HC595 (Clock)
int dataPin = 11;    // Pino conectado ao pino 14 do 74HC595 (Data)
```

Depois, você declara uma variável de tipo `byte`, inicializando-a como 0. Ela armazenará o número do quadro atualmente exibido na animação de oito quadros:

```
byte frame = 0;   // variável para armazenar o quadro atual sendo exibido
```

Na sequência, você prepara um array bidimensional de tipo `byte`:

```
byte led[8][8] = { {0, 56, 92, 158, 158, 130, 68, 56}, // 8 quadros de uma animação
           {0, 56, 124, 186, 146, 130, 68, 56},
           {0, 56, 116, 242, 242, 130, 68, 56},
           {0, 56, 68, 226, 242, 226, 68, 56},
           {0, 56, 68, 130, 242, 242, 116, 56},
           {0, 56, 68, 130, 146, 186, 124, 56},
           {0, 56, 68, 130, 158, 158, 92, 56},
           {0, 56, 68, 142, 158, 142, 68, 56} };
```

Arrays foram apresentados no capítulo 3. Um array é um conjunto de variáveis que podem ser acessadas utilizando um número de índice. O array que utilizamos agora é diferente, pois tem dois conjuntos de números de índice para os elementos. No capítulo 3, você declarou um array unidimensional desta forma:

```
byte ledPin[] = {4, 5, 6, 7, 8, 9, 10, 11, 12, 13};
```

Aqui, você tem de criar um array bidimensional, com dois conjuntos de números de índice. Nesse caso, seu array tem 8 x 8, ou 64 elementos no total. Um array bidimensional é praticamente idêntico a uma tabela bidimensional: você pode acessar uma célula individual, referenciando os números de linha e de coluna correspondentes. A tabela 7.2 mostra como acessar os elementos em seu array.

As linhas representam o primeiro número do índice do array, por exemplo `byte led[7][..]`, e as colunas representam o segundo índice, por exemplo `byte led[..][7]`. Para acessar o número `158` na linha 6, coluna 4, você utilizaria `byte led[6][4]`.

Tabela 7.2 – Elementos em seu array

	0	1	2	3	4	5	6	7
0	0	56	92	158	158	130	68	56
1	0	56	124	186	146	130	68	56
2	0	56	116	242	242	130	68	56
3	0	56	68	226	242	226	68	56
4	0	56	68	130	242	242	116	56
5	0	56	68	130	146	186	124	56
6	0	56	68	130	158	158	92	56
7	0	56	68	142	158	142	68	56

Note que, ao declarar o array, você também o inicializou com dados. Para inicializar dados em um array bidimensional, você coloca todos eles dentro de chaves globais, e cada conjunto de dados dentro de suas próprias chaves com uma vírgula ao final, da seguinte maneira:

```
byte led[8][8] = { {0, 56, 92, 158, 158, 130, 68, 56},
           {0, 56, 124, 186, 146, 130, 68, 56},
           {0, 56, 116, 242, 242, 130, 68, 56},    // etc., etc.
```

O array bidimensional armazenará os oito quadros de sua animação. O primeiro índice do array fará referência ao quadro da animação, e o segundo a qual das oito linhas de números de 8 bits formará o padrão de LEDs que devem acender e apagar. Para economizar espaço no código, os números foram convertidos de binários para decimais. Se você visse os números binários, poderia discernir a animação da figura 7.3.

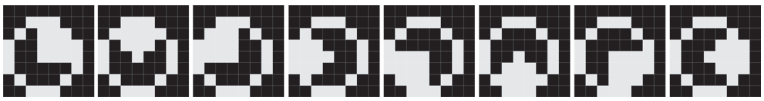


Figura 7.3 – Animação da roda girando.

Evidentemente, você pode alterar essa animação para representar o que quiser, aumentando ou diminuindo o número de quadros. Esboce sua animação no papel, converta as linhas para números binários de 8 bits, e coloque-os em seu array.

No loop de inicialização, você define os três pinos novamente como saída, cria um objeto de timer com duração de 10 mil microssegundos e anexa a função `screenUpdate()` à interrupção:

```
void setup() {
  pinMode(latchPin, OUTPUT);   // define os 3 pinos digitais como saídas
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);

  Timer1.initialize(10000);         // define um timer com duração de 10000 microssegundos
  Timer1.attachInterrupt(screenUpdate);    // anexa a função screenUpdate
}
```

No loop principal, você faz um loop percorrendo cada uma das oito linhas do sprite, assim como no projeto 19. Entretanto, este loop está dentro de outro loop, que se repete oito vezes e controla qual frame você deseja exibir:

```
void loop() {
  for (int i=0; i<8; i++) {    // faz um loop por todos os 8 quadros da animação
    for (int j=0; j<8; j++) {    // faz um loop pelas 8 linhas em cada quadro
```

Na sequência, você pega todos os elementos do array, um de cada vez, e desloca seus valores uma posição para a esquerda. Entretanto, utilizando um pequeno e interessante truque de lógica, você garante que todos os bits deslocados para fora, no lado esquerdo, retornem no lado direito. Isso é feito com o seguinte comando:

```
led[i][j]= led[i][j] << 1 | led[i][j] >> 7;   // rotação bit a bit
```

Aqui, o elemento atual do array, escolhido pelos inteiros `i` e `j`, é deslocado uma posição para a esquerda. Entretanto, você, depois, pega esse resultado e aplica um OU lógico com o número de valor `led [i][j]` deslocado sete posições para a direita. Vejamos como isso funciona.

Suponha que o valor atual de `led[i][j]` seja `156`, ou o número binário 10011100. Se esse número for deslocado para a esquerda uma posição, você terá 00111000. Agora, se você pegar o mesmo número, `156`, e deslocá-lo para a direita sete vezes, terá como resultado 00000001. Em outras palavras, fazendo isso, você desloca o dígito binário da extrema esquerda para o lado direito. Em seguida, você realiza uma operação lógica bit a bit OU nos dois números. Lembre-se de que o cálculo bit a bit OU produzirá um número 1 se qualquer um dos dígitos for 1, da seguinte maneira:

```
00111000 |
00000001 =

--------

00111001
```

Com isso, você deslocou o número uma posição para a esquerda, e utilizou um OU, comparando esse resultado com o mesmo número deslocado sete posições para a direita. Como você pode ver no cálculo, o resultado é o mesmo de deslocar o número para a esquerda uma vez, e deslocar para o lado direito qualquer dígito que tenha saído pelo lado esquerdo. Isso é conhecido como *rotação bit a bit* ou *deslocamento circular*, técnica frequentemente utilizada em criptografia digital. Você pode realizar uma rotação bit a bit em um dígito binário de qualquer extensão, utilizando o seguinte cálculo:

```
i << n | i >> (a - n);
```

No qual `n` é o número de dígitos que você deseja rotacionar e `a` é a extensão, em bits, de seu dígito original.

Na sequência, você incrementa o valor de `frame` em `1`, verifica se ele é maior que `7` e, caso seja, define o número novamente como 0. Isso fará um ciclo por cada um dos oito quadros da animação, um quadro de cada vez, até que atinja o fim dos quadros, e depois repetirá a operação. Por fim, há uma espera de 100 milissegundos.

```
frame++;          // vai para o próximo quadro na animação
if (frame>7) { frame =0;}  // certifica-se de retornar ao frame 0 depois de passar do 7
delay(100);          // espera um pouco entre cada frame
```

Depois, você executa as funções `screenUpdate()` e `shiftIt()`, da mesma forma que nos projetos anteriores, com base em registradores de deslocamento. No próximo projeto, você utilizará novamente uma matriz de pontos LED, mas dessa vez não fará uso de registradores de deslocamento. Em vez disso, você utilizará o popular chip MAX7219.

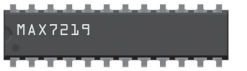
Projeto 21 – Display de matriz de pontos LED – Mensagem com rolagem horizontal

Há muitas formas diferentes de controlar LEDs. Utilizar registradores de deslocamento é apenas uma das opções, e tem suas vantagens. Entretanto, há muitos outros CIs disponíveis, especificamente projetados para controlar displays LED, facilitando muito a sua vida. Um dos CIs controladores de LEDs mais populares na comunidade do Arduino é o MAX7219, um controlador de displays de LED com interface serial de oito dígitos, feito pela Maxim. Esses chips são projetados para controlar displays de LED numéricos de sete segmentos com até oito dígitos, displays de gráficos de barras, ou displays LED de matriz de pontos 8 x 8, sendo este o uso que faremos deles. O IDE do Arduino vem como uma biblioteca, Matrix, que acompanha código de exemplo escrito especificamente para os chips MAX7219. Essa biblioteca facilita, e muito, a utilização desses chips. Entretanto, neste projeto você não utilizará nenhuma biblioteca externa. Em vez disso, você escolherá o caminho mais difícil, escrevendo todo o código você mesmo. Dessa forma, você aprenderá exatamente como o chip MAX7219 funciona, e poderá transferir suas habilidades para o uso de qualquer outro chip controlador de LEDs.

Componentes necessários

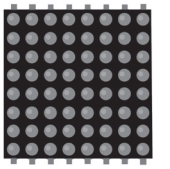
Será necessário um CI controlador de LED MAX7219. Como alternativa, você pode utilizar um AS1107 da Austria Microsystems, praticamente idêntico ao MAX7219, e que funcionará sem necessidade de alterações no seu código ou circuito. Dessa vez, o display de matriz de pontos 8 x 8 deve ser do tipo cátodo comum, uma vez que o chip MAX não funcionará com um display de ânodo comum.

MAX7219 (ou AS1107)



Resistor limitador de corrente

Display de matriz de pontos 8 x 8 (C-)



Conectando os componentes

Analise o diagrama da figura 7.4 cuidadosamente. Certifique-se de que seu Arduino esteja desligado enquanto conecta os fios. A fiação do MAX7219 para o display de matriz de pontos, na Figura 7.4, foi criada de acordo com a unidade de display específica que utilizei. Os pinos de seu display podem ser diferentes. Isso não é relevante, simplesmente conecte os pinos que saem do MAX7219 aos pinos de coluna e linha apropriados em seu display (consulte a tabela 7.3 para a pinagem). Uma leitura horizontal mostrará quais dispositivos estão conectados a quais pinos. No display, as colunas são os cátodos, e as linhas, os ânodos. Em meu display, verifiquei que a linha 1 estivesse na base e a linha 8 no topo. Você pode ter de inverter a ordem em seu display, se notar que as letras estão invertidas ou ao contrário. Conecte os 5 V do Arduino ao barramento positivo da protoboard, e o terra ao barramento do terra.

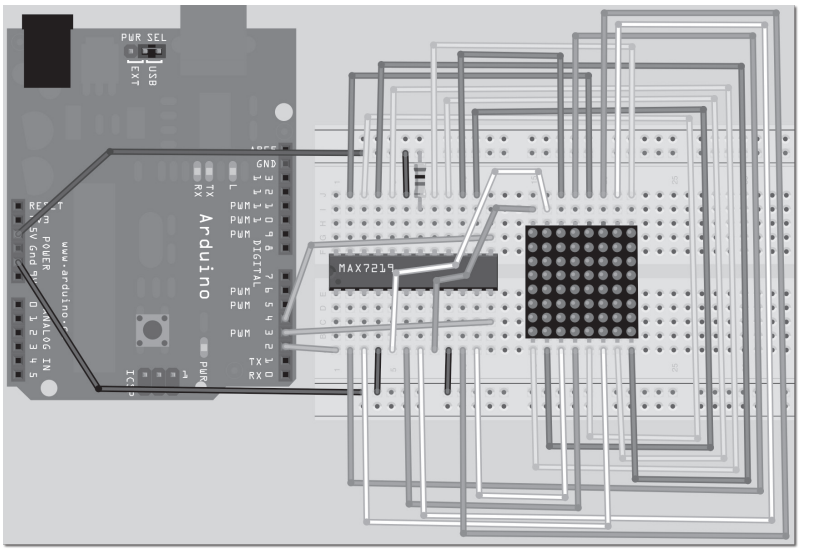


Figura 7.4 – Circuito para o Projeto 21 (consulte o site da Novatec para versão colorida).

Tabela 7.3 – Pinagem entre o Arduino, o CI e o display de matriz de pontos

Arduino	MAX7219	Display	Outros
Digital 2	1 (DIN)		
Digital 3	12 (LOAD)		
Digital 4	13 (CLK)		
	4,9		Gnd (Terra)
	19		+5 V
	18 (ISET)		Resistor para +5 V
	2 (DIG 0)	Coluna 1	
	11 (DIG 1)	Coluna 2	
	6 (DIG 2)	Coluna 3	
	7 (DIG 3)	Coluna 4	
	3 (DIG 4)	Coluna 5	
	10 (DIG 5)	Coluna 6	
	5 (DIG 6)	Coluna 7	
	8 (DIG 7)	Coluna 8	
	22 (SEG DP)	Linha 1	
	14 (SEG A)	Linha 2	
	16 (SEG B)	Linha 3	
	20 (SEG C)	Linha 4	
	23 (SEG D)	Linha 5	
	21 (SEG E)	Linha 6	
	15 (SEG F)	Linha 7	
	17 (SEG G)	Linha 8	

Verifique suas conexões antes de ligar o Arduino.

Digite o código

Digite e faça o upload do código da listagem 7.3.

Listagem 7.3 – Código para o projeto 21

```
#include <avr/pgmspace.h>
#include <TimerOne.h>

int DataPin = 2;  // Pino 1 no MAX
int LoadPin = 3;  // Pino 12 no MAX
int ClockPin = 4; // Pino 13 no MAX
byte buffer[8];

static byte font[][8] PROGMEM = {
```

```
// Apenas os caracteres ASCII imprimíveis (32-126)
{B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00000000},
{B00000100, B00000100, B00000100, B00000100, B00000100, B00000100, B00000000, B00000100},
{B00001010, B00001010, B00001010, B00000000, B00000000, B00000000, B00000000, B00000000},
{B00000000, B00001010, B00011111, B00001010, B00011111, B00001010, B00011111, B00001010},
{B00000111, B00001100, B00010100, B00001100, B00000110, B00000101, B00000110, B00011100},
{B00011001, B00011010, B00000010, B00000100, B00000100, B00001000, B00001011, B00010011},
{B00000110, B00001010, B00010010, B00010100, B00001001, B00010110, B00010110, B00001001},
{B00000100, B00000100, B00000100, B00000000, B00000000, B00000000, B00000000, B00000000},
{B00000010, B00000100, B00001000, B00001000, B00001000, B00001000, B00000100, B00000010},
{B00001000, B00000100, B00000010, B00000010, B00000010, B00000010, B00000100, B00001000},
{B00010101, B00001110, B00011111, B00001110, B00010101, B00000000, B00000000, B00000000},
{B00000000, B00000000, B00000100, B00000100, B00011111, B00000100, B00000100, B00000000},
{B00000000, B00000000, B00000000, B00000000, B00000000, B00000110, B00000100, B00001000},
{B00000000, B00000000, B00000000, B00000000, B00001110, B00000000, B00000000, B00000000},
{B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00000100},
{B00000001, B00000010, B00000010, B00000100, B00000100, B00001000, B00001000, B00010000},
{B00001110, B00010001, B00010011, B00010001, B00010101, B00010001, B00011001, B00001110},
{B00000100, B00001100, B00010100, B00000100, B00000100, B00000100, B00000100, B00011111},
{B00001110, B00010001, B00010001, B00000010, B00000100, B00001000, B00010000, B00011111},
{B00001110, B00010001, B00000001, B00001110, B00000001, B00000001, B00010001, B00001110},
{B00010000, B00010000, B00010100, B00010100, B00011111, B00000100, B00000100, B00000100},
{B00011111, B00010000, B00010000, B00011110, B00000001, B00000001, B00000001, B00011110},
{B00000111, B00001000, B00010000, B00011110, B00010001, B00010001, B00010001, B00001110},
{B00011111, B00000001, B00000001, B00000001, B00000010, B00000100, B00001000, B00010000},
{B00001110, B00010001, B00010001, B00001110, B00010001, B00010001, B00010001, B00001110},
{B00001110, B00010001, B00010001, B00001111, B00000001, B00000001, B00000001, B00000001},
{B00000000, B00000100, B00000100, B00000000, B00000000, B00000100, B00000100, B00000000},
{B00000000, B00000100, B00000100, B00000000, B00000000, B00000100, B00000100, B00001000},
{B00000001, B00000010, B00000100, B00001000, B00001000, B00000100, B00000010, B00000001},
{B00000000, B00000000, B00000000, B00011110, B00000000, B00011110, B00000000, B00000000},
{B00010000, B00001000, B00000100, B00000010, B00000010, B00000100, B00001000, B00010000},
{B00001110, B00010001, B00010001, B00000010, B00000100, B00000100, B00000000, B00000100},
{B00001110, B00010001, B00010001, B00010101, B00010101, B00010001, B00010001, B00011110},
{B00001110, B00010001, B00010001, B00010001, B00011111, B00010001, B00010001, B00010001},
{B00011110, B00010001, B00010001, B00011110, B00010001, B00010001, B00010001, B00011110},
{B00000111, B00001000, B00010000, B00010000, B00010000, B00010000, B00001000, B00000111},
{B00011100, B00010010, B00010001, B00010001, B00010001, B00010001, B00010010, B00011100},
{B00011111, B00010000, B00010000, B00011110, B00010000, B00010000, B00010000, B00011111},
{B00011111, B00010000, B00010000, B00011110, B00010000, B00010000, B00010000, B00010000},
{B00001110, B00010001, B00010000, B00010000, B00010111, B00010001, B00010001, B00001110},
{B00010001, B00010001, B00010001, B00011111, B00010001, B00010001, B00010001, B00010001},
{B00011111, B00000100, B00000100, B00000100, B00000100, B00000100, B00000100, B00011111},
{B00011111, B00000100, B00000100, B00000100, B00000100, B00000100, B00010100, B00001000},
{B00010001, B00010010, B00010100, B00011000, B00010100, B00010010, B00010001, B00010001},
{B00010000, B00010000, B00010000, B00010000, B00010000, B00010000, B00010000, B00011111},
```

```
{B00010001, B00011011, B00011111, B00010101, B00010001, B00010001, B00010001, B00010001},
{B00010001, B00011001, B00011001, B00010101, B00010101, B00010011, B00010011, B00010001},
{B00001110, B00010001, B00010001, B00010001, B00010001, B00010001, B00010001, B00001110},
{B00011110, B00010001, B00010001, B00011110, B00010000, B00010000, B00010000, B00010000},
{B00001110, B00010001, B00010001, B00010001, B00010001, B00010101, B00010011, B00001111},
{B00011110, B00010001, B00010001, B00011110, B00010100, B00010010, B00010001, B00010001},
{B00001110, B00010001, B00010000, B00001000, B00000110, B00000001, B00010001, B00001110},
{B00011111, B00000100, B00000100, B00000100, B00000100, B00000100, B00000100, B00000100},
{B00010001, B00010001, B00010001, B00010001, B00010001, B00010001, B00010001, B00001110},
{B00010001, B00010001, B00010001, B00010001, B00010001, B00010001, B00001010, B00000100},
{B00010001, B00010001, B00010001, B00010001, B00010001, B00010101, B00010101, B00001010},
{B00010001, B00010001, B00001010, B00000100, B00000100, B00001010, B00010001, B00010001},
{B00010001, B00010001, B00001010, B00000100, B00000100, B00000100, B00000100, B00000100},
{B00011111, B00000001, B00000010, B00000100, B00001000, B00010000, B00010000, B00011111},
{B00001110, B00001000, B00001000, B00001000, B00001000, B00001000, B00001000, B00001110},
{B00010000, B00001000, B00001000, B00000100, B00000100, B00000010, B00000010, B00000001},
{B00001110, B00000010, B00000010, B00000010, B00000010, B00000010, B00000010, B00001110},
{B00000100, B00001010, B00010001, B00000000, B00000000, B00000000, B00000000, B00000000},
{B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00011111},
{B00001000, B00000100, B00000000, B00000000, B00000000, B00000000, B00000000, B00000000},
{B00000000, B00000000, B00000000, B00001110, B00010010, B00010010, B00010010, B00001111},
{B00000000, B00010000, B00010000, B00010000, B00011100, B00010010, B00010010, B00011100},
{B00000000, B00000000, B00000000, B00001110, B00010000, B00010000, B00010000, B00001110},
{B00000000, B00000001, B00000001, B00000001, B00000111, B00001001, B00001001, B00000111},
{B00000000, B00000000, B00000000, B00011100, B00010010, B00011110, B00010000, B00001110},
{B00000000, B00000011, B00000100, B00000100, B00000110, B00000100, B00000100, B00000100},
{B00000000, B00001110, B00001010, B00001010, B00001110, B00000010, B00000010, B00001100},
{B00000000, B00010000, B00010000, B00010000, B00011100, B00010010, B00010010, B00010010},
{B00000000, B00000000, B00000100, B00000000, B00000100, B00000100, B00000100, B00000100},
{B00000000, B00000010, B00000000, B00000010, B00000010, B00000010, B00000010, B00001100},
{B00000000, B00010000, B00010000, B00010100, B00011000, B00011000, B00010100, B00010000},
{B00000000, B00010000, B00010000, B00010000, B00010000, B00010000, B00010000, B00001100},
{B00000000, B00000000, B00000000, B00001010, B00010101, B00010001, B00010001, B00010001},
{B00000000, B00000000, B00000000, B00010100, B00011010, B00010010, B00010010, B00010010},
{B00000000, B00000000, B00000000, B00001100, B00010010, B00010010, B00010010, B00001100},
{B00000000, B00011100, B00010010, B00010010, B00011100, B00010000, B00010000, B00010000},
{B00000000, B00001110, B00010010, B00010010, B00001110, B00000010, B00000010, B00000001},
{B00000000, B00000000, B00000000, B00001010, B00001100, B00001000, B00001000, B00001000},
{B00000000, B00000000, B00001110, B00010000, B00001000, B00000100, B00000010, B00011110},
{B00000000, B00010000, B00010000, B00011100, B00010000, B00010000, B00010000, B00001100},
{B00000000, B00000000, B00000000, B00010010, B00010010, B00010010, B00010010, B00001100},
{B00000000, B00000000, B00000000, B00010001, B00010001, B00010001, B00001010, B00000100},
{B00000000, B00000000, B00000000, B00010001, B00010001, B00010001, B00010101, B00001010},
{B00000000, B00000000, B00000000, B00010001, B00001010, B00000100, B00001010, B00010001},
{B00000000, B00000000, B00010001, B00001010, B00000100, B00001000, B00001000, B00010000},
{B00000000, B00000000, B00000000, B00011111, B00000010, B00000100, B00001000, B00011111},
```

```
{B00000010, B00000100, B00000100, B00000100, B00001000, B00000100, B00000100, B00000010},
{B00000100, B00000100, B00000100, B00000100, B00000100, B00000100, B00000100, B00000100},
{B00001000, B00000100, B00000100, B00000100, B00000010, B00000100, B00000100, B00001000},
{B00000000, B00000000, B00000000, B00001010, B00011110, B00010100, B00000000, B00000000}
};

void clearDisplay() {
  for (byte x=0; x<8; x++) {
    buffer[x] = B00000000;
  }
  screenUpdate();
}

void initMAX7219() {
  pinMode(DataPin, OUTPUT);
  pinMode(LoadPin, OUTPUT);
  pinMode(ClockPin, OUTPUT);
  clearDisplay();
  writeData(B00001011, B00000111);    // limite de varredura definido entre 0 e 7
  writeData(B00001001, B00000000);    // modo de decodificação desligado
  writeData(B00001100, B00000001);    // define o registrador de desligamento (shutdown) para a
                                      // operação normal
  intensity(15);               // Apenas valores de 0 a 15 (4 bits)
}

void intensity(int intensity) {
  writeData(B00001010, intensity);    // B00001010 é o Registrador de Intensidade (Intensity Register)
}

void writeData(byte MSB, byte LSB) {
  byte mask;
  digitalWrite(LoadPin, LOW);           // deixa o LoadPin pronto para receber dados
  // Envia o bit mais significativo (most significant byte, ou MSB)
  for (mask = B10000000; mask > 0; mask >>= 1) { // itera, percorrendo a máscara de bits
    digitalWrite(ClockPin, LOW);
    if (MSB & mask){             // se o E bit a bit for verdadeiro
      digitalWrite(DataPin,HIGH);       // envia 1
    }
    else{                       // se o E bit a bit for falso
      digitalWrite(DataPin,LOW);        // envia 0
    }
    digitalWrite(ClockPin, HIGH);         // clock no estado alto, os dados entram
  }
```

```
  // Envie o bit menos significativo para os dados
  for (mask = B10000000; mask>0; mask >>= 1) {   // itera, percorrendo a máscara de bits
    digitalWrite(ClockPin, LOW);
    if (LSB & mask) {                // se o E bit a bit for verdadeiro
      digitalWrite(DataPin,HIGH);    // envia 1
    }
    else {                       // se o E bit a bit for falso
      digitalWrite(DataPin,LOW);     // envia 0
    }
    digitalWrite(ClockPin, HIGH);    // clock no estado alto, os dados entram
  }
  digitalWrite(LoadPin, HIGH);   // trava os dados
  digitalWrite(ClockPin, LOW);
}


void scroll(char myString[], int speed) {
  byte firstChrRow, secondChrRow;
  byte ledOutput;
  byte chrPointer = 0;   // inicializa o ponteiro de posição da string
  byte Char1, Char2;     // os dois caracteres que serão exibidos
  byte scrollBit = 0;
  byte strLength = 0;
  unsigned long time;
  unsigned long counter;

  // Incrementa a contagem, até que o final da string seja alcançado
  while (myString[strLength]) {strLength++;}

  counter = millis();

  while (chrPointer < (strLength-1)) {
    time = millis();
    if (time > (counter + speed)) {
      Char1 = myString[chrPointer];
      Char2 = myString[chrPointer+1];
      for (byte y= 0; y<8; y++) {
        firstChrRow = pgm_read_byte(&font[Char1 - 32][y]);
        secondChrRow = (pgm_read_byte(&font[Char2 - 32][y])) << 1;
        ledOutput = (firstChrRow << scrollBit) | (secondChrRow >> (8 - scrollBit) );
        buffer[y] = ledOutput;
      }
      scrollBit++;
      if (scrollBit > 6) {
      scrollBit = 0;
```

```
        chrPointer++;
        }
        counter = millis();
      }
    }
  }

  void screenUpdate() {
    for (byte row = 0; row < 8; row++) {
      writeData(row+1, buffer[row]);
    }
  }

  void setup() {
    initMAX7219();
    Timer1.initialize(10000);   // inicializa timer1 e define o período de interrupção
    Timer1.attachInterrupt(screenUpdate);
  }

  void loop() {
    clearDisplay();
    scroll("   BEGINNING ARDUINO   ", 45);
    scroll("   Chapter 7 - LED Displays   ", 45);
    scroll("   HELLO WORLD!!!  :)   ", 45);
  }
```

Quando você fizer o upload do código, verá uma mensagem de rolagem horizontal no display.

Projeto 21 – Display LED de matriz de pontos – Mensagem com rolagem horizontal – Análise do hardware

Para facilitar a compreensão do código, você primeiro deve saber como funciona o chip MAX7219, por isso, analisaremos o hardware antes do código.

O MAX7219 opera de modo muito semelhante ao dos registradores de deslocamento que vimos antes, no sentido de que você faz a entrada de dados de modo serial, bit a bit. Um total de 16 bits deve ser carregado no dispositivo de cada vez. O chip é de fácil utilização, e usa apenas três pinos do Arduino. O pino digital 2 vai para o pino 1 do MAX, que é o pino de entrada de dados (Data In, ou DIN). O pino digital 3 vai para o pino 12 do MAX, o LOAD, e o pino digital 4 vai para o pino 13 do MAX, o clock (CLK). Consulte a figura 7.5 para a pinagem do MAX7219.

O pino LOAD é colocado no estado baixo, e o primeiro bit de dados é definido como HIGH ou LOW no pino DIN. O pino CLK é definido para oscilar entre LOW e HIGH.

No extremo ascendente do pulso do clock, o bit no pino DIN é deslocado para o registrador interno. Então, o pulso do clock cai para LOW, e o próximo bit de dados é definido no pino DIN antes que o processo se repita. Depois de todos os 16 bits de dados terem sido colocados no registrador, conforme o clock sobe e desce 16 vezes, o pino LOAD será finalmente definido como HIGH, o que travará os dados no registrador.

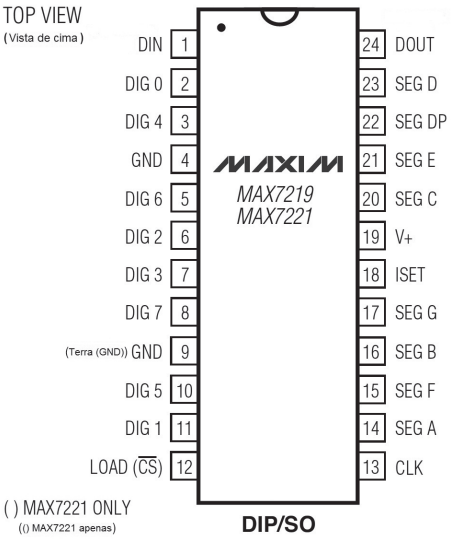


Figura 7.5 – Diagrama de pinos para o MAX7219.

A figura 7.6 é o diagrama de sincronismo do datasheet do MAX7219, e mostra como os três pinos são manipulados para enviar os bits de dados, de D0 a D15, para o dispositivo. O pino DOUT (pino 24) não é utilizado neste projeto. Porém, se você tivesse encadeado mais um chip MAX7219, o DOUT do primeiro chip estaria conectado ao DIN do segundo, e assim por diante. Dados saem do pino DOUT durante o extremo descendente do ciclo do clock.

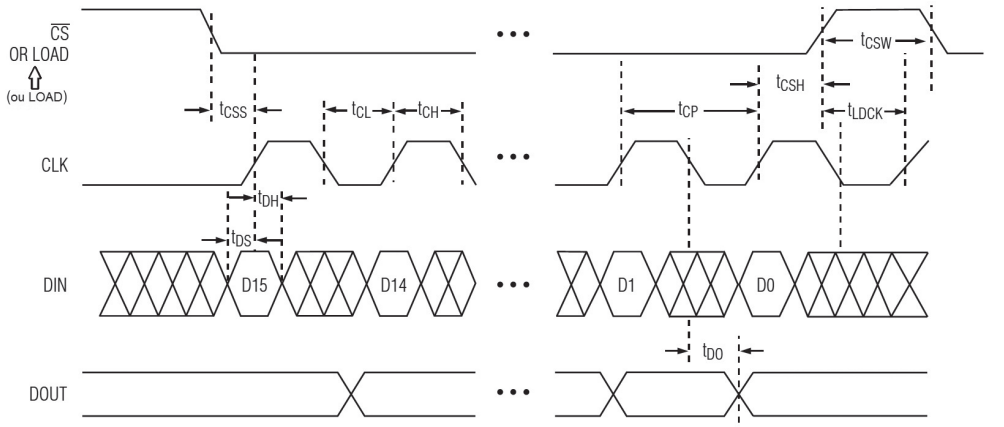


Figura 7.6 – Diagrama de tempo para o MAX7219.

Você tem de recriar essa sequência de sincronismo em seu código, para conseguir enviar os códigos apropriados ao chip. O chip pode receber uma corrente de até 100 mA, mais do que suficiente para a maioria dos displays de matriz de pontos. Caso você queira ler o datasheet do MAX7219, pode fazer seu download no site da Maxim, em *http://datasheets.maxim-ic.com/en/ds/MAX7219-MAX7221.pdf*.

O dispositivo aceita dados em 16 bits. O D15, ou bit mais significativo (*most significant bit*, ou MSB), é enviado primeiro, por isso a ordem decresce de D15 para D0, o bit menos significativo (*least significant bit*, ou LSB). Os primeiros quatro bits são bits "don't care" (sem importância), ou seja, bits que não serão utilizados pelo CI, por isso podem ser qualquer coisa. Os próximos quatro bits representam o endereço do registrador, e os oito bits finais representam os dados. A tabela 7.4 mostra o formato dos dados seriais, e a tabela 7.5 mostra o mapa de endereço dos registradores.

Tabela 7.4 – Formato de dados seriais (16 bits) do MAX7219

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	X	ENDEREÇO				MSB			DADOS			LSB	
											X				
											X				
											X				
											X				
											X				
											X				
											X				

Tabela 7.5 – Mapa de endereço de registradores do MAX7219

Registro	Endereço					Código Hexadecimal
	D15-D12	D11	D10	D9	D8	
No-Op	X	0	0	0	0	0xX0
Dígito 0	X	0	0	0	1	0xX1
Dígito 1	X	0	0	1	0	0xX2
Dígito 2	X	0	0	1	1	0xX3
Dígito 3	X	0	1	0	0	0xX4
Dígito 4	X	0	1	0	1	0xX5
Dígito 5	X	0	1	1	0	0xX6
Dígito 6	X	0	1	1	1	0xX7
Dígito 7	X	1	0	0	0	0xX8
Modo de decodificação (Decode Mode)	X	1	0	0	1	0xX9
Intensidade (Intensity)	X	1	0	1	0	0xXA
Limite de varredura (Scan Limit)	X	1	0	1	1	0xXB
Desligamento (Shutdown)	X	1	1	0	0	0xXC
Teste do display (Display test)	X	1	1	1	1	0xXF

Por exemplo, como você pode ver pelo mapa de endereço dos registradores na tabela 7.5, o endereço para o registrador de intensidade é 1010 binário. O registrador de intensidade define o brilho do display com valores que vão do menos intenso, em 0, ao mais intenso, em 15 (B000 a B1111). Para definir a intensidade como 15 (máxima), você enviaria os 16 bits a seguir, com o bit mais significativo (o bit mais à esquerda) sendo enviado primeiro, e o bit menos significativo (o bit mais à direita) sendo enviado por último (ou seja, o número está na ordem inversa dos bits):

```
0000101000001111
```

Os quatro bits menos significativos dos primeiros oito bits têm o valor de B1010, endereço do registrador de intensidade. Os quatro bits mais significativos dos primeiros oito bits são bits "don't care", por isso você envia B0000. Os oito bits seguintes são os dados que estão sendo enviados ao registro. Nesse caso, você deseja enviar o valor B1111 ao registrador de intensidade. Os primeiros quatro bits são novamente do tipo "don't care", por isso você envia B0000. Enviando esses 16 bits ao dispositivo, você define a intensidade do display como máxima. O valor inteiro de 16 bits que você deseja enviar é B0001010000001111, mas como são enviados primeiro os MSBs (bits mais significativos) e depois os LSBs (bits menos significativos), o número é enviado em ordem inversa, nesse caso, B1111000000101000.

Outro endereço que você utilizará é o do limite de varredura (*scan limit*). Lembre-se de que o MAX7219 é projetado para trabalhar com displays LED de sete segmentos (Figura 7.7).

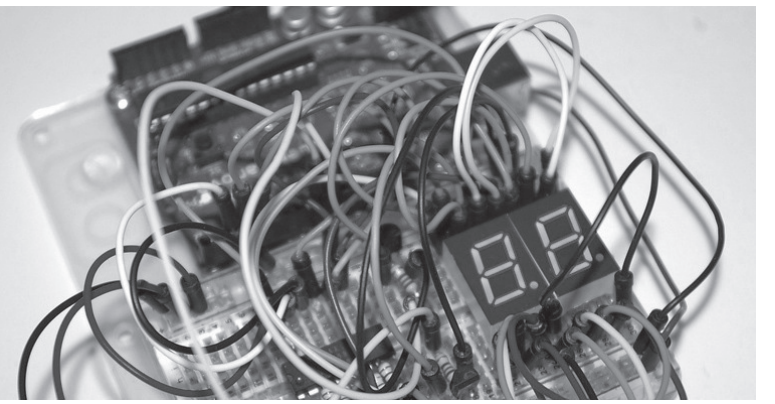


Figura 7.7 – Display LED de sete segmentos (imagem por Tony Jewell).

O limite de varredura decide quantos dos oito dígitos devem ser acesos. Em seu caso, você não está utilizando displays de sete segmentos, mas displays de matriz de pontos 8 x 8. Os dígitos correspondem às colunas em seu display. Você deseja que todas as oito colunas estejam sempre habilitadas, por isso o registrador do limite de varredura será definido como B00000111 (dígitos de 0 a 7, e 7 em binário é B111).

O registrador do modo de decodificação (*decode mode register*) é relevante apenas se você estiver utilizando displays de sete segmentos, por isso será definido como B00000000, para desligar a decodificação.

Por fim, você definirá o registro de desligamento (*shutdown register*) como B00000001, para garantir que ele esteja em operação normal, e não em modo de desligamento. Se você definir o registro de desligamento como B00000000, todas as fontes de alimentação serão direcionadas para o terra, deixando o display em branco.

Para mais informações sobre o CI MAX7219, leia seu datasheet. Procure as partes do texto relevantes ao seu projeto, e você verá que é muito mais fácil compreendê-las do que parece à primeira vista.

Agora que você (assim esperamos) compreende como o MAX7219 funciona, vamos analisar o código e descobrir como exibir o texto com rolagem horizontal.

Projeto 21 – Display LED de matriz de pontos – Mensagem com rolagem horizontal – Análise do código

Sua primeira ação no início do sketch é carregar as duas bibliotecas que serão utilizadas no código:

```
#include <avr/pgmspace.h>
#include <TimerOne.h>
```

A primeira biblioteca é a `pgmspace`, ou a biblioteca dos utilitários do Program Space, cujas funções permitem ao seu programa acessar dados armazenados em espaço de programa ou na memória flash. O Arduino com chip ATmega328 tem 32 kB de memória flash (2 kB são utilizados pelo bootloader, por isso 30 kB estão disponíveis). O Arduino Mega tem 128 kB de memória flash, 4 kB dos quais são utilizados pelo bootloader. O espaço de programa é exatamente o que indica seu nome: o espaço em que seu programa será armazenado. Você pode utilizar o espaço livre na memória flash empregando os utilitários do Program Space. É nela que você armazenará o extenso array bidimensional com a fonte de seus caracteres.

A segunda biblioteca é a `TimerOne`, utilizada pela primeira vez no projeto 19. Na sequência, declaramos os três pinos digitais que farão interface com o MAX7219:

```
int DataPin = 2;     // Pino 1 no MAX
int LoadPin = 3;     // Pino 12 no MAX
int ClockPin = 4;    // Pino 13 no MAX
```

Depois, você cria um array de tipo byte com oito elementos:

```
byte buffer[8];
```

Esse array armazenará o padrão de bits, que decidirá quais LEDs devem estar acesos ou apagados quando o display estiver ativo.

Depois, temos um extenso array bidimensional de tipo byte:

```
static byte font[][8] PROGMEM = {
// Apenas os caracteres ASCII que podem ser impressos (32-126)
{B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00000000, B00000000},
{B00000100, B00000100, B00000100, B00000100, B00000100, B00000100, B00000000, B00000100},
..etc
```

Esse array está armazenando o padrão de bits que forma a fonte utilizada para exibir o texto no display. Trata-se de um array bidimensional de tipo `static byte`. Depois da declaração do array, você também adicionou o comando `PROGMEM`, uma função dos utilitários do Program Space que diz ao compilador para armazenar esse array na memória flash, em vez de armazenar na SRAM (Static Random Access Memory).

SRAM é o espaço na memória do chip ATmega normalmente utilizado para armazenar as variáveis e strings de caracteres de seu sketch. Quando utilizados, esses dados são copiados do espaço de programa para a SRAM. Entretanto, o array utilizado para armazenar a fonte de texto é formado de 96 caracteres, compostos de oito bits cada. O array tem 96 x 8 elementos, o que corresponde a 768 elementos no total, e cada elemento é um byte (8 bits). A fonte, portanto, ocupa 768 bytes no total. O chip ATmega328 tem apenas 2 kB, ou aproximadamente 2.000 bytes de espaço na memória para variáveis. Quando você soma isso às outras variáveis e strings de texto utilizadas no programa, corre um sério risco de rapidamente ficar sem memória.

O Arduino não tem como lhe avisar de que a memória está acabando. Em vez disso, ele simplesmente para de funcionar. Para impedir que isso ocorra, você armazena esse array na memória flash, em vez de armazenar na SRAM, uma vez que ela tem muito mais espaço disponível. O sketch tem cerca de 2.800 bytes, e o array, pouco menos de 800 bytes, por isso você utilizará algo em torno de 3,6 kB dos 30 kB de memória flash disponíveis.

Em seguida, você cria as diversas funções que serão necessárias para o programa. A primeira simplesmente limpa o display. Quaisquer bits armazenados no array `buffer` serão exibidos na matriz. A função `clearDisplay()` simplesmente percorre todos os oito elementos do array e define seus valores como 0, para que nenhum LED esteja aceso e o display fique em branco. Depois, ela chama a função `screenUpdate()`, que exibe na matriz o padrão armazenado no array `buffer[]`. Nesse caso, como o buffer contém apenas zeros, nada será exibido.

```
void clearDisplay() {
  for (byte x=0; x<8; x++) {
    buffer[x] = B00000000;
  }
  screenUpdate();
}
```

A função seguinte, `initMAX7219()`, prepara o chip MAX7219 para uso. Primeiramente, os três pinos são definidos como `OUTPUT`:

```
void initMAX7219() {
  pinMode(DataPin, OUTPUT);
  pinMode(LoadPin, OUTPUT);
  pinMode(ClockPin, OUTPUT);
```

Depois, limpamos o display:

```
clearDisplay();
```

O limite de varredura é definido como `7` (em binário), o modo de decodificação é desligado e o registrador de desligamento é definido para a operação normal:

```
writeData(B00001011, B00000111);  // limite de varredura definido entre 0 e 7
writeData(B00001001, B00000000);  // modo de decodificação desligado
writeData(B00001100, B00000001);  // define o registrador de desligamento (shutdown) para a
                                  // operação normal
```

Então, a intensidade é definida como máxima, chamando a função `intensity()`:

```
intensity(15); // Apenas valores de 0 a 15 (4 bits)
```

Em seguida, temos a função `intensity()` em si, que simplesmente pega o valor transmitido a ela e escreve esse dado no registrador de intensidade, chamando a função `writeData()`:

```
void intensity(int intensity) {
  writeData(B00001010, intensity);  // B0001010 é o Registrador de Intensidade (Intensity Register)
}
```

A função seguinte realiza a maioria do trabalho pesado. Seu objetivo é escrever os dados no MAX7219, um bit de cada vez. A função exige dois parâmetros, ambos do tipo `byte`, representando o byte (e não bit) mais significativo e o byte menos significativo do número de 16 bits.

```
void writeData(byte MSB, byte LSB) {
```

Uma variável de tipo `byte`, chamada `mask`, é declarada:

```
byte mask;
```

Ela será utilizada como uma máscara de bits (conceito apresentado no projeto 17), para escolha do bit correto a ser enviado.

Na sequência, o `loadPin` é definido como `LOW`. Isso destrava os dados no registro do CI, deixando-o pronto para receber novos dados:

```
digitalWrite(LoadPin, LOW); // deixa o LoadPin pronto para receber dados
```

Agora, você deve enviar o byte mais significativo do número de 16 bits para o chip, com o bit mais à esquerda (mais significativo) sendo enviado primeiro. Para tanto, você utiliza dois conjuntos de loops `for`, um para o MSB e outro para o LSB. O loop utiliza uma máscara de bits para percorrer todos os oito bits. O uso de uma função bit a bit E (`&`) decide se o bit atual é `1` ou `0`, e define o `dataPin` de acordo, como `HIGH` ou `LOW`. O `clockPin` é definido como `LOW`, e o valor `HIGH` ou `LOW` é escrito no `dataPin`:

```
// Envia o bit mais significativo
for (mask = B10000000; mask>0; mask >>= 1) {   // itera, percorrendo a máscara de bits
  digitalWrite(ClockPin, LOW);
  if (MSB & mask) {                            // se o E bit a bit for verdadeiro
    digitalWrite(DataPin,HIGH);                // envia 1
  }
  else {                                       // se o E bit a bit for falso
    digitalWrite(DataPin,LOW);                 // envia 0
  }
  digitalWrite(ClockPin, HIGH);                // clock no estado alto, os dados entram
}
```

Por fim, o `loadPin` é definido como `HIGH`, para garantir que os 16 bits sejam travados no registro do chip, e o `clockPin` é definido como `LOW`, uma vez que o último pulso havia sido `HIGH` (o clock deve oscilar entre `HIGH` e `LOW`, para que os dados pulsem corretamente):

```
digitalWrite(LoadPin, HIGH);  // trava os dados
digitalWrite(ClockPin, LOW);
```

Na sequência, temos a função `scroll()`, que exibe no display os caracteres apropriados da string de texto. A função aceita dois parâmetros: o primeiro é a string que você deseja exibir, e o segundo, a velocidade na qual você deseja que a rolagem ocorra entre as atualizações, em milissegundos:

```
void scroll(char myString[], int speed) {
```

Depois, duas variáveis de tipo `byte` são preparadas; elas armazenarão uma das oito linhas de padrões de bits que compõem o caractere específico sendo exibido:

```
byte firstChrRow, secondChrRow;
```

Outra variável `byte` é declarada, `ledOutput`, que armazenará o resultado de um cálculo realizado sobre o primeiro e o segundo padrão de bits dos caracteres, e decidirá quais LEDs devem estar acesos ou apagados (isso será explicado em breve):

```
byte ledOutput;
```

Mais uma variável de tipo `byte` é declarada, `chrPointer`, e inicializada como 0. Ela armazenará a posição atual na string de texto que está sendo exibida, iniciando em 0 e incrementando até alcançar o comprimento da string:

```
byte chrPointer = 0; // Inicializa o ponteiro de posição da string
```

Outras duas variáveis `byte` são declaradas, que armazenarão o caractere atual e o seguinte na string:

```
byte Char1, Char2;  // os dois caracteres que serão exibidos
```

Essas variáveis são diferentes de `firstChrRow` e `secondChrRow`, pois armazenam o valor ASCII (American Standard Code for Information Interchange) do caractere atual, a ser exibido, e do seguinte na string, enquanto `firstChrRow` e `secondChrRow` armazenam o padrão de bits que forma as letras a serem exibidas.

Todas as letras, números, símbolos etc., que podem ser exibidos em uma tela de computador ou enviados via linha serial, têm um código ASCII, que é simplesmente um número de índice, indicando um caractere correspondente na tabela ASCII. Os caracteres de 0 a 31 são códigos de controle e não serão utilizados, uma vez que não podem ser exibidos em seu display de matriz de pontos. Você utilizará os caracteres ASCII de 32 a 126, os 95 caracteres que podem ser impressos, iniciando no número 32, que corresponde a um espaço, e indo até o 126, o símbolo de til (~). Os caracteres ASCII imprimíveis estão listados na tabela 7.6.

Tabela 7.6 – Caracteres ASCII imprimíveis

```
!"#$%&'()*+,-./0123456789:;ó?@
ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`
abcdefghijklmnopqrstuvwxyz{|}~
```

Outra variável byte é declarada e inicializada como 0. Ela armazenará a quantidade de bits, do padrão de caracteres do conjunto atual de letras, que devem ser deslocados para dar a impressão de que o texto está rolando da direita para a esquerda:

```
byte scrollBit = 0;
```

Mais uma variável `byte` armazenará o comprimento da string de caracteres. Ela é inicializada como 0:

```
byte strLength = 0;
```

Então, duas variáveis de tipo `unsigned long` são declaradas. Uma armazenará o tempo atual, em milissegundos, transcorrido desde que o chip do Arduino for inicializado ou reinicializado, e outra, o mesmo valor, só que dessa vez depois de executar uma rotina `while`. Quando tomadas em conjunto, essas variáveis garantem que os bits sejam deslocados apenas depois de transcorrido um intervalo de tempo especificado em milissegundos, para que a rolagem ocorra em uma velocidade que permita a leitura do texto:

```
unsigned long time;
unsigned long counter;
```

Agora, você tem de descobrir quantos caracteres há na string. Há muitas formas de fazê-lo, mas, em seu caso, você simplesmente prepara um loop `while`, que verifica se há dados no array atual de índice, `strLength` (inicializado como 0), e, se afirmativo,

incrementa a variável `strLength` em uma unidade. O loop então se repete até que a condição de `myString[strLength]` seja falsa, ou seja, quando não houver mais caracteres na string e `strLength`, incrementada em uma unidade a cada iteração, armazenar o comprimento da string:

```
while (myString[strLength]) {strLength++;}
```

Na sequência, você define o valor de `counter` como o valor de `millis()`. Você viu `millis()` no projeto 4, e sabe que ela armazena o valor, em milissegundos, do tempo transcorrido desde que o Arduino foi ligado ou reinicializado:

```
counter = millis();
```

Agora, um loop `while` é executado, com a condição de que a posição atual do caractere seja menor do que o comprimento da string, menos um:

```
while (chrPointer < (strLength-1)) {
```

A variável `time` é definida com o valor atual de `millis()`:

```
time = millis();
```

Então, uma instrução `if` verifica se o tempo atual é maior do que o último tempo armazenado, mais o valor em `speed`, ou seja, 45 milissegundos, e, se afirmativo, executa seu bloco de código:

```
if (time > (counter + speed)) {
```

`Char1` é carregada com o valor do caractere ASCII indicado em `chrPointer` no array `myString`, e `Char2` com o valor seguinte:

```
Char1 = myString[chrPointer];
Char2 = myString[chrPointer+1];
```

Agora, um loop `for` faz a iteração, percorrendo cada uma das oito linhas:

```
for (byte y= 0; y<8; y++) {
```

Em seguida, você lê o array `font` e coloca o padrão de bits da linha atual, de um total de oito, em `firstChrRow`, e a segunda linha em `secondChrRow`. Lembre-se de que o array `font` está armazenando os padrões de bits que formam os caracteres na tabela ASCII, mas apenas os referentes aos caracteres que podem ser impressos, de 32 a 126. O primeiro elemento do array é o código ASCII do caractere (menos 32, uma vez que você não está utilizando os caracteres de 0 a 31), e o segundo elemento armazena as oito linhas de padrões de bits que compõem esse caractere. Por exemplo, as letras `A` e `Z` são os caracteres ASCII 65 e 90, respectivamente. Você subtrai desses números um valor de 32, para obter o índice de seu array.

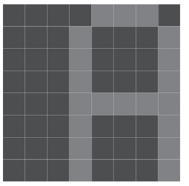
Assim, a letra `A`, código ASCII 65, é armazenada no elemento 33 do array (65 - 32), e a segunda dimensão do array naquele índice armazena os oito padrões de bits que formam essa letra. A letra `Z`, código ASCII 90, é o número de índice 58 no array. Os dados em `font[33][0...8]`, para a letra `A`, correspondem a:

```
{B00001110, B00010001, B00010001, B00010001, B00011111, B00010001, B00010001, B00010001},
```

Se você posicionar esses dados individualmente, para analisá-los mais claramente, terá como resultado:

```
B00001110
B00010001
B00010001
B00010001
B00011111
B00010001
B00010001
B00010001
```

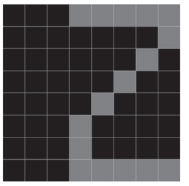
Se analisar mais de perto, você verá o padrão seguinte, que forma a letra `A`:



Para a letra `Z`, os dados no array são:

```
B00011111
B00000001
B00000010
B00000100
B00001000
B00010000
B00010000
B00011111
```

O que corresponde ao seguinte padrão de bits no LED:



Para ler esse padrão de bits, você tem de acessar a fonte de texto, armazenada no espaço de programa, e não na SRAM, como costuma ocorrer. Para tanto, você usa um dos utilitários da biblioteca pgmspace, `pgm_read_byte`:

```
firstChrRow = pgm_read_byte(&font[Char1 - 32][y]);
secondChrRow = (pgm_read_byte(&font[Char2 - 32][y])) << 1;
```

Quando você acessa o espaço de programa, obtém dados armazenados na memória flash. Para fazê-lo, você deve saber o endereço na memória em que os dados estão armazenados (cada local de armazenamento na memória tem um número de endereço único).

Para isso, você utiliza o símbolo `&` à frente de uma variável. Quando o faz, não lê os dados nessa variável, mas, sim, o endereço em que eles estão armazenados. O comando `pgm_read_byte` tem de saber o endereço na memória flash dos dados que você deseja recuperar, por isso você coloca um símbolo `&` à frente de `font[Char1 - 32][y]`, criando `pgm_read_byte(&font[Char1 - 32][y])`. Isso significa, simplesmente, que você lê o byte no espaço de programa armazenado no endereço de `font[Char1 - 32][y]`.

O valor de `secondChrRow` é deslocado uma posição para a esquerda, simplesmente para fazer com que o intervalo entre as letras seja menor, dessa forma facilitando sua leitura no display. Isso ocorre porque, em todos os caracteres, há bits não utilizados à esquerda por três espaços. Você poderia utilizar um deslocamento de duas posições para aproximá-las ainda mais, mas isso dificultaria a leitura.

A linha seguinte carrega o padrão de bits, da linha relevante, em `ledOutput`:

```
ledOutput = (firstChrRow << scrollBit) | (secondChrRow >> (8 - scrollBit) );
```

Como você deseja que as letras tenham rolagem horizontal da direita para a esquerda, você desloca para a esquerda a primeira letra, pela quantidade de vezes em `scrollBit`, e a segunda letra para a direita, por `8 - scrollBit` vezes. Então você aplica um OU lógico aos resultados, para mesclá-los no padrão de 8 bits necessário para exibição. Por exemplo, se as letras exibidas fossem `A` e `Z`, então os padrões para ambas seriam:

```
B00001110 B00011111
B00010001 B00000001
B00010001 B00000010
B00010001 B00000100
B00011111 B00001000
B00010001 B00010000
B00010001 B00010000
B00010001 B00011111
```

Assim, o cálculo anterior na primeira linha, quando `scrollBit` está definida como `5` (ou seja, quando as letras rolaram cinco pixels para a esquerda), seria:

```
B11000000 B00000011
```

O que corresponde à primeira linha de `A` deslocada para a esquerda cinco vezes, e à primeira linha do `Z` deslocada para a direita três vezes (8 - 5). Você pode ver que o padrão da esquerda é o que você obtém ao deslocar a letra `A` cinco pixels para a esquerda, e o padrão da direita é o que você obteria se a letra `Z` rolasse para a direita por três pixels. O OU lógico, símbolo `|`, tem o efeito de mesclar esses dois padrões, criando:

```
B11000011
```

que corresponde ao resultado obtido se as letras `A` e `Z` estivessem lado a lado, e rolassem cinco pixels para a esquerda.

A linha seguinte carrega esse padrão de bits na linha apropriada do buffer de tela:

```
buffer[y] = ledOutput;
```

`scrollBit` é acrescida em uma unidade:

```
scrollBit++;
```

Então uma instrução `if` verifica se o valor de `scrollBit` atingiu `7`. Se afirmativo, ela o define novamente como 0 e eleva `chrPointer` em uma unidade, para que, da próxima vez que for chamada, a função exiba os dois próximos conjuntos de caracteres:

```
if (scrollBit > 6) {
  scrollBit = 0;
  chrPointer++;
}
```

Por fim, o valor de `counter` é atualizado com o valor mais recente de `millis()`:

```
counter = millis();
```

A função `screenUpdate()` simplesmente toma as oito linhas de padrões de bits que você carregou nos oito elementos do array de buffer, e escreve esses dados no chip; o chip, por sua vez, exibe o resultado na matriz:

```
void screenUpdate() {
  for (byte row = 0; row < 8; row++) {
    writeData(row+1, buffer[row]);
  }
}
```

Depois de preparar essas seis funções, você finalmente atinge as funções `setup()` e `loop()` do programa. Na primeira, o chip é inicializado chamando `initMAX7219()`, um timer é criado e definido com um período de atualização de 10 mil microssegundos, e a função `screenUpdate()` é anexada. Assim como antes, isso garante que `screenUpdate()` seja ativada a cada 10 mil microssegundos, independentemente do que mais estiver ocorrendo.

```
void setup() {
  initMAX7219();
  Timer1.initialize(10000);  // inicializa timer1 e define o período de interrupção
  Timer1.attachInterrupt(screenUpdate);
}
```

Por fim, o loop principal do programa tem apenas quatro linhas. A primeira limpa o display, e as três seguintes chamam a rotina de rolagem, para exibir as três linhas de texto e fazê-las rolar horizontalmente no display.

```
void loop() {
  clearDisplay();
  scroll("   BEGINNING ARDUINO   ", 45);
  scroll("   Chapter 7 - LED Displays   ", 45);
  scroll("   HELLO WORLD!!!  :)   ", 45);
}
```

É evidente que você pode alterar o texto no código, fazendo com que o display apresente a mensagem que você quiser. O projeto 21 foi bem complexo no que se refere ao seu código. Como eu disse no início, todo esse trabalho poderia ter sido evitado se você tivesse, simplesmente, utilizado uma das bibliotecas preexistentes para matriz de LEDs, disponíveis em domínio público. Mas, ao fazê-lo, você não teria aprendido como funciona o chip MAX7219, nem como controlá-lo. Essas habilidades podem ser empregadas em praticamente qualquer outro CI externo, uma vez que os princípios envolvidos são muito semelhantes.

No próximo projeto, você utilizará essas bibliotecas, e verá como elas facilitam sua vida. Vamos pegar seu display e nos divertir um pouco com ele.

Projeto 22 – Display de matriz de pontos LED – Pong

O projeto 21 não foi fácil, e apresentou a você muitos conceitos novos. Assim, para o projeto 22, você criará um simples joguinho, utilizando o display de matriz de pontos e um potenciômetro. Dessa vez, você utilizará uma das muitas bibliotecas disponíveis para controle de displays de matriz de pontos LED, e verá como elas facilitam seu trabalho de codificação.

Componentes necessários

Os mesmo do projeto 21, mais:

Potenciômetro de 10 kΩ

Conectando os componentes

Deixe o circuito como no projeto 21 e adicione um potenciômetro. Os pinos da esquerda e da direita vão para o terra e para os +5 V, respectivamente, e o pino central vai para o pino analógico 5.

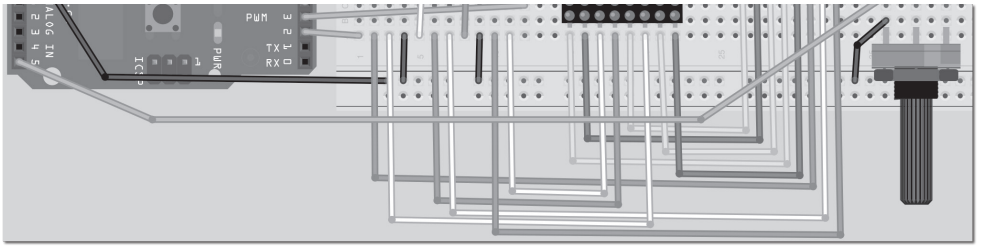


Figura 7.8 – Adicione um potenciômetro ao circuito do projeto 21 (consulte o site da Novatec para versão colorida).

Upload do código

Faça o upload do código da listagem 7.4. Quando o programa for executado, uma bola iniciará em um ponto aleatório na esquerda e avançará para a direita. Utilizando o potenciômetro, você controla a raquete para rebater a bola de volta à parede. Conforme o tempo passa, a velocidade da bola aumenta cada vez mais, até que você não consegue mais acompanhá-la.

Quando a bola ultrapassar a raquete, a tela piscará e o jogo reiniciará. Veja por quanto tempo você consegue acompanhar a bola, até que o jogo reinicie.

Listagem 7.4 – Código para o projeto 22

```
//Projeto 22
#include "LedControl.h"

LedControl myMatrix = LedControl(2, 4, 3, 1);         // cria uma instância de uma Matriz

int column = 1, row = random(8)+1;     // decide em que ponto a bola deve iniciar
int directionX = 1, directionY = 1;      // certifica-se de que ela vai primeiro da esquerda para
                                         // a direita
int paddle1 = 5, paddle1Val;        // pino e valor do potenciômetro
int speed = 300;
int counter = 0, mult = 10;

void setup() {
  myMatrix.shutdown(0, false);   // habilita o display
  myMatrix.setIntensity(0, 8);   // define o brilho como médio
  myMatrix.clearDisplay(0);      // limpa o display
  randomSeed(analogRead(0));
}
```

```
void loop() {
  paddle1Val = analogRead(paddle1);
  paddle1Val = map(paddle1Val, 200, 1024, 1,6);
  column += directionX;
  row += directionY;
  if (column == 6 && directionX == 1 && (paddle1Val == row || paddle1Val+1 == row ||
    paddle1Val+2 == row)) {directionX = -1;}
  if (column == 0 && directionX == -1 ) {directionX = 1;}
  if (row == 7 && directionY == 1 ) {directionY = -1;}
  if (row == 0 && directionY == -1 ) {directionY = 1;}
  if (column == 7) { oops(); }
  myMatrix.clearDisplay(0);  // limpa a tela para o próximo quadro de animação
  myMatrix.setLed(0, column, row, HIGH);
  myMatrix.setLed(0, 7, paddle1Val, HIGH);
  myMatrix.setLed(0, 7, paddle1Val+1, HIGH);
  myMatrix.setLed(0, 7, paddle1Val+2, HIGH);
  if (!(counter % mult)) {speed -= 5; mult * mult;}
  delay(speed);
  counter++;
}

void oops() {
  for (int x=0; x<3; x++) {
  myMatrix.clearDisplay(0);
  delay(250);
    for (int y=0; y<8; y++) {
      myMatrix.setRow(0, y, 255);
    }
    delay(250);
  }
  counter=0;        // reinicia todos os valores
  speed=300;
  column=1;
  row = random(8)+1;   // escolhe uma nova posição inicial
}
```

Projeto 22 – Display de matriz de pontos LED – Pong – Análise do código

O código para o projeto 22 é muito simples. Afinal, estamos nos recuperando do trabalho que tivemos no projeto 21.

Primeiro, inclua a biblioteca `LedControl.h` em seu sketch. Como antes, você terá de baixá-la e instalá-la na pasta `libraries`. A biblioteca, assim como informações adicionais, pode ser encontrada em *www.arduino.cc/playground/Main/LedControl*.

```
#include "LedControl.h"
```

Depois, crie uma instância de um objeto `LedControl`, da seguinte maneira:

```
LedControl myMatrix = LedControl(2, 4, 3, 1);          // cria uma instância de uma Matriz
```

Isso cria um objeto `LedControl`, `myMatrix`. O objeto `LedControl` requer quatro parâmetros. Os primeiros três são números de pinos para o MAX7219, na ordem Data In, Clock e Load. O número final é o número do chip (caso você esteja controlando mais de um MAX7219 e mais de um display).

Depois, você decide em qual coluna e linha a bola iniciará. A linha é escolhida utilizando um número aleatório.

```
int column = 1, row = random(8)+1;   // decide em que ponto a bola deve iniciar
```

Agora, dois inteiros são declarados, para decidir a direção em que a bola viajará. Se o número for positivo, ela avançará da esquerda para a direita e da base para o topo; se negativo, ela fará o caminho inverso.

```
int directionX = 1, directionY = 1;    // certifica-se de que ela vai primeiro da esquerda para
                                       // a direita
```

Você, então, decide qual pino está sendo utilizado para a raquete (o potenciômetro), e declara um inteiro para armazenar o valor lido no pino analógico:

```
int paddle1 = 5, paddle1Val;    // pino e valor do potenciômetro
```

A velocidade da bola é declarada em milissegundos:

```
int speed = 300;
```

Depois você declara e inicializa um contador como `0`, e seu multiplicador como `10`:

```
int counter = 0, mult = 10;
```

A função `setup()` habilita o display, assegurando que o modo de economia de energia esteja definido como `false`. A intensidade é definida como média e limpamos o display, deixando-o preparado para receber o jogo. Antes de você iniciar, `randomSeed` é definida com um valor aleatório, lido a partir de um pino analógico não utilizado.

```
void setup() {
  myMatrix.shutdown(0, false);   // habilite o display
  myMatrix.setIntensity(0, 8);   // Defina o brilho como médio
  myMatrix.clearDisplay(0);      // limpe o display
  randomSeed(analogRead(0));
}
```

No loop principal, você inicia lendo o valor analógico da raquete:

```
paddle1Val = analogRead(paddle1);
```

Então, esses valores são mapeados entre `1` e `6`:

```
paddle1Val = map(paddle1Val, 200, 1024, 1,6);
```

O comando `map` requer cinco parâmetros. O primeiro é o número a ser mapeado. Depois, temos os valores mínimo e máximo do número, e os valores menor e maior aos quais você deseja mapear. Em seu caso, você está tomando o valor em `paddle1Val`, que é a voltagem lida no pino analógico 5. Esse valor vai de `0`, em 0 V, a `1024` em 5 V. Você deseja que esses números sejam mapeados para o intervalo de 1 a 6, uma vez que essas são as linhas em que a raquete será exibida quando desenhada no display.

As coordenadas de coluna e linha são aumentadas pelos valores em `directionX` e `directionY`:

```
column += directionX;
row += directionY;
```

Agora, você tem de decidir se a bola atingiu uma parede ou a raquete e, se afirmativo, deve rebatê-la (a exceção sendo quando ela ultrapassa a raquete). A primeira instrução `if` verifica se a bola atingiu a raquete. Isso é feito decidindo se a coluna da bola está na coluna 6 e (E lógico, `&&`) se ela também está avançando da esquerda para a direita:

```
if (column == 6 && directionX == 1 && (paddle1Val == row || paddle1Val+1 == row
  || paddle1Val+2 == row)) {directionX = -1;}
```

Há três condições que devem ser atendidas para que a direção da bola se altere. A primeira é a de que a coluna seja 6, a segunda, de que a direção seja positiva (ou seja, da esquerda para a direita), e a terceira, de que a bola esteja na mesma linha de qualquer um dos três pontos que formam a raquete. Isso é feito aninhando um conjunto de comandos OU lógico (`||`) dentro de parênteses. O resultado desse cálculo é verificado primeiro, e depois adicionado às três operações `&&` no primeiro conjunto de parênteses.

Os três conjuntos de instruções `if`, a seguir, verificam se a bola atingiu as paredes do topo, da base ou do lado esquerdo, e, se afirmativo, invertem a direção da bola:

```
if (column == 0 && directionX == -1 ) {directionX = 1;}
if (row == 7 && directionY == 1 ) {directionY = -1;}
if (row == 0 && directionY == -1 ) {directionY = 1;}
```

Por fim, se a bola está na coluna 7, obviamente não atingiu a raquete, mas a ultrapassou. Se esse for o caso, chame a função `oops()` para piscar o display e reinicializar os valores:

```
if (column == 7) { oops(); }
```

Em seguida, limpamos o display para apagar pontos prévios:

```
myMatrix.clearDisplay(0);  // limpa a tela para o próximo frame de animação
```

A bola é desenhada na localização da coluna e da linha. Isso é feito com o comando `.setLed` da biblioteca LedControl:

```
myMatrix.setLed(0, column, row, HIGH);
```

O comando `.setLed` requer quatro parâmetros. O primeiro é o endereço do display. Depois, temos as coordenadas x e y (ou de coluna e linha) e, finalmente, um `HIGH` ou `LOW` para estados de ligado e desligado. Esses parâmetros são utilizados para desenhar os três pontos que formam a raquete na coluna 7 e na linha de valor `paddle1Val` (somando um além dela, e mais um depois disso).

```
myMatrix.setLed(0, 7, paddle1Val, HIGH);
myMatrix.setLed(0, 7, paddle1Val+1, HIGH);
myMatrix.setLed(0, 7, paddle1Val+2, HIGH);
```

Então, você verifica se o módulo de `counter % mult` não (NÃO lógico, `!`) é verdadeiro e, caso isso ocorra, diminui a velocidade em cinco, e multiplica o multiplicador por ele mesmo. Módulo é o resto obtido quando você divide um inteiro por outro. Em seu caso, você divide `counter` por `mult` e verifica se o resto é um número inteiro ou não. Isso basicamente garante que a velocidade apenas aumente depois de um intervalo de tempo definido, e que o tempo se eleve em proporção ao delay decrescente.

```
if (!(counter % mult)) {speed -= 5; mult * mult;}
```

Uma espera em milissegundos com o valor de `speed` é ativada, e o valor de `counter`, acrescido em uma unidade:

```
delay(speed);
counter++;
}
```

Por fim, a função `oops()` provoca um loop `for` dentro de outro loop `for` para limpar o display, e depois preenche todas as linhas repetidamente, com um delay de 250 milissegundos entre cada operação. Isso faz com que todos os LEDs pisquem, acendendo e apagando para indicar que a bola saiu de jogo e que o jogo está prestes a reiniciar. Então, todos os valores de `counter`, `speed` e `column` são definidos novamente em suas posições de início, e um novo valor aleatório é escolhido para `row`.

```
void oops() {
  for (int x=0; x<3; x++) {
  myMatrix.clearDisplay(0);
  delay(250);
    for (int y=0; y<8; y++) {
      myMatrix.setRow(0, y, 255);
    }
    delay(250);
  }
  counter=0;        // reinicia todos os valores
  speed=300;
  column=1;
  row = random(8)+1;   // escolhe uma nova posição de início
}
```

O comando `.setRow` opera transmitindo o endereço do display, o valor da linha e o padrão binário que indica quais dos LEDs devem acender e apagar. Nesse caso, você deseja todos acesos, o que, em binário, é representado por 11111111, e em decimal, por 255.

O propósito do projeto 22 foi mostrar como é mais fácil controlar um chip controlador de LEDs se você empregar uma biblioteca de código projetada para o chip. No projeto 21 você escolheu o caminho mais difícil, codificando todos os elementos a partir do zero; no projeto 22, o trabalho pesado foi todo feito nos bastidores. Há outras bibliotecas de matriz disponíveis; de fato, o próprio IDE do Arduino vem com uma, a `matrix`. Como tive melhores resultados com a biblioteca `LedControl.h`, preferi utilizá-la. Utilize a biblioteca que melhor atender às suas necessidades.

No próximo capítulo, você verá um tipo diferente de matriz de pontos, o LCD.

EXERCÍCIO

Tome os conceitos dos projetos 21 e 22 e combine-os. Crie um jogo Pong, mas faça com que o código mantenha uma pontuação (determinada pelos milissegundos transcorridos desde que a partida iniciou). Quando a bola sair de jogo, utilize a função de texto com rolagem horizontal para mostrar a pontuação da partida que acabou de terminar (os milissegundos que o jogador sobreviveu), e a pontuação mais alta até então.

Resumo

O capítulo 7 apresentou alguns tópicos bem complexos, incluindo o uso de CIs externos. Você ainda não completou nem a metade dos projetos, e já sabe como controlar um display de matriz de pontos utilizando tanto registradores de deslocamento quanto um CI controlador de LED dedicado. Assim, também, você primeiro aprendeu a codificar suas criações da forma mais difícil, para, em seguida, criar código com facilidade, incorporando uma biblioteca projetada especificamente para seu CI controlador de LED.

Você também aprendeu o conceito, por vezes desconcertante, da multiplexação, uma habilidade que será de grande utilidade em muitas outras situações, além do uso em displays de matriz de pontos.

Assuntos e conceitos abordados no capítulo 7:

- como conectar um display de matriz de pontos;
- como instalar uma biblioteca externa;
- o conceito da multiplexação (multiplexing, ou muxing);
- como utilizar a multiplexação para acender 64 LEDs individualmente, utilizando apenas 16 pinos de saída;

- o conceito básico de timers;
- como utilizar a biblioteca TimerOne para ativar códigos específicos, independentemente do que mais estiver acontecendo;
- como incluir bibliotecas externas em seu código utilizando a diretiva `#include`;
- como utilizar números binários para armazenar imagens LED;
- como inverter um número binário utilizando um NÃO bit a bit (~);
- como aproveitar a persistência da visão para enganar os olhos;
- como armazenar quadros de animação em arrays multidimensionais;
- como declarar e inicializar um array multidimensional;
- como acessar dados em um elemento específico de um array bidimensional;
- como realizar uma rotação bit a bit (ou deslocamento circular);
- como controlar displays de matriz de pontos LED utilizando registradores de deslocamento;
- como controlar displays de matriz de pontos LED utilizando CIs MAX7219;
- como coordenar corretamente os pulsos para entrada e saída de dados em CIs externos;
- como armazenar uma fonte de caracteres em um array bidimensional;
- como utilizar os registros do MAX7219;
- como ir além dos limites da SRAM e armazenar dados no espaço de programa;
- como inverter a ordem dos bits;
- como fazer a rolagem de textos e de outros símbolos em um display de matriz de pontos;
- o conceito da tabela de caracteres ASCII;
- como escolher um caractere a partir de seu código ASCII;
- como descobrir o tamanho de uma string de texto;
- como escrever e ler dados no espaço de programa;
- como obter o endereço na memória de uma variável utilizando o símbolo `&`;
- como utilizar a biblioteca `LedControl.h` para controlar LEDs individuais e linhas de LEDs;
- como utilizar operadores lógicos;
- como facilitar sua vida, e desenvolver código mais rapidamente, utilizando bibliotecas de código.