Capítulo 10

Memória EEPROM no Arduino Mega 2560

A memória EEPROM interna é uma poderosa ferramenta de armazenamento de dados no Arduino. Essa memória garante o armazenamento permanente de dados, mesmo na falta de energia, o que garante continuidade de operação em sistemas que precisam voltar a operar automaticamente após o restabelecimento da energia.

10.1 Memória EEPROM interna

Em alguns casos, existe a necessidade do armazenamento permanente de dados referentes ao processo que se deseja controlar, mesmo com a possibilidade de falta de energia. Por exemplo, em uma fechadura eletrônica codificada, ao se cadastrar uma senha, deseja-se que esta não seja "esquecida" pelo sistema no caso de uma falta de energia. Isso garante que, após o restabelecimento da alimentação do microcontrolador, o sistema volte a operar normalmente, tendo como base a senha previamente cadastrada. Essa senha pode ser armazenada localmente em uma memória não volátil, cuja melhor opção para implementação dessa tarefa se dá a partir do uso de uma memória EEPROM.

O Arduino Mega 2560 possui uma memória EEPROM interna com capacidade de 4 kB que podem ser usados livremente pelo projetista para armazenamento não volátil de dados. Internamente, a EEPROM é dividia em bytes, podendo cada byte ser acessado individualmente a partir do seu endereço.

No Arduino, para escrever um dado em uma posição de memória qualquer, usa-se o método `EEPROM.write(endereço, dado)`, onde o parâmetro endereço, do tipo inteiro sem sinal, corresponde ao endereço de memória no qual se deseja escrever algo. Como a memória possui 4 kB de capacidade, a faixa endereçável inicia em 0 e vai até 4095. O parâmetro dado é um byte que corresponde à informação que se deseja armazenar no endereço especificado.

O *datasheet* do microcontrolador ATmega 2560 especifica que um ciclo de escrita em uma posição da EEPROM dura aproximadamente 3,3 ms e essa memória apresenta uma vida útil de aproximadamente 100.000 ciclos de escrita/apagamento. Nesse contexto, o método EEPROM.update(endereço, dado) se apresenta como uma forma alternativa de escrita na memória EEPROM. Esse método escreve o dado no endereço especificado apenas se a informação for diferente daquela previamente armazenada. Isso pode exigir um maior tempo para o processo de escrita na memória, pois nesse método é previamente realizada uma leitura do valor armazenado que é então comparado com o valor que se deseja escrever. Entretanto, isso traz como vantagem a economia de ciclos de escrita na memória, aumentando seu tempo de vida útil.

Para ler uma palavra em uma posição de memória qualquer, usa-se o método EEPROM.read(endereço), onde endereço correspondendo ao endereço de memória no qual se deseja ler o seu conteúdo.

Para exemplificar o uso da memória EEPROM interna, consideremos o circuito apresentado na figura 1. Nesse circuito, por simplicidade foram omitidos os componentes e as conexões que ajustam o contraste do display. A montagem possui 6 chaves do tipo NA, sendo duas superiores e 4 inferiores, conectadas nos pinos de 8 a 13 do Arduino UNO. As 4 chaves inferiores são usadas como entrada para uma senha de 4 bits, as outras duas realizam a operação de escrita (chave esquerda – pino 8) e leitura (chave direita – pino 9) na memória EEPROM interna. Quando a chave superior esquerda é pressionada, o estado das chaves inferiores é lido e armazenado na posição zero da memória EEPROM. Quando a chave superior direita é pressionada, o código de 4 bits referente ao estado das chaves, gravado na memória, é exibido no LCD em formato hexadecimal. O programa que implementa essas ações é mostrado no quadro seguinte.

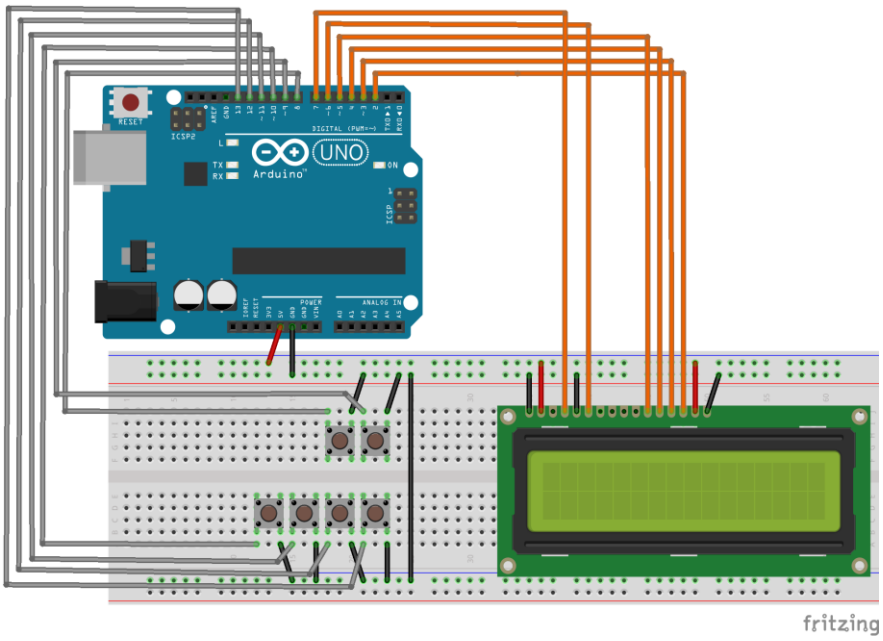

Figura 1 – Circuito para teste do programa com memória EEPROM interna.

```
#include <LiquidCrystal.h>
#include <EEPROM.h>
#define  rs  7
#define  en  6
#define  d4  5
#define  d5  4
#define  d6  3
#define  d7  2
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

void setup(){
  lcd.begin(16, 2);
  pinMode(8, INPUT_PULLUP);
  pinMode(9, INPUT_PULLUP);
  pinMode(10, INPUT_PULLUP);
  pinMode(11, INPUT_PULLUP);
  pinMode(12, INPUT_PULLUP);
  pinMode(13, INPUT_PULLUP);
  EEPROM.write(0,0);
}

void loop(){
  if(!digitalRead(9))
  {
    lcd.clear();
    lcd.print(EEPROM.read(0), HEX);
    delay(500);
  }
  if(!digitalRead(8))
  {
    byte senha=0;
    for(byte contador=0; contador<=3; ++contador)
    {
      if(!digitalRead(13-contador))
        bitSet(senha, contador);
    }
    EEPROM.write(0, senha);
    delay(500);
  }
}
```

Como apresentado no programa, para usar as funções de manipulação da memória EEPROM é necessária a inclusão da biblioteca <EEPROM.h>.

A função setup( ) inicializa o LCD, configura os pinos das chaves como entradas com resistores de pullup e escrevem no endereço 0 o valor inicial como sendo zero. Na função loop( ), o programa testa continuamente se alguma das chaves conectadas aos pinos 8 e 9 foi pressionada.

A primeira estrutura **if** testa se a chave do pino 9 foi pressionada. Se a chave tiver sido pressionada, o bloco de código limpa o LCD pelo uso do método `lcd.clear( )`, além de imprimir o valor lido do endereço zero da EEPROM em formato hexadecimal pelo método `lcd.print(EEPROM.read(0), HEX)`. Após isso, é inserido um delay de 500 ms.

A segunda estrutura **if** testa se a chave do pino 8 foi pressionada. Se a chave tiver sido pressionada, o bloco de código inicialmente declara a variável senha e a inicializa com o valor zero. A estrutura **for** subsequente declara e inicializa com zero a variável contador e executa o bloco de código interno enquanto essa variável for menor ou igual a 3. Essa estrutura lê sequencialmente os pinos 13, 12, 11 e 10, onde estão ligadas as chaves codificadoras da senha, e seta, ou não, os 4 bits menos significativos da variável contador com a função interna bitSet( ), conforme a chave tenha sido pressionada ou não. Em seguida, a variável é armazenada na posição zero da EEPROM. Um delay de 500 ms também é inserido ao final da estrutura **if**.

Nas duas estruturas **if**, o delay de 500 ms foi inserido a fim de evitar que as estruturas sejam continuamente repetidas enquanto o botão estiver pressionado. Dessa forma, esse tempo de 500 ms apenas produz um atraso suficiente para que o usuário possa soltar o botão antes que a estrutura seja executada novamente a partir das chamadas da função loop( ).

10.2 Armazenamento de outros tipos de dados primitivos na memória EEPROM interna

A memória EEPROM pode ser usada para armazenar qualquer tipo de dado, não apenas do tipo byte. O operador EEPROM[ ] pode ser usado para referenciar a memória EEPROM como um array. Assim, por exemplo, ao se referir a EEPROM[0], estamos nos referindo ao byte armazenado no endereço zero da memória.

A estrutura de dados array é usada para conter dados do mesmo tipo. Dados de tipos diferentes também podem ser agregados em tipos especiais chamados estruturas (**struct**). Primeiro, o tipo estrutura é declarado (é necessário especificar que tipos de variáveis serão combinados dentro da estrutura), e então variáveis deste novo tipo podem ser definidas (de maneira similar à que usamos para definir variáveis do tipo int ou char).

Uma declaração de estrutura declara um tipo **struct**. Cada tipo struct recebe um nome (ou *tag*). Refere-se àquele tipo pelo *nome* precedido da palavra struct. Cada unidade de dados na estrutura é chamada *membro* e possui um *nome de membro*. Os membros de uma estrutura podem ser de qualquer tipo primitivo. Declarações de estrutura não são definições. Não é alocada memória, simplesmente é introduzido um novo tipo de estrutura.

Geralmente declarações de estruturas são globais, assim elas são visíveis por todas as funções (embora isto dependa de como a estrutura está sendo usada).

A forma padrão de declaração de uma estrutura é:

```
struct nome-estrutura{
  declaração dos membros
} definição de variáveis (opcional);
```

Abaixo é apresentado um exemplo de um tipo struct que contém um membro do tipo int e um outro membro do tipo char:

```
struct My_structure{
  int num;
  char ch;
};
```

Esta declaração cria um novo tipo de dado estruturado, chamado **My_structure**, que contém um inteiro chamado num e um caractere chamado ch. Como acontece com qualquer outro tipo de dado, variáveis de tipos estruturados são definidas fornecendo o nome do tipo e o nome da variável. Considere a definição abaixo relativa a uma variável com o nome Estrutura1 que é do tipo My_structure:

```
My_structure Estrutura1;
```

Com essa definição, memória suficiente será alocada para guardar um int e um char (nesta ordem). Como qualquer outra variável, Estrutura1 tem um nome, um tipo, e um endereço associados.

Variáveis estruturadas armazenam também valores, e como outras variáveis locais, se elas não têm atribuídas um valor específico, seu valor é indefinido. É possível definir variáveis durante a declaração do tipo estrutura, como em:

```
struct My_structure{
  int num;
  char ch;
} Estrutura1;
```

Nomes de membros (tais como num e ch) podem ser usados como nomes de outras variáveis independentes (fora do tipo estrutura definido), ou como nomes de membros em outros tipos de estrutura.

Dada uma variável de estrutura, um membro específico é referenciado usando o nome da variável seguida de . (ponto) e pelo nome do membro da estrutura, como no exemplo a seguir:

`Estrutura1.num` se refere ao membro com nome num na estrutura Estrutura1;
`Estrutura1.ch` se refere ao membro com nome ch na estrutura Estrutura1.

O exemplo abaixo mostra alguns exemplos do uso de membros de estrutura:

```
Estrutura1.ch = 'G';
Estrutura1.num = 42;

Estrutura1.num++;

if(Estrutura1.ch == 'H'){
  lcd.print(Estrutura1.num);
}
```

Uma variável de estrutura pode ser tratada como um objeto simples no todo, com um valor específico associado a ela (a estrutura Estrutura1 tem um valor que agrega valores de todos os seus membros). Note a diferença com arrays: se arr[ ] é um array de tamanho 2 definido como int arr[2] = {0,1};, o nome arr2 não se refere ao valor coletivo de todos os elementos do array. Na verdade, arr2 é um ponteiro constante e se refere ao endereço de memória onde o array se inicia. Variáveis de estrutura são diferentes.

De todas as variações de atribuição (incluindo o incremento e decremento) atribuição de estruturas pode ser usada apenas com o operador (=). O uso de outros operadores de atribuição ou de incremento causará um erro de compilação. A atribuição de um valor de estrutura para outro copia todos os membros de uma estrutura para outra. Mesmo que um dos membros seja um array ou outra estrutura, ela é copiada integralmente. As duas estruturas envolvidas na atribuição devem ser do mesmo tipo struct. Considere o seguinte exemplo:

```
struct My_struture{
  int num;
  char ch;
};

My_structure Estrutura1, Estrutura2;

Estrutura1.num = 3;
Estrutura1.ch = 'C';

Estrutura1 = Estrutura2;
```

Como em outros tipos, variáveis de estruturas podem ser inicializadas ao serem declaradas. Esta inicialização é análoga ao que é feito no caso de arrays. O exemplo abaixo ilustra a inicialização de estruturas:

```
struct My_struture{
  int num;
  char ch;
} Estrutura1 = {3, 'C'};
```

Os valores de inicialização devem estar na mesma ordem dos membros na declaração da estrutura.

Ainda é possível ter estruturas como argumentos de função e valores de retorno, arrays de estruturas, e estruturas aninhadas.

Para escrever um dado na memória EEPROM, cujo tamanho e tipo sejam diferentes de um byte, usa-se o método EEPROM.put(endereço, dado). O parâmetro endereço, nesse caso, é o endereço do primeiro byte do dado que se deseja armazenar na EEPROM, e o parâmetro dado pode ser qualquer tipo primitivo, como inteiro, float ou uma estrutura. Este método usa, internamente, o método EEPROM.update( ), assim ele não escreve o dado na EEPROM se o dado já armazenado for igual ao que se deseja gravar.

Para ler um dado na memória EEPROM, cujo tamanho e tipo sejam diferentes de um byte, usa-se o método EEPROM.get(endereço, dado). O parâmetro endereço, nesse caso, é o endereço do primeiro byte do dado que se deseja ler da EEPROM, e o parâmetro dado pode ser qualquer tipo primitivo, como inteiro, float ou uma estrutura onde será armazenado a leitura feita. O tamanho em bytes da variável dado, nesse caso, corresponde à quantidade de bytes que serão lidos da EEPROM.

O programa mostrado a seguir faz uso do operador EEPROM[ ] e dos métodos EEPROM.put( ) e EEPROM.get( ). O programa armazena 6 diferentes tipos de dados na memória EEPROM e apresenta o seu conteúdo no LCD a partir do pressionamento de uma das 6 chaves da montagem da figura 1. No quadro abaixo é mostrado o cabeçalho inicial do programa e a função setup( ). O quadro da página seguinte apresenta a função loop( ).

```
#include <LiquidCrystal.h>
#include <EEPROM.h>
#define  rs  7
#define  en  6
#define  d4  5
#define  d5  4
#define  d6  3
#define  d7  2
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

struct My_structure{
  int numero;
  char letra;
};

void setup(){
  lcd.begin(16, 2);
  pinMode(8, INPUT_PULLUP);
  pinMode(9, INPUT_PULLUP);
  pinMode(10, INPUT_PULLUP);
  pinMode(11, INPUT_PULLUP);
  pinMode(12, INPUT_PULLUP);
  pinMode(13, INPUT_PULLUP);
  EEPROM[0]=0;

  My_structure Estrutura[5];
  for(byte contador=0; contador < 5; ++contador)
  {
    Estrutura[contador] = {contador+1, 'A'+contador};
    EEPROM.put(1+contador*sizeof(Estrutura[contador]), Estrutura[contador]);
  }
}
```

No cabeçalho há a declaração da estrutura My_structure que contém dois membros, sendo um inteiro e um caractere.

Na função setup( ), além da inicialização do LCD e da configuração dos pinos de entrada, é armazenado o valor zero no endereço zero da memória EEPROM com o uso do operador EEPROM[ ] .

Em seguida, há a definição de uma array de 5 elementos do tipo My_structure, que recebe o nome Estrutura, o qual é inicializado e armazenado sequencialmente na memória EEPROM no laço **for** seguinte. Nesse laço, é criada a variável de controle, contador, que recebe o valor zero na sua inicialização. No primeiro comando do laço, os elementos do array são indexados e endereçados pela variável de controle, onde é feita a inicialização dos seus membros. O primeiro membro, do tipo inteiro, recebe o valor contador+1, e o segundo membro, do tipo char, recebe o caractere correspondente à operação 'A'+contador. Isso garante que, com o incremento da variável de controle do laço, os membros inteiros dos elementos do array serão inicializados sequencialmente com os valores 1, 2, 3, 4, e 5 e os membros char serão inicializados com os caracteres 'A', 'B', 'C', 'D' e 'E'.

No segundo comando do laço for, isto é, o método EEPROM.put( ), é armazenado na memória EEPROM, sequencialmente e em ordem crescente, os elementos do array. São necessários 3 bytes para armazenar cada elemento do array, sendo 2 bytes do membro inteiro e um do membro char. Para isso, a memória é endereçada pelo primeiro parâmetro do método, 1+contador*sizeof(Estrutura[contador]). O valor resultante dessa expressão é sempre o endereço inicial de escrita do dado, uma vez que a função sizeof( ) retorna o tamanho em bytes necessários para armazenar um elemento do array de estruturas.

Por exemplo, quando o contador é igual a zero, o resultado da expressão é igual a 1, que é o endereço inicial para armazenamento do primeiro elemento do array, já que o endereço zero foi inicialmente usado para armazenar o valor zero durante a utilização do operador EEPROM[ ]. Quando o contador é igual a 1, o resultado da expressão é igual a 1+1*3 = 4. O valor 4 é portanto o endereço inicial para armazenamento do segundo elemento do array. Isso ocorre sequencialmente até que os 5 elementos estejam armazenados, quando a função setup é então encerrada.

Na função loop( ), abaixo, as chaves são constantemente escaneadas de forma sequencial. A variável de controle do laço **for** é responsável pelo escaneamento e leitura nas chaves conectadas aos pinos de 8 a 13.

Se a chave conectada no pino 8 for pressionada, o bloco de códigos do **if** mais interno é executado. Esse bloco limpa o conteúdo exibido pelo display, seta o cursor no primeiro caractere da primeira linha, de onde exibe a sequência de caracteres “dado 1:”, seta o caractere no primeiro caractere da segunda linha, de onde exibe o valor lido da posição zero da memória EEPROM.

Se alguma outra chave for pressionada, o bloco **else** é executado. Esse bloco define a variável Estrutura do tipo My_structure, faz a leitura de 3 bytes, correspondente ao tamanho da variável Estrutura, a partir do endereço definido pela expressão contida no primeiro parâmetro do método EEPROM.get, e armazena o resultado nessa variável. Em seguida, limpa o conteúdo exibido pelo cursor e imprime na primeira linha do LCD a sequência “Dado X”, onde X é um número de 2 a 6, dependendo de qual chave foi pressionada. Finalmente, é impresso na segunda linha do LCD os valores dos dois membros da estrutura armazenada, separados por um caractere de espaço. ■

```
void loop()
{
  for(byte contador=0; contador < 6; ++contador)
  {
    if(!digitalRead(8+contador))
    {
      if(!contador)
      {
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("Dado 1:");
        lcd.setCursor(0,1);
        lcd.print(EEPROM[0]);
      }
      else
      {
        My_structure Estrutura;
        EEPROM.get(1+(contador-1)*sizeof(Estrutura), Estrutura);
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("Dado ");
        lcd.print(contador+1);
        lcd.print(":");
        lcd.setCursor(0,1);
        lcd.print(Estrutura.numero);
        lcd.print(" ");
        lcd.print(Estrutura.letra);
      }
      delay(1000);
    }
  }
}
```