**Biblioteca para a SRAM (23LC1024) e SPI para a Caixa Preta**

01/05/2020, Versão 1.0

SPI operando em 8 MHz.

Sram\_zera() gasta 694 mseg.

São duas memórias de 128 KB que vão trabalhar como uma única de 256 KB.

O acesso precisa ficar limitado a uma página de 128 KB.

A partir do endereço, a função decide qual memória acessar.

Funções

|  |  |  |
| --- | --- | --- |
|  |  |  |
| void | sram\_flash | (void) |
| void | sram\_op\_dados | (void) |
| void | sram\_op\_mostra | (void) |
| void | sram\_dump | (long adr, long qtd) |
| void | sram\_zera\_mpu\_gps | (void) |
| int | sram\_rd\_32b | (long adr) |
| void | sram\_wr\_32b | (long adr, int dado) |
| int | sram\_rd\_16b | (long adr) |
| void | sram\_wr\_16b | (long adr, int dado) |
| void | sram\_rd\_str | (long adr, byte \*msg, word qtd) |
| void | sram\_wr\_str | (long adr, byte \*msg) |
| void | sram\_zera | (void) |
| void | sram\_wr\_blk | (long adr, byte \*vet, word qtd) |
| void | sram\_rd\_blk | (long adr, byte \*vet, word qtd) |
| void | sram\_wr | (long adr, byte dado) |
| byte | sram\_rd | (long adr) |
| void | sram\_modo\_wr | (byte qual, byte dado) |
| byte | sram\_modo\_rd | (byte qual) |
| void | spi\_cs0 | (void) |
| void | spi\_CS0 | (void) |
| void | spi\_cs1 | (void) |
| void | spi\_CS1 | (void) |
| void | spi\_cs2 | (void) |
| void | spi\_CS2 | (void) |
| byte | spi\_transf | (byte dado) |
| void | spi\_config | (byte clk){ |
|  |  |  |
|  |  |  |

* void **sram\_flash** (void)

Copiar toda a SRAM para a FLASH. Gasta 9,4 seg.

Deveria gastar 6,144 seg (2.048 pag x 3ms = 6,144 seg) + 3 seg em SPI e TWI

// #define FLASH\_PAG 128

// #define GPS\_ADR\_FIM 0x40000L //Fim área GPS

* void **sram\_op\_dados** (void)

Mostrar toda a configuração ao ligar o carro na SRAM.

Não imprime qualquer tipo de rótulo, só os dados. Ver exemplo no final.

* void **sram\_op\_mostra** (void)

Mostrar toda a configuração ao ligar o carro na SRAM

Imprime os dados rotulados. Ver exemplo no final.

* void **sram\_dump** (long adr, long qtd)

Dump da SRAM. Mostra qtd bytes a partir do endereço adr.

Arredonda “adr” para o múltiplo de 16 inferior (evita problema com página de 128K) (adr &= 0xFFFFFF)

Usa a função ser\_dump\_memo (long adr, char \*vet) para mostrar uma linha.

* void **sram\_zera\_mpu\_gps** (void)

Zerar somente área usada pelo MPU e GPS. Salva Calibração ao ligar, zera e depois a reescreve.

* long **sram\_rd\_32b** (long adr)

Ler um valor de 32 bits da SRAM, Big Endian.

* void **sram\_wr\_32b** (long adr, long dado)

Escrever um valor de 32 bits na SRAM, Big Endian.

* int **sram\_rd\_16b** (long adr)

Ler um valor inteiro de 16 bits da SRAM. Big Endian.

* void **sram\_wr\_16b** (long adr, int dado)

Escrever um valor inteiro de 16 bits da SRAM. Big Endian.

* void **sram\_rd\_str** (long adr, byte \*msg, word qtd)

Ler uma string a partir do endereço adr da SRAM. Copia o zero final.

Qtd indica a quantidade máxima, e caso se chegue a esse valor, o último byte é o zero final.

* void **sram\_wr\_str** (long adr, byte \*msg)

Gravar uma string na SRAM, incluindo o zero final.

* void **sram\_zera** (void)

Zerar toda a SRAM. Com SCLK = 8 MHz, gasta 694 mseg.

* void **sram\_wr\_blk** (long adr, byte \*vet, word qtd)

Escrever uma sequência de qtd bytes a partir do endereço adr.

Faz a transição de 0x1FFFF para 0x20000.

\*\*🡺 Não faz a volta de 0x3FFFF para 0x00000 (fica na pag 1: volta para 0x20000).

Se adr = 0x0 0000 -> 0x1 FFFF, SRAM0 #CS0.

Se adr = 0x2 0000 -> 0x3 FFFF, SRAM1 #CS1.

* void **sram\_rd\_blk** (long adr, byte \*vet, word qtd)

Ler uma sequência de qtd bytes e guardar no vetor vet.

Faz a transição de 0x1FFFF para 0x20000.

\*\*🡺 Não faz a volta de 0x3FFFF para 0x00000 (fica na pag 1: volta para 0x20000).

Se adr = 0x0 0000 -> 0x1 FFFF, SRAM0 #CS0.

Se adr = 0x2 0000 -> 0x3 FFFF, SRAM1 #CS1.

* void **sram\_wr** (long adr, byte dado)

Escrever o dado no endereço adr da SRAM.

Se adr = 0x0 0000 -> 0x1 FFFF, SRAM0 #CS0.

Se adr = 0x2 0000 -> 0x3 FFFF, SRAM1 #CS1.

* byte **sram\_rd** (long adr)

Ler o endereço adr da SRAM e retornar o dado lido.

Se adr = 0x0 0000 -> 0x1 FFFF, SRAM0 #CS0.

Se adr = 0x2 0000 -> 0x3 FFFF, SRAM1 #CS1.

* void **sram\_modo\_wr** (byte qual, byte dado)

Escrever no registrador de modo.

Se qual=0 --> SRAM0 #CS0 (0x0 0000 -> 0x1 FFFF).

Se qual=1 --> SRAM1 #CS1 (0x2 0000 -> 0x3 FFFF).

* byte **sram\_modo\_rd** (byte qual)

Ler o registrador de modo.

Se qual=0 --> SRAM0 #CS0 (0x0 0000 -> 0x1 FFFF).

Se qual=1 --> SRAM1 #CS1 (0x2 0000 -> 0x3 FFFF).

* void **spi\_cs0** (void)

CS0 (PL0) = LOW, SRAM 0 (23LC1024)

* void **spi\_CS0** (void)

CS0 (PL0) = HIGH SRAM 0 (23LC1024)

* void **spi\_cs1** (void)

CS1 (PL1) = LOW, SRAM 0 (23LC1024)

* void **spi\_CS1** (void)

CS1 (PL1) = HIGH, SRAM 0 (23LC1024)

* void **spi\_cs2** (void)

CS2 (PL2) = LOW, SRAM 2 (W25Q64)

* void **spi\_CS2** (void)

CS2 (PL2) = HIGH, SRAM 2 (W25Q64)

* byte **SPI\_transf** (byte dado)

Enviar um Byte pela porta SPI.

Rotina envia e recebe um Byte ao mesmo tempo.

* void **spi\_config** (byte clk)

Inicializar porta SPI do Mega

#define FLASH\_PAG 128 //Tamanho da página para gravação

Exemplo de void **sram\_op\_mostra** (void)

--- SRAM: Dados da Configuração ao Ligar o carro ---

Caixa Preta Não Acidentada.

Sef Test = NOK.

Calibracao de Fabrica = 8 medidas para Calibrar ao Ligar

Calibra Escala Acel = +/- 2g

Calibra Escala Giro = +/- 250 gr/s

Calibracao (ax-ay-az-tp-gx-gy-gz): +05103 +00500 +10773 +00969 +02997 +02895 +02559

Freq de Amostragem = 100 Hz =

Opera Escala Acel = +/- 8g

Opera Escala Giro = +/- 2000 gr/s

Limiares de disparo (ax-ay-az-gx-gy-gz): 4 4 4 1000 1000 1000

Disparo no endereco: 0x00000000

Quem Disparou: AX AY AZ GX GY GZ

N N N N N N

Data do acidente: dd/mm/yy

Hora do acidente: hh:mm:ss

--- SRAM: Fim dos Dados da Configuração ao Ligar o carro ---

Exemplo de void **sram\_op\_dados** (void)

20046

20046

00083

8

0

0

+05155 +00556 +10846 +01068 +03807 +03589 +03199

0

0

0 0 0

9

2

3

0

+00004 +00004 +00004 +01000 +01000 +01000

0

0

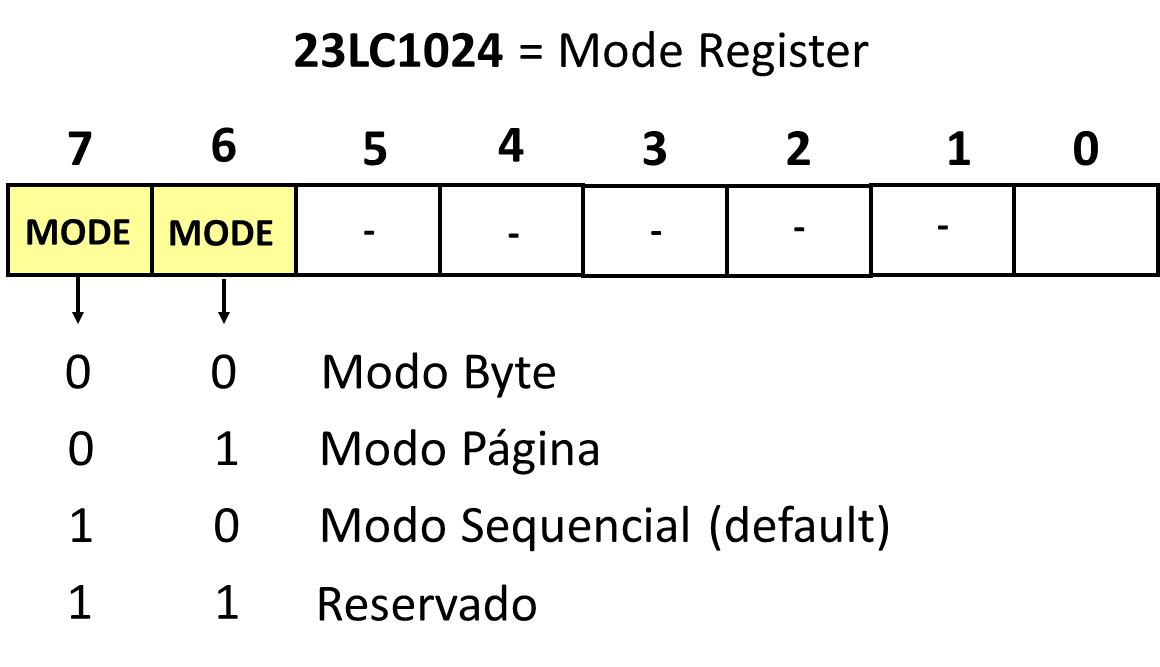
+20046 +20046 +20046 +20046 +20046 +20046

ddmmyy

hhmmss.sss

*Tabela 9.6. Conjunto de instruções para a memória 23LC1024*

|  |  |  |  |
| --- | --- | --- | --- |
| **Instrução** | **Formato** | **Hexa** | **Descrição** |
| READ | 0000 0011 | 0x03 | Ler a memória a partir do endereço selecionado |
| WRITE | 0000 0010 | 0x02 | Escrever na memória a partir do endereço selecionado |
| EDIO | 0011 1011 | 0x3B | Habilitar o acesso Dual |
| EQIO | 0011 1000 | 0x38 | Habilitar o acesso Quad |
| RSTIO | 1111 1111 | 0xFF | Reset do acesso Dual e Quad |
| RDMR | 0000 0101 | 0x05 | Ler o Registrador de Modo |
| WRMR | 0000 0001 | 0x01 | Escrever no Registrador de Modo |



*Figura 9.10. Registrador de Modo da memória 23LC1024.*

*Gabarito para ajudar na configuração dos registradores da porta SPI*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **SPCR** | SPIE | SPE | DORD | MSTR | CPOL | CPHA | SPR1 | SPR0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0,1 | 0,1 |
| **SPSR** | SPIF | WCOL | - | - | - | - | - | SPI2X |
| 0 | 0 | - | - | - | - | - | 0,1 |

Para modo mestre (MSTR) é preciso SS=1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SPI2X** | **SPR1** | **SPR0** | **Frequência SCK** | **Arduino fosc=16 MHz** |
| 0 | 0 | 0 | fosc/4 | 4 MHz |
| 0 | 0 | 1 | fosc/16 | 1 MHz |
| 0 | 1 | 0 | fosc/64 | 250 kHz |
| 0 | 1 | 1 | fosc/128 | 125 kHz |
| 1 | 0 | 0 | fosc/2 | 8 MHz |
| 1 | 0 | 1 | fosc/8 | 2 MHz |
| 1 | 1 | 0 | fosc/32 | 500 kHz |
| 1 | 1 | 1 | fosc/64 | 250 kHz |

// Velocidades SPI, verificar o dobrador (SPI2X)

#define SPI\_125K 0 //SCL=125KHz, SPI2X=0

#define SPI\_250K 1 //SCL=250KHz, SPI2X=0

#define SPI\_500K 2 //SCL=500KHz, SPI2X=1

#define SPI\_1M 3 //SCL=1MHz, SPI2X=0

#define SPI\_2M 4 //SCL=2MHz SPI2X=1

#define SPI\_4M 5 //SCL=4MHz SPI2X=0

#define SPI\_8M 6 //SCL=8MHz SPI2X=1

#define MISO 50 //Master Input

#define MOSI 51 //Master Output

#define SCK 52 //Saída do relógio

#define CS0 49 //(PL0) Controla o estado do #CS0

#define CS1 48 //(PL1) Controla o estado do #CS1

#define CS2 47 //(PL2) Controla o estado do #CS1

#define SS 53 //Controla o estado do #CS do SD Card