**Biblioteca do Projeto Arapuca - PUKA**

12/02/2022 - Passada à limpo.

Biblioteca LCD

Imprimir no LCD.

Funções

|  |  |  |
| --- | --- | --- |
| void | lcd\_crono | (unsigned long c) |
| void | lcd\_float | (float fx, char prec) |
|  |  |  |
| void | lcd\_dec32 | (long dt) |
| void | lcd\_dec32u | (unsigned long dt); |
| void | lcd\_dec32nz | (long dt) |
| void | lcd\_dec32unz | (unsigned long dt) |
| void | lcd\_hex32 | (long dt) |
|  |  |  |
| void | lcd\_dec16 | (int dt) |
| void | lcd\_dec16u | (unsigned int dt) |
| void | lcd\_dec16nz | (int dt) |
| void | lcd\_dec16unz | (unsigned int dt) |
| void | lcd\_hex16 | (unsigned int dt) |
|  |  |  |
| void | lcd\_dec8 | (char dt) |
| void | lcd\_dec8u | (char dt) |
| void | lcd\_dec8nz | (char dt) |
| void | lcd\_dec8unz | (char dt) |
| void | lcd\_hex8 | (char dt) |
|  |  |  |
| void | lcd\_str\_lc | (char lin, char col, char \*pt) |
| void | lcd\_str | (char \*pt) |
| void | lcd\_spc | (char c) |
| void | lcd\_cursor\_lc | (char lin, char col) |
| void | lcd\_cursor | (char x) |
|  |  |  |
| void | lcd\_char | (char c) |
| void | lcd\_char\_aux | (char x) |
| void | lcd\_cmdo | (char c) |
| void | lcd\_cmdo\_aux | (char x) |
|  |  |  |
| char | lcd\_ocupado | (void) |
| char | lcd\_status | (void) |
| void | lcd\_status\_prep | (char c1, char c2) |
| char | lcd\_status\_ler | (void) |
|  |  |  |
| void | lcd\_inic | (void) |
| void | lcd\_aux | (char dado) |
| void | pcf\_STT\_STP | (void) |
|  |  |  |
| void | lcd\_BL | (void) |
| void | lcd\_bl | (void) |
| void | lcd\_Bl | (void) |
|  |  |  |
| void | pcf\_write | (char dado) |
| int | pcf\_qual\_adr | (void) |
| int | pcf\_ack | (int adr) |
| void | delay | (long limite) |
| void | i2c\_config | (void) |
|  |  |  |
|  |  |  |
|  |  |  |

* void **lcd\_crono** (unsigned long c);

Imprimir leitura do cronometro.

* void **lcd\_float** (float f, char prec)

Escreve no LCD o float fx com prec casas após a vírgula e apresenta o sinal.

Formato = + xxx xxx xxx , ddd ddd ddd ddd (usar char msg[24])

Limite da parte inteira = 9 dígitos. Limite da parte fracionária = 12 dígitos.

Caso ultrapasse esses limites imprime ###, ###

O máximo é 999.999.999,999999. Se ultrapassar o máximo, escreve ###,###.

Na verdade, o máximo é 999.999.999,999967. Exemplos:

999.999.999,0 🡪 imprime 999.999.936,000000 (por causa da precisão da representação)

876.543.210,123456789 🡪 imprime 876543232,000000 (por causa da precisão da representação)

* void **lcd\_dec32** (long dt)

Escrever no LCD 32 bits em decimal, com sinal e com zeros à esquerda.

* void **lcd\_dec32u** (long dt)

Escrever no LCD 32 bits em decimal, sem sinal e com zeros à esquerda.

* void **lcd\_dec32nz** (long dt)

Escrever no LCD 32 bits em decimal, com sinal e sem zeros à esquerda.

* void **lcd\_dec32unz** (long dt)

Escrever no LCD 32 bits em decimal, sem sinal e sem zeros à esquerda.

* void **lcd\_hex32** (long dt)

Escrever no LCD 32 bits em hexadecimal.

* void **lcd\_dec16** (unsigned int dt)

Escrever no LCD 16 bits em decimal, com sinal e com zeros à esquerda.

* void **lcd\_dec16u** (unsigned int dt)

Escrever no LCD 16 bits em decimal, sem sinal e com zeros à esquerda.

* void **lcd\_dec16nz** (unsigned int dt)

Escrever no LCD 16 bits em decimal, com sinal e sem zeros à esquerda.

* void **lcd\_dec16unz** (unsigned int dt)

Escrever no LCD 16 bits em decimal, sem sinal e sem zeros à esquerda.

* void **lcd\_hex16** (unsigned int dt)

Escrever no LCD 16 bits em hexadecimal.

* void **lcd\_dec8** (char dt)

Escrever no LCD 8 bits em decimal, com sinal e com zeros à esquerda.

* void **lcd\_dec8u** (char dt)

Escrever no LCD 8 bits em decimal, sem sinal e com zeros à esquerda.

* void **lcd\_dec8nz** (char dt)

Escrever no LCD 8 bits em decimal, com sinal e sem zeros à esquerda.

* void **lcd\_dec8unz** (char dt)

Escrever no LCD 8 bits em decimal, sem sinal e sem zeros à esquerda.

* void **lcd\_hex8** (char dt)

Escrever no LCD 8 bits em hexadecimal.

* void **lcd\_str\_lc** (char lin, char col, char \*pt){

No LCD, escrever string terminada em zero na posição "lin" e "col"

* void **lcd\_str** (char \*pt)

No LCD escrever string terminada em zero na posição atual do cursor.

* void **lcd\_spc** (char qtd)

Escrever no LCD qtd de espaços. Máximo de 16 espaços.

* void **lcd\_cursor\_lc** (char lin, char col)

No LCD, posicionar cursor na posição linha=0,1, ... e coluna=0,1,2... .

É preciso definir NRC (qtd de colunas). Aqui não usa, mas é preciso definir NRL (qtd de linhas).

* void **lcd\_cursor** (char x)

No LCD, posicionar cursor no endereço x da RAM de Dados do LCD.

* void **lcd\_char** (char dt)

Escrever no LCD um char. Ignora o Busy.

* void **lcd\_char\_aux** (char x)

Pulsar o Enable, é auxiliar da função lcd\_char().

* void **lcd\_cmdo** (char dt)

Escrever um comando. Ignora o Busy.

* void **lcd\_cmdo\_aux** (char x)

Pulsar o Enable, é auxiliar da função lcd\_char().

* char **lcd\_ocupado** (void)

LCD Ocupado? Se ocupado retorna TRUE

* char **lcd\_status** (void)

Retorna o byte de STATUS do LCD.

* void **lcd\_status\_prep** (char c1, char c2)

Usado pela função lcd\_status().

Gera START e prepara PCF e LCD para a leitua - Não gera STOP.

c1 = 0xF0 | lcd\_lght | RS=0 | RW=1

c2 = 0xF0 | lcd\_lght | RS=0 | RW=1 | E=1

* char **lcd\_status\_ler** (void)

Usado pela função lcd\_status().

Faz a leitura de um nibble do LCD - Gera STOP

* void **lcd\_inic** (void)

Inicializar LCD no modo 4 bits.

* void **lcd\_aux** (char dado)

Auxiliar inicialização do LCD (RS=RW=0)

\*\*\* Só serve para a inicialização \*\*\*

* void **pcf\_STT\_STP** (void)

Gerar vários START e STOP para tentar colocar PCF em estado conhecido.

Só é usado na inicialização. É para prevenir algum galho.

* void **lcd\_BL** (void)

Ligar Back Light.

* void **lcd\_bl** (void)

Desligar Back Light.

* void **lcd\_Bl** (void)

Inverter Back Light.

* void **pcf\_write** (char dado)

Escrever dado em PCF.

* int **pcf\_qual\_adr** (void)

Descobrir endereço do PCF que aciona o LCD 0x3F ou 0x27.

Retorna -1 se não encontrou.

* int **pcf\_ack** (int adr)

Testar um endereço para ver se o PCF responde.

ACK --> retorna TRUE e NACK --> retorna FALSE.

* void **delay** (long limite)

Delay para usar logo após STOP

* void **i2c\_config** (void)

Configurar Pinos I2C - UCSB0

P3.0 --> SDA e P3.1 --> SCL

Leds\_chaves

Variáveis externas:

extern int **ps1**; 🡪 estado anterior da chave S1

extern int **ps2**; 🡪 estado anterior da chave S2

Funções para Leds e Chaves

|  |  |  |
| --- | --- | --- |
| int | chave\_s1 | (void) |
| int | chave\_s2 | (void) |
| void | chaves\_config | (void) |
| void | leds\_config | (void) |
| void | led\_VM | (void) |
| void | led\_vm | (void) |
| void | led\_Vm | (void) |
| void | led\_VD | (void) |
| void | led\_vd | (void) |
| void | led\_Vd | (void) |
| void | SCP1 | (void) |
| void | scp1 | (void) |
| void | Scp1 | (void) |
| void | SCP2 | (void) |
| void | scp2 | (void) |
| void | Scp2 | (void) |
| void | debounce | (unsigned int x); |
|  |  |  |
|  |  |  |

* int **chave\_s1** (void)

TRUE se S1 passou de aberta para fechada.

FALSE para os demais casos.

* int **chave\_s2** (void)

TRUE se S2 passou de aberta para fechada.

FALSE para os demais casos.

* void **chaves\_config** (void)

Configura as chaves para uso: S1(P2.1) e S0 (P1.1).

* void **leds\_config** (void)

Configura os leds e os pinos (scp) para osciloscópio.

LED1 (P1.0) = Vermelho e LED2 (P4.7) = Verde.

SCP1(P2.4) e SCP2(P2.5).

* void **led\_VM** (void)

Acende o led vermelho, led1, P1.0.

* void **led\_vm** (void)

Apaga o led vermelho, led1, P1.0.

* void **led\_Vm** (void)

Inverte o estado do led vermelho, led1, P1.0.

* void **led\_VD** (void)

Acende o led verde, led2, P4.7.

* void **led\_vd** (void)

Apaga o led verde, led2, P4.7.

* void **led\_Vd** (void)

Inverte o estado do led verde, led2, P4.7.

* void **SCP1** (void)

Coloca em nível alto o pino 1 (P2.4) para osciloscópio.

* void **scp1** (void)

Coloca em nível baixo o pino 1 (P2.4) para osciloscópio.

* void **Scp1** (void)

Inverte o estado do pino 1 (P2.4) para osciloscópio.

* void **SCP2** (void)

Coloca em nível alto o pino 2 (P2.5) para osciloscópio.

* void **scp2** (void)

Coloca em nível baixo o pino 2 (P2.5) para osciloscópio.

* void **Scp2** (void)

Inverte o estado do pino 2 (P2.5) para osciloscópio.

* void **debounce** (unsigned int x)

Gera um pequeno atraso. Usado no debounce.

Biblioteca Serial - Serve para o Bluetooth

Semelhante à biblioteca LCD

Funções

|  |  |  |
| --- | --- | --- |
| void | ser\_crono | (unsigned long c) |
|  |  |  |
| void | ser\_double | (double dx, char prec) |
| void | ser\_float | (float fx, char prec) |
|  |  |  |
| void | ser\_dec32 | (long dt) |
| void | ser\_dec32u | (unsigned long dt); |
| void | ser\_dec32nz | (long dt) |
| void | ser\_dec32unz | (unsigned long dt) |
| void | ser\_hex32 | (long dt) |
|  |  |  |
| void | ser\_dec16 | (int dt) |
| void | ser\_dec16u | (unsigned int dt) |
| void | ser\_dec16nz | (int dt) |
| void | ser\_dec16unz | (unsigned int dt) |
| void | ser\_hex16 | (unsigned int dt) |
|  |  |  |
| void | ser\_dec8 | (char dt) |
| void | ser\_dec8u | (char dt) |
| void | ser\_dec8nz | (char dt) |
| void | ser\_dec8unz | (char dt) |
| void | ser\_hex8 | (char dt) |
|  |  |  |
| void | ser\_crlf | (unsigned int qtd) |
| void | ser\_cr | (unsigned int qtd) |
| void | ser\_lf | (unsigned int qtd) |
| void | ser\_spc | (unsigned int qtd) |
| void | ser\_str | (char \*pt) |
| void | ser\_char | (char x) |
| void | ser\_config | (long brate) |
|  |  |  |
|  |  |  |

* void **ser\_double** (double dx, char prec){

Imprimir double = + xxx xxx xxx xxx xxx xxx , ddd ddd ddd ddd ddd ddd (40 posições)

18 posições = limite da parte inteira

18 posições = limite da parte fracionária

Caso ultrapasse os limites imprime ### , ###

* void **ser\_float** (float fx, char prec)

Escreve na Serial o float fx com prec casas após a vírgula e apresenta o sinal.

Formato = + xxx xxx xxx , ddd ddd ddd ddd (usar char msg[24])

Limite da parte inteira = 9 dígitos. Limite da parte fracionária = 12 dígitos.

Caso ultrapasse esses limites imprime ###, ###

O máximo é 999.999.999,999999. Se ultrapassar o máximo, escreve ###,###.

Na verdade, o máximo é 999.999.999,999967. Exemplos:

999.999.999,0 🡪 imprime 999.999.936,000000 (por causa da precisão da representação)

876.543.210,123456789 🡪 imprime 876543232,000000 (por causa da precisão da representação)

* void **ser\_dec32** (long dt)

Escrever na Serial 32 bits em decimal, com sinal e com zeros à esquerda.

* void **ser\_dec32u** (long dt)

Escrever na Serial 32 bits em decimal, sem sinal e com zeros à esquerda.

* void **ser\_dec32nz** (long dt)

Escrever na Serial 32 bits em decimal, com sinal e sem zeros à esquerda.

* void **ser\_dec32unz** (long dt)

Escrever na Serial 32 bits em decimal, sem sinal e sem zeros à esquerda.

* void **ser\_hex32** (long dt)

Escrever na Serial 32 bits em hexadecimal.

* void **ser\_dec16** (unsigned int dt)

Escrever na Serial 16 bits em decimal, com sinal e com zeros à esquerda.

* void **ser\_dec16u** (unsigned int dt)

Escrever na Serial 16 bits em decimal, sem sinal e com zeros à esquerda.

* void **ser\_dec16nz** (unsigned int dt)

Escrever na Serial 16 bits em decimal, com sinal e sem zeros à esquerda.

* void **ser\_dec16unz** (unsigned int dt)

Escrever na Serial 16 bits em decimal, sem sinal e sem zeros à esquerda.

* void **ser\_hex16** (unsigned int dt)

Escrever na Serial 16 bits em hexadecimal.

* void **ser\_dec8** (char dt)

Escrever na Serial 8 bits em decimal, com sinal e com zeros à esquerda.

* void **ser\_dec8u** (char dt)

Escrever na Serial 8 bits em decimal, sem sinal e com zeros à esquerda.

* void **ser\_dec8nz** (char dt)

Escrever na Serial 8 bits em decimal, com sinal e sem zeros à esquerda.

* void **ser\_dec8unz** (char dt)

Escrever na Serial 8 bits em decimal, sem sinal e sem zeros à esquerda.

* void **ser\_hex8** (char dt)

Escrever na Serial 8 bits em hexadecimal.

* void **ser\_crlf** (unsigned int qtd)

Escrever na Serial qtd de pares Carriage Return (0xD) e Line Feed (0xA).

* void **ser\_cr** (unsigned int qtd)

Escrever na Serial qtd de Carriage Return (0xD).

* void **ser\_lf** (unsigned int qtd)

Escrever na Serial qtd de Line Feed (0xA).

* void **ser\_spc** (unsigned int qtd)

Escrever na Serial qtd de espaços.

* void **ser\_str** (char \*pt)

Na Serial escrever string terminada em zero na posição atual do cursor.

* void **ser\_char** (char x)

Enviar um caracter, antes espera UCTXIFG = 1.

Pode prender execução

* void **ser\_config** (long brate)

Configurar USCI\_A0 com SMCLK = 20 MHz.

Pronto para 9.600, 38.400, 460.800 e 250.000 bps

P3.3 = TXD --> pino 5 conector Bluetooth.

P3.4 = RXD --> pino 4 conector Bluetooth.

Biblioteca Strings para LCD ou Serial

Gerar strings para facilitar a impressão no LCD, na porta serial, etc.

Funções

|  |  |  |
| --- | --- | --- |
| void | str\_crono | (unsigned long c, char \*v) |
| void | str\_rmvz\_u | (char \*msg) |
| void | str\_rmvz\_s | (char \*msg) |
|  |  |  |
| void | str\_double | (float d, char prec, char \*msg) |
| void | str\_float | (float f, char prec, char \*msg) |
|  |  |  |
| void | str\_dec32 | (long c, char \*msg) |
| void | str\_dec32u | (long c, char \*msg) |
| void | str\_hex32 | (long c, char \*msg) |
|  |  |  |
| void | str\_dec16u | (word c, char \*msg) |
| void | str\_dec16 | (int c, char \*msg) |
| void | str\_hex16 | (word c, char \*msg) |
|  |  |  |
| void | str\_dec8u | (char c, char \*msg) |
| void | str\_dec8 | (char c, char \*msg) |
| void | str\_hex8 | (char c, char \*msg) |
|  |  |  |
| void | str\_spc | (char qtd, char \*msg) |
| void | str\_crlf | (char qtd, char \*msg) |
| void | str\_cr | (char qtd, char \*msg) |
| void | str\_lf | (char qtd, char \*msg) |
|  |  |  |
|  |  |  |

* void **str\_crono** (unsigned long c, char \*v)

Gera string com o valor lido do cronômetro.

* void **str\_rmvz\_u** (char \*msg)

Remove os zeros à esquerda da string de número sem sinal que está em msg.

* void **str\_rmvz\_s** (char \*msg)

Remove os zeros à esquerda da string de número com sinal que está em msg.

* void **str\_double** (double d, char prec, char \*msg)

Imprimir double = + xxx xxx xxx xxx xxx xxx , ddd ddd ddd ddd ddd ddd (40 posições).

18 posições = limite da parte inteira.

18 posições = limite da parte fracionária.

Caso ultrapasse os limites imprime ### , ###.

* void **str\_float** (float f, char prec, char \*msg)

Escreve em msg o float fx com prec casas após a vírgula e apresenta o sinal.

Formato = + xxx xxx xxx , ddd ddd ddd ddd (usar char msg[24])

Limite da parte inteira = 9 dígitos.

Limite da parte fracionária = 12 dígitos.

Caso ultrapasse os limites imprime ###, ###

O máximo é 999.999.999,999999. Se ultrapassar o máximo, escreve ###,###.

Na verdade, o máximo é 999.999.999,999967. Exemplos:

999.999.999,0 🡪 imprime 999.999.936,000000 (por causa da precisão da representação)

876.543.210,123456789 🡪 imprime 876543232,000000 (por causa da precisão da representação)

* void **str\_dec32** (long c, char \*msg)

Escreve em msg o (long) decimal 32 bits com sinal e com zeros à esquerda.

Usar char msg[12], pois +4 294 967 295 \0 - 12 posições.

* void **str\_dec32u** (long c, char \*msg)

Escreve em msg o (unsigned long) decimal 32 bits sem sinal e com zeros à esquerda.

Usar char msg[12], pois +4 294 967 295 \0 - 12 posições.

* void **str\_hex32** (long c, char \*msg)

Escreve em msg o (long) hexadecimal de 32 bits. Usar char msg[9].

* void **str\_dec16** (int c, char \*msg)

Escreve em msg o (int) decimal 16 bits com sinal e com zeros à esquerda.

Usar char msg[7], pois +67 295 \0 - 7 posições.

* void **str\_dec16u** (word c, char \*msg)

Escreve em msg o (word) decimal 16 bits sem sinal e com zeros à esquerda.

Usar char msg[7], pois +67 295 \0 - 7 posições.

* void **str\_hex16** (word c, char \*msg)

Escreve em msg o (word) hexadecimal de 16 bits. Usar char msg[5].

* void **str\_dec8** (char c, char \*msg)

Escreve em msg o (char) decimal 8 bits com sinal e com zeros à esquerda.

Usar char msg[5], pois +123 \0 - 5 posições.

* void **str\_dec8u** (char c, char \*msg)

Escreve em msg o (char) decimal 8 bits sem sinal e com zeros à esquerda.

Usar char msg[5], pois +295 \0 - 5 posições.

* void **str\_hex8** (char c, char \*msg)

Escreve em msg o (char) hexadecimal de 8 bits. Usar char msg[3].

* void **str\_spc**(char qtd, char \*msg)

Escrever em msg uma qtd de espaços = 0x20 (Espaço em Branco). Prever msg com tamanho adequado.

* void **str\_crlf**(char qtd, char \*msg)

Escrever em msg uma qtd de pares CR (‘\r’=0xD) e LF (‘\n’=0xA). Prever msg com tamanho adequado.

* void **str\_cr**(char qtd, char \*msg)

Escrever em msg uma qtd de CR (‘\r’=0xD). Prever msg com tamanho adequado.

* void **str\_lf**(char qtd, char \*msg)

Escrever em msg uma qtd de LF (‘\n’=0xA). Prever msg com tamanho adequado.

Biblioteca Timer e também cronômetro

Funções

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| unsigned long | crono\_ler | (void) |
| void | crono\_inic | (void) |
| void | crono\_start | (void) |
| void | crono\_stop | (void) |
| void | crono\_zera | (void) |
| unsigned char | crono\_calibra | (void) |
| void | mclk\_20MHz | (void) |
|  |  |  |
|  |  |  |

* unsigned long **crono\_ler** (void)

Ler valor cronometro.

* void **crono\_config** (void)

Configurar o cronometro. Usa TA2 com SMCLK = 20 MHz.

Habilita divisor por 20 para que clock = 1 MHz.

* void **crono\_start** (void)

Disparar cronometro, apenas liga TA2.

* void **crono\_stop** (void)

Parar cronometro, apenas pausa TA2, pode continuar de forma a somar tempos parciais.

* void **crono\_zera** (void)

Zerar cronometro. Usa TACLR para zerar TA2 e zera contador crono\_cont.

* unsigned char **crono\_calibra** (void)

Determinar erro entre Start/Stop. Faz 4 medidas e retorna a média.

#pragma vector = 43 //TIMER2\_A1\_VECTOR

* \_\_interrupt void **isr\_ta2ifg** (void){

Interrupção, apenas incrementa crono\_cont.

* void **mclk\_20MHz** (void)

Programar os relógios do MSP.

ACLK = XT1 => 32.768 Hz. SMCLK = DCO => 20.000.000 Hz. MCLK = DCO => 20.000.000 Hz.

Cronômetro com precisão de 24 bits: 0 -> 16.777.215 μs.

|  |  |
| --- | --- |
| Crono\_cont | TA2R |
| 🡨 8 bits 🡪 | 🡨 16 bits 🡪 |

|  |  |
| --- | --- |
| Segundos | Frações |
| 🡨 4 bits 🡪 | 🡨 20 bits 🡪 |