DTMF SYSTEM CONTROL DOCUMENTATION

Inclusão de Bibliotecas

O código a seguir inclui várias bibliotecas necessárias para o funcionamento do projeto no Arduino.

```
#include <Arduino.h>
#include <Adafruit_GFX.h>    // Core graphics library
#include <Adafruit_ST7789.h> // Hardware-specific library for ST7789
#include <SPI.h>             // Arduino SPI library
```

Biblioteca `Arduino.h`

- **Descrição:** Esta é a biblioteca principal do Arduino. Ela fornece a funcionalidade básica para programação com Arduino, incluindo definições e funções essenciais como `setup()` e `loop()`.
- **Funções Principais:**
  - `setup()`: Função chamada uma vez quando o programa inicia. É usada para inicializações.
  - `loop()`: Função chamada repetidamente após a execução de `setup()`. É onde o código principal do programa é executado continuamente.

Biblioteca `Adafruit_GFX.h`

- **Descrição:** Esta é a biblioteca de gráficos central da Adafruit. Ela fornece uma interface genérica para gráficos, que pode ser usada com diferentes tipos de displays.
- **Funções Principais:**
  - Desenho de formas geométricas (linhas, círculos, retângulos, etc.).
  - Renderização de texto em diferentes fontes.
  - Manipulação de pixels individuais.

Biblioteca `Adafruit_ST7789.h`

- **Descrição:** Esta é uma biblioteca específica para displays que utilizam o controlador ST7789. Ela depende da `Adafruit_GFX` para a renderização gráfica.
- **Funções Principais:**
  - Inicialização e configuração do display ST7789.
  - Controle de hardware específico do ST7789.
  - Suporte para funcionalidades avançadas de display, como rotação de tela e preenchimento de cores.

Biblioteca `SPI.h`

- **Descrição:** Esta é a biblioteca de interface de Periféricos Seriais (SPI) do Arduino. SPI é um protocolo de comunicação serial síncrona utilizado para transferir dados de forma rápida entre microcontroladores e dispositivos periféricos.
- **Funções Principais:**
  - `SPI.begin()`: Inicializa a interface SPI.
  - `SPI.transfer()`: Envia e recebe dados via SPI.
  - Configuração de parâmetros como velocidade do clock, modo SPI e ordem dos bits.

Definições de Pinos do Display ST7789

O código define os pinos usados para a conexão do display ST7789 ao Arduino:

```
#define TFT_DC   9    // DC (data/command)  DC
#define TFT_RST  8    // RST (reset) RES
#define TFT_MOSI 11   // MOSI (SPI data pin) SDA
#define TFT_SCLK 13   // SCLK (SPI sclk pin) SCK
#define TFT_CS   10   // CHIP SELECT  NC NULL
```

Definições de Caracteres DTMF

Os caracteres DTMF (Dual Tone Multi Frequency) são definidos com base nas suas frequências:

```
#define DTMF_1          0x01  //  1 - (frequências: 697 Hz e 1336 Hz)
#define DTMF_2          0x02  //  2 - (frequências: 697 Hz e 1477 Hz)
#define DTMF_3          0x03  //  3 - (frequências: 770 Hz e 1209 Hz)
#define DTMF_4          0x04  //  4 - (frequências: 770 Hz e 1336 Hz)
#define DTMF_5          0x05  //  5 - (frequências: 770 Hz e 1477 Hz)
#define DTMF_6          0x06  //  6 - (frequências: 852 Hz e 1209 Hz)
#define DTMF_7          0x07  //  7 - (frequências: 852 Hz e 1336 Hz)
#define DTMF_8          0x08  //  8 - (frequências: 852 Hz e 1477 Hz)
```

```
#define DTMF_9          0x09  //  9 - (frequências: 941 Hz e 1209 Hz)
#define DTMF_0          0x0A  //  0 - (frequências: 697 Hz e 1209 Hz)
#define DTMF_HASH       0x0B  //  # - (frequências: 941 Hz e 1477 Hz)
#define DTMF_ASTERISK   0x0C  // .* - (frequências: 941 Hz e 1209 Hz)
```

Pinos do Módulo DTMF MT8870

Os pinos do módulo DTMF MT8870 são mapeados para os pinos analógicos do Arduino, é de extrema importância esta ordem, os caracteres declarados do DTMF equivalem ao registrador PINC, sendo assim Q1 o bit LSB e Q4 MSB.

```
#define Q1      PC0             //  OUTPUT LSB DTMF   - Pino A0 do Arduino Uno (PORTC, Bit 0)
#define Q2      PC1             //  OUTPUT 1 DTMF     - Pino A1 do Arduino Uno (PORTC, Bit 1)
#define Q3      PC2             //  OUTPUT 2 DTMF     - Pino A2 do Arduino Uno (PORTC, Bit 2)
#define Q4      PC3             //  OUTPUT MSB DTMF   - Pino A3 do Arduino Uno (PORTC, Bit 3)
#define STQ     PC4             //  OUTPUT STATE      - Pino A4 do Arduino Uno (PORTC, Bit 4)
```

Pino para Interrupção

O pino de interrupção é definido para o Arduino Uno:

```
#define PIN_INTERRUPT PCINT21  // PD5 [ PINO 5 ARDUINO UNO]
```

Constantes

Algumas constantes importantes para o funcionamento do código são definidas:

```
#define KEYWORD_MAX_LENGTH  4       // SEQUENCIA MAXIMA de tons DTMF
#define CURSOR_POS0_X      20       // POSIÇÃO INICIAL CURSOR DISPLAY COLUNA
#define CURSOR_POS0_Y      30       // POSIÇÃO INICIAL CURSOR DISPLAY LINHA
#define CURSOR_INCREMENT_X 50       // INCREMENTO INICIAL PARA  CURSOR  X LINHA
```

Instância da Biblioteca Adafruit ST7789

Uma instância da biblioteca Adafruit ST7789 é criada:

```
Adafruit_ST7789 tft = Adafruit_ST7789(TFT_CS, TFT_DC, TFT_RST);
```

Protótipos de Funções

As funções são declaradas antes de serem definidas para permitir o uso em todo o código:

```
uint8_t check_password(uint8_t *password, uint8_t keyword_read[]);
uint8_t disp_asterisk(bool x);
uint8_t dtmf_dispChar (uint8_t keyword_value);
uint8_t disp_asterisk(bool x);
uint8_t relay_enable(uint8_t active);
uint8_t relay_disable(uint8_t desactive);
void    loading(void);
void    reading(void);
void    invalid_keyword(void);
```

Senhas

As sequências de tons DTMF que representam as senhas são definidas por um array:

```
uint8_t password_A1E[] = {DTMF_1, DTMF_2, DTMF_3, DTMF_4};  // PASSWORD: ATIVA  A1
uint8_t password_A2E[] = {DTMF_3, DTMF_4, DTMF_5, DTMF_6};  // PASSWORD: ATIVA  A2
uint8_t password_A3E[] = {DTMF_5, DTMF_6, DTMF_7, DTMF_8};  // PASSWORD: ATIVA  A3
uint8_t password_A4E[] = {DTMF_7, DTMF_8, DTMF_9, DTMF_0};  // PASSWORD: ATIVA  A4

uint8_t password_A1D[] = {DTMF_1, DTMF_2, DTMF_3, DTMF_3};     // PASSWORD: DESATIVA  A1
uint8_t password_A2D[] = {DTMF_3, DTMF_4, DTMF_5, DTMF_HASH};  // PASSWORD: DESATIVA  A2
```

```
uint8_t password_A3D[] = {DTMF_5, DTMF_6, DTMF_7, DTMF_HASH};  // PASSWORD: DESATIVA  A3
uint8_t password_A4D[] = {DTMF_7, DTMF_8, DTMF_9, DTMF_HASH};  // PASSWORD: DESATIVA  A4
```

Variáveis Globais

Algumas variáveis globais são definidas para uso no código:

```
uint8_t keyword_read[KEYWORD_MAX_LENGTH]; // Armazenar sequência de sinal
uint8_t keyword_sequence = 0;             // iniciando na posição zero.
uint8_t keyword;                          // Sinal lido
uint8_t cursor_posX = CURSOR_POS0_X;      // função auxiliar para display
uint8_t cursor_posY = CURSOR_POS0_Y;      // função auxiliar para display
volatile bool flag = 0;                   // flag para interrupção
```

Função Principal `main`

A função principal do programa, onde a configuração inicial é feita e o loop principal é executado:

```
int main(void)
{
  UCSR0B  &= ~0x18;            // desabilita RX (bit4) e TX(bit3) para trabalho com os pinos do PORTD no Arduino
  DDRD    |= 0xC0;             // Define os bits 6 e 7 como saída (11000000 em binário)
  PORTD    = 0x00;             // garante os pinos PD0 à PD7 iniciem em LOW
  DDRC    &= ~0x7F;            // DEFINE PORTC COMO ENTRADA
  PORTC   &= ~0x7F;            // Garante que os pinos A0 à A6 iniciem em zero;
  PCICR   |= (1 << PCIE2);     // Habilita a interrupção via PORTD (PCINT16-23)
  PCMSK2  |= (1 << PCINT21);   // habilita interrupção para PCINT21 (PD5) (pino 5 arduino)

// Teste e Animação inicial do Display

  Serial.begin(9600);
  tft.init(240, 240, SPI_MODE2);
  tft.setRotation(2);
  tft.fillScreen(ST77XX_BLACK);
  loading();
  reading();
  tft.fillScreen(ST77XX_BLACK);
  disp_asterisk(1);
  _delay_ms(200);
```

Leitura do Sinal

Este loop é o cerne do firmware, a cada sinal lido ele armazena o valor em uma sequência, assim que a sequência for completa ele inicia uma interrupção por mudança de estado definido pela logica e faz a checagem com as senhas recebidas.

Perceba-se que antes da interrupação a flag é atualizada, e então a interrupção global é ativa e em seguida ocorre a mudança de estado do pino que gera a interrupção.

```
while (true) // LOOP LEITURA DTMF KEYS
{
  bool signal = (PINC &(1 << STQ));                 // Lê o estado do STQ
  if (signal)
  {
      keyword = (PINC & 0x0F);  // ADD AO KYWORD O VALOR NO PINC da forma 0x00
      _delay_ms(100);               //isso garante que o Keyword terá o mesmo formato dos caracteres definidos DTMF_ 0x00
      if (keyword_sequence < KEYWORD_MAX_LENGTH)
      {
        keyword_read[keyword_sequence++] = keyword; // ADD AO ARRAY O VALOR KEYWORD E INCREMENTA POSIÇÃO
        disp_asterisk(0);
        dtmf_dispChar(keyword);                     // MOSTRA NO DISPLAY O TON RECEBEIDO
        cursor_posX+= CURSOR_INCREMENT_X;           // MUDA A POSIÇÃO DO CURSOR
        _delay_ms(100);
      }//fim keyword_read
  }
  else if (!signal && !(keyword_sequence < KEYWORD_MAX_LENGTH))
  {
    flag = 1;
    SREG    |= 0x80;               // Habilitar interrupções globais bit 7(I = 1)
    PORTD |= (1 << PIN_INTERRUPT); // ATIVA INTERRUPÇÃO
```

```
      _delay_ms(2000);
      keyword_sequence = 0;
      cursor_posX = CURSOR_POS0_X;
      cursor_posY = CURSOR_POS0_Y;

      tft.fillScreen(ST77XX_BLACK);
      reading();
      tft.fillScreen(ST77XX_BLACK);
      disp_asterisk(1);
    }// fim else
  } // fim leitura signal
} // fim main
```

Detalhes função `main`

- `UCSR0B` &= ~0x18; // desabilita RX (bit4) e TX (bit3) para trabalho com os pinos do PORTD no Arduino
- `DDRD` |= 0xC0; // Define os bits 6 e 7 como saída (11000000 em binário)
- `PORTD` = 0x00; // Garante que os pinos PD0 a PD7 iniciem em LOW
- `DDRC` &= ~0x7F; // Define PORTC como entrada (A0 a A6)
- `PORTC` &= ~0x7F; // Garante que os pinos A0 a A6 iniciem em zero
- `PCICR` |= (1 << PCIE2); // Habilita a interrupção via PORTD (PCINT16-23)
- `PCMSK2` |= (1 << PCINT21); // Habilita interrupção para PCINT21 (PD5) (pino 5 do Arduino)
- `SREG` |= 0x80; // Habilita interrupções globais (bit 7, I = 1)

`Serial.begin(9600)` // Inicializa a comunicação serial a 9600 bps
`tft.init(240, 240, SPI_MODE2)` // Inicializa o display ST7789
`tft.setRotation(2)` // Define a rotação do display
`tft.fillScreen(ST77XX_BLACK)` // Preenche a tela do display com a cor preta
`loading()` // Chama a função de animação de carregamento
`reading()` // Chama a função de animação de leitura
`tft.fillScreen(ST77XX_BLACK)` // Preenche a tela do display com a cor preta novamente
`disp_asterisk(1)` // Exibe asteriscos no display
`_delay_ms(10);` // Aguarda 10 milissegundos

O loop principal (`while (true)`) realiza a leitura dos tons DTMF e atualiza o display:

- `bool signal = (PINC &(1 << STQ));`: Lê o estado do pino STQ.
- `if (signal)`: Se o sinal for verdadeiro, lê o valor dos pinos e armazena na variável `keyword`.
- `keyword = (PINC & 0x0F);` aqui ele le os valores do PINC que estão relacionados ao ao dtmf
- `if (keyword_sequence < KEYWORD_MAX_LENGTH)`: Se a sequência de palavras-chave for menor que o comprimento máximo, armazena o valor na array `keyword_read`.
- `else if (!signal && !(keyword_sequence < KEYWORD_MAX_LENGTH))`: Se o sinal for falso e a sequência estiver completa, habilita a interrupção e reinicia a leitura.

Interrupção `ISR(PCINT2_vect)`

A função de interrupção verifica a sequência de palavras-chave e ativa ou desativa os relés correspondentes, em seguida limpa a flag da variavel e do registrador, então desativa a interrupção global, desta forma é uma garantia que possiveis flutuações causem interrupções indevidas em outros trechos do código, com isso troca o estado do pino de interrupção.

```
ISR(PCINT2_vect)
{
  if(flag){
    if      (check_password(password_A1E,keyword_read))   relay_enable(1);
    else if (check_password(password_A2E,keyword_read))   relay_enable(2);
    else if (check_password(password_A3E,keyword_read))   relay_enable(3);
    else if (check_password(password_A4E,keyword_read))   relay_enable(4);
    else if (check_password(password_A1D,keyword_read))   relay_disable(1);
    else if (check_password(password_A2D,keyword_read))   relay_disable(2);
    else if (check_password(password_A3D,keyword_read))   relay_disable(3);
    else if (check_password(password_A4D,keyword_read))   relay_disable(4);
    else invalid_keyword();
  }
  flag = 0;
  PCIFR |= (1 << PCIF2);
  cli();
  PORTD &= ~(1 << PIN_INTERRUPT);
}
```

Funções Auxiliares

check_password

Verifica se a sequência de tons corresponde a uma senha:

```
uint8_t check_password(uint8_t *password, uint8_t keyword_read[]) // Função para verificar se a sequência de tons corresponde a uma senha
{
  for (uint8_t i = 0; i < KEYWORD_MAX_LENGTH; i++)
  {
    if (keyword_read[i] != password[i]) // Verifica cada elemento da sequência com a senha
      return false;                     // Se houver uma diferença, retorna falso
  }
  return true; // Se a sequência corresponder à senha, retorna verdadeiro
} //fim
```

dtmf_dispChar

Exibe o caractere DTMF no display fazendo a comparação do sinal unico recebido com o valor dos caracteres e printa no display o correspondente.

```
uint8_t dtmf_dispChar (uint8_t keyword_value){

extern uint8_t cursor_posX;
extern uint8_t cursor_posY;

tft.setTextSize(3);
tft.setCursor(cursor_posX, cursor_posY);
tft.setTextColor(ST77XX_BLACK);
tft.println(" * ");
tft.setTextColor(ST77XX_YELLOW);

// CONVERSÃO DUAL-TONE EM CARACTERE
switch (keyword_value)
{

case DTMF_1:           tft.setCursor(cursor_posX, cursor_posY); tft.println(" 1 ");   break;

case DTMF_2:           tft.setCursor(cursor_posX, cursor_posY); tft.println(" 2 ");   break;

case DTMF_3:           tft.setCursor(cursor_posX, cursor_posY); tft.println(" 3 ");   break;

case DTMF_4:           tft.setCursor(cursor_posX, cursor_posY); tft.println(" 4 ");   break;

case DTMF_5:           tft.setCursor(cursor_posX, cursor_posY); tft.println(" 5 ");   break;

case DTMF_6:           tft.setCursor(cursor_posX, cursor_posY); tft.println(" 6 ");   break;

case DTMF_7:           tft.setCursor(cursor_posX, cursor_posY); tft.println(" 7 ");   break;

case DTMF_8:           tft.setCursor(cursor_posX, cursor_posY); tft.println(" 8 ");   break;

case DTMF_9:           tft.setCursor(cursor_posX, cursor_posY); tft.println(" 9 ");   break;

case DTMF_0:           tft.setCursor(cursor_posX, cursor_posY); tft.println(" 0 ");   break;

case DTMF_HASH:        tft.setCursor(cursor_posX, cursor_posY); tft.println(" # ");   break;

case DTMF_ASTERISK:    tft.setCursor(cursor_posX, cursor_posY); tft.println(" # ");   break;

default:  tft.println("ERROR");
break;
}

tft.setTextColor(ST77XX_YELLOW);
tft.setCursor(cursor_posX, cursor_posY);

return keyword_value;
}
```

loading

Animação de carregamento:

```
void loading(void)
{
  const int centerX = 120;
  const int centerY = 120;
  const int barWidth = 20;
  const int barHeight = 10;

  tft.setTextColor(ST77XX_BLUE);
  tft.setCursor(40, 200);
  tft.setTextSize(1);
  tft.println("Created by: Yasser");

  tft.setTextColor(ST77XX_CYAN);
  tft.setTextSize(4);

  tft.setCursor(40, 10);
  tft.println("LOADING");

  for (int i = 0; i <= 100; i += 4)
  {
    int barLength = map(i, 0, 100, 0, 2 * centerX - barWidth);
    tft.fillRect(centerX - barLength / 2, centerY - barHeight / 2, barLength, barHeight, ST77XX_WHITE);
    delay(50);
    tft.fillRect(centerX - barLength / 2, centerY - barHeight / 2, barLength, barHeight, ST77XX_BLACK);
  }
  tft.fillScreen(ST77XX_BLACK);
}
```

reading

Animação de leitura:

```
void reading(void)
{
  const int centerX = 120;
  const int centerY = 120 + 30;
  const int radius = 40;
  const int numDots = 200;
  const int dotSize = 6;

  tft.setTextColor(ST77XX_BLUE);
  tft.setTextSize(dotSize);
  uint16et)

  tft.setTextSize(3);
  tft.setTextColor(ST77XX_YELLOW);
  tft.setCursor(45, 20);
  tft.println("RECIVING");
  tft.setCursor(35, 45);
  tft.println("DUAL-TONE");
```

```
int i = 0;
int j = 1;

for (int i = 0; i < numDots; ++i)
{
  int x = centerX + int(radius * cos(i * 2 * PI / numDots));
  int y = centerY + int(radius * sin(i * 2 * PI / numDots));

  tft.fillCircle(x, y, dotSize / 2, dotColor);
}
for (i = 0; i < numDots; ++i)
{
  int x0 = centerX + int(radius * cos(i * 2 * PI / numDots));
  int y0 = centerY + int(radius * sin(i * 2 * PI / numDots));

  int x1 = centerX + int(radius * cos(j) * 2 * PI / numDots);
  int y1 = centerY + int(radius * sin(j) * 2 * PI / numDots);

  tft.fillCircle(x0, y0, dotSize / 2, ST77XX_YELLOW);

  _delay_ms(10);
  tft.fillCircle(x1, y1, dotSize / 2, dotColor);
  j++;
```

```
    }
  }
```

disp_asterisk

Exibe asteriscos no display:

```
uint8_t disp_asterisk(bool x){


  extern uint8_t cursor_posX;
  extern uint8_t cursor_posY;

  uint8_t set_cursor = cursor_posX;

  tft.setCursor(set_cursor, cursor_posY);

  tft.setTextSize(3);
  tft.setTextColor(ST77XX_RED);


  if(x){
  for (int i = 0; i < KEYWORD_MAX_LENGTH; i++)
  {

  tft.setCursor(set_cursor, cursor_posY);
  tft.println(" * ");
  set_cursor += CURSOR_INCREMENT_X;
  }


  }
  else{

  tft.setTextSize(3);
  tft.setTextColor(ST77XX_BLACK);
  tft.setCursor(cursor_posX, cursor_posY);
  tft.println(" * ");
  }
  return x;
}
```

relay_enable

Habilita os relés:

```
uint8_t relay_enable(uint8_t enable){

tft.fillScreen(ST77XX_BLACK);
tft.setTextColor(ST77XX_GREEN);


uint8_t set_textsizeA = 9;
uint8_t set_cursorAX = 70;
uint8_t set_cursorAY = 40;

uint8_t set_textE = 4;
uint8_t set_cursorEX = 35;
uint8_t set_cursorEY = 150;



if  (enable == 1)
{

tft.setTextSize(set_textsizeA);
tft.setCursor(set_cursorAX, set_cursorAY);
tft.println("A1");

tft.setTextSize(set_textE);
tft.setCursor(set_cursorEX, set_cursorEY);
tft.println("ENABLED");
```

```
    }

    else if (enable ==  2)
    {

    tft.setTextSize(set_textsizeA);
    tft.setCursor(set_cursorAX, set_cursorAY);
    tft.println("A2 ");

    tft.setTextSize(set_textE);
    tft.setCursor(set_cursorEX, set_cursorEY);
    tft.println("ENABLED");

    }
    else if (enable == 3)
    {

    tft.setTextSize(set_textsizeA);
    tft.setCursor(set_cursorAX, set_cursorAY);
    tft.println("A3");

    tft.setTextSize(set_textE);
    tft.setCursor(set_cursorEX, set_cursorEY);
    tft.println("ENABLED");

    }else if(enable == 4) {


    tft.setTextSize(set_textsizeA);
    tft.setCursor(set_cursorAX, set_cursorAY);
    tft.println("A4");

    tft.setTextSize(set_textE);
    tft.setCursor(set_cursorEX, set_cursorEY);
    tft.println("ENABLED");

    }
    return enable;
  }
```

relay_disable

Desabilita os relés:

```
  uint8_t relay_disable(uint8_t disable){

    tft.fillScreen(ST77XX_BLACK);
    tft.setTextColor(ST77XX_RED);

    uint8_t set_textsizeA = 9;
    uint8_t set_cursorAX = 70;
    uint8_t set_cursorAY = 40;

    uint8_t set_textE = 4;
    uint8_t set_cursorEX = 35;
    uint8_t set_cursorEY = 150;



    if  (disable == 1)
    {

    tft.setTextSize(set_textsizeA);
    tft.setCursor(set_cursorAX, set_cursorAY);
    tft.println("A1");

    tft.setTextSize(set_textE);
    tft.setCursor(set_cursorEX, set_cursorEY);
    tft.println("DISABLE");
    }

    else if (disable == 2)
    {

    tft.setTextSize(set_textsizeA);
    tft.setCursor(set_cursorAX, set_cursorAY);
```

```
    tft.println("A2 ");

    tft.setTextSize(set_textE);
    tft.setCursor(set_cursorEX, set_cursorEY);
    tft.println("DISABLE");

    }
    else if (disable == 3)
    {

    tft.setTextSize(set_textsizeA);
    tft.setCursor(set_cursorAX, set_cursorAY);
    tft.println("A3");

    tft.setTextSize(set_textE);
    tft.setCursor(set_cursorEX, set_cursorEY);
    tft.println("DISABLE");

    }else if(disable == 4) {


    tft.setTextSize(set_textsizeA);
    tft.setCursor(set_cursorAX, set_cursorAY);
    tft.println("A4");

    tft.setTextSize(set_textE);
    tft.setCursor(set_cursorEX, set_cursorEY);
    tft.println("DISABLE");
    }
    return disable;
  }
```

invalid_keyword

Exibe mensagem de senha inválida:

```
  void invalid_keyword(void){

    uint8_t set_textsizeA = 4;
    uint8_t set_cursorAX = 35;
    uint8_t set_cursorAY = 80;

    uint8_t set_cursorEX = 30;
    uint8_t set_cursorEY = 130;

    tft.fillScreen(ST77XX_RED);
    tft.setTextColor(ST77XX_WHITE);

    tft.setTextSize(set_textsizeA);
    tft.setCursor(set_cursorAX, set_cursorAY);
    tft.println("INVALID");
    tft.setCursor(set_cursorEX, set_cursorEY);
    tft.println("PASSWORD");

    _delay_ms(1000);
    disp_asterisk(true);

  }
```