جامعة عفت

EFFAT UNIVERSITY

CS 4111 - 1 - Programming Embedded Systems

Programming Embedded Systems Project Report: Ultrasonic Radar Sensor Module

Student Names:

Ameera Attiah - S21107316

Mawaddah Alagha - S20106707

Lujain Almarri - S20106753

Instructor: **Dr. Zain Balfagih**

Date Last Edited: December 12, 2023

Contents

1 Introduction

The simulation ultrasonic radar sensor module, at its core, represents a harmonious blend of hardware and software ingenuity, encapsulating an Arduino microcontroller, a servo motor, an ultrasonic sensor, and a 1.8-inch SPI TFT screen. This compact yet powerful system functions as a 180-degree scanning radar, unveiling a high-level overview of its capabilities.

At the heart of its functionality lies the ability to detect objects within a range of 2 to 300 cm, capturing their directional positions through the servo motor and offering real-time visual feedback on the TFT screen. The radar's prowess extends to dynamic distance measurement, obstacle detection, and continuous monitoring of the spatial landscape.

Our journey through this report will unravel the intricate circuitry of the embedded board, detailing the roles of each component and shedding light on the power supply dynamics. Simultaneously, we'll explore the device interfacing aspect, focusing on the ultrasonic sensor as the input device and the TFT screen as the output device.

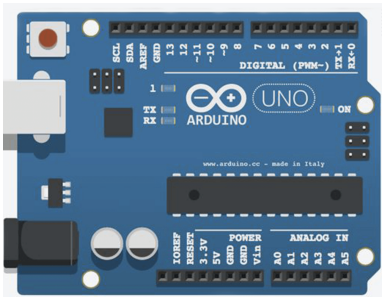
It's essential to note that this report emphasizes the practical implementation of the ultrasonic radar sensor module, spotlighting what is currently operational in the system. We will distinguish between successful implementations and aspects that, despite aspirations, did not materialize due to various constraints. Join us in uncovering the working mechanics of this innovative radar system and gaining insights into the blend of hardware and software functionalities that propel it forward.

2 Hardware

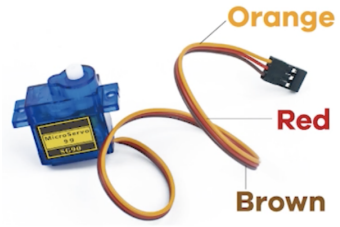
2.1 Embedded Board Description

The embedded board used for the simulation ultrasonic radar sensor module is based on the Arduino platform. It serves as the central processing unit for interfacing with the ultrasonic sensor, servo motor, and TFT screen. The key hardware components include:

1. **Microcontroller:** The board utilizes an Arduino microcontroller to execute the control logic, manage data processing, and communicate with the connected peripherals.

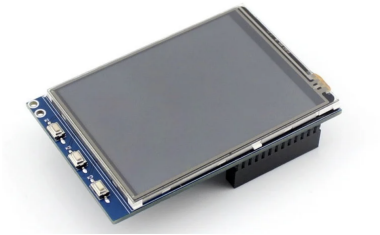


2. **Servo Motor:** The servo motor is employed to achieve a 180-degree scanning range for the ultrasonic sensor. It is controlled by the Arduino to enable directional sensing of objects.

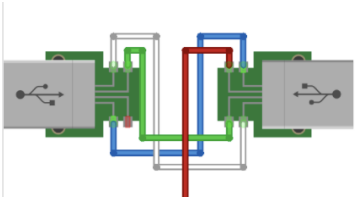


3. **Ultrasonic Sensor:** This sensor is responsible for measuring distances between the module and detected objects. The Arduino communicates with the ultrasonic sensor to obtain distance data, facilitating both distance measurement and obstacle detection.

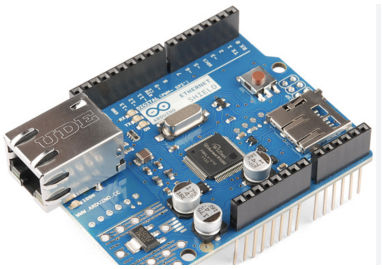
4. **TFT Screen:** The 1.8-inch SPI TFT screen serves as the output display, providing visual feedback of the detected targets. The Arduino controls the TFT screen to represent targets within the specified range using different colored dots.



5. **Power Supply:** The board can be powered either by an external source with a voltage ranging from 7-12V or through a 5V power supply via the USB interface.



6. **Circuit Schematics and Shields:** Include detailed circuit schematics illustrating the connections between the Arduino, servo motor, ultrasonic sensor, and TFT screen. If any additional shields or modules are used for interfacing, provide information on their roles and connections within the circuit.



7. **Piezo Buzzer:** The piezo buzzer is used for auditory distance alerts. It emits different tones based on object proximity, detected by the ultrasonic sensor, enhancing the mini radar kit's functionality with sound feedback.

2.2 Input device description

The ultrasonic sensor is a key component of the simulation ultrasonic radar sensor module. It is responsible for measuring distances between the sensor and objects in the environment.

2.2.1 Access in Software:

The Arduino microcontroller interacts with the ultrasonic sensor by utilizing specific software commands and functions. The Arduino programming environment provides a set of libraries and functions that facilitate easy communication with the ultrasonic sensor. These functions typically include commands for triggering the sensor, capturing and processing echo signals, and extracting distance information.

2.2.2 Communication Protocol:

The communication between the Arduino and the ultrasonic sensor involves a simple yet crucial protocol. Here is a generalized overview:

1. **Triggering the Measurement:** The Arduino sends a trigger signal to the ultrasonic sensor to initiate a distance measurement. The Arduino sends a trigger signal to the ultrasonic sensor to initiate a distance measurement.

2. **Echo Reception:** After triggering, the ultrasonic sensor emits ultrasonic pulses. When these pulses encounter an object, they bounce back as echoes.

3. **Echo Detection and Processing:** The Arduino waits for the echo signals and detects their return. The time taken for the echoes to return is measured.

4. **Distance Calculation:** Using the known speed of sound, the Arduino calculates the distance between the sensor and the detected object. The formula for distance calculation is often expressed as: Distance = (Time taken for echo to return * Speed of sound) / 2.

5. **Data Retrieval:** The calculated distance information is then retrieved by the Arduino for further processing and decision-making.

2.2.3 Additional Considerations:

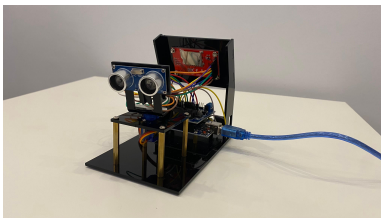
Timing is crucial in this communication protocol, and the Arduino must be programmed to handle the timing intricacies involved in sending triggers, capturing echoes, and calculating distances accurately. The communication protocol might involve error-checking mechanisms to ensure the reliability of distance measurements.

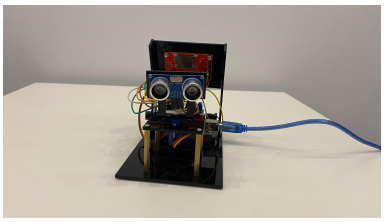
2.3 Output Device description

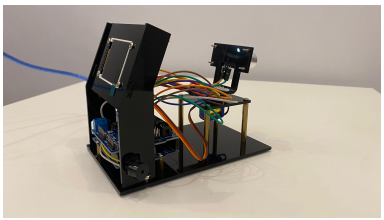
The TFT screen functions as the output device, displaying visual feedback based on the data received from the ultrasonic sensor. Targets detected within a 1-meter range are represented by red dots, while targets beyond 1 meter are represented by yellow dots. The Arduino controls the TFT screen to dynamically update and visualize the position and distance information of the detected objects.

2.4 Final Model:

Here is our final model with every components integrated:







3 Software

3.1 Sketches

This Arduino sketch creates a mini radar system using an ultrasonic sensor, a servo motor, and a display. It begins by setting up the necessary components: the ultrasonic sensor for distance measurement, the servo motor for movement, and a piezo buzzer for audio feedback. The Ucglib graphics library is used for the display operations. The setup function initializes the display, showcasing "Mini Radar," and tests the servo motor's rotation.

The main function, loop, drives the radar's operation. It moves the servo, mimicking a radar sweep, and uses the ultrasonic sensor to detect object distances. This data is then displayed on the screen as a radar interface with sweeping lines and distance markers. Functions like playSoundBasedOnDistance() and calculateDistance() enhance the radar's functionality, providing sound feedback based on proximity and calculating distances, respectively. This sketch effectively combines hardware and software to simulate a radar system, showcasing real-time object detection and distance visualization.

```
/*mini radar kit*/
#include <Servo.h>
#include <SPI.h>
#include "Ucglib.h"
#define  trigPin   6          //Ultrasonic Module Trig->D6
#define  echoPin   5          //Ultrasonic Module Echo->D5
#define  ServoPin  3          //Servo Signal->D3
#define buzzerPin 7   // Piezo Buzzer->D7

int Ymax = 128;               //vertical pixels of the screen
int Xmax = 160;               //horizontal pixels of the screen
int Xcent = Xmax / 2;         //Horizontal screen center position
int base = 118;               //baseline position
int scanline = 105;           //Radar scan line length
Servo baseServo;
Ucglib_ST7735_18x128x160_HWSPI ucg(/*cd=*/ 9, /*cs=*/ 10, /*reset=*/ 8);


void setup(void)
{
      ucg.begin(UCG_FONT_MODE_SOLID); //initialization screen
      ucg.setRotate90();              //Set to horizontal screen If the screen display direction is reversed, you can modify the function setRotate90 or setRotate270

      pinMode(trigPin, OUTPUT);       //Set the trig Pin port mode
      pinMode(echoPin, INPUT);        //Set echo Pin port mode
      Serial.begin(115200);             //Set the serial port transmission rate
      baseServo.attach(ServoPin);     //Initialize servos

      pinMode(buzzerPin, OUTPUT); // Set buzzer Pin as output mode


      //screen startup interface
      ucg.setFontMode(UCG_FONT_MODE_TRANSPARENT);
      ucg.setColor(0, 0, 100, 0);
      ucg.setColor(1, 0, 100, 0);
      ucg.setColor(2, 20, 20,20);
      ucg.setColor(3, 20, 20, 20);
      ucg.drawGradientBox(0, 0, 160, 128);
      ucg.setPrintDir(0);
      ucg.setColor(0, 5, 0);
      ucg.setPrintPos(27,42);
      ucg.setFont(ucg_font_logisoso18_tf);
      ucg.print("Mini Radar");
      ucg.setColor(0, 255, 0);
      ucg.setPrintPos(25,40);
      ucg.print("Mini Radar");
      ucg.setFont(ucg_font_helvB08_tf);
      ucg.setColor(0, 255, 0);
      ucg.setPrintPos(40,100);
      ucg.print("Testing...");
      baseServo.write(90);
      ucg.setColor(0, 255, 0);

      //Test the operation of the base, pay attention to the position and rotation posture of the base, and whether there is any jamming (or wire winding).
      for(int x=0;x<180;x+=5)
          { baseServo.write(x);
            delay(50);
           }
```

```
      ucg.print("OK!");
      delay(500);
      ucg.setColor(0, 0, 0, 0);
      ucg.setColor(1, 0, 0, 0);
      ucg.setColor(2, 0, 0,0);
      ucg.setColor(3, 0, 0, 0);

      //clear screen
      //ucg.clearScreen();
      cls();

      ucg.setFont(ucg_font_orgv01_hr);
      ucg.setFontMode(UCG_FONT_MODE_SOLID);
}


void cls()
{
  //clear screen
  ucg.setColor(0, 0, 0, 0);

  for(int s=0;s<128;s+=8)
  for(int t=0;t<160;t+=16)
  {
    ucg.drawBox(t,s,16,8);
   // delay(1);
  }


}

void playSoundBasedOnDistance(int distance) {
   int toneFrequency;

   if (distance < 30) {  // If object is very close
       toneFrequency = 2000; // High tone
   } else if (distance < 100) { // If object is moderately close
       toneFrequency = 1000; // Medium tone
   } else {
       toneFrequency = 500; // Low tone
   }

   tone(buzzerPin, toneFrequency, 100); // Play tone for 100 milliseconds
   delay(100); // Delay to avoid continuous sound
}


int calculateDistance()
{
      long duration;
      //power off trigPin and wait 2 microseconds
      digitalWrite(trigPin, LOW);
      delayMicroseconds(2);
      //TrigPin power on delay 10 microseconds and then power off
      digitalWrite(trigPin, HIGH);
      delayMicroseconds(10);
      digitalWrite(trigPin, LOW);
      //Reading the echoPin returns the travel time of the sound wave (in microseconds)
      duration = pulseIn(echoPin, HIGH);
      //Convert echo time to distance value
      return duration*0.034/2;
}

void fix_font()
{
      ucg.setColor(0, 180, 0);
      ucg.setPrintPos(70,128-120+7);
      ucg.print("100cm");
```

```
      ucg.setPrintPos(70,128-85-11);
      ucg.print("75cm");
      ucg.setPrintPos(70,128-60-8);
      ucg.print("50cm");
      ucg.setPrintPos(70,128-35-4);
      ucg.print("25cm");
}

void fix()
{

      ucg.setColor(0, 40, 0);
      //Draw a background image
      ucg.drawDisc(Xcent, base+1, 3, UCG_DRAW_ALL);
      ucg.drawCircle(Xcent, base+1, 115, UCG_DRAW_UPPER_LEFT);
      ucg.drawCircle(Xcent, base+1, 115, UCG_DRAW_UPPER_RIGHT);
      ucg.drawCircle(Xcent, base+1, 86, UCG_DRAW_UPPER_LEFT);
      ucg.drawCircle(Xcent, base+1, 86, UCG_DRAW_UPPER_RIGHT);
      ucg.drawCircle(Xcent, base+1, 58, UCG_DRAW_UPPER_LEFT);
      ucg.drawCircle(Xcent, base+1, 58, UCG_DRAW_UPPER_RIGHT);
      ucg.drawCircle(Xcent, base+1, 29, UCG_DRAW_UPPER_LEFT);
      ucg.drawCircle(Xcent, base+1, 29, UCG_DRAW_UPPER_RIGHT);
      ucg.drawLine(0, base+1, Xmax,base+1);

      ucg.setColor(0, 120, 0);
      //draw scale
       for(int i= 40;i < 140; i+=2)
       {

        if (i % 10 == 0)
          ucg.drawLine(105*cos(radians(i))+Xcent,base - 105*sin(radians(i)) , 113*cos(
   radians(i))+Xcent,base - 113*sin(radians(i)));
        else

         ucg.drawLine(110*cos(radians(i))+Xcent,base - 110*sin(radians(i)) , 113*cos(radians
   (i))+Xcent,base - 113*sin(radians(i)));
       }

       //draw some decorative patterns
       ucg.setColor(0,200,0);
       ucg.drawLine(0,0,0,18);
       for(int i= 0;i < 5; i++)
       {
          ucg.setColor(random(255),random(255),random(255));
          ucg.drawBox(2,i*4,random(14)+2,3);
       }

       ucg.setColor(0,0,180);
       ucg.drawFrame(146,0,14,14);
       ucg.setColor(0,0,60);
       ucg.drawHLine(148,0,10);
       ucg.drawVLine(146,2,10);
       ucg.drawHLine(148,13,10);
       ucg.drawVLine(159,2,10);

       ucg.setColor(random(255),random(255),random(255));
       //ucg.setColor(0,220,0);
       ucg.drawBox(148,2,4,4);
       ucg.setColor(0,220,0);
       ucg.drawBox(148,8,4,4);
       ucg.setColor(random(255),random(255),random(255));
       //ucg.setColor(100,0,0);
       ucg.drawBox(154,8,4,4);
       ucg.setColor(random(255),random(255),random(255));
       //ucg.setColor(0,0,150);
       ucg.drawBox(154,2,4,4);

       ucg.setColor(0,0,90);
```

```
        ucg.drawTetragon(62,123,58,127,98,127,102,123);
        ucg.setColor(0,0,160);
        ucg.drawTetragon(67,123,63,127,93,127,97,123);
        ucg.setColor(0,255,0);
        ucg.drawTetragon(72,123,68,127,88,127,92,123);
}



void loop(void)
{

  int distance;

  fix();
  fix_font();  //Repaint screen background elements

  for (int x=180; x > 4; x-=2){       //The base servo rotates from 180 to 0 degrees

      baseServo.write(x);             //Adjust the steering gear angle

      //Draw Radar Scanlines
      int f = x - 4;
      ucg.setColor(0, 255, 0);
      ucg.drawLine(Xcent, base, scanline*cos(radians(f))+Xcent,base - scanline*sin(radians(f
    )));
      f+=2;
      ucg.setColor(0, 128, 0);
      ucg.drawLine(Xcent, base, scanline*cos(radians(f))+Xcent,base - scanline*sin(radians(f
    )));
      f+=2;
      ucg.setColor(0, 0, 0);
      ucg.drawLine(Xcent, base, scanline*cos(radians(f))+Xcent,base - scanline*sin(radians(f
    )));
      ucg.setColor(0,200, 0);
      //Get the distance value
      distance = calculateDistance();


   // Play sound based on distance
   playSoundBasedOnDistance(distance);

      //Draw a point at the corresponding position according to the measured distance
      if (distance < 100)
      {
        ucg.setColor(255,0,0);
        ucg.drawDisc(1.15*distance*cos(radians(x))+Xcent,-(1.15*distance*sin(radians(x)))+
    base, 1, UCG_DRAW_ALL);
      }
      else
      { //If it is more than 1 meter, it is indicated by a yellow painting on the edge area
        ucg.setColor(255,255,0);
        ucg.drawDisc(116*cos(radians(x))+Xcent,-116*sin(radians(x))+base, 1, UCG_DRAW_ALL);
      }


      //Debug code, output angle and range value
      Serial.print("Degree:  ");
      Serial.print(x);
      Serial.print("    ,Distance:   ");
      Serial.println(distance);


      if (x > 70 and x < 110)  fix_font();  //When the scan line and the number coincide,
    redraw the number

      ucg.setColor(0,0,155,  0);
      ucg.setPrintPos(0,126);
```

```
256       ucg.print("DEG: ");
257       ucg.setPrintPos(24,126);
258       ucg.print(x);
259       ucg.print("  ");
260       ucg.setPrintPos(125,126);
261       ucg.print("  ");
262       ucg.print(distance);
263       ucg.print("cm  ");
264
265   }
266   delay(50);
267   cls();
268   fix();
269   fix_font();          //Repaint screen background elements
270
271   for (int  x=1; x < 176; x+=2){
272       baseServo.write(x);             //Adjust the steering gear angle
273
274       //Draw Radar Scanlines
275       int f = x + 4;
276       ucg.setColor(0, 255, 0);
277       ucg.drawLine(Xcent, base, scanline*cos(radians(f))+Xcent,base - scanline*sin(radians(f
    )));
278       f-=2;
279       ucg.setColor(0, 128, 0);
280       ucg.drawLine(Xcent, base, scanline*cos(radians(f))+Xcent,base - scanline*sin(radians(f
    )));
281       f-=2;
282       ucg.setColor(0, 0, 0);
283       ucg.drawLine(Xcent, base, scanline*cos(radians(f))+Xcent,base - scanline*sin(radians(f
    )));
284       ucg.setColor(0, 200, 0);
285
286       distance = calculateDistance();
287       playSoundBasedOnDistance(distance);
288
289       //Draw a point at the corresponding position according to the measured distance
290       if (distance < 100)
291       {
292         ucg.setColor(255,0,0);
293         ucg.drawDisc(1.15*distance*cos(radians(x))+Xcent,-(1.15*distance*sin(radians(x)))+
    base, 1, UCG_DRAW_ALL);
294       }
295       else
296       { //If it is more than 1 meter, it is indicated by a yellow painting on the edge area
297         ucg.setColor(255,255,0);
298         ucg.drawDisc(116*cos(radians(x))+Xcent,-116*sin(radians(x))+base, 1, UCG_DRAW_ALL);
299       }
300
301       //Debug code, output angle and range value
302       Serial.print("Degree:  ");
303       Serial.print(x);
304       Serial.print("    ,Distance:   ");
305       Serial.println(distance);
306
307       if (x > 70 and x < 110)  fix_font();  //When the scan line and the number coincide,
    redraw the number
308
309       ucg.setColor(0,0,155,  0);
310       ucg.setPrintPos(0,126);
311       ucg.print("DEG: ");
312       ucg.setPrintPos(24,126);
313       ucg.print(x);
314       ucg.print("   ");
315       ucg.setPrintPos(125,126);
316       ucg.print("   ");
317       ucg.print(distance);
318       ucg.print("cm   ");
```

```
319
320   }
321  delay(50);
322  cls();
323
324 }
```

Listing 1: This is the arduino code for our ultrasonic radar sensor

3.2 Libraries

This Arduino code uses three libraries, each providing specific functionality for the mini radar kit project:

1. **Servo.h:** This library is used to control servo motors. In your code, it's utilized to rotate the base of the radar setup. The `Servo` class provides methods to attach a servo to a pin, and to set its angle. You can see this in action with `baseServo.attach(ServoPin);` and `baseServo.write(x);`, where the servo motor is controlled to sweep the radar.

2. **SPI.h:** The Serial Peripheral Interface (SPI) library is used for communication between the Arduino and peripheral devices, typically sensors or SD cards. However, in your code, it seems to be included as a dependency for the Ucglib library but not directly used.

3. **Ucglib.h:** Ucglib is a graphics library for Arduino, used to control different types of displays. In your project, it's used to manage a graphical display for the radar. This library provides functions for drawing text, shapes, and images on the screen. The object `ucg` is an instance of `Ucglib_ST7735_18x128x160_HWSPI`, a specific driver for the ST7735 display controller. The library is extensively used throughout your code for drawing the radar interface, updating distance measurements, and creating visual feedback on the display.

These libraries collectively provide the necessary interface and control over the hardware components of the radar system: the servo motor for movement, the SPI for communication, and the Ucglib for display management.

4 Related Work

4.1 Has anyone done a project like this before?

Although our team has not undertaken an identical project, similar initiatives exist in the broader technological community. Projects involving Arduino-based microcontrollers and ultrasonic sensors for object detection and distance measurement are common in fields like robotics and automation. Our project's uniqueness lies in its application as a 180-degree scanning radar, achieved through the integration of a servo motor. This distinctive feature, coupled with real-time visual feedback, color-coded target representation, and Piezo actuator sets our project apart within this domain.

4.2 How does your project compare to existing similar projects?

While many projects employ ultrasonic sensors with Arduino for distance measurement and object detection, the uniqueness of the simulation ultrasonic radar sensor module lies in its specific application as a 180-degree scanning radar. The incorporation of a servo motor allows for a sweeping motion, expanding the field of view and providing a comprehensive spatial overview. Also the addition of Piezo made our project unique to regular radar modules.

Compared to traditional static ultrasonic sensor applications, the radar-like functionality adds a dynamic element, making it well-suited for scenarios requiring continuous monitoring over a wide range. The real-time visual feedback on the TFT screen enhances user interaction and situational awareness.

The color-coded representation of detected targets based on distance further distinguishes this project. Red dots signify objects within a 1-meter range, while yellow dots indicate objects beyond 1 meter. This feature facilitates quick and intuitive interpretation of the spatial layout.

In summary, the simulation ultrasonic radar sensor module introduces a unique blend of hardware and software elements to create a versatile radar system with enhanced visualization capabilities.

5 Conclusion

5.1 Team Work

Our team collaborated effectively to bring the simulation ultrasonic radar sensor module to fruition. The breakdown of responsibilities was as follows:

- Hardware Integration: [Mawaddah Alagha] led the efforts in integrating the embedded board components, including the Arduino microcontroller, servo motor, ultrasonic sensor, and TFT screen. This involved meticulous circuit design, soldering, and testing.
- Software Development: [Ameera Attiah] took charge of the software development aspect. This included writing the Arduino sketch to control the radar system, interface with the ultrasonic sensor, and manage the visual output on the TFT screen.
- Documentation and Report: [Lujain Almarri] contributed to documenting the project's details and compiling this comprehensive report. This involved explaining the hardware setup, detailing the software implementation, and providing insights into the project's significance.

5.2 Challenges

1. Ensuring that the servo motor, ultrasonic sensor, and display work in harmony was challenging. Precise calibration is needed for accurate distance measurement and proper display of the radar sweep.
2. Writing and debugging the code to control the hardware components was complex. This included managing the timing of sensor readings, controlling the servo motor's movement, and updating the display in real-time.
3. Creating a user-friendly interface on a small display screen was difficult. You need to ensure that the radar sweep, distance readings, and other information are presented clearly and accurately.

5.3 Future Work

1. Implementing advanced algorithms to distinguish between different types of obstacles, enhancing the system's object recognition capabilities.
2. Introducing machine learning algorithms for the radar system to adapt and learn from its environment, improving performance in diverse scenarios.
3. Implementing wireless communication capabilities to enable data transmission to external devices, expanding the module's applications in remote monitoring and control.
4. Exploring power-efficient components and strategies to optimize energy consumption, particularly for scenarios where the module operates on battery power.
5. Improving the TFT screen interface by incorporating touch controls and additional visual indicators for enhanced user interaction and a more user-friendly experience.