AN0049
Application Note

AT-SURF-F437 Board Application Note

Introduction

This application note describes various features of the AT32F437 series MCUs with specific examples. Each of the examples is provided with software and hardware design requirements and detailed descriptions.

The example cases give a full description of main features of the AT32F437 series, along with many useful programs. All application examples have passed through MDK5 compiler, so that the users only need download the corresponding codes to the AT-SURF-F437 evaluation board for quick verification.

Applicable products:

Part number	AT32F437 series

Contents

1 Overview 11

2 AT32F437 architecture 12

2.1 ARM Cortex-M4F processor core 13

2.2 BusMatrix 14

3 Environment requirements 15

3.1 Hardware configuration 15

3.2 Software configuration 15

4 Application examples 16

4.1 Example 1: Serial interface print application 16

4.1.1 Introduction 16

4.1.2 Resource requirements 16

4.1.3 Hardware design 16

4.1.4 Software design 17

4.1.5 Download and verify 18

4.2 Example 2: RGB LED application 19

4.2.1 Introduction 19

4.2.2 Resource requirements 20

4.2.3 Hardware design 20

4.2.4 Software design 21

4.2.5 Download and verify 22

4.3 Example 3: Buzzer application 23

4.3.1 Introduction 23

4.3.2 Resource requirements 23

4.3.3 Hardware design 23

4.3.4 Software design 25

4.3.5 Download and verify 26

4.4 Example 4: Touch screen application 27

4.4.1 Introduction 27

4.4.2 Resource requirements 28

4.4.3 Hardware design 28

4.4.4 Software design.. 29

4.4.5 Download and verify... 30

4.5 Example 5: ERTC application 31

4.5.1 Introduction... 31

4.5.2 Resource requirements 31

4.5.3 Hardware design 31

4.5.4 Software design... 32

4.5.5 Download and verify.. 34

4.6 Example 6: Button application 35

4.6.1 Introduction... 35

4.6.2 Resource requirements 35

4.6.3 Hardware design 35

4.6.4 Software design... 36

4.6.5 Download and verify.. 37

4.7 Example 7: ADC sample application 38

4.7.1 Introduction... 38

4.7.2 Resource requirements 38

4.7.3 Hardware design 38

4.7.4 Software design... 39

4.7.5 Download and verify.. 40

4.8 Example 8: DAC output 41

4.8.1 Introduction... 41

4.8.2 Resource requirements 41

4.8.3 Hardware design 41

4.8.4 Software design... 42

4.8.5 Download and verify.. 43

4.9 Example 9: PWM DAC output 44

4.9.1 Introduction... 44

4.9.2 Resource requirements 44

4.9.3 Hardware design 44

4.9.4 Software design... 45

4.9.5 Download and verify.. 46

4.10 Example 10: RS485 communications 47

4.10.1 Introduction... 47

4.10.2 Resource requirements 47
4.10.3 Hardware design 47
4.10.4 Software design 48
4.10.5 Download and verify 50
4.11 Example 11: CAN communication 51
4.11.1 Introduction 51
4.11.2 Resource requirements 51
4.11.3 Hardware design 52
4.11.4 Software design 53
4.11.5 Download and verify 57
4.12 Example 12: Gaming pad application 58
4.12.1 Introduction 58
4.12.2 Resource requirements 58
4.12.3 Hardware design 58
4.12.4 Software design 60
4.12.5 Download and verify 62
4.13 Example 13: EEPROM communication 63
4.13.1 Introduction 63
4.13.2 Resource requirements 64
4.13.3 Hardware design 64
4.13.4 Software design 65
4.13.5 Download and verify 68
4.14 Example 14: TFT LCD application 69
4.14.1 Introduction 69
4.14.2 Resource requirements 69
4.14.3 Hardware design 69
4.14.4 Software design 71
4.14.5 Download and verify 72
4.15 Example 15: SD card communication 73
4.15.1 Information 73
4.15.2 Resource requirements 73
4.15.3 Hardware design 73
4.15.4 Software design 75
4.15.5 Download and verify 77
4.16 Example 16: OTG test 78

4.16.1 Introduction........78
4.16.2 Resource requirements........78
4.16.3 Hardware design........78
4.16.4 Software design........79
4.16.5 Download and verify........81
4.17 Example 17: SDRAM test........82
4.17.1 Introduction........82
4.17.2 Resource requirements........82
4.17.3 Hardware design........82
4.17.4 Software design........84
4.17.5 Download and verify........86
4.18 Example 18: IrDA application........87
4.18.1 Introduction........87
4.18.2 Resource requirements........88
4.18.3 Hardware design........88
4.18.4 Software design........89
4.18.5 Download and verify........90
4.19 Example 19: Low-power mode........91
4.19.1 Introduction........91
4.19.2 Resource requirements........91
4.19.3 Hardware design........91
4.19.4 Software design........92
4.19.5 Download and verify........94
4.20 Example 20: QSPI FLASH application........95
4.20.1 Introduction........95
4.20.2 Resource requirements........95
4.20.3 Hardware design........95
4.20.4 Software design........96
4.20.5 Download and verify........98
4.21 Example 21: QSPI SRAM application........99
4.21.1 Introduction........99
4.21.2 Resource requirements........99
4.21.3 Hardware design........99
4.21.4 Software design........100
4.21.5 Download and verify........102

4.22 Example 22: Audio player application 103
4.22.1 Information 103
4.22.2 Resource requirements 103
4.22.3 Hardware design 103
4.22.4 Software design 105
4.22.5 Download and verify 106
4.23 Example 23: Camera application 107
4.23.1 Introduction 107
4.23.2 Resource requirements 108
4.23.3 Hardware design 108
4.23.4 Software design 109
4.23.5 Download and verify 111
4.24 Example 24: Network communication application 112
4.24.1 Introduction 112
4.24.2 Resource requirements 112
4.24.3 Hardware design 112
4.24.4 Software design 114
4.24.5 Download and verify 116

5 Revision history 119

List of tables

Table 1. Hardware resources 16
Table 2. Hardware resources 20
Table 3. Hardware resources 23
Table 4. PCA9555 23
Table 5. Resistive vs. capacitive touch screens 27
Table 6. Hardware resources 28
Table 7. Hardware resources 31
Table 8. Hardware resources 35
Table 9. Hardware resource 38
Table 10. Hardware resources 41
Table 11. Hardware resources 44
Table 12. Hardware resources 47
Table 13. Hardware resources 52
Table 14. Hardware resources 58
Table 15. PCA9555 58
Table 16. Hardware resources 64
Table 17. Hardware resources 69
Table 18. PCA9555 connector 70
Table 19. Hardware resources 73
Table 20. PCA9555 73
Table 21. Hardware resources 78
Table 22. SDRAM vs. SRAM 82
Table 23. Hardware resources 82
Table 24. Hardware resources 88
Table 25. Hardware resources 91
Table 26. Hardware resources 95
Table 27. Hardware resources 99
Table 28. PCA9555 103
Table 29. Hardware resources 103
Table 30. Hardware resources 108
Table 31. Hardware resources 112
Table 32. PCA9555 112
Table 33. Document revision history 119

List of figures

Figure 1. Internal block diagram of Cortex-M4F 13
Figure 2. Block diagram of AHB BusMatrix 14
Figure 3. AT32 SUFR Board 15
Figure 4. Serial interface schematic diagram 16
Figure 5. Test result 18
Figure 6. RGB-LED schematic diagram 20
Figure 7. PCA9555 schematic diagram 24
Figure 8. Buzzer schematic diagram 24
Figure 9. Schematic diagram of touch screen 28
Figure 10. Test result 30
Figure 11. Schematic diagram of external high-speed crystal oscillator 31
Figure 12. Schematic diagram of battery powered circuit 32
Figure 13. Test result 34
Figure 14. Schematic diagram of button circuit 35
Figure 15. Test result 37
Figure 16. Schematic diagram of adjustable resistor 38
Figure 17. Test result 40
Figure 18. Schematic diagram of DAC circuit 41
Figure 19. Test result 43
Figure 20. PWM DAC schematic diagram 44
Figure 21. Test result 46
Figure 22. Test process 47
Figure 23. Schematic diagram of 485 chip 48
Figure 24. PC side serial port assistant 50
Figure 25. SUFR side 50
Figure 26. Schematic diagram of CAN1 52
Figure 27. Schematic diagram of CAN2 52
Figure 28. Schematic diagram of Jumper 52
Figure 29. Test result 57
Figure 30. Five-direction joystick principle 58
Figure 31. Schematic diagram of PCA9555 connector 59
Figure 32. Schematic diagram of gaming pad 59
Figure 33. Test result 62
Figure 34. Application example of I^2C bus 63

Figure 35. I2C bus data format....64
Figure 36. Schematic diagram of 24C02....64
Figure 37. Test result....68
Figure 38. Schematic diagram of TFT LCD....70
Figure 39. Schematic diagram of PCA9555....70
Figure 40. Test result....72
Figure 41. Schematic diagram of SD card....74
Figure 42. Schematic diagram of PCA9555....74
Figure 43. Test result....77
Figure 44. Schematic diagram of OTG circuit....79
Figure 45. PC side serial port assistant....81
Figure 46. SUFR board....81
Figure 47. Schematic diagram of SDRAM....84
Figure 48. Test result....86
Figure 49. NEC start bit transfer....87
Figure 50. NEC bit transfer format....87
Figure 51. NEC data frame format....88
Figure 52. Schematic diagram of infrared receiver....88
Figure 53. Test result....90
Figure 54. Schematic diagram of button circuit....91
Figure 55. Test result....94
Figure 56. Schematic diagram of QSPI FLASH....95
Figure 57. Test result....98
Figure 58. Schematic diagram of QSPI SRAM....99
Figure 59. Schematic diagram of QSPI SRAM Jumper....99
Figure 60. Test result....102
Figure 61. Schematic diagram of WM8988....104
Figure 62. Schematic diagram WM8988 jumper....104
Figure 63. Schematic diagram WM8988 jumper....104
Figure 64. Test result....106
Figure 65. Camera....107
Figure 66. Schematic diagram of camera....109
Figure 67. Test result....111
Figure 68. Schematic diagram of PHY....113
Figure 69. Schematic diagram of RJ45....113

Figure 70. Schematic diagram of PCA9555 114
Figure 71. Computer internet configuration 117
Figure 72. PC side test result 117
Figure 73. SUFR board side test result 118

1 Overview

The ultra-high performance AT32F437 series is based on 32-bit ARM® Cortex®-M4 core operating at a frequency of up to 288 MHz. The device features an embedded single precision floating-point unit (FPU), digital signal processor (DSP), memory protection unit (MPU), rich peripherals and flexible clock control mechanism for a wide range of applications. Additionally, it embeds up to 4032 KB Flash memory and 512 KB SRAM, far beyond its counterparts in terms of performance.

The AT32F437 series is equipped with a security library (sLib), allowing users to set any part of the internal Flash memory as a password-protected area. At the same time, the sLib mechanism makes it more convenient and secure for solution providers to program their core algorithms in the sLib area and provide a room for them to conduct secondary development.

Besides, AT32F437 series incorporates 2x OTG controllers (Xtal-less in device mode), 2x QSPIs for external SPI Flash memory or SPI RAM extension, 8x UARTs, 2x CANs, 4x SPIs/I^{2}Ss (2x full-duplex), 3x high-speed ADC engines (5.33 Msps), 8~14-bit digital video parallel interface (DVP), XMC for extended SDRAM, SRAM and PSRAM, and IEEE-802.3 10/100Mbps Ethernet port controller for IoT applications, greatly improving the reliability while lowering the costs.

AT32F437 series perform well in the temperature range of -40 to 105 °C. It also provides a variety of chips for selection in response to diverse memory demand. With its powerful on-chip resources, higher integration and cost-effectiveness, the AT32F437 series products are best suited to the applications that seek for higher computation power and larger memory, including industrial automation, motor control, IoT and consumer electronics.

Strengths with AT32F437 series:

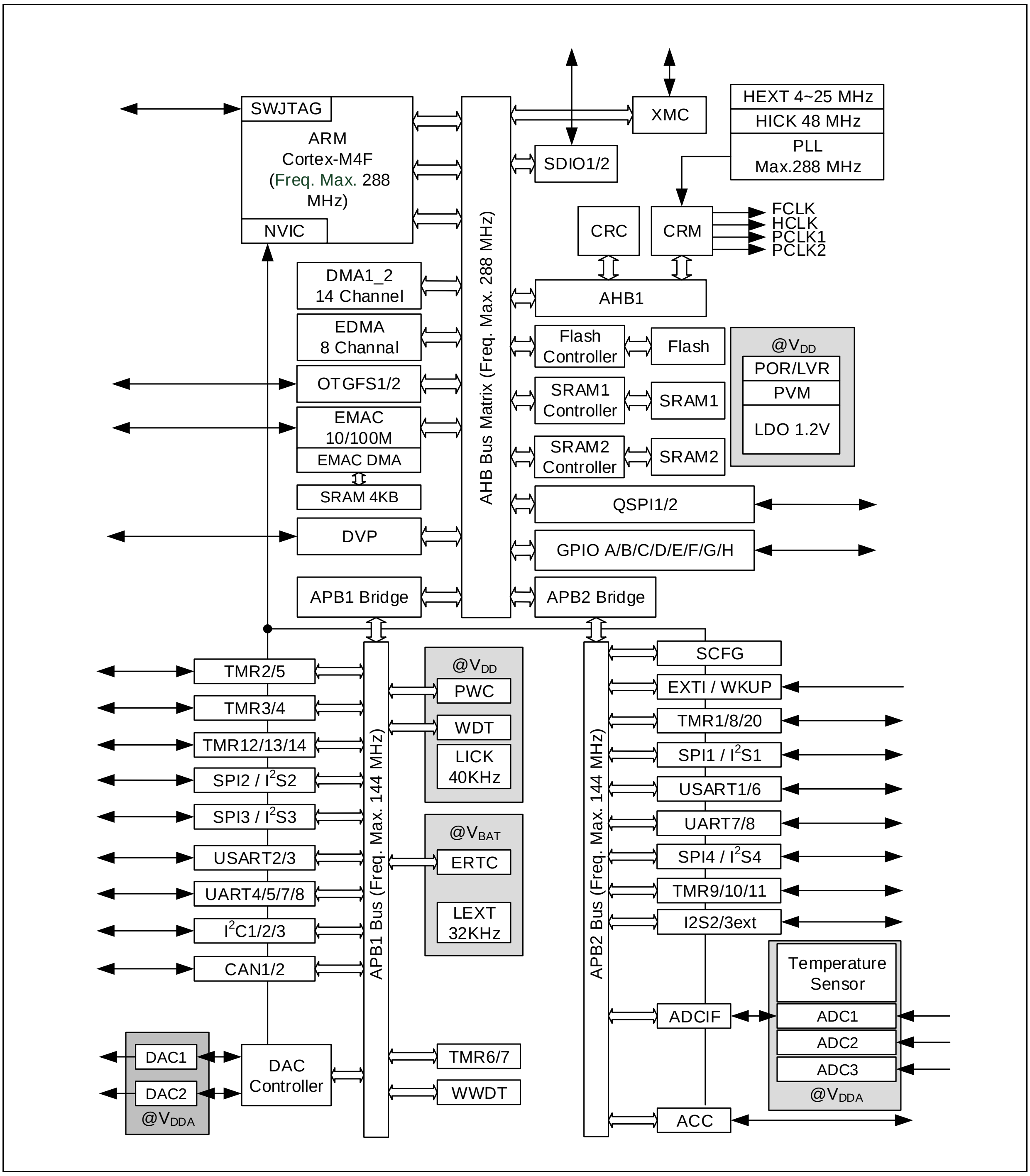
- Maximum frequency: 288 MHz
- Operating voltage: 2.6-3.6 V
- Operating temperature: -40-105 °C
- Main features
  Up to 4032KB Flash/512KB SRAM, 10/100 Mbps Ethernet, SDRAM, dual QSPI, dual OTG, DVP and 5.33 Msps ADC
- Main applications: IoT gateway, serial server, micro printer, stage lighting, HMI, LED display, QR code scanner, surveillance, industrial control, 5G

This guideline offers many examples to help readers get a quick understand of codes.
The examples in this document cover main features of the AT32F437 series and offer many useful programs. All examples have passed through MDK5 compiler, so that the users only need download programs to the AT-SURF-F437 evaluation board and get a quick start with their verification.
Due to space reasons, this application note does not further explain the basic functions of peripherals.
Reference documents:

- AT32F437 product selector on Artery official website
- AT32 MCU APnote
- AT32 MCU FAQ
- AT32 MCU Sample Code

2 AT32F437 architecture

AT32F437 series microcontrollers incorporate a 32-bit ARM® Cortex®-M4F processor core, multiple16-bit and 32-bit timers, Infrared Transmitter (IRTMR), DMA controller, EDMA controller, ERTC, communication interfaces such as SPI, QSPI, I²C, USART/UART and SDIO, CAN bus controller, external memory controller (XMC), USB2.0 full-speed interface, Ethernet MAC, parallel digital camera interface, HICK with automatic clock calibration (ACC), 12-bit ADC, 12-bit DAC, programmable voltage monitor (PVM), rich peripherals and memories. The Cortex®-M4F processer supports enhanced high-performance DSP instruction set, including extended single-cycle 16-bit/32-bit multiply accumulator (MAC), dual 16-bit MAC instructions, optimized 8-bit/16-bit SIMD operation and saturation operation instructions, and single-precision (IEEE-754) floating point unit (FPU), as shown in the figure below.

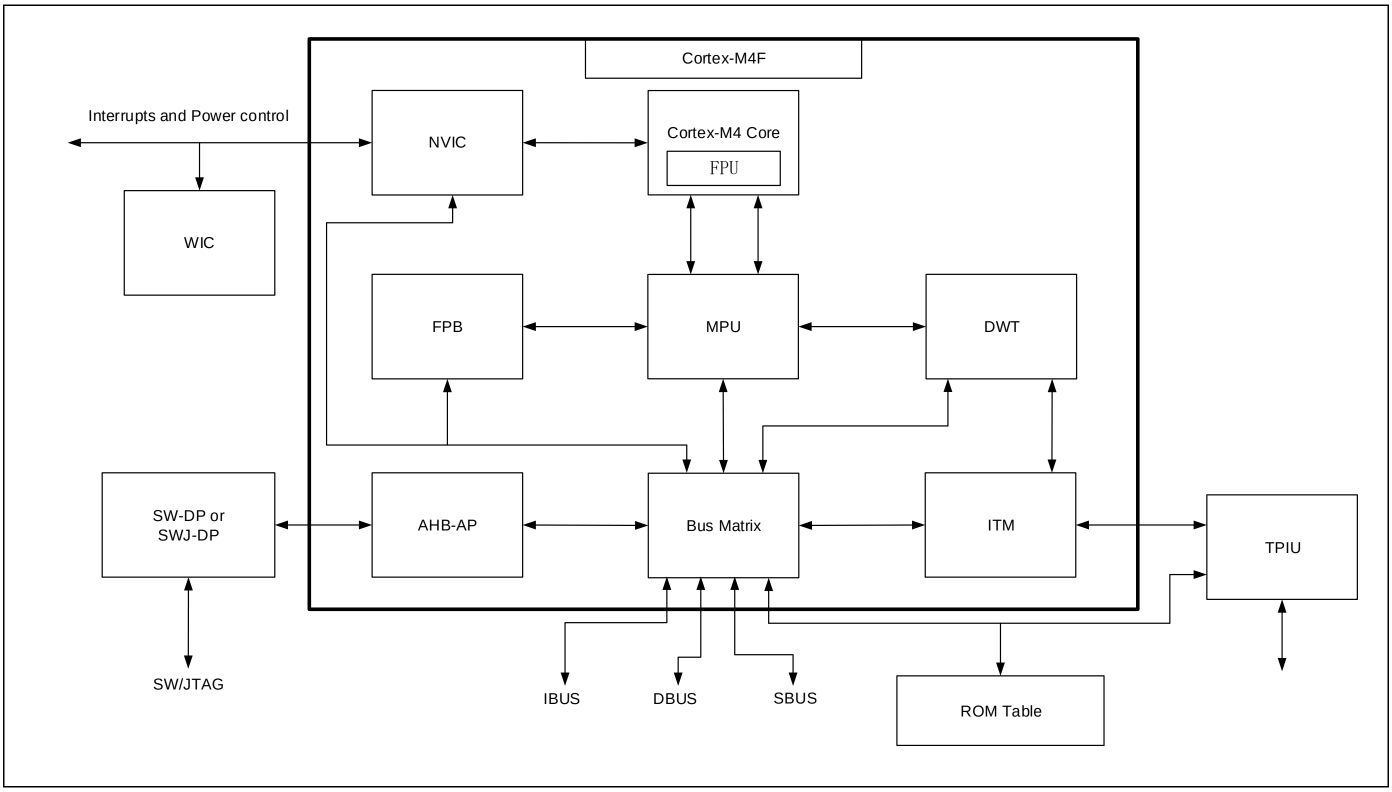


2.1 ARM Cortex-M4F processor core

Cortex®-M4F processor is a low-power consumption processor featuring low gate count, low interrupt latency, and low-cost debug. It supports DSP instruction set and FPU, and thus is best suited to the embedded applications that require quicker response to interrupts. Cortex®-M4F processor is based on ARMv7-M architecture, supporting both Thumb instruction set and DSP instruction set.

Figure 1 shows the internal block diagram of Cortex®-M4F processor. Please refer to *ARM Cortex® -M4 Technical Reference Manual* for more information.

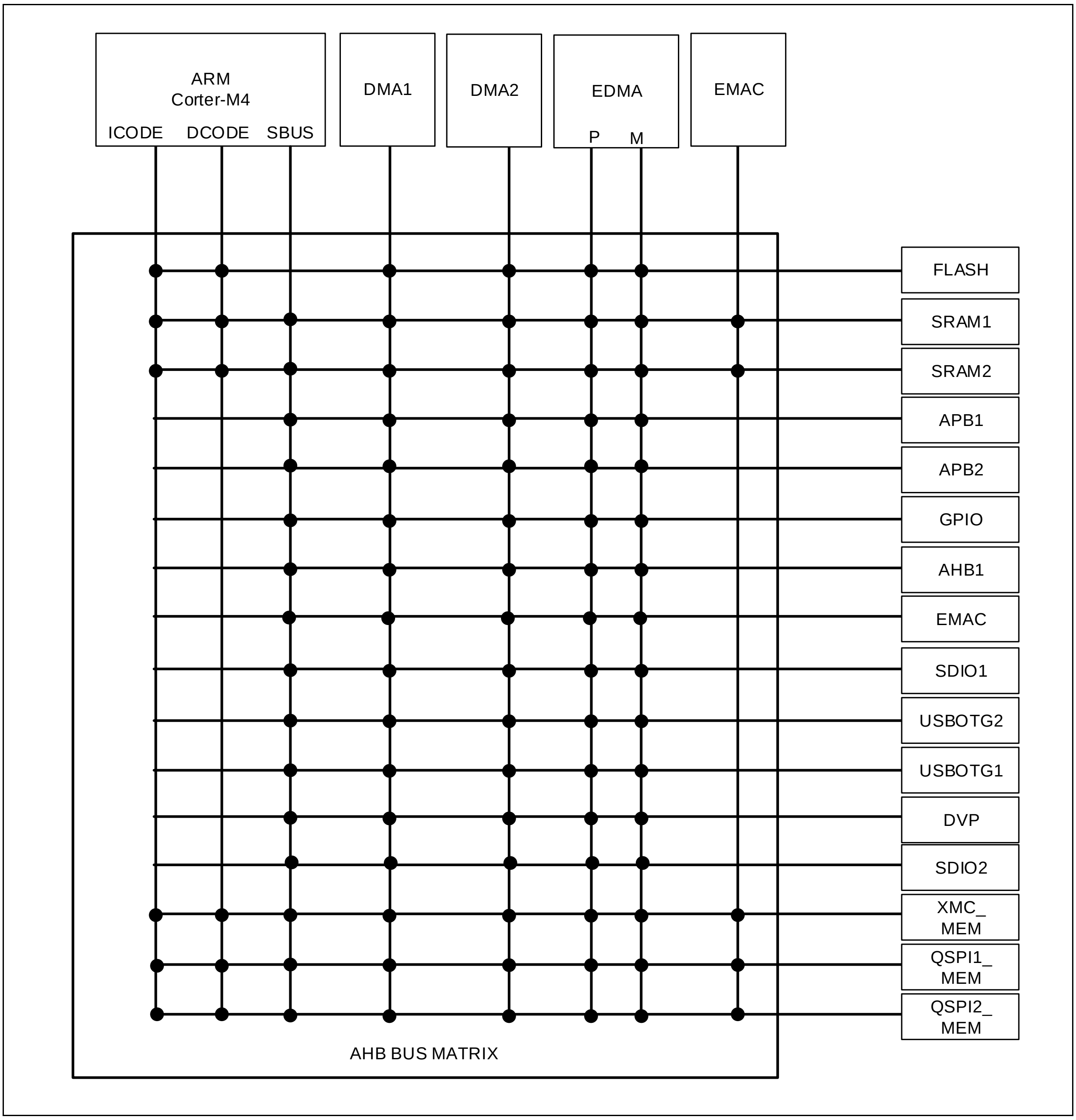
Figure 1. Internal block diagram of Cortex-M4F



2.2 BusMatrix

Figure 2 shows the block diagram of AHB BusMatrix.

Figure 2. Block diagram of AHB BusMatrix



3 Environment requirements

3.1 Hardware configuration

The AT32 SUFR board is equipped with many chips to better demonstrate the functions of peripherals of the AT32F437 series. Each peripheral feature is showcased with a specific example case, with the aim of making it quicker and easier for users to learn about the AT32F437 series and speeding up development.

Figure 3. AT32 SUFR Board

3.2 Software configuration

For AT32 SUFR board, the demo program files are organized as follows:

Libraries: AT32 library program

Middlewares: middleware program

project\at_surf_f437_board: drivers for peripherals

project\at_surf_f437\examples: example cases

project\at_surf_f437\applications: application cases

4 Application examples

4.1 Example 1: Serial interface print application

4.1.1 Introduction

The serial interface print is used to output key information during debugging and development. It is usually done by re-directing the output of the printf function to a serial interface and then calling the printf function to print information.

4.1.2 Resource requirements

- Hardware resources

  AT-SURF-F437 Board

- Software resources

  AT32F435_437_Firmware_Library_V2.x.x\project\at_sufr_f437\examples\uart_printf

4.1.3 Hardware design

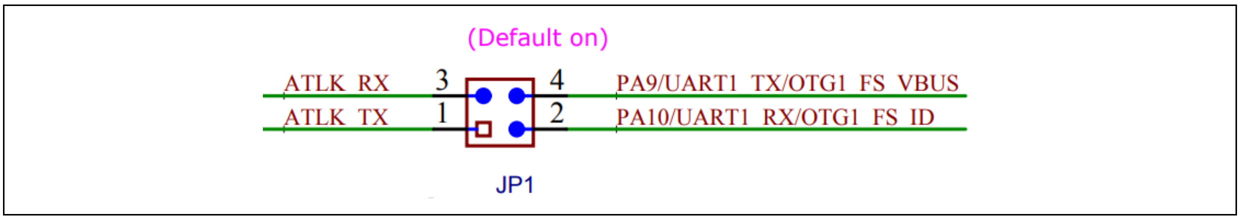
The hardware resources used in the application example are LCD and serial interface 1.
Table 1 shows the corresponding pins:

Table 1. Hardware resources

No.	PIN Name	Peripheral function	Description
1	PA9	USART1 TX	serial interface transmit

Figure 4 shows the schematic diagram of the serial interface.

Figure 4. Serial interface schematic diagram



4.1.4 Software design

1) Serial interface print test
   - Initialize serial interface
   - Print information via the interface per second
2) Code
   - Main function

```
int main(void)
{
   /* Initialize system clock */
   system_clock_config();

   /* Initialize interrupt priority group */
   nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

   /* Initialize delay function */
   delay_init();

   /* Initialize LCD */
   lcd_init(LCD_DISPLAY_VERTICAL);

   /* Initialize serial interface */
   uart_print_init(115200);

   /* Show information */
   lcd_string_show(10, 20, 200, 24, 24, (uint8_t *)"UART Print Test");

   while(1)
   {
      delay_ms(1000);

      /* Serial interface print information */
      printf("Artery 2022 \r\n");
   }
}
```

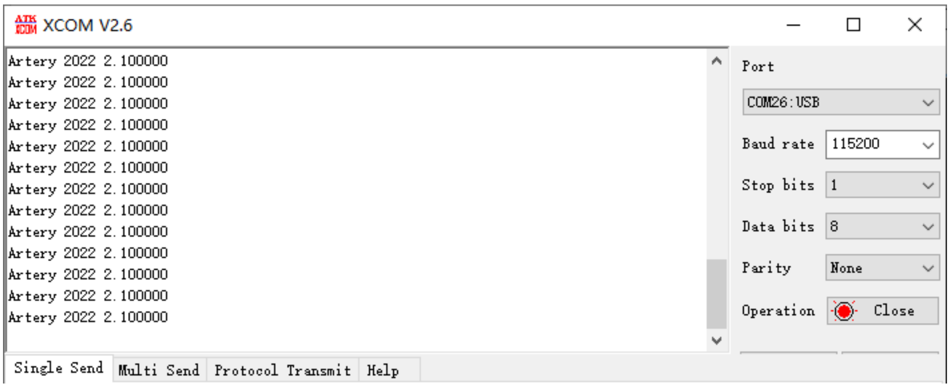
   - void uart_print_init(uint32_t baudrate) function

```
/**
   * @brief   initialize uart
   * @param   baudrate: uart baudrate
   * @retval none
   */
void uart_print_init(uint32_t baudrate)
```

4.1.5 Download and verify

- Connect the serial interface to PC, open serial port assistant, and print information once per second.

Figure 5. Test result



4.2 Example 2: RGB LED application

4.2.1 Introduction

RGB refers to the three primary colors, red, green, and blue. Different colors can be generated by powering each LED. Besides, blue LED can work with yellow phosphor, and ultraviolet LED with RGB phosphor. The mixed color effects can be achieved by mixing the three primary colors in different proportions. That's why in most cases, we can notice that some LED backlights are very clear and bright, and even comparable to the high-definition Television set.

RGB is designed based on the principle of color luminescence. Simply speaking, the color mixing mode acts like three lights in red, green and blue. When the lights of red, green and blue are combined, it produces mixing brightness that equals to those of three lights. Thus, the more mixed, the stronger the brightness, that is so-called additive color mixture.

With a combination of the red, green and blue colors, the center white area has the highest brightness. Each of the three colors, red, green and blue, ranges from 0 to 255 in terms of brightness. At 0, the "light" is the dimmest — it is turned off, while at 255, the "light" is the brightest. When the tricolor is set with the same grayscale values, it ends up forming a gray tone that features different grayscale. In other words, when the three primary colors are set at 0 in grayscale, it turns out black, the darkest color. When they are set at 255, it turns out white, the brightest color.

The RGB color model is an additive one. Red, Green and Blue values (All light rays are reflected back to eyes) are combined to reproduce white color. The additive color is widely used for lighting, Television sets and computer screens. For instance, the colors on computer screens are generated by red, green and blue phosphors that emit lights. In fact, a vast majority of visible spectrum can be seen a mixture of red, green and blue (RGB) colors based on different proportions and intensity. As long as such colors are combined, it turns out cyan, magenta and yellow.

How to control RGB LED

- **Option 1: use GPIOs to control RGB LED (regular mode)**
  PB10 is used to turn on/off RED;
  PD13 is used to turn on/off GREEN
  PB5 is used to turn on/off BLUE.
  Besides, different colors can be generated through a combination of two or three GPIOs.
  This method is easy to control as it only use GPIOs, without the need of any other peripherals.
  But the brightness of LED cannot be adjusted in this mode.
- **Option 2: Use TMR for RGB LED control (breathing light mode)**
  TMR2 channel 3 is used to control RED ON/OFF and its brightness
  TMR4 channel 2 is used to control GREEN ON/OFF and its brightness
  TMR3 channel 2 is used to control BLUE ON/OFF and its brightness
  Different colors can be generated through a combination of different channels.
  RGB LED brightness depends on the duty cycle of TMR channels.
  This method requires both GPIO and TMR. And colors and brightness are adjustable.

The example concentrates on the first control mode.

4.2.2 Resource requirements

- Hardware resources
  AT-SURF-F437 Board
- Software resources
  AT32F435_437_Firmware_Library_V2.x.x\project\at_start_f437\ examples\rgb_led

4.2.3 Hardware design

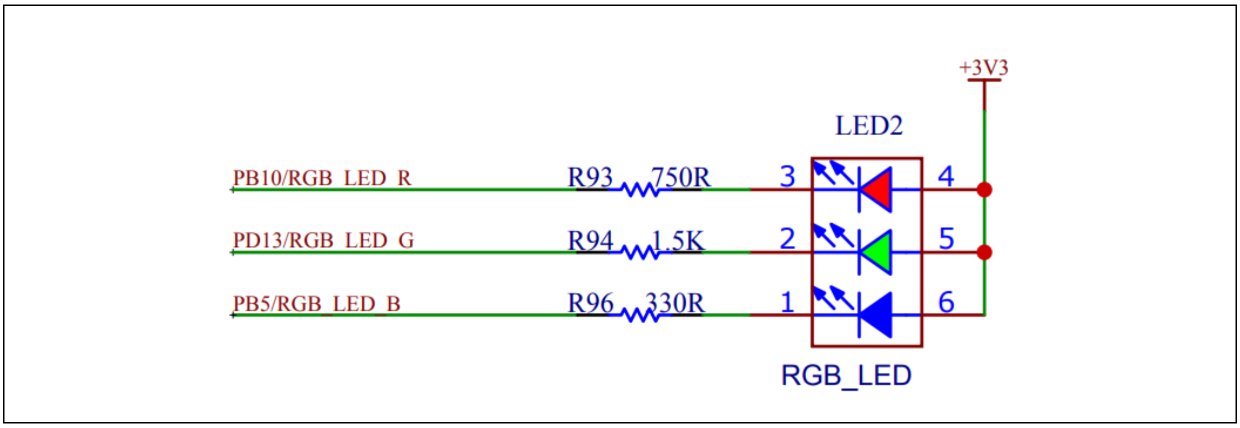
The hardware resources in the application example are LCD and RGB-LED0
Table 2 shows the corresponding pins.

Table 2. Hardware resources

No.	PIN Name	Peripheral function	Description
1	PB10	GPIO	RGB LED R
2	PD13	GPIO	RGB LED G
3	PB5	GPIO	RGB LED B

Figure 6 shows the schematic diagram of RGB-LED.

Figure 6. RGB-LED schematic diagram



4.2.4 Software design

1) Use GPIO to control RGB LED
   - Set GPIO as push-pull output
   - Select GPIO output high/low to control RGB LED
2) Code
   - main function

```
int main(void)
{
    /* Initialize system clock */
    system_clock_config();

    /* Initialize interrupt priority group*/
    nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

    /* Initialize delay function */
    delay_init();

    /* Initialize LCD */
    lcd_init(LCD_DISPLAY_VERTICAL);

    /* Show information */
    lcd_string_show(10, 20, 200, 24, 24, (uint8_t *)"RGB LED Test");

    /* Initialize RGB LED */
    rgb_led_init();

    /* LED OFF */
    rgb_led_off();

    rgb_led_set(RGB_LED_GBLUE);
    delay_ms(500);

    rgb_led_set(RGB_LED_PURPLE);
    delay_ms(500);

    rgb_led_set(RGB_LED_WHITE);
    delay_ms(500);

    rgb_led_set(RGB_LED_YELLOE);
    delay_ms(500);

    while(1)
    {
        rgb_led_toggle(RGB_LED_RED);
        delay_ms(500);
```

```
    }
}
```

- void rgb_led_init(void)

```
/**
  * @brief   initialize rgb led
  * @param   none
  * @retval none
  */
void rgb_led_init(void)
```

- void rgb_led_set(uint16_t color)

```
/**
  * @brief   set rgb led color,and turn on.
  * @param   color: rgb led color
  *              this parameter can be one of the following values:
  *              - RGB_LED_RED
  *              - RGB_LED_GREEN
  *              - RGB_LED_BLUE
  *              - RGB_LED_YELLOE
  *              - RGB_LED_GBLUE
  *              - RGB_LED_PURPLE
  *              - RGB_LED_WHITE
  * @retval flag_status (SET or RESET)
  */
void rgb_led_set(uint16_t color)
```

- void rgb_led_off(void) function

```
/**
  * @brief   turn off reg led.
  * @param   none
  * @retval none
  */
void rgb_led_off(void)
```

- void rgb_led_toggle(uint16_t color) function

```
/**
  * @brief   reg led toggle.
  * @param   none
  * @retval none
  */
void rgb_led_toggle(uint16_t color)
```

4.2.5 Download and verify

- After power-on, RGB LED shifts from one color to another in regular mode.

4.3 Example 3: Buzzer application

4.3.1 Introduction

The AT32-SUFR board is equipped with an easy-to-use buzzer. The buzzer goes off as long as it is powered.

The buzzer on the AT32-SUFR board is not directly connected to the MCU IO but to an IO extension chip PCA9555. The PCA9555 is linked to the MCU through I^2C bus. The PCA9555 output register is configured by MCU through I^2C bus in order to control IO level on the PCA9555. Just like AT32 MCU GPIO output control logic, writing 0 to the output register will trigger the corresponding IO output low level; writing 1 to the output register will trigger the corresponding IO output high level.

4.3.2 Resource requirements

- Hardware resources

  AT-SURF-F437 Board
- Software resources

  AT32F435_437_Firmware_Library_V2.x.x\project\at_sufr_f437\examples\buzz

4.3.3 Hardware design

The hardware resources used in the application example are buzzer and PCA9555.
Table 3 shows the corresponding pin.

Table 3. Hardware resources

No.	PIN Name	Peripheral function	Description
1	PH2	I2C2 SCL	PCA9555 SCL line
2	PH3	I2C2 SDA	PCA9555 SDA line
3	PG3	GPIO	PCA9555 INT line

Table 4. PCA9555

No.	PIN Name	Pin function	Description
1	IO0_4	Buzzer control	-

Figure 7 shows the schematic diagram of PCA9555.

Figure 7. PCA9555 schematic diagram

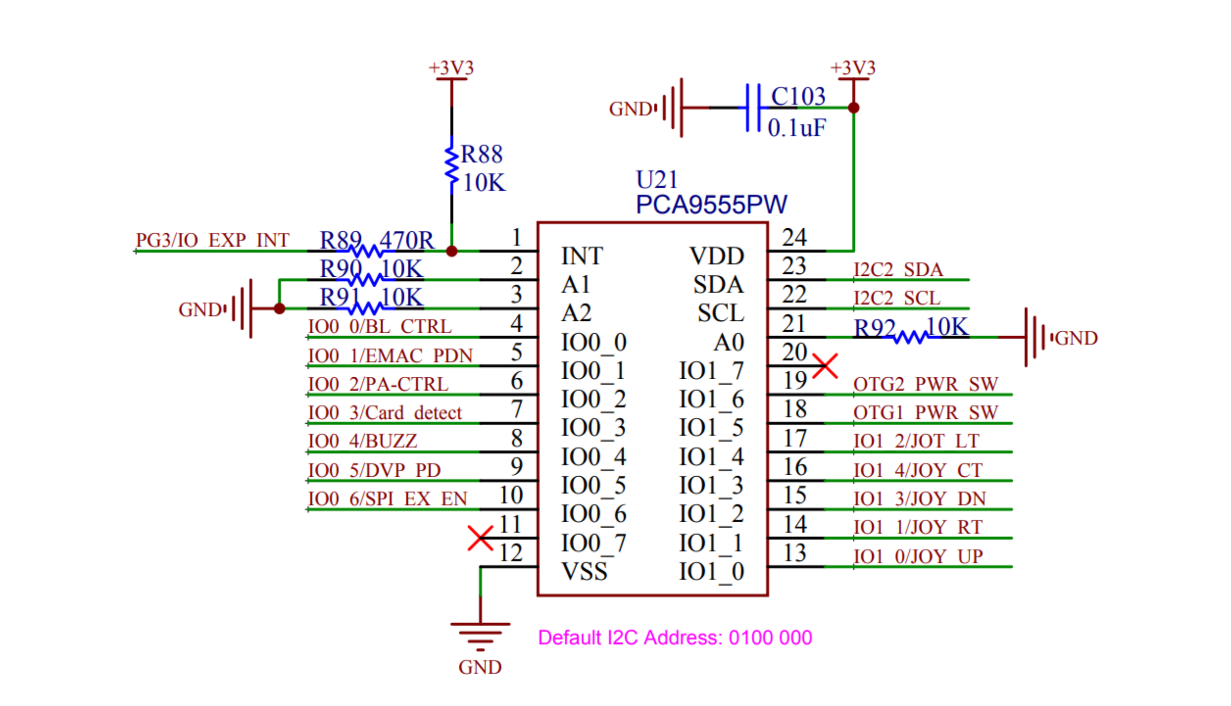
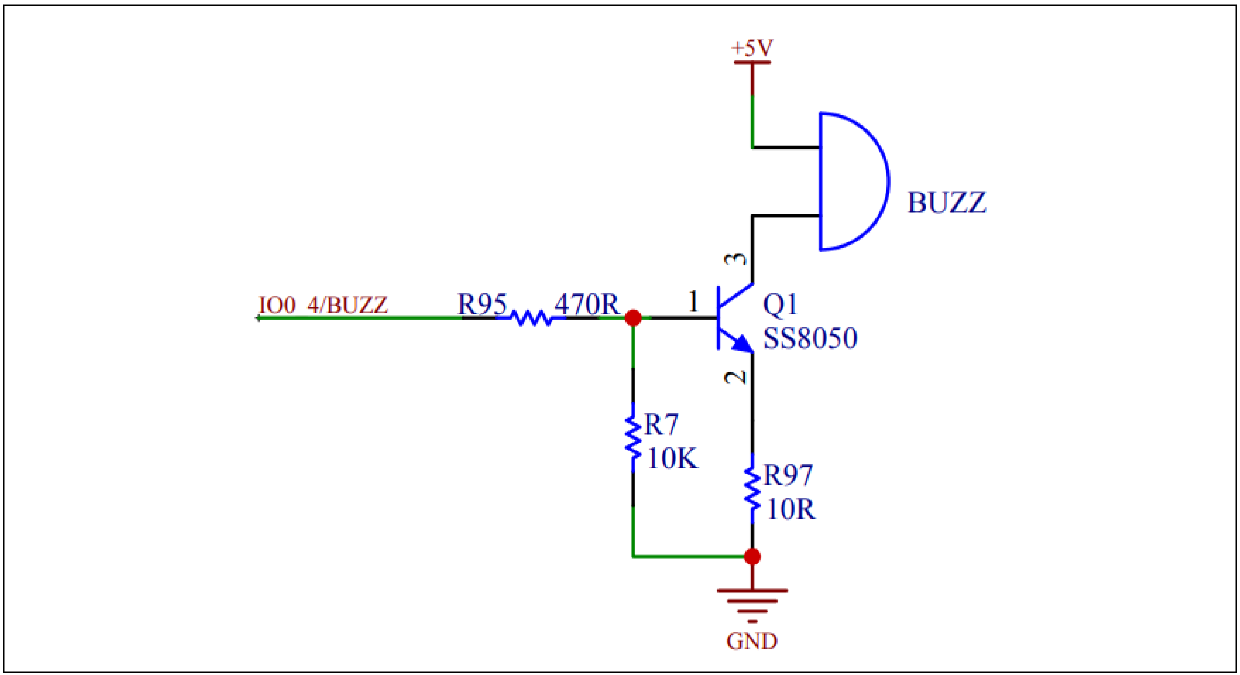


Figure 8. Buzzer schematic diagram



4.3.4 Software design

1) Buzzer test
   - Initialize I²C interface
   - Set PCA9555 IO (connected to buzzer) port as output mode
   - Drive buzzer through PCA9555 IO
2) Code
   - main function

```
int main(void)
{
   /* Initialize system clock */
   system_clock_config();

   /* Initialize interrupt priority group */
   nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

   /* Initialize delay function */
   delay_init();

   /* Initialize LCD */
   lcd_init(LCD_DISPLAY_VERTICAL);

   /* Initialize buzzer */
   buzz_init();

   /* Show information*/
   lcd_string_show(10, 20, 200, 24, 24, (uint8_t *)"Buzz Test");

   while(1)
   {
      /* Turn on buzzer */
      BUZZ_ON();

      delay_ms(100);

      /* Turn off buzzer*/
      BUZZ_OFF();

      delay_ms(5000);

      /* Turn on buzzer */
      BUZZ_ON();

      delay_ms(40);
```

```
        /* Turn off buzzer */
        BUZZ_OFF();

        delay_ms(5000);

    }
```

- void buzz_init(void)

```
/**
  * @brief   buzz init.
  * @param   none.
  * @retval none.
  */
void buzz_init(void)
```

4.3.5 Download and verify

- Buzzer makes long and short sounds alternately.

4.4 Example 4: Touch screen application

4.4.1 Introduction

There are two common touch screens, capacitive touch screen and resistive touch screen.

— **Resistive touch screen:**
When touching the screen by a finger, a micro shape change is detected on the screen. With the touch screen, the x/y location of the point of contact is then converted into the voltage of the x/y coordinates. The X and Y coordinates are calculated by collecting the voltages through a dedicated touch chip. The MCU then gets the accurate location of the point of contact by reading the touch chip.

— **Capacitive touch screen:**
When touching the screen by a finger, change on the capacitance of the point of contact takes place. The accurate point of contact is calculated by a dedicated touch chip. The MCU gets the location of the point of contact by reading the touch chip.

Table 5. Resistive vs. capacitive touch screens

No.	Item	Capacitive touch screen	Resistive touch screen
1	Environmental adaptability	Prone to water, dust and other external factors	Can withstand dust, water and harsh environment
2	Transmittance and definition	Superior to resistive touch screen	Low light transmittance
3	Touch accuracy	Difficult to touch small targets (limited by finger size)	Accuracy up to a single display pixel
4	Touch sensitivity	Sensitive to the finger touch Finger cots and gloves are not supported	Pressure is applied to the screen to enable contact among layers of the screen
5	Multi-point touch control	Support	Not support
6	Calibration	No calibration is required before use	Calibration is required before use
7	Price	10~50% more expensive than resistive screen	Inexpensive

The AT32-SUFR uses a capacitive touch screen, and touch chip GT911. The touch chip is connected to AT32 MCU through I^2C bus.

4.4.2 Resource requirements

- Hardware resource
  AT-SURF-F437 Board
- Software resource
  AT32F435_437_Firmware_Library_V2.x.x\project\at_sufr_f437\examples\touch

4.4.3 Hardware design

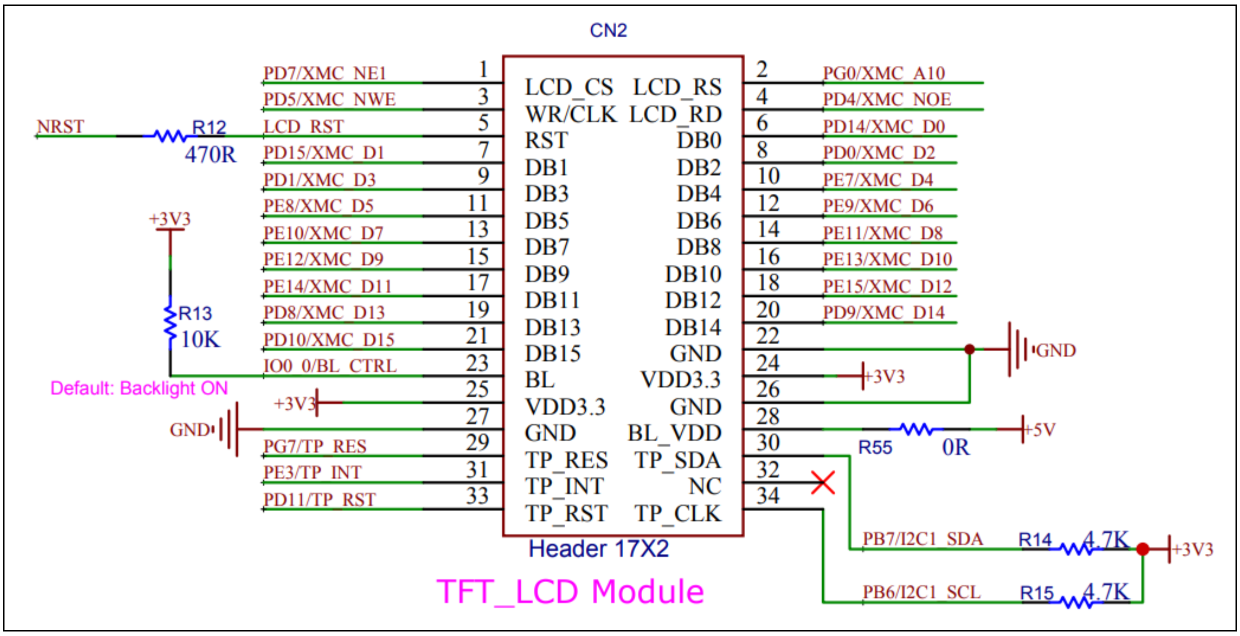
The hardware resource used in the application example is LCD.
Table 6 shows the corresponding pins:

Table 6. Hardware resources

No.	PIN Name	Peripheral function	Description
1	PB6	I2C1 SCL	SCL
2	PB7	I2C1 SDA	SDA
3	PE3	TP INT	Touch event interrupt line
4	PD11	TP RST	Touch chip reset line

Figure 9 shows the schematic diagram of touch screen circuit.

Figure 9. Schematic diagram of touch screen



4.4.4 Software design

1) Touch test
   - Initialize touch chip
   - When the point of contact is detected, the coordinates will be shown on LCD
2) Code
   - main function

```
int main(void)
{
   /* Initialize system clock */
   system_clock_config();

   /* Initialize interrupt priority group*/
   nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

   /* Initialize delay function */
   delay_init();

   /* Initialize LCD */
   lcd_init(LCD_DISPLAY_VERTICAL);

   /* Initialize touch chip*/
   touch_init(TOUCH_SCAN_VERTICAL);

   /* Show information*/
   lcd_string_show(10, 20, 200, 24, 24, (uint8_t *)"Touch Test");

   while(1)
   {

      lcd_string_show(10, 60, 200, 24, 24, (uint8_t *)"X:");

      lcd_string_show(10, 90, 200, 24, 24, (uint8_t *)"Y:");

      /* Read touch point coordinates */
      if(touch_read_xy(x_dot, y_dot) == SUCCESS)
      {
         /* Show X coordinate */
         lcd_num_show(40, 60, 200, 24, 24, x_dot[0], 3);

         /* Show Y coordinate */
         lcd_num_show(40, 90, 200, 24, 24, y_dot[0], 3);
      }  }
}
```

- error_status touch_init(void)

```
/**
  * @brief   this function is init touch.
  * @param   none
  * @retval function execution result.
  *             - SUCCESS.
  *             - ERROR.
  */
error_status touch_init(void)
```

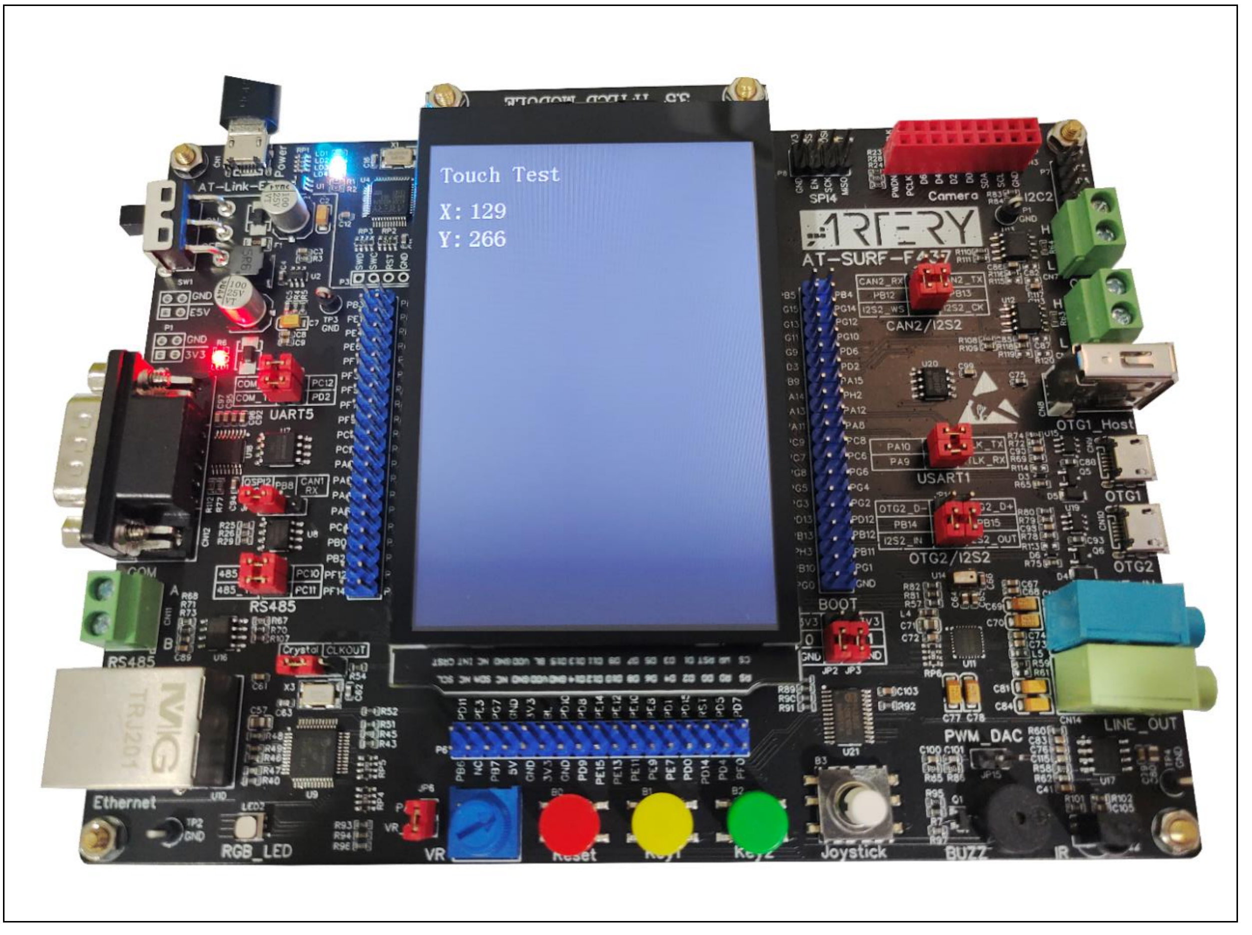
- error_status touch_read_xy(uint16_t *x, uint16_t *y)

```
/**
  * @brief   this function is read data from touch.
  * @param   x/y : coordinate value.
  * @retval function execution result.
  *             - SUCCESS.
  *             - ERROR.
  */
error_status touch_read_xy(uint16_t *x, uint16_t *y)
```

4.4.5 Download and verify

- LCD display shows the coordinates of the contact point as soon as a touch event is detected.

Figure 10. Test result



4.5 Example 5: ERTC application

4.5.1 Introduction

The real-time clock (ERTC) is an independent BCD counter. The ERTC provides a time-of-day clock/calendar. It is located in the battery powered domain (BPR). As long as the BPR is powered, the ERTC never stops, regardless of the device status (system reset and VDD power-down).

ERTC main features:

— Calendar with seconds, minutes, hours, day, date, month and year

— Alarm A and Alarm B

— Periodic wakeup

— Tamper detection

— Calibration: smooth and coarse calibration

This example case demonstrates how to use ERTC calendar feature, and how to show the calendar on LCD display.

4.5.2 Resource requirements

- Hardware resources

  AT-SURF-F437 Board

- Software resources

  AT32F435_437_Firmware_Library_V2.x.x\project\at_sufr_f437\examples\calendar

4.5.3 Hardware design

Hardware resources in the application example include 32768Hz crystal and battery.
Table 7 gives the corresponding pins.

Table 7. Hardware resources

No.	PIN Name	Peripheral function	Description
1	VBAT	VBAT	Battery powered pin
2	PC14	OSC_IN	Crystal oscillator pin
3	PC15	OSC_OUT	Crystal oscillator pin

Figure 11 shows the schematic diagram of external low-speed crystal oscillator.

Figure 11. Schematic diagram of external high-speed crystal oscillator

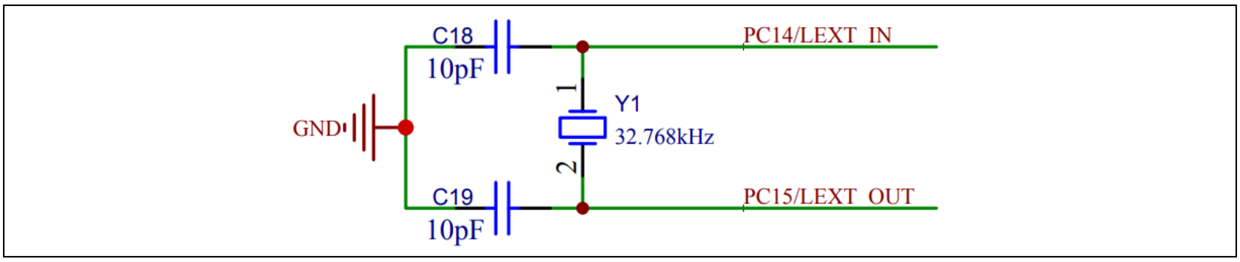
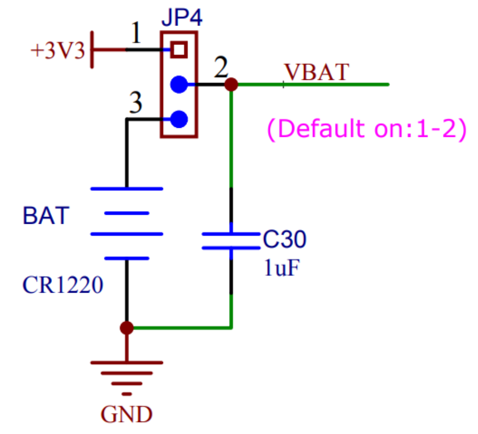


Figure 12. Schematic diagram of battery powered circuit



4.5.4 Software design

1) Calendar test
   - Initialize ERTC
   - Show calendar on LCD display
2) Code
   - main function

```
int main(void)
{
    uint8_t temp = 0;
    ertc_time_type time;

    /* Initialize system clock */
    system_clock_config();

    /* Initialize interrupt priority group */
    nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

    /* Initialize delay function */
    delay_init();

    /* Initialize LCD */
    lcd_init(LCD_DISPLAY_VERTICAL);

    /* Initialize calendar */
    calendar_init();

    /* Show information */
    lcd_string_show(10, 20, 200, 24, 24, (uint8_t *)"Calendar Test");

    /* Display symbols */
    lcd_string_show(10, 60, 200, 24, 24, (uint8_t *)"      -  -        :  :  ");
```

```
    while(1)
    {
      /* Get current time */
      ertc_calendar_get(&time);

      if(temp != time.sec)
      {
        temp = time.sec;

        /*Show year information*/
        lcd_num_show(10, 60, 200, 24, 24, time.year + 2000, 4);

        /* Show month information */
        lcd_num_show(70, 60, 200, 24, 24, time.month, 2);

        /* Show day information */
        lcd_num_show(106, 60, 200, 24, 24, time.day, 2);

        /* Show hour information */
        lcd_num_show(142, 60, 200, 24, 24, time.hour, 2);

        /* Show minute information */
        lcd_num_show(178, 60, 200, 24, 24, time.min, 2);

        /* Show second information */
        lcd_num_show(214, 60, 200, 24, 24, time.sec, 2);
      }
    }
}
```

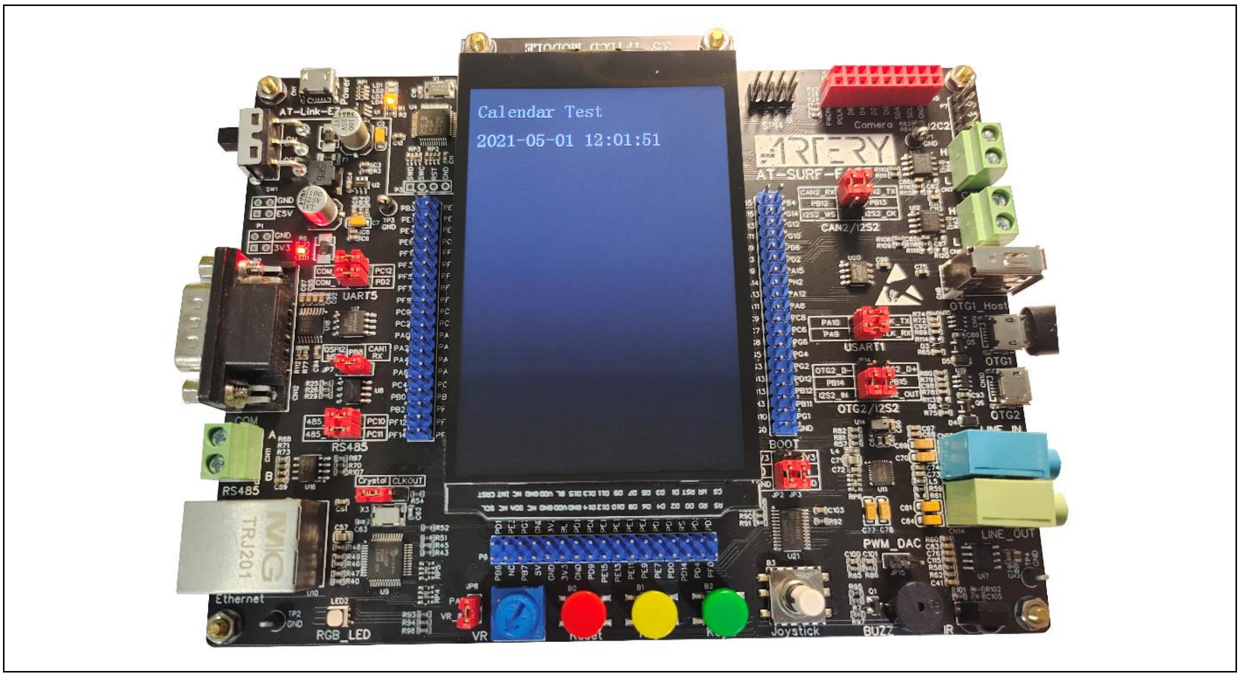
- void calendar_init(void)

```
/**
  * @brief   calendar init.
  * @param   none.
  * @retval none.
  */
void calendar_init(void)
```

4.5.5 Download and verify

- Show calendar information on LCD.

Figure 13. Test result



4.6 Example 6: Button application

4.6.1 Introduction

AT32-SUFR has two buttons that are connected to the IO ports of MCU. Pressing the buttons, the IO level becomes high, otherwise, the IO low. By checking IO level, the MCU can know whether the buttons are pressed or not.

4.6.2 Resource requirements

- Hardware resources

  AT-SURF-F437 Board

- Software resources

  AT32F435_437_Firmware_Library_V2.x.x\project\at_sufr_f437\examples\key

4.6.3 Hardware design

Hardware resources in the example are two separate buttons.

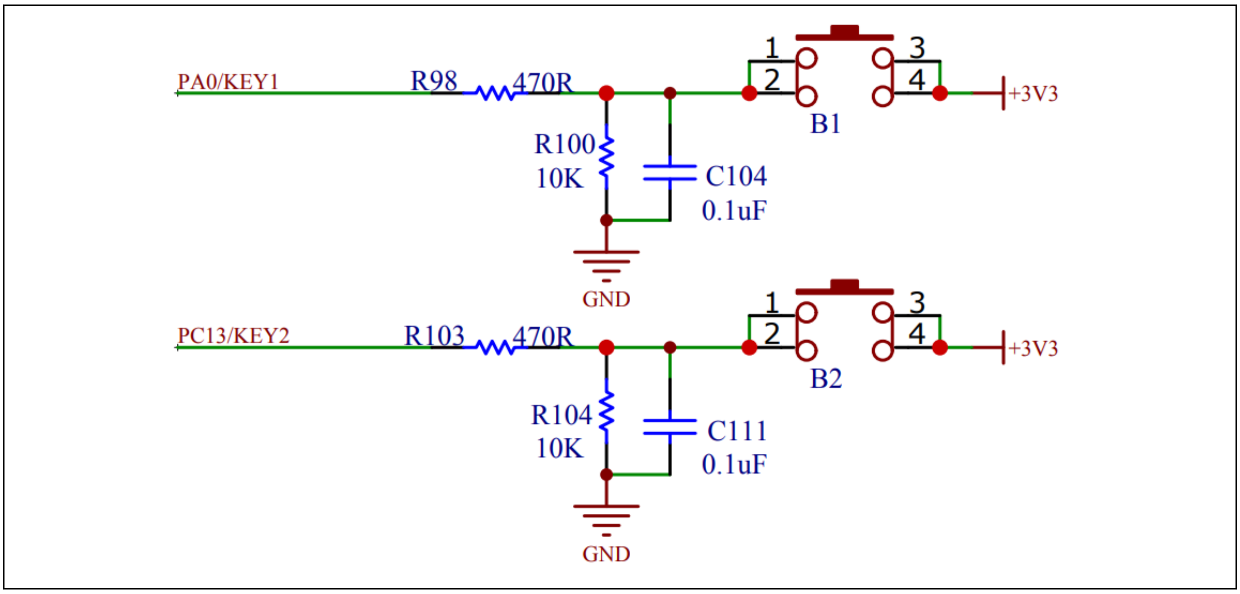
Table 8 gives the corresponding pins.

Table 8. Hardware resources

No.	PIN Name	Peripheral function	Description
1	PA0	GPIO	Button 1
2	PC13	GPIO	Button 2

Figure 14 shows the schematic diagram of button circuit.

Figure 14. Schematic diagram of button circuit



4.6.4 Software design

1) Button test
   - Set IO ports connected to both buttons as input mode
   - Read IO status to check whether the buttons are pressed or not.
2) Code
   - main function

```
int main(void)
{
  /* Initialize system clock */
  system_clock_config();

  /* Initialize interrupt priority group */
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

  /* Initialize delay function */
  delay_init();

  /* Initialize LCD */
  lcd_init(LCD_DISPLAY_VERTICAL);

  /* Initialize buttons */
  key_init();

  /* Show information */
  lcd_string_show(10, 20, 200, 24, 24, (uint8_t *)"Key Test");
  lcd_string_show(10, 60, 280, 24, 24, (uint8_t *)"Press any key to begin");

  while(1)
  {
    /* Check button status */
    key_value = key_press();

    switch(key_value)
    {
      /* Button 1 is pressed */
      case KEY_1:
        lcd_string_show(10, 100, 310, 24, 24, (uint8_t *)"key value: key 1");
        break;

      /* Button 2 is pressed */
      case KEY_2:
        lcd_string_show(10, 100, 310, 24, 24, (uint8_t *)" key value: key 2");
        break;

      default:
```

```
            break;
        }
    }
}
```

■ void key_init(void)

```
/**
  * @brief key init.
  * @param none.
  * @retval none.
  */
void key_init(void)
```

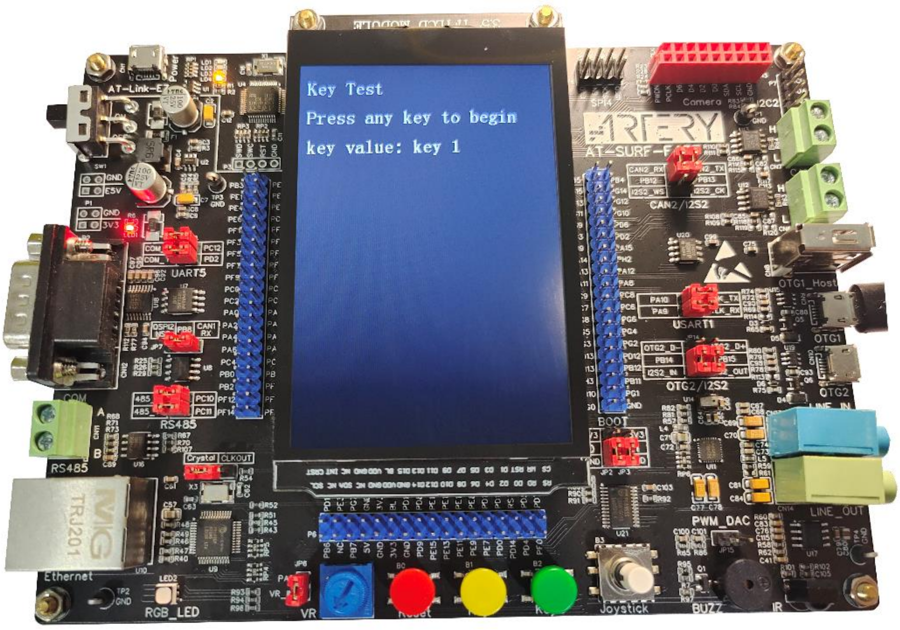
■ key_type at32_key_press(void)

```
/**
  * @brief  returns which key have press down
  * @param  none
  * @retval the key have press down
  */
key_type at32_key_press(void)
```

4.6.5 Download and verify

■ If there is any of two buttons is being pressed, this status information will be displayed on the LCD screen.

Figure 15. Test result



4.7 Example 7: ADC sample application

4.7.1 Introduction

The ADC is a peripheral that converts an analog input signal into a 12-bit/10-bit/8-bit/6-bit digital signal. The AT32F437 has up to 19 channels, with up to 5.33MSPS of sampling rate.
For AT32F437, its ADC main features:

- 12-bit, 10-bit, 8-bit or 6-bit resolution
- ADC conversion time is 0.1875 µs at 80MHz (in 12-bit resolution)
- ADC conversion time is 0.1125 µs at 80MHz (in 16-bit resolution)
- Channel-wise programmable sampling time
- Conversion sequence management mechanism supports various conversion modes
- Multiple data alignment mode
- DMA transfer support

AT32-SUFR embeds a variable resistor that is connected to the ADC. Thus it is possible for users to get the current voltage of the variable resistor through ADC sampling.

4.7.2 Resource requirements

- Hardware resources
  AT-SURF-F437 Board
- Software resources
  AT32F435_437_Firmware_Library_V2.x.x\project\at_sufr_f437\examples\variable_resistor

4.7.3 Hardware design

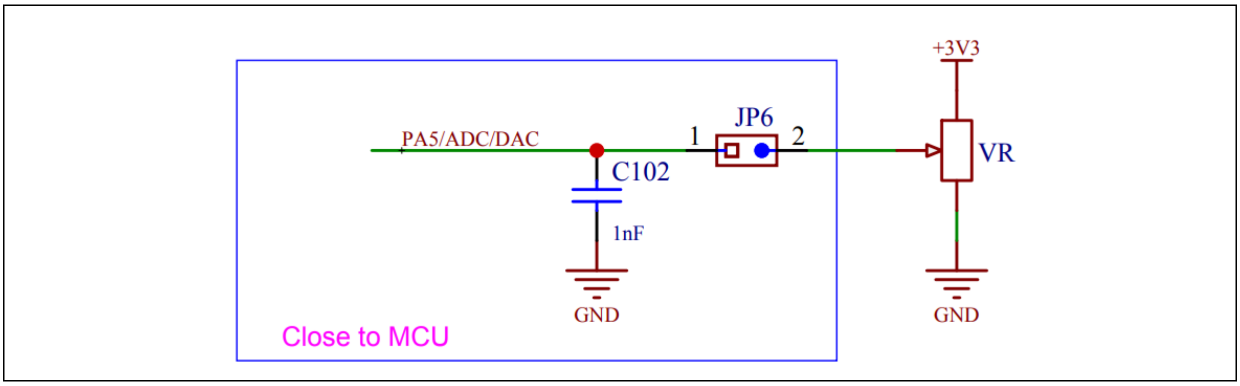
This application example use a variable resistor as hardware resource.
Table 9 gives the corresponding pins.

Table 9. Hardware resource

No.	PIN Name	Peripheral function	Description
1	PA5	ADC_CHANNEL_5	ADC1 channel 5

Figure 16 shows the schematic diagram of the variable resistor.

Figure 16. Schematic diagram of adjustable resistor



4.7.4 Software design

1) ADC test

- Initialize timers for triggering ADC conversion at a frequency of 10 Hz
- Initialize DMA for ADC data transfer
- Initialize ADC
- ADC-captured voltage values are shown on LCD

2) Code

- main function

```
int main(void)
{
  /* Initialize system clock */
  system_clock_config();

  /* Initialize interrupt priority group */
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

  /* Initialize delay function */
  delay_init();

  /* Initialize LCD */
  lcd_init(LCD_DISPLAY_VERTICAL);

  /* Initialize variable resistor */
  variable_resistor_init();

  /* Show information */
  lcd_string_show(10, 20, 310, 24, 24, (uint8_t *)"Variable Resistor Test");

  while(1)
  {
    /* ADC conversion overflow */
    if(adc_overflow_flag != 0)
    {
      /* Show error information */
      lcd_string_show(10, 120, 310, 24, 24, (uint8_t *)"Error occur: ");

       /* Show error times */
      lcd_num_show(114, 60, 310, 24, 24, adc_overflow_flag, 3);
    }

    /* ADC conversion complete */
    else if(dma_trans_complete_flag == 1)
    {
      /* Clear transfer complete flag */
```

```
        dma_trans_complete_flag = 0;

        /* Calculate current voltage */
        voltage = 3.3 * adc_convert_value / 4095;

        /* Show voltage */
        lcd_string_show(10, 60, 310, 24, 24, (uint8_t *)"Voltage:");

        /* Show voltage value */
        lcd_float_num_show(114, 60, 310, 24, 24, voltage, 3);
      }
    }
}
```

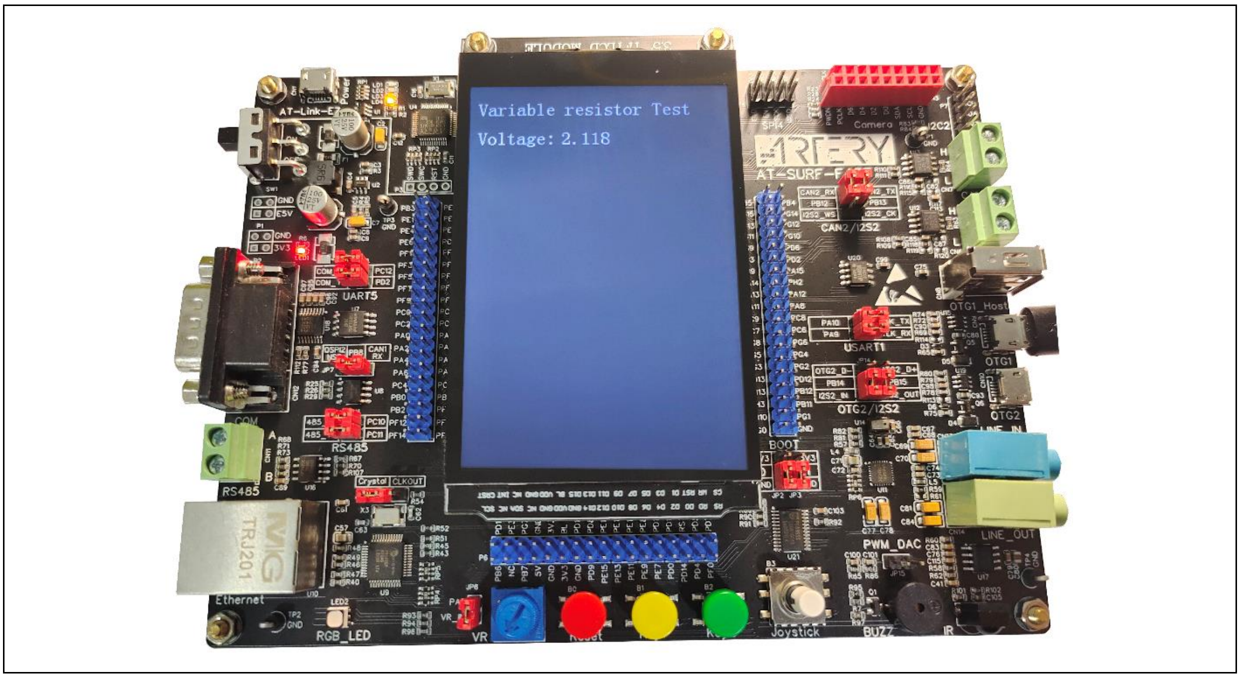
- void variable_resistor_init(void)

```
/**
  * @brief   variable_resistor init.
  * @param   none.
  * @retval none.
  */
void variable_resistor_init(void)
```

4.7.5 Download and verify

- Modify the variable resistor and view the voltage value on LCD display.

Figure 17. Test result



4.8 Example 8: DAC output

4.8.1 Introduction

The DAC module is a digital-to-analog converter. The AT32F437 has two interdependent DACs so that conversions can be done independently or simultaneously. The DAC can be configured in 8- or12-bit mode, generating an analog output between 0 and reference voltage. The input reference voltage VREF+ is available for better conversion accuracy.

AT32F437 DAC has the main features below:

— 8-bit or 12-bit mode
— Left or right alignment for a single or dual DAC
— Reference voltage VREF+
— DMA support
— Noise-wave/triangular-wave generation
— Independent conversion for dual DAC or a single DAC (DAC1 or DAC2)
— DMA capability for DAC1/DAC
— Software-triggered or external-triggered conversion

As the DAC of the AT32-SUFR board is connected to a variable resistor, it is necessary to disconnect the JP6 jumper.

4.8.2 Resource requirements

■ Hardware resources

AT-SURF-F437 Board

■ Software resources

AT32F435_437_Firmware_Library_V2.x.x\project\at_sufr_f437\examples\dac

4.8.3 Hardware design

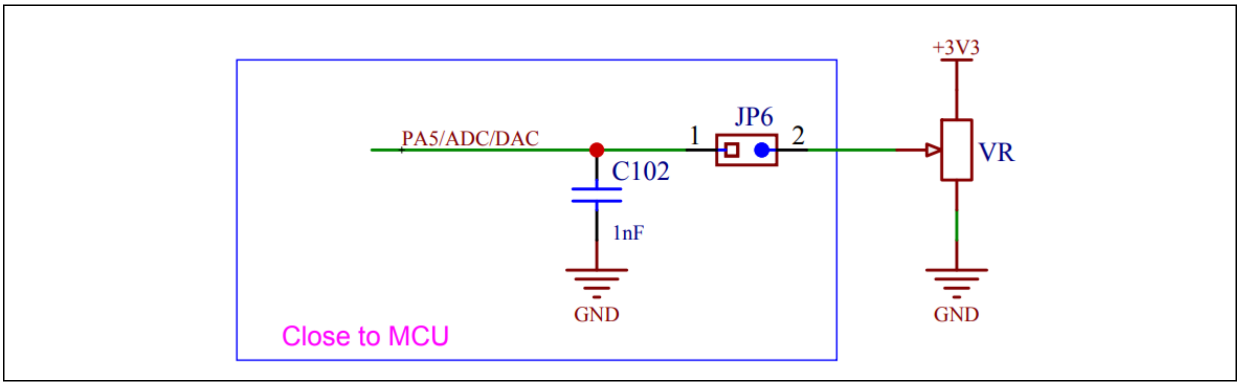
Hardware resources in the application example are TFT LCD display and DAC.
Table 10 gives the corresponding pins.

Table 10. Hardware resources

No.	PIN Name	Peripheral function	Description
1	PA5	DAC2	-

Figure 18 shows the schematic diagram of DAC.

Figure 18. Schematic diagram of DAC circuit



4.8.4 Software design

1) DAC test
   - Initialize DAC
   - Increase 0.1V per 300ms, and show the output voltage on LCD display
2) Code
   - main function

```
int main(void)
{
   uint16_t voltage = 0;

   /* Initialize system clock */
   system_clock_config();

   /* Initialize interrupt priority group */
   nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

   /* Initialize delay function */
   delay_init();

   /* Initialize LCD */
   lcd_init(LCD_DISPLAY_VERTICAL);

   /* Initialize DAC */
   dac_init();

   /* Show information */
   lcd_string_show(10, 20, 200, 24, 24, (uint8_t *)"DAC Test");

   while(1)
   {
      /* Add 0.1V for each output*/
      voltage += 100;

      if(voltage > 3300)
      {
         voltage = 0;
      }

      /* Show title */
      lcd_string_show(10, 60, 310, 24, 24, (uint8_t *)"Output Voltage:");

      /* Show output voltage */
      lcd_float_num_show(200, 60, 310, 24, 24, voltage / 1000.0, 1);

      /* DAC output setting */
```

```
        dac_output_voltage_set(voltage);

        delay_ms(300);
    }
}
```

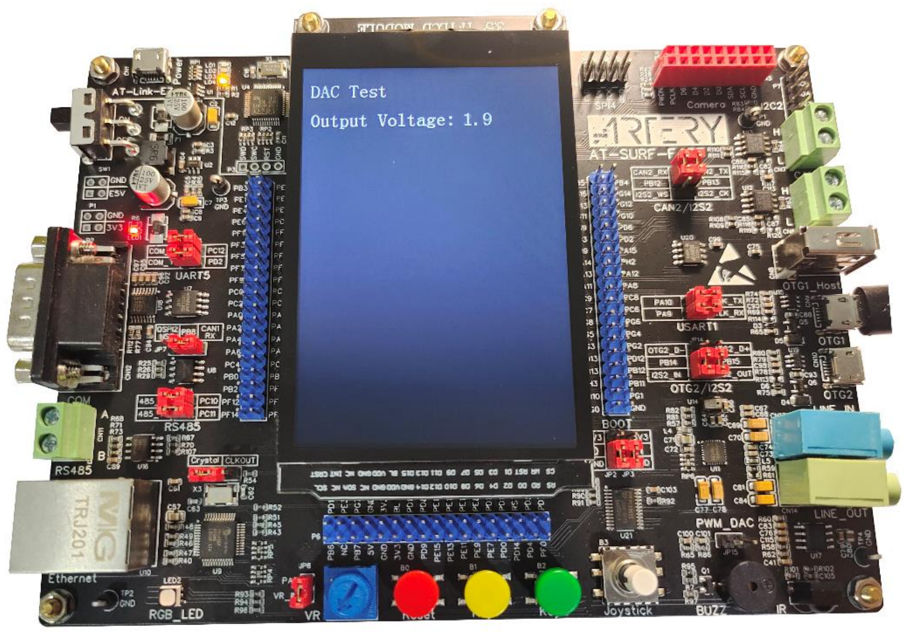
- void dac_init(void)

```
/**
    * @brief   dac init.
    * @param    none.
    * @retval none.
    */
void dac_init(void)
```

4.8.5 Download and verify

- Increase 0.1V output per 300ms, and show output voltage on LCD display.
- Measure the voltage on PA5 via multimeter, and you can see that it matches the output voltage on LCD display.

Figure 19. Test result



4.9 Example 9: PWM DAC output

4.9.1 Introduction

PWM DAC refers to the implementation of DAC feature through PWM. The PWM signal is a digital signal with a fixed frequency pulse width change. Generally speaking, the PWM signal voltage can be considered as an analog signal, after going through a filter, which can act as a DAC with low accuracy. This is particularly useful for the devices without ADC peripherals as it lowers development costs greatly by saving one DAC chip.

4.9.2 Resource requirements

- Hardware resources

  AT-SURF-F437 Board

- Hardware resources

  AT32F435_437_Firmware_Library_V2.x.x\project\at_sufr_f437\examples\pwm_dac

4.9.3 Hardware design

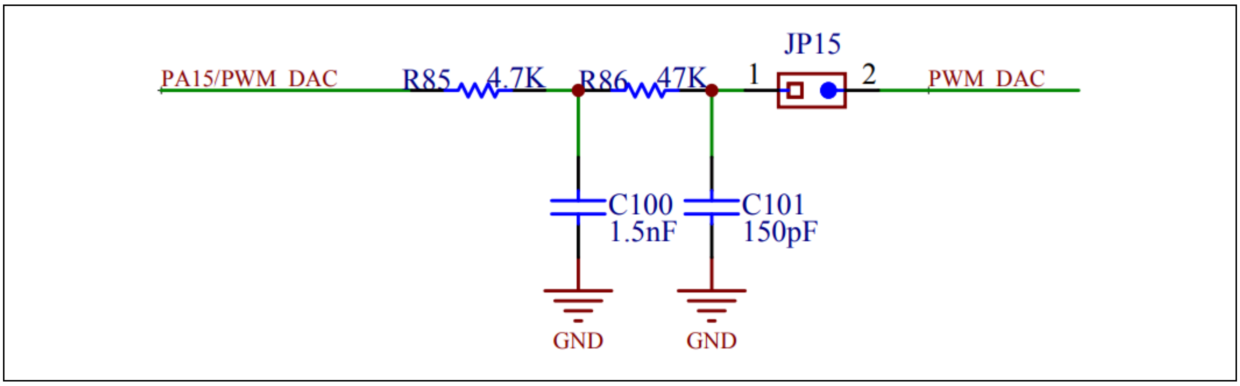
Hardware resources in the application example are TFT LCD display and TMR.
Table 11 gives the corresponding pins:

Table 11. Hardware resources

No.	PIN Name	Peripheral function	Description
1	PC7	TMR8_CH2	PWM output

Figure 20 shows the schematic diagram of PWM DAC.

Figure 20. PWM DAC schematic diagram



4.9.4 Software design

1) PWM DAC test
   - Initialize PWM output of TMR
   - Increase 0.1V output per 300ms, and show the output voltage on LCD display
2) Code
   - main function

```
int main(void)
{
   uint16_t voltage = 0;

   /* Initialize system clock */
   system_clock_config();

   /* Initialize interrupt priority group */
   nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

   /* Initialize delay function */
   delay_init();

   /* Initialize LCD */
   lcd_init(LCD_DISPLAY_VERTICAL);

   /* Initialize PWM DAC */
   pwm_dac_init();

   /* Show information */
   lcd_string_show(10, 20, 200, 24, 24, (uint8_t *)"PWM DAC Test");

   while(1)
   {
      /* Add 0.1V for each output */
      voltage += 100;

      if(voltage > 3300)
      {
         voltage = 0;
      }

      /* Show title */
      lcd_string_show(10, 60, 310, 24, 24, (uint8_t *)"Output Voltage:");

      /* Show output voltage */
      lcd_float_num_show(200, 60, 310, 24, 24, voltage / 1000.0, 1);

      /* PWM DAC output setting */
```

```
        pwm_dac_output_voltage_set(voltage);

        delay_ms(300);
    }
}
```

- void pwm_dac_init(void)

```
/**
  * @brief   pwm dac init.
  * @param   none.
  * @retval none.
  */
void pwm_dac_init(void)
```

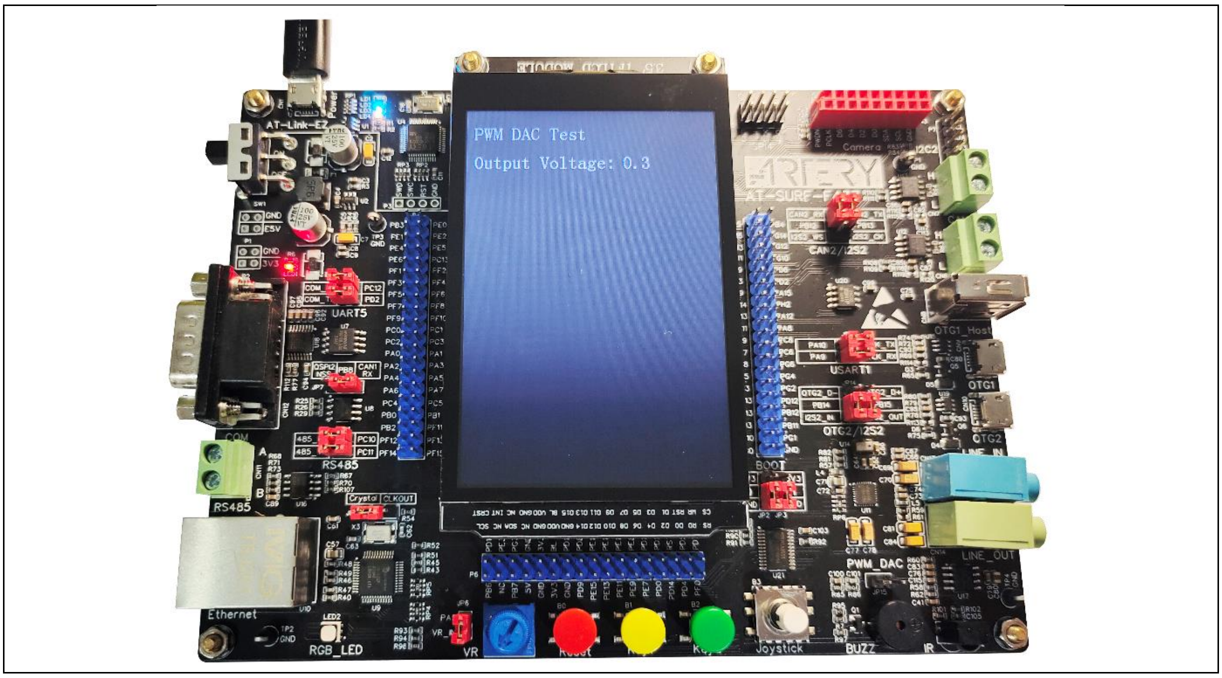
- void pwm_dac_output_voltage_set(uint16_t voltage)

```
/**
  * @brief   pwm dac output voltage set.
  * @param   voltage: output voltage
  *              the range is 0~3300 representing 0~3.300V.
  * @retval none.
  */
void pwm_dac_output_voltage_set(uint16_t voltage)
```

4.9.5 Download and verify

- Increase 0.1V output per 300ms, and show the output voltage on LCD display.
- Measure the JP15 voltage via a multimeter, and we can see that the reading matches the output voltage on LCD.

Figure 21. Test result



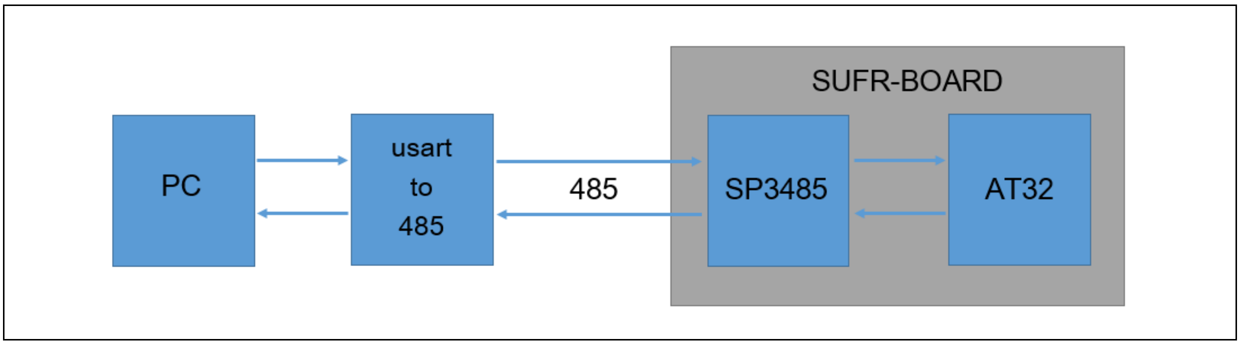
4.10 Example 10: RS485 communications

4.10.1 Introduction

The RS485 is usually based on two-wire bus. It operates in half-duplex mode and is widely used in industrial control applications. The RS485 communication method is able to achieve a maximum of 1200 meters of transmission distance, theoretically. Its bus can be connected to multiple devices. For users, it is only necessary to connect A and B ports of a device to A and B lines on the bus. The RS485 signals are differential and thus enjoy stronger anti-interference. In terms of bus communication, master/slave communication mode is often used, that is, a master is connected to several slaves.

AT32 SUFR board embeds a 485 chip which is connected to AT32 MCU via a serial interface. In data transmission mode, the data are first sent to 485 chip by AT32 MCU via a serial interface, and then converted into differentiated signals via 485 chip before being transmitted to the bus. In data reception mode, the differentiated signals on the bus are sent to the serial interface via the 485 chip for AT32 MCU to read.

Figure 22. Test process



4.10.2 Resource requirements

- Hardware resources

  AT-SURF-F437 Board

- Software resources

  AT32F435_437_Firmware_Library_V2.x.x\project\at_sufr_f437\examples\rs485

4.10.3 Hardware design

The application example uses 485 chip as hardware resource.

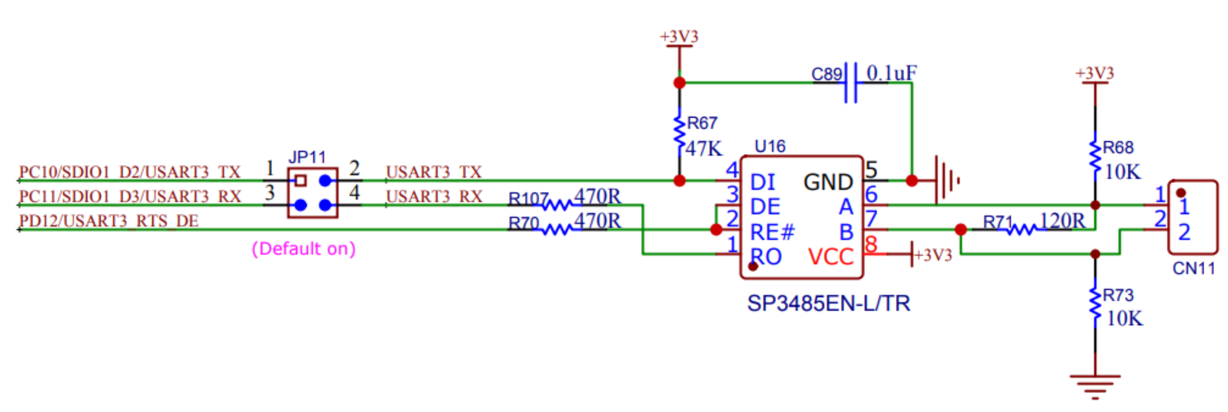
Table 12 gives the corresponding pins:

Table 12. Hardware resources

No.	PIN Name	Peripheral function	Description
1	PC10	USART3_TX	Serial interface transmit pin
2	PC11	USART3_RX	Serial interface receive pin
3	PD12	DE	485 chip TX/RX mode switch

Figure 23 shows the schematic diagram of 485 chip.

Figure 23. Schematic diagram of 485 chip



4.10.4 Software design

1) RS485 test

- Initialize RS485
- Enter reception mode and wait to receive data
- PC side sends data to SUFR board
- After the data are received by SUFR board, they are shown on LCD display
- Send a second data to the bus and enter reception mode, repeat above steps

2) Code

- main function

```
int main(void)
{
    /* Initialize system clock */
    system_clock_config();

    /* Initialize interrupt priority group */
    nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

    /* Initialize delay function */
    delay_init();

    /* Initialize LCD */
    lcd_init(LCD_DISPLAY_VERTICAL);

    /* Initialize RS485 */
    rs485_init();

    /* Show information */
    lcd_string_show(10, 20, 200, 24, 24, (uint8_t *)"RS485 Test");

    lcd_string_show(10, 60, 200, 24, 24, (uint8_t *)"RX data:");
```

```
  while(1)
  {
    /* Receive data */
    if(rs485_rx_number)
    {
      /* Clear up the last byte for presentation purposes */
      rx_buf[rs485_rx_number] = 0;

      /* Copy dat to rx_buf */
      rs485_data_receive(rx_buf, rs485_rx_number);

      /* Clear display area */
      lcd_fill(10, 100, 310, 124, WHITE);

      /* Show received data */
      lcd_string_show(10, 100, 200, 24, 24, rx_buf);

      /* Send data */
      rs485_data_send(tx_buf, 17);
    }
  }
}
```

■ void rs485_init(void)

```
/**
  * @brief  initializes rs485.
  * @param  none
  * @retval none
  */
void rs485_init(void)
```

■ void rs485_data_send(uint8_t* pdata, uint16_t num)

```
/**
  * @brief  send data.
  * @param  pdata: data buffer.
  * @param  num: data size.
  * @retval none.
  */
void rs485_data_send(uint8_t* pdata, uint16_t num)
```

■ void rs485_data_receive(uint8_t* pdata, uint16_t num)

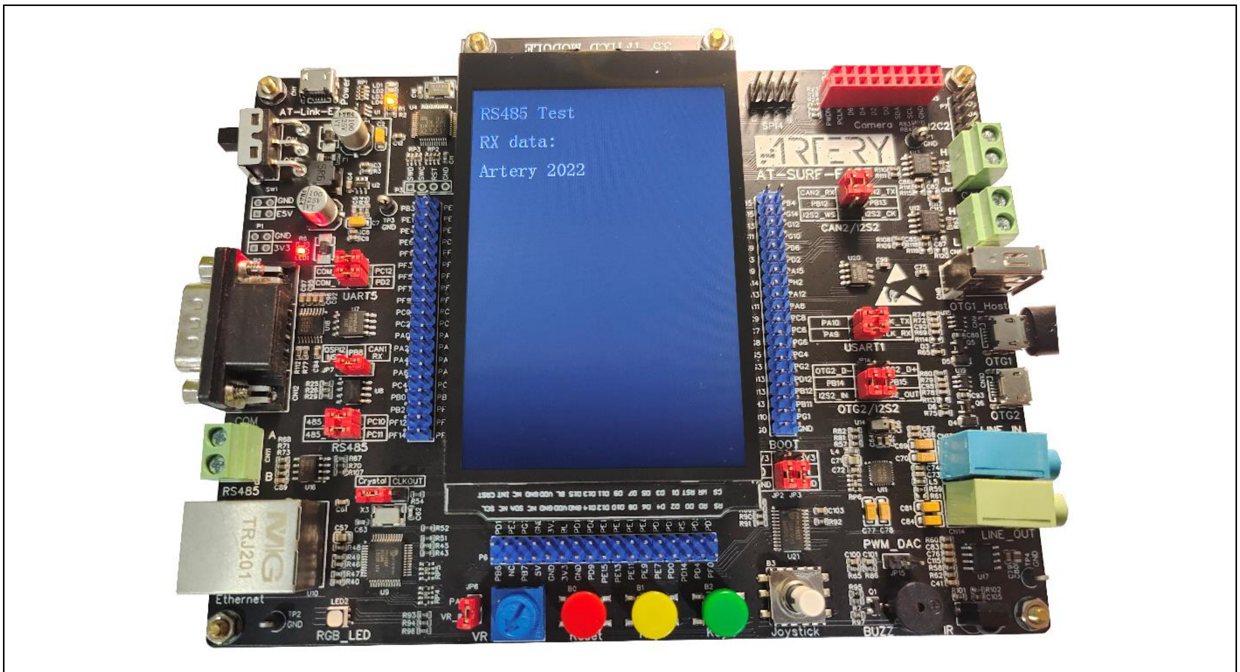
```
/**
  * @brief  receive data.
  * @param  pdata: data buffer.
  * @param  num: data size.
  * @retval none.
  */
void rs485_data_receive(uint8_t* pdata, uint16_t num)
```

4.10.5 Download and verify

- PC sends “Artery 2022” to SUFR board via a serial port assistant.
- The received data is shown on LCD display.
- SUFR board sends another data “AT32-SUFR-BOARD” to PC.

Figure 24. PC side serial port assistant

Figure 25. SUFR side



4.11 Example 11: CAN communication

4.11.1 Introduction

CAN is the acronym of the Controller Area Network. It is firstly developed by BOSCH Company in Germany and later recognized as an international standard. As one of the extensively-used buses across the world, CAN features high performance and strong reliability and has been widely applied in various fields covering industrial automation, shipping, automotive and industrial equipment.

AT32F437 CAN has the following main features:

- Supports CAN protocol version 2.0A and 2.0B
- Baud rates up to 1M bit/s for software-triggered or external-triggered conversions
- Time triggered communication option
- Interrupt generation and maskable
- Configurable auto retransmission mode

Transmission

- Three transmit mailboxes
- Configurable transmit priority
- Time stamp on transmission

Reception

- Two receive FIFOs with three stages
- 28 filter banks
- Identifier list mode
- Identifier mask mode
- FIFO overrun management
- Time triggered communication mode
- 16-bit timer
- Time stamp on transmission

The AT32 SUFR board embeds two CANs, SN65HVD230DR (part number). Attention should be paid to the settings of Jumper caps when testing. The H and L signal lines of one CAN must be connected to those of another CAN, respectively.

4.11.2 Resource requirements

- Hardware resources

  AT-SURF-F437 Board

- Software resources

  AT32F435_437_Firmware_Library_V2.x.x\project\at_sufr_f437\examples\can

4.11.3 Hardware design

Hardware resources used in the application example are TFT LCD, two CANs (part number: SN65HVD230DR).

Table 13 lists the corresponding pins:

Table 13. Hardware resources

No.	PIN Name	Peripheral function	Description
1	PB8	CAN1 RX	CAN1 transmit
2	PB9	CAN1 TX	CAN1 receive
2	PB12	CAN2 RX	CAN2 receive
4	PB13	CAN2 TX	CAN2 transmit

Figure 26 shows the schematic diagram of CAN1.

Figure 26. Schematic diagram of CAN1

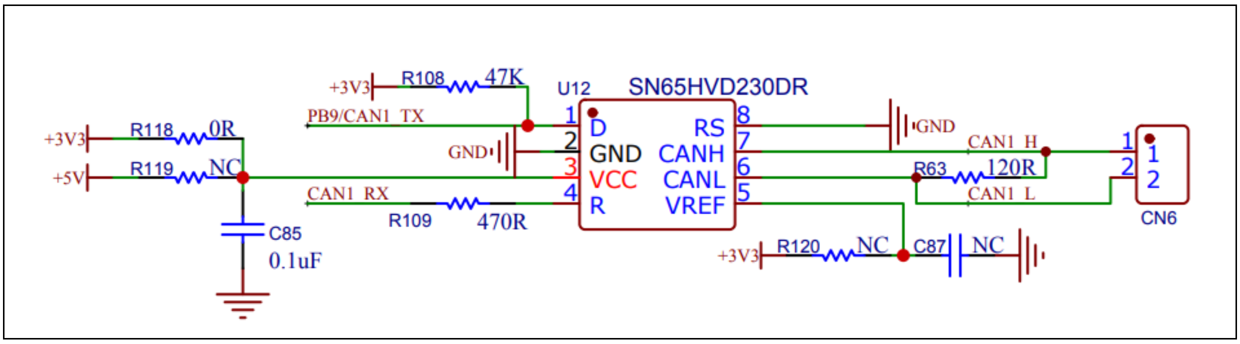


Figure 27. Schematic diagram of CAN2

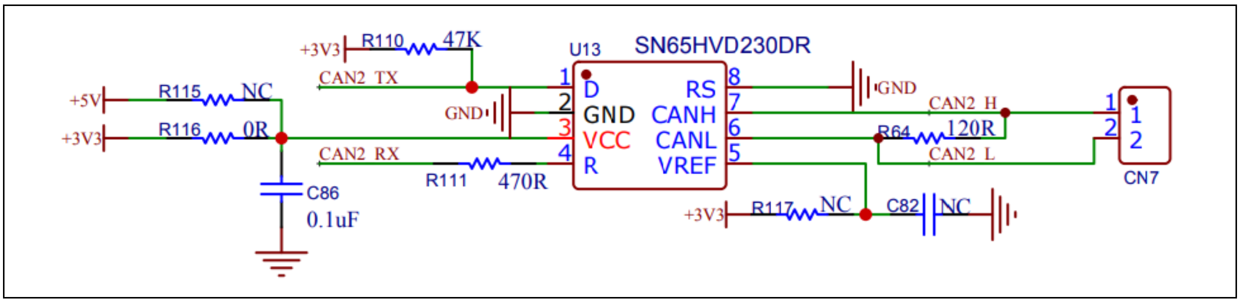
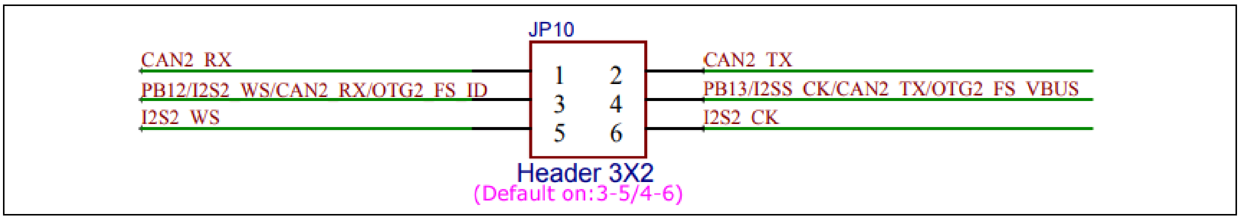


Figure 28. Schematic diagram of Jumper



4.11.4 Software design

1) CAN communication test
   - Initialize TFT LCD
   - Initialize CAN1 and CAN2
   - CAN1 sends data to CAN2, while CAN2 sends data to CAN1
2) Code
   - main function

```
int main(void)
{
    uint32_t cnt = 0;

    /* Initialize system clock */
    system_clock_config();

    /* Initialize interrupt priority group */
    nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

    /* Initialize delay function */
    delay_init();

    /* Initialize LCD */
    lcd_init(LCD_DISPLAY_VERTICAL);

    /* Initialize can 1 */
    can_init(CAN1);

    /* Initialize can 2 */
    can_init(CAN2);

    /* Display information*/
    lcd_string_show(10, 20, 200, 24, 24, (uint8_t *)"CAN Test");

    while(1)
    {
        /* can 1 data initialization */
        can_data_init(CAN1);

        /* Transmit data through can 1 */
        can_transmit_data(CAN1, can1_tx_message);

        /* can 2 data initialization */
        can_data_init(CAN2);

        /* Transmit data through can 2 */
```

```
can_transmit_data(CAN2, can2_tx_message);

/* can 1 receives data */
if(can1_rx_flag != 0)
{
  /* Clear can 1 received data flag */
  can1_rx_flag = 0;

  /* Show title */
  lcd_string_show(10, 60, 200, 24, 24, (uint8_t *)"can 1 received");

  /* Show filter index */
  lcd_string_show(10, 100, 200, 24, 24, (uint8_t *)"filter_index:");
  lcd_num_show(170, 100, 200, 24, 24, can1_rx_message.filter_index, 1);

  if (can1_rx_message.id_type == CAN_ID_EXTENDED )
  {
    /* Show extended id */
    lcd_string_show(10, 130, 200, 24, 24, (uint8_t *)"extended_id:");
    lcd_num_show(170, 130, 200, 24, 24, can1_rx_message.extended_id, 1);
  }
  else
  {
    /* Show standard id */
    lcd_string_show(10, 130, 200, 24, 24, (uint8_t *)"standard_id:");
    lcd_num_show(170, 130, 200, 24, 24, can1_rx_message.standard_id, 1);
  }

  if (can1_rx_message.frame_type == CAN_TFT_REMOTE )
  {
    /* No data */
    lcd_string_show(10, 160, 200, 24, 24, (uint8_t *)"remote frame: no data");
  }
  else
  {
     /* Data compare */
    if(buffer_compare(can2_tx_message.data, can1_rx_message.data, 8) == 0)
    {
      /* Data reception success */
      lcd_string_show(10, 160, 310, 24, 24, (uint8_t *)"can 1 Data reception success");
    }
    else
    {
      /* Data reception error */
      lcd_string_show(10, 160, 310, 24, 24, (uint8_t *)"can 1 Data reception error ");
    }
```

```
    }
  }

  /* can 2 receives data */
  if(can2_rx_flag != 0)
  {
    /* Clear can 2 received data flag */
    can2_rx_flag = 0;

    /* Show title */
    lcd_string_show(10, 220, 200, 24, 24, (uint8_t *)"can 2 received");

    /* Show filter index */
    lcd_string_show(10, 260, 200, 24, 24, (uint8_t *)"filter_index:");
    lcd_num_show(170, 260, 200, 24, 24, can2_rx_message.filter_index, 1);

    if (can2_rx_message.id_type == CAN_ID_EXTENDED )
    {
      /* Show extended id */
      lcd_string_show(10, 290, 200, 24, 24, (uint8_t *)"extended_id:");
      lcd_num_show(170, 290, 200, 24, 24, can2_rx_message.extended_id, 1);
    }
    else
    {
      /* Show standard id */
      lcd_string_show(10, 290, 200, 24, 24, (uint8_t *)"standard_id:");
      lcd_num_show(170, 290, 200, 24, 24, can2_rx_message.standard_id, 1);
    }

    if (can2_rx_message.frame_type == CAN_TFT_REMOTE )
    {
      /* No data */
      lcd_string_show(10, 320, 200, 24, 24, (uint8_t *)"remote frame: no data");
    }
    else
    {
       /* Data compare */
      if(buffer_compare(can1_tx_message.data, can2_rx_message.data, 8) == 0)
      {
        /* Data reception success */
        lcd_string_show(10, 320, 310, 24, 24, (uint8_t *)"can 2 Data reception success");
      }
      else
      {
        /* Data reception error */
        lcd_string_show(10, 320, 310, 24, 24, (uint8_t *)"can 2 Data reception error ");
```

```
            }
        }
    }

    cnt++;

    /* Show transfer times */
    lcd_string_show(10, 380, 200, 24, 24, (uint8_t *)"transfer number:");
    lcd_num_show(206, 380, 250, 24, 24, cnt, 1);

    delay_ms(1000);
  }
}
```

- void can_init(can_type *can_x)

```
/**
  * @brief   initialize can.
  * @param   can_x: select the can peripheral.
  *          this parameter can be one of the following values:
  *          CAN1, CAN2.
  * @retval none
  */
void can_init(can_type *can_x)
```

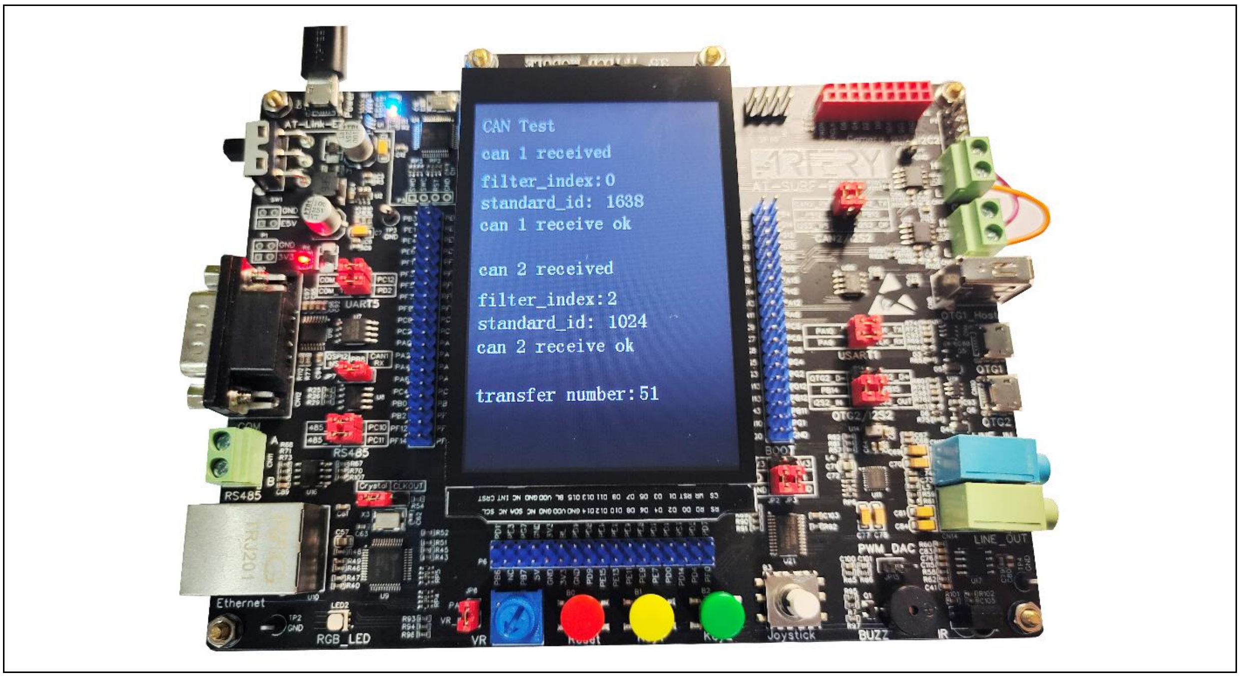
- void can_transmit_data(can_type *can_x, can_tx_message_type tx_message_struct)

```
/**
  * @brief   can transmit data
  * @param   can_x: select the can peripheral.
  *          this parameter can be one of the following values:
  *          CAN1, CAN2.
  *          tx_message_struct: transmitmessage
  * @retval none
  */
void can_transmit_data(can_type *can_x, can_tx_message_type tx_message_struct)
```

4.11.5 Download and verify

- CAN1 sends data to CAN2, and CAN2 sends data to CAN1.
- Compare the data received by CAN1 and CAN2 to verify if both are correct.
- LCD screen shows communication information of CAN1 and CAN2.

Figure 29. Test result

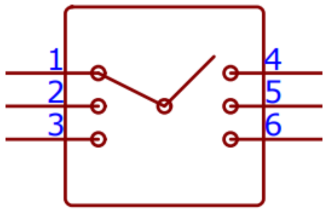


4.12 Example 12: Gaming pad application

4.12.1 Introduction

AT32-SUFR features a gaming pad (five-direction joystick with a selection button). The principle of the five-direction joystick is shown in Figure 30 below. There are a total of five communication paths, indicating that the buttons can output five values.
This section describes how to read and print the key values of the gaming pad.

Figure 30. Five-direction joystick principle



The gaming pad on the SUFR board is not connected to AT32 MCU IO, but to an extended IO connector PCA9555. The PCA9555 is linked to AT32 MCU through I^2C bus. When the IO level change on PCA9555 is detected, a falling edge is generated on the INT pin. After the falling edge is detected, the AT32 MCU is able to get the current IO status by reading the IO status register of PCA9555 via I^2C bus.

4.12.2 Resource requirements

- Hardware resources

  AT-SURF-F437 Board

- Software resources

  AT32F435_437_Firmware_Library_V2.x.x\project\at_sufr_f437\examples\joystick

4.12.3 Hardware design

Hardware resources in the application example are a gaming pad and PCA9555.
Table 14 to 15 list the corresponding pins:

Table 14. Hardware resources

No.	PIN Name	Peripheral function	Description
1	PH2	I2C2 SCL	PCA9555 SCL
2	PH3	I2C2 SDA	PCA9555 SDA
3	PG3	GPIO	PCA9555 INT

Table 15. PCA9555

No.	PIN Name	Peripheral function	Description
1	IO1_0	Up	Five-direction joystick key
2	IO1_1	Right	Five-direction joystick key
3	IO1_2	Down	Five-direction joystick key
4	IO1_3	Center	Five-direction joystick key
5	IO1_4	Left	Five-direction joystick key

Figure 31 shows the schematic diagram of PCA9555 connector.

Figure 31. Schematic diagram of PCA9555 connector

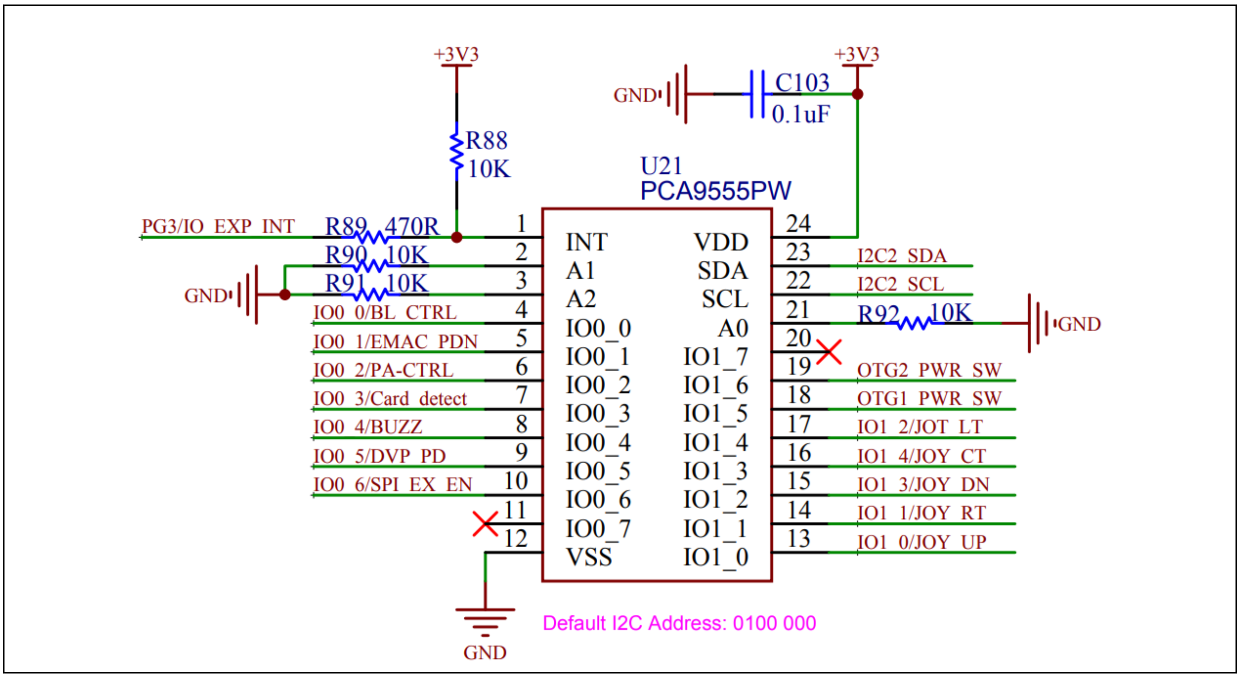
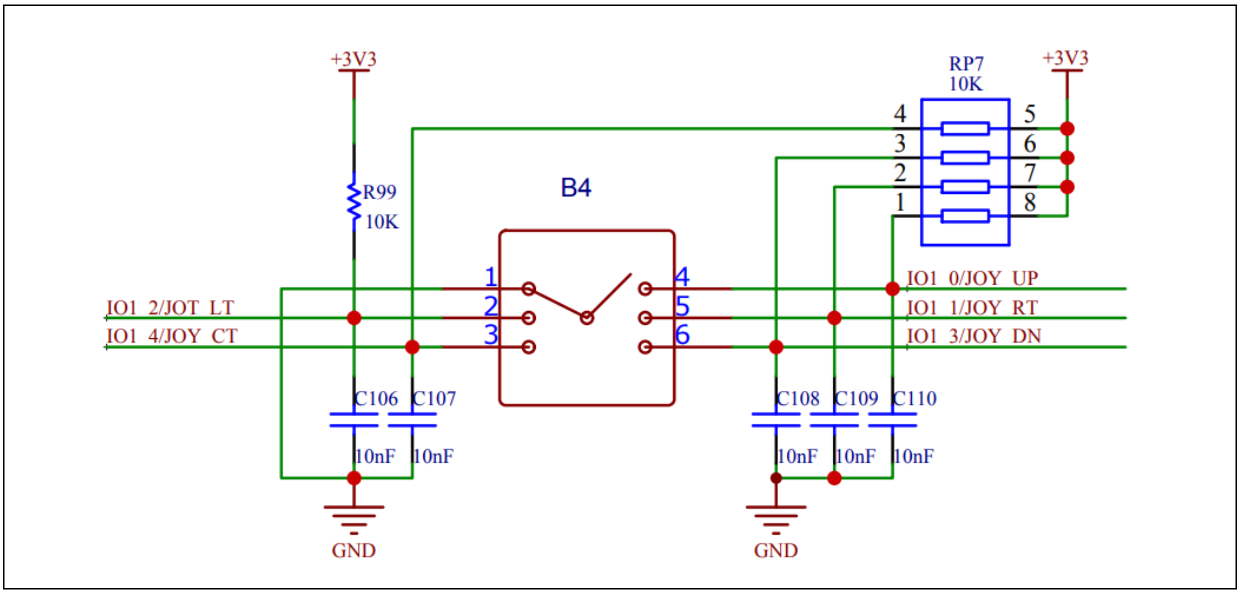


Figure 32. Schematic diagram of gaming pad



4.12.4 Software design

1) Gaming pad test
   - Initialize I^2C interface
   - Set the PCA9555 IO as input mode, which is connected to gaming pad keys
   - Read the status of PCA9555 IO to confirm where the keys are pressed or not.
2) Code
   - main function

```
int main(void)
{
    uint16_t x = 0, i;

    joy_type key;

    /* Initialize system clock */
    system_clock_config();

    /* Initialize interrupt priority group */
    nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

    /* Initialize delay function */
    delay_init();

    /* Initialize LCD */
    lcd_init(LCD_DISPLAY_VERTICAL);

    /* Initialize five-direction joystick keys */
    joystick_init();

    lcd_clear(BLACK);

    /* Display information */
    lcd_string_show(80, 20, 200, 24, 24, (uint8_t *)"Joystick Test");

    lcd_string_show(50, 400, 200, 24, 24, (uint8_t *)"key value:");

    while(1)
    {
        /* Read PCA9555 IO status */
        pca9555_io_scan();

        /* Get key value */
        key = joystick_press();

        switch(key)
```

```
{
  case JOY_LEFT:
    lcd_string_show(200, 400, 100, 24, 24, (uint8_t *)"left ");
    pitch = 10;
    roll= 0;
    yaw= 0;
    break;
  case JOY_RIGHT:
    lcd_string_show(200, 400, 100, 24, 24, (uint8_t *)"right");
    pitch = -10;
    roll= 0;
    yaw= 0;
    break;
  case JOY_UP:
    lcd_string_show(200, 400, 100, 24, 24, (uint8_t *)"up    ");
    pitch = 0;
    roll= -10;
    yaw=0;
    break;
  case JOY_DOWN:
    lcd_string_show(200, 400, 100, 24, 24, (uint8_t *)"down ");
    pitch = 0;
    roll= 10;
    yaw=0;
    break;
  case JOY_ENTER:
    lcd_string_show(200, 400, 100, 24, 24, (uint8_t *)"enter");
    pitch = 0;
    roll= 0;
    yaw = 10;
    break;
  case JOY_NONE:
    break;
  default:
    break;
}

lcd_fill(80,80,230,220,BLACK);

for(i=0; i<8; i++)
rotate(cube[i], pitch/360, roll/360, yaw/360);
for(i=0; i<28; i+=2)
{
  lcd_draw_line(160+cube[lineid[i]-1][0], 150+cube[lineid[i]-1][1], 160+cube[lineid[i+1]-1][0], 150+cube[lineid[i+1]-1][1], WHITE);
}
```

```
        delay_ms(10);
    }
}
```

■ void joystick_init(void)

```
/**
  * @brief  joystick_init.
  * @param  none.
  * @retval none.
  */
void joystick_init(void)
```

■ void dac_init(void)

```
/**
  * @brief  dac init.
  * @param  none.
  * @retval none.
  */
void dac_init(void)
```

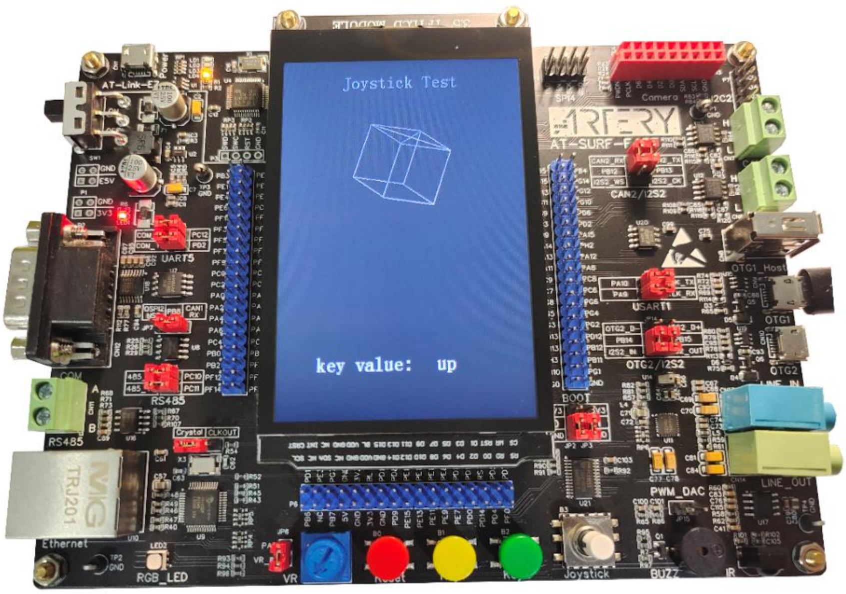
■ void dac_output_voltage_set(uint16_t voltage)

```
/**
  * @brief  dac output voltage set.
  * @param  voltage: output voltage
  *             the range is 0~3300 representing 0~3.300V.
  * @retval none.
  */
void dac_output_voltage_set(uint16_t voltage)
```

4.12.5 Download and verify

- When a key is pressed, the corresponding key value is displayed on the LCD screen.

Figure 33. Test result



4.13 Example 13: EEPROM communication

4.13.1 Introduction

AT32-SUFR features an EEPROM connector (part number 24C02) which has a 256-byte density and is connected to AT32 MCU via I²C bus. This section describes how to read/write from/to the EEPROM connector via AT32 MCU I²C interface as well as how to print the read/write results through a serial interface.

I²C (Inter-Integrated Circuit bus) is a bidirectional and two-wire bus developed by Philips Semiconductor Company. The bus interface serves as an interface between microcontrollers and the serial I²C bus. The I²C bus standard has been used in more than 1000 ICs over more than 50 companies. Besides, the I²C bus may be used for purposes, including SMBus (system management bus) and PMBus (power management bus).

I²C main features:

— Two buses: SDA and SCL

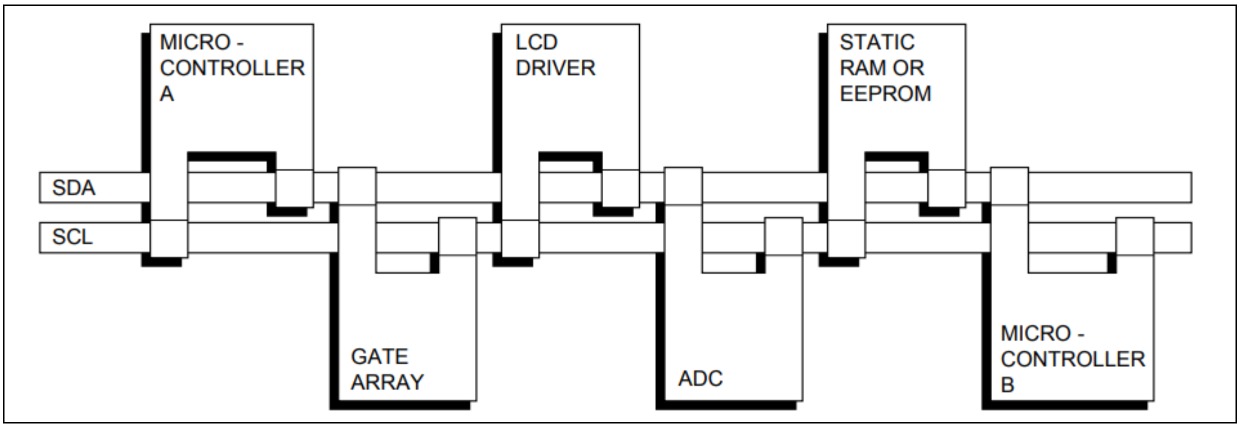
— Each slave connected to the bus has a unique address for host to acknowledge them based on their particular slave addresses

— Multi-controller bus, including conflict detection and arbitration, to avoid two or more controllers' simultaneous enabling data transfer

— Support different communication speeds:
up to 100 kbit/s in Standard mode;
up to 400 kbit/s in Fast mode;
up to 1 Mbit/s in Fast mode plus

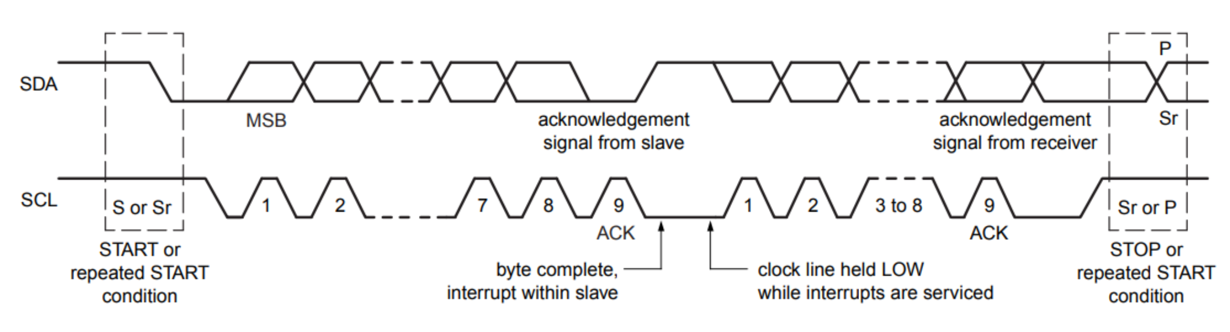
— The total number of IC that can be connected to the same bus only depends on the maximum bus capacitance

Figure 34. Application example of I²C bus



The I²C bus communication always starts with a START condition and terminates with a STOP condition. After the completion of each byte (8 bits) transfer, the receiver of the 9th bit data will pull low the SDA line and send an ACK signal. If the SDA line is not pulled low, an NACK is sent. The host stops the communication upon the receipt of NACK signal.

Figure 35. I²C bus data format



This application example gives a description of three communication modes used to access EEPROM. The three communication modes are available for user's selection according to their needs.

— Polling mode: transmit data by software polling
— Interrupt mode: transmit data by interrupt operation
— DMA mode: transmit data by DMA

4.13.2 Resource requirements

- Hardware resources

  AT-SURF-F437 Board

- Software resources

  AT32F435_437_Firmware_Library_V2.x.x\project\at_sufr_f437\examples\eeprom

4.13.3 Hardware design

Hardware resources used in the application example is 24C02.

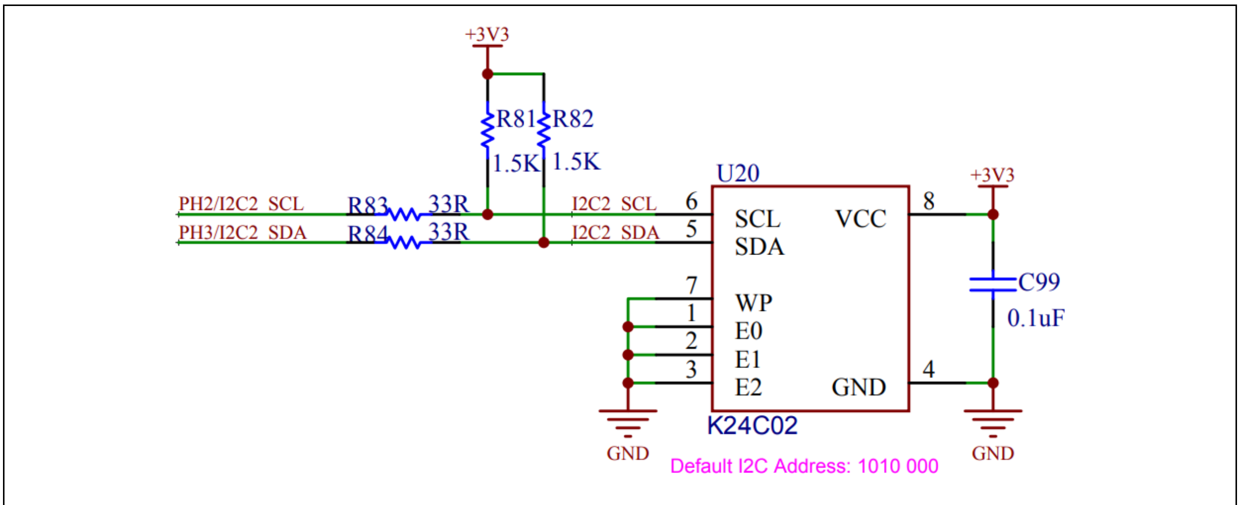
Table 16 gives the corresponding pins:

Table 16. Hardware resources

No.	PIN Name	Peripheral function	Description
1	PH2	I2C2 SCL	SCL line
2	PH3	I2C2 SDA	SDA line

Figure 36 shows the schematic diagram of 24C02.

Figure 36. Schematic diagram of 24C02



4.13.4 Software design

1) Use different modes to communicate with EEPROM
   - Initialize I^{2}C interface
   - Write data through polling mode
   - Read data through polling mode
   - Write data through interrupt mode
   - Read data through interrupt mode
   - Write data through DMA mode
   - Read data through DMA mode
2) Code
   - main function

```
int main(void)
{
  i2c_status_type i2c_status;

  /* Initialize system clock */
  system_clock_config();

  /* Initialize interrupt priority group*/
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

  /* Initialize delay function */
  delay_init();

  /* Initialize LCD */
  lcd_init(LCD_DISPLAY_VERTICAL);

  /* Initialize EEPROM */
  eeprom_init(EE_I2C_CLKCTRL_400K);

  /* Display information */
  lcd_string_show(10, 20, 200, 24, 24, (uint8_t *)"EEPROM Test");

  /* Write data to EEPROM in polling mode*/
  if((i2c_status = eeprom_data_write(&hi2c2, EE_MODE_POLL,
EEPROM_I2C_ADDRESS, 0, tx_buf1, BUF_SIZE, I2C_TIMEOUT)) != I2C_OK)
  {
    error_handler(i2c_status);
  }

  delay_ms(1);

  /* Read from EEPROM in polling mode */
  if((i2c_status = eeprom_data_read(&hi2c2, EE_MODE_POLL, EEPROM_I2C_ADDRESS,
```

```
0, rx_buf1, BUF_SIZE, I2C_TIMEOUT)) != I2C_OK)
  {
    error_handler(i2c_status);
  }

  delay_ms(1);

  /* Write data to EEPROM in interrupt mode */
  if((i2c_status = eeprom_data_write(&hi2c2, EE_MODE_INT, EEPROM_I2C_ADDRESS,
0, tx_buf2, BUF_SIZE, I2C_TIMEOUT)) != I2C_OK)
  {
    error_handler(i2c_status);
  }

  delay_ms(1);

  /* Read from EEPROM in interrupt mode */
  if((i2c_status = eeprom_data_read(&hi2c2, EE_MODE_INT, EEPROM_I2C_ADDRESS, 0,
rx_buf2, BUF_SIZE, I2C_TIMEOUT)) != I2C_OK)
  {
    error_handler(i2c_status);
  }

  delay_ms(1);

  /* Write data to EEPROM in DMA mode */
  if((i2c_status = eeprom_data_write(&hi2c2, EE_MODE_DMA, EEPROM_I2C_ADDRESS,
0, tx_buf3, BUF_SIZE, I2C_TIMEOUT)) != I2C_OK)
  {
    error_handler(i2c_status);
  }

  delay_ms(1);

  /* Read from EEPROM in DMA mode */
  if((i2c_status = eeprom_data_read(&hi2c2, EE_MODE_DMA, EEPROM_I2C_ADDRESS,
0, rx_buf3, BUF_SIZE, I2C_TIMEOUT)) != I2C_OK)
  {
    error_handler(i2c_status);
  }

  /* Compare the TX and RX data */
  if((buffer_compare(tx_buf1, rx_buf1, BUF_SIZE) == 0) &&
     (buffer_compare(tx_buf2, rx_buf2, BUF_SIZE) == 0) &&
     (buffer_compare(tx_buf3, rx_buf3, BUF_SIZE) == 0))
  {
```

```
    lcd_string_show(10, 60, 310, 24, 24, (uint8_t *)"eeprom write/read ok");
  }
  else
  {
    error_handler(i2c_status);
  }

  while(1)
  {
  }
}
```

■ void eeprom_init(void)

```
/**
  * @brief   eeprom_init.
  * @param   none.
  * @retval none.
  */
void eeprom_init(void)
```

■ eeprom_data_read()

```
/**
  * @brief   read data from eeprom.
  * @param   hi2c: the handle points to the operation information.
  * @param   mode: i2c transfer mode.
  *            - EE_MODE_POLL: transmits data through polling mode.
  *            - EE_MODE_INT: transmits data through interrupt mode.
  *            - EE_MODE_DMA: transmits data through dma mode.
  * @param   address: eeprom address.
  * @param   mem_address: eeprom memory address.
  * @param   pdata: data buffer.
  * @param   number: the number of data to be transferred.
  * @param   timeout: maximum waiting time.
  * @retval i2c status.
  */
i2c_status_type eeprom_data_read(i2c_handle_type* hi2c, eeprom_i2c_mode_type mode,
uint16_t address, uint16_t mem_address, uint8_t* pdata, uint16_t number, uint32_t timeout)
```

■ void eeprom_data_write()

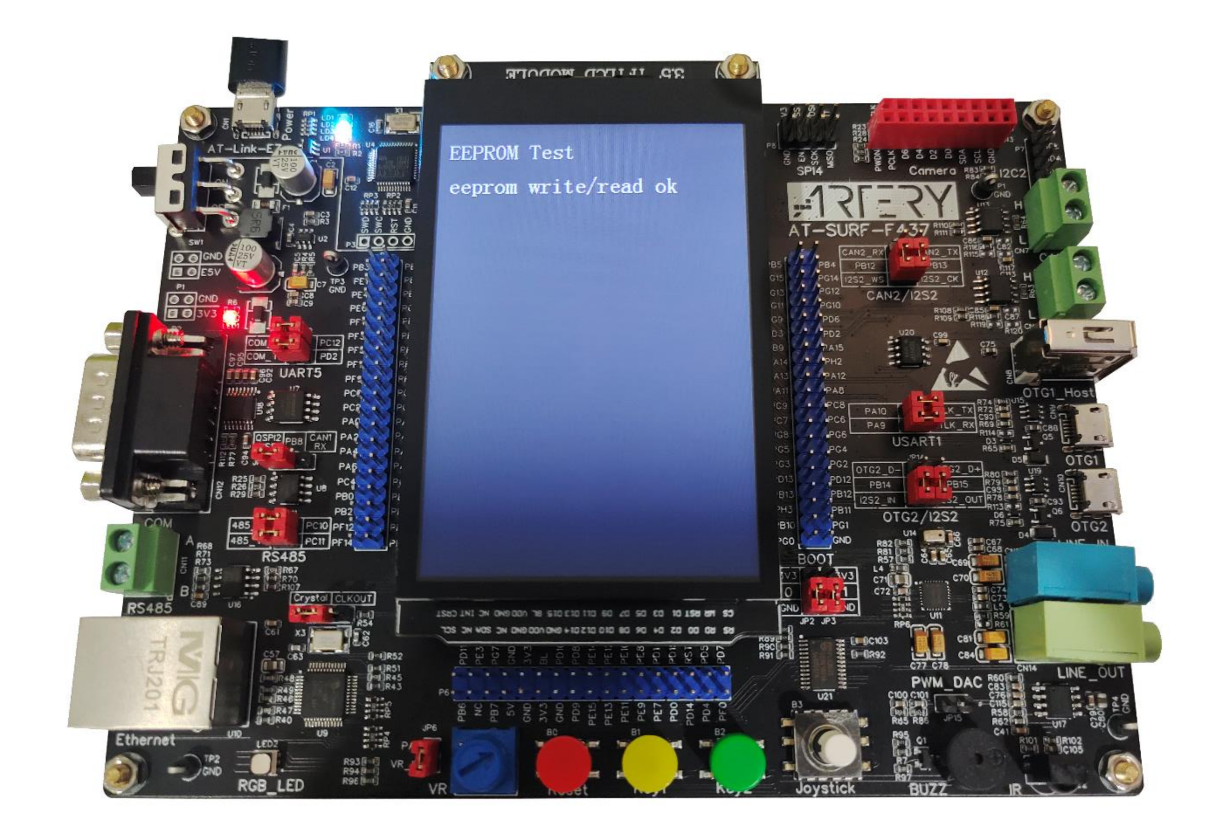
```
/**
  * @brief   write data to eeprom.
  * @param   hi2c: the handle points to the operation information.
  * @param   mode: i2c transfer mode.
  *            - EE_MODE_POLL: transmits data through polling mode.
  *            - EE_MODE_INT: transmits data through interrupt mode.
  *            - EE_MODE_DMA: transmits data through dma mode.
  * @param   address: eeprom address.
  * @param   mem_address: eeprom memory address.
```

```
  * @param    pdata: data buffer.
  * @param    number: the number of data to be transferred.
  * @param    timeout: maximum waiting time.
  * @retval i2c status.
  */
i2c_status_type eeprom_data_write(i2c_handle_type* hi2c, eeprom_i2c_mode_type mode,
uint16_t address, uint16_t mem_address, uint8_t* pdata, uint16_t number, uint32_t timeout)
```

4.13.5 Download and verify

- If communication OK and the data written and read are identical, the "eeprom write/read ok" message is shown on LCD.
- If communication error, the "eeprom write/read error" message is shown on LCD.

Figure 37. Test result



4.14 Example 14: TFT LCD application

4.14.1 Introduction

The TFT LCD refers to the liquid crystal display with thin film transistor. As each pixel in TFT LCD is controllable, each node is relatively independent and can also be controlled together. This feature ensures quicker display response and precise control of display color gradation, producing better color fidelity of TFT LCD. Despite its good luminance, high contrast, enhanced layering and bright colors, there is still room of improvement for the TFT LCD in terms of power consumption and higher costs.

AT32 SUFR board features a 3.5-inch TFT LCD display with a resolution of 480*320, connected to AT32 MCU using an XMC peripheral.

4.14.2 Resource requirements

- Hardware resources

  AT-SURF-F437 Board

- Software resources

  AT32F435_437_Firmware_Library_V2.x.x\project\at_sufr_f437\examples\tft_lcd

4.14.3 Hardware design

Hardware resources used in the application example are TFT LCD display and PCA9555 IO extension connector. Table 17 lists the corresponding pins:

Table 17. Hardware resources

No.	PIN Name	Peripheral function	Description
1	PG0	XMC_A10	-
2	PD14	XMC_D0	-
3	PD15	XMC_D1	-
4	PD0	XMC_D2	-
5	PD1	XMC_D3	-
6	PE7	XMC_D4	-
7	PE8	XMC_D5	-
8	PE9	XMC_D6	-
9	PE10	XMC_D7	-
10	PE11	XMC_D8	-
11	PE12	XMC_D9	-
12	PE13	XMC_D10	-
13	PE14	XMC_D11	-
14	PE15	XMC_D12	-
15	PD8	XMC_D13	-
16	PD9	XMC_D14	-
17	PD10	XMC_D15	-
18	PD7	XMC_NE1	-
19	PD5	XMC_NWE	-
20	PD4	XMC_NOE	-
21	NRST	NRST	LCD reset

Table 18. PCA9555 connector

No.	PIN Name	Peripheral function	Description
1	IO0_0	LCD_BL_CTRL	LCD backlight control

Figure 38 shows the schematic diagram of TFT LCD.

Figure 38. Schematic diagram of TFT LCD

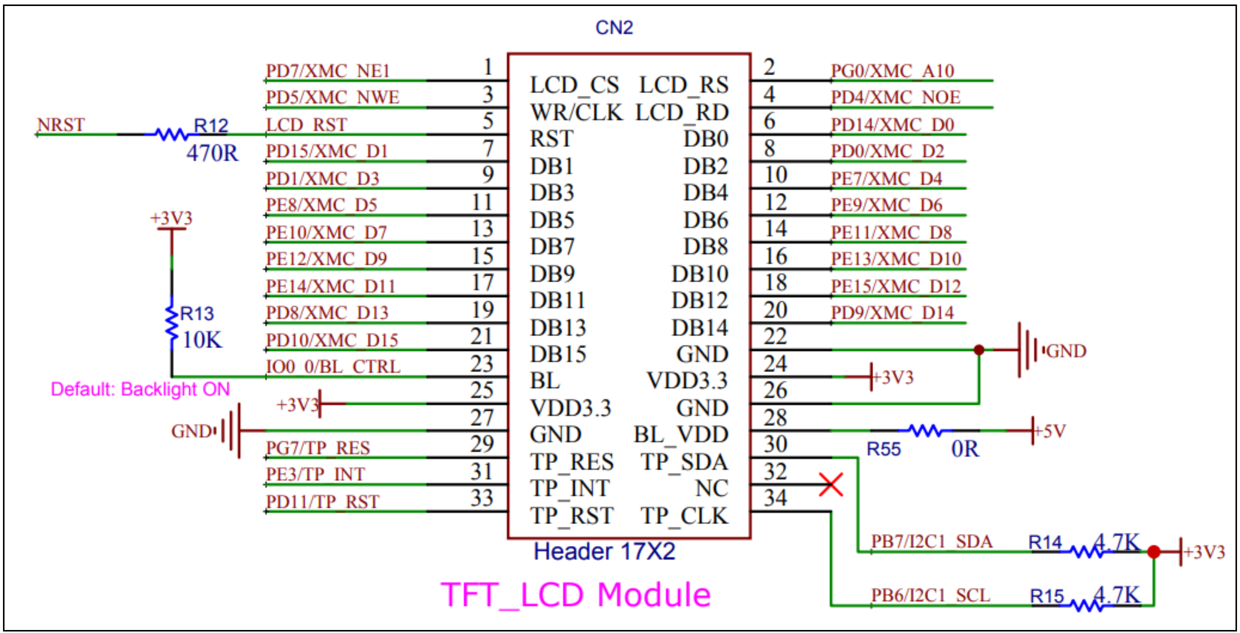
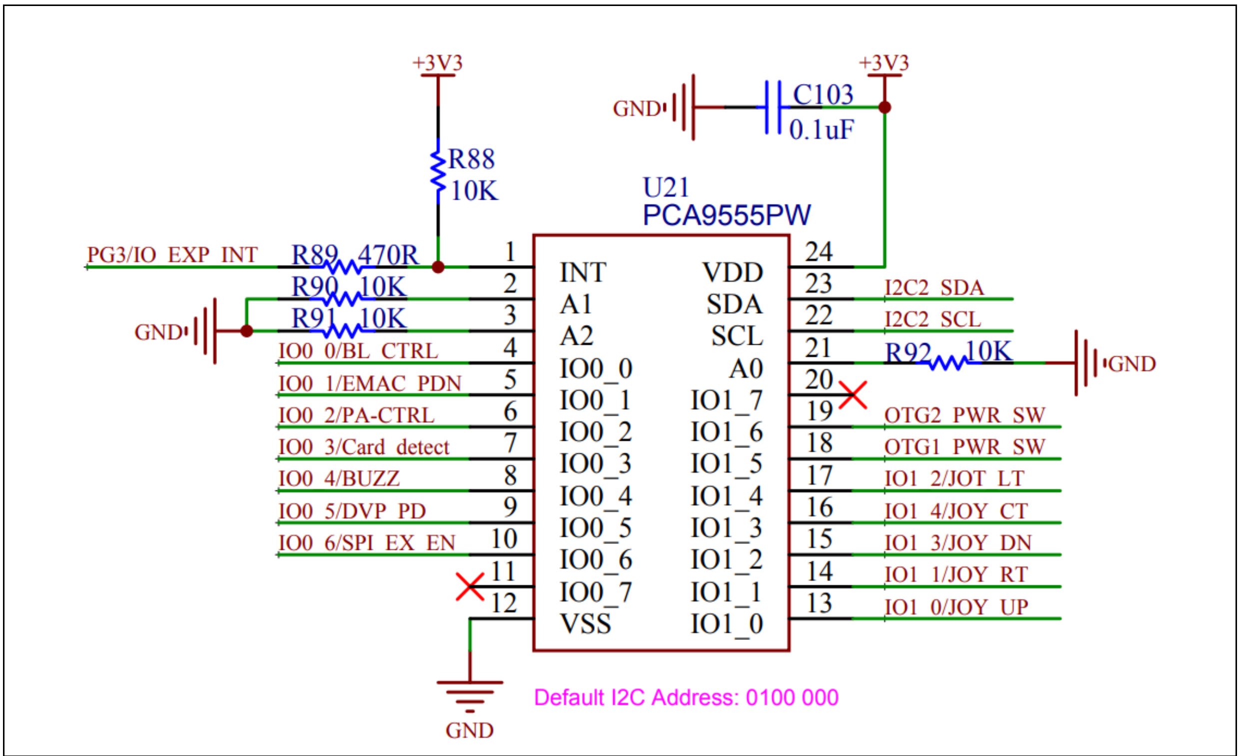


Figure 39. Schematic diagram of PCA9555



4.14.4 Software design

1) TFT LCD test
   - Initialize TFT LCD
   - Show information on LCD
2) Code
   - main function

```
int main(void)
{
    uint8_t step = 0;

    /* Initialize system clock */
    system_clock_config();

    /* Initialize interrupt priority group */
    nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

    /* Initialize delay function */
    delay_init();

    /* Initialize LCD */
    lcd_init(LCD_DISPLAY_VERTICAL);

    while(1)
    {
        /* Switch display color */
        switch(step)
        {
            case 0: lcd_clear(WHITE ); break;
            case 1: lcd_clear(BLUE    ); break;
            case 2: lcd_clear(BRED    ); break;
            case 3: lcd_clear(GBLUE ); break;
            case 4: lcd_clear(RED      ); break;
            case 5: lcd_clear(BRRED ); break;
            case 6: lcd_clear(GREEN ); break;
            case 7: lcd_clear(YELLOW); break;
            default: step = 0; break;
        }

        /* Display information */
        lcd_string_show(10, 20, 200, 24, 24, (uint8_t *)"TFT LCD Test");
        lcd_string_show(10, 60, 200, 24, 24, (uint8_t *)"2021-01-20");

        step++;

        if(step == 7)
```

```
        {
            step = 0;
        }

        delay_ms(1000);
    }
}
```

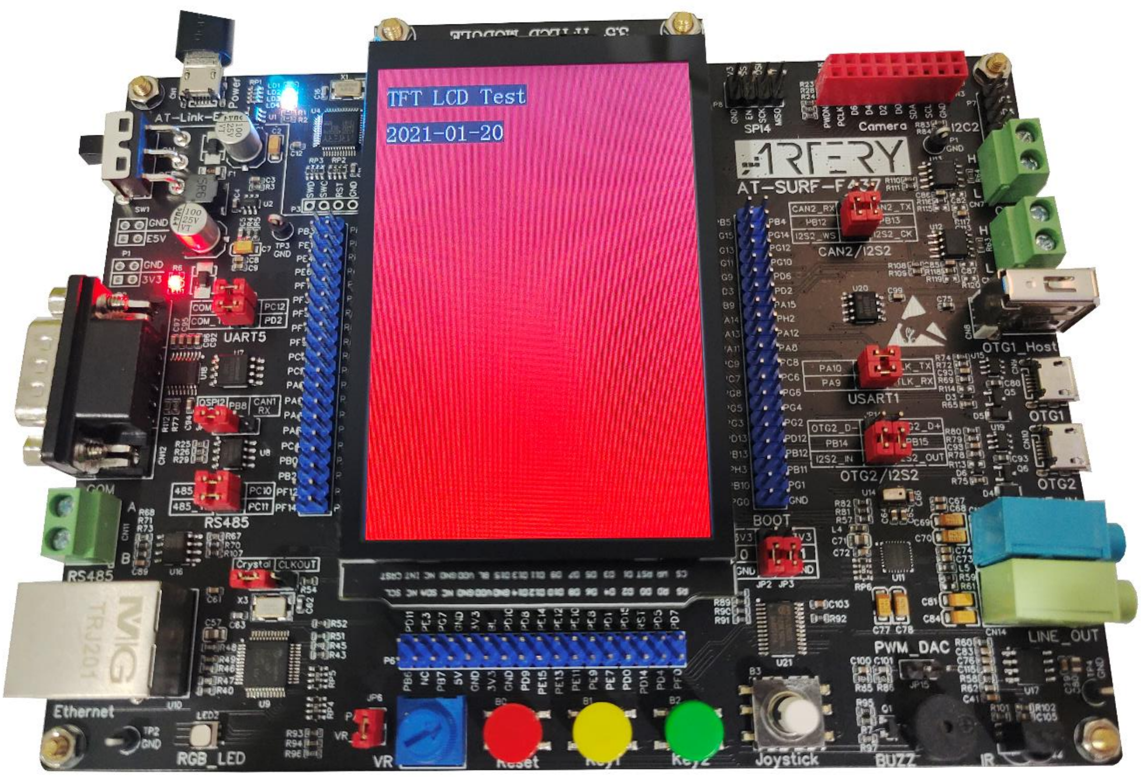
■ void lcd_init(void)

```
/**
  * @brief   initialization lcd screen
  * @param   direction: display direction
  * @retval none
  */
void lcd_direction(uint8_t direction)
```

4.14.5 Download and verify

■ Show information on LCD screen, and switch display backgroud per second.

Figure 40. Test result



4.15 Example 15: SD card communication

4.15.1 Information

The SD card is a flash-based memory card. It is extensively used in portable devices because it has many advantages such as small size, high capacity, fast data transfer and hot insertion feature.

The SD card specifications were originally defined by MEI (Matsushita Electric Company), Toshiba Corporation and SanDisk Corporation in 1999. They were built on MMC (MultiMediaCard), and thus MMC forward compatibility was kept. The SD card communication follows a standard communication protocol, which is available in the SD card protocol document.

Should the SD card is used to communicate with a microcontroller, just need read and write SD card directly. Should there are data in the SD card which need to be recognized by a computer, such data have to be in line with the format of a computer file system. There are many file systems applicable to microcontrollers, including FatFS, ThreadX FileX, RL-FlashFS and FreeRTOS FAT.

In our demo, FatFS file system is used. The FatFS is a generic file system module for small embedded systems. The FatFS is completely separated from the disk I/O player. Therefore it is independent of the hardware platform. It can be incorporated into small microcontrollers with limited resources.

AT32-SUFR board features a SD card which is connected to AT32F437 MCU through SDIO interface.

4.15.2 Resource requirements

- Hardware resources

  AT-SURF-F437 Board

- Software resources

  AT32F435_437_Firmware_Library_V2.x.x\project\at_sufr_f437\examples\sd_card

4.15.3 Hardware design

Hardware resources used in the application example are TFT LCD and SD card.
Table 19 lists the corresponding pins:

Table 19. Hardware resources

No.	PIN Name	Pin function	Description
1	PB4	SDIO_D0	SD card data pin 0
2	PC9	SDIO_D1	SD card data pin 1
3	PC10	SDIO_D2	SD card data pin 2
4	PC11	SDIO_D3	SD card data pin 3
5	PC12	SDIO_CLK	SD card clock pin
6	PD2	SDIO_CMD	SD card command pin

Table 20. PCA9555

No.	PIN Name	Pin function	Description
1	IO0_3	Card_detect	SD card detection pin

Figure 41 shows the schematic diagram of SD card.

Figure 41. Schematic diagram of SD card

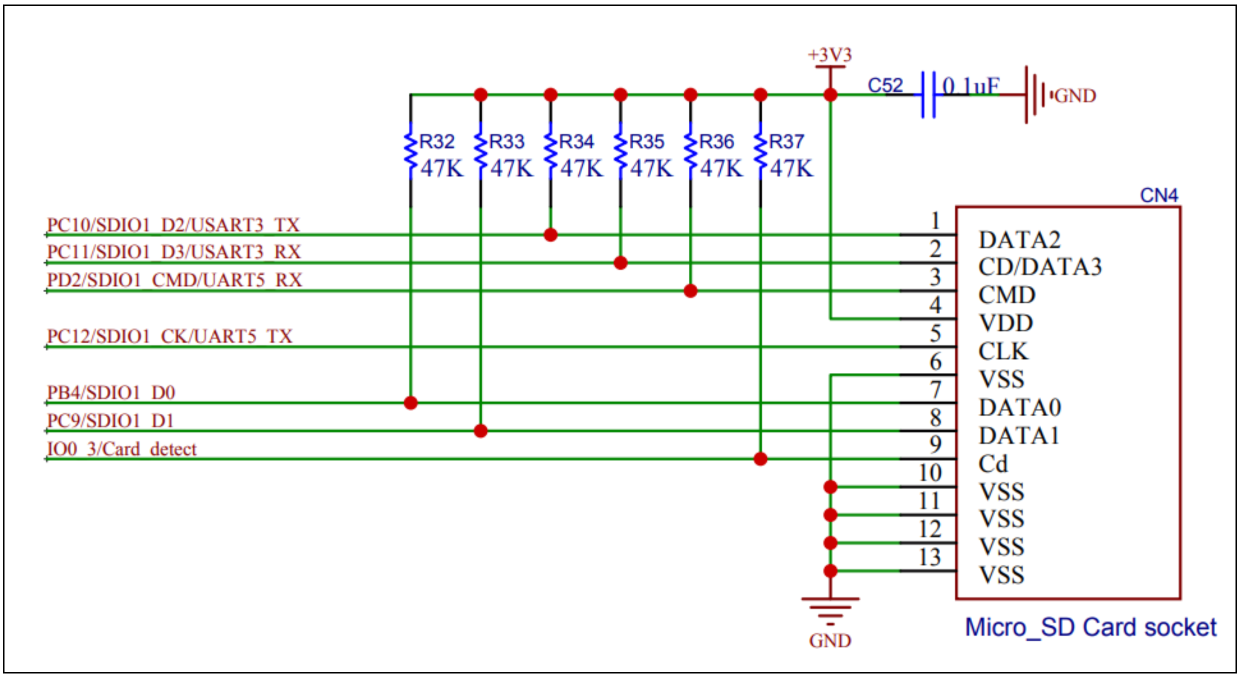
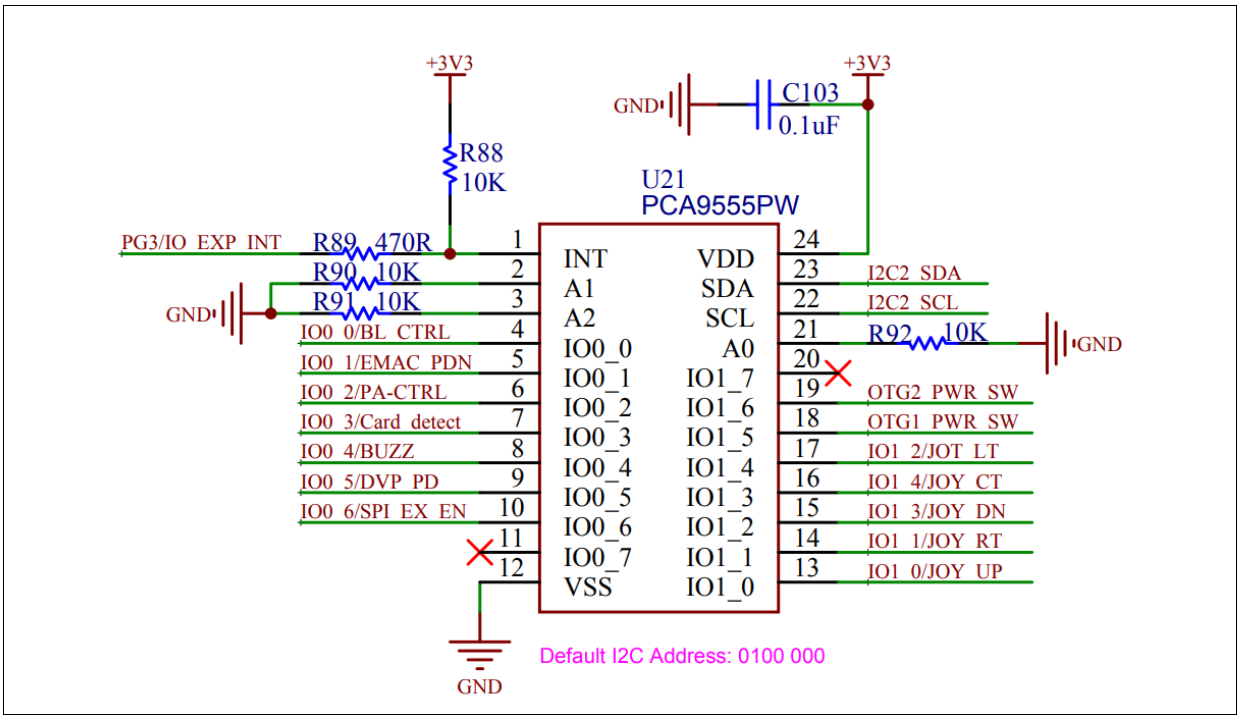


Figure 42. Schematic diagram of PCA9555



4.15.4 Software design

1) SD card test
   - Initialize TFT LCD
   - Initialize SD card
   - Read and write SD card
   - Display information on LCD
2) Code
   - main function

```
int main(void)
{
  FRESULT ret;
  UINT bytes_written = 0;
  UINT bytes_read = 0;
  DWORD fre_clust, fre_sect, tot_sect;
  FATFS* pt_fs;

  /* Initialize system clock*/
  system_clock_config();

  /* Initialize interrupt priority group */
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

  /* Initialize delay function */
  delay_init();

  /* Initialize LCD */
  lcd_init(LCD_DISPLAY_VERTICAL);

  /* Display information*/
  lcd_string_show(10, 20, 200, 24, 24, (uint8_t *)"SD Card Test");

  /* Initialize SD card and fatfs file system*/
  if(file_system_init() == SUCCESS)
  {
    lcd_string_show(10, 55, 200, 24, 24, (uint8_t *)"sd card init ok");
  }
  else
  {
    lcd_string_show(10, 55, 200, 24, 24, (uint8_t *)"sd card init error");
  }

  /* Open a file, or create one if no file is availabel */
  if((ret = f_open(&file, filename, FA_READ | FA_WRITE | FA_CREATE_ALWAYS)) != 0)
  {
```

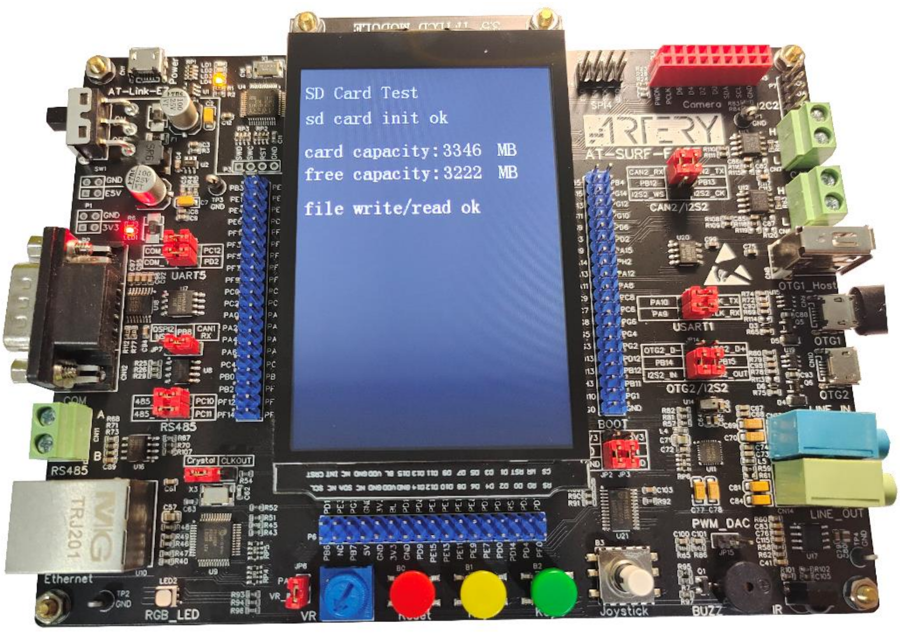
```
    error_handler(ret);
  }

  /* Write data to the file*/
  if((ret = f_write(&file, write_buf, sizeof(write_buf), &bytes_written)) != 0)
  {
    error_handler(ret);
  }

  /* Move the file pointer to the start location of the file */
  f_lseek(&file, 0);

  /* Read the file */
  if((ret = f_read(&file, read_buf, sizeof(read_buf), &bytes_read)) != 0)
  {
    error_handler(ret);
  }

  /* Close the file*/
  if((ret = f_close(&file)) != 0)
  {
    error_handler(ret);
  }

  pt_fs = &fs;

  /* Get free disk place */
  ret = f_getfree("1:", &fre_clust, &pt_fs);

  if(ret == FR_OK)
  {
    /* Get the total space and remaining space (unit: page) */
    tot_sect = (pt_fs->n_fatent - 2) * pt_fs->csize;
    fre_sect = fre_clust * pt_fs->csize;

    /* Calculate capacity */
    tot_sect = tot_sect * 512 / 1024 / 1024;
    fre_sect = fre_sect * 512 / 1024 / 1024;

    /* Show total capacity */
    lcd_string_show(10, 100, 300, 24, 24, (uint8_t *)"card capacity:       MB");
    lcd_num_show(182, 100, 200, 24, 24, tot_sect, 1);

    /* Show free capacity */
    lcd_string_show(10, 130, 300, 24, 24, (uint8_t *)"free capacity:       MB");
    lcd_num_show(182, 130, 200, 24, 24, fre_sect, 1);
```

```
    }

    /* Cancel work area */
    ret = f_mount(NULL, "1:", 1);

    /* Compare the data written and read */
    if(buffer_compare((uint8_t*)read_buf, (uint8_t*)write_buf, sizeof(write_buf)) == 0)
    {
      lcd_string_show(10, 175, 310, 24, 24, (uint8_t *)"file write/read ok");
    }
    else
    {
      lcd_string_show(10, 175, 310, 24, 24, (uint8_t *)"file write/read fail");
    }

    while(1)
    {

    }
}
```

4.15.5 Download and verify

- Read and write SD card, and compare the data written and read to check if they are correct.
- Display information on LCD.

Figure 43. Test result



4.16 Example 16: OTG test

4.16.1 Introduction

AT32F437 MCU embeds two separated OTGFS modules which support control transfer, bulk transfer, interrupt transfer and synchronous transfer.

The OTGFS module consists of OTGFS controller, PHY and a dedicated 1280-byte SRAM.

OTGFS is a USB full-speed dual-role device controller. It can be configured as a host or a slave through ID line. When the ID line is in floating mode, the OTGFS is used as a device; when the ID line is grounded, the OTGFS is used as a host. The OTG PHY has an internal 1.5KΩ pull-up resistor and 15KΩ pull-down resistor to support both device and host functions.

In device mode, the OTGFS supports one bi-directional control endpoint, seven IN endpoints, seven OUT endpoints;

In host mode, the OTGFS supports 16 host channels.

The suspend mode is also supported. After entering suspend mode, the OTGFS runs in power-saving mode.

As device, the OTGFS allocates a unified FIFO buffer for all OUT endpoints, and a separate FIFO buffer for each individual IN endpoint. As host, the OTGFS allocates a unified receive FIFO for all receive channels, and a unified transmit FIFO for all non-periodic transmit channels, and a unified transmit FIFO for all periodic transmit channels.

Suspend mode is supported in OTGFS module. The OTGFS will enter the suspend mode if no further bus signals are received within 3ms after the STOPPCLK bit is set in the OTGFS_PCGCCTL register. In addition, disabling PHY receive feature by setting the LP_MODE bit of the OTGFS_GCCFG register can help reduce power consumption.

The application example demonstrates how to use OTG as a virtual serial interface. Prior to use, a USB cable is needed to connect the OTG1 interface of SUFR board to a computer.

4.16.2 Resource requirements

- Hardware resources

  AT-SURF-F437 Board

- Software resources

  AT32F435_437_Firmware_Library_V2.x.x\project\at_sufr_f437\examples\otg

4.16.3 Hardware design

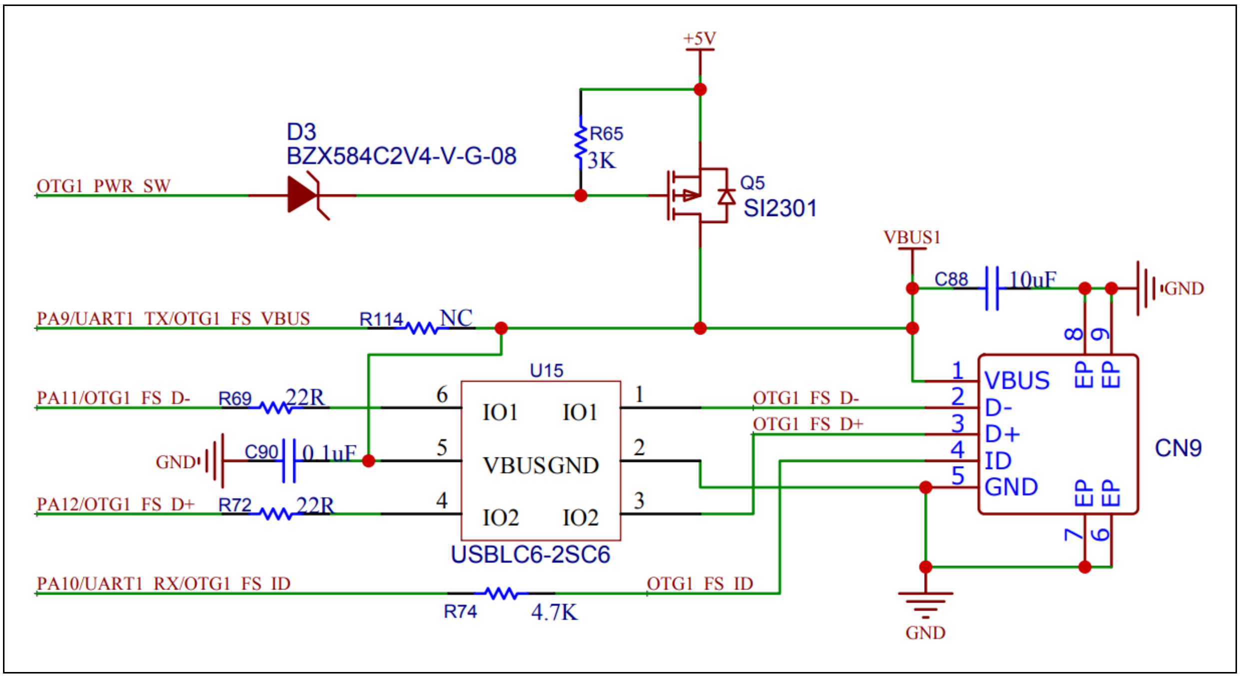
Hardware resources used in the example are TFT LCD display and USB.
Table 21 lists the corresponding pins:

Table 21. Hardware resources

No.	PIN Name	Pin function	Description
1	PA11	OTG1_FS_DM	-
2	PA12	OTG1_FS_DP	-
3	PA9	OTG1_FS_VBUS	-

Figure 44 shows the schematic diagram of OTG circuit.

Figure 44. Schematic diagram of OTG circuit



4.16.4 Software design

1) OTG test
   - Initialize OTG
   - PC side sends data to SUFR board
   - After the data are received by SUFR board, they are shown on LCD display
   - Send the received data back to PC side
2) Code
   - main

```
int main(void)
{
  uint16_t data_len;
  uint32_t timeout;
  uint8_t send_zero_packet = 0;

  /* Initialize system clock */
  system_clock_config();

  /* Initialize interrupt priority group */
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

  /* Initialize delay function */
  delay_init();

  /* Initialize LCD */
  lcd_init(LCD_DISPLAY_VERTICAL);

  /* Initialize otg */
```

```
    otg_init();

    /* Display information */
    lcd_string_show(10, 20, 300, 24, 24, (uint8_t *)"OTG Test");

    /* Display information */
    lcd_string_show(10, 50, 300, 24, 24, (uint8_t *)"Virtual Serial Port");

    while(1)
    {
      /* Get the received data */
      data_len = usb_vcp_get_rxdata(&otg_core_struct.dev, usb_buffer);

      if(data_len > 0 || send_zero_packet == 1)
      {
        /* Display the received data */
        lcd_fill(10, 80, 300, 140, WHITE);
        lcd_string_show(10, 80, 300, 24, 24, (uint8_t *)"Receive data:");
        lcd_string_show(10, 110, 300, 24, 24, usb_buffer);

        if(data_len > 0)
          send_zero_packet = 1;

        if(data_len == 0)
          send_zero_packet = 0;

        timeout = 5000000;
        do
        {
          /* Send data to host */
          if(usb_vcp_send_data(&otg_core_struct.dev, usb_buffer, data_len) == SUCCESS)
          {
            break;
          }
        }while(timeout --);
      }
    }
}
```

■ void otg_init(void)

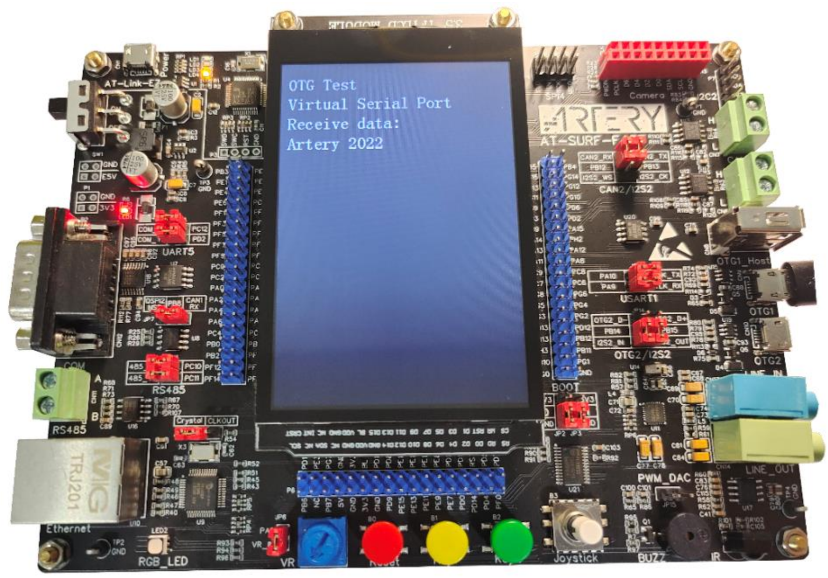
```
/**
  * @brief   otg init.
  * @param   none.
  * @retval none.
  */
void otg_init(void)
```

4.16.5 Download and verify

- PC sends “Artery 2022” to SUFR board via a serial port assistant.
- The received message is shown on LCD display.
- SUFR board sends the received message “Artery 2022” back to PC.

Figure 45. PC side serial port assistant

Figure 46. SUFR board



4.17 Example 17: SDRAM test

4.17.1 Introduction

The SDRAM is an acronym of Synchronous Dynamic random access memory. It is being widely used in the computer sector. In terms of microcontroller-based applications, the SRAM and SDRAM are both used as extended MCU memories. In other words, the SDRAM is generally used as high-density extended memory, and the SRAM as low-density extended memory.

The SRAM is directly accessible, and can save data without the need of refresh operations. In contrast, the SDRAM has an access command mechanism, and can save data only after being refreshed and charged within a certain period of time, otherwise, data loss may occur.

It is necessary to send corresponding commands before reading/writing SDRAM. These commands include pre-charge command, read command, write command and others. For more information, please refer to the SDRAM data sheet.

Table 22. SDRAM vs. SRAM

No.	SDRAM	SRAM
1	Periodic refresh and charge (up to 64ms) required	Without periodic refresh and charge
2	Inexpensive, high density	Expensive, low density
3	Lower power consumption	Higher power consumption
5	With bank mechanism	Without bank mechanism
5	Complicated control timing	Simple control timing

The SUFR board features a 32MB SDRAM chip (part number: W9825G6KH-6). The SDRAM has a 16-bit data width, and is connected to AT32 MCU via an XMC interface. The SDRAM clock is 288/3= 96 MHz, at a maximum frequency of 288M.

4.17.2 Resource requirements

- Hardware resources

  AT-SURF-F437 Board
- Software resources

  AT32F435_437_Firmware_Library_V2.x.x\project\at_sufr_f437\examples\sdram

4.17.3 Hardware design

Hardware resources used in the application example are TFT LCD and SDRAM (W9825G6KH-6). Table 23 lists the corresponding pins:

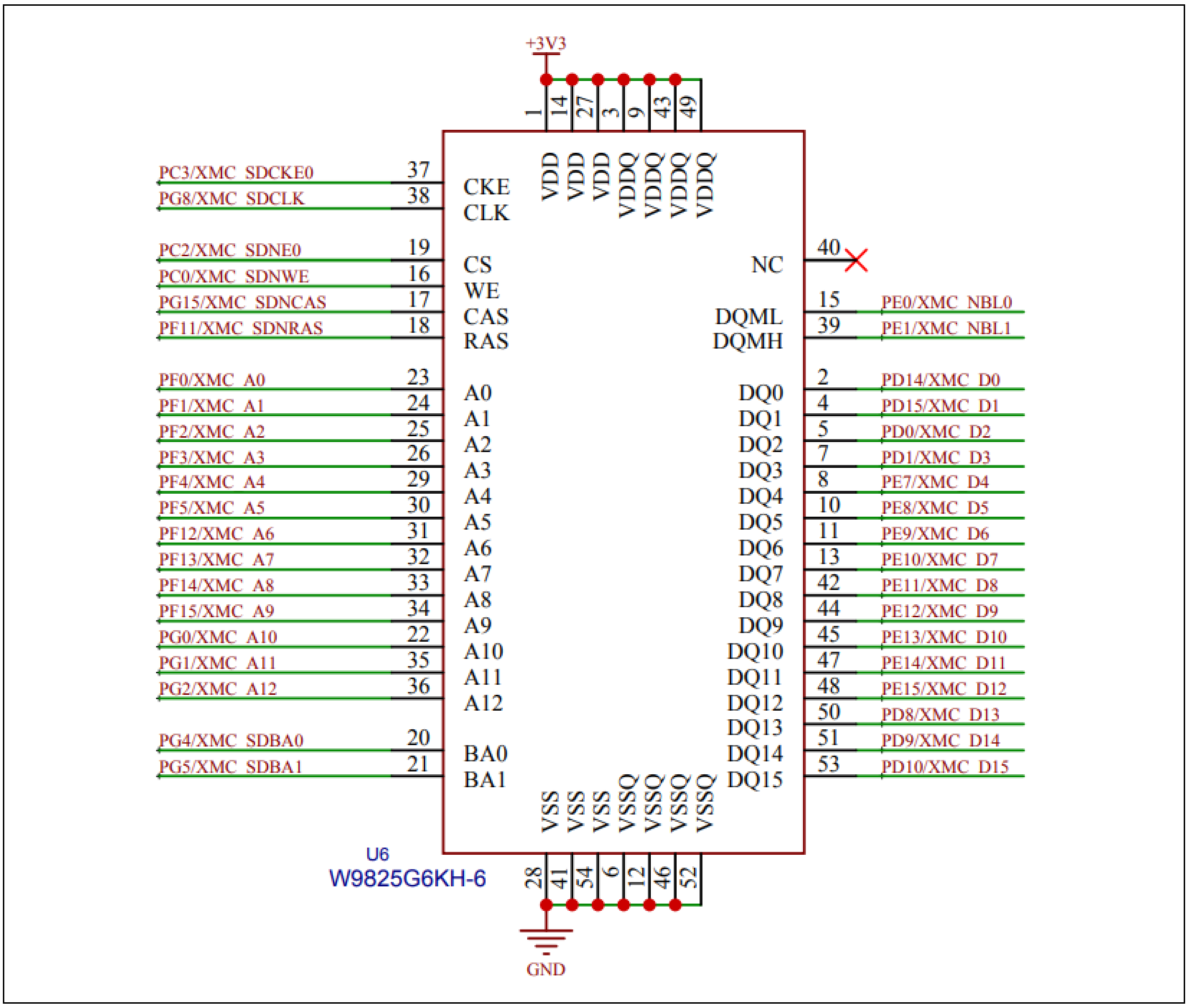
Table 23. Hardware resources

No.	PIN Name	Pin function	Description
1	PD14	XMC_D0	-
2	PD15	XMC_D1	-
3	PD0	XMC_D2	-
4	PD1	XMC_D3	-
5	PE7	XMC_D4	-
6	PE8	XMC_D5	-
7	PE9	XMC_D6	-
8	PE10	XMC_D7	-

No.	PIN Name	Pin function	Description
9	PE11	XMC_D8	-
10	PE12	XMC_D9	-
11	PE13	XMC_D10	-
12	PE14	XMC_D11	-
13	PE15	XMC_D12	-
14	PD8	XMC_D13	-
15	PD9	XMC_D14	-
16	PD10	XMC_D15	-
17	PF0	XMC_A0	-
18	PF1	XMC_A1	-
19	PF2	XMC_A2	-
20	PF3	XMC_A3	-
21	PF4	XMC_A4	-
22	PF5	XMC_A5	-
23	PF12	XMC_A6	-
24	PF13	XMC_A7	-
25	PF14	XMC_A8	-
26	PF15	XMC_A9	-
27	PG0	XMC_A10	-
28	PG1	XMC_A11	-
29	PG2	XMC_A12	-
30	PC3	XMC_SDCKE0	-
31	PG8	XMC_SDCLK	-
32	PC2	XMC_SDNE0	-
33	PC0	XMC_SDNWE	-
34	PG15	XMC_SDNCAS	-
35	PF11	XMC_SDNRAS	-
36	PG4	XMC_SDBA0	-
37	PG5	XMC_SDBA1	-
38	PE0	XMC_NBL0	-
39	PE1	XMC_NBL1	-

Figure 47 shows the schematic diagram of SDRAM.

Figure 47. Schematic diagram of SDRAM



4.17.4 Software design

1) SDRAM test
    - Initialize TFT LCD
    - Initialize SDRAM
    - Write data to SDRAM
    - Read data from SDRAM
    - Display information on LCD screen
2) Code
    - main function

```
int main(void)
{
    uint16_t i;

    /* Initialize system clock*/
    system_clock_config();
```

```
    /* Initialize interrupt priority group */
    nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

    /* Initialize delay function */
    delay_init();

    /* Initialize LCD */
    lcd_init(LCD_DISPLAY_VERTICAL);

    /* initialize sdram */
    sdram_init();

    /* Display informaiton*/
    lcd_string_show(10, 20, 200, 24, 24, (uint8_t *)"SDRAM Test");

    /* Initialize data*/
    for(i = 0; i < BUF_SIZE; i++)
    {
      write_buf[i] = i;
    }

    /* Write data to SDRAM */
    sdram_data_write(0, write_buf, BUF_SIZE);

    /* Read data from SDRAM */
    sdram_data_read(0, read_buf, BUF_SIZE);

    /* Compare the data written and read */
    if(buffer_compare((uint8_t *)write_buf, (uint8_t *)read_buf, BUF_SIZE * 2) == 0)
    {
      lcd_string_show(10, 60, 310, 24, 24, (uint8_t *)"sdram write/read ok");
    }
    else
    {
      lcd_string_show(10, 60, 310, 24, 24, (uint8_t *)"sdram write/read ok");
    }

    while(1)
    {

    }
}
```

- void sdram_init(void)

```
/**
  * @brief   init sdram interface
  * @param   none
  * @retval none
  */
void sdram_init(void)
```

- sdram_data_write

```
/**
  * @brief   writes a half-word buffer to the sdram memory.
  * @param   pbuffer : pointer to buffer.
  * @param   writeaddr : sdram memory internal address from which the data will be
  *            written.
  * @param   numhalfwordtowrite : number of half-words to write.
  * @retval none
  */
void sdram_data_write(uint32_t writeaddr, uint16_t* pbuffer, uint32_t numhalfwordtowrite)
```

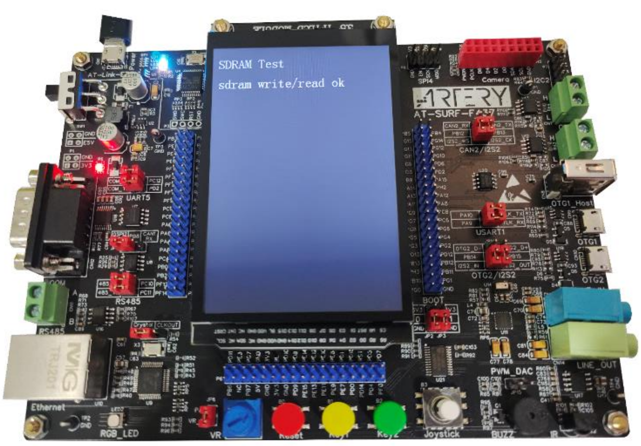
- sdram_data_read

```
/**
  * @brief   reads a block of data from the sdram.
  * @param   pbuffer : pointer to the buffer that receives the data read from the
  *            sdram memory.
  * @param   readaddr : sdram memory internal address to read from.
  * @param   numhalfwordtoread : number of half-words to read.
  * @retval none
  */
void sdram_data_read(uint32_t readaddr, uint16_t* pbuffer, uint32_t numhalfwordtoread)
```

4.17.5 Download and verify

- Write data to SDRAM.
- Read data from SDRAM.
- Compare the data written and read, and display test result on LCD.

Figure 48. Test result



4.18 Example 18: IrDA application

4.18.1 Introduction

As a wireless and contactless remote control, the infrared remote control features strong anti-interference, low power consumption and low cost, and thus it is widely used in the household appliances remote control devices. The principle of it is to transmit and receive data based on infrared rays. In other words, the transmitter sends the modulated infrared rays through an infrared-emitting diode, and the receive side receives the infrared rays through an IR receiver and demodulate the received data.

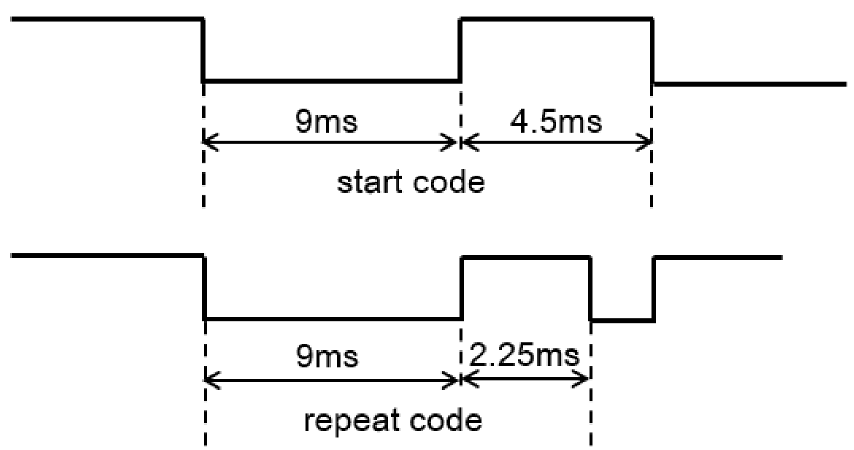
The basic principle of infrared remote control is similar among the application systems, with the only difference being the infrared coding. For infrared remote control, the NEC and Philips RC5 are the commonly used infrared coding protocols. Besides, the infrared coding custom is also supported. This application example uses NEC as an infrared coding protocol.

NEC protocol introduction

Start code transfer

Each data transfer starts with a Start bit, in the form of 9ms low level + 4.5ms high level. Keeping the button pressed will cause a repeat frame, instead of a data frame, to be transferred. The repeat frame is sent in the format of 9ms low level + 2.25 ms high level.

Figure 49. NEC start bit transfer

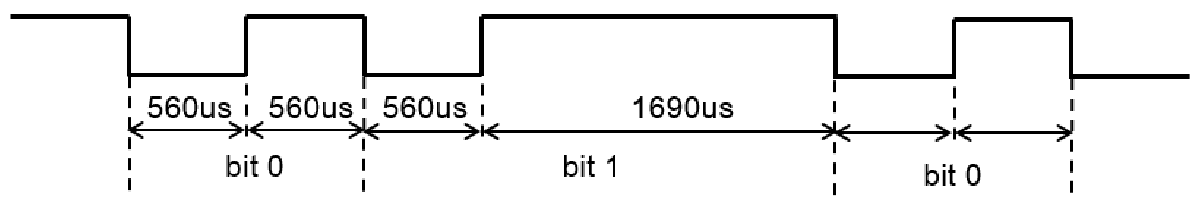


Bit transfer

0 transfer: 1.12ms cycle, 560us low level, 560us high level

1 transfer: 2.25ms cycle, 560us low level, 1690us high level

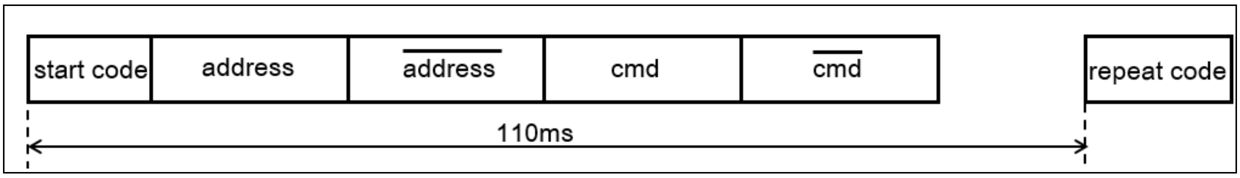
Figure 50. NEC bit transfer format



Frame format

The data frame format consists of four bytes, start code + address + inverse code of address + command + inverse code of command. The inverse codes of address and command are used to verify data integrity. Keeping the button pressed will cause the repeat code to be sent every 110ms.

Figure 51. NEC data frame format



In the example code, the data is analyzed by measuring the level width through input capture feature of a timer.

4.18.2 Resource requirements

- Hardware resources

  AT-SURF-F437 Board

- Software resources

  AT32F435_437_Firmware_Library_V2.x.x\project\at_sufr_f437\examples\infrared_receiver

4.18.3 Hardware design

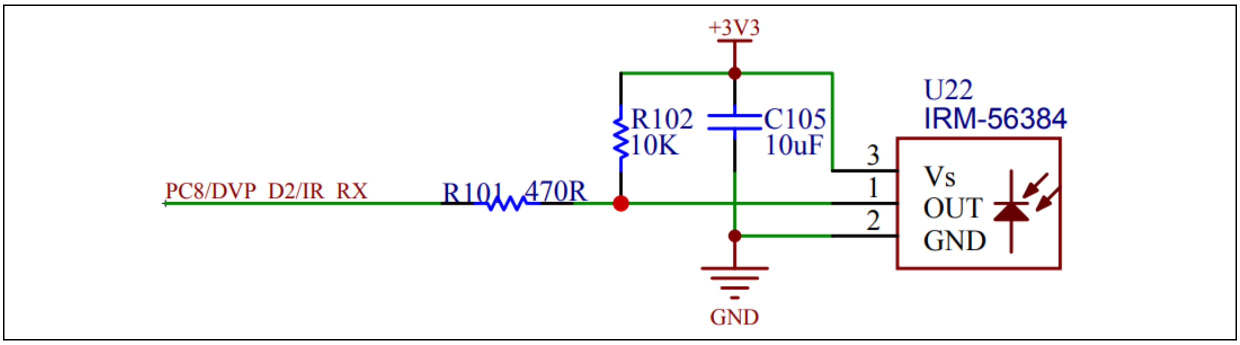
Hardware resources used in the application example are TFT LCD display and IRM-56384 IR receiver. Table 23 lists the corresponding pins:

Table 24. Hardware resources

No.	PIN Name	Peripheral function	Description
1	PC8	TMR3_CH3	Input capture

Figure 52 shows the schematic diagram of infrared receiver.

Figure 52. Schematic diagram of infrared receiver



4.18.4 Software design

1) Infrared receive test
   - Initialize TFT LCD
   - Initialize TMR for input capture
   - Display information on LCD
2) Code
   - main function

```
int main(void)
{
   uint8_t key_value;

   /* Initialize system clock */
   system_clock_config();

   /* Initialize interrupt priority group */
   nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

   /* Initialize delay function */
   delay_init();

   /* Initialize LCD */
   lcd_init(LCD_DISPLAY_VERTICAL);

   /* Initialize infrared receiver */
   infrared_receiver_init();

   /* Display information*/
   lcd_string_show(10, 20, 200, 24, 24, (uint8_t *)"Infrared receiver Test");

   while(1)
   {
      /* Get infrared receiver key */
      if(infrared_receiver_key_get(&key_value) == SUCCESS)
      {
         /* Show key address and command */
         lcd_string_show(10, 90, 310, 24, 24, (uint8_t *)"key   address:     ");
         lcd_num_show(178, 90, 310, 24, 24, (uint8_t)(key_value >> 8), 3);

         lcd_string_show(10, 120, 310, 24, 24, (uint8_t *)"key        cmd:     ");
         lcd_num_show(178, 120, 310, 24, 24, (uint8_t)(key_value & 0xFF), 3);
      }
   }
}
```

■ void infrared_receiver_init(void)

```
/**
  * @brief   infrared receiver init.
  * @param   none.
  * @retval none.
  */
void infrared_receiver_init(void)
```

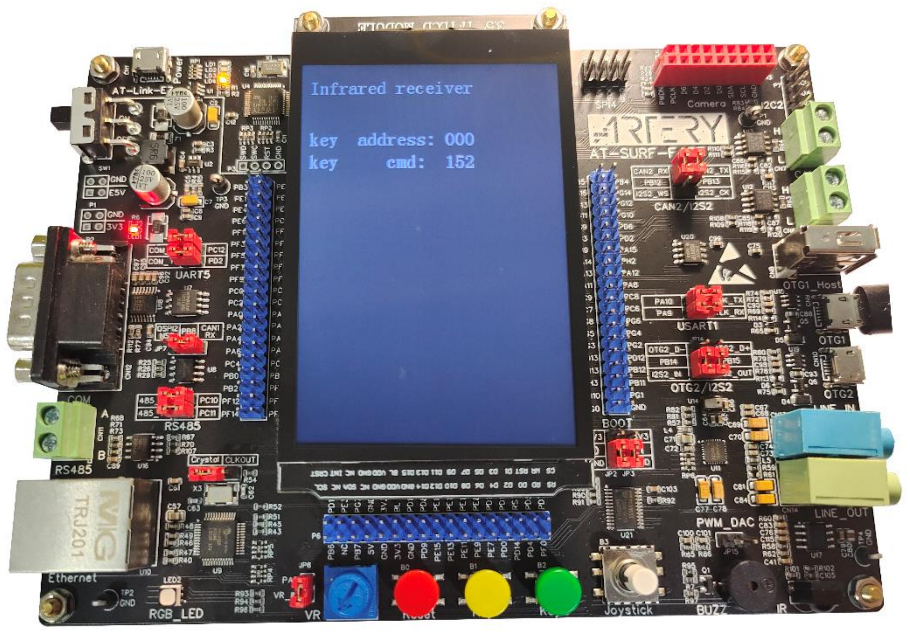
■ error_status infrared_receiver_key_get(uint16_t *val)

```
/**
  * @brief   get infrared key.
  * @param   the pointer of key value.
  * @retval error_status.
  */
error_status infrared_ receiver_key_get(uint16_t *val)
```

4.18.5 Download and verify

- Initialize TMR to receive data.
- Point a remote control at the IR receiver and press the button on it.
- After valid data are received, they are shown on LCD.

Figure 53. Test result



4.19 Example 19: Low-power mode

4.19.1 Introduction

AT32F437 series MCU requires a 2.6V to 3.6V operating voltage supply. There are three low-power modes available to save power when the CPU does not need to be kept running. They are Sleep mode, Deepsleep mode and Standby mode. It is up to the user to select the mode that gives the best compromise between low-power consumption, CPU run time and speed. In terms of the level of power consumption, they are Sleep mode > Deepsleep mode > Standby mode.

4.19.2 Resource requirements

- Hardware resources

  AT-SURF-F437 Board

- Software resources

  AT32F435_437_Firmware_Library_V2.x.x\project\at_sufr_f437\examples\low_power_mode

4.19.3 Hardware design

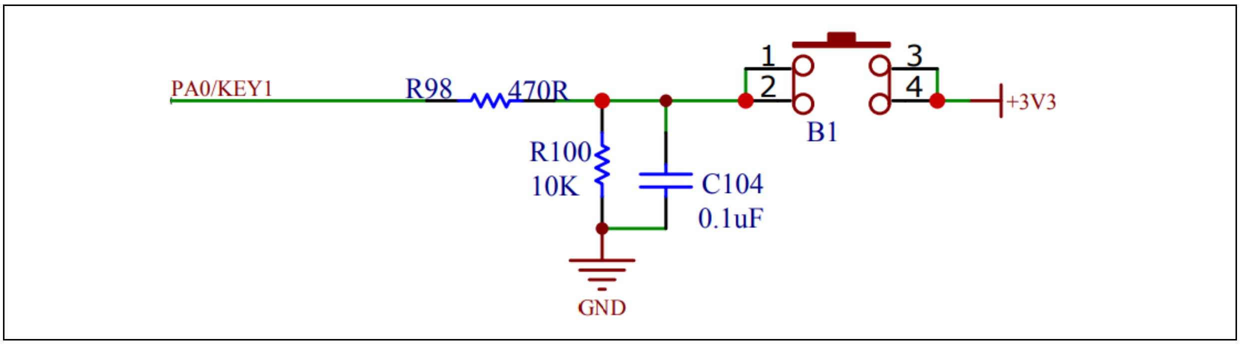
Hardware resources used in the application example are two separate buttons.
Table 25 lists the corresponding pins:

Table 25. Hardware resources

No.	PIN Name	Peripheral function	Description
1	PA0	EXINT line 0	Button for waking up from low-power mode

Figure 54 shows the schematic diagram of button circuit.

Figure 54. Schematic diagram of button circuit



4.19.4 Software design

1) Low-power consumption test
   - Set EXINT rising edge triggered mode for the button to wake up low-power mode
   - MCU enters low-power mode
   - Press the button and wake up low-power mode
2) Code
   - main function

```
int main(void)
{
    /* Initialize system clock */
    system_clock_config();

    /* Initialize interrupt priority group */
    nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

    /* Initialize delay function */
    delay_init();

    /* Initialize LCD */
    lcd_init(LCD_DISPLAY_VERTICAL);

    /* Initialize wakeup pin (button 2) */
    wakeup_pin_init();

    /* Enable PWC clock */
    crm_periph_clock_enable(CRM_PWC_PERIPH_CLOCK, TRUE);

    /* Show information */
    lcd_string_show(5, 20, 310, 24, 24, (uint8_t *)"Low power mode test");

    /* Delay 1s */
    delay_ms(1000);

#if defined TEST_SLEEP_MODE

    /* Show information */
    lcd_string_show(5, 60, 310, 24, 24, (uint8_t *)"Enter sleep mode");
    lcd_string_show(5, 90, 310, 24, 24, (uint8_t *)"Prese button 2 to wakeup");

    /* Enter sleep mode */
    pwc_sleep_mode_enter(PWC_SLEEP_ENTER_WFI);

    /* Wake up from sleep mode */
    lcd_string_show(5, 140, 310, 24, 24, (uint8_t *)"Wakeup from sleep mode");
```

```
#elif defined TEST_DEEPSLEEP_MODE

    /* Show information*/
    lcd_string_show(5, 60, 310, 24, 24, (uint8_t *)"Enter deepsleep mode");
    lcd_string_show(5, 90, 310, 24, 24, (uint8_t *)"Prese button 2 to wakeup");

    /* Set LDO mode */
    pwc_voltage_regulate_set(PWC_REGULATOR_LOW_POWER);

    /* Enter deepsleep mode */
    pwc_deep_sleep_mode_enter(PWC_DEEP_SLEEP_ENTER_WFI);

    /* Initialize system clock */
    system_clock_config();

    /* Wake up from deepsleep mode */
    lcd_string_show(5, 140, 310, 24, 24, (uint8_t *)"Wakeup from deepsleep mode");

#endif

    while(1)
    {

    }
}
```

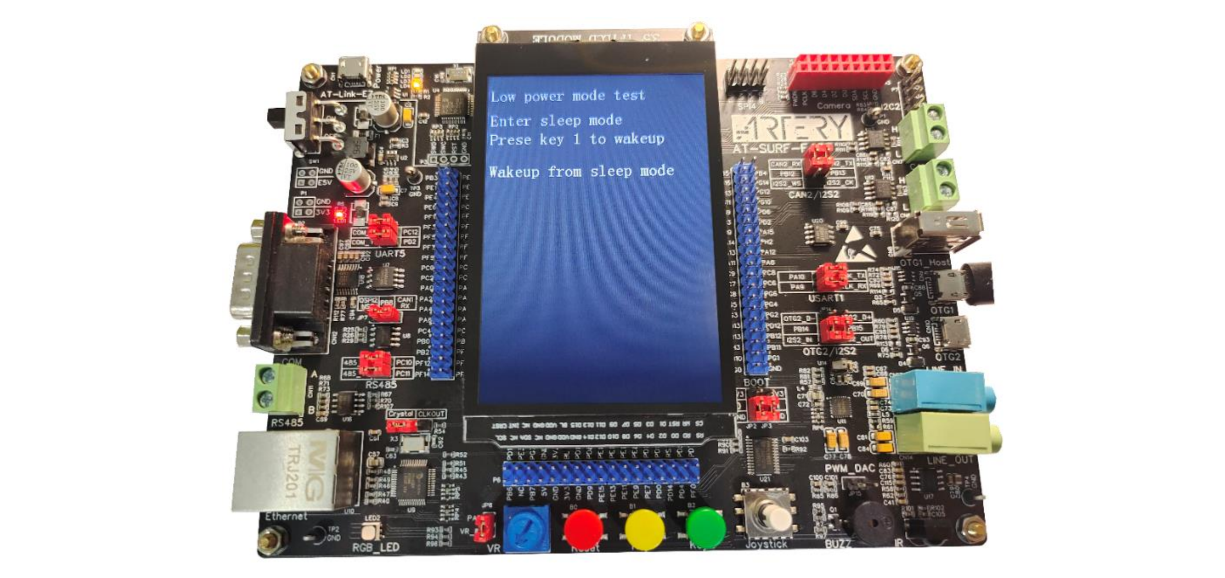
- void wakeup_pin_init(void)

```
/**
    * @brief   wakeup pin init.
    * @param   none.
    * @retval none.
    */
void wakeup_pin_init(void)
```

4.19.5 Download and verify

- Enter low-power mode one second after power-on
- When pressing button 2, exit low-power mode.

Figure 55. Test result



4.20 Example 20: QSPI FLASH application

4.20.1 Introduction

The Quad SPI is a 6-wire SPI interface. It is faster than traditional 4-wire SPI (two one-way data lines) as Quad SPI uses four data lines as opposed to just two data lines (one-way). In theory, the Quad SPI is four times faster than traditional ones, and therefore, the QSPI FLASH is almost four times (actually it is not the case because it involves clearing command bytes) as fast as that of traditional 4-wire SPI Flash. In addition, QSPI can be used as Flash extension area for program run.

The AT32 SUFR board features a 16 Mbyte Flash, part number W25Q128JVSIQ.

4.20.2 Resource requirements

- Hardware resources

  AT-SURF-F437 Board

- Software resources

  AT32F435_437_Firmware_Library_V2.x.x\project\at_sufr_f437\examples\qspi_flash

4.20.3 Hardware design

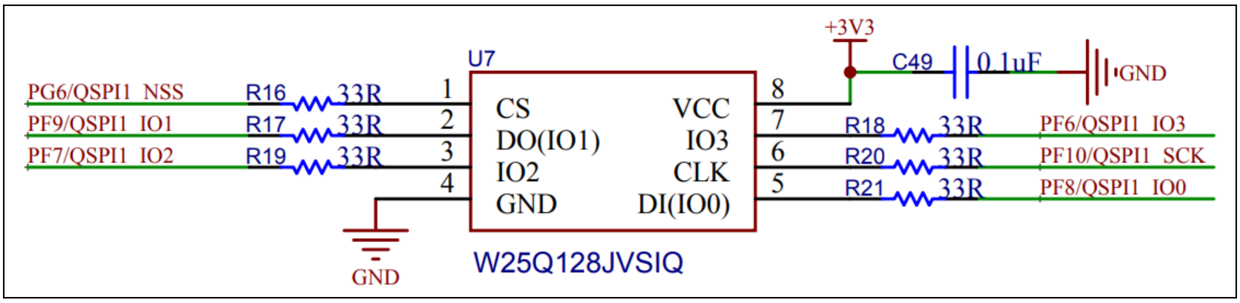
Hardware resources used in the application example are TFT LCD display and Flash (W25Q128JVSIQ). Table 26 lists the corresponding pins:

Table 26. Hardware resources

No.	PIN Name	Peripheral function	Description
1	PF10	QSPI1_CLK	Clock pin
2	PG6	QSPI1_NSS	Chip select pin
3	PF8	QSPI1_D0	Data pin 0
4	PF9	QSPI1_D1	Data pin 1
5	PF7	QSPI1_D2	Data pin 2
6	PF6	QSPI1_D3	Data pin 3

Figure 56 shows the schematic diagram of QSPI Flash.

Figure 56. Schematic diagram of QSPI FLASH



4.20.4 Software design

1) QSPI FLASH test
   - Initialize TFT LCD
   - Initialize QSPI FLASH
   - Write data to QSPI FLASH
   - Read data from QSPI FLASH
   - Display information on LCD screen
2) Code
   - main function

```
int main(void)
{
  uint16_t i;

  /* Initialize system clock */
  system_clock_config();

  /* Initialize interrupt priority group */
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

  /* Initialize delay function */
  delay_init();

  /* Initialize LCD */
  lcd_init(LCD_DISPLAY_VERTICAL);

  /* Initialize QSPI FLASH */
  qspi_flash_init();

  /* Show information*/
  lcd_string_show(10, 20, 200, 24, 24, (uint8_t *)"QSPI Flash Test");

  /* Initialize data */
  for(i = 0; i < BUF_SIZE; i++)
  {
    write_buf[i] = i % 256;
  }

  /* Erase sector 0 */
  qspi_flash_erase(0);

  /* Write data to QSPI FLASH */
  qspi_flash_data_write(0, write_buf, BUF_SIZE);

  /* Read QSPI FLASH */
```

```
    qspi_flash_data_read(0, read_buf, BUF_SIZE);

    /* Compare the data written and read */
    if(buffer_compare(write_buf, read_buf, BUF_SIZE) == 0)
    {
      lcd_string_show(10, 60, 310, 24, 24, (uint8_t *)"flash write/read ok");
    }
    else
    {
      lcd_string_show(10, 60, 310, 24, 24, (uint8_t *)"flash write/read ok");
    }

    while(1)
    {

    }
}
```

- void qspi_flash_init(void)

```
/**
  * @brief   initializes quad spi flash.
  * @param   none
  * @retval none
  */
void qspi_flash_init(void)
```

- void qspi_flash_data_write(uint32_t addr, uint8_t* buf, uint32_t total_len)

```
/**
  * @brief   qspi flash write data
  * @param   addr: the address for write
  * @param   total_len: the length for write
  * @param   buf: the pointer for write data
  * @retval none
  */
void qspi_flash_data_write(uint32_t addr, uint8_t* buf, uint32_t total_len)
```

- void qspi_flash_data_read(uint32_t addr, uint8_t* buf, uint32_t total_len)

```
/**
  * @brief   qspi flash read data
  * @param   addr: the address for read
  * @param   total_len: the length for read
  * @param   buf: the pointer for read data
  * @retval none
  */
void qspi_flash_data_read(uint32_t addr, uint8_t* buf, uint32_t total_len)
```

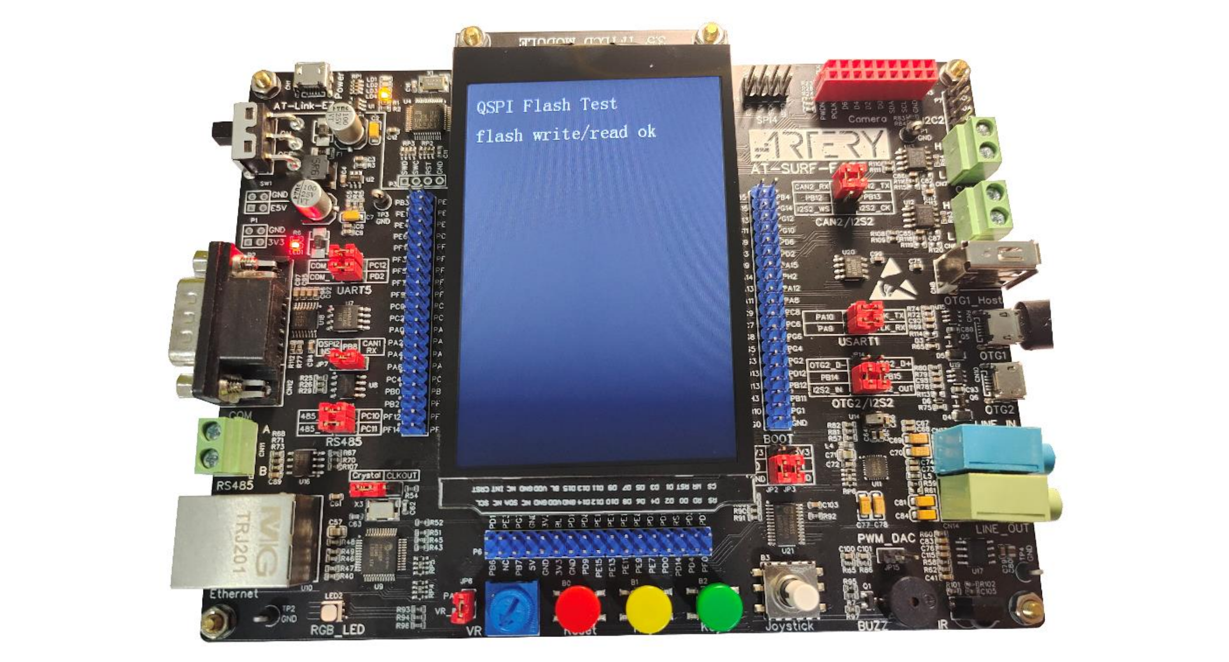
- void qspi_flash_erase(uint32_t sec_addr)

```
/**
  * @brief   qspi flash erase data
  * @param   sec_addr: the sector address for erase
  * @retval none
  */
void qspi_flash_erase(uint32_t sec_addr)
```

4.20.5 Download and verify

- Write data to QSPI FLASH.
- Read data from QSPI FLASH.
- Compare the data written and read to verify data integrity, and show result on LCD.

Figure 57. Test result



4.21 Example 21: QSPI SRAM application

4.21.1 Introduction

The QSPI SRAM is a QSPI interface-based SRAM. The Quad SPI is a 6-wire SPI interface, and faster than traditional 4-wire SPI (two one-way data lines) as Quad SPI uses four data lines as opposed to just two data lines (one-way). In theory, the Quad SPI is four times faster than traditional ones.

The AT32 SUFR board features an 8M byte-SRAM (part number: LY68L6400SLI). It should be noted that the Jumper caps must be set correctly when in use.

4.21.2 Resource requirements

- Hardware resources

  AT-SURF-F437 Board

- Software resources

  AT32F435_437_Firmware_Library_V2.x.x\project\at_sufr_f437\examples\qspi_sram

4.21.3 Hardware design

Hardware resources used in the application example are TFT LCD display and SRAM (part number: LY68L6400SLI). Table 27 lists the corresponding pins:

Table 27. Hardware resources

No.	PIN Name	Peripheral function	Description
1	PB1	QSPI2_CLK	Clock pin
2	PB8	QSPI2_NSS	Chip select pin
3	PB0	QSPI2_D0	Data pin 0
4	PG12	QSPI2_D1	Data pin 1
5	PG10	QSPI2_D2	Data pin 2
6	PA3	QSPI2_D3	Data pin 3

Figure 58 shows the schematic diagram of QSPI SRAM.

Figure 58. Schematic diagram of QSPI SRAM

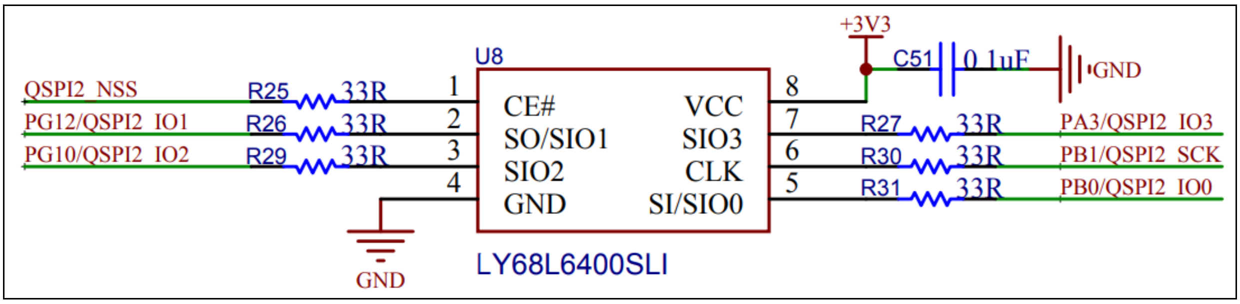
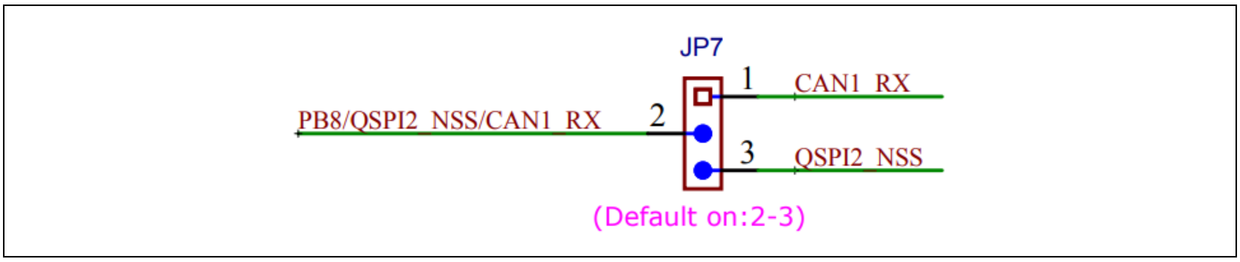


Figure 59. Schematic diagram of QSPI SRAM Jumper



4.21.4 Software design

1) QSPI SRAM test
   - Initialize TFT LCD
   - Initialize QSPI SRAM
   - Write data to QSPI SRAM
   - Read from QSPI SRAM
   - Display information on LCD screen
2) Code
   - main function

```
int main(void)
{
  uint16_t i;

  /* Initialize system clock */
  system_clock_config();

  /* Initialize interrupt priority group */
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

  /* Initialize delay function */
  delay_init();

  /* Initialize LCD */
  lcd_init(LCD_DISPLAY_VERTICAL);

  /* Initialize QSPI SRAM */
  qspi_sram_init();

  /* Display information*/
  lcd_string_show(10, 20, 200, 24, 24, (uint8_t *)"QSPI Sram Test");

  /* Initialize data */
  for(i = 0; i < BUF_SIZE; i++)
  {
    write_buf[i] = i % 256;
  }

  /* Write data to SRAM */
  qspi_sram_data_write(0, write_buf, BUF_SIZE);

  /* Read from SRAM */
  qspi_sram_data_read(0, read_buf, BUF_SIZE);

  /* Compare the data written and read */
```

```
    if(buffer_compare(write_buf, read_buf, BUF_SIZE) == 0)
    {
      lcd_string_show(10, 60, 310, 24, 24, (uint8_t *)"sram write/read ok");
    }
    else
    {
      lcd_string_show(10, 60, 310, 24, 24, (uint8_t *)"sram write/read ok");
    }

    while(1)
    {

    }
}
```

- void qspi_sram_init(void)

```
/**
  * @brief  initializes quad spi sram.
  * @param  none
  * @retval none
  */
void qspi_sram_init(void)
```

- void qspi_sram_data_read(uint32_t addr, uint8_t* buf, uint32_t total_len)

```
/**
  * @brief  qspi sram read data
  * @param  addr: the address for read
  * @param  total_len: the length for read
  * @param  buf: the pointer for read data
  * @retval none
  */
void qspi_sram_data_read(uint32_t addr, uint8_t* buf, uint32_t total_len)
```

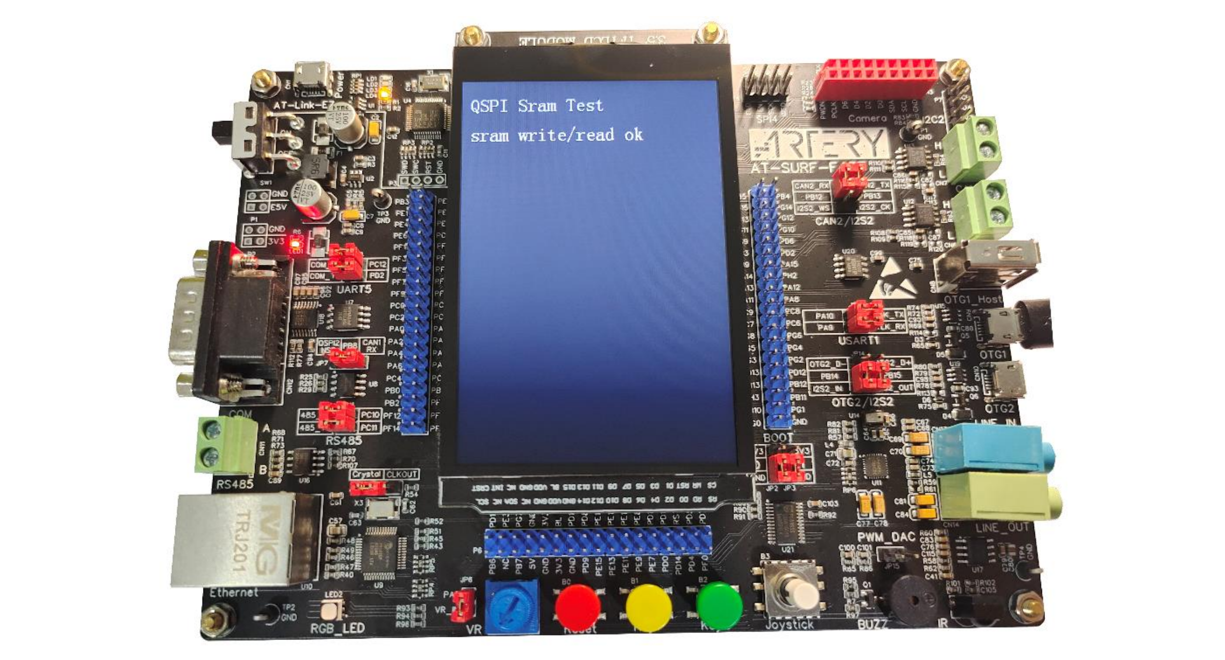
- void qspi_sram_data_write(uint32_t addr, uint8_t* buf, uint32_t total_len)

```
/**
  * @brief  qspi sram write data
  * @param  addr: the address for write
  * @param  total_len: the length for write
  * @param  buf: the pointer for write data
  * @retval none
  */
void qspi_sram_data_write(uint32_t addr, uint8_t* buf, uint32_t total_len)
```

4.21.5 Download and verify

- Write data to QSPI SRAM.
- Read data from QSPI SRAM.
- Compare the data written and read, and show test result on LCD.

Figure 60. Test result



4.22 Example 22: Audio player application

4.22.1 Information

The AT32 SUFR board embeds a WM898 audio codec. The WM8988 is a low power, high quality stereo CODEC designed for portable digital audio applications. The device integrates complete interfaces to two stereo headphone or line out ports. External components requirements are drastically reduced.

The WM8988 can operate as a master or a slave, with various master clock frequencies including 12 or 24 MHz for USB devices, or standard 256fs rates like 12.288MHz and 24.576MHz. Different audio sample rates such as 96 kHz, 48 kHz, and 44.1 kHz are supported.

The WM8988 operates at supply voltage of 1.8 to 3.6V. It is supplied in a very small and thin 4x4mm COL package, ideal for use in hand-held and portable systems.

The demo gives an example of reading a music song from the SD card, and sending the audio data, after being decoded by software, to the WM8988 to play the music. There are many formats for music play, including MP3, WAV, APE and FLAC. It should be noted that music documents must be stored in “MUSIC” folder under SD card boot directory prior to use.

4.22.2 Resource requirements

- Hardware resources

  AT-SURF-F437 Board

- Software resources

  AT32F435_437_Firmware_Library_V2.x.x\project\at_sufr_f437\examples\audio

4.22.3 Hardware design

Hardware resources used in the application example are TFT LCD display, PCA9555 IO extension connector, WM8988, SD card and buttons. Table 28 to 29 list the corresponding pins:

Table 28. PCA9555

No.	PIN Name	Pin function	Description
1	IO0_2	Power amplifier switch	-

Table 29. Hardware resources

No.	PIN Name	Peripheral function	Description
1	PH2	I2C2 SCL	-
2	PH3	I2C2 SDA	-
3	PB12	I2S2_WS	-
4	PB13	I2S2_CK	-
5	PB14	I2S2_SDIN	-
6	PB15	I2S2_SDOUT	-

Figure 61 shows the schematic diagram of WM8988.

Figure 61. Schematic diagram of WM8988

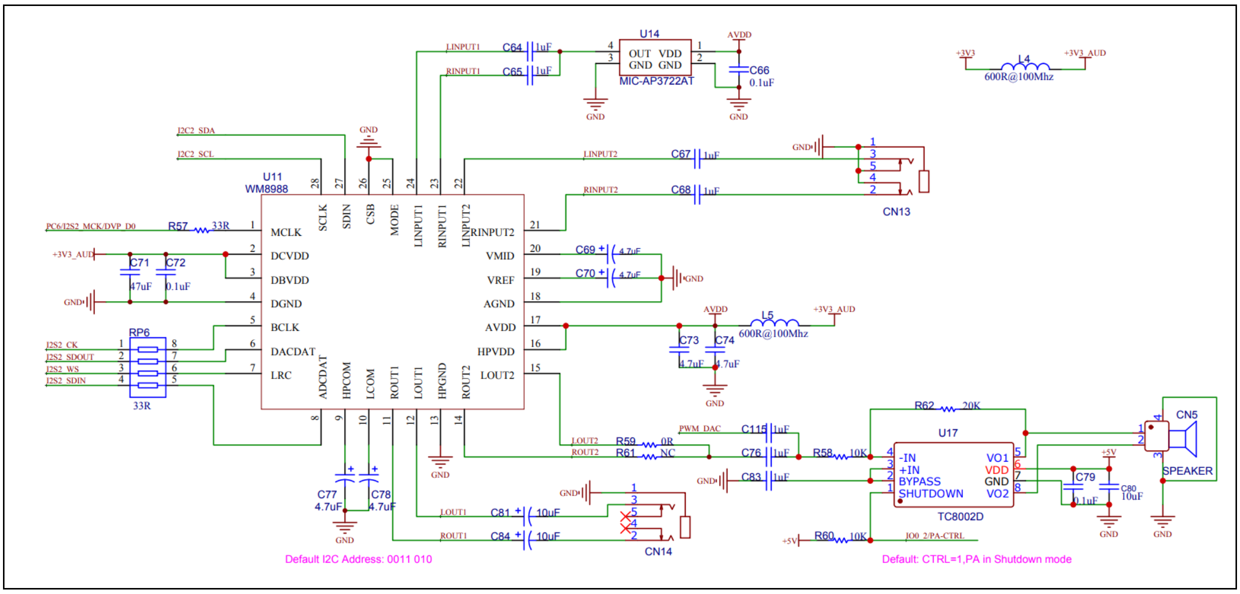


Figure 62. Schematic diagram WM8988 jumper

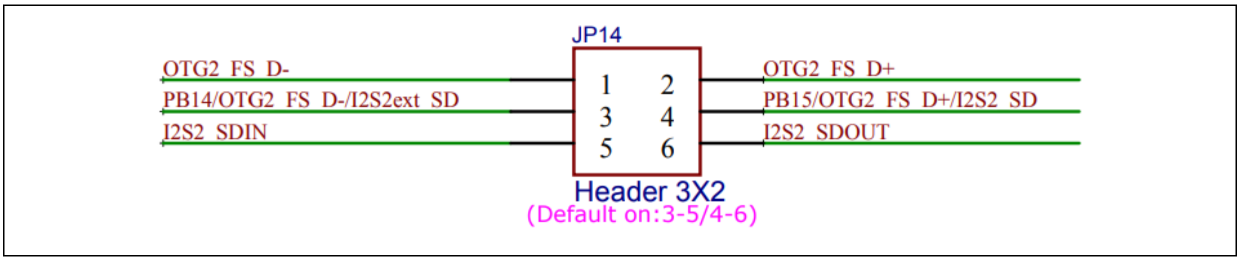
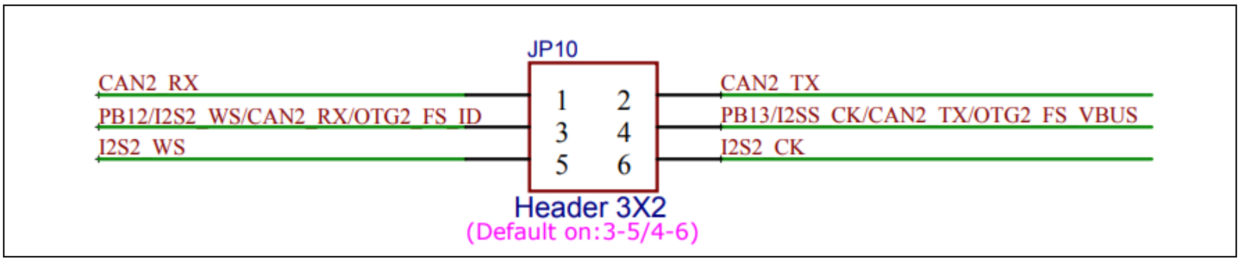


Figure 63. Schematic diagram WM8988 jumper



4.22.4 Software design

1) Audio test
   - Initialize TFT LCD
   - Initialize SD card
   - Initialize WM8988
   - Play music
   - Show music information on LCD
   - Switch, play and pause music through button control
   - Control sound volume using a variable resistor
2) Code
   - main function

```
int main(void)
{
   /* Initialize system clock */
   system_clock_config();

   /* Initialize interrupt priority group */
   nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

   /* Initialize delay function*/
   delay_init();

   /* Initialize LCD */
   lcd_init(LCD_DISPLAY_VERTICAL);

   /* Show information */
   lcd_string_show(10, 20, 200, 24, 24, (uint8_t *)"Audio Test");

   /* Initialize file system */
   if(file_system_init() != SUCCESS)
   {
      lcd_string_show(10, 55, 300, 24, 24, (uint8_t *)"sd card init error");
      while(1);
   }

   /* Initialize IO extension connector */
   pca9555_init(PCA_I2C_CLKCTRL_100K);

   /* Initialize keys */
   key_init();

   /* Initialize variable resistor */
   variable_resistor_init();
```

```
    /* Initialize audio */
    audio_init();

    /* Play music */
    music_play(&audio_info);

    while(1)
    {

    }
}
```

■ void audio_init(void)

```
/**
  * @brief  audio codec init
  * @param  none
  * @retval error status
  */
error_status audio_init(void)
```

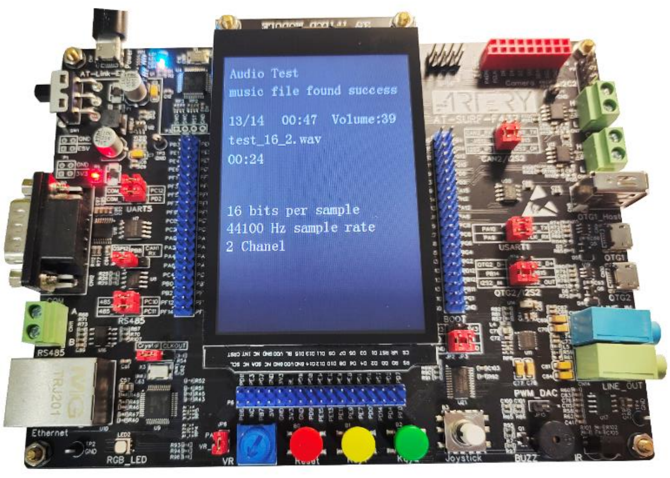
■ void music_play(audio_type *audio)

```
/**
  * @brief  play music.
  * @param  audio: audio information structure.
  * @retval none
  */
void music_play(audio_type *audio)
```

4.22.5 Download and verify

- After power-on, automatically search for and play music documents in the SD card.
- Switch songs with “KEY1” and “KEY2”, and pause or play music using JoyStick “OK” button.
- Adjust sound volume using variable resistor.
- Show test result on LCD display.

Figure 64. Test result



4.23 Example 23: Camera application

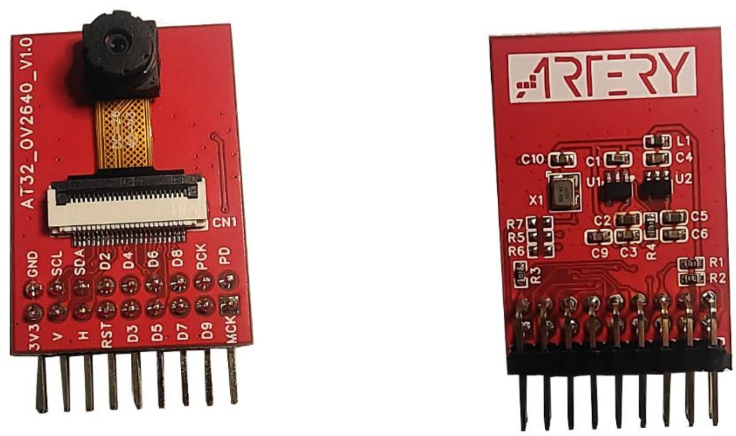
4.23.1 Introduction

The digital video parallel interface (DVP) is used to capture parallel data from CMOS video camera. It offers hardware synchronization mode and embedded synchronization mode, which can be selected based on camera output, to achieve frame synchronization and line synchronization. With frame rate control feature, it is also possible to adjust the number of frames captured per second according to the actual needs. With the crop window feature, it enables the user to reserve or discard certain areas. With the image resizing feature, it allows the user to reduce the number of pixels or lines according to certain proportions. The use of DMA (direct memory access) causes captured data to be sent to memory unit without having to occupy CPU resources. In this case, the user only needs to monitor the status of data reception through status interrupts and error interrupts.

— 8-bit, 10-bit, 12-bit and 14-bit parallel interfaces
— Support parallel interface data alignment
— Support hardware synchronization and embedded synchronization mode
— Capture data in a single frame or in continuous mode
— Frame rate control
— Crop window feature
— Image resizing feature
— Grayscale image binaryzation
— DMA single or burst transfer
— Interrupts upon frame capture complete, vertical synchronization status and horizontal synchronization status
— Error interrupts upon output buffer overflow and embedded synchronization code error

AT32 SUFR board features a 2-million-pixel camera, part number OV2640, connected to AT32 MCU via DVP interface.

Figure 65. Camera



The OV2640 image sensor is a low voltage CMOS device that provides the full functionality of a single-chip UXGA (1632x1232) and image sensor in a small footprint package. It provides full-frame, scaled or windowed 8-bit/10-bit images in a wide range of formats, controlled through the Serial Camera Control Bus (SCCB) interface. This product has an image array capable of operating at up to 15 frames per second in UXGA resolution with complete user control over image quality, formatting and output data transfer. All required image processing functions, including exposure control, gamma, white balance, color saturation, hue control, white pixel cancelling, noise cancelling, and more, are also programmable through the SCCB interface. The OV2640 also includes a compression engine for increased processing power. In addition, the product uses proprietary sensor technology to improve image quality by reducing or eliminating common lighting/electrical sources of image contamination, such as fixed pattern noise, smearing, etc., to produce a clean, fully stable color image.

— High sensitivity for low-light operation
— Standard SCCB interface
— Output support for Raw RGB, RGB565, RGB 555, GRB422, YUV(422/420) and YCbCr(4:2:2)
— Support image sizes: UXGA、SXGA、SVGA
— Automatic Exposure Control (AEC), Automatic Gain Control (AGC), Automatic White Balance (AWB), Automatic Band Filter (ABF) and Automatic Black-level Calibration (ABLC)
— Image quality controls including color saturation, gamma, sharpness (edge enhancement), lens correction, white pixel cancelling and noise cancelling

4.23.2 Resource requirements

- Hardware resources

  AT-SURF-F437 Board

- Software resources

  AT32F435_437_Firmware_Library_V2.x.x\project\at_sufr_f437\examples\camera

4.23.3 Hardware design

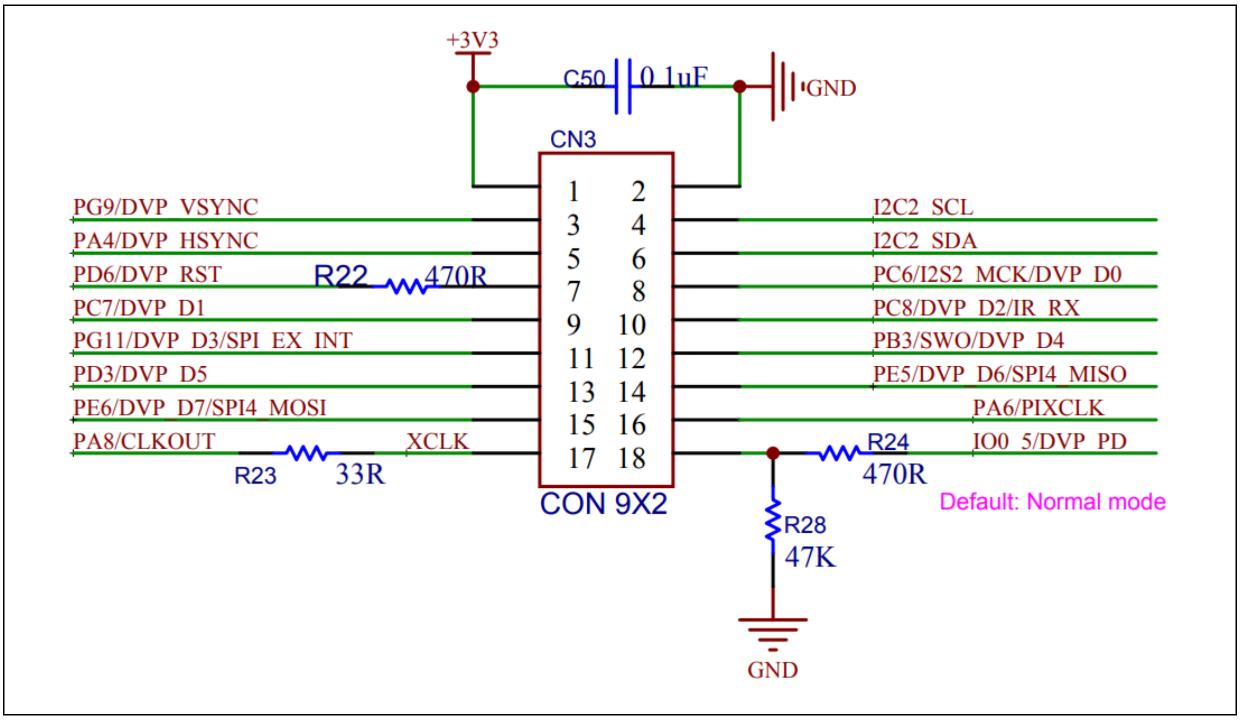
Hardware resources used in the application example are TFT LCD and OV2640 camera module. Table 30 lists the corresponding pins:

Table 30. Hardware resources

No.	PIN Name	Peripheral function	Description
1	PC6	DVP_D0	-
2	PC7	DVP_D1	-
3	PC8	DVP_D2	-
4	PG11	DVP_D3	-
5	PB3	DVP_D4	-
6	PD3	DVP_D5	-
7	PE5	DVP_D6	-
8	PE6	DVP_D7	-
9	PA6	DVP_VSYNC	-
10	PA4	DVP_HSYNC	-
11	PH2	I2C2_SCL	-
12	PH3	I2C2_SDA	-
13	PD6	DVP_RST	Reset camera

Figure 66 shows the schematic diagram of camera circuit.

Figure 66. Schematic diagram of camera



4.23.4 Software design

1) Camera test

- Initialize TFT LCD
- Initialize camera
- Show image on LCD

2) Code

- main function

```
int main(void)
{
    /* Initialize system clock */
    system_clock_config();

    /* Initialize interrupt priority group */
    nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

    /* Initialize delay function */
    delay_init();

    /* Initialize LCD */
    lcd_init(LCD_DISPLAY_VERTICAL);

    /* Show information*/
    lcd_string_show(10, 20, 200, 24, 24, (uint8_t *)"DVP Test");

    delay_ms(500);
```

```
    /* Initialize ov2640 */
    while(ov2640_init() != SUCCESS)
    {
      lcd_string_show(10, 140, 200, 24, 24, (uint8_t *)"ov2640 init err");

      delay_ms(400);
    }

    /* Show information */
    lcd_string_show(10, 60, 300, 24, 24, (uint8_t *)"ov2640 init ok");

    lcd_string_show(10, 100, 300, 24, 24, (uint8_t *)"dvp initializing...");

    /* Start image capture */
    ov2640_capture();

    while(1)
    {

    }
}
```

■ error_status ov2640_init(void)

```
/**
  * @brief  initialize ov2640
  * @param  none
  * @retval error_status: SUCCESS, ERROR
  */
error_status ov2640_init(void)
```

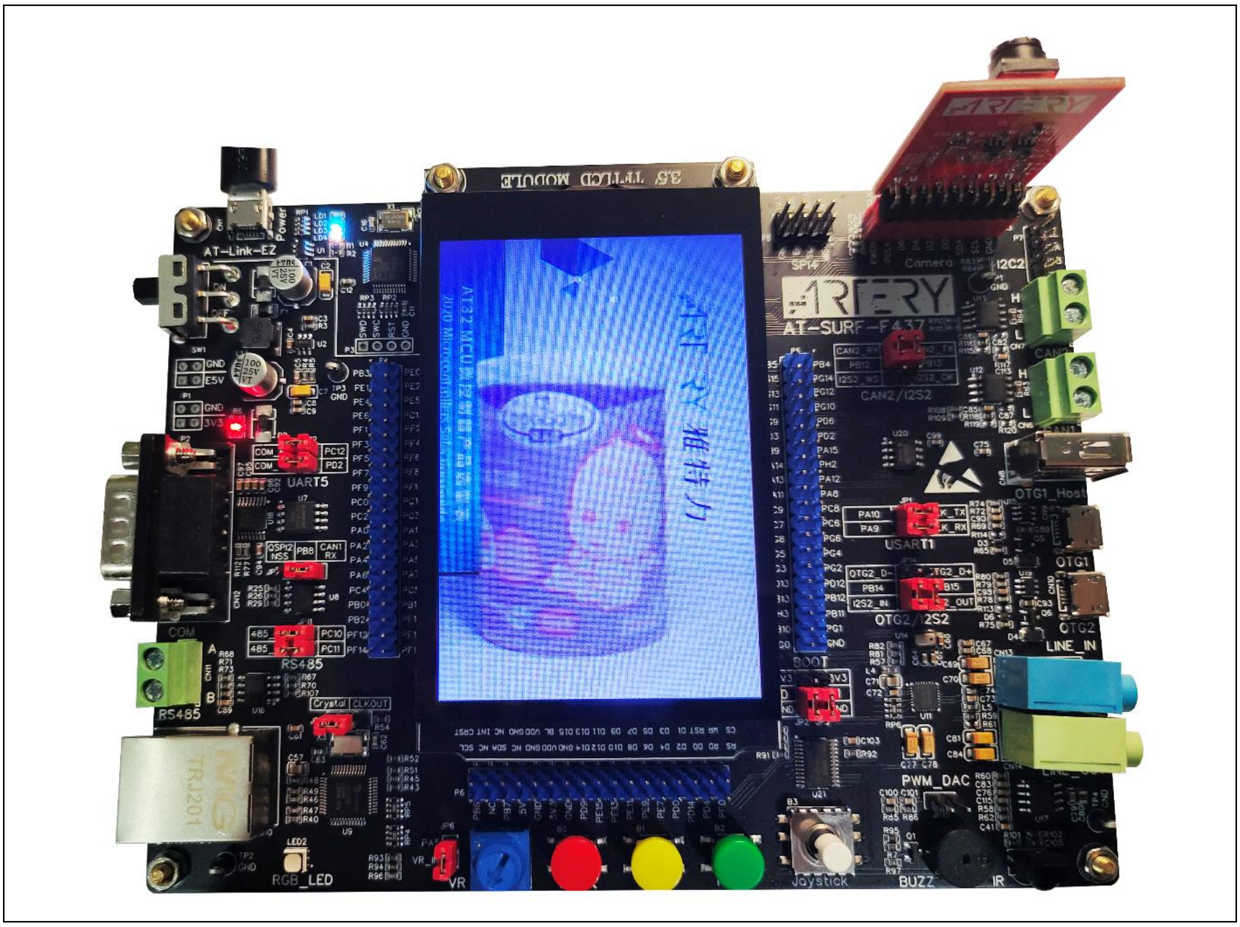
■ void ov2640_capture(void)

```
/**
  * @brief  start image capture and display on lcd
  * @param  none
  * @retval none
  */
void ov2640_capture(void)
```

4.23.5 Download and verify

- After the initialization of the camera, show test result on LCD.

Figure 67. Test result



4.24 Example 24: Network communication application

4.24.1 Introduction

The Ethernet peripheral enables the AT32F437 to transmit and receive data (10/100Mbps) over Ethernet in compliance with the IEEE 802.3-2002 standard. It supports two industry standard interfaces to the external physical layer (PHY): the default media independent interface (MII) defined in the IEEE 802.3 specifications and the reduced media independent interface (RMII).

The SUFR board has a PHY chip, part number DM9162, RMII interface. This example demonstrates the implementation of TCP Server function using LWIP TCP/IP protocol stack.

LWIP is a lightweight TCP/IP stack and operational without any operating system. The focus of the LWIP implementation is to reduce RAM usage while still having full scale TCP. This making LWIP suitable for use in embedded systems with tenths of KB of free RAM and room for around 40 KB of code ROM.

4.24.2 Resource requirements

- Hardware resources

  AT-SURF-F437 Board

- Software resources

  AT32F435_437_Firmware_Library_V2.x.x\project\at_sufr_f437\examples\tcp_server

4.24.3 Hardware design

Hardware resources used in the example are TFT LCD, PCA9555 IO extension connector and DM9162 chip. Table 31 to 32 list the corresponding pins:

Table 31. Hardware resources

No.	PIN Name	Peripheral function	Description
1	PA1	RMII_REF_CLK	-
2	PC1	EMAC_MDC	-
3	PA2	EMAC_MDIO	-
4	PG13	RMII_TXD0	-
5	PG14	RMII_TXD1	-
6	PB11	RMII_TX_EN	-
7	PC4	RMII_RXD0	-
8	PC5	RMII_RXD1	-
9	PA7	RMII_CRS_DV	-

Table 32. PCA9555

No.	PIN Name	Pin function	Description
1	IO0_1	EMAC_PDN	PHY chip power-on control

Figure 68 shows the schematic diagram of PHY.

Figure 68. Schematic diagram of PHY

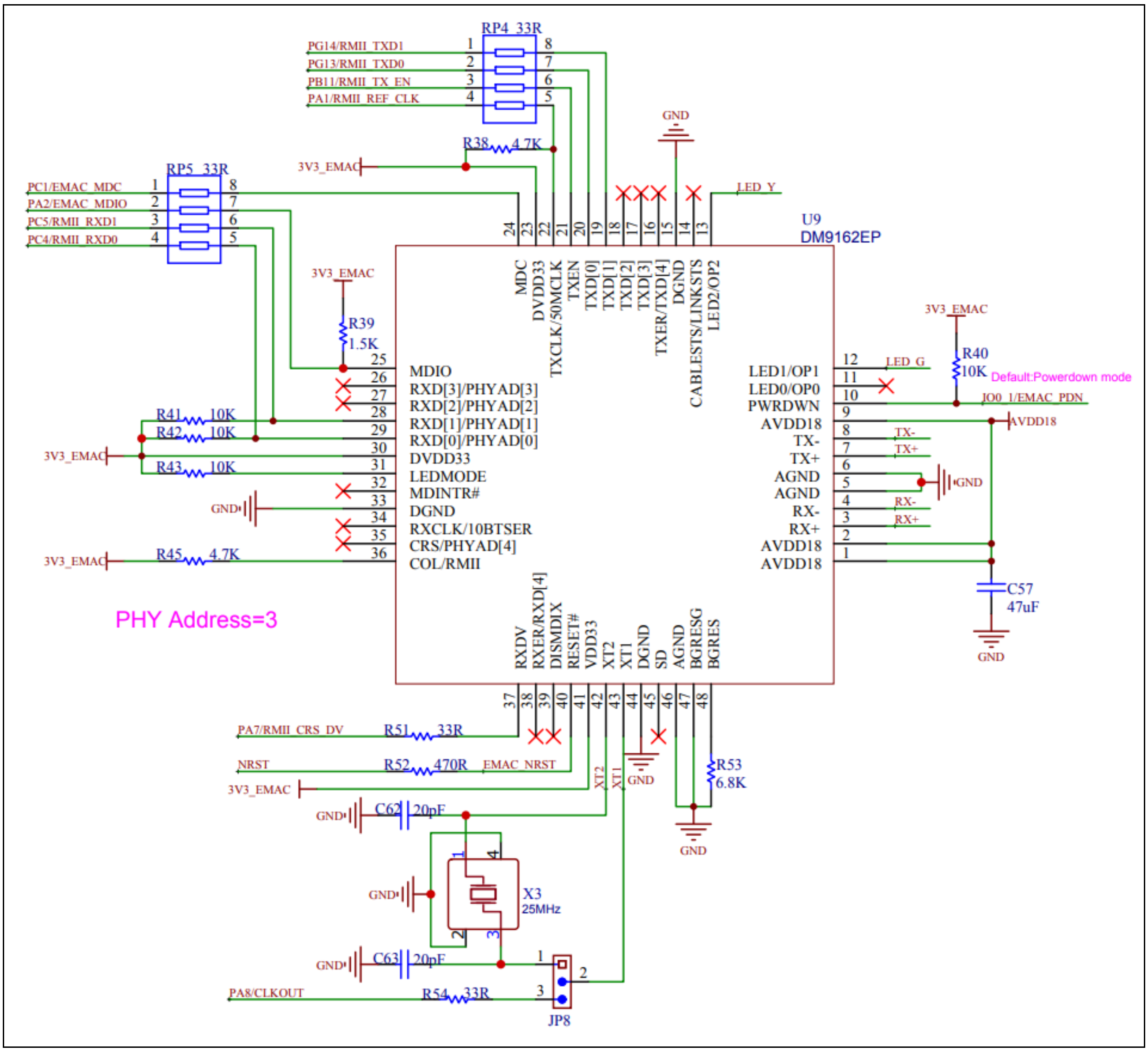


Figure 69. Schematic diagram of RJ45

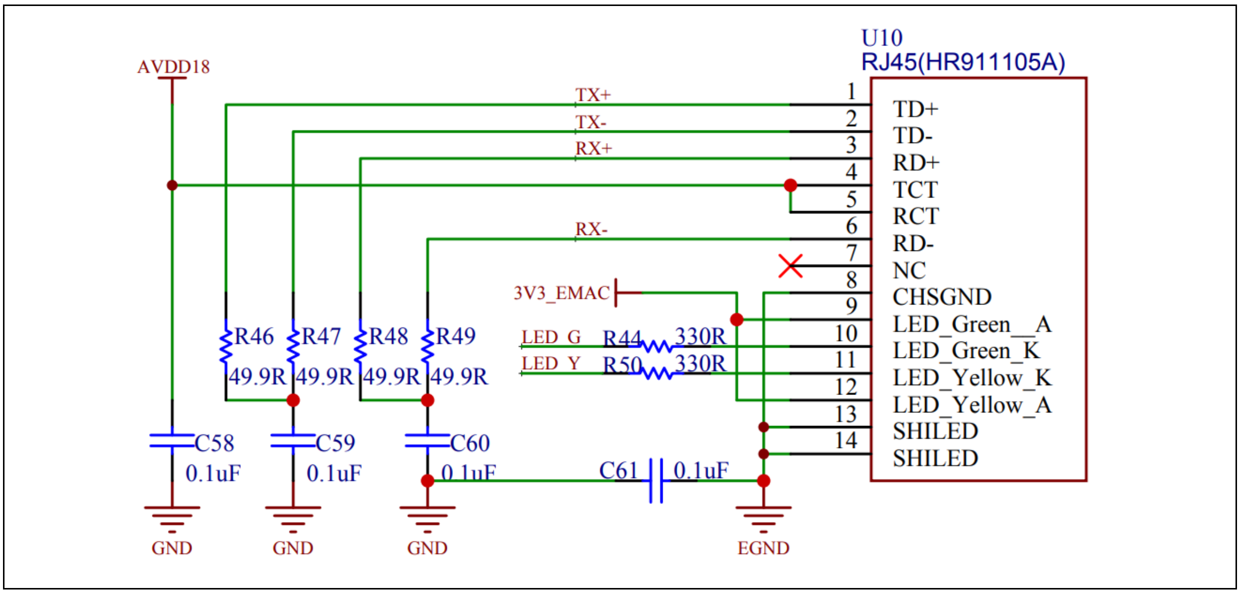
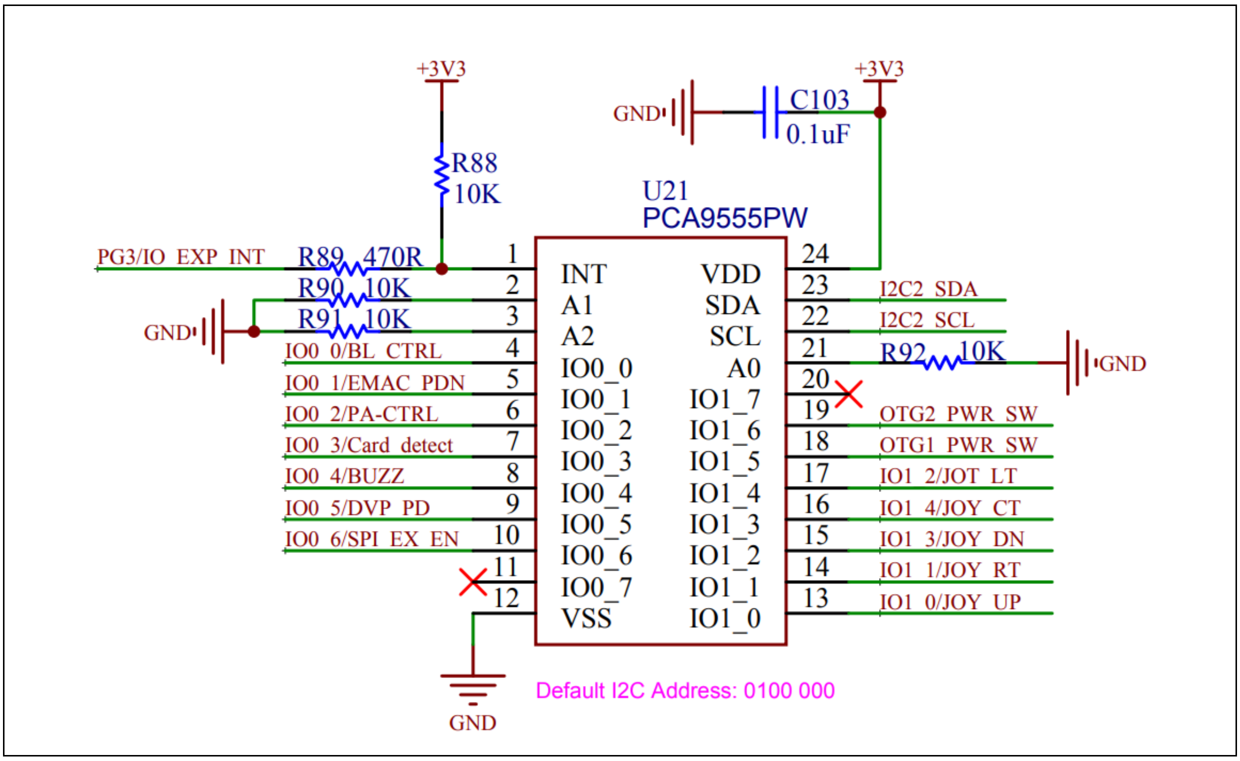


Figure 70. Schematic diagram of PCA9555



4.24.4 Software design

1) TCP Server test

- Initialize TFT LCD
- Initialize TCP Server
- Wait for client connection
- After client connection, send “Hello” to client
- After data are received from client, they will be shown on LCD

2) Code

- main function

```
int main(void)
{
   /* Initialize system clock */
   system_clock_config();

   /* Initialize interrupt priority group */
   nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

   /* Initialize delay function*/
   delay_init();

   /* Initialize LCD */
   lcd_init(LCD_DISPLAY_VERTICAL);

   /* Show information*/
   lcd_string_show(10, 20, 200, 24, 24, (uint8_t *)"TCP Server Test");
```

```
/* Initialize pca9555 IO extension connector */
pca9555_init(PCA_I2C_CLKCTRL_400K);

/* Initialize emac */
if(emac_system_init() == SUCCESS)
{
  lcd_string_show(10, 60, 200, 24, 24, (uint8_t *)"emac init ok");
}
else
{
  lcd_string_show(10, 60, 200, 24, 24, (uint8_t *)"emac init fail");
  while(1);
}

/* Initialize tcpip */
tcpip_stack_init();

/* Initialize tcp server */
tcp_server_init();

/* Show ip */
lcd_string_show(10, 90, 300, 24, 24,  (uint8_t *)"ip  :    .    .    .    ");
lcd_num_show(82,  90, 200, 24, 24, local_ip[0], 3);
lcd_num_show(130, 90, 200, 24, 24, local_ip[1], 3);
lcd_num_show(178, 90, 200, 24, 24, local_ip[2], 3);
lcd_num_show(226, 90, 200, 24, 24, local_ip[3], 3);

/* Show gateway */
lcd_string_show(10, 120, 300, 24, 24, (uint8_t *)"gw  :    .    .    .    ");
lcd_num_show(82,  120, 200, 24, 24, local_gw[0], 3);
lcd_num_show(130, 120, 200, 24, 24, local_gw[1], 3);
lcd_num_show(178, 120, 200, 24, 24, local_gw[2], 3);
lcd_num_show(226, 120, 200, 24, 24, local_gw[3], 3);

/* Show mask */
lcd_string_show(10, 150, 300, 24, 24, (uint8_t *)"mask:    .    .    .    ");
lcd_num_show(82,  150, 200, 24, 24, local_mask[0], 3);
lcd_num_show(130, 150, 200, 24, 24, local_mask[1], 3);
lcd_num_show(178, 150, 200, 24, 24, local_mask[2], 3);
lcd_num_show(226, 150, 200, 24, 24, local_mask[3], 3);

/* Show server port */
lcd_string_show(10, 180, 300, 24, 24, (uint8_t *)"port:     ");
lcd_num_show(82,  180, 200, 24, 24, TCP_LOCAL_PORT, 1);
```

```
    while(1)
    {
    }
}
```

- error_status emac_system_init(void)

```
/**
  * @brief  enable emac clock and gpio clock
  * @param  none
  * @retval success or error
  */
error_status emac_system_init(void)
```

- void tcpip_stack_init(void)

```
/**
  * @brief  initializes the lwip stack
  * @param  none
  * @retval none
  */
void tcpip_stack_init(void)
```

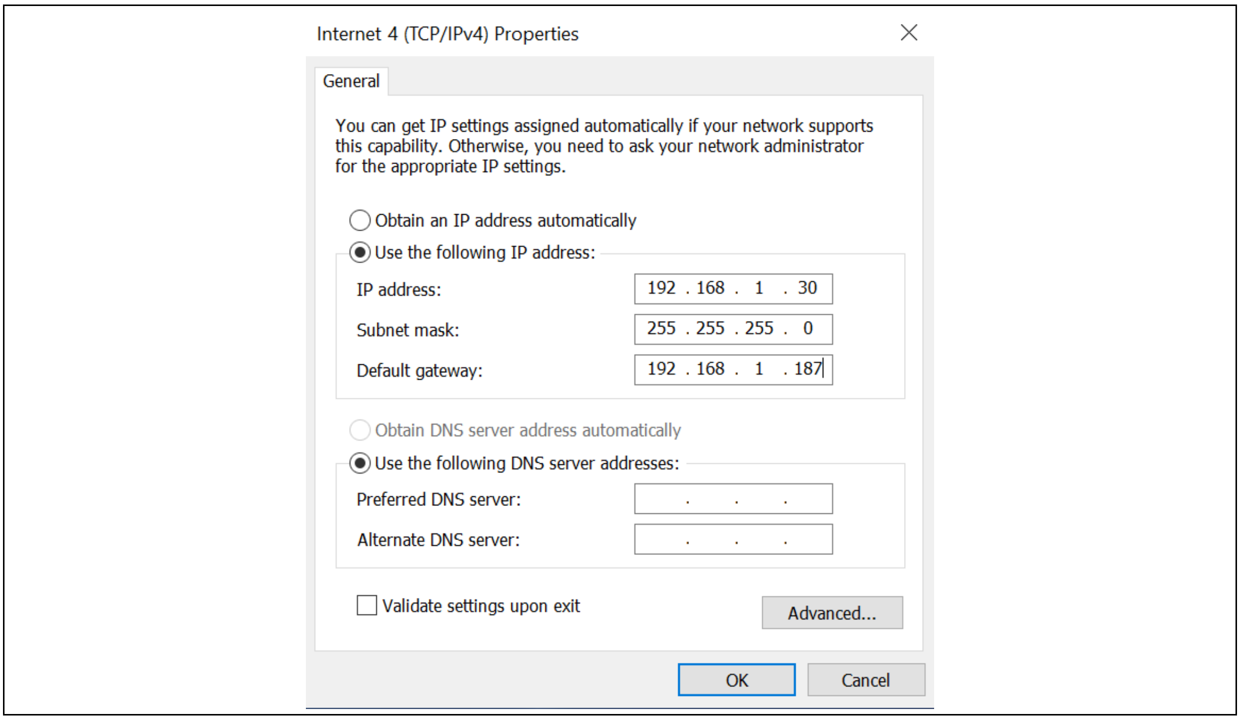
- void tcp_server_init(void)

```
/**
  * @brief  initialize tcp server
  * @param  none
  * @retval none
  */
void tcp_server_init(void)
```

4.24.5 Download and verify

- Connect SUFR to a PC via an internet cable.
- Configure computer internet.

Figure 71. Computer internet configuration



- Power supply the SURF board, initialize TCP Server, Server IP is 192.168.1.37, 1030 port.
- The computer side is connected to TCP Server via “Net Assistant”. After successful connection, the SUFR board sends “Hello” message.
- The computer side sends data to the SUFR board via “Net Assistant”. The received data will be shown on LCD before being sending back to the computer side.

Figure 72. PC side test result

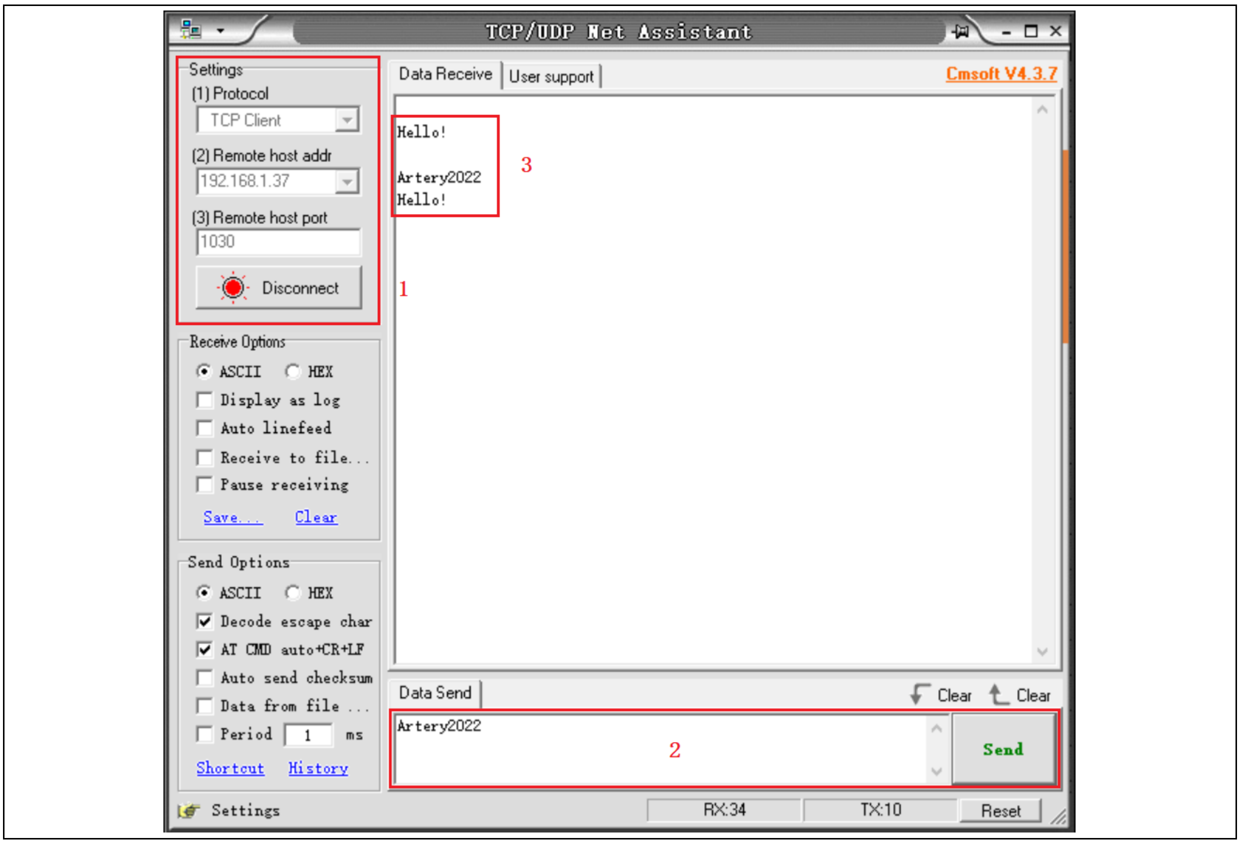
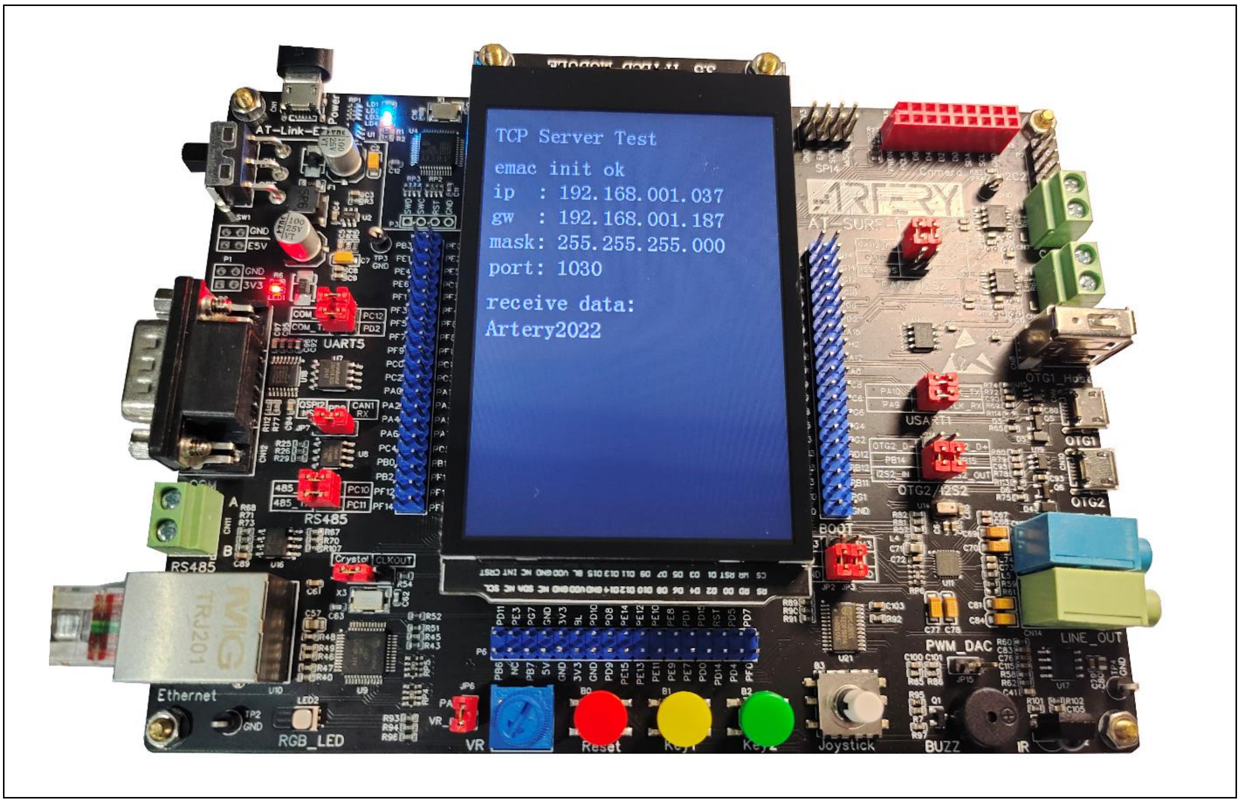


Figure 73. SUFR board side test result



5 Revision history

Table 33. Document revision history

Date	Revision	Changes
2022.3.11	2.0.0	Initial release
2022.4.26	2.0.1	Added *4.22 Example 22: Audio player application*
2022.6.14	2.0.2	Modified the LCD pin descriptions of *4.14 Example 14: TFT LCD application*
2022.9.22	2.0.3	Changed some figures in Chinese to English ones.