Multi-Functional Clock

Contents

1 Basic functions and user interface **1**
- 1.1 Basic functions . . . 1
- 1.2 LCD UI& Operating mode . . . 2
- 1.3 Key Function: . . . 3

2 System circuit design **3**

3 System code design **4**
- 3.1 API Function & File system . . . 4
- 3.2 Timer interrupt & Timer multiplexing . . . 5

4 LCD1602 driver module **6**
- 4.1 Basic write/read operation . . . 6

5 DS1302 driver module **7**
- 5.1 Basic write/read operation . . . 7
- 5.2 Burst read/write operation . . . 8
- 5.3 String converter function . . . 9

6 NEC IR Remote Control module **9**
- 6.1 NEC protocal . . . 9
- 6.2 NEC transmittion procedure . . . 10
- 6.3 NEC key encoding . . . 12

7 DS18B20 driver module **12**
- 7.1 Write and read operation of DS18B20 . . . 12
- 7.2 Flow chart of DS18B20 . . . 13

8 EEPROM (24C02) I2C bus module **14**
- 8.1 I2C bus protocal . . . 14
- 8.2 EEPROM page read and write . . . 14

9 Music player module **15**

10 UART module, timer1 multiplexing **16**
- 10.1 Baud rate(1200) calculation . . . 16
- 10.2 Timer1 multiplexing . . . 17

11 FSM of clock **18**

12 Main function & user serve function **19**

13 Appendix **20**

1 Basic functions and user interface

1.1 Basic functions

This electronic clock mainly has the following functions:

1. The **DS1302** clock chip is used to accurately display **time, date and day of the week**.
2. The real-time temperature is collected by the **DS18B20** temperature sensor chip, and it has a temperature alarm function.
3. **Buzzer sounds every hour**, and the buzzer can be turned off in real time by the control of the remote controller.
4. The time can be stored by **EEPROM**, and the power-down storage of the time can be controlled by the remote controller.
5. The time can be adjusted and the clock state can be changed by the **infrared remote controller**.
6. With multi-tasking function, each task can be performed simultaneously.
7. Can **play music**, "Only the mother is good in the world".
8. The **LCD 1602** is used for time display.
9. Communicate with the computer via the **UART** to transfer real-time time or time stored in the EEPROM.

1.2 LCD UI& Operating mode

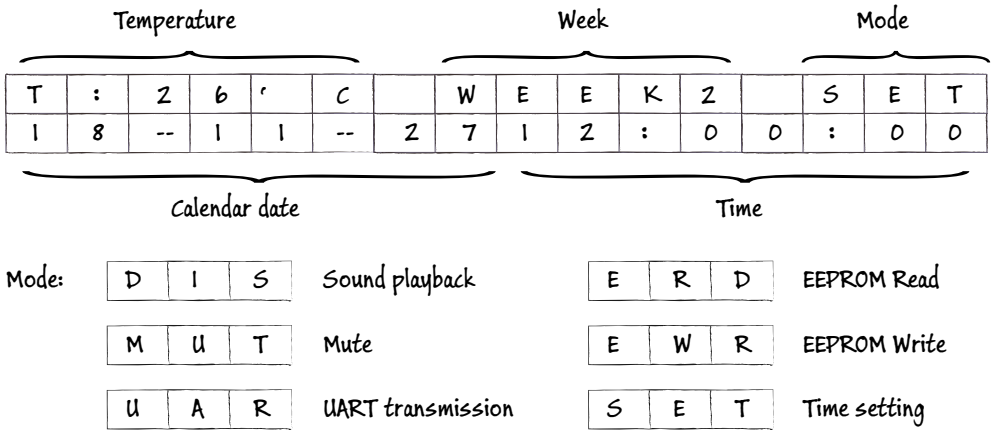

Figure 1: UI of this multi-task clever clock

The clock interface consists of five parts, the date display part, the time display part, the working mode display part, the day of the week display part, and the temperature display part. Next, we will focus on our clock working mode. The working mode of the clock can be seen in the upper right corner of the LCD in real time: **MUT, DIS, ERD, EWR, UAR, SET mode**. In the finite state machine of the clock, we have two states: **time setting state** and **normal walking state**. Among them (MUT, DIS, ERD, EWR, UAR) are completed in the normal walking state, only the SET mode is in the time setting state. We will introduce these working modes separately below.

DIS: In the DIS mode, if the temperature exceeds the predetermined threshold clock, it will alarm, or the clock will just go for an hour. Our buzzer can sound normally, and the clock can play music, which is the opposite of the MUT mode.

MUT: In the DIS mode, if the temperature exceeds the predetermined threshold clock, it will alarm, or the clock will just go for an hour. Our **buzzer can not sound.**

UAR: In UAR mode, our clock can communicate with the computer through the UART communication protocol, mainly transmitting real-time clock data or data stored in the EEPROM to the computer.

ERD/EWR: In ERD/EWR mode, we write the current clock data to the EEPROM or read the corresponding historical data from the EEPROM. Note that these two modes can only be turned on after the UAR mode is turned on, and the ERD/EWR mode should be completely turned off by turning off the UAR mode. This is because we can only transfer data to the computer after UAR mode is turned on. Since **ERD/EWR is time consuming**, remember to turn off both modes in time.

SET: We adjust the time, date, and week by entering the SET mode. Their adjustments can be made via the remote control.

1.3 Key Function:

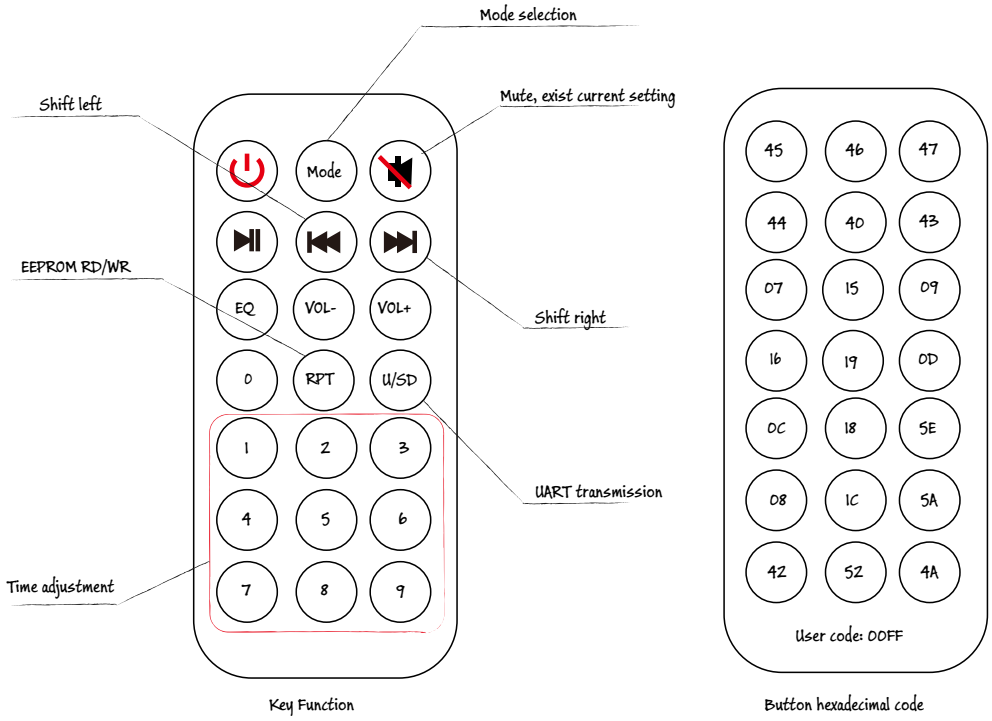

Figure 2: Key function.

Note that the Mode button here can convert the state of the clock. The default is the DIS state. When pressed once, it changes to the time setting state. The mute button has two functions: when the clock is in the DIS state, press the Mute button to turn off the music playback function. When the clock is in the SET state, press the Mute button to abandon the current setting and the clock remains at the original time. The RPT button also has two functions. The first time you press the ERD state, you can transfer data to the computer if you turn on the UAR at the same time. The second time you press the RPT button, the EWR state is entered. At this time, the clock continuously writes data to the EEPROM. . U/SD can turn on UAR or turn off UAR. Press UAR for the first time open the UAR, press it second time you turn off UAR. The (0,1,2,3,4,5,6,7,8,9) button is the clock setting button. When in the SET state, the corresponding button is pressed and the corresponding bit becomes the pressed value. The left or right shift key is responsible for adjusting the position of the cursor, they are only valid in the SET state.

2 System circuit design

We first introduce the composition of the circuit. This electronic clock consists of seven modules. They are: LCD display module, DS1302 clock module, DS18B20 temperature measurement module, passive buzzer sound playback module, EEPROM 24C02C power-down storage module, MCU control module, infrared wireless communication module.The schematic of the circuit is as follows.

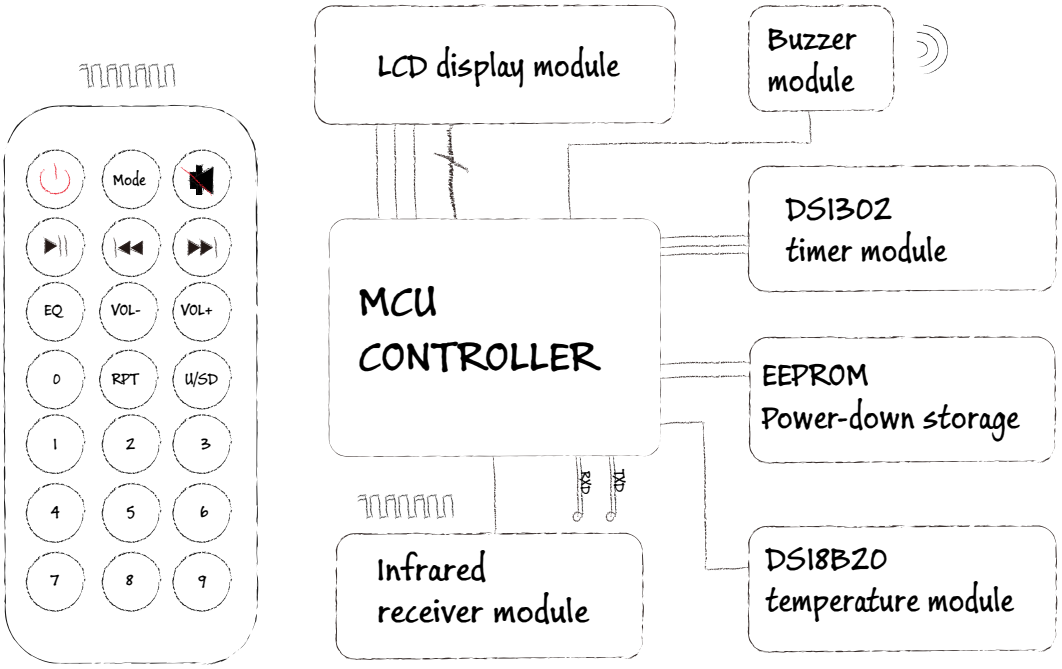

Figure 3: System circuit block

3 System code design

3.1 API Function & File system

Due to the many functions of this program and many low-level driver modules, this project adopts modular programming. The program consists mainly of five parts and a lot of header files. The five parts are: DS1302 driver module, DS18B20 driver module, EEPROM driver module (I2C module), LCD1602 driver module, and the final clock function module. We mainly use function calls and finite state machine programming. The first four modules are mainly responsible for the hardware driver, generating the timing required by the corresponding chip. The last module is mainly responsible for the interaction with the user, the clock state conversion, the infrared button Detection, LCD display mode.

Here we first introduce the DS1302 driver module, DS18B20 driver module, EEPROM driver module (I2C module), LCD1602 driver module. Here is a brief introduction to the API functions provided by these four modules for the upper CLOCK.c file. A more detailed discussion will be introduced next.

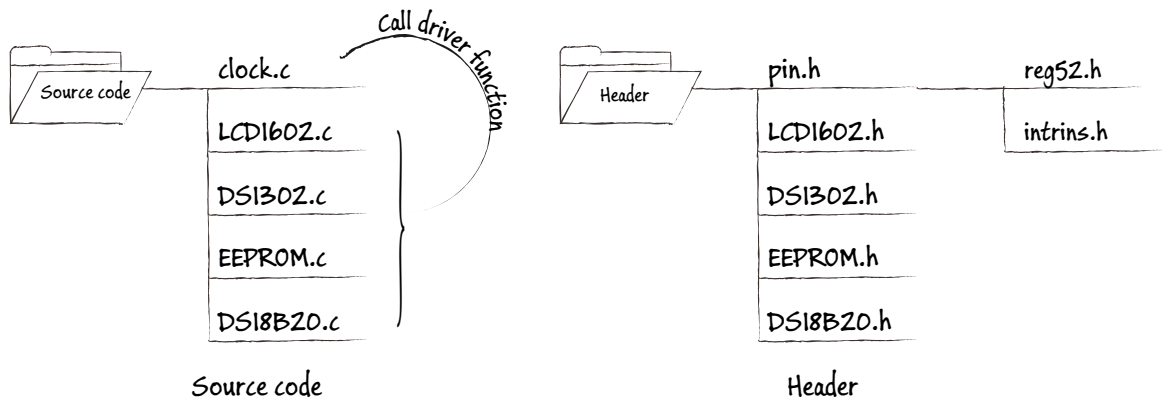

Figure 4: File relation in project

Below are all the functions used in the program, which are listed in the table below. On the left are the basic driver functions, the right interrupt service function and the user key service function, and the state machine conversion function.

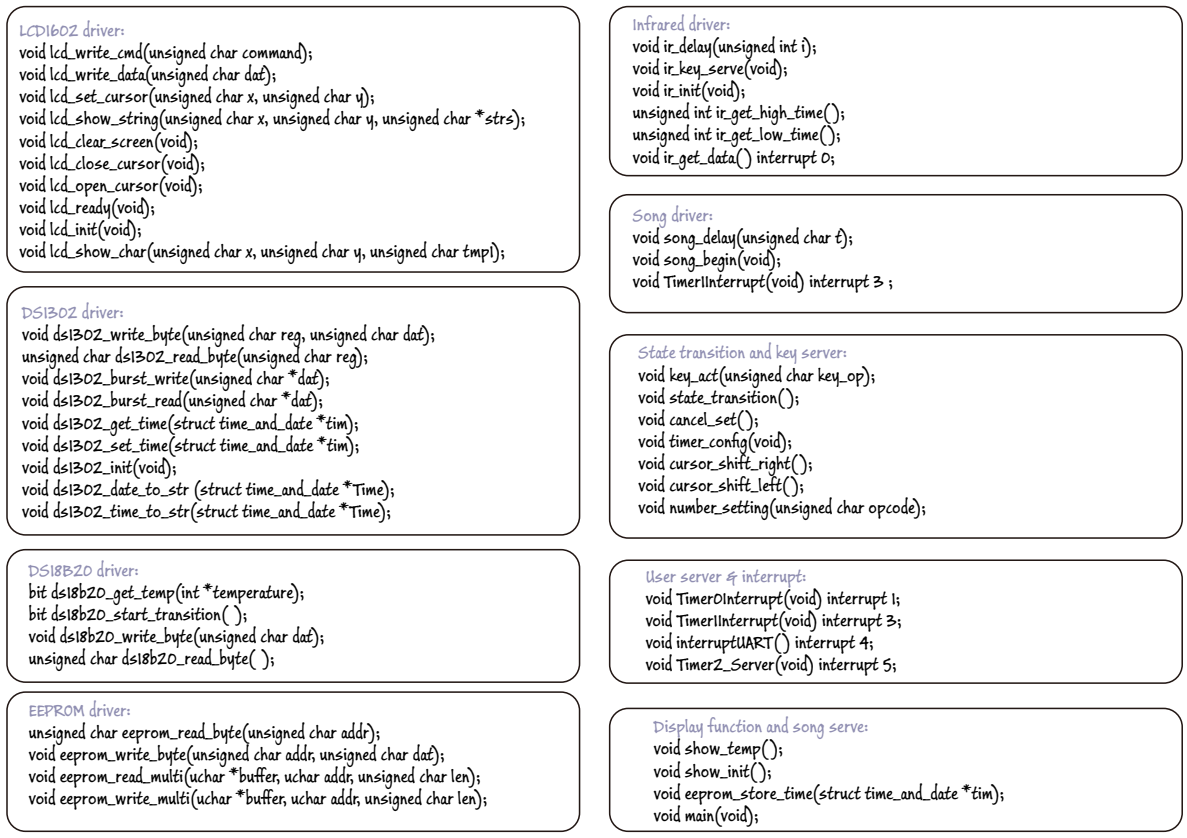

Figure 5: Block function. Driver API

3.2 Timer interrupt & Timer multiplexing

Below we focus on the timer interrupt. Since our program requires a total of 4 timers, there are only three timers in the 52 MCU, so we have to share the timer. Careful study of the clock application scenario, when we send data to the computer, prove that we are already paying attention to the clock time, we no longer need the alarm clock function, so here we choose UART serial port module and singing module to share the timer one,

Because Our infrared button detection module also uses a timer to detect the length of time the signal is received. Therefore, the infrared module also uses a timer. We use timer 0 here. We have not multiplexed timer 0 here. This is because the button detection should always be used, because our clock should respond to the user's needs in time, in order to bring a good user experience. So we ended up only timing the timer 0 function, and it was not multiplexed.

Finally, we use Timer 2 to run our button detection, time update, temperature update program, and Timer 2 interrupts every 500ms, which not only ensures timely response to user input, but also ensures that when playing music The music has been interrupted.

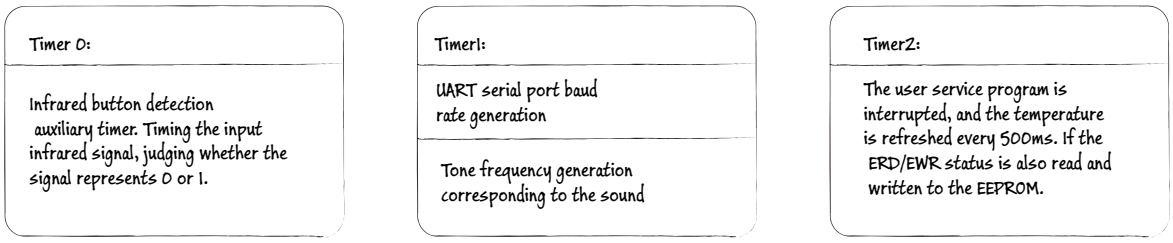

Figure 6: Timer multiplexing and timer interrupt

4 LCD1602 driver module

4.1 Basic write/read operation

The below image shows the timing diagram for sending the data to the LCD. As shown in the timing diagram the data is written after sending the RS and RW signals. It is still ok to send the data before these signals. The only important thing is the data should be available on the databus before generating the High-to-Low pulse on EN pin.

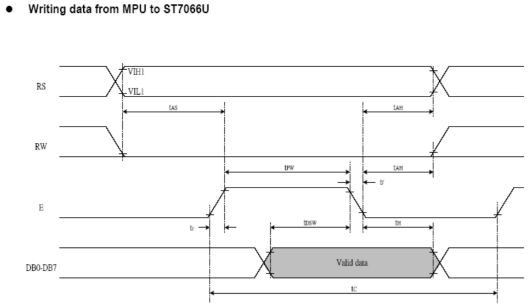

Figure 7: Time diagram of writing data

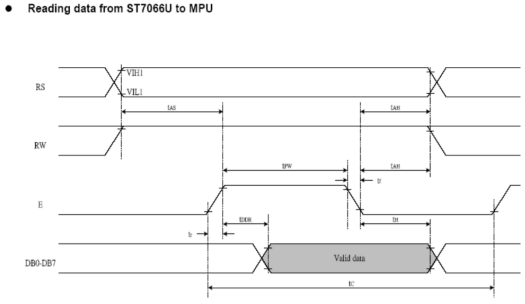

Figure 8: Time diagram of reading data

Steps for Sending Command:

1. Send the I/P command to LCD.
2. Select the Control Register by making RS low.
3. Select Write operation making RW low.
4. Send a High-to-Low pulse on Enable PIN with some delay_us.

Steps for Sending Data:

1. Send the character to LCD.
2. Select the Data Register by making RS high.
3. Select Write operation making RW low.

4. Send a High-to-Low pulse on Enable PIN with some delay_us.

The timings are similar as above only change is that RS is made high for selecting Data register.In the LCD driver file we provide the following main functions, lcd_set_cursor sets the position of the cursor, lcd_show_string shows the string in the corresponding coordinates.The specific function name and corresponding call method are listed in the above table, and will not be repeated here. And each function name is also self-explanatory.

Table 1: Pin of LCD1602

Pin Number	Symbol	Pin Function
1	VSS	Ground
2	VCC	+5v
3	VEE	Contrast adjustment (VO)
4	RS	Register Select. 0:Command, 1: Data
5	R/W	Read/Write, R/W=0: Write & R/W=1: Read
6	EN	Enable. Falling edge triggered
7	D0	Data Bit 0
8	D1	Data Bit 1
9	D2	Data Bit 2
10	D3	Data Bit 3
11	D4	Data Bit 4
12	D5	Data Bit 5
13	D6	Data Bit 6
14	D7	Data Bit 7/Busy Flag
15	A/LED+	Back-light Anode(+)
16	K/LED-	Back-Light Cathode(-)

5 DS1302 driver module

5.1 Basic write/read operation

DS1302 is a real-time clock chip, we can use the microcontroller to write time or read the current time data. The DS1302 is a real-time clock chip that provides information on seconds, minutes, hours, dates, months, years, and more.There is also the ability to automatically adjust the software, you can configure AM / PM to decide whether to use the 24-hour format or the 12-hour format. It has 31 bytes of data storage RAM.

Pins of DS1302: 1 pin VCC2 is the main power positive pin, 2 pin X1 and 3 pin X2 are crystal oscillator input and output pins, 4 pin GND is negative, 5 pin CE is enable pin, connect to MCU IO Port, 6-pin I/O is the data transmission pin, connected to the IO port of the microcontroller. The 7-pin SCLK is the communication clock pin. It is connected to the IO port of the MCU. The 8-pin VCC1 is the standby power pin.

One instruction of DS1302 has 8 bytes in total, and the 7th bit (ie, the highest bit) is fixed to 1. If this bit is 0, it is invalid. The sixth bit is to choose RAM or CLOCK. We mainly talk about the use of CLOCK clock. Its RAM function is not used, so if you choose CLOCK function, the 6th bit is 0. If you want to use RAM, the 6th bit is 1. From the 5th to the 1st, the 5-bit address of the register is determined, and the 0th bit is the read/write bit. If it is to be written, this bit is 0. If it is to be read, this bit is 1. The intuitive byte assignment of the instruction byte is shown in the figure.

7	6	5	4	3	2	1	0
1	RAM / $\overline{CK}$	A4	A3	A2	A1	A0	RD / $\overline{WR}$

Figure 9: Instruction byte of DS1302

In the timing of the DS1302, the microcontroller must pre-write a byte instruction indicating the address of the register to be written and the subsequent operation is a write operation, and then write a byte of data.

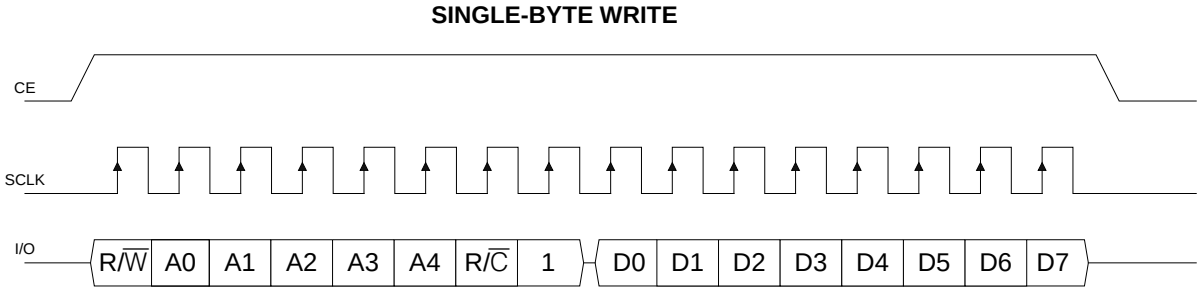

Figure 10: Writing operation of DS1302

The arrows on the timing diagram of the DS1302 are for the DS1302. Therefore, when reading, the first byte instruction is written first. When the rising edge is used, the DS1302 latches the data. On the falling edge, we use the microcontroller to send data. To the second word data, since our timing process is equivalent to CPOL=0/CPHA=0, the leading edge sends data, the trailing edge reads data, and the second byte is DS1302 falling edge output data, our microcontroller rises along Read, so the arrow appears on the falling edge from the DS1302 perspective.

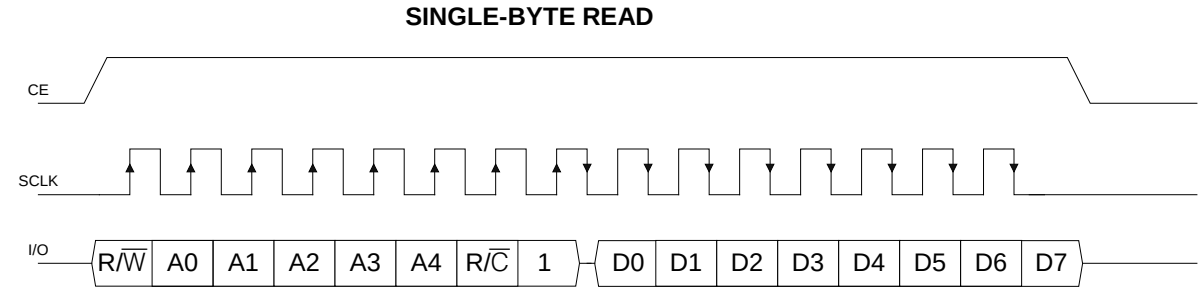

Figure 11: Reading operation of DS1302

5.2 Burst read/write operation

If our current time is 00:00:59, we will read the second first, the second is 59, and then read the minute, and just after reading the second to the time when the minute has not yet started, just time to enter. It becomes 00:01:00 At this time, the minute we read is 01, and a 00:01:59 will appear on the LCD. This time is obviously wrong. The probability of this problem is extremely small, but it is actually possible. So in order to prevent the data read by the microcontroller is wrong, we use Burst read and Burst write.

When we write the instruction to the DS1302, as long as we write 1 address of the 5-bit address to be written, the read operation uses 0xBF, and the write operation uses 0xBE. After such instruction is sent to DS1302, it will automatically recognize that it is burst mode. All 8 bytes are immediately latched

into the other 8 byte register buffer so that the clock continues to go and the data we read is read from another buffer.

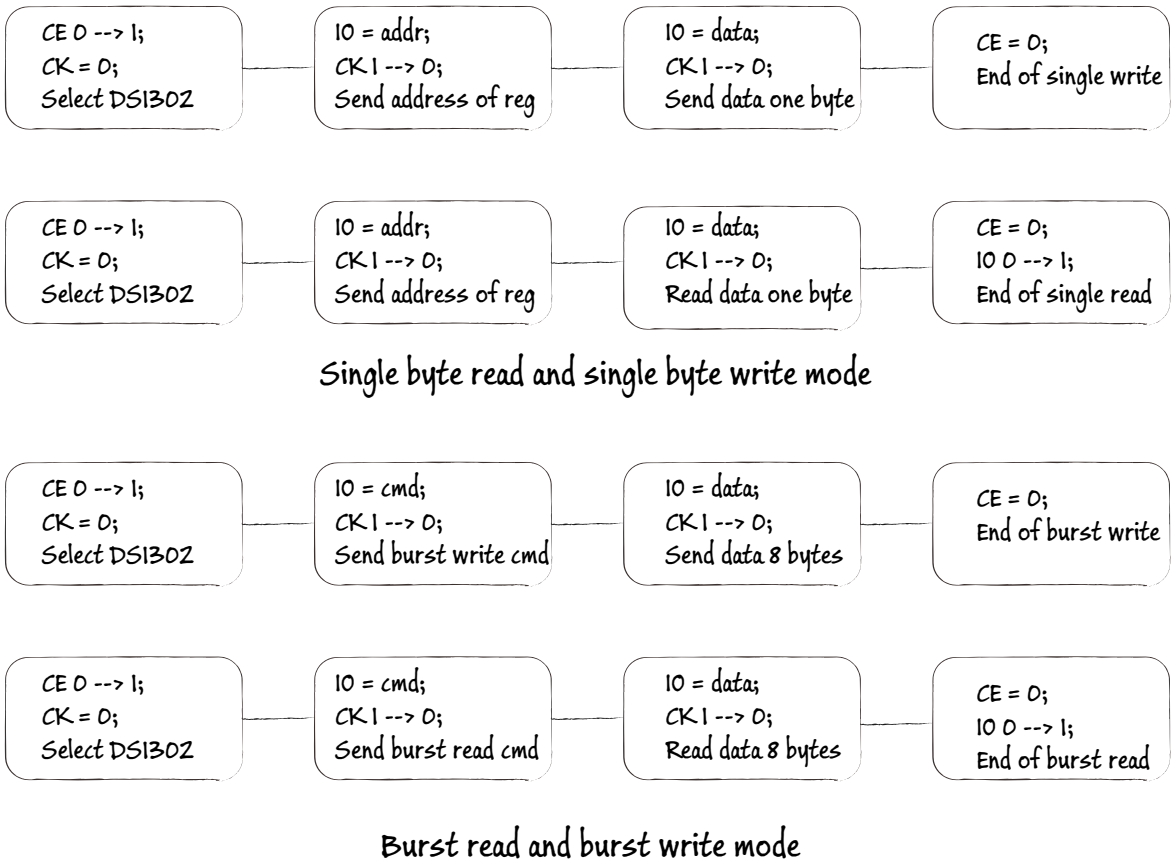

Figure 12: Flow chart of DS1302

5.3 String converter function

The following program converts the read data into a string that is output to the LCD.

```
1  void ds1302_time_to_str(struct time_and_date *Time)
2  {
3          Time->TimeString[0] = (Time->hour >> 4) + '0';
4          Time->TimeString[1] =(Time->hour & 0x0F) + '0';
5          Time->TimeString[2] = ':';
6          Time->TimeString[3] = (Time->minute >> 4) + '0';
7          Time->TimeString[4] = (Time->minute & 0x0F) + '0';
8          Time->TimeString[5] = ':';
9          Time->TimeString[6] = (Time->second>> 4) + '0';
10         Time->TimeString[7] = (Time->second & 0x0F) + '0';
11         Time->TimeString[8] = '\0';
12 }
```

6 NEC IR Remote Control module

6.1 NEC protocal

IR Remote Controllers and receivers follow standard protocols for sending and receiving the data. Some of the standard protocols are NEC , JVC , SIRC (Sony Infrared Remote Control) etc. We will be discussing only the NEC protocol.

NEC IR protocol encodes the keys using a 32bit frame format as shown below. Each bit is transmitted using the pulse distance as shown in the image.

Logical '0': A 562.5s pulse burst followed by a 562.5s space, with a total transmit time of 1.125ms

Logical '0': A 562.5s pulse burst followed by a 1.6875ms space, with a total transmit time of 2.25ms

NEC Frame Format			
Address	Complement of Address	Command	Complement of Command
LSB-MSB(0-7)	LSB-MSB(8-15)	LSB-MSB(16-23)	LSB-MSB(24-31)

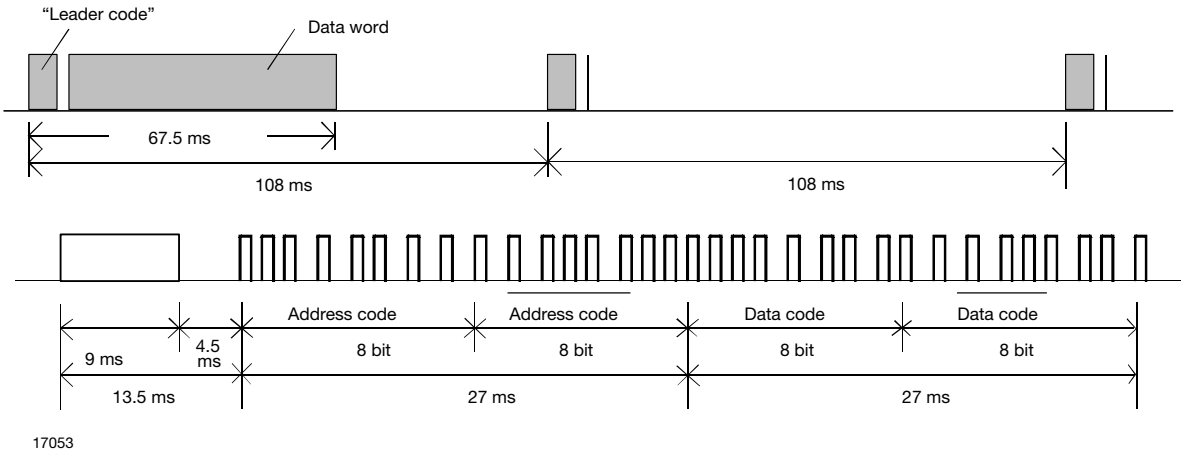

Figure 13: NEC protocal

When a key is pressed on the remote controller, the message transmitted consists of the following, in order: The four bytes of data bits are each sent least significant bit first. Below image illustrates the format of an NEC IR transmission frame, for an address of 00h (00000000b) and a command of ADh (10101101b). A total of 67.5ms is required to transmit a message frame. It needs 27ms to transmit the 16 bits of address (address + inverse) and the 16 bits of command (command + inverse).

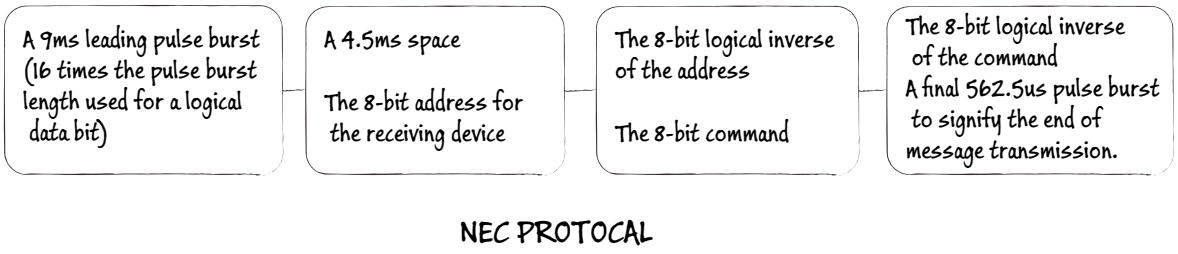

Figure 14: message transmittion order

6.2 NEC transmittion procedure

Since the input signal length is to be measured in the program, we use timer 0 to measure the signal length. Since we want to respond to the signal in real time, we use external interrupt 0 here and use edge jump.

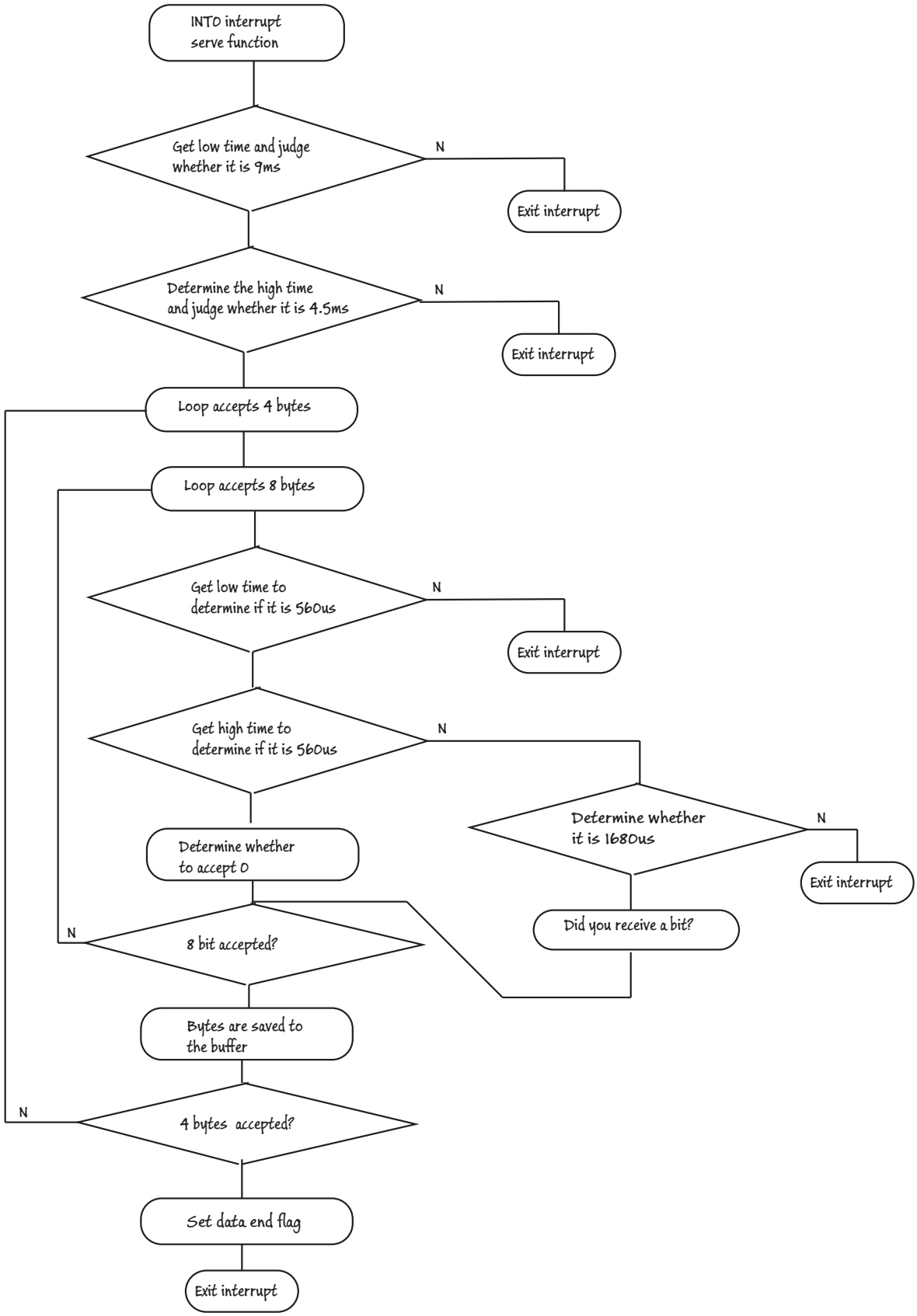

Figure 15: message transmittion order

6.3 NEC key encoding

The buttons of the wireless remote control have corresponding codes. After measuring the coding of each button, we encode it twice, making the program more readable, and the program can be made more in the final button adjustment phase. simple.

```
const uint8 code ir_map[][2] = {
        {0x45,'s'}, {0x46,'o'}, {0x47,'m'},              //shutdown   Mode
              Mute
        {0x44,'u'}, {0x40,'l'}, {0x43,'r'},              //unknown    left
              Right
        {0x07,'e'}, {0x15,'d'}, {0x09,'i'},              // EQ        Inc
              Dec
        {0x16,'0'}, {0x19,'x'}, {0x0D,'y'},              //'0'->'0'   RPT
              U/SD
        {0x0C,'1'}, {0x18,'2'}, {0x5E,'3'},              //'1'->'1'
            '2'->'2'    '3'->'3'
        {0x08,'4'}, {0x1C,'5'}, {0x5A,'6'},              //'4'->'4'
            '5'->'5'    '6'->'6'
        {0x42,'7'}, {0x52,'8'}, {0x4A,'9'},              //'7'->'7'
            '6'->'8'    '9'->'9'
};
```

7 DS18B20 driver module

7.1 Write and read operation of DS18B20

The DS18B20 is a temperature sensor from Maxim that allows the microcontroller to communicate with the DS18B20 via the 1-Wire protocol and ultimately read the temperature. The hardware interface of the 1-Wire bus is very simple. Just connect the data pin of the DS18B20 to an IO port of the microcontroller. The simplicity of the hardware, along with the complexity of the software timing. The timing of the 1-Wire bus is more complicated.

Initialization. Similar to the I2C addressing, the 1-Wire bus also needs to detect the presence of the DS18B20 on this bus. If there is a DS18B20 on this bus, the bus will return a low-level pulse according to the timing requirement. If it does not exist, it will not return a pulse, that is, the bus remains high, so it is customary to detect the presence of a pulse. There is a pulse detection process. First, the microcontroller should pull this pin low for about 480us to 960us. Our program lasts for 500us. Then, the MCU releases the bus, which is to give a high level. After the DS18B20 waits for about 15 to 60us, it will actively pull the pin down to about 60 to 240us, and then the DS18B20 will actively release the bus, so that the IO port will be pulled automatically by the pull-up resistor.

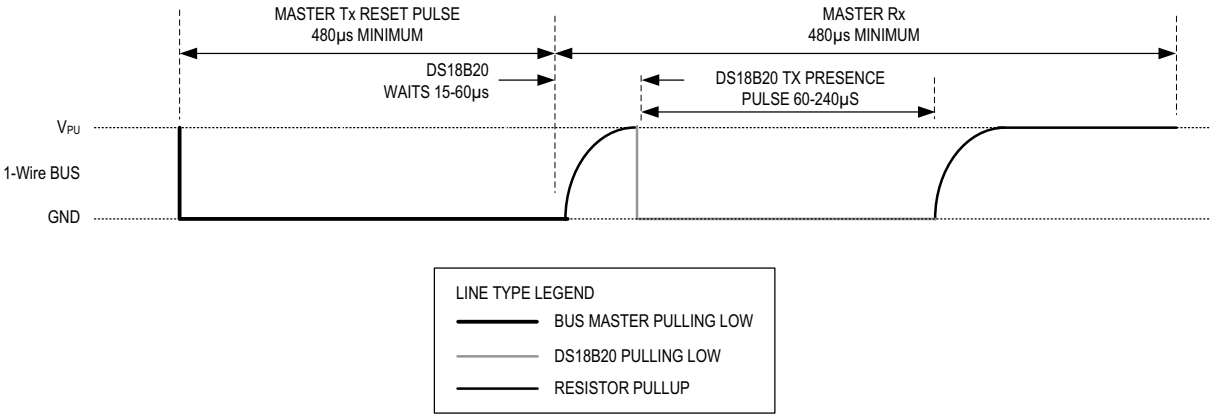

Figure 16: Initialization timing diagram of DS18B20

When writing 0 to the DS18B20, the microcontroller directly pulls the pin low for more than 60us and less than 120us.When writing a 1 to the DS18B20, the microcontroller first pulls this pin low, pulling the time low for more than 1us, and then immediately releasing the bus, that is, pulling the pin high, and the duration is also greater than 60us. Similar to writing 0, the DS18B20 will read this 1 between 15us and 60us.

When reading the DS18B20 data, our microcontroller must first pull this pin low for at least 1us, then release the pin and read it as soon as possible after the release. From pulling this pin low to reading the pin state, it cannot exceed 15us. As can be seen from the figure, the host sampling time, which is MASTER SAMPLES, must be completed within 15us.

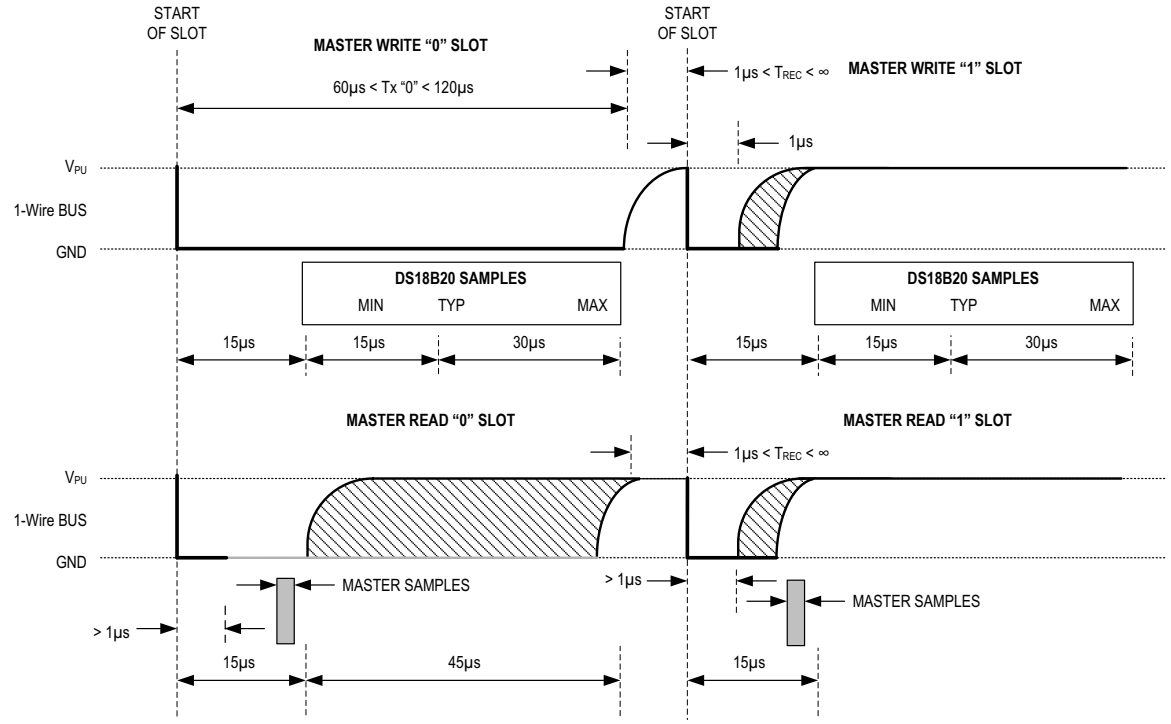

Figure 17: Reading and writing diagram of DS18B20

7.2 Flow chart of DS18B20

In the module of DS18B20, we provide the function of reading and writing to DS18B20, and provide the function of obtaining temperature. Note that in the obtained temperature function, our function requires passing in a pointer of type INT, and the function reads the data internally. The conversion returns to the function called.

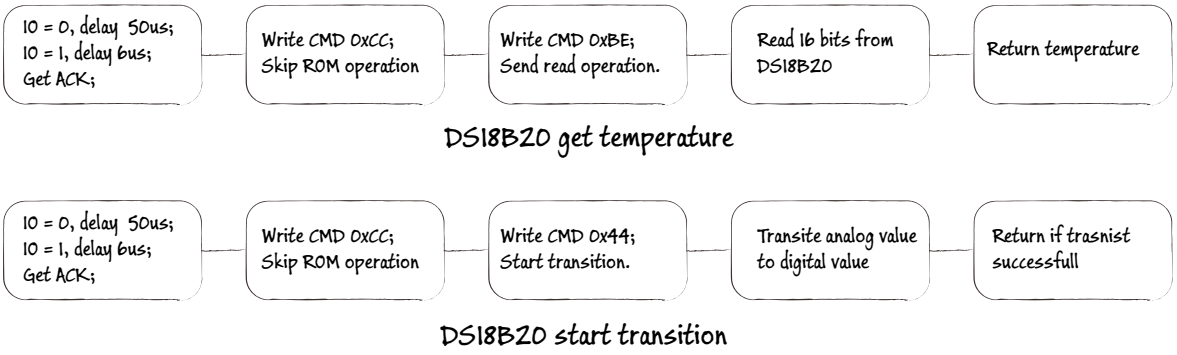

Figure 18: Flow chart of DS18B20

8 EEPROM (24C02) I2C bus module

8.1 I2C bus protocal

The I2C bus is a two-wire serial bus developed by PHILIPS, which is used to connect microprocessors and their peripheral chips.

Start signal: UART communication is a low level flag start bit from a continuous high level; the start signal of I2C communication is defined as SCL is high level, SDA is high level to low level. The change produces a falling edge indicating the start signal.

Data transfer: First, the UART is low and the high bit is after; the I2C communication is high and the low. Secondly, the UART communication data bit is a fixed length, and the baud rate is one of the bits. One bit of fixed time can be sent. I2C does not have a fixed baud rate, but there are timing requirements. When SCL is low, SDA is allowed to change. That is, the sender must first keep SCL low to change the data line SDA. Output one bit of the current data to be transmitted; and when SCL is high, SDA must not change, because at this time, the receiver has to read whether the current SDA level signal is 0 or 1, so it is guaranteed The stability of SDA, the change of each bit of data, is in the low position of SCL. The 8-bit data bit is followed by an acknowledge bit.

Stop signal: The stop bit of UART communication is a fixed high level signal; while the I2C communication stop signal is defined as SCL is high level, SDA changes from low level to high level to generate a rising edge, indicating the end signal.

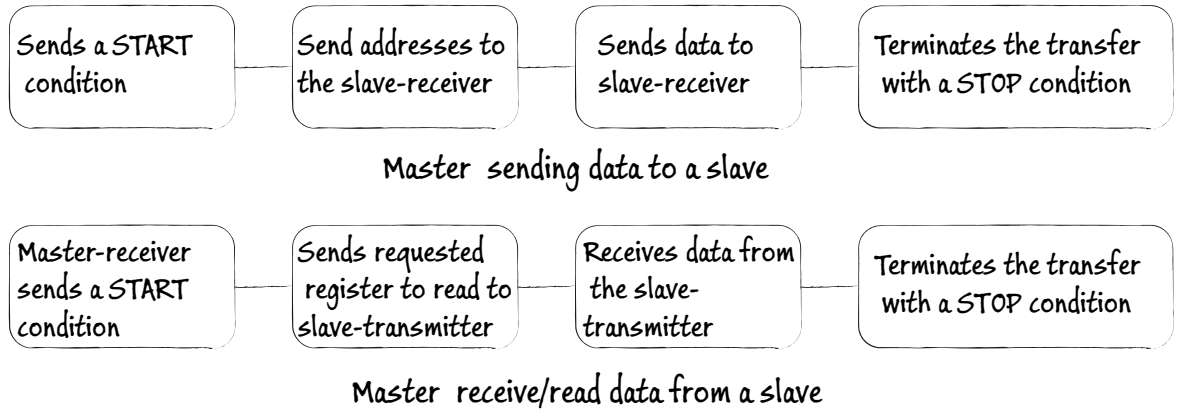

Figure 19: Flow chart of I2C bus

8.2 EEPROM page read and write

In some cases, we need to record some data, and they often need to be changed or updated. After power failure, the data can not be lost. For example, our household meter power, channel memory inside the TV, generally use EEPROM. The data is saved, and the feature is that it is not lost after power failure. The device used on our board is 24C02, which is a EEPROM with a capacity of 2Kbits, which is 256 bytes.

When we read the EEPROM, it was very simple. The EEPROM sent the data directly according to the timing we sent, but writing EEPROM was not that simple. After sending data to the EEPROM, it is saved in the EEPROM buffer. The EEPROM must move the data in the buffer to the "non-volatile" area to achieve the effect of power loss without loss. It takes a certain amount of time to write to the non-volatile area. Each device is not exactly the same. ATMEL's 24C02 has a write time of no more than 5ms. In the process of writing to the non-volatile area, the EEPROM will not respond to our access. Not only will we not receive our data, we will use the I2C standard addressing mode to address, and the EEPROM will not respond, just like There is no such device on this bus. After the data is written to the nonvolatile area, the EEPROM returns to normal again and can be read and written normally.

When writing multiple bytes of data to the EEPROM continuously, if you wait a few ms for each byte written, the overall write efficiency is too low. So **here we use page writes to improve the efficiency of the program.** After the page is allocated, if we continuously write a few bytes in the same page, the timing of the stop bit is finally sent. After the EEPROM detects this stop bit, it will write the data of this page to the non-volatile area at one time. If the data we write spans the page, after writing a page, we will send a stop bit, then Waiting and detecting the idle mode of the EEPROM, until the previous page of data is completely written to the non-volatile area, and then writing to the next page, which can greatly improve the data writing efficiency.

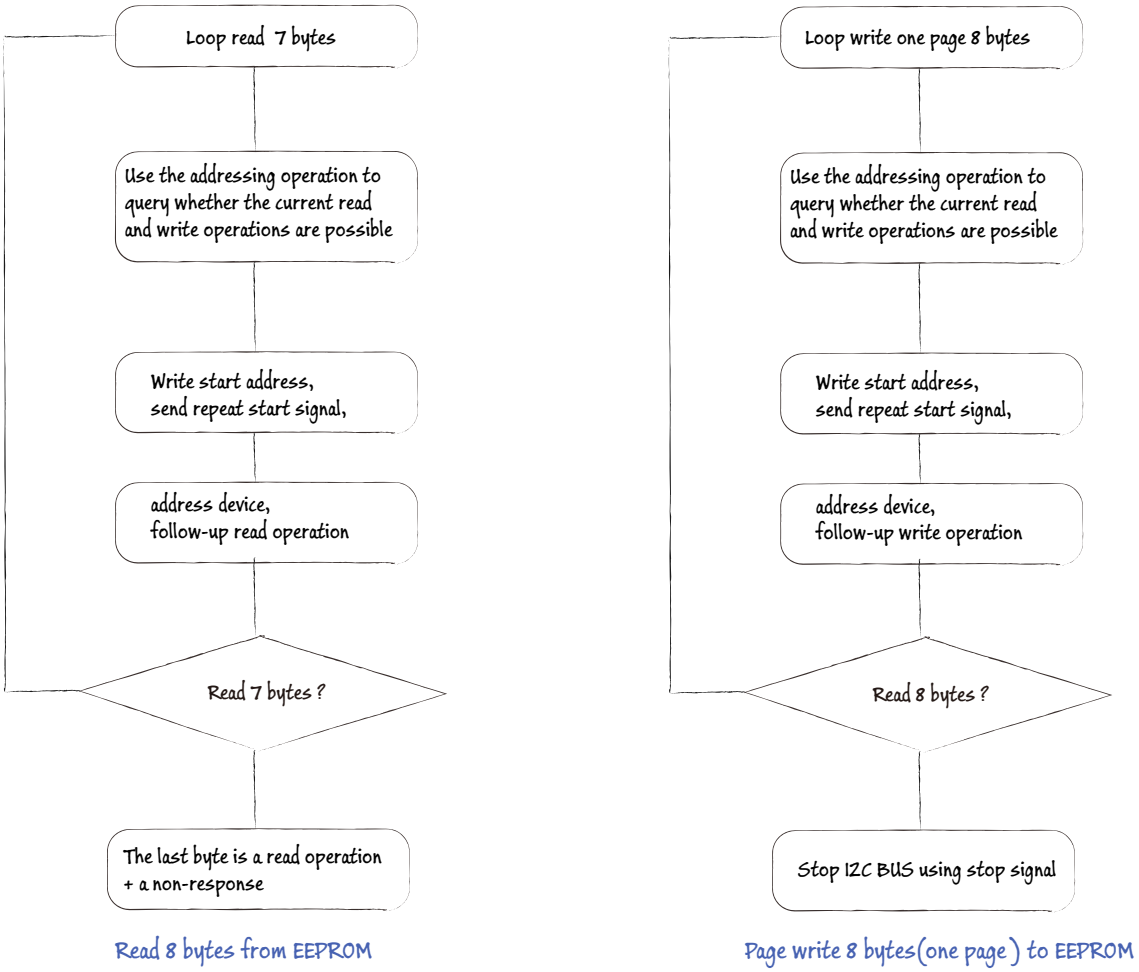

Figure 20: Page read and write operation of EEPROM

9 Music player module

Our music player uses Timer 1 to generate the frequencies needed for music. Our program can distinguish between treble, super-high, mid-range, and bass to make musical tonal changes. We change the pitch by setting the initial value of Timer 1. The initial values of the timer are shown below.

```
1 code unsigned char FH[] = {
    //Initila value for timer1 high eight bits
2       0xF2,0xF3,0xF5,0xF5,0xF6,0xF7,0xF8,          //Low
          frequency
3       0xF9,0xF9,0xFA,0xFA,0xFB,0xFB,0xFC,          //Medium
          freqency.
4       0xFC,0xFC,0xFD,0xFD,0xFD,0xFD,0xFE,          //High
          frequency.
```

```
5           0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFF                    //Super
                high freqency.
6   };
7
8   code unsigned char FL[] = {
        //Initila value for timer1 low eight bits
9           0x42, 0xC1, 0x17, 0xB6, 0xD0, 0xD1, 0xB6,       //Low frequency
10          0x21, 0xE1, 0x8C, 0xD8, 0x68, 0xE9, 0x5B,       //Medium
                freqency.
11          0x8F,0xEE, 0x44, 0x6B, 0xB4, 0xF4, 0x2D,        //High frequency
                .
12          0x47, 0x77, 0xA2, 0xB6, 0xDA, 0xFA, 0x16        //Super high
                freqency.
13  };
```

Music Encoding: A note has three numbers. The first is the first few, the middle is the first octave, and the latter is the duration (in half beats). 6, 2, 3 respectively represent: 6, midrange, 3 half beats; 5, 2, 1 respectively represent: 5, midrange, 1 half beat;

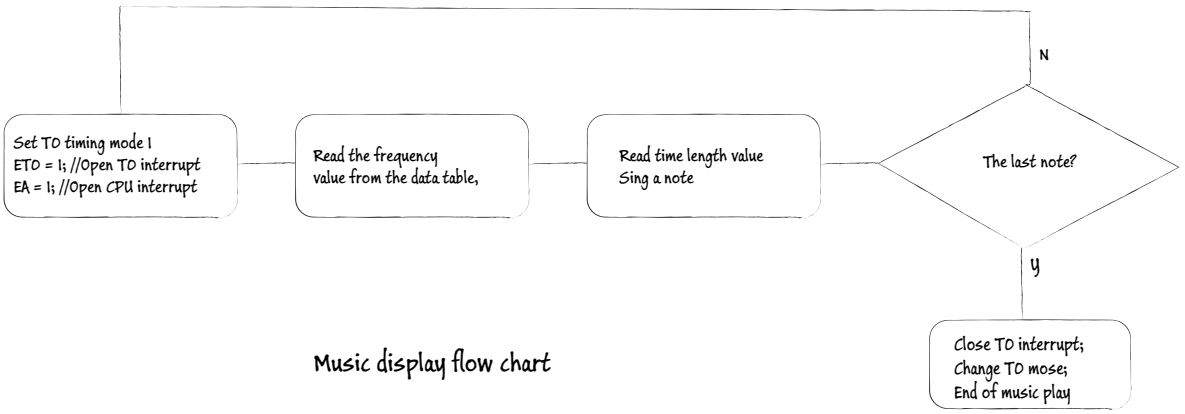

Figure 21: Music play flow chart

10 UART module, timer1 multiplexing

10.1 Baud rate(1200) calculation

Here we use baud rate 1200 since the value of crystal of board is 12MHZ to minimize the error we should use the low boud rate. At boud rate 1200 we have TH1 = 0xCC. We use the UART protocal to communicate with computer getting the command. The main criteria for UART communication is its baud rate. Both the devices Rx/Tx should be set to same baud rate for successful communication. For the 8051 the Timer 1' is used to generate the baud rate in Auto reload mode.Usually, an 12 Mhz crystal oscillator is used to provide the clock to 8051. We need to load Timer1(TH1) in Mode2 in order to generate the required baud rate. The final formula for baud rate is as below.

$$Baudrate = Fosc/(32 \times 12 \times (256 - TH1))$$

$$TH1 = 256 - (Fosc/(32 \times 12 \times Baudrate))$$

//If(SMOD==0 in PCON register)

$$TH1 = 256 - (Fosc/(32 \times 6 \times Baudrate))$$

//If(SMOD==1 in PCON register) Now with Fosc = 12Mhz, TH1 value for 1200 baudrate will be:

$$TH1 = 256 - (12.0 \times 10^6)/(32 \times 12 \times 1200) = 0XCC$$

There are four steps to init a UART serial port:

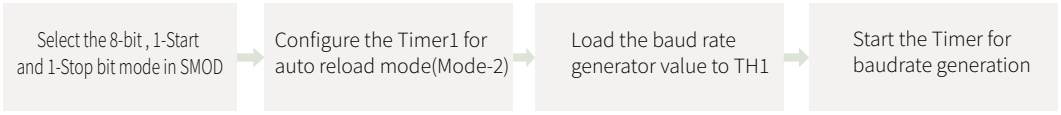

Figure 22: Initial UART

Steps of transmission:

1. Load the new char to be transmitted int SBUF.
2. Wait till the char is transmitted.
3. TI will be set when the data in SBUF is transmitted.
4. Clear the TI for next cycle.

Wait procedure:

1. Wait till the Data is received.
2. RI will be set once the data is received in SBUF register.
3. Clear the receiver flag(RI) for next cycle.
4. Copy/Read the received data from SBUF register.

10.2 Timer1 multiplexing

Due to the tightness of the timer resources, we have multiplexed Timer 1. The music playback module and the UART module share a timer. When the clock is in DIS mode, our timer 1 is set to the music playback mode. When the clock is in MUT, ERD, EWD, UAR mode, the timer is in the UART transfer setting.The settings of the timer in different states are as follows:

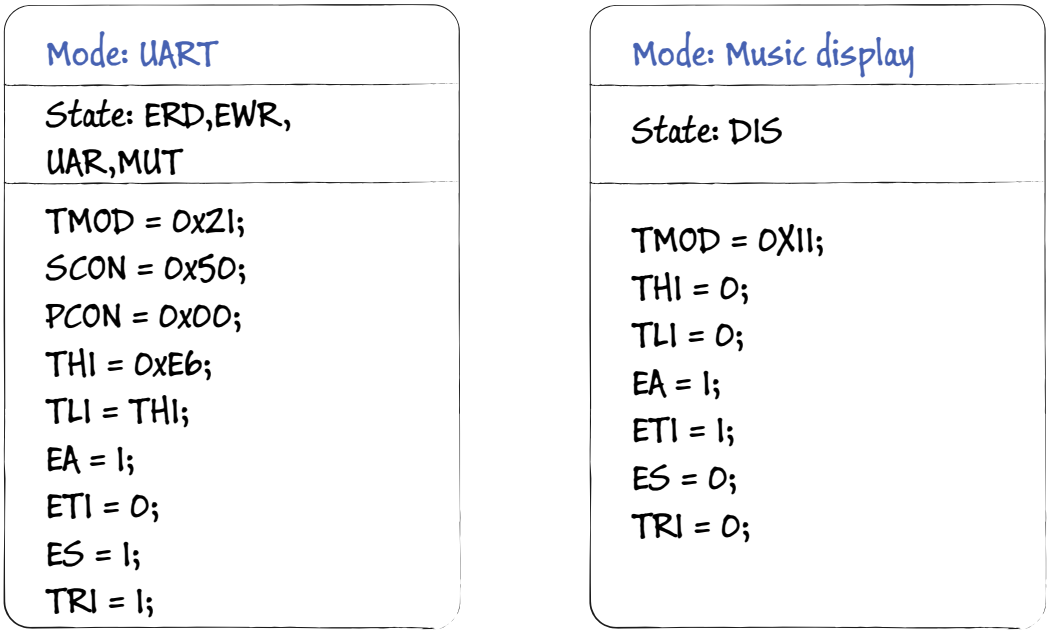

Figure 23: Different configeration of Timer1 at different state

11 FSM of clock

In the finite state machine of the clock, we have two main states: **time setting state** and **normal walking state**. Among them (MUT, DIS, ERD, EWR, UAR) are completed in the normal walking state, only the SET mode is in the time setting state. We will introduce these working modes separately below.

DIS: In the DIS mode, if the temperature exceeds the predetermined threshold clock, it will alarm, or the clock will just go for an hour. Our buzzer can sound normally, and the clock can play music, which is the opposite of the MUT mode.

MUT: In the DIS mode, if the temperature exceeds the predetermined threshold clock, it will alarm, or the clock will just go for an hour. Our **buzzer can not sound.**

UAR: In UAR mode, our clock can communicate with the computer through the UART communication protocol, mainly transmitting real-time clock data or data stored in the EEPROM to the computer.

ERD/EWR: In ERD/EWR mode, we write the current clock data to the EEPROM or read the corresponding historical data from the EEPROM. Note that these two modes can only be turned on after the UAR mode is turned on, and the ERD/EWR mode should be completely turned off by turning off the UAR mode. This is because we can only transfer data to the computer after UAR mode is turned on. Since **ERD/EWR is time consuming**, remember to turn off both modes in time.

SET: We adjust the time, date, and week by entering the SET mode. Their adjustments can be made via the remote control.

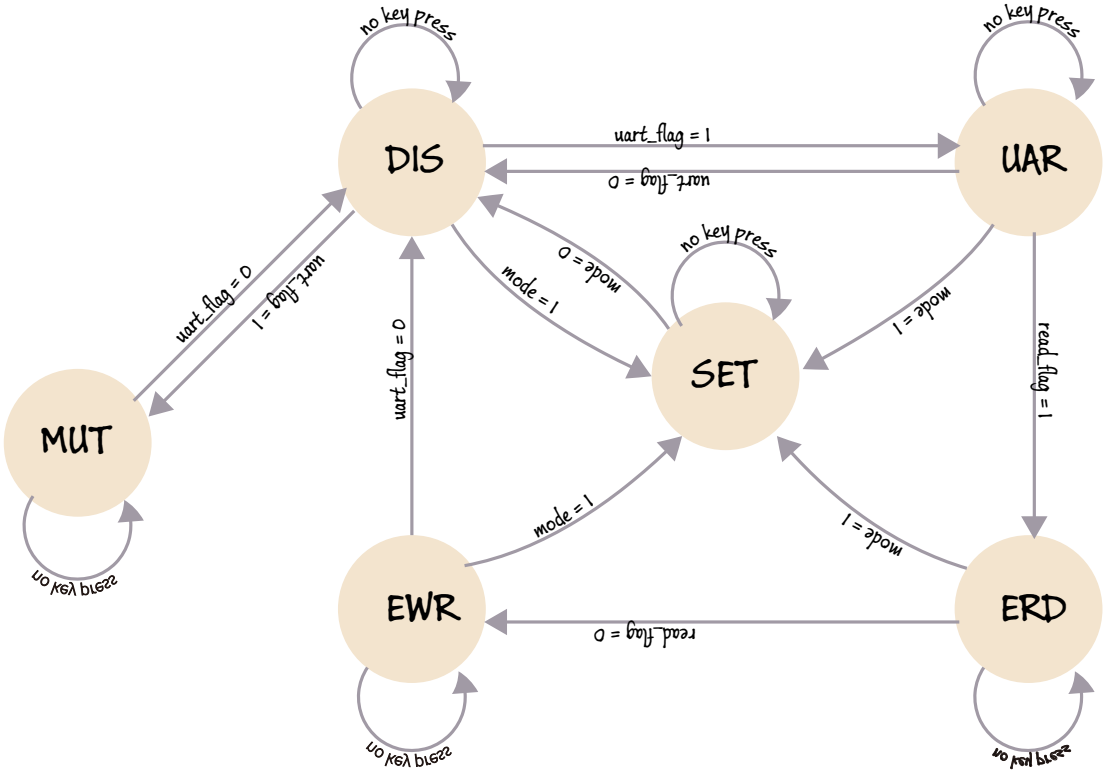

Figure 24: FSM of system

Because the user setting state is very important, we will focus on explaining the setting state below. When the left shift button or the right shift button is not pressed in the setting state, if any of the number setting buttons is pressed at this time, the corresponding cursor position is changed to the corresponding setting number. And the cursor position is automatically shifted to the right. When the left shift button is pressed, the cursor position is automatically shifted to the left. When the right shift button is pressed, the cursor position is automatically shifted to the right. The button operation of this program is very convenient. When the corresponding 1-9 button is pressed, the corresponding bit directly becomes the pressed number.

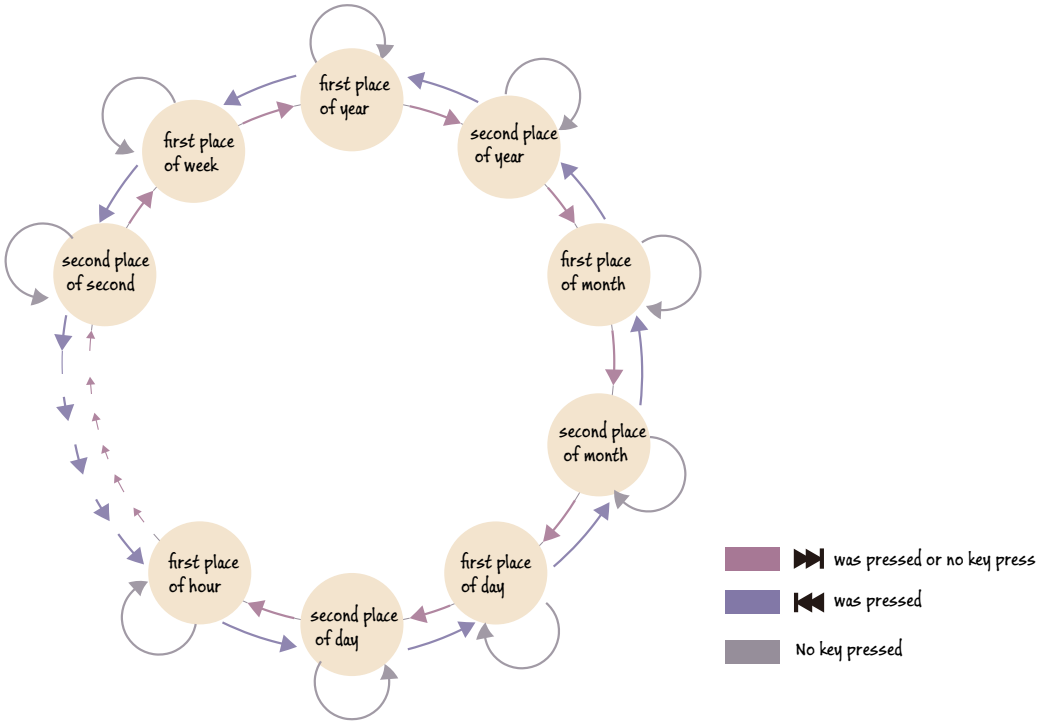

Figure 25: FSM of setting state

12 Main function & user serve function

In the main function, we only perform the function of music playing. When the detection program needs to alarm or play music, the main function calls the corresponding music playing function. Our user service programs are all placed in the timer interrupt. The timer interrupts the main function every 500ms. In the interrupt service function, we mainly carry out the update of the temperature value, the reading of the clock data, the display of the temperature and clock, and the button testing work. When it is detected that the EEPROM is to be operated, we also read and write the EEPROM in the interrupt.

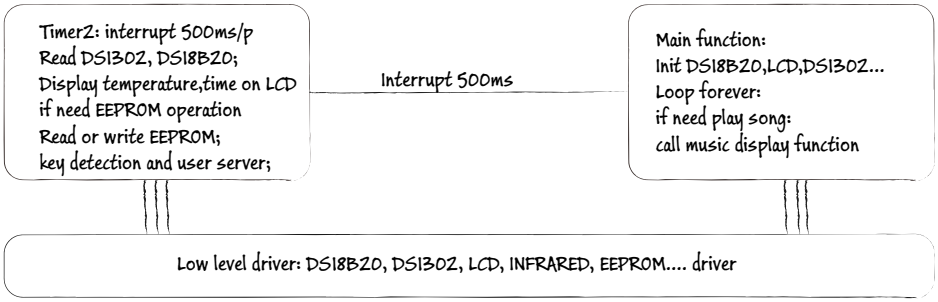

Figure 26: System block

13 Appendix

Simulation result:

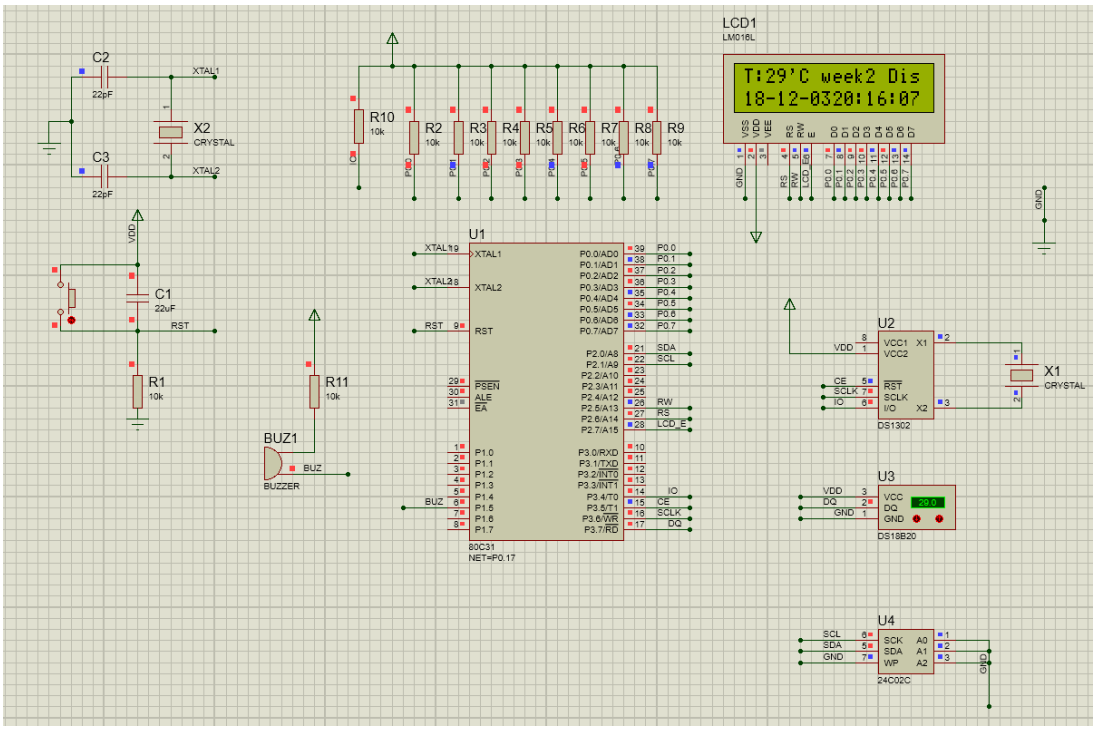

Figure 27: Simulation result

Real circuit:

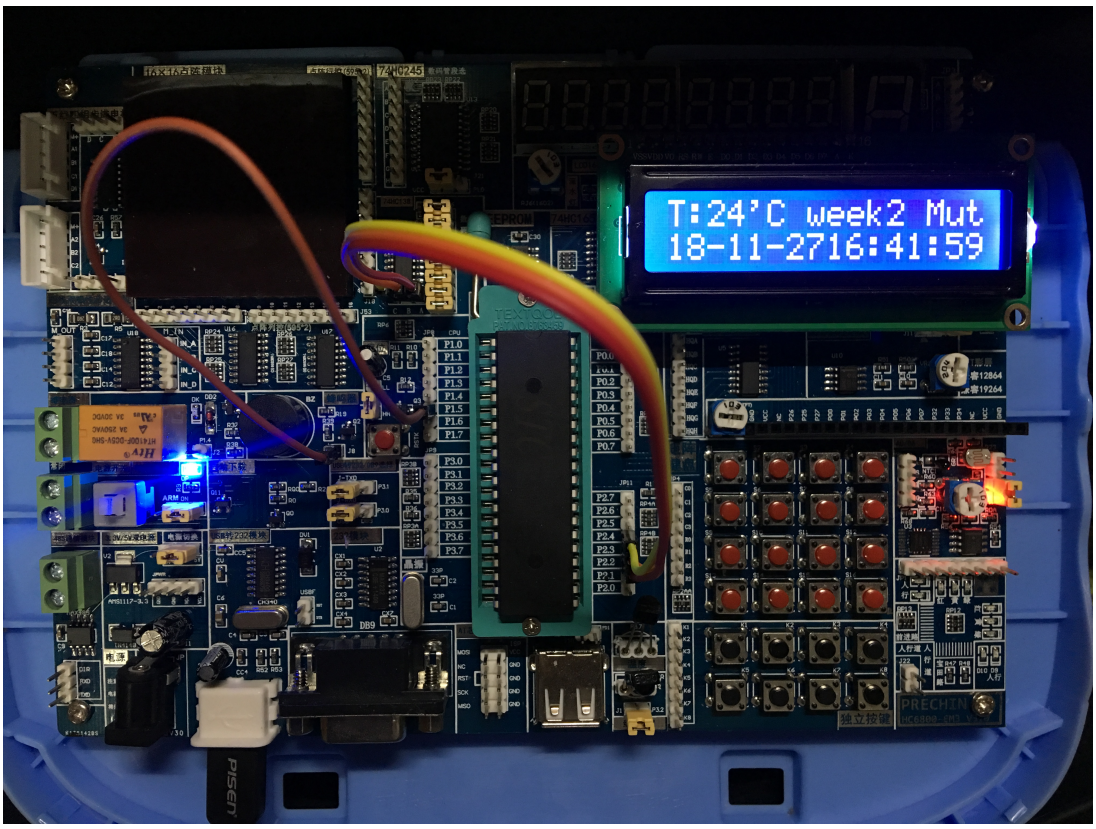

Figure 28: Real circuit result

UART result:

Figure 29: UART real time

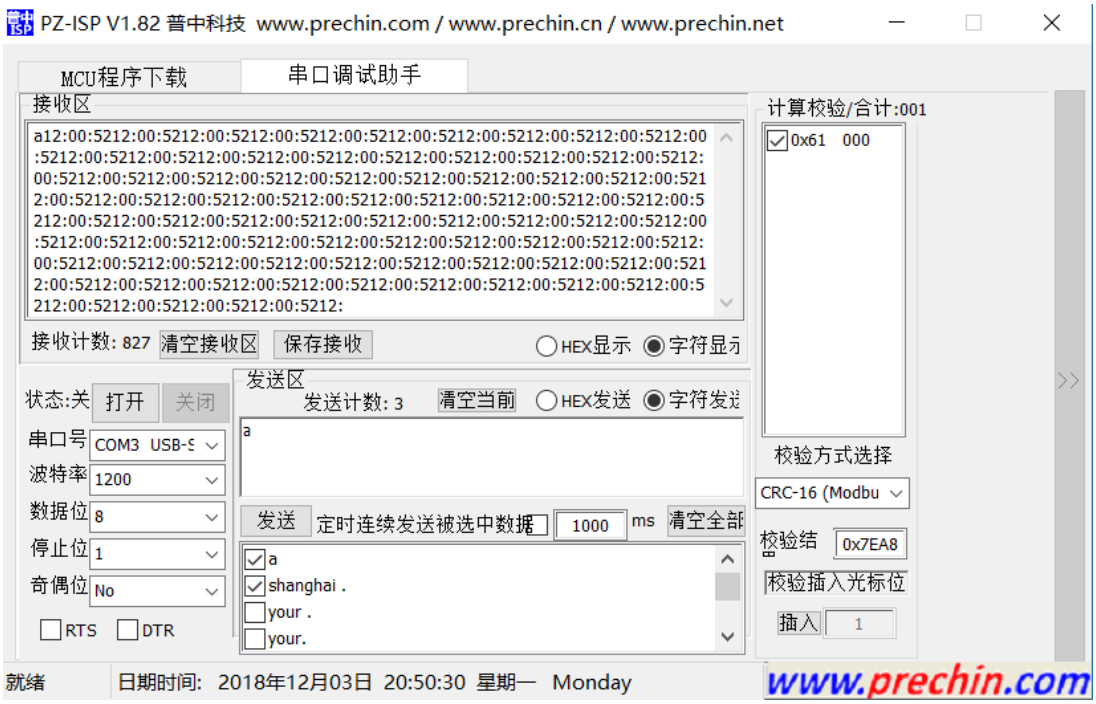

Figure 30: EEPROM read