Lecture Notes on Wireless Networks and Communications (2018) 1: 1-16
Clausius Scientific Press, Canada

I2C data transfer program design and communication protocol improvement

Yuxin DUAN

Department of Communication Engineering, SiChuan University, 1st Ring Road, Chengdu Sichuan, China
Yiyubai@126.com

Keywords: *I2C communication protocol, eUIDE, EM78P259N, AT24C02, assembly*

Abstract: This basic concepts and principles of the I2C communication protocol were introduced in this paper. Based on the I2C communication protocol, the eUIDE software platform and the ELAN EM78P259N single-chip microcomputer was used to realize the data transmission and reception between EM78P259N and AT24C02 in assembly language, and the timing relationship diagram of data transmission verification and data transmission SDA and SCL was given. The defects of the existing I2C communication protocol were analyzed, and the improvement schemes were given from the physical layer and the protocol layer, and the advantages and disadvantages between the schemes were evaluated.

1. Introduction

Data communication protocol, also known as data communication control protocol, aims to ensure the efficiency and reliability of communication between two parties in a data communication network.

The data communication protocol makes a series of conventions for data format, data transmission order and rate, confirmation or rejection, error detection, retransmission control and interrogation. This paper mainly introduces the Inter-Integrated Circuit (hereinafter referred to as I2C) bus communication protocol, and gives the timing diagram and protocol improvement of data transmission verification, data transmission serial SDA (hereinafter referred to as SDA) and serial SCL (hereinafter referred to as SCL).

The definition, basic concept and working principle of I2C bus communication protocol are introduced in this paper. eUIDE software platform and ELAN EM78P259N single-chip microcomputer are used to realize data transmission and reception between EM78P259N and AT24C02 chips in assembly language, and the corresponding verification results are given.

Although the I2C bus communication protocol saves the pins of the microcontroller and the additional logic chips, the printed circuit board is simpler and costs less. However, when one device occupies the SCL as the host during communication, the other I2C devices can only act as slaves, which passively respond to the commands of host but do not have the initiative to send and receive data. In response to this major flaw, this paper will present improvements and compare the advantages and disadvantages of several scenarios.

2. I2C bus communication protocol

The I2C bus is a simple two-wire synchronous serial bus developed by Philips. I2C is both a bus and a communication protocol.

In embedded development, communication protocols can be divided into physical layer and protocol layer. The physical layer guarantees the transmission of data on the physical medium; the protocol layer mainly stipulates the communication logic, and uniform standards for packing and unpacking the data transmitted by both parties.

2.1. Physical layer

The wiring diagram of the I2C communication system is shown in Figure 1.

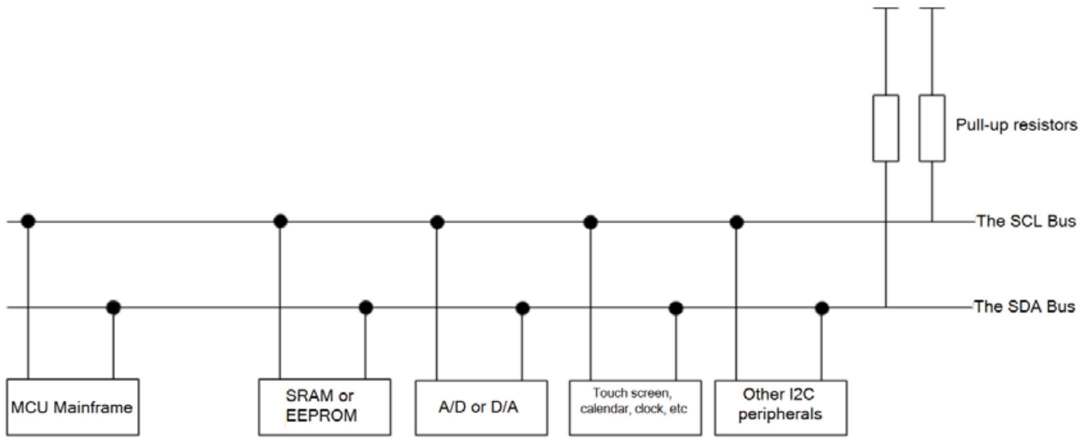


Figure 1 I2C system wiring diagram (two-wire system)

I2C communication requires only two bidirectional buses, SDA and SCL. SDA is used to transmit data, and SCL is used to synchronize data transmission and reception.

Multiple I2C communication devices can be connected to the bus. Each device has an independent 7-bit address code. The upper 4 bits are fixed. The device type is specified by the manufacturer. The lower 3 bits are the device pin custom address. The host is a logic module with a central processing unit CPU that initializes the data transfer of the bus and generates a clock signal that allows transmission. The host uses each device's independent address to access other devices, and any addressed device is called a slave. The I2C communication bus supports data transfer between one host and multiple slaves, i.e. ‘one-to-many’.

Both SDA and SCL require a pull-up resistor. When the bus is idle, both lines are high level. Any device connected to the bus outputs low level will pull the bus signal low.

2.2. Protocol layer [1]

The protocol layer specifies the data validity of the communication, start and stop signals, responses, data read and write sequences, address broadcasts, and so on.

2.2.1. Data validity

When the bus performs data transmission, each bit of data has a corresponding clock pulse (or synchronous control), that is, each bit of data is serially transmitted bit by bit on the SDA in cooperation with SCL. While SCL is high, the data on SDA must remain stable, i.e. there must be no level changes. The high and low states on SDA are allowed to change only while SCL is low.

As shown in Figure 2 below, to transmit the first bit of data, the SDA level must be changed while SCL is still low.Then SCL is high, the slave receives the first bit of data. SDA can change according to the transmission data value when SCL goes low. when SCL is high again, SDA can't change level, slave receives second bit data, and so on, until reaches stop signal.

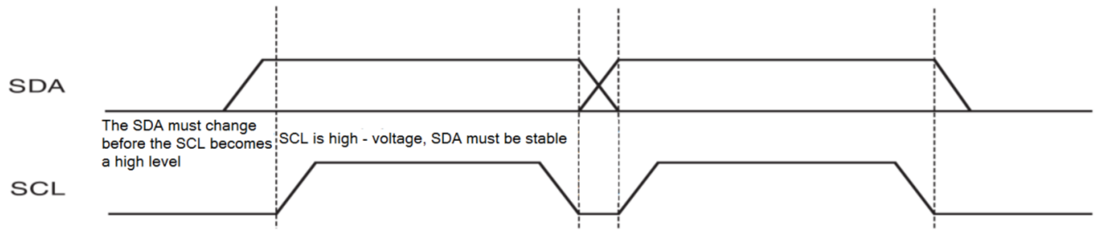

SDA
The SDA must change before the SCL becomes a high level
SCL is high - voltage, SDA must be stable
SCL

Figure 2 Data validity diagram

2.2.2. Start and stop signals

The start and stop signals mark the start and end of data transmission, which is different from the normal transmission data.

It is the start signal when SDA changes from high to low level during the high level of SCL. The stop signal is that SDA changes from low to high level when SCL is high level. There is contradiction to: SDA level must be fixed when SCL is high. i.e. ‘violation.’ It is this special change that distinguishes the start and stop signals from normal data transmission.

Both the start and stop signals are sent by the master.The bus is occupied after the start signal is generated, . After the stop signal is generated, the bus is in an idle state, and both SDA and SCL are at a high level.

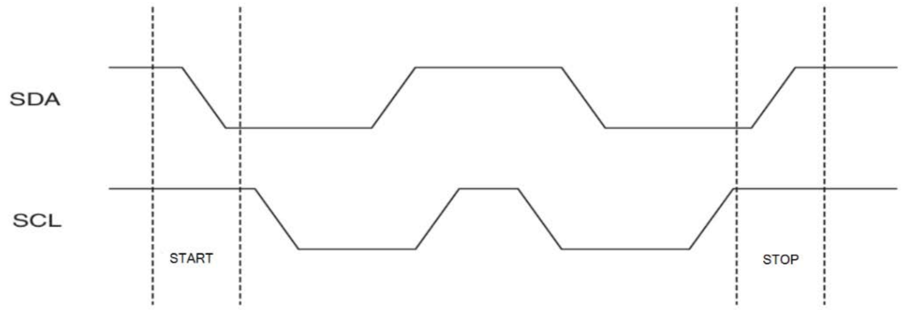


Figure 3 Start and stop signal timing diagram

2.2.3. Valid response and non-response

All data on the I2C bus is transmitted in 8-bit bytes. The SCL level changes once and the SDA transmits 1-bit data as described in 2.2.1. Each time the host transmits one byte (8 bits), it releases the SDA during the ninth clock pulse and waits for the slave to feed back the acknowledge signal (high/low level of one pulse width).

When the response signal is a low pulse, it is specified as a valid response (hereinafter referred to as ACK), indicating that the slave has successfully received the byte; when the response signal is a high pulse, it is specified as a non-acknowledgement (hereinafter referred to as NACK), indicating The machine failed to receive the byte.

If the slave pulls SDA low during the low period before the ninth clock pulse and ensures that SDA is stable low during SCL high level, then ACK is considered to be fed back.

If the host requests data from the slave and receives it, the host is the requesting party, and it can know whether having received or not at the receiving end, and does not need to feed back to the slave. However, after the host receiving the last byte, a NACK signal is sent to keep synchronized with SCL, the slave is notified to end the data transmission, and the SDA is released, so that the host sends a stop signal.

If the master pulls SDA high during the low period before the ninth clock pulse, and ensures that SDA is stable high during high level of SCL, it is considered to be NACK.

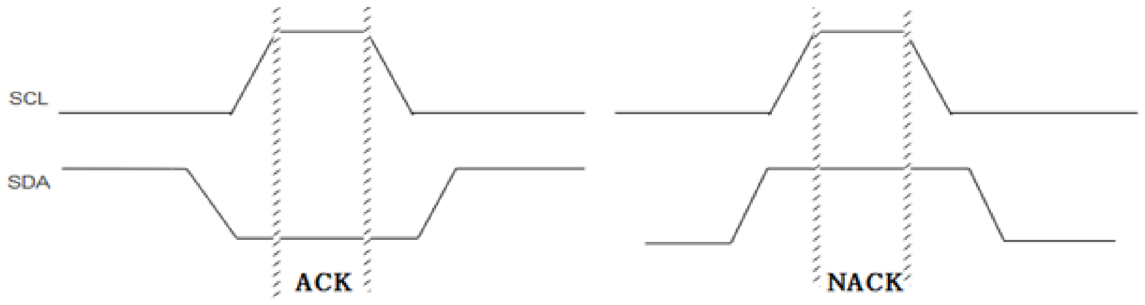


Figure 4 Response signal

2.2.4. I2C read and write process

1 Write data flow

The standard process for writing data is:

(1) The host first initiates the START signal; then sends the I2C address code (7bit, see 2.1) and write operation 0 (1bit), waiting for the slave to respond to the ACK. This process is called address broadcast.

(2) The slave sends an ACK;

(3) The host sends the register address (8bit), that is, the data is written to the slave, waiting for the slave to respond to the ACK;

(4) The slave sends an ACK;

(5) The host sends data (8 bits), that is, the data to be written into the register, waiting for the slave to respond to the ACK;

(6) The slave sends an ACK;

(7) Steps 5 and 6 can be repeated multiple times, that is, sequentially writing a plurality of registers starting from the register address indicated in step 3, as shown in FIGURE6

(8) The host sends a stop signal.

2 Read the data flow

Use a read-write composite format. During the transmission, the host sends 2 start signals: the first time the master finds the slave through the slave address, sends the slave internal register or memory address; and reads the contents of the address the second time . When the host wants to read the slave data, it will release the control of the SDA. The slave controls the SDA, and the host is responsible for receiving. When the host wants to write data to the slave, the SDA is controlled by the master, and the slave is responsible for receiving.

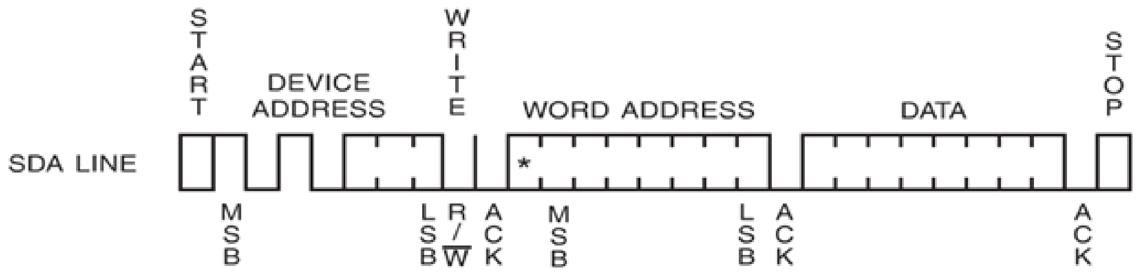


Figure 5 Write the 1byte data flow

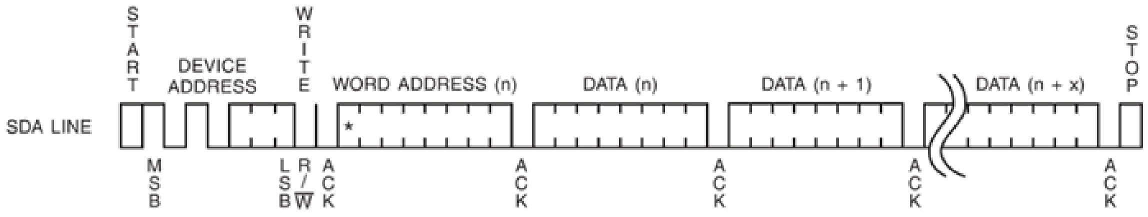


Figure 6 Write the nbyte data flow

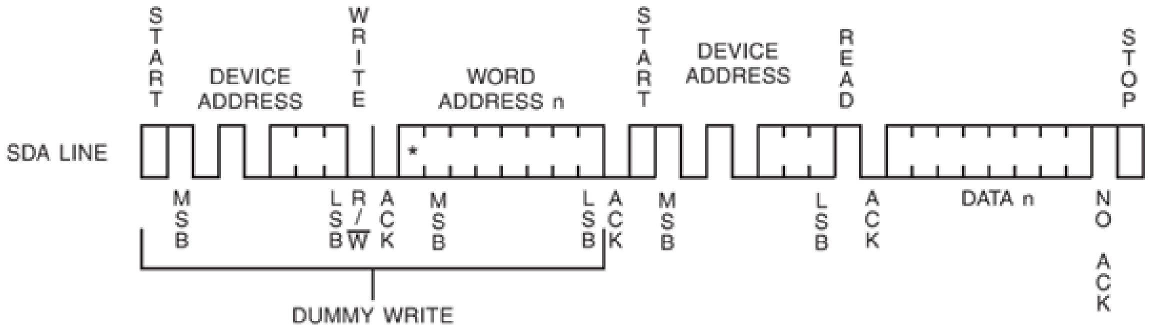


Figure 7 Read the 1byte data flow

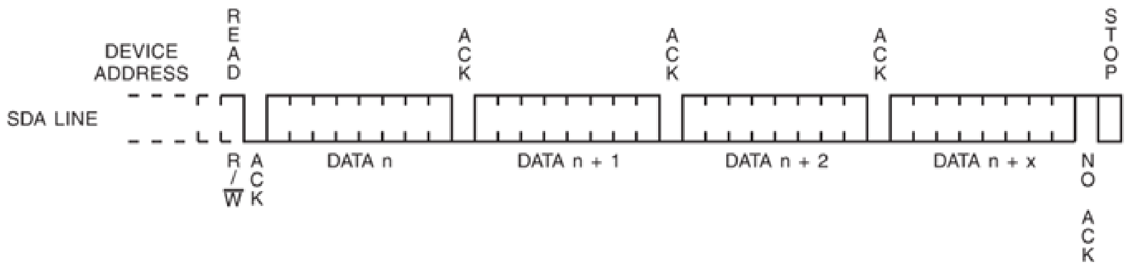


Figure 8 Read the nbyte data flow

The standard process for reading the data is :

(1) The host first initiates a START signal; then sends an I2C address code (7bit) and a write operation 0 (1bit), waiting for the slave to respond to the ACK;

(2) The slave sends an ACK;

(3) The host sends the register address (8bit), that is, the data to be read is stored in the slave position, waiting for the slave to respond to the ACK;

(4) The slave sends an ACK;

(5) The host initiates the START signal again; sends the I2C address code (7bit) and read operation 1 (1bit), prepares to read the data, and waits for the slave to respond to the ACK;

(6) The slave sends an ACK;

(7) At this time, the slave transmits data from the previously received address, and the host receives (8 bits);

(8) The host sends a NACK;

(9) Steps 7 and 8 may be repeated many times, and the slave sequentially issues a number of sets of data starting from the register address indicated by step 3. Note that each time the host receives 1 Byte of data, it must send an ACK to the slave so that the slave can continue to send data, except for the last 1 Byte. As is shown in Figure 8;

(10) The host sends a stop signal.

2.2.5. Bus Timing Summary

SDA and SCL have strict timing correspondence. When SCL is low, the slave does not receive data. At this time, the host can change the SDA level. When SCL is low, the SDA must be solidified, and the data is waiting for reception. If SCL is low, the SDA level changes, that is, a violation occurs, and the start and stop signals can be judged according to the order of the level change.

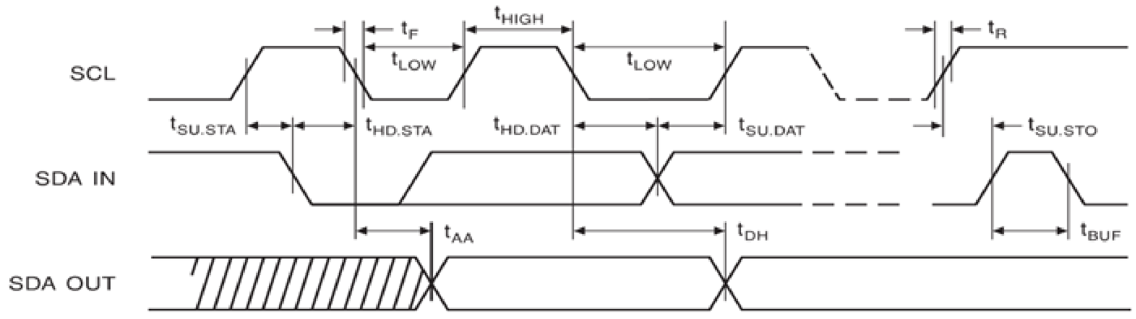


Figure 9 The bus timing

The data transmitted on the I2C bus includes an address signal and a data signal. After the host sends a start signal, all slaves on the bus start waiting for the host address broadcast. The host transmits 1 byte of data. The first 7 bits are the slave address, the 8th bit is the transmission direction, ‘0’ indicates that the host sends data (write), and ‘1’ indicates reception (read). When the broadcast address is the same as a device address, the device is selected and an ACK or NACK is sent to the host. After the host receives the ACK, it continues to send/receive data, and the unselected device ignores the subsequent data signal.

If the host is conFigured to write data direction, the host starts to transfer data to the slave. The data packet size is 8 bits. Each time the host waits for the ACK of the slave it sends 1 byte of data. The process is repeated when multi-byte data is sent. If conFigured as read direction the slave starts to return data to the host, and the packet size is still 8 bits. Similarly, Each time the slave waits for the ACK of the host it sends one byte of data, and repeating this process will return multiple data. When the host wants to stop receiving data, it will feed back NACK and the data transmission will end. Each data transfer is always terminated by the host generating the end signal, but if the host wishes to continue to occupy the bus, the end signal may not be generated, but the start signal is again sent to the other slave.

3 I2C data transmission program design and verification

This section takes a host and a slave as an example By using the eUIDE software platform and the ELAN EM78P259N microcontroller, the data is transmitted and received between the EM78P259N and AT24C02 chips in assembly language. The EM78P259N acts as the master and the AT24C02 chip acts as the slave. Finally, the verification result of data transmission, the timing relationship diagram between SDA and SCL, the oscilloscope data pulse display, SCL and SDA timing pulse display are given.

3.1 Physical layer hardware connection

The pin 1 of EM78P259N is connected to pin 5 of AT24C02 chip, pin 2 is connected with pin 6 ; pin 6 of AT24C02 chip is SCL, pin 5 is SDA, as shown in Figure 10;pin 1 corresponds to P56, pin 2 corresponds to P52, as shown in Figure 11. Among them, P5 is Port 5, corresponding to IOC50 input and output controller; SCL corresponds to the second bit of Port 5, and SDA corresponds to the sixth bit of Port 5.

The output of the 78L05 module is 5V, and the pull-up resistors on SCL and SDA are connected to it.

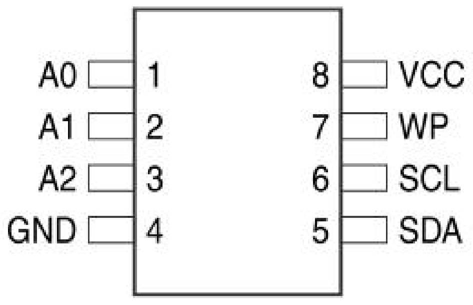


Figure 10 AT24C02 pin Figure

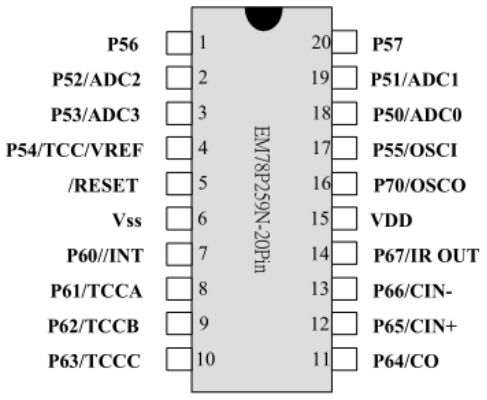


Figure 11 EM78P259N pin Figure[2]

3.2 Programme design

3.2.1 1Byte data transmission and reception

The program is divided into two parts. According to the program flow described in 2.2, the program includes start and stop signal, the host sending data to the slave, the host receiving the data sent by the slave, and the slave responding to the host with the ACK and the host NACK.

1. The host sends data to the slave

The ‘Start START_CON’sub-program is called first, and the host then issues the slave device addressing code. Note here that the AT24C02 high 4-bit device type code is 1010; according to Figure 10, A0, A1, A2 address bits of the slave AT24C02 are grounded, the device address code is

000, and finally a write signal 0 is added, and the host issues 8 bits 1010 0000, the timing relationship is shown in Figure 12.

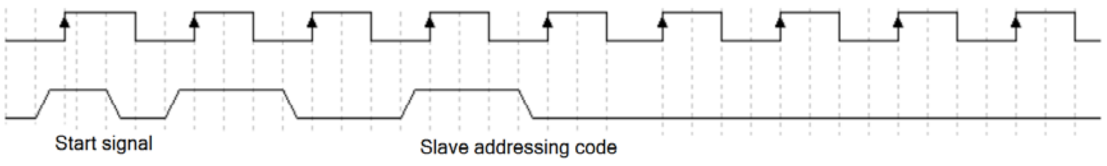


Figure 12 Start signal and send addressing codes

‘DATA_OUT’sending data subprogram is called , firstly pulling down SCL, changing host pin 1 as output, then judging output buffer register ‘OUT_buffer’ bit by bit from the highest bit (7 bits) (10100000 is stored in it), If it is 1 SDA will be Pulling high, which means outputting data 1, and vice versa, indicating outputting data 0. Continue to cycle the left ‘OUT_buffer’ and output bit by bit from the highest bit.

After receiving from the slave, the host waiting for the slave response subprogram ‘ACK_tran’ is called. Firstly SCL is pulled low, SDA pulled high, that is, changing host pin 1 as input, and then check if SDA is low. If it is low, the data is correctly received. Pull pin 1 low again, set 0, the direction becomes output, ready to output data again; otherwise, ‘DATA_OUT’ is called again to retransmit.

Next, send the address where the slave stores the data, here set to hexadecimal 0x01, the process is as described above.

Finally, the data is transmitted (here, 06 as an example is 0000 0110). After receiving the ACK of the slave, the host calls the end signal subprogram ‘STOP_CON’, and the data transmission process ends. The timing relationship is shown in Figure 13.

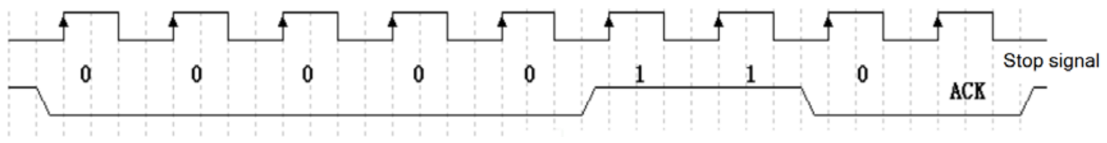


Figure 13 The timing diagrams of the data 06, ACK and end signals

2 The host requests data from the slave

The "START_CON" subprogram is called first, and the host then calls the "DATA_OUT" subprogram to issue the slave device addressing code 1010 000 and write data signal 0.

The slave call "ACK_tran"after receiving information, then send the data store address 0x01of the slave .

Different from sending data, the next step is to call the "START_CON" subprogram again, call the "DATA_OUT" subprogram again to issue the slave device addressing code 1010 000 and read the data signal 1, and call "ACK_tran" to determine if the slave received this set of data. then call "DATA_IN" data receiving subprogram. Pull down SCL and change the host pin 1 as input. Then, move left to the output buffer register "OUT_buffer" ,continuously judge whether SDA is high or low. If it is high, the lowest position of OUT_buffer is set to 1, otherwise it is set to 0. The data transferred by the slave is finally stored in the OUT_buffer.

The host calls the "NO_ACK" subprogram to send a NACK. Finally, the stop signal subprogram "STOP_CON" is called and the receiving data process ends. The relationship between NACK and end timing is shown in Figure 14 below.The DATA_OUT and DATA_IN subprogram flowcharts are shown in the following Figure.15 and Figure 16.

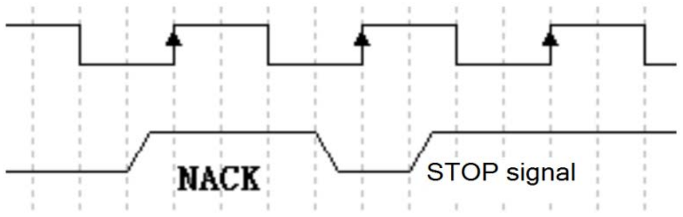
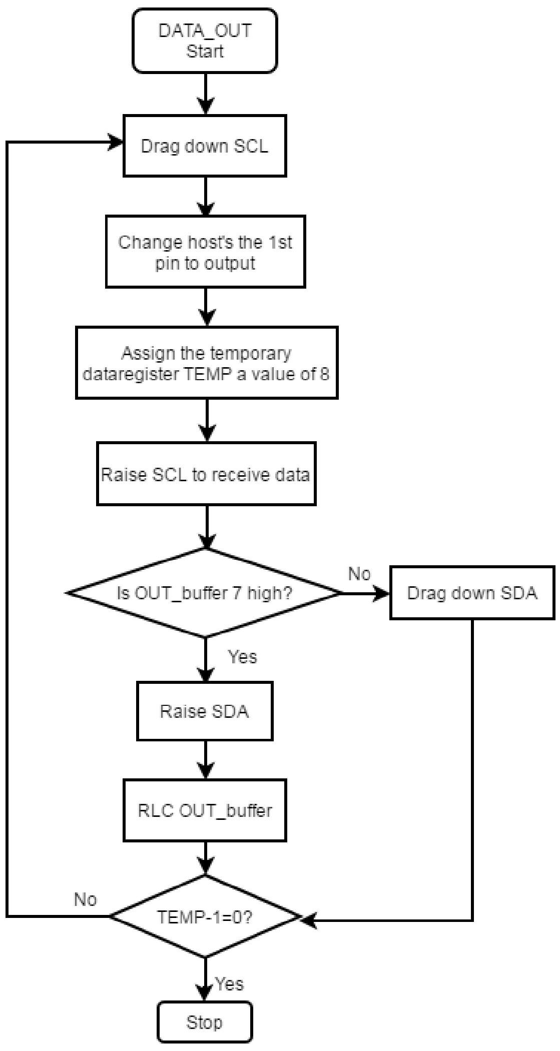


Figure 14 NACK and end time sequence


DATA_OUT Start
Drag down SCL
Change host's the 1st pin to output
Assign the temporary dataregister TEMP a value of 8
Raise SCL to receive data
Is OUT_buffer 7 high?
No
Drag down SDA
Yes
Raise SDA
RLC OUT_buffer
No
TEMP-1=0?
Yes
Stop

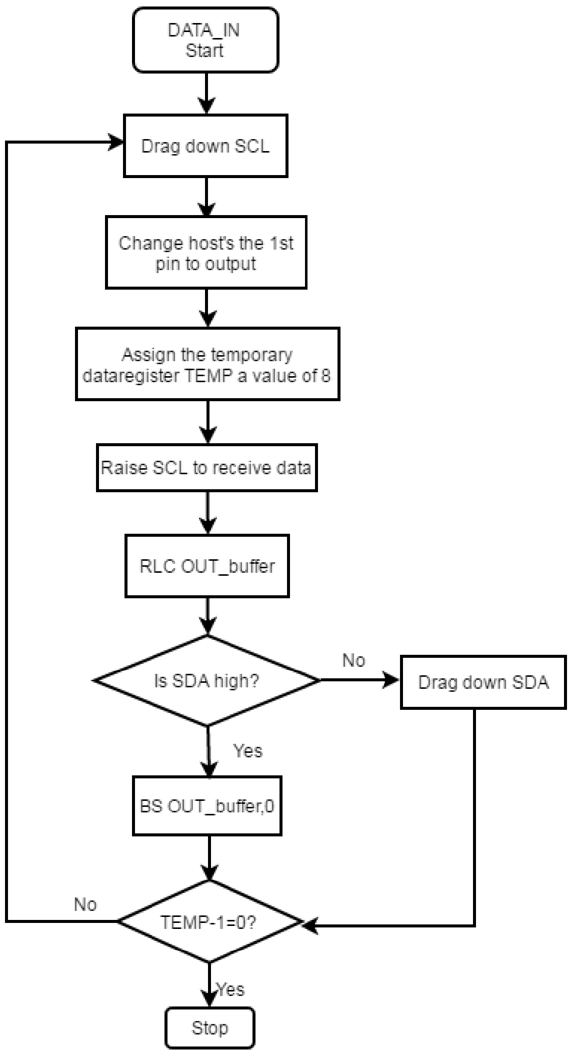

DATA_IN Start
Drag down SCL
Change host's the 1st pin to output
Assign the temporary dataregister TEMP a value of 8
Raise SCL to receive data
RLC OUT_buffer
Is SDA high?
No
Drag down SDA
Yes
BS OUT_buffer,0
No
TEMP-1=0?
Yes
Stop

Figure 15 DATA_OUT subprogram Figure 16 DATA_IN subprogram

3.2.2 Verification results

Add the unipolar non-return to zero code in the program, and display of the transmission and reception pulse of 06:0000 0110 is as shown below Figure 18.

The DuPont line is used to test the output wave forms of pins 5 and 6 of the AT24C02 chip, that is, the timing relationship between SCL and SDA. In this case, in order to continuously display the timing relationship on the oscilloscope, it is necessary to add a loop program to the main program, but this will damage the chip life, and it is necessary to turn off the power in time after observation.

The start, end, and data 0000 0110 timing diagrams are as follows. The pulses shown in Figure.21 are, in order, a device addressing code and a read signal 10100001, data 0000 0110, a NACK signal, and stop signal.

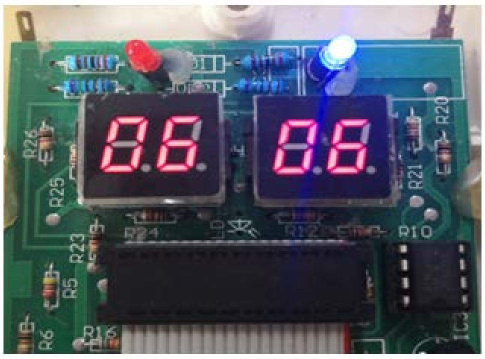


Figure 17 Data transmission results

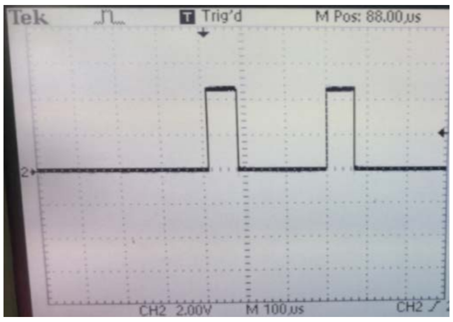


Figure 18 Data pulse

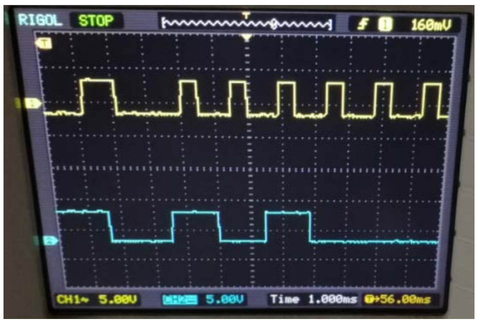
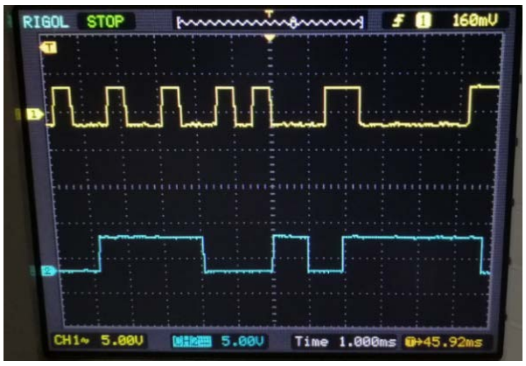


Figure 19 Start signal
(SCL is high, SDA is high to low)



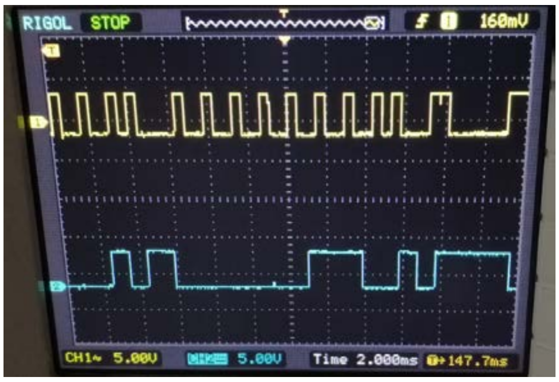


Figure 20 The stop signal(SCL is low, SDA is low to high, see last rising edge)

Figure 21 Data reception

Disadvantages of the I2C bus:

1 Although in the high-speed mode, the serial 8-bit bidirectional data transmission bit rate can theoretically reach 3.4 Mbps,most I2C devices do not support high-speed transmission at present .

2 slave passive responding to host

The host is mainly used to drive the SCL, and the slave responds to the host; both can transmit data, but the slave cannot initiate the transmission, and the transmission is controlled by the host; the slave passively receives the data sent by the host, or responds to the request of data from host.

Advantages of the I2C bus:

(1) The current consumption is extremely low;

(2) Resistance of high noise interference;

(3) The power supply voltage range is wide;

(4) Wide operating temperature range;

(5) Fault diagnosis and debugging are simple, and faults can be traced immediately;

(6) Adding or deleting ICs in the system will not affect other circuits in the bus;

(7) The portability is good, and different devices conforming to the protocol can be driven by the same set of codes;

(8) The structure is simple. The host establishes a multi-machine communication mechanism through the address code, eliminating the chip selection line of the peripheral device, and the system is still a two-wire structure regardless of how many devices are connected onto the bus.

5 I2C communication protocol improvements

This section is aimed at the shortcomings of the slave's passive response to host, and proposes improvements from the physical layer and the protocol layer .

5.1 Physical layer - adding new communication line

The original bus protocol physical layer has only one SDA and one SCL. Any device with a logical CPU occupies SCL. Other devices can only act as slaves in response to the device which occupies SCL and the data is transmitted in accordance with the rhythm of the host SCL. In order to realize the two-way communication between the master and the slave on the basis of the original two-wire system, a new SCL is added to form a three-wire structure, and the physical connection is as follows.

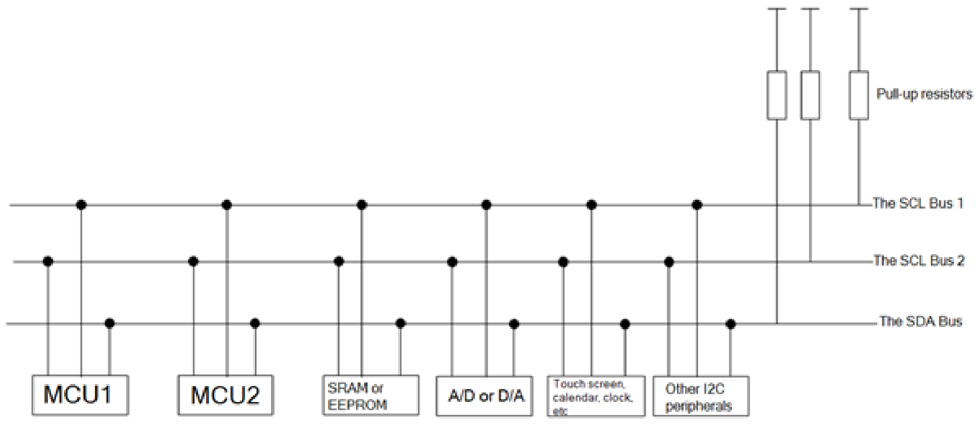


Figure 23 Three wire system

The status of two SCLs of All devices with control SCL function must be detected before initiating data transmission. Here, SCL is low to indicate idle, and high to indicate device occupancy.

It is assumed that the clocks of the devices are different, but the difference is much smaller than the pulse with the narrowest width. Before the device is to transmit data, it sends a piece of data as the command data for detecting conflicts. The specific content can be customized. Here is a 2-byte reference scheme. The first four digits of the first byte are the data priority, and the host user customizes the urgency of the data to be sent. The highest priority is assumed here to be 1. The larger the value, the smaller the priority. The last four digits are spare extension bits, which can be expanded when the number of devices is too large.

The first four bits of the second byte are the device priority and are initially 0001. The benefit of the spare extension bits placed between device and data priority is that data prioritization can be extended backwards as needed, and device priority can be scaled forward as needed. In general, the priority of level 15 can basically meet the needs; the device priority will overflow after 15 conflicts, and the probability of two devices transmitting data at the same time is small, which can be satisfied for a certain number of devices. In case of special circumstances, the default is extended by two.

The last four digits are fixed device type codes, as specified by the manufacturer.

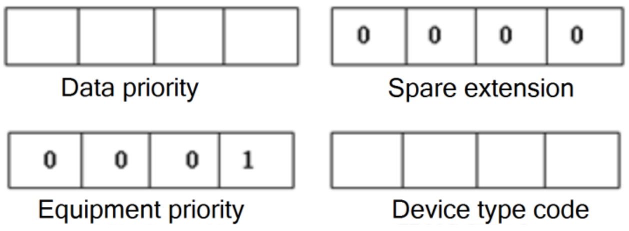


Figure 24 Detecting conflicting instruction data

The protocol layer of the data transmission remains unchanged, and the data is transmitted in units of 8 bits. Here is the agreement:

1. Each device sends and detects data according to its own SCL frequency, but once a device occupies SDA as the host, the frequency of the host prevails;

2. By using the "line and" feature of I2C itself, if two devices send data to SDA at the same time, the level state on SDA presents the result of the two.

The specific steps of data transmission are as follows:

The device detects the status of two SCLs before initiating data communication, and can occupy as long as one SCL is low. After occupying SCL, It delays 8 pulses according to its own clock frequency, and checks the state of SDA at the SCL frequency. If the SDA level changes during this time, it is indicating that the device is transmitting data, the delay continues until the “stop” signal is detected.

When the "stop" signal is detected, immediately send 2 bytes of command data to detect the conflicting, and at the same time, check whether the level of SDA is the same as the issued command according to its own SCL frequency. If a conflict occurs, whichever device first detects the difference, the transmission is abandoned, and the other device is used as the host. The abandoned device continues to delay until the "stop" signal, and the device priority of both devices is increased by one. Repeat the above steps. After the data is completely transferred, SCL and SDA are released.

Since the level on SDA is the result of phase-to-phase, the higher the priority, the later the high level appears, and the difference between the device clocks is much smaller than the pulse with the narrowest width. After the data transmission is completed, the device will release SCL and SDA. Therefore, it is basically guaranteed that data and devices with higher priority can preferentially occupy SDA transmission data.

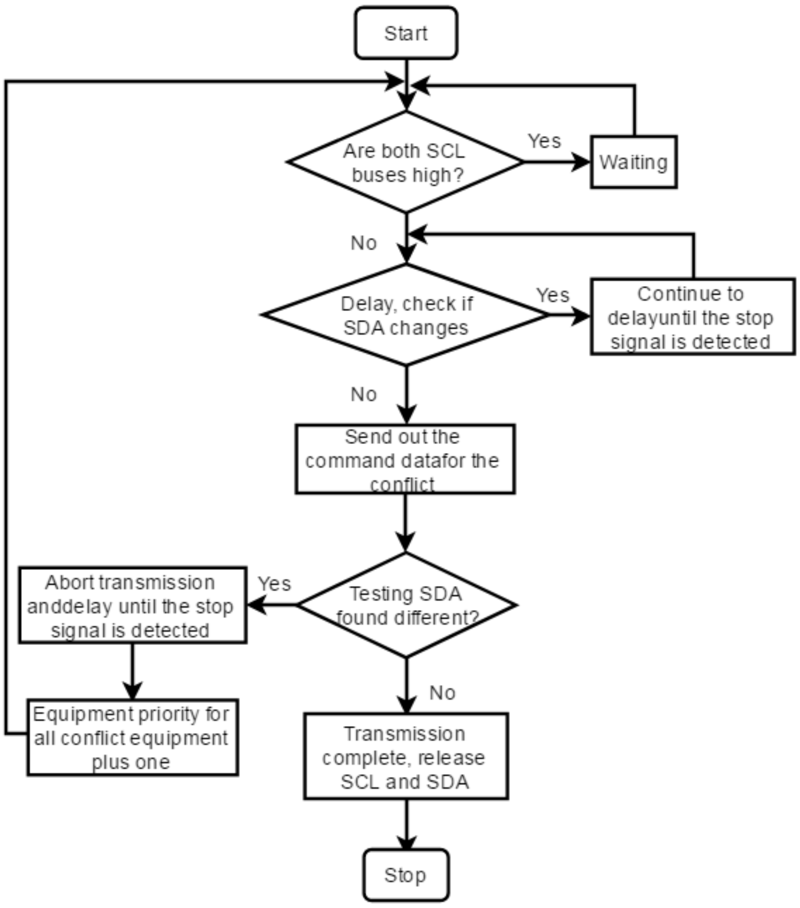

Start
Are both SCL buses high?
Yes
Waiting
No
Delay, check if SDA changes
Yes
Continue to delayuntil the stop signal is detected
No
Send out the command datafor the conflict
Testing SDA found different?
Yes
Abort transmission anddelay until the stop signal is detected
Equipment priority for all conflict equipment plus one
No
Transmission complete, release SCL and SDA
Stop

Figure 25 Physical optimization flow chart

5.2 Protocol layer

Each device continuously detects the SCL level. If the level is low, the SCL is idle, and the device can become a temporary host. However, in the case where multiple devices with SCL control capabilities are attached to the bus, conflicts are most likely to occur. In the following, under the premise of preserving two-wire structure of the physical layer and keeping the signal timing of data transmission unchanged, several optimization schemes are proposed.

1. Carrier sense with collision detection and backoff algorithm

(1) Collision Detection

When two or more devices transmit data at the same time, because the SCL frequency is different, the Superposition zero point of the start signal (first high and then low) is different from the start signal of the device itself, thereby the collision can be detected.

(2) Stop sending

After the conflict occurs, all data transmission is stopped.

(3) Delay determination

The backoff algorithm by a truncated binary exponential type is used to determine the delay to resend data .Firstly determine the basic backoff time, assuming a millisecond. This can be defined based on the performance of the device and the average delay time.

Let K=Min[current collision number, 10], randomly take a number from the discrete integer set $\{0,1,...,2^{k-1}\}$, and takes a value between 0 and 1 as the first retransmission . takes one between 0, 1, 2, 3 as the second retransmission, 0 to 7 as the third , ..., and so on, which is equivalent to the lower priority of the device. Recorded as r[3].

$$\text{Delay} = r\cdot a \tag{1}$$

Repeat the data after the delay and repeat the above steps.

2."low priority" arbitration principle

Suppose there are two host devices with SCL control in a system, which are recorded as temporary host 1 and 2 respectively, and their output data are DATA1 and DATA2 respectively. Both of them send a start signal to SDA one after the other and the temporary host 1 is slightly ahead. Given the " wired and " nature of the I2C system, the signal level obtained on SDA is the result of DATA1 and DATA2.

The following is an example. After setting the start signal, the temporary host 1 sends the data "101...", and the temporary host 2 sends the data "100101...". The two temporary hosts must detect the SDA signal level of their own output every time a data bit is sent. As long as the test result matches the level sent by itself, it will continue to occupy SDA.

The third bit of the temporary host 1 is expected to transmit "1", that is, to send a high level in the third clock cycle, but the third bit of the temporary host 2 is expected to transmit "0". During the high period of the clock cycle, in view of the "line-and-" feature, the temporary host 1 will detect a mismatched low level, at which time the temporary host 1 should abandon the SDA control, making the temporary host 2 the only host of the bus. Thus bus arbitration is implemented.

From the above process, it can be concluded that neither the temporary host 1 nor the temporary host 2 loses data; each temporary host has no priority level, even if the temporary host 1 that first sends the start signal does not finally get the SDA control.

The system actually follows the "low-priority" arbitration principle, which awards the bus to a temporary host that sends a low level on SDA first, while other temporary hosts that send a high level lose control of SDA [4]. As shown in Figure 27.

Of course, you can also design a "high priority" mode with the same steps.

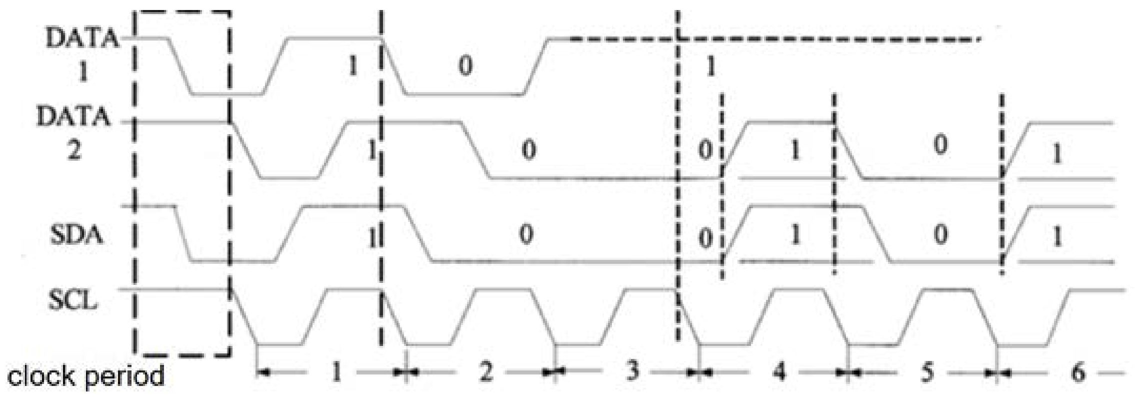


Figure 26 The arbitration principle of low level priority

5.3 Physical Layer and Protocol Layer - Token-ring network

Token Ring is a physical bus structure. The devices attached to it form a logical ring structure. Tokens are transmitted sequentially between devices along the ring bus. The device can only send data frames after obtaining the token, so no conflicts will occur. Since tokens are passed in order on the ring, access is fair to all devices.

The inherent disadvantage of Token Ring is the need to maintain tokens, which require special equipment for monitoring and management, and cannot work once the token is lost. In view of this inherent shortcoming, tokens are rare in the entire computer LAN. Most of the vendors that originally provided token network devices have also withdrawn from the market [5], so this method will not be described again. Only the possibility is provided.

5.4 Comparison of advantages and disadvantages

Physical layer

Advantages:

1. Basically optimize the problem of two-way communication between devices;
2. The SCL is detected by data blocks of two bytes wherein four bits of device priority, four bits of data priority, and four bits are used as alternate adjustment bits, which can be adjusted according to the number of devices connected in the bus. For a certain number of devices, there is basically no conflict.

Disadvantages:

1. One more SCL is used, the system complexity increases, and the cost increases;
2. Because the conflict is to be eliminated, data block for detection of the SCL is at least 2 bytes, which needs additional data cost.
3. If the SDA level is to be judged, and just the device sends 8 bits of all "0" data, it may disrupt the data transmission of the current device.

Protocol layer

A common advantage of protocol layer optimization is that there is no need to increase hardware cost and system complexity.

1. Carrier sense /collision detection and backoff algorithm

Advantages:

The problem of conflict is solved. When a conflict occurs, all conflicting devices stop sending data without additional data interference.

Disadvantages:

A certain system delay is introduced, and the priority of the data to be transmitted cannot be determined. It is very likely that a very important piece of data is delayed because the number of collisions is too many.

2. "low priority" arbitration principle

Advantages:

1. No data is lost between devices that have conflicts;

2. Each temporary host does not have a priority level, which reduces the additional cost to a certain extent.

Disadvantages:

The bus control right is randomly determined. Even if the device that sends the data first does not have the control of the bus, it may cause important messages to be delayed.

6 Conclusions

The I2C bus is a simple two-wire synchronous serial bus. The physical layer only needs two buses, SDA and SCL. SDA is used to transmit data, SCL is used for synchronization for data transmission and reception; protocol layer stipulates data validity of communication, start and stop signals, response, data read and write sequence, address broadcast and so on. SDA and SCL have strict timing correspondence. If a violation occurs, the start and stop signals can be judged according to the order of the level change.

Aiming at the shortcomings of the slave passive response to host, an improved scheme is proposed based on the communication bus added from the physical layer, carrier sense /collision detection and backoff algorithm for the protocol layer, and the "low-level priority" arbitration principle .The improvement schemes of the physical layer and the protocol layer each have advantages and disadvantages, and the actual application can combine the two to complement each other.

References

[1] ELAN Microelectronics Co., Ltd. EM78P259N 8-bit OTP Microcontroller Product Specification [S]. Taiwan, Jan 2011

[2] ATMEL Corporation. Two-wire Automotive Temperature Serial EEPROM[S].USA,2007

[3] Xiao Debao. Computer Network. Huazhong University of Science and Technology Press,Oct.1st 2007

[4] Li Xuehai. Practical Guide to PIC Microcontrollers. Beijing Aerospace University Press, Jun 1st,2002

[5] Zhang Dong, Wu Chunming, Jiang Ming. A Survey of Consistent Algorithms for Resource Allocation in Distributed Systems. Zhejiang: Hangzhou University of Electronic Science and Technology, 2009, Issue 1. 37-40