### T5L1-CAN Parameter configuration description



# Introduction to CAN Protoco

CAN is the abbreviation of Controller Area Network. It is an ISO internationally standardized serial communication protocol. It has two standards after ISO standardization: ISO11898 and ISO11519-2. The former is a high-speed communication standard of 125Kbps~1Mbps, and the latter is a low-speed communication standard for the communication rate below 125Kbps.

The CAN bus adopts a multi-master competitive bus structure, featuring serial bus with multi-master operation and decentralized arbitration as well as broadcast communication.

The CAN bus adopts a multi-master competitive bus structure, featuring serial bus with multi-master operation and decentralized arbitration as well as broadcast communication. With non-destructive arbitration technology, when two nodes transmit data to the network at the same time, the node with lower priority will stop sending data voluntarily, while the node with higher priority will continue transmitting data without being affected, effectively avoiding bus conflicts, so any node on CAN bus can send information to other nodes on the network voluntarily at any time without prioritizing, and free communication can be realized among nodes.

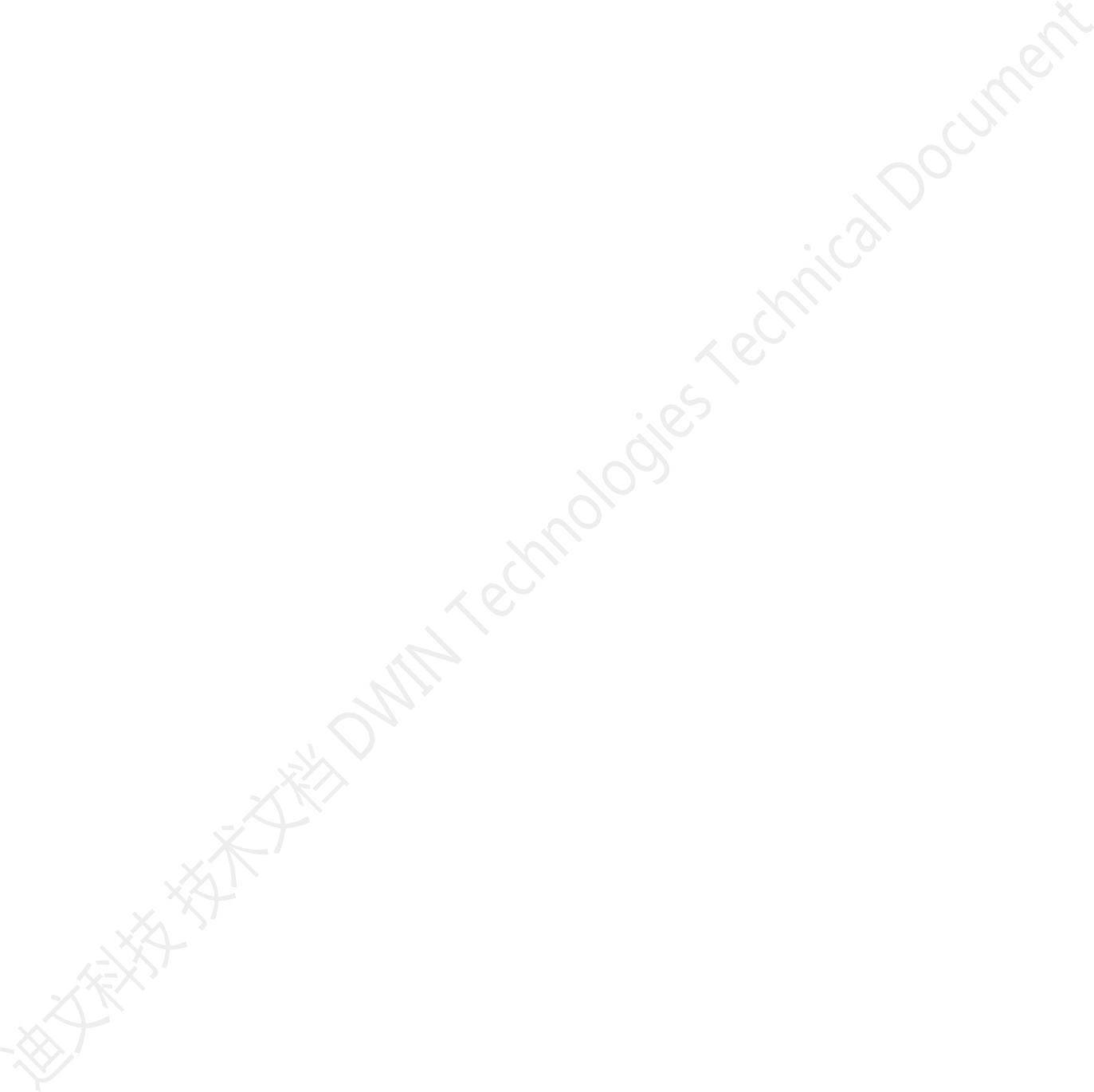
1. **CAN interface definition of T5L1**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **CPU** | **PIN#** | **Definition** | | | | | |
| **Function1** | **Description** | **Function2** | **Description** | **Function3** | **Description** |
| OS | 125 | P0.2 | I/O port | CAN\_TX | CAN interface data transmission |  |  |
| OS | 126 | P0.3 | I/O port | CAN\_TX | CAN interface data reception |  |  |

**3. CAN development steps of T5L1**



As with other serial protocol configurations of T5L1, it is only necessary to configure the SFR and related DGUS variables of the CAN interface.

* 1. ******SFR related to CAN interface**

|  |  |  |
| --- | --- | --- |
| **SFR** | **Address** | **Description** |
| MUX\_SEL | 0xC9 | .7 1=CAN interface leads to P0.2, P0.3, 0=CAN interface doesn’t lead out, it’s IO port. |
| CAN\_CR | 0x8F | CAN interface control register.  .7 1=CAN interface on 0=CAN interface off. Initially closed.  .6 CAN interface mode, 1=reset, 0=normal.  .5 Write 1 to configure the CAN interface configuration data (0xFF0060-0xFF0062) once, and clear it after hardware processing.  .4 Set the speed mode, 1=1 sampling, 0=3 sampling.  .3 Set filter mode, 1=double，0=single.  .2 Write 1 to start a transmission, and it will be cleared after hardware processing (successful transmission, arbitration failure, EI (CAN\_IR.3) error, software reset).  .1 overload control bit. 0=default value. 1=actively send an overload frame once.  .0 auto\_transmit. 0= the default value, if an error occurs, or the arbitration fails, the sending node will automatically resend;  1=If an error occurs, or the arbitration fails, the sending node will not automatically re-send, and software needs to be reconfigured to send. |
| CAN\_IR | 0x91 | CAN interface interrupt register.  .7 RF\_IF, remote frame receiving interrupt flag, set by hardware, it needs to be cleared.  .6 CAN\_RX\_IF, After CAN receiving the interrupt flag, it is set by hardware and needs to be cleared. During the set period, the hardware no longer updates the data.  .5 CAN\_TX\_IF, CAN send successfully interrupt flag, set by hardware, it needs to be cleared.  .4 OI, receive overflow flag, set by hardware, it needs to be cleared.  .3 EI, error flag, when an error occurs in CAN\_ET[4:0], this bit is set and needs to be cleared.  .2 JI, send arbitration flag, 1=failed, 0=succeeded.  .1-0 Undefined. |
| CAN\_ET | 0XE8 | The CAN interface error type register is set by hardware and needs to be cleared. Bit addressable.  .7 Node suspension flag.  .6 Active error identification.  .5 Passive error identification.  .4 CRC check error identification.  .3 Response error identification.  .2 Format error identification.  .1 Bit fill error flag.  .0 Bit error flag. |

The SFR related to the CAN interface is as shown in the above table, and the following parts of CAN\_CR and CAN\_IR need to be noted:

CAN\_CR register: CAN\_CR.6 Write 1 to indicate software reset. It needs to write 0 during normal operation. It defaults to 1 after power-on. If you need to use the CAN interface, you need to clear this bit. If you need software to reset CAN, you need to set this bit to 1 and then clear it to zero. The remaining bits are described in the table above.

CAN\_IR register: CAN\_IR.6 is set to 1 as long as there is correct received data, if the reception is a remote frame, CAN\_IR.7 is set to 1, otherwise it is 0. CAN\_IR.3 This bit needs to be used in conjunction with the CAN\_ET register. When a related error occurs, the software needs to control the related action. See the above table for the description of the remaining bits.

* 1. **DGUS variables related to CAN interface**

The DGUS variables related to the CAN interface are shown in the following table. Among them, 0xFF0060-0xFF0062 needs to be configured before CAN can be used. When configuring, you need to write data in advance and then set CAN\_CR.5 to 1. The introduction is as follows:

| Address | Site | Length | Definition | Description |
| --- | --- | --- | --- | --- |
| 0xFF:0060 | D3 | 1 | BRP | Baud rate divider register. |
| D2 | 1 | BTR0 | Bit Timing Register 0 [7:4] - Synchronous Jump Width SJW;  [3:0] TSEG2 Time Segment 2, configurable 0-7 according to the protocol |
| D1 | 1 | BTR1 | Bit Timing Register 1 [4:0]TSEG1 Time segment 1, including PROP SEG, PHASE SEG1, configurable according to protocol 0-F |
| D0 | 1 | DEBUG0 | The default value is 0. The user can configure it as 0, do not configure it casually |
| 0xFF:0061 | D3:D0 | 4 | ACR3:0 | Accept the code register. |
| 0xFF:0062 | D3:D0 | 4 | AMR3:0 | Accept the mask register. |
| 0xFF:0063 | D3 | 1 | RXERR | Receive error count register. |
| D2 | 1 | TXERR | Send error count register. |
| D1 | 1 | DEBUG1 | The default value is 0. The user can configure it as 0, do not configure it casually |
| D0 | 1 | Interrupt mask | The default value is 0.[7:2] corresponds to the interrupt mask bits of CAN\_IR. If this bit is 1,  then mask the interrupt source. |
| 0xFF:0064 | D3 | 1 | CAN\_TX\_BUFFER | [7] IDE, [6] RTR, [3:0] DLC, frame data length. |
| D2:D0 | 3 | Undefined | Write 0. |
| 0xFF:0065 | D3 | 1 | ID3 | [7:0] The first 8 bits of standard frame and extended frame ID. |
| D2 | 1 | ID2 | [7:5] Standard frame, the last 3 digits of ID, 11 digits in total; [7:0] The following 8 digits of extended frame ID. |
| D1 | 1 | ID1 | [7:0] The extended frame ID is followed by 8 bits, and the standard frame is invalid. |
| D0 | 1 | ID0 | [7:3] The last 5 digits of the extended frame ID, 29 digits in total, are invalid for the standard frame. |
| 0xFF:0066 | D3:D0 | 4 | Send data | DATA1-DATA4。 |
| 0xFF:0067 | D3:D0 | 4 | Send data | DATA5-DATA8。 |
| 0xFF:0068 | D3 | 1 | CAN\_RX\_BUFFER | [8] IDE, [6] RTR, [3:0] DLC, frame data length. |
| D2:D0 | 3 | Undefined | Invalid data. |
| 0xFF:0069 | D3 | 1 | ID3 | [7:0] The first 8 bits of standard frame and extended frame ID. |
| D2 | 1 | ID2 | [7:5] Standard frame, the last 3 digits of ID, 11 digits in total; [7:0] The following 8 digits of extended frame ID. |
| D1 | 1 | ID1 | [7:0] The extended frame ID is followed by 8 bits, and the standard frame is invalid. |
| D0 | 1 | ID0 | [7:3] The last 5 digits of the extended frame ID, 29 digits in total, are invalid for the standard frame. |
| 0xFF:006A | D3:D0 | 4 | Receive data | DATA1-DATA4。 |
| 0xFF:006B | D3:D0 | 4 | Receive data | DATA5-DATA8。 |

* + 1. **DGUS variable assignment**

In order to meet the baud rate required by the user, it is necessary to adjust BRP, SJW, TSEG1, TSEG2 reasonably, adjust the baud rate deviation to the minimum, and complete the allocation of the DGUS variable space.

* + - 1. Actual BRP calculation method

The actual BRP is obtained by calculating the target BRP.

Target BRP calculation method: target BRP=f/(SUM\*2\*band)-1, where: f, main frequency; SUM, time share; band, required baud rate band. For example, f=206438400HZ, SUM=25, band=150000HZ,

It can be calculated that the target BRP=26.52512. Actual BRP=ROUND(target BRP,0)=26.

* + - 1. Calculate the actual baud rate R

R = f/{[(BRP+1)\*2]\*[ Sync\_SEG + (TSeg1+1)+ (TSeg2+1)]} =f/[( BRP+1)\*2\*(TSeg1+TSeg2+3) ]

Among them: f, main frequency; BRP, actual baud rate; Sync\_SEG, synchronization segment, fixed at 1; TSeg1+TSeg2+3=sum. The actual baud rate R=152917HZ.

* + - 1. Calculate the baud rate deviation

Baud rate deviation=(actual baud rate/target baud rate-1)\*100%=1.9%.

Then, the baud rate deviation can be adjusted by adjusting the value of BRP and sum.

Notice:

* + - * 1. The sum of the time share is as large as possible, and the maximum value of 25 starts to decrease and try to find a suitable value；
        2. TSEG1 and TSEG2 need to be set according to the needs of the manufacturer. Manually adjust TSEG1 and TSEG2, you can change the sampling point position, you can consider the suggested sampling point at about 75%;
        3. SJW is configured according to the actual situation. You can try between [1,4], adjusting from small to large.

1. **CAN development C code example for T5L1**



* 1. **CUP initialization**

According to 3.1 initialization configuration in "Development Guide of T5L ASIC", following the default values, or setting instructions, the C51 reference code is as follows：

void T5LINIT()

{

EA = 0;

CKCON = 0; //CPU runs in 1T mode

T2CON = 0x70; //Timer T2 runs in Autoload mode

DPC = 0; //DPC fixed value

PAGESEL = 1; //64K code space

D\_PAGESEL = 2; //MOVX access RAM space settings

MUX\_SEL= 0x80; //Lead out the CAN interface to P0.2, P0.3

PORTDRV = 1; //Configure IO port drive capability

RAMMODE = 0; //Clear the DGUS variable interface register

}

* 1. **CAN initialization reference code**

Set in the code: the required baud rate is 150000HZ, the time share is 25, the main frequency is 206438400HZ, and the SJW is set to 1. Actual BRP=0x1A, BTR0=0x17, BTR1=0x0F, DBUG0=0.

The initialization reference code is as follows：

void CanInit()

{

P0MDOUT = 0x04; //P0.2(CAN\_TX) configure as output

P0 = 0xFF; //Output high leve

ADR\_H = 0xFF; //Configure DGUS variable memory address

ADR\_M = 0x00;

ADR\_L = 0x60;

ADR\_INC = 1; //Configure address increment

RAMMODE = 0x8F; //Write mode

while(!APP\_ACK); //Waiting for confirmation，

APP\_ACK Hardware response to 8051 occupied variable memory request, 1=OK, 0=BUSY, need to continue to wait.

DATA3 = 0x1A; //DGUS variable memory address 0xFF:0060 assignment.

DATA2 = 0x17; DATA1 = 0x0F; DATA0 = 0;

APP\_EN = 1;

while(APP\_EN); //Wait for the data operation to complete, and clear it after the operation is complete.

DATA3 = 0; //Acceptance register 0xFF:0061 assignment clear.

DATA2 = 0;

DATA1 = 0;

DATA0 = 0;

APP\_EN = 1;

while(APP\_EN); //Acceptance register 0xFF:0061 assignment clear.

DATA3 = 0xFF; //Acceptance mask registers 0xFF:0062 are all set to 1, and acceptance is not performed.



DATA2 = 0xFF;

DATA1 = 0xFF; DATA0 = 0xFF; APP\_EN = 1;

while(APP\_EN); //Wait for the data operation to complete, and clear it after the operation is complete.

RAMMODE = 0; //End access to DGUS variable memory

CAN\_CR = 0xA0; //Open CAN interface and configure FF0060-FF0062

while(CAN\_CR&0x20); //Execute configuration FF0060-FF0062 actions

ECAN = 1; //Turn on CAN interrupt

EA = 1; //Turn on total interrupt

}

## CAN send reference code

In the process of CAN interaction, refer to the CANopen protocol to send and receive data in the form of standard frames. The first 8 bits of the 11-bit ID are used as the device ID, and the last 3 bits are used as commands for data interaction. The reference code is as follows:

In the C language code, the sending sub-function void CAN\_tx(unsigned char type, unsigned char cmd) is as follows:

Among them, type represents the setting of IDE, RTR, DLC and other data; cmd is the command sent, 0 represents the control status data, 1 represents the main page display data; ID is a fixed value, represents the ID of the current device; the array status[8] is the sending data cache array.

void CAN\_tx(unsigned char type,unsigned char cmd)

{

EA = 0; //Make sure interrupts are turned off before sending data

ADR\_H = 0xFF; //Configure DGUS variable memory address

ADR\_M = 0x00;

ADR\_L = 0x64;

ADR\_INC = 1; //Configure address increment

RAMMODE = 0x8F; //Write operation

while(!APP\_ACK); //Waiting for confirmation，

APP\_ACK Hardware response to 8051 occupied variable memory request, 1=OK, 0=BUSY, need to continue to wait.

DATA3 = type; //DGUS variable memory 0xFF:0064 address CAN\_TX\_BUFFER configuration value

DATA2 = 0;

DATA1 = 0;

DATA0 = 0;

APP\_EN = 1;

while(APP\_EN); //Write RTR, IDE, DLC and other data in CAN\_TX\_BUFFER

DATA3 = ID; //DGUS variable memory 0xFF:0065 address stores the first 8 bits of the ID

DATA2 = cmd; //DGUS variable memory 0xFF:0065 3bit after address storage ID

DATA1 = 0;

DATA0 = 0;

APP\_EN = 1;

while(APP\_EN); //Write ID data

DATA3 = status[0]; //DGUS variable memory 0xFF:0066 address stores the first 4 bytes of transmit data

DATA2 = status[1]; DATA1 = status[2];

DATA0 = status[3]; APP\_EN = 1;



while(APP\_EN); //Write the first 4 bytes of the transmit data

DATA3 = status[4]; //DGUS variable memory 0xFF:0067 address stores the last 4 bytes of transmit data

DATA2 = status[5]; DATA1 = status[6]; DATA0 = status[7]; APP\_EN = 1;

while(APP\_EN); //Write the first 4 bytes of the transmit data

EA = 1; //Turn on the interrupt, and turn on the interrupt after the data writing of the DGUS variable memory is completed

RAMMODE = 0; //End access to DGUS variable memory

CAN\_CR |= 0x04; //Start sending

## CAN receive interrupt reference code

Like the UART serial port, when the main program processes the received data, it can be processed in the interrupt. When the receive interrupt flag is triggered, the interrupt service function is executed. If the interrupt flag is not triggered, other programs are executed to improve the efficiency of the CPU. The CAN interrupt entry is 0x004B, and the interrupt serial number is interrupt 9. The specific interrupt sub-function void CAN\_Isr() interrupt 9 is as follows, among them, rx\_buf[12] is the received data buffer area, and rx\_done is the reception success flag bit.

void CAN\_Isr() interrupt 9

{

EA = 0; //After entering the CAN interrupt service routine, turn off the total interrupt

if((CAN\_IR&0xC0) == 0xC0) //The remote frame reception interrupt flag is set and the reception complete flag is set

{

CAN\_IR &= 0x3F; //The remote frame reception interrupt flag is cleared and the reception complete flag is cleared

ADR\_H = 0xFF; //Configure DGUS variable memory address

ADR\_M = 0x00; ADR\_L = 0x69;

ADR\_INC = 1; //Configure address increment

RAMMODE = 0xAF; //Read operation

while(!APP\_ACK); //Waiting for confirmation，

APP\_ACK Hardware response to 8051 occupied variable memory request, 1=OK, 0=BUSY, need to continue to wait.

APP\_EN = 1;

while(APP\_EN); //receive data into the buffer

rx\_buf[0] = DATA3;

rx\_buf[1] = DATA2; rx\_buf[2] = DATA1; rx\_buf[3] = DATA0;

RAMMODE = 0; //End access to DGUS variable memory

rx\_done = 1; //Receive complete

}

else if((CAN\_IR&0x40) == 0x40) //Set the reception completion flag to 1

{

CAN\_IR &= 0xBF; //Receive complete flag bit cleared

ADR\_H = 0xFF;

ADR\_M = 0x00; ADR\_L = 0x69; ADR\_INC = 1;



RAMMODE = 0xAF; //Read operation

while(!APP\_ACK); APP\_EN = 1;

while(APP\_EN);

rx\_buf[0] = DATA; rx\_buf[1] = DATA2; rx\_buf[2] = DATA1; rx\_buf[3] = DATA0; APP\_EN = 1;

while(APP\_EN); rx\_buf[4] = DATA3; rx\_buf[5] = DATA2; rx\_buf[6] = DATA1; rx\_buf[7] = DATA0; APP\_EN = 1;

while(APP\_EN); rx\_buf[8] = DATA3; rx\_buf[9] = DATA2; rx\_buf[10] = DATA1; rx\_buf[11] = DATA0; RAMMODE = 0;

rx\_done = 2;

}

if((CAN\_IR&0x20) == 0x20) //If sending marker position 1

{

CAN\_IR &= 0xDF; //Clear the transmit frame flag bit

}

if((CAN\_IR&0x10) == 0x10) //If the receive overflows mark the bit as 1

{

CAN\_IR &= 0xEF; //Clear the receive overflow flag bit

}

if((CAN\_IR&0x08) == 0x08) //If wrong mark position 1

{

CAN\_IR &= 0xF7; //Clear error flag

CAN\_ET &= 0xE0; //Clear error flag

}

if((CAN\_IR&0x04) == 0x04) //If arbitration loss flag bit as 1

{

CAN\_IR &= 0xFB; //Clear the arbitration loss flag bit

CAN\_CR |= 0x04; //Restart sending

}

EA = 1;

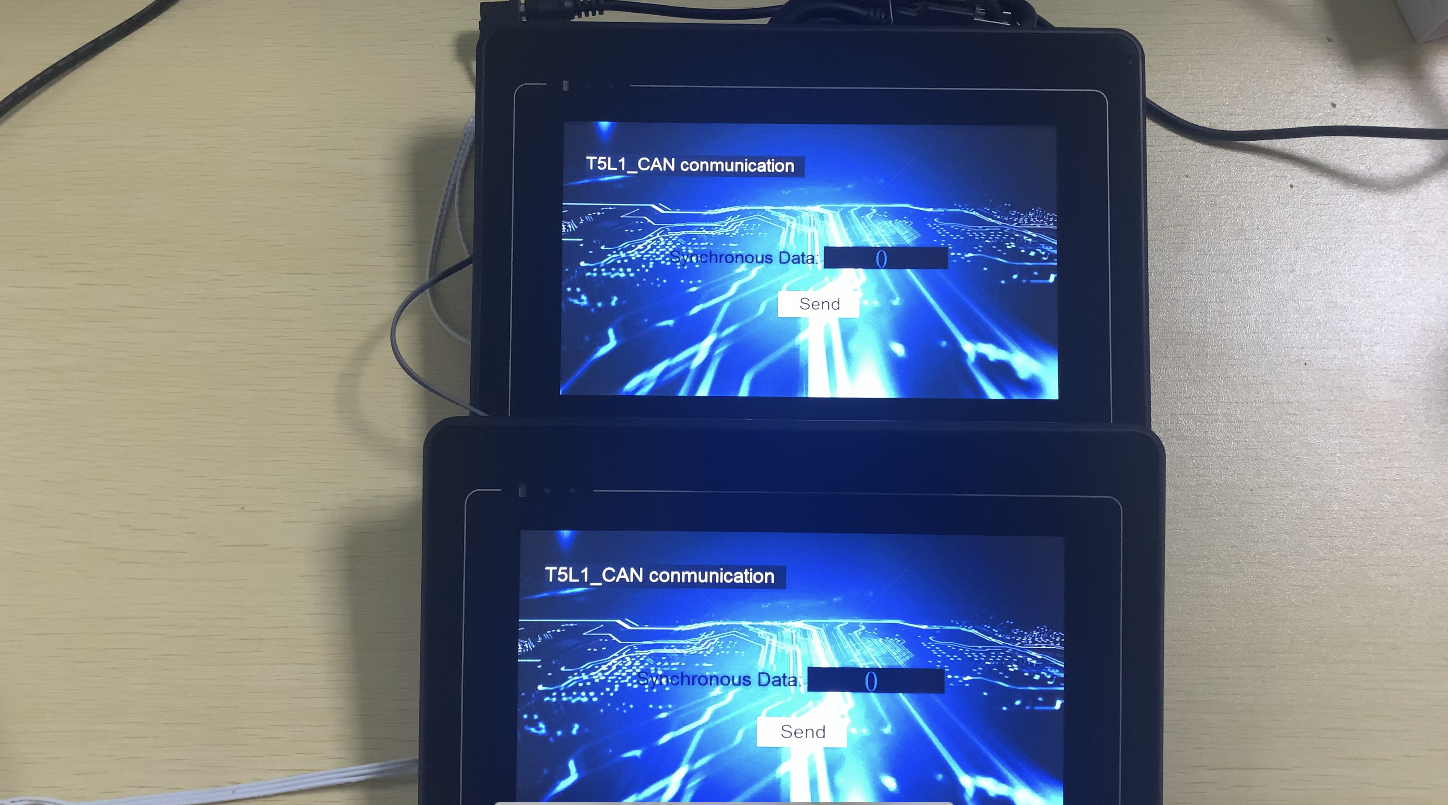
}

# DEMO description



The DEMO implements the function of synchronizing data entry on the T5L1 smart screen,

User-defined requirements can also be achieved by accessing DGUS, such as synchronous icon display, synchronous page cutting, etc.



Instructions, related demo routines, C source code and demonstration video are in the attachment.