make a figure

Robotics and Electronics

Gicren’s RS485 protocol (G485_V11)

Introduction

In order to regulate the RS485 communication protocol of **Gicren**’s product line, a set of all the devices that are connected to the same RS485 bus is called a RS485 system. It is called "***System-Master***"(the address of the *System-Master* is fixed to 127) that plays a leading and coordinating role in the RS485 system, while the other devices are called “***System-Slave***”(the address of the *System-Slave* can be set from 1 to 126). The *System-Master* needs to use the broadcast address(“0”), if all the *System-Slave* are required to respond to an instruction packet from it at the same time. Note that the broadcast instruction packet must be a *User-Register-Write* operation(the corresponding packet identifier is RPID) or a *Set* operation(the corresponding packet identifier is SPID). In other words, it is not allowed to get data from the *System-Slave* by broadcasting. In addition, if there is no special handling mechanism, the RS485 bus does not support the *Multi-Master* arbitration, so the *System-Master* is always the master on the RS485 bus. All the data transmissions are initiated by the *System-Master* and strictly comply with the *Master-Slave* structure.

All functions can be performed by reading or writing the user register, except for a few special operations(the corresponding packet identifiers are HPID, EPID and SPID). There are two different modes to write the user register: *Real-Time-Write* and *Non-Real-Time-Write*. If you need to update the current value of the user register to EEPROM, please send a *Set* instruction packet with 0x7c(PID value).

Real-Time-Write mode : It is the default write mode after power-on, which can be enabled at any time by a *Set* instruction packet with 0x79(PID value). The write operation will be performed immediately once the user register changed in this mode.
Non-Real-Time-Write mode : Even if the user register has been changed, it could not be performed before a *Set* instruction packet with 0x77(PID value) is correctly received. When some *System-Slave* need to be synchronized, configuring the user register in the *Non-Real-Time-Write* mode first, and then the *System-Master* broadcasts a *Set* instruction packet with 0x77(PID value). This write mode can be enabled at any time by a *Set* instruction packet with 0x78(PID value).

Convention

- A complete RS485 transmission is called a **Transaction**. It consists of one or more **Instruction Packets** which is composed of some **Fields**.
- **Output** : From the *System-Master* to the *System-Slave*.
  Input : From the *System-Slave* to the *System-Master*.
- **Parameters**:

Start Bit	–	1
Data Bit	–	8
Stop Bit	–	1
Parity Bit	–	None
Flow Control	–	No

- **W** : (0) RS485 write control bit.
- **R** : (1) RS485 read control bit.
- **SMA** : The *System-Master* address.
- **SSA** : The *System-Slave* address.
- **DI_x** : The input data.
- **DO_x** : The output data.
- **RCBN** : The number of bytes which will be continuously returned.
- **MDOC** : The check value of some fields(output) at the *System-Master* side.
- **MDIC** : The check value of some fields(input) at the *System-Master* side.
- **SDOC** : The check value of some fields(output) at the *System-Slave* side.
- **SDIC** : The check value of some fields(input) at the *System-Slave* side.
- **ERRH** : The higher byte of the *System-Slave*'s error word.
- **ERRL** : The lower byte of the *System-Slave*'s error word.
- **RPID** : It is the identifier of the *User-Register-Write*, *User-Register-Read-Request* or *User-Register-Read-Response* instruction packet. It indicates the first address of user registers which will be read or written continuously. The range is from 1 to 100.
- **HPID** : It is the identifier of the Handshake instruction packet, which is fixed to 0x7f. As a *System-Master*, strictly

speaking, it is necessary to confirm that not only all the output data have been correctly received by the *System-Slave*, but also all the input data have been correctly received by itself. If an abnormal transmission is detected, the *System-Master* can be programmed whether to re-establish a new transmission or not. However, the *System-Slave* only needs to respond to the correct transmission. The Handshake instruction packet is used to get the check byte from the *System-Slave* to verify the instruction packet has been correctly transmitted(or received) or not. According to the previous instruction packet, there are two cases as follows:

1. If the previous instruction packet is a *User-Register-Write* or a *Set* operation, some fields(see below) are checked by the *System-Master* during data outputting and this check byte is **MDOC**. In the meantime, the *System-Slave* checks the same data and this check byte is **SDOC**, which can be returned by a Handshake instruction packet. If the *System-Master* finds that MDOC is not equal to SDOC, it indicates current transmission has failed. Whether a new transmission will be established or not is according to the user code.
2. If the previous instruction packet is a *User-Register-Read-Response* or a *Get-Error-Word-Response* operation, some fields(see below) are checked by the *System-Slave* during data inputting and this check byte is **SDIC**, which can be returned by a Handshake instruction packet. In the meantime, the *System-Master* checks the same data and this check byte is **MDIC**. If the *System-Master* finds that MDIC is not equal to SDIC, it indicates current transmission has failed. Whether a new transmission will be established or not is according to the user code.

- **EPID** : It is the identifier(fixed to 0x7e) of the *Get-Error-Word-Request* or *Get-Error-Word-Response* instruction packet.
- **SPID** : It is the identifier of the *Set* instruction packet. There are 10 sub-types as follows:
  1. 0x7d : Restore factory settings.
  2. 0x7c : Update the current value of the user register to EEPROM.
  3. 0x7b : Reset the device.
  4. 0x7a : Clear the error word.
  5. 0x79 : Enable the *Real-Time-Write* mode.
  6. 0x78 : Enable the *Non-Real-Time-Write* mode.
  7. 0x77 : Perform the *Non-Real-Time-Write* operation.
  8. 0x76 : Enable the control transmission mode which is the default mode after power-on. In this mode, theoretically, any valid instruction packet should never be discarded.
  9. 0x75 : Enable the bulk transmission mode, when there is a function which the priority is higher than serial interruption. In this mode, the serial data would be selectively discarded.

10. 0x74 : The baud rate will not be detected automatically, keeping the last adaptive value. You can restore the function of automatic baud rate detection by restoring it to factory settings, please refer to the device's user manual for details.

Field

Field is the minimum unit of an instruction packet. There are 6 types of fields as follows:

- **Start field** : 2 bytes(0xff + 0xff), the start data of an instruction packet.
- **Address field** : 1 byte, it consists of 7 address bits and a read/write control bit ((SSA<<1)+W/R or (SMA<<1)+R). They are simplified as "SSA_W", "SSA_R" and "SMA_R" below.
- **Length field** : 1 byte, the number of bytes in the instruction packet(after this field).
- **PID field** : 1 byte, there are 4 main-types, such as RPID, HPID, EPID and SPID.
- **Data field** : It consists of one or more bytes.
- **Check field** : 1 byte, see below for details. Note that "~" is the Not Bit operator. For example, ~0x55 = 0xaa.

Packet

Packet is made up of several fields and is also the unit of a transaction. There are 8 types of instruction packets as follows:

Background color	Description
	From the *System-Master*
	From the *System-Slave*

- ***User-Register-Write***

Start	Address	Length	PID	Data			Check
0xff 0xff	SSA_W	x+2	RPID	DO_1	……	DO_x	MDOC

MDOC=~(LowerByte(SSA_W+(x+2)+RPID+DO_1+……+DO_x))

- ***User-Register-Read-Request***

Start	Address	Length	PID	Data	Check
0xff 0xff	SSA_R	0x03	RPID	RCBN	MDOC

MDOC= ~(LowerByte(SSA_R+0x03+RPID+RCBN))

- ***User-Register-Read-Response***

Start	Address	Length	PID	Data			Check
0xff 0xff	SMA_R	x+2	RPID	DI_1	……	DI_x	SDIC

SDIC= ~(LowerByte(SMA_R+(x+2)+RPID+DI_1+.....+DI_x))

- ***Handshake-Request***

Start	Address	Length	PID	Check
0xff 0xff	SSA_R	0x02	HPID	MDOC

MDOC= ~(LowerByte(SSA_R+0x02+HPID))

- ***Handshake-Response***

Start	Address	Length	PID	Data	Check
0xff 0xff	SMA_R	0x03	HPID	SDOC/SDIC	SDIC

1. SDIC= ~(LowerByte(SMA_R+0x03+HPID+SDOC)) (The previous instruction packet is a *User-Register-Write* operation)
2. SDIC= ~(LowerByte(SMA_R+0x03+HPID+SDIC)) (The previous instruction packet is a *User-Register-Read-Response* or a *Get-Error-Word-Response* operation)

Note that the “SDIC” in the check field is different from that in the data field because of the different data source.

- ***Get-Error-Word-Request***

Start	Address	Length	PID	Check
0xff 0xff	SSA_R	0x02	EPID	MDOC

MDOC= ~(LowerByte(SSA_R+0x02+EPID))

- ***Get-Error-Word-Response***

Start	Address	Length	PID	Data		Check
0xff 0xff	SMA_R	0x04	EPID	ERRH	ERRL	SDIC

SDIC= ~(LowerByte(SMA_R+0x04+EPID+ERRH+ERRL))

- ***Set***

Start	Address	Length	PID	Check
0xff 0xff	SSA_W	0x02	SPID	MDOC

MDOC= ~(LowerByte(SSA_W+0x02+SPID))

Transaction

A transaction consists of one or more instruction packets, which is called a complete transmission. There are 4 types of transactions as follows:

- ***User-Register-Read*** **transaction**

  "*User-Register-Read-Request*" + "*User-Register-Read-Response*" + "*Check Operation*"(it can be omitted for less critical applications)

  The check operation can be realized via two methods as follows:

  1. Handshake: "*Handshake-Request*" + "*Handshake-Response*". It is suggested to add a time-out handling after *Handshake-Request*.
  2. Reread the current value of the user register for comparison.

- ***User-Register-Write*** **transaction**

  "*User-Register-Write*" + "*Check Operation*"(it can be omitted for less critical applications)

  The check operation can be realized via two methods as follows:

  1. Handshake: "*Handshake-Request*" + "*Handshake-Response*". It is suggested to add a time-out handling after *Handshake-Request*.

2. Read the current value of the user register for comparison

- ***Get-Error-Word* transaction**

  "*Get-Error-Word-Request*" + "*Get-Error-Word-Response*" + "*Check Operation*"(it can be omitted for less critical applications)

  The check operation can be realized via two methods as follows:

  1. Handshake: "*Handshake-Request*" + "*Handshake-Response*". It is suggested to add a time-out handling after *Handshake-Request*.
  2. Reread the error word for comparison.

- ***Set* transaction**

  "*Set*" + "*Check Operation*"(it can be insteaded by sending the *Set* instruction packet for several(3 is recommended) times in less critical applications.)

  It is different from the other three transactions, because there is only one method to confirm that whether the *Set* instruction packet has been correctly received or not by the *System-Slave*. At this time, we must think over the reliability of the Handshake operation. Is it really flawless? No, the problem is that the check byte exists before the *System-Slave* receives a *Set* instruction packet, it just be synchronously updated during data outputting. For example, if the *Set* instruction packet has not been correctly received by the *System-Slave*, however, the current check byte is exactly equal to the check value of the *Set* instruction packet and returned by the Handshake instruction packet, the *System-Master* will mistake that the *Set* instruction packet has been correctly received. Because the **Gicren**'s product line is mainly designed for the robotic fan, with simplicity as the motive, so we do not want to design too complex communication protocol. About the reliability of Handshake, we can do the *User-Register-Read* operation(Note that this operation will probably more than once. You should change the user register address to be read until the check byte of the *User-Register-Read-Response* instruction packet is different from that of the *Set* instruction packet to be sent) before sending the *Set* instruction packet.

User Register Definition

The User register consists of the common register and the private register.

	Address	Name	Store	Access	Initial Value	Description
Common Register (For each device)	1 (0x01)	ProtocolVer	ROM	R	*	Protocol Version
	2 (0x02)	DeviceMainClass	ROM	R	*	Device Main Class
	3 (0x03)	DeviceSubClass	ROM	R	*	Device Sub Class
	4 (0x04)	HardwareVer	ROM	R	*	Hardware Version
	5 (0x05)	SoftwareVer	ROM	R	*	Software Version
	6 (0x06)	Command	RAM	W	0x00	Command Byte
	7 (0x07)	ID	ROM	R/W	0x01	Device Address (1~126)
Private Register (Depend on device)	8 (0x08)	*	*	*	*	*
	……	*	*	*	*	*
	100 (0x64)	*	*	*	*	*