make a figure

Robotics and Electronics

Gicren's I2C protocol (GI2C_V11)

Introduction

In order to regulate the I2C communication protocol of **Gicren**'s product line, a set of all the devices that are connected to the same I2C bus is called an I2C 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 I2C 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, although I2C is a true *Multi-Master* bus, given the characteristics of Gicren's product line, the *System-Master* is always the master on the I2C bus. All the data transmissions are initiated by the *System-Master* and strictly comply with the *Master-Slave* structure. An automatic bidirectional level conversion circuit is embedded in the device, when using the I2C interface, you just make sure the appropriate pull-up resistors have been connected to the SDA and SCL of the *System-Master*(4.7KΩ to +5V is recommended).

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 0xf8(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 0xf2(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 0xef(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 0xef(PID value). This write mode can be enabled at any time by a *Set* instruction packet with 0xf1(PID value).

Convention

- A complete I2C 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*.
- **S** : I2C start signal.
- **P** : I2C stop signal.
- **A** : I2C ACK signal.
- **NA** : I2C NACK signal.
- **W** : (0) I2C write control bit.
- **R** : (1) I2C read control bit.
- **SMA** : The *System-Master* address.
- **SSA** : The *System-Slave* address.
- **DI_x** : The input data.
- **DO_x** : The output data.
- **MDOC** : The check value of data fields(output) at the *System-Master* side.
- **MDIC** : The check value of data fields(input) at the *System-Master* side.
- **SDOC** : The check value of data fields(output) at the *System-Slave* side.
- **SDIC** : The check value of data 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-Read* or *User-Register-Write* instruction packet. It indicates the first address of user registers which will be read or written continuously. The range is from 1 to 100 by shifting right 1 bit. The higher 7 bits are available, the LSB is an odd check bit of the higher 7 bits for checking the byte itself. For example, "0b10101011" indicates that the first address is "0b10101011>>1=0b01010101=85".
- **HPID** : It is the identifier of the Handshake instruction packet, which is fixed to 0b11111110(0xfe). 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* operation, all the data fields 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* operation, all the data fields 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 0b11111101(0xfd)) of the *Get-Error-Word* instruction packet.
- **SPID** : It is the identifier of the *Set* instruction packet. There are 7 sub-types as follows:

1. 0b11111011(0xfb) : Restore factory settings.
2. 0b11111000(0xf8) : Update the current value of the user register to EEPROM.
3. 0b11110111(0xf7) : Reset the device.
4. 0b11110100(0xf4) : Clear the error word.
5. 0b11110010(0xf2) : Enable the *Real-Time-Write* mode.
6. 0b11110001(0xf1) : Enable the *Non-Real-Time-Write* mode.
7. 0b11101111(0xef) : Perform the *Non-Real-Time-Write* operation.

Field

Field is the minimum unit of an instruction packet. There are 7 types of fields as follows:

- **Start field** : 1 bit, the start signal of an instruction packet.
- **Stop field** : 1 bit, the stop signal of an instruction packet.
- **Acknowledge field** : 1 bit, A(ACK) or NA(NACK).
- **Address field** : 1 byte, it consists of 7 address bits and a read/write control bit ((SSA<<1)+W/R). They are simplified as "SSA_W" and "SSA_R" below.
- **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, ~(LowerByte(DO_1+......+DO_x)) or ~(LowerByte(DI_1+......+DI_x)). Note that "~" is the Not Bit operator. For example, x =3, DO_1=0x55, DO_2=0x66, DO_3=0x77, and the check byte is ~(LowerByte(0x55+0x66+0x77)) =~0x32=0xcd.

Packet

Packet is made up of several fields and is also the unit of a transaction. There are 5 types of instruction packets as follows:

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

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

Start	Address	Acknowledge	PID	Acknowledge	Data + Acknowledge					Check	Acknowledge	Stop
S	SSA_W	A	RPID	A	DO_1	A	……	DO_x	A	MDOC	A	P

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

Start	Address	Acknowledge	PID	Acknowledge	Stop	Start	Address	Acknowledge	Data + Acknowledge					Stop
S	SSA_W	A	RPID	A	P	S	SSA_R	A	DI_1	A	……	DI_x	NA	P

- ***Handshake***

Start	Address	Acknowledge	PID	Acknowledge	Stop	Start	Address	Acknowledge	Data	Acknowledge	Stop
S	SSA_W	A	HPID	A	P	S	SSA_R	A	SDOC/SDIC	NA	P

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

Start	Address	Acknowledge	PID	Acknowledge	Stop	Start	Address	Acknowledge	Data + Acknowledge				Stop
S	SSA_W	A	EPID	A	P	S	SSA_R	A	ERRH	A	ERRL	NA	P

- ***Set***

Start	Address	Acknowledge	PID	Acknowledge	Stop
S	SSA_W	A	SPID	A	P

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*" + "*Check Operation*" (it can be omitted for less critical applications)

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

  1. Handshake:
  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:
  2. Read the current value of the user register for comparison.

- ***Get-Error-Word* transaction**
  "*Get-Error-Word*" + "*Check Operation*" (it can be omitted for less critical applications)
  There is only one method to realize the check operation, that is, reread the error word for comparison.

- ***Set* transaction**
  "*Set*"
  Because of the inherent response mechanism of the I2C bus, the *System-Master* can detect whether the PID field of the *Set* instruction packet has been transmitted or not, however, it is uncertain whether it is correctly received or not. Repeatedly sending the *Set* instruction packet for several times, which is an effective solution.

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)	*	*	*	*	*