OTG15X Series I²C Software User Manual V2.3

◆ General Rules

OTG15X generally means processor used in OTG15 series modules, user mainframe generally means MCU developed by user and communicates with OTG15X through I²C bus.

If 16-bit parameter is composed of two bytes, the first byte is low bit and the second byte is high bit.

If 32-bit parameter is composed of four bytes, the first byte is low bit and the fourth byte is high bit.

0xnn means the described value is uncertain, it may be any values, but its value is within range fixed originally, for instance, instruction length is 2 to 137.

B7 means the seventh bit of byte, B6 means the sixth bit of byte, and the rest may be deduced by analogy.

I²C address that user mainframe writes OTG15X is 0x5e, i.e. 01011110B. I²C address that user mainframe reads OTG15X is 0x5f, i.e. 01011111B.

The length of register is 8 bits in general, and user mainframe only needs one byte for read and write. Another registers that mark byte length need several bytes for read and write, should do multibyte read and write according to need.

In addition to I²C's SCL and SDA ports, OTG15X provides a INT output port extra, which is used to inform of user mainframe after OTG15X's internal state changes.

INT is high level under normal work, it is raised by pull-up resistor in OTG15X, and can be lowered by external circuit, but cannot be raised by external circuit, and interface is the same as the interface of 8051 single chip.

When applicable interrupt appears in OTG15X, OTG15X lowers INT. OTG15X will raise INT after user mainframe eliminates corresponding interrupt state.

User mainframe can detect INT with low level at any time, and read INTRD register after detecting INT with low level. INTRD is 16-bit register, and each bit means a kind of interrupt, there are 16 kinds of interrupts in all from INT0 to INT15. Every interrupt can be set or cleared by INTEN register. It is able to set corresponding interrupt number to clear generated interrupt after reading interrupt.

Interrupt mechanism just provides a fast processing mode, and user mainframe can read corresponding register according to interrupt number, which can improve the using time of I²C bus very much.

There is the register of "INTx" behind read-only or read-write register, it's better to read register after user mainframe reads corresponding interrupt in order to get most timely information; if read register without generating interrupt, generally read information not updated last time.

Hard & Soft Technology Co., LTD.
http://www.HSAV.com
Address: second floor, No.199, Longyin 2nd Road, Xixiang, Shenzhen, China
TEL: 86-0755-27951479 27950879 FAX: 86-0755-27950879-213
Technology support: support@HSAV.com Business contact: Sales@HSAV.com

◆ OTG15X System Setup Register

Address	Name	Description
0x00	INTCLR	Clear interrupt register (write only) Interrupt number instruction: 16-bit interrupt number is consist of byte 0 and byte 1, B0 of byte 0 is INT0, B7 of byte 1 is INT15, and the rest may be deduced by analogy. INT0=1, OTG15X initialization. INT1=1, USB/CF card interface plug-pull interrupt, need to read “DISKSTA” register. INT2=1, file structure scan is finished in USB/CF, it is okay to read related information. INT3=1, file information finishes, it is okay to read “FILEINFO” register, filename, or ID3. INT4=1, playing time change. INT5=1, stop playing songs. INT6=1, cannot find appointed file. INT7=1, cannot play files. INT8=1, stop recoding. INT9=1, U disk/CF card’s available space is full/write protection. INT10=0, reserved. INT11=1, mark finishing character library read, it is okay to read related information. INT12=1, real time clock change. INT13=0, reserved. INT14=1, OTG15X, computer, and network communication receive interrupt. INT15=1, additional function interrupt.
0x01	INTRD	Read interrupt register (read only) Interrupt number and clear interrupt register are the same. **Attention: please clear corresponding register number after read, or interrupt will go on without stop. Clear corresponding interrupt through writing 1.**
0x02	INTENA	Interrupt enable register (write only) Set corresponding interrupt enable, OTG15X will generate corresponding interrupt and lowers INT pin when state changes, user mainframe need to check INT pin, read interrupt value and do corresponding processing. Interrupt number and read/write interrupt register are corresponding, enable corresponding interrupt when corresponding bit is 1, and disable corresponding interrupt when corresponding bit is 0.
0x03	DISKSTA	Interface state register (read only/INT1) B0=1, USB interface has plug or pull action. B5=1, CF card has plug or pull action. B6=1, USB device and PC have plug or pull action. B7=1 is interface plug type, B7=0 is interface pull type.
0x09	APPSTA	Additional function register (read only/INT15) B0=1, OTG15X temperature change. B1 to B7 are reserved, and value is 0.
0x0c	SETRTC	System real time clock setup (write only) Byte 0 to byte 6 is respectively second/minute/hour/week/day/month/year.
0x0d	READRTC	System real time clock read (write only/INT12) Byte 0 to byte 6 is respectively second/minute/hour/week/day/month/year.
0x0f	SYSVER	System version read (read only) Byte 0 to byte 3 is respectively hour/day/month/year, i.e. the date of system date. Byte 4 is OTG15X module hardware code: =0x15is OTG15E, =0x16 is OTG15F, and =0x18 is OTG15H

◆ OTG15X Play/Record Register

Address	Name	Description
0x10	DISKSEL	Drive letter selection register (write only) =0x00, use USB interface. =0x05, use CF card interface. Attention: this register is effective for all operation, it is necessary to select it in advance.
0x12	PLAYMODE	Play, pause, stop, record, stop recording mode selection (write only) =0x00, stop mode. =0x01, playing mode. =0x02, pause mode. =0x03, recording mode. =0x04, recording pauses. =0x05, stop recording.
0x14	SPECPLAY	Select songs to play. (write only) (note: arrange the sequence of directory and songs according to date that creates directory and songs) Byte 0 to byte 1 is song number. Byte 2 to byte 3 is directory number.
0x16	AUDMODE	Sound and channel mode selection (write only) =0x00, mute when stop mode. =0x01, select ADC sound when stop mode. =0x02, select internal FLASH when stop mode. =0xff, mute.
0x18	RECMODE	Record format setup (write only) For byte 0 to bye 3, please refer to appendix “Record format setup”.
0x1a	PLAYLIST	List play (read/write) I^2C transmits list station number that user selects in Playlist.txt file under CF card directory; register records list number that user select to play (note: Playlist.txt file content should be written according to prescribed format, and list play file is placed in AUDIO folder). List format is as follows: [1] station NO 1= 1 // Station quantity. list NO 1=2 // The quantity of speech list. playfile 1= 0001.mp3+000a.mp3 // Current station song name that is consist of 4 letters or digits.
0x1b	NEWFOLDER	New folder register (write only) Directly send folder name to create a new folder. Filename supports 8 bytes at most.

◆ OTG15X File Information Register

Address	Name	Description
0x20	NEWFILE	New folder register (write only) Create a new file in U disk. =0x00, new file, default filename is NEW_XX.15X, and XX is from 01 to 99. =0x01, close and save new file.
0x21	TRACKTOL	Songs quantity in U disk (read only/INT3)
0x22	COPYFILE	Copyfile register (write only)
0x23	DIRTOL	Directories quantity in U disk (read only/INT3)
0x24	DELFILE	Register deleting file (write only) Delete appointed file, 16-bit register, high 8 bits is directory number, and low 8 bits is filename.
0x25	DIRTRACK	Songs quantity under current directory (read only)
0x27	PLAYTIME	Play time (read only/INT4) Note: time playing songs.
0x29	FILEINFO	File information that is playing (read only/INT4) Byte 0 to byte 1 is song number. Byte 2 to byte 3 is directory number. Byte 4 to byte 5 is total time playing songs. Byte 6 is song type. Byte 7 is song code stream rate. Byte 8 is song sampling rate. Divide total time playing song by 60, if you get integer, it is minute, and the rest is second. Byte 6 file type=0x01, it is MP3 file; =0x02, it is WMA file. For byte 7 code stream rate, if file type is MP3, multiplying 8 by the value of byte 7 is the code stream rate of song; if file type is WMA, the value of byte 7 is the same as code stream rate. Byte 8 sampling rate=0x08, it is 8K; =0x0b, it is 11.025KHz; =0x0c, it is 12K; =0x10, it is 16K; =0x16, it is 22.05KHz; =0x18, it is 24K; =0x20, it is 32K; =0x2c, it is 44.1KHz; =0x30, it is 48K.
0x2b	FILENAME	Song filename internal code, it is double-byte character string and ends in 0x0000 (read only/INT4)
0x2d	ID3NAME	Song ID3 internal code, it is double-byte character string and ends in 0x0000 (read only/INT4)

◆ OTG15X File Edit Register

Address	Name	Description
0x30	FONTTYPE	Character font format selection (write only) Byte 0 to byte 1 is limiting character font buffer byte length, and 0x0000 means there is no limit. Byte 2 is character font format selection. For more information, please refer to appendix 2 “Character font format instructions”. Byte 3 is character font internal code selection, 0 is GBK, and 1 is UNICODE. Byte 4 is font selection, 0 is AUTO, 1 is simplified Chinese, and 2 is traditional Chinese. Write when register received INTRD-> system abnormity-> OTG15X initialization.
0x31	FONTLENG	Effective character font total byte length (read only/INT11) Byte 0 to byte 1 is effective character font total length, may read after “GETFONT” induces INTRW to produce interrupt.
0x32	GETFONT	Get appointed character string’ character font (write only) Byte 0 is the path selection of input character string. When byte 0 is 0xff: Appoint offset position from FONTBUFF character font that has been gotten. Byte 1to byte 2 is offset position. When byte 0 is 0xfe: Byte 1 is the length of input internal code character string. Byte 2 to byte n is input internal code character string, and user mainframe can input any characters. When byte 0 is 0xfc: Use current song filename internal code as input internal code character string. When byte 0 is 0xfb: Use current song ID3 internal as input internal code character string. When byte 0 is less than 0x80, the same as “interface selection register directory number”: Byte 1 to byte 2 is song number. Byte 3 to byte 4 is directory number. Attention: at this moment, it is possible to induce INTRW to generate interrupt, and user mainframe must wait for “character font read finish mark” and then read “FONTBUFF register”.
0x33	FONTBUFF	Character font buffer (read only/INT11) Byte 0 to byte n-1 is character font data. Byte n is check sum, and its value is the sum adding byte 0 to byte n-1.
0x36	T9INPUT	Chinese input method register (write only)

Appendix 1:

1 Record Format Setup

Byte 0 is to appoint recording format, default is MP3 format 128Kbps/44.1KHz/Stereo.

Byte 1 is to appoint recording stop mode, 0 means that stop when recording length is over, and 1 means that filename number and 1 and continue recording after recording length is over.

Byte 2 to byte 3 is to appoint the length of recording file, unit is MB, 0x0000 is default 50MB, and 0x0001 is 1MB.

Use MP3 compression recording format and extension is *. MP3.	
0x00: 320kbps/44.1kHz/Stereo 0x01: 320kbps/48kHz/Stereo 0x02: 256kbps/44.1kHz/Stereo 0x03: 256kbps/48kHz/Stereo 0x04: 192kbps/44.1kHz/Stereo 0x05: 192kbps/48kHz/Stereo 0x06: 192kbps/32kHz/Stereo 0x07: 160kbps/44.1kHz/Stereo 0x08: 160kbps/48kHz/Stereo 0x09: 160kbps/32kHz/Stereo 0x0a: 128kbps/44.1kHz/Stereo 0x0b: 128kbps/32kHz/Stereo 0x0c: 96kbps/44.1kHz/Dual channel 0x0d: 96kbps/32kHz/Dual channel 0x0e:96kbps/44.1kHz/Single channel 0x0f: 96kbps/48kHz/Single channel 0x10: 64kbps/44.1kHz/Single channel 0x11: 48kbps/44.1kHz/Single channel 0x12: 48kbps/32kHz/Single channel	0x13: 64kbps/22.05kHz/Dual channel 0x14: 64kbps/24kHz/Dual channel 0x15: 48kbps/22.05kHz/Dual channel 0x16: 48kbps/16kHz/Dual channel 0x17: 32kbps/16kHz/Dual channel 0x18: 24kbps/11.025kHz/Dual channel 0x19: 48kbps/22.05kHz/Single channel 0x1a: 48kbps/24kHz/Single channel 0x1b: 32kbps/22.05kHz/Single channel 0x1c: 32kbps/24kHz/Single channel 0x1d: 24kbps/16kHz/Single channel 0x1e: 16kbps/16kHz/Single channel 0x1f: 16kbps/11.025kHz/Single channel 0x20: VBR 256kbps/44.1kHz/Stereo/VBR 0x21: VBR 192kbps/44.1kHz/Stereo/VBR 0x22: VBR 192kbps/48kHz/Stereo/VBR 0x23: VBR 160kbps/44.1kHz/Stereo/VBR 0x24: VBR 160kbps/32kHz/Stereo/VBR 0x25: VBR 128kbps/44.1kHz/Stereo/VBR 0x26: VBR 96kbps/44.1kHz/Dual channel/VBR
Use WAVE nonloss compression recording format and extension is *.WAV.	
0x30: 44.1kHz/Stereo 0x32: 22.05kHz/Stereo 0x34: 11.025kHz/Stereo	0x31: 44.1kHz/Stereo 0x33: 22.05kHz/Stereo 0x35: 11.025kHz/Stereo

Attention:

1) The larger the length of recording file, the longer time that recording initialization needs.
2) 1MB = 1,048,576 bytes.
3) If recording format is 128Kbps/44.1KHz/Stereo, it is able to record for one hour (bit rate/8 is byte quantity that this bit rate records for one second, and unit is KB).
4) Dual-channel is two irrelevant channels.
5) VBR is variable bit rate recording format.
6) Default is 128kbps/44.1kHz/Stereo。
7) There is need to appoint corresponding code stream and sampling rate before recording, or record according to default.

Appendix 2:

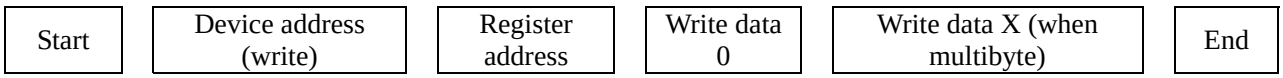
2 Character Font Format Instruction and Usage

OTG15X supports any character font arrangement mode.

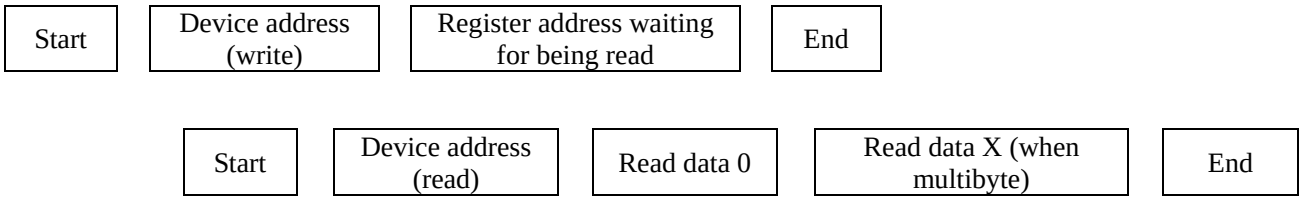
Appendix 3:

3 OTG15X Read and Write Register

OTG15X write register diagram：



OTG15X read register diagram：



First use the device address of write to write register address waiting for being read, and then use the device address of read to read corresponding data.

Need to receive the ninth ACK bit when write each byte including data and address for I^2C, ACK bit is 0 outputted by OTG15X. User mainframe can know if OTG15X is working normally according to ACK.

Need to send the ninth ACK bit when read each byte for I^2C, ACK bit is 0 outputted by user mainframe, but the last byte needs to send the ninth NAK bit, NAK bit is 1 outputted by user mainframe.

Appendix 4:

4 Source Code Instructions that OTG15X Uses IO Port to Simulate I^2C Time Order

```
void main(){
    BYTE gLocal_1;
    // Another initialization;
    // I2C bus and INT initialization
    while (1){
        if (pMED_INT){
            gLocal_1 = MMED_ReadByte(cADD_INTRD);          // read OTG15X interrupt
            if (gLocal_1 & 0x01){                          // B0 system is not normal
                MMED_WriteByte(cADD_INTCLR, 0x01);         //   clear corresponding interrupt
                // another processing
            }
            else if (gLocal_1 & 0x02){                     // B1 and another processing
                // another corresponding processing
            }
```

```
            }
          // another processing
      }
}

// the following functions can be used in all I2C bus
void MMED_WriteByte(BYTE gLocal_1, BYTE gLocal_2){      // write single-byte register
      MMEDStart();                                          // I2C start
      MMEDWrite(0x32);                                      // write device address write address
      MMEDWrite(gLocal_1);                                  // write register address
      MMEDWrite(gLocal_2);                                  // write register value
      MMEDStop();                                           // I2C stop
     return;
}

BYTE MMED_Read_BYTE(BYTE gLocal_1){                         // read 8-bit register
      MMEDStart();                                          // I2C start
      MMEDWrite(0x5e);                                      // write device address write address
      MMEDWrite(gLocal_1);                                  // write register address
      MMEDStop();                 // this moment OTG15X prepares for corresponding date used to read

      MMEDStart();                                          // I2C start
      MMEDWrite(0x5e+1);                                    // write device address write address
      gLocal_1 = MMEDRead(1);       // read register ACK bit, stop reading if NAK is 1
      MMEDStop();                                           // I2C stop
     return gLocal_1;                                       // respond register value
}

BOOL MMEDWrite(BYTE gLocal_1){              // I2C writes byte，gLocal is data waiting for being writen
      BOOL FLocal_1;                        // return ACK/NAK marks
      BYTE gLocal_2;                        // bit counter

      gLocal_2 = 8;                         // 8 bits, high bit first outputs
      do {
            pMED_SCL(0);                    // lower I2C clock wire so as to change data
            if (gLocal_1 & 0x80){           // if data bit is 1
                  pMED_SDA(1);              //raise I2C data wire
            }
            else {                          // if data bit is 0
                  pMED_SDA(0);              // lower I2C data wire
            }
```

```
        MUSDELAY(5);                        // delay 5ms
        gLocal_1 <<= 1;                     // prepare for next data
        pMED_SCL(1);                        // raise I2C clock wire so as to keep data stable
       MUSDELAY(5);                         // delay 5ms
    } while (--gLocal_2 != 0);              // finish 8 bits data

    pMED_SCL(0);                            // lower I2C clock wire, and prepare for ACK bit
    pMED_SDA(1);                            // raise I2C data wire, ACK bit is 1
    MUSDELAY(5);                            // delay 5ms
    pMED_SCL(1);                            // raise I2C clock wire
    MUSDELAY(5);                            // delay 5ms
    FLocal_1 = 0;                           // NCK mark means failure
    if (!pMED_SDA_HIGH){
        FLocal_1 = 1;                       // ACK mark means success
    }
    pMED_SCL(0);
    return FLocal_1;                        // return ACK/NAK marks
}


void MMEDStart(){                           // I2C start
    pMED_SDA(1);                            // raise data wire, data wire is idle
    pMED_SCL(1);                            // raise clock wire, clock wire is idle
    MUSDELAY(5);                            // delay 5ms, keep state stable
    pMED_SDA(0);        // for data wire, high level switches to low level when clock is high level,
which means start
    MUSDELAY(5);                            // delay 5ms
    pMED_SCL(0);                            // lower clock wire, ready to receive or send data
    return;
}


BYTE MMEDRead(BOOL FLocal_NAK){             // I2C read
    BYTE gLocal_1;                          // data temporary memory
    BYTE gLocal_2;                          // bit counter

    pMED_SDA(1);                            // SDA is ready to input
    MUSDELAY(5);                            // delay 5ms
    gLocal_2 = 8;                           // 8 bits
    do{
        pMED_SCL(1);            // raise I2C clock wire, data is effective when clock wire is high
level
        gLocal_1 <<= 1;                     // if data wire is high level, data temporary
memory is 1
        if (pMED_SDA_HIGH) gLocal_1 |= 0x01;
        MUSDELAY(5);                        // delay 5ms, keep state stable
```

```
        pMED_SCL(0);                    // lower I2C clock wire, receive next bit
        MUSDELAY(5);                        // delay 5ms, keep state stable
    } while (--gLocal_2 != 0);          // repeat 8 times, finish receiving one byte

    if (!FLocal_NAK){                   // if ACK mark is 0
        pMED_SDA(0);                        // lower data wire, continue read if ACK is 0
    }
    else {                              // if ACK mark is 1
        pMED_SDA(1);                    // raise data wire, read is over if ACK is 1
    }
    MUSDELAY(5);                            // delay 5ms
    pMED_SCL(1);                        // send ACK/NAK bit
    MUSDELAY(5);                            // delay 5ms
    pMED_SCL(0);                        // lower I2C clock wire
    return gLocal_1;                    //return received data
}

void MMEDStop(){                        // I2C stop
    pMED_SDA(0);                            //lower data wire, ready to stop
    MUSDELAY(5);                            // delay 6ms, keep state stable
    pMED_SCL(1);                        // raise clock wire
    MUSDELAY(5);                            // delay 6ms, keep state stable
    pMED_SDA(1);            // for data wire, low level switches to high level when clock is high
level, which means end
    return;
}
void MUSDELAY(BYTE gLocal_1){
    // delay according to user mainframe' status
    return;
}
```

Appendix 5:

5 Principles That HSAV uses C Language to Compile Source Code

1、Naming principle

Naming of all variable, constant and function is composed of three portions.

For example, FAUD_Mute is divided to three parts, namely: F, AUD, and _Mute.

The first portion is composed of one letter or one letter and one number, which mean the type of the definition.

Content	Meaning
Capital 'M'	Means function.
Capital 'F'	Means indexed variable, 1-bit variable.
Lower case 'g'	Means 8-bit variable.
Lower case 'g2'	Means 16-bit variable.
Lower case 'g4'	Means 32-bit variable.
Lower case 'g8'	Means 64-bit variable.
Lower case 'c'	Means constant.
Lower case 'p'	Means IO port.

The second part is composed of three to four English capital letters that mean the document the naming belongs to. For example, if variable is used in H06_AUD.C, the second part is AUD. Documents used usually are as follows,

Content	Meaning
AUD	Universal audio processing file
VOL	Multi-channel volume processing file
SUR	Multi-channel with surround sound processing file
SUB	The function of main file is being expanded. There is no need too many functions in the main file to prevent deteriorating effect.
DOS	Operating system processing file with USB mainframe or hard disk interface.
MED	Processing files with multimedia audio playing such as mp3.

The third part is concrete content that has one word or several words generally. The first letter of each word is capital and underline can be added between each word. As the capital letters separate each word, there is no need to add underline. The principle is that if it doesn't look good or the word is abbreviated (It's usually capitalized) underline can be used.

2、Principles of global and local variables

Content	Meaning (compatible with VC++)	C language standard
1-bit indexed variable	EXTR BOOL FAUD_Mute	Nonexistence
8-bit non-mark variable	EXTR BYTE gAUO_Volume	Unsigned char
16-bit non-mark variable	EXTR WORD g2AUO_EQ_Mode	Unsigned int
32-bit non-mark variable	EXTR DWORD g4AUO_Mute_Timer	Unsigned long
Pointer variable	EXTR BYTE *gpAUD_Pointer	Unsigned char
Local variable	EXTR BYTE gLocal_1	Unsigned char

Local variable absolutely forbids using 1 or several letters, e.g., when ‘X’ is the variable, it is difficult to copy and point out how many bits there are. All writing should be named for the first time and copying is necessary in the process of application, rewriting the same name is not suggested.

As for indexed local variable BOOL FLocal_1 and 8-bit local variable BYTE gLocal_1 etc, the first part of local variable and global variable are the same, the second part uses ‘Local_’ character string to mean local variable, and the third part is composed of numbers from 1 to 9 and lower case letters from “a” to “z.”