



## 4. Application Source Code

```

/*****/
/*
/* Prj. : QS1600 Application for mobile system
/* Programed by Jeong K. H. 2002 / 07
/* [ CPU : 8051 / CLOCK : 20Mhz / ROM : 64Kbyte / RAM : 256Byte ]
/*
/*****/
typedef unsigned char BYTE;
typedef unsigned int WORD;

#include <cfg8051.h>
#include <sample.h>

#define ON 1
#define OFF 0

/*****/
/* Define Interface Port Index and Command */
/*****/
#define HPPI_DATA (*(char *)0x018000) /* HPPI Data Port */
#define HPPI_ADDR (*(char *)0x018001) /* HPPI Address Port */

#define REG_SMF_CTRL 0x03
#define CMD_SMF_PLAY 0x11
#define CMD_SMF_STOP 0x12
#define CMD_SMF_PAUSE 0x13

#define REG_SMF_VOL 0x04

#define REG_SMF_REV1 0x05

#define REG_PIO_MIDI 0x06

#define REG_WAVE_CHAN 0x07
#define CMD_NON_WAVE 0x00
#define CMD_ONE_WAVE 0x01
#define CMD_TWO_WAVE 0x02

#define REG_WAV1_CTRL 0x08
#define CMD_WAV1_PLAY 0x11
#define CMD_WAV1_STOP 0x12
#define CMD_WAV1_PAUSE 0x13

#define REG_WAV1_VOL 0x09

#define REG_WAV1_PITCH 0x0A

#define REG_WAV1_TYPE 0x0B

#define REG_WAV2_CTRL 0x0C

```

```

#define CMD_WAV2_PLAY  0x11
#define CMD_WAV2_STOP  0x12
#define CMD_WAV2_PAUSE 0x13

#define REG_WAV2_VOL    0x0D

#define REG_WAV2_PITCH 0x0E

#define REG_WAV2_TYPE  0x0F

#define REG_BAUDRATE   0x18

#define REG_MIDI_BUF    0x11

#define REG_WAV1_BUF    0x12

#define REG_WAV2_BUF    0x13

#define REG_START_UP    0x14
#define CMD_START_UP    0x11

#define REG_OUT_MODE    0x16

#define REG_POWER_DOWN 0x17

#define REG_PWM_CLOCK   0x19

#define CMD_POWER_DOWN 0xAA

#define REG_IREQ_TYPE   0x1A

#define Q16_STATUS1_RD 0x1C
#define Q16_STATUS2_RD 0x1D
#define Q16_STATUS1_WR 0x1E
#define Q16_STATUS2_WR 0x1F

/*****
/* Define Variables and Decalre Functions */
/*****
#pragma memory=idata

bit FG_midi_play;
bit FG_wave1_play;
bit FG_wave2_play;
bit FG_q16_request;
bit FG_switch_press;
bit FG_restart;

BYTE WaveChannels;
WORD Ptr_midi,Ptr_wav1,Ptr_wav2;
WORD SizeMidi,SizeWav1,SizeWav2;
WORD One_mSec;
BYTE PreSwitchState

```

```

void Wait10mSec(WORD);
void Send_Wav1_Packet_Data(void);
void Send_Wav2_Packet_Data(void);
void Send_MIDI_Packet_Data(void);
void TransferMidiBlock(void);
void TransferWav1Block(void);
void TransferWav2Block(void);

/*****
/* Interrupt Vectors */
/*****
interrupt void INT_external0(void) /* request wave or midi from Qs1600 */
{
    FG_q16_request = ON;
}
interrupt void INT_external1(void) /* Not Used */
{
}

interrupt void INT_timer0(void) /* Not Used */
{
}

interrupt void INT_timer1(void) /* 1mSecond time counter */
{
    TH1 = 0xF9;
    TL1 = 0x7D;
    One_mSec++;
}

interrupt void INT_serial(void) /* Not Used */
{
}

/*****
/* Main Procedure Functions */
/*****
void CommandOut(idata BYTE reg, idata BYTE val)
{
    BYTE bdata, i;

    RECOMMAND:
    /* clear QS1600 busy status bit */
    HPPI_ADDR = Q16_STATUS2_WR;
    HPPI_DATA = 0x00;

    /* set command index "reg" and transfer command "val" */
    HPPI_ADDR = reg;
    HPPI_DATA = val;

    /* wait on for acknowledge from QS1600 */
    for(i=0; i<20; i++) {

```

```

        HPPI_ADDR = Q16_STATUS2_RD;
        bdata = HPPI_DATA;
        if(bdata == reg) break;
    }

    /* if non-acknowledge, repeat transfer command process */
    if(i==20) goto RECOMMAND;
}

void Wait10mSec(WORD time)
{
    One_mSec = 1;
    while(clk_delay < time) ;
}

void Send_MIDI_Packet_Data(void)
{
    HPPI_ADDR = REG_MIDI_BUF;
    TransferMidiBlock();
    TransferMidiBlock();
    HPPI_ADDR = 5;
    HPPI_DATA = 5;
    if(Ptr_midi >= SizeMidi) {
        Ptr_midi = 0x0000;
        CommandOut(REG_SMF_CTRL,CMD_SMF_STOP);
        FG_midi_play = OFF;
        if(FG_midi_play == OFF) {
            /* TEST_MIDI
            FG_restart = ON; */
        }
    }
}

void Send_Wav1_Packet_Data(void)
{
    if(WaveChannels == 1) {
        HPPI_ADDR = REG_WAV1_BUF;
        TransferWav1Block();
        TransferWav1Block();
    }
    else if(WaveChannels == 2) {
        HPPI_ADDR = REG_WAV1_BUF;
        TransferWav1Block();
    }
    if(Ptr_wav1 >= SizeWav1) {
        Ptr_wav1 = 0x0004;
        CommandOut(REG_WAV1_CTRL,CMD_WAV1_STOP);
        FG_wave1_play = OFF;
        if(FG_wave2_play == OFF) {
            /* TEST_WAVE
            FG_restart = ON; */
        }
    }
}

```

```

void Send_Wav2_Packet_Data(void)
{
    if(WaveChannels == 1) {
        HPPI_ADDR = REG_WAV1_BUF;
        TransferWav2Block();
        TransferWav2Block();
    }
    else if(WaveChannels == 2) {
        HPPI_ADDR = REG_WAV2_BUF;
        TransferWav2Block();
    }

    if(Ptr_wav2 >= SizeWav2) {
        Ptr_wav2 = 0x0004;
        CommandOut((BYTE)REG_WAV2_CTRL,(BYTE)CMD_WAV2_STOP);
        FG_wave2_play = OFF;
        if(FG_wave1_play == OFF) {
            /* TEST_WAVE
            FG_restart = ON; */
        }
    }
}

void TransferMidiBlock(void)
{
    BYTE i;

    for(i=0;i<64;i++) HPPI_DATA = TBL_midi_data[Ptr_midi++];
}

void TransferWav1Block(void)
{
    BYTE i;

    for(i=0;i<64;i++) HPPI_DATA = TBL_wav1_data[Ptr_wav1++];
}

void TransferWav2Block(void)
{
    BYTE i;

    for(i=0;i<64;i++) HPPI_DATA = TBL_wav2_data[Ptr_wav2++];
}

void main(void)
{
    BYTE bdata;
    WORD cnt;

    /* intialize SFR */
    IP = 0x01;
    TMOD = 0x11;
    SCON = 0x00;
}

```

```
TH1 = 0xF9;
TL1 = 0x7D;
TCON = 0x45;
IE = 0x00;
```

```
/* initialize flags */
P3.5 = OFF; /* wake up QS1600 active */
Wait10mSec(1);
P3.5 = ON; /* wake up QS1600 */
FG_midi_play = OFF;
FG_wave1_play = OFF;
FG_wave2_play = OFF;
FG_q16_request = OFF;
FG_switch_press = OFF;
FG_restart = OFF;
```

```
/* wait on QS1600's warm boot */
for(;;) {
    HPPI_ADDR = Q16_STATUS1_RD;
    bdata = HPPI_DATA;
    if(bdata == 0x83) {
        HPPI_ADDR = REG_START_UP;
        HPPI_DATA = CMD_START_UP;
        break;
    }
}
```

```
/* initialize system */
P1 = 0xFF;
PreSwitchState = 0xFF;
HPPI_ADDR = REG_PWM_CLOCK;
HPPI_DATA = 0x02;
HPPI_ADDR = REG_WAVE_CHAN;
HPPI_DATA = CMD_NON_WAVE;
HPPI_ADDR = REG_OUT_MODE;
HPPI_DATA = 0x48;
IE = 0x89;
```

```
/* set data size to play */
/*
SizeMidi = 0XXXXX;
SizeWav1 = 0XXXXX;
SizeWav2 = 0XXXXX;
*/
```

```
/* Main Loop Begin */
while(1){
    if(PreSwitchState != P1) {
        bdata = PreSwitchState ^ P1;
        PreSwitchState = P1;
        switch(bdata) {
            case 0x01 : /* midi play start */
                if(P1.0 == OFF && FG_midi_play == OFF) {
```

```

        FG_midi_play = ON;
        CommandOut(REG_SMF_CTRL,CMD_SMF_PLAY);
        Ptr_midi = 0x0000;
        Send_MIDI_Packet_Data();
    }
    break;
case 0x02 : /* one channel wave  play start */
    if(P1.1 == OFF && FG_wave1_play == OFF) {
        FG_wave1_play = ON;
        HPPI_ADDR = REG_WAVE_CHAN;
        HPPI_DATA = CMD_ONE_WAVE;
        WaveChannels = CMD_ONE_WAVE;
        Ptr_wav1 = 0x0004;
        HPPI_ADDR = REG_WAV1_TYPE;
        HPPI_DATA = TBL_wav1_data[3];
        Send_Wav1_Packet_Data();
        CommandOut(REG_WAV1_CTRL,CMD_WAV1_PLAY);
    }
    break;
case 0x04 : /* two channel wave  play start */
    if(P1.2 == OFF && FG_wave2_play == OFF) {
        FG_wave2_play = ON;
        FG_wave1_play = ON;
        HPPI_ADDR = REG_WAVE_CHAN;
        HPPI_DATA = CMD_TWO_WAVE;
        WaveChannels = CMD_TWO_WAVE;
        Ptr_wav2 = 0x0004;
        Ptr_wav1 = 0x0004;
        HPPI_ADDR = REG_WAV1_TYPE;
        HPPI_DATA = TBL_wav1_data[3];
        Send_Wav1_Packet_Data();
        HPPI_ADDR = REG_WAV2_TYPE;
        HPPI_DATA = TBL_wav2_data[3];
        Send_Wav2_Packet_Data();
        CommandOut(REG_WAV1_CTRL,CMD_WAV1_PLAY);
    }
    break;
case 0x08 : /* volume up */
    if(P1.3) {

    }
    break;
case 0x10 : /* midi play stop */
    if(P1.4 == OFF && FG_midi_play == ON) {
        FG_midi_play = OFF;
        Ptr_midi = 0x0000;
        CommandOut(REG_SMF_CTRL,CMD_SMF_STOP);
    }
    break;
case 0x20 : /* wave play stop */
    if(P1.5==OFF&&(FG_wave2_play == ON || FG_wave1_play == ON)) {
        FG_wave2_play = OFF;
        FG_wave1_play = OFF;
        Ptr_wav1 = 0x0000;

```

```

        Ptr_wav2 = 0x0000;
        CommandOut(REG_WAV1_CTRL,CMD_WAV1_STOP);
        CommandOut(REG_WAV2_CTRL,CMD_WAV2_STOP);
    }
    break;
case 0x40 : /* reserved */
    if(P1.6) {

        }
    break;
case 0x80 : /* volume down */
    if(P1.7) {

        }
    break;
default : break;
}
}

if(FG_q16_request == ON){
    IE = 0x00;
    HPPI_ADDR = REG_IREQ_TYPE;
    bdata = HPPI_DATA;
    if(FG_midi_play == ON) {
        if(bdata & 0x04) Send_MIDI_Packet_Data();
    }
    if(FG_wave2_play == ON) {
        if(bdata & 0x01) Send_Wav2_Packet_Data();
    }
    if(FG_wave1_play == ON) {
        if(bdata & 0x02) Send_Wav1_Packet_Data();
    }
    FG_q16_request = OFF;
    IE = 0x89;
}
}
}

```