

ALC250 EQUALIZER APPLICATION NOTES

0. Revision History	1	2-2. Equalizer Control by Serial Bus	2
1. Hardware Digital Equalizer	1	2-3. Serial Bus Connection	14
2. Equalizer Control.....	2	3. Equalizer Coefficient Generator – EQ250.EXE.....	15
2-1. Equalizer Control by ACLINK	2		

0. Revision History

Version 0.10:

(1) Preliminary version

1. Hardware Digital Equalizer

The ALC250 integrates seven bands of hardware digital equalizer cascades one low-pass filter, five band-pass filters and one High-pass filter. It is for optimizing frequency response of speaker on laptop PCs. Figure 1 indicates the cascaded equalizer architecture. The ALC250 also built in a **dedicated Analog-to-Digital converter (EQ ADC)** used to **digitize input analog audio into PCM format sent to DAC**, that makes digital equalizer can also compensate frequency response for analog inputs.

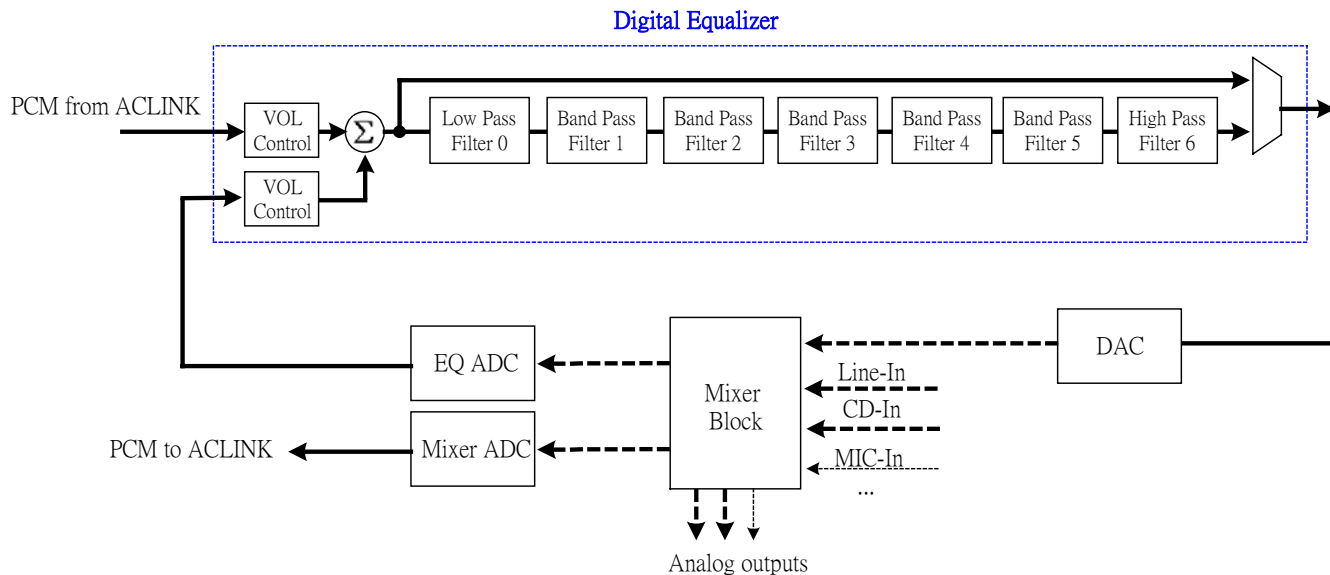


Figure 1-1 Cascaded digital equalizer

2. Equalizer Control

The integrated digital equalizer can be configured in two situations. One is system BIOS or device driver can program chipset through ACLINK to control equalizer coefficients, it is for system is in normal operation. Another one is system is in power off mode, but uP (ex. Audio DJ®) still play audio signal to Codec, the ALC250 supports 2-wire of serial bus to let uP can control equalizer coefficients and volume setting.

2-1. Equalizer Control by ACLINK

BIOS or driver can access all the equalizer coefficients through ACLINK. To accommodate audio driver supplied by chipset vendor or Microsoft, those equalizer coefficients will not be cleared by 'Cold Reset' on ACLINK, BIOS or driver must take care the set and reset state for equalizer coefficients. Please refer to data sheet to get detail coefficient definition.

2-2. Equalizer Control by Serial Bus

The ALC250 supports 2-wire of serial bus to let uP can control equalizer coefficients and volume setting when system is in power off mode. Due to both serial bus and ACLINK can access equalizer coefficient, a serial Bus Control (SBC) register only controlled by serial bus is used to hang shake register access to keep away from bus competition.

WORD Address-80h Serial Bus Control (SBC)

Bit	Type	Function
15	R	AC'97 Mixer Interface Status 0: Idle, AC-LINK is not accessing mixers. 1: Busy, AC-LINK is accessing mixers.
14:3		Reserved
2	R/W	DPE & EQ Register Reset Control (DPE index-00h ~index-24h) 0: Not affected by ACLINK Cold/Warm/Register reset. (Default) 1: Can be reset by ACLINK Cold/Warm/Register reset.
1	R/W	AC-LINK Mixer Interface Control (exclusive read/write control)❶❷ 0: All registers except SBC registers are accessed via AC-LINK. (default) 1: SBC can access AC-LINK register, AC-LINK access register and get normal data while MX74.7 is zero. (Registers will not be reset by AC-LINK Cold/Warm/Register reset.)
0	R/W	Reset Control 0: Reset ALC250. 1: Normal. (Default) This bit is gated (AND) with AC97_RESET# to reset all registers except SBC registers to default value. This bit should be back to 1 automatically after H/W has finish reset.

❶ The reset condition of Digital Equalizer and its registers is:

(Power on reset) | (SBC[0]) | (SBC[2] & Cold/Warm/Register reset)

The reset condition of registers except Digital Equalizer and its registers is:

(Power on reset) | (SBC[0]) | SBC[1]==0 & Cold/Warm/Register reset)

❷ Serial bus programmer must check SBC[15] before setting SBC[1].

Programmer can use following procedures in different system power state.

In normal power on mode: (BIOS programs equalizer, OS driver doesn't take care equalizer)

(+3.3V is on, +5VA is on, AC-LINK is on, Equalizer is supported)

1. Serial bus set SBC[2]=1 after check SBC[15]=0.
2. Serial bus set SBC[1]=1 to get control of mixer interface.
3. Serial bus programs Digital Equalizer and its coefficient registers.
4. Serial bus set SBC[1]=0 to release control of mixer interface.
5. AC-LINK programs ALC250 under OS.

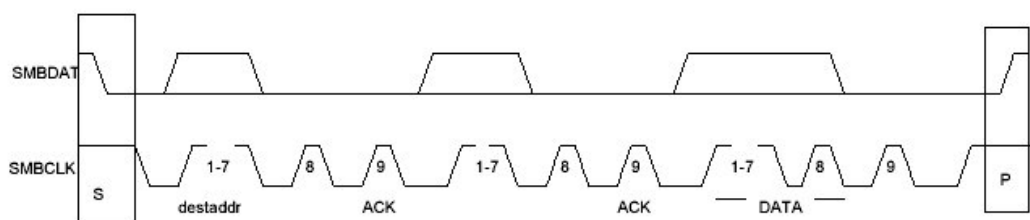
In system shut off mode: (uP programs equalizer through serial bus)

(+3.3V is on, +5VA is on, AC-LINK is off, Equalizer is supported)

1. Serial bus set SBC[2]=1 after check SBC[15]=0.
2. Serial bus set SBC[1]=1 to get control of mixer interface.
3. Serial bus programs Digital Equalizer and its coefficient registers.
4. Serial bus programs mixers to do volume control. (CD to output path can be turn on via serial bus programming.)

③ ALC250 supports WORD mode of serial bus Read/Write: (Low byte first)

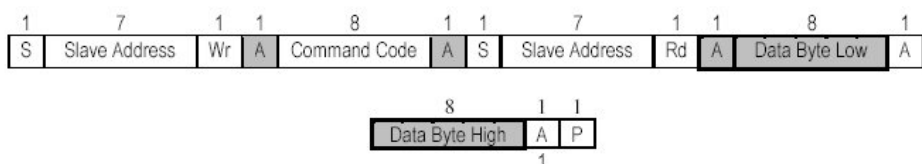
Data transfer over Serial Bus:



Write WORD Protocol:



Read WORD Protocol:



S: Start Condition

Slave Address: 7-bit Device Address is '0010000b'.

Wr: 0 for Write Command

Rd: 1 for Read Command

Command Code: 8-bit Mixer Address

A: 0 for ACK, 1 for NACK

Data Byte: 16-bit Mixer data

□: Master-to-Slave

■: Slave-to-Master

④ Sample C code for uP Read/Write Command

// Assumption:

// Use GPIO0 as SCK, GPIO1 as SDA in uP

// A 32-bit register in IOBASE+0x4C is GPIO pin control

// Bit[31:16] configure properties of GPIO[15:0]. 0:input 1:output

// Bit[15:0] are Read(for input)/Write(for output) states at GPIO[15:0]. 0:Low 1:High

int uPWrite(unsigned int address, unsigned int data)

```
{
    int i,n;
    int lowbyte,highbyte;
    lowbyte = data&0x00ff;
    highbyte = (data&0xff00)>>8;

    //Initial a 'Start Bit'
    axx=inx(IOBASE+0x4c)|0x00030003;
    outx(IOBASE+0x4c,axx);           //Configure both GPIO0/GPIO1 are output, and driven high
    delay(0.02);
    axx=inx(IOBASE+0x4c)&0x00030001;
    outx(IOBASE+0x4c,axx);           //Drive GPIO0(SCK)=high, GPIO1(SDA)=low, to initial a 'Start Bit'
    delay(0.02);
    axx=inx(IOBASE+0x4c)&0x00030000;
    outx(IOBASE+0x4c,axx);
    delay(0.02);
```

```
//Slave addressress
axx=inx(IOBASE+0x4c)|0x00030001;    //prepare data
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030003;    //latch data bit7=0
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030000;
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)|0x00030001;    //prepare data
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030003;    //latch data bit6=0
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030000;
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)|0x00030002;    //prepare data
outx(IOBASE+0x4c,axx);
axx=inx(IOBASE+0x4c)|0x00030003;    //prepare data
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030003;    //latch data bit5=1
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030002;
outx(IOBASE+0x4c,axx);
axx=inx(IOBASE+0x4c)&0x00030000;
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)|0x00030001;    //prepare data
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030003;    //latch data bit4=0
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030000;
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)|0x00030001;    //prepare data
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030003;    //latch data bit3=0
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030000;
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)|0x00030001;    //prepare data
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030003;    //latch data bit2=0
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030000;
outx(IOBASE+0x4c,axx);
```

```
delay(0.02);
axx=inx(IOBASE+0x4c)|0x00030001;    //prepare data
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030003;    //latch data bit1=0
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030000;
outx(IOBASE+0x4c,axx);
delay(0.02);

//Wr Bit
axx=inx(IOBASE+0x4c)|0x00030001;    //prepare data
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030003;    //latch data WR=0
outx(IOBASE+0x4c,axx);
delay(0.01);
axx=inx(IOBASE+0x4c)&0x00030002;
outx(IOBASE+0x4c,axx);
axx=inx(IOBASE+0x4c)&0x00010000;
outx(IOBASE+0x4c,axx);
delay(0.01);

//Wait for ACK singnal
axx=inx(IOBASE+0x4c)|0x00010001;
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030003;
outx(IOBASE+0x4c,axx);
for(i=0;i<10;i++)
{
    axx=inx(IOBASE+0x4c)&0x00000002;
    if((axx!=0))
    {
        gotoeol(50,11);
        printf("Do not receive ACK(1)");
        return(1);
    }
}
//printf("Receive ACK signal form CODEC(1)");
axx=inx(IOBASE+0x4c)|0x00030000;
axx=axx&0x00030000;
outx(IOBASE+0x4c,axx);
delay(0.02);

//Command code
n=0x80;
for(i=0;i<8;i++)
{
    if((address&n)!=0) axx=inx(IOBASE+0x4c)|0x00030002;    //prepare data
    else axx=inx(IOBASE+0x4c)|0x00030000;
    outx(IOBASE+0x4c,axx);
    axx=inx(IOBASE+0x4c)|0x00030001;
    outx(IOBASE+0x4c,axx);
    delay(0.02);
    axx=inx(IOBASE+0x4c)&0x00030003;    //latch data
    outx(IOBASE+0x4c,axx);
    delay(0.02);
}
```

```
    axx=inx(IOBASE+0x4c)&0x00030002;
    outx(IOBASE+0x4c,axx);
    n=n>>1;
    axx=inx(IOBASE+0x4c)&0x00030000;
    outx(IOBASE+0x4c,axx);
    delay(0.02);
}

//Wait for ACK singnal
axx=inx(IOBASE+0x4c)|0x00010001;
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030003;
outx(IOBASE+0x4c,axx);
for(i=0;i<10;i++)
{
    axx=inx(IOBASE+0x4c)&0x00000002;
    if(axx!=0)
    {
        gotoeol(50,11);
        printf("Do not receive ACK(2)");
        return(1);
    }
}

//printf("Receive ACK signal form CODEC(2)");
axx=inx(IOBASE+0x4c)|0x00030000;
axx=axx&0x00030000;
outx(IOBASE+0x4c,axx);
delay(0.02);

//Low Byte
n=0x80;
for(i=0;i<8;i++)
{
    if((lowbyte&n)!=0)    axx=inx(IOBASE+0x4c)|0x00030002;    //prepare data
    else    axx=inx(IOBASE+0x4c)|0x00030000;
    outx(IOBASE+0x4c,axx);
    axx=inx(IOBASE+0x4c)|0x00030001;
    outx(IOBASE+0x4c,axx);
    delay(0.02);
    axx=inx(IOBASE+0x4c)&0x00030003;    //latch data
    outx(IOBASE+0x4c,axx);
    delay(0.02);
    axx=inx(IOBASE+0x4c)&0x00030002;
    outx(IOBASE+0x4c,axx);
    n=n>>1;
    axx=inx(IOBASE+0x4c)&0x00030000;
    outx(IOBASE+0x4c,axx);
    delay(0.02);
}

//Wait for ACK singnal
axx=inx(IOBASE+0x4c)|0x00010001;
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030003;
outx(IOBASE+0x4c,axx);
for(i=0;i<10;i++)
```

```
{
  axx=inx(IOBASE+0x4c)&0x00000002;
  if(axx!=0)
  {
    gotoeol(50,11);
    printf("Do not receive ACK(3)");
    return(1);
  }
}

//printf("Receive ACK signal form CODEC(3)");
axx=inx(IOBASE+0x4c)|0x00030000;
axx=axx&0x00030000;
outx(IOBASE+0x4c,axx);
delay(0.02);

//High Byte
n=0x80;
for(i=0;i<8;i++)
{
  if((highbyte&n)!=0)      axx=inx(IOBASE+0x4c)|0x00030002;    //prepare data
  else      axx=inx(IOBASE+0x4c)|0x00030000;
  outx(IOBASE+0x4c,axx);
  axx=inx(IOBASE+0x4c)|0x00030001;
  outx(IOBASE+0x4c,axx);
  delay(0.02);
  axx=inx(IOBASE+0x4c)&0x00030003;    //latch data
  outx(IOBASE+0x4c,axx);
  delay(0.02);
  axx=inx(IOBASE+0x4c)&0x00030002;
  outx(IOBASE+0x4c,axx);
  n=n>>1;
  axx=inx(IOBASE+0x4c)&0x00030000;
  outx(IOBASE+0x4c,axx);
  delay(0.02);
}
//Wait for ACK singnal
axx=inx(IOBASE+0x4c)|0x00010001;
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030003;
outx(IOBASE+0x4c,axx);
for(i=0;i<10;i++)
{
  axx=inx(IOBASE+0x4c)&0x00000002;
  if(axx!=0)
  {
    gotoeol(50,11);
    printf("Do not receive ACK(4)");
    return(1);
  }
}

//printf("Receive ACK signal form CODEC(4)");
axx=inx(IOBASE+0x4c)|0x00030000;
axx=axx&0x00030000;
outx(IOBASE+0x4c,axx);
delay(0.02);
```

```
//Stop Bit
axx=inx(IOBASE+0x4c)|0x00030001;
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)|0x00030003;
outx(IOBASE+0x4c,axx);
delay(1);
return(0);
} // End of 'Write' subroutine
```

```
//=====
```

unsigned int uPRead(unsigned int address)

```
{
    int i,n;
    unsigned int lowbyte,highbyte;
    unsigned int data;
//Start bit
axx=inx(IOBASE+0x4c)|0x00030003;
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030001;
outx(IOBASE+0x4c,axx);
delay(0.05);
axx=inx(IOBASE+0x4c)&0x00030000;
outx(IOBASE+0x4c,axx);
delay(0.02);
//Slave address
axx=inx(IOBASE+0x4c)|0x00030001; //prepare data
outx(IOBASE+0x4c,axx);
axx=inx(IOBASE+0x4c)&0x00030003; //latch data bit7=0
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030000;
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)|0x00030001; //prepare data
outx(IOBASE+0x4c,axx);
axx=inx(IOBASE+0x4c)&0x00030003; //latch data bit6=0
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030000;
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)|0x00030002; //prepare data
outx(IOBASE+0x4c,axx);
axx=inx(IOBASE+0x4c)|0x00030003; //prepare data
outx(IOBASE+0x4c,axx);
delay(0.015);
axx=inx(IOBASE+0x4c)&0x00030003; //latch data bit5=1
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030002;
outx(IOBASE+0x4c,axx);
axx=inx(IOBASE+0x4c)&0x00030000;
outx(IOBASE+0x4c,axx);
delay(0.015);
axx=inx(IOBASE+0x4c)|0x00030001; //prepare data
outx(IOBASE+0x4c,axx);
```

```

axx=inx(IOBASE+0x4c)&0x00030003;    //latch data bit4=0
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030000;
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)|0x00030001;    //prepare data
outx(IOBASE+0x4c,axx);
axx=inx(IOBASE+0x4c)&0x00030003;    //latch data bit3=0
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030000;
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)|0x00030001;    //prepare data
outx(IOBASE+0x4c,axx);
axx=inx(IOBASE+0x4c)&0x00030003;    //latch data bit2=0
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030000;
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)|0x00030001;    //prepare data
outx(IOBASE+0x4c,axx);
axx=inx(IOBASE+0x4c)&0x00030003;    //latch data bit1=0
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030000;
outx(IOBASE+0x4c,axx);
delay(0.02);

//Wr Bit
axx=inx(IOBASE+0x4c)|0x00030001;    //prepare data
outx(IOBASE+0x4c,axx);
axx=inx(IOBASE+0x4c)&0x00030003;    //latch data WR=0
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030002;
outx(IOBASE+0x4c,axx);
delay(0.005);
axx=inx(IOBASE+0x4c)&0x00010000;
outx(IOBASE+0x4c,axx);
delay(0.01);

//Wait for ACK singnal
axx=inx(IOBASE+0x4c)|0x00010001;
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030003;
outx(IOBASE+0x4c,axx);
for(i=0;i<500;i++)
{
    axx=inx(IOBASE+0x4c)&0x00000002;
    if(axx!=0)
    {
        gotoeol(50,11);
        printf("Do not receive ACK(1)");
        return(5);
    }
}

```

```
    else
        break;
}
//printf("Receive ACK signal form CODEC(1)");
axx=inx(IOBASE+0x4c)|0x00030000;
axx=axx&0x00030000;
outx(IOBASE+0x4c,axx);
delay(0.02);
//Command code
n=0x80;
for(i=0;i<8;i++)
{
    if((address&n)!=0)        axx=inx(IOBASE+0x4c)|0x00030002;    //prepare data
    else                    axx=inx(IOBASE+0x4c)|0x00030000;
    outx(IOBASE+0x4c,axx);
    axx=inx(IOBASE+0x4c)|0x00030001;
    outx(IOBASE+0x4c,axx);
    axx=inx(IOBASE+0x4c)&0x00030003;    //latch data
    outx(IOBASE+0x4c,axx);
    delay(0.02);
    axx=inx(IOBASE+0x4c)&0x00030002;
    outx(IOBASE+0x4c,axx);
    n=n>>1;
    axx=inx(IOBASE+0x4c)&0x00030000;
    outx(IOBASE+0x4c,axx);
    delay(0.02);
}

//Wait for ACK signal
axx=inx(IOBASE+0x4c)|0x00010001;
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030003;
outx(IOBASE+0x4c,axx);
for(i=0;i<100;i++)
{
    axx=inx(IOBASE+0x4c)&0x00000002;
    if(axx!=0)
    {
        gotoeol(50,11);
        printf("Do not receive ACK signal form CODEC(2)");
        return(5);
    }
    else break;
}
//printf("Receive ACK signal form CODEC(2)");
axx=inx(IOBASE+0x4c)|0x00030000;
axx=axx&0x00030000;
outx(IOBASE+0x4c,axx);
delay(0.02);

//Start bit
axx=inx(IOBASE+0x4c)|0x00030002;
outx(IOBASE+0x4c,axx);
axx=inx(IOBASE+0x4c)|0x00030003;
outx(IOBASE+0x4c,axx);
delay(0.01);
axx=inx(IOBASE+0x4c)&0x00030001;
outx(IOBASE+0x4c,axx);
```

```

delay(0.03);
axx=inx(IOBASE+0x4c)&0x00030000;
outx(IOBASE+0x4c,axx);
delay(0.02);

//Slave address
axx=inx(IOBASE+0x4c)|0x00030001;    //prepare data
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030003;    //latch data bit7=0
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030000;
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)|0x00030001;    //prepare data
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030003;    //latch data bit6=0
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030000;
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)|0x00030002;    //prepare data
outx(IOBASE+0x4c,axx);
axx=inx(IOBASE+0x4c)|0x00030003;    //prepare data
outx(IOBASE+0x4c,axx);
delay(0.01);
axx=inx(IOBASE+0x4c)&0x00030003;    //latch data bit5=1
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030002;
outx(IOBASE+0x4c,axx);
axx=inx(IOBASE+0x4c)&0x00030000;
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)|0x00030001;    //prepare data
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030003;    //latch data bit4=0
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030000;
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)|0x00030001;    //prepare data
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030003;    //latch data bit3=0
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030000;
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)|0x00030001;    //prepare data
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030003;    //latch data bit2=0

```

```
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030000;
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)|0x00030001;    //prepare data
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030003;    //latch data bit1=0
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030000;
outx(IOBASE+0x4c,axx);
delay(0.02);

//Rd Bit
axx=inx(IOBASE+0x4c)|0x00030002;    //prepare data
outx(IOBASE+0x4c,axx);
axx=inx(IOBASE+0x4c)|0x00030003;    //prepare data
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030003;    //latch data RD=1
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030002;
outx(IOBASE+0x4c,axx);
delay(0.005);
axx=inx(IOBASE+0x4c)&0x00010000;
outx(IOBASE+0x4c,axx);
delay(0.012);

//Wait for ACK signal
axx=inx(IOBASE+0x4c)|0x00010001;
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030003;
outx(IOBASE+0x4c,axx);
for(i=0;i<100;i++)
{
    axx=inx(IOBASE+0x4c)&0x00000002;
    if(axx!=0)
    {
        gotoeol(50,11);
        printf("Do not receive ACK(3)");
        return(5);
    }
    else break;
}

//printf("Receive ACK signal form CODEC(3)");
axx=inx(IOBASE+0x4c)|0x00010000;
axx=axx&0x00010000;
outx(IOBASE+0x4c,axx);
delay(0.02);

//Low byte
data=0;
for(i=0;i<8;i++)
{
```

```
axx=inx(IOBASE+0x4c)|0x00010001;
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030003;
outx(IOBASE+0x4c,axx);
delay(0.01);
axx=inx(IOBASE+0x4c)&0x00000002;
if(axx!=0) data=data|0x0001;
```

```
data=data<<1;
axx=inx(IOBASE+0x4c)&0x00030002;
outx(IOBASE+0x4c,axx);
axx=inx(IOBASE+0x4c)&0x00030000;
outx(IOBASE+0x4c,axx);
delay(0.015);
}
axx=inx(IOBASE+0x4c)|0x00030000;
axx=axx&0x00030000;
outx(IOBASE+0x4c,axx);
```

//Send ACK signal

```
axx=inx(IOBASE+0x4c)|0x00030001;    //prepare data
outx(IOBASE+0x4c,axx);
axx=inx(IOBASE+0x4c)&0x00030003;    //latch data
outx(IOBASE+0x4c,axx);
delay(0.03);
axx=inx(IOBASE+0x4c)&0x00030002;
outx(IOBASE+0x4c,axx);
axx=inx(IOBASE+0x4c)&0x00010000;
outx(IOBASE+0x4c,axx);
delay(0.01);
```

//High byte

```
for(i=0;i<8;i++)
{
    axx=inx(IOBASE+0x4c)|0x00010001;
    outx(IOBASE+0x4c,axx);
    delay(0.02);
    axx=inx(IOBASE+0x4c)&0x00030003;
    outx(IOBASE+0x4c,axx);
    delay(0.01);
    axx=inx(IOBASE+0x4c)&0x00000002;
    if(axx!=0) data=data|0x0001;
    if(i!=7) data=data<<1;
    axx=inx(IOBASE+0x4c)&0x00030000;
    outx(IOBASE+0x4c,axx);
    delay(0.03);
}
axx=inx(IOBASE+0x4c)|0x00030000;
outx(IOBASE+0x4c,axx);
delay(0.02);
```

//Send NACK signal

```
axx=inx(IOBASE+0x4c)|0x00030003;    //prepare data
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)&0x00030003;    //latch data
outx(IOBASE+0x4c,axx);
delay(0.02);
```

```

axx=inx(IOBASE+0x4c)&0x00030000;
outx(IOBASE+0x4c,axx);
delay(0.03);

//Stop Bit
axx=inx(IOBASE+0x4c)|0x00030001;
outx(IOBASE+0x4c,axx);
delay(0.02);
axx=inx(IOBASE+0x4c)|0x00030003;
outx(IOBASE+0x4c,axx);

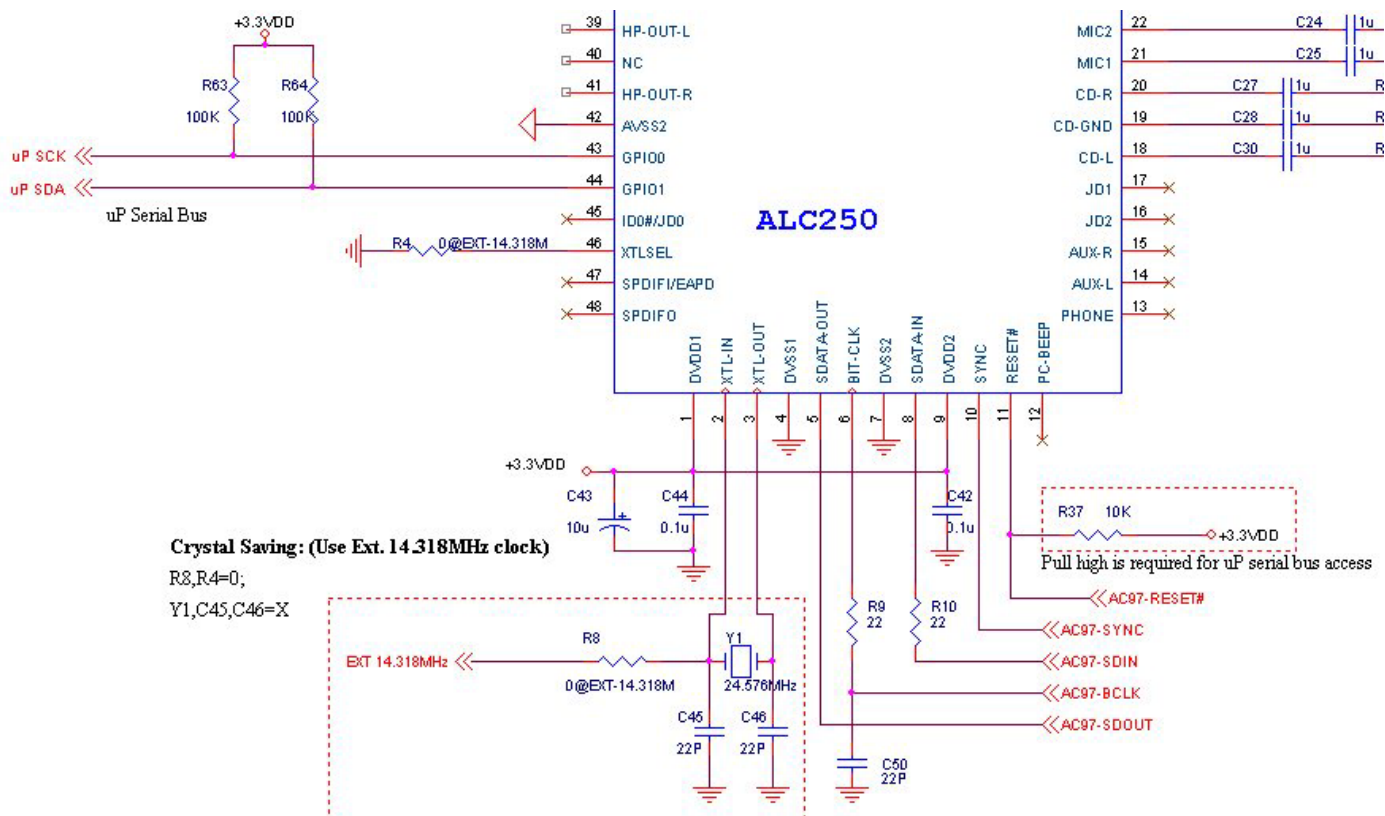
//Swap
lowbyte = (data&0xff00)>>8;
highbyte = (data&0x00ff)<<8;
data =highbyte | lowbyte;
axx=inx(IOBASE+0x4c)&0x00000000;
outx(IOBASE+0x4c,axx);
return(data);
} // End of 'Read' subroutine

```

2-3. Serial Bus Connection

If uP want to program equalizer through serial bus when system is in shut down mode (south bridge is off), system makers must aware some tricky points in system connection.

1. The uP serial bus at pin-43 and pin-44 must be external pull high. A pull high resistor at pin-11 (AC97 RESET#) is also required to avoid an unstable RESET# state may reset ALC250 when south bridge is shut down.
2. The integrated digital equalizer is powered by +3.3V digital power, the +3.3V power must be still supplied to keep equalizer is operating.
3. The clock source is necessary to keep equalizer is operating.



3. Equalizer Coefficient Generator – EQ250.EXE

Figure 3-1 indicates the control panel of 'EQ250.EXE'. The 'Filter Characteristic Description' field is used to define center frequency, bandwidth and gain control for each filter. 'Enable Filter' checkbox controls whether corresponding filter will be selected or not. The 'Control Item' decide the plotting behavior, 'Plot' button will draw the predicted response curve based on current setting in Filter Characteristic and Enable Filter. 'Original' button will reset curve to initial state had been assigned. 'Reset' button will clear current plotting, and reset all setting to zero state.

The 'EQ Parameter' field indicates the generated EQ coefficients according to setting in Filter Characteristic, which can be written into ALC250's EQ registers to turn on equalizer function.

Figure 3-2 to Figure 3-17 demo a method of operations in step-by- step.

Figure 3-1. The control panel of 'EQ250.EXE'

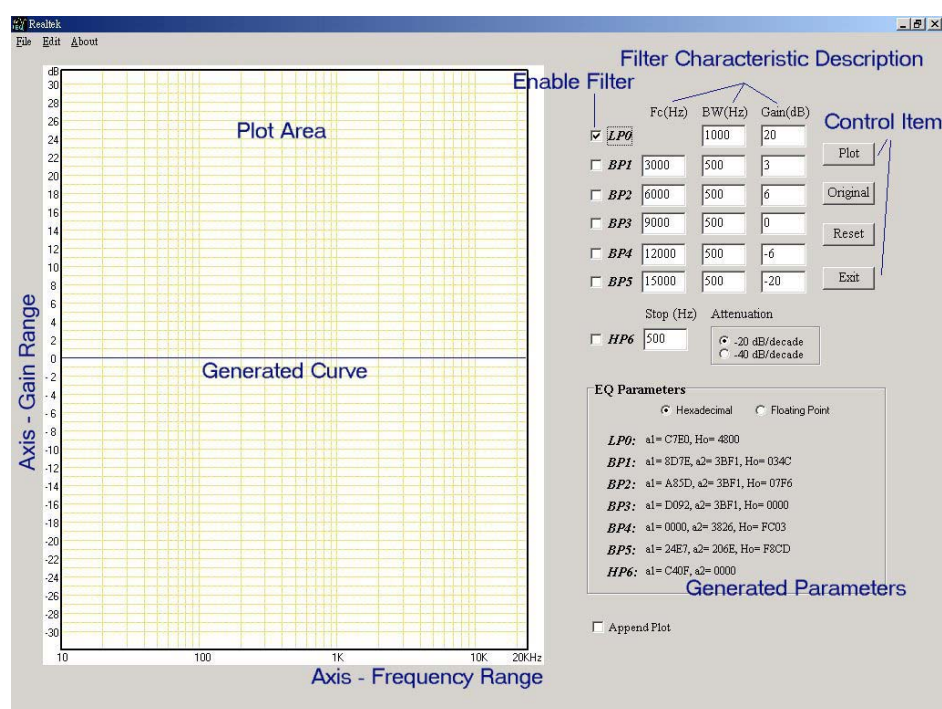


Figure 3-2. Set 'BW(Hz)' and 'Gain(dB)' for Low Pass Filter 0 (LP0), and enable LP0

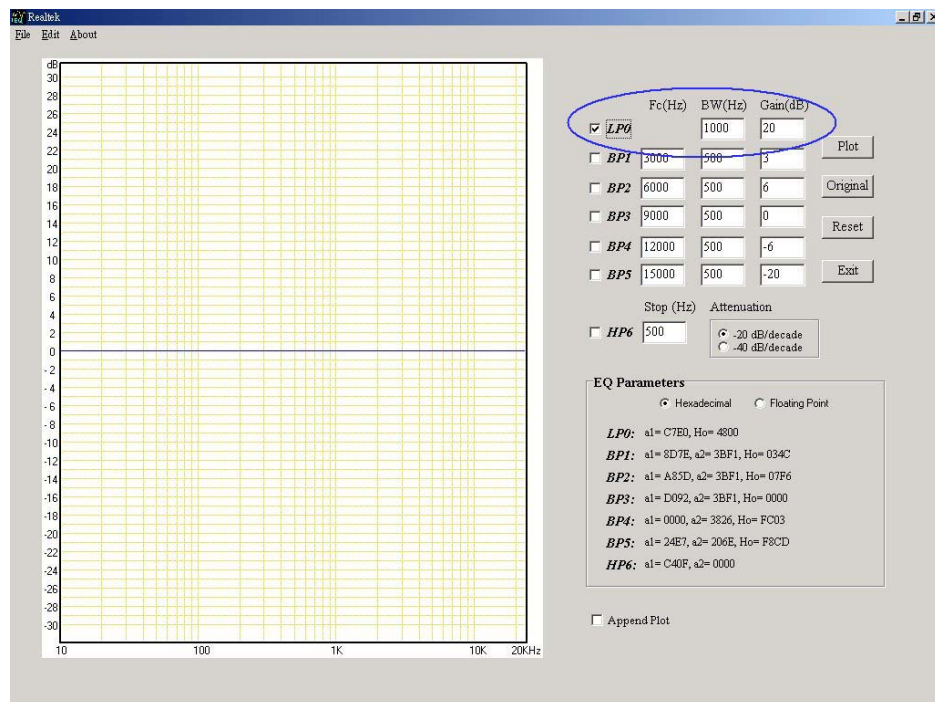


Figure 3-3. Press 'Plot' button to draw the frequency response

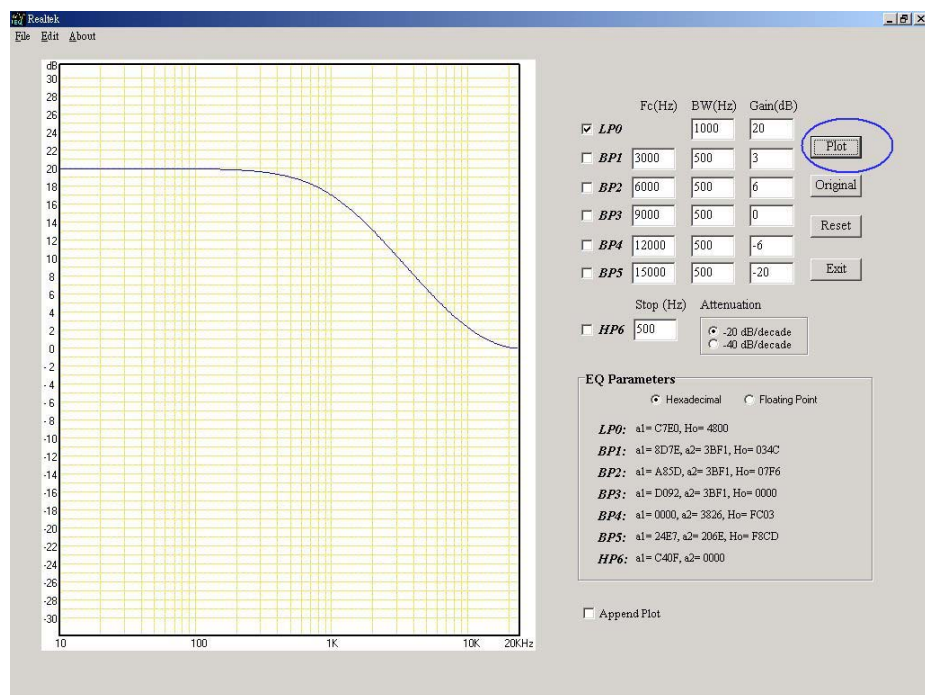


Figure 3-4. Enable ‘Append Plot’ checkbox which will accumulate previous curves, repeat procedures in Figure 3-2/3-3 to control new LP0 and BP1, then press ‘Plot’ button to get current curve and previous curves simultaneously.

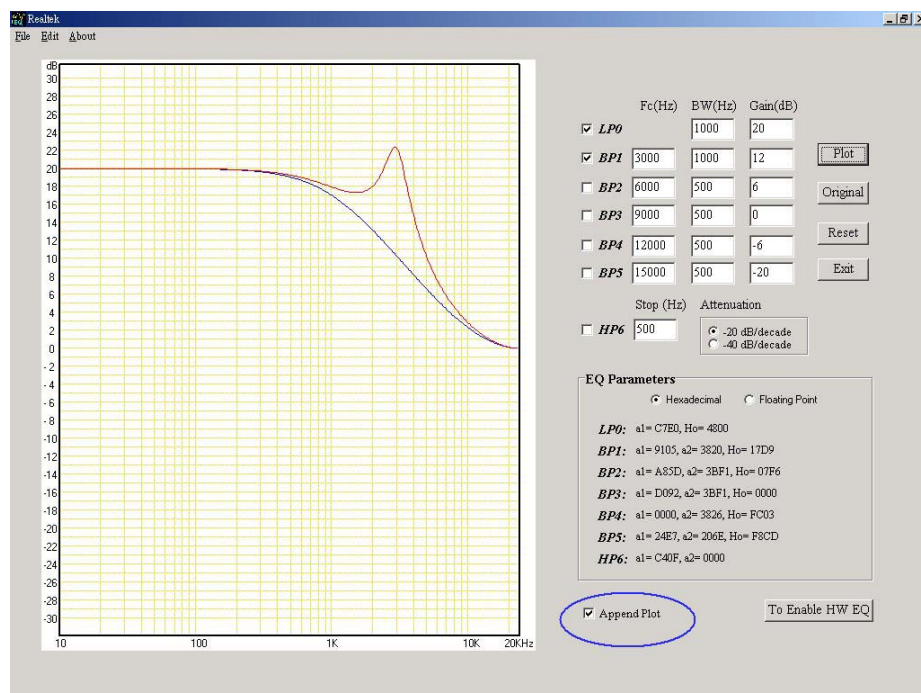


Figure 3-5. Setting and plot will be clear once ‘Reset’ button is pressed.

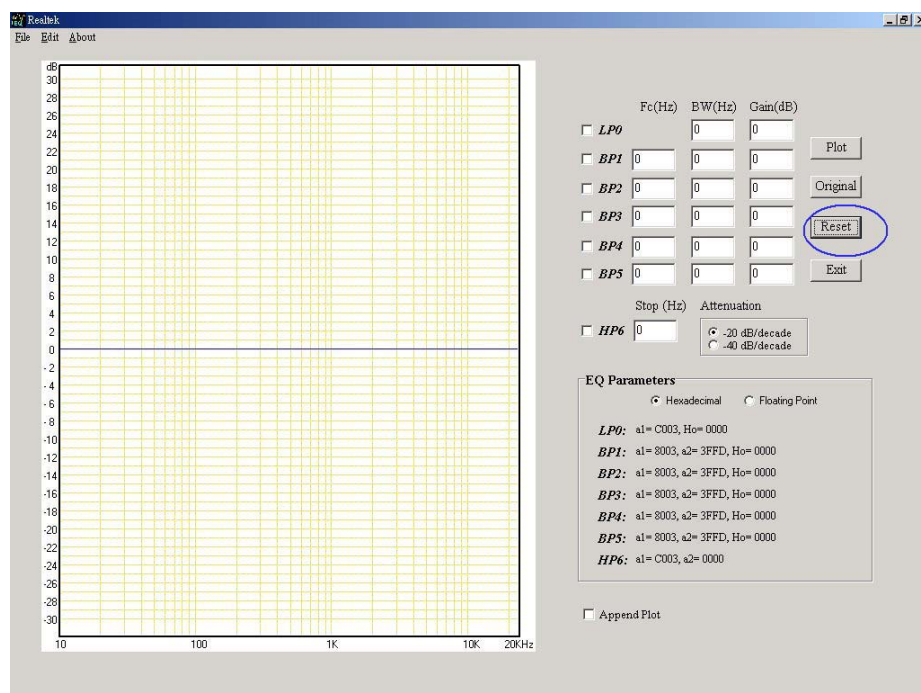


Figure 3-6. Save current setting and EQ coefficient, select ‘Save Parameters To File’ in ‘File’ menu.

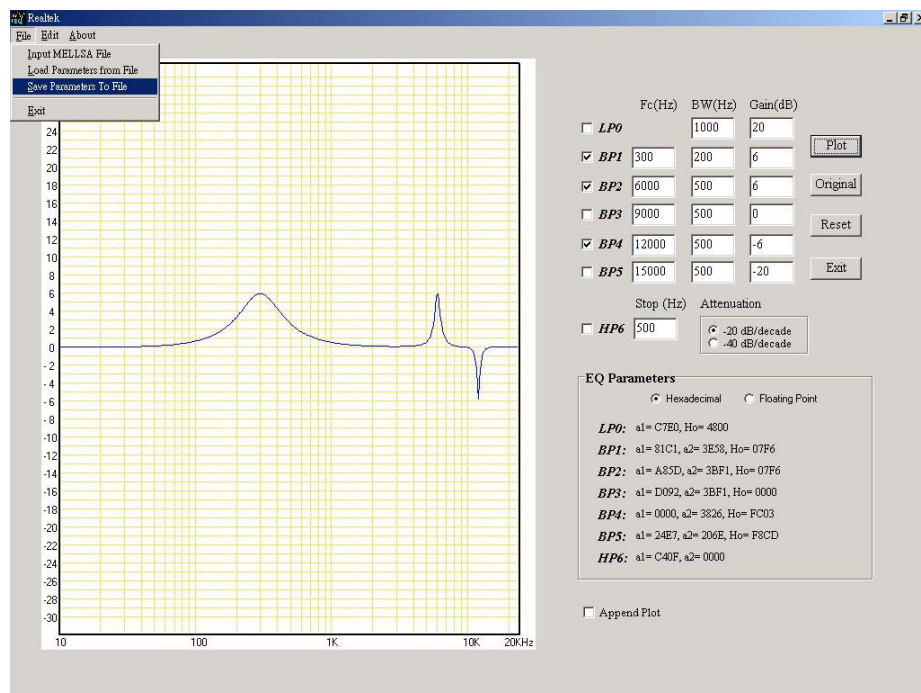


Figure 3-7. Assign file name ‘EQ250.dat’, and press ‘OK’ button to save all settings. Pass the EQ250.dat to Realtek to implement your EQ setting in driver.

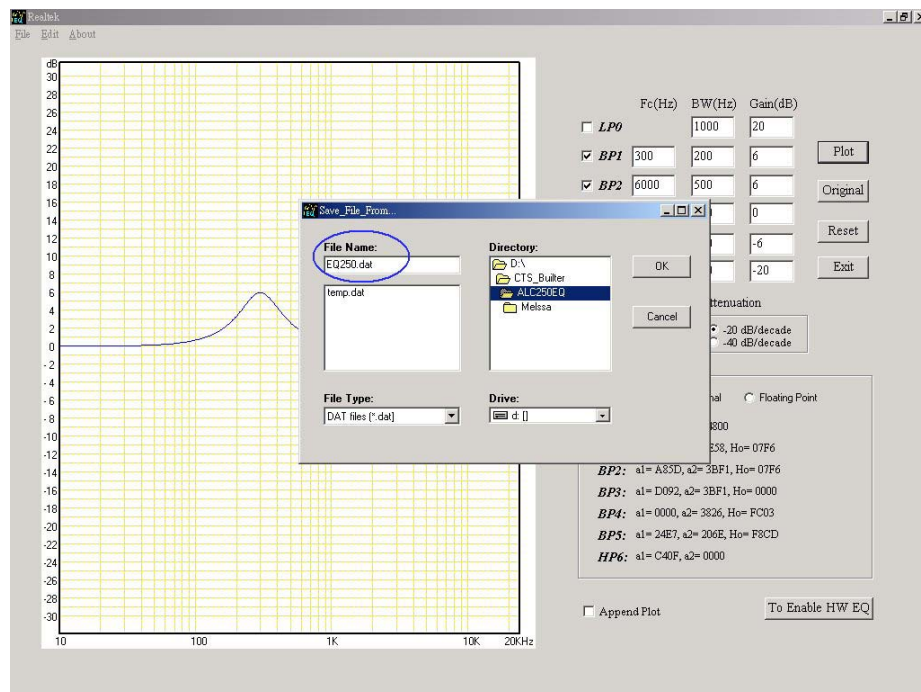


Figure 3-8. Setting was reset, the previous setting can be loaded by select 'File/Load Parameters from File' menu.

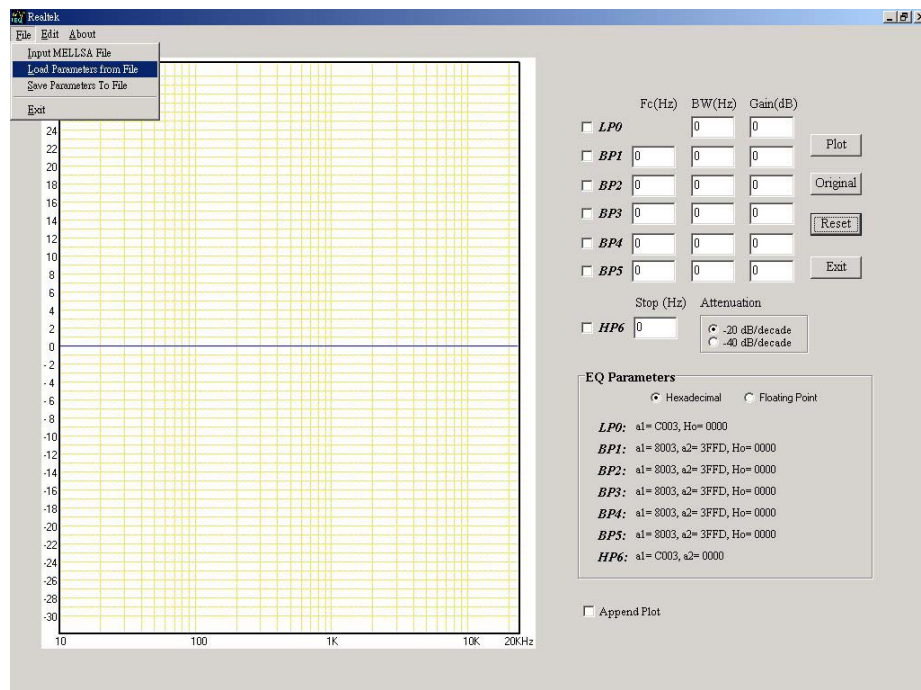


Figure 3-9. Choice the '*.dat' file, and press 'OK' button to load previous setting.

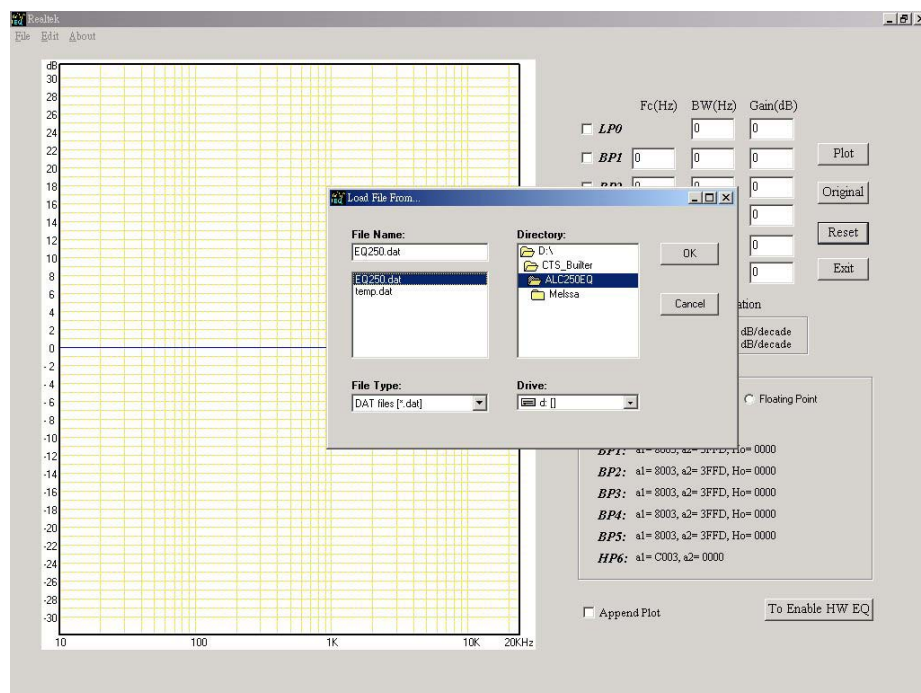


Figure 3-10. The previous curve and setting have been loaded.

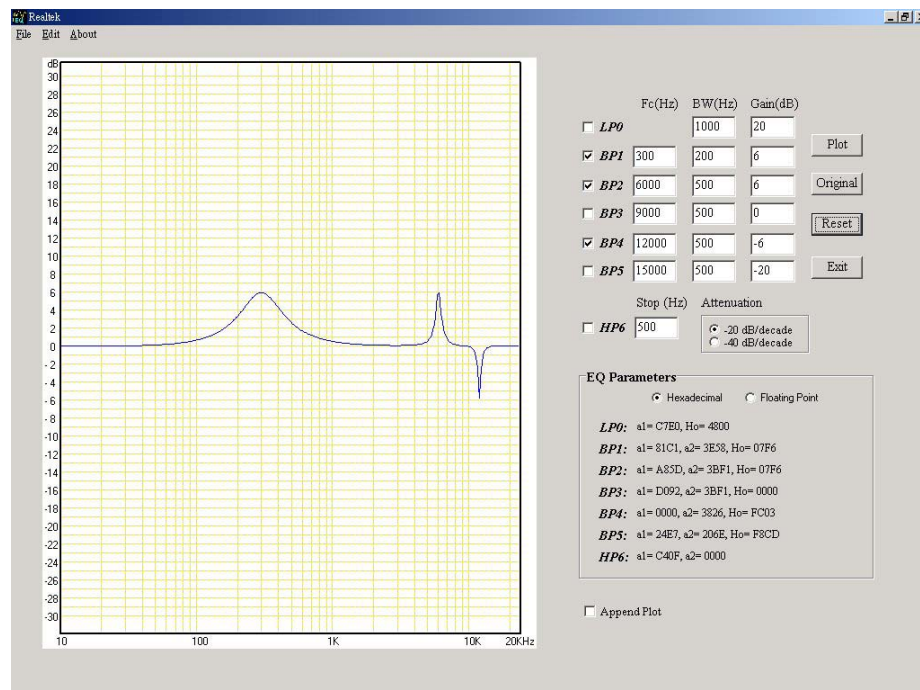


Figure 3-11. To simulate the speaker's frequency response supplied by speak manufacture. Set the simulated curve as the initial curve before start compensating. Select 'File/Set Current Plot as Original' menu to set current plot as initial curve.

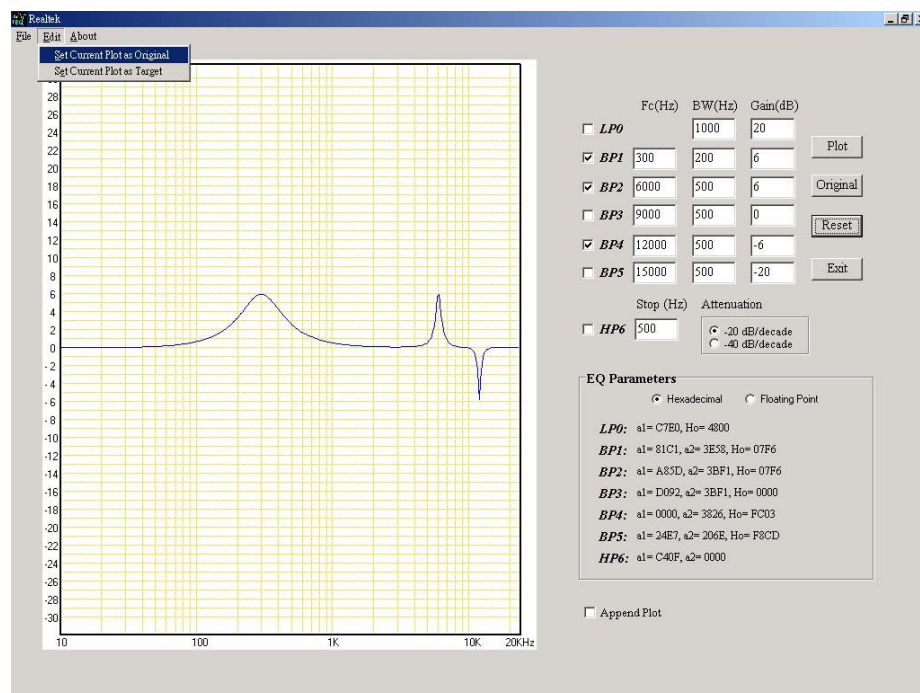


Figure 3-12. The initial curve will be the one you had set when each time the ‘Original’ button is pressed.

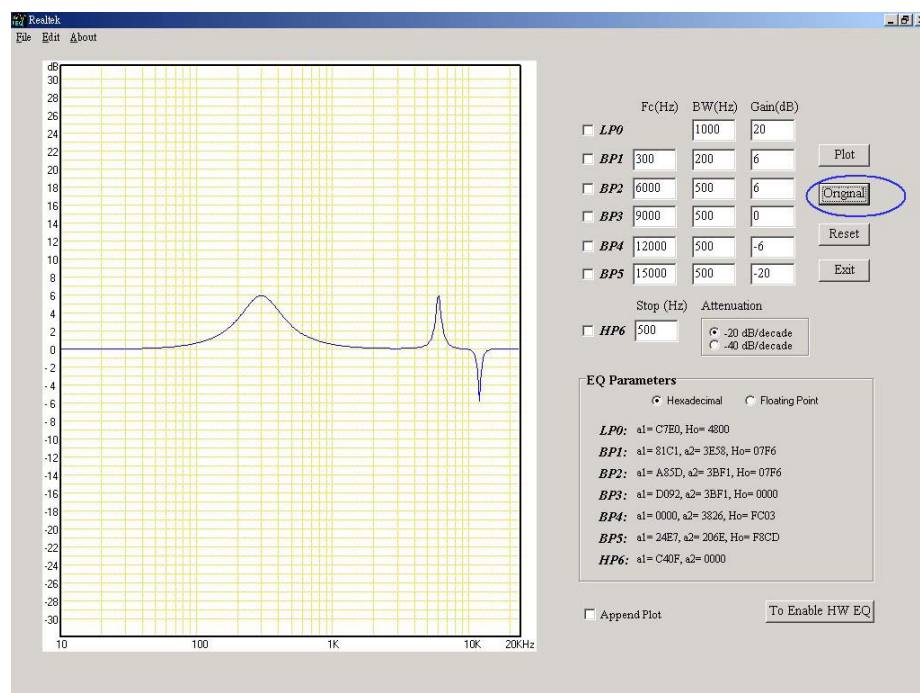


Figure 3-13. Another case may need to approach an expected target curve. Select ‘File\Input MELSSA File’ menu to input the expected target curve.

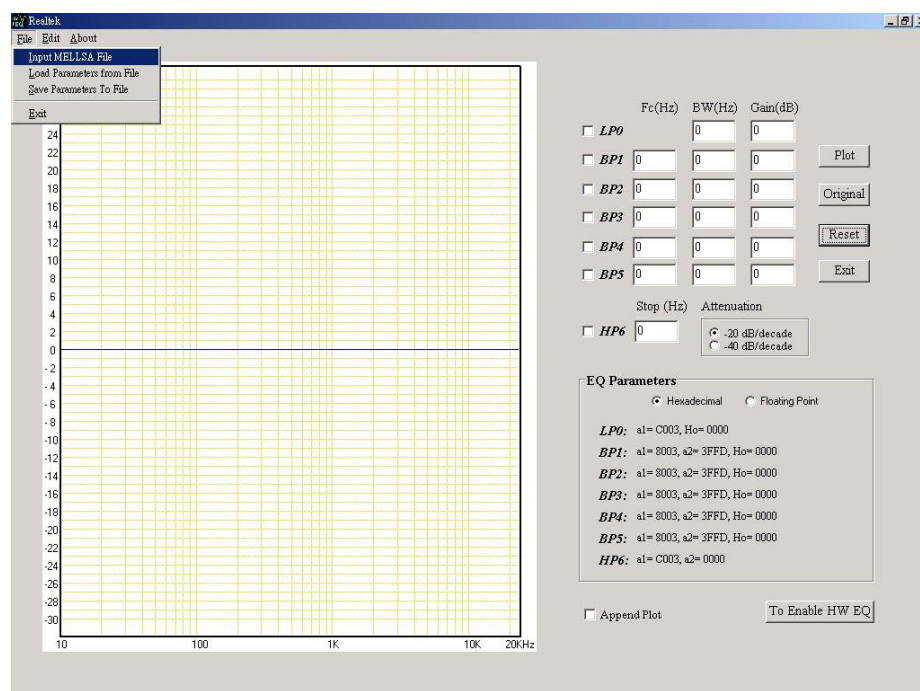


Figure 3-14. Select File 1 and frequency range, File 2 and its frequency range, then press 'OK' button.

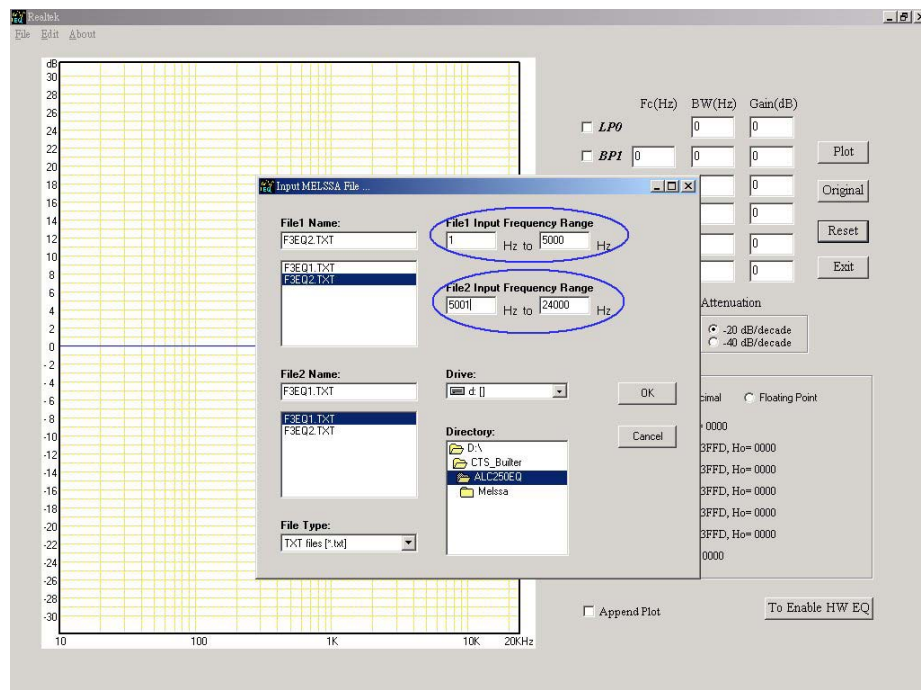


Figure 3-15. The expected target curve is input.

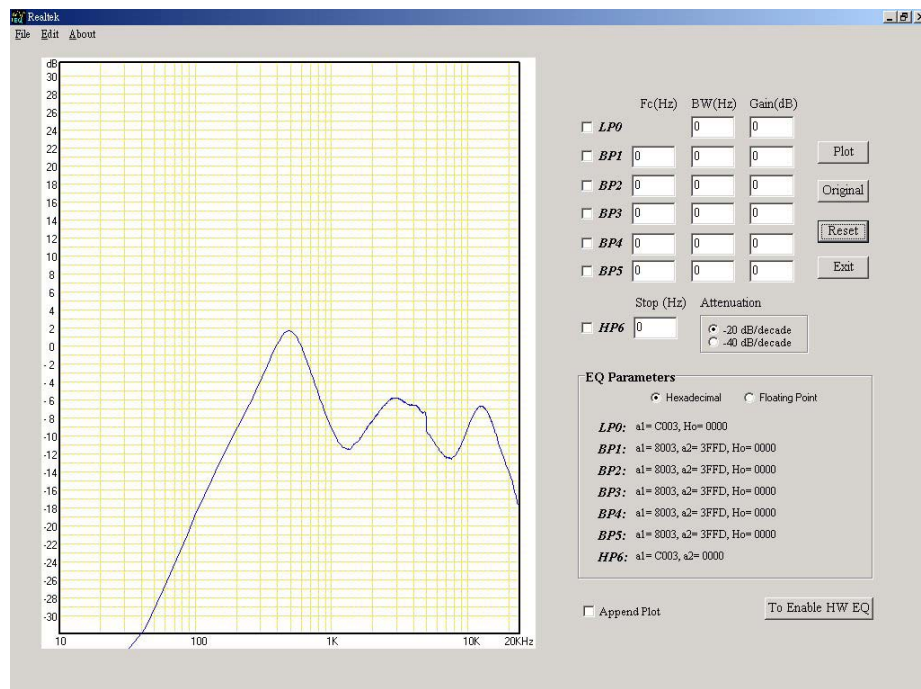


Figure 3-16. Select 'File\Set Current Plot as Target' menu to set it as an expected curve want to be approached.

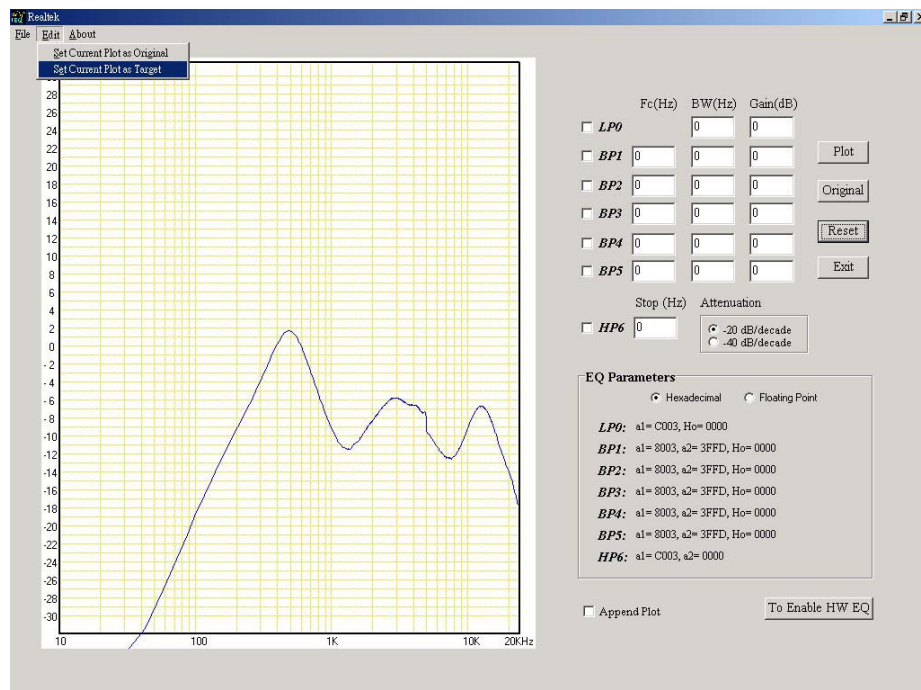


Figure 3-17. The expected curve will be drawn as reference in gray color. Now you can set the Filter characteristic to approach the reference curve.

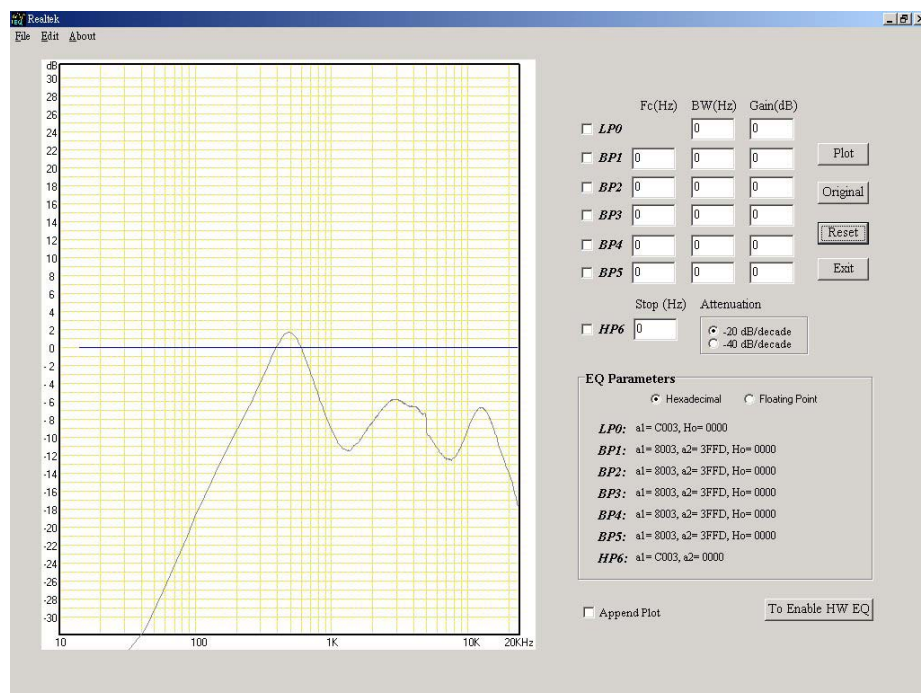


Figure 3-18. After simulate the frequency response, press ‘To Enable HW EQ’ button to directly enable the hardware equalizer in ALC250.

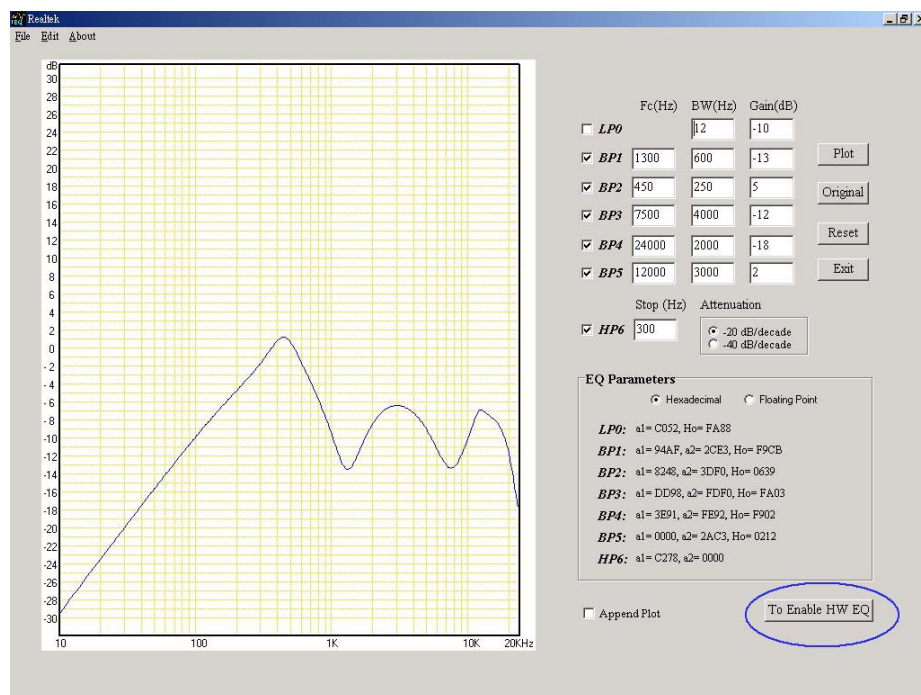
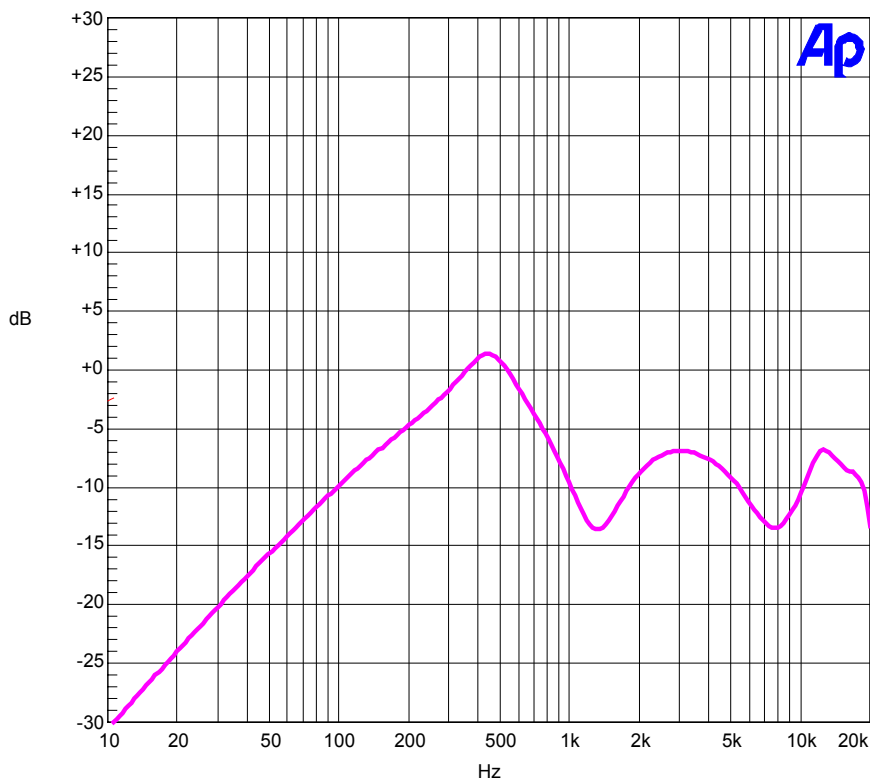


Figure 3-19 Audio Precision generates multi-tone signal from 10Hz to 20KHz through LINE-IN jack, Audio Precision analyzer scans the frequency response of output signal from LINE-OUT or HP-OUT jack, **the practical response does match the simulated curve!**



Realtek Semiconductor Corp.**Headquarters**

1F, No. 2, Industry East Road IX, Science-based
Industrial Park, Hsinchu, 300, Taiwan, R.O.C.
Tel : 886-3-5780211 Fax : 886-3-5774713

WWW: www.realtek.com.tw

Email: pcotech@realtek.com.tw
