应用笔记: AN01008

XMOS 特許一級代理商

------茂晶有限公司

國內服務據點：
深圳： Tel: 86-755-8828 5788-1207 Email: Shenzhen@gfei.com.hk
北京： Tel: 86-10-5126 6624 Email: Beijing@gfei.com.hk
上海： Tel: 86-21-54453155 Email: Shanghai@gfei.com.hk
武漢： Tel: 86-27-8730 6822, 8784 0783 Email: Wuhan@gfei.com.hk
青島： Tel: 86-532-8573 1420 Email: Qingdao@gfei.com.hk
成都： Tel: 86-028-85548390 Email: Chengdu@gfei.com.hk
廈門： Tel: 86-0592-5302668 Email: Xiamen@gfei.com.hk
海外服務據點：
香港： Tel: 852-3741 0662-2293 Email: Hongkong@gfei.com.hk
台灣： Tel: 886-2-89132200 Email: Service@gfei.com.hk
技術支援：
深圳： Tel: 86-755-8828 5788-1207 Email: royl@gfei.com.hk
上海： Tel: 86-21-54453155
北京： Tel: 86-10-8429 8668 Email: jackl@gfei.com.hk

增加 DSP 功能到 USB2.0 L1 音频参考设计

本应用笔记展示了如何将 DSP 处理集成到 XMOS USB 音频解决方案中，这个方案使用一个 Tile 的 L8 系列产品在单芯片上实现 USB 接口和音频处理功能。这里的 DSP 核利用 64 位单周期 MAC 指令实现音频处理功能，例如：EQ。
通过打开 sc_dsp_filters 模块从代码到支持二价滤波器（注 1）来提供 DSP 功能。这个模块可以生成任意级联二价过滤器，其在合适的系数在 dB 范围宽。这些系数是在编译时生成的头文件用于构建最终的可执行文件。 本应用笔记是基于 bigquad_xta_appnote_1v0 记录在上面的存储库里。

USB XUD 核需要至少 80 MIPS,这限制了这个例子使用 6 核。 这个设计中有一个核是可以使用的，所以我们将使用这个核来实现所有可用的 DSP 性能。 为简单起见， 这个功能要求产品运行在 500Mhz，并且不支持 MIDI 和 SPDIF 功能。

一个典型应用就是使用 USB 有源扬声器，在这里 DSP 功能是为了弥补扬声器外壳的物理局限性或者更高端应用于数字交叉处理的需求。

本应用笔记是在假设读者熟悉 XMOS XS1-L1 DSP 性能应用笔记（注 2）的基础上，提到的应用笔记是如何使用二价滤波器库测量性能。

[1] https://github.com/xcore/sc_dsp_filters
[2] http://www.xmos.com/published/an01011

1 DSP 功能和性能介绍

音频处理,单个二价过滤器可以用来提供一个单一的频率转换功能,如波峰上获得(音量增强)或一个峰值频率的功能。

使用多个级联过滤器可以用于生产更复杂的功能,比如多波段均衡器或数字交叉。

XS1-L 架构支持单个周期 32 * 32 MACC 指令产生真正的 64 位定点 DSP 操作结果是理想的。

单个二价过滤器需要 5 MACC 操作。在这个操作应用加载系数和检查溢出。在项目上从开源 XCore 库,一个优化实现的组装 sc_dsp_filters 库是可用的。

1.1 构建 DSP 到应用程序

通过拼接一个新的核之间的解耦和音频处理器，可将 DSP 核添加到 USB 音频参考设计中。

```
int main()
{
  chan c_sof;
  chan c_xud_out[NUM_EP_OUT];  /* Endpoint channels for XUD */
  chan c_xud_in[NUM_EP_IN];
  chan c_aud_ctl;
  chan c_aud_out;
  chan c_mix_out;
  .....
  {
    set_thread_fast_mode_on();
    dsp(c_mix_out, c_aud_out, p_button_a, p_button_b);
  }

  {
      thread_speed();
    /* Audio I/O (pars additional S/PDIF TX core) */
    audio(c_aud_out, null, null);
  }
  ....
```

按钮是 A 和 B 端口连接到 USB 音频 L1 音频上的按钮。

DSP 功能本身可以写在一个单独的文件——示例源代码是在附录中。

核应用程序显示在图的图表示 1:

DSP 核心实现基于混频器的功能从 USB 音频多通道设计解耦和音频核心之间的缓冲区数据。USB 音频设计有一个非常简单的设计解耦和输出之间缓冲数据的核心基于音频数据率。

音频核输出一个命令来表示它准备接收采样。核与它会用一个控制令牌显示采样率的变化,或与一个连接命令表明正常的音频数据。音频核心然后发送 1 采样每个输入通道和接收一个示例输出通道之前做的 I/O 操作端口。

随着 DSP 核心之间解耦和音频,它必须看起来像音频核心分离,分离核心音频。这种行为非常类似于混合机的 USB 音频多通道设计。

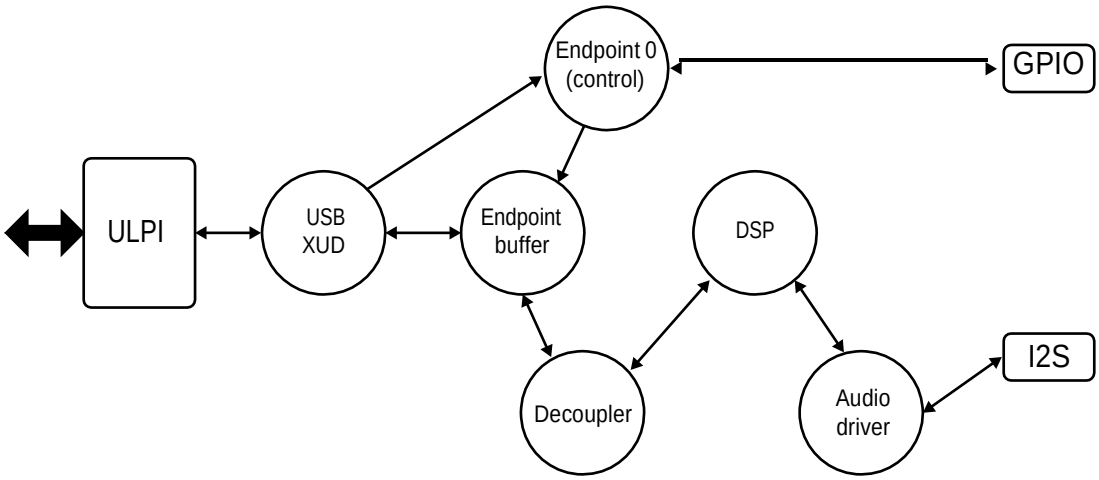

图 1: L1 USB 音频使用 DSP 功能软件核框图

因此,DSP 核心是基于这个功能。

反过来,它接受从音频核心请求数据,并将其传递到送出。然后从解耦处理握手并将其传递回音频。然后传递和接收样品的要求,每个方向的核心缓冲每个通道一个样品。DSP 应用于样例,收到分离之前存储在本地核心缓冲区。

DSP 只应用于数据从主机到音频输出(即输出通道),所以只有两个渠道 DSP 应用。这种影响有多少过滤器将适合在一个单一的核心。有一些指令来处理传入的音频数据,但这是最小的。

DSP 应用程序笔记（注 3）显示设备运行中的性能可用 6 核在 500 mhz - 80 MIPS 每核心和大约 2.5 MIPS 二价过滤器(或每立体声双通道 5 MIPS),我们可以支持 16 级联每通道的 2 通道的设计。这将允许非常细微的 EQ 或精确调谐的输出响应,以适应特定的扬声器应用程序。

下面的示例代码是基于使用 4 级联二价过滤器——这是很多演示效果显然没有重叠的过滤器可能会导致失真。

[3] http://www.xmos.com/published/an01011

2 样例应用

正如上面提到的,这个文档的附录包括 DSP 核心的完整源代码被包括在 USB 音频 L1 参考设计。对于这个示例添加了一些额外的特性使 DSP 的影响清晰可闻。
在开放源码库中使用的系数是计算使用非常基本的算法——然而定制系数由 DSP 专家可以被使用。
这个代码已经写入显示有什么可能 XMOS L1 设备。这包括生活改变 EQ 设置通过预设或改变一个滤波器的系数。DSP 是配置总是启用,但使用以下设置:

- 关闭 ODB 响应
- 低音增强 (Makefile 文件中的过滤器低于最低频率)
- 高音提升 (Makefile 文件中的过滤器高于最高频率)
- 低音增强，高音提升和峰值过滤器 (在 Makefile 文件中规定)

切换的选择是使用按钮切换上述预设。此外,按钮 B 可以独立选择的低音增强(3 设置,减少,正常和提高)。

二价过滤器的更多细节请参考文档提供的二价模块。
(https://github.com/xcore/sc_dsp_filters)

所选择的值在上面的例子中旨在创造一个轻松的音响效果,在播放歌曲的时候，可能会导致大面积的失真。

3 Measuring audio performance

在构建应用程序和下载到板子的时候，一些测量可以通过频谱分析工具(如光谱 PC 应用程序)。下面的情节,测试电脑设置玩扫描正弦波(20 hz - 20 khz)在-40 分贝基线输出电平通过 XMOS USB 音频设备配置了一个二进制 DSP 代码运行。USB 音频板的输出连接到一个模拟输入的记录电脑 EMU-24k 声卡。光谱是用来记录波形用峰值保持 FFT 绘制频率响应。输出电平设置为-40 分贝最小化任何剪裁引入的失真。红线是一个“持有”峰值线从被输入正弦波。

首先,让我们来检查频率响应在默认位置。虽然 biquad 启用过滤器,它们配置为产生一个平坦的频率响应:

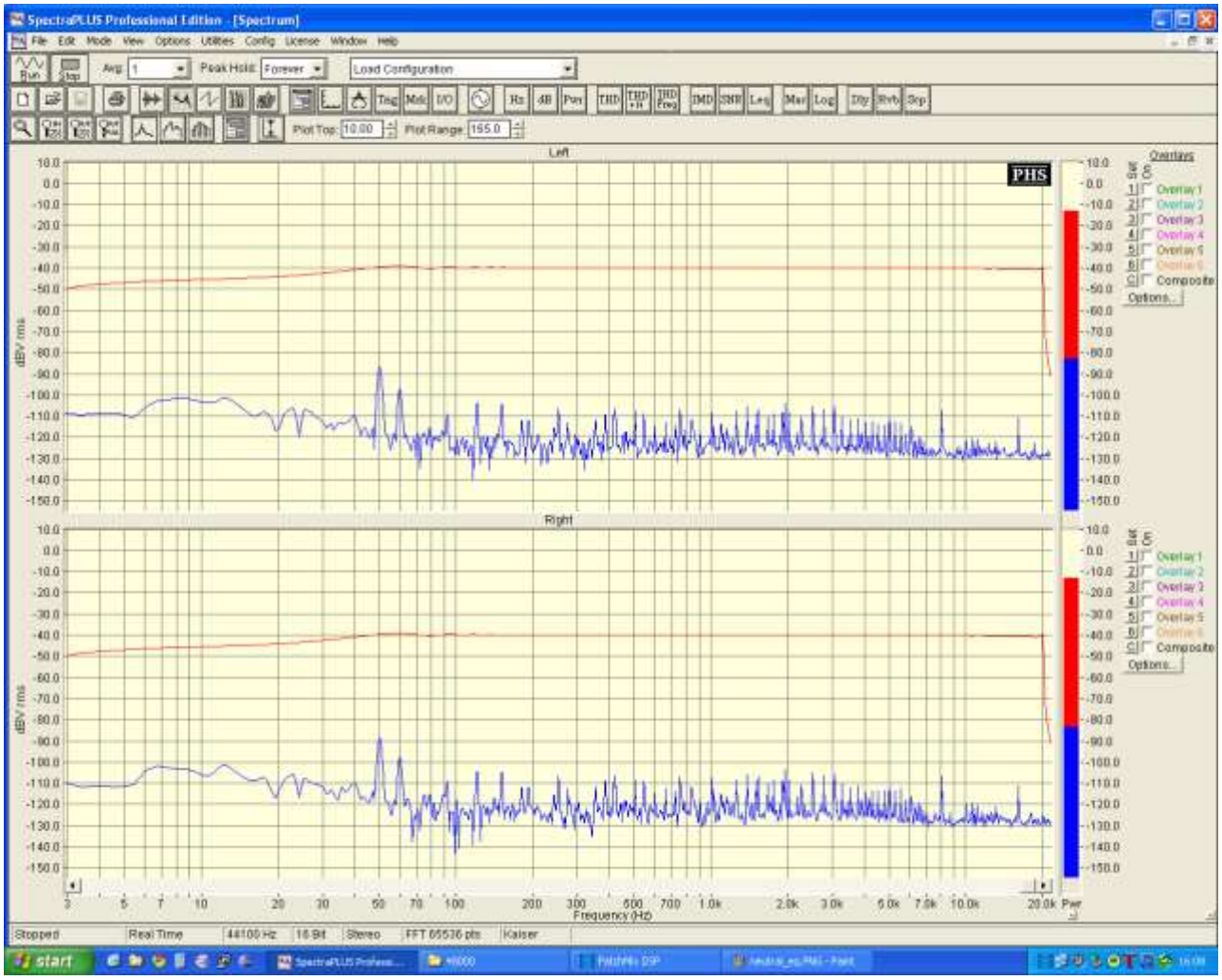

图 2: 频率响应为中性 EQ 设置

这看起来不错,基线-40 db,尽管有一个滚边低于 50 赫兹。接下来,按下按钮一旦选择低音增强架子过滤器,在 250 赫兹,+ 10

db 的相对水平。这将生成以下响应:

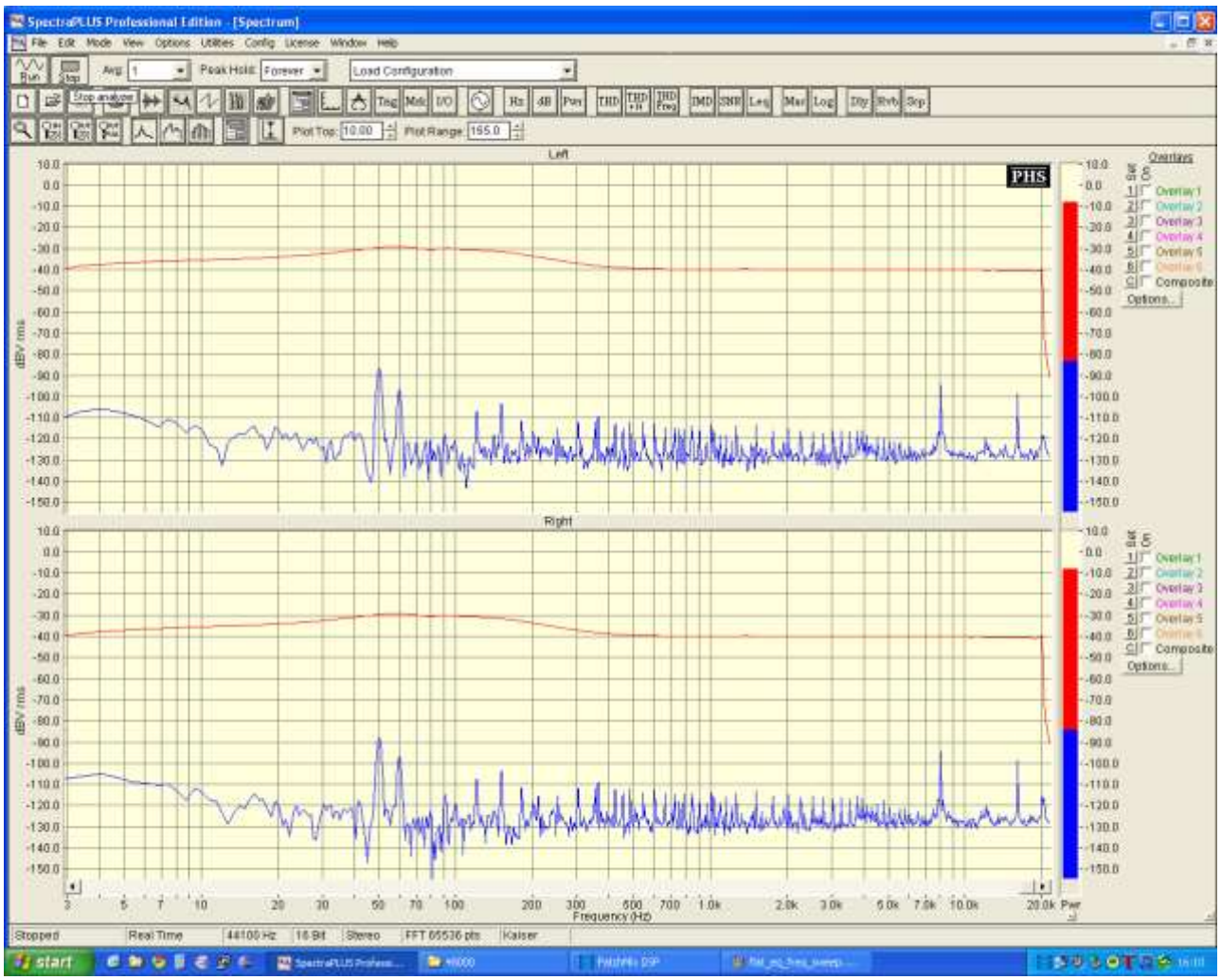

图 3: 频率响应的低音增强 EQ 设置

再一次,这看起来很符合我们的预期,提高低于 250 赫兹和滚边低于 50 赫兹一样最初的痕迹。第三个设置是三倍的提高又只有一个角落 2 khz 的频率和+ 10 db 的相对水平。这将生成以下:

进一步按的按钮选择低音增强,三倍的提高和峰值过滤器在 500 赫兹和 1000 赫兹:

多个过滤器的重叠使计算总体响应不简单。

3.1 Makefile

下面的 sc_dsp_filters 的目录内容，应该复制到 USB 音频源代码的目录中。

- module_cascading_biquad
- build_biquad_coefficients

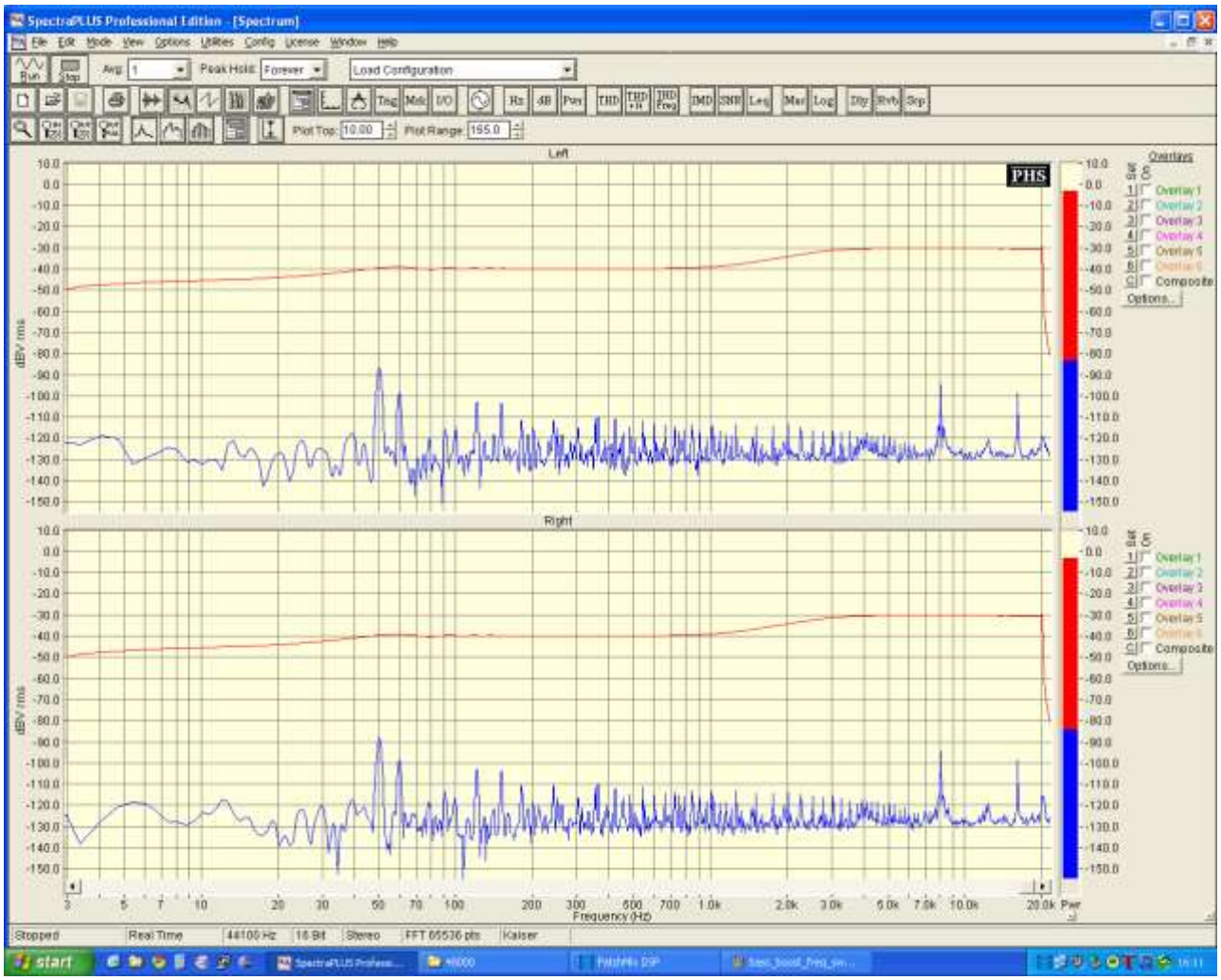

Figure 4: Frequency response for bass boost EQ setting

在 app_aud_l1 的 Makefile 文件中被修改生成需要的系数文件，并且在工程中包含二价模块下面的章节需要添加到所有的目标工程准中：

```
coefficients:
    make -f ../build_biquad_coefficients/Makefile \
              FILTER='-min -20 -max 20 -step 1 -bits 27 -low 250 -high 2000
              -peaking 500 1 -peaking 1000 1' \
              INCLUDEFILE=src/coeffs.h \
              XCFILE=src/coeffs.xc \
              CSVFILE=bin/response.csv

all: coefficients $(BIN_DIR)/usb_audio.xe
```

上述参数将建立一个二价级联 4 过滤器。更可以包含通过添加进一步的参数要求。参数描述的详细信息,请查阅包含在 sc_dsp_filter/doc directory 中的 biquad .rst 文件。

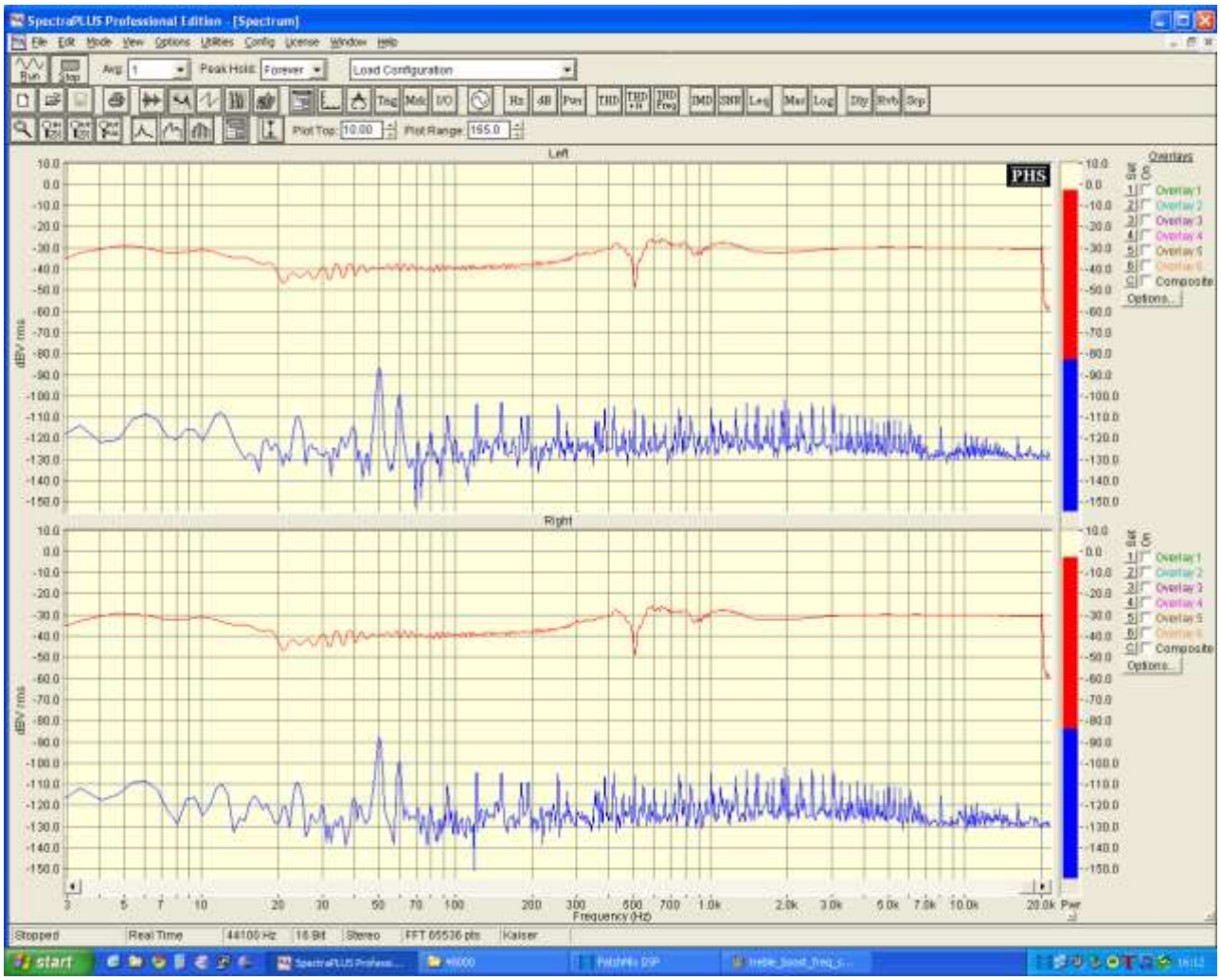

图 5: 所有过滤器的频率响应

3.2 源文件

从 DSP 的时序分析组件文档中，,粗略的经验法则是 5 MIPS / 二价过滤器 48 khz,2 通道音频。核速率 80 mips,这表明最高 16

biquads 每个通道是可能的。上面的例子 Makefile 文件中，当尝试最小化任何失真，使用 4 保持效果轻松声响。

核框图 图 1 说明了 DSP 核如何在之间的解耦和音频处理器设置。

4 结论

本应用笔记演示了如何将代码从开源 DSP 库集成到一个参考设计中，并且在只有一个简单而快速的重新编译代码的 DSP 参数，展示了如何快速的逐渐反复变化。

APPENDIX A - DSP function source code

```
// DSP core.

#include <xs1.h>
#include "devicedefines.h"
#include <print.h>
#include "coeffs.h"

extern int biquadAsm(int xn, biquadState &state);

// BANKS is defined in coeffs.h and generated by makefile.

// The change_dsp funcion simply iterates over some presets to demonstrate
// the sorts of possible effects.
// These are chosen for being easily noticed, rather than providing
// subtle sound quality improvements.
// The program assumes that the Bass shelf filter is the first value in the Makefile generation
// and the treble shelf is the second value. Putting the filters in a different order in the makefile
// will change the effects here.
#pragma unsafe arrays
void change_dsp(biquadState &bq, int
  eq_select){ switch(eq_select) {
  case 0: // "Off"
#pragma loop unroll
    for (int i =0; i < BANKS; i++)
      { bq.desiredDb[i] = 20 ; //zeroDb value
    }
    break;
  case 1: // Bass boost only
#pragma loop unroll
    for (int i =1; i < BANKS; i++)
      { bq.desiredDb[i] = 20; // zeroDb value
    }
    bq.desiredDb[0] = 30; // Set bass filter to boost
    break;
  case 2: // Treble boost only
#pragma loop unroll
    for (int i =0; i < BANKS; i++)
      { bq.desiredDb[i] = 20; // zeroDb value
    }
    //    bq.desiredDb[1] = 30; // set treble filter to boost
    break;
  case 3: // bass boost, treble boost + all peaking filters (All filters turned on).
    // As the number of filters defined in the makefile increase, it is
    // suggeted you reduce desiredDb towards 20 (no change) to minimise
    // distortion.
    // With 5 banks, a setting of 30 is okay for most music types
    // With 10 banks, a setting of 25 is okay for most music types
#pragma loop unroll
    for (int i =1; i < BANKS; i++)
      { bq.desiredDb[i] = 25;
    }
    break;
  default:
    break;
  }
}
```

```
//give/get functions based on mixer.xc then simplified.

#pragma unsafe arrays
void giveSamplesToHost(chanend c, const int samples[])
{

#pragma loop unroll
  for (int i=0;i<NUM_USB_CHAN_IN;i++)
  {
        int sample;
        sample = samples[i + NUM_USB_CHAN_OUT];
        outuint(c,sample);

  }
}

#pragma unsafe arrays
static void getSamplesFromHost(chanend c, int samples[], biquadState bs[])
{
  unsigned int l_samp[NUM_USB_CHAN_OUT];
#pragma loop unroll
  for (int i=0;i<NUM_USB_CHAN_OUT;i++)
    {
      int sample, x;
      /* Receive sample from decouple */
      l_samp[i] = inuint(c);
    }

  // timing fails if we do dsp in between receiving for both channels.
  // Instead receive into a local buffer, then do the dsp on both channels.
  // XTA times a single call to biquadAsm as 234 core cycles with 4 banks.
  // 48000 sample freq * 2 channels => ~100000 operations/second or 10us
  // allowance between samples.
  // With an 80 MIPS core speed (500MHz/6 cores), this gives ~36 cascaded biquads as the
  // maximum number possible at a max sample freq of 48kHz (ie BANKS=18).
#pragma loop unroll
  for (int i=0;i<NUM_USB_CHAN_OUT;i++)
    {
      int sample;
      sample = biquadAsm(l_samp[i],bs[i]);
          samples[i] = sample;
    }
}

#pragma unsafe arrays
void giveSamplesToDevice(chanend c, const int samples[])
{
#pragma loop unroll
    for (int i=0;i<NUM_USB_CHAN_OUT;i++)
    {
        int sample;
        sample = samples[i];
        outuint(c, sample);
    }
}

#pragma unsafe arrays
void getSamplesFromDevice(chanend c, int samples[])
{
#pragma loop unroll
    for (int i=0;i<NUM_USB_CHAN_IN;i++)
    {
        int sample;
        sample = inuint(c);
                samples[NUM_USB_CHAN_OUT+i] = sample;
  }
}
```

```
void dsp ( chanend c_audio,
           chanend c_decouple,
           in port p_button_a,
           in port p_button_b) {
   /* One larger for an "off" channel for mixer sources" */
  int samples[NUM_USB_CHAN_OUT + NUM_USB_CHAN_IN + MAX_MIX_COUNT + 1];

  timer ta, tb;
  unsigned time_a, time_b;
  unsigned button_a_val = 0, button_a_active = 1;
  unsigned biquad_offset = 0, button_b_active = 1, button_b_val = 0;
  unsigned bass_filter = 10;
  biquadState bs[NUM_USB_CHAN_OUT];

  // 20 is the central value (approx no-eq).
  // As values tend to the extremes, distortion will become more prevalent.
  // The spread of available values is defined in the Makefile where
  // the filter coefficients are set.
  // If a smaller coefficient array is used, these values will need to be adjusted.
  initBiquads(bs[0], 20);
  initBiquads(bs[1], 20);
  set_port_inv (p_button_a);
  set_port_inv (p_button_b);
  // zero samples buffer.
  for (int i=0;i<NUM_USB_CHAN_OUT + NUM_USB_CHAN_IN + MAX_MIX_COUNT;i++)
    {
      samples[i] = 0;
    }

  while(1){ //Implements channel protocol between decouple and audio cores
      inuint(c_audio);                    //Get sample request from audio
      outuint(c_decouple, 0);             //Send sample request to decouple
      if(testct(c_decouple)){             //Test for sample frequency change .
          command = inct(c_decouple);     //Get the CT and command - See commands.h
          value = inuint(c_decouple);     // Get command value from decouple core - normally SR value
          outct (c_audio, command);       //Send control token to audio (SR change)
          outuint (c_audio, value);       //Now send value to audio
          chkct (c_audio, XS1_CT_END);    //wait for handshake
          outct (c_decouple, XS1_CT_END); //Forward handshake to decouple
      }

      else{  // Normal audio loop
          underflow = inuint(c_decouple);   // Get confirmation from decouple indicating we're ready
          outuint (c_audio, underflow);     // Pass it on to audio

          getSamplesFromDevice (c_audio, samples);//Always get input samples from device and send to audio
          if (!underflow) giveSamplesToDevice (c_audio, samples); //Only send audio if not in underflow
          giveSamplesToHost (c_decouple, samples);//Always send samples to decouple
          if (!underflow) getSamplesFromHost (c_decouple, samples, bs);//Only get samples from decouple if no
            ↪ underflow
      }
    }
  }
```

```
    // The section below demonstrates how a user interface can
    // be used to change the EQ settings on the fly using the A and B buttons
    // on the L1 reference board.

    // Sample the buttons values with a timeout for debounce.
    // Button A changes the Bass shelf filter through 3 settings (cut, normal, boost)
    p_button_a :> button_a_val;
    if (button_a_val && button_a_active)
      { bass_filter = bs[0].desiredDb[0];
      bass_filter += 10;
      if (bass_filter > 30)
        { bass_filter = 10;
      }
      bs[0].desiredDb[0] = bass_filter;
      bs[1].desiredDb[0] = bass_filter;
      ta:>time_a;
      button_a_active = 0;
    }

    // Button B iterates over the 4 DSP presets
    // B will clear any changes made with button A.
    p_button_b :> button_b_val;
    if (button_b_val && button_b_active)
      { biquad_offset ++;
      biquad_offset &= 3;
      change_dsp(bs[0], biquad_offset);
      change_dsp(bs[1], biquad_offset);
      tb:>time_b;
      button_b_active = 0;
    }

    // 0.5s timeout on button press for B, 0.3s on A
    // Holding the buttons down will then iterate over the possible settings with this time period
    // between changes.
    select {
    case (button_b_active == 0 ) => tb when timerafter (time_b + 50000000) :> int _:
      button_b_active = 1;
      break;
    case (button_a_active == 0 ) => ta when timerafter (time_a + 30000000) :> int _:
      button_a_active = 1;
      break;
     default:
      break;
    }
  }
}
```

XMOS 特許一級代理商

茂晶有限公司
國內服務據點：
深圳：Tel: 86-755-8828 5788-1207 Email: Shenzhen@gfei.com.hk
北京：Tel: 86-10-5126 6624 Email: Beijing@gfei.com.hk
上海：Tel: 86-21-54453155 Email: Shanghai@gfei.com.hk
武漢：Tel: 86-27-8730 6822, 8784 0783 Email: Wuhan@gfei.com.hk
青島：Tel: 86-532-8573 1420 Email: Qingdao@gfei.com.hk
成都：Tel: 86-028-85548390 Email: Chengdu@gfei.com.hk
廈門：Tel: 86-0592-5302668 Email: Xiamen@gfei.com.hk
海外服務據點：
香港：Tel: 852-3741 0662-2293 Email: Hongkong@gfei.com.hk
台灣：Tel: 886-2-89132200 Email: Service@gfei.com.hk
技術支援：
深圳：Tel: 86-755-8828 5788-1207 Email: royl@gfei.com.hk
上海：Tel: 86-21-54453155 Email: SampsonL@gfei.com.hk
北京：Tel: 86-10-8429 8668 Email: jackl@gfei.com.hk