PHASER AUDIO EFFECT

Gabriel Rivas • July 4, 2011 • 2 comments • Coded in C

This phaser implementation was created using cascaded 2nd order variable notch filters to take advantage of the strong phase shift produced near the center notch frequency. The Q parameter determines the frequency band for the notch filter, and for lower Q values the frequency band will be wider and the phase shift effect around the notch frequency will be more pronounced.

To generate the output the input and the phase shifted signal are scaled and added, this will produce phase cancellations or enhancements in some frequencies present on the input signal as the center frequency notch is varied in a frequency range.

In this code is used the code in my previous entry at http://www.dsprelated.com/showcode/173.php

as a building block to implement the variable notch filter stages.

Here is a sample of how it sounds like:

http://www.youtube.com/watch?v=zTANtpuLD8s

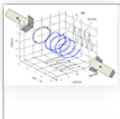
Notice that this code is not optimized to any particular DSP architecture but you can use it as a reference code to further optimize it.

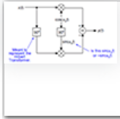
I hope you find this useful.

```
/*
Phaser audio effect:
```

Blogs - Hall of Fame

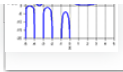
A FIXED-POINT INTRODUCTION BY EXAMPLE
Chrisopher Felton


HANDLING SPECTRAL INVERSION IN BASEBAND PROCESSING
Eric Jacobsen


UNDERSTANDING THE PHASING METHOD OF SINGLE SIDEBAND MODULATION
Rick Lyons

AN INTERESTING

```
Xin                                                                       Yout
------------------------------[dir_mix]--------------------->[+]--------->
      |                                                        ^
      |                                                        |
      |-->[VNS1]-->[VNS2]-->[VNS3]...-->[VNSN]-->[pha_mix]------

            ^        ^        ^           ^
            |        |        |           |
            |--------|--------|---...------
            |
          [LFO]

VNS = Variable notch stage

*/

#include "br_iir.h"
#include "Phaser.h"

/*This defines the phaser stages
that is the number of variable notch blocks
*/
#define PH_STAGES 20

static short center_freq;    /*Center frequency counter*/
static short samp_freq;      /*Sampling frequency*/
static short counter;        /*Smaple counter*/
static short counter_limit;  /*Smaple counter limit*/
static short control;        /*LFO Control*/
static short max_freq;       /*Maximum notch center frequency*/
static short min_freq;       /*Minimum notch center frequency*/
static double pha_mix;       /*Filtered signal mix*/
static short f_step;         /*Sweep frequency step*/
static double dir_mix;       /*Direct signal mix*/
static struct br_filter H[PH_STAGES]; /*Array of notch filters stages*/

/*
This funtion initializes the phaser control variables
and the variable notch filter coefficients array
*/
void Phaser_init(short effect_rate,short sampling,short maxf,short minf,short Q,do
    /*Initialize notch filter coefficients set array*/
    br_iir_init(sampling,gainfactor,Q,freq_step, minf);

    /*Initializes the phaser control variables*/
    center_freq = 0;
    samp_freq = sampling;
    counter = effect_rate;
    control = 0;
    counter_limit = effect_rate;

    /*Convert frequencies to integer indexes*/
    min_freq = 0;
    max_freq = (maxf - minf)/freq_step;

    pha_mix = pha_mixume;
    f_step = freq_step;
    dir_mix = dmix;
}

/*
This function does the actual phasing processing
1. It takes the input sample and pass it trough the
cascaded notch filter stages
2. It takes tha output of the cascaded notch filters
and scales it, scales the input sample and generate
the output effect sample.
*/
double Phaser_process(double xin) {
    double yout;
    int i;

    yout = br_iir_filter(xin,&H[0]);

    for(i = 1; i < PH_STAGES; i++) {
        yout = br_iir_filter(yout,&H[i]);
    }

    yout = dir_mix*xin + pha_mix*yout;

    return yout;
}
```



```
/*
This function makes vary the center notch frequency
in all the cascaded notch filter stages by a simulated
triangle wave LFO that goes up and down
*/
void Phaser_sweep(void) {
    int i;

    if (!--counter) {
        if (!control) {
            center_freq+=f_step;

            if (center_freq > max_freq) {
                control = 1;
            }
        }
        else if (control) {
            center_freq-=f_step;

            if (center_freq == min_freq) {
                control = 0;
            }
        }
        for(i = 0; i < PH_STAGES; i++) {
            br_iir_setup(&H[i],center_freq);
        }
        counter = counter_limit;
    }
}

/************

Phaser.h

***********/

#ifndef __PHASER_H__
#define __PHASER_H__

extern void Phaser_init(short effect_rate,short sampling,short maxf,short minf,sho
extern double Phaser_process(double xin);
extern void Phaser_sweep(void);

#endif
```

Quick Links

- Home
- Blogs
- Forums
- Courses
- Tutorials
- Books
- Free Books
- Free PDFs
- Code
- comp.dsp

About DSPRelated.com

- Advertise
- Contact
- Privacy Policy
- Terms of Service
- Cookies Policy

Social Networks

The Related Media Group

- EmbeddedRelated.com
- DSPRelated.com
- Electronics-Related.com
- FPGARelated.com