C programmes used in the product development of the BitVia Media Player

Sine wave generator

```
#include <stdio.h>
#include <stdint.h>
#include <math.h>

#define FILESIZE 48000   // number of samples
#define OMEGA 1000      // wave frequency (Hz)
#define SCALETIME 48000 // sampling frequency (Hz)
#define AMPLITUDE 1000  // sine wave amplitude
#define PI 3.141592654
#define CLIPPING_VALUE 800
#define MOD_FREQUENCY 5
#define DELAY_MS 10.0

int main ()
{
    FILE *inputData;
    FILE *TBData;
    FILE *Tremolo;
    FILE *Delay;
    const char *inputFilePath = "C:\\Users\\Matt\\Documents\\C_programming\\GroupProjectFiles\\sine_1kHz.
dat";
    const char *TBFilePath = "C:\\Users\\Matt\\Documents\\C_programming\\GroupProjectFiles\\sine_1kHz_CLI
PPED.dat";
    const char *TremoloFilePath = "C:\\Users\\Matt\\Documents\\C_programming\\GroupProjectFiles\\tremolo_
1kHz.dat";
    const char *DelayFilePath = "C:\\Users\\Matt\\Documents\\C_programming\\GroupProjectFiles\\delay_1kHz
.dat";

    double time = 0.0;
    double wave = 0.0;
    double carrierWave = 0.0;
    double maxClip = CLIPPING_VALUE;
    double minClip = -CLIPPING_VALUE;
    const double delayTime = (DELAY_MS*1000) / SCALETIME;
    int i = 0; // iteration control

    inputData = fopen(inputFilePath, "w");
    TBData = fopen(TBFilePath, "w");
    Tremolo = fopen(TremoloFilePath, "w");
    Delay = fopen(DelayFilePath, "w");

    /* Load data into file */
    for(i=0;i<FILESIZE;i++)
    {
        if(i>0)
            time = (double)(i)/SCALETIME;
            wave = AMPLITUDE*sin(2*PI*OMEGA*time);
            carrierWave = 1*sin(2*PI*MOD_FREQUENCY*time);
```

```
        // Delay output
        fprintf(Delay, "%lf\t%lf\n", time+delayTime, wave);

        // Tremolo
        fprintf(Tremolo, "%lf\t%lf\n", time, carrierWave*wave);

        // Clipping
        if(wave < maxClip && wave > minClip)
        {
            fprintf(TBData, "%lf\t%lf\n", time, wave);
        }
        else if (wave < minClip)
        {
            fprintf(TBData, "%lf\t%lf\n", time, minClip);
        }
        else if (wave > maxClip)
        {
            fprintf(TBData, "%lf\t%lf\n", time, maxClip);
        }

        // Sine wave output
        fprintf(inputData, "%lf\t%lf\n", time, wave);
    }

    fclose(inputData);
    fclose(TBData);
    fclose(Tremolo);
    fclose(Delay);

    return 0;
}
```

Triangle wave generator

```c
#include <stdio.h>
#include <conio.h>
#include <stdint.h>
#include <stdlib.h>
#include <math.h>

#define FILESIZE 48000    // number of samples
#define OMEGA 40           // wave frequency (Hz)
#define SCALETIME 48000   // sampling frequency (Hz)
#define AMPLITUDE 1.0     // wave amplitude
#define PI 3.141592654

int main ()
{
    FILE *triangleData;
    FILE *GNUPLOT_TRIANGLE;
    FILE *GNU_TREMOLO;
    FILE *tremolo_Ref;
    const char *triangleFilePath = "C:\\Users\\Matt\\Documents\\C_programming\\GroupProjectFiles\\triangl
eWave_TB.dat";
    const char *GNU_TRIANGLE_PATH = "C:\\Users\\Matt\\Documents\\C_programming\\GroupProjectFiles\\triang
leWave_GNU.dat";
    const char *GNU_TREMOLO_PATH = "C:\\Users\\Matt\\Documents\\C_programming\\GroupProjectFiles\\tremolo
_GNU.dat";
    const char *tremolo_Ref_PATH = "C:\\Users\\Matt\\Documents\\C_programming\\GroupProjectFiles\\tremolo
_Ref.dat";

    double triangleArray[FILESIZE];
    double time = 0.0;
    double sample = 0.0;
    double sinSample, sinTriOut = 0.0;
    const double dataConvert = 16777216; // 2^24 (convert fractional values to integers)
    int i = 0; // iteration control

    triangleData = fopen(triangleFilePath, "w");
    GNUPLOT_TRIANGLE = fopen(GNU_TRIANGLE_PATH, "w");
    GNU_TREMOLO = fopen(GNU_TREMOLO_PATH, "w");
    tremolo_Ref = fopen(tremolo_Ref_PATH, "w");

    /* Generate a triangle wave */
    for(i=1;i<FILESIZE+1;i++)
    {
        if(i>0)
            {time = (double)(i)/SCALETIME;}

        // (2*100/PI()) * ASIN( SIN( ( (2*PI()) /20) * (ROW()-1) ) )
        // https://www.quora.com/What-is-the-code-to-create-a-triangle-waveform-in-Excel-using-VBA-
coding-given-the-frequency-of-10kHz-and-amplitude-of-%2B-1
        sample = (2*(AMPLITUDE/PI))*asin(sin((2*PI*OMEGA)*(time-1)));  // modulating wave

        // Tremolo output for GNUplot
        sinSample = 1000*sin(2*PI*1000*time); // audio input - 1 kHz tone
```

```
        fprintf(tremolo_Ref, "%lf\t%1.10lf\n", time, sinSample); // include time column (i) for gnuplot
        sinTriOut = sinSample * sample; // tremolo output
        fprintf(GNU_TREMOLO, "%lf\t%1.10lf\n", time, sinTriOut);

        // Check the triangle wave output
        fprintf(GNUPLOT_TRIANGLE, "%lf\t%1.10lf\n", time, sample); // include time column (i) for gnuplot

        // create the file for the hardware ROM
        if(sample >= 0)
        {
            fprintf(triangleData, "%d\n", (int)(sample*dataConvert)); // fixed point samples for TB
            //fprintf(triangleData, "%lf %d\n", time, (int)round(sample)); // with time
            //fprintf(triangleData, "%d => X\"%06X\", \n", i-
1, (int)(sample*dataConvert)); // samples for VHDL testbench
        }
        if(sample < 0)
        {
            fprintf(triangleData, "%d\n", (int)(sample*dataConvert)); // fixed point samples for TB
            //fprintf(triangleData, "%lf %d\n", time, (int)round(sample)); // with time
            //fprintf(triangleData, "%d => X\"%6X\", \n", i-
1, (int)(sample*dataConvert)); // samples for VHDL testbench
        }

    }

    fclose(triangleData);
    fclose(GNUPLOT_TRIANGLE);
    fclose(GNU_TREMOLO);
    fclose(tremolo_Ref);

    return 0;
}
```

Add time column

```c
#include <stdio.h>
#include <stdint.h>
#include <math.h>

/*
    // copy files from the xsim directory, into the local dir for
    // this C programme.
    // The reference tone and output tone will then be copied into
    // a file with a time base, which matches the sampling frequency.
    // This is then used to plot the output on an x,y graph.
*/

#define SAMPLE_FREQ 48000 // sampling frequency (Hz)
#define isRef 1 // set 1 if outputting reference file, 0 for output tone

#if isRef == 0
    #define IN_FILENAME "outputReference.dat"
    #define OUT_FILENAME "outputReference_GNU.dat"
#elif isRef == 1
    #define IN_FILENAME "outputTone_1kHz_TB.dat"
    #define OUT_FILENAME "outputTone_1kHz_GNU.dat"
#elif isRef == 2
    #define IN_FILENAME "clippedTriangle_TB.dat"
    #define OUT_FILENAME "clippedTriangle_GNU.dat"
#elif isRef == 3
    #define IN_FILENAME "tremoloOutput_TB.dat"
    #define OUT_FILENAME "tremoloOutput_GNU.dat"
#else
    printf("Error: Incorrect filename \n\n");
#endif

int main ()
{
    FILE *inputData;
    FILE *outputData;
    const char *inputFilePath = "C:\\Users\\Matt\\Documents\\C_programming\\GroupProjectFiles\\"IN_FILENA
ME;
    const char *outputFilePath = "C:\\Users\\Matt\\Documents\\C_programming\\GroupProjectFiles\\"OUT_FILE
NAME;

    const double time_Fs = 0.0000208333;    // 1/Fs - sampling frequency time period to 10dp
    double time = 0.0;  // captures the time base for the output waves
    int copySample = 0; // move sample from input file to output file

    inputData = fopen(inputFilePath, "r");
    outputData = fopen(outputFilePath, "w");

    /* Load data into file */
    printf("Creating new file\n");
    printf("Sample frequency: %d Hz\n", SAMPLE_FREQ);
    while(!feof(inputData))
    {
```

```
        time = time + time_Fs;

        fscanf(inputData, "%d", &copySample);
        fprintf(outputData, "%.10lf %d\n", time, copySample); // for GNU plot, add time in

    }

    fclose(inputData);
    fclose(outputData);

    printf("Finished\n");

    return 0;
}
```

Convert hex values

```
#include "logo.h"
#include <stdio.h>
#include <stdint.h>

#define SCREEN_SIZE 1152000 // (480 x 800 x 3) - must be multiple of 3

int main () {

    // reorder //
    // char rawData[SCREEN_SIZE] = { 0xFA, 0xFB, 0xFC, 0xA0, 0xA1, 0xA2, 0xD0, 0xD1, 0xD2, 0xE0, 0xE1, 0x
E2 };
    static uint32_t rawData_reordered[SCREEN_SIZE] = {0};
    uint32_t RED,BLUE,GREEN = 0;
    // group //
    uint32_t char1, char2, char3 = 0;
    static uint32_t rawData_grouped[SCREEN_SIZE/3] = {0};
    uint32_t pixel_value = 0;
    const uint32_t MASK = 0x000000FF;
    // general //
    int i, y = 0; // iteration control

/*****************************************************************************************/
/*
    Reorder chars
    The 24-bit BMP file stores pixel value data in the order of BLUE GREEN RED.
    The TFT frame buffers require RED BLUE GREEN.
    This section of code takes each set of three bytes and reorders them, storing
    the results in a new array: rawData_reordered.
*/
/*****************************************************************************************/

    /* Reorder RGB chars */
    printf("Reorder RGB values...");
    for(i=0;i<SCREEN_SIZE;i+=3)
    {
        // extract RGB values from array and convert
        // them into unsigned 32-bit integers
        BLUE = (uint32_t)rawData[i];
        GREEN = (uint32_t)rawData[(i+1)];
        RED = (uint32_t)rawData[(i+2)];

        // remove any signed extensions, leaving just
        // the 8-bit values
        BLUE = BLUE & MASK;
        GREEN = GREEN & MASK;
        RED = RED & MASK;

        // input the converted values into the new array
        // in the order of RED BLUE GREEN
        rawData_reordered[i] = RED;
        rawData_reordered[(i+1)] = BLUE;
        rawData_reordered[(i+2)] = GREEN;
```

```
    }
    printf("successful\n");

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

/*****************************************************************************************/
/*
    Group chars
*/
/*****************************************************************************************/
    for(i=0;i<SCREEN_SIZE;i+=3)
    {
        // pass the RGB values into holders ready for
        // concatenation into a single pixel value
        RED = rawData_reordered[i];
        BLUE = rawData_reordered[(i+1)];
        GREEN = rawData_reordered[(i+2)];

        // remove any unwanted information from the RGB
        // values (protects against unwanted sign extensions)
        // And bitshift values to the correct position for
        // the final value
        RED = (RED&MASK) << 16;
        BLUE = (BLUE&MASK) << 8;
        GREEN = GREEN&MASK;

        // us a logical OR operation to concatenate values
        // into a single pixel value
        pixel_value = (RED | BLUE | GREEN);

        // load values into a new array of uint32
        // array position is incremented by one each
        // iteration of the for-loop
        rawData_grouped[y] = pixel_value;
        y++;
    }
/*****************************************************************************************/

/*****************************************************************************************/
/*
    Write to file
*/
/*****************************************************************************************/

    FILE *outputData;
    const char *outputFilePath = "C:\\Users\\Matt\\Documents\\C_programming\\GroupProjectFiles\\imageArra
y.txt";
    outputData = fopen(outputFilePath, "w");

    /* Load data into file */
    printf("Printing array...");
    fprintf(outputData, "uint32_t BitVia_Logo[] = { \n\t" );
    for(i=0;i<SCREEN_SIZE/3;i++)
    {
```

```
        fprintf(outputData, "0x%08X, ", rawData_grouped[i] );
        if(9 == i%10)
        {
            fprintf(outputData, "\n\t");
        }
    }
    fprintf(outputData, "\n};\n");

    fclose(outputData);

    printf("successful\n");


    return 0;
}
```