UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Real-time audio signal processing

Cristescu Vlad-Andrei

Contents

1. Introduction

Digital Signal Processing (DSP) is a branch of engineering that focuses on the analysis, modification, and synthesis of signals using digital techniques. Signals are functions that convey information, and they can be either continuous (analog) or discrete (digital). In DSP, we primarily deal with digital signals, which are sequences of numbers that represent sampled values of a continuous signal.

Real-time audio filtering is the process of applying digital filters to audio signals in real time, meaning that the processing is done simultaneously with the signal's acquisition and playback. This is crucial in applications such as live sound reinforcement, audio recording, broadcasting, and communication systems.

Digital Signal Processors (DSPs) are specialized microprocessors designed specifically for efficiently executing signal processing algorithms. They are widely used in applications that require real-time processing of audio, video, radar, sonar, and other data types. The ADSP-BF533, part of Analog Devices' Blackfin family, is a popular DSP for embedded applications due to its high performance and low power consumption.

These are designed to handle intensive mathematical operations efficiently, which are common in signal processing algorithms. The hardware architecture of DSPs is optimized to execute operations such as Multiply-Accumulate (MAC), decimation, and convolution at high speed due to its specialized arithmetic units, parallelism and pipelining, efficient memory architecture, circular buffering and so on.

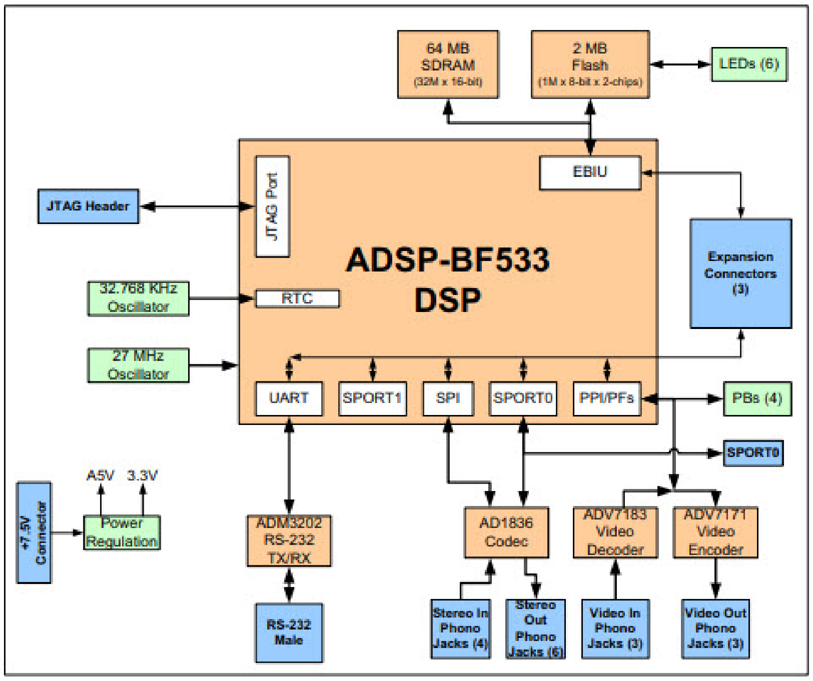

Figure 1: ADSP-BF533 Block Diagram

2. Theoretical Fundamentals

The starting point of this project is the C_Talkthrough_TDM.prj project example provided by Analog Devices. In its given form, the program simply copies the signal from the input and sends it straight to the output after ensuring all the necessary initializations and setups. The goal of this project is to somehow process this signal while making sure that the computation time does not exceed the time interval between 2 consecutive samples.

2.1. Volume

The easiest modification that can be done is volume control. This simply requires that we multiply the input by a certain number before sending it to the output. A button can be used to toggle between different values, all that needs to be done beforehand is configure the button as an input and enable its pull-up resistance in the software to define a base level for the button pin.

2.2. Filter

With the filter I am trying to achieve an echo effect, so we need to somehow delay the samples at the input. Since this must be done in real time, the output must only depend on previous samples, so we need a non-recursive filter such as a FIR filter (Finite Impulse Response).

The following equation can be used to express a simple non-recursive filter:

$$y(n) = x(n) + g*x(n-m),$$

where m is the delay (in number of samples), and $g \leq 1$ is the gain. These filters can be used to implement delay or echo sound effects.

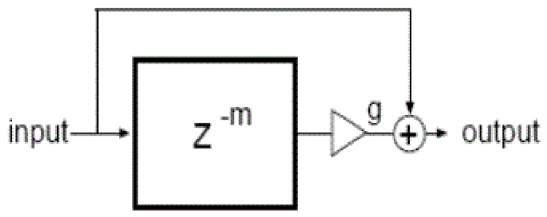

Figure 2: Non-recursive Filter Diagram

The transfer function can be written as follows:

$$H(z) = 1 + 0.25*z^{-delay}$$

2.2.1. Circular buffer

For the effect to be audible we would like the delay to be around 1 second, so we can set the number of samples at 45000. Since this would exceed the memory buffer, a common technique to overcome this problem is circular addressing. This allows the DSP to efficiently manage buffers of data in a manner that simplifies the implementation of various signal processing algorithms.

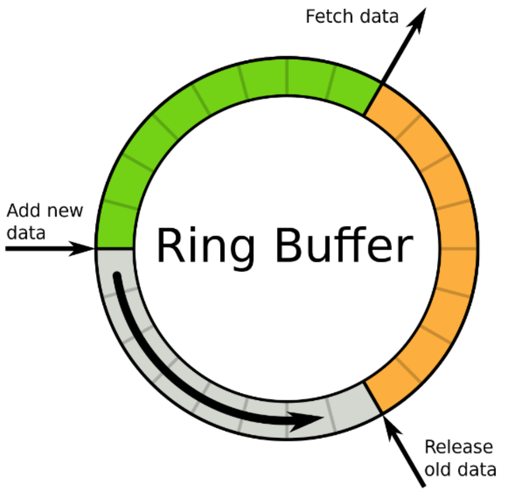

Figure 3: Circular buffer Schematic

DSPs have dedicated registers to hold the base address and length of the buffer, as well as the current address pointer. When an address pointer reaches the end of the buffer, it wraps around to the beginning.

3. Implementation

Since the starting point already provides the setup files, which cover most of the necessary initializations, we only have to configure the buttons: mark them as inputs and enable their internal pull-up resistors in “*Initialize.c*” and call the function in “*main.c*”.

```
void Init_Buttons(void) {
    // Configure PF10 and PF11 as input
    *pFIO_DIR &= ~(1 << 10);
    *pFIO_DIR &= ~(1 << 11);
    // Enable pull-up resistor
    *pFIO_INEN |= (1 << 10);
    *pFIO_INEN |= (1 << 11);
}
```

The rest of the program is all written in “*Process_data.c*” :

```
#include "Talkthrough.h"
#include <stdbool.h>
#include <cycle_count.h>
#include <stdio.h>

//--------------------------------------------------------------------------//
// Function:          Process_Data()

// Description: This function is called from inside the SPORT0 ISR every        //
                      time a complete audio frame has been received. The new
                      input samples can be found in the variables iChannel0LeftIn,//
                      iChannel0RightIn, iChannel1LeftIn and iChannel1RightIn
                      respectively. The processed    data should be stored in
                      iChannel0LeftOut, iChannel0RightOut, iChannel1LeftOut,
                      iChannel1RightOut, iChannel2LeftOut and   iChannel2RightOut
                      respectively.

//--------------------------------------------------------------------------//
```

```
#define ECHO_BUFFER_SIZE 50000
#define ECHO_DELAY 4500

int volumeFactor = 1;       // Initial volume factor
int filterState = 0;  // 0: no filter, 1: echo

int echoBuffer[ECHO_BUFFER_SIZE];
int echoIndex = 0;

// Function to check state of PF10 button
bool isVolumeButtonPressed(void) {
    return !(*pFIO_FLAG_D & (1 << 10)); // true when PF10 is pressed
}

// Function to check state of PF11 button
bool isEchoButtonPressed(void) {
    return !(*pFIO_FLAG_D & (1 << 11)); // true when PF11 is pressed
}

// Apply 1s echo filter to a signal using circular addressing
int echo(int input) {
    int output;
    echoBuffer[echoIndex] = input;
    int delayedIndex = (echoIndex - ECHO_DELAY + ECHO_BUFFER_SIZE) %
ECHO_BUFFER_SIZE;
    output = input + (echoBuffer[delayedIndex] >> 2); // 0.25 * delayed signal

    // Update echo index
    echoIndex = (echoIndex + 1) % ECHO_BUFFER_SIZE;
    return output;
}



// Volume control and optional echo effect
void Process_Data(void) {

      // Start counting cycles
      cycle_t start_count;
      cycle_t final_count;
      START_CYCLE_COUNT(start_count);

    // Check if volume button is pressed
    static bool lastVolumeButtonState = false;
    bool volumeButtonState = isVolumeButtonPressed();
    if (volumeButtonState && !lastVolumeButtonState) {
        volumeFactor = (volumeFactor + 1) % 5; // Increase volume factor up to 4 then reset
    }
    lastVolumeButtonState = volumeButtonState;

    // Check if filter button is pressed
    static bool lastEchoButtonState = false;
    bool echoButtonState = isEchoButtonPressed();
    if (echoButtonState && !lastEchoButtonState) {
        filterState = (filterState + 1) % 2; // Cycle through filter states: 0, 1
    }
    lastEchoButtonState = echoButtonState;
```

```
  // Apply selected filter and volume
 switch (filterState) {
   case 1: // Echo and volume
     iChannel0LeftOut = echo(iChannel0LeftIn) * volumeFactor;
     iChannel0RightOut = echo(iChannel0RightIn) * volumeFactor;
     break;
   default: // Just volume
        iChannel0LeftOut = iChannel0LeftIn * volumeFactor;
     iChannel0RightOut = iChannel0RightIn * volumeFactor;
     break;
 }

  // leave channel 1 as is for testing
  iChannel1LeftOut = iChannel0LeftIn;
  iChannel1RightOut = iChannel0RightIn;

  // Stop counting cycles and determine if real-time is achieved
  STOP_CYCLE_COUNT(final_count,start_count);
  PRINT_CYCLES("Number of cycles: ",final_count);
}
```

Bibliography

- Information Processing Technologies courses, Romulus Terebeş
- https://www.arrow.com/en/reference-designs/adzs-bf533-ezlite-adsp-bf533-ez-kit-lite-evaluation-system-based-on-blackfin-digital-signal-processors-dsps/f2a9466d89e91252ea05beddc4dc8fc0d410297c4dce
- https://circuitglobe.com/difference-between-fir-filter-and-iir-filter.html
- https://commons.wikimedia.org/wiki/File:Ring_buffer.svg