Ambisonic Surround Sound System

Ben Bloomberg · 6.111 Final Project Proposal · November 9, 2009

This document describes a system which is capable of taking several channels of real-time audio from a client computer and panning them across an arbitrary number of speakers arranged in any configuration.

Clock Cycles
The audio clock domain will be a 48Khz based system. All DSP will run at 65Mhz in order to provide 1300 effective DSP cycles for each clock cycle.

There will be several component systems that work to achieve the panning process in real time:

Computer Streaming System
This will be a python script on the computer which writes audio data to a virtual com port. The data will sit in a USB buffer on the host computer until it is requested by the labkit at a rate of 64 bits every audio clock cycle. Four incoming streams of audio will be sampled at 48khz and 16-bits per sample.

USB Receiver System
This module will read 1-byte at a time from the USB module. A sample will consist of 2 bytes, so 8 bytes must be read at the beginning of each audio clock cycle and passed to each of 4 encoders.

Coordinate Control System & UI
This system will provide an alternate to a UI with interactive mouse control. Each of the sources will be displayed on the screen and it will be possible to recall several programable preset locations for the sources. The coordinates for each source's location will be 7-bits of resolution.

Encoding Module
This component receives a coordinate from the UI system and a 2-byte sample. It uses a lookup table stored in FLASH to find two coefficients based on 6-bit coordinates. The seventh coordinate bit serves as an interpolation bit. The final coordinate is passed to the smoothing module. Each input has 16 encoding multipliers.

Smoothing Module
This module compares a smoothly varying, constantly changing multiplier to the coefficient sent from the encoder module. If the input coefficient is different, the interpolator increments or decrements the encoding multiplier by a constant amount that will not cause clicks in the audio stream. Both the multiplier and the coefficient are 16-bit resolution. The rate at which the multipliers change will be fixed and changeable based on the value of the labkit switches. This will allow preset-recall to happen at different rates.

Encoder Multiplier
This is a shared 16-bit multiplier that computes a new 32-bit product of each sample and encoder multiplier. This happens 16 times for each sample and the resulting sixteen 32-bit channels compose that inputs encoded ambisonic stream.

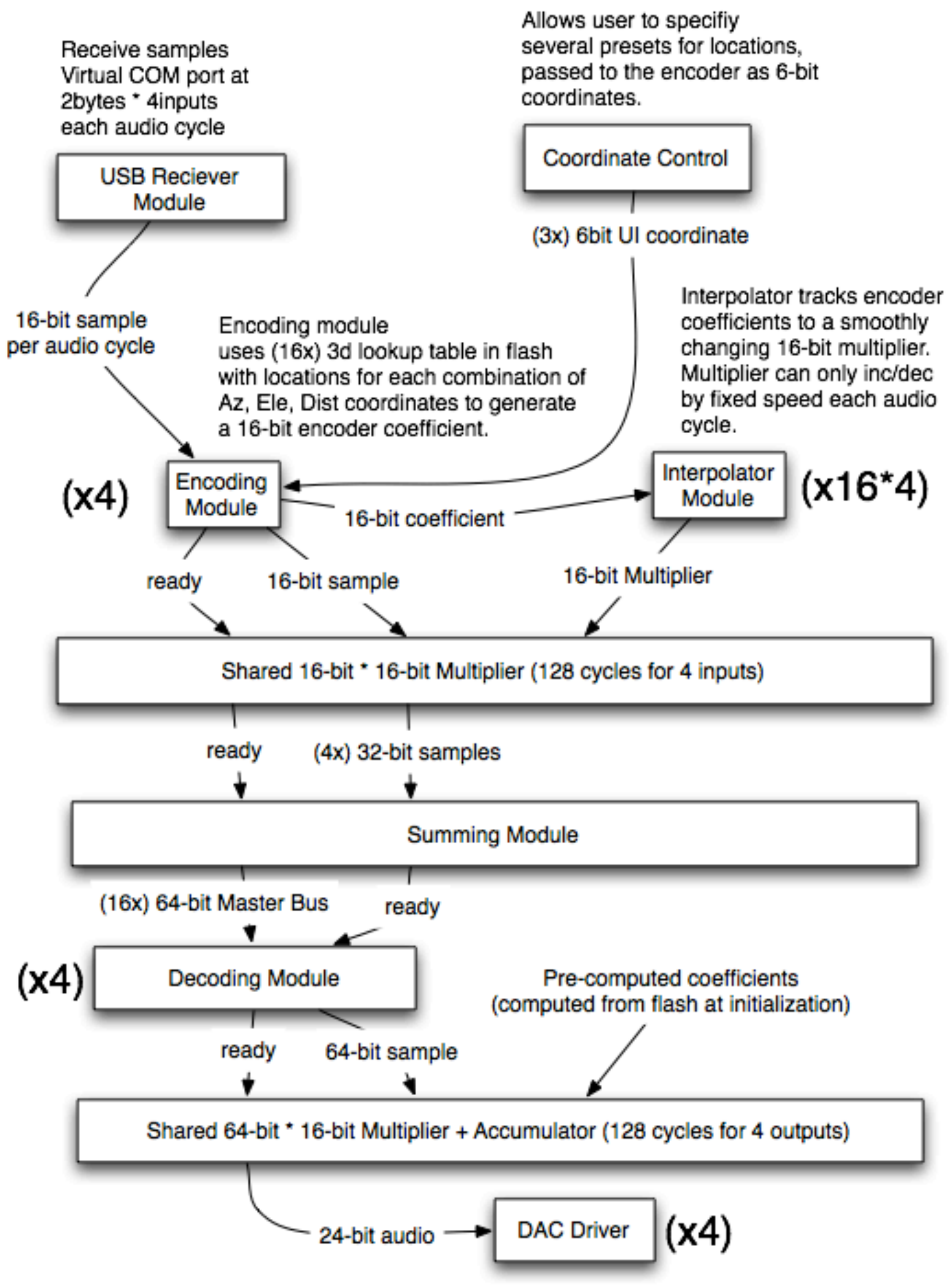
Summing Module
The summing system sums together 4 encoded streams and combines them into a single high resolution 16-channel stream representing the sum of all respective channels in each stream. The summing module provides a master 64-bit bus which all signals are summed into.

Decoding Module
A module that takes a high resolution 16-channel stream and performs appropriate decoding for one speaker. The coefficients for the decoder will be generated at initialization using the look-up table.

Decoder Multiplier
This module uses a single multiplier-accumulator to create the final 24-bit stream which can be sent to the output stage for each output. This will use a 64x16 bit multiplier.

Output Stage Module
This module drives an 8 channel DAC to provide high quality audio output. It will take the 24 most significant bits from each of the decoder outputs.

Block Diagram


Receive samples Virtual COM port at 2bytes * 4inputs each audio cycle
USB Reciever Module
16-bit sample per audio cycle
Allows user to specifiy several presets for locations, passed to the encoder as 6-bit coordinates.
Coordinate Control
(3x) 6bit UI coordinate
Encoding module uses (16x) 3d lookup table in flash with locations for each combination of Az, Ele, Dist coordinates to generate a 16-bit encoder coefficient.
Interpolator tracks encoder coefficients to a smoothly changing 16-bit multiplier. Multiplier can only inc/dec by fixed speed each audio cycle.
(x4)
Encoding Module
16-bit coefficient
Interpolator Module
(x16*4)
ready
16-bit sample
16-bit Multiplier
Shared 16-bit * 16-bit Multiplier (128 cycles for 4 inputs)
ready
(4x) 32-bit samples
Summing Module
(16x) 64-bit Master Bus
ready
(x4)
Decoding Module
Pre-computed coefficients (computed from flash at initialization)
ready
64-bit sample
Shared 64-bit * 16-bit Multiplier + Accumulator (128 cycles for 4 outputs)
24-bit audio
DAC Driver
(x4)

Optional Modules:

Dithering Module (Optional)
In order to convert a high bandwidth signal to a low bandwidth signal (i.e. 32-bit audio to 24-bit audio) it is possible to scale the original signal with noise shaping instead of just truncating the stream to the

correct width. This preserves audio resolution for quiet content. It makes sense to include this module as a stub which truncates the stream initially, to be filled in later.

Summing Serializer (Optional)
This module would allow the summing module output stream to be passed to another FPGA, effectively allowing multiple devices to be chained together for unlimited encoding and decoding. This could be accomplished using a high bandwidth interconnect, capable of transmitting 16-channels of 24-32 bit audio at higher than 48khz. For real-world applications, a serial interface would be needed, but this implementation could utilize the user IO pins in a parallel configuration.

RS-232 Automation (Optional)
This would publish all speaker location data to the client machine via RS-232 to allow automation via commercial show control software.

Doppler Simulation (Optional)
By running the encoding process at a speed faster than real time, it is possible to over-sample the incoming audio and change its speed by altering the sampling period for the encoding process. As long as it is possible to stream audio from the client computer at speeds above the output sampling rate, it should be possible to perform simple doppler correction for moving sources.

Room Simulation (Optional)
This would allow the specification of an ambisonic impulse response for each source, which could be used with a convolution algorithm to simulate 3 dimensional reverb. B-format impulse responses are available for many famous locations (York Minster, St. Andrews, etc..). These could be loaded into flash or even ZBT RAM because they are very short. Unfortunately, this is a very computationally intensive operation requiring 16*n^2 multiplications for n samples in the IR to convert the B-format to ambisonics. The algorithm would be based on GPL source code for the popular convolution engine JConv.