Audio Synthesis Basics

Analog Synthesis

Intro to Digital Oscillators

CSE466

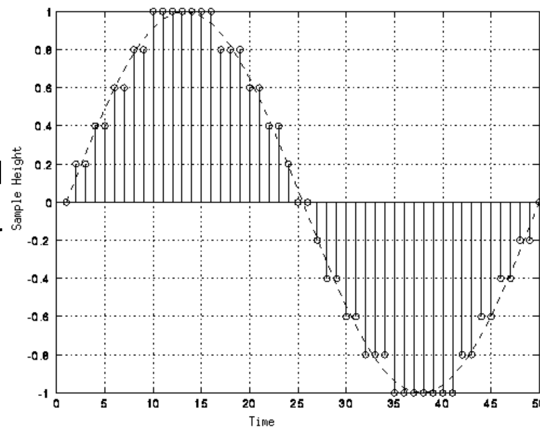
What is SAMPLING?

- Process by which an analog signal is measured or reconstructed, often millions of times per second for video, in order to convert the analog signal to digital.

Page 2

Quantization

- The height of each vertical bar can take on only certain values, shown by horizontal dashed lines, which are sometimes higher and sometimes lower than the original signal, indicated by the dashed curve.



Nyquist Theorem

- A theorem, developed by Harry Nyquist, which states that an analog signal waveform may be uniquely reconstructed, without error, from samples taken at equal time intervals.

Nyquist Theorem

- The sampling rate must be equal to, or greater than, twice the highest frequency component in the analog signal.

Page 5

Nyquist Theorem

- Stated differently:
- The highest frequency which can be accurately represented is one-half of the sampling rate.

Page 6

Analog Synthesis Overview

- Sound is created by controlling electrical current within synthesizer, and amplifying result.
- Basic components:
  - **Oscillators**
  - Filters
  - **Envelope generators**
  - Noise generators
- Voltage control

Page 7

Oscillators

- Creates periodic fluctuations in current, usually with selectable waveform.
- Different waveforms have different harmonic content, or *frequency spectra*.

Page 8

Filters

- Given an input signal, attenuate or boost a frequency range to produce an output signal
- Basic Types:
  - Low pass
  - High pass
  - Band pass
  - Band reject (notch)

Envelope Generators

- Generate a control function that can be applied to various synthesis parameters, including amplitude, pitch, and filter controls.

Noise Generators

- Generate a random, or semi-random fluctuation in current that produces a signal with all frequencies present.

Page 11

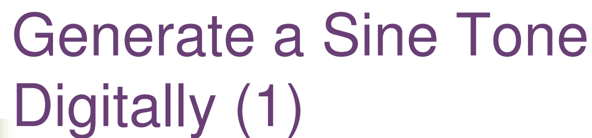
Digital Synthesis Overview

- Sound is created by manipulating numbers, converting those numbers to an electrical current, and amplifying result.
- Numerical manipulations are the same whether they are done with software or hardware.
- Same capabilities (components) as analog synthesis, plus significant new abilities

Page 12

Digital Oscillators

- Everything is a Table
  - A table is an indexed list of elements (or values)
  - The index is the address used to find a value

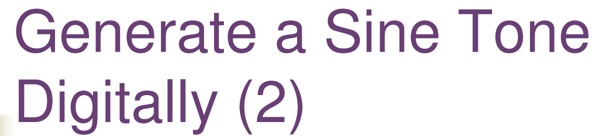


Generate a Sine Tone Digitally (1)

- Compute the sine in real time, every time it is needed.
  - equation:

$$signal(t) = r\sin(\omega t)$$

  - t = a point in time; r = the radius, or amplitude of the signal; w (omega) = 2pi*f the frequency
  - Advantages: It's the perfect sine tone. Every value that you need will be the exact value from the unit circle.
  - Disadvantages: must generate every sample of every oscillator present in a synthesis patch from an algorithm. This is very expensive computationally, and most of the calculation is redundant.



Generate a Sine Tone Digitally (2)

- Compute the sine tone once, store it in a table, and have all oscillators look in the table for needed values.
  - Advantages: Much more efficient, hence faster, for the computer. You are not, literally, re-inventing the wheel every time.
  - Disadvantages: Table values are discrete points in time. Most times you will need a value that falls somewhere in between two already computed values.

Table Lookup Synthesis

- Sound waves are very repetitive.
- For an oscillator, compute and store one cycle (period) of a waveform.
- Read through the wavetable repeatedly to generate a periodic sound.

- The Sample Rate doesn't change within a synthesis algorithm.
- You can change the speed that the table is scanned by skipping samples.
- skip size is the increment, better known as the phase increment.

  phase increment is a very important concept

Page 17

- Basic, two-step program:
  - $phase_index = \mathrm{mod}_L(previous_phase + increment)$
  - $output = amplitude \times wavetable[phase_index]$
- $increment = \dfrac{(TableLength \times DesiredFrequency)}{SampleRate}$

Page 18

If You’re Wrong, it’s Noise

- What happens when the phase increment doesn’t land exactly at an index location in the table?
  - It simply looks at the last index location passed for a value.
    In other words, the phase increment is truncated to the integer.
- Quantization
- Noise
- The greater the error, the more the noise.

Interpolation

- Rather than truncate the phase location…
  - look at the values stored before and after the calculated phase location
  - calculate what the value would have been at the calculated phase location if it had been generated and stored.

    Interpolate
- More calculations, but a much cleaner signal.

How to select the sample

- Important: Desired sample is between real samples!
- We can:
  - 1: Select the nearest sample
  - 2: Linearly interpolate between samples
  - 3: Resample

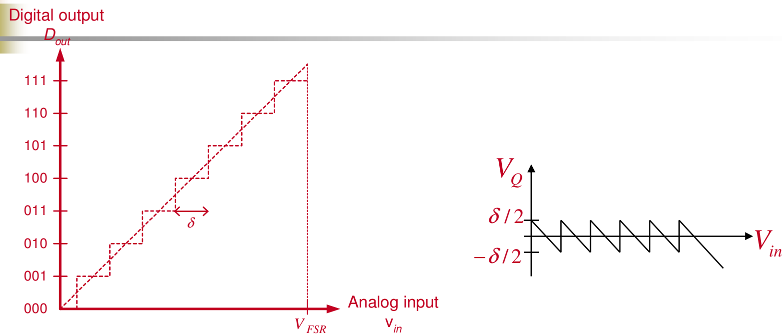
Page 21

Selecting the nearest sample

- Simply round and access your wave table
  - return wave(int(sample + 0.5));
- Works, but is somewhat noisy

Page 22

Quantization noise



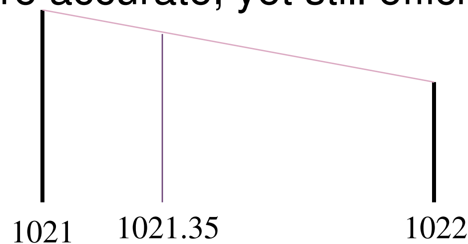
•N-bit converter: $\delta = \frac{V_{FSR}}{2^N}$

Linear interpolation

- Interpolate between two audio samples

```
double inbetween = fmod(sample, 1);
return (1. – inbetween) * wave[int(sample)] +
            inbetween * wave[int(sample) + 1];
```

- More accurate, yet still efficient



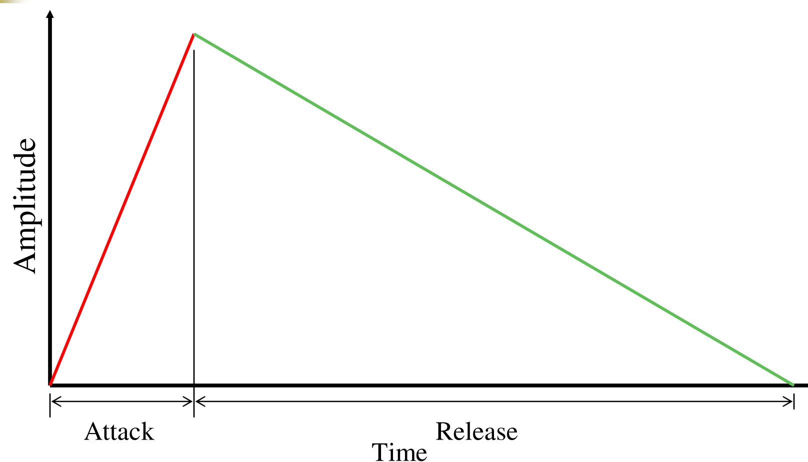
Envelopes

- What if we use looping to make an efficient piano sound?
  - Looping does not decay, but a piano sound does

- We commonly will make samples with fixed amplitudes, then make a synthetic *envelope* for the sound event.

Page 25

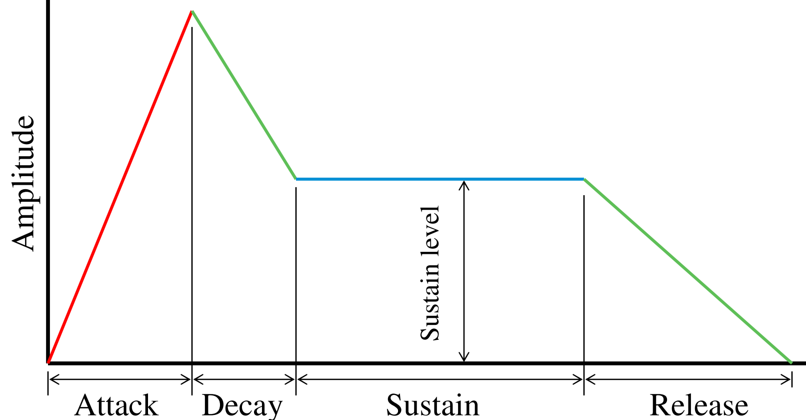
Attack and Release



Page 26

ADSR

- ADSR: Attack, decay, sustain, release

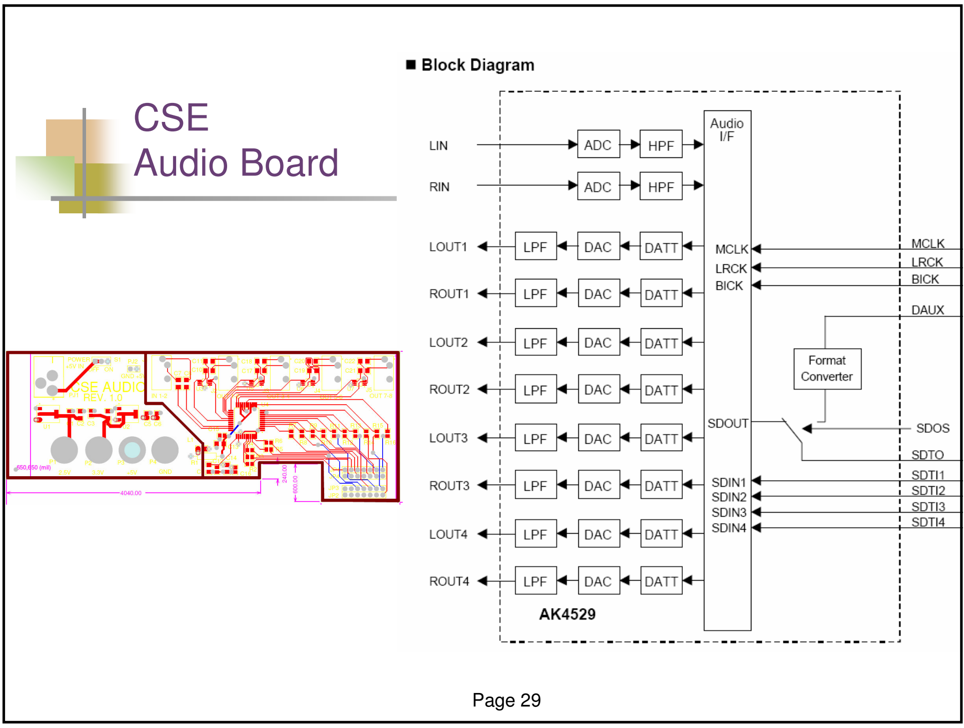


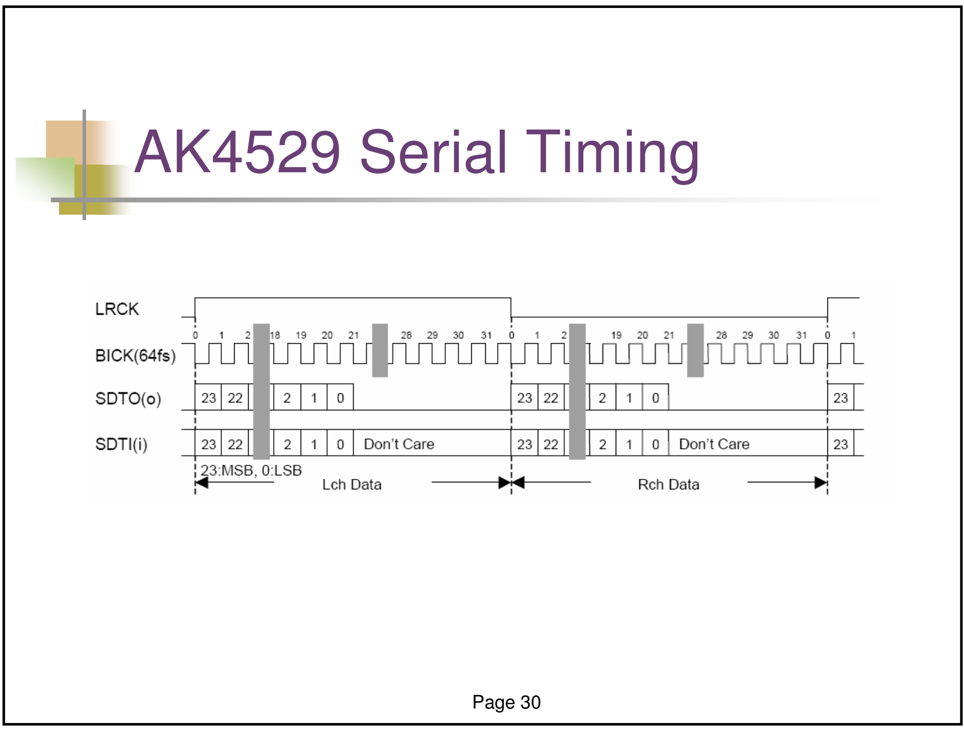
Page 27

OPB Audio Controller

- The opb_audio_controller unit is used to communicate with the audio boards mounted on the AFX BG560 boards. The chip requires that all communications with it be in a serial format.
- This controller allows you to have memory mapped I/O to the component so that you can build software to communicate with the audio chip.

Page 28


CSE
Audio Board
Block Diagram
LIN
RIN
ADC
HPF
Audio I/F
LOUT1
ROUT1
LOUT2
ROUT2
LOUT3
ROUT3
LOUT4
ROUT4
LPF
DAC
DATT
MCLK
LRCK
BICK
DAUX
Format Converter
SDOUT
SDOS
SDTO
SDIN1
SDIN2
SDIN3
SDIN4
SDTI1
SDTI2
SDTI3
SDTI4
AK4529
Page 29


AK4529 Serial Timing
LRCK
BICK(64fs)
SDTO(o)
SDTI(i)
Don't Care
23:MSB, 0:LSB
Lch Data
Rch Data
Page 30

Audio Controller Register Map

BASEADDR + 0x00	Control register	R/W
BASEADDR + 0x04	Input Left channel	R
BASEADDR + 0x08	Input Right channel	R
BASEADDR + 0x0C	Output1 Left channel	R/W
BASEADDR + 0x10	Output1 Right channel	R/W

- All registers are 32 bits wide, however, only the lower order 24 bits are used in the input and output registers. They are signed 24-bit values, however they are sign extended to 32-bits when you read them.

Audio Controller Register Map

BASEADDR + 0x00	Control register	R/W
BASEADDR + 0x04	Input Left channel	R
BASEADDR + 0x08	Input Right channel	R
BASEADDR + 0x0C	Output1 Left channel	R/W
BASEADDR + 0x10	Output1 Right channel	R/W

- There are only 2 flags in the control register. The lowest order bit (0x00000001) is the enable bit and enables the codec. It is tied to the PDN pin. The next bit (0x00000002) is the interrupt enable. When this is high, an interrupt is generated every time the codec is ready for a new sample. The interrupt is cleared by writing or reading from any register in the codec.

OPB Audio Controller

- The codec requires several different clocks so this component contains a DLL to generate these clock signals.
- All other component's clock inputs to use an internal net that is connected to the SYS_CLK output of the opb_audio_controller.
- Connect the XTAL_CLK input of the opb_audio_controller to the off-chip crystal clock source (probably an external net that gets connected to AL17).

Lab- Interrupt routine

```
void audio_interrupt_handler(void *InstancePtr)
{
	/*
	 * TODO: calculate the next sample and give it to the audio controller.
	 *
	 * Note that this interrupt will happen every 23 microseconds, or
	 * at 43.4kHz (the sample rate of the codec)
	 */

	/* this currently makes a triangle wave */
	if(curr_value == HIGH_VALUE) climbing = 0;
	if(curr_value == LOW_VALUE) climbing = 1;
	if(climbing)
	{
		curr_value++;
	}
	else
	{
		curr_value--;
	}

	XAudio_mWriteOutput(XPAR_AUDIO_BASEADDR, LEFT, 0, curr_value << SHIFT_AMOUNT);
}
```

Lab- Main

```
int main()
{
      /* TODO: initialization code should go here */

      curr_value = 0;
      climbing = 1;

   /* register for the interrupts */
   XIntc_InterruptVectorTable[0].Handler = audio_interrupt_handler;
   XIntc_InterruptVectorTable[0].CallBackRef = NULL;
   XIntc_mEnableIntr(XPAR_INTC_SINGLE_BASEADDR, XPAR_AUDIO_INTERRUPT_MASK);
   XIntc_mMasterEnable(XPAR_INTC_SINGLE_BASEADDR);

   /* globally enable the interrupts on the microblaze */
   microblaze_enable_interrupts();

      /* enable the audio codec and make it interrupt me every time it wants new data */
   XAudio_mSetControlReg(XPAR_AUDIO_BASEADDR, AUDIO_CR_INT_ENABLE_MASK
      AUDIO_CR_ENABLE_MASK);

   for(;;)
   {
     /* do nothing, just let the interrupts handle the rest of the work */
   }
   return 0;  /* never reached */
}
```

Tidbits

- The sample rate of the codec is 43.4kHz. Use a table length of 256 entries.
- The Microblaze has no floating point operations. If you use floating point, the compiler will emulate it, however, it will be extremely slow, so you should not use floating point.
- The Microblaze does not have a hardware multiplier, so multiplies are done in software. As a result, they are very slow. You probably have about enough time between samples to do about 2 multiplies, maybe 3.
- Don't even think about doing a divide by a number other than a power of 2 (bit shift).
- The audio codec takes 24-bit signed numbers, centered at 0. This means that:
  - The highest value it can receive is 2^23 - 1, or 8388607, or 0x007FFFFF.
  - The lowest value it can receive is -(2^23) or -8388608, or 0xFF800000.