Computer Arithmetic on the 8bit Floating-Point Formats G.711 alaw and ulaw

Christoph Lauter

College of Engineering, Computer Science Department, University of Texas at El Paso

RAIM JMM 2025, Lyon, France

IEEE 754 or *L'évangile selon Jean-Michel*

- FP numbers are $(-1)^s \, 2^{E-b} \, h.m_1 m_2 \cdots_{k-1}$
- E is a biased exponent on w bits, $b = 2^{w-1} - 1$ is the bias
- Precision is k, $k-1$ bits are explicitly stored, h is hidden
- Common formats

Name	w	k	Storage space	Decimals	Abbreviation
double	11	53	8 bytes	≈ 16	E11M52
single	8	24	4 bytes	≈ 7	E8M23
half	5	11	2 bytes	≈ 3	E5M10

- Operations $+, -, \times, /, \sqrt{}$, FMA are correctly rounded

$$a \oplus b = \circ\,(a+b)$$

- We have a signed ± 0.0
- There are special encodings of $\pm\infty$ and tons of NaNs

The AI Race for Small Formats

Literature Survey on DNNs and Precision

- Micikevicius, Narang et al. “Mixed precision training”:
  $k = 16$ bits is enough
- Microsoft Brainwave Project:
  $k = 11$ bits is enough
- Agrawal, Mueller et al. “Dlfloat: a 16-b floating-point...”:
  $k = 9$ bits is enough
- Henry, Tang, Heinecke. “Leveraging the bfloat16 artificial...”:
  $k = 8$ bits is enough
- Banner, Nahshan et al.. “Post training 4-bit quantization...”:
  $k = 4$ bits is enough
- Hubara, Courbiaux et al. “Binarized neural networks”:
  $k = 1$ bit is enough

P3109 8bit FP Formats

- The IEEE P3109 Working Group is standardizing AI formats
- The P3109 formats are 1 byte formats
- Still $(-1)^s \, 2^{E-b} \, h.m_1 m_2 ... m_{k-1}$
- Most Common Proposed Formats

Name	w	k	Storage space	Decimals	Abbreviation
–	3	4	1 byte	≈ 1	**E3M4**
–	4	3	1 byte	< 1	E4M3

- Operations are still correctly rounded, but nobody cares
- We have a single 0 with **no** sign
- There are special encodings of $\pm\infty$ and a single NaN

Reinventing the Wheel?

Claim 1 of this talk

- P3109 is reinventing the wheel with E3M4.
- CCITT/ITU G.711 alaw essentially **is** E3M4
- G.711 alaw was standardized in $\leq$1980
- G.711 alaw is used in billions of phone calls daily
- G.711 ulaw, a variant, dates back to the 1950ies

Claim 2 of this talk

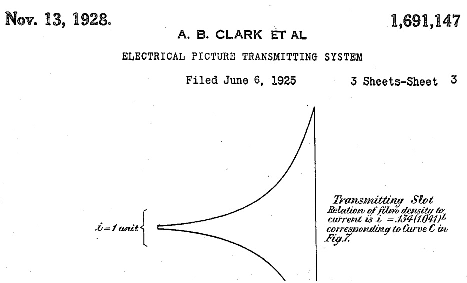
- Even though G.711 alaw meant to be a storage format, we **can** do FP arithmetic in G.711 alaw
- We present correctly rounded G.711 alaw arithmetic
- We implement FFT in G.711 alaw
- We can learn from this experience for AI applications

Signal Transmission and Signal Processing

- Signals are sampled at a certain sampling rate
    - G.711 subtly implies a 8kHz sampling rate
    - That sampling rate is confirmed by tons of other standards
- Signals are then quantized
    - The quantization is a type of rounding
    - In signal processing, a single rounding does not matter much
    - Rounding is modeled as noise
    - Signal-Noise-Ratio (SNR) is the important measure
- Signals can be processed with Digital Signal Processing
    - Mixing
    - FFTs
    - Filtering
    - . . .

Analog Logarithmic Signal Representation

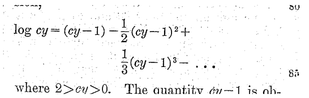
- Analog companders represent signals by their logarithm
- The companding improves the Signal-Noise-Ratio (SNR)
- The idea leverages the Weber-Fechner law of human perception
- US Patent 1691147 filed in 1925


Nov. 13, 1928. A. B. CLARK ET AL 1,691,147

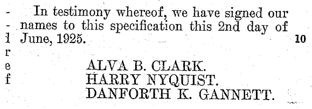
ELECTRICAL PICTURE TRANSMITTING SYSTEM

Filed June 6, 1925 3 Sheets-Sheet 3

$i = 1$ unit
Transmitting Slot
Relation of film density to current is $i = .134(1.041)^L$ corresponding to Curve C in Fig. 7.


$$\log cy = (cy-1) - \frac{1}{2}(cy-1)^2 + \frac{1}{3}(cy-1)^3 - \ldots$$

where $2 > cy > 0$. The quantity $cy-1$ is ob-

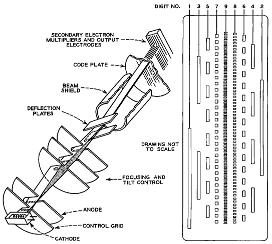

In testimony whereof, we have signed our names to this specification this 2nd day of June, 1925.

ALVA B. CLARK.
HARRY NYQUIST.
DANFORTH K. GANNETT.

Digital Signal Representation - ADCs

- End of 1940ies: the Bell Labs work on digital transmission
- Three ideas considered
  - First analog companding, then linear analog-digital conversion
  - Immediately do logarithmic analog-digital conversion
  - Fine-grained linear analog-digital conversion, compand to FP

☞ G.711 ulaw and alaw



Decoding G.711 alaw: *The Sound of Silence*

- Let's look at some G.711 alaw recordings of silence
- Silence should be all zeros, this will explain how 0 is encoded
- G.711 alaw is a 1 byte format, so we can look at single bytes

Decoding G.711 alaw: *The Sound of Silence*

- Let's look at some G.711 alaw recordings of silence
- Silence should be all zeros, this will explain how 0 is encoded
- G.711 alaw is a 1 byte format, so we can look at single bytes
- We see

```
0xd5 0xd5 0xd5 0xd5 0xd5 0xd5 0xd5 0xd5 0xd5 0xd5
0xd5 0xd5 0xd5 0xd5 0xd5 0xd5 0xd5 0xd5 0xd5 0xd5
0xd5 0xd5 0xd5 0xd5 0xd5 0xd5 0xd5 0xd5 0xd5 0xd5
```

Decoding G.711 alaw: *The Sound of Silence*

- Let's look at some G.711 alaw recordings of silence
- Silence should be all zeros, this will explain how 0 is encoded
- G.711 alaw is a 1 byte format, so we can look at single bytes
- We see

```
0xd5  0xd5  0xd5  0xd5  0xd5  0xd5  0xd5  0xd5  0xd5  0xd5
0xd5  0xd5  0xd5  0xd5  0xd5  0xd5  0xd5  0xd5  0xd5  0xd5
0xd5  0xd5  0xd5  0xd5  0xd5  0xd5  0xd5  0xd5  0xd5  0xd5
```

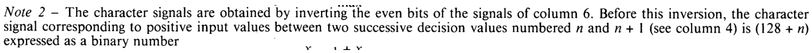
  or

```
0x55  0x55  0x55  0x55  0x55  0x55  0x55  0x55  0x55  0x55
0x55  0x55  0x55  0x55  0x55  0x55  0x55  0x55  0x55  0x55
0x55  0x55  0x55  0x55  0x55  0x55  0x55  0x55  0x55  0x55
```

- This does not look at all like zero. Or does it?

Decoding G.711 alaw: *Bit-Inversions*

- `0x55` = 01010101
- Transmission line engineers love zero-one-zero edges
  ☞ An edge provides for clock resynchronization
- A first look into the G.711 Standard. There's a footnote:

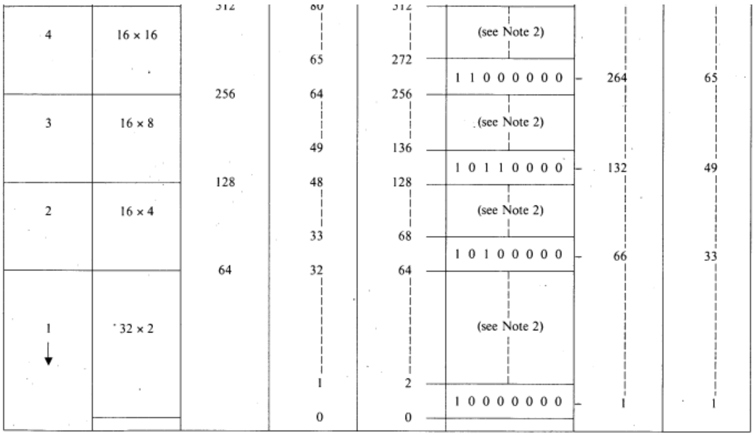


Note 2 – The character signals are obtained by inverting the even bits of the signals of column 6. Before this inversion, the character signal corresponding to positive input values between two successive decision values numbered n and $n + 1$ (see column 4) is $(128 + n)$ expressed as a binary number

- So `0x55` is actually 00000000 with even bits flipped
  and `0xd5` is actually 10000000 with even bits flipped
- That could be -0.0 and $+0.0$. (And it is.)

A look into the G.711 Standard

- A second look into the G.711 Standard. There's a table:



		512	80	512			
4	16 × 16				(see Note 2)		
			65	272			
					1 1 0 0 0 0 0 0	264	65
		256	64	256			
3	16 × 8				(see Note 2)		
			49	136			
					1 0 1 1 0 0 0 0	132	49
		128	48	128			
2	16 × 4				(see Note 2)		
			33	68			
					1 0 1 0 0 0 0 0	66	33
		64	32	64			
1 ↓	32 × 2				(see Note 2)		
			1	2			
					1 0 0 0 0 0 0 0	1	1
			0	0			

- It's beginning to look a lot like Floating-Point…

Decoding G.711 alaw: *The Revelation*

- G.711 alaw: we can get rid of the bit-inversions first
- The MSB is the sign bit, 1 is $+$, 0 is $-$, another inversion
- There are 3 bits for an exponent after the MSB, bias $b = 3$
- There are 4 bits for a mantissa ☞ **E3M4**
- There is a hidden bit, just as in IEEE754 or P3109
- There are subnormals, just as in IEEE754 or P3109
- There are no infinities, quantization uses round-toward-zero
- There are no NaNs, which mean nothing for signals
- Overall

$$(-1)^{s+1}\, 2^{E-b}\, h.m_1 m_2 m_3 m_4$$

No Absolute Reference

- IEEE754 and P3109 explain how decoding of FP to $\mathbb{R}$ works
- G.711 alaw explains how decoding to an integer *"interval"* $v \in \mathbb{Z}$ works
- The scale σ of one fixed-point value v is given nowhere
- G.711 gives a sequence of 8 alaw bytes s_i which
  - can be decoded to 8 values v_i
  - and scaled with a scale σ s.t.
  - when they are played as $\sigma\, v_i$ repeatedly,
  - the corresponding signal has a mean **power** of 1mW

$$\frac{8}{8000\text{Hz}} \sum_{i=0}^{8-1} \sigma^2\, v_i^2 = 1\text{mW}$$

- However, G.711 alaw is so close to IEEE754 that we can also chose σ s.t. the bias becomes $b = 2^{w-1} - 1$ for $w = 3$.

G.711 ITU Reference Code vs. IEEE754 Habits

- The ITU website gives reference code for G.711 alaw
- The reference code is in pre-ANSI C (and is unreadable)
- The code provides two functions
  - to decode an `unsigned char` to `signed short` v and
  - to encode an `signed short` v to `unsigned char`.
- Figuring out the scale σ' to apply on v is…

G.711 ITU Reference Code vs. IEEE754 Habits

- The ITU website gives reference code for G.711 alaw
- The reference code is in pre-ANSI C (and is unreadable)
- The code provides two functions
  - to decode an `unsigned char` to `signed short` v and
  - to encode an `signed short` v to `unsigned char`.
- Figuring out the scale σ' to apply on v is...
- ...(again) left as an exercise to the reader: $\sigma' = 2^{-10}$
- The functions behave differently than classic IEEE754
  - the encode (rounding) function performs round-toward-zero
  - the **decode** function systematically adds $\pm 1/2$ ulp()
  - in an attempt to simulate round-to-nearest,
  - loosing the ability to produce ± 0.0

G.711 alaw FP Numbers at a Last Glance

- $1.0 =$`0xe5`
- $-1.0 =$`0x65`
- $+0.0 =$`0xd5`
- $-0.0 =$`0x55`
- Largest Normal $2^5 - \mathrm{ulp}(2^5) = 31 =$`0xaa`
  ☞Everything greater than 31 rounds to 31
- Smallest Normal $2^{-2} =$`0xc5`
- Smallest Subnormal $2^{-6} =$`0xd4`
- $\circ\left(\sqrt{2}/2\right) = 0.71875 =$`0xf2`
- $\circ\left(\pi\right) = 3.125 =$`0x9c`

Computer Arithmetic on G.711 alaw

Contribution:

- Complete reimplementation of G.711 alaw from scratch in C
- Implementation of correctly rounded G.711 alaw arithmetic
  - $+, -, \times, /, \text{FMA}, \sqrt{\ }$
  - $\text{FMMA}(a, b, c, d) = \circ\,(ab + cd)$ for complex arithmetic
  - Conversions from and to double precision
  - Conversions from and to integers
  - Comparisons
  - No elementary functions, yet (anyway just 256byte tables)
- Binding in C++ with all C++ bells and whistles
  - Overloaded operators
  - Casts
  - Compound operators
  - Move-assignments

☞ Code with G.711 alaw like with `double`

A Signal Processing Example: FFT

- FFT is one of the most basic Signal Processing Algorithms

$$\begin{aligned} X_k &= \sum_{i=0}^{N-1} x_i \, e^{-j\,2\pi\,\frac{ki}{N}}, \quad j^2 = -1 \\ x_i &= \frac{1}{N} \sum_{k=0}^{N-1} X_k \, e^{j\,2\pi\,\frac{ik}{N}} \end{aligned}$$

- Can we implement FFT on G.711 alaw FP arithmetic
- and get reasonable results in terms of FP and round-off noise?

A Signal Processing Example: FFT

- FFT is one of the most basic Signal Processing Algorithms

$$\begin{aligned} X_k &= \sum_{i=0}^{N-1} x_i \, e^{-j\,2\pi\,\frac{ki}{N}}, \quad j^2 = -1 \\ x_i &= \frac{1}{N} \sum_{k=0}^{N-1} X_k \, e^{j\,2\pi\,\frac{ik}{N}} \end{aligned}$$

- Can we implement FFT on G.711 alaw FP arithmetic
- and get reasonable results in terms of FP and round-off noise?

- Blunt answer: Нет!

FFT: Unitary It Shall Be

- e.g. X_0 is the DC component, for $N > 31$, X_0 overflows
- We can implement the unusual *unitary* FFT

$$X_k = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} x_i \, e^{-j\,2\pi\,\frac{ki}{N}}, \quad j^2 = -1$$

$$x_i = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} X_k \, e^{j\,2\pi\,\frac{ik}{N}}$$

- For unitary FFTs, Parseval's theorem is satisfied
- The usual FFT split into 2 recursions does not work anymore
  - $\frac{1}{\sqrt{N}} = \frac{1}{\sqrt{2\,N'}} = \frac{1}{\sqrt{2}} \cdot \frac{1}{\sqrt{N}}$
  - But the precision of G.711 alaw $\circ\left(\frac{\sqrt{2}}{2}\right)$ is too bad

☞ Let's implement an FFT split with 4 recursions
  - $\frac{1}{\sqrt{4}} = 1/2$, which is exact

G.711 alaw FFT - Rounding Errors

- Implement $N = 256$ point unitary FFT and IFFT in alaw
- Generate vectors x with random alaw-encoded x_i
- Compute $\tilde{x} = \text{IFFT}\left(\text{FFT}\left(x\right)\right)$
- Compute energy in x and in noise $\tilde{x} - x$

$$E = \sum_{i=0}^{N-1} x_i^2 \text{ vs. } \Delta = \sum_{i=0}^{N-1} \left(\tilde{x}_i - x_i\right)^2$$

- Deduce Signal-to-Noise Ratio SNR$=10 \log_{10} \frac{E}{\Delta}$dB
- Experimental result: G.711 alaw FP FFT gives

$$\text{SNR} = 6.2\text{dB}$$

The case of G.711 ulaw

- The G.711 actually defines two formats
  - alaw, as discussed
  - ulaw, which is even older
- G.711 alaw is used everywhere but…
- …in the US, Canada and Japan which use G.711 ulaw
- Key differences:
  - ulaw has a fancy representation:

$$(-1)^s \left(2^{E-b} \cdot (33 + 2 \cdot m) - 33\right)$$

  - ulaw has no classical subnormals
  - ulaw does not use the even-bit inversion of alaw
  - ulaw extends the bit inversion of the sign on the rest of the bits
  - ulaw guarantees that the LSB can be dropped

G.711 FP - Lessons Learned

- G.711 alaw is just E3M4 Floating-Point with bit inversions
  ☞ When you call your spouse on the phone, you talk on FP!
- G.711 ulaw is *baroque*
- G.711 alaw can be extended to full FP Arithmetic
  - Implemented $+, -, \times, /, \sqrt{\ }$, FMA, FMMA, conversions
  - With correct rounding
  - In C++, so you can code as easily as with `double`
  - Sufficiently useful to do tasks like FFTs
- G.711 alaw **constantly** overflows and underflows
  ☞ as does P3109 E3M4
- G.711 alaw makes us think in *transmission levels* in dB
  ☞ This made us make the FFT unitary
  ☞ **We should adopt the same point of view for AI**

Questions & Answers

Слава Україні!

Thank you & Merci!

Questions?