Adventures in Ambient Audio

Kuy Mainwaring
@kuymainwaring

Rewind to 2014

2014.

Goal: Clean, responsible solar audio.

Objectives

- Low-power (<5W per unit)
- Run from bank of 12V batteries
- Build 16 to 20 units
- Under $250 budget!

Wiring

How do you do it?

Idea: Each unit is just a speaker

- Requires one central amplifier
- The amplifier must have many outputs
- Lots of wiring
- Power loss over long wire runs
- Needs thicker wires, which increases cost
- Where do you get a 20-channel amp?
- Does that source run efficiently on 12V?

Wiring

How do you do it?

Idea: Each unit contains its own amplifier

\+ Each unit takes power and audio inputs

\+ Power, audio wires can be in parallel

\+ More distributed and failure-tolerant

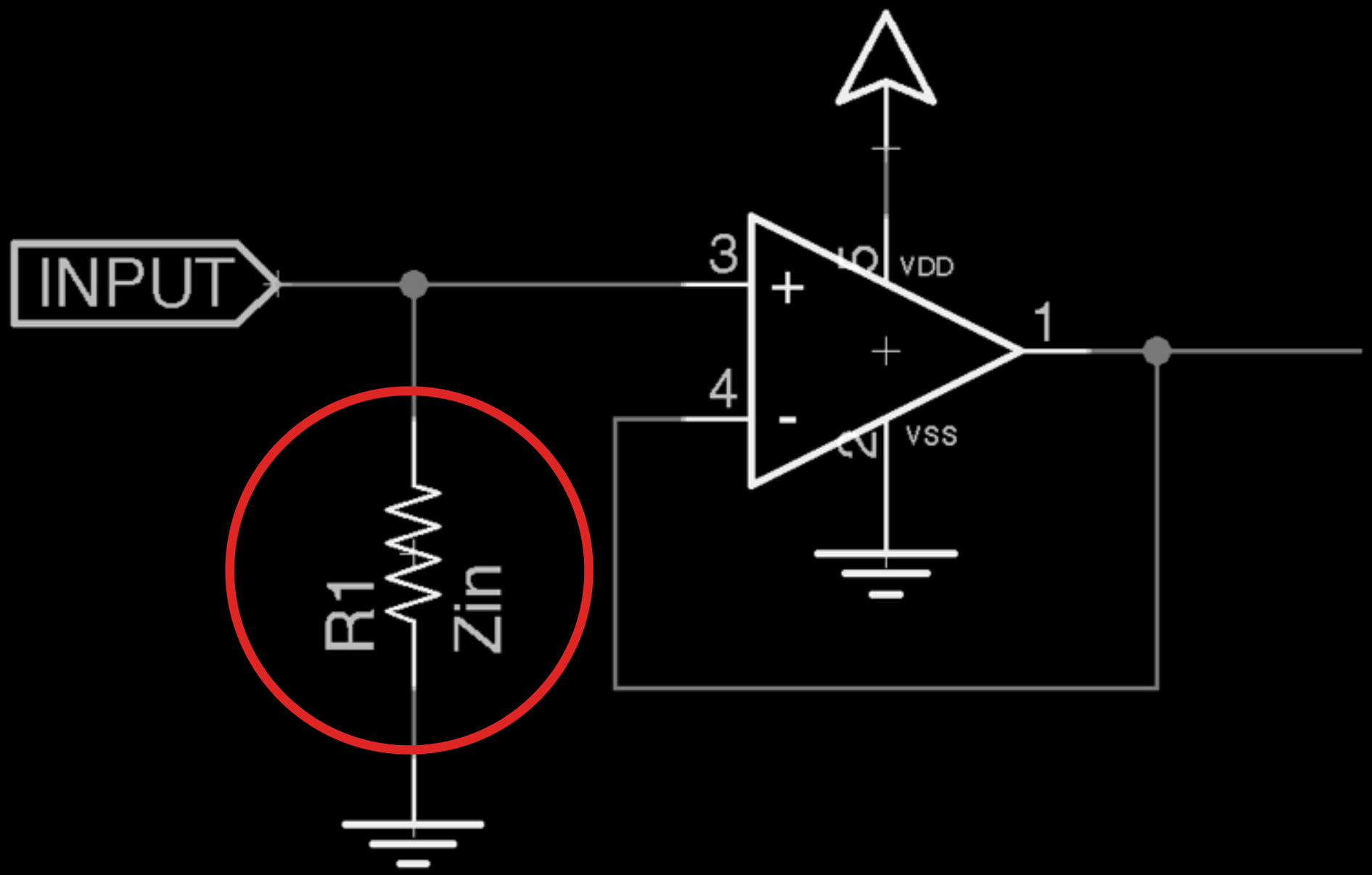
\+ Can make my own small amplifiers

\+ Can split into two (or more) systems

\- Analog input parallel impedance

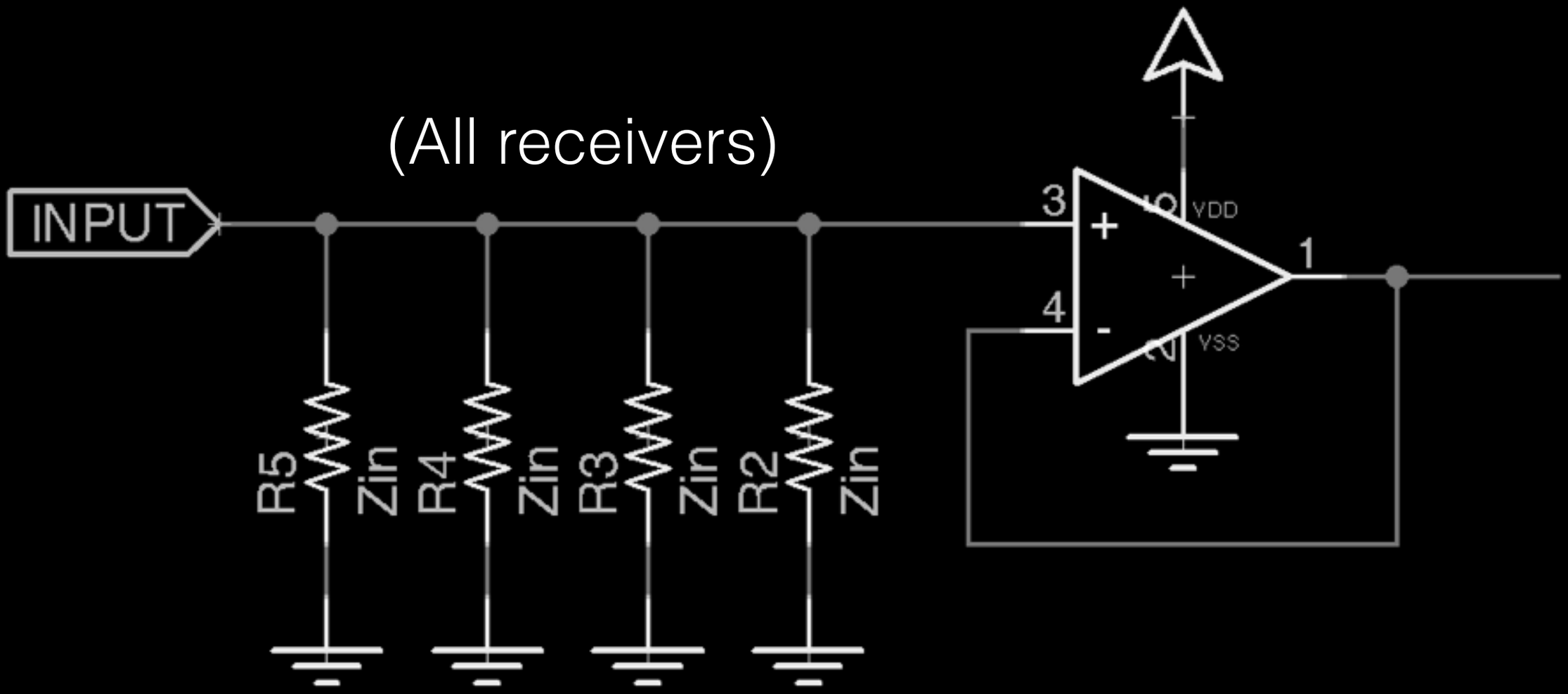
\- Noise

Analog input impedance



Analog input impedance

(All receivers)



(Source sees Z1 || Z2 || Z3 || … impedance)

Wiring

How do you do it?

*Idea: Each unit contains its own **digital** amplifier*

\+ All the benefits of an integrated amplifier
\+ Digital inputs can be very noise-tolerant
\+ Not affected by amplifier input impedance
\+ Bi-directional communication possible
\+ Can send other data than just audio
\- Transmitter is non-trivial
\- Network may need repeaters, termination

But digital could be more complex!

Behold, the humble serial port.

Asynchronous serial
is very flexible and durable.

We could use a single data line (plus ground.)

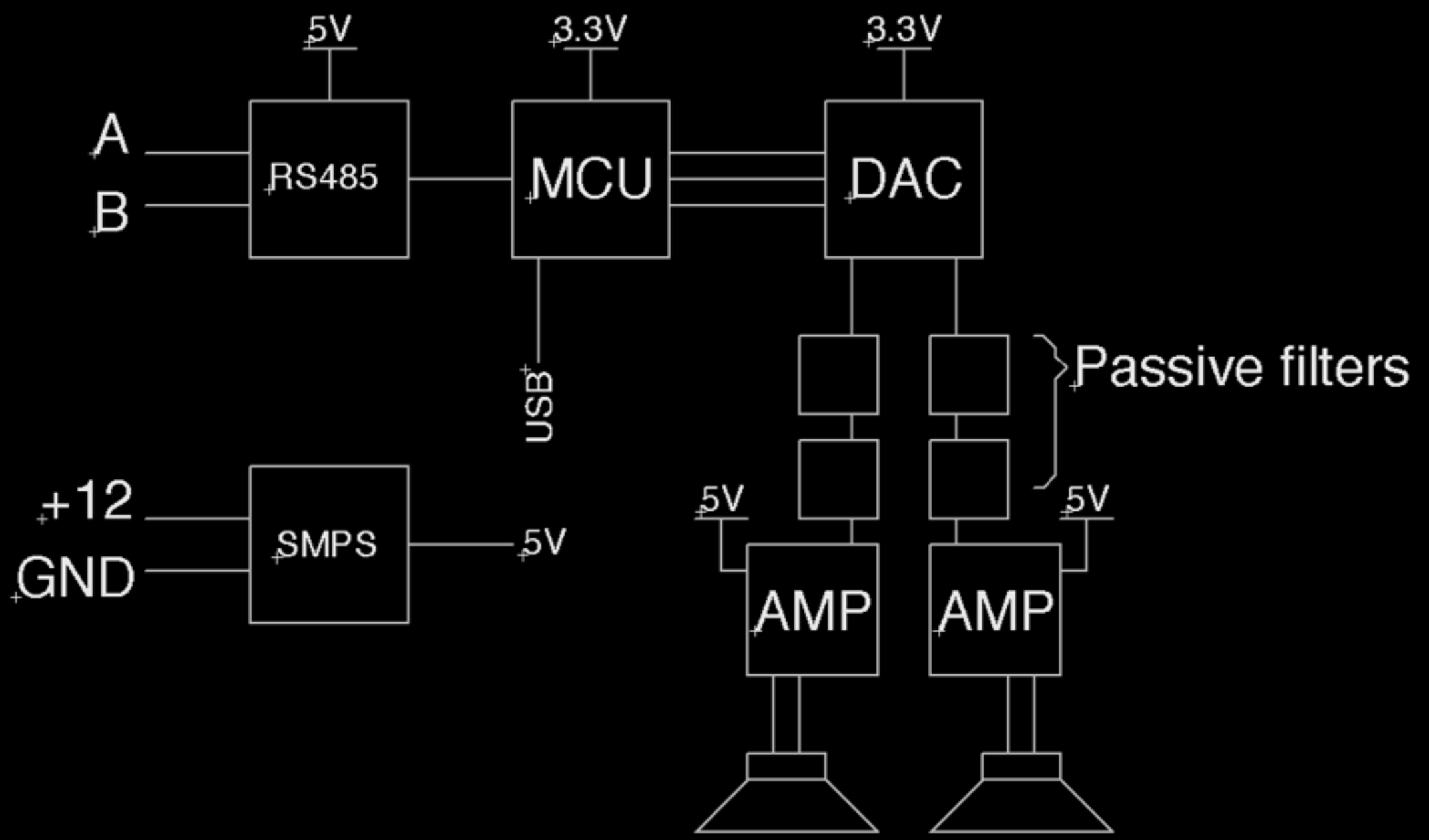
But that’s not reliable over long wires.
We can do better than that!

RS-485 is a way to send a stream of bits
over differential wiring to other devices.

It has no dedicated clock, just an
“A” line and a “B” line. Electrically,
“A” and “B” are opposites.

A \ B	L	H
L	?	0
H	1	?

Block Diagram (Receiver)



That's all great, but
how do I get started?

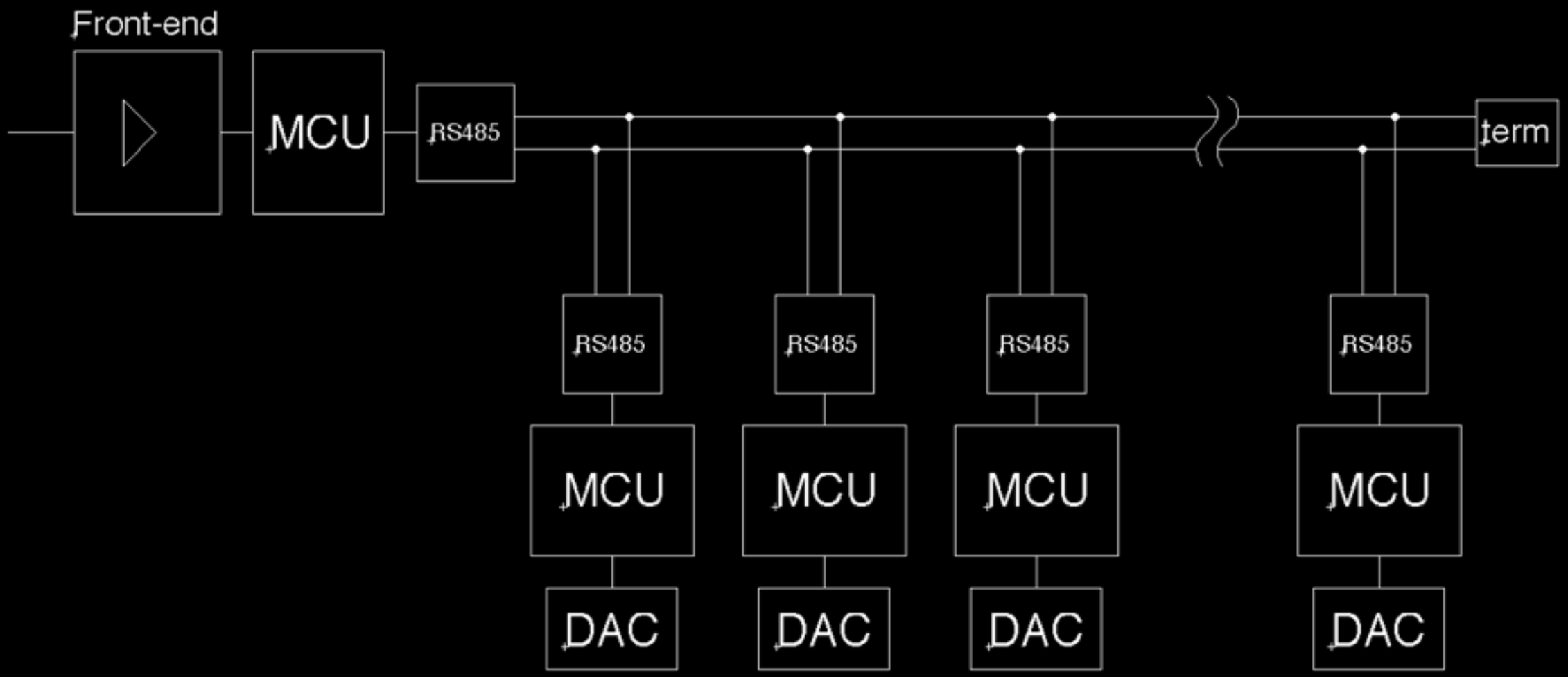
Start with what you know
and what you have on-hand

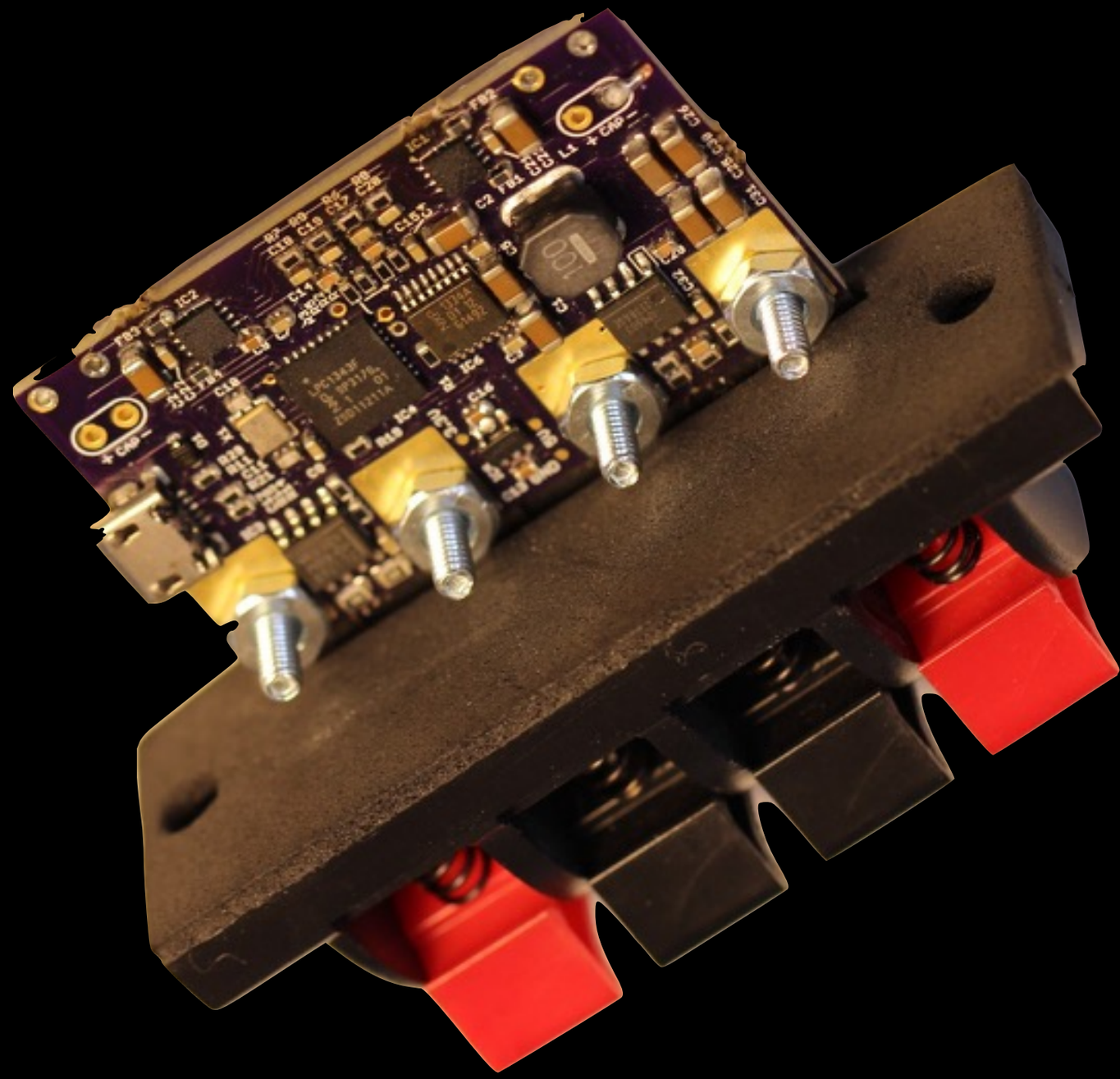
This was the key to
staying under budget.

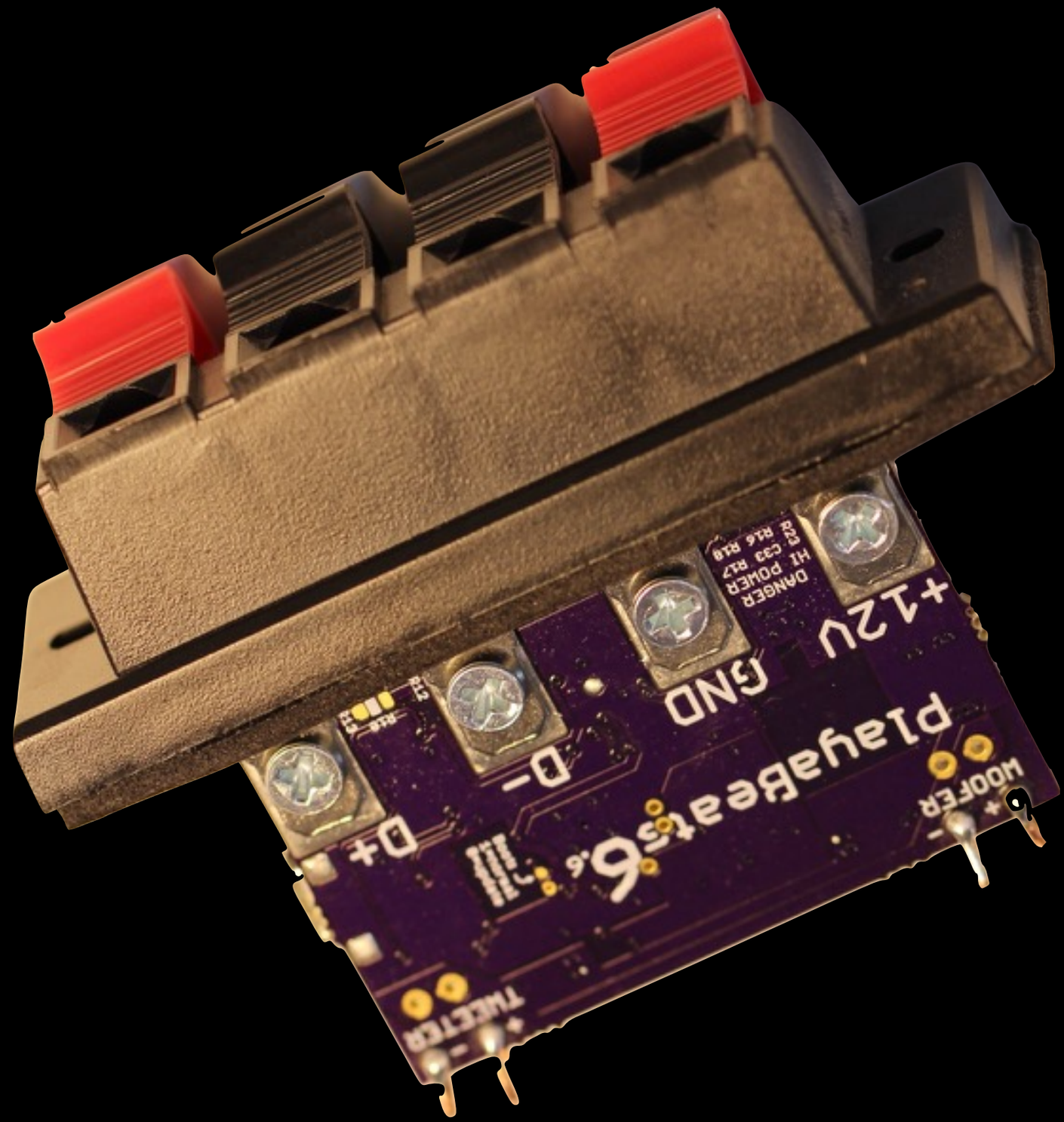
A bunch of leftover parts

- NXP LPC1343 Microcontroller
  ARM Cortex-M3, USB Bootloader, 72MHz
- NXP UDA1334 Stereo DAC
  Accepts I2S and similar
- TI TPA2006A Audio Amplifier
  Small, simple, efficient and effective
- Diodes AP6503 SMPS
  *Not a fan but had a **lot** of them lying around*

System block diagram




+CAN−
+CAN−

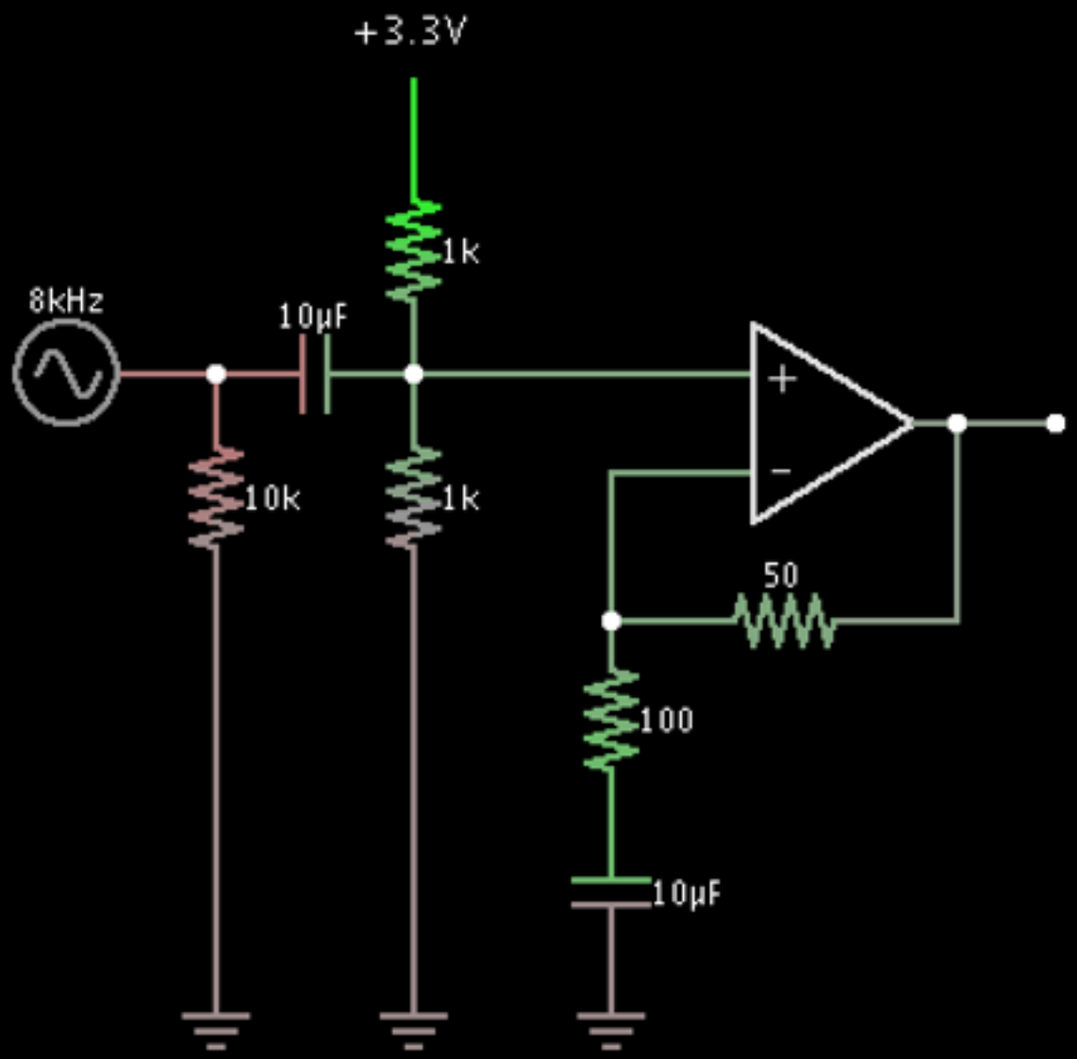

DANGER
HI POWER
+12V
GND
D-
D+
playBeats6
WOOFER
TWEETER

Let’s talk data

Audio: a continuous function consisting of a sum of oscillations ranging from 20Hz to 20kHz.

We can sample audio with an ADC.

Front-end



Line-in audio is 2Vpp centered at 0V and expects Z=10kΩ

The front-end scales and offsets the input to Vdd/2 to match the ADC

The audio source is endless
so we break it into *packets*

The transmitter samples data,
compresses it, sends packets.

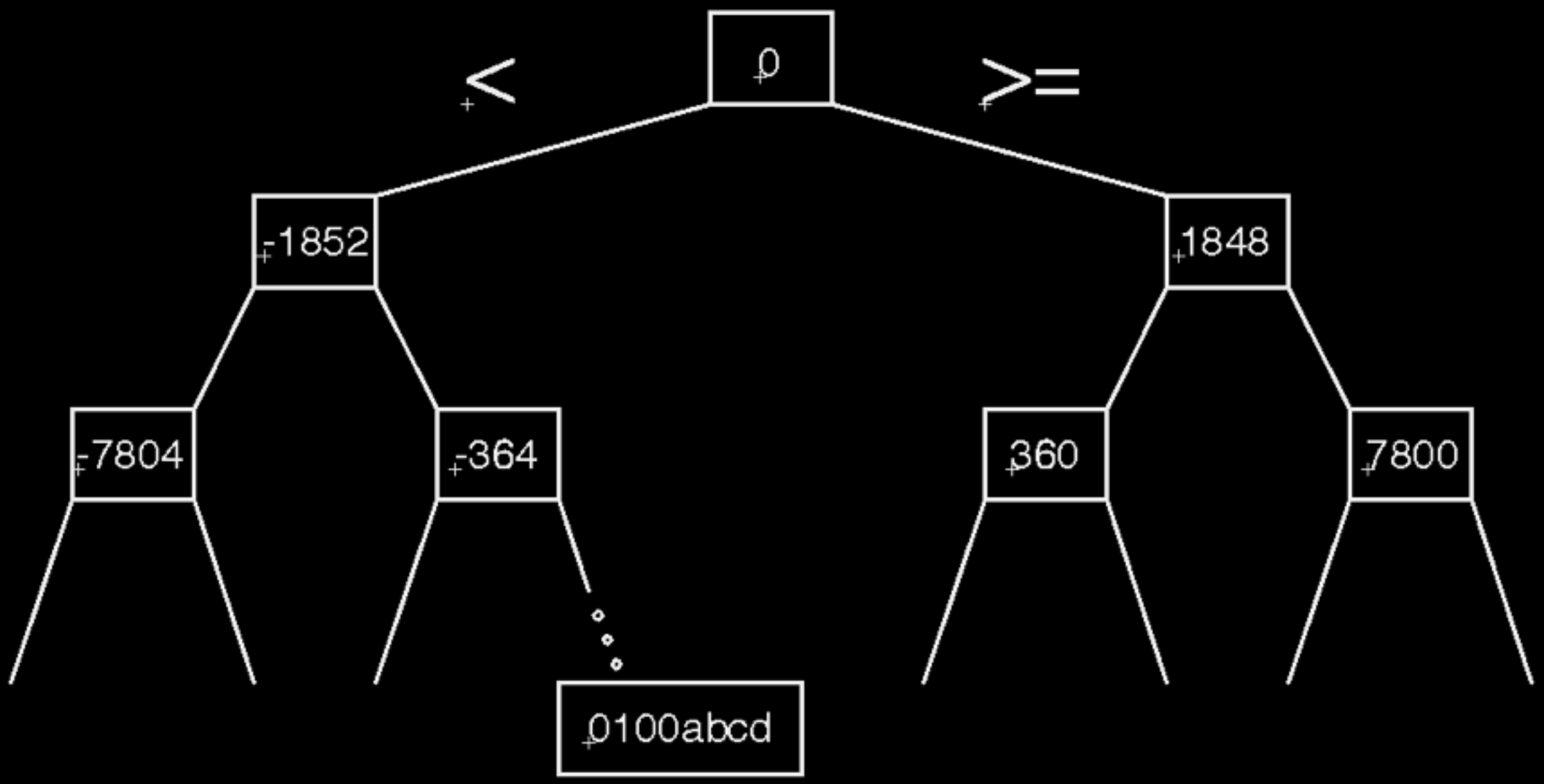
The receiver decompresses it,
does DSP and feeds the DAC.

Audio Compression (G.711)

Linear input value [note 1]	Compressed code XOR 11111111	Linear output value [note 2]
s00000001abcdx	s000abcd	s00000001abcd1
s0000001abcdxx	s001abcd	s0000001abcd10
s000001abcdxxx	s010abcd	s000001abcd100
s00001abcdxxxx	s011abcd	s00001abcd1000
s0001abcdxxxxx	s100abcd	s0001abcd10000
s001abcdxxxxxx	s101abcd	s001abcd100000
s01abcdxxxxxxx	s110abcd	s01abcd1000000
s1abcdxxxxxxxx	s111abcd	s1abcd10000000

Source: https://en.wikipedia.org/wiki/G.711

Audio Compression (G.711)



Audio Compression (G.711)

```
1   unsigned char uLawCompress(signed short sample)
2   {
3       // encode G.711 u-Law
4       if(sample > 31608)  sample = 31608;
5       if(sample < -31616) sample = -31616;
6
7       if(sample >= 0)
8       {
9           if(sample > 1848)
10          {
11              if(sample > 7800)
12              {
13                  if(sample > 15736)  return(0x80 + ((31608 - sample) >> 10));
14                  else                return(0x90 + ((15736 - sample) >> 9));
15              }
16              else
17              {
18                  if(sample > 3832)   return(0xA0 + ((7800 - sample) >> 8));
19                  else                return(0xB0 + ((3832 - sample) >> 7));
20              }
21          }
22          else
23          {
24              if(sample > 360)
25              {
26                  if(sample > 856)    return(0xC0 + ((1848 - sample) >> 6));
27                  else                return(0xD0 + ((856 - sample) >> 5));
28              }
29              else
30              {
31                  if(sample > 112)    return(0xE0 + ((360 - sample) >> 4));
32                  ...
```

Audio packets

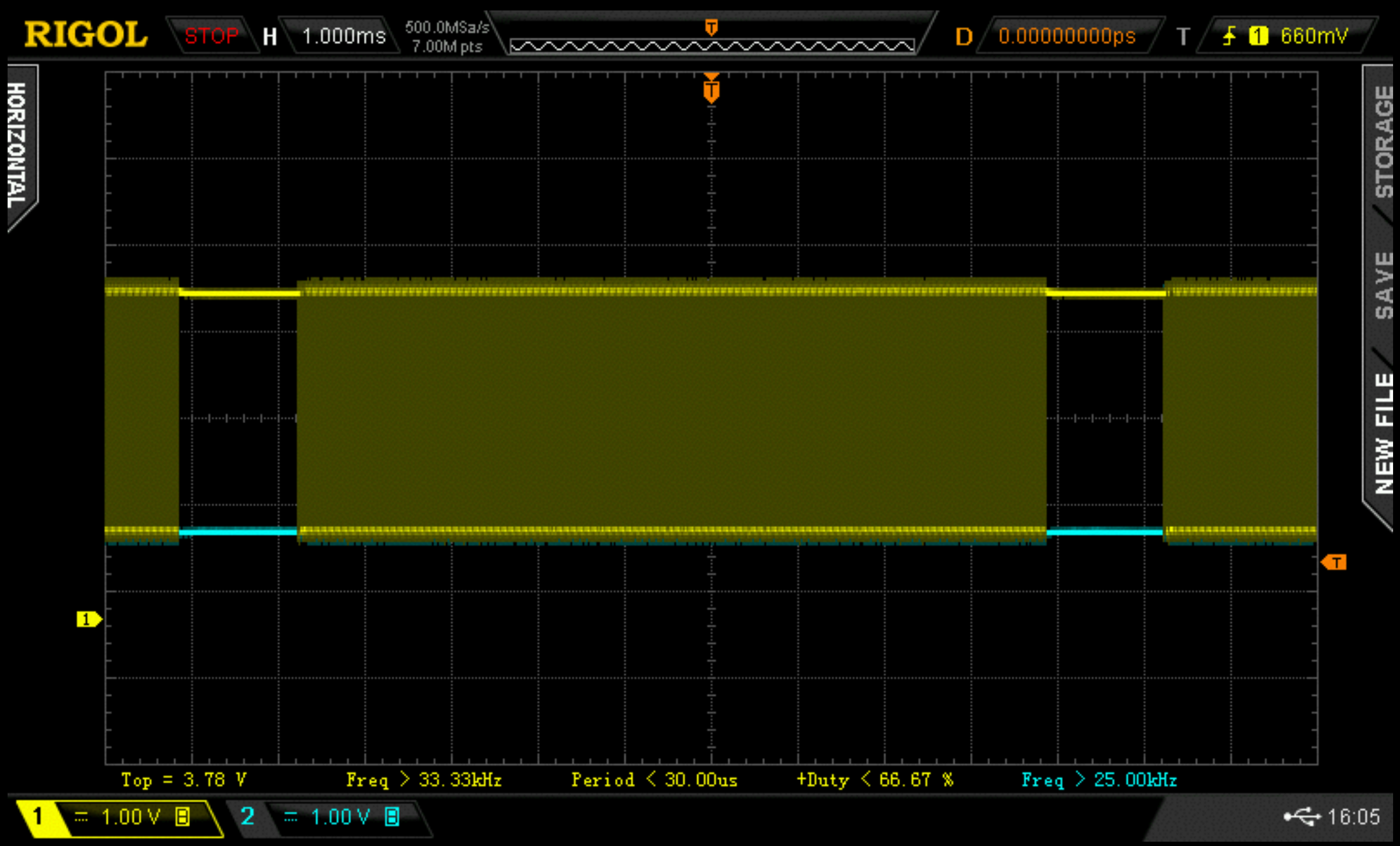
Untitled

0	7F	E0	01	A0	32	DC	34	40	EB	39	C0	21	70	F5	D2	D8
16	C1	A2	9C	20	3E	B2	F6	A0	25	40	04	09	46	EB	0C	F3
32	5B	61	0B	81	61	C3	C0	F8	68	87	22	C8	F2	D3	67	3C
48	0C	CE	F6	76	CB	BC	A9	16	66	54	BB	40	6E	C6	11	FD
64	7A	9A	01	02	9A	A0	48	59	7E	EB	A5	DB	77	E4	FE	BE
80	8C	4E	DC	32	2E	CC	7F	4F	12	03	DD	83	3A	7A	19	4E
96	0F	CC	AE	A4	1E	60	92	26	15	6B	74	78	00	90	14	50
112	AE	62	26	0C	C4	0B	98	90	3C	2B	94	37	EC	7B	E4	D2
128	E2	84	A0	FC	1B	1B	D2	3A	BD	D2	3E	0A	23	35	0E	E2
144	32	16	1A	48	C4	0E	47	93	E8	F4	8F	21	99	50	66	3E
160	44	8E	31	78	FA	E0	DC	44	66	8E	AE	55	DF	F9	2F	81
176	F7	62	4D	CC	F9	03	55	AF	5D	75	27	2A	FC	F2	35	E0
192	F8	99	68	9C	39	A1	DF	E9	0C	31	5F	53	DA	08	F1	11
208	89	87	AF	B2	BB	FB	0D	3D	B1	DD	04	D9	54	09	A5	83
224	7D	92	59	62	8C	2B	05	F7	86	F0	D2	E7	C5	40	32	42
240	35	77	F1	76	65	01	64	A2	68	91	DF	6B	56	3F	93	32
256	4E	EB	5F	94	DC	93	1C	C7	CB	3E	14	9E	13	0E	D0	DF
272	12	88	07	86	1C	0C	75	9C	2A	34	CB	53	50	1D	58	0B
288	EA	2C	24	A0	CD	77	02	BB	AA	9F	C3	EB	B7	05	DA	DF
304	D3	BE	F8	FC	F8	A5	ED	97	B8	ED	DC	17	9B	B8	67	4D
320	ED	01	65	4E												

4 bytes selected at offset 0 out of 324 bytes

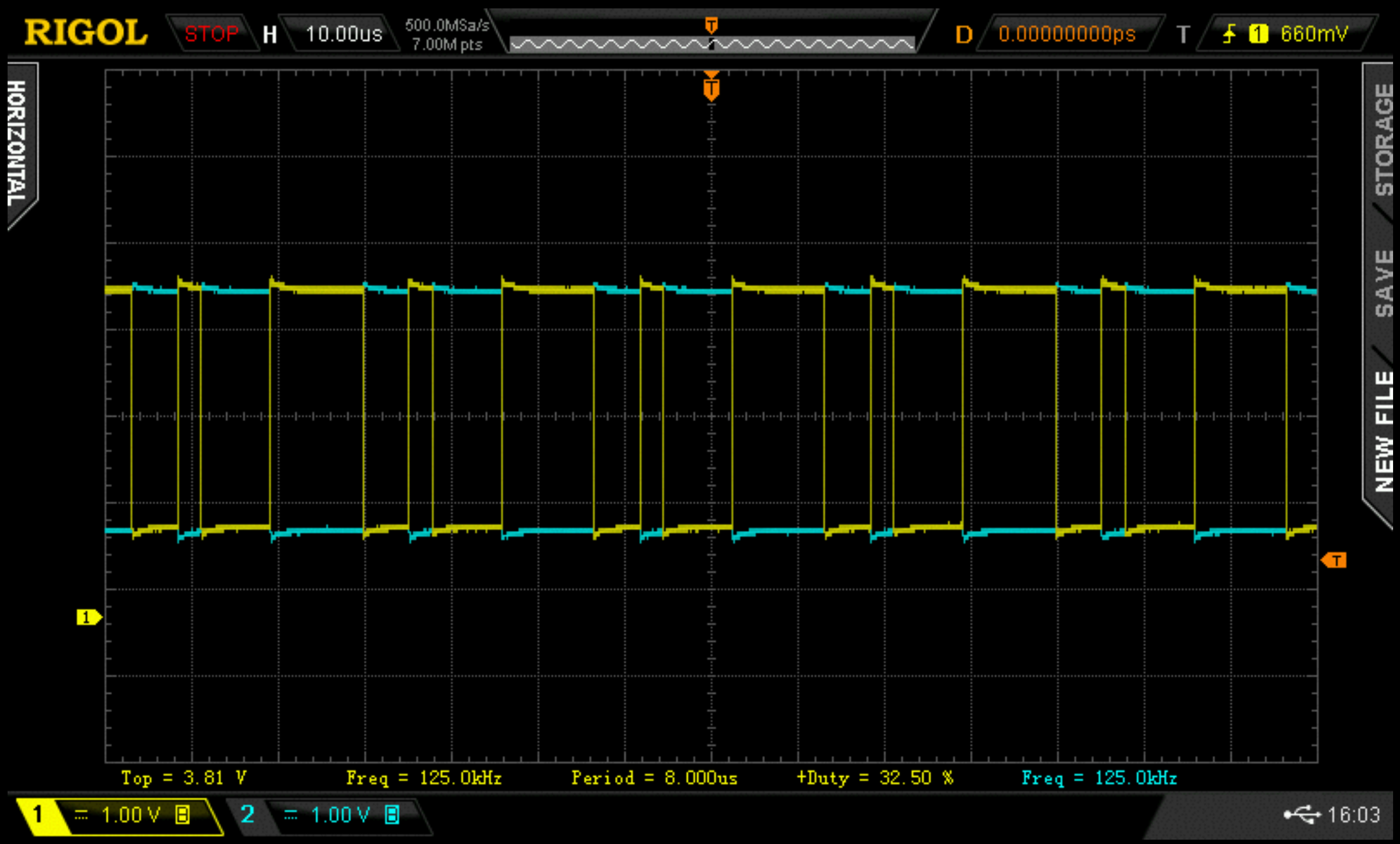
Audio packet (header highlighted)

Audio packets



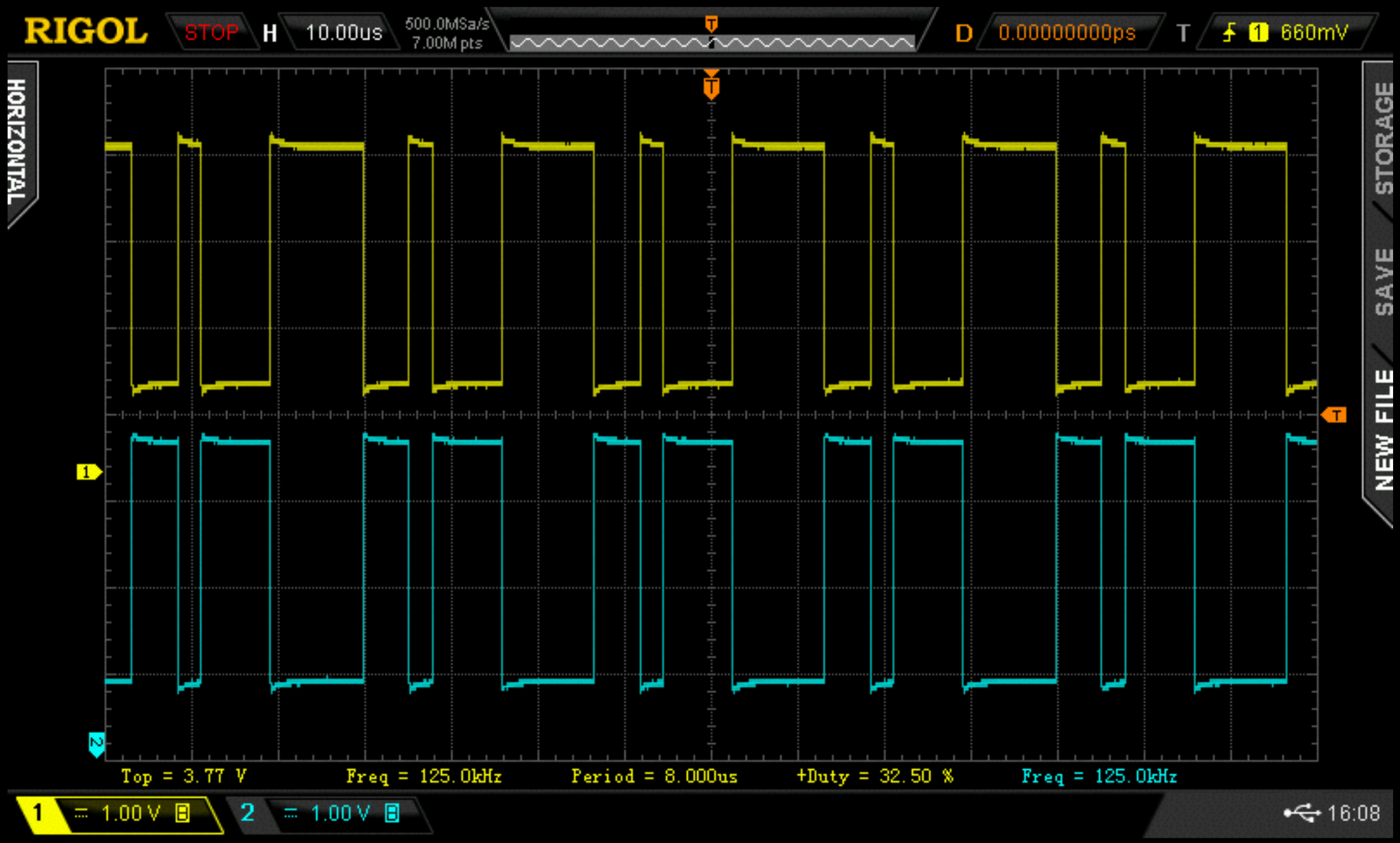
RS-485 waveform carrying audio data

Audio packets



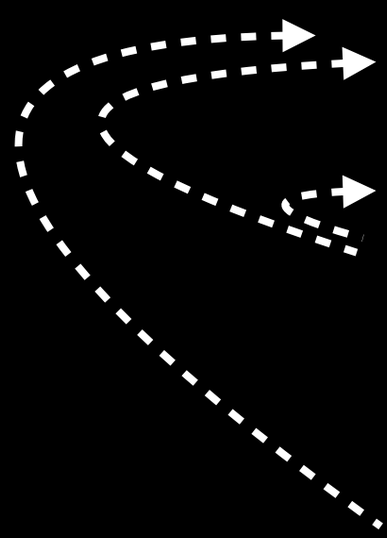
RS-485 waveform carrying audio data

Audio packets



RS-485 waveform carrying audio data

Synchronization and Decoding



- Synchronize to beginning of packet
- Decode command phase
- Decompress data section
- Desynchronize and repeat

Audio Decompression (G.711)

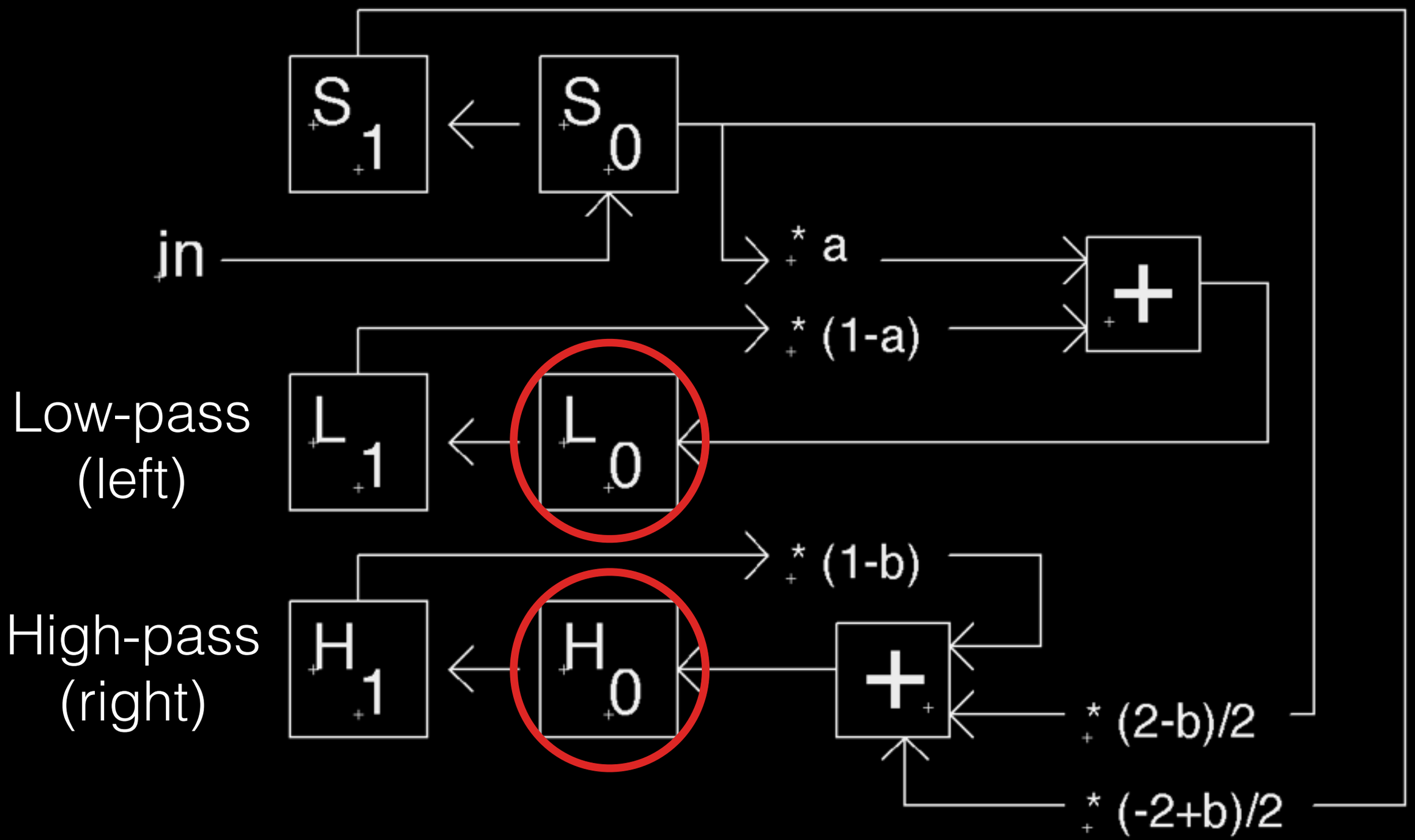
Linear input value [note 1]	Compressed code XOR 11111111	Linear output value [note 2]
s00000001abcdx	s000abcd	s00000001abcd1
s0000001abcdxx	s001abcd	s0000001abcd10
s000001abcdxxx	s010abcd	s000001abcd100
s00001abcdxxxx	s011abcd	s00001abcd1000
s0001abcdxxxxx	s100abcd	s0001abcd10000
s001abcdxxxxxx	s101abcd	s001abcd100000
s01abcdxxxxxxx	s110abcd	s01abcd1000000
s1abcdxxxxxxxx	s111abcd	s1abcd10000000

Source: https://en.wikipedia.org/wiki/G.711

Audio Decompression (G.711)

```
1   int g711ULawDecode0(int v) {return(-31616 + (v << 10));}
2   int g711ULawDecode1(int v) {return(-15744 + (v << 9));}
3   int g711ULawDecode2(int v) {return(-7804 + (v << 8));}
4   int g711ULawDecode3(int v) {return(-3836 + (v << 7));}
5   int g711ULawDecode4(int v) {return(-1852 + (v << 6));}
6   int g711ULawDecode5(int v) {return(-860 + (v << 5));}
7   int g711ULawDecode6(int v) {return(-364 + (v << 4));}
8   int g711ULawDecode7(int v) {return(-116 + (v << 3));}
9   int g711ULawDecode8(int v) {return(31608 - (v << 10));}
10  int g711ULawDecode9(int v) {return(15736 - (v << 9));}
11  int g711ULawDecodeA(int v) {return(7800 - (v << 8));}
12  int g711ULawDecodeB(int v) {return(3832 - (v << 7));}
13  int g711ULawDecodeC(int v) {return(1848 - (v << 6));}
14  int g711ULawDecodeD(int v) {return(856 - (v << 5));}
15  int g711ULawDecodeE(int v) {return(360 - (v << 4));}
16  int g711ULawDecodeF(int v) {return(112 - (v << 3));}
17
18  typedef int (*G711ULawDecoderFunction)(int v);
19
20  static G711ULawDecoderFunction G711ULawDecoder[16] =
21  {
22      &g711ULawDecode0, &g711ULawDecode1, &g711ULawDecode2, &g711ULawDecode3,
23      &g711ULawDecode4, &g711ULawDecode5, &g711ULawDecode6, &g711ULawDecode7,
24      &g711ULawDecode8, &g711ULawDecode9, &g711ULawDecodeA, &g711ULawDecodeB,
25      &g711ULawDecodeC, &g711ULawDecodeD, &g711ULawDecodeE, &g711ULawDecodeF
26  };
27
```

Digital Signal Processing



We turn the digital samples
into analog with a DAC.

The DAC supplies analog
audio to the amplifiers.

Left is for the woofer, right for the tweeter.

Conclusions

- It works!
- Not powerful enough
- Needs a programming button
- Should have input protection
- SMPS is not efficient at low current

Plan for next version

- All-digital DAC+amplifier (TI TAS57xx?)
- Big SPI flash for stored sound effects
- Volume control knob per speaker
- More efficient SMPS (TI TPS62xxx)
- May change back-panel connector

Questions?

Adventures in Ambient Audio

Kuy Mainwaring
@kuymainwaring