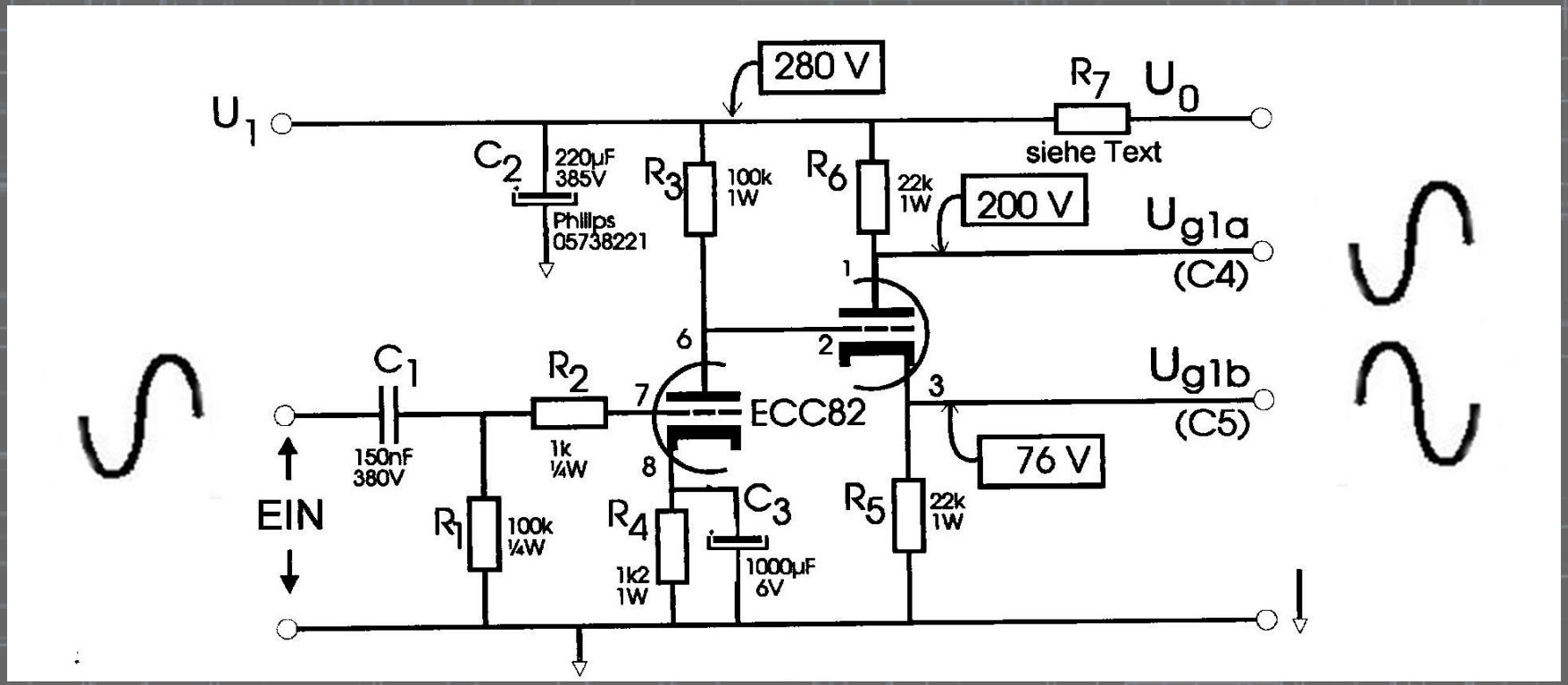
# Analysis of the WAVE file format in order to generate test signals

# Motivation: tube amplifier

-Creation of a asymmetric test signal to trace electronic circuits

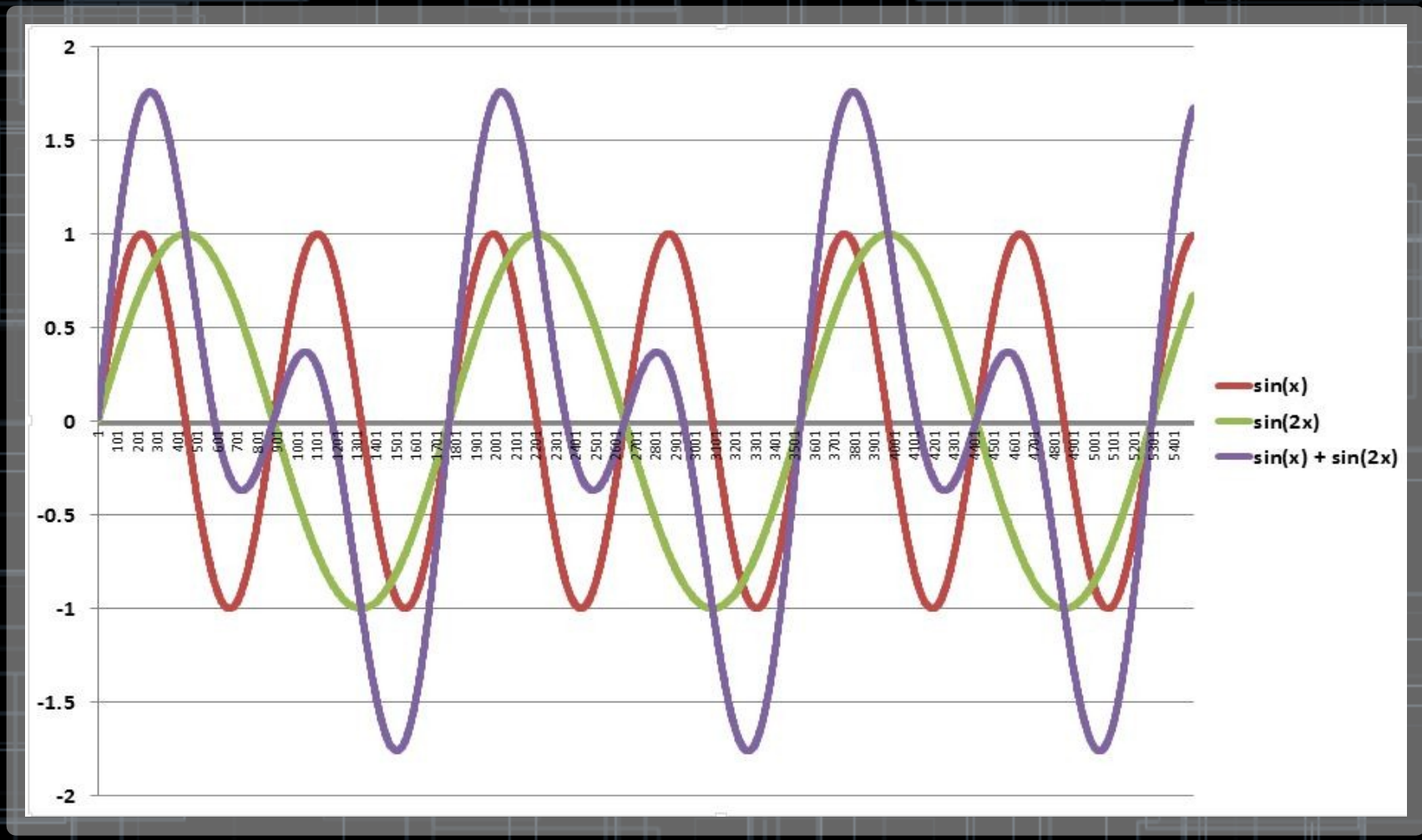
# Requirement: asymmetric signal to demonstrate function of phase splitter



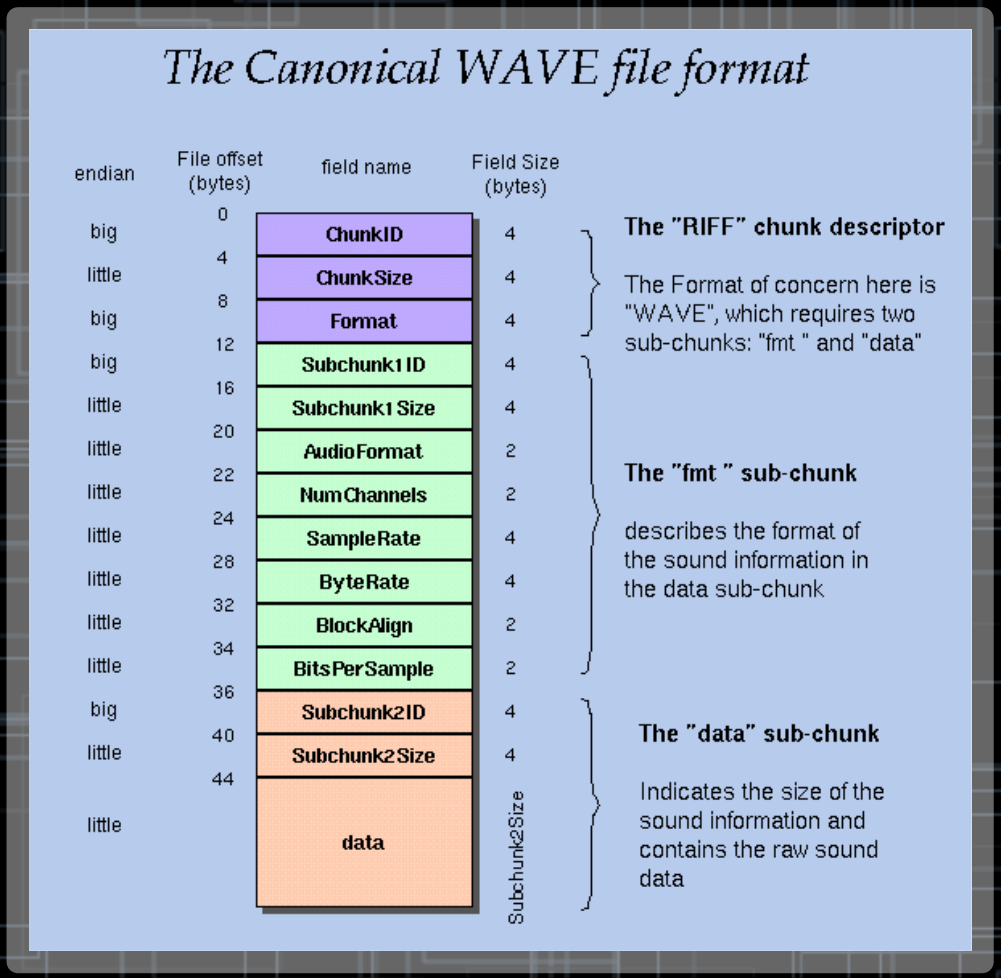
# Plan

- Analyze WAVE file format
- Calculate signal by adding multiple sine curves
- Write script to generate WAVE file
- Use pc sound card to play file to provide test signal

# Adding multiple sine curves



# Websearch


The Canonical WAVE file format
endian
File offset (bytes)
field name
Field Size (bytes)
big
little
big
big
little
little
little
little
little
little
big
little
little
0
4
8
12
16
20
22
24
28
32
34
36
40
44
ChunkID
ChunkSize
Format
Subchunk1ID
Subchunk1Size
AudioFormat
NumChannels
SampleRate
ByteRate
BlockAlign
BitsPerSample
Subchunk2ID
Subchunk2Size
data
4
4
4
4
4
2
2
4
4
2
2
4
4
Subchunk2Size
The "RIFF" chunk descriptor
The Format of concern here is "WAVE", which requires two sub-chunks: "fmt " and "data"
The "fmt " sub-chunk
describes the format of the sound information in the data sub-chunk
The "data" sub-chunk
Indicates the size of the sound information and contains the raw sound data

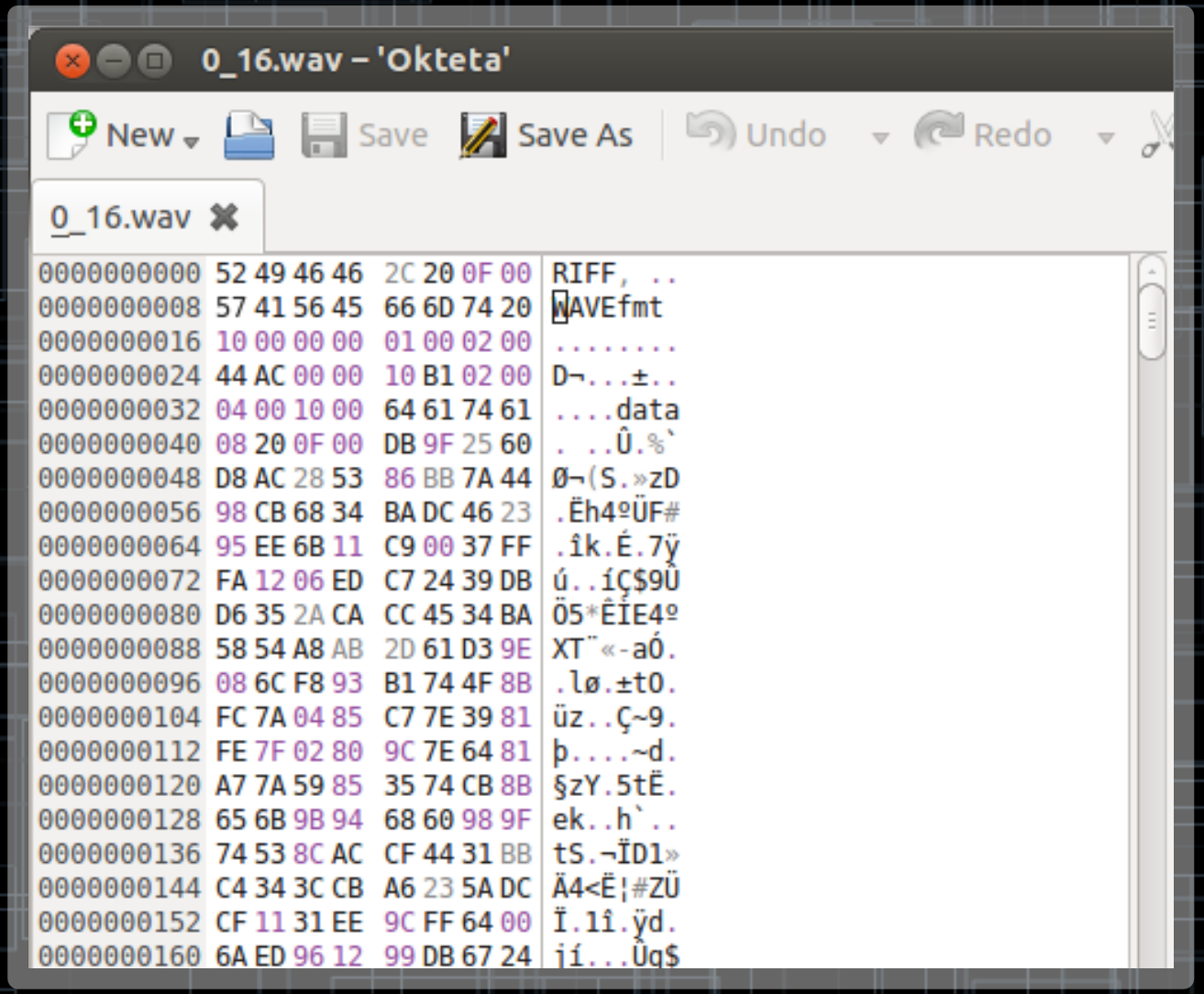
# Analysis of an existing .wav file

- 5.3 second long 1khz 16 bit stereo sine wave, 0 dBFS ( „0 decibel full scale“ meaning: using maximal possible amplitude at sine wave peak), 44.1 kHz sampling rate

- http://www.rme-audio.de/old/download/audtest.htm

- Hex file editor

# Display of test file in a hex editor



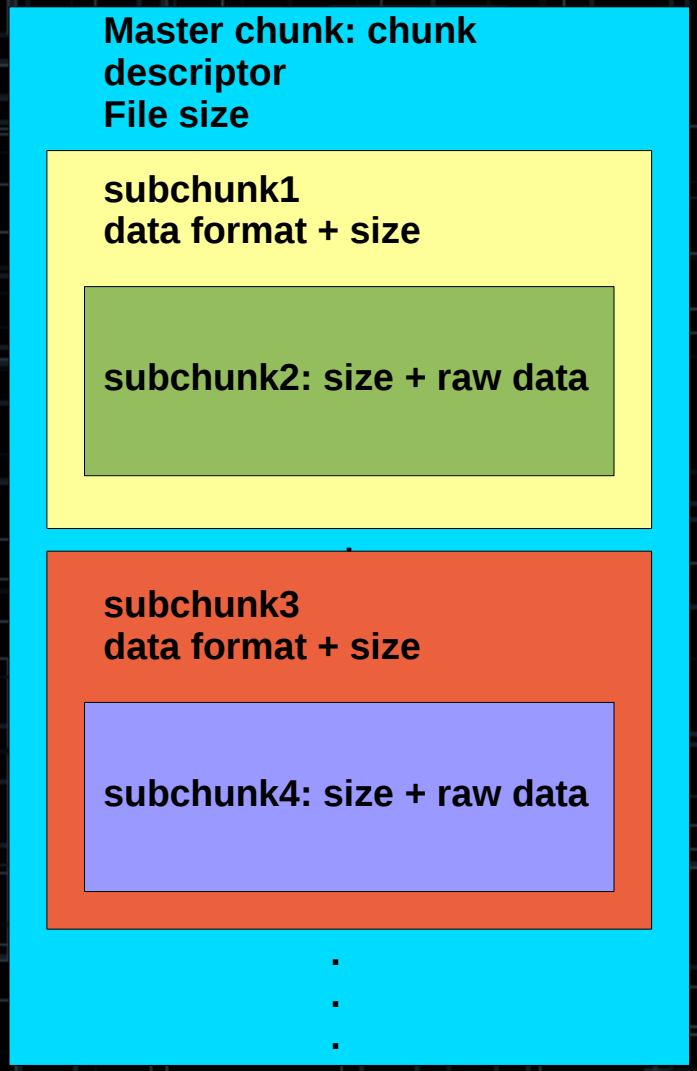
# RIFF: Resource Interchange File Format

- Developed by Microsoft + IBM 1991

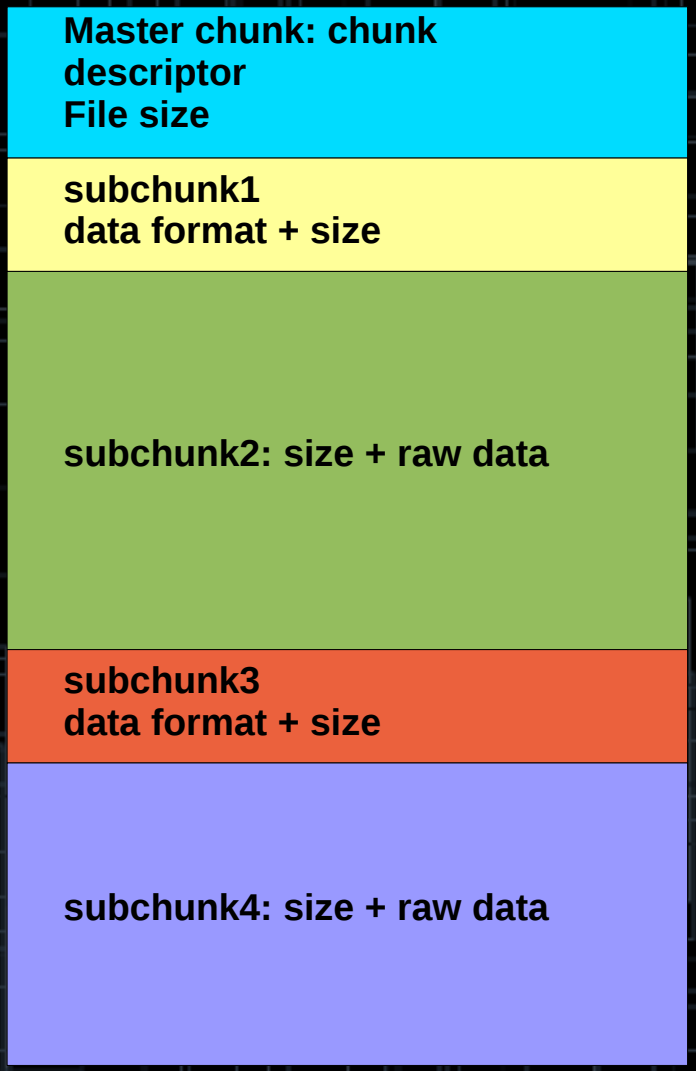
  generic container format for storage of multimedia data
- A RIFF file is composed of multiple discrete sections of data called chunks
- Chunk = Brocken / Klumpen

# RIFF: Container format for multimedia data

## logical structure




## implementation

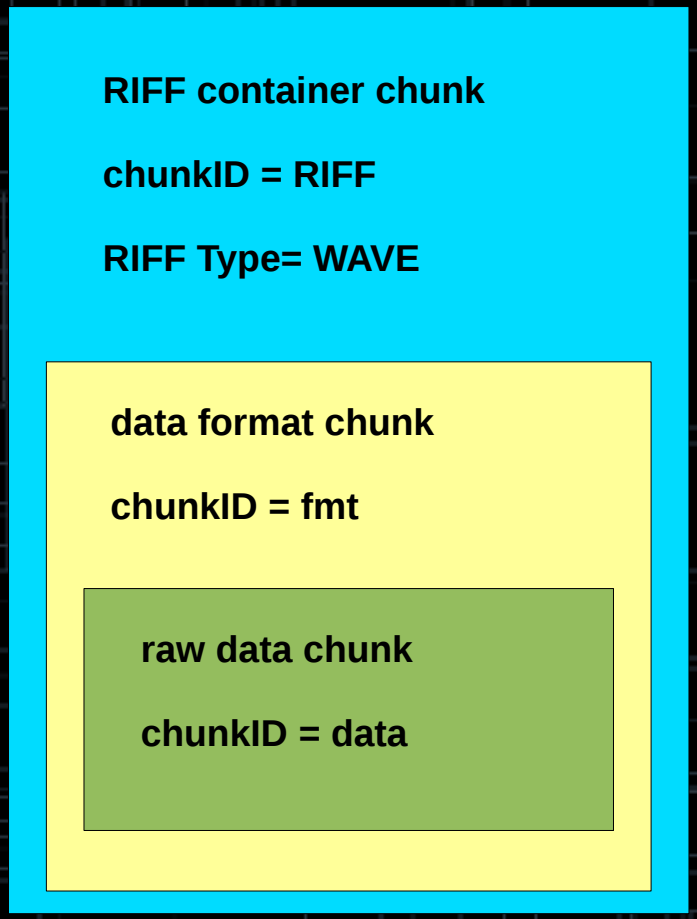


# WAVE: implementation of RIFF

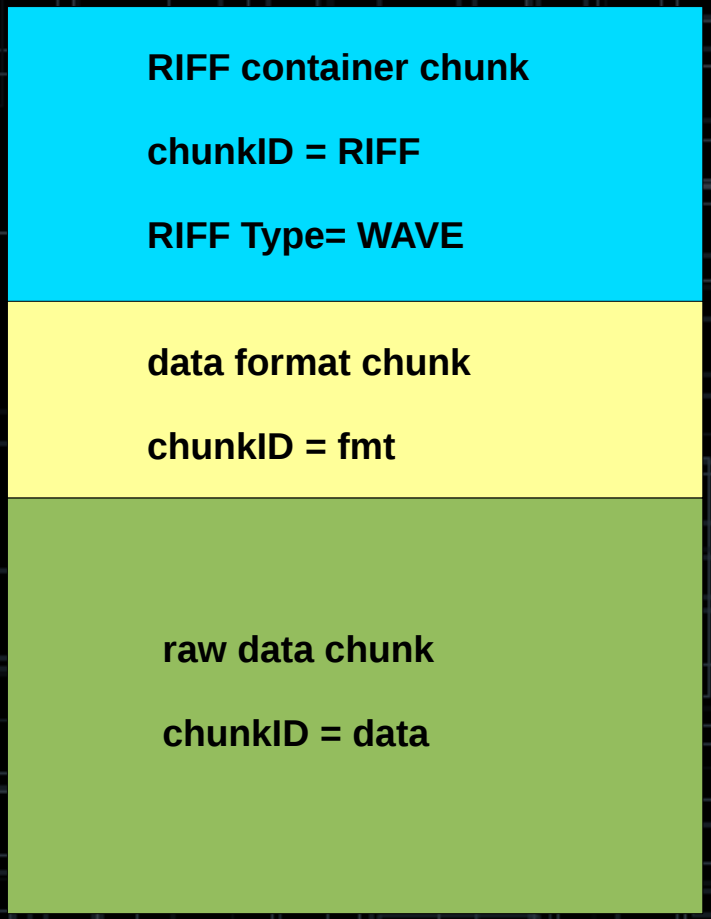
- WAVE: Waveform Audio File Format
- Implementation of RIFF for audio data
- Consists of three chunks
- RIFF container chunk
- fmt data format chunk
- data chunk with raw data
- Pulse code modulated data (PCM)

# WAVE file structure

logical structure




implementation



# What is Endianess or Byte Order ?

- Numbers exeeding memory register width (8bit) need to be distributed over several registers
- Byte order: order by which several one byte segments are read into 8bit memory

# About byte order

32 bit decimal integer  439'041'101

in binary is: 11010001010110011110001001101

| | | | | |
|---|---|---|---|---|
| binary | 00011010 | 00101011 | 00111100 | 01001101 |
| hexadecimal | 1A | 2B | 3C | 4D |

Big endian

1A 2B 3C 4D

Little endian

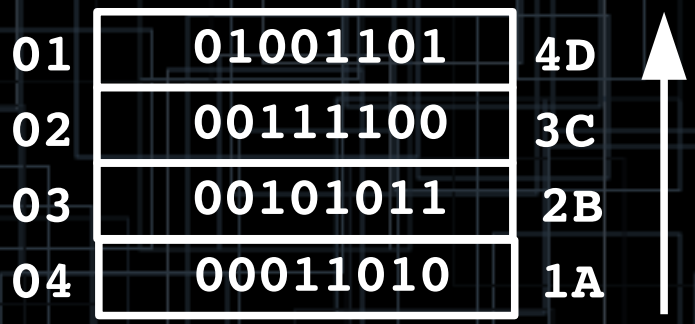
4D 3C 2B 1A

# About byte order

8 Bit Memory Register

Little Endian

4D 3C 2B 1A

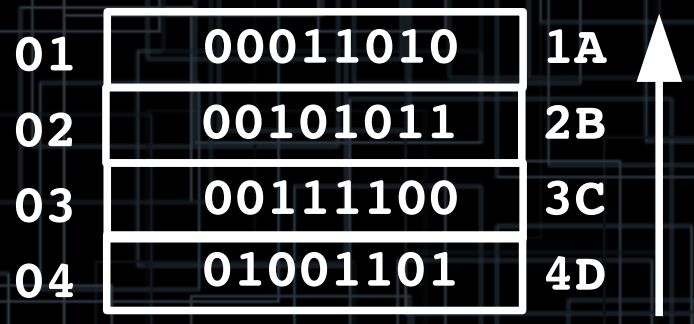



least significant byte

at samllest address

Big Endian

1A 2B 3C 4D

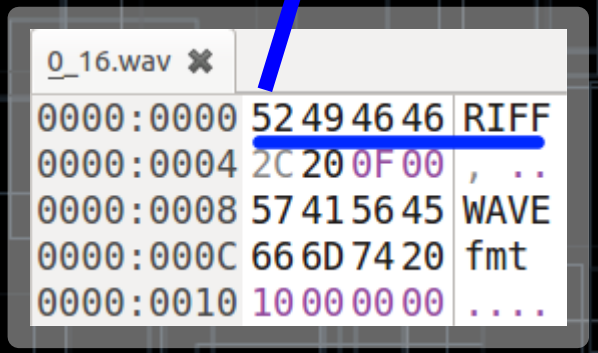



most significant byte

at samllest address

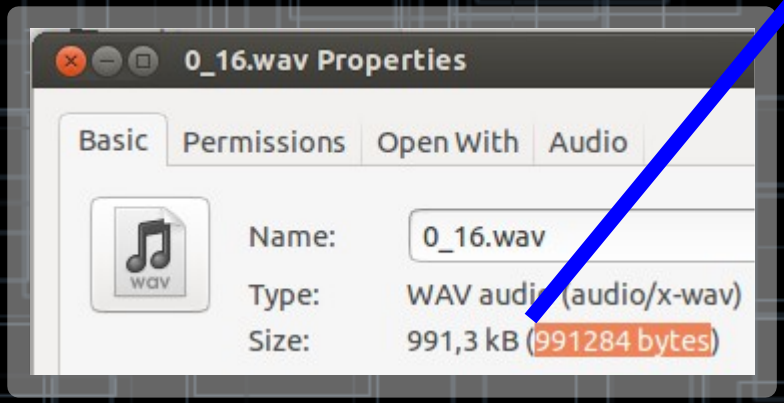
# Master chunk: ChunkID

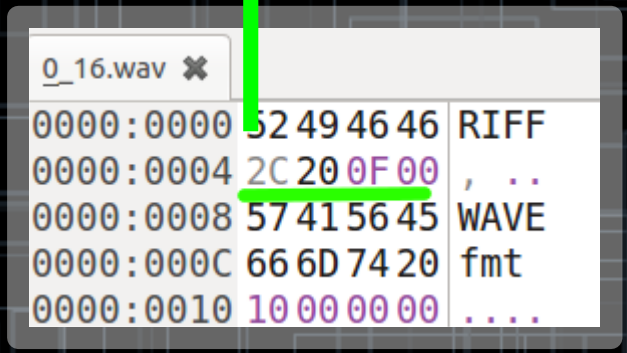
| Fieldname | ChunkID |
|---|---|
| Description | Contains the letters "RIFF" in ASCII |
| Decimal | 82 73 70 70 |
| Hex Big Endian | 52 49 46 46 |
| Hex Little Endian | n. a. |



# Master chunk: ChunkSize

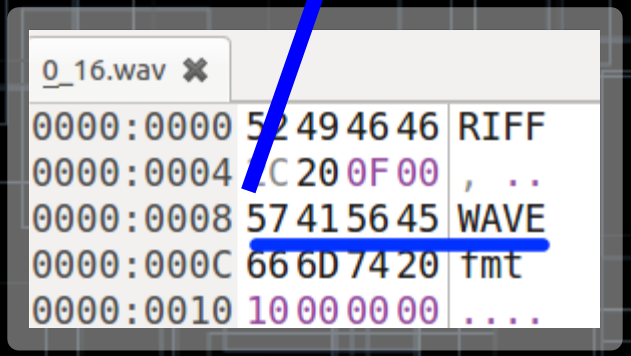
| Fieldname | ChunkSize |
|---|---|
| Description | Size of the entire file in bytes minus 8 bytes of the leading two fields |
| Decimal | 991'284 – 8 = 991'276 |
| Hex Big Endian | 00 0F 20 2C |
| Hex Little Endian | 2C 20 0F 00 |



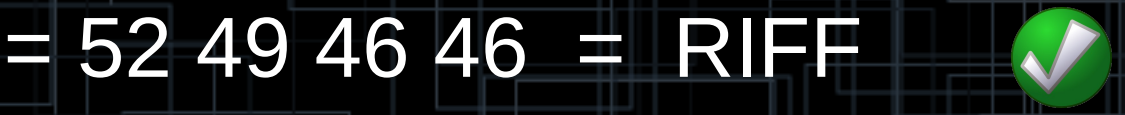
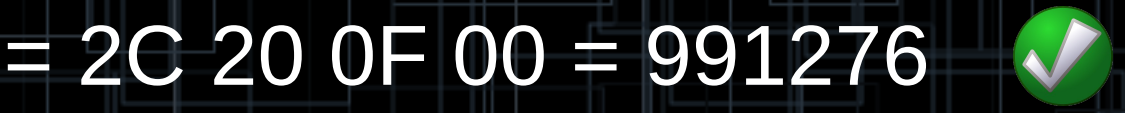
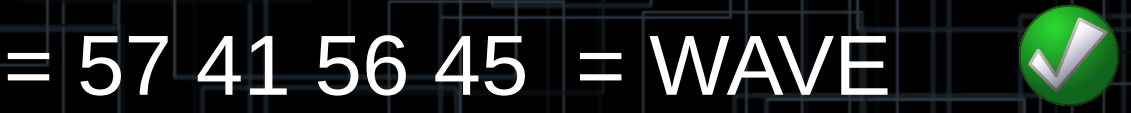



# Master chunk: Format

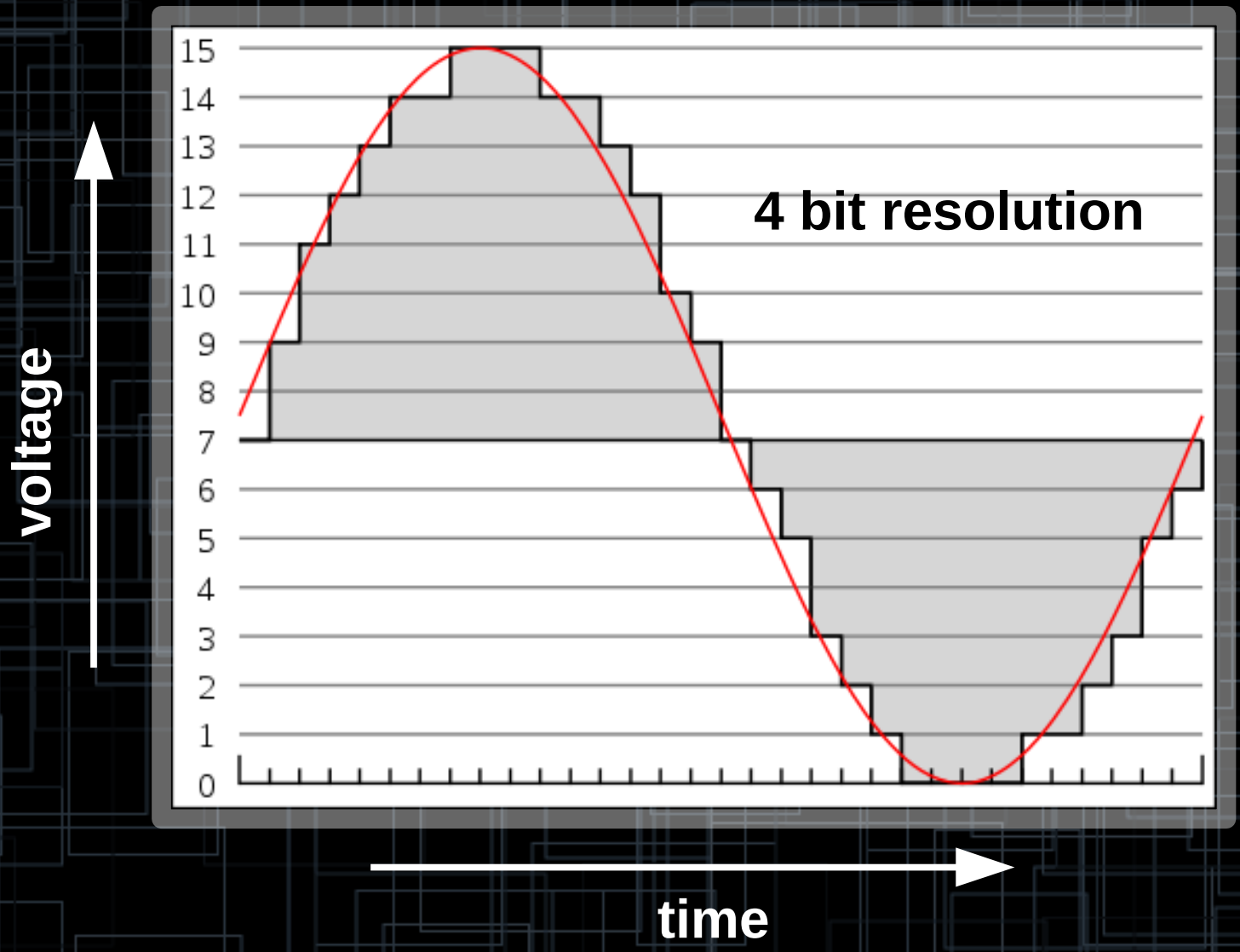
| Fieldname | Format |
|---|---|
| Description | Contains the letters "WAVE" in ASCII |
| Decimal | 87 65 86 69 |
| Hex Big Endian | 57 41 56 45 |
| Hex Little Endian | n. a. |



## RIFF chunk descriptor: done 

- ChunkID = 52 49 46 46 = RIFF 
- ChunkSize = 2C 20 0F 00 = 991276 
- Format = 57 41 56 45 = WAVE 

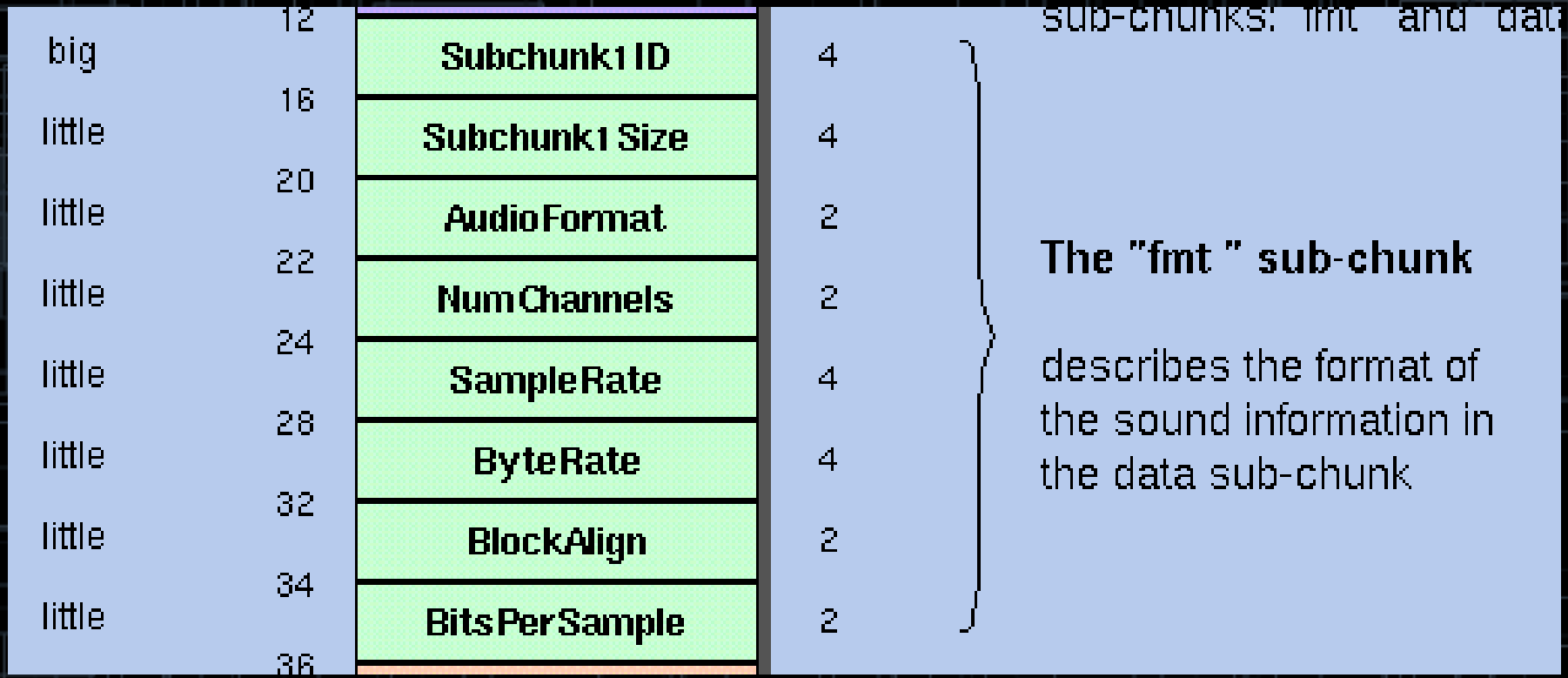
# PCM: Pulse Code Modulation



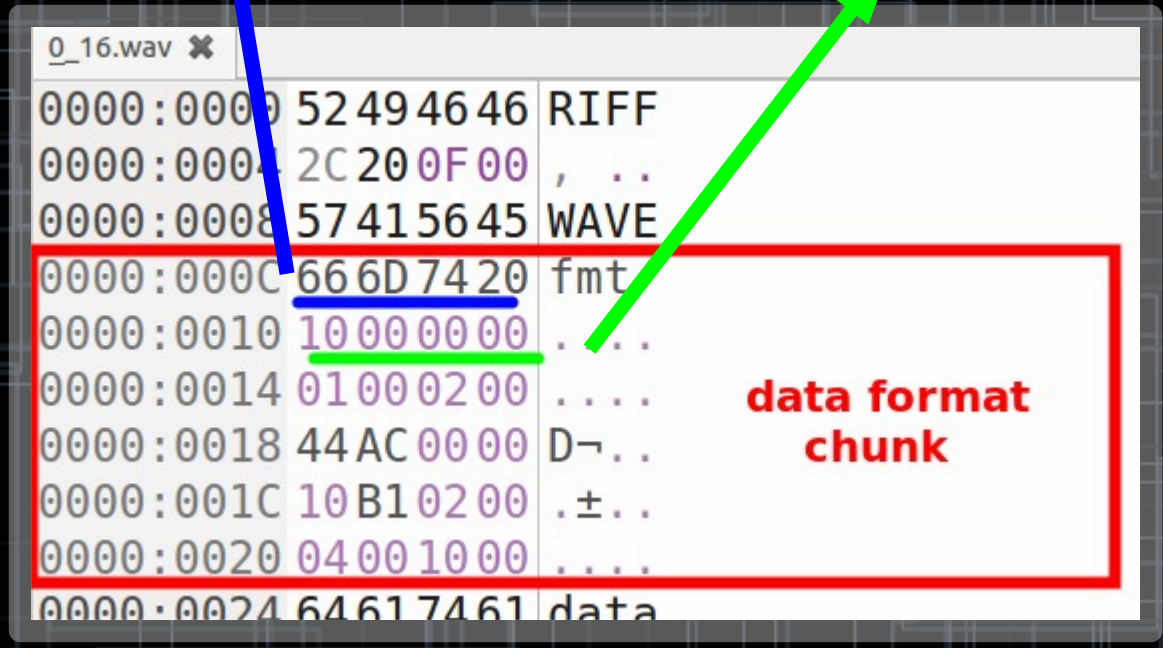
# Common PCM format

- CD (compact disc) format
- 2 Channels (stereo)
- 44'100 Hz sampling rate
- 16 bit resolution

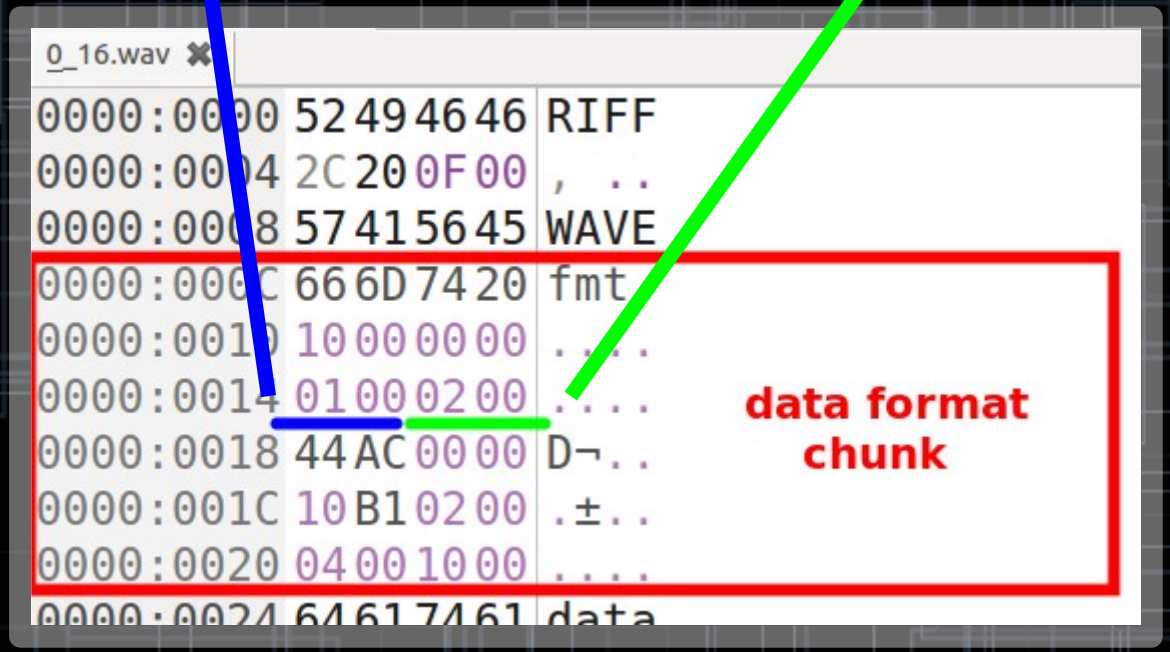
# The format, “fmt” sub-chunk



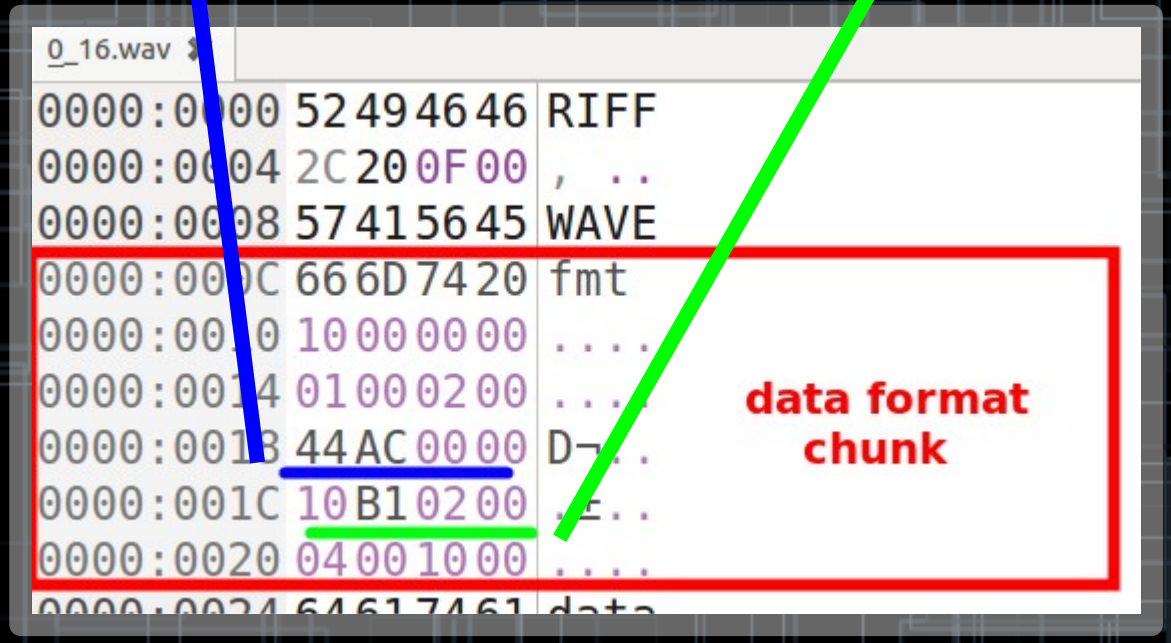
| Fieldname | Subchunk1ID | Subchunk1Size |
|---|---|---|
| Description | Contains the letters "fmt " in ASCII | Subchunk1 size is 16 for PCM. This is the size of the rest of the subchunk which follows this number. |
| Decimal | 102 109 116 032 | 16 |
| Hex Big Endian | 66 6D 74 20 | 00 00 00 10 |
| Hex Little Endian | n.a. | 10 00 00 00 |



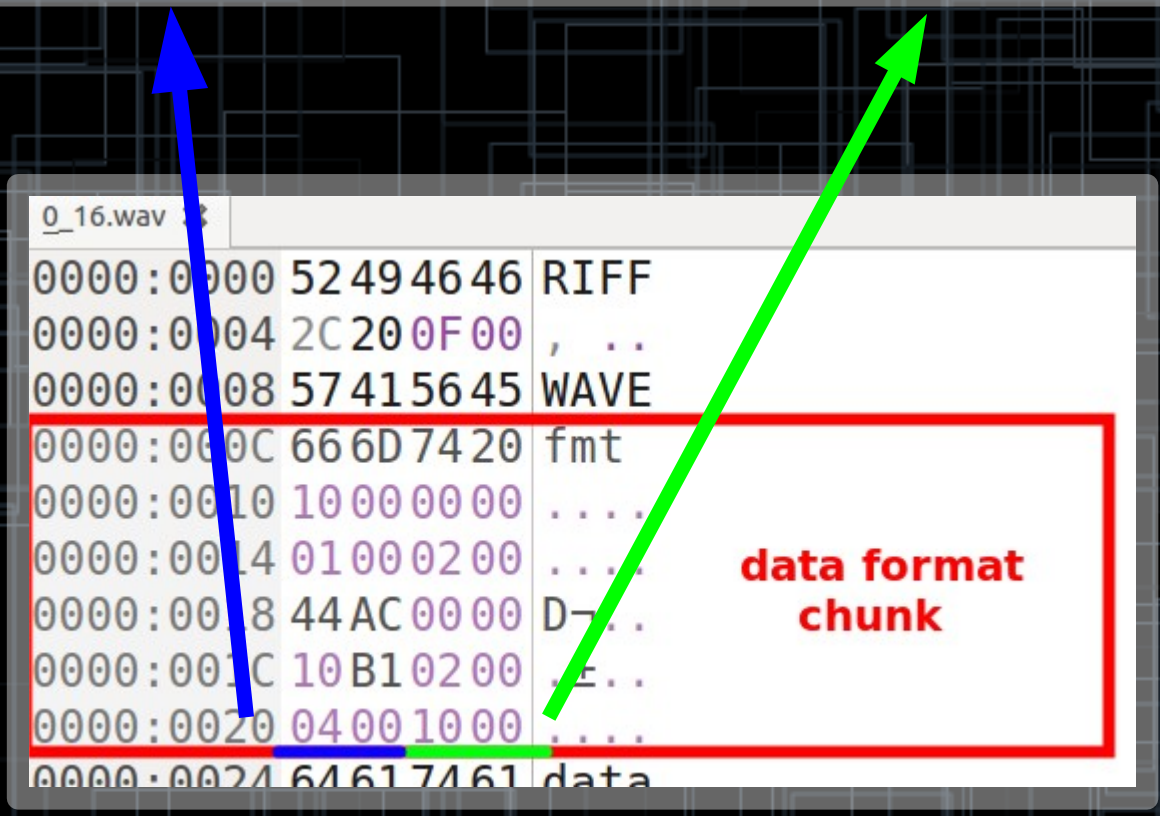
| Fieldname | AudioFormat | NumChannels |
|---|---|---|
| Description | PCM = 1 (i.e. Linear quantization) Values other than 1 indicate some form of compression. | Mono = 1, Stereo = 2, etc. |
| Decimal | 1 | 2 |
| Hex Big Endian | 00 01 | 00 02 |
| Hex Little Endian | 01 00 | 02 00 |



| Fieldname | SampleRate | ByteRate |
|---|---|---|
| Description | 8000 Hz , 44100 Hz, etc. | (SampleRate * NumChannels * BitsPerSample)/8 |
| Decimal | 44100 | 44100 x 2 x 16 / 8<br>= 176400 |
| Hex Big Endian | 00 00 AC 44 | 00 02 B1 10 |
| Hex Little Endian | 44 AC 00 00 | 10 B1 02 00 |



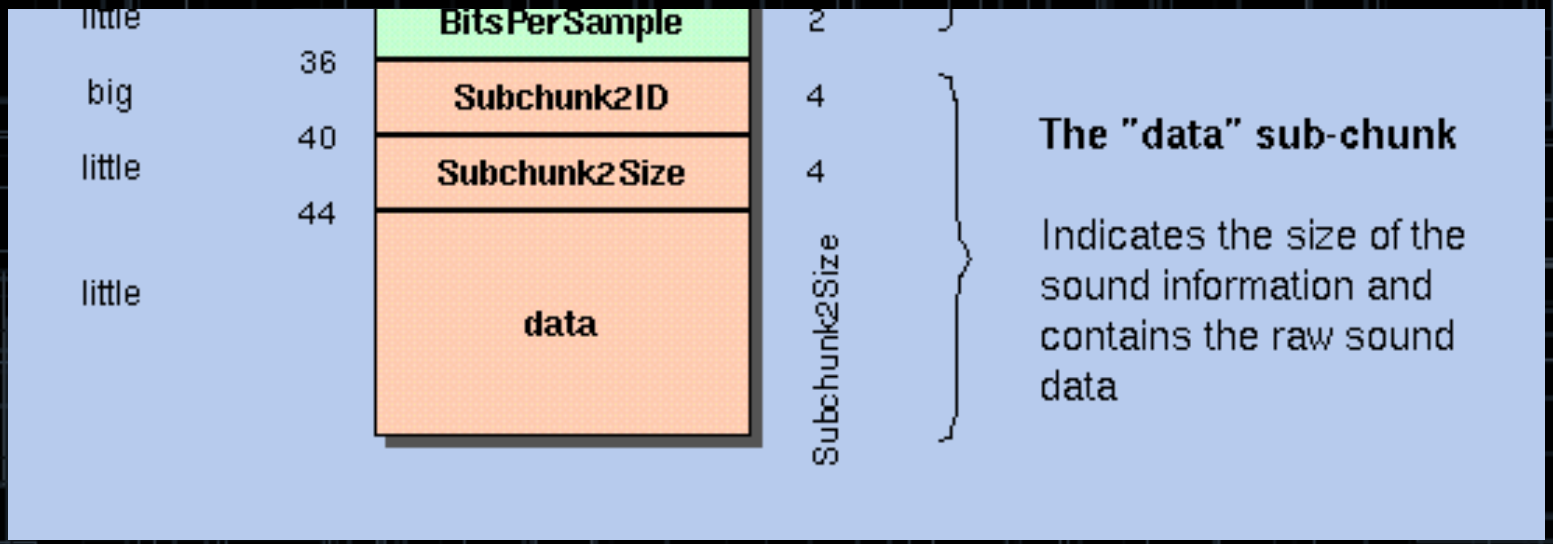
| Fieldname | BlockAlign | BitsPerSample |
|---|---|---|
| Description | The number of bytes for one sample including all channels | 8 bits = 8, 16 bits = 16, etc. |
| Decimal | 2 x 16 / 8 = 4 | 16 |
| Hex Big Endian | 00 04 | 00 10 |
| Hex Little Endian | 04 00 | 10 00 |



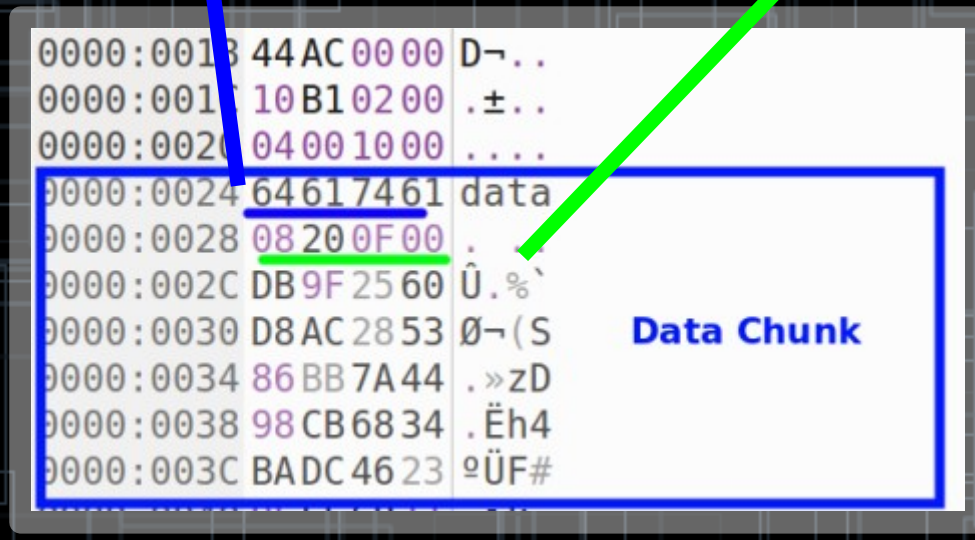
## “fmt” sub-chunk: done

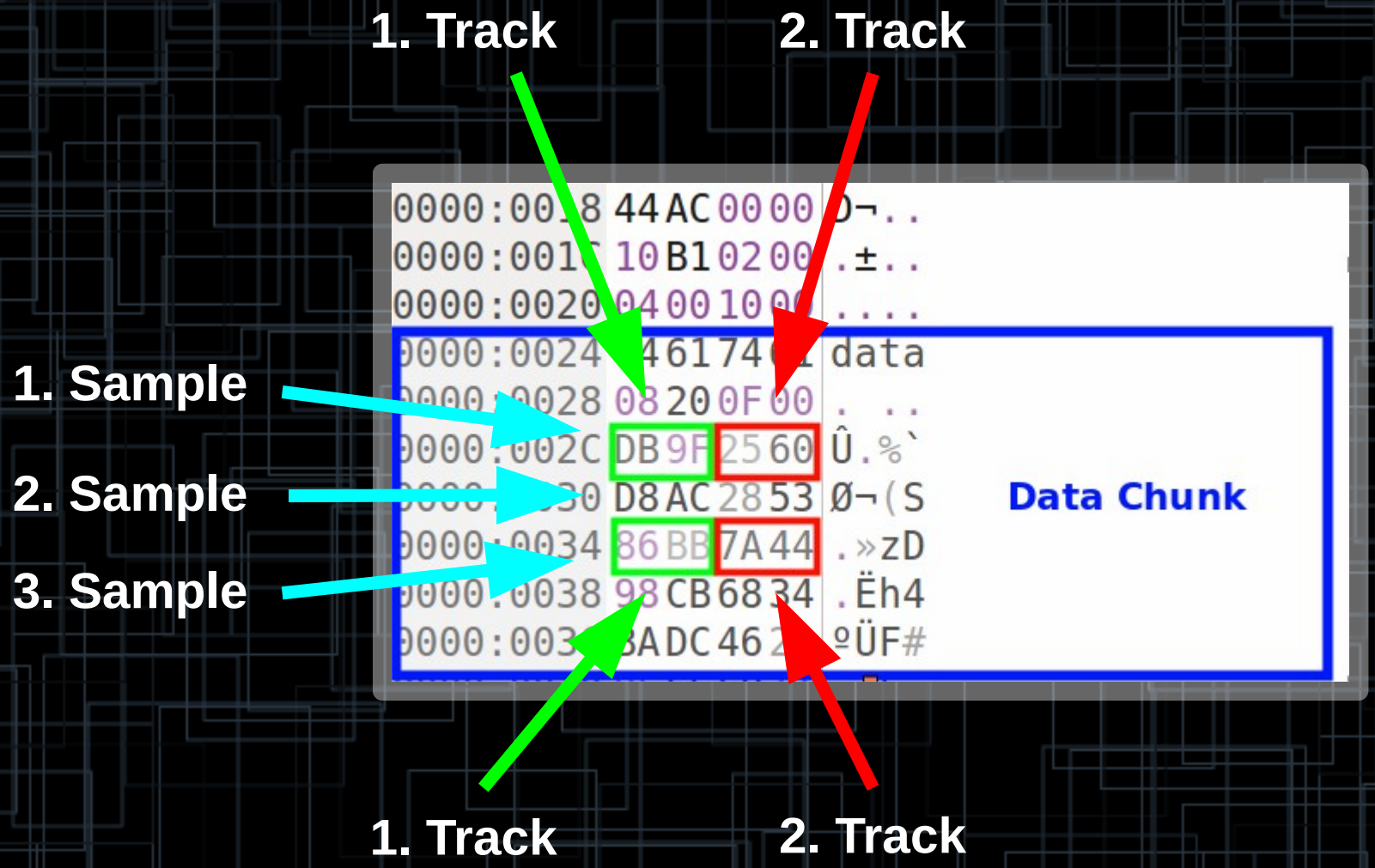
- Subchunk1ID = 66 6D 74 20 = fmt 
- Subchunk1Size = 10 00 00 00 = 16 
- AudioFormat = 01 00 = 1 
- NumChannels = 02 00 = 2 
- SampleRate = 44 AC 00 00 = 44100 
- ByteRate = 10 B1 02 00 = 176400 
- BlockAlign = 04 00 = 4 
- BitsPerSample = 10 00 = 16 

# The data sub-chunk



| Fieldname | Subchunk2ID | Subchunk2Size |
|---|---|---|
| Description | Contains the letters "data“ in ASCII | NumSamples * NumChannels * BitsPerSample/8<br>This is the number of bytes in the data. |
| Decimal | 100 97 116 100 | (5.619274376 sec x 44100) x 2 x 16/8 = 991240 |
| Hex Big Endian | 64 61 74 61 | 00 0F 20 08 |
| Hex Little Endian | n. a. | 08 20 0F 00 |

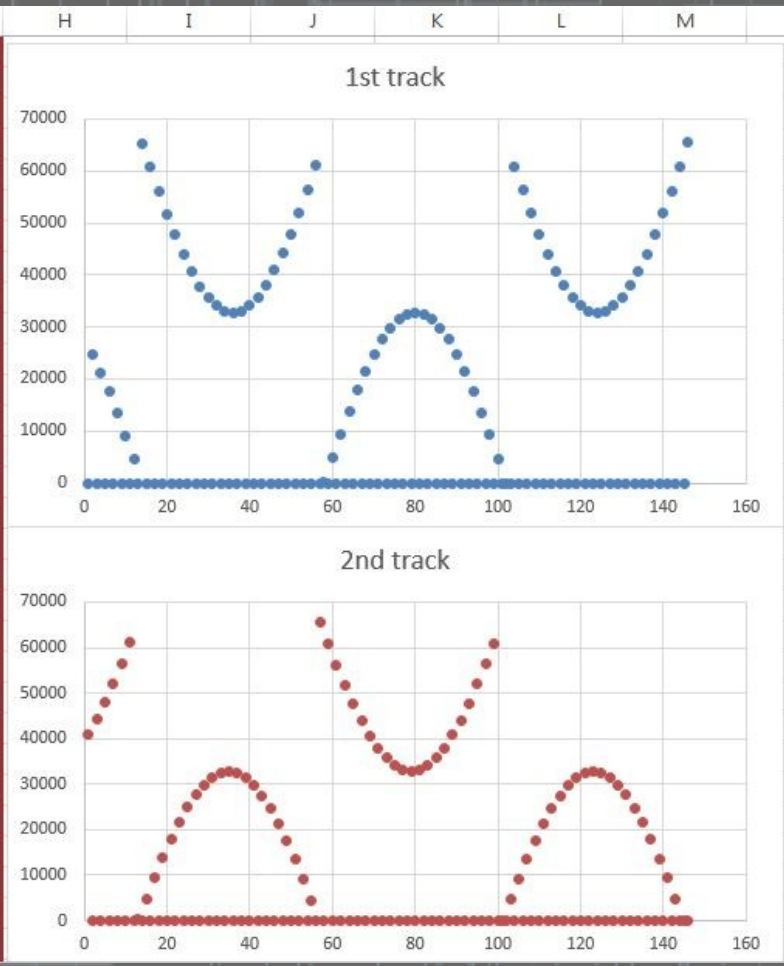



1. Track
2. Track
1. Sample
2. Sample
3. Sample
0000:0024 61 74 61
data
0000:0028 08 20 0F 00
0000:002C DB 9F 25 60
0000:0030 D8 AC 28 53
0000:0034 86 BB 7A 44
0000:0038 98 CB 68 34
Data Chunk
1. Track
2. Track

# Korn shell script “wave.sh”

# raw data “two's complement”

| | raw data (little endian) | little to big Endian | hex to dec | counter | 1st track | 2nd track |
|---|---|---|---|---|---|---|
| 2 | DB 9F | 9FDB | 40923 | 2 | FALSE | 40923 |
| 3 | 25 60 | 6025 | 24613 | 3 | 24613 | FALSE |
| 4 | D8 AC | ACD8 | 44248 | 4 | FALSE | 44248 |
| 5 | 28 53 | 5328 | 21288 | 5 | 21288 | FALSE |
| 6 | 86 BB | BB86 | 48006 | 6 | FALSE | 48006 |
| 7 | 7A 44 | 447A | 17530 | 7 | 17530 | FALSE |
| 8 | 98 CB | CB98 | 52120 | 8 | FALSE | 52120 |
| 9 | 68 34 | 3468 | 13416 | 9 | 13416 | FALSE |
| 10 | BA DC | DCBA | 56506 | 10 | FALSE | 56506 |
| 11 | 46 23 | 2346 | 9030 | 11 | 9030 | FALSE |
| 12 | 95 EE | EE95 | 61077 | 12 | FALSE | 61077 |
| 13 | 6B 11 | 116B | 4459 | 13 | 4459 | FALSE |
| 14 | C9 00 | 00C9 | 201 | 14 | FALSE | 201 |
| 15 | 37 FF | FF37 | 65335 | 15 | 65335 | FALSE |
| 16 | FA 12 | 12FA | 4858 | 16 | FALSE | 4858 |
| 17 | 06 ED | ED06 | 60678 | 17 | 60678 | FALSE |
| 18 | C7 24 | 24C7 | 9415 | 18 | FALSE | 9415 |
| 19 | 39 DB | DB39 | 56121 | 19 | 56121 | FALSE |
| 20 | D6 35 | 35D6 | 13782 | 20 | FALSE | 13782 |
| 21 | 2A CA | CA2A | 51754 | 21 | 51754 | FALSE |
| 22 | CC 45 | 45CC | 17868 | 22 | FALSE | 17868 |
| 23 | 34 BA | BA34 | 47668 | 23 | 47668 | FALSE |
| 24 | 58 54 | 5458 | 21592 | 24 | FALSE | 21592 |
| 25 | A8 AB | ABA8 | 43944 | 25 | 43944 | FALSE |
| 26 | 2D 61 | 612D | 24877 | 26 | FALSE | 24877 |
| 27 | D3 9E | 9ED3 | 40659 | 27 | 40659 | FALSE |
| 28 | 08 6C | 6C08 | 27656 | 28 | FALSE | 27656 |
| 29 | F8 93 | 93F8 | 37880 | 29 | 37880 | FALSE |
| 30 | B1 74 | 74B1 | 29873 | 30 | FALSE | 29873 |



# Resolution: 16 bit

$2^{16} = 65'536$

1, 2, 3 … 65535,65536

0, 1, 2 … 65534,65535

-32768,-32767,...,-1,0,1,...,32767

~~-32768~~ -32767,...,-1,0,1,...,32767

# two's complement

0,1,...,32767,32768,...,65535

-32768,.....,-1,0,1,...,32767

**+ 65536**

32768,...65535,0,1,...,32767

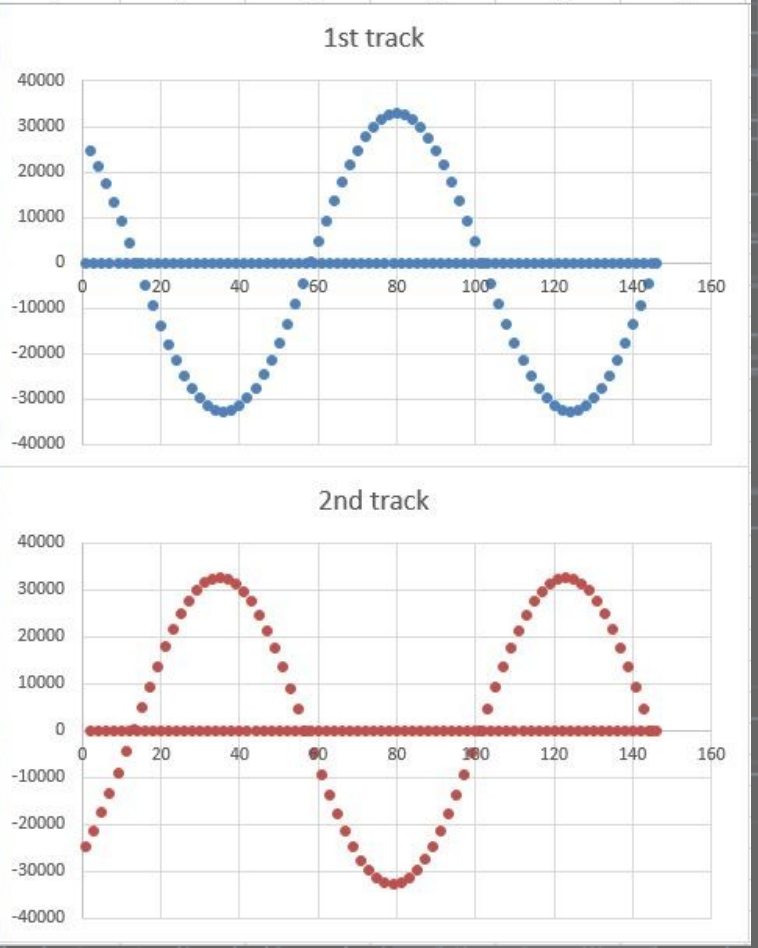
# why two's complement ?

- Allows to do binary substraction by addition
- Allows to drop cary bit

➔ simplifies design of computing units

# raw data transformed

| | A | B | C | D | E | G | H |
|---|---|---|---|---|---|---|---|
| 1 | raw data (little endian) | | little to big Endian | hex to dec | 2s complement | 1st track | 2nd track |
| 2 | DB | 9F | 9FDB | 40923 | -24613 | FALSE | -24613 |
| 3 | 25 | 60 | 6025 | 24613 | 24613 | 24613 | FALSE |
| 4 | D8 | AC | ACD8 | 44248 | -21288 | FALSE | -21288 |
| 5 | 28 | 53 | 5328 | 21288 | 21288 | 21288 | FALSE |
| 6 | 86 | BB | BB86 | 48006 | -17530 | FALSE | -17530 |
| 7 | 7A | 44 | 447A | 17530 | 17530 | 17530 | FALSE |
| 8 | 98 | CB | CB98 | 52120 | -13416 | FALSE | -13416 |
| 9 | 68 | 34 | 3468 | 13416 | 13416 | 13416 | FALSE |
| 10 | BA | DC | DCBA | 56506 | -9030 | FALSE | -9030 |
| 11 | 46 | 23 | 2346 | 9030 | 9030 | 9030 | FALSE |
| 12 | 95 | EE | EE95 | 61077 | -4459 | FALSE | -4459 |
| 13 | 6B | 11 | 116B | 4459 | 4459 | 4459 | FALSE |
| 14 | C9 | 00 | 00C9 | 201 | 201 | FALSE | 201 |
| 15 | 37 | FF | FF37 | 65335 | -201 | -201 | FALSE |
| 16 | FA | 12 | 12FA | 4858 | 4858 | FALSE | 4858 |
| 17 | 06 | ED | ED06 | 60678 | -4858 | -4858 | FALSE |
| 18 | C7 | 24 | 24C7 | 9415 | 9415 | FALSE | 9415 |
| 19 | 39 | DB | DB39 | 56121 | -9415 | -9415 | FALSE |
| 20 | D6 | 35 | 35D6 | 13782 | 13782 | FALSE | 13782 |
| 21 | 2A | CA | CA2A | 51754 | -13782 | -13782 | FALSE |
| 22 | CC | 45 | 45CC | 17868 | 17868 | FALSE | 17868 |
| 23 | 34 | BA | BA34 | 47668 | -17868 | -17868 | FALSE |
| 24 | 58 | 54 | 5458 | 21592 | 21592 | FALSE | 21592 |
| 25 | A8 | AB | ABA8 | 43944 | -21592 | -21592 | FALSE |
| 26 | 2D | 61 | 612D | 24877 | 24877 | FALSE | 24877 |
| 27 | D3 | 9E | 9ED3 | 40659 | -24877 | -24877 | FALSE |
| 28 | 08 | 6C | 6C08 | 27656 | 27656 | FALSE | 27656 |
| 29 | F8 | 93 | 93F8 | 37880 | -27656 | -27656 | FALSE |
| 30 | B1 | 74 | 74B1 | 29873 | 29873 | FALSE | 29873 |



## data sub-chunk: done

- ChunkID = 64 61 74 61 = data

- Subchunk2Size = 08 20 0F 00 = 991240

- data = two's complement

## program

- 1. part : master chunk, subchunk1, subchunk2 (headers)
  - Easy: constant for identical size ( 1 minute)
- 2. part : payload
  - For 1 minute : 44100 x 60 iterations of sin(x) and its transformation to little endian  2's complement

# from sin(x) to raw data

| sine function | scale to 32767 |
|---|---|
| | |
| sin(x) | sin(x) * 32767 |
| | |
| | |
| -1 ... 0 ... 1 | -32767 ... 0 ... 32767 |
| | |
| sin( 0.05) = 0.0499791 | 1637.6674 |
| sin(-0.05) = -0.0499791 | -1637.6674 |
| sin( 0.5 ) = 0.4794255 | 15709.337 |
| sin(-0.5 ) = -0.4794255 | -15709.337 |
| sin( 1.2 ) = 0.9320391 | 30540.125 |
| sin(-1.2 ) = -0.9320391 | -30540.125 |

# from sin(x) to raw data

| scale to 32767 | round to 16 bit integer |
|---|---|
| sin(x) * 32767 | round(sin(x) * 32767) |
| -32767 ... 0 ... 32767 | -32767 ... 0 ... 32767 |
| 1637.6674 | 1638 |
| -1637.6674 | -1638 |
| 15709.337 | 15709 |
| -15709.337 | -15709 |
| 30540.125 | 30540 |
| -30540.125 | -30540 |

# from sin(x) to raw data

| round to 16 bit integer | two's complent |
|---|---|
| | |
| round(sin(x) * 32767) | sin(x) > 0: round(sin(x) * 32767)<br>sin(x) < 0: round(sin(x) * 32767) + 65536 |
| | |
| -32767 ... 0 ... 32767 | 32769 ... 65535, 0, 1 ... 32767 |
| | |
| 1638 | 1638 |
| -1638 | 63898 |
| 15709 | 15709 |
| -15709 | 49827 |
| 30540 | 30540 |
| -30540 | 34996 |

# from sin(x) to raw data

| two's complent | hex big endian |
|---|---|
| | |
| sin(x) > 0: round(sin(x) * 32767)<br>sin(x) < 0: round(sin(x) * 32767) + 65536 | dec to hex |
| | |
| 32769 ... 65535, 0, 1 ... 32767 | 0x8001 ... 0xFFFF, 0x0000, 0x0001 ... 0x7FFF |
| | |
| 1638 | 0x0666 |
| 63898 | 0xF99A |
| 15709 | 0x3D5D |
| 49827 | 0xC2A3 |
| 30540 | 0x774C |
| 34996 | 0x88B4 |

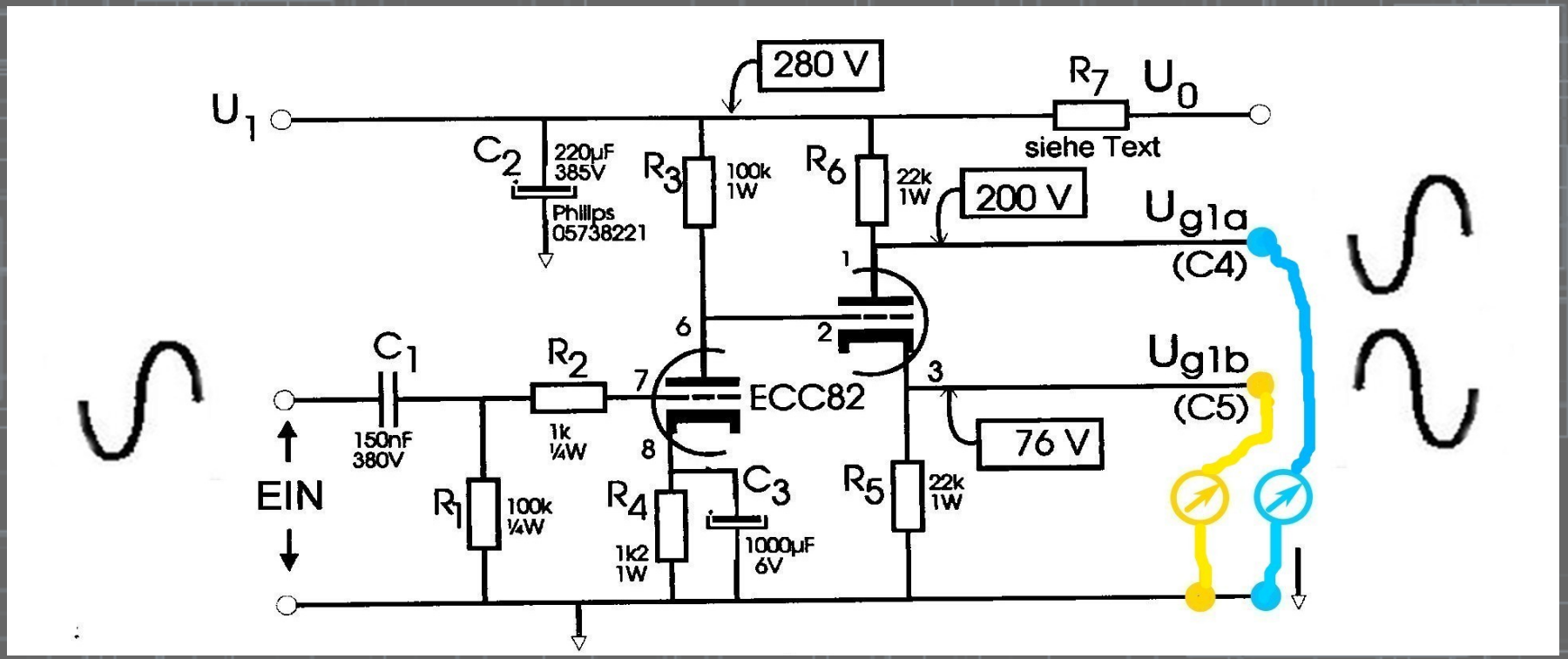
# from sin(x) to raw data

| hex big endian | hex little endian |
|---|---|
| | |
| dec to hex | big to little endian |
| | |
| | |
| 0x8001 ... 0xFFFF, 0x0000, 0x0001 ... 0x7FFF | 0x0180 ... 0xFFFF, 0x0000, 0x0100 ... 0xFF7F |
| | |
| 0x0666 | 0x6606 |
| 0xF99A | 0x9AF9 |
| 0x3D5D | 0x5D3D |
| 0xC2A3 | 0xA3C2 |
| 0x774C | 0x4C77 |
| 0x88B4 | 0xB488 |

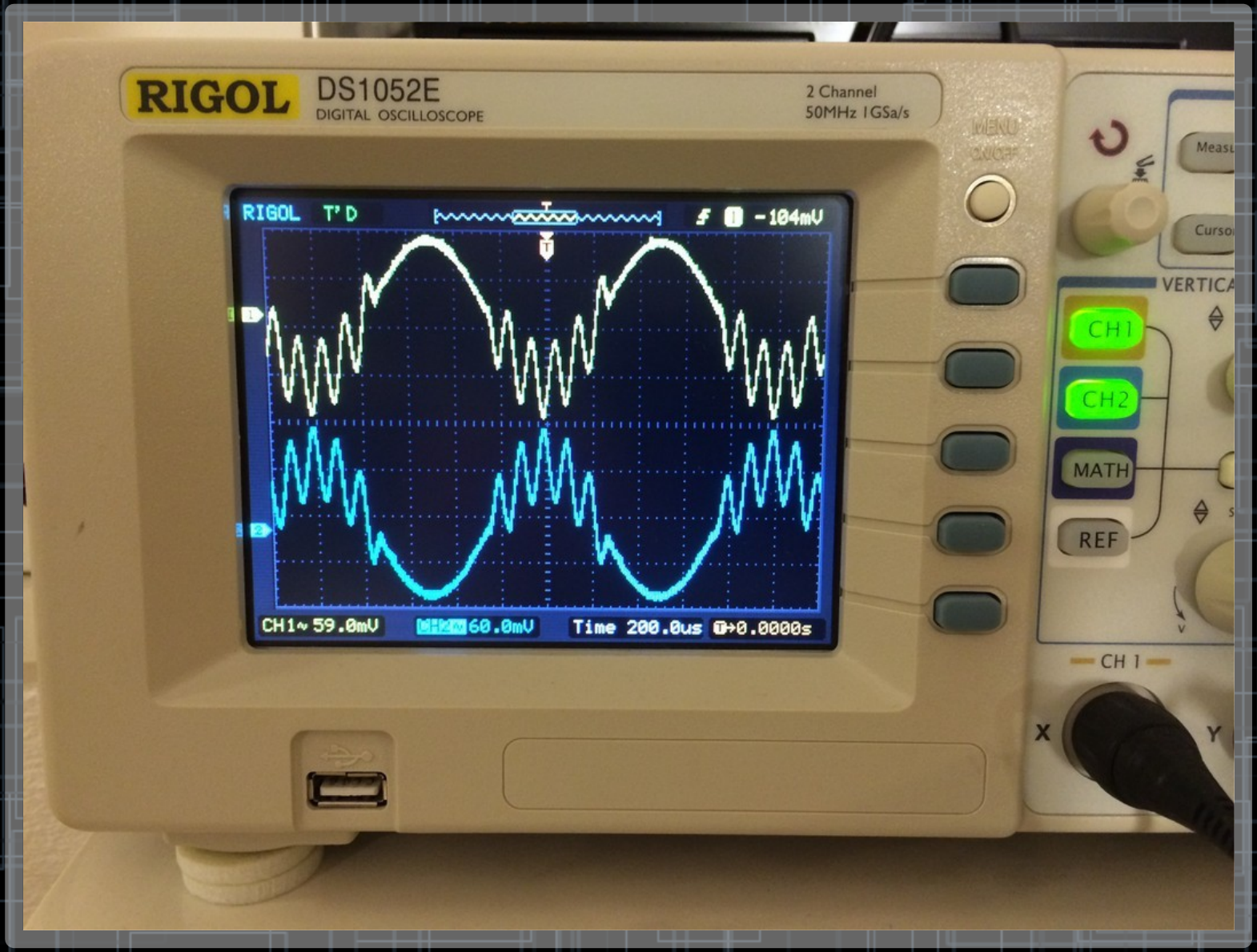
# Korn shell script “wave.sh”

- Why shell script ?
- Training for LPIC
- Train vi editor
- Getting familiar with shell scripting
- Why Korn shell ?
- Supports sine function
- Can deal with floating point numbers

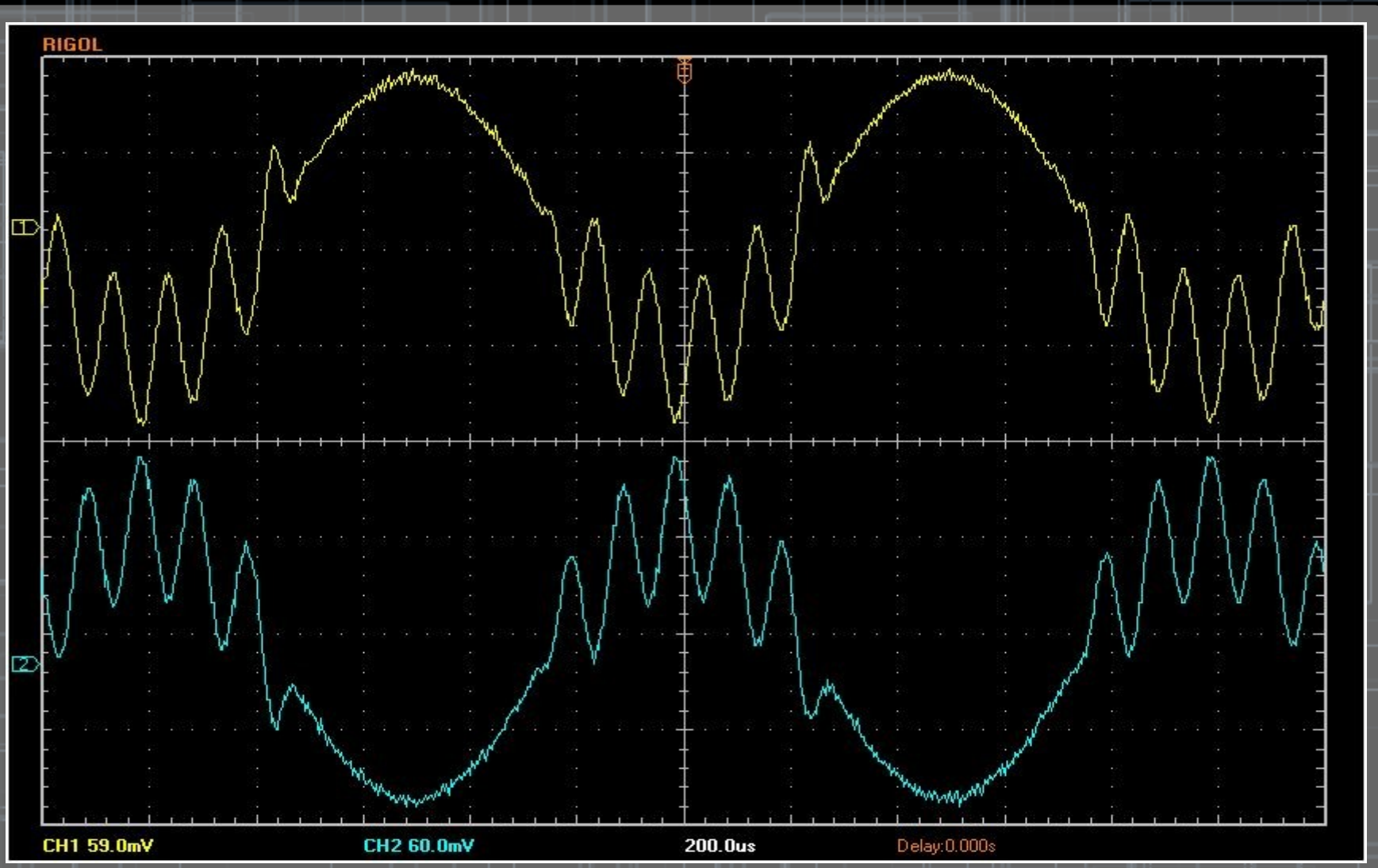
# Action of phase splitter circuit



# Action of phase splitter circuit



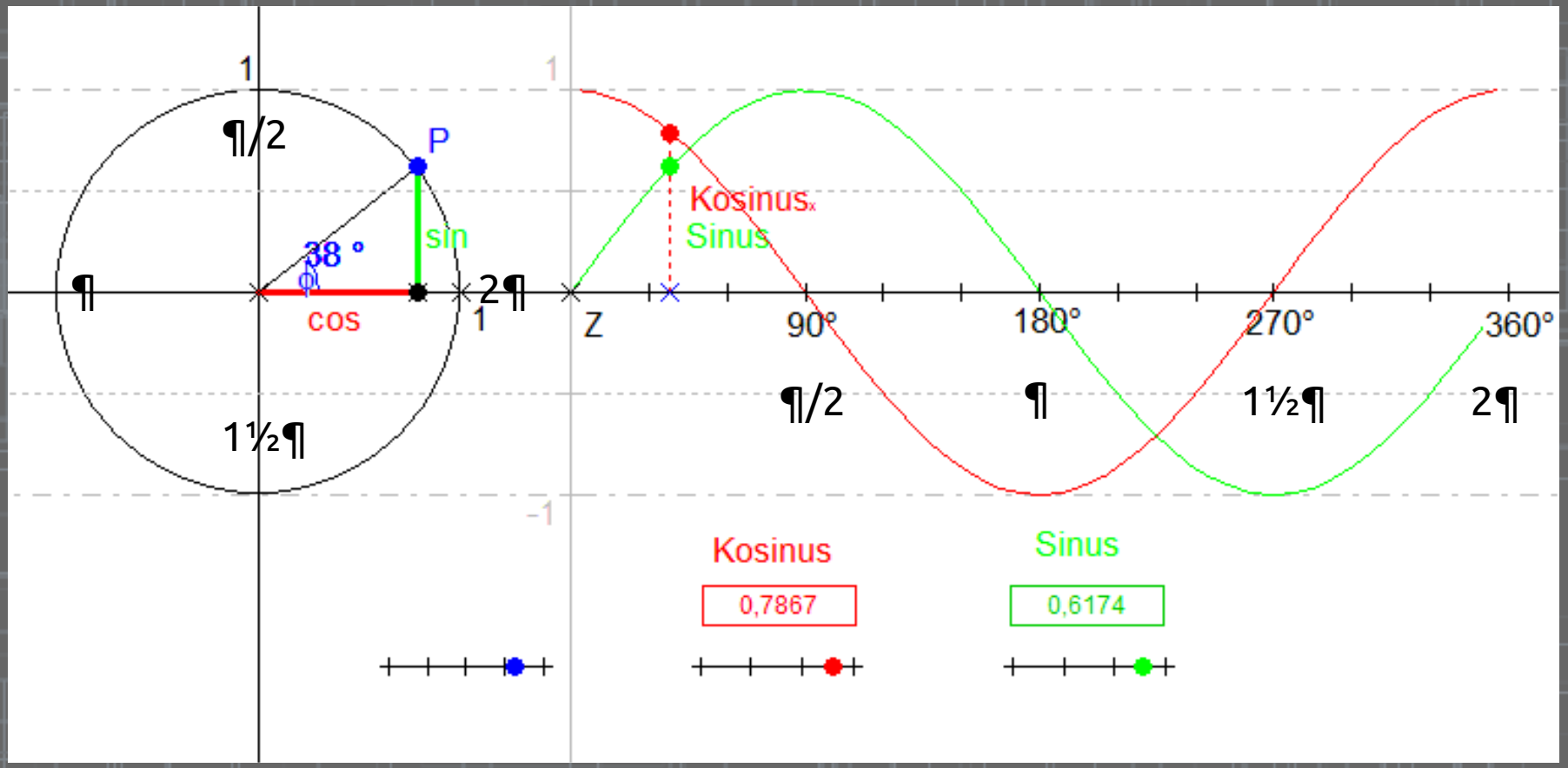
# Action of phase splitter circuit



# Outlook: further steps

- Measure rest of tube amp circuit
- Linux Alsa Sound System
- Porting Korn shell script do a different programming language
- Other PCM formats: CD, .au, .aiff
- Compressed audio formats: mp3
- Other RIFF file containers: .avi
- Torture sound card: throw nasty signals at it

# Geometrical definition of the sine function



# References

- RIFF: http://de.wikipedia.org/wiki/Resource_Interchange_File_Format
- WAVE: http://de.wikipedia.org/wiki/RIFF_WAVE
- WAVE: https://ccrma.stanford.edu/courses/422/projects/WaveFormat/

# References

- Endianess: http://de.wikipedia.org/wiki/Byte-Reihenfolge
- Endianess: http://www.ietf.org/rfc/ien/ien137.txt
- PCM: http://en.wikipedia.org/wiki/Pulse-code_modulation