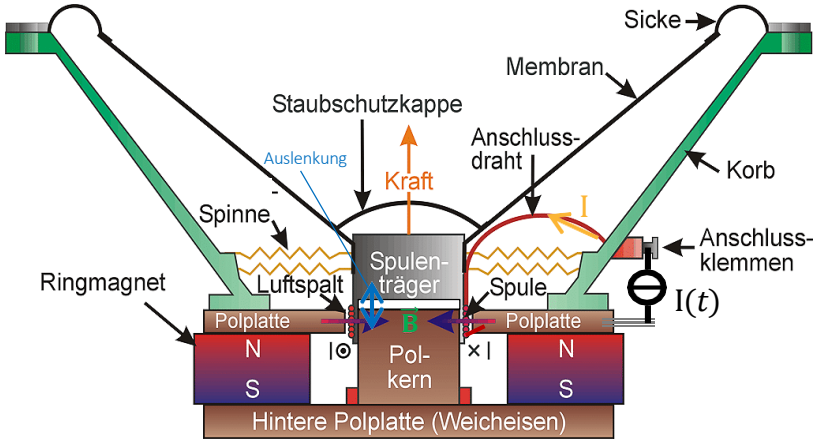
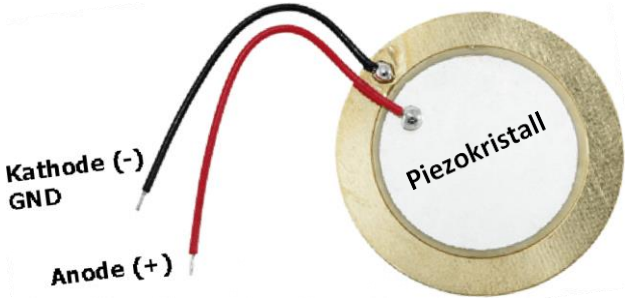
Digitalaudio

Schall erzeugen

- Schallwellen lassen sich mit dem Lautsprecher künstlich erzeugen. In einer klassischen Lautsprechereinheit wird durch einen Elektromagneten (Lautsprecherspule) eine Membran und somit die umliegende Luft zum schwingen angeregt.
- Je mehr Elektronen pro Sekunde durch den gewickelten Draht der Lautsprecherspule fließen, desto weiter wird die Membran ausgelenkt.
- Das an der Lautsprecherspule angelegte elektrische Signal (Wechselstrom $\mathrm{I}(t)$) wird als sogenanntes Audiosignal bezeichnet.



© Gottfried Behler



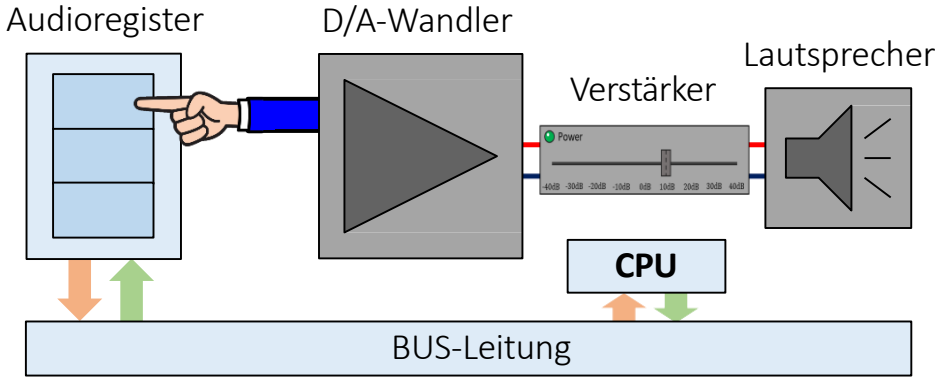
© Turanis

Audiokarte

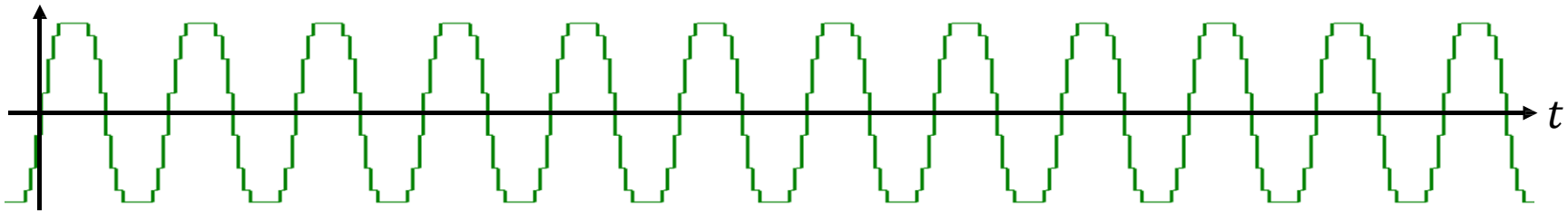
Eine Audiokarte, auch Tonkarte oder im englischen als Soundcard bezeichnet, erzeugt, mischt und zeichnet Tonsignale auf. Bei der Erzeugung eines Tonsignals schreibt (☞ Stream) die CPU ein Sample in das Audioregister. Ein D/A-Wandler erzeugt aus dem Sample ein analoges Signal, das ggf. verstärkt werden kann. Das Verfahren wird bei Aufnahmen genau umgekehrt angewendet.

Oak Mozart 16

© Appaloosa



Diskretisierte Audiosignale



Sinussignal

(Bittiefe: 4bit)

Aufbau einer WAVE-Datei

Offset:	00	01	02	03	04	05	06	07	ASCII
0x0000	52	49	46	46	32	F5	00	00	R I F F 2 õ
0x0008	57	41	56	45	66	6D	74	20	W A V E f m t
0x0010	10	00	00	00	01	00	01	00	
0x0018	11	2B	00	00	22	56	00	00	+ “ V
0x0020	02	00	10	00	64	61	74	61	d a t a
0x0028	EE	F4	00	00	76	00	E0	FB	î ô v fi û
0x0030	11	F8	22	F9	09	01	85	07	ø “ ù …
0x0038	3F	FE	50	F1	A9	F1	CO	00	? þ P ñ © ñ À

WAVE-Kopf

```
typedef struct WavHeader {
    char riff[4];               // "RIFF"
    unsigned int size;
    char wave[4];               // "WAVE"
    char fmt[4];                // "fmt "
    unsigned int fmt_size;
    unsigned short audio_format;
    unsigned short num_channels;
    unsigned int sample_rate;
    unsigned int byte_rate;
    unsigned short block_align;
    unsigned short bits_per_sample;
    char data[4];               // "data"
    unsigned int data_size; } wavHeader;
```

Aufgabe: WAVE-Dateien mischen I

```
int main(int argc, char* argv[]) {
    CHK0(argc, argv);
    WavHeader reference_header, header;
    int input_count = argc - 2;
    const char* output_path = argv[argc - 1];
    FILE* test, output;
    FILE** inputs = malloc(sizeof(FILE*) * input_count); CHK1(inputs);
    int16_t* samples = malloc(sizeof(int16_t) * input_count); CHK1(samples);

    // Check that output file does not exist
    test = fopen(output_path, "rb");
    if (test != NULL) {
        fclose(test)
        ERR0(output_path);
        return 1;
    }
    ...
```

Aufgabe: WAVE-Dateien mischen II

```
int main(int argc, char* argv[]) {
    ...
    // Open all input files and check headers
    for (int i = 0; i < input_count; ++i) {
        inputs[i] = fopen(argv[i + 1], "rb");
        // --> ERR1: Could not open input file
        if (!inputs[i]) {
            for (int j = 0; j < i; ++j) { fclose(inputs[j]); }
            free(inputs);
            ERR1(argv[i + 1]);
            return 1;
        }
        ...
        }
    }
    ...
```

Aufgabe: WAVE-Dateien mischen III

```
int main(int argc, char* argv[]) {
    ...
    // Open all input files and check headers
    for (int i = 0; i < input_count; ++i) {
        ...
        // Rewind + --> ERR2: Could not read WAV header
        rewind(inputs[i]);
        if (fread(&header, 1, HEADER_SIZE, inputs[i]) != HEADER_SIZE) {
            for (int j = 0; j <= i; ++j) { fclose(inputs[j]); }
            free(inputs);
            ERR2(argv[i + 1]);
            return 1;
        }
        ...
    }
    ...
```

Aufgabe: WAVE-Dateien mischen IV

```
int main(int argc, char* argv[]) {
    ...
    // Open all input files and check headers
    for (int i = 0; i < input_count; ++i) {
        ...
        // --> ERR3: Incompatible WAV file
        if (i == 0) {
            reference_header = header;
        } else if (!headers_are_compatible(&reference_header, &header)) {
            for (int j = 0; j <= i; ++j) { fclose(inputs[j]); }
            free(inputs);
            ERR3(argv[i + 1]);
            return 1;
        }
    }
    ...
```

Aufgabe: WAVE-Dateien mischen V

```
int main(int argc, char* argv[]) {
    ...
    // Create output file
    output = fopen(output_path, "wb");
    if (!output) {
        for (int i = 0; i < input_count; ++i) { fclose(inputs[i]); }
        free(inputs);
        ERR4(output_path);
        return 1;
    }

    // Write placeholder header
    fwrite(&reference_header, 1, HEADER_SIZE, output);
    ...
}
```

Aufgabe: WAVE-Dateien mischen VI

```
int main(int argc, char* argv[]) {
    ...
    // Calculate channels and sample count
    int sample_count = reference_header.data_size
                     / (reference_header.bits_per_sample / 8);
    int channels = reference_header.num_channels;
    int total_samples = sample_count / channels;
    int16_t sample = 0;
    int16_t mixed = 0;
    int sum = 0;
    int count = 0;
    long int final_pos = 0;
    ...
}
```

Aufgabe: WAVE-Dateien mischen VII

```
int main(int argc, char* argv[]) {
    ...
    // Mix samples
    for (int i = 0; i < total_samples; ++i) {
        for (int ch = 0; ch < channels; ++ch) {
            sum = 0; count = 0;

            for (int j = 0; j < input_count; ++j) {
                if (fread(&sample, sizeof(int16_t), 1, inputs[j]) == 1) {
                    sum += sample; count++; }
            }

            mixed = count ? (sum / count) : 0;
            fwrite(&mixed, sizeof(int16_t), 1, output);
        }
    }
    ...
```

Aufgabe: WAVE-Dateien mischen VIII

```
int main(int argc, char* argv[]) {
    ...
    // Update header with correct data size
    final_pos = ftell(output);
    reference_header.data_size = final_pos - HEADER_SIZE;
    reference_header.size = final_pos - 8;

    rewind(output);
    fwrite(&reference_header, 1, HEADER_SIZE, output);

    for (int i = 0; i < input_count; ++i) { fclose(inputs[i]); } // Cleanup
    fclose(output);
    free(inputs);
    free(samples);
    MIXING_COMPLETED(output_path);
    return EXIT_SUCCESS;
}
```