Pliki cyfrowe audio wykorzystywane są przede wszystkim do przechowywania strumieni danych z przetworników analogowo-cyfrowych A/C lub wypełniania strumieni do przetworników cyfrowo-analogowych C/A.

Konwertery A/C C/A kodeków AC97 i kart dźwiękowych mogą pracować w trybie równoległym (strumienie danych są przetwarzane jednocześnie w obu kierunkach FullDUPLEX).
Tory sygnałowe kodeków i kart dźwiękowych (część analogowa torów oraz konwertery A/C C/A są zaprojektowane na pasmo częstotliwości akustycznych (50 - 22000)Hz.

Podstawowe parametry konwerterów A/C C/A:

1. Długość słowa przetwarzanego (b_r) – 8 bitów / 16 bitów / 20 bitów / 24 bity / zmienna dł.,

2. Częstotliwość próbkowania (zmienna, ale ograniczona do zbioru najczęściej stosowanych):

- 44 100 Hz / 22 050 Hz / 11 025 Hz
- 48 000 Hz / 32 000 Hz / 24 000 Hz / 16 000 Hz / 12 000 Hz / 8 000 Hz
- **96 000 Hz**(maks.) **-** Sound Blaster z procsorem DSP EMU10K2 oraz nowsze kodeki AC97

3. Kodowanie próbek sygnału (wartości chwilowych) do systemu binarnego najczęściej wykonywane jest metodą modulacji PCM lub ADPCM / MS ADPCM (ADPCM - *Adaptive Differential Pulse Code Modulation*.

4. Zakres dynamiki sygnału przetwarzanego w torach sygnałowych kart i kodeków od 50 dB do 100 dB.

$$X_{br} = 20\log_{10}(b_r)$$

$20\log_{10}(2^8) = 20\log_{10}(256) = 48\text{dB}$ – dynamika toru A/C z przetwornikiem 8-bitowym
$20\log_{10}(2^{12}) = 20\log_{10}(4096) = 72\text{dB}$ – dynamika toru A/C z przetwornikiem 12-bitowym
$20\log_{10}(2^{16}) = 20\log_{10}(65536) = 96\text{dB}$ – dynamika toru A/C z przetwornikiem 16-bitowym

Zapis/odczyt i organizacja danych (próbek sygnału) w plikach audio *.WAV zgodnie ze standardem RIFF (*Resource Interchange File Format*).

Standard RIFF wykorzystuje 4-elementowe (4 x 1bajt) pola znaczników do identyfikacji bloków pliku. Pola znaczników wypełnianie są znakami ASCII. 4-bajtowe znaczniki tworzą drzewa kodu **FOURCC**. Funkcje i operatory dla znaczników FourCC zdefiniowane są pliku nagłówkowym w **mmsystem.h**

Blok danych (*chunk*) w pliku typu RIFF wygląda następująco:

- rozpoczyna się identyfikatorem znacznika "RIFF" (4 bajty FourCC),
- pole określające długość bloku danych objętych znacznikiem RIFF (4 bajty DWORD),
- identyfikator typu znacznika (4 bajty FourCC),
- pole określające położenie bloku danych względem początku pliku (4 bajty DWORD),
- pole flag (4 bajty DWORD),

Struktura znaczników w kodzie FOURCC:

Znaczniki **RIFF** i **LIST** mają najwyższy status (*parent chunk*) w hierarchii znaczników i mogą zawierać znaczniki niższego poziomu (*subchunks*).
Przemieszczanie po drzewie znaczników umożliwiają funkcje mmioAscend i mmioDescend.

PRZYKŁADY predefiniowanych znaczników:

```
#define RIFFINFO_IARL      mmioFOURCC ('I', 'A', 'R', 'L')     Archival location
#define RIFFINFO_IART      mmioFOURCC ('I', 'A', 'R', 'T')     Artist
#define RIFFINFO_ICMS      mmioFOURCC ('I', 'C', 'M', 'S')     Commissioned
#define RIFFINFO_ICMT      mmioFOURCC ('I', 'C', 'M', 'T')     Comments
#define RIFFINFO_ICOP      mmioFOURCC ('I', 'C', 'O', 'P')     Copyright
#define RIFFINFO_ICRD      mmioFOURCC ('I', 'C', 'R', 'D')     Creation date of subject
#define RIFFINFO_ICRP      mmioFOURCC ('I', 'C', 'R', 'P')     Cropped
#define RIFFINFO_IDIM      mmioFOURCC ('I', 'D', 'I', 'M')     Dimensions
#define RIFFINFO_IDPI      mmioFOURCC ('I', 'D', 'P', 'I')     Dots per inch
#define RIFFINFO_IENG      mmioFOURCC ('I', 'E', 'N', 'G')     Engineer
#define RIFFINFO_IKEY      mmioFOURCC ('I', 'K', 'E', 'Y')     Keywords
#define RIFFINFO_ILGT      mmioFOURCC ('I', 'L', 'G', 'T')     Lightness settings
#define RIFFINFO_IMED      mmioFOURCC ('I', 'M', 'E', 'D')     Medium  */
#define RIFFINFO_INAM      mmioFOURCC ('I', 'N', 'A', 'M')     Name of subject
```

Wszystkie deklaracje funkcji, zmiennych itd. Znajdują się w plikach nagłówkowych:

mmsystem.h, mmreg.h, msacm.h, mciavi.h, digitalv.h, vcr.h, vfw.h

```
#define WAVE_INVALIDFORMAT     0x00000000     nieznany format */
#define WAVE_FORMAT_1M08       0x00000001      11.025 kHz, Mono,   8-bitów
#define WAVE_FORMAT_1S08       0x00000002      11.025 kHz, Stereo, 8-bitów
#define WAVE_FORMAT_1M16       0x00000004      11.025 kHz, Mono,   16-bitów
#define WAVE_FORMAT_1S16       0x00000008      11.025 kHz, Stereo, 16-bitów
#define WAVE_FORMAT_2M08       0x00000010      22.050  kHz, Mono,   8-bitów
#define WAVE_FORMAT_2S08       0x00000020      22.050  kHz, Stereo, 8-bitów
#define WAVE_FORMAT_2M16       0x00000040      22.050  kHz, Mono,   16-bitów
#define WAVE_FORMAT_2S16       0x00000080      22.050  kHz, Stereo, 16-bitów
#define WAVE_FORMAT_4M08       0x00000100      44.100   kHz, Mono,   8-bitów
#define WAVE_FORMAT_4S08       0x00000200      44.100   kHz, Stereo, 8-bitów
#define WAVE_FORMAT_4M16       0x00000400      44.100   kHz, Mono,   16-bitów
#define WAVE_FORMAT_4S16       0x00000800      44.100   kHz, Stereo, 16-bitów
```

Organizacja danych w plikach *.WAV

Typowa struktura pliku składa się z:

NAGŁÓWEK	*'R','I','F','F'* + Rozmiar pliku (DWORD)
	'W','A','V','E'
	'F','M','T',' ' + rozmiar struktury formatu (DWORD)
	PCMWAVEFORMAT / WAVEFORMAT / WAVEFORMATEX (**struct**) itd.
DANE	*'D','A','T','A'* + Rozmiar bloku danych (DWORD) próbki sygnału L, P, L, P, L .....

PRZYKŁAD STRUKTURY BLOKU NAGŁÓWKOWEGO W PLIKU
(PCM – WAVEFORM Audio) *.WAV

```
struct TWavHeader {
                char riff[4];
                unsigned int BlockSize;
                char wave[4];
                char fmt[4];
                unsigned int FormatLength ;
                PCMWAVEFORMAT  pcmWaveFormat;
                char data[4];
                unsigned int DataSize;
      } WavHeader ={"RIFF", 0, "WAVE", "fmt\0", 0,{ 0²,0²,0⁴,0⁴,0²,0²},"data", 0};
```

```
typedef struct  waveformat_tag {
                              unsigned short wFormatTag;
                              unsigned short nChannels;
                              unsigned int   SamplesPerSec;
                              unsigned int   nAvgBytesPerSec;
                              unsigned short nBlockAlign;
                            } WAVEFORMAT, *PWAVEFORMAT;
(FormatLength = 14)
```

```
typedef struct pcmwaveformat_tag {
                                WAVEFORMAT     wf;
                                unsigned short  wBitsPerSample;
                          } PCMWAVEFORMAT, *PPCMWAVEFORMAT;

(FormatLength = 16)
```

```
typedef struct tWAVEFORMATEX
                              {
                              unsigned short   wFormatTag;
                              unsigned short   nChannels;
                              unsigned int     nSamplesPerSec;
                              unsigned int     nAvgBytesPerSec;
                              unsigned short   nBlockAlign;
                              unsigned short   wBitsPerSample;
                              unsigned short   cbSize;
                } WAVEFORMATEX, *PWAVEFORMATEX;
(FormatLength = 18)
```

POZOSTAŁE definicje struktur (ponad 120)

FILE *PLIKWAV;

```
1. PLIKWAV = fopen( "DEMO.WAV", "r+b" );
```

// PWAVEFORMAT – rozmiar nagłówka + 8 bajtów ("data"+ Rozmiar bloku danych (DWORD))

```
2. fread(PWAVEFORMAT, sizeof(PWAVEFORMAT ), 1, PLIKWAV );

3. fseek(PWAVEFORMAT, sizeof(PWAVEFORMAT), SEEK_SET);

   unsigned char  TABLPROBEK[Rozmiar bloku danych];
```

4. lub

```
   short int TABLPROBEK[Rozmiar bloku danych];
```

//Tablica na próbki (8 lub 16 bitów) mono/stereo

```
  fread(TABLPROBEK, sizeof(unsigned char)*Rozmiar bloku danych,1, PLIKWAV);
```

5. lub

```
  fread(TABLPROBEK, sizeof(short int)* Rozmiar bloku danych, 1, PLIKWAV);

...
```

Zakres zmienności wartości próbek reprezentowanych jako 8-bitowe i 16-bitowe

format	Maximum	Minimum	Średnia
8-bitów PCM	255	0	128
16-bitów PCM	32,767	- 32,768	0

OBSŁUGA INTERFEJSU AUDIO (WAVEFORM AUDIO INTERFACE)

Funkcje dostępu do urządzeń audio dostępne są z poziomu:

1. funkcje interfejsu API wysokiego poziomu do odtwarzania cyfrowych plików audio,

2. Funkcje interfejsu MCI (Media Control Interface) do sterowania urządzeniami multimedialnymi w PC (MIDI, CD, DV, DVD, inne) – zapewnia izolację warstwy programowej od sprzętu,

3. Funkcje Waveform Audio API (WA - API), dostęp do zasobów sprzętowych urządzeń Audio zainstalowanych w systemie operacyjnym

4. Biblioteki DirectX funkcji API:
   - DirectSound,
   - DirectMusic,
   - DirectX Audio

Waveform Audio API

Funkcje poprzedzone przedrostkiem wave na przykład: waveIn, waveOut.

Funkcje identyfikujące zasoby systemowe:

```
UINT waveInGetNumDevs(void);
```

Funkcja zwraca liczbę zainstalowanych i dostępnych przez interfejs urządzeń wyposażonych w tory przetwarzania A/C (wejścia audio); 0 – brak ; wart. niezerowa – liczba dostępnych urzadzeń

```
UINT waveOutGetNumDevs(void);
```

Funkcja zwraca liczbę zainstalowanych i dostępnych przez interfejs urządzeń wyposażonych w tory przetwarzania C/A (wyjścia audio);Zwracana wartość j.w.

```
MMRESULT waveInGetDevCaps(UINT uDeviceID, LPWAVEINCAPS pwic, UINT sbwic);
```

Funkcja zwraca do struktury `pwic` o rozmiarze `sbwic` bajtów, informację o producencie urządzenia, nazwie, sterownikach, obsługiwanych formatach i liczbie wejść audio urządzenia identyfikowanego w systemie zmienną `uDeviceID`.

Struktura `WAVEINCAPS (pwic)`

```
typedef struct {
                WORD wMid;
                WORD wPid;
                MMVERSION vDriverVersion;
                CHAR szPname[MAXPNAMELEN];
                DWORD dwFormats;
                WORD wChannels;
                WORD wReserved1;} WAVEINCAPS; *pwic;
```

```
MMRESULT waveOutGetDevCaps(UINT uDeviceID, LPWAVEINCAPS pwoc, UINT sbwoc);
```

Funkcja zwraca do struktury `pwoc` o rozmiarze `sbwoc` bajtów, informację o producencie urządzenia, nazwie, sterownikach, obsługiwanych formatach i liczbie wejść audio urządzenia identyfikowanego w systemie zmienną `uDeviceID`.

Struktura `WAVEOUTCAPS (pwoc)` zawiera te same pola co struktura WAVEINCAPS oraz dodatkowe pole dwSupport identyfikujące o opcjonalnych możliwościach wyjść audio (niezależna reg. głośności wyjść, rozdz. regulacji, regulacja szybkości odtwarzania i częstotliwości próbkowania przy rekonstrukcji sygnału).

```
typedef struct WAVEOUTCAPS_tag {
                .
                .
                pola jak w WAVEINCAPS
                .
                .
                DWORD dwSupport;} WAVEOUTCAPS;*pwoc;
```

Funkcje dostępu do zasobów systemowych:

```
MMRESULT waveInOpen(LPHWAVEIN phwi, UINT uDeviceID, LPWAVEFORMATEX pwfx,
                    DWORD dwCallback, DWORD dwCallbackInstance, DWORD fdwOpen);
```

Funkcja otwierająca strumień wejściowy audio urządzenia `uDeviceID (lub WAVE_MAPPER)` i identyfikowany zmienną `phwi` . Format danych przekazywanych strumieniem określa struktura `pwfx`. Zmienne `dwCallback` oraz `dwCallbackInstance` określają adresy funkcji oraz instancje odwołań do których będą przekazywane komunikaty odwołań. Zmienna `fdwOpen` określa flagi dostępu do urządzenia i komunikacji odwołań (`CALLBACK_EVENT, CALLBACK_FUNCTION, CALLBACK_WINDOW` i inne).

```
MMRESULT waveOutOpen(LPHWAVEOUT phwo, UINT uDeviceID, LPWAVEFORMATEX pwfx,
                     DWORD dwCallback, DWORD dwCallbackInstance, DWORD fdwOpen);
```

Funkcja otwierająca strumień wyjściowy audio (interpretacja parametrów funkcji jak przy `waveInOpen`).

Funkcje buforowania strumieni wejścia/wyjścia audio:

```
MMRESULT waveInPrepareHeader (HWAVEIN hwi,LPWAVEHDR pwh,UINT cbwh);
```

Funkcja przygotowująca bufor dla danych przekierowanych strumieniem o identyfikatorze `hwi` z przetwornika A/C. Struktura `WAVEHDR` o rozmiarze `cbwh` określa parametry bufora (adres, rozmiar poziom zapełnienia, aktualny status bufora, liczbę powtórzeń i wskaźnik na kolejny bufor). Bufor należy przygotować przed wowłaniem funkcji `waveInPrepareHeadera`.

```
typedef struct wavehdr_tag {
                                LPSTR       lpData;
                                DWORD       dwBufferLength;
                                DWORD       dwBytesRecorded;
                                DWORD       dwUser;
                                DWORD       dwFlags;
                                DWORD       dwLoops;
                                struct wavehdr_tag FAR *lpNext;
                                DWORD       reserved;
                            } WAVEHDR, *PWAVEHDR, *pwh;
```

```
MMRESULT waveInAddBuffer(HWAVEIN hwi,LPWAVEHDR pwh,UINT cbwh);
```

Funkcja wykonująca przekazanie bufora do strumienia `hwi` danych przychodzących z przetworników A/C.

```
MMRESULT waveOutPrepareHeader (HWAVEOUT hwo,LPWAVEHDR pwh,UINT cbwh);
```

Funkcja bufora dla strumienia wyjściowego (interpretacja parametrów funkcji jak przy `waveInAddBuffer`).Nie ma funkcji definicji /`waveOutAddBuffer/`!

Funkcje sterowania strumieniami:

```
MMRESULT waveInStart(HWAVEIN hwi);
```

Funkcja wyzwalająca przetwornik A/C do wypełniania buforowanego strumienia hwi próbkami sygnału analogowego podanego na wejścia audio. Funkcja wykonuje wyzwalanie bez względu czy jest przekazany bufor czy nie. Brak informacji o buforowaniu powoduje że próbki nie są nigdzie zapisywane i nie ma możliwości ich odtworzenia. Zapisywanie do buforów odbywa się zawsze od pozycji początkowej i zwracana jest informacja tylko w przypadku całkowitego wypełnienia bufora.

```
MMRESULT waveInReset(HWAVEIN hwi);
```

Funkcja wyzwalająca sygnał zatrzymania przetwornika A/C i ustawiająca wskaźnik buforowania strumienia `hwi` na początek bufora bez względu na jego aktualne położenie. (dotyczy wszystkich zdefiniowanych buforów).

```
MMRESULT waveInStop(HWAVEIN hwi);
```

Funkcja wyzwalająca sygnał zatrzymania przetwornika A/C i ustawiająca wskaźnik buforowania strumienia `hwi` w miejscu jego aktualnego położenia. (dotyczy tylko wypełnianego bufora).

```
MMRESULT waveOutWrite(HWAVEOUT hwo);
```

Funkcja wyzwalająca wypełnianie buforowanego strumienia hwo przetwornika C/A próbkami sygnału dyskretnego podanego do rekonstrukcji na wyjście audio. Funkcja wykonuje wyzwalanie bez względu czy jest przygotowany bufor czy nie. Brak informacji o przygotowanym funkcją `waveOutPrepareHeader` buforowaniu powoduje że wywoływana jest funkcja `waveOutPause`. Odczyt z buforów odbywa się zawsze od pozycji początkowej i zwracana jest informacja tylko w przypadku zakończenia odczytu z bufora.

```
MMRESULT waveOutRestart((HWAVEOUT hwo);
```

Funkcja wznawiająca odczyt z buforów próbek do przetwornika C/A wywoływana tylko po zatrzymaniu strumieniowania funkcją `waveOutPause`.

```
MMRESULT waveOutReset(HWAVEOUT hwo);
```

Funkcja wyzwalająca sygnał zatrzymania przetwornika C/A - analogicznie jak w przypadku funkcji `waveInReset`.

Funkcje zwalniania zasobów buforowania strumieni:

```
MMRESULT waveInUnprepareHeader (HWAVEIN hwi,LPWAVEHDR pwh,UINT cbwh);
```

Funkcja zwalniająca bufor dla danych przekierowanych strumieniem o identyfikatorze `hwi` z przetwornika A/C. Struktura `WAVEHDR` o rozmiarze `cbwh` określa parametry bufora (adres, rozmiar poziom zapełnienia, aktualny status bufora, liczbę powtórzeń i wskaźnik na kolejny bufor). Bufor można zwolnić funkcją `waveInUnprepareHeader` po zakończeniu przetwarzania i uzyskanym potwierdzeniu tego informacją w strukturze `WAVEHDR`.

```
MMRESULT waveOutUnprepareHeader (HWAVEOUT hwo,LPWAVEHDR pwh,UINT cbwh);
```

Funkcja zwalniająca bufor danych dla przetwornika C/A - analogicznie jak w przypadku funkcji `waveInUnprepareHeader`.

Funkcje zamykania strumieni:

```
MMRESULT waveInClose(HWAVEIN hwi);
```

Funkcja zamykająca dostęp do strumienia związanego z przetwornikiem A/C i zwalniająca identyfikator uchwytu `hwi` do strumienia WAVEIN. Wykonanie funkcji wymaga zatrzymania wszystkich przetworników A/C i ustawienia wskaźników buforowania w pozycji początkowej (0). Można wykonać to funkcją `waveInReset`.

```
MMRESULT waveOutClose(HWAVEOut hwo);
```

Funkcja zamykająca dostęp do strumienia związanego z przetwornikiem C/A - analogicznie jak w przypadku funkcji `waveInClose`.

Funkcje nastaw i regulacji wzmocnienia:

```
MMRESULT waveOutSetVolume(HWAVEOut hwo,DWORD dwVolume);
```

Funkcja regulacji głośności wyjścia audio. Charakterystyka regulacji odbywa się logarytmicznie i zależy od rozdzielczości regulacyjnej (4-16 bitów). Dla urządzeń z regulacją o rozdzielczości mniejszej niż 16-bitów zmiany nie są wykonywane i sygnalizowane.

```
MMRESULT waveOutGetVolume(HWAVEOut hwo,DWORD pdwVolume);
```

Funkcja odczytu nastaw regulacji głośności wyjścia audio. Zwraca wartość nastaw jak dla regulacji 16-bitowej.

```
WaveInSetProperty / WaveInGetProperty / WaveOutSetProperty / WaveOutGetProperty
```

Funkcje zmiany ustawień parametrów urządzeń audio priorytety zmiany nastaw regulacyjnych, parametry klasyfikacji strumieni przekazujących próbki (domyślne, przekaz, audio, efekty itd.)