ENEE408G Multimedia Signal Processing
Design Project on Digital Audio Processing

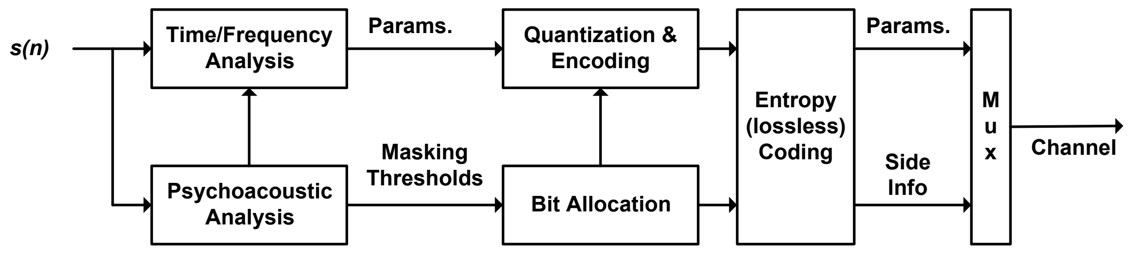
The Goals

1. Learn the fundamentals of perceptual coding of audio and intellectual rights protection from multimedia.
2. Design a digital audio watermarking system in time and frequency domain.
3. Explore the synthetic audio: MIDI and MPEG4 Structured Audio.

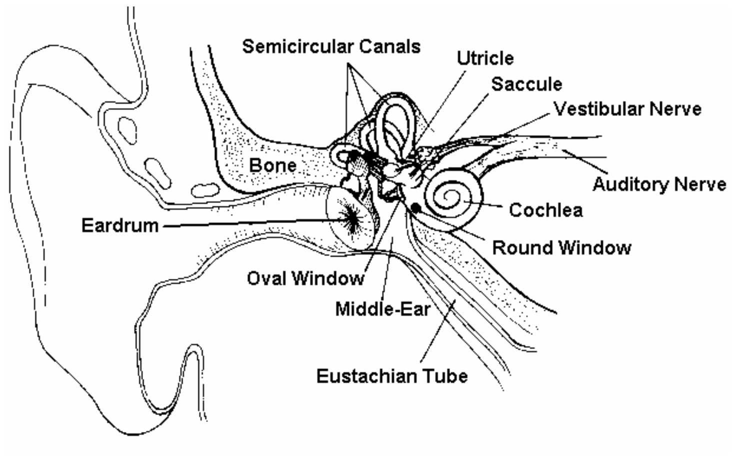
Note: The symbol ✐ means to put your discussion, flowchart, block diagram, or plots in your report. The symbol ♫ indicates that you should put the obtained multimedia data in your report. The symbol 🖫 means to put your source codes (Matlab, Basic, or C/C++) in your report.

Part I. Perceptual Coding and MP3

In modern audio coding algorithms, four key technologies play important roles: perceptual coding, frequency-domain coding, window switching, and dynamic bit allocation. The figure below shows a generic block diagram for modern audio encoders. In this part of the design project, we investigate the fundamentals of perceptual coding in the MP3 technology.



1. Psychoacoustic Models and Perceptual Coding



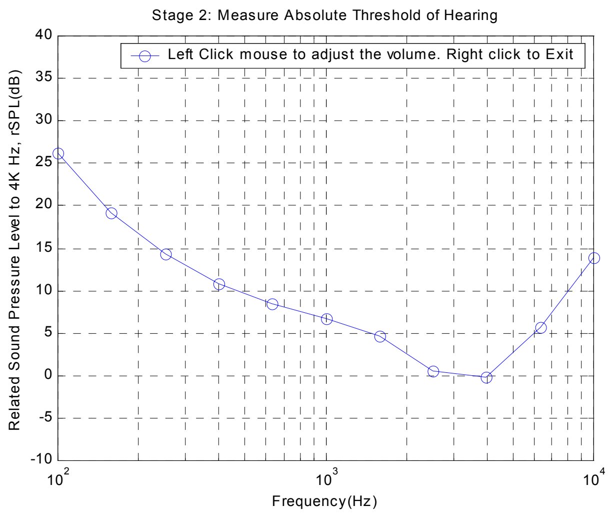
The figure[1] above shows the anatomy of human ear. Many researchers in the field of psychoacoustics exploit the "irrelevant" signal information that is not detectable even by a well-trained or sensitive listener. These studies lead to five psychoacoustic principles.

a. Absolute Threshold of Hearing: This threshold represents the minimal amount of energy if a listener is able to detect a pure tone in a noiseless environment. If a given tone is too weak, we cannot hear it, so we do not have to encode it.
b. Critical Bands Frequency Analysis: Cochlea can be modeled as a non-uniform filter bank that consists of 25 highly overlapping bandpass filters. Critical bands are the passbands of those filters.
c. Simultaneous Masking: In each critical band, one sound (the maskee) is rendered inaudible due to the presence of another sound (the masker). We can identify a masker and do not encode the inaudible, masked tones.
d. Spread of Masking: Masking in a critical band can be spread to its neighboring bands.
e. Non-Simultaneous Masking: Masking can be done in time domain, too.

For perceptual coding, the encoder generates a global masker according to the above principles and provides parameters for further processing. In this part, we investigate the absolute threshold of hearing and simultaneous marking principles.

(a) Absolute Threshold of Hearing:

In this section, we use *PM_Abs_Thre_Hearing.m* to explore the absolute threshold of hearing. The goal of this experiment is to find out a volume threshold that is just auditable at a specific frequency. In other words, given a tone with the same frequency, if it has a slightly lower volume than this threshold, it becomes inaudible.



[1] From http://www.vestibular.org/gallery.html

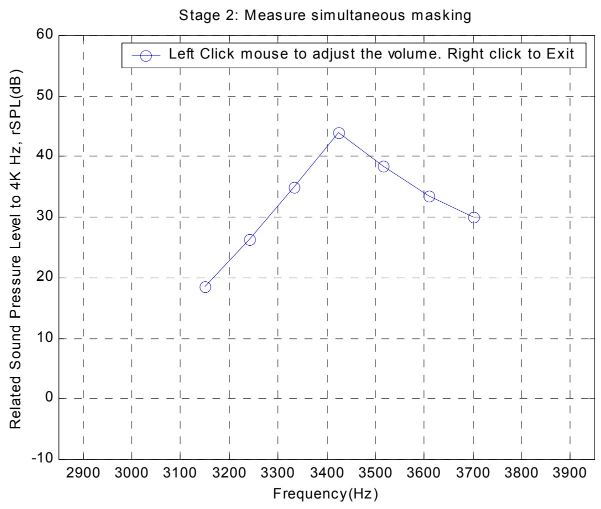
First, we calibrate the minimal audible volume for 4KHz[2]. After setting this volume, you will see the figure above. Each circle represents a frequency component and we can increase/decrease its volume by simply left clicking mouse in the upper/lower side of this circle. You can right click mouse to exit this program.

Find out the thresholds for your ear at the 11 frequencies shown in the figure. Copy your result figure using *Edit* → *Copy Figure* from the menu bar of the figure window and paste it to your report ♬.

(b) Simultaneous Masking:

In this section, we use *PM_Simu_Masking.m* to explore the simultaneous masking, which means that a tone could become inaudible if there is a simultaneous tone with higher volume at a neighboring frequency. For each critical band, we can fix the volume of the central frequency. By adding a neighboring tone with different amplitude, we can find out a threshold that this neighboring tone is inaudible if the amplitude is lower than the threshold, but audible if the amplitude is higher than the threshold.

In this experiment, we also need to calibrate the minimal audible volume for 4KHz. After calibration, you will see another figure that is illustrated below. There are seven frequencies indicated by circles. The middle one represents the central frequency of a specific critical band. When we click the other neighboring frequency, this program will generate an audio signal consisting of the central frequency and the selected frequency. We can increase/ decrease its volume by simply left clicking mouse in the upper/lower side of this circle. After finding out the thresholds of those six frequencies (except the central frequency), right click mouse to exit this program.



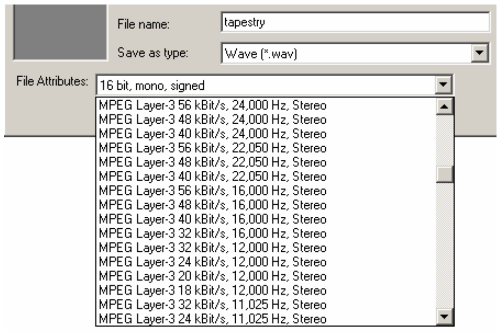
[2] Since 4KHz is the most sensitive frequency for human ear, to obtain the whole dynamic range of volume, we first calibrate the minimal audible volume at 4KHz.

Select two different critical bands[3] and find out the simultaneous masking of your ear. Copy these two figures using *Edit* → *Copy Figure* from the menu bar of this figure window and paste them in your report ♬ . For each critical band, compare the mask of higher neighboring frequency with the lower one ✐.

2. Audio Extraction and MP3

MP3 (MPEG1 Audio Layer 3) is an audio coding/compression standard that uses perceptual coding technologies. In this part, we use *GoldWave* to extract a piece of music and compress it in MP3 format. We will observe how psychoacoustic model works by comparing the frequency spectrum of the raw signal and MP3 file:

(a) CD Audio Extraction: Click *Tools*→*CD audio extraction* from *GoldWave*'s menu bar. An audio extraction window will pop up. Use this tool to extract about ten seconds' music[4] from your favorite music CD and save it as a *16-bit stereo signed WAV* file. Name it as *original.wav*. ♬



File name: tapestry
Save as type: Wave (*.wav)
File Attributes: 16 bit, mono, signed

```
MPEG Layer-3 56 kBit/s, 24,000 Hz, Stereo
MPEG Layer-3 48 kBit/s, 24,000 Hz, Stereo
MPEG Layer-3 40 kBit/s, 24,000 Hz, Stereo
MPEG Layer-3 56 kBit/s, 22,050 Hz, Stereo
MPEG Layer-3 48 kBit/s, 22,050 Hz, Stereo
MPEG Layer-3 40 kBit/s, 22,050 Hz, Stereo
MPEG Layer-3 56 kBit/s, 16,000 Hz, Stereo
MPEG Layer-3 48 kBit/s, 16,000 Hz, Stereo
MPEG Layer-3 40 kBit/s, 16,000 Hz, Stereo
MPEG Layer-3 32 kBit/s, 16,000 Hz, Stereo
MPEG Layer-3 32 kBit/s, 12,000 Hz, Stereo
MPEG Layer-3 24 kBit/s, 12,000 Hz, Stereo
MPEG Layer-3 20 kBit/s, 12,000 Hz, Stereo
MPEG Layer-3 18 kBit/s, 12,000 Hz, Stereo
MPEG Layer-3 32 kBit/s, 11,025 Hz, Stereo
MPEG Layer-3 24 kBit/s, 11,025 Hz, Stereo
```

(b) Downsampling: To compare with MP3, we need downsample this WAV file. Click *Effect*→ *Resample* and choose 16000Hz. Save this file as *downsample.wav*. ♬

(c) Generate MP3 file by *File* → *Save As*. Choose *Save as type* as *Wave Audio*. Adjust the parameters in the *File Attributes* as *MPEG Layer 3, 32kbps, 16000Hz, stereo*. Name this new file as *wav2mp3.wav*. ♬

(d) Convert MP3 to WAVE: Reload *wav2mp3* file and save it as *16-bit stereo signed* WAV file. Name it as *reconstructed_wav.wav*. ♬

(e) Compare Spectrum: Use *Compare_Spectrum.m*[5] to plot spectrum for *downsample.wav* and *reconstructed_wav.wav* ✐. Compare the difference between

[3] *[LinX,volumnY]=PM_Simu_Masking(ith_CB)* ; you can select critical band by specifying *ith_CB*
[4] You can use your own favorite Compact Disk.
[5] *Compare_Spectrum(Original_Filename,Reconstructed_Filename,NFFTorder,Len_order,shift)*
% Original_Filename : file name of the original signal.
% Reconstructed_Filename : file name of the reconstructed signal
% NFFTorder: Number of FFT points, NFFT=power(2,NFFTorder), e.g. NFFT order = 9 for
% 512-point FFT;

the two audio chips and discuss on which frequency range the psychoacoustic model has a significant impact ✎.

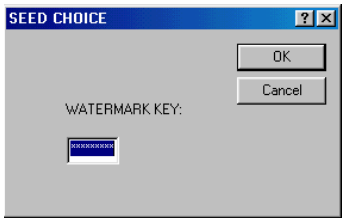
(f) Repeat (b)~(e) but change the MP3 to MPEG Layer 3, 56kbps, 24000Hz, stereo ♬. Compare the result with previous one ✎.

Part II. Digital Audio Watermarking

1. Watermark Embedding, Detection, and Attack

Digital Watermarks: Due to the advancement in network communications and multimedia signal processing network, exchanging and distributing multimedia become easier and popular. The accompanying problem with this advancement is how to protect the copyright of intellectual property of digital content. Ownership/copyright protection and integrity verification are two key issues. Digital watermarking is a class of techniques that embed copyright and other protection information in multimedia. In this part, we use *AudioMark* Demo[6] to embed and detect watermark in audio and explore the weakness of this demo software.

(a) Embedding of Digital Audio Watermark: By *Watermark*→ *Cast* to load the *sample.wav* and specify a KEY (the range of demo version is between 100000000~100000100) and the filename of watermarked file.



(a) Detection of Digital Audio Watermark: By *Watermark* → *Detect* and specify the key/seed, the detection module will determine the existence of the watermark.

(b) Attack on Robust Audio Watermark: An adversary may modify a piece of watermarked multimedia to try to remove the watermark without degrading the perceptual quality of multimedia too much. To increase the robustness of watermarking, designers should understand typical attacks and prevent them in advance. In this part, you are asked to attack the watermarked wave file using the tools provided by *GoldWave* and test the existence of watermark in the attacked file using *AudioMark* Demo. Try to keep the quality of music acceptable. Here are a few possible ways in which you can attack.

% Len_order: Number of samples for calculating spectrum, Len=power(2,Len_order);
% shift: start position to compare two signals

[6] AudioMark Demo: http://www.alphatecltd.com/watermarking/audiomark/audiomark.html .

(1) MP3 compression: Choose *mp3* by *File* → *Save As*.

(2) Echoing: Add *echo* by *Effect*→*Echo*.

(3) Enhancement and filtering techniques, such as low pass filtering, quantization, and equalization: By *Effect*→ *Filter*→*Low/HighPass, Equalizer,* or *Effect*→ *Resample*.

(4) Noise addition: By *Tools*→*Expression Evaluator*.

Develop an attack scheme that can preserve reasonable good sound quality yet remove watermark embedded by *AudioMark Demo.* Describe under what conditions the watermark will be destroyed. ✐

2. Design Your Own Audio Watermarking Systems

There are three basic issues in designing an audio watermarking system:

a. *Transparency*: The digital watermark should not degrade the perceptual quality of the signal.

b. *Robustness*: For watermark conveying owner's rights, adversaries would have incentives to remove the watermark by modifying and attacking the watermarked audio. The watermarks in these applications should be robust enough to survive a wide range of attacks. On the other hand, watermark that is fragile to processing can be useful for detecting tampering, where the change in watermark will signal and locate the altered regions of a multimedia signal. As multimedia is often stored in compressed format for efficient storage and compression, even in the case of fragile watermark, it is often desirable to design the watermark to sustain moderate compression.

c. *Capacity / Payload*: It is desirable in many applications to allow the embedded watermarks to carry enough payload bits for representing various types of information.

In this section, you will design two digital audio watermarking systems, and evaluate them in terms of transparency, robustness, and payload.

(a) System #1

One of the simplest approaches to "hide" a message in audio is to convert the message into bits and putting them into the least-significant-bits (LSBs) of audio samples. To help detector make a more reliable decision, you may repeatedly embed each message bit in a number of audio samples at the embedder's side, and do a majority voting at the detector's side.

(1) Implement this LSB-based watermarking in Matlab to embed and detect the following message in an audio file *sample.wav* :

“`(c)Spring 2003. DO NOT SELL. DO NOT TAMPER. Go Terps!`”

Your implementation may include a message encoding function, watermark embedding and detection functions, and a message decoding function. 🖫

Note 1: The *sample.wav* can be downloaded from course webpage. It is a stereo audio file. For simplicity, you can just embed watermarks in one channel here.

Note 2: Matlab supports the I/O of WAVE file format. *wavread.m* and *wavwrite.m* are for reading and writing wave files, respectively. The format of *sample.wav* file is signed 16-bit, i.e. the range of its signed integer representation is between [-32768, 32767]. However, the value obtained by *wavread.m* is in stored as a “double” between [-1, 1). To embed watermarks in the LSBs in the Matlab environment, you may find it convenient to convert the [-1, 1) values to 16-bit unsigned integer values [0 65535] using the formula $(x+1)*2^{15}$.

Note 3: The following Matlab built-in/toolbox functions may be helpful to your implementation: dec2bin( ), bin2dec( ), char( ), double( ).

Note 4: Try to implement your watermarking system in a flexible way to accommodate the embedding in the n^{th} LSB bits, n = 1, 2, 3 …. See the instructions below regarding the transparency and robustness.

(2) Transparency: Listen to the watermarked audio whose 1st LSBs carries your message. Does the watermark affect the quality of the audio? Change your embedding function (and correspondingly the detection) to put your message in the 2nd LSBs and answer the above question again. How about 3rd LSBs and 4th LSBs? ✐

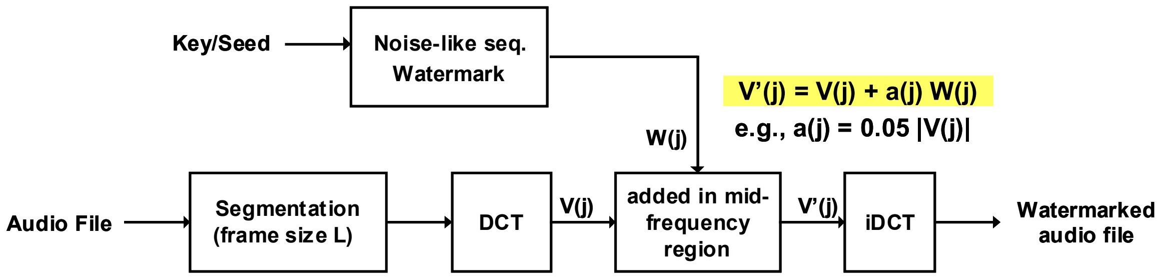
(3) Robustness, Security, and Applications: How robust is the watermarking system that embeds message in the 1st LSBs? How about the 2nd, 3rd, and 4th LSBs? How does the repeating time affect the robustness? Can an unauthorized person change the embedded message? Design tests and use the results to justify your answers 🖫 ✐ .

Note: You can use the bit error rate (BER), the percentage of bits that are incorrectly decoded, to measure the robustness.

(b) System #2

The watermark can be embedded either in time domain or frequency domain. In this section, we employ spread-spectrum embedding to put a watermark in the 1-D DCT domain.

Embedder:



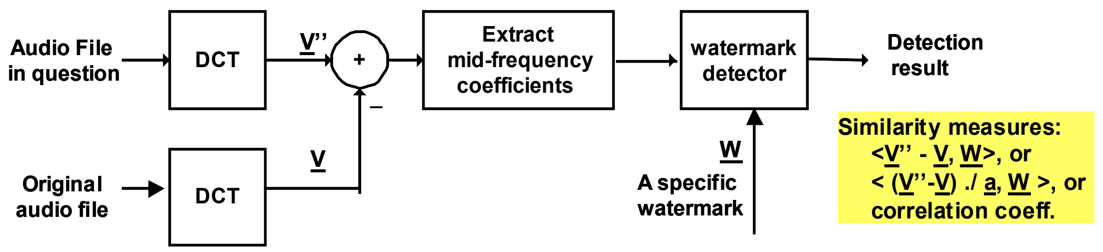
The basic procedures of spread-spectrum embedding is as the follows:

- First, segment an audio file into non-overlapped frames with size L=1024 and apply a 1-D DCT for each frame.
- Second, construct a noise-like vector ***w*** of length L as your watermark (such as through "randn( )"). Normalize the strength of your watermark, for example, use a random number generator to generate each element w_i with a variance of 1.
- For each frame of the audio, add watermark according to $v'_i = v_i + \alpha_i w_i$, $i = 1, \ldots,$ L, where v_i denotes an original coefficient and v'_i denotes the watermarked version. The scaling factor α_i controls the strength of your overall watermark. Here we apply the following simple rules: set α_i to zero except for mid-frequency DCT coefficients (i.e. $\alpha_i = 0$ for i < T1 and i > T2, where the frequency thresholds T1 < T2 are determined by your experiments). For each mid-frequency coefficient v_i, set α_i to be 3-10% of $|v_i|$. You can determine the exact setting empirically.

  As can be seen from this simple rule, a watermark is only embedded in the mid-frequency DCT coefficients of an audio frame. This is because information in the mid-frequency is more important than the other part for human auditory system. An adversary does not want to sacrifice too much quality of audio by hacking this informative part. Accordingly, the watermark is more likely to survive. A more sophisticated choice of α_i for mid-frequency part should be guided by human auditory model.
- After embedding, perform an inverse DCT to convert the signal back to time domain. Clip the amplitude of the watermarked signal to the range of the original audio sample [-1, +1]. Repeat the process for every frame.

Detector:

We use the original unmarked audio file to help determine the existence of a specific watermark. Thus the detector would know the original audio file, a watermark sequence, and of course, the watermarking method and all the related parameters.



The basic procedures of spread-spectrum embedding is as the follows:

- We perform DCT on both the original and the audio file in question, and compute the difference between the corresponding elements of these two sets of DCT coefficients. We shall denote this difference vector as $\underline{z}$.
- Retain only the elements of $\underline{w}$ and $\underline{z}$ that correspond to the mid-frequency part in which the embedder chooses to embed watermark. We denote the retained vectors as $\underline{w}^{(m)}$ and $\underline{z}^{(m)}$.
- Measuring the similarity between $\underline{w}^{(m)}$ and $\underline{z}^{(m)}$ by computing the correlation coefficient. A high positive correlation coefficient indicates that with high probability the audio frame in question comes from adding $\underline{w}^{(m)}$ to the original audio frame. You may also try to take into account the scaling factor $\underline{a}$ when measuring the similarity.
- Repeat the process for other frames. Plot the correlation coefficients you obtained from all frames.

Here is your To-Do list:

(1) Use Matlab scripts/functions to implement an embedder and a detector according to the procedures described above 💾.

Note: The following Matlab built-in/toolbox functions may be helpful to your implementation: dct( ), idct( ), randn( ), corrcoef( ).

(2) Generate two different spread-spectrum watermarks $\underline{w}_1$ and $\underline{w}_2$. Produce a watermarked audio files from *sample.wav* with $\underline{w}_1$ embedded, and name the watermarked file as *marked1.wav*; produce a 2nd watermarked audio files from *sample.wav* with $\underline{w}_2$ embedded, and name the watermarked file as *marked2.wav*. ♫

Use your detector to determine whether $\underline{w}_1$ can be found in *marked1.wav* and *marked2.wav*, respectively. In a single plot of correlation coefficients vs. audio frames, include and compare your detection results of the two cases. ✎

(3) Transparency and Robustness - Adjust various parameters in your watermarking system (L, α_i, T1 and T2) and examine their impact on transparency and robustness:

i. Listen to the watermarked audio. Does the watermark affect the quality of the audio? ✐

ii. Add noise[7] with different amplitude to the watermarked audio. Does the detector detect the existence of the watermark that was generated from the original user key? ✐

iii. Use GoldWave to add echo in the watermarked audio. Try echoes with short delay a small volume and with long delay a large volume, respectively. Observe the detection result. ✐

iv. Use GoldWave to apply MP3 compression to the watermarked audio. Can your watermark resist the compression attack? ✐

v. Discuss the tradeoff between the transparency and robustness ✐. Include in your report a watermarked audio signal using the parameter settings that you believe giving the best tradeoff ♫.

(4) Compare the transparency and robustness of the watermarks for the two systems investigated above. List their advantages, disadvantages, and potential applications. Discuss how to improve these two systems. ✐

(5) Bonus part: Extend your System#2 to hide a meaningful message such as the message we have used in System#1 🖫 ✐.

Hint: you can use hide one bit in one frame by adding the watermark if to embed a bit "1" ($v'_i = v_i + \alpha_i w_i$) and subtracting the watermark if to embed a bit "0" ($v'_i = v_i - \alpha_i w_i$). You can reuse your message encoding and decoding functions from System#1. If needed, you can repeatedly embed the same bit in a few frames and do a majority voting at the detector's side.

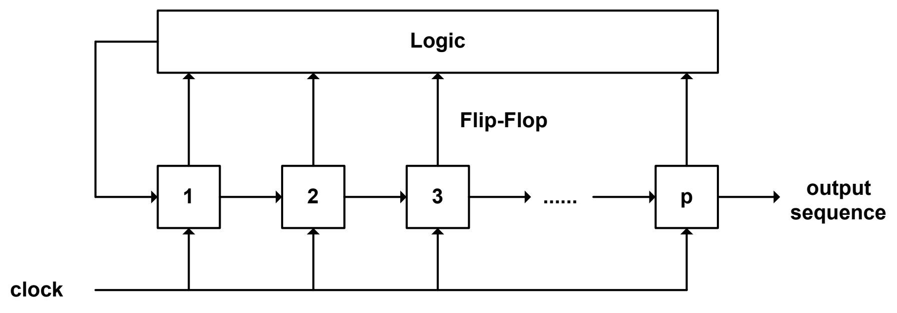
Appendix of Part-II for further exploration: Pseudo-Noise Sequence

Spread-spectrum watermarking uses a noise-like sequence $\underline{\boldsymbol{w}}$ as watermark. Such noise-like signals are less likely to introduce perceivable distortions than structured signal (such as a periodic squared wave). In addition, they have good statistical properties to achieve high noise resistance and help detector make reliable decisions.

In System#2, you have experienced real-valued watermark generated through a Gaussian random number generator. Another simpler choice of $\underline{\boldsymbol{w}}$ is a binary periodic

[7] You can use *rand.m* to generate "uniformly distributed" random sequences or *randn.m* "Normally distributed" random sequences. For example, *noise=A*randn(1,100)* represents 100 normally distributed random numbers with amplitude scalar *A*.

pseudo-noise sequence[8], or PN sequence in short. PN sequence can be generated by a series of shift registers with a feedback logic that is shown in the figure below.



The feedback logic can be expressed by a polynomial,

$$f(X)=g_1X+g_2X^2+\ldots+g_pX^p$$

where X^i indicated the ith Flip-Flop and $g_i \in \{0,1\}$ controls the logic. When the clock triggers, the system will shift all of its values by one unit, output a bit as the PN sequence, and then send the $f(X)$ back to the first Flip-Flop. One implementation is to employ *Galois Field prime polynomials*[9] in the logic feedback part to obtain maximal-length sequence (or m-sequence in short).

Setting p=10 gives us 60 possible polynomials to use[10]. Each polynomial can generate 2^{10}-1 = 1023 different output sequences. You can generate a pseudo random sequence of this kind using a provided MATLAB function *PNsequence.m*[11] with a given polynomial and a seed (which is to initialize the shift register). We can observe the auto-correlation and cross-correlation function of the m-sequence by the following MATLAB programs:

- *Auto-correlation property:*

```
Seq1_index= ?   % Specify your selected key (seed) here
seq1_poly= GFprimMrx(seq1_index,2:end)
seq1 = PNsequence(seq1_poly,bitget(1,1:p));

for seq2_index=1:2^p-1
    seq2 = PNsequence(seq1_poly,bitget(seq2_index,1:p));
    [corrf]=PNcorrelation(seq1, seq2);
    plot(corrf); axis([0, length(corrf)+1, ,0 max(corrf)+1 ]);
    pause;
end
```

[8] For details, please refer to chapter 7 in: Simon Haykin: Communication Systems, 4th edition, Wiley, 2000.

[9] For fixed order *p*, the prime polynomials can be obtained using the Matlab built-in function.
GFprimMrx = gfprimfd(p,'all')
Each row represents a "key" for each user: *GFprimMrx (key,2:end) =* $[g_1\ g_2\ \ldots\ g_p]$
You can store those polynomials as a *.mat* file (by *save* command*)* since searching for the polynomials is time-consuming.

[10] W.W. Peterson and E.J. Weldon, Error-Correcting Codes, Cambridge: MIT Press, 1972.

[11] *out_sequence = PNsequence(GFprimMrx (key,2:end), seed)*

- *Cross-correlation property:*

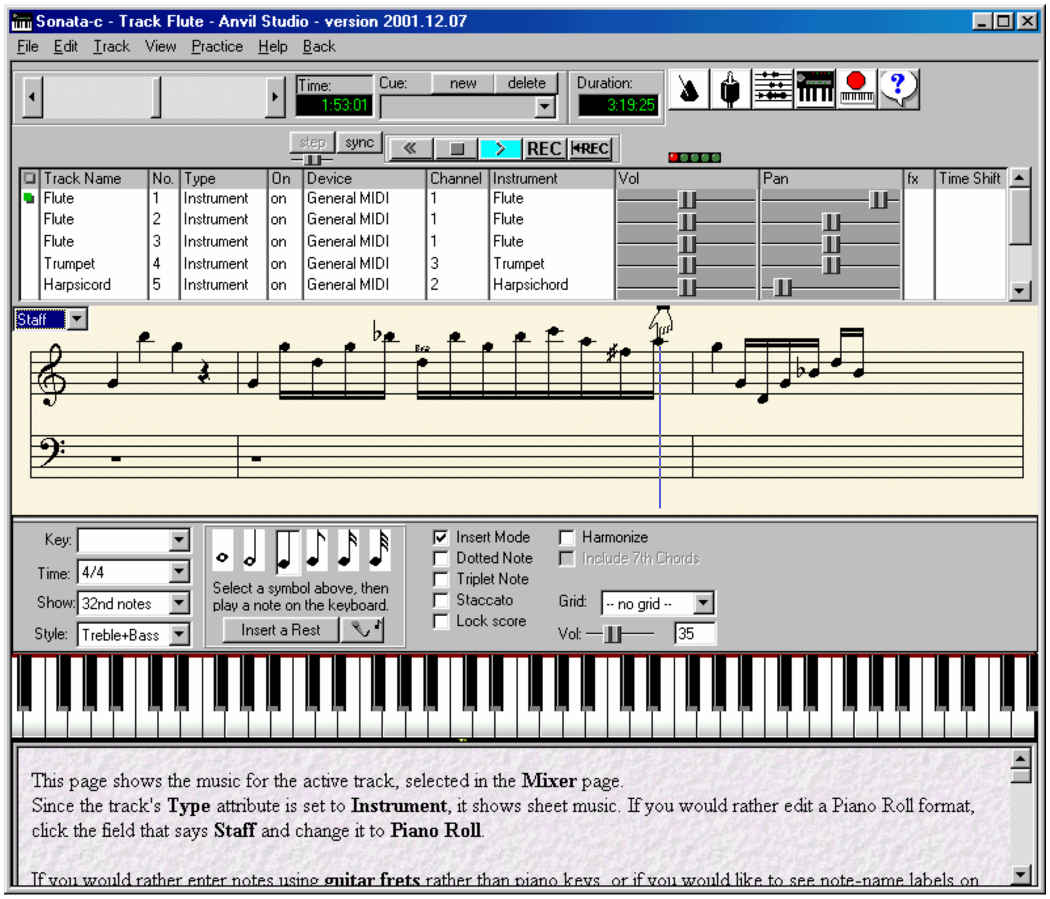
```
Seq1_index= ?    % Specify your selected key (seed) here
NumGF=size(GFprimMrx,1);
maxTable=zeros(1,NumGF);
seq1_poly= GFprimMrx(seq1_index,2:end)
seq1 =  PNsequence(seq1_poly, rand(1,length(seq1_poly))>=0.5);

for seq2_index=1:NumGF
   seq2_poly = GFprimMrx(seq2_index,2:end);
   seq2 =  PNsequence(seq2_poly,rand(1,length(seq2_poly))>=0.5);
  [corrf]=PNcorrelation(seq1, seq2);
   plot(corrf); axis([0, length(corrf)+1, 0 max(corrf)+1 ]);
  maxTable(seq2_index)=max(corrf);
end

plot(maxTable);
find(maxTable==max(maxTable))
```

Part III. Synthetic Audio (1): Musical Instrument Digital Interface (MIDI)

Musical Instrument Digital Interface (MIDI)[12] is different from the digital sampled audio, such as PCM. It can be thought of as instructions telling music synthesizer when to play and what notes to play instead of sending waveform to the speakers. There are several advantages using this synthesis approach. For instances, it requires much less storage space and PC bandwidth in I/O bus. In this part, we use *Anvil Studio*[13] to study the MIDI Protocol. As shown in the figure below, there are several panels (from top to bottom) in this software: play panel, track editor panel, stave panel, note editor panel, and keyboard panel.


Sonata-c - Track Flute - Anvil Studio - version 2001.12.07
File Edit Track View Practice Help Back
Time:
Cue: new delete
Duration:
step sync
REC
Staff

Time: 1:53:01
Duration: 3:19:25

Track Name	No.	Type	On	Device	Channel	Instrument	Vol	Pan	fx	Time Shift
Flute	1	Instrument	on	General MIDI	1	Flute				
Flute	2	Instrument	on	General MIDI	1	Flute				
Flute	3	Instrument	on	General MIDI	1	Flute				
Trumpet	4	Instrument	on	General MIDI	3	Trumpet				
Harpsicord	5	Instrument	on	General MIDI	2	Harpsichord				

Key:
Time: 4/4
Show: 32nd notes
Style: Treble+Bass

Select a symbol above, then play a note on the keyboard.
Insert a Rest

[✓ Insert Mode]
Dotted Note
Triplet Note
Staccato
Lock score
Harmonize
Include 7th Chords
Grid: -- no grid --
Vol: 35

This page shows the music for the active track, selected in the **Mixer** page.
Since the track's **Type** attribute is set to **Instrument**, it shows sheet music. If you would rather edit a Piano Roll format, click the field that says **Staff** and change it to **Piano Roll**.

If you would rather enter notes using **guitar frets** rather than piano keys, or if you would like to see note-name labels on

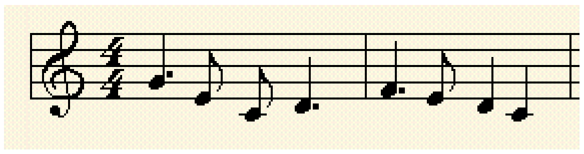
(a) Load the *Sonata-c.mid* by *File→Open Song*. Double click the *play* button in the play panel. Then modify the track editor panel, such as, the channel and instrument[14]. The channel setting is for assigning an audio channel for each track and the instrument setting is to choose what instrument to play.

[12] A useful Tutorial of MIDI: http://www.harmony-central.com/MIDI/Doc/tutorial.html
[13] Anvil Studio: http://www.anvilstudio.com/upgraden.htm
[14] There are 16 logical channels and 128 instruments in the General MIDI (GM) systems. Those number of instruments are standardized, thus different music synthesizer will not play the different instrument while reading the same instrument number form MIDI file. However, this does not mean those music

(b) Notes in MIDI file are defined as the following table (each note takes 4 bits).

Octave	Music Notes											
	C	**C#**	**D**	**D#**	**E**	**F**	**F#**	**G**	**G#**	**A**	**A#**	**B**
0	0	1	2	3	4	5	6	7	8	9	10	11
1	12	13	14	15	16	17	18	19	20	21	22	23
2	24	25	26	27	28	29	30	31	32	33	34	35
3	36	37	38	39	40	41	42	43	44	45	46	47
4	48	49	50	51	52	53	54	55	56	57	58	59
5	60	61	62	63	64	65	66	67	68	69	70	71
6	72	73	74	75	76	77	78	79	80	81	82	83
7	84	85	86	87	88	89	90	91	92	93	94	95
8	96	97	98	99	100	101	102	103	104	105	106	107
9	108	109	110	111	112	113	114	115	116	117	118	119
10	120	121	122	123	124	125	126	127				

The *Anvil Studio* has provided a keyboard interface at the bottom of the program's window, which can automatically translate notes keyed in into the above note numbers. Create a new MIDI file by *File* → *New Song* and use the keyboard panel to key in the following score. Save this MIDI file. ♫



(c) To check whether the MIDI file is generated successfully, you can use the Windows Media Player to listen to this MIDI file.

Part IV. Essays: Digital Rights Protection of Multimedia and related Ethics issues for Engineers

Intellectual Property (IP), such as copyright, plays an important role in contemporary society. This is among several important ethics issues that engineers may face in their career. We will have a lecture on ethics discussion. There are reading and writing assignments associated with the ethics discussion, and they are to be completed individually by a separate deadline. The details will be announced shortly.

synthesizers will play the same waveform. In fact, every music synthesizer has its own approach to generate any specified note of instrument. Usually, there are two approaches, Frequency Modulation and Wave Table. The quality of the latter approach is much better than the former. Except hardware synthesizer, there exists software synthesizer generating and mixing waveforms. *Microsoft GS Wavetable SW Synth* is the popular one under Windows system, which adopts Roland instrument sounds. You can try to change the MIDI playback device, no matter hardware of software synthesizer, and listen the differences among them.

Part V: Mobile Computing and Pocket PC Programming

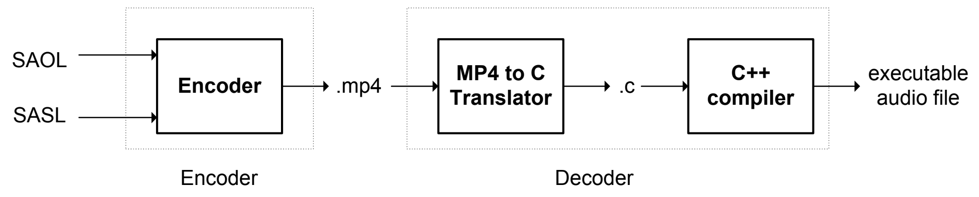
From the above three parts, we have learned the fundamentals of digital audio processing. Now design a simple Pocket PC application related to digital audio processing using the Microsoft eMbedded Tools. You can refer to "*ENEE408G Multimedia Signal Processing Mobile Computing and Pocket PC Programming Manual*" and extend the examples there. ✎🖫

Bonus Part I. Synthetic Audio (2): MPEG4- Structured Audio (MP4-SA)

MPEG4 synthetic audio coding standard consists of two methods, namely, Structured Audio (SA) and Text-to-Speech (TTS). In the SA part, MPEG4 defines the Structured Audio Orchestra Language (SAOL) and Structured Audio Score Language (SASL)[15]. Instead of using the frequency modulation and wavetable technique as in MIDI, SAOL encodes a sound signal according to its structure. This technique can achieve extremely high compression ratio, about 100:1~10,000:1. In this section, we explore these two languages, SAOL and SASL, using the *SPlay* and *SNet*[16].

SAOL and SASL:

The role of SAOL is sound modeling, which encodes algorithms on how instruments generate sounds. On the other hand, SASL is for sound sequencing, i.e. a timing table giving instructions to each instrument on when and how to play notes. The figure below illustrates the framework. The SAOL and SASL are in plain text file format. The encoder encodes those two files into binary form known as the MP4 format. A MP4 file contains the structure of instruments and the scores of the music instead of digitalized sample waveforms. The decoder converts the MP4 file into a C file and then compiles it as an executable audio file.



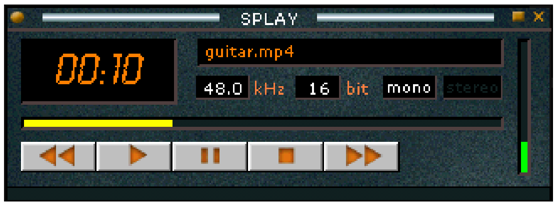
SPlay:

SPlay is a software program that implements a decoder. Download several MP4 files from http://student-kmt.hku.nl/~saol/ and listen to the results. Notice that the file size is quite small. The behavior of this audio player is different from that of a waveform

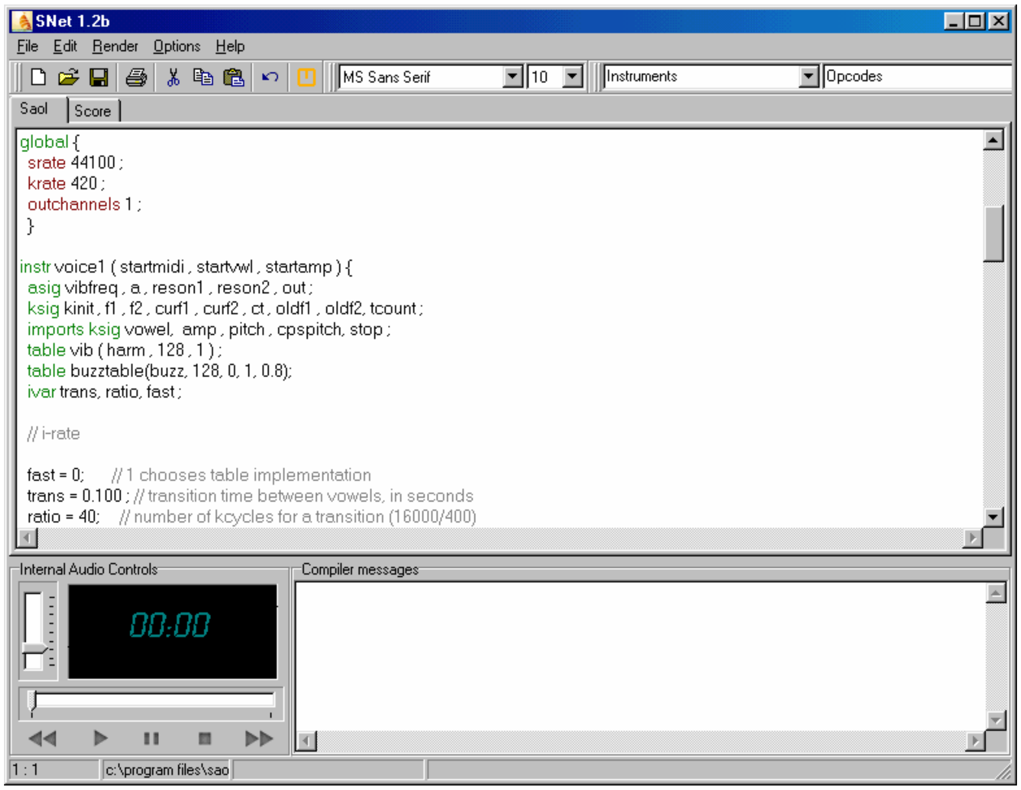
[15] MP4-SA Language Standard: http://www.cs.berkeley.edu/~lazzaro/sa/book/append/fdis/SA-FDIS.pdf.
(a) A useful online book explains how to use this new language: *The MPEG-4 Standard Structured Audio book*, http://www.cs.berkeley.edu/~lazzaro/sa/book/ by John Lazzaro and John Wawrzynek.
(b) You can watch a short presentation about MP4-SA by John Wawrzynek at http://bmrc.berkeley.edu/bibs/instance?prog=1&group=13&inst=35

[16] SPlay and SNet are available at: http://student-kmt.hku.nl/~saol/

player, such as players for .wav and .mp3 files. Since an MP4 player translates the MP4 file into C file and compiles it, this player will take more time than a waveform player.



(a) In this sub-section, we explore SAOL and SASL using *SNet*. *SNet* is a GUI wrapping the kernel *sfront*. You can refer to John Lazzaro's online book, *The MPEG-4 Standard Structured Audio Book*, to learn more details about how to use these languages.



```
global {
 srate 44100 ;
 krate 420 ;
 outchannels 1 ;
 }

instr voice1 ( startmidi , startvwl , startamp ) {
 asig vibfreq , a , reson1 , reson2 , out ;
 ksig kinit , f1 , f2 , curf1 , curf2 , ct , oldf1 , oldf2, tcount ;
 imports ksig vowel,  amp , pitch , cpspitch, stop ;
 table vib ( harm , 128 , 1 ) ;
 table buzztable(buzz, 128, 0, 1, 0.8);
 ivar trans, ratio, fast ;

 // i-rate

 fast = 0;      // 1 chooses table implementation
 trans = 0.100 ; // transition time between vowels, in seconds
 ratio = 40;    // number of kcycles for a transition (16000/400)
```

(1) Read the online book Part I – A Tutorial Introduction.

(2) Play those three examples (*sine*, *vsine* and *vcsine*) by *SNet*. First, copy *.saol file and then paste on the *SAOL* tab. Save it. Then, copy the *.sasl file and paste on the *Score* tab. Save them by *Render* → *Render to .mp4*. ♬ .

(3) Use *SPlay* to play the MP4 files. Discuss the advantages and disadvantages of waveform coding (PCM) and synthetic audio (e.g. MIDI and MP4-SA) ✎.