Laboratory Exercise 8

Audio CODEC

The purpose of this exercise is to learn how to use the Audio enCOder/DECoder (CODEC) on an Altera DE-series Board. We will connect a microphone and speakers to the DE-series board to allow recording and playback of sounds. To access the microphone and speakers we will use the DE-series Media Computer, which is described in the *DE-series Media Computer for the Altera DE-series Board* manual.

Background

Sounds, such as speech and music, are signals that change over time. The amplitude of a signal determines the volume at which we hear it. The way the signal changes over time determines the type of sounds we hear. For example, an 'ah' sound is represented by a waveform shown in Figure 1.

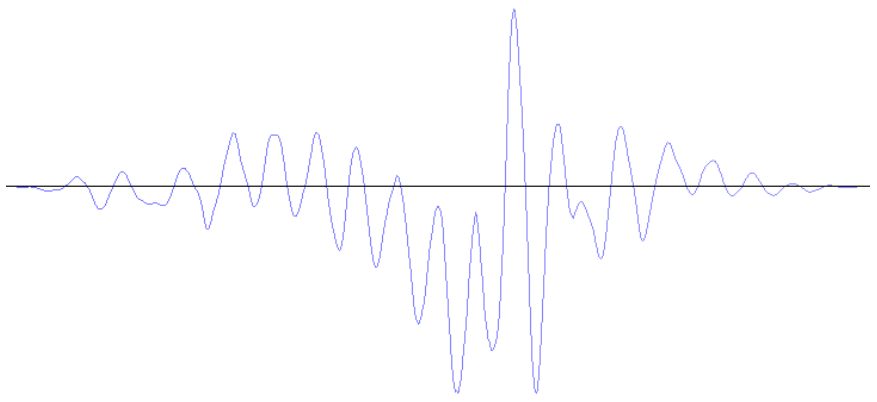


Figure 1: A waveform for an 'ah' sound.

The waveform is an analog signal, which can be stored in a digital form by using a relatively small number of samples that represent the analog values at certain points in time. The process of producing such digital signals is called *sampling*.

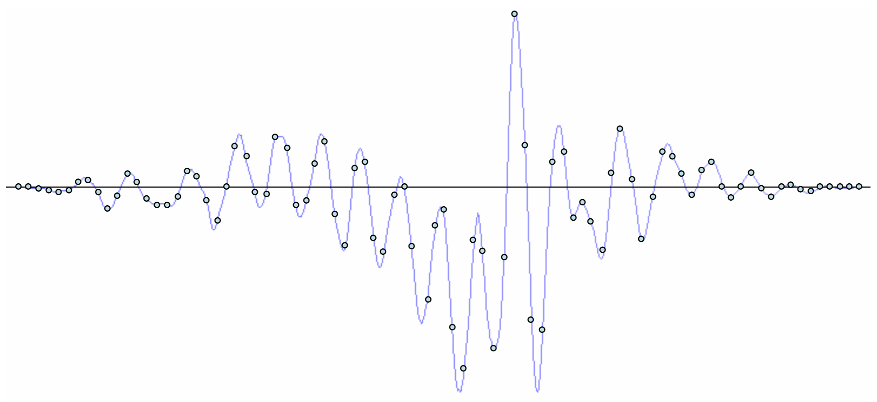


Figure 2: A sampled waveform for an 'ah' sound.

The points in Figure 2 provide a sampled waveform. All points are spaced equally in time and they trace the original waveform.

To be able to record and play back sounds we will use the DE-series Media Computer. It is a computer system very similar to the DE-series Basic Computer, except that it contains a few additional components useful for running multimedia programs. In particular, it contains an Audio Port to access microphone and speakers, Video-Out Port to display graphics on a computer screen and a PS/2 Port to connect a keyboard or a mouse directly to the DE-series board. In this exercise, we will use the DE-series Media Computer to access the Audio CODEC for recording and playback. Figure 3 shows the part of the DE-series Media Computer we will use.

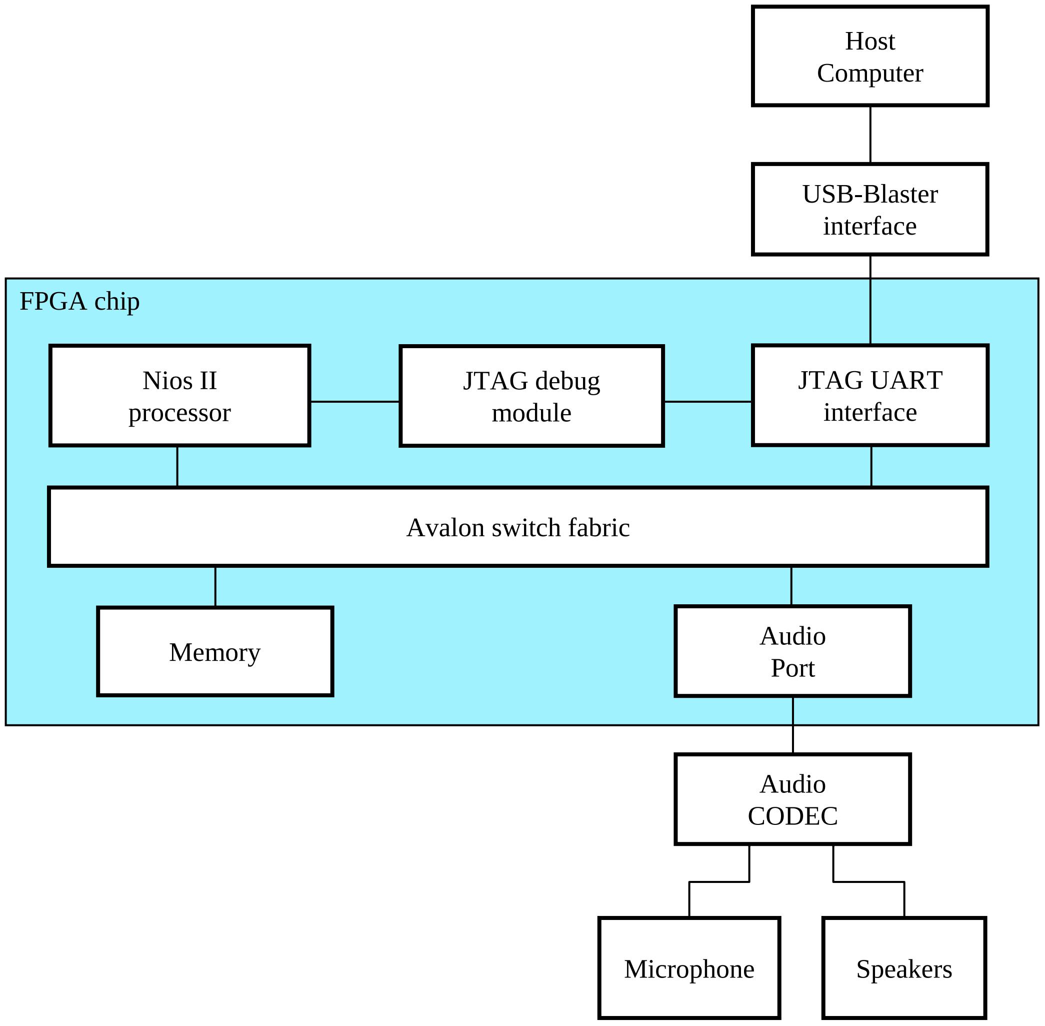


Figure 3: Portion of the DE-series Media Computer used in this exercise.

The Audio Port provides a link between a program executed by the Nios II processor and the Audio CODEC connected to the external microphone and speaker system, as shown in Figure 4. A user program executed by the Nios II processor can record and play back sounds by using the Audio Port.

Sound can be recorded by the system using a microphone. The microphone converts the sound into an analog waveform, such as the one shown in Figure 1. The CODEC samples the waveform every 1/48000th of a second, and stores the sample in the Record FIFOs. The Record FIFOs can then be read by the processor. To produce an output waveform, the Audio Port accepts a sampled waveform as an input from the processor and stores it in the Playback FIFOs. The CODEC then converts the data in the FIFOs into an analog waveform and sends it to the speakers.

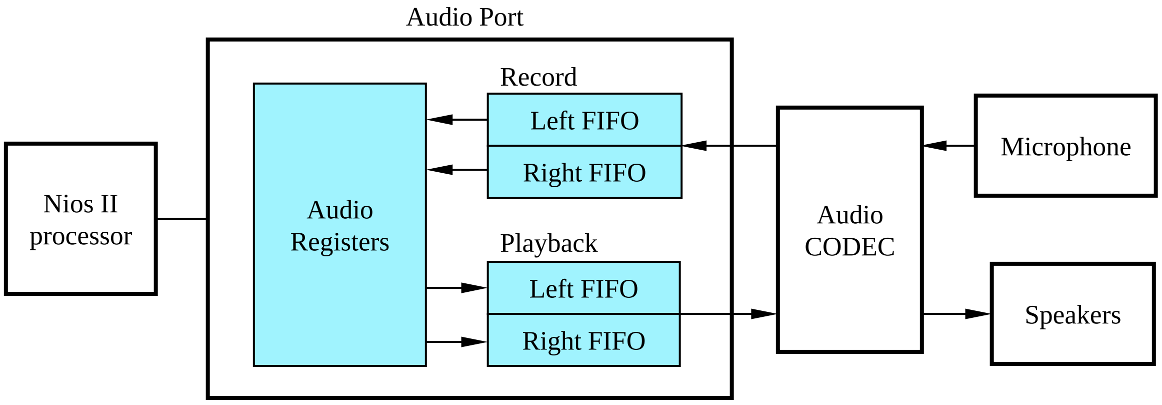


Figure 4: The audio subsystem.

User programs can access the Audio Port through its I/O interface in the DE-series Media Computer. The interface includes four memory-mapped registers. Figure 5 shows the registers and their location in the memory address space.

Address	31 ··· 24	23 ··· 16	15 ··· 10	9	8	7 ···	3	2	1	0	
0x10003040	Unused			WI	RI		CW	CR	WE	RE	Control register
0x10003044	WSLC	WSRC	RALC			RARC					Fifospace register
0x10003048	Left data										Leftdata register
0x1000303C	Right data										Rightdata register

Figure 5: Audio access registers on the DE-series Media Computer.

The *control register* contains six bits that are used to control the Audio Port:

- RE and WE bits enable the Audio Port to generate an interrupt request when the Record FIFO is 75% full (RE), or the Playback FIFO is less than 25% full (WE), or both
- CR and CW bits clear the Record and Playback FIFO buffers, respectively.
- RI and WI bits are set to 1 when interrupt requests are raised. RI becomes 1 when the Record FIFO is at least 75% full; WI bit becomes 1 when the Playback FIFO is less than 25% full.

The *Fifospace register* indicates the current state of each of the FIFOs. Each FIFO has space for 128 samples. The number of samples available in the Left/Right Record FIFO can be read from the RALC/RARC field. Similarly, the amount of space available for samples in the Left/Right Playback FIFO can be read from the WSLC/WSRC field.

The *Leftdata* and *Rightdata* registers are used either to load data into the Playback FIFOs (left or right) by writing to the registers, or to retrieve sound from the Record FIFOs by reading these registers. Users should take care to check the state of Record and Playback FIFOs before reading or writing to *Leftdata* and *Rightdata* registers. If the Record FIFOs are empty, then the contents of the *Leftdata* and *Rightdata* registers are invalid. Also, if the Playback FIFOs are full, then writes to the *Leftdata* and *Rightdata* registers will be ignored by the Audio Port.

Part I

In this part of the exercise you will create a 3-second recording and play it back. To do so, a microphone and speakers need to be connected to the DE-series board. Speakers should be connected to the green jack, labeled LINE-OUT, and the microphone should be connected to the pink jack labeled MIC.

You are to write a C program that uses the Audio Port to record and play back sounds. Your program should read the data from the Audio Port and store it in memory when the user presses the KEY_1 pushbutton. The recording should be played back by writing samples stored in memory to the Audio Port when the user presses the KEY_2 pushbutton.

The playback of the recording should also be controlled by slider switches SW_{1-0}. When switch SW_1 is 1, then the playback should skip every other sample when playing back the recording. If SW_0 is 1, then the playback should repeat every sample from the recording. Otherwise, the recorded sound should be played back normally. Follow these steps to complete this part of the exercise:

1. Write a C program described above.
2. Start the Altera Monitor Program and create a new project. When choosing the system for your project, select the DE-series Media Computer from the drop-down list.
3. Download the DE-series Media Computer onto the DE-series board and compile your program.
4. Download your program into the system and run it.
5. Test the effect of switches SW_{1-0}.

Part II

Echo is an effect we often hear in large halls or caverns, where a sound we make comes back to us with some delay. This effect occurs because the sound bounces off the walls and returns to us. The reflected sound is somewhat attenuated.

In this part of the exercise you are to write a C program that reads sounds from the microphone and produces the sound back through the speakers. However, moments later the same sound should repeat, at a lower volume, and again a few moments after that to simulate the effect of an echo. The echo effect should be superimposed on (added on top of) any sound being provided through the microphone.

Part III

Sound tones can also be generated using the Audio Port. Each tone is a sinusoidal waveform of a specific frequency. For example, the tones in the fourth octave on a piano have the following frequencies:

C = 262 Hz
C# = 278 Hz
D = 292 Hz
D# = 310 Hz
E = 328 Hz
F = 348 Hz
F# = 370 Hz
G = 392 Hz
G# = 416 Hz
A = 440 Hz
A# = 466 Hz
B = 494 Hz

To generate a sinusoidal waveform of a given frequency you have to supply points that trace that sinusoid over time. This can be accomplished using the *sin* function from the *math.h* library. Recall that

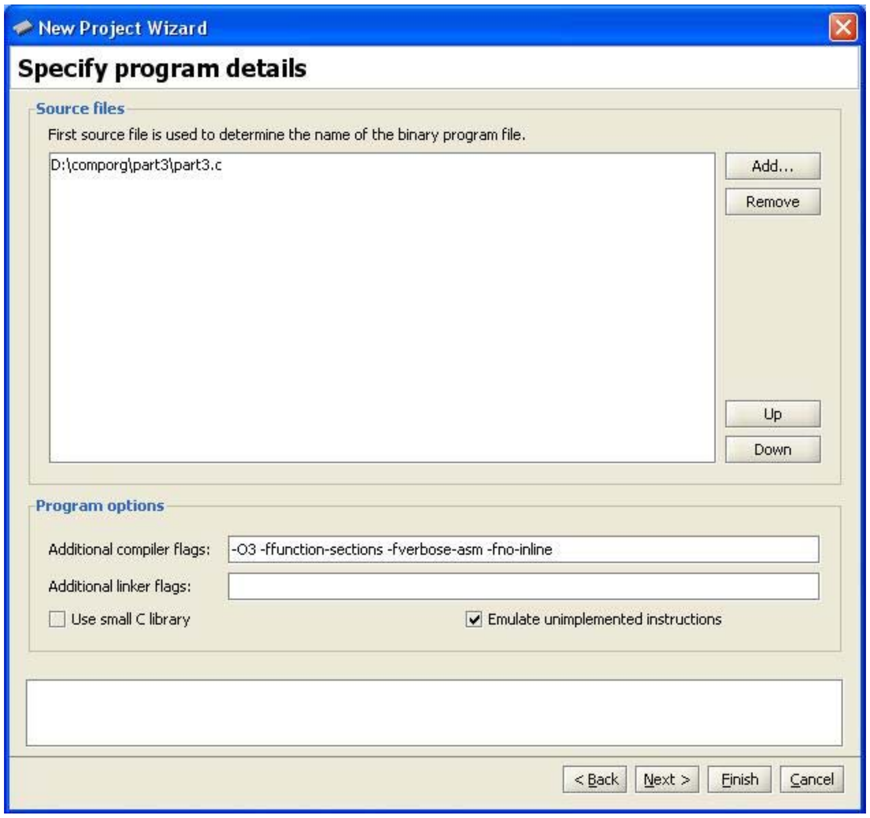
$$v(t) = A \cdot sin(2 \cdot \pi \cdot \frac{t}{T})$$

where A is the amplitude of the sine wave, T is its period, and t is the time index. Thus, a sinusoidal waveform can be created by supplying $v(t)$ as input to the Audio Port, where values of $v(t)$ are generated at Δt intervals.

Create a piano-playing program that plays tones from the fourth octave. The user will play the "piano" by pressing keys on the keyboard. Use keys a, w, s, e, d, f, t, g, y, h, u, and j to represent each of the tone, respectively. When any of these keys is pressed, your program should play the corresponding tone for approximately half of a second.

Do the following:

1. Create a header file, *tones.h*, with 12 tables that represent the tones in the fourth octave. Each table should consist of 24000 entries that trace the waveform for a given tone. The amplitude of each waveform should be $2 * 10^7$.
2. Write a piano-playing program in C language. Make sure to include the *tones.h* file in your program.
3. Create a project in the Altera Monitor Program for the DE-series Media Computer. When specifying compiler options in the window shown in Figure 6, make a change in the field called *Additional compiler flags*. The parameter -O1 should be changed to -O3, as shown in Figure 6, to configure the C compiler to use the highest optimization level. This change should allow your program to be executed fast enough to read keys from the keyboard and play back several tones at the same time.



Specify program details

Source files

First source file is used to determine the name of the binary program file.

D:\comporg\part3\part3.c

Program options

Additional compiler flags: -O3 -ffunction-sections -fverbose-asm -fno-inline

Additional linker flags:

[ ] Use small C library

[✓] Emulate unimplemented instructions

Figure 6: Program Details screen in Altera Monitor Program.

4. Compile and run your program using the Altera Monitor Program.

Hint: You will need to use UART interrupts, or access the UART directly, to be able to read key strokes without interrupting the music. You should not use C functions such as *scanf* to read keystrokes.

Preparation

The recommended preparation for this laboratory exercise includes:

1. C code for Part I.
2. C code for Part II.
3. A header file for Part III, called tones.h, with 12 integer tables - one for each tone. Each table should contain 24000 elements, which trace a waveform that corresponds to one of the tones in the fourth octave.