IN325: Project – CD player

Author : Christophe Garion <garion@isae.fr>
Audience : 3A IN
Date :

Abstract

The objectives of this project is to implement the core of a real-time CD player.

1 Problem

The software inside a CD player has to deliver 16-bit values to a digital-to-analog converter (DAC) for both left and right channel at a 44.1 kHz frequency. We want to build the software that drives such a CD player. The objective is to write Java code to handle the data flow from the CD drive to the DAC. Figure 1 presents the three components that you need to develop and the associated data flows.

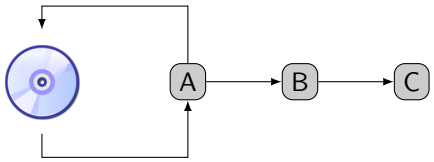

Figure 1: The three components for the CD player

- component A reads 2352-syze sectors from the disk, divides the sector into 16-bit samples and place the samples in queue B. When the queue is too full or empty, it adjusts the disk drive's motor. The disk reader needs to average 75 sectors per second.

  Hardware transfers data from the disk to a buffer in memory with a double buffering system. Each time is buffer is filled, the disk drive sends a signal to the software.

- component B is a queue. Its size will be at least 89,376 bytes in order to be able to handle read errors.

- component C writes sample pairs (left/right) to the DAC. It has to write them very punctually at one pair every 22.7 μs.

2 Classes to be implemented

2.1 Device drivers

There are two devices drivers to be implemented: the disk controller (class `DiskDriver`) and the DAC driver. Both drivers has to be able to access raw memory:

- the disk controller has dedicated RAM for two 2352-byte buffers located at physical addresses 0x00385000 and 0x00386000. It has also a control register (1 byte) at physical adress 0x0038700 that regulates the rotational velocity of the disk. Meaningful values for the control byte range from 0x00 (stop the disk) to 0x0f (maximum speed)

- the DAC driver pilots two digital analog converters (one for the left channel, one for the right channel). Each DAC has a 16-bit register for data and a 8-bit control register. Writing 0x01 to the control register causes the value in the data register to be converted to an analog value. The data register for the left channel is at physical address 0x00384000, its control register at physical address 0x00384028 and the right channel's registers are 0x10 bytes higher.

We will simulate raw memory access using a `RawMemoryAccess` class that you can find in the archive available on http://www.tofgarion.net/lectures/IN325. The `RawMemoryAccess` class has the following public constructor and methods:

- `RawMemoryAccess(long b, int size)`: creates a raw memory area of size `size` at address `b`

- `byte getByte(long offset)` and `short getShort(long offset)`: get the byte or short located at address `b + offset` where `b` is the creation address of the raw memory area

- `setByte(long offset, byte v)` and `setShort(long offset, short v)`: set the byte or short located at address `b + offset` where `b` is the creation address of the raw memory area

Skeletons of `DiskDriver` and `DacDriver` classes are provided in the archive.

2.2 The queue

The queue between the disk reader and the DAC writer will be modelled by two classes:

- `QueueEntry`, a class that encapsulates a single sample of left and right 16-bit values
- `Queue`, a class that represents the queue. The queue has unbounded size and the size of the queue should be accessible. The queue never blocks on dequeue even if the queue is empty (it returns a **`null`** value in this case)

2.3 The DAC-writing mechanism

The DAC writing mechanism will be implemented by the `DacWriting` class. When creating the thread in the main program, its priority should be `NORM_PRIORITY + 1`.

2.4 The disk-reading mechanism

The disk-reading task is released once for each disk sector. Its main goal is to transfer samples from a disk buffer to the queue, but it also regulates the disk motor's speed. If n is the number of entries per disk buffer, then when there are less that $45 \times n$ entries in the queue, it will accelerate the disk and when there are more than $60 \times n$ entries in the queue, it will decelerate the disk. We suppose that the disk-reading mechanism will not get sectors faster than every $2,666666$ ms.

2.5 The simulator

In order to simulate hardware, a `Simulator` class is available in the archive. *This class is not complete*. The class generate fake sinusoidal signals to fill the buffers. It should be used by the DAC-writing mechanism in order to print values to the console to "simulate" speakers.

3 Questions

1. what kind of threads or handlers will you choose for components A and C? Justify your answer.
2. complete or implement the application classes.
   - for the `DiskDriver` class, add the following methods:
     - **`short`** `readSample(int selector, int index)`: read the sample for bugger `selector` at index `index`
     - **`void`** `faster()` and **`void`** `slower()`: change speed of the CD drive
   - for the `DacDriver` class, add the following method:
     - **`void`** `write(short left, short right)`: write the corresponding values in the raw memory area
   - implement the `QueueEntry` and `Queue` classes
   - implement the DAC-writing and disk-reading mechanisms
   - complete the `run` method of the `Simulator` class
   - create a `CDPlayer` application that launch all necessary threads and watch the sound ☺

All developped or completed classes should be documented and put into the `src` directory of your Subversion repository that is located at https://eduforge.isae.fr/repos/IN325/login where `login` is you login name at ISAE. A Makefile or Ant file will be also provided and you can provide comments on your work in a `README.txt` file. **Final version of your files should be committed before January, 25th 2013 23:00**.

References

[1] Greg Bollella et al. *The Real-Time Specification for Java*. 2005. URL: http://www.rtsj.org/specjavadoc/book_index.html.