Homework for Lab-5: Note Generation in C

Up to now, we have used assembly programming to implement various tasks. Assembly programming permits direct access to memory, interrupts and on-board peripherals of the micro-controller. This has provided us an insight into the working of the micro-controller.

However, it is also important to learn the use of high level languages like C with micro-controller based systems. Programming is easier in high level languages because we do not have to manage the exact locations where variables are stored. Also, there are built in constructs for looping, arrays and structure handling etc. Constructs in high level languages are self-documenting to a greater extent.

The standard version of C is designed to insulate the programmer from hardware details of the system. However, in micro-controller based applications, we do need to control interrupts and peripherals etc. We shall use the C compiler which is provided by the Keil μVision program. It allows one to interact with the hardware through constructs which are additional to standard C statements. (This version of C is called "embedded C" by Keil). A tutorial on embedded C has been uploaded to moodle. Please browse through it.

In this lab, we shall generate musical notes as was done in Lab-3 (– wasn't that fun!), this time using C.

You are already familiar with the external hardware required by the system. Re-read the material which had been put up for that lab to refresh your memory.

Homework

1. Write a program which will use three constant arrays as follows:
   (a) The first array stores half periods of 12 semitones in an octave as 16 bit integers.
   (b) The second array stores the sequence of notes to be played. Each entry in this array is a byte. The upper 4 bits of the byte store octave information, while the lower four bits store which note is to be played.

(c) The third array contains bytes representing the duration of each note as a multiple of 10 ms.

Your program should initialize a "note_index" to 0. Then it should enter a loop in which

- it uses this note_index to read the duration from the duration array (c). If the duration is 0, the program should terminate. If it is not 0, the duration should be stored in a variable.
- The same note_index should be used to fetch a byte from the note sequence array (b). The program should separate the upper and lower nibble of the fetched byte. The lower nibble should be used as an index in the half period array (a) to fetch a 16 bit integer, which should be stored in a variable.
- The upper nibble represents the octave. If it is zero, nothing needs to be done. If it is 1, the half period should be halved. If it is 2, the half period should be doubled. If it is > 2, the half period should be made 0xFFFF.
- The program should then display the note_index, duration, octave and half period on the LCD in the following format:
  Line 1: "Note = ??, D=??"
  Line 2: "Oct=? HP= ?? ??" where each '?' is a hex digit.

After this, the main program should increment the note_index and repeat the loop till a 0 value is found for duration.

Test the program with the following values for the arrays:
Halfperiod: 3900, 3681, 3476, 3279, 3096, 2925, 2757, 2600, 2456, 2319, 2189, 2066
Durations: 200, 100, 100, 100, 200, 100, 100, 200, 100, 200, 100, 100, 200, 100, 100, 100, 200, 00 Notes: 0, 2, 4, 5, 7, 9, 11, 16, 64, 16, 11, 9, 7, 5, 4, 2, 0, 0.

Compile and debug the program using the debugger first. Then download to the kit and run it.