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.

Lab Assignments

We are going to modify the homework problem so that the notes are played through the H bridge and speaker, rather than being displayed on the LCD.

For this, we need to write interrupt service routines (ISRs) for timer T0 which generates a square wave with the desired half period.

1. Write an interrupt service routine (ISR) which will be activated when Timer T0 overflows. This routine should use a global enable flag and a global 16 bit integer representing the half period.

   The ISR should should first check the enable flag. If it is set, it should toggle a port pin which will generate the square wave, reload TH0, TL0 with the negative value of the 16 bit integer variable and re-start the timer.

2. The main program is essentially the same as the home work problem.

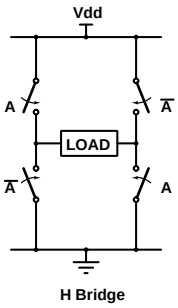
   It should initialize the note_index to 0 and then enter a loop in which

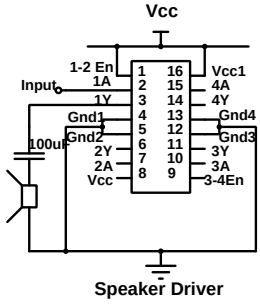
   - it uses note_index to read the duration from the duration array (c). If the duration is 0, the program should terminate. If it is not 0, the duration should be stored in a variable.
   - The same note_index should be used to fetch a byte from the note sequence array (b). The program should separate the upper and lower nibble of the fetched byte. The lower nibble should be used as an index in the half period array to fetch a 16 bit integer, which should be stored in a temporary 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.
     After this adjustment, the half period value should be stored in the global variable used by the T0 ISR to load TH0 and TL0. If the octave value is > 2, it represents a silence period and so the global enable flag should be cleared, so that T0 stops toggling the port pin. Otherwise, the enable flag should be set and the note starts playing.
   - Having started the note, we start a loop which is set to 5 times the delay value. (This is because duration is specified in units of 10 ms, but T1 will be programmed to 2 ms). In each iteration of this loop, we load Timer 1 with values calculated to generate a 2 ms delay and start it. Then we keep waiting till T1 flag is set. When that happens, we clear the flag, and go to the next iteration of the loop.
   - When this loop terminates, the note_index is incremented and go back to the note_index loop till we read a 0 value for the duration.

   As for the homework problem, 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.

Hardware

The impedance of a speaker is quite low ($< 10\Omega$). A port output cannot provide sufficient current to drive this impedance. So we connect the port pin which provides the square wave to what is called an H bridge. This is just a collection of buffers with enable inputs which can connect the output to supply or ground depending on the current input. The equivalent circuit of the H bridge is shown in the figure on the left below. It is called an H bridge because of the shape of the circuit. The data sheet of the H bridge we shall be using has been uploaded to your moodle page. However, for our speaker driver, we need not use the whole H bridge. We can use the simpler circuit shown below on the right.





Notice that the IC used by us has 4 such half bridges which can provide 2 independent H bridges. Connect up the circuit on the bread board (using 5V for Vcc) and connect your port pin output to the input.

When you run your program, you will hear a bar of music. You can change the duration and index arrays to play any sequence of notes. The mapping of notes to index values is shown below:

index	note	index	note	index	note	index	note	index	note
0	sa	1	komal re	2	re	3	komal ga	4	ga
5	ma	6	tivra ma	7	pa	8	komal dha	9	dha
10	komal ni	11	ni						

This is contained in the lower nibble. The upper nibble is 0 for the main octave, 1 for the upper octave, 2 for the lower octave and 3 (or more) for silence.
Durations are specified as multiples of 10 ms in the duration array. You end the sequence by giving a 0 value for duration.

For the musically oriented

You can implement a "glide" by using a key > 3 in the note array. When the main program sees this key, it should read ahead to the next note and divide the duration into equal intervals over which the half period is changed continuously from one note to the next.