Lab-3: Timers and Note Generation

In this set of experiments, we learn how to use built-in hardware timers in 8051 to generate delays without tying up the processor. We then make a musical note generator using these timers.

Before attempting these exercises, please go through the notes on timers and interrupts uploaded to the moodle site.

Homework

1. Write a subroutine which will use a 16 bit value stored at 81H/82H in the indirectly addressable memory to program the timer T0 in order to generate a delay proportional to this count.

   Recall that the 8051 timers count *up*. These generate an interrupt (if enabled to do so) when the count wraps around from 0FFFFH to 0000. If we want a timer to time out after n cycles, it should be loaded with -n (2's complement of n).

   So the subroutine should subtract the stored 16 bit number from 0000H and load the result as the initial count in T0.

   (While debugging the program with single stepping, you could initialize locations 81H/82H with 0001, which results in loading the timer with -1 = 0FFFFH, so that it overflows at the first increment. In actual use, a different count will be stored, of course.)

2. Write a program which will use the above subroutine to blink LEDs (as in Lab-1) such that these are ON for one second and OFF for one second endlessly. Adjust the timer count and the number of times the delay subroutine is called to make the ON and OFF period as close to 1 second as possible. (Measure the time over a large number of cycles to make this adjustment).

Lab Assignments

1. We want to make a programmable note generator. For this, we shall generate square waves whose frequencies are selected from a table. 32 bytes are stored in the program

memory using the DB directive. These represent 16 values, each being 2 bytes wide. (The entries represent candidate values for the half period of a square wave to be generated). We shall program a timer for a duration proportional to this 16 bit value and toggle a port pin at every timeout. Since two toggles will complete 1 cycle of the square wave, the stored value is the half period of the resulting square wave.

The program should enable the user to pick any one of these 16 half periods for generating a square wave. This is done by reading a 4 bit index from the slide switches on the card. This index will select which of the 16 values will be used to set the half period of the square wave.

(In C terminology, we have an array of 16 bit ints, say p[]. We read i from the slide switches, then p[i] is the desired half period.)

After reading the index, it is used for fetching the selected entry. (This is similar to what was done in Lab-1, exercise 2). Please note that each entry in this array is 2 bytes wide and therefore, the offset of the desired entry is 2*i from the start of the array, if i is the index.

The 16 bit value fetched from the array will be used to program the delay of timer T0. Every time the timer times out, an interrupt should be generated. The interrupt service routine should toggle a port bit and re-start the timer. This will generate the desired square wave.

Use the following array to test this program:

1AH, 11H, 24H, 10H, 3CH, 0FH, 61H, 0EH, 93H, 0DH, 0CFH, 0CH, 17H, 0CH, 6AH, 0BH,
0C6H, 0AH, 2BH, 0AH, 99H, 09H, 0FH, 09H, 8DH, 08H, 12H, 08H, 9EH, 07H, 31H, 07H

Note that bytes constituting a 16 bit word have been placed in this array with LSB first and MSB second. (This convention is known as Little Endian and is followed by Intel). These 16 bit values need to be subtracted from 0 before being loaded in the timer, as was explained for the homework problem.

2. Modify the above program to read the index from another array of bytes rather than from switches.

   The top 4 bits of each byte represent a duration in units of 100 milliseconds. The bottom 4 bits provide the index (which was read from the slide switches in the previous exercise).

   Bytes are read sequentially from this additional array. The duration and index are separated out from this byte. Using the index, half period of the square wave should be

fetched from the half period array as in the previous exercise. Timer T0 should now be used to generate a square wave of the given half-period.

Timer T1 should be used to control the duration (in units of 100 ms) for which the square wave is generated. (This can be done by programming T1 for a delay of 100ms using the duration value as the count for a loop in which this delay is called).

At the end of this period, we stop T0 by clearing its run flag and fetch the next byte from the new array. This should continue till a duration value of 0 is read.

Test this with the following array for a sequence of durations and indices into the half period array:

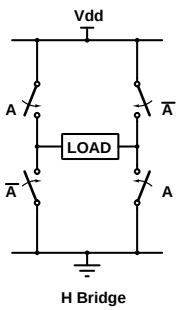
0F2H, 0F4H, 0F6H, 0F7H, 0F9H, 0FBH, 0FDH, 0FEH, 0FEH,
0FEH, 0FDH, 0FBH, 0F9H, 0F7H, 0F6H, 0F4H, 0F2H, 0F2H, 02H

The Fun Part

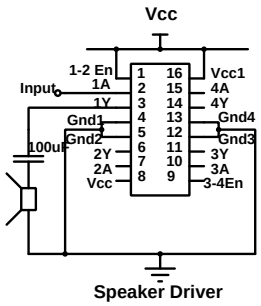
We have made a square wave generator which can be programmed to generate a selected frequency corresponding to a musical note for a selectable duration. This can be used to make a musical note generator!

To hear the results of your efforts, you need to do a bit of work though.

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.



H Bridge



Speaker Driver

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 array to play any sequence of notes. The mapping of notes to index values is shown below:

index	note	index	note	index	note
0	lower komal ni	1	lower ni	2	sa
3	komal re	4	re	5	komal ga
6	ga	7	ma	8	tivra ma
9	pa	10	komal dha	11	dha
12	komal ni	13	ni	14	upper sa
15	upper komal re				

You end the sequence by giving 0 duration (upper nibble = 0).