CS341 - Computer Architecture and Organization Bob Wilson
MP3: Timer Interrupts
Assigned: Class 14 Due: at start of Class 18

The purpose of this assignment is to build a package of timing routines for the Tutor VM and use it to generate a message at a user entered time interval.

Read the "Intel 8254 Programmable Interval Timer Data Sheet" on my website.
Look at $pcex/timer.c for an example of C code for accessing the timer.
Copy all the files needed for this project from ~bobw/cs341/mp3.

1. The timing package.
A package is a set of utilities that can be called by a program that wants to use them. When designing a package we carefully consider the things in it that must be visible to the caller, specify those, and require that everything else be internal and invisible. The visible parts of the timing package you will build are in "tickpack.h" (as function prototypes, symbolic constants, and comments about using the package). A customer for the package's services includes that header file (using #include) and links his or her object code with "tickpack.opc". You are to finish writing "tickpack.c" to provide a timer service for our PC-Tutor.

The tickpack.h file defines four function prototypes. The code in tickpack.c should do the following for each of these functions:

void init_ticks(void); This code should initialize the timer table and set up the timer chip to interrupt about 18 times per second (full count down of 16 bit timer). The timer table is an array of five timer entry structs. Each struct contains a function pointer to a callback function to be executed every time the timer expires, an int for the number of timer interrupts before the first call to the callback function, and an int for the number of timer interrupts for each subsequent call to the callback function. For each struct in the array, the initial values should be NULL for the callback function and 0 for two int values.

int set_timer(IntHandler *function, int time, int reinittime); This code finds the first empty position in the timer table. An empty position would have a NULL value for the callback function. If there are no empty positions left in the table or if there is a duplicate of the function pointer argument already in the table, the function should return a value of false (0). Otherwise, it loads its arguments into the empty struct and returns true (1). The time and reinittime arguments are in seconds, but the timer ISR decrements the time every interrupt and interrupts occur 18 times per second. Therefore, the two int argument values must be multiplied by 18 when moved to the timer entry struct. This will cause the callback function to be executed by the ISR every time its corresponding time value is decremented to zero.
int stop_timer(IntHandler *function); This code looks for the position in the timer table corresponding to the callback function. If it does not find it in the table, it returns false (0). Otherwise, it sets the elements of this struct to their initial values so that this timer will no longer be processed by the ISR

void shutdown_ticks(void); This code should program the PIC to stop the timer interrupts.

There is one more key function that is not a part of the API defined in tickpack.h. This internal ISR is called every time the timer interrupts:

void irq0inthandc(void); This code tells the PIC that we are servicing the interrupt. Then, it looks through the timer table and checks each entry. If there is a non-NULL value for the callback function and its time value is greater than zero, it decrements the time value. If the time value has gone to zero, it executes the callback function and reinitializes the time value from the reinit time value. A reinit time value of zero will cause the callback function to be executed only once when the timer expires the first time. This is a so called “one shot” timer and we’ll use it in mp4.

NOTE: Since we do not have any underlying operating system, our callback functions actually execute at interrupt level. We need to be concerned about “thread safety” (e.g. no “interrupt windows” between the ISR and background code) in our design. Be sure to disable interrupts in the background code where needed. If we were writing this code to run under a real operating system, the callback functions would be tasks scheduled under the OS to be run later in the background.

Every package should have a test program showing that it works by calling it in all important ways. This test program is called a "driver" because it sits on top of the package and drives it like we test-drive a car – start up, do this, do that, stop, shut down. It is also called a "unit test" because it tests just this one package separate from any other package in a larger program. If we suspect something's wrong in a certain package, we try to make its unit test fail, and then we debug the problem in the relatively simple environment of the unit test, rather than in the larger program. The test program for tickpack.c is test_tickpack.c. You can build test_tickpack.lnx right away and run it. However, it will fail until you complete the code in tickpack.c. When you have completed the code in the package you should get a display like this with no error messages:

```
Tutor> ~downloading test_tickpack.lnx
......Done.
Download done, setting EIP to 100100.
Tutor> go 100100
Wait for a while to observe timer printouts.
******
Wait for a while to observe timer printouts have stopped.
Done!
Exception 3 at EIP=00100110: Breakpoint
Tutor>
```

2. PC-Tutor Timer Interrupt Displays

In part two, you’ll integrate your tested tickpack.c code into a version of the PC-Tutor program that you worked with in mp1. The tutor main program calls init_ticks for you, but nothing will happen until a timeon or timeoff command is entered. Your stop command should call shutdownticks() so that interrupts are not left enabled when PC-Tutor returns to Tutor. This code has been provided to you in the cmds.c file. You just need to run it and check that it works with your tickpack code.

When you enter the timeon command with an integer value for the time interval, the PC-Tutor program should print out a running value for the number of timer

interrupts that have occurred since the timeon command was entered. These print outs should occur once every time interval based on the value you entered. Because each printout may take a few seconds, you shouldn't enter time interval values less than about 5 seconds.

Since these printouts are triggered by timer interrupts, the background PC-Tutor program will still be running and waiting for entry of further commands on the COM2: port. Verify that normal PC-Tutor commands such as md and ms work correctly while timer interrupts are printing. Note that if you start entering a command and the timer interrupt occurs, you can just complete entering the command after the timer printout.

The timeoff command stops the printouts.

Sample Display:

```
~downloading tutor.lnx
..........Done.
Download done, setting EIP to 100100.
Tutor> go 100100
cmd help message
timeon Timer On: timeon <interval in secs>
timeoff Timer Off: timeoff
md Memory display: md <addr>
ms Memory set: ms <addr>
h Help: h <cmd> or h (for all)
q Quit
PC-tutor> timeon
Proper usage : Timer On: timeon <interval in secs>
PC-tutor> timeon 5
timer on
PC-tutor> (1)
(2)
(3)
(4)
(5)
PC-tutor> timeoff
timer off
PC-tutor> timeon 5
timer on
PC-tutor> md 100100
00100100 bc f0 ff 3f 00 bd 00 00 00 00 e8 01 00 00 00 cc ...?............
PC-tutor> (1)
md 100100(2)
00100100 bc f0 ff 3f 00 bd 00 00 00 00 e8 01 00 00 00 cc ...?............
PC-tutor> (3)
timeo(4)
ff
timer off
PC-tutor>
```

---

discussion.txt

Write a discussion describing how you tested your code and what interesting things you discovered while doing so. Put your discussion in discussion.txt. Include small portions of scripts showing output caused by the interrupt handlers and the callback functions to support your observations and your conclusions.

Turn-in
In your mp3 directory, capture and print/turn in hard copies of typescript files (one from system building on ulab and one from mtip/tutor interactions on tutor vserver).

The ulab system building typescript file should show:
pwd
ls –l
cat timepack.c
make clean
make test_timepack.lnx
make tutor.lnx
cat discussion.txt

The tutor vserver typescript file should should be uploaded to your ulab mp3 directory and show test runs of test_tickpack.lnx and tutor.lnx.

FINAL NOTE:
In the event that you are unable to correctly complete this assignment by the due date, do not remove the work you were able to accomplish. Submit your report on time – partial credit is always better than none.