

# Laboratory 8

(Due date: Nov. 22<sup>nd</sup>)

## OBJECTIVES

- Learn basic mechanisms for Real-Time Systems: handle signals (setup and detect).
- Configure and test Real-Time Clock.

## REFERENCE MATERIAL

- Refer to the [board website](#) or the [Tutorial: Embedded Intel](#) for User Manuals and Guides.
- Refer to the [Tutorial: High-Performance Embedded Programming with the Intel® Atom™ platform](#) → *Tutorial 8* for associated examples.

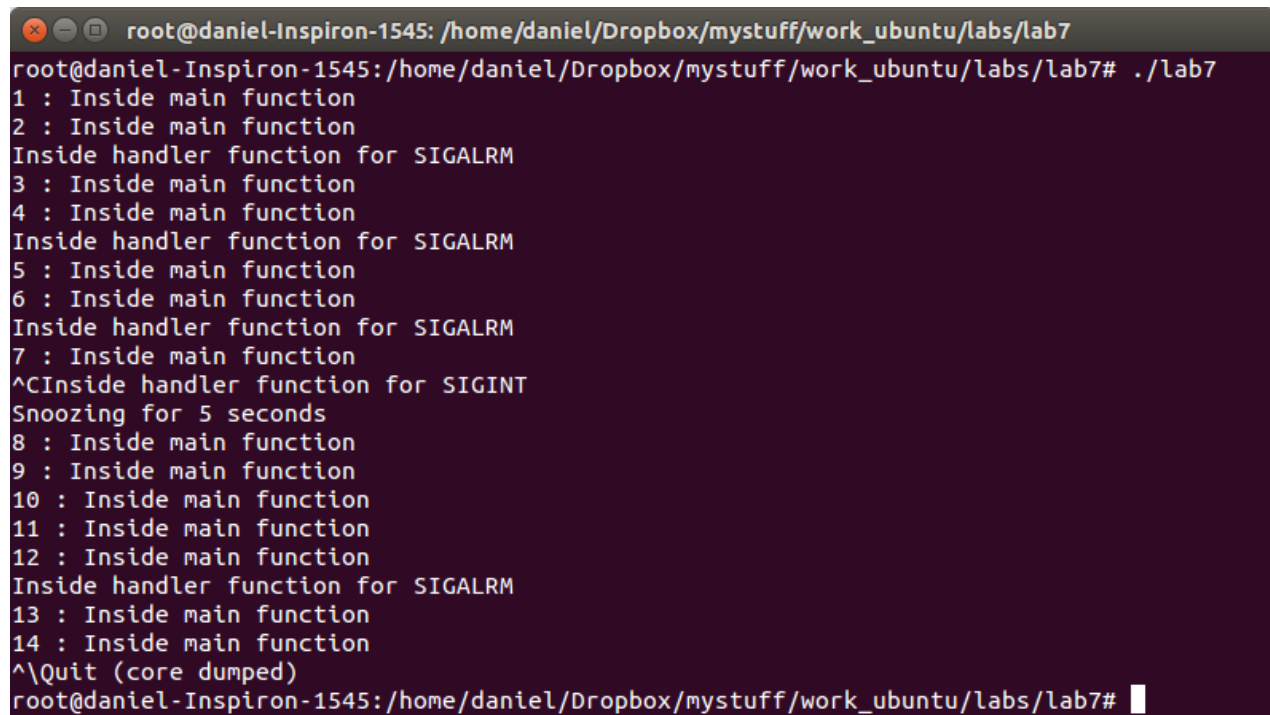
## ACTIVITIES

### FIRST ACTIVITY: HANDLING SIGNALS (60/100)

- In this experiment, you are asked to implement an application (.c) that implements the following:
  - ✓ Prints the message "i: Inside main function" every second. i = 1,2, ...
  - ✓ Every 2 seconds, an alarm (SIGALRM signal) goes off that interrupts the execution of the main function and prints the message "Inside handler function for SIGALRM".
  - ✓ The user should have the ability to snooze the alarm for 5 seconds. This is done via the SIGINT signal (Ctrl-c); here, the message "Inside handler function for SIGINT" is printed.
  - ✓ To exit the program, the user can use the SIGQUIT signal (Ctrl-\\).

### Suggestions

- You need to setup a *handler function* for both the SIGALRM and SIGINT signals.
- You can set up the 2-second alarm before an infinite loop. In order to continuously setup the 2-second alarm, you might want to setup (restart) the 2-second alarm every time the *handler function* for SIGALRM is executed.
- When the user issues the SIGINT signal, you can set up a 5-second alarm in the *handler function* for SIGINT.
- Take a screenshot of the software running on the Terminal. It should show: i) the messages being printed every second, ii) the alarm going off every 2 seconds, and iii) the user generating the SIGINT signal. Fig. 1 shows an execution example.

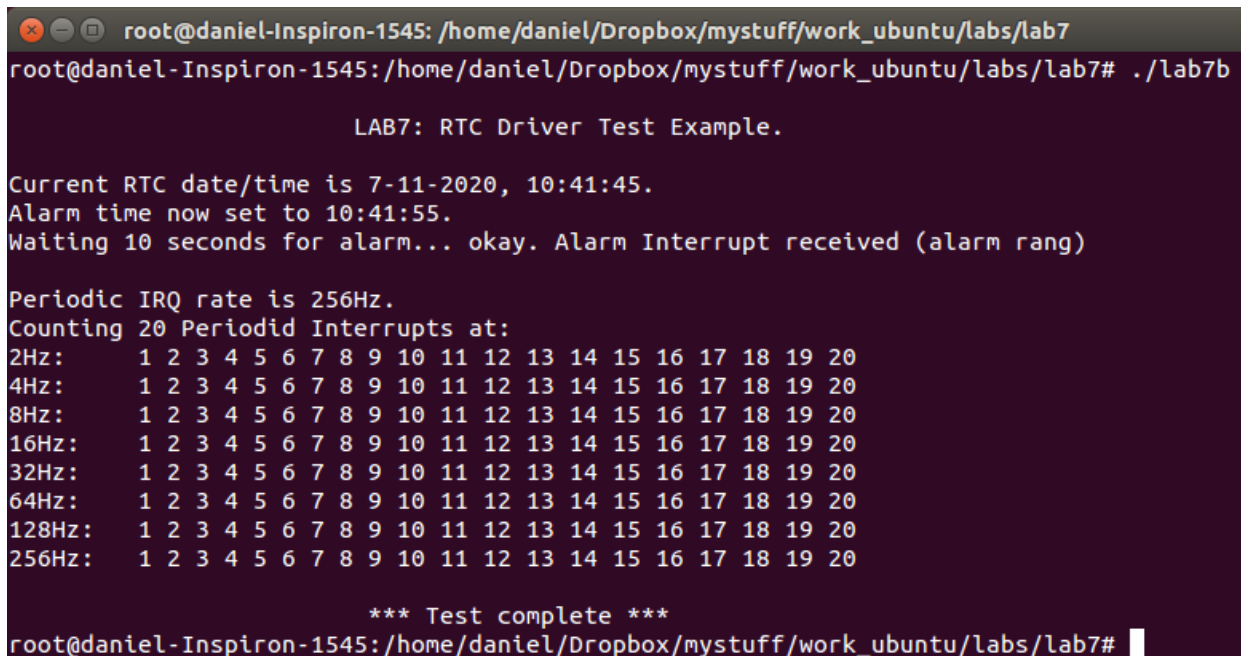


```
root@daniel-Inspiron-1545: /home/daniel/Dropbox/mystuff/work_ubuntu/labs/lab7
root@daniel-Inspiron-1545: /home/daniel/Dropbox/mystuff/work_ubuntu/labs/lab7# ./lab7
1 : Inside main function
2 : Inside main function
Inside handler function for SIGALRM
3 : Inside main function
4 : Inside main function
Inside handler function for SIGALRM
5 : Inside main function
6 : Inside main function
Inside handler function for SIGALRM
7 : Inside main function
^CInside handler function for SIGINT
Snoozing for 5 seconds
8 : Inside main function
9 : Inside main function
10 : Inside main function
11 : Inside main function
12 : Inside main function
Inside handler function for SIGALRM
13 : Inside main function
14 : Inside main function
^\\Quit (core dumped)
root@daniel-Inspiron-1545: /home/daniel/Dropbox/mystuff/work_ubuntu/labs/lab7#
```

Figure 1. Sample execution for the application in the First Activity.

## SECOND ACTIVITY: REAL-TIME CLOCK CONFIGURATION (40/100)

- In this experiment, you are asked to implement an application that:
  - ✓ Read data (current date/time) from RTC.
  - ✓ Configure and test the *Alarm Interrupt*.
  - ✓ Configure and test *Periodic Interrupts*.
- You need to use the RTC driver template (`rtctst.c`) available from the [Tutorial: High-Performance Embedded Programming with the Intel® Atom™ platform](#) → *Tutorial 8*. Refer to this tutorial for a detailed explanation of the code.
- You are asked to perform the following (these are minor modifications to `rtctst.c`):
  - ✓ Read the RTC time/date. Print it in format `mm-dd-yy, hours:minutes:seconds`.
  - ✓ Configure and test *Alarm Interrupt*:
    - Set the Alarm Interrupt to **10** seconds in the future.
    - Read current alarm settings. Print the time the alarm is set to go off: `hours:minutes:seconds`.
    - Enable Alarm Interrupts.
    - Wait until Alarm Interrupt comes by executing a blocking `read()` on RTC.
    - Disable Alarm Interrupts.
  - ✓ Configure and test *Periodic Interrupts*:
    - Read periodic IRQ rate (it will print the last one it has been used)
    - For a set of periodic interrupts (from 2 to 256 Hz, only powers of 2), do: set the frequency, enable period interrupts, detect 20 interrupt of a given frequency, and disable periodic interrupts.
      - Set frequency (2, 4, 8, 16, 32, 64, 128, 256)
      - Enable Periodic Interrupts
      - For a given frequency, wait for 20 periodic interrupts; use a blocking `read()` to detect each.
      - Disable Periodic Interrupts
- Note that you need be `root` to execute this code (use `sudo -i`).
- Take a screenshot of the software running on the Terminal. It should show (Fig. 2 shows an execution example)
  - ✓ The current RTC data/time,
  - ✓ The time the alarm is set to go off, and when the alarm is detected.
  - ✓ The current Periodic Interrupt rate, and the detection of 20 periodic interrupts for each frequency (2, 4, 6, 16, 32, 64, 128, 256).



```
root@daniel-Inspiron-1545: /home/daniel/Dropbox/mystuff/work_ubuntu/labs/lab7
root@daniel-Inspiron-1545:/home/daniel/Dropbox/mystuff/work_ubuntu/labs/lab7# ./lab7b

LAB7: RTC Driver Test Example.

Current RTC date/time is 7-11-2020, 10:41:45.
Alarm time now set to 10:41:55.
Waiting 10 seconds for alarm...okay. Alarm Interrupt received (alarm rang)

Periodic IRQ rate is 256Hz.
Counting 20 Periodic Interrupts at:
2Hz:    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
4Hz:    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
8Hz:    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
16Hz:   1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
32Hz:   1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
64Hz:   1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
128Hz:  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
256Hz:  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

*** Test complete ***
root@daniel-Inspiron-1545:/home/daniel/Dropbox/mystuff/work_ubuntu/labs/lab7#
```

Figure 2. Sample execution for the application in the Second Activity.

## SUBMISSION

- Demonstration: In this Lab 8, the requested screenshot of the software routine running in the Terminal suffices.
  - ✓ If you prefer, you can request a virtual session (Zoom) with the instructor and demo it (using a camera).
- Submit to Moodle (an assignment will be created):
  - ✓ Two .zip files (one for the 1<sup>st</sup> Activity and one for the 2<sup>nd</sup> Activity).
    - 1<sup>st</sup> Activity: The .zip file must contain the source files (.c, .h, Makefile) and the requested screenshot.
    - 2<sup>nd</sup> Activity: The .zip file must contain the source files (.c, .h, Makefile) and the requested screenshot.

TA signature: \_\_\_\_\_

Date: \_\_\_\_\_