# PP1 - 2020

## 1. The L3223

In this programming test you will have to develop a Minix privileged program that interfaces with a device, the **L3223**, **similar** to the i8254, the timer/counter used in Lab 2. However, this device has different features. **Please read carefully the description of the timer/counter, to ensure that your implementation is according to its specification in this test.**

The L3223 device has three identical 16-bit timers. It has a one 8-bit register per timer at ports 0x20 to 0x22. Port 0x23 is used to access both a control register and a status register, both with 8 bits. Each timer operates independently, depending on its configuration. A timer can operate in one of two modes: periodic, which is similar to i8254's mode 3 that was used in Lab 2, and alarm. In the alarm mode, the L3223 generates one interrupt after a programmed interval. The L3223 can use 3 time units to specify either the period or the time interval: 1 microsecond, 1 millisecond and 1 second. Like the i8254, the period or the alarm interval, i.e. the initial timer value, is loaded via one 8-bit port, one per timer. The initial value must always be in binary, and the least significant byte (LSB) must be loaded before the most significant byte (MSB).

Writing to port 0x23 writes the control register, whose relevant bits are:

| **Bit** | 7 | 6 | 5 | **4** | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| **CTRL. REG.** | Timer-MSbit | Timer-LSbit | - | **Alarm** | - | - | Unit-MSbit | Unit-LSbit |

**bits 7 and 6**

the `Timer Selection` bits, which select the timer to configure. The possible values are:
**00** for selecting timer 0
**01** for selecting timer 1
**10** for selecting timer 2

**bit 4**

the `Alarm` bit, which sets the mode of operation:
**0** for periodic mode
**1** for alarm mode

**bits 1 and 0**

the `Time Units` bits, which select the unit of the initialization value. The possible values are:
**00** for selecting timer microsecond
**01** for selecting timer millisecond
**10** for selecting timer second

According to the documentation of the L3223, the remaining bits of the control register should be set to 0 for future compatibility.

The configuration of a timer follows a protocol similar to that used in Lab 2:

1. Write the control word to the control register
2. Write the period or the interval to the selected timer register: first the LSB and then the MSB.

Reading from port 0x23 reads the status register, whose relevant bits are:

| **Bit** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| **STA. REG.** | - | - | - | - | - | TIMER2-INT | TIMER1-INT | TIMER0-INT |

**bit 2**

the TIMER2-INT bit, which if set to 1 indicates a pending interrupt from Timer 2

**bit 1**

the TIMER1-INT bit, which if set to 1 indicates a pending interrupt from Timer 1

**bit 0**

the TIMER0-INT bit, which if set to 1 indicates a pending interrupt from Timer 0

# 2. The Problem

## 2.1. The function to develop

In this test, you must develop the following function:

```
int pp_test_alarm(int timer, int interval, enum l3223_time_units unit);
```

which must configure the timer specified in its first argument, to generate an alarm, i.e, an interrupt, after an interval specified in the second argument, whose time unit is specified in its third argument. After that time interval, it should invoke the following function that we provide you:

```
int pp_print_alarm(int timer, int interval, enum l3223_time_units
unit);
```

where `enum l3223_time_units unit` is the following C enumeration type:

```
enum l3223_time_units {
    l3223_microsecond,
    l3223_millisecond,
    l3223_second
};
```

Use interrupts with the `IRQ_REENABLE` policy only. The L3223 uses IRQ line 10.

**IMP**: you should develop your code in the `/home/exame/pp/` directory that was created by the setup.sh script. The pp.c file in that directory already includes the main() function, which you must not change, and a skeleton for `pp_test_alarm()`, which you are supposed to complete.

# 2.2. Building, running and testing

In order to build your program you should use the Makefile that is available in `/home/lcom/labs/pp/`, by executing in that directory:

```
minix$ make depend && make
```

**Hint**: Remember that you can always login remotely on Minix by running the following command on a terminal (in Linux):

```
$ ssh lcom@localhost -p 2222
```

and use the newly created shell to run (and build) your program.

To learn how to run your program, you should use the command `lcom_run` and specify no program arguments:

```
minix $ lcom_run pp
Usage:
    lcom_run pp "alarm <timer - one of 0, 1 or 2> <interval - decimal>
<unit - one of u, m or s> -t <test no.>"
    Use 'u' for microseconds, 'm' for milliseconds and 's' for seconds
```

It will output a usage message that describes the command line arguments that you can use. The `<test no.>` argument is an integer between 1 and 3. For all test no. 's, the behavior is deterministic, therefore you should get always the same results for the same set of arguments The following table summarizes each test case:

| Test No. | Test description | Guaranteed score |
|---|---|---|
| 1 | Tests timer control word only. (So that we can give partial grade: this will be worth 40%.) | 25% |
| 2 | Tests timer configuration. (I.e. configuring also the initial value. This is worth 10%.) | 15% |
| 3 | Tests interrupts. | 30% |

If your program terminates normally, it will print either:

```
Test succeeded.
```

indicating success, or

```
Test FAILED!
```

indicating failure. In case of failure, you should examine carefully the output on the terminal or in the output.txt file and the trace with the calls your program performed in the trace.txt file. (Both text files are in the directory where you executed the `lcom_run` command.)