

Fontys University of Applied Sciences

Software & Technology - 6th Semester
ES

Assignment 1: **Peek & Poke**

Elviro Pereira Junior, **2719584**

Rafal Grasman, **289290**

Part 1:

On the first part of the assignment we're asked to write a userspace program which reads the RTC values from it's address and execute it on a lubuntu virtual machine and on the LPC3250 board. The goal of the assignment is to observe it's behaviour on both targets.

Research Results:

To start the assignment, we first looked into the datasheet of the LPC3250 board to find the address and some of the characteristics of the RTC Up Counter.

The RTC UC address is (hex) 0x4002 4000 (page 34 of the DataSheet-UM10326). The RTC has a frequency of 1 Hz (page 24) (but is driven by a 32768 Hz oscillator so we can calculate the error if needed) and uses a 32-bit register to store the increments of the clock, which should take 136 years to overflow.

Design Decisions:

To get the value of the RTC UC, the address 0x4002 4000 will need to be read (4 bytes). This means a `uint32_t` should suffice:

```
volatile uint32_t* rtc_upcounter = (uint32_t*) 0x40024000;
```

Because we are reading memory that can be externally modified, we need to tell the C compiler that the memory is volatile, which will mean the memory won't be cached - the result will be the actual contents of the memory every time it's dereferenced. Because the memory sizes are pre-defined, explicit definition of the memory layout is required (int cannot be used as the size can be either 64 bits, 32 bits or even 16 bits, the C standard specifies a minimum of 16 bits for int).

Testing:

To be able to test if the address corresponds to the RTC clock we will need to read the value, then, after 10 seconds, read the value again and it should be incremented by about 10 (+/- 1).

This will result in the following code:

```
uint32_t a = *rtc_upcounter;
usleep(10* 1000 * 1000); // sleep 1 second (10 million microseconds)
uint32_t b = *rtc_upcounter;
printf("a: %d, b: %d, difference(b-a): %d", a, b, b - a);
```

The "difference" is expected to be around 10. This can be off because of scheduling (e.g. the `usleep` call will wake up too soon or too late) and small errors in the RTC (can be off by 0.03 ms).

The full code that will be compiled:

```
#include <stdio.h>
#include <unistd.h>
#include <stdint.h>

int main()
{
    volatile uint32_t* rtc_upcounter = (uint32_t*) 0x40024000;

    uint32_t a = *rtc_upcounter;
    usleep(10* 1000 * 1000); // sleep 10 seconds (10 million microseconds)
    uint32_t b = *rtc_upcounter;

    printf("a: %d, b: %d, difference(b-a): %d", a, b, b - a);

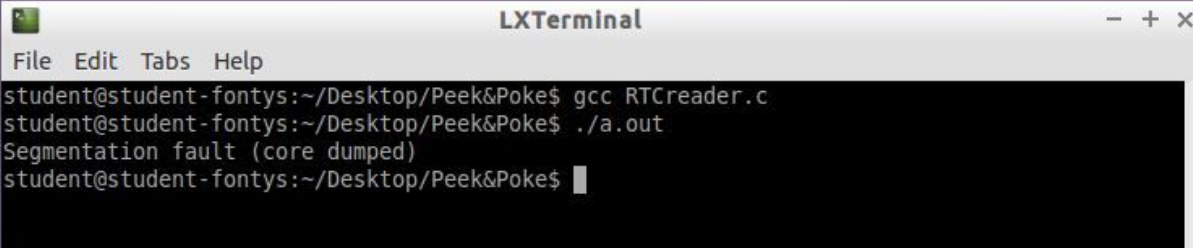
    return 0;
}
```

Running this code (in userspace) according to the assignment should result in the RTC up counter values being printed, but running this code in userspace we expect this code to either cause a segfault or produce garbage. This is because on other machines this memory is (usually) not mapped and not accessible. Also, userspace applications (in linux and windows) cannot access hardware registers.

Side Note:

It can be noticed the LPC3250 board does not have /dev/mem, this would allow to read/write physical memory. The code in testing and design decision will always fail because the linux user space programs, and linux kernel too (albeit in a different way), use virtual memory, this means that even if you know the physical address, if you use it, it will not correspond to the same physical address you want (because it's virtual mapped to some other physical region). You need to get to the physical memory in some other way (e.g. /dev/mem but the LPC3250 linux doesn't have this), in the kernel this can be done with the function 'ioremap', which remaps virtual memory (the return value) to physical memory (the input value).

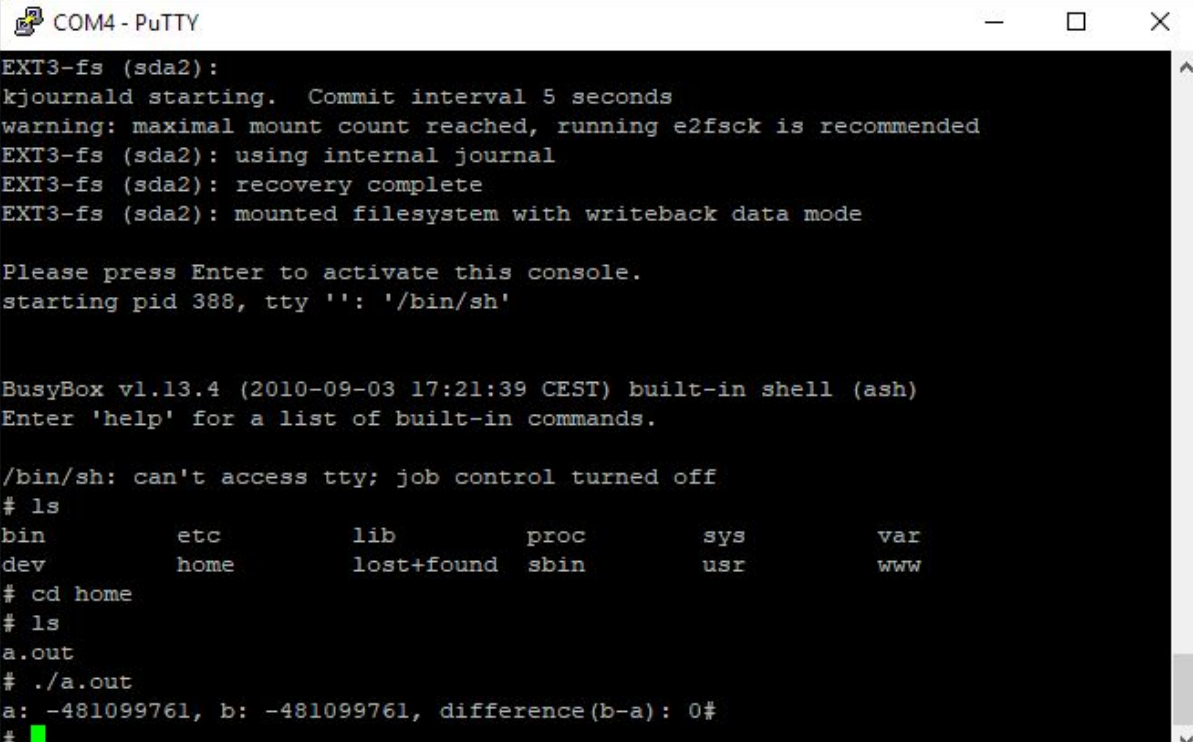
Results:



```
LXTerminal
File Edit Tabs Help
student@student-fontys:~/Desktop/Peek&Poke$ gcc RTCTreader.c
student@student-fontys:~/Desktop/Peek&Poke$ ./a.out
Segmentation fault (core dumped)
student@student-fontys:~/Desktop/Peek&Poke$
```

Target - PC (lubuntu vm)

On the lubuntu vm we get a segfault, this is expected as described in testing.



```
COM4 - PuTTY
EXT3-fs (sda2):
kjournald starting. Commit interval 5 seconds
warning: maximal mount count reached, running e2fsck is recommended
EXT3-fs (sda2): using internal journal
EXT3-fs (sda2): recovery complete
EXT3-fs (sda2): mounted filesystem with writeback data mode

Please press Enter to activate this console.
starting pid 388, tty '': '/bin/sh'

BusyBox v1.13.4 (2010-09-03 17:21:39 CEST) built-in shell (ash)
Enter 'help' for a list of built-in commands.

/bin/sh: can't access tty; job control turned off
# ls
bin          etc          lib          proc         sys          var
dev          home        lost+found  sbin        usr          www
# cd home
# ls
a.out
# ./a.out
a: -481099761, b: -481099761, difference(b-a): 0#
#
```

Target - LPC3250

Part 2:

The second part of the assignment request's us to build a kernel module which can access and modify any hardware register.

Research Results:

Running the code in kernel space on the LPC3250 board (because the RTC up counter address is board-specific) should work and produce the value of the RTC UC. Before that however, the code needs to be updated to map a physical address to a virtual address, like this:

```
volatile uint32_t* rtc_upcounter = (uint32_t*) ioremap(0x40024000, 4);
```

The step above is required because there are certain addresses which are intercepted by the motherboard between the RAM and CPU and then redirected to other hardware, the `ioremap` function, is a kernel function which allows access to hardware through a mechanism called I/O mapped memory.

Once the value of the register is mapped from the virtual memory we can now read or write to same position. Once we no longer need the memory to be accessible to us, we need to unmap I/O memory from the virtual kernel address space of the register once again using the following function:

```
iounmap((void*)address);
```

In order to have a module that can interact/communicate with user-space and change any hardware register we had to create a directory and file within the system filesystem so that we could call it from the userspace and have the results (the file being `/sys/kernel/es6/hw`). This directory and file are created in the system filesystem when the module is launched and are removed when the module is closed.

To be able to read data from the registers it is not recommended to use pointers/dereferencing. Instead the use of specialised functions called `ioread32` and `iowrite32` is recommended to read and write 32 bit registers.

Design Decisions:

A simple protocol is implemented where writing "`r <hexadecimal address> <amount (in hex)>`" to `/sys/kernel/es6/hw` will result in `<amount>` of registers to be read starting at address `<hexadecimal address>`. For example, to read the RTC Uptime register:

```
echo "r 40024000 1" > /sys/kernel/es6/hw
cat /sys/kernel/es6/hw
```

Will output, for example:

```
11cb
```

To write a value (e.g. decimal 26) to a register (at address `0x44556677`):

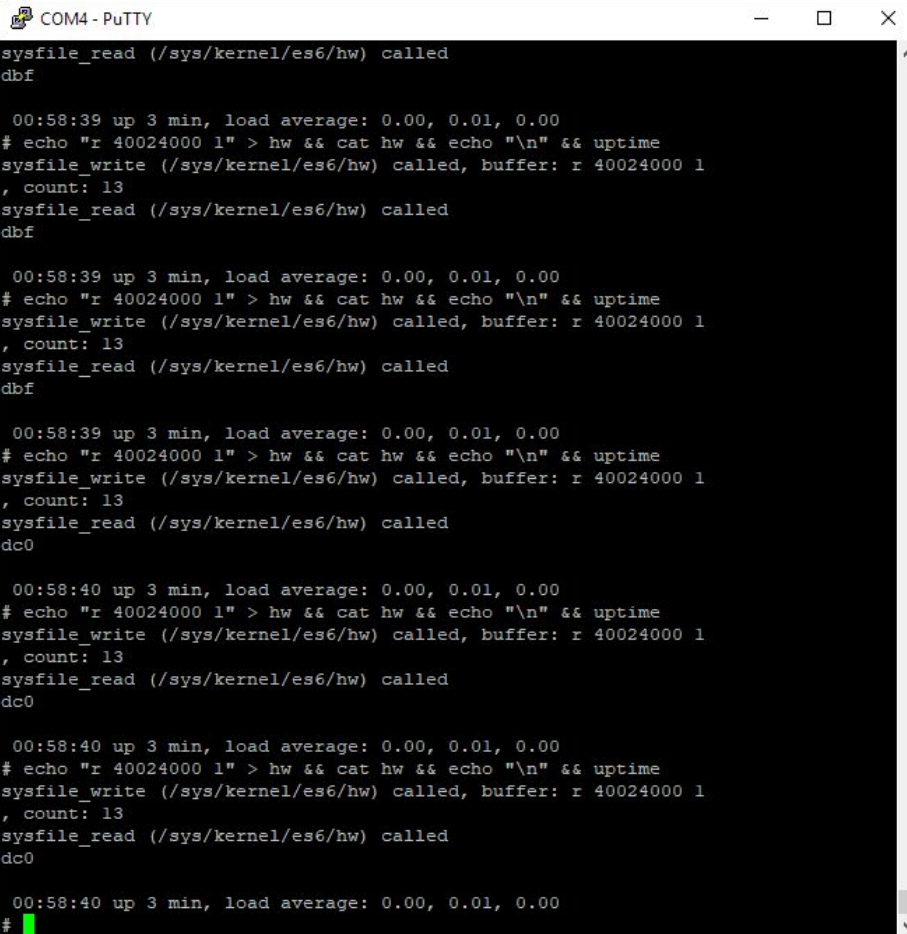
```
echo "w 44556677 1a" > /sys/kernel/es6/hw
```

The format is: "`w <hexadecimal address> <hexadecimal value>`"

Debugging messages can be enabled by doing `'dmesg -n 7'` and then (after operations) `'dmesg'`.

Testing:

To test the read function, the RTC Uptime clock (@40024000) is read continually, and checked against the system uptime:



```
COM4 - PuTTY
sysfile_read (/sys/kernel/es6/hw) called
dbf

00:58:39 up 3 min, load average: 0.00, 0.01, 0.00
# echo "r 40024000 1" > hw && cat hw && echo "\n" && uptime
sysfile_write (/sys/kernel/es6/hw) called, buffer: r 40024000 1
, count: 13
sysfile_read (/sys/kernel/es6/hw) called
dbf

00:58:39 up 3 min, load average: 0.00, 0.01, 0.00
# echo "r 40024000 1" > hw && cat hw && echo "\n" && uptime
sysfile_write (/sys/kernel/es6/hw) called, buffer: r 40024000 1
, count: 13
sysfile_read (/sys/kernel/es6/hw) called
dbf

00:58:39 up 3 min, load average: 0.00, 0.01, 0.00
# echo "r 40024000 1" > hw && cat hw && echo "\n" && uptime
sysfile_write (/sys/kernel/es6/hw) called, buffer: r 40024000 1
, count: 13
sysfile_read (/sys/kernel/es6/hw) called
dc0

00:58:40 up 3 min, load average: 0.00, 0.01, 0.00
# echo "r 40024000 1" > hw && cat hw && echo "\n" && uptime
sysfile_write (/sys/kernel/es6/hw) called, buffer: r 40024000 1
, count: 13
sysfile_read (/sys/kernel/es6/hw) called
dc0

00:58:40 up 3 min, load average: 0.00, 0.01, 0.00
# echo "r 40024000 1" > hw && cat hw && echo "\n" && uptime
sysfile_write (/sys/kernel/es6/hw) called, buffer: r 40024000 1
, count: 13
sysfile_read (/sys/kernel/es6/hw) called
dc0

00:58:40 up 3 min, load average: 0.00, 0.01, 0.00
#
```

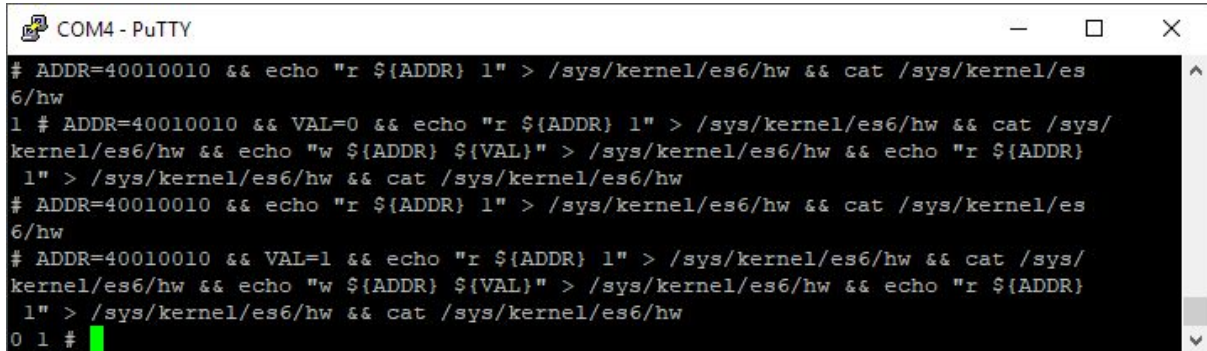
The RTC uptime increments by 1 each uptime second, which means we correctly read the register.

To test the write a read will be performed to GPIO Activation Type Register (@40010010), then a write (0), then a read again, then a write again (1), then a read again, this can be done quickly two one-liners:

```
ADDR=40010010 &&
VAL=0 &&
echo "r ${ADDR} 1" > /sys/kernel/es6/hw &&
cat /sys/kernel/es6/hw &&
echo "w ${ADDR} ${VAL}" > /sys/kernel/es6/hw &&
echo "r ${ADDR} 1" > /sys/kernel/es6/hw &&
cat /sys/kernel/es6/hw
```

```
ADDR=40010010 &&
VAL=1 &&
echo "r ${ADDR} 1" > /sys/kernel/es6/hw &&
```

```
cat /sys/kernel/es6/hw &&  
echo "w ${ADDR} ${VAL}" > /sys/kernel/es6/hw &&  
echo "r ${ADDR} 1" > /sys/kernel/es6/hw &&  
cat /sys/kernel/es6/hw
```



```
COM4 - PuTTY  
# ADDR=40010010 && echo "r ${ADDR} 1" > /sys/kernel/es6/hw && cat /sys/kernel/es  
6/hw  
1 # ADDR=40010010 && VAL=0 && echo "r ${ADDR} 1" > /sys/kernel/es6/hw && cat /sys/  
kernel/es6/hw && echo "w ${ADDR} ${VAL}" > /sys/kernel/es6/hw && echo "r ${ADDR}  
1" > /sys/kernel/es6/hw && cat /sys/kernel/es6/hw  
# ADDR=40010010 && echo "r ${ADDR} 1" > /sys/kernel/es6/hw && cat /sys/kernel/es  
6/hw  
# ADDR=40010010 && VAL=1 && echo "r ${ADDR} 1" > /sys/kernel/es6/hw && cat /sys/  
kernel/es6/hw && echo "w ${ADDR} ${VAL}" > /sys/kernel/es6/hw && echo "r ${ADDR}  
1" > /sys/kernel/es6/hw && cat /sys/kernel/es6/hw  
0 1 #
```

The value changed, which means register writing works.