Peek and Poke

Tim Ackermans & Dominic Voets

Opdracht 1

Wij hebben onderstaande code eerst gebruikt op onze eigen linux versie. Dit gaf onderstaande resultaat.

```c
#define RTC_UCOUNT  0x40024000

int main(int argc, char const *argv[])
{
    int* upcount = RTC_UCOUNT;

    while(1)
    {
        printf("%d\n", *upcount);
        if(getchar() == 'x')
            break;
    }

    /* code */
    return 0;
}
```

```
student@student-fontys:~/Github/S6T/ES/PeekPoke$ ./peek
Segmentation fault (core dumped)
```

Als we dezelfde code uitvoeren op de LPC3250 krijgen we onderstaande resultaat:

```
# ./home/upload/peek
-481099761

-481099761

-481099761

-481099761

-481099761
```

Wij hadden verwacht een resultaat te krijgen dat steeds van waarde veranderd aangezien het om een teller gaat. Wij hebben het resultaat vergeleken met andere studenten en zij hadden hetzelfde getal als uitkomst. Ons vermoeden is dat dit komt doordat het programma geen rechten heeft om het register uit te lezen.

Opdracht 2

```
reading from 40024000
addres: f4024000, value: 4992
# echo "r 40024000 1" > /sys/kernel/ES6/hw
sysfile_write (/sys/kernel/ES6/hw) called
buffer: r 40024000 1

reading from 40024000
addres: f4024000, value: 4993
# echo "r 40024000 1" > /sys/kernel/ES6/hw
sysfile_write (/sys/kernel/ES6/hw) called
buffer: r 40024000 1

reading from 40024000
addres: f4024000, value: 4994
# echo "r 40024000 1" > /sys/kernel/ES6/hw
sysfile_write (/sys/kernel/ES6/hw) called
buffer: r 40024000 1

reading from 40024000
addres: f4024000, value: 4994
```

Dit zijn de waardes die wij hebben uitgelezen bij de RTC up counter, we hebben hier de waarde een aantal keer achter elkaar opgevraagd en er is duidelijk te zien dat de waarde steeds groter wordt. Dit is bewijs dat het lezen van registers werkt.

Nadat het leesgedeelte van de opdracht gemaakt was zijn wij begonnen met het schrijfgedeelte. Om er zeker van te zijn dat we niet in een of ander belangrijk register begonnen te schrijven hebben we gekozen voor het register dat in de opdracht stond geschreven. Dit hebben wij getest door de volgende stap:

```
$ echo "w 400a8014 3ff" > /sys/kernel/ES6/hw
```

Gevolgd door

```
$ echo "r 400a8014 1" > /sys/kernel/ES6/hw
```

Krijgen wij de waarde:

```
reading from 400a8014
addres: f40a8014, value: 3ff
```

Daarna hebben we met andere waardes getest met het volgende resultaat:

```
$ echo "w 400a8014 4ff" > /sys/kernel/ES6/hw
```

Gevolgd door

```
$ echo "r 400a8014 1" > /sys/kernel/ES6/hw
```

Krijgen wij de waarde:

```
reading from 400a8014
addres: f40a8014, value: ff
```

Daarna hebben we

$ echo "w 400a8014 5ff" > /sys/kernel/ES6/hw

Gevolgd door

$ echo "r 400a8014 1" > /sys/kernel/ES6/hw

Krijgen wij de waarde:

```
reading from 400a8014
addres: f40a8014, value: 1ff
```

Eerst vonden wij deze uitkomst erg raar. Maar nadat we logisch zijn gaan nadenken kwamen we erop uit dat het register wel eens maximaal 10 bits zou kunnen zijn. En Freddy kennende was dit ook zo! Hiermee was het duidelijk waar het probleem zat. Zie onderstaand plaatje:

23.5.7 I2Cn Slave Address (I2Cn_ADR - 0x400A 0014, 0x400A 8014)

The I2Cn_ADR register holds the I2C bus slave address.

Table 510. I2Cn Slave Address (I2Cn_ADR - 0x400A 0014, 0x400A 8014)

I2Cn_ADR	Function	Description	Reset value
9:0	ADR	ADR is the I2C bus slave address. Bits 6:0 are enabled if the I2C is configured for slave mode. Bits 9:7 are enabled only if it is configured for slave mode and '10-bit addressing'. Note: A soft-reset disables Bits 9:7, see the description of Bits 8 and 9 in the I2Cn_CTRL registers.	0x06E

Als laatste test hebben wij nog de waarde 35 geschreven om te controleren of waarden onder de 10 bits wel altijd werkten:

$ echo "w 400a8014 35" > /sys/kernel/ES6/hw

Gevolgd door

$ echo "r 400a8014 1" > /sys/kernel/ES6/hw

```
reading from 400a8014
addres: f40a8014, value: 35
```

Dit is bewijs dat het schrijven van registers werkt.