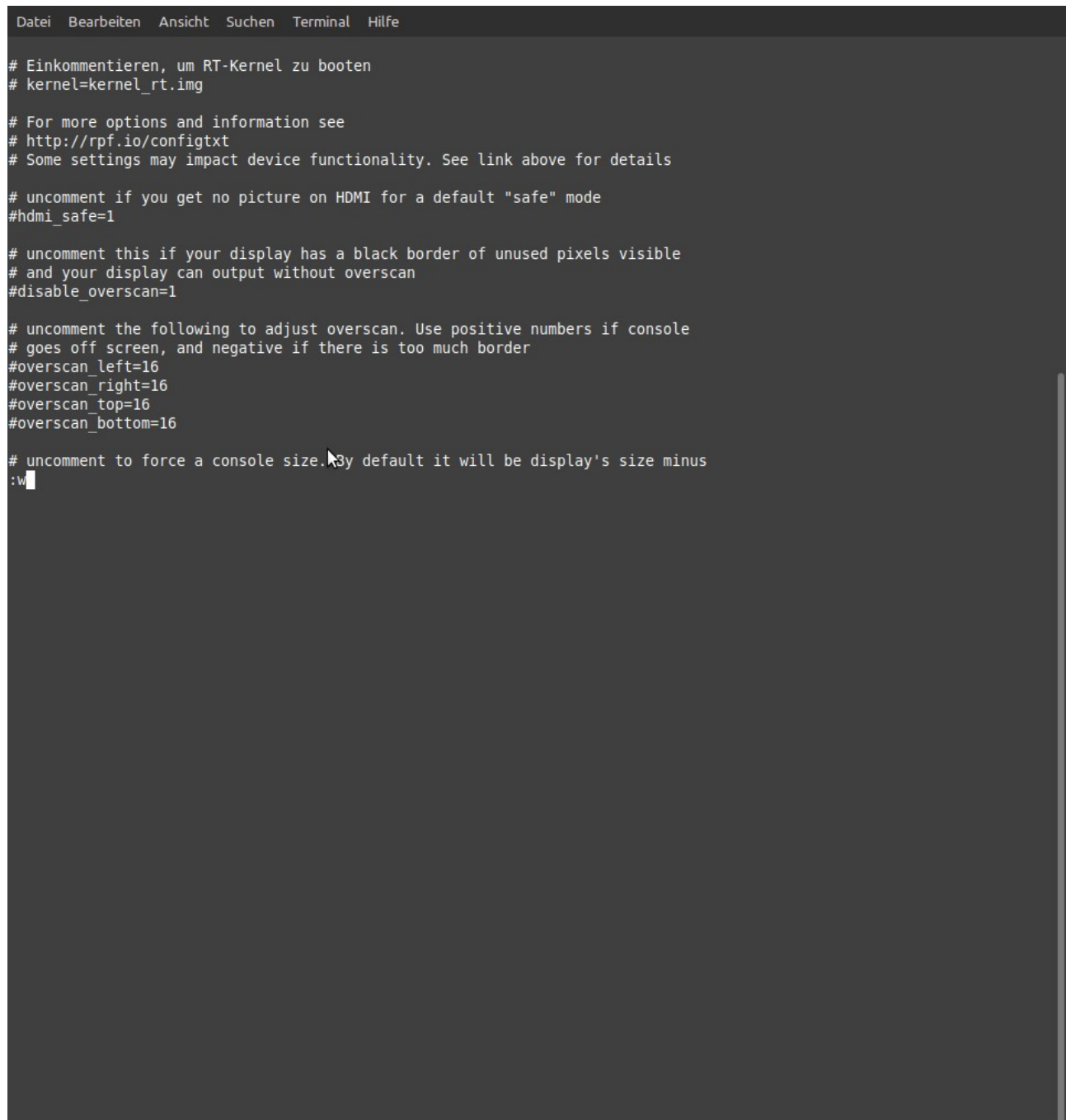


Georgios Markou EZV3

1)



```
Datei Bearbeiten Ansicht Suchen Terminal Hilfe

# Einkommentieren, um RT-Kernel zu booten
# kernel=kernel_rt.img

# For more options and information see
# http://rpf.io/configtxt
# Some settings may impact device functionality. See link above for details

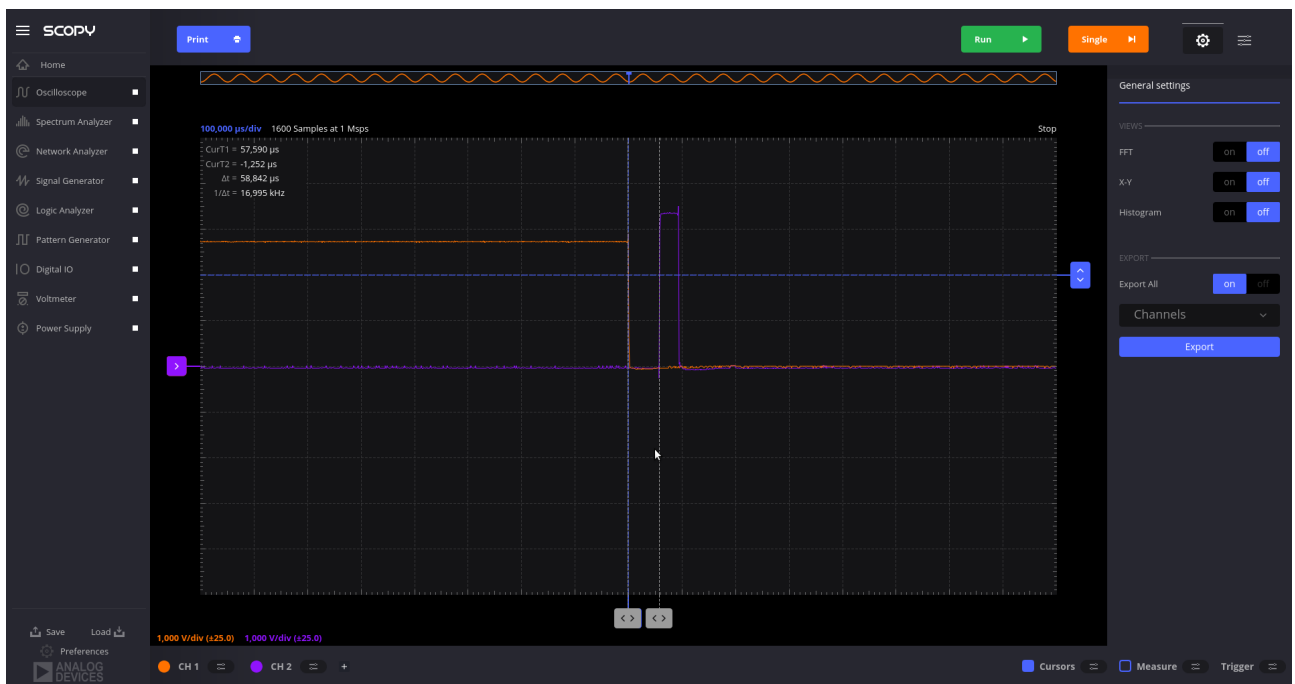
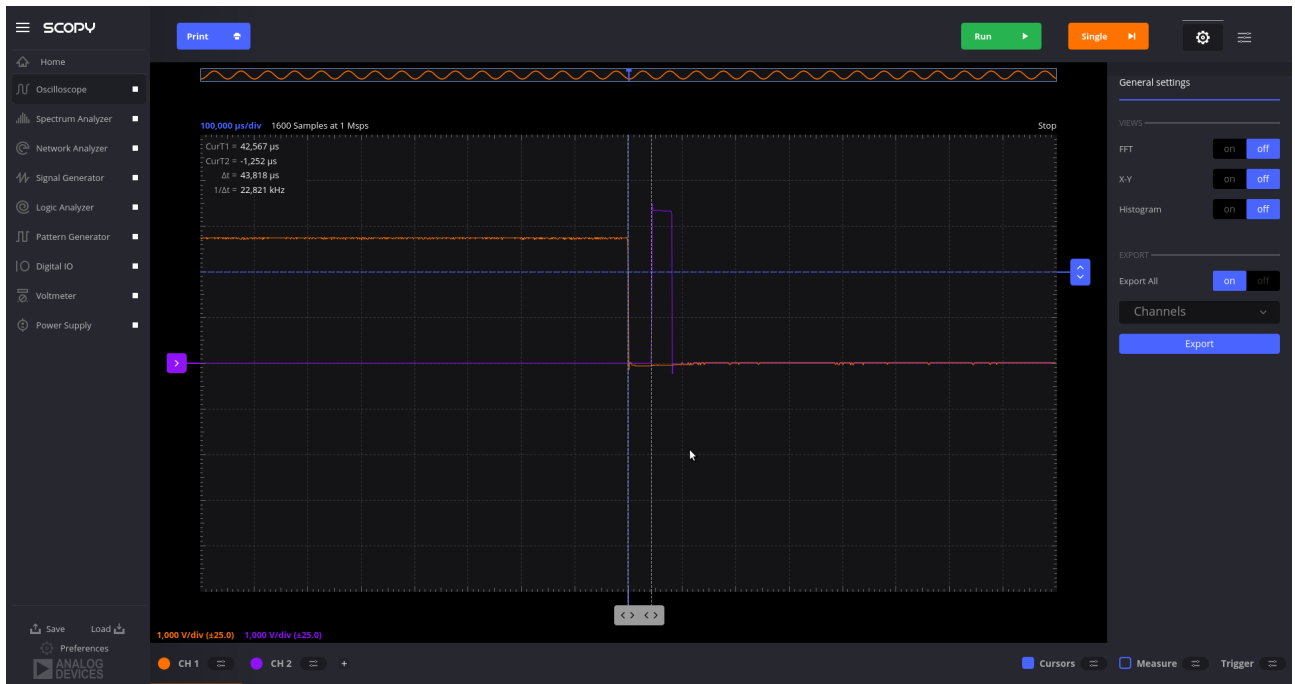
# uncomment if you get no picture on HDMI for a default "safe" mode
#hdmi_safe=1

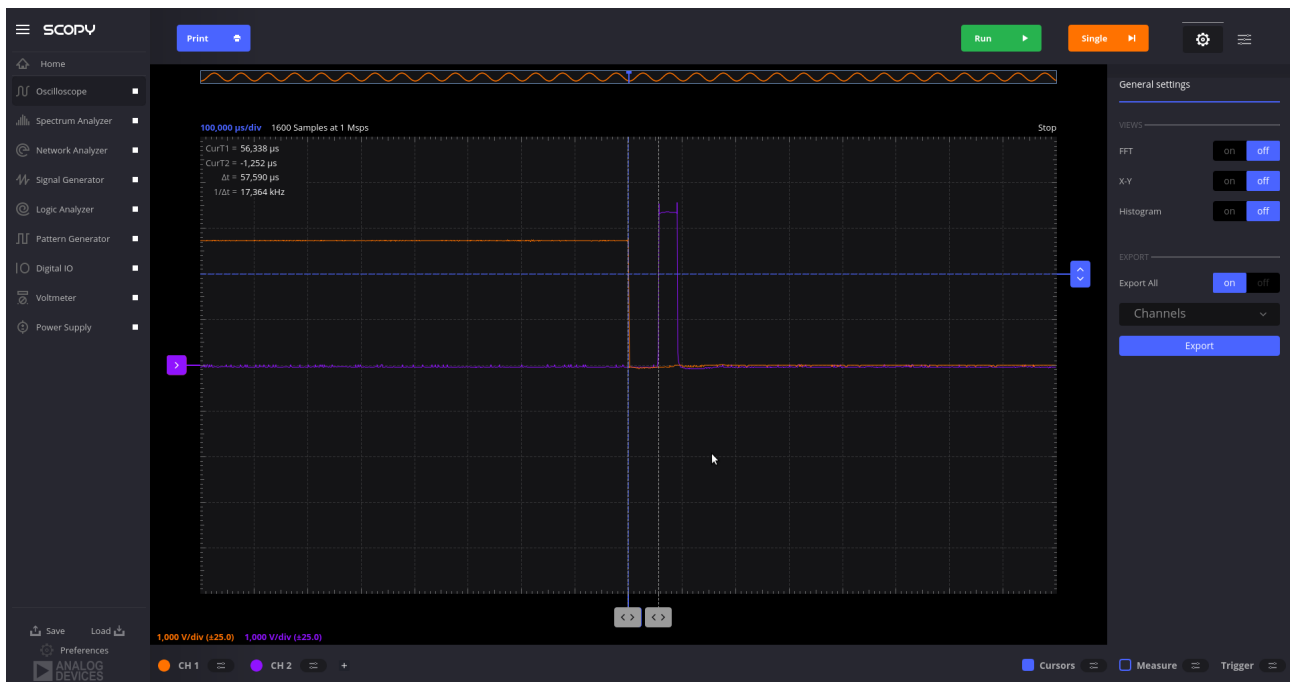
# uncomment this if your display has a black border of unused pixels visible
# and your display can output without overscan
#disable_overscan=1

# uncomment the following to adjust overscan. Use positive numbers if console
# goes off screen, and negative if there is too much border
#overscan_left=16
#overscan_right=16
#overscan_top=16
#overscan_bottom=16

# uncomment to force a console size. By default it will be display's size minus
#w
```

Damit das Real-Time Modul funktioniert muss die zweite Zeile der Datei `"/boot/config.txt"` auskommentiert werden (wie man im Bild oben sieht). Danach kann man mit `"sudo insmod ezv_rt.ko"` der Real-Time Kernel geladen werden.





Wie man sieht war die beste Reaktionszeit 47 µS und der Mittelwert liegt bei ca. 58 µS.

2)

Quelltext:

```
pi@raspberrypi:~/EZV3$ cat 2.c
#include <stdio.h>
#include <stdlib.h>
#include <wiringPi.h>
#include <stdint.h>
#include <stdbool.h>
```

```
const int pinIn = 28;
const int pinOut = 1;
```

```
size_t microsLow;
size_t microsHigh;
size_t microsAverage;
```

```
long long pollingIntervall = -1; //do not use polling if == -1
```

```
size_t startMicros = 0;
```

```
void checkMicros()
{
    if ((micros() - startMicros) >= 5000000)
    {
        startMicros = micros();
        printf("min : %d ; max : %d ; avg : %d\n", (int)microsLow,
(int)microsHigh, (int)microsAverage);
    }
}
```

```
}
```

```
size_t calculateDelta(size_t micros1, size_t micros2)
```

```
{
```

```
    size_t delta = micros2 - micros1;
```

```
    if (delta < microsLow && delta > 0)
```

```
    {
```

```
        microsLow = delta;
```

```
    }
```

```
    else if (delta > microsHigh)
```

```
    {
```

```
        microsHigh = delta;
```

```
    }
```

```
    microsAverage += delta;
```

```
    microsAverage /= 2; //calculate average
```

```
    return delta;
```

```
}
```

```
void poll()
```

```
{
```

```
    //digitalWrite(gpioPin, LOW);
```

```
    uint8_t signalsCount = 0;
```

```
    size_t micros1 = micros();
```

```
    startMicros = micros1;
```

```
    int state = digitalRead(pinIn);
```

```
    while (1)
```

```
    {
```

```
        if (signalsCount == 2)
```

```
        {
```

```
            calculateDelta(micros1, micros());
```

```
            micros1 = micros();
```

```
            signalsCount = 0;
```

```
        }
```

```
        int current = digitalRead(pinIn);
```

```
        if (current != state)
```

```
        {
```

```
            state = current;
```

```
            signalsCount++;
```

```
        }
```

```
        delayMicroseconds(pollingIntervall);
```

```
        checkMicros();
```

```
    }
```

```
}
```

```
size_t microsISR = 0;  
int signalCountISR = 0;  
bool startISR = false;
```

```
void ISRhandler()  
{  
    printf("ISR\n");  
    if (!startISR)  
    {  
        startMicros = micros();  
        startISR = true;  
    }  
  
    if (signalCountISR == 0)  
    {  
        microsISR = micros();  
    }  
  
    if (signalCountISR == 2)  
    {  
        calculateDelta(microsISR, micros());  
        microsISR = 0;  
        signalCountISR = 0;  
    }  
  
    signalCountISR++;  
  
    if (microsISR == 0)  
    {  
        microsISR = micros();  
    }  
}
```

```
int main(int argc, char** argv)  
{  
    wiringPiSetup();  
    pinMode(pinIn, INPUT);  
    //pinMode(pinOut, OUTPUT);  
  
    if (argc == 2) //use polling  
    {  
        printf("POLLING\n");  
        pollingIntervall = strtoll(argv[1], NULL, 10);  
        poll();  
    }  
  
    else  
    {  
        printf("INTERRUPT\n");  
        wiringPiISR(pinIn, INT_EDGE_FALLING, ISRhandler);  
    }  
}
```

```

        while (1)
        {
            checkMicros();
            delay(100);
        }
    }

    return 0;
}

```

Leider funktionieren die Interrupts nicht weder bei steigenden noch bei fallenden Flanken wird meine Interrupt-Service-Routine nicht aufgerufen.

Die Theorie heißt Nyquist-Shannon-Abtasttheorem. Sie besagt, dass man um ein Signal mit Frequenz f_1 , mindestens mit der Abtastfrequenz $f_2 = f_1 * 2$ das Signal abtasten muss. Wladimir Kotelnikow formulierte dieses Theorem. Shannon hat dieses Theorem mithilfe von Nyquists Arbeit entwickelt.

Man kann das Poti leider nicht fein einstellen, aber der Frequenz $f = 4.9$ kHz schafft es mein Pi nicht die Frequenz oft genug abzustaten. Das Argument 0 steht für 0 μ S delay.

```

pi@raspberrypi:~/EZV3$ ./2 0
POLLING
min : 0 ; max : 50260 ; avg : 11302
min : 0 ; max : 56722 ; avg : 5973
min : 0 ; max : 56722 ; avg : 8714
min : 0 ; max : 77241 ; avg : 14148
min : 0 ; max : 77241 ; avg : 7680
min : 0 ; max : 77241 ; avg : 4056
min : 0 ; max : 77241 ; avg : 3942
^C
pi@raspberrypi:~/EZV3$ 

```