/Pico_AD2 Diodentest

```
#include "pico/stdlib.h"  // SDK (Software Development Kit) Ein Software Development Kit (SDK) besteht aus mehreren Tools, die (typischerweise) vom Hersteller einer Hardwareplattform – hier RPi-Pico - , eines Betriebssystems oder einer Programmiersprache bereitgestellt werden. Hier: „pico/stdlib.h“ und „hardware/adc.h“

#include "hardware/adc.h"

void setup() {

  int iir=0;                    // definiert Variable „iir“ als Ganzzahl, Wert 0

  Serial.begin(115200);      // Initialisiert die serielle Kommunikation mit der
                                Geschwindigkeit in Baud

  adc_init();   // Initialisiert das ADC-Modul des  Boards mit den Standardeinstellungen

  adc_gpio_init(26); // initialisiert Port 26 als ADC-Eingang auf Kanal 0 (Port wird hochohmig)
```

gpio_pull_up(26); // Port 26 wird mit einem Pull-up Widerstand eingeschaltet

adc_select_input(0); // ADC-Eingangsauswahl. Die ADC-Eingänge 0...3 sind jeweils die GPIOs 26...28. Eingang 4 ist zur internen Spannungsmessung auf dem Board, Eingang 5 ist der integrierte Temperatursensor auf dem Board (Funktion: analogReadTemp()).

uint32_t t = time_us_32(); //die Variable „t“ wird als Variablentyp „**uint32_t**“ festgelegt (vorzeichenlose Ganzzahl mind. 32Bit groß)
Time_us_32() → gibt einen 32-Bit-Zeitstempelwert in Mikrosekunden zurück → gibt die unteren 32 Bits des Hardware-Timers zurück.

while(true){ // Eine *while* - Schleife testet immer auf wahr oder falsch (*true* oder *false*). Solange der Wert in dem Schleifenkopf *true* ist, wird der gesamte Inhalt des Blockes wiederholt. Das endet, wenn man den Wert auf *false* setzt oder die Schleife mit *break* verlässt

uint32_t ad = adc_read()*3300/4095; // die Variable „ad“ wird als Variablentyp **„uint32_t“** festgelegt (vorzeichenlose Ganzzahl mind. 32Bit groß)

adc_read()*3300/4095 → Führt eine ADC-Konvertierung durch, wartet auf das Ergebnis und gibt es dann zurück. Die Abfrage liefert einen 12-Bit-Wert zwischen 0 und 4095

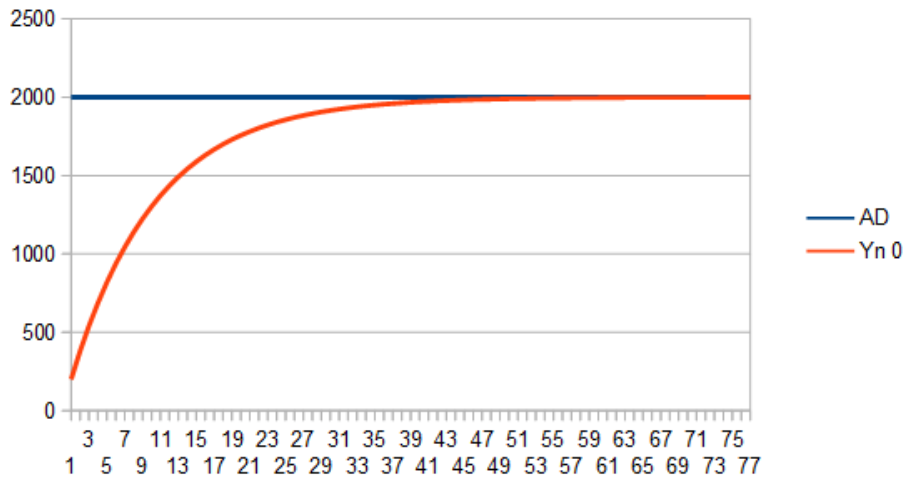
```
    for (int n=0; n<10; n++){   // In der For-Schleife wird n mit 0 initialisiert und so lange um 1 erhöht, solange n kleiner 10 wahr ist

      ad = adc_read()*3300/4095;  // adc_read()*3300/4095 → Führt eine ADC-Konvertierung durch, wartet auf das Ergebnis und gibt es dann zurück. Die Abfrage liefert einen 12-Bit-Wert zwischen 0 und 4095


      iir = (iir*9+ad)/10;       // Mittelwertbildung von 10 Messwerten/IIR-Filter (infinite impulse response)
    }
     Serial.print(iir); // Print-Ausgabe des Mittelwerts /ser.Monitor/Plotter)

    Serial.print(" mV   ");

    Serial.print(ad); // Print-Ausgabe des aktuellen ad-Wertes

    Serial.println(" mV ad-Wert");
```

```
  sleep_ms(500); // →warte die angegebene Zeit in Millisekunden


  }
}

void loop() {
}
```

Erläuterungen von Reinhard zum Code AD2:

Man sieht, dass der Ausgangswert gegen AD konvergiert. Man muss natürlich mehr als 10 Durchläufe berechnen.
Die Schleife mit den 10 Durchläufen gibt es ja nur, weil alle 10 Durchläufe ein Wert geplottet wird.
Ansonsten läuft die Schleife ja permanent durch wegen while(true).

Der erste Befehl in der while Schleife ist eigentlich überflüssig, weil der gelesene AD-Wert ja in der for-Schleife gleich wieder überschrieben wird.
Auch in dem Befehl t= time_us_32() sehe ich keinen Sinn, da die Variable t im Folgenden nicht benutzt wird.

Yn = (Yn-1 * 9 + AD) / 10 = 0,9*Yn-1 +0,1*AD; für AD=const: Yn = 0,9*Yn-1 + k

AD	Yn
	0
2000	200
2000	380
2000	542
2000	687,8
2000	819,02
2000	937,12
2000	1043,41
2000	1139,07
2000	1225,16
2000	1302,64
2000	1372,38
2000	1435,14
2000	1491,63
2000	1542,46
2000	1588,22
2000	1629,4
2000	1666,46
2000	1699,81
2000	1729,83
2000	1756,85
2000	1781,16
2000	1803,05
2000	1822,74
2000	1840,47
2000	1856,42
2000	1870,78
2000	1883,7
2000	1895,33
2000	1905,8
2000	1915,22
2000	1923,7
2000	1931,33
2000	1938,19
2000	1944,37
2000	1949,94
2000	1954,94
2000	1959,45
2000	1963,5