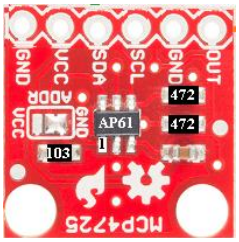
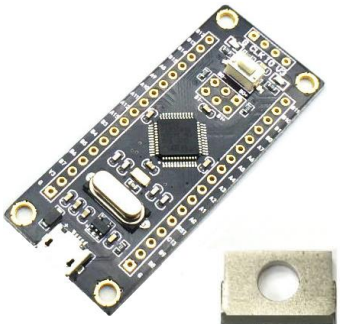
DC-Waveform Generator for Higher Current and Lower Frequencies

In this project we use a STM32F103C8 – mounted on a Black Pill – to control via I2C a 12-Bit Digital-to-Analog Converter (DAC) to generate signals like e.g. Sinus-, Triangle-, Rectangle-signals, in a voltage range of e.g. U_0=0V up to U_{max}=28V with currents of e.g. I=1A.

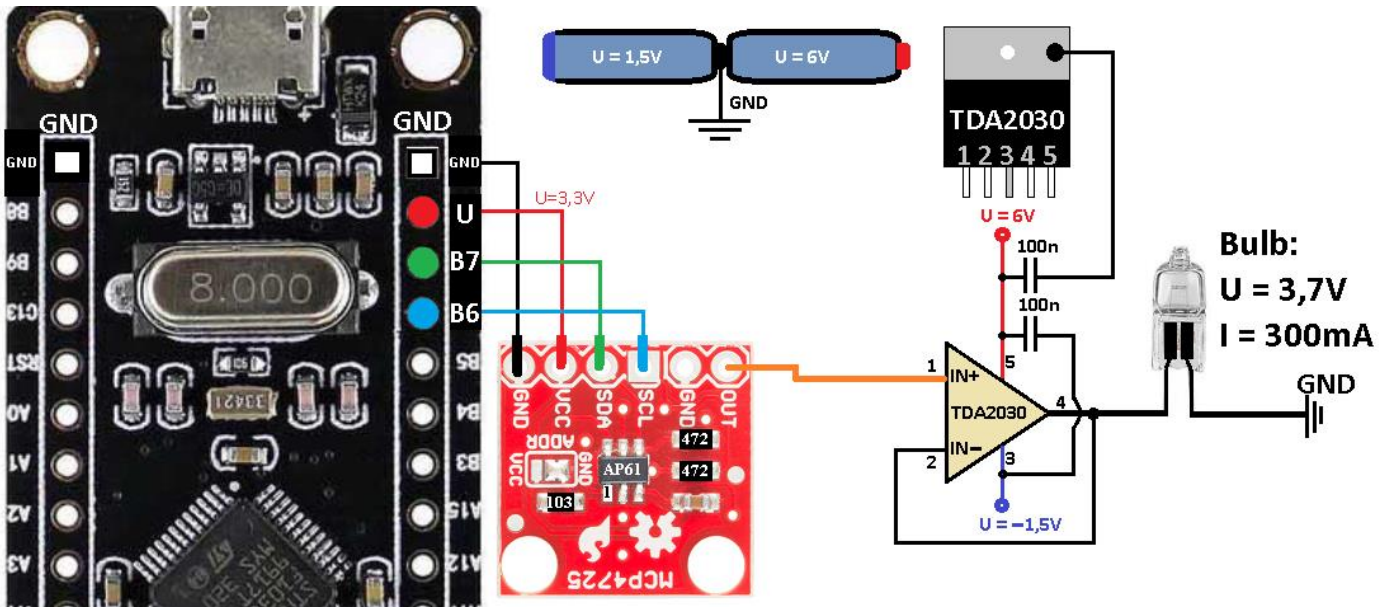




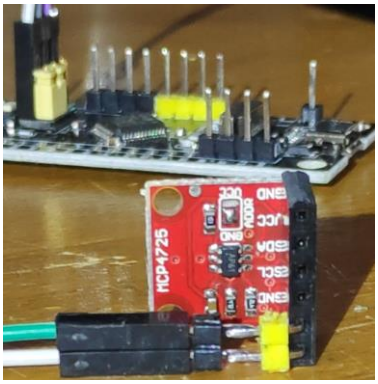
For that we need a linear working power amplifier – we use: TDA2030, that works in a desired voltage rang of U_0 up to U_{max} and a necessary current to conduct some experiments.
„The TDA2030 is a monolithic integrated circuit. At ±14V or 28V, the guaranteed output power is 12W on a 4Ω load and 8W on an 8Ω."

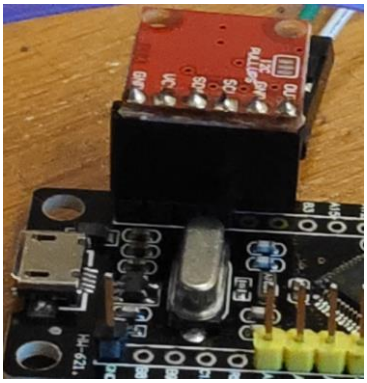
1. Problem: How can we drive a TDA2030 with a U = 3,3V Black Pill?

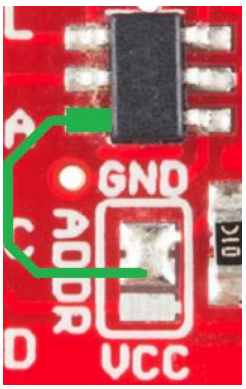
In a first step we need a little bit higher currents to demonstrate the variable resistance of a bulb's coiled filament which we switch on and off by a triangle ramp.
For this purpose, a voltage of U_{max} = 3,3V is enough, but the current has to be I_{max} = 300mA.



Because of a favorable order of pins, we can plug a MCP4725-module simply on the Black Pill as shown in the pictures. Our module has an address 0x62, since the green IC-pin is on GND.



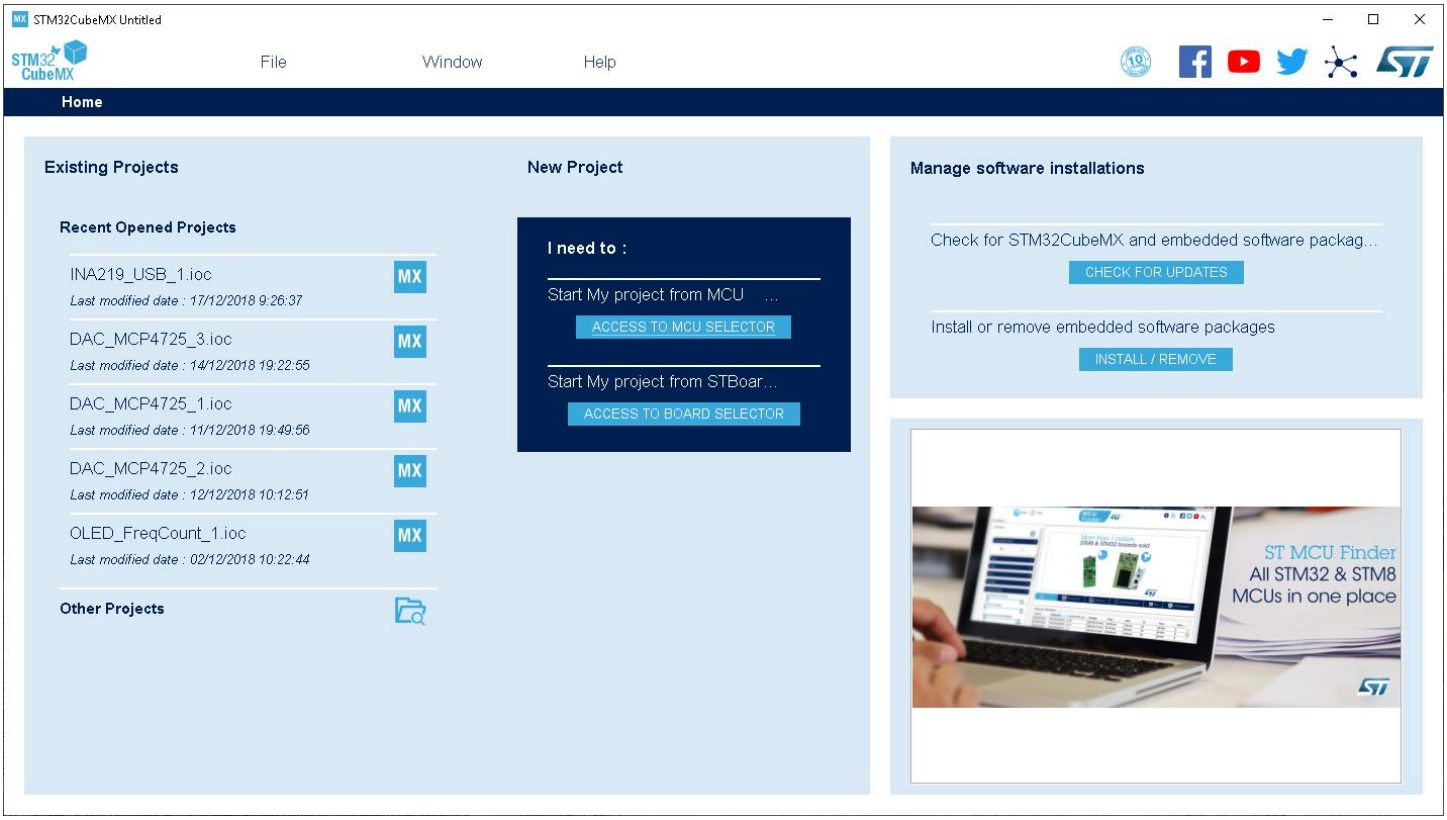


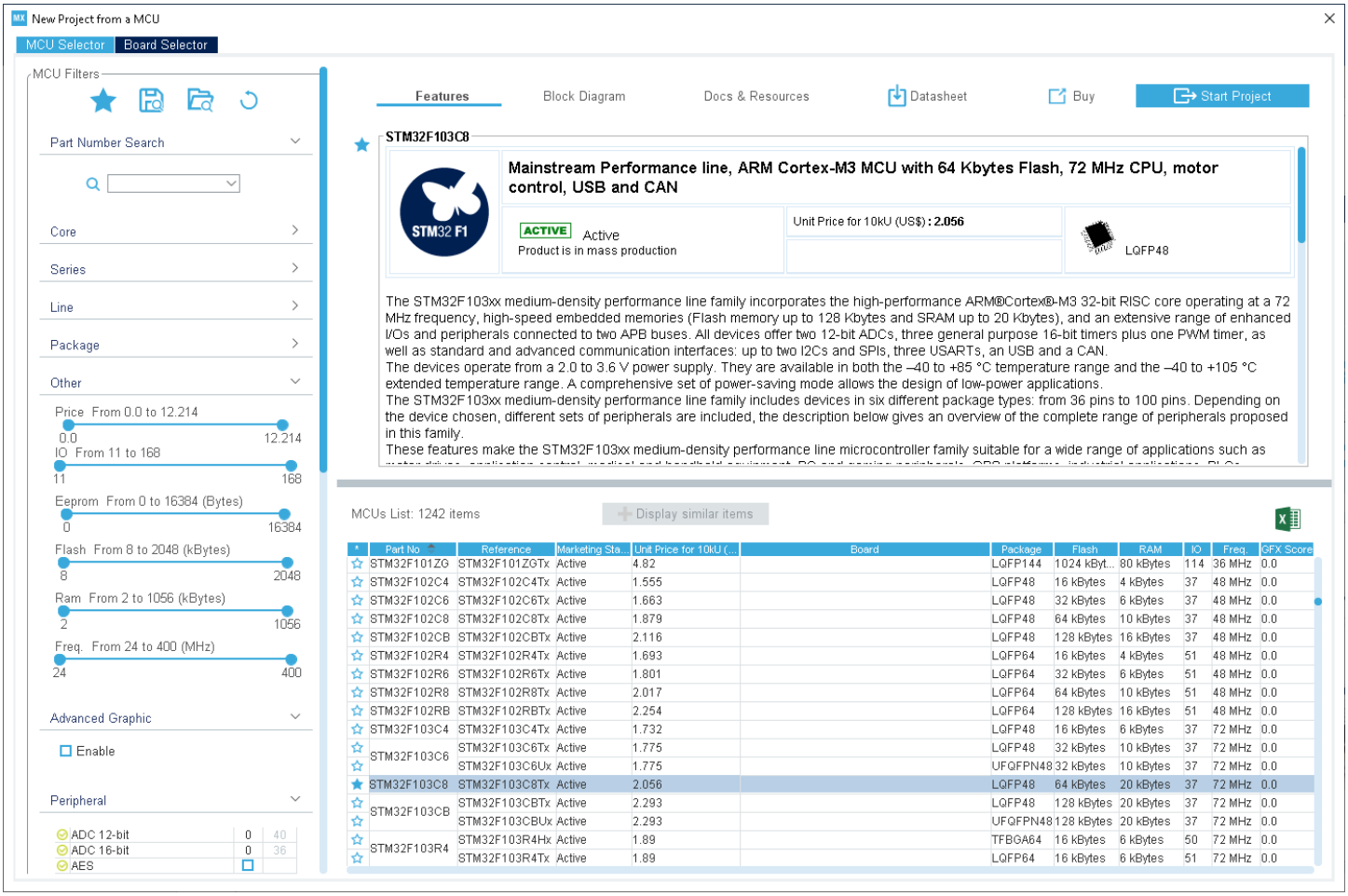


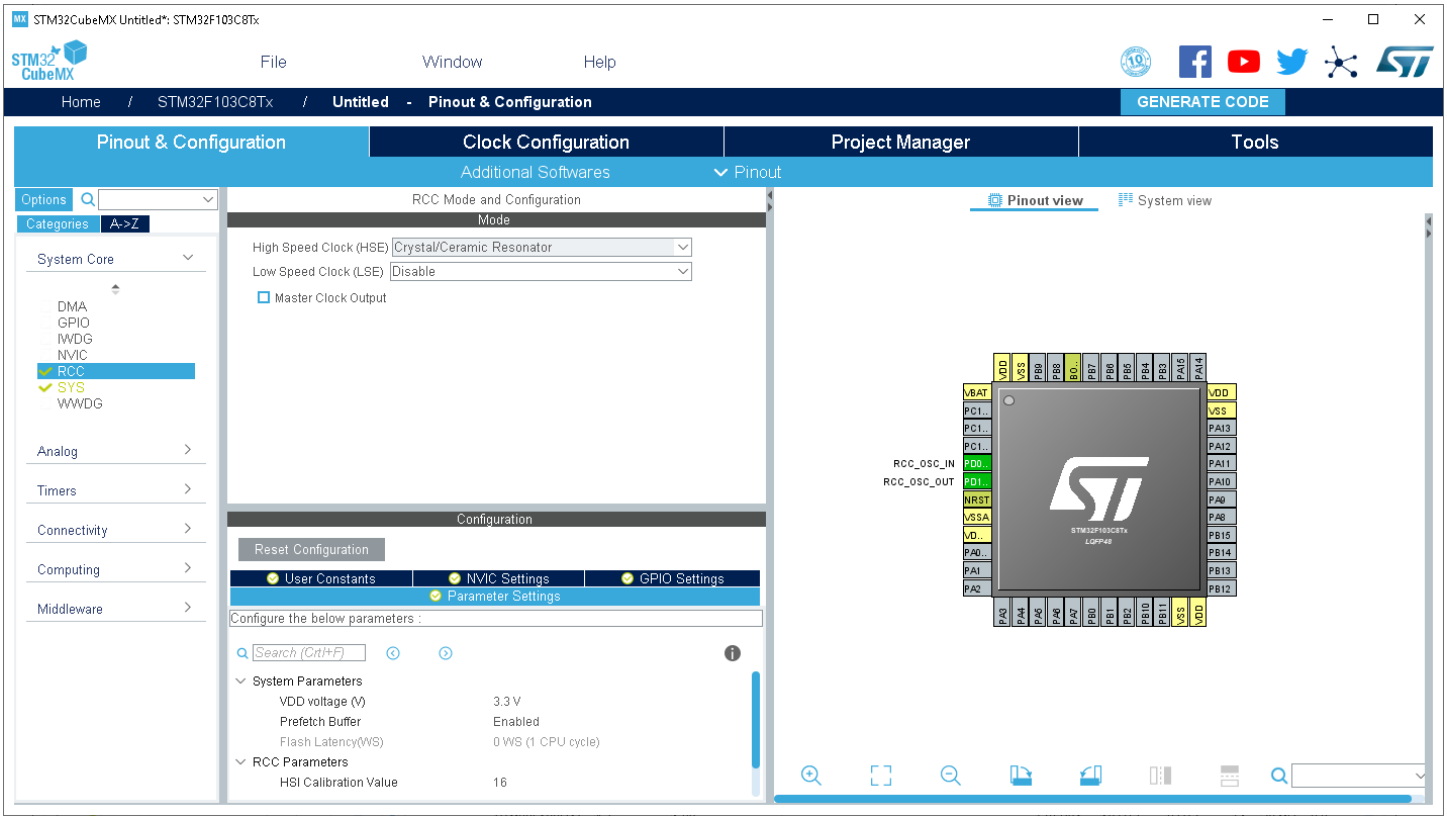
To check that MCP4725-DAC-module works well we convert its output voltage with an ADC of STM32F103 and transmit the result via USB-Black-Pill-jack to a terminal program. STM32CubeMX 4.26.1 creates the basic code of our program, whereby the version 4.26.1 is not important.

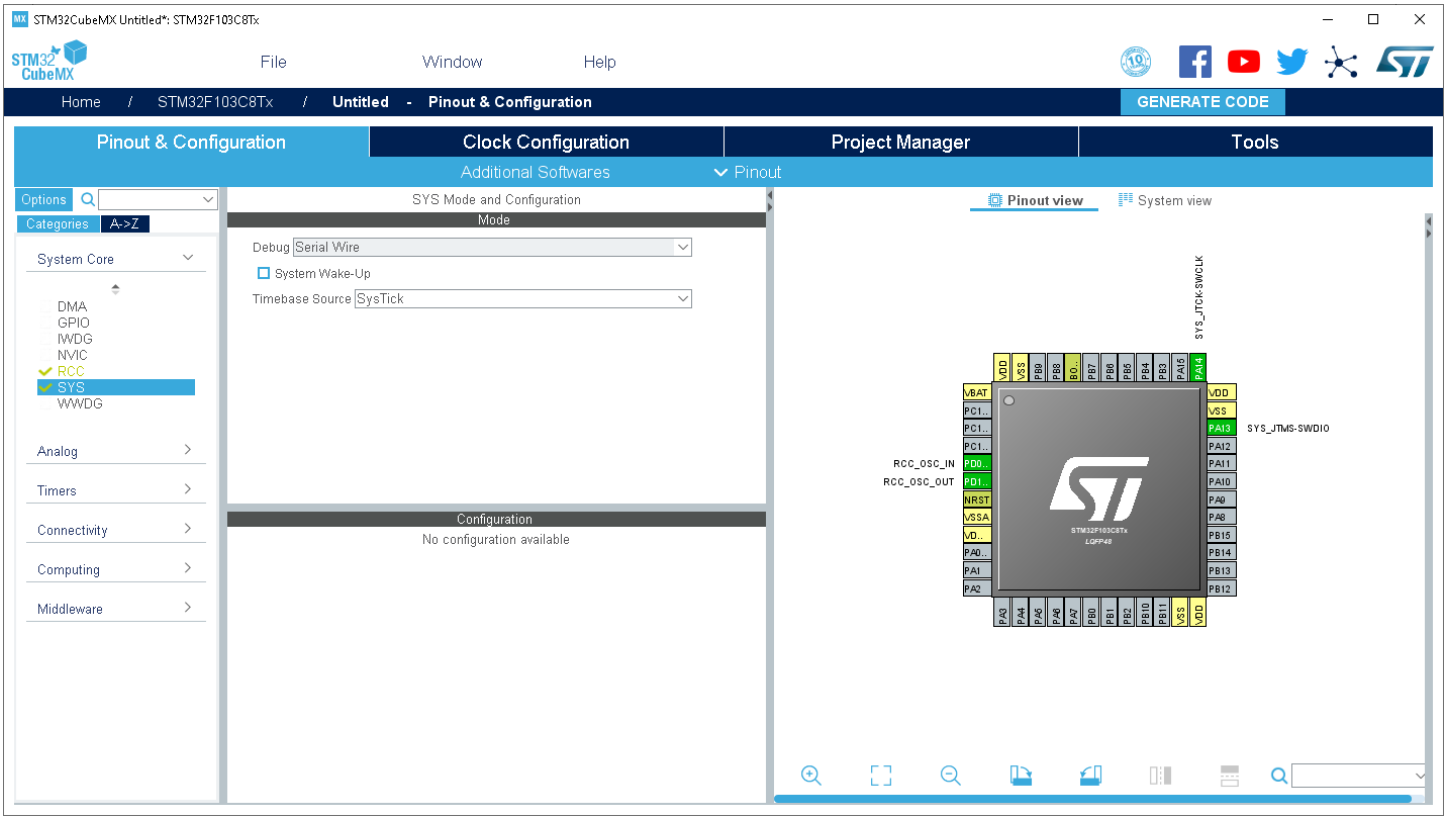
1.: Setup and check of a USB communication with a terminal program:

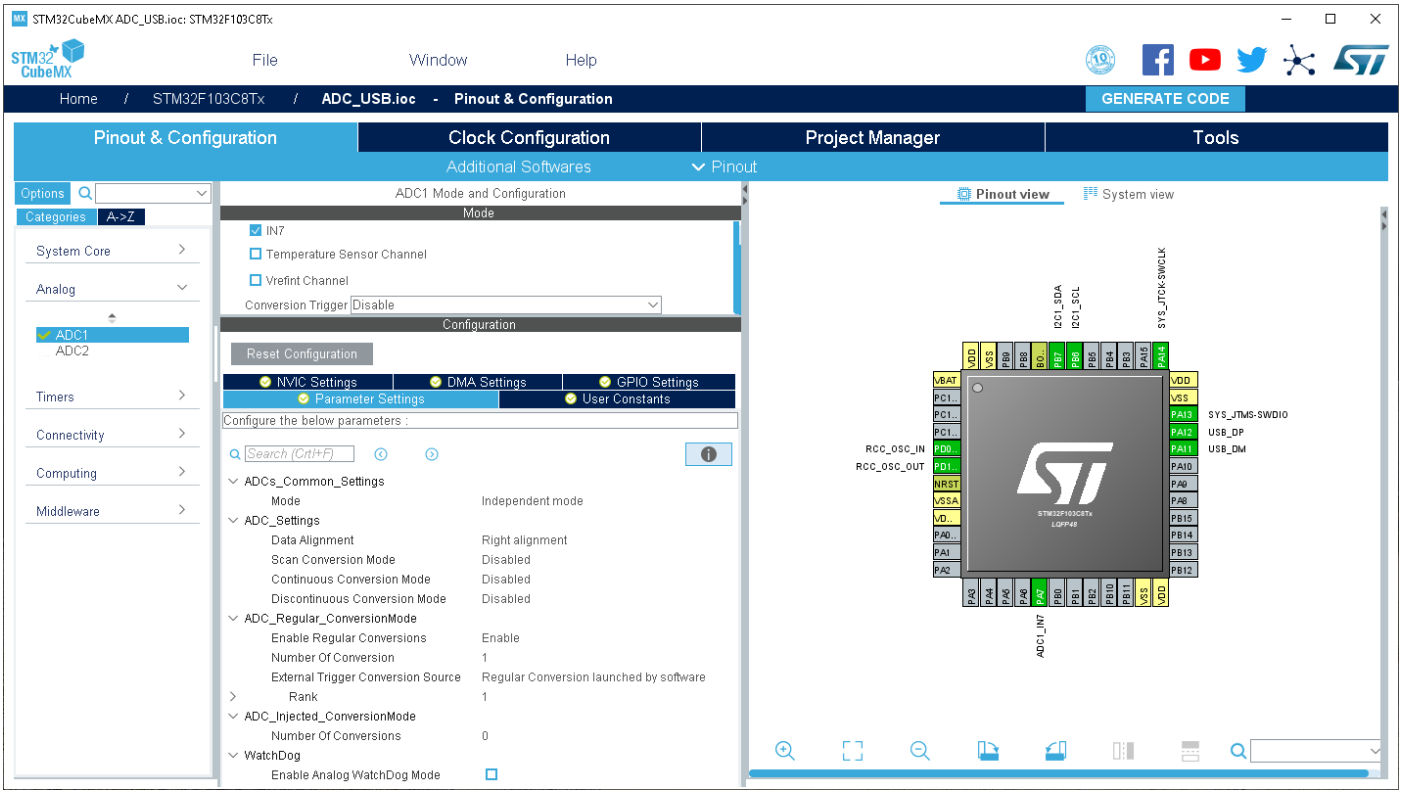
Setup a Black Pill STM21F103C8 Project With STM32CubeMX 4.26.1

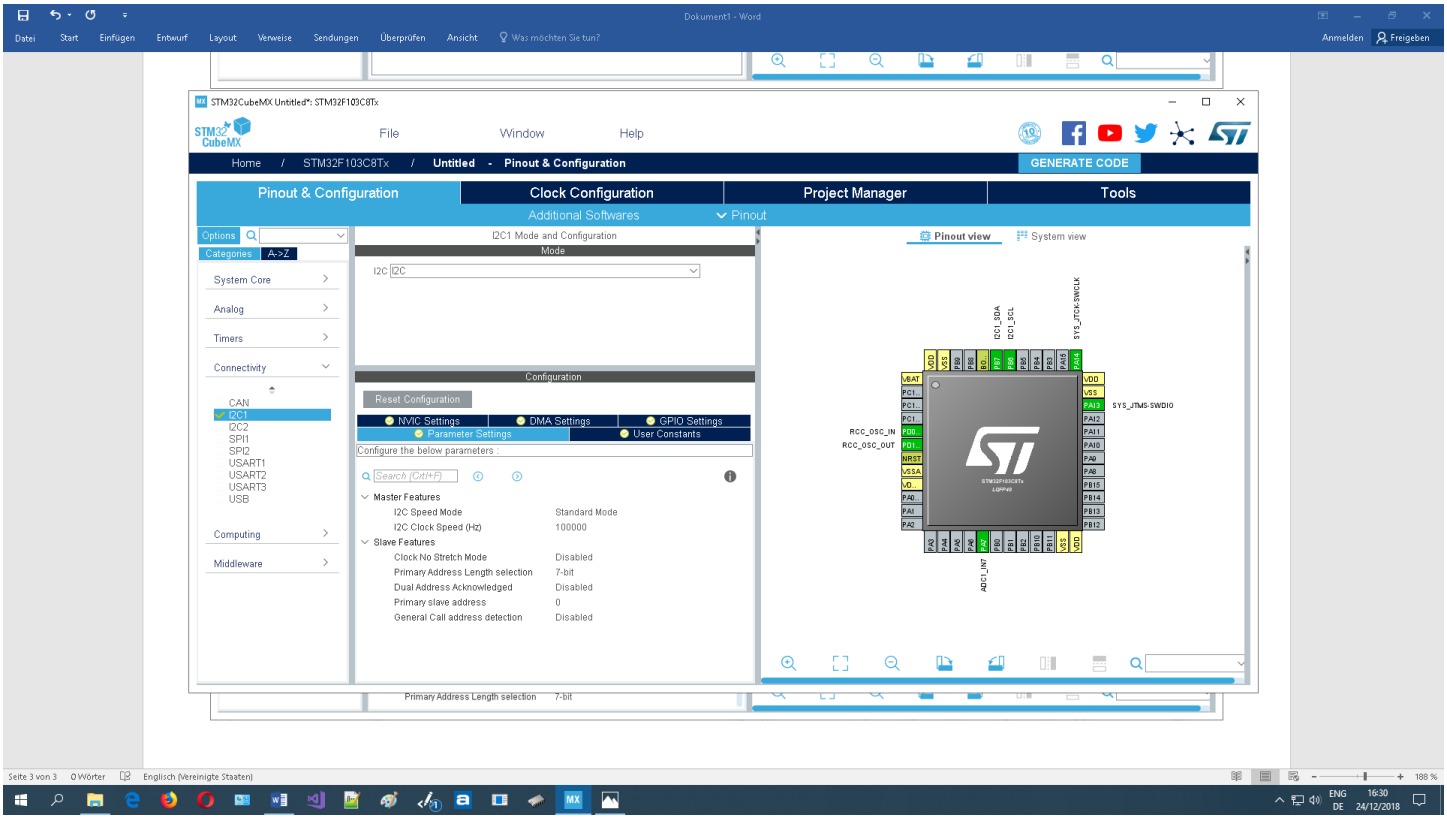


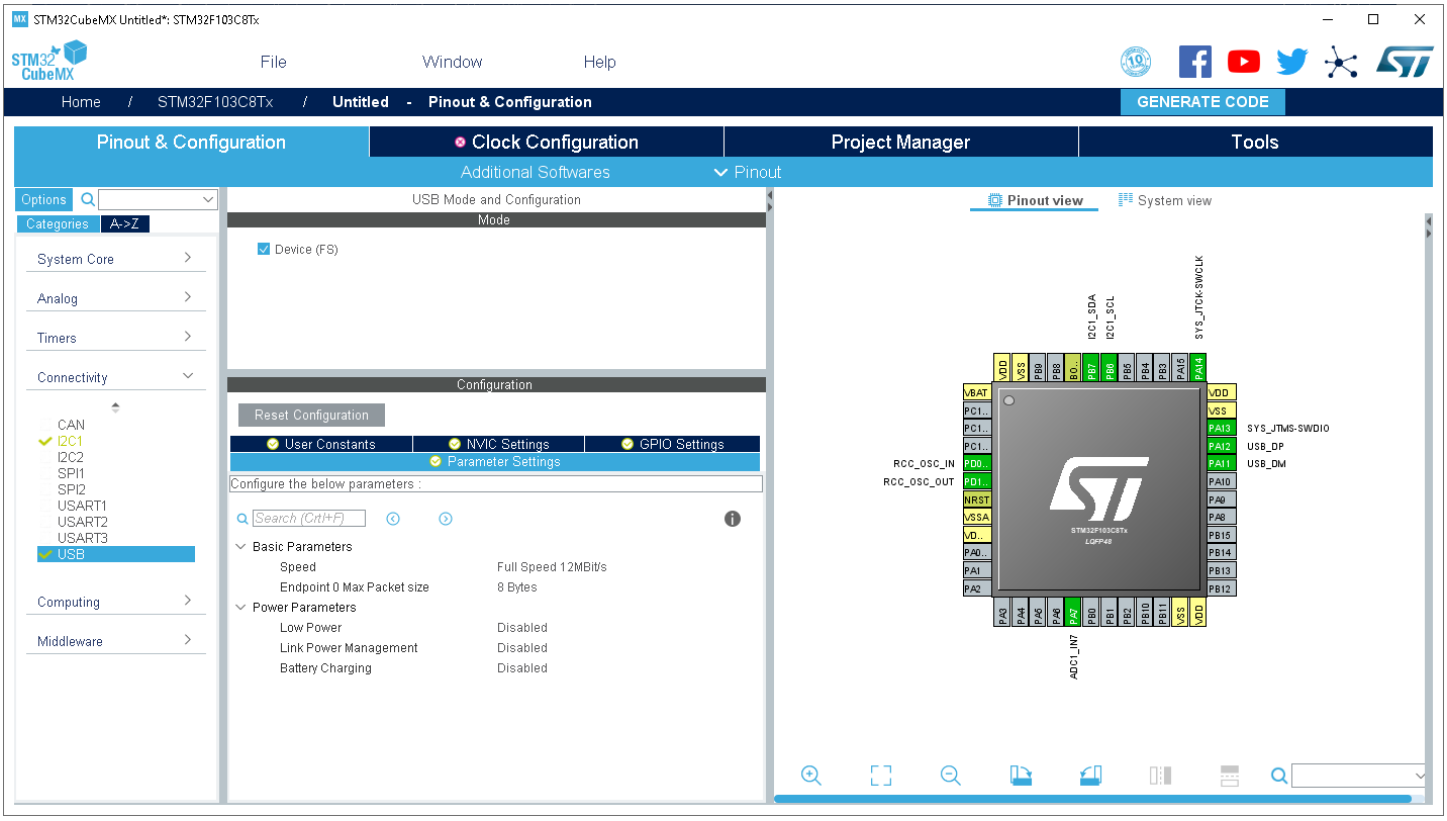

New Project from a MCU
MCU Selector
Board Selector
MCU Filters
Part Number Search
Core
Series
Line
Package
Other
Price From 0.0 to 12.214
IO From 11 to 168
Eeprom From 0 to 16384 (Bytes)
Flash From 8 to 2048 (kBytes)
Ram From 2 to 1056 (kBytes)
Freq. From 24 to 400 (MHz)
Advanced Graphic
Enable
Peripheral
ADC 12-bit
ADC 16-bit
AES
Features
Block Diagram
Docs & Resources
Datasheet
Buy
Start Project
STM32F103C8
Mainstream Performance line, ARM Cortex-M3 MCU with 64 Kbytes Flash, 72 MHz CPU, motor control, USB and CAN
ACTIVE
Active
Product is in mass production
Unit Price for 10kU (US$) : 2.056
LQFP48
The STM32F103xx medium-density performance line family incorporates the high-performance ARM®Cortex®-M3 32-bit RISC core operating at a 72 MHz frequency, high-speed embedded memories (Flash memory up to 128 Kbytes and SRAM up to 20 Kbytes), and an extensive range of enhanced I/Os and peripherals connected to two APB buses. All devices offer two 12-bit ADCs, three general purpose 16-bit timers plus one PWM timer, as well as standard and advanced communication interfaces: up to two I2Cs and SPIs, three USARTs, an USB and a CAN.
The devices operate from a 2.0 to 3.6 V power supply. They are available in both the –40 to +85 °C temperature range and the –40 to +105 °C extended temperature range. A comprehensive set of power-saving mode allows the design of low-power applications.
The STM32F103xx medium-density performance line family includes devices in six different package types: from 36 pins to 100 pins. Depending on the device chosen, different sets of peripherals are included, the description below gives an overview of the complete range of peripherals proposed in this family.
These features make the STM32F103xx medium-density performance line microcontroller family suitable for a wide range of applications such as
MCUs List: 1242 items
Display similar items
Part No
Reference
Marketing Sta...
Unit Price for 10kU (...
Board
Package
Flash
RAM
IO
Freq.
GFX Score
STM32F101ZG STM32F101ZGTx Active 4.82 LQFP144 1024 kByt... 80 kBytes 114 36 MHz 0.0
STM32F102C4 STM32F102C4Tx Active 1.555 LQFP48 16 kBytes 4 kBytes 37 48 MHz 0.0
STM32F102C6 STM32F102C6Tx Active 1.663 LQFP48 32 kBytes 6 kBytes 37 48 MHz 0.0
STM32F102C8 STM32F102C8Tx Active 1.879 LQFP48 64 kBytes 10 kBytes 37 48 MHz 0.0
STM32F102CB STM32F102CBTx Active 2.116 LQFP48 128 kBytes 16 kBytes 37 48 MHz 0.0
STM32F102R4 STM32F102R4Tx Active 1.693 LQFP64 16 kBytes 4 kBytes 51 48 MHz 0.0
STM32F102R6 STM32F102R6Tx Active 1.801 LQFP64 32 kBytes 6 kBytes 51 48 MHz 0.0
STM32F102R8 STM32F102R8Tx Active 2.017 LQFP64 64 kBytes 10 kBytes 51 48 MHz 0.0
STM32F102RB STM32F102RBTx Active 2.254 LQFP64 128 kBytes 16 kBytes 51 48 MHz 0.0
STM32F103C4 STM32F103C4Tx Active 1.732 LQFP48 16 kBytes 6 kBytes 37 72 MHz 0.0
STM32F103C6 STM32F103C6Tx Active 1.775 LQFP48 32 kBytes 10 kBytes 37 72 MHz 0.0
STM32F103C6Ux Active 1.775 UFQFPN48 32 kBytes 10 kBytes 37 72 MHz 0.0
STM32F103C8 STM32F103C8Tx Active 2.056 LQFP48 64 kBytes 20 kBytes 37 72 MHz 0.0
STM32F103CB STM32F103CBTx Active 2.293 LQFP48 128 kBytes 20 kBytes 37 72 MHz 0.0
STM32F103CBUx Active 2.293 UFQFPN48 128 kBytes 20 kBytes 37 72 MHz 0.0
STM32F103R4 STM32F103R4Hx Active 1.89 TFBGA64 16 kBytes 6 kBytes 50 72 MHz 0.0
STM32F103R4Tx Active 1.89 LQFP64 16 kBytes 6 kBytes 51 72 MHz 0.0

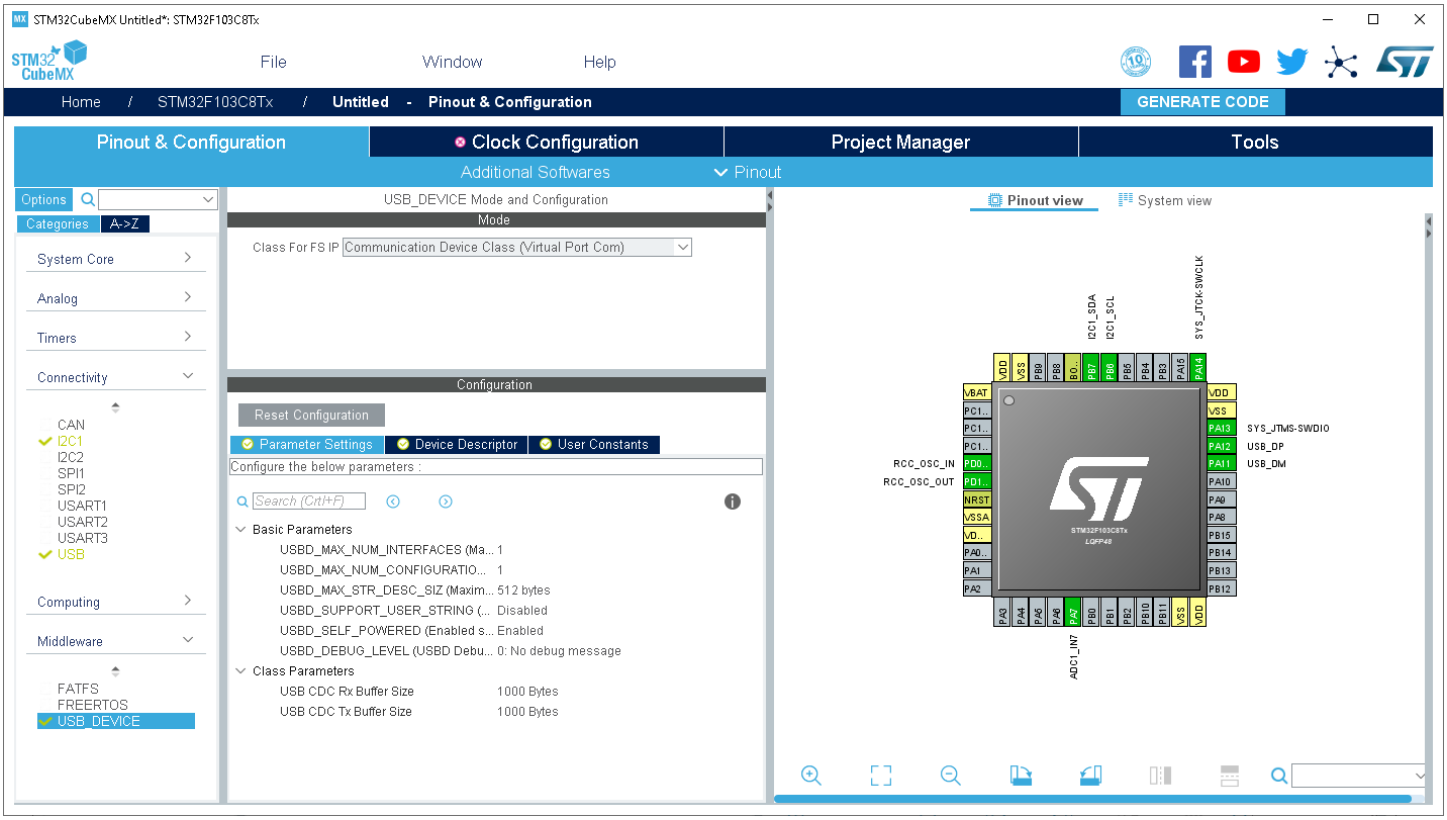

STM32CubeMX Untitled*: STM32F103C8Tx
File
Window
Help
Home / STM32F103C8Tx / Untitled - Pinout & Configuration
GENERATE CODE
Pinout & Configuration
Clock Configuration
Project Manager
Tools
Additional Softwares
Pinout
Options
Categories
A->Z
System Core
DMA
GPIO
IWDG
NVIC
RCC
SYS
WWDG
Analog
Timers
Connectivity
Computing
Middleware
RCC Mode and Configuration
Mode
High Speed Clock (HSE) Crystal/Ceramic Resonator
Low Speed Clock (LSE) Disable
Master Clock Output
Configuration
Reset Configuration
User Constants
NVIC Settings
GPIO Settings
Parameter Settings
Configure the below parameters :
Search (Crtl+F)
System Parameters
VDD voltage (V) 3.3 V
Prefetch Buffer Enabled
Flash Latency(WS) 0 WS (1 CPU cycle)
RCC Parameters
HSI Calibration Value 16
Pinout view
System view
RCC_OSC_IN
RCC_OSC_OUT

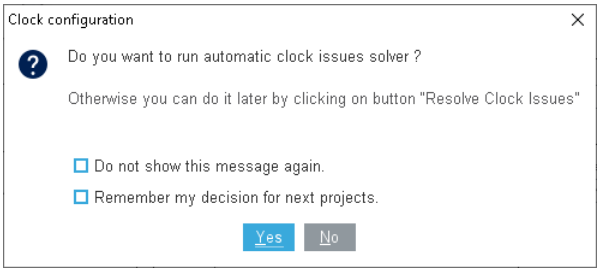

STM32CubeMX Untitled*: STM32F103C8Tx
File
Window
Help
Home / STM32F103C8Tx / Untitled - Pinout & Configuration
GENERATE CODE
Pinout & Configuration
Clock Configuration
Project Manager
Tools
Additional Softwares
Pinout
Options
Categories
A->Z
System Core
DMA
GPIO
IWDG
NVIC
RCC
SYS
WWDG
Analog
Timers
Connectivity
Computing
Middleware
SYS Mode and Configuration
Mode
Debug
Serial Wire
System Wake-Up
Timebase Source
SysTick
Configuration
No configuration available
Pinout view
System view
SYS_JTCK-SWCLK
SYS_JTMS-SWDIO
RCC_OSC_IN
RCC_OSC_OUT

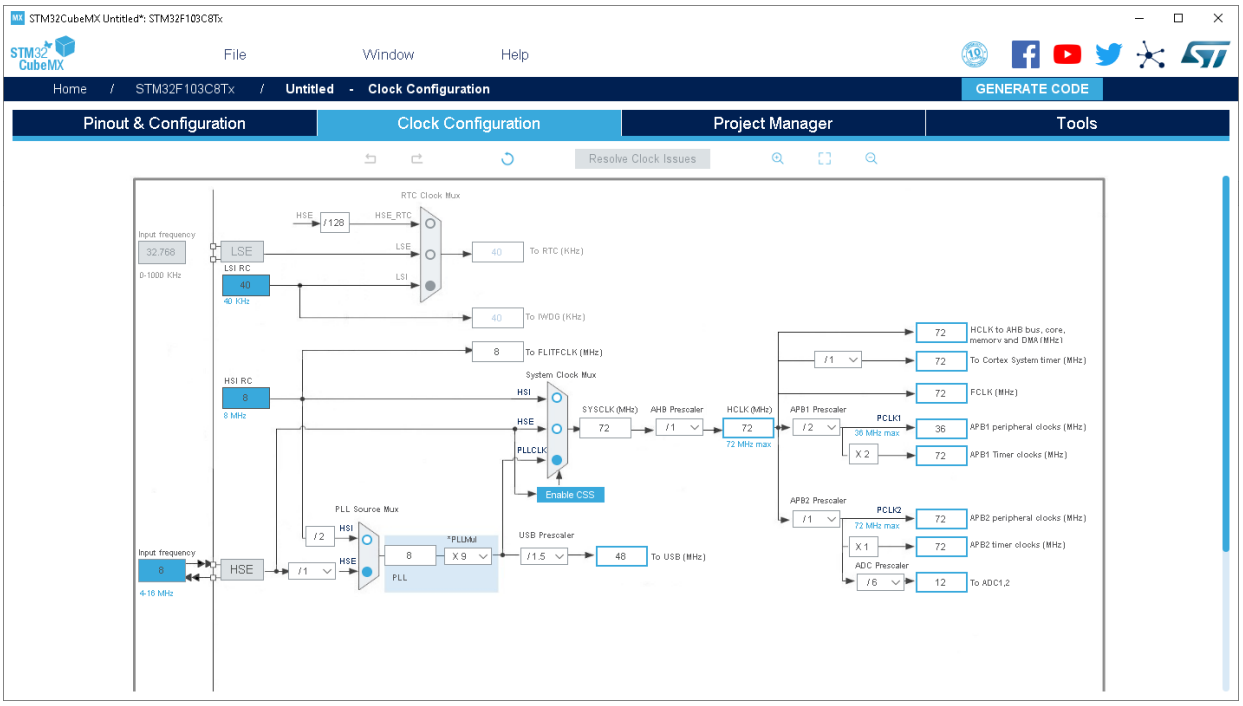

STM32CubeMX ADC_USB.ioc: STM32F103C8Tx
File
Window
Help
Home / STM32F103C8Tx / ADC_USB.ioc - Pinout & Configuration
GENERATE CODE
Pinout & Configuration
Clock Configuration
Project Manager
Tools
Additional Softwares
Pinout
Options
Categories
A->Z
System Core
Analog
ADC1
ADC2
Timers
Connectivity
Computing
Middleware
ADC1 Mode and Configuration
Mode
IN7
Temperature Sensor Channel
Vrefint Channel
Conversion Trigger
Disable
Configuration
Reset Configuration
NVIC Settings
DMA Settings
GPIO Settings
Parameter Settings
User Constants
Configure the below parameters :
Search (Crtl+F)
ADCs_Common_Settings
Mode
Independent mode
ADC_Settings
Data Alignment
Right alignment
Scan Conversion Mode
Disabled
Continuous Conversion Mode
Disabled
Discontinuous Conversion Mode
Disabled
ADC_Regular_ConversionMode
Enable Regular Conversions
Enable
Number Of Conversion
1
External Trigger Conversion Source
Regular Conversion launched by software
Rank
1
ADC_Injected_ConversionMode
Number Of Conversions
0
WatchDog
Enable Analog WatchDog Mode
Pinout view
System view
I2C1_SDA
I2C1_SCL
SYS_JTCK-SWCLK
SYS_JTMS-SWDIO
USB_DP
USB_DM
RCC_OSC_IN
RCC_OSC_OUT
ADC1_IN7

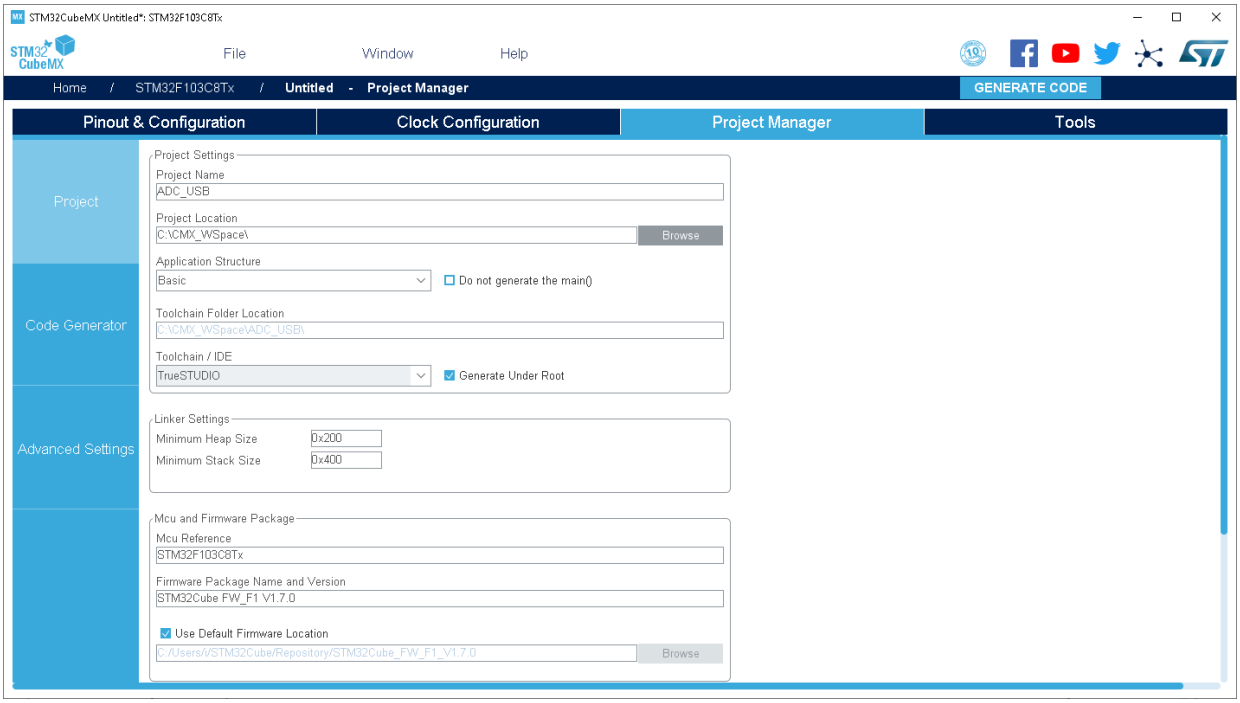

Dokument1 - Word
STM32CubeMX Untitled*: STM32F103C8Tx
File
Window
Help
Home / STM32F103C8Tx / Untitled - Pinout & Configuration
GENERATE CODE
Pinout & Configuration
Clock Configuration
Project Manager
Tools
Additional Softwares
Pinout
I2C1 Mode and Configuration
Mode
I2C
Configuration
Reset Configuration
NVIC Settings
DMA Settings
GPIO Settings
Parameter Settings
User Constants
Configure the below parameters :
Search (Crtl+F)
Master Features
I2C Speed Mode
Standard Mode
I2C Clock Speed (Hz)
100000
Slave Features
Clock No Stretch Mode
Disabled
Primary Address Length selection
7-bit
Dual Address Acknowledged
Disabled
Primary slave address
0
General Call address detection
Disabled
Pinout view
System view
I2C1_SDA
I2C1_SCL
SYS_JTCK-SWCLK
SYS_JTMS-SWDIO
RCC_OSC_IN
RCC_OSC_OUT
ADC1_IN7


STM32CubeMX Untitled*: STM32F103C8Tx
File
Window
Help
Home / STM32F103C8Tx / Untitled - Pinout & Configuration
GENERATE CODE
Pinout & Configuration
Clock Configuration
Project Manager
Tools
Additional Softwares
Pinout
Options
Categories
A->Z
System Core
Analog
Timers
Connectivity
CAN
I2C1
I2C2
SPI1
SPI2
USART1
USART2
USART3
USB
Computing
Middleware
USB Mode and Configuration
Mode
Device (FS)
Configuration
Reset Configuration
User Constants
NVIC Settings
GPIO Settings
Parameter Settings
Configure the below parameters :
Search (Crtl+F)
Basic Parameters
Speed
Full Speed 12MBit/s
Endpoint 0 Max Packet size
8 Bytes
Power Parameters
Low Power
Disabled
Link Power Management
Disabled
Battery Charging
Disabled
Pinout view
System view
I2C1_SDA
I2C1_SCL
SYS_JTCK-SWCLK
SYS_JTMS-SWDIO
USB_DP
USB_DM
RCC_OSC_IN
RCC_OSC_OUT
ADC1_IN7


STM32CubeMX Untitled*: STM32F103C8Tx
File
Window
Help
Home / STM32F103C8Tx / Untitled - Pinout & Configuration
GENERATE CODE
Pinout & Configuration
Clock Configuration
Project Manager
Tools
Additional Softwares
Pinout
Options
Categories
A->Z
System Core
Analog
Timers
Connectivity
CAN
I2C1
I2C2
SPI1
SPI2
USART1
USART2
USART3
USB
Computing
Middleware
FATFS
FREERTOS
USB_DEVICE
USB_DEVICE Mode and Configuration
Mode
Class For FS IP
Communication Device Class (Virtual Port Com)
Configuration
Reset Configuration
Parameter Settings
Device Descriptor
User Constants
Configure the below parameters :
Search (Crtl+F)
Basic Parameters
USBD_MAX_NUM_INTERFACES (Ma... 1
USBD_MAX_NUM_CONFIGURATIO... 1
USBD_MAX_STR_DESC_SIZ (Maxim... 512 bytes
USBD_SUPPORT_USER_STRING (... Disabled
USBD_SELF_POWERED (Enabled s... Enabled
USBD_DEBUG_LEVEL (USBD Debu... 0: No debug message
Class Parameters
USB CDC Rx Buffer Size 1000 Bytes
USB CDC Tx Buffer Size 1000 Bytes
Pinout view
System view
I2C1_SDA
I2C1_SCL
SYS_JTCK-SWCLK
SYS_JTMS-SWDIO
USB_DP
USB_DM
RCC_OSC_IN
RCC_OSC_OUT
ADC1_IN7


Clock configuration
Do you want to run automatic clock issues solver ?
Otherwise you can do it later by clicking on button "Resolve Clock Issues"
Do not show this message again.
Remember my decision for next projects.
Yes
No


STM32CubeMX Untitled*: STM32F103C8Tx
File
Window
Help
Home / STM32F103C8Tx / Untitled - Clock Configuration
GENERATE CODE
Pinout & Configuration
Clock Configuration
Project Manager
Tools
Resolve Clock Issues
RTC Clock Mux
HSE
/128
HSE_RTC
LSE
LSI
To RTC (KHz)
To IWDG (KHz)
To FLITFCLK (MHz)
HSI RC
System Clock Mux
HSI
HSE
PLLCLK
SYSCLK (MHz)
AHB Prescaler
HCLK (MHz)
APB1 Prescaler
APB2 Prescaler
PCLK1
PCLK2
HCLK to AHB bus, core, memory and DMA (MHz)
To Cortex System timer (MHz)
FCLK (MHz)
APB1 peripheral clocks (MHz)
APB1 Timer clocks (MHz)
APB2 peripheral clocks (MHz)
APB2 timer clocks (MHz)
ADC Prescaler
To ADC1,2
Enable CSS
PLL Source Mux
PLL
USB Prescaler
To USB (MHz)
Input frequency
LSE
LSI RC
HSI RC
HSE



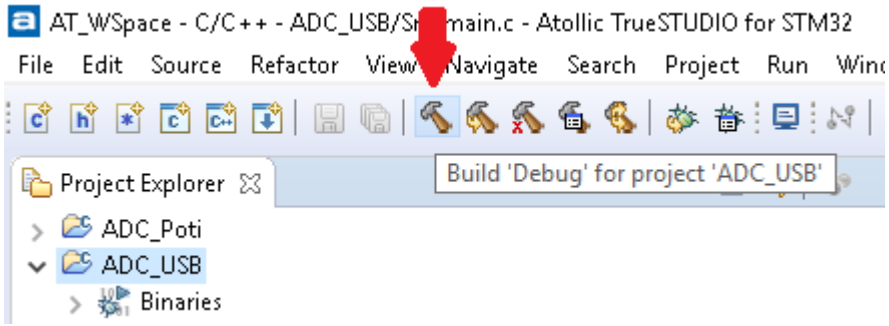
Let’s add some gray highlighted code to the code of CubeMX:

```
#include "main.h"
#include "usb_device.h"
#include "usbd_cdc_if.h"//By reason of: CDC_Transmit_FS(uint8_t*, uint16_t);

ADC_HandleTypeDef hadc1;
I2C_HandleTypeDef hi2c1;
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
static void MX_I2C1_Init(void);

uint16_t slen(const char*);
void add_txt(char* , char* );
void USB_Info(char*);

int main(void) {
HAL_Init();
SystemClock_Config();
MX_GPIO_Init();
MX_ADC1_Init();
MX_I2C1_Init();
MX_USB_DEVICE_Init();
while (1){
  USB_Info("test123456789");
  HAL_Delay(500);
}
}

uint16_t slen(const char* s) {
    uint16_t i;
    for (i = 0; s[i] != 0; i++);
    return i;//s[0] not 0 then i=1;
}
```

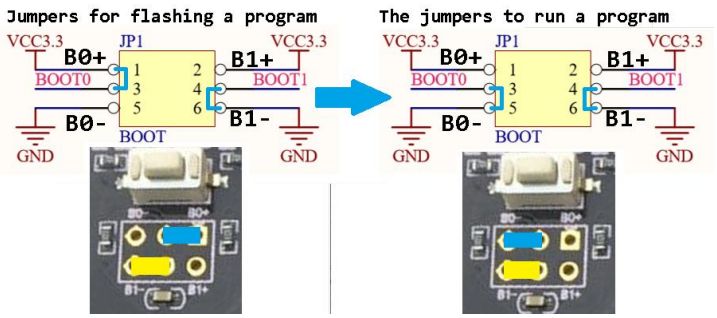
```
void add_txt(char* out, char* in) {
  while (*out != 0) out++;
  while (*in != 0) {
    *out++ = *in++;
  }
  *out = 0;
}

void USB_Info(char *str) {
 char txt[64] = {};
 add_txt( txt, str);
 add_txt( txt, "\n\r");
 CDC_Transmit_FS((uint8_t *)txt, slen(txt));
}

void SystemClock_Config(void)
{
RCC_OscInitTypeDef RCC_OscInitStruct = {0};
RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};
RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
RCC_OscInitStruct.HSEState = RCC_HSE_ON;
RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
RCC_OscInitStruct.HSIState = RCC_HSI_ON;
RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK) Error_Handler();
RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK) Error_Handler();
PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC|RCC_PERIPHCLK_USB;
PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV6;
PeriphClkInit.UsbClockSelection = RCC_USBCLKSOURCE_PLL_DIV1_5;
if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK) Error_Handler();
}
static void MX_ADC1_Init(void)
{
ADC_ChannelConfTypeDef sConfig = {0};
hadc1.Instance = ADC1;
hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
hadc1.Init.ContinuousConvMode = DISABLE;
hadc1.Init.DiscontinuousConvMode = DISABLE;
hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
hadc1.Init.NbrOfConversion = 1;
if (HAL_ADC_Init(&hadc1) != HAL_OK) Error_Handler();
sConfig.Channel = ADC_CHANNEL_7;
sConfig.Rank = ADC_REGULAR_RANK_1;
sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK) Error_Handler();
```

```
}
static void MX_I2C1_Init(void)
{
hi2c1.Instance = I2C1;
hi2c1.Init.ClockSpeed = 100000;
hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
hi2c1.Init.OwnAddress1 = 0;
hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
hi2c1.Init.OwnAddress2 = 0;
hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
if (HAL_I2C_Init(&hi2c1) != HAL_OK) Error_Handler();
}
static void MX_GPIO_Init(void)
{
__HAL_RCC_GPIOD_CLK_ENABLE();
__HAL_RCC_GPIOA_CLK_ENABLE();
__HAL_RCC_GPIOB_CLK_ENABLE();
}
void Error_Handler(void){}
#ifdef  USE_FULL_ASSERT
void assert_failed(uint8_t *file, uint32_t line){}
#endif //P-End
```

Build the project:

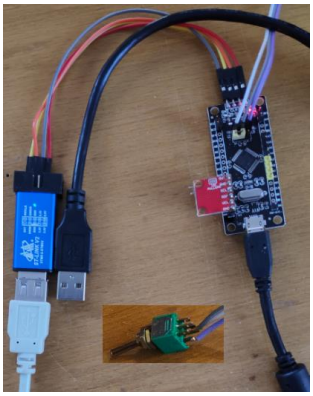


Put the jumper B0 in position FLASHING:

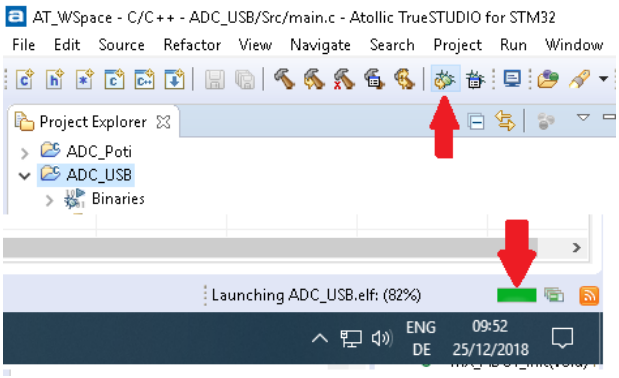


and plug ST-Link as shown:

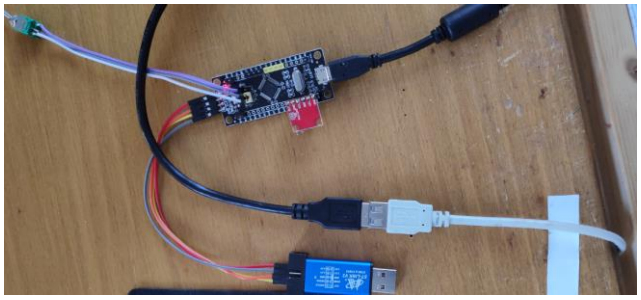
We use a switch as B0 jumper.



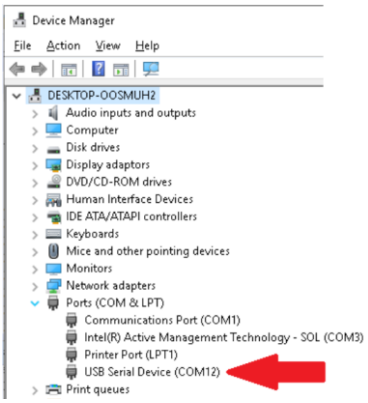
Transfer your program to Black Pill:



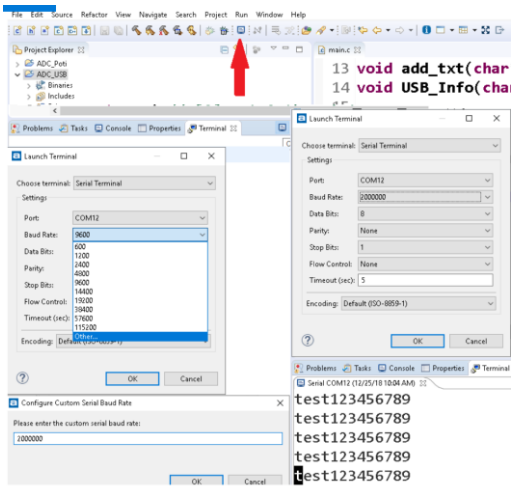
To use USB, disconnect ST-Link programmer, move B0 jumper to RUN mode and at last connect the black USB cable:



A new COM port appears to use with any terminal program:



We launch TrueSTUDIO's terminal program with a Baud Rate of 2000000 or more:

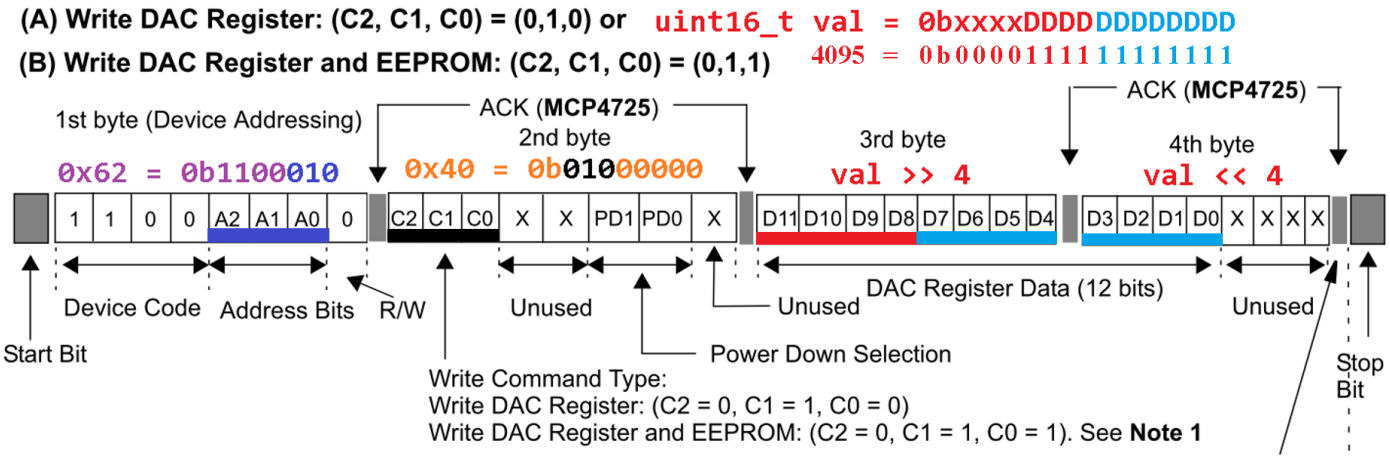


The Terminal-Sheet contains our test string "test123456789"; every second appears another one.

2.: Procedures to drive MCP4725-Module:

In CubeMX we choose I2C1 for communication with pin PB6(CLOCK) and PB7(DATA). With these adjacent pins GND, U = 3,3V, DATA and CLOCK, we can easy connect the MCP4725-module on the Black Pill.

The DAC splits the voltage intervall [0V, 3,3V] into 4096 interim values, with 0 for U = 0V and 4095 for U = 3,3V. The data sheet explains how to write the 12-Bit DAC-Register:



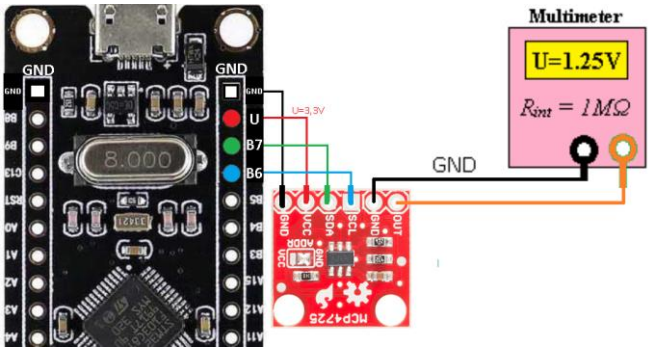
To write DAC-Register we can use one of the HAL-procedures:

HAL_I2C_Master_Transmit or HAL_I2C_Mem_Write. Since HAL_I2C cares for the R/W-Bit, we have to use **0x62 << 1** in code instead of **0x62** to end up with 8 Bit.

```
void MCP4725_WriteReg(uint16_t val)
{//0x62 is MCP4725-modul
    uint8_t data[] = { 0x40, val >> 4, val << 4 };
  HAL_I2C_Master_Transmit(&hi2c1, 0x62 << 1, data, 3, HAL_MAX_DELAY);
  HAL_Delay(1);
}
void MCP4725_Mem_WriteReg(uint16_t val)
{//0x62 is MCP4725-modul
  uint8_t data[] = { val >> 4, val << 4 };
```

```
  HAL_I2C_Mem_Write(&hi2c1, 0x62 << 1, 0x40, 1, data, 2, HAL_MAX_DELAY);
  HAL_Delay(1);
}
```

Let's check initially the output voltage with a digital multimeter. Later we use the ADC and the USB-jack of the Black Pill.



In **main{ … }** we change in our previous program the while-loop from:

```
while (1){
  USB_Info("test123456789");
  HAL_Delay(500);
}
```

to:

```
while (1){
  for(int n = 0; n <= 4095; n += 9){
    MCP4725_Mem_WriteReg(n);
      HAL_Delay(100);
  }
  for(int n = 4086; n >= 0; n -= 9){
      MCP4725_Mem_WriteReg(n);
      HAL_Delay(100);
  }
}
```

The green highlighted lines of code are the changes. Although the USB-code is still there, we do not use Black Pill's USB-functionality in this 2. step.

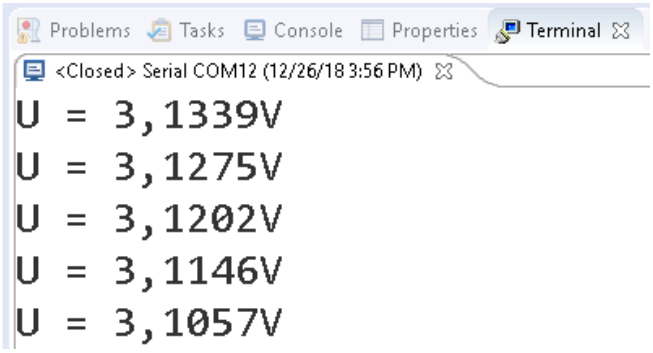
```
#include "main.h"
#include "usb_device.h"
#include "usbd_cdc_if.h"//By reason of: CDC_Transmit_FS(uint8_t*, uint16_t);

ADC_HandleTypeDef hadc1;
I2C_HandleTypeDef hi2c1;
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
static void MX_I2C1_Init(void);

void MCP4725_Mem_WriteReg(uint16_t);
uint16_t slen(const char*);
void add_txt(char* , char* );
```

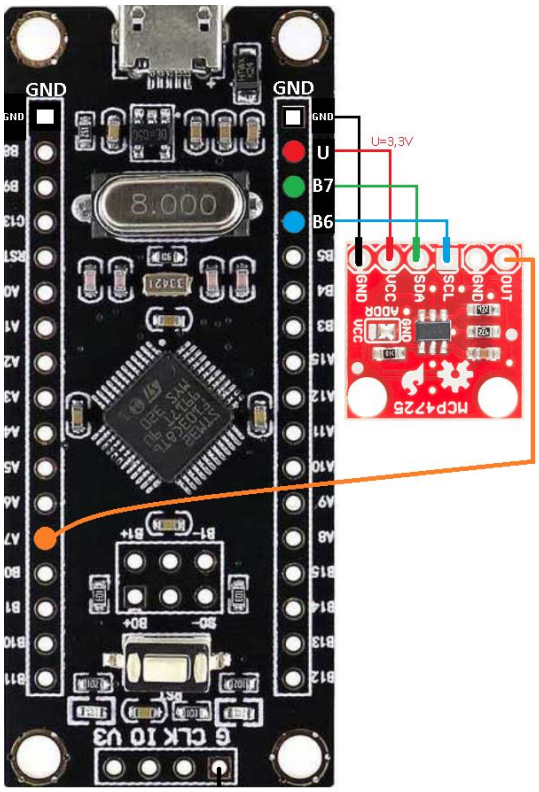
```
void USB_Info(char*);

int main(void)
{
HAL_Init();
SystemClock_Config();
MX_GPIO_Init();
MX_ADC1_Init();
MX_I2C1_Init();
MX_USB_DEVICE_Init();
while (1){
  for(int n = 0; n <= 4095; n += 9){
    MCP4725_Mem_WriteReg(n);
      HAL_Delay(100);
  }
  for(int n = 4086; n >= 0; n -= 9){
      MCP4725_Mem_WriteReg(n);
      HAL_Delay(100);
  }
}
}

void MCP4725_Mem_WriteReg(uint16_t val)
{//0x62 is MCP4725-modul
   uint8_t data[] = { val >> 4, val << 4 };
   HAL_I2C_Mem_Write(&hi2c1, 0x62 << 1, 0x40, 1, data, 2, HAL_MAX_DELAY);
   HAL_Delay(1);
}

uint16_t slen(const char* s) {
   uint16_t i;
   for (i = 0; s[i] != 0; i++);
   return i;//s[0] not 0 then i=1;
}
```

… and the remaining code until … **#endif** //P-End

3.: Now we replace the digital multimeter by an ADC1 at Pin PA7, the USB-functionality of the Black Pill and a Terminal program.

Connect MCP4725-OUT with ADC1-Pin PA7:





We can start and stop an ADC on STM32F108. Is in the ADC initialization procedure
static void MX_ADC1_Init(void) a continuous conversion disabled:
hadc1.Init.ContinuousConvMode = *DISABLE*, we have to start ADC locally first, to read an ADC value:
HAL_ADC_Start(&hadc1); … HAL_ADC_GetValue(&hadc1); … HAL_ADC_Stop(&hadc1);
If we set hadc1.Init.ContinuousConvMode = *ENABLE* , we should start ADC after the line of code:
MX_USB_DEVICE_Init();.
We convert the ADC voltage values U into a **char string** with procedure **char** ***my_ftoa(float**, **char***) and send this string via USB to a terminal program. For a conversion one can use sprintf(str, "%d", rawValue) too.

We extent our previous program with the yellow highlighted code.

```
#include "main.h"
#include "usb_device.h"
#include "usbd_cdc_if.h"//By reason of: CDC_Transmit_FS(uint8_t*, uint16_t);

volatile uint32_t rawValue = 0;
volatile float U = 0.0;
ADC_HandleTypeDef hadc1;
I2C_HandleTypeDef hi2c1;
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
static void MX_I2C1_Init(void);

void MCP4725_Mem_WriteReg(uint16_t);
uint16_t slen(const char*);
```

```
void add_txt(char* , char* );
char *my_ftoa(float, char*);
void USB_Info(char*);
void USB_Info_tft(char*, float, char*);

int main(void)
{
HAL_Init();
SystemClock_Config();
MX_GPIO_Init();
MX_ADC1_Init();
MX_I2C1_Init();
MX_USB_DEVICE_Init();

while (1){
  for(int n = 0; n < 4095; n += 9){
    MCP4725_Mem_WriteReg(n);
    HAL_Delay(20);
    HAL_ADC_Start(&hadc1);//local ADC-start
    while(HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY) != HAL_OK);
    rawValue = HAL_ADC_GetValue(&hadc1);
    HAL_ADC_Stop(&hadc1);//local ADC-stop
    U = 3.3 * rawValue / 4095.0;
    USB_Info_tft("U = ", U, "V");
  }
  for(int n = 4095; n >= 0; n -= 9){
    MCP4725_Mem_WriteReg(n);
    HAL_Delay(20);
    HAL_ADC_Start(&hadc1);
    while(HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY) != HAL_OK);
    rawValue = HAL_ADC_GetValue(&hadc1);
    HAL_ADC_Stop(&hadc1);
    U = 3.3 * rawValue / 4095.0;
    USB_Info_tft("U = ", U, "V");
  }
}
}

void MCP4725_Mem_WriteReg(uint16_t val)
{//0x62 is MCP4725-modul
    uint8_t data[] = { val >> 4, val << 4 };
    HAL_I2C_Mem_Write(&hi2c1, 0x62 << 1, 0x40, 1, data, 2, HAL_MAX_DELAY);
    HAL_Delay(1);
}

uint16_t slen(const char* s) {
    uint16_t i;
    for (i = 0; s[i] != 0; i++);
    return i;//s[0] not 0 then i=1;
}

void add_txt(char* out, char* in) {
   while (*out != 0) out++;
   while (*in != 0) {
      *out++ = *in++;
```

```
   }
   *out = 0;
}

char *my_ftoa(float val, char *str)
{
  char *cp; cp=str;
  int v, v0, rest, rest0;
  char c;
  if(val < 0){     //   cp=0
      val = -val;  //cp[0   ][1   ][2][3][4][5][6][7][8][9]
      *cp++ = '-'; //  [0: -] cp=1
  }
  v0 = (int)val; v=v0;
  //rest0=(int)((val-(int)val)*100000000);//genauer!
  rest0=(int)((val-(int)val)*10000);
  rest = rest0;
  do {
    v /= 10;
    cp++;
  } while(v != 0);
  do {
    rest /= 10;
    cp++;
  } while(rest != 0);
  cp++; //wegen ','
  *cp-- = 0;
  do {
    c = rest0 % 10;
    rest0 /= 10;
    c += '0';
    *cp-- = c;
  } while(rest0 != 0);
  *cp-- = ',';
  do {
    c = v0 % 10;
    v0 /= 10;
    c += '0';
    *cp-- = c;
  } while(v0 != 0);
  return  cp;
}

void USB_Info(char *str)
{
  char txt[64] = {};
  add_txt( txt, str);
  add_txt( txt, "\n\r");
  CDC_Transmit_FS((uint8_t *)txt, slen(txt));
}

void USB_Info_tft(char *str, float n, char *str1)
{
  char txt[64] = {}, h[32] = {};
  add_txt( txt, str);
```

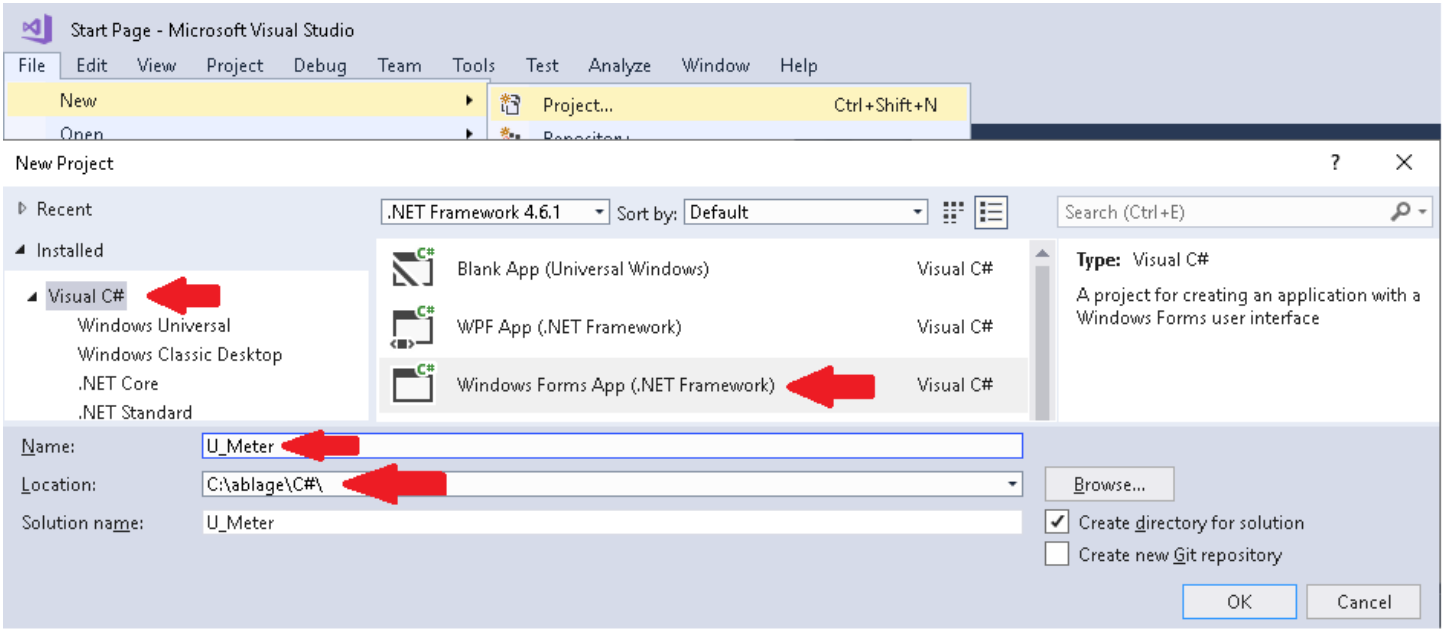
```
  my_ftoa(n, h);
  add_txt( txt, h);
  add_txt( txt, str1);
  add_txt( txt, "\n\r");
  CDC_Transmit_FS((uint8_t *)txt, slen(txt));
}
```

void SystemClock_Config(void) … and the remaining code until … **#endif** //P-End

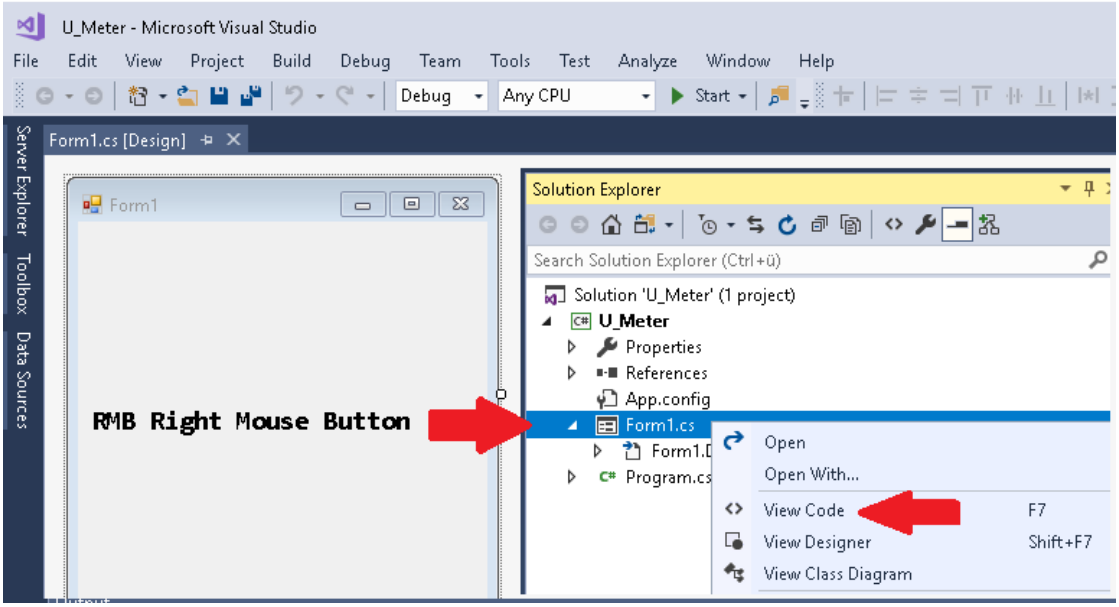
Later we want to draw as an application of the DC-Waveform Generator a current-voltage-characteristics of a 3.7V-bulb. For that we have to develop a plotting program. As an exercise to practice C# we now develop first a small voltage display – our own terminal program.

U-Meter with C#

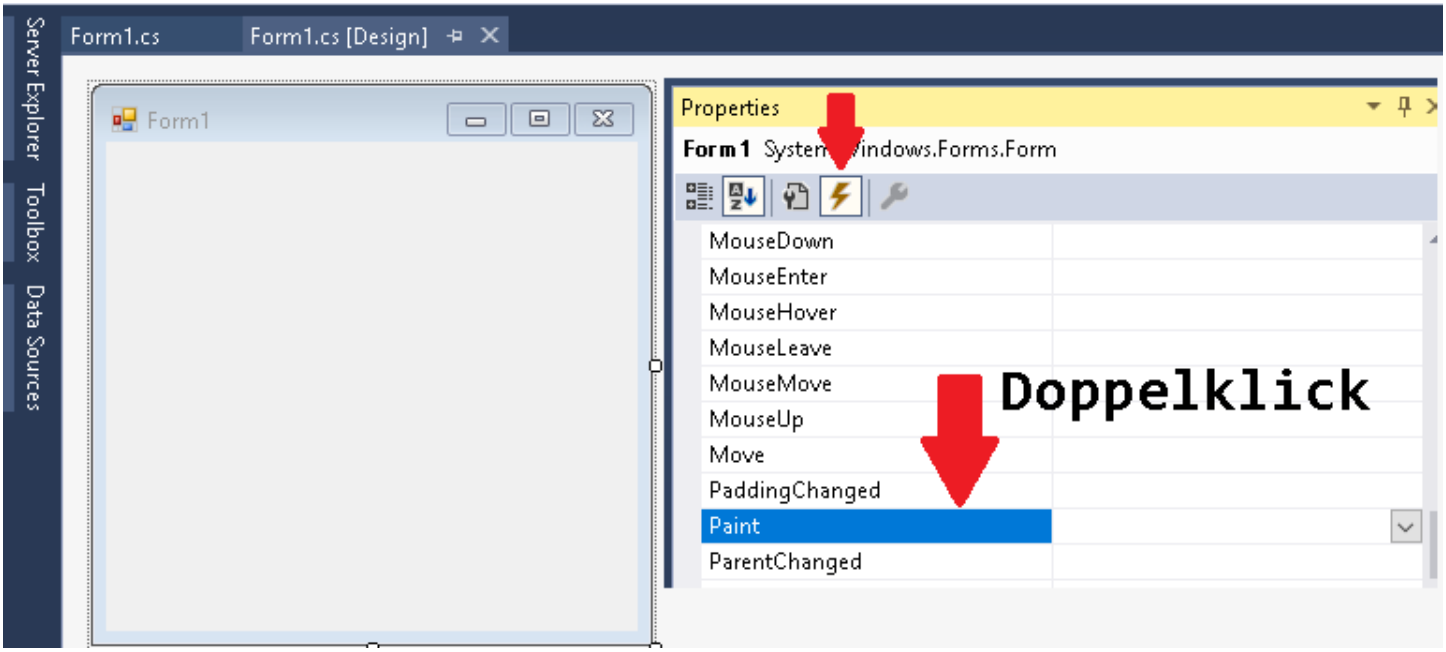
Create a project:



View Code:

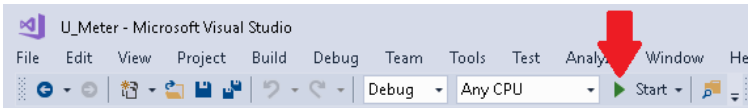


Add a Paint event:



For a first C# **Hello** program let's extent this Windows Forms Template with the gray highlighted lines of code:

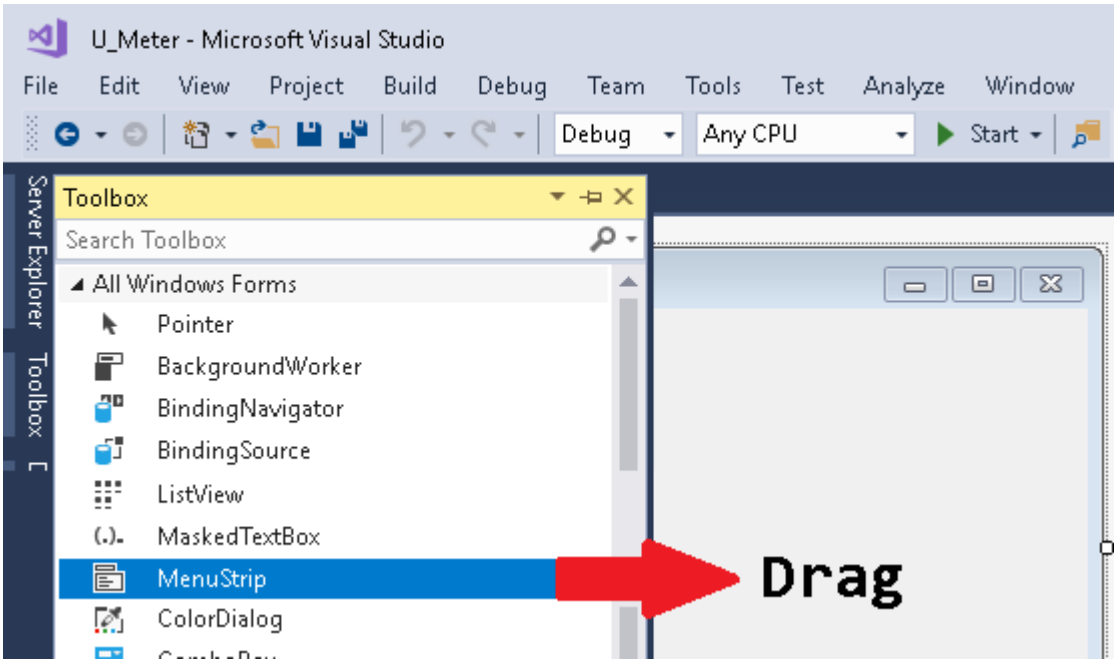
```
using System.Drawing;
using System.Windows.Forms;

namespace U_Meter
{
   public partial class Form1 : Form
   {
      Bitmap _backBuffer;

      public Form1()
      {
         DoubleBuffered = true; //faster drawing
         InitializeComponent();
      }

      private void Form1_Paint(object sender, PaintEventArgs e)
      {
         if (_backBuffer == null)
         {
            _backBuffer = new Bitmap(this.ClientSize.Width, this.ClientSize.Height);
         }
         Graphics g = Graphics.FromImage(_backBuffer);
         g.Clear(Color.Black);
         Font fn = new Font("Verdana", 50);
         Brush br = new SolidBrush(Color.LightGreen);
         g.DrawString("Hallo", fn, br, 30, 50);
         g.Dispose();
         e.Graphics.DrawImageUnscaled(_backBuffer, 0, 0);//Copy back buffer to screen
      }
```



Button zum compilieren und starten des Programms.

```
    }
}
```

We need more than this **Hallo** and insert a menu:

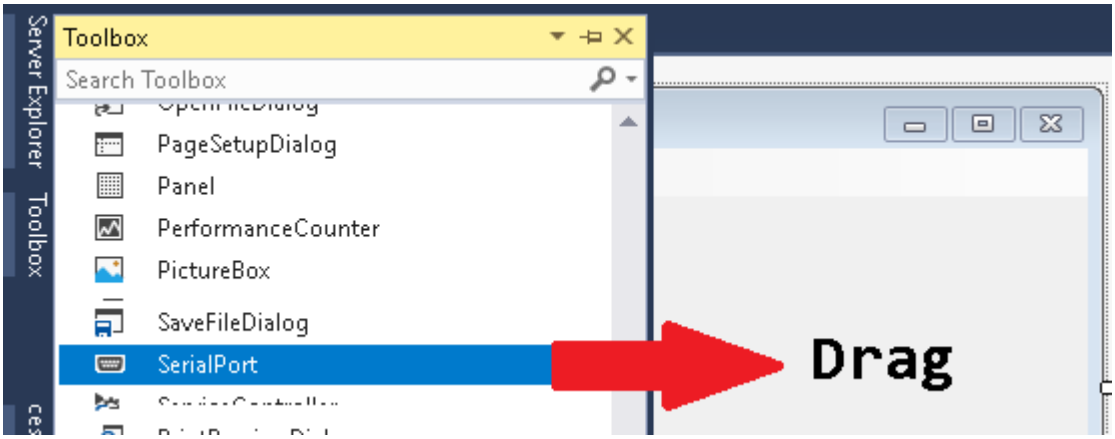


For testing this menu strip create File/Exit:

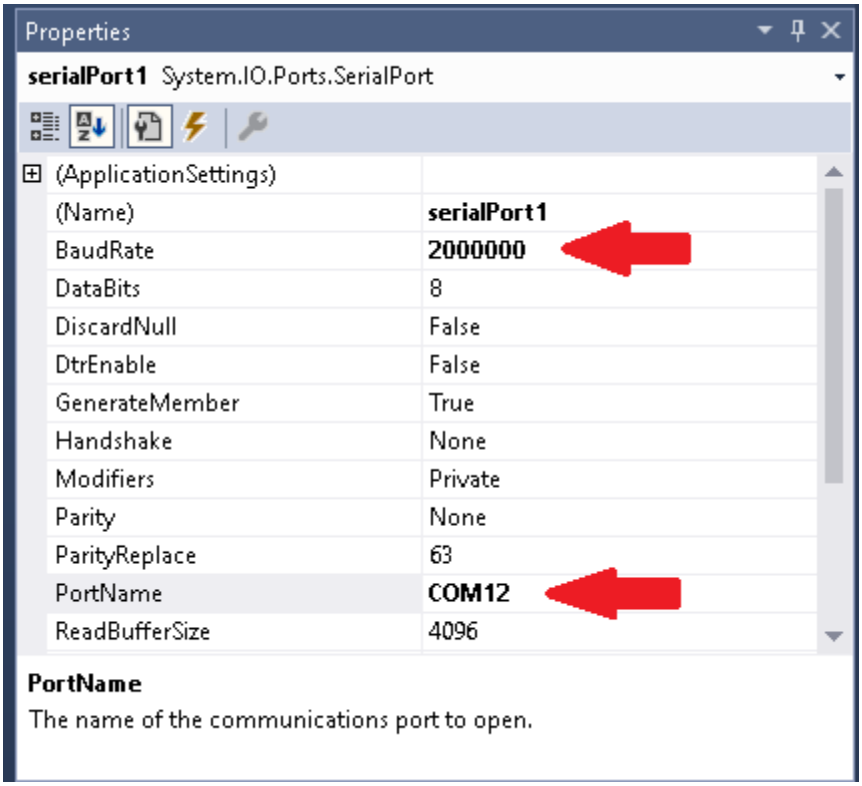
and breathe life into it with **Close();**

```
private void exitToolStripMenuItem_Click(object sender, System.EventArgs e)
{ Close(); }
```

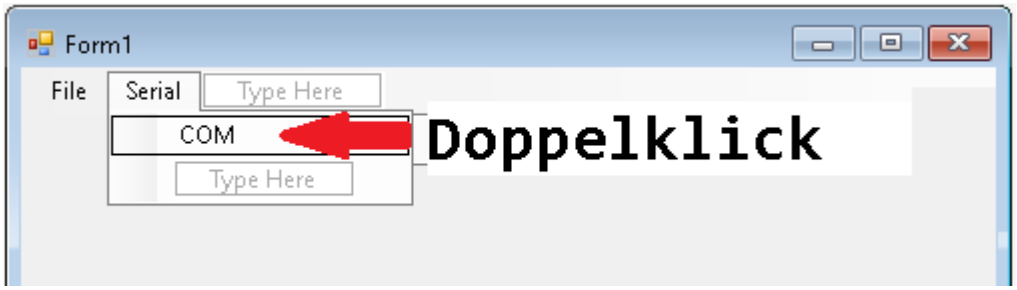
With the Toolbox on the left side we implement a Serial Port too:



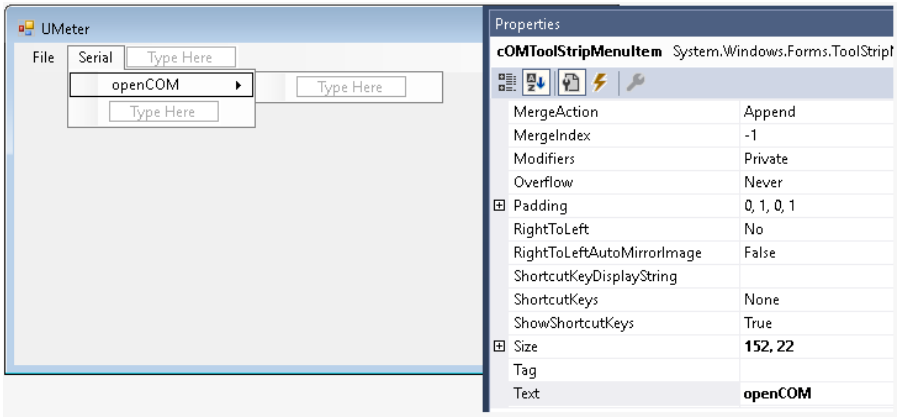
Enter Baud Rate and individual COM Port Number – we have COM12:



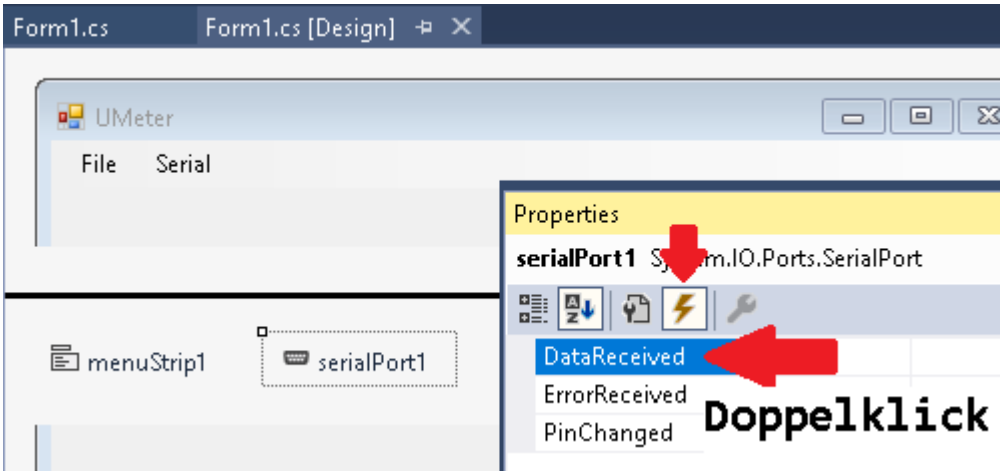
Let‘s extend the menu:



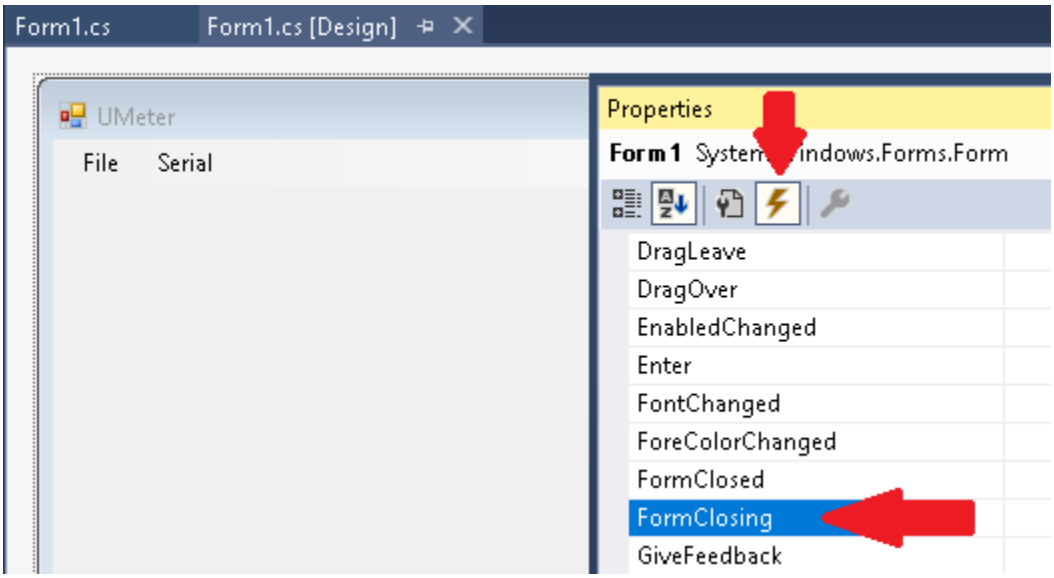
Inside the sheet **Properties** we can edit many attributes, such as change text **COM** in text **openCOM** or title **FORM1** to **UMeter**:



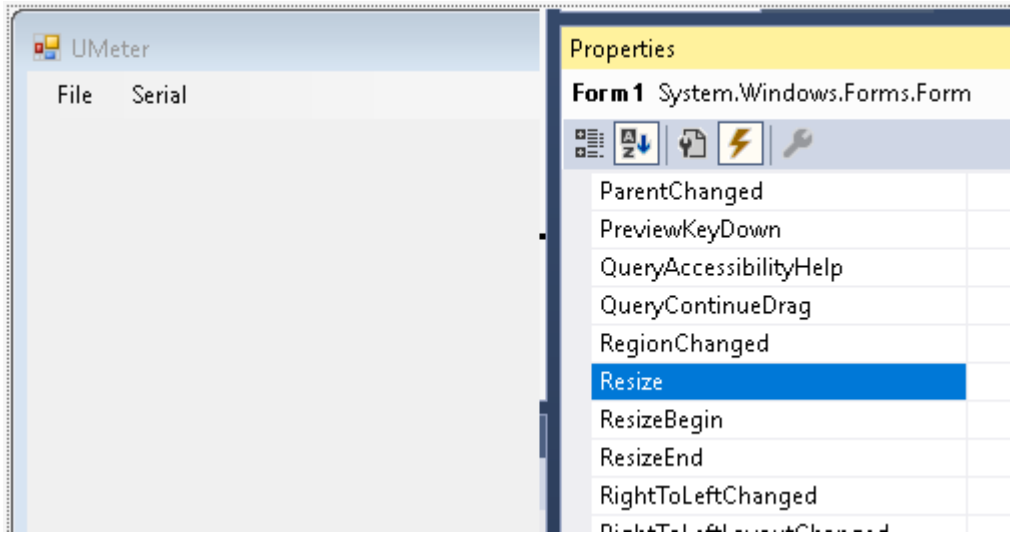
Receive data is an Event:



Evan to close an open COM Port is to be learnt:



Those who want to change the size of the application window have to introduce Event **RESIZE**:



The C# code of our program is now as follows:

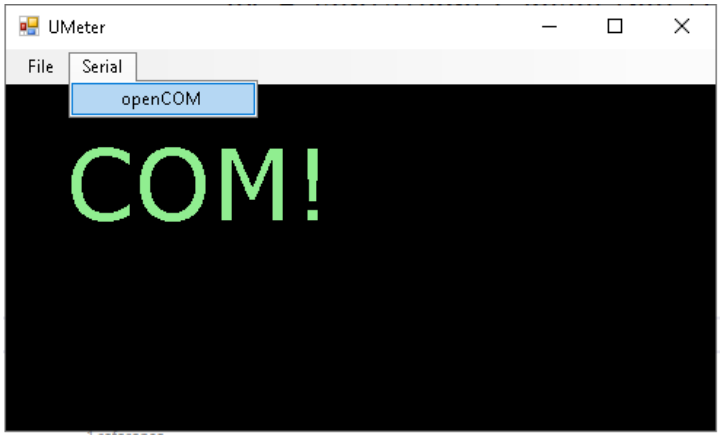
```
using System.Drawing;
using System.Windows.Forms;
using System;
```

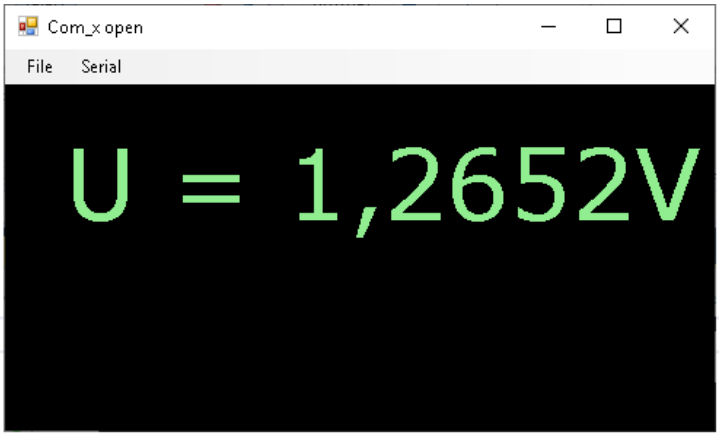
```
namespace U_Meter
{
   public partial class Form1 : Form
   {
      Bitmap _backBuffer;
      bool busy = false;
      String ex = "COM!";

      public Form1()
      {
         DoubleBuffered = true; //faster drawing
         InitializeComponent();
      }

      private void opencom(object sender, EventArgs e)
      {
         if (serialPort1 != null)
         {
            try
            {
               serialPort1.Open();
               serialPort1.DiscardInBuffer();
               serialPort1.DiscardOutBuffer();
               Text = "Com_x open";
               cOMToolStripMenuItem.Text = "closeCOM";
            }
            catch (Exception ex)
            {
               MessageBox.Show("Sonstiger Fehler: " + ex.Message);
            }
         }
      }

      private void closecom(object sender, EventArgs e)
      {
         if (serialPort1.IsOpen && !busy)
         {
            try
            {
               serialPort1.Close();
               this.Text = "Com_x closed";
               cOMToolStripMenuItem.Text = "openCOM";
            }
            catch (Exception ex)
            {
               MessageBox.Show("Sonstiger Fehler: " + ex.Message);
            }
         }
      }

      private void Form1_Paint(object sender, PaintEventArgs e)
      {
         if (_backBuffer == null)
         {
            _backBuffer = new Bitmap(this.ClientSize.Width, this.ClientSize.Height);
```

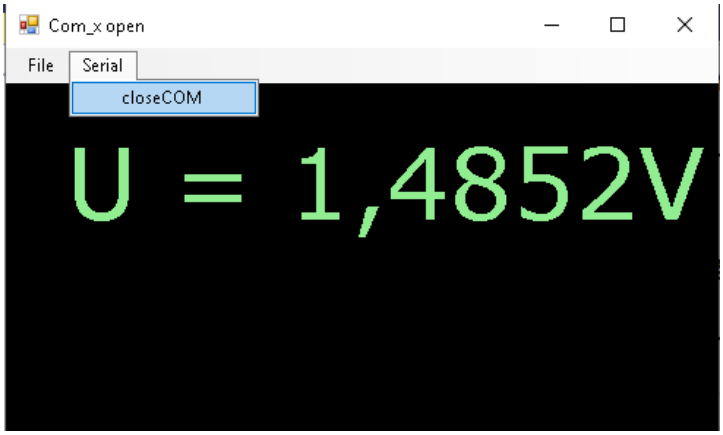
```
        }
        Graphics g = Graphics.FromImage(_backBuffer);
        g.Clear(Color.Black);
        Font fn = new Font("Verdana", 50);
        Brush br = new SolidBrush(Color.LightGreen);
        g.DrawString(ex, fn, br, 30, 50);
        g.Dispose();
        e.Graphics.DrawImageUnscaled(_backBuffer, 0, 0);//Copy back buffer to screen
    }

    private void exitToolStripMenuItem_Click(object sender, System.EventArgs e)
    {
        Close();
    }

    private void cOMToolStripMenuItem_Click(object sender, System.EventArgs e)
    {
        string txt = cOMToolStripMenuItem.Text;
        if (txt == "openCOM")
        {
            BeginInvoke(new EventHandler(opencom));
        }
        else
        {
            BeginInvoke(new EventHandler(closecom));
        }
    }

    private void serialPort1_DataReceived(object sender, System.IO.Ports.SerialDataReceivedEventArgs
e)
    {
        busy = true;
        try
        {
            ex = serialPort1.ReadLine();
        }
        catch (TimeoutException) { }
        try
        {
            serialPort1.ReadExisting();
        }
        catch (TimeoutException) { }
        busy = false;
        Invalidate();
    }

    private void Form1_FormClosing(object sender, FormClosingEventArgs e)
    {
        this.BeginInvoke(new EventHandler(closecom));
    }

    private void Form1_Resize(object sender, EventArgs e)
    {
        _backBuffer.Dispose();
        _backBuffer = null;
```

```
            Refresh();
        }
    }
}
```

and with a connected Black Pill you should see:

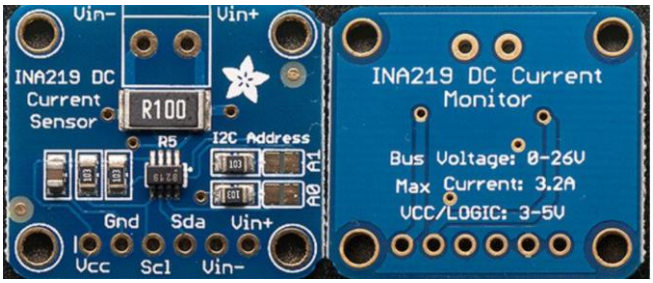




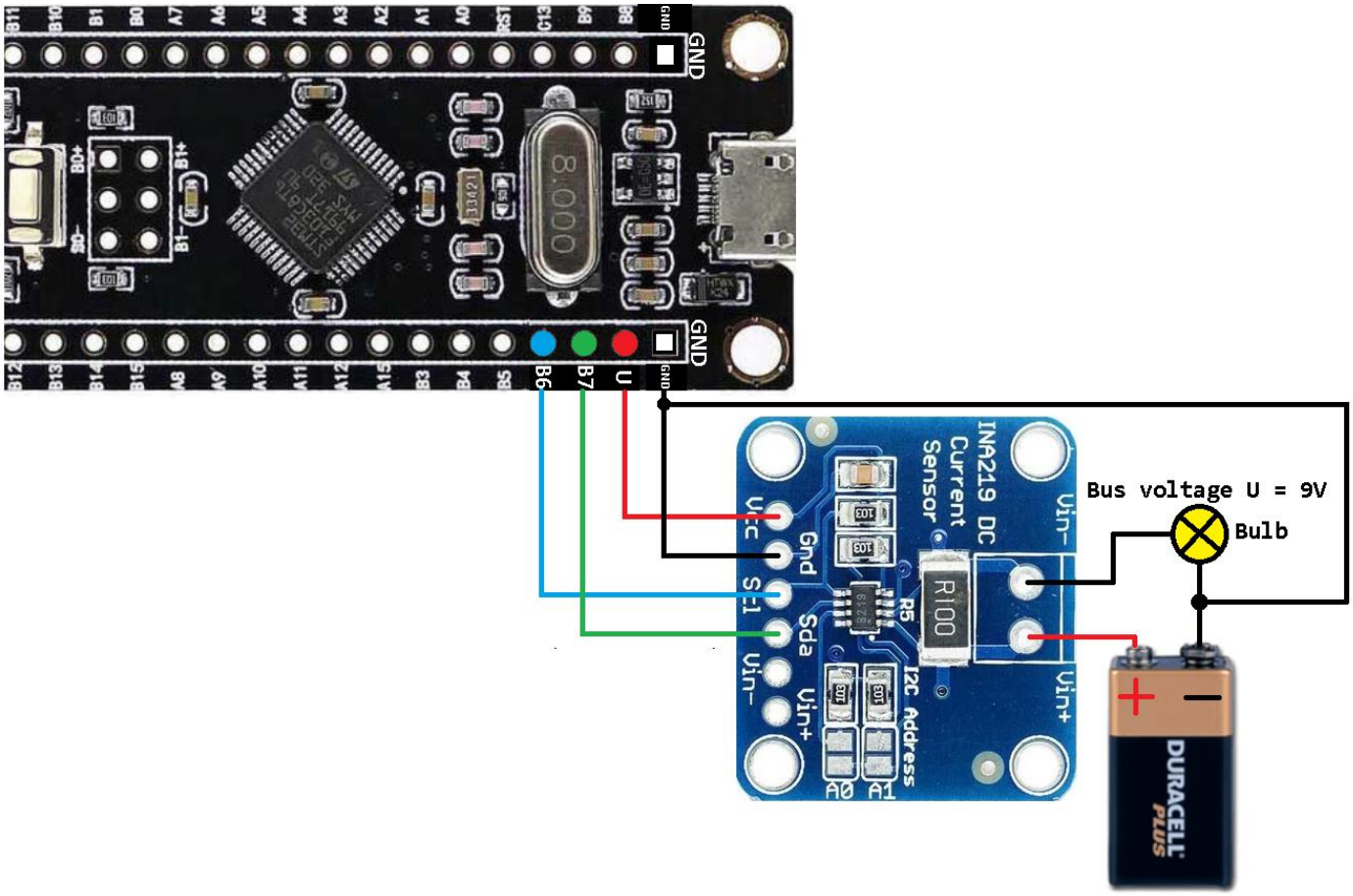


For now, we stop with our little C# training. Later we extend this terminal program to a plotting program to draw an U = U(I) graph, to demonstrate a variable rather than a constant resistor of a 3.7V bulb.

Module INA219

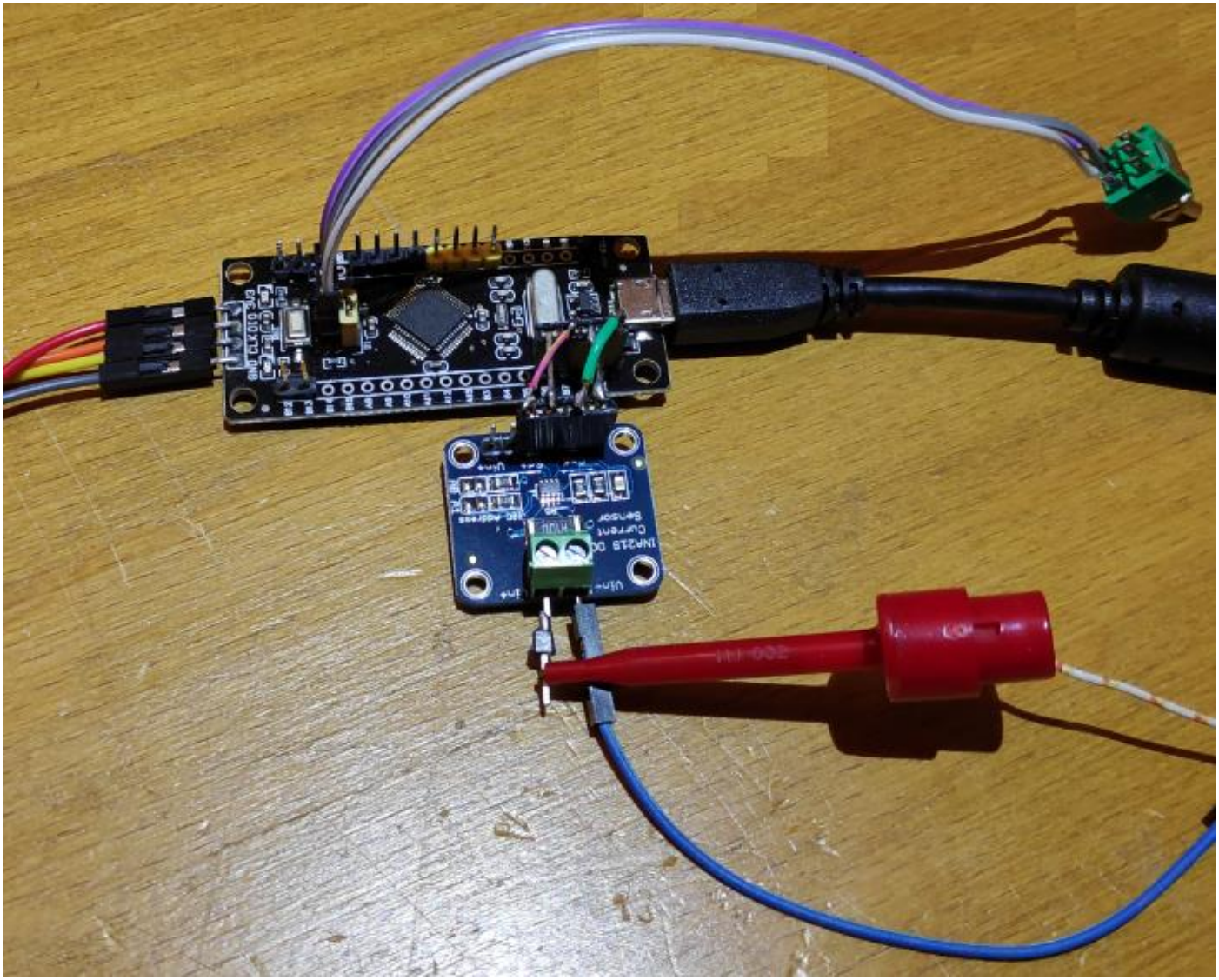


We record the U = U(I) curve with a chip INA219, manufactured by Texas Instruments: „The INA219 is a current shunt and power monitor with an I2C- or SMBUS-compatible interface. The device monitors both shunt voltage drop and **bus supply voltage**, with programmable conversion times and filtering. A programmable calibration value, combined with an internal multiplier, enables **direct readouts of current in amperes**.“



This module enables us to measure U = U(I) without to use an ADC of Black Pill.

Problem: How can we send the sequence $(I_n, U_n)_{n\in N}$ of measurements with USB coherent to a computer, i.e. without to disturb the dual pairing?



Module INA219 we have to configure, to calibrate, to write and to read. This we achieve with two procedures in addition – compared with MCP4725. We waive averaging by the module and configure INA219 to work with 12-Bit resolution, which are 4096 steps – from 0 up to 4095.

```
void ina219_Mem_WriteReg(uint8_t reg, uint16_t val)
{//0x40 is INA219-modul
        uint8_t data[] = { val >> 8, val };
        HAL_I2C_Mem_Write(&hi2c1, 0x40 << 1, reg, 1, data, 2, HAL_MAX_DELAY);
        HAL_Delay(1);
}

void ina219_Mem_ReadReg(uint8_t reg, uint16_t *val)
{//0x40 is INA219-modul; for BusVoltage reg=2; for Current reg=4
        uint8_t data[] = {0, 0};
        HAL_I2C_Mem_Read(&hi2c1, 0x40 << 1, reg, 1, data, 2, HAL_MAX_DELAY);
        *val = (data[0] << 8 ) | data[1];
}

void ina219_setCalibration_16V_400mA(void)
{//Divider_I_mA = 10 = 1000 / I_LSB; I_LSB = 100uA ≈ 400mA / 4096 pro Bit
        ina219_Mem_WriteReg(0x05, 4096);//calibration_Reg = 0x05, 12-Bit resolution
        ina219_Mem_WriteReg(0x00, 0b0000000110011111);//config_Reg = 0x00
}
```

The address of module INA219 is **0x40**, and the W/R-Bits makes this to **0x40 << 1**.

Table 2. Summary of Register Set `Value for Calibration without averaging is 4096`

POINTER ADDRESS HEX	REGISTER NAME	FUNCTION	POWER-ON RESET BINARY	POWER-ON RESET HEX	TYPE(1)
00	Configuration	All-register reset, settings for bus voltage range, PGA Gain, ADC resolution/averaging.	00111001 10011111	399F	R/$\overline{W}$
01	Shunt voltage	Shunt voltage measurement data.	Shunt voltage	—	R
02	Bus voltage	Bus voltage measurement data.	Bus voltage	—	R
03	Power(2)	Power measurement data.	00000000 00000000	0000	R
04	Current(2)	Contains the value of the current flowing through the shunt resistor.	00000000 00000000 needs Calibration	0000	R
05	Calibration	Sets full-scale range and LSB of current and power measurements. Overall system calibration.	00000000 00000000	0000	R/$\overline{W}$

(1) Type: **R** = Read only, **R/$\overline{W}$** = Read/Write.
(2) The Power register and Current register default to 0 because the Calibration register defaults to 0, yielding a zero current value until the Calibration register is programmed.

Configuration Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RST	—	BRNG	PG1	PG0	BADC 4	BADC 3	BADC 2	BADC 1	SADC 4	SADC 3	SADC 2	SADC 1	MODE 3	MODE 2	MODE 1
R/W-0	R/W-0	R/W-1	R/W-1	R/W-1	R/W-0	R/W-0	R/W-1	R/W-1	R/W-0	R/W-0	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

RST: **Reset Bit**

Bit 15 Setting this bit to '1' generates a system reset that is the same as power-on reset. Resets all registers to default values; this bit self-clears. **RESET = 0x399F**

BRNG: **Bus Voltage Range**

Bit 13 0 = 16V FSR 1 = 32V FSR (default value)

PG: **PGA (Shunt Voltage Only)**

Bits 11, 12 Sets PGA gain and range. Note that the PGA defaults to ÷8 (320mV range). Table 4 shows the gain and range for the various product gain settings.

PG1	PG0	GAIN	Range
0	0	1	±40 mV

BADC4	BADC3	BADC2	BADC1	Mode/Samples
0	0	0	0	9 bit res.: 0 .. 511
0	0	0	1	10 bit res.: 0 .. 1023
0	0	1	0	11 bit res.: 0 .. 2047
0	0	1	1	12 bit res.: 0 .. 4095

SADC4	SADC3	SADC2	SADC1	Mode/Samples	Conversion Time
1	0	0	0	12 bit	532 µs 1000 or 0011, TI.pdf

MODE3	MODE2	MODE1	MODE
0	0	0	Power-down
1	0	1	Shunt voltage, continuous
1	1	0	Bus voltage, continuous
1	1	1	Shunt and bus, continuous

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	1	0	0	1	1	1	1	1

0b0000000110011111

Bus voltage register (0x02) and current register (0x04) contain still to be processed values, to get extern voltage U in V and current I in mA. We are not interested in negative currents, but we take care of the minus sign in the code.

```
float ina219_get_I_mA(void)
{
	uint16_t tmp;
	ina219_Mem_ReadReg(0x04 , &tmp);//I_Reg = 0x04
	if(getBit(15, tmp)){//0b1000000000000000 is minus sign; TI_INA219.pdf
		tmp = ~tmp;//take complement
		tmp += 1;//and add one, to calc I=-0.123mA
		return tmp / -10.0; //Divider_I_mA = 10;
	} else return tmp / 10.0; //Divider_I_mA = 10;
}

float ina219_get_UBus_V(void) {//U=12.34V
  uint16_t tmp;
  ina219_Mem_ReadReg(0x02 , &tmp);//U_Reg = 0x02
  tmp = tmp >> 3;//TI_INA219_pdf:Bus Voltage Register contents must be shifted right by three bits.
  return tmp * 0.004;
}
```

TI_INA219.pdf: “When reading from the INA219, the last value stored in the register pointer by a **write operation** determines which register is read during a read operation. … **This write** is accomplished by issuing a slave address byte with the R/W bit LOW, followed by the register pointer byte. **No additional data are required**.”
We translate this note into C-code:

```
while(HAL_I2C_IsDeviceReady(&hi2c1, 0x40 << 1, 0x04, HAL_MAX_DELAY) != HAL_OK);
```

to then read the current value and

```
while(HAL_I2C_IsDeviceReady(&hi2c1, 0x40 << 1, 0x02, HAL_MAX_DELAY) != HAL_OK);
```

to then read the bus voltage.

We drive module INA219 with the blue highlighted lines of code:

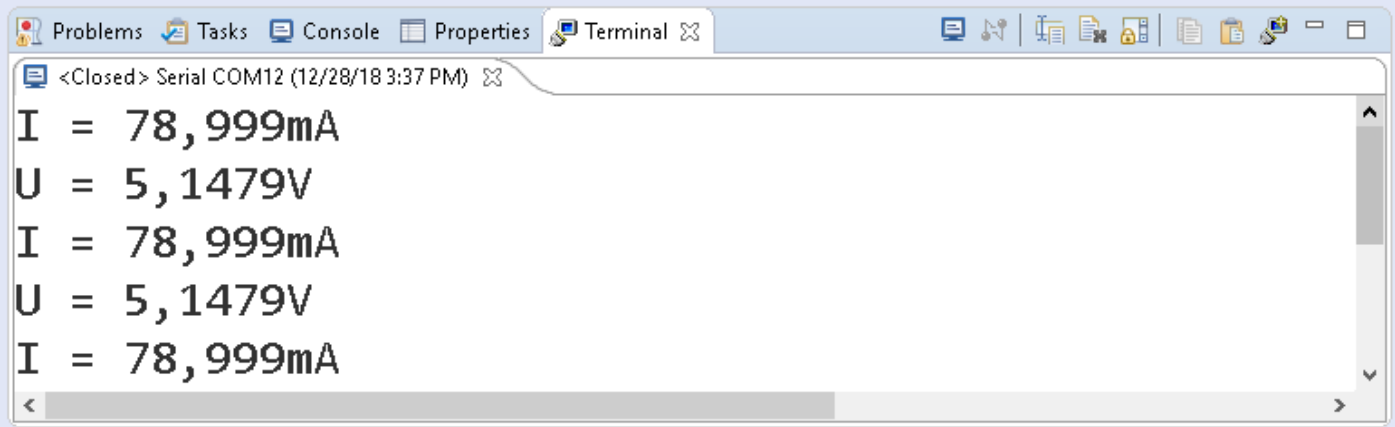
```
#include "main.h"
#include "usb_device.h"
#include "usbd_cdc_if.h"//By reason of: CDC_Transmit_FS(uint8_t*, uint16_t);

#define getBit(bit, value) ((value >> bit) & 0x00000001)

volatile uint32_t rawValue = 0;
volatile float U = 0.0;
ADC_HandleTypeDef hadc1;
I2C_HandleTypeDef hi2c1;
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
static void MX_I2C1_Init(void);

void MCP4725_Mem_WriteReg(uint16_t);
void ina219_Mem_WriteReg(uint8_t, uint16_t);
void ina219_Mem_ReadReg(uint8_t, uint16_t*);
void ina219_setCalibration_16V_400mA(void);
float ina219_get_I_mA(void);
```

```
float ina219_get_UBus_V(void);
uint16_t slen(const char*);
void add_txt(char* , char* );
char *my_ftoa(float, char*);
void USB_Info(char*);
void USB_Info_tft(char*, float, char*);

int main(void)
{
HAL_Init();
SystemClock_Config();
MX_GPIO_Init();
MX_ADC1_Init();
MX_I2C1_Init();
MX_USB_DEVICE_Init();
ina219_setCalibration_16V_400mA();
HAL_Delay(500);
while (1){
  while(HAL_I2C_IsDeviceReady(&hi2c1, 0x40 << 1, 0x04, HAL_MAX_DELAY) != HAL_OK);
  USB_Info_tft("I = ", ina219_get_I_mA(), "mA");
  HAL_Delay(1000);
  while(HAL_I2C_IsDeviceReady(&hi2c1, 0x40 << 1, 0x02, HAL_MAX_DELAY) != HAL_OK);
  USB_Info_tft("U = ", ina219_get_UBus_V(), "V");
  HAL_Delay(1000);
}
}

void MCP4725_Mem_WriteReg(uint16_t val)
{//0x62 is MCP4725-modul
    uint8_t data[] = { val >> 4, val << 4 };
    HAL_I2C_Mem_Write(&hi2c1, 0x62 << 1, 0x40, 1, data, 2, HAL_MAX_DELAY);
    HAL_Delay(1);
}

void ina219_Mem_WriteReg(uint8_t reg, uint16_t val)
{//0x40 is INA219-modul
    uint8_t data[] = { val >> 8, val };
    HAL_I2C_Mem_Write(&hi2c1, 0x40 << 1, reg, 1, data, 2, HAL_MAX_DELAY);
    HAL_Delay(1);
}

void ina219_Mem_ReadReg(uint8_t reg, uint16_t *val)
{//0x40 is INA219-modul; for BusVoltage reg=2; for Current reg=4
    uint8_t data[] = {0, 0};
    HAL_I2C_Mem_Read(&hi2c1, 0x40 << 1, reg, 1, data, 2, HAL_MAX_DELAY);
    *val = (data[0] << 8 ) | data[1];
}

void ina219_setCalibration_16V_400mA(void)
{//Divider_I_mA = 10 = 1000 / I_LSB; I_LSB = 100uA ~ 400mA / 4096 pro Bit
    ina219_Mem_WriteReg(0x05, 4096);//calibration_Reg = 0x05, 12-Bit resolution
    ina219_Mem_WriteReg(0x00, 0b0000000110011111);//config_Reg = 0x00
}

float ina219_get_I_mA(void)
```

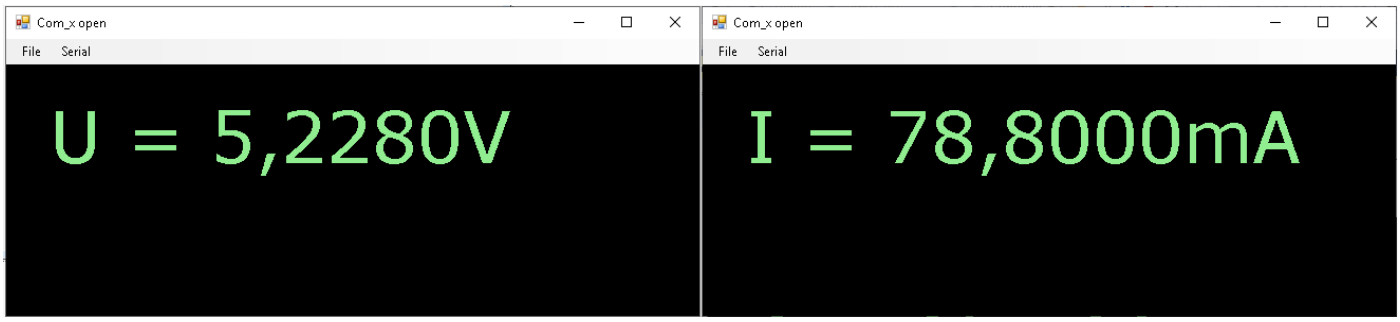
```
{
    uint16_t tmp;
    ina219_Mem_ReadReg(0x04 , &tmp);//I_Reg = 0x04
    if(getBit(15, tmp)){//0b1000000000000000 is minus sign; TI_INA219.pdf
        tmp = ~tmp;//take complement
        tmp += 1;//and add one, to calc I=-0.123mA
        return tmp / -10.0; //Divider_I_mA = 10;
    } else return tmp / 10.0; //Divider_I_mA = 10;
}


float ina219_get_UBus_V(void) {//U=12.34V
  uint16_t tmp;
  ina219_Mem_ReadReg(0x02 , &tmp);//U_Reg = 0x02
  tmp = tmp >> 3;//TI_INA219_pdf:Bus Voltage Register contents must be shifted right by three bits.
  return tmp * 0.004;
}
```

uint16_t **slen**(**const char*** s) { … und der Rest bis … **#endif** //P-End

The result is displayed by the terminal program included in TrueSTUDIO:



You get this output with the C# program, we developed above:



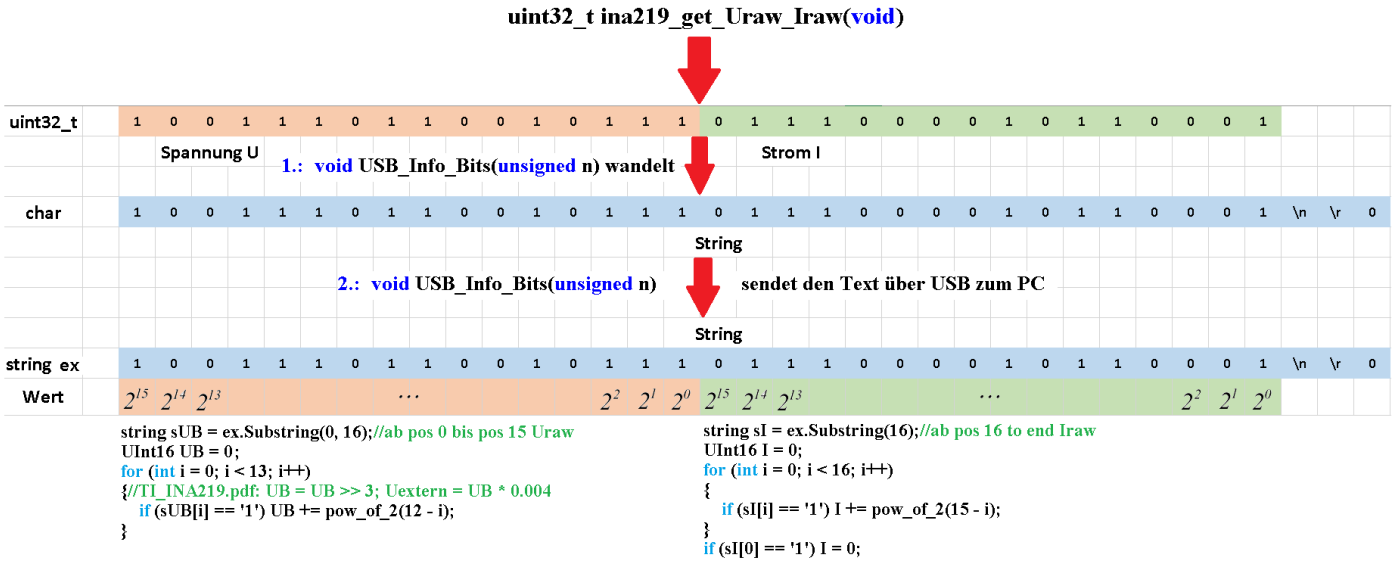
An uint32_t variable helps us to solve the problem of coherence to plot U = U(I). We put both values – I-value and U-value – together in this one value:

uint32_t ina219_get_Uraw_Iraw(void) {

```
  uint16_t x=0, y=0;
  while(HAL_I2C_IsDeviceReady(&hi2c1, 0x40 << 1, 0x02, HAL_MAX_DELAY) != HAL_OK);
  ina219_Mem_ReadReg(0x02 , &x);//U_Reg = 0x02
  while(HAL_I2C_IsDeviceReady(&hi2c1, 0x40 << 1, 0x04, HAL_MAX_DELAY) != HAL_OK);
  ina219_Mem_ReadReg(0x04 , &y);//I_Reg = 0x04
  return (x << 16) | y;
}
```

Subsequently we put one bit after another of the uint32_t number in a **char** array and send the content via USB to a computer.

```
void USB_Info_Bits(unsigned n)
{
  char txt[64] = {};
  for(int i = 31; i >= 0; i--){
     if(getBit(i, n)) add_txt( txt, "1"); else add_txt( txt, "0");
  }//uint8_t i = 31; causes USB-crash because of i==0 so i-- not def.
  add_txt( txt, "\n\r");
  CDC_Transmit_FS((uint8_t *)txt, slen(txt));
}
```



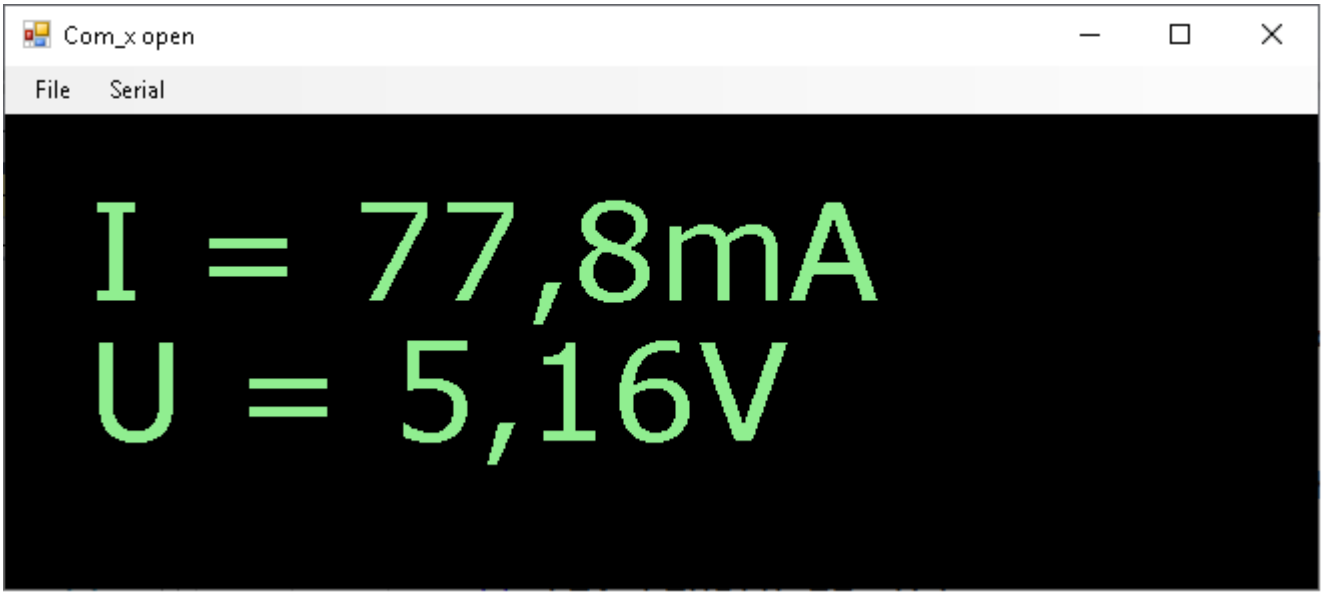
Inside a C# program we obtain bus voltage and current in the same way, used in the Black Pill program with UB*0.004 respectively I/10.0. The Black Pill program for itself we change as follows:

```
#include "main.h"
#include "usb_device.h"
#include "usbd_cdc_if.h"//By reason of: CDC_Transmit_FS(uint8_t*, uint16_t);

#define getBit(bit, value) ((value >> bit) & 0x00000001)

volatile uint32_t rawValue = 0;
volatile float U = 0.0;
ADC_HandleTypeDef hadc1;
I2C_HandleTypeDef hi2c1;
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
static void MX_I2C1_Init(void);
```

```
void MCP4725_Mem_WriteReg(uint16_t);
void ina219_Mem_WriteReg(uint8_t, uint16_t);
void ina219_Mem_ReadReg(uint8_t, uint16_t*);
void ina219_setCalibration_16V_400mA(void);
float ina219_get_I_mA(void);
float ina219_get_UBus_V(void);
uint32_t ina219_get_Uraw_Iraw(void);
uint16_t slen(const char*);
void add_txt(char* , char* );
char *my_ftoa(float, char*);
void USB_Info(char*);
void USB_Info_tft(char*, float, char*);
void USB_Info_Bits(unsigned);

int main(void)
{
HAL_Init();
SystemClock_Config();
MX_GPIO_Init();
MX_ADC1_Init();
MX_I2C1_Init();
MX_USB_DEVICE_Init();
ina219_setCalibration_16V_400mA();
HAL_Delay(500);
while (1){
  USB_Info_Bits(ina219_get_Uraw_Iraw());
  HAL_Delay(1000);
}
}

void MCP4725_Mem_WriteReg(uint16_t val) {…}

void ina219_Mem_WriteReg(uint8_t reg, uint16_t val) {…}

void ina219_Mem_ReadReg(uint8_t reg, uint16_t *val) {…}

void ina219_setCalibration_16V_400mA(void){…}

float ina219_get_I_mA(void){…}

float ina219_get_UBus_V(void) {…}

uint32_t ina219_get_Uraw_Iraw(void) {
  uint16_t x=0, y=0;
  while(HAL_I2C_IsDeviceReady(&hi2c1, 0x40 << 1, 0x02, HAL_MAX_DELAY) != HAL_OK);
  ina219_Mem_ReadReg(0x02 , &x);//U_Reg = 0x02
  while(HAL_I2C_IsDeviceReady(&hi2c1, 0x40 << 1, 0x04, HAL_MAX_DELAY) != HAL_OK);
  ina219_Mem_ReadReg(0x04 , &y);//I_Reg = 0x04
  return (x << 16) | y;
}

uint16_t slen(const char* s) {…}

void add_txt(char* out, char* in) {…}
```

```
char *my_ftoa(float val, char *str) {…}

void USB_Info(char *str) {…}

void USB_Info_tft(char *str, float n, char *str1) {…}

void USB_Info_Bits(unsigned n)
{
  char txt[64] = {};
  for(int i = 31; i >= 0; i--){
    if(getBit(i, n)) add_txt( txt, "1"); else add_txt( txt, "0");
  }
  add_txt( txt, "\n\r");
  CDC_Transmit_FS((uint8_t *)txt, slen(txt));
}

void SystemClock_Config(void) … und der Rest bis … #endif //P-End
```

Obviously, only our still easy to modify C# terminal program works with the above Black-Pill program, because the others do not know about our bit shiftings. We get this output with the following C# code changes:

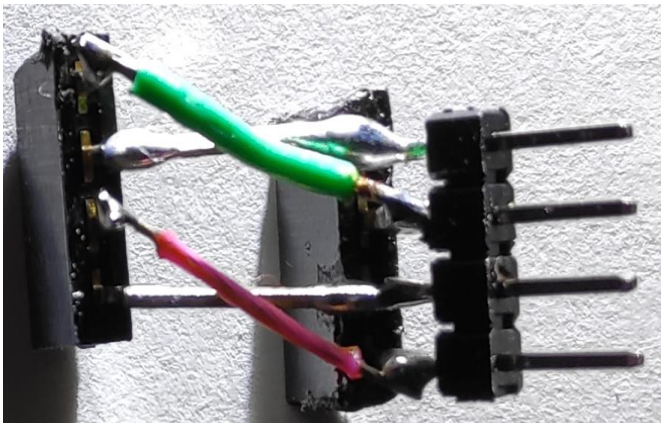


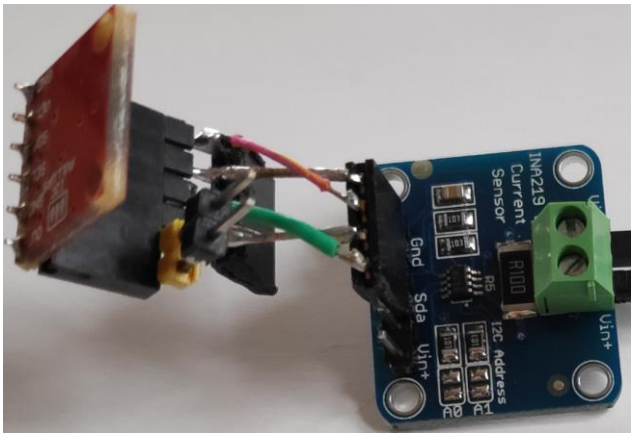

```
using System.Drawing;
using System.Windows.Forms;
using System;

namespace U_Meter
{
   public partial class Form1 : Form
   {
      Bitmap _backBuffer;
      bool busy = false;
      String ex = "COM!";

      public Form1()
      {
```

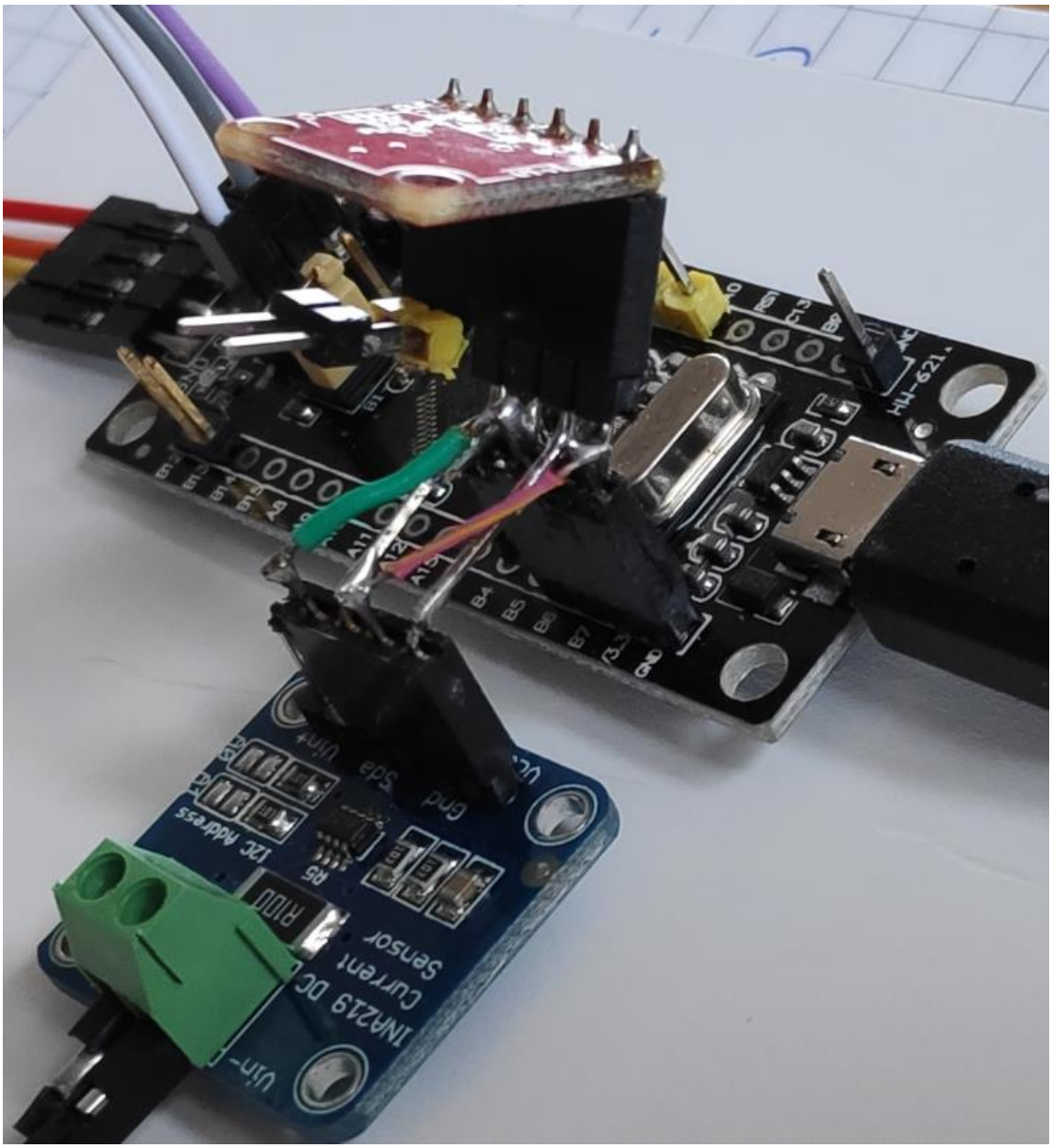
```
        DoubleBuffered = true; //faster drawing
        InitializeComponent();
    }

    private void opencom(object sender, EventArgs e) {…}

    private void closecom(object sender, EventArgs e) {…}

    UInt16 pow_of_2(int n)
    {
        return (UInt16)(1 << n);
    }

    private void Form1_Paint(object sender, PaintEventArgs e)
    {
        if (_backBuffer == null)
        {
            _backBuffer = new Bitmap(this.ClientSize.Width, this.ClientSize.Height);
        }
        Graphics g = Graphics.FromImage(_backBuffer);
        g.Clear(Color.Black);
        Font fn = new Font("Verdana", 50);
        Brush br = new SolidBrush(Color.LightGreen);
        if (ex.Length == 32)
        {
            string sUB = ex.Substring(0, 16);//ab pos 0 to pos 15
            string sI = ex.Substring(16);//ab pos 16 to end
            UInt16 UB = 0, I = 0;
            for (int i = 0; i < 13; i++)
            {//TI_INA219.pdf: UB = UB >> 3; Uextern = UB * 0.004
                if (sUB[i] == '1') UB += pow_of_2(12 - i);
            }
            for (int i = 0; i < 16; i++)
            {
                if (sI[i] == '1') I += pow_of_2(15 - i);
            }
            if (sI[0] == '1') I = 0;
            g.DrawString("I = " + (I/10.0).ToString() + "mA", fn, br, 30, 50);
            g.DrawString("U = " + (UB * 0.004).ToString() + "V", fn, br, 30, 120);
        }
        g.Dispose();
        e.Graphics.DrawImageUnscaled(_backBuffer, 0, 0);//Copy back buffer to screen
    }

    private void exitToolStripMenuItem_Click(object sender, System.EventArgs e) {…}

    private void cOMToolStripMenuItem_Click(object sender, System.EventArgs e) {…}

    private void serialPort1_DataReceived(object sender, System.IO.Ports.SerialDataReceivedEventArgs e) {…}

    private void Form1_FormClosing(object sender, FormClosingEventArgs e) {…}

    private void Form1_Resize(object sender, EventArgs e) {…}
  }
}
```

Black Pill with MCP4725- and INA219-module at I2C1

Now we come to the announced xy-curve – here: IU-curve – with data, sent from the Black Pill.

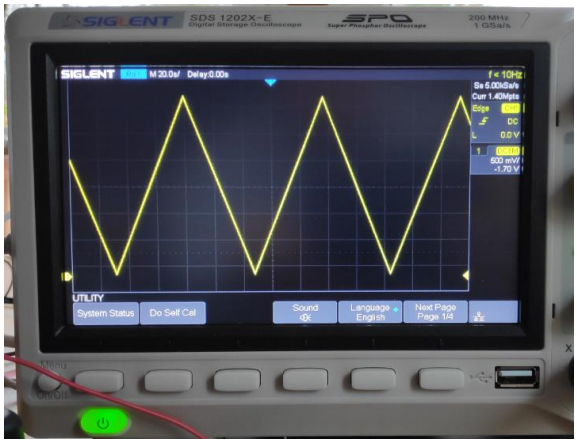






On the same I2C bus, both modules must have different addresses, which is the case with 0x62 for MCP4725 and 0x40 for INA219.
In the Black Pill program we have to change almost nothing, except in the while loop in **main {...}**, which now looks like this:

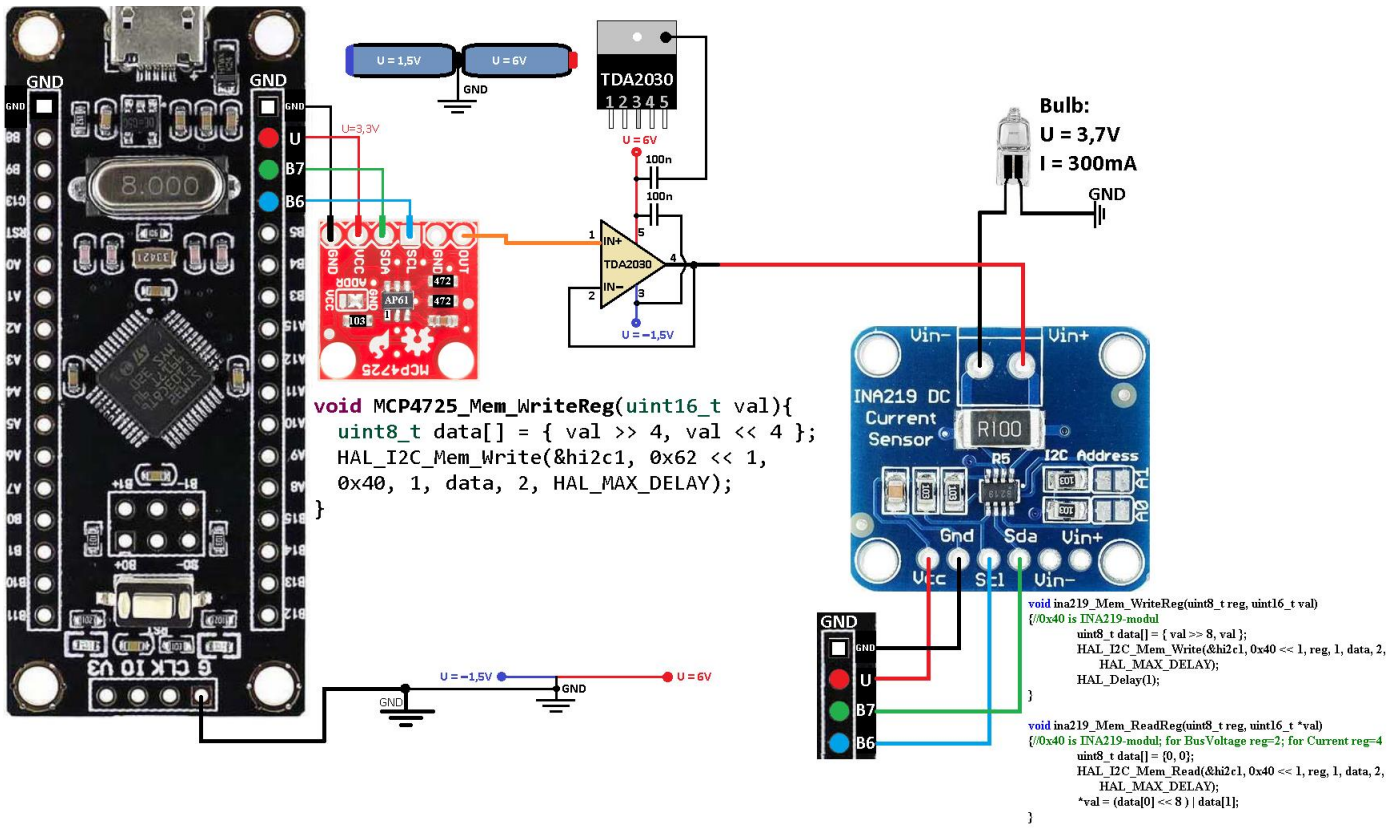
```
int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_USB_DEVICE_Init();
    MX_I2C1_Init();

    ina219_setCalibration_16V_400mA();
    HAL_Delay(500);

    while (1){
       for(int n = 0; n <= 4095; n += 9){
        MCP4725_Mem_WriteReg(n);
        USB_Info_Bits(ina219_get_Uraw_Iraw());
        //HAL_Delay(50);
       }
       for(int n = 4086; n >= 0; n -= 9){
        MCP4725_Mem_WriteReg(n);
        USB_Info_Bits(ina219_get_Uraw_Iraw());
        //HAL_Delay(50);
       }
    }
}
```



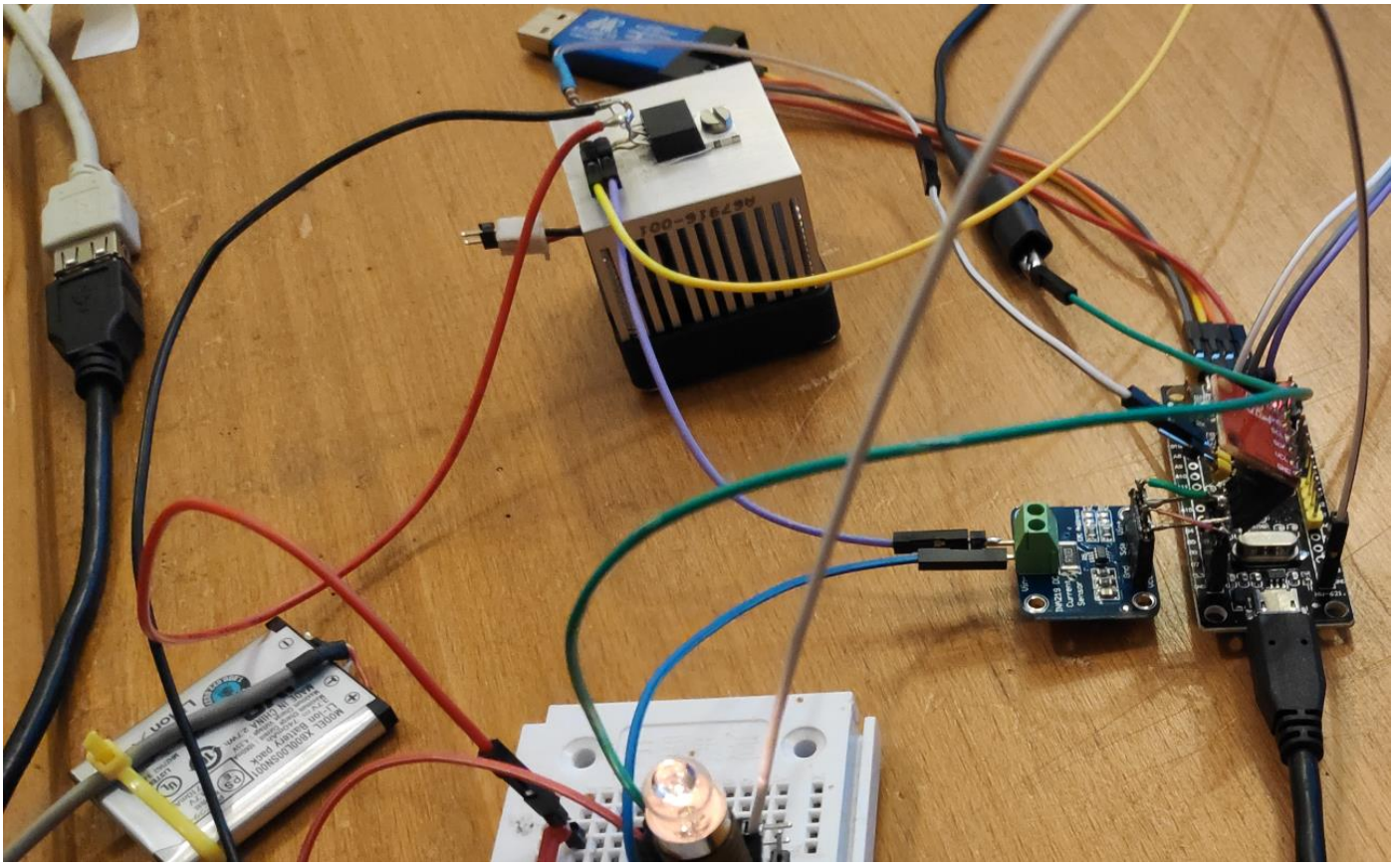
With this MCP4725-DAC voltage ramp, which rises linearly from U = 0 V (GND) up to U = 3.3V, and then linearly decay again to GND level, we feed the TDA2030 power amplifier IC.

The power supply of the power amplifier TDA2030 is arbitrary within wide limits. Instead of the U = –1.5V we use as shown in the photo below an old mobile phone battery with U = –3.7V at Pin3, and at Pin5 are U = +6V from a laboratory power supply.

The circuit with the crucial I2C procedures:



The TDA2030 power amplifier is mounted on the heat sink and can be controlled directly without the R = 100kΩ resistor.



In the following C# plotting program we need transformation functions, e.g. Transfer a curve drawn on a piece of paper to the dimensions of the monitor. We need a coordinate system, a plot procedure, and a list in which we store the (U, I) values, sent by the Black Pill. We draw this constantly updated list continuously as an IU-curve. All this we have to do without the Visual Studio Designer.

Plotting Program

Step 1: Coordinate transformation functions int xX(double x) and int yY(double y) as well as the Koordinatensystem(Graphics g, double dx, double dy, double hier_xAchse, double hier_yAchse)

We extend our terminal program accordingly:

```
using System.Collections.Generic;


public partial class Form1 : Form
{
    bool busy = false;
    String ex = "COM!";
    Bitmap _backBuffer;
    double xmin, xmax, ymin, ymax, yA = 0, xA = 0;
    double xStreckfaktor, yStreckfaktor;
    List<double> LI, LU;

public Form1()  {
    xmax = 150;
    xmin = -5;
    ymax = 4;
    ymin = -0.5;
    LI = new List<double>();
    LU = new List<double>();
… }

public int xX(double x) {
    return (int)Math.Round(xStreckfaktor * (x - xmin));
}
public int yY(double y) {
    return (int)Math.Round(yStreckfaktor * (ymax - y));
}

public void Koordinatensystem(Graphics g, double dx, double dy, double hier_xAchse, double hier_yAchse)
{
    Font fn = new Font("Verdana", 12);
    Brush br = new SolidBrush(Color.DimGray);
    Pen pen = new Pen(Color.Red, 1);

    if (LU.Count > 20)
    {
        xmax = 0;
        for (int i = 0; i < LI.Count; i++)
            if (xmax < LI[i]) xmax = LI[i];
        xmax = 1.1 * xmax;
        xmin = -0.05 * xmax;
        ymax = 0;
        for (int i = 0; i < LU.Count; i++)
            if (ymax < LU[i]) ymax = LU[i];
        ymax = 1.1 * ymax;
```

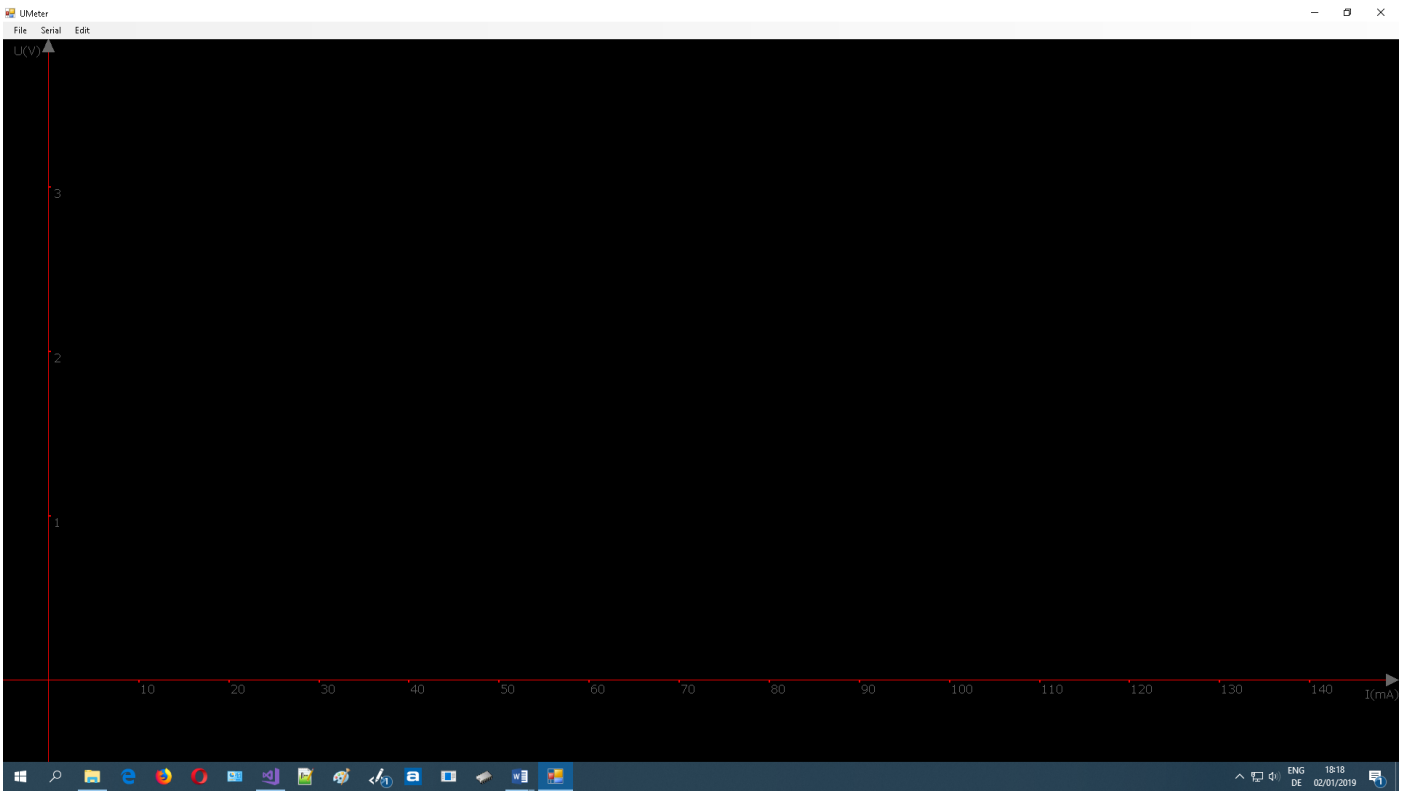
```
    ymin = -0.05 * ymax;
}
xStreckfaktor = this.ClientRectangle.Width / (xmax - xmin);
yStreckfaktor = this.ClientRectangle.Height / (ymax - ymin);
Point[] pxA =
{
    new Point(xX(xmax),yY(hier_xAchse)),
    new Point(xX(xmax) - 18,yY(hier_xAchse) - 9),
    new Point(xX(xmax) - 18,yY(hier_xAchse) + 9),
    new Point(xX(xmax),yY(hier_xAchse))
};
int h = menuStrip1.Height;
Point[] pyA =
{
    new Point(xX(hier_yAchse) + 9, yY(ymax) + 18 + h),
    new Point(xX(hier_yAchse), yY(ymax)-2 + h),
    new Point(xX(hier_yAchse) - 9, yY(ymax) + 18 + h),
    new Point(xX(hier_yAchse) + 9, yY(ymax) + 18 + h)
};

g.DrawLine(pen, xX(xmin), yY(hier_xAchse), xX(xmax), yY(hier_xAchse));//x-Achse
g.FillPolygon(br, pxA);//Pfeil der x-Achse
g.DrawString("I(mA)", fn, br, xX(xmax)-50, yY(hier_xAchse) + 10);
g.DrawLine(pen, xX(hier_yAchse), yY(ymin), xX(hier_yAchse), yY(ymax));//y-Achse
g.FillPolygon(br, pyA);//Pfeil der y-Achse
g.DrawString("U(V)", fn, br, xX(hier_yAchse) - 50, yY(ymax) + 30);

double x = dx;
while (x < xmax)
{
    g.DrawString(x.ToString("#.#"), fn, br, xX(x), yY(hier_xAchse) + 3);
    g.DrawRectangle(pen, xX(x), yY(hier_xAchse), 1, 3);
    x += dx;
}
x = -dx;
while (x > xmin)
{
    g.DrawString(x.ToString(), fn, br, xX(x), yY(hier_xAchse) + 3);
    g.DrawRectangle(pen, xX(x), yY(hier_xAchse), 1, 3);
    x -= dx;
}
x = dy;
while (x < ymax)
{
    g.DrawString(x.ToString("0.#####"), fn, br, xX(hier_yAchse) + 5, yY(x));
    g.DrawRectangle(pen, xX(hier_yAchse), yY(x), 3, 1);
    x += dy;
}
x = -dy;
while (x > ymin)
{
    g.DrawString(x.ToString("0.#####"), fn, br, xX(hier_yAchse) + 5, yY(x));
    g.DrawRectangle(pen, xX(hier_yAchse), yY(x), 3, 1);
    x -= dy;
}
```

```
}
```

… and we check this with:

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    if (_backBuffer == null)
    {
        _backBuffer = new Bitmap(this.ClientSize.Width, this.ClientSize.Height);
    }
    Graphics g = Graphics.FromImage(_backBuffer);
    g.Clear(Color.Black);
    Font fn = new Font("Verdana", 50);
    Brush br = new SolidBrush(Color.LightGreen);
    Koordinatensystem(g, 10, 1, 0, 0);
    g.Dispose();
    e.Graphics.DrawImageUnscaled(_backBuffer, 0, 0);//Copy back buffer to screen
}
```

We get the output:



Step 2: The Plot Procedure:

```
public void draw_points(Graphics g, Color c)
{
    Brush br = new SolidBrush(c);
    for (int i = 0; i < LU.Count; i++)
        g.FillRectangle(br, xX(LI[i]), yY(LU[i]), 5, 5);
}
```

For testing we need values in the lists LI (current) and LU (voltage). To do this we extend **Form1_Paint** as follows:

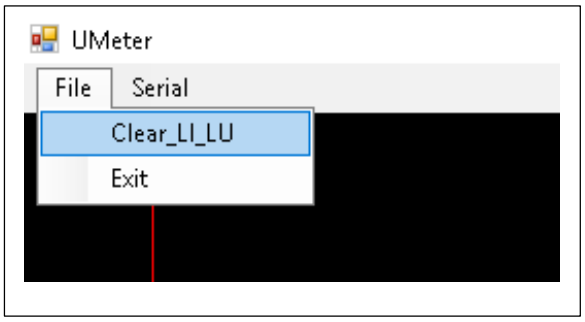
```
Koordinatensystem(g, 10, 1, 0, 0);
if (ex.Length == 32)
{
    string sUB = ex.Substring(0, 16);//ab pos 0 to pos 15
    string sI = ex.Substring(16);//ab pos 16 to end
    UInt16 UB = 0, I = 0;
    for (int i = 0; i < 13; i++)
    {//TI_INA219.pdf: UB = UB >> 3; Uextern = UB * 0.004
        if (sUB[i] == '1') UB += pow_of_2(12 - i);
    }
    for (int i = 0; i < 16; i++)
    {
        if (sI[i] == '1') I += pow_of_2(15 - i);
    }
    if (sI[0] == '1') I = 0;
    LU.Add(UB * 0.004);
    LI.Add(I / 10.0);
    draw_points(g, Color.Blue);
}
g.Dispose();
```

Step 3.: Sometimes, while the curve

is drawn, you want to start a new curve.

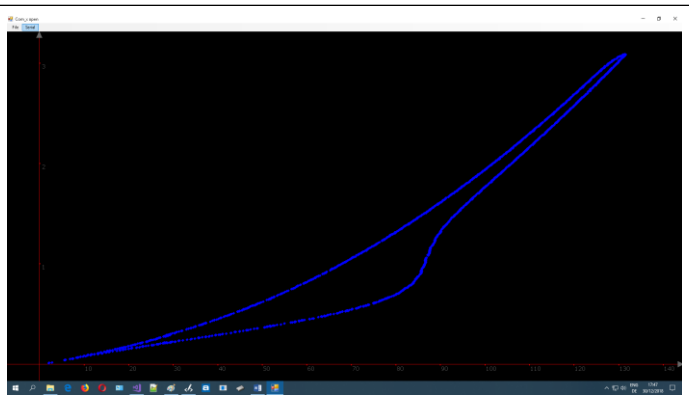
To do this we add the option in the menu under **File**:

Clear_LI_LU to delete all previous list entries.

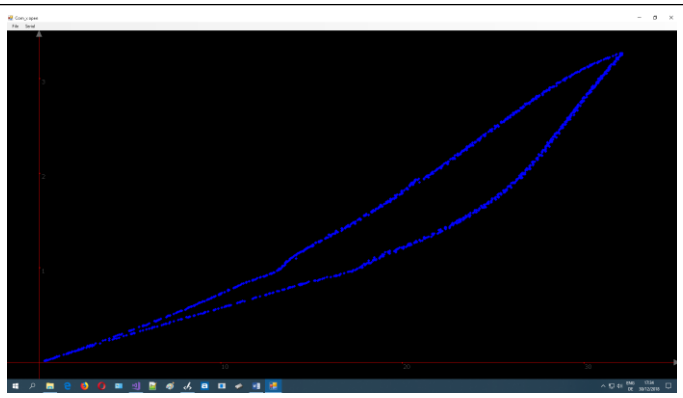



```
private void clearLILUToolStripMenuItem_Click(object sender, EventArgs e)
{
    ex = "";
    LI.Clear(); LU.Clear();
}
```

The keyboard key **Druck S-Abf** can be used to take screenshots. In e.g. **Paint** we use paste and save to store them. Here are two examples:



U = 3,7V Lampe



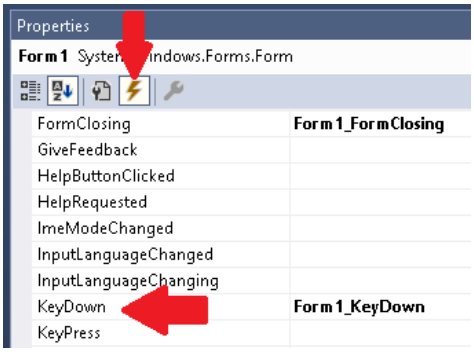
U = 12V Lampe

Problem: With the U = 12V lamp you want more voltage than the currently only about U = 3V. We will come to these higher voltages soon.

Let's playing around, to change the color of the graph during recording it. We do this with:

```
double xStreckfaktor, yStreckfaktor;
List<double> LI, LU;
List<Color> SysCol;
Color colx;
int nCol;


DoubleBuffered = true; //faster drawing
colx = Color.DarkOrange;
nCol = -1;
foreach (Color col in new ColorConverter().GetStandardValues())
{
    SysCol.Add(col);
}
InitializeComponent();
```

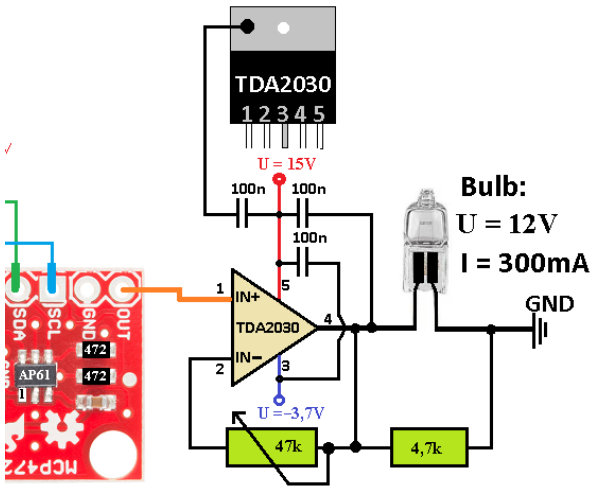
```
LU.Add(UB * 0.004);
LI.Add(I / 10.0);
draw_points(g, colx);
```



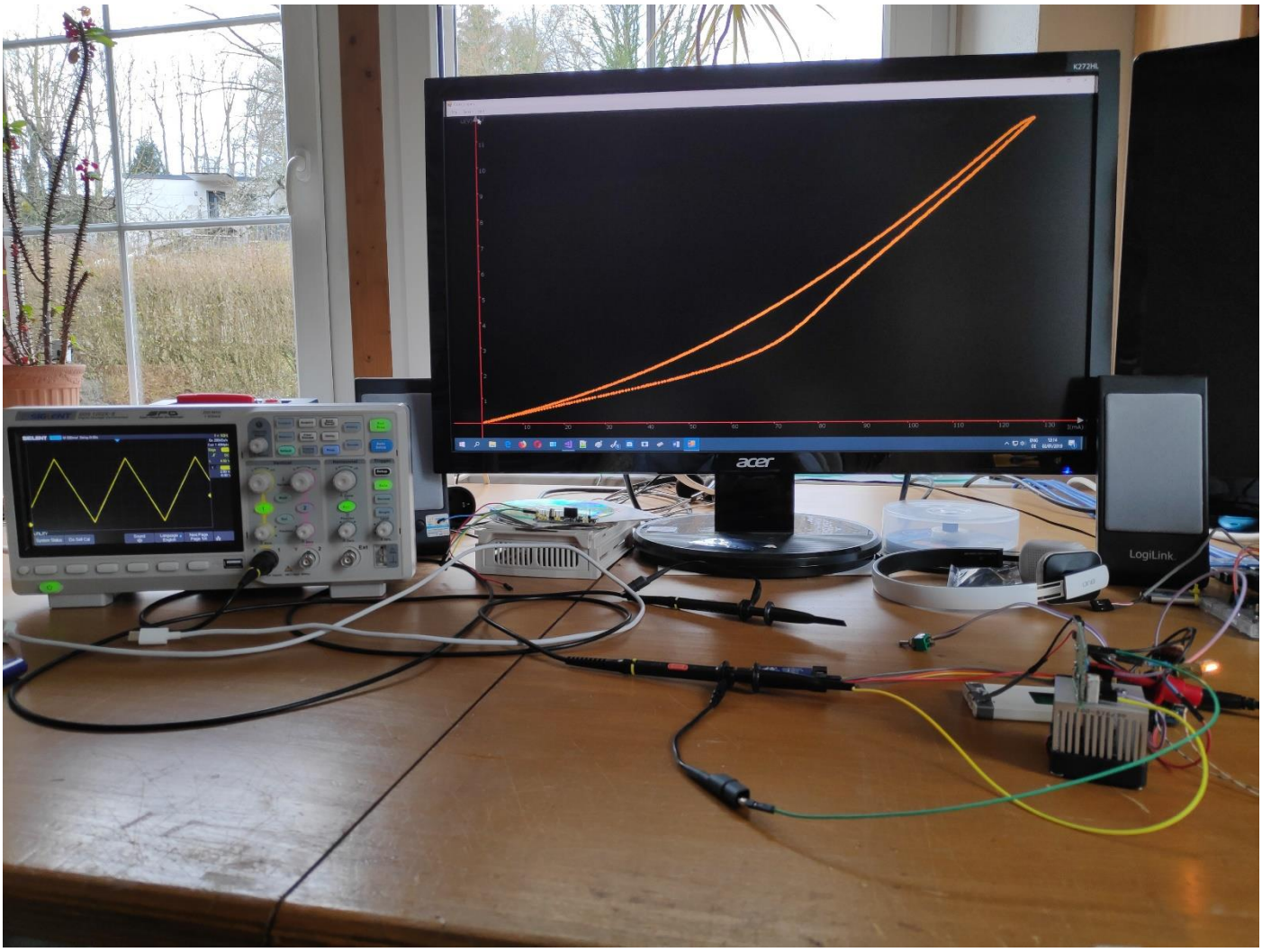

```
private void Form1_KeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.Right)
    {
        nCol++;
        if (nCol < SysCol.Count) colx = SysCol[nCol];
        else { nCol = 0; colx = SysCol[nCol]; }
        Refresh();
    }
}
```

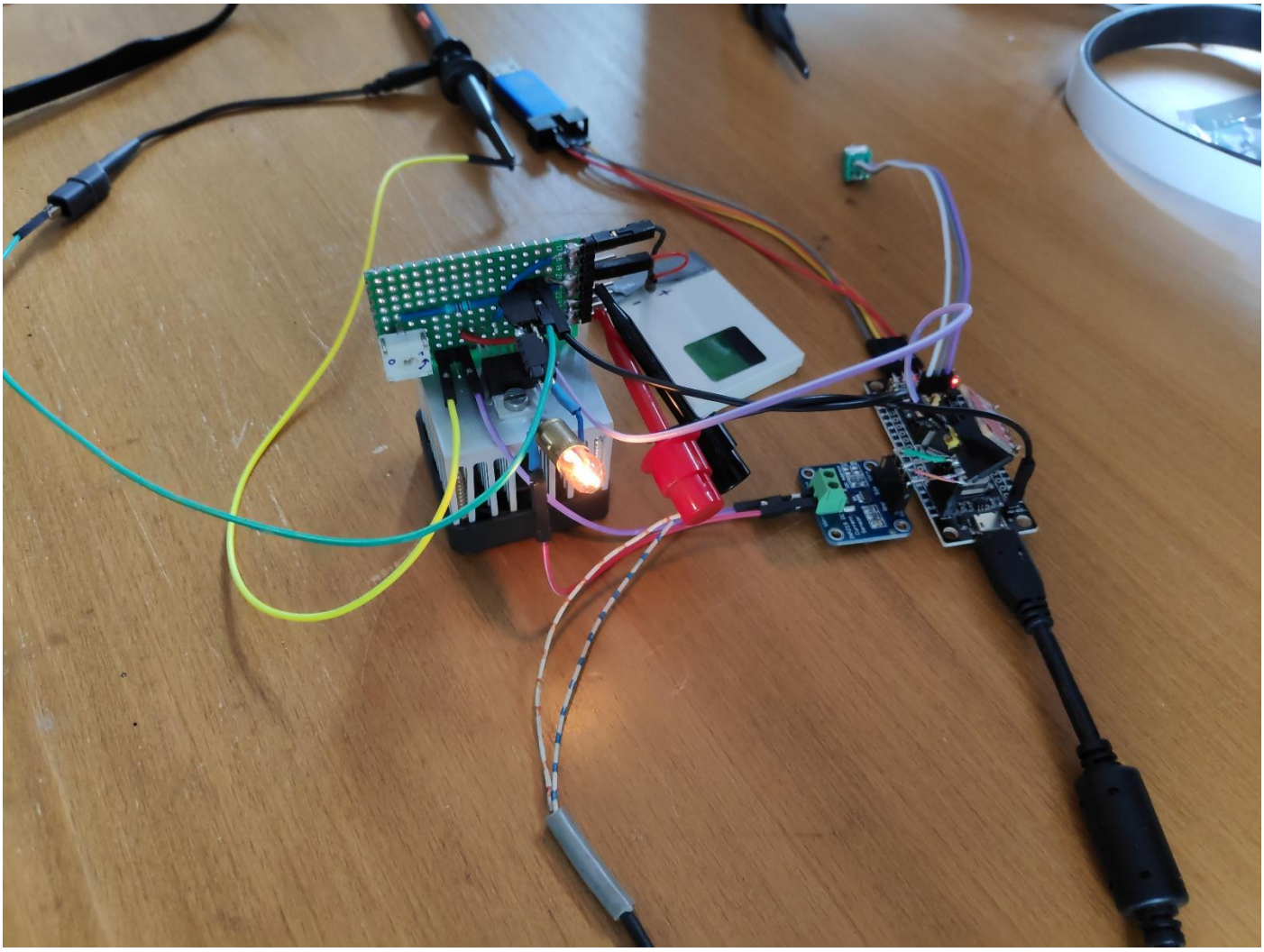
The problem of getting higher voltages can be solved by doing the following: At the moment we are running the power amplifier like an impedance converter: pin4 (OUT) and pin2 (IN−) are connected directly. The gain is then G = 1.

The amplification factor $G = 1 + \frac{R_2}{R_1}$, where we choose R_2 variable (poti), we get with adjacent modification. With the C = 100nF capacitors we avoid resonances.



The entire experiment with a bulb looks like this:





Two C = 100nF capacitors are soldered on the back side of the circuit board of TDA2030

Finally, the last used **main.c** file for programming the Black Pill:

```
#include "main.h"
#include "usb_device.h"
#include "usbd_cdc_if.h"//By reason of: CDC_Transmit_FS(uint8_t*, uint16_t);

#define getBit(bit, value) ((value >> bit) & 0x00000001)

volatile uint32_t rawValue = 0;
volatile float U = 0.0;
ADC_HandleTypeDef hadc1;
I2C_HandleTypeDef hi2c1;
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
static void MX_I2C1_Init(void);

void MCP4725_Mem_WriteReg(uint16_t);
void ina219_Mem_WriteReg(uint8_t, uint16_t);
void ina219_Mem_ReadReg(uint8_t, uint16_t*);
void ina219_setCalibration_16V_400mA(void);
float ina219_get_I_mA(void);
float ina219_get_UBus_V(void);
uint32_t ina219_get_Uraw_Iraw(void);
```

```
uint16_t slen(const char*);
void add_txt(char* , char* );
char *my_ftoa(float, char*);
void USB_Info(char*);
void USB_Info_tft(char*, float, char*);
void USB_Info_Bits(unsigned);

int main(void)
{
HAL_Init();
SystemClock_Config();
MX_GPIO_Init();
MX_ADC1_Init();
MX_I2C1_Init();
MX_USB_DEVICE_Init();
ina219_setCalibration_16V_400mA();
HAL_Delay(500);
while (1){
    for(int n = 0; n <= 4095; n += 9){
      MCP4725_Mem_WriteReg(n);
      USB_Info_Bits(ina219_get_Uraw_Iraw());
      //HAL_Delay(50);
    }
    for(int n = 4086; n >= 0; n -= 9){
      MCP4725_Mem_WriteReg(n);
      USB_Info_Bits(ina219_get_Uraw_Iraw());
      //HAL_Delay(50);
    }
}
}

void MCP4725_Mem_WriteReg(uint16_t val)
{//0x62 is MCP4725-modul
   uint8_t data[] = { val >> 4, val << 4 };
   HAL_I2C_Mem_Write(&hi2c1, 0x62 << 1, 0x40, 1, data, 2, HAL_MAX_DELAY);
   HAL_Delay(1);
}

void ina219_Mem_WriteReg(uint8_t reg, uint16_t val)
{//0x40 is INA219-modul
    uint8_t data[] = { val >> 8, val };
    HAL_I2C_Mem_Write(&hi2c1, 0x40 << 1, reg, 1, data, 2, HAL_MAX_DELAY);
    HAL_Delay(1);
}

void ina219_Mem_ReadReg(uint8_t reg, uint16_t *val)
{//0x40 is INA219-modul; for BusVoltage reg=2; for Current reg=4
    uint8_t data[] = {0, 0};
    HAL_I2C_Mem_Read(&hi2c1, 0x40 << 1, reg, 1, data, 2, HAL_MAX_DELAY);
    *val = (data[0] << 8 ) | data[1];
}

void ina219_setCalibration_16V_400mA(void)
{//Divider_I_mA = 10 = 1000 / I_LSB; I_LSB = 100uA ~ 400mA / 4096 pro Bit
    ina219_Mem_WriteReg(0x05, 4096);//calibration_Reg = 0x05, 12-Bit resolution
```

```
    ina219_Mem_WriteReg(0x00, 0b0000000110011111);//config_Reg = 0x00
}

float ina219_get_I_mA(void)
{
    uint16_t tmp;
    ina219_Mem_ReadReg(0x04 , &tmp);//I_Reg = 0x04
    if(getBit(15, tmp)){//0b1000000000000000 is minus sign; TI_INA219.pdf
        tmp = ~tmp;//take complement
        tmp += 1;//and add one, to calc I=-0.123mA
        return tmp / -10.0; //Divider_I_mA = 10;
    } else return tmp / 10.0; //Divider_I_mA = 10;
}

float ina219_get_UBus_V(void) {//U=12.34V
  uint16_t tmp;
  ina219_Mem_ReadReg(0x02 , &tmp);//U_Reg = 0x02
  tmp = tmp >> 3;//TI_INA219_pdf:Bus Voltage Register contents must be shifted right by three bits.
  return tmp * 0.004;
}

uint32_t ina219_get_Uraw_Iraw(void) {
  uint16_t x=0, y=0;
  while(HAL_I2C_IsDeviceReady(&hi2c1, 0x40 << 1, 0x02, HAL_MAX_DELAY) != HAL_OK);
  ina219_Mem_ReadReg(0x02 , &x);//U_Reg = 0x02
  while(HAL_I2C_IsDeviceReady(&hi2c1, 0x40 << 1, 0x04, HAL_MAX_DELAY) != HAL_OK);
  ina219_Mem_ReadReg(0x04 , &y);//I_Reg = 0x04
  return (x << 16) | y;
}

uint16_t slen(const char* s) {
    uint16_t i;
    for (i = 0; s[i] != 0; i++);
    return i;//s[0] not 0 then i=1;
}

void add_txt(char* out, char* in) {
   while (*out != 0) out++;
   while (*in != 0) {
      *out++ = *in++;
   }
   *out = 0;
}

char *my_ftoa(float val, char *str)
{
  char *cp; cp=str;
  int v, v0, rest, rest0;
  char c;
  if(val < 0){     //   cp=0
      val = -val;  //cp[0   ][1   ][2][3][4][5][6][7][8][9]
      *cp++ = '-'; //  [0: -] cp=1
  }
  v0 = (int)val; v=v0;
  //rest0=(int)((val-(int)val)*100000000);//genauer!
```

```
  rest0=(int)((val-(int)val)*10000);
  rest = rest0;
  do {
    v /= 10;
    cp++;
  } while(v != 0);
  do {
    rest /= 10;
    cp++;
  } while(rest != 0);
  cp++; //wegen ','
  *cp-- = 0;
  do {
    c = rest0 % 10;
    rest0 /= 10;
    c += '0';
    *cp-- = c;
  } while(rest0 != 0);
  *cp-- = ',';
  do {
    c = v0 % 10;
    v0 /= 10;
    c += '0';
    *cp-- = c;
  } while(v0 != 0);
  return  cp;
}

void USB_Info(char *str)
{
  char txt[64] = {};
  add_txt( txt, str);
  add_txt( txt, "\n\r");
  CDC_Transmit_FS((uint8_t *)txt, slen(txt));
}

void USB_Info_tft(char *str, float n, char *str1)
{
  char txt[64] = {}, h[32] = {};
  add_txt( txt, str);
  my_ftoa(n, h);
  add_txt( txt, h);
  add_txt( txt, str1);
  add_txt( txt, "\n\r");
  CDC_Transmit_FS((uint8_t *)txt, slen(txt));
}

void USB_Info_Bits(unsigned n)
{
  char txt[64] = {};
  for(int i = 31; i >= 0; i--){
    if(getBit(i, n)) add_txt( txt, "1"); else add_txt( txt, "0");
  }
  add_txt( txt, "\n\r");
  CDC_Transmit_FS((uint8_t *)txt, slen(txt));
```

```
}

void SystemClock_Config(void)
{
RCC_OscInitTypeDef RCC_OscInitStruct = {0};
RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};
RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
RCC_OscInitStruct.HSEState = RCC_HSE_ON;
RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
RCC_OscInitStruct.HSIState = RCC_HSI_ON;
RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK) Error_Handler();
RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK) Error_Handler();
PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC|RCC_PERIPHCLK_USB;
PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV6;
PeriphClkInit.UsbClockSelection = RCC_USBCLKSOURCE_PLL_DIV1_5;
if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK) Error_Handler();
}
static void MX_ADC1_Init(void)
{
ADC_ChannelConfTypeDef sConfig = {0};
hadc1.Instance = ADC1;
hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
hadc1.Init.ContinuousConvMode = DISABLE;
hadc1.Init.DiscontinuousConvMode = DISABLE;
hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
hadc1.Init.NbrOfConversion = 1;
if (HAL_ADC_Init(&hadc1) != HAL_OK) Error_Handler();
sConfig.Channel = ADC_CHANNEL_7;
sConfig.Rank = ADC_REGULAR_RANK_1;
sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK) Error_Handler();
}
static void MX_I2C1_Init(void)
{
hi2c1.Instance = I2C1;
hi2c1.Init.ClockSpeed = 100000;
hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
hi2c1.Init.OwnAddress1 = 0;
hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
hi2c1.Init.OwnAddress2 = 0;
hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
if (HAL_I2C_Init(&hi2c1) != HAL_OK) Error_Handler();
```

```
}
static void MX_GPIO_Init(void)
{
__HAL_RCC_GPIOD_CLK_ENABLE();
__HAL_RCC_GPIOA_CLK_ENABLE();
__HAL_RCC_GPIOB_CLK_ENABLE();
}
void Error_Handler(void){}
#ifdef  USE_FULL_ASSERT
void assert_failed(uint8_t *file, uint32_t line){}
#endif
```

and the **Form1.cs** file of the C# plotting program. Use the arrow key **Key.Right** to change the color of the drawn graph.

```
using System.Drawing;
using System.Windows.Forms;
using System;
using System.Collections.Generic;

namespace U_Meter
{
   public partial class Form1 : Form
   {
      Bitmap _backBuffer;
      bool busy = false;
      String ex = "COM!";
      double xmin, xmax, ymin, ymax, yA = 0, xA = 0;
      double xStreckfaktor, yStreckfaktor;
      List<double> LI, LU;
      List<Color> SysCol;
      Color colx;
      int nCol;

   public Form1()
      {
         xmax = 150;
         xmin = -5;
         ymax = 4;
         ymin = -0.5;
         LI = new List<double>();
         LU = new List<double>();
         SysCol = new List<Color>();
         DoubleBuffered = true; //faster drawing
         colx = Color.DarkOrange;
         nCol = -1;
         foreach (Color col in new ColorConverter().GetStandardValues())
         {
            SysCol.Add(col);
         }
         InitializeComponent();
      }

      public int xX(double x)
      {
```

```
        return (int)Math.Round(xStreckfaktor * (x - xmin));
    }
    public int yY(double y)
    {
        return (int)Math.Round(yStreckfaktor * (ymax - y));
    }

    public void Koordinatensystem(Graphics g, double dx, double dy, double hier_xAchse, double
hier_yAchse)
    {
        Font fn = new Font("Verdana", 12);
        Brush br = new SolidBrush(Color.DimGray);
        Pen pen = new Pen(Color.Red, 1);

        if (LU.Count > 20)
        {
            xmax = 0;
            for (int i = 0; i < LI.Count; i++)
                if (xmax < LI[i]) xmax = LI[i];
            xmax = 1.1 * xmax;
            xmin = -0.05 * xmax;
            ymax = 0;
            for (int i = 0; i < LU.Count; i++)
                if (ymax < LU[i]) ymax = LU[i];
            ymax = 1.1 * ymax;
            ymin = -0.05 * ymax;
        }
        xStreckfaktor = this.ClientRectangle.Width / (xmax - xmin);
        yStreckfaktor = this.ClientRectangle.Height / (ymax - ymin);
        Point[] pxA =
        {
            new Point(xX(xmax),yY(hier_xAchse)),
            new Point(xX(xmax) - 18,yY(hier_xAchse) - 9),
            new Point(xX(xmax) - 18,yY(hier_xAchse) + 9),
            new Point(xX(xmax),yY(hier_xAchse))
        };
        int h = menuStrip1.Height;
        Point[] pyA =
        {
            new Point(xX(hier_yAchse) + 9, yY(ymax) + 18 + h),
            new Point(xX(hier_yAchse), yY(ymax)-2 + h),
            new Point(xX(hier_yAchse) - 9, yY(ymax) + 18 + h),
            new Point(xX(hier_yAchse) + 9, yY(ymax) + 18 + h)
        };

        g.DrawLine(pen, xX(xmin), yY(hier_xAchse), xX(xmax), yY(hier_xAchse));//x-Achse
        g.FillPolygon(br, pxA);//Pfeil der x-Achse
        g.DrawString("I(mA)", fn, br, xX(xmax)-50, yY(hier_xAchse) + 10);
        g.DrawLine(pen, xX(hier_yAchse), yY(ymin), xX(hier_yAchse), yY(ymax));//y-Achse
        g.FillPolygon(br, pyA);//Pfeil der y-Achse
        g.DrawString("U(V)", fn, br, xX(hier_yAchse) - 50, yY(ymax) + 30);
        double x = dx;
        while (x < xmax)
        {
            g.DrawString(x.ToString("#.#"), fn, br, xX(x), yY(hier_xAchse) + 3);
```

```
            g.DrawRectangle(pen, xX(x), yY(hier_xAchse), 1, 3);
            x += dx;
        }
        x = -dx;
        while (x > xmin)
        {
            g.DrawString(x.ToString(), fn, br, xX(x), yY(hier_xAchse) + 3);
            g.DrawRectangle(pen, xX(x), yY(hier_xAchse), 1, 3);
            x -= dx;
        }
        x = dy;
        while (x < ymax)
        {
            g.DrawString(x.ToString("0.#####"), fn, br, xX(hier_yAchse) + 5, yY(x));
            g.DrawRectangle(pen, xX(hier_yAchse), yY(x), 3, 1);
            x += dy;
        }
        x = -dy;
        while (x > ymin)
        {
            g.DrawString(x.ToString("0.#####"), fn, br, xX(hier_yAchse) + 5, yY(x));
            g.DrawRectangle(pen, xX(hier_yAchse), yY(x), 3, 1);
            x -= dy;
        }
    }

    public void draw_points(Graphics g, Color c)
    {
        Brush br = new SolidBrush(c);
        for (int i = 0; i < LU.Count; i++)
            g.FillRectangle(br, xX(LI[i]), yY(LU[i]), 5, 5);
    }



    private void opencom(object sender, EventArgs e)
    {
        if (serialPort1 != null)
        {
            try
            {
                serialPort1.Open();
                serialPort1.DiscardInBuffer();
                serialPort1.DiscardOutBuffer();
                Text = "Com_x open";
                cOMToolStripMenuItem.Text = "closeCOM";
            }
            catch (Exception ex)
            {
                MessageBox.Show("Sonstiger Fehler: " + ex.Message);
            }
        }
    }

    private void closecom(object sender, EventArgs e)
```

```
{
   if (serialPort1.IsOpen && !busy)
   {
      try
      {
         serialPort1.Close();
         this.Text = "Com_x closed";
         cOMToolStripMenuItem.Text = "openCOM";
      }
      catch (Exception ex)
      {
         MessageBox.Show("Sonstiger Fehler: " + ex.Message);
      }
   }
}

private void clearLILUToolStripMenuItem_Click(object sender, EventArgs e)
{
   ex = "";
   LI.Clear(); LU.Clear();
}

private void Form1_KeyDown(object sender, KeyEventArgs e)
{
   if (e.KeyCode == Keys.Right)
   {
      nCol++;
      if (nCol < SysCol.Count) colx = SysCol[nCol];
      else { nCol = 0; colx = SysCol[nCol]; }
      Refresh();
   }
}

UInt16 pow_of_2(int n)
{
   return (UInt16)(1 << n);
}

private void Form1_Paint(object sender, PaintEventArgs e)
{
   if (_backBuffer == null)
   {
      _backBuffer = new Bitmap(this.ClientSize.Width, this.ClientSize.Height);
   }
   Graphics g = Graphics.FromImage(_backBuffer);
   g.Clear(Color.Black);
   Font fn = new Font("Verdana", 30);
   Brush br = new SolidBrush(colx);
   Koordinatensystem(g, 10, 1, 0, 0);
   if (ex.Length == 32)
   {
      string sUB = ex.Substring(0, 16);//ab pos 0 to pos 15
      string sI = ex.Substring(16);//ab pos 16 to end
      UInt16 UB = 0, I = 0;
      for (int i = 0; i < 13; i++)
```

```
            {//TI_INA219.pdf: UB = UB >> 3; Uextern = UB * 0.004
                if (sUB[i] == '1') UB += pow_of_2(12 - i);
            }
            for (int i = 0; i < 16; i++)
            {
                if (sI[i] == '1') I += pow_of_2(15 - i);
            }
            if (sI[0] == '1') I = 0;
            LU.Add(UB * 0.004);
            LI.Add(I / 10.0);
            draw_points(g, colx);
        }
        //g.DrawString(SysCol.Count.ToString() + colx.ToString() + nCol.ToString(), fn, br, 50, 30);
        g.Dispose();
        e.Graphics.DrawImageUnscaled(_backBuffer, 0, 0);//Copy back buffer to screen
    }

    private void exitToolStripMenuItem_Click(object sender, System.EventArgs e)
    {
        Close();
    }

    private void cOMToolStripMenuItem_Click(object sender, System.EventArgs e)
    {
        string txt = cOMToolStripMenuItem.Text;
        if (txt == "openCOM")
        {
            BeginInvoke(new EventHandler(opencom));
        }
        else
        {
            BeginInvoke(new EventHandler(closecom));
        }
    }

    private void serialPort1_DataReceived(object sender, System.IO.Ports.SerialDataReceivedEventArgs
e)
    {
        busy = true;
        try
        {
            ex = serialPort1.ReadLine();
        }
        catch (TimeoutException) { }
        try
        {
            serialPort1.ReadExisting();
        }
        catch (TimeoutException) { }
        busy = false;
        Invalidate();
    }

    private void Form1_FormClosing(object sender, FormClosingEventArgs e)
    {
```

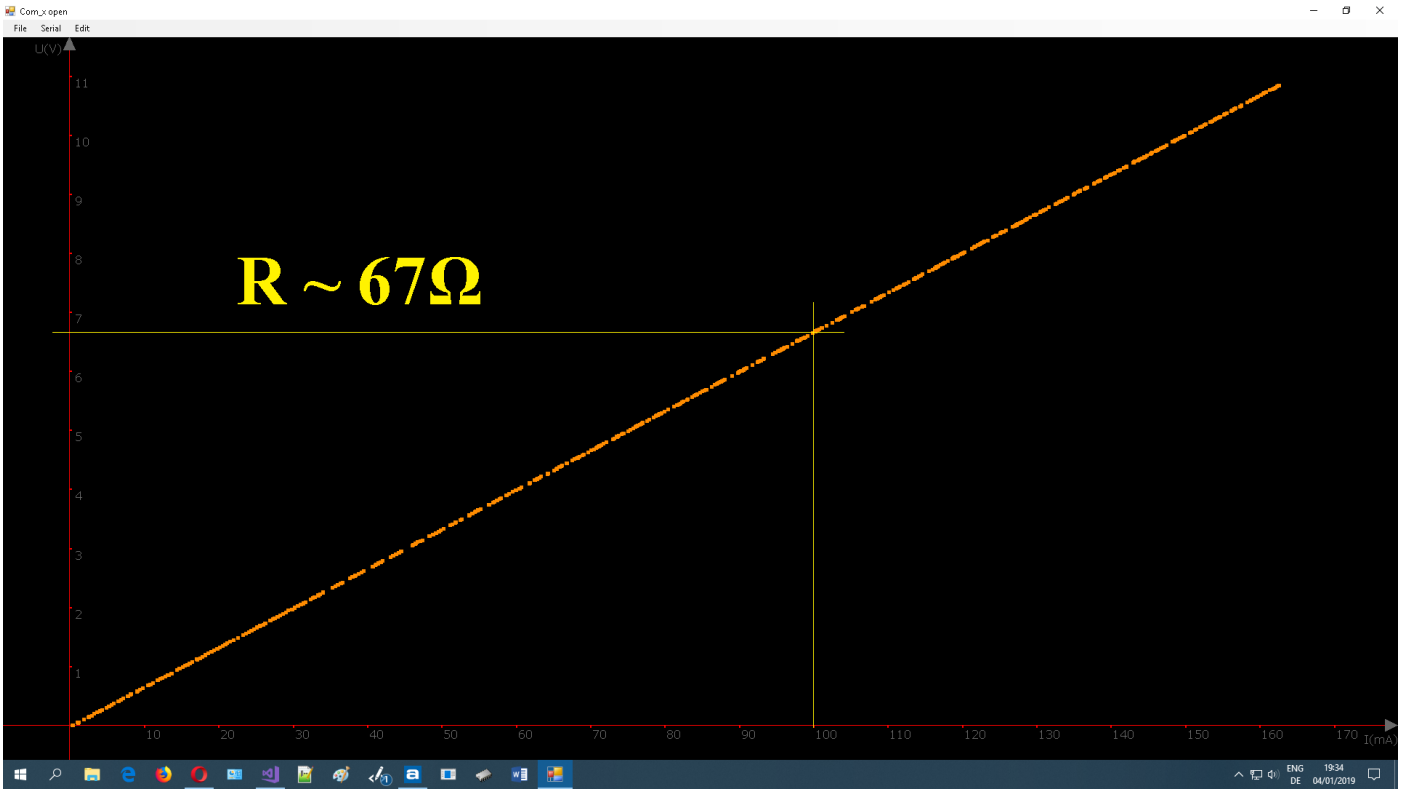
```
            this.BeginInvoke(new EventHandler(closecom));
        }

        private void Form1_Resize(object sender, EventArgs e)
        {
            _backBuffer.Dispose();
            _backBuffer = null;
            Refresh();
        }
    }
}
```

Exercise 1: Change I2C speed from 100000 to 400000.
Measure the frequency of the triangle ramp with an oscilloscope for both speeds.

Exercise 2: Uncomment **//HAL_Delay(50);** in Black Pill's program in the **while**-loop in **main{…}**.
Record and plot U = U(I).
Compare the new graph with an old one, which is not delayed.
Explain the difference of both graphs.

Exercise 3: Record and plot a graph of a e.g. R = 67Ω resistor.
Determine the slope of the graph.

Exercise 4: Ohm's law remain valid or lose validity?

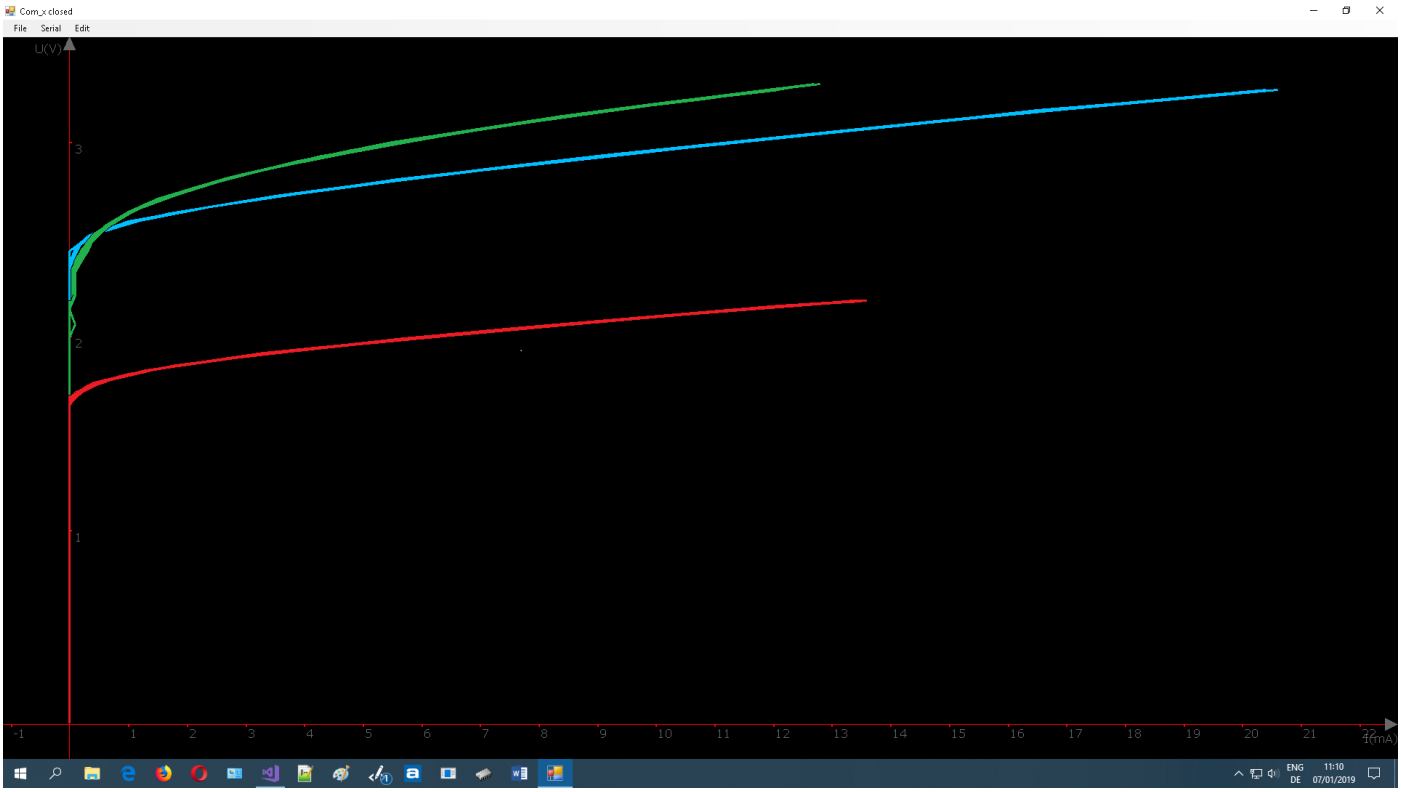


Exercise 5: Extent the C# program to use the arrow key **Key.Left** to go backwards in the list of colors **SysCol[nCol]**.

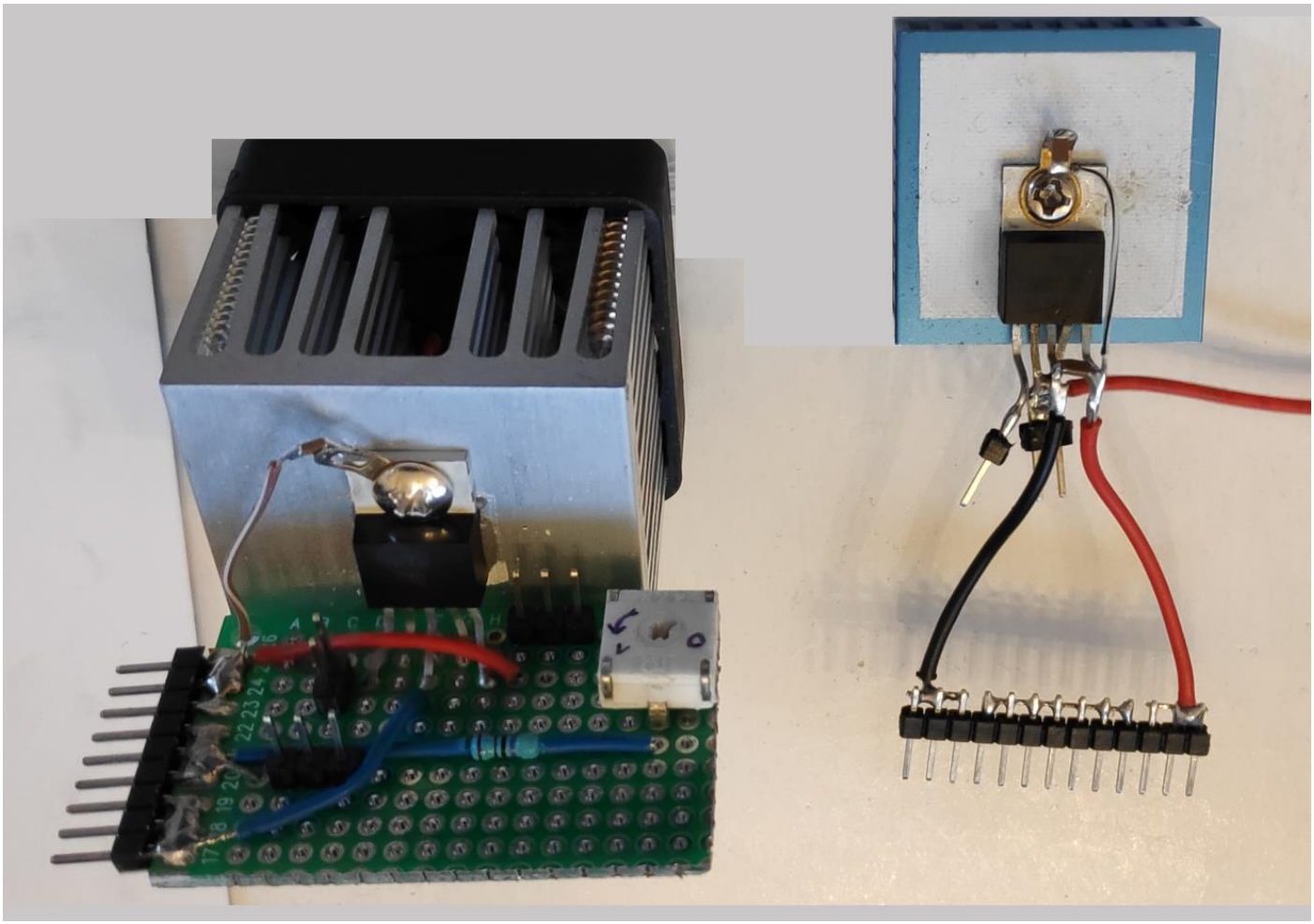
Exercise 6: Change the C# drawing procedure **draw_points**, to draw lines instead of points.

```
public void draw_line(Graphics g, Color c)
{
    //Brush br = new SolidBrush(c);
    Pen pen = new Pen(c, 3);
    for (int i = 0; i < LU.Count; i++)
        //g.FillRectangle(br, xX(LI[i]), yY(LU[i]), 5, 5);
        if(i < (LU.Count - 1) ) g.DrawLine(pen, xX(LI[i]), yY(LU[i]), xX(LI[i+1]), yY(LU[i+1]));
}
```

Exercise 7: Take a three color LED and measure the forward voltage of the three colors. Close the COMport while changing the wiring, to measure the forward voltage of another color. Use a TDA2030 with G = 1, and a series resistor to limit the forward current of the LED.



We used a R = 10Ω resistor to limit the forward current of the LED. With a little help of MS-Paint you get this result.



OpAmp TDA2030 with $G = 1 + \frac{R_2}{R_1}$, and OpAmp TDA2030 with G = 1.

… have fun with STM32 and C#!
edgarmarx@t-online.de