Digital Instrumentation

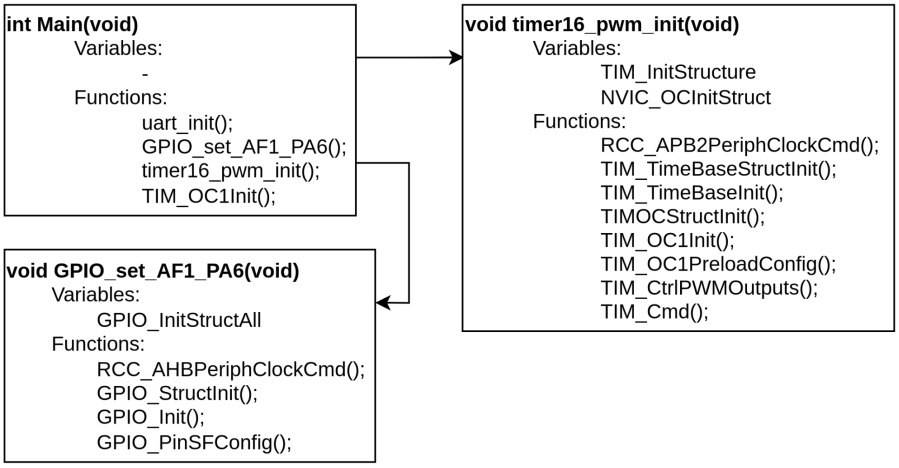
Journal for Exercise 3

Participants: **Tibor Illés**
Benedikt Klingebiel
Christian Jehle

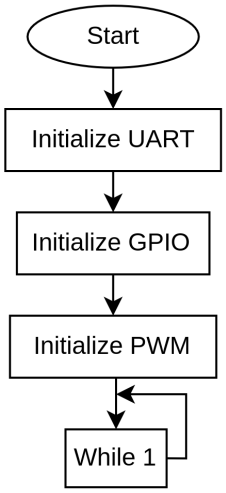
Sep 20, 2022

1. Generating PWM signal

Block Diagram:



Flowchart:



Function description:

main

Syntax `int main(void);`

Parameters -

The function is the main function. It initializes the GPIO port, the TIMER for PWM and the UART port. It contains the main program loop that is continuously executed and does nothing in this case.

timer16_pwm_init

Syntax `void timer16_pwm_init(void);`

Parameters -

The function initializes the timer with a PWM output with the required parameters.

GPIO_set_AF1_PA6

Syntax `void GPIO_set_AF1_PA6(void);`

Parameters -

The function designates the PA6 GPIO port for the timer as output with the required settings.

Source code:

```
#include "stm32f30x_conf.h" // STM32 config
#include "30010_io.h"           // Input/output library for this course
#include "flash.h"
#include "lcd.h"

void timer16_pwm_init(void){
	//step1
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM16, ENABLE);
	//step3
	TIM_TimeBaseInitTypeDef TIM_InitStructure;
	TIM_TimeBaseStructInit(&TIM_InitStructure);
	TIM_InitStructure.TIM_ClockDivision = 0;
	TIM_InitStructure.TIM_Period = 1000; //set the maximum period
	TIM_InitStructure.TIM_Prescaler = 64; //for 1MHz counting frequency
	TIM_TimeBaseInit(TIM16,&TIM_InitStructure);
	//step4
	TIM_OCInitTypeDef TIM_OCInitStruct;
	TIM_OCStructInit(&TIM_OCInitStruct);
	TIM_OCInitStruct.TIM_OCMode = TIM_OCMode_PWM1; //pwm mode
	TIM_OCInitStruct.TIM_OutputState = TIM_OutputState_Enable;
	TIM_OCInitStruct.TIM_Pulse = 250; //25% pwm
	TIM_OCInitStruct.TIM_OCPolarity = TIM_OCPolarity_High; // polarity
	//step5
	TIM_OC1Init(TIM16,&TIM_OCInitStruct);
	//step6
	TIM_OC1PreloadConfig(TIM16,TIM_OCPreload_Enable);
	//step7
	TIM_CtrlPWMOutputs(TIM16, ENABLE); //pwm output enable
	TIM_Cmd(TIM16,ENABLE); // timer enable
}

void GPIO_set_AF1_PA6(void){
	//step2
	// Enable clock for GPIO Port B
	RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA,ENABLE);
	GPIO_InitTypeDef GPIO_InitStructAll; // Define typedef struct for setting
pins
	GPIO_StructInit(&GPIO_InitStructAll);
	// Then set things that are not default.
	GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_AF;
	GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_6;
	GPIO_InitStructAll.GPIO_PuPd = GPIO_PuPd_NOPULL;
	GPIO_InitStructAll.GPIO_Speed = GPIO_Speed_50MHz;
	// Setup of GPIO with the settings chosen
	GPIO_Init(GPIOA, &GPIO_InitStructAll);
	//set gpio af
```

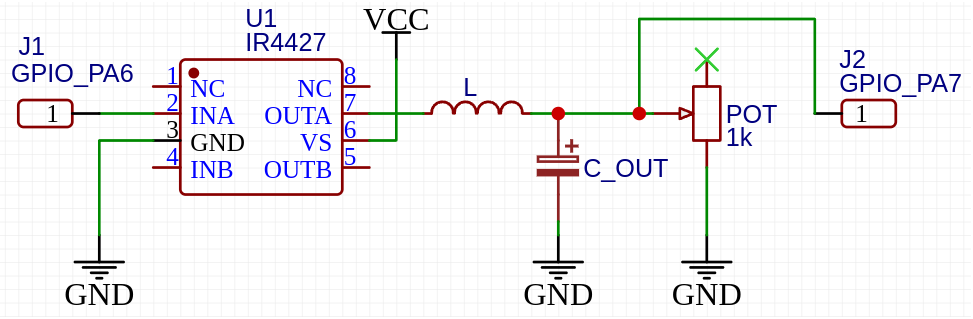
```
	GPIO_PinAFConfig(GPIOA, GPIO_PinSource6, GPIO_AF_1);
}

int main(void)
{
	uart_init(9600);
	GPIO_set_AF1_PA6();
	timer16_pwm_init();
	while(1){

	}
}
```

Results:

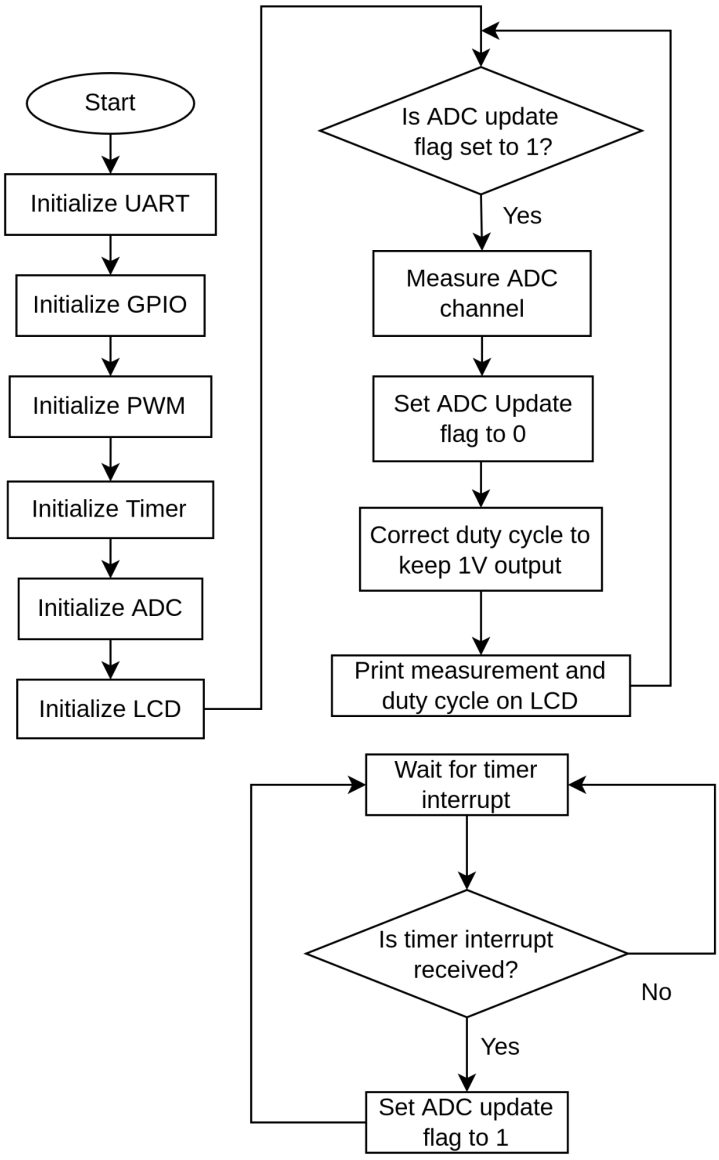
We measured the GPIO output and PWM signal was correct with the set 25% duty cycle.

2. Implementing the circuit

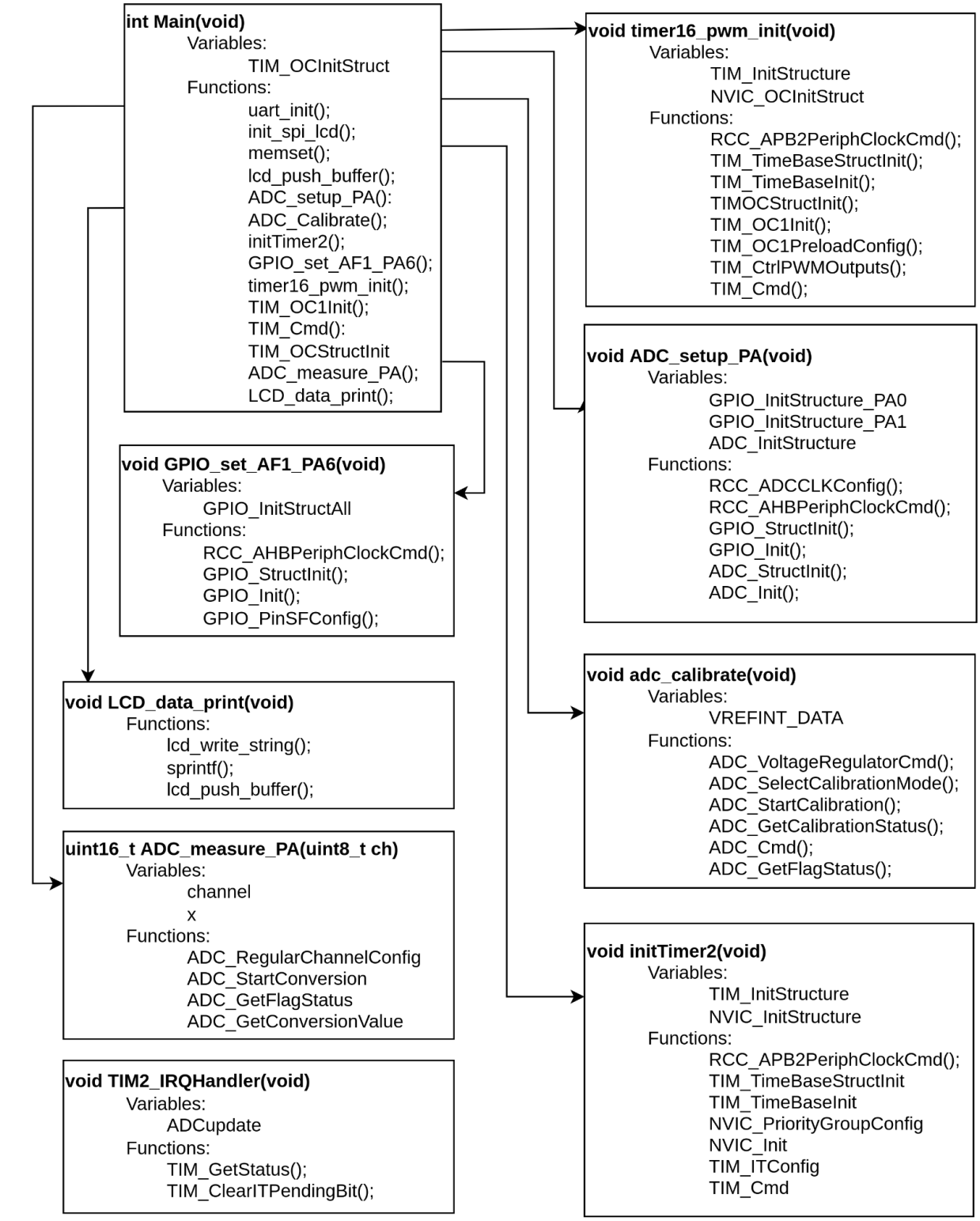
Circuit diagram:



Flowchart:



Block Diagram:



Function description:

main

Syntax `int main(void);`

Parameters -

The function is the main function. It initializes the GPIO port, the TIMER for PWM, the UART port and the ADC. It contains the main program loop that is continuously executed, each time checking the ADCupdate flag. If the flag is valid, it measure the output of the circuit, prints the data on the LCD and corrects the PWM output data cycle to reach a steady 1V circuit output voltage.

timer16_pwm_init

Syntax `void timer16_pwm_init(void);`

Parameters -

The function initializes the timer with a PWM output with the required parameters.

GPIO_set_AF1_PA6

Syntax `void GPIO_set_AF1_PA6(void);`

Parameters -

The function designates the PA6 GPIO port for the timer as output with the required settings.

initTimer2

Syntax `void initTimer(void);`

Parameters -

The function initializes the required timer for setting the ADCupdate flag every 10ms.

TIM2_IRQHandler

Syntax `void TIM2_IRQHandler(void);`

Parameters -

The function is activated when a timer interrupt occurs. Sets the ADCupdate flag every 10 milliseconds.

ADC_setup_PA

Syntax `void ADC_setup_PA(void);`

Parameters -

This function configures the ADC. The parameters have been set according to the assignment description.

ADC_Calibrate

Syntax `void ADC_Calibrate(void);`

Parameters -

This function calibrates the ADC according to the assignment description.

ADC_measure_PA

Syntax `uint16_t ADC_measure_PA(uint8_t ch);`

Parameters ch: ADC channel to be measured

This function reads either ADC channel 1 or 15 (this is the output of the circuit) depending on the input. For this purpose the channel is collected for 1.5 clock cycles and then the measurement is started. As soon as this is completed, the measured value is output in the form of a 16-bit integer.

LCD_data_print

Syntax `void LCD_data_print(void);`

Parameters -

This function prints the ADC Absolute voltage and the PWM duty cycle to the lcd.

Source Code:

```
#include "stm32f30x_conf.h" // STM32 config
#include "30010_io.h"           // Input/output library for this course
#include "flash.h"
#include "lcd.h"
#include <string.h>

#define VREFINT_CAL *((uint16_t*) ((uint32_t) 0x1FFFF7BA))
float volatile V_ABS;
char  str[16];
uint8_t fbuffer[512];
uint16_t volatile adc1;
uint16_t volatile adc2;
uint8_t volatile ADCupdate=0;
uint8_t volatile maxcount=100;
uint8_t volatile count=50;


void initTimer2(void){
      RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2,ENABLE);
      NVIC_InitTypeDef NVIC_InitStructure;
      // NVIC for timer
      NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
      NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
      NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
      NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
      NVIC_Init(&NVIC_InitStructure);
      TIM_ITConfig(TIM2,TIM_IT_Update,ENABLE);
      TIM_Cmd(TIM2,ENABLE);
      //Settings timer
      RCC->APB1ENR |= RCC_APB1Periph_TIM2; // Enable clock line to timer 2
      TIM2->CR1=0xB01;
      TIM2->PSC=6399; //change pre-scaler frequency to 10kHz
```

```
      TIM2->ARR=999; //count up to 100
      TIM2->DIER |= 0x0001; // Enable timer 2 interrupts
    //NVIC settings
      NVIC_SetPriority(TIM2_IRQn, 1); // Set interrupt priority interrupts
      NVIC_EnableIRQ(TIM2_IRQn); // Enable interrupt
      TIM_Cmd(TIM2,DISABLE);
}

void ADC_setup_pA(void){
      RCC_ADCCLKConfig(RCC_ADC12PLLCLK_Div8); //adc clk
      RCC_AHBPeriphClockCmd(RCC_AHBPeriph_ADC12, ENABLE); //enable adc
      RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA, ENABLE); //gpio clock

      GPIO_InitTypeDef GPIO_InitStructAll; // Define typedef struct for setting
pins

      GPIO_StructInit(&GPIO_InitStructAll); // Initialize GPIO struct
      GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_AN; // Set as input
      GPIO_InitStructAll.GPIO_PuPd = GPIO_PuPd_DOWN; // Set as pull down
      GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_7; // Set so the configuration is on
pin 4
      GPIO_Init(GPIOA, &GPIO_InitStructAll); // Setup of GPIO with the settings
chosen

      GPIO_StructInit(&GPIO_InitStructAll); // Initialize GPIO struct
      GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_AN; // Set as input
      GPIO_InitStructAll.GPIO_PuPd = GPIO_PuPd_DOWN; // Set as pull down
      GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_1; // Set so the configuration is on
pin 4
      GPIO_Init(GPIOA, &GPIO_InitStructAll); // Setup of GPIO with the settings
chosen

      ADC_InitTypeDef ADC_InitStructAll; //struct for adc config

      ADC_StructInit(&ADC_InitStructAll); //settings for the adc
      ADC_InitStructAll.ADC_ContinuousConvMode = DISABLE;
      ADC_InitStructAll.ADC_Resolution = ADC_Resolution_12b;
      ADC_InitStructAll.ADC_ExternalTrigEventEdge =
ADC_ExternalTrigEventEdge_None;
      ADC_InitStructAll.ADC_DataAlign = ADC_DataAlign_Right;
      ADC_InitStructAll.ADC_NbrOfRegChannel = 1;
      ADC_Init(ADC1,&ADC_InitStructAll); // init the adc settings
      ADC_Cmd(ADC1,ENABLE); //enable adc
      // set internal reference voltage source and wait
}

void ADC_Calibrate(){
```

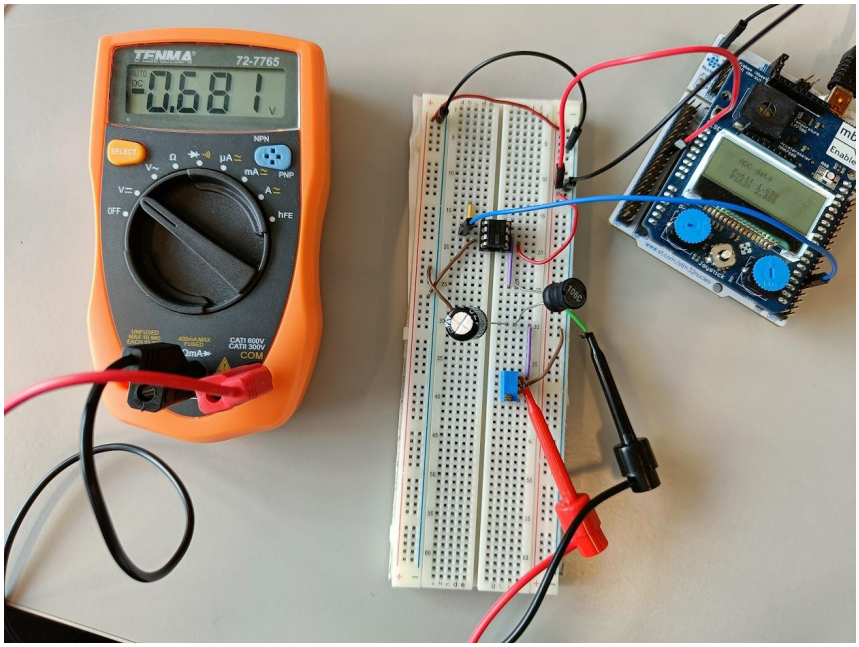
```
	ADC_VoltageRegulatorCmd(ADC1,ENABLE);
	//Wait for at least 10uS before continuing...
	for(uint32_t i = 0; i<10000;i++);

	ADC_Cmd(ADC1,DISABLE);
	while(ADC_GetDisableCmdStatus(ADC1)){} // wait for disable of ADC

	ADC_SelectCalibrationMode(ADC1,ADC_CalibrationMode_Single); //select
calibration mode
	ADC_StartCalibration(ADC1); //calibrate adc
	while(ADC_GetCalibrationStatus(ADC1)){} //wait for calibration
	for(uint32_t i = 0; i<100;i++);//wait more

	ADC_VrefintCmd(ADC1,ENABLE); // setup ref voltage to channel 18
	for(uint32_t i = 0; i<10000;i++); // wait for some time

	ADC_Cmd(ADC1,ENABLE);// turn on ADC
	while((!ADC_GetFlagStatus(ADC1,ADC_FLAG_RDY))){     } //wait for adc to turn
on

	ADC_RegularChannelConfig(ADC1, ADC_Channel_18, 1, ADC_SampleTime_19Cycles5);
//wait for 2.2us
	ADC_StartConversion(ADC1); // Start ADC read
	while (ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == 0); // Wait for ADC read

	uint16_t VREFINT_DATA = ADC_GetConversionValue(ADC1); // save measured data
	V_ABS = ((3.3 * (VREFINT_CAL / VREFINT_DATA)) / 4095); // calculate the
voltage/adc step
}

uint16_t ADC_measure_PA(uint8_t channel){
	uint16_t x;
	ADC_RegularChannelConfig(ADC1, channel, 1, ADC_SampleTime_1Cycles5);
	ADC_StartConversion(ADC1); // Start ADC read
	while (ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == 0); // Wait for ADC read
	x = ADC_GetConversionValue(ADC1) ; // savemeasured data
	return x;
}

void TIM2_IRQHandler(void) { //timer interrupt handler
	if(TIM_GetITStatus(TIM2,TIM_IT_Update) != RESET){ //if interrupt occurs
		TIM_ClearITPendingBit(TIM2,TIM_IT_Update); // Clear interrupt bit
		ADCupdate=1;
	}
}

void LCD_data_print(void){
```

```
	lcd_write_string((uint8_t*)"ADC data", fbuffer, 20, 0);
	sprintf(str,"V_out: %0.3f",(double)adc1* (double)V_ABS);
	lcd_write_string(str, fbuffer, 20, 2);
	sprintf(str,"PWM Duty: %0.3f",(double)count/(double)maxcount);
	lcd_write_string(str, fbuffer, 20, 3);
	lcd_push_buffer(fbuffer);
}

void timer16_pwm_init(void){
	//step1
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM16, ENABLE);
	//step3
	TIM_TimeBaseInitTypeDef TIM_InitStructure;
	TIM_TimeBaseStructInit(&TIM_InitStructure);
	TIM_InitStructure.TIM_ClockDivision = 0;
	TIM_InitStructure.TIM_Period = maxcount; //set the maximum period
	TIM_InitStructure.TIM_Prescaler = 64; //for 1MHz counting frequency
	TIM_TimeBaseInit(TIM16,&TIM_InitStructure);
	//step4
	TIM_OCInitTypeDef TIM_OCInitStruct;
	TIM_OCStructInit(&TIM_OCInitStruct);
	TIM_OCInitStruct.TIM_OCMode = TIM_OCMode_PWM1;
	TIM_OCInitStruct.TIM_OutputState = TIM_OutputState_Enable;
	TIM_OCInitStruct.TIM_Pulse = count;
	TIM_OCInitStruct.TIM_OCPolarity = TIM_OCPolarity_High;
	//step5
	TIM_OC1Init(TIM16,&TIM_OCInitStruct);
	//step6
	TIM_OC1PreloadConfig(TIM16,TIM_OCPreload_Enable);
	//step7
	TIM_CtrlPWMOutputs(TIM16, ENABLE);
	TIM_Cmd(TIM16,ENABLE);
}

void GPIO_set_AF1_PA6(void){
	//step2
	RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA,ENABLE); // Enable clock for GPIO
Port B
	GPIO_InitTypeDef GPIO_InitStructAll; // Define typedef struct for setting
pins
	GPIO_StructInit(&GPIO_InitStructAll);
	// Then set things that are not default.
	GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_AF;
	GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_6;
	GPIO_InitStructAll.GPIO_PuPd = GPIO_PuPd_NOPULL;
	GPIO_InitStructAll.GPIO_Speed = GPIO_Speed_50MHz;
	GPIO_Init(GPIOA, &GPIO_InitStructAll); // Setup of GPIO with the settings
```

```
chosen
	GPIO_PinAFConfig(GPIOA, GPIO_PinSource6, GPIO_AF_1);
}

int main(void)
{
	uart_init(9600);
	init_spi_lcd();
	memset(fbuffer,0x00,512); // Sets each element of the buffer to 0xAA
	lcd_push_buffer(fbuffer);
	ADC_setup_pA();
	ADC_Calibrate();
	initTimer2();
	GPIO_set_AF1_PA6();
	timer16_pwm_init();
	TIM_Cmd(TIM2,ENABLE);
	TIM_OCInitTypeDef TIM_OCInitStruct;
	TIM_OCStructInit(&TIM_OCInitStruct);

	while(1){
		if(ADCupdate == 1){
			adc1 = ADC_measure_PA(15);
			//adc2 = ADC_measure_PA(2);
			if( ((double)adc1* (double)V_ABS) < 1){count++;}
			else count--;
			TIM_OCInitStruct.TIM_OCMode = TIM_OCMode_PWM1;
			TIM_OCInitStruct.TIM_OutputState = TIM_OutputState_Enable;
			TIM_OCInitStruct.TIM_Pulse = count;
			TIM_OCInitStruct.TIM_OCPolarity = TIM_OCPolarity_High;
			TIM_OC1Init(TIM16,&TIM_OCInitStruct);
			LCD_data_print();
			ADCupdate=0;
		}
	}
}
```

Results:

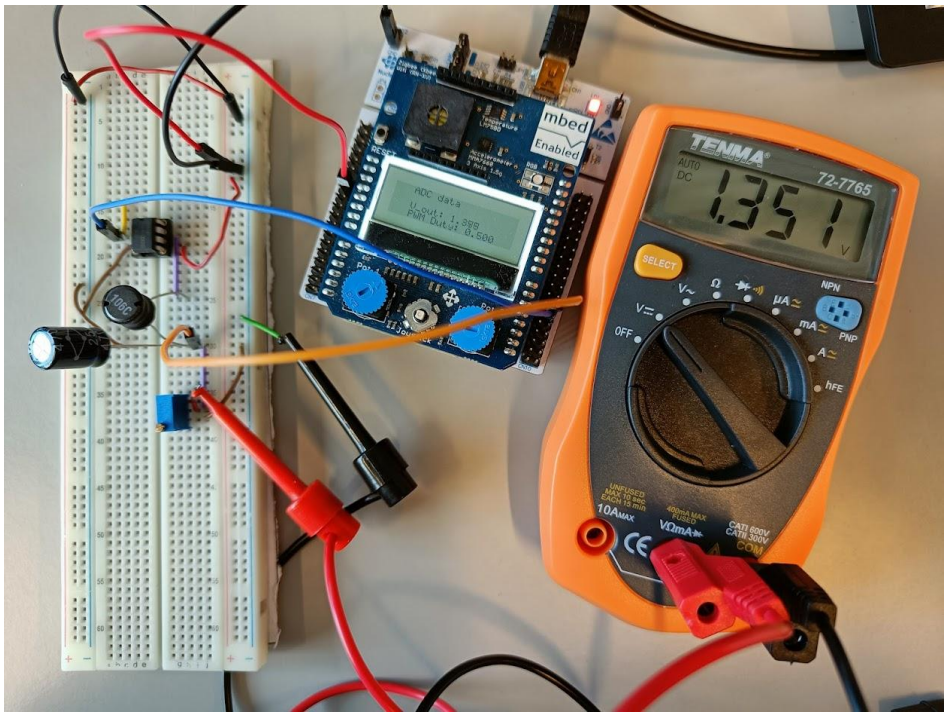
First we needed to build the circuit. We built it on a breadboard according to the circuit diagram.



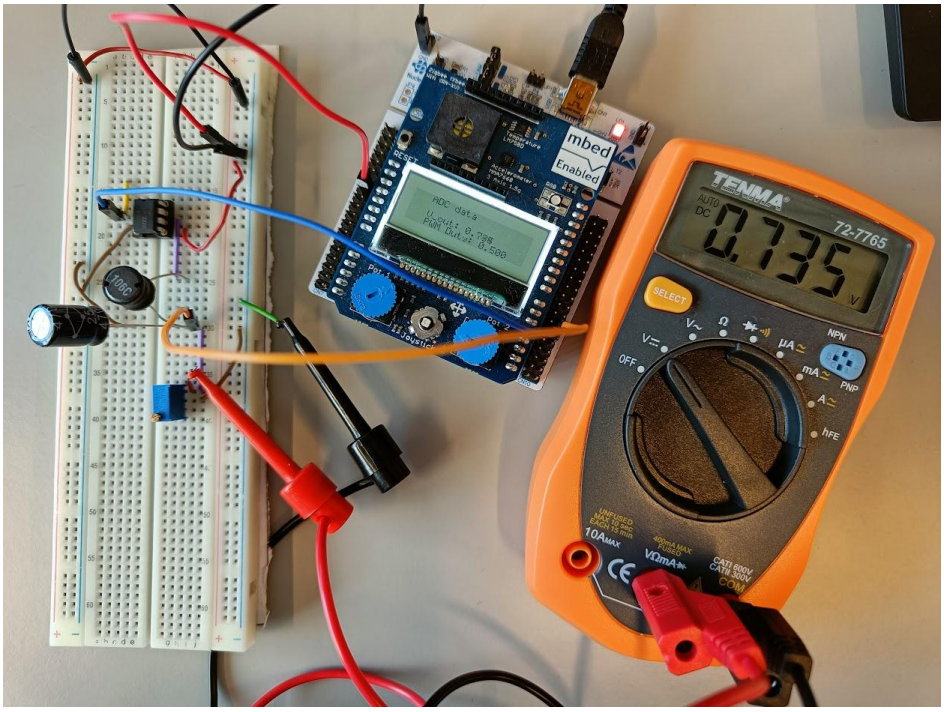
After the initial test was done to ensure the circuit is working properly we proceeded with the work. We set the load to 100 Ω, the duty cycle to 50% and connected the multimeter to the output. First we measured the output without any load and then repeated the measurement with the 100 Ω load. The results are collected in this table:

Load	Output Voltage
0 Ω	735 mV
100 Ω	1351 mV

Unloaded output (the load cable was actually disconnected, but on the photo it cannot be seen clearly):



Loaded output:

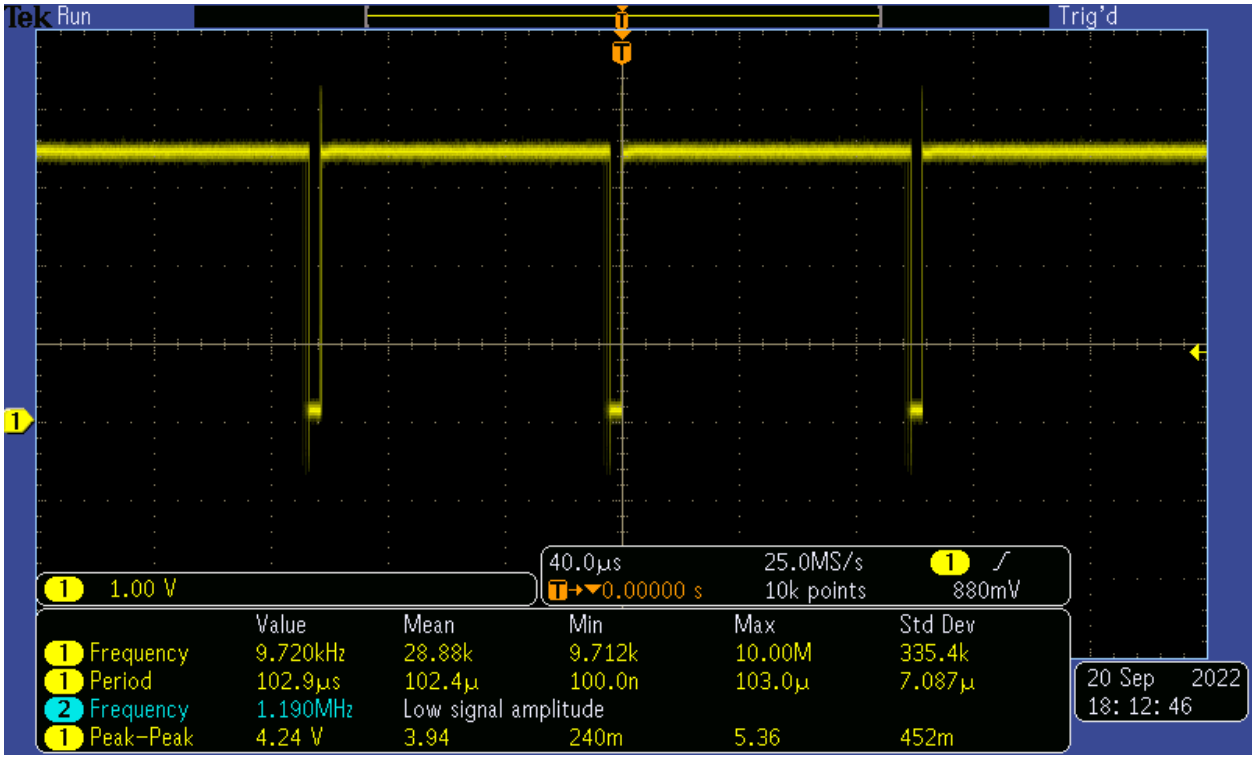


For the last exercise we added a feedback loop to the software for automatic PWN duty cycle correction. We tested the new software by measuring the output with Nol and Maximal load. To calculate the maximum load we measured the PWM signal with an oscilloscope. To determine the exact value we increased the load until the PWM reached almost 100% duty cycle and the output voltage was still 1 V. After this we measured the resistance of the potentiometer, it was 37.2 Ω, this is the maximum load. To calculate the maximum output current we need to solve this simple equation:

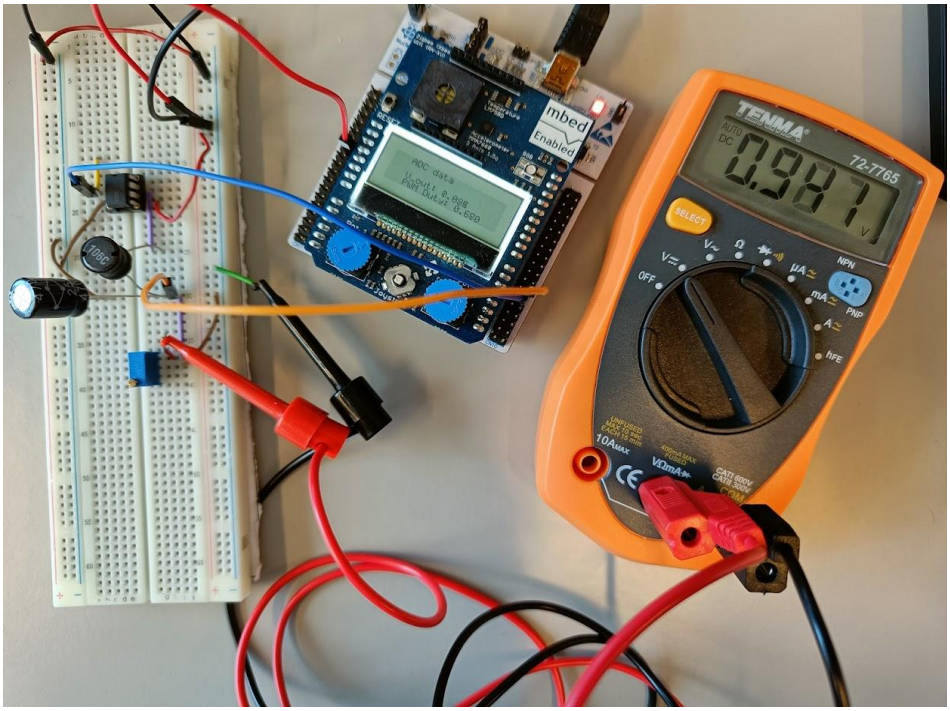
$$I_{Outmax} = \frac{U_{Out}}{R_{min}} = \frac{1\,V}{37.2\,\Omega} = 26.8\,mA$$

We took some pictures during the measurements. On the first the oscilloscope measurement can be seen to determine the maximum current and the test setup with a multimeter. On the other two photos the load was removed and the oscilloscope measurement was also recorded.

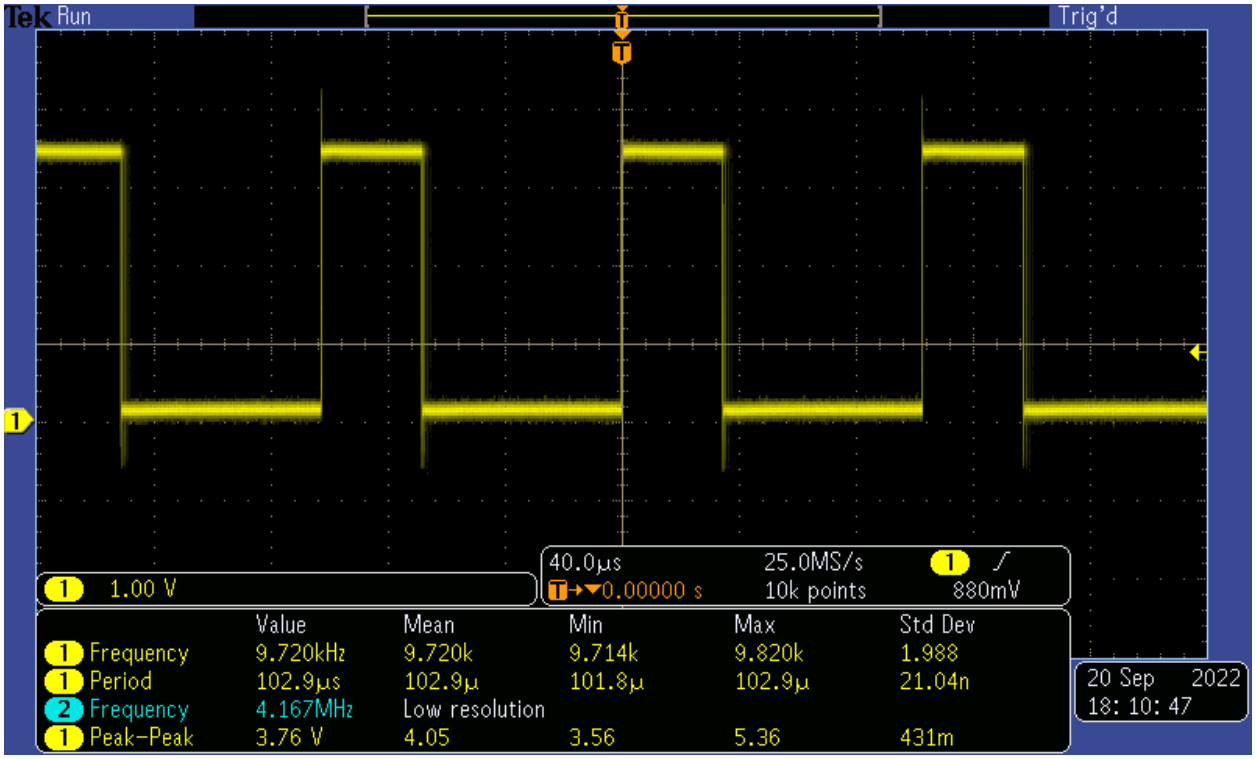
Picture of the oscilloscope measurement for maximum duty cycle:



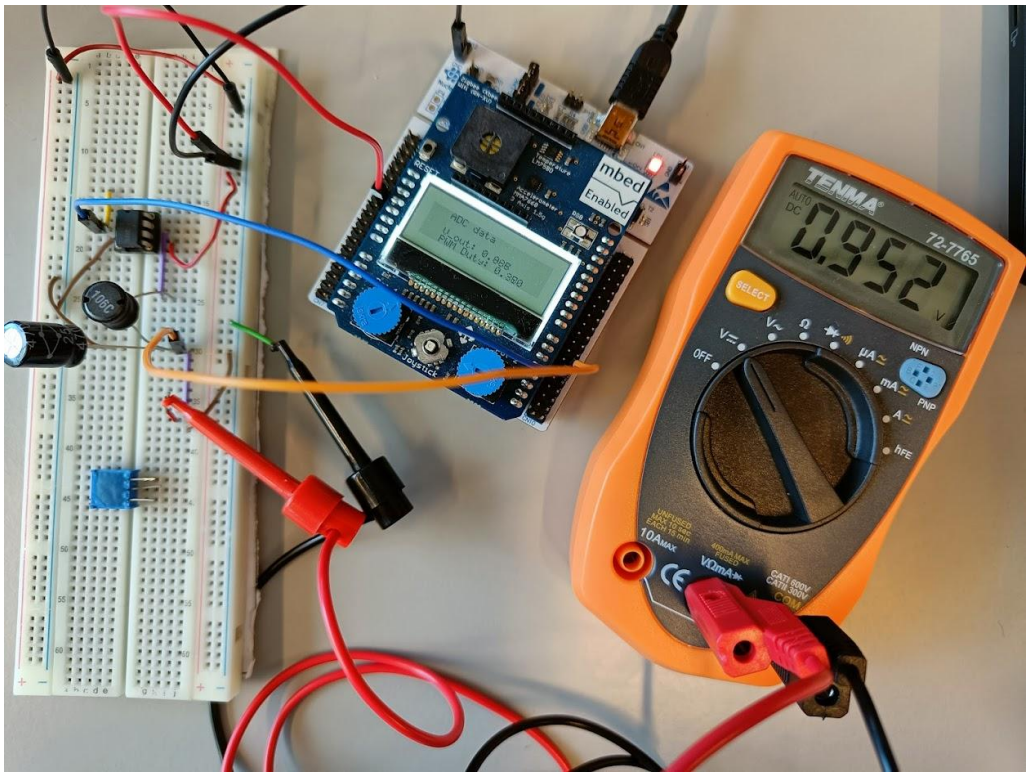
Test setup with maximum load attached:



Oscilloscope measurement for no load:

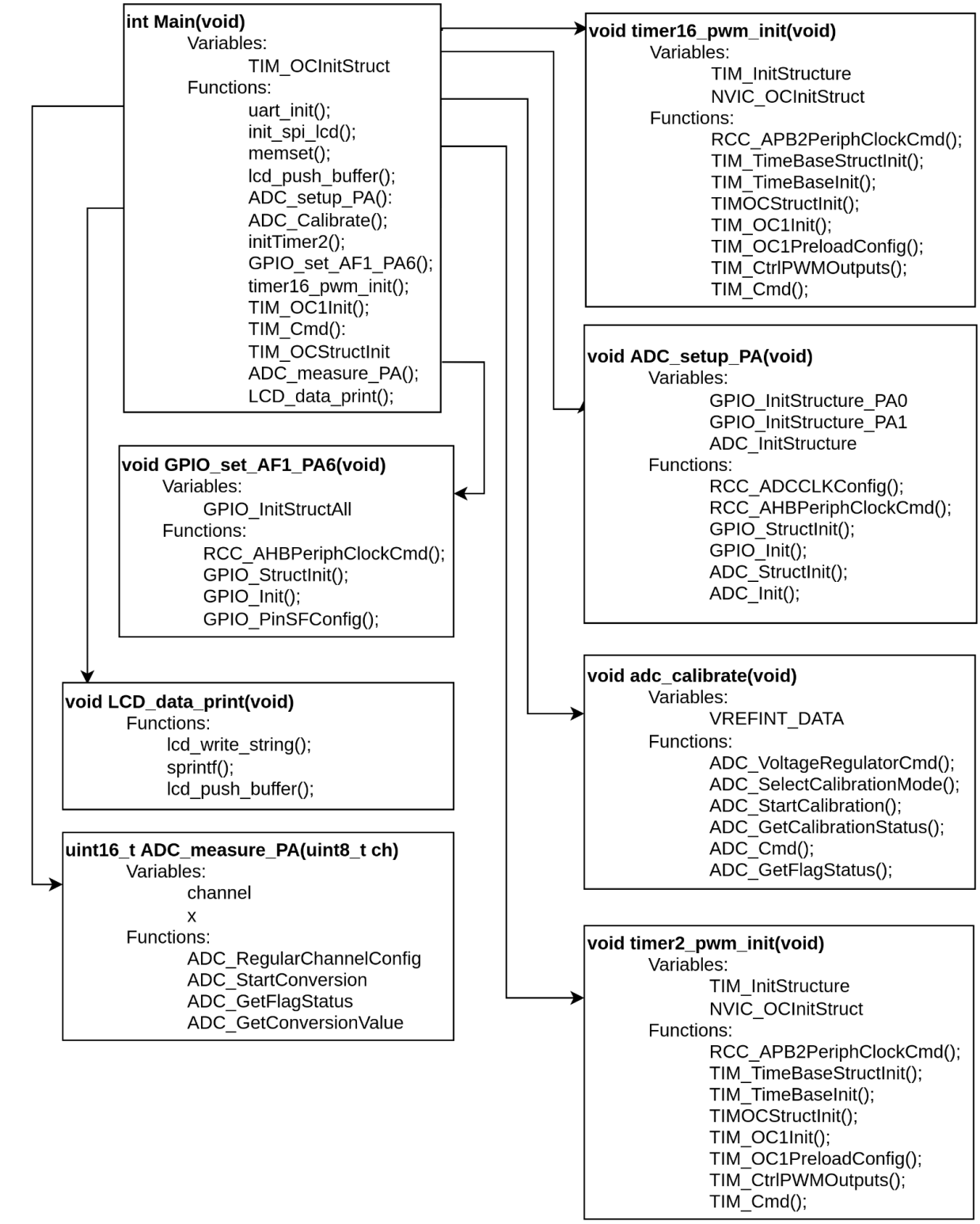


Test setup with no load attached:

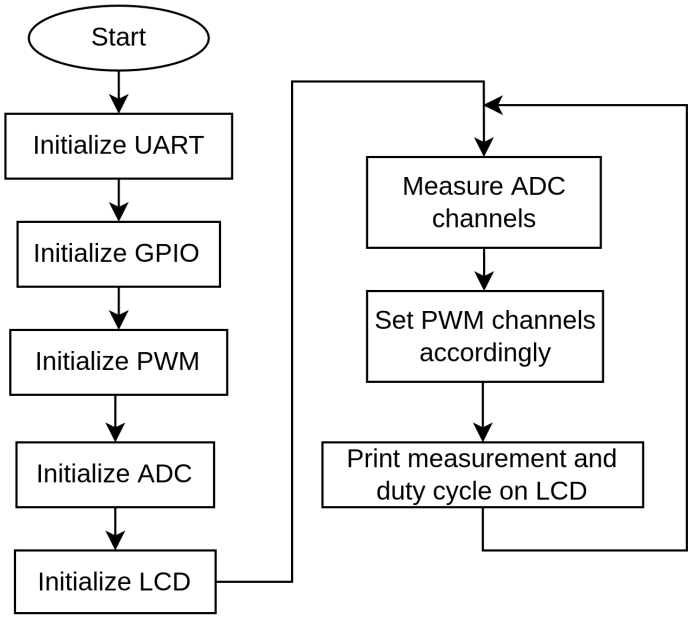


3. Servo Motor

Block diagram:



Flowchart:



Function description:

main

Syntax `int main(void);`

Parameters -

The function is the main function. It initializes the GPIO port, the TIMERs for PWM, the UART port and the ADC. It contains the main program loop that is continuously executed. During every cycle it measure the potentiometers with the ADC channels, transforms the data into PWM signals for the servos, sets the PWM duty cycle accordingly and prints the PWM pulse width in milliseconds to the LCD.

timer16_pwm_init

Syntax `void timer16_pwm_init(void);`

Parameters -

The function initializes the timer with a PWM output with the required parameters.

GPIO_set_AF1_PA6

Syntax `void GPIO_set_AF1_PA6(void);`

Parameters -

The function designates the PA6 GPIO port for the timer as output with the required settings.

timer2_pwm_init

Syntax `void initTimer(void);`

Parameters -

The function initializes the timer with a PWM output with the required parameters.

ADC_setup_PA

Syntax `void ADC_setup_PA(void);`

Parameters -

This function configures the ADC. The parameters have been set according to the assignment description.

ADC_Calibrate

Syntax `void ADC_Calibrate(void);`

Parameters -

This function calibrates the ADC according to the assignment description.

ADC_measure_PA

Syntax `uint16_t ADC_measure_PA(uint8_t ch);`

Parameters ch: ADC channel to be measured

This function reads either ADC channel 1 or 2 depending on the input. For this purpose the channel is collected for 1.5 clock cycles and then the measurement is started. As soon as this is completed, the measured value is output in the form of a 16-bit integer.

LCD_data_print

Syntax `void LCD_data_print(void);`

Parameters -

This function prints the PWM pulse widths to the lcd.

Source Code:

```
#include "stm32f30x_conf.h" // STM32 config
#include "30010_io.h"           // Input/output library for this course
#include "flash.h"
#include "lcd.h"
#include <string.h>

#define VREFINT_CAL *((uint16_t*) ((uint32_t) 0x1FFFF7BA))
float volatile V_ABS;
char  str[16];
uint8_t fbuffer[512];
uint16_t volatile adc1;
uint16_t volatile adc2;
uint8_t volatile ADCupdate=0;
uint32_t map1=0;
uint32_t map2=0;

void ADC_setup_pA(void){
      RCC_ADCCLKConfig(RCC_ADC12PLLCLK_Div8); //adc clk
      RCC_AHBPeriphClockCmd(RCC_AHBPeriph_ADC12, ENABLE); //enable adc
      RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA, ENABLE); //gpio clock
      GPIO_InitTypeDef GPIO_InitStructAll; // Define typedef struct for setting
```

```
pins
	GPIO_StructInit(&GPIO_InitStructAll); // Initialize GPIO struct
	GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_AN; // Set as input
	GPIO_InitStructAll.GPIO_PuPd = GPIO_PuPd_DOWN; // Set as pull down
	GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_0; // Set so the configuration is on
pin 4
	GPIO_Init(GPIOA, &GPIO_InitStructAll); // Setup of GPIO with the settings
chosen
	GPIO_StructInit(&GPIO_InitStructAll); // Initialize GPIO struct
	GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_AN; // Set as input
	GPIO_InitStructAll.GPIO_PuPd = GPIO_PuPd_DOWN; // Set as pull down
	GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_1; // Set so the configuration is on
pin 4
	GPIO_Init(GPIOA, &GPIO_InitStructAll); // Setup of GPIO with the settings
chosen
	ADC_InitTypeDef ADC_InitStructAll; //struct for adc config
	ADC_StructInit(&ADC_InitStructAll); //settings for the adc
	ADC_InitStructAll.ADC_ContinuousConvMode = DISABLE;
	ADC_InitStructAll.ADC_Resolution = ADC_Resolution_12b;
	ADC_InitStructAll.ADC_ExternalTrigEventEdge =
ADC_ExternalTrigEventEdge_None;
	ADC_InitStructAll.ADC_DataAlign = ADC_DataAlign_Right;
	ADC_InitStructAll.ADC_NbrOfRegChannel = 1;
	ADC_Init(ADC1,&ADC_InitStructAll); // init the adc settings
	ADC_Cmd(ADC1,ENABLE); //enable adc
	// set internal reference voltage source and wait
}

void ADC_Calibrate(){
	ADC_VoltageRegulatorCmd(ADC1,ENABLE);
	//Wait for at least 10uS before continuing...
	for(uint32_t i = 0; i<10000;i++);
	ADC_Cmd(ADC1,DISABLE);
	while(ADC_GetDisableCmdStatus(ADC1)){} // wait for disable of ADC
	ADC_SelectCalibrationMode(ADC1,ADC_CalibrationMode_Single); //select
calibration mode
	ADC_StartCalibration(ADC1); //calibrate adc
	while(ADC_GetCalibrationStatus(ADC1)){} //wait for calibration
	for(uint32_t i = 0; i<100;i++);//wait more
	ADC_VrefintCmd(ADC1,ENABLE); // setup ref voltage to channel 18
	for(uint32_t i = 0; i<10000;i++); // wait for some time
	ADC_Cmd(ADC1,ENABLE);// turn on ADC
	while((!ADC_GetFlagStatus(ADC1,ADC_FLAG_RDY))){     } //wait for adc to turn
on
	ADC_RegularChannelConfig(ADC1, ADC_Channel_18, 1, ADC_SampleTime_19Cycles5);
//wait for 2.2us
	ADC_StartConversion(ADC1); // Start ADC read
```

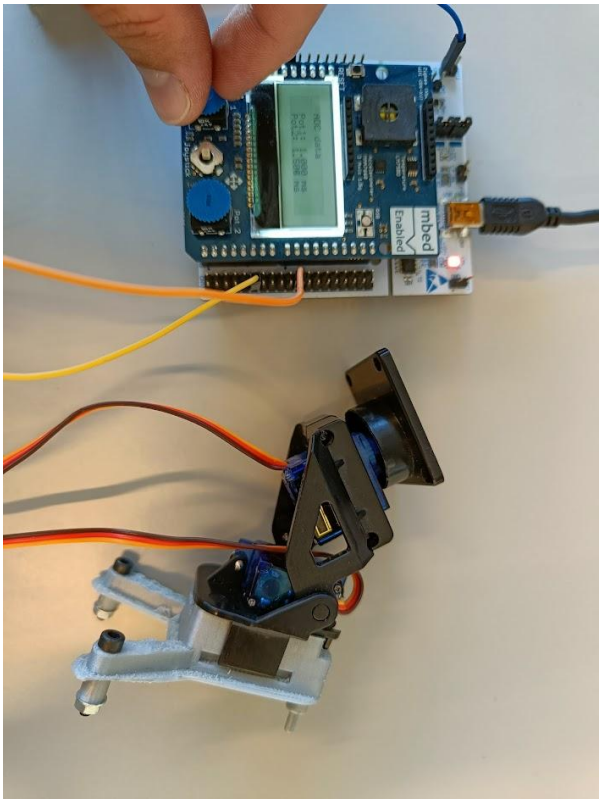
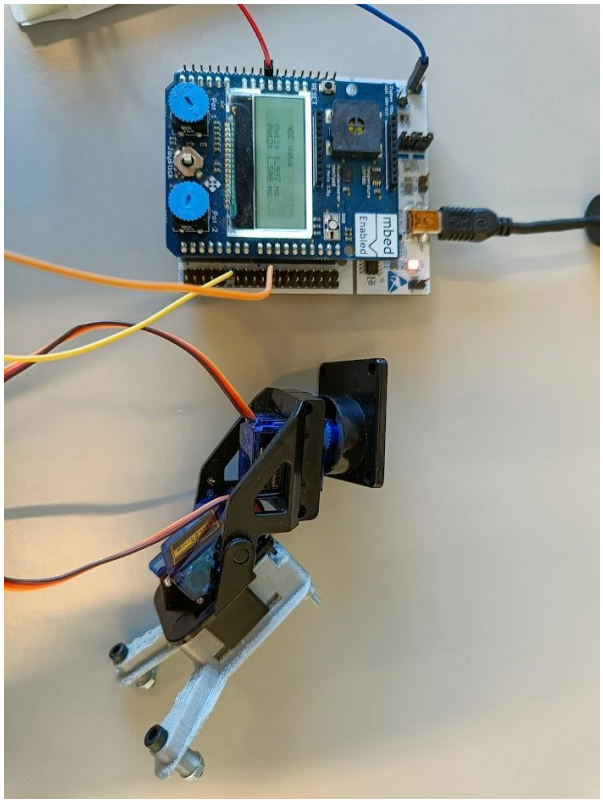
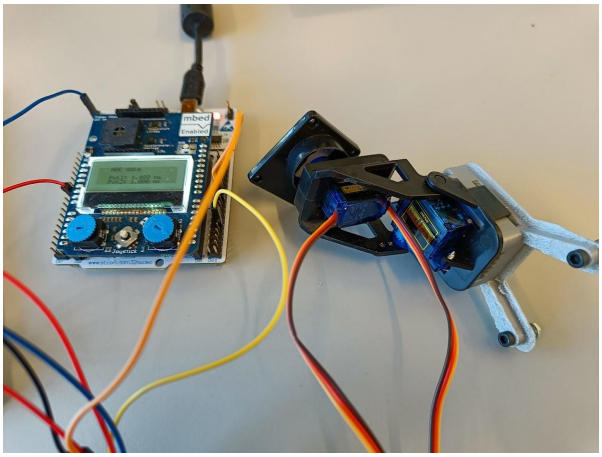
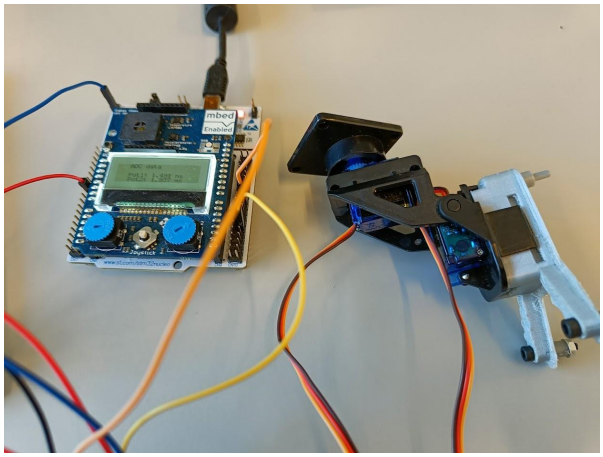
```
	while (ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == 0); // Wait for ADC read
	uint16_t VREFINT_DATA = ADC_GetConversionValue(ADC1); // save measured data
	V_ABS = ((3.3 * (VREFINT_CAL / VREFINT_DATA)) / 4095); // calculate the
voltage/adc step
}

uint16_t ADC_measure_PA(uint8_t channel){
	uint16_t x;
	ADC_RegularChannelConfig(ADC1, channel, 1, ADC_SampleTime_1Cycles5);
	ADC_StartConversion(ADC1); // Start ADC read
	while (ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == 0); // Wait for ADC read
	x = ADC_GetConversionValue(ADC1) ; // savemeasured data
	return x;
}

void LCD_data_print(void){
	lcd_write_string((uint8_t*)"ADC data", fbuffer, 20, 0);
	sprintf(str,"Pot1: %0.3f ms",(double)map1/1000);
	lcd_write_string(str, fbuffer, 20, 2);
	sprintf(str,"Pot2: %0.3f ms",(double)map2/1000);
	lcd_write_string(str, fbuffer, 20, 3);
	lcd_push_buffer(fbuffer);
}

void timer16_pwm_init(void){
	//step1
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM16, ENABLE);
	//step3
	TIM_TimeBaseInitTypeDef TIM_InitStructure;
	TIM_TimeBaseStructInit(&TIM_InitStructure);
	TIM_InitStructure.TIM_ClockDivision = 0;
	TIM_InitStructure.TIM_Period = 20000; //set the maximum period
	TIM_InitStructure.TIM_Prescaler = 64; //for 1MHz counting frequency
	TIM_TimeBaseInit(TIM16,&TIM_InitStructure);
	//step4
	TIM_OCInitTypeDef TIM_OCInitStruct;
	TIM_OCStructInit(&TIM_OCInitStruct);
	TIM_OCInitStruct.TIM_OCMode = TIM_OCMode_PWM1;
	TIM_OCInitStruct.TIM_OutputState = TIM_OutputState_Enable;
	//TIM_OCInitStruct.TIM_Pulse = 0x01f4; // 0x0000 <-> 0xFFFF
	TIM_OCInitStruct.TIM_Pulse = 1500; // 0 - 20000, 1000-2000
	TIM_OCInitStruct.TIM_OCPolarity = TIM_OCPolarity_High;
	//step5
	TIM_OC1Init(TIM16,&TIM_OCInitStruct);
	//step6
	TIM_OC1PreloadConfig(TIM16,TIM_OCPreload_Enable);
	//step7
```

```
	TIM_CtrlPWMOutputs(TIM16, ENABLE);
	TIM_Cmd(TIM16,ENABLE);
}

void GPIO_set_AF1_PA6(void){
	//step2
	RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA,ENABLE); // Enable clock for GPIO
Port B
	//RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOB,ENABLE); // Enable clock for
GPIO Port B
	GPIO_InitTypeDef GPIO_InitStructAll; // Define typedef struct for setting
pins
	GPIO_StructInit(&GPIO_InitStructAll);
	// Then set things that are not default.
	GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_AF;
	//GPIO_InitStructAll.GPIO_OType = GPIO_OType_PP;
	GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_6;
	//GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_4;
	GPIO_InitStructAll.GPIO_PuPd = GPIO_PuPd_NOPULL;
	GPIO_InitStructAll.GPIO_Speed = GPIO_Speed_50MHz;
	GPIO_Init(GPIOA, &GPIO_InitStructAll); // Setup of GPIO with the settings
chosen
	//GPIO_Init(GPIOB, &GPIO_InitStructAll); // Setup of GPIO with the settings
chosen
	GPIO_PinAFConfig(GPIOA, GPIO_PinSource6, GPIO_AF_1);
	//GPIO_PinAFConfig(GPIOB, GPIO_PinSource4, GPIO_AF_1);
}

void timer2_pwm_init(void){
	//step1
	RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
	//step3
	TIM_TimeBaseInitTypeDef TIM_InitStructure;
	TIM_TimeBaseStructInit(&TIM_InitStructure);
	TIM_InitStructure.TIM_ClockDivision = 0;
	TIM_InitStructure.TIM_Period = 20000; //set the maximum period
	TIM_InitStructure.TIM_Prescaler = 64; //for 1MHz counting frequency
	TIM_TimeBaseInit(TIM2,&TIM_InitStructure);
	//step4
	TIM_OCInitTypeDef TIM_OCInitStruct;
	TIM_OCStructInit(&TIM_OCInitStruct);
	TIM_OCInitStruct.TIM_OCMode = TIM_OCMode_PWM1;
	TIM_OCInitStruct.TIM_OutputState = TIM_OutputState_Enable;
	TIM_OCInitStruct.TIM_Pulse = 1500; // 0 - 20000, 1000-2000
	TIM_OCInitStruct.TIM_OCPolarity = TIM_OCPolarity_High;
	//step5
	TIM_OC4Init(TIM2,&TIM_OCInitStruct);
```

```
	//step6
	TIM_OC4PreloadConfig(TIM2,TIM_OCPreload_Enable);
	//step7
	TIM_CtrlPWMOutputs(TIM2, ENABLE);
	TIM_Cmd(TIM2,ENABLE);
}

void GPIO_set_AF1_PB11(void){
	//step2
	//RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA,ENABLE); // Enable clock for
GPIO Port B
	RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOB,ENABLE); // Enable clock for GPIO
Port B
	GPIO_InitTypeDef GPIO_InitStructAll; // Define typedef struct for setting
pins
	GPIO_StructInit(&GPIO_InitStructAll);
	// Then set things that are not default.
	GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_AF;
	GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_11;
	GPIO_InitStructAll.GPIO_PuPd = GPIO_PuPd_NOPULL;
	GPIO_InitStructAll.GPIO_Speed = GPIO_Speed_50MHz;
	GPIO_Init(GPIOB, &GPIO_InitStructAll); // Setup of GPIO with the settings
chosen
	GPIO_PinAFConfig(GPIOB, GPIO_PinSource11, GPIO_AF_1);
}

int main(void)
{
	uart_init(9600);
	init_spi_lcd();
	memset(fbuffer,0x00,512); // Sets each element of the buffer to 0xAA
	lcd_push_buffer(fbuffer);
	ADC_setup_pA();
	ADC_Calibrate();
	GPIO_set_AF1_PA6();
	timer16_pwm_init();
	GPIO_set_AF1_PB11();
	timer2_pwm_init();
	TIM_OCInitTypeDef TIM_OCInitStruct;
	TIM_OCStructInit(&TIM_OCInitStruct);

	while(1){
		printf("alma\n");
		adc1 = ADC_measure_PA(1);
		adc2 = ADC_measure_PA(2);
		map1 = 1000+(adc1*1000)/4096;
		map2 = 1000+(adc2*1000)/4096;
```

```
            //printf("%d\n",map1);
            TIM_OCInitStruct.TIM_OCMode = TIM_OCMode_PWM1;
            TIM_OCInitStruct.TIM_OutputState = TIM_OutputState_Enable;
            TIM_OCInitStruct.TIM_Pulse = map1;
            TIM_OCInitStruct.TIM_OCPolarity = TIM_OCPolarity_High;
            TIM_OC1Init(TIM16,&TIM_OCInitStruct);

            TIM_OCInitStruct.TIM_OCMode = TIM_OCMode_PWM1;
            TIM_OCInitStruct.TIM_OutputState = TIM_OutputState_Enable;
            TIM_OCInitStruct.TIM_Pulse = map2;
            TIM_OCInitStruct.TIM_OCPolarity = TIM_OCPolarity_High;
            TIM_OC4Init(TIM2,&TIM_OCInitStruct);

            LCD_data_print();
      }
}
```

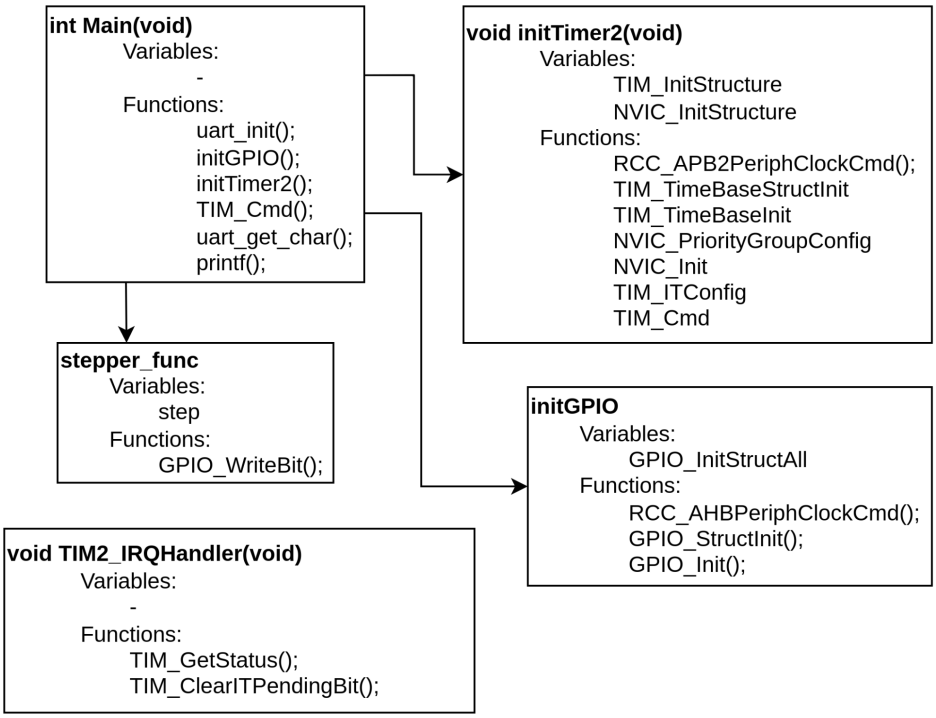
Results:
We tested the circuit by setting both potentiometers to left, center and right positions. The servos moved accordingly, and worked perfectly. Some photos of the results are presented in the table below.

 Potentiometer 1 most left position	 Potentiometer 1 most right position
 Potentiometer 2 most left position	 Potentiometer 2 most right position

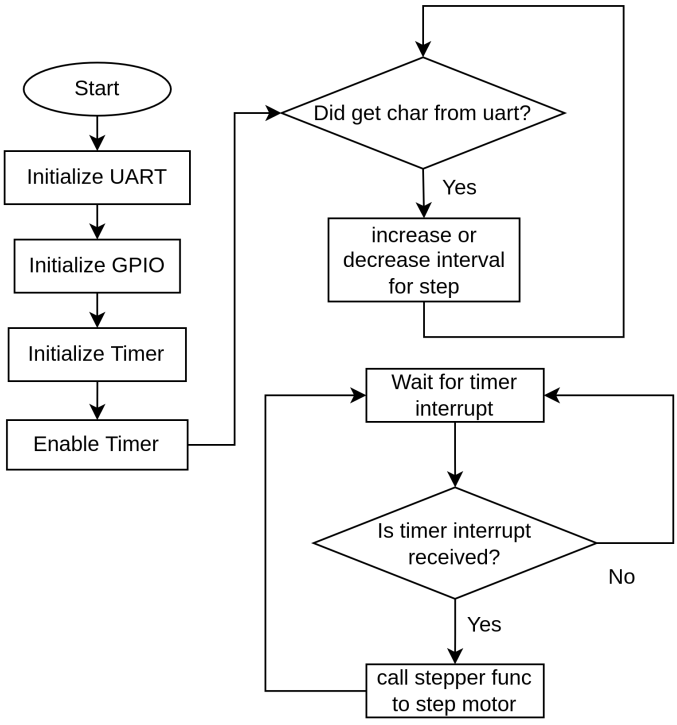
We also recorded a video for better documentation:
https://photos.app.goo.gl/kXdfVRokPhfbez7Q9

4. Stepper Motor

Block Diagram:



Flow Chart:



Function Description:

main

Syntax `int main(void);`

Parameters -

The function is the main function. It initializes the GPIO ports, the TIMER and the UART port. It contains the main program loop that is continuously executed. During every cycle it waits for an uart character. If the right character is received it increases or decreases the step interval accordingly.

initTimer2

Syntax `void initTimer(void);`

Parameters -

The function initializes the required timer for executing a step according to the defined interval.

TIM2_IRQHandler

Syntax `void TIM2_IRQHandler(void);`

Parameters -

The function is activated when a timer interrupt occurs. Increases the step to execute the next step and then executes it by running the stepper_func function.

initGPIO

Syntax `void initGPIO(void);`

Parameters -

The function initializes the four GPIO ports as outputs.

stepper_func

Syntax `void stepperFunc(uint8_t step);`

Parameters -

The function executes the correct step according to the value of the step variable.

Source code:

```
#include "stm32f30x_conf.h" // STM32 config
#include "30010_io.h"           // Input/output library for this course
#include "flash.h"
#include "lcd.h"
#include <string.h>

//uint8_t positions[4] = { 0b0101, 0b1001, 0b1010, 0b0110 };
uint16_t interval=100;
uint8_t current_step=0;

void initTimer2(void){
```

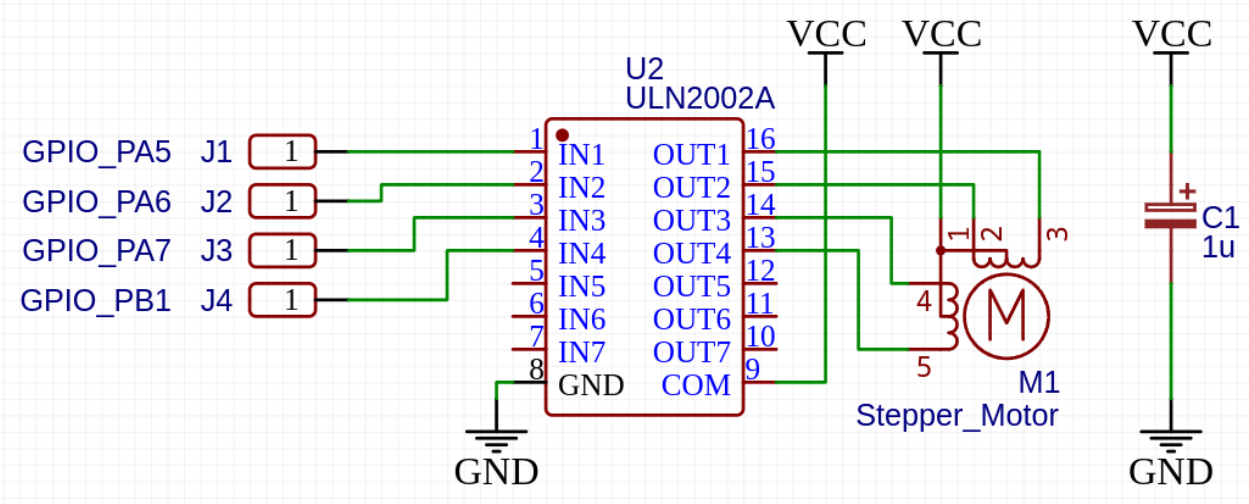
```
	RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2,ENABLE);
	NVIC_InitTypeDef NVIC_InitStructure;
	// NVIC for timer
	NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
	NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
	NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
	NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
	NVIC_Init(&NVIC_InitStructure);
	TIM_ITConfig(TIM2,TIM_IT_Update,ENABLE);
	TIM_Cmd(TIM2,ENABLE);
	//Settings timer
	RCC->APB1ENR |= RCC_APB1Periph_TIM2; // Enable clock line to timer 2
	TIM2->CR1=0xB01;
	TIM2->PSC=6399; //change pre-scaler frequency to 10kHz
	TIM2->ARR=interval; //count up to 100
	TIM2->DIER |= 0x0001; // Enable timer 2 interrupts
    //NVIC settings
	NVIC_SetPriority(TIM2_IRQn, 1); // Set interrupt priority interrupts
	NVIC_EnableIRQ(TIM2_IRQn); // Enable interrupt
	TIM_Cmd(TIM2,DISABLE);
}

void TIM2_IRQHandler(void) { //timer interrupt handler
	if(TIM_GetITStatus(TIM2,TIM_IT_Update) != RESET){ //if interrupt occurs
		stepper_func(current_step);
		current_step=(current_step+1)%4;
		//printf("%d\n", current_step);
		TIM_ClearITPendingBit(TIM2,TIM_IT_Update); // Clear interrupt bit
	}
}

void initGPIO(void){
	//RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOC,ENABLE); // Enable clock for
GPIO Port C
	RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOB,ENABLE); // Enable clock for GPIO
Port B
	RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA,ENABLE); // Enable clock for GPIO
Port A

	GPIO_InitTypeDef GPIO_InitStructAll; // Define typedef struct for setting
pins

	// Sets PA9 to output
	GPIO_StructInit(&GPIO_InitStructAll); // Initialize GPIO struct
	GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_OUT; // Set as output
	GPIO_InitStructAll.GPIO_OType = GPIO_OType_PP; // Set as Push-Pull
	GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_5; // Set so the configuration is on
```

```
pin 9
	GPIO_InitStructAll.GPIO_Speed = GPIO_Speed_2MHz; // Set speed to 2 MHz
	// For all options see SPL/inc/stm32f30x_gpio.h
	GPIO_Init(GPIOA, &GPIO_InitStructAll); // Setup of GPIO with the settings
chosen

	// Sets PA9 to output
	GPIO_StructInit(&GPIO_InitStructAll); // Initialize GPIO struct
	GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_OUT; // Set as output
	GPIO_InitStructAll.GPIO_OType = GPIO_OType_PP; // Set as Push-Pull
	GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_6; // Set so the configuration is on
pin 9
	GPIO_InitStructAll.GPIO_Speed = GPIO_Speed_2MHz; // Set speed to 2 MHz
	// For all options see SPL/inc/stm32f30x_gpio.h
	GPIO_Init(GPIOA, &GPIO_InitStructAll); // Setup of GPIO with the settings
chosen

	// Sets PA9 to output
	GPIO_StructInit(&GPIO_InitStructAll); // Initialize GPIO struct
	GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_OUT; // Set as output
	GPIO_InitStructAll.GPIO_OType = GPIO_OType_PP; // Set as Push-Pull
	GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_7; // Set so the configuration is on
pin 9
	GPIO_InitStructAll.GPIO_Speed = GPIO_Speed_2MHz; // Set speed to 2 MHz
	// For all options see SPL/inc/stm32f30x_gpio.h
	GPIO_Init(GPIOA, &GPIO_InitStructAll); // Setup of GPIO with the settings
chosen

	// Sets PA9 to output
	GPIO_StructInit(&GPIO_InitStructAll); // Initialize GPIO struct
	GPIO_InitStructAll.GPIO_Mode = GPIO_Mode_OUT; // Set as output
	GPIO_InitStructAll.GPIO_OType = GPIO_OType_PP; // Set as Push-Pull
	GPIO_InitStructAll.GPIO_Pin = GPIO_Pin_1; // Set so the configuration is on
pin 9
	GPIO_InitStructAll.GPIO_Speed = GPIO_Speed_2MHz; // Set speed to 2 MHz
	// For all options see SPL/inc/stm32f30x_gpio.h
	GPIO_Init(GPIOB, &GPIO_InitStructAll); // Setup of GPIO with the settings
chosen
}

void stepper_func(uint8_t step){
	switch(step){
		case 0 :
			GPIO_WriteBit(GPIOA , GPIO_Pin_5, 0); //set red led to enabled
or disabled
			GPIO_WriteBit(GPIOA , GPIO_Pin_6, 1); //set green led to
enabled or disabled
```

```
                    GPIO_WriteBit(GPIOA , GPIO_Pin_7, 0); //set blue led to enabled
or disabled
                    GPIO_WriteBit(GPIOB , GPIO_Pin_1, 1); //set red led to enabled
or disabled
             break;

             case 1 :
                    GPIO_WriteBit(GPIOA , GPIO_Pin_5, 1); //set red led to enabled
or disabled
                    GPIO_WriteBit(GPIOA , GPIO_Pin_6, 0); //set green led to
enabled or disabled
                    GPIO_WriteBit(GPIOA , GPIO_Pin_7, 0); //set blue led to enabled
or disabled
                    GPIO_WriteBit(GPIOB , GPIO_Pin_1, 1); //set red led to enabled
or disabled
             break;

             case 2 :
                    GPIO_WriteBit(GPIOA , GPIO_Pin_5, 1); //set red led to enabled
or disabled
                    GPIO_WriteBit(GPIOA , GPIO_Pin_6, 0); //set green led to
enabled or disabled
                    GPIO_WriteBit(GPIOA , GPIO_Pin_7, 1); //set blue led to enabled
or disabled
                    GPIO_WriteBit(GPIOB , GPIO_Pin_1, 0); //set red led to enabled
or disabled
             break;

             case 3 :
                    GPIO_WriteBit(GPIOA , GPIO_Pin_5, 0); //set red led to enabled
or disabled
                    GPIO_WriteBit(GPIOA , GPIO_Pin_6, 1); //set green led to
enabled or disabled
                    GPIO_WriteBit(GPIOA , GPIO_Pin_7, 1); //set blue led to enabled
or disabled
                    GPIO_WriteBit(GPIOB , GPIO_Pin_1, 0); //set red led to enabled
or disabled
             break;
      }
}

int main(void)
{
      uart_init(9600);
      initGPIO();
      initTimer2();
      TIM_Cmd(TIM2,ENABLE);
```

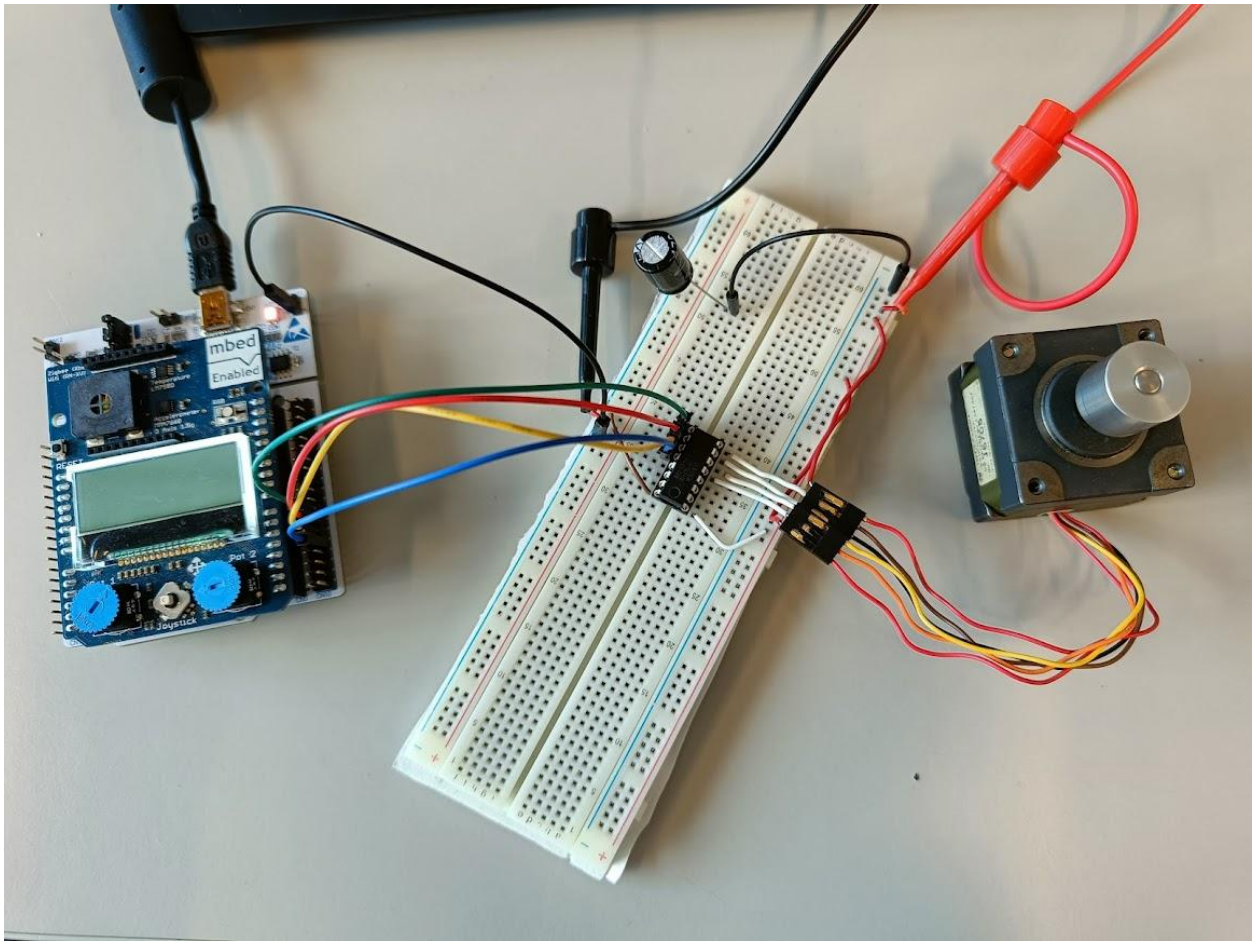
```
	while(1){
		switch(uart_get_char()){
		case 'w' :
if(interval<=1000){interval+=1;TIM2->ARR=interval;printf("%d\n",interval);};
		break;

		case 's' :
if(interval>30){interval-=1;TIM2->ARR=interval;printf("%d\n",interval);}
		break;
		}
	}
}
```

Results:
After writing the code we built the circuit based on this circuit diagram.



For testing the circuit we set the step interval to 5 ms and the input voltage for the motor to 5 V as it was told in the exercise. Thes stepper motor did not run correctly. For the motor to run correctly we had to increase the step interval to 5.5 ms. While setting the interval we also tested the uart input and it worked correctly, by setting the right step value and printing it to the console. After some testing we concluded that the motor needs at least 5.5 ms step interval to start up, but after it is running we can lower the step interval to 5 ms and it still rotates correctly. So the exact value for the motor to run correctly in every scenario we decided that the minimum step value should be chosen as 5.5 ms for 5 V input voltage. If we would increase the input voltage this value can be lowered significantly. With a 9 V input the minimum interval was as low as 4 ms. In the below pictures you can see the built circuit. We also included a video file for better documentation.

The breadboard circuit:



Voltage settings on the external power supply:

Video of the stepper motor while running:
https://photos.app.goo.gl/bp95AYjG3QgTUxNq8