Final Project

DC Motor Speed Controller

EGC433– Embedded Systems

MAY 16, 2022

Christopher Lepore

Abstract

In this project we designed a Speed Controller for a DC motor that can maintain the RPM when friction is applied to the wheel using a Nucleo-64 microcontroller. This project also includes an LCD that displays the current RPM and a potentiometer that is used to set the speed of the motor when the speed control mode is off. The potentiometer is read by an ADC and the DC motor output is controlled by a DAC. The speed controller's mode is determined by an external switch. When the speed control is on, the potentiometer is no longer used, and the microcontroller will maintain the RPM of the motor found when the mode was enabled. The method used to maintain the RPM is called bang bang which will slowly increase or decrease the output till it gets to its target value (RPM), but after it reached the value, the output will oscillate slightly at the target value. The RPM of the motor is read through a light diode/LED pair sensor which will output a logic 1 when there is no obstruction between them. A disk with a hole at the edge is attached to the motor and the light diode and LED are placed on opposite sides of the disk. Using input capture, the frequency of the sensor input (PA6) was found and multiplied by 60 to give the motor's RPM. These features allow the microcontroller to maintain the motor's RPM even when external friction is applied.

Table of contents

Introduction

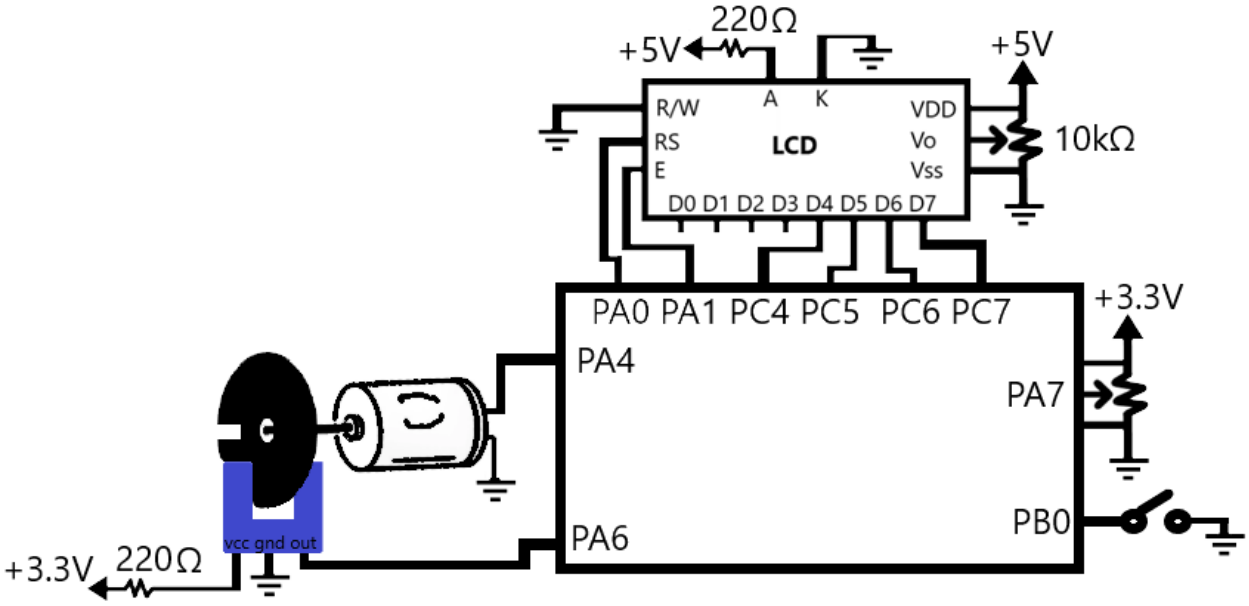
In this project we designed a Speed Controller for a DC motor that can maintain the RPM when friction is applied to the wheel using a Nucleo-64 microcontroller. This project also includes an LCD that displays the current RPM and a potentiometer that is used to set the speed of the motor when the speed control mode is off. The speed controller's mode is determined by an external switch. This project introduced coding with analog inputs and outputs in C programing. This also introduced the input capture feature of a microcontroller and wiring a high number of desired input and output ports of the STM32F446RE circuit board to different sensors. The Speed Controller determines the mode and RPM of the motor and can increase and decrease the motor's power as needed through the DAC. By designing this speed controller, a deeper comprehension of programing with analog for the STM32F446RE circuit board was achieved.

Theory

Speed control has existed since the early 1900s in cars such as the Wilson-Pilcher. These cars had a lever which could be used to set the speed that will be maintained by the engine. The modern speed control was developed in 1948 by Ralph Teetor who was a mechanical engineer and a blind inventor. He developed this idea due to his frustration with people speeding up and slowing down consistently on the road. A big factor in the development of speed control was the 35-mph speed limit which was put in place by the United States during World War 2. This was done to reduce gas use and tire wear. This method of speed control provided resistance to the gas peddle to prevent the driver from going above the desired speed. Teetor's idea included a dashboard speed selector which sets at what angle the resistance will be applied. The first car with Teetor's speed control was the 1958 Chrysler Imperial which was nicknamed the auto-pilot. Then in 1956 the American Motors Corporation (AMC) brought to market a low-priced speed control system for large cars with automatic transmissions. Another revolution in speed control was introduced by Daniel Aaron Wisner who included digital memory to his speed control system. Wisner's system was the first electronic device to control a car in 1968. The speed control system became even more popular in the U.S after the 1973 oil crisis and rising fuel prices due to its ability to save gas. Then the speed control system became commercially developed by Motorola in the late 1980s which included an integrated circuit which allowed it to be integrated with engine management and accident-avoidance systems.

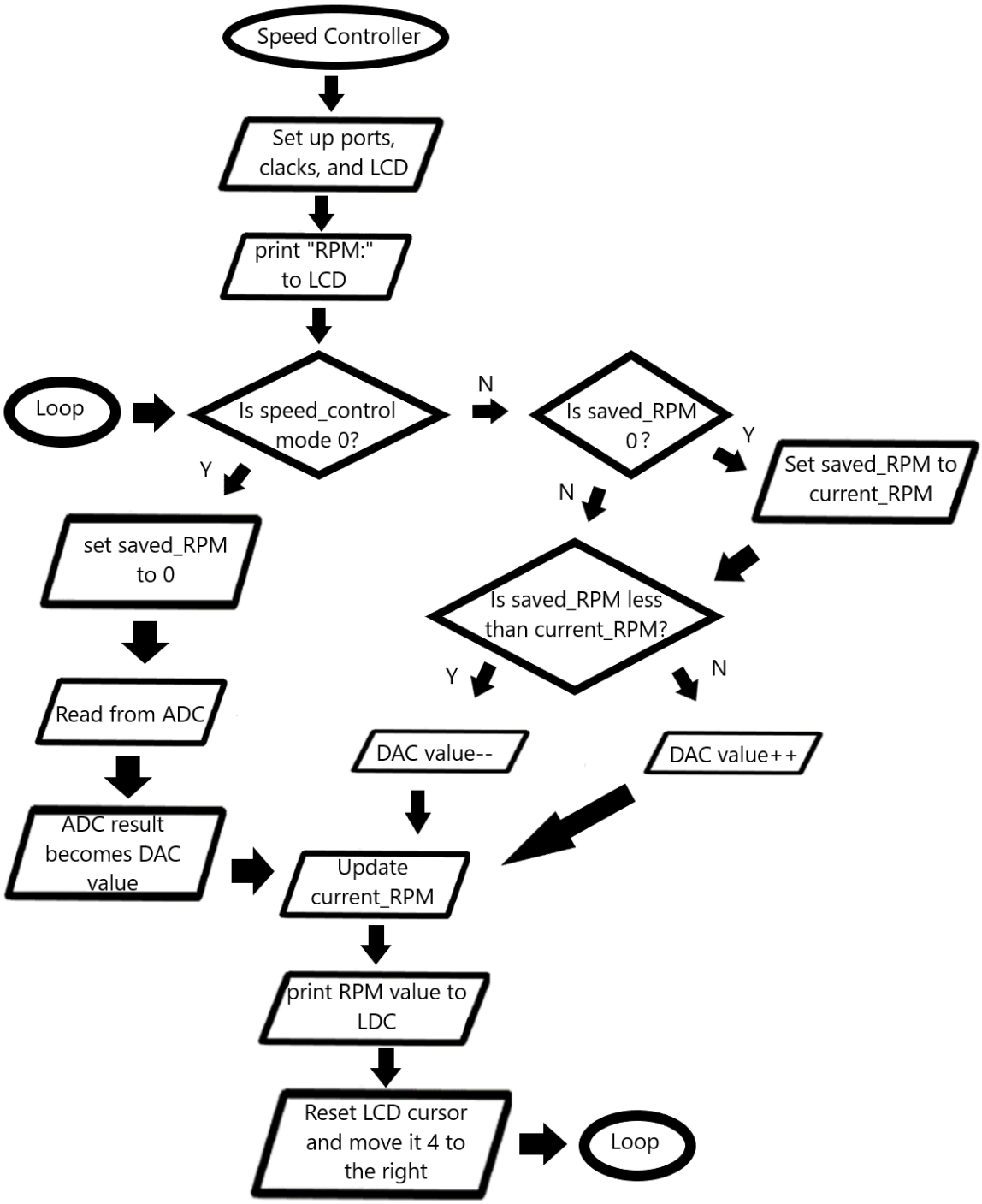
Design

Hardware (Block Diagrams)



For the speed controller circuit, port PA4 was set to analog output mode for the DAC and was connected to the DC motor. The other wire of the DC motor was grounded. Port PA7 was set to analog input mode for the ADC and was connected to a potentiometer with a range of 3.3V to 0V. The mode of the speed controller was controlled by port PB0 which was set to input mode with pull up resistor. Port PB0 was connected to a switch which was grounded. Port PC4 to PC7 were set to output mode and then each connected to the LCD data pins from D4 to D7 accordingly. The cathode and anode of the display are connected to ground and 5 V accordingly. Port PA0 is set to register select(RS), PA1 to the enable(E), and the read/write(R/W) pin was grounded. Port PA6 was connected to a light diode/LED pair sensor which outputted logic 1 when there is nothing between them. The Vcc of the light diode/LED pair sensor is connected to 3.3V and it's GND pin is grounded.

Software (Flow Chart)

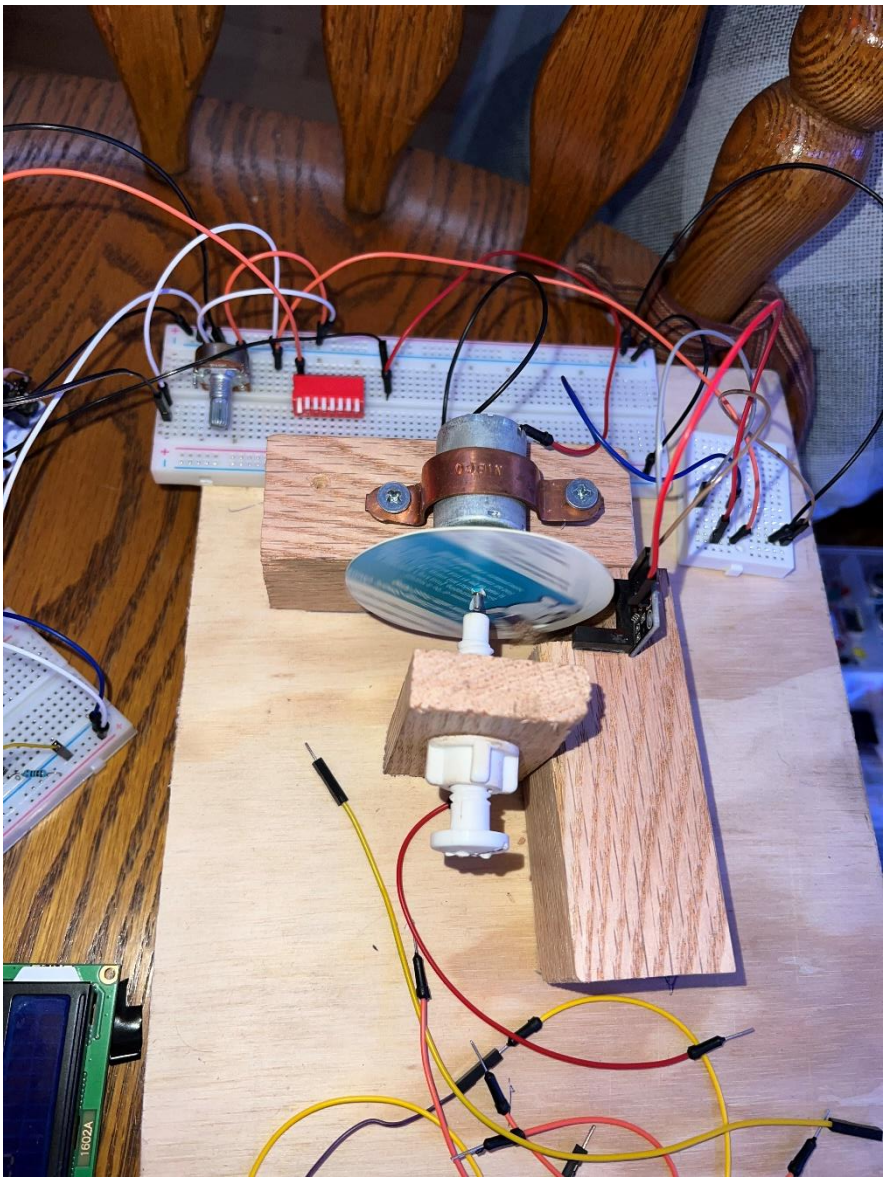


In the speed controller program, the code will first set up the ports, clocks, and LCD. Then it will print “RPM:” to the LCD and check if the mode is 0 or 1. If the mode is 0 then the program will do the following: set the saved_RPM variable to 0, read from the ADC register, and then set DAC’s input with the value read from the ADC. If the mode is 1 then the speed controller is turned on and the

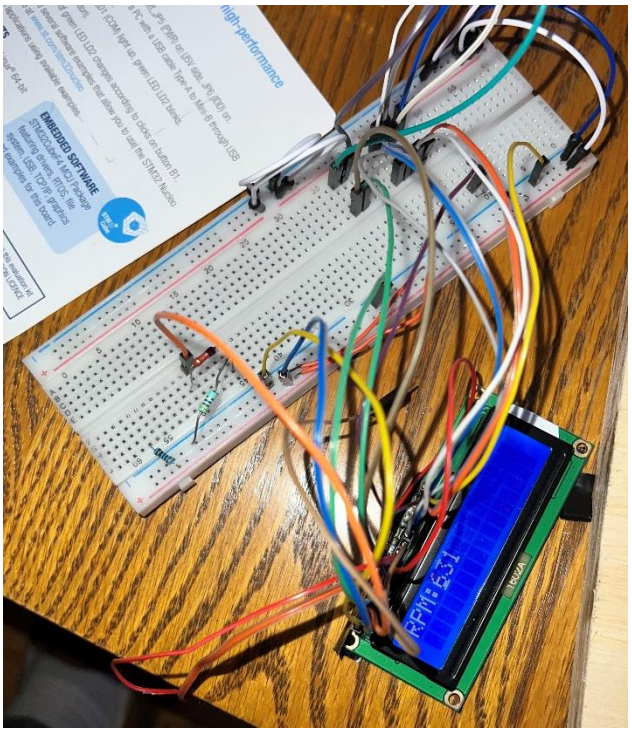
program will first set the saved_RPM variable to current_RPM's value if the saved_RPM value is 0. Second, the program will increase the DAC output if the saved_RPM's value is greater than the current_RPM's value, if it is less than the DAC output will decrease. Both modes will then converge to update the current_RPM variable, print the current_RPM value to the LCD, and then set the LCD cursor 4 to the right from its home location. The program will then loop back to the mode check if statement.

Verification

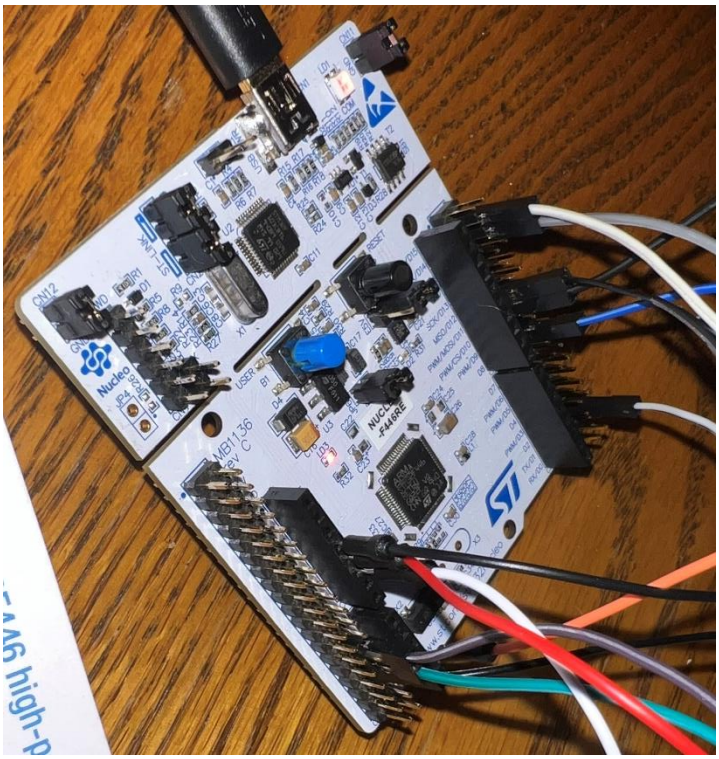
Nucleo and breadboard circuit



Breadboard wiring (motor, potentiometer, and mode switch)



Breadboard wiring (LCD set up)



NUCLEO circuit board

Summary and conclusion

In Speed Controller, there are 2 modes that can be chosen including speed control enabled and speed control disabled. These modes are controlled by switch PB0. When speed control is disabled, the microcontroller will read from a potentiometer (PA7) to determine the DC motor output (PA4). The microcontroller reads the analog value through an ADC which functions by starting a conversion and waiting for the conversion to complete (SR register's 2^{nd} bit becomes logic 1); then the DR register will hold the value of the analog input. The value of the ADC is then outputted to the DC motor by taking the value of the DR register and putting it in the DHR12R1 register of the DAC. The output range of the DAC is set so the potentiometer cannot make the DC motor spin at max speed or stop; this gives the microcontroller' DAC additional range to correct the RPM of the motor. At the start of this mode the saved_RPM variable will be cleared; this is used when the speed control is enabled to indicate the saved_RPM needs to be updated. This mode does not maintain the RPM, the speed of the motor is only determined by the potentiometer.

When the speed control is enabled, the potentiometer is no longer used, and the microcontroller will maintain the RPM of the motor found when the mode was enabled. In this mode, the saved_RPM variable will be set as the current_RPM if it was initially 0. Then the program will increase the DAC output by 1 if the saved_RPM's value is greater than the current_RPM's value, if it is less than the DAC output will decrease by 1. This method is called bang bang which will slowly increase or decrease the output till it gets to its target value (RPM), but after it reached the value, the output will oscillate slightly at the target value.

Then both modes will update the current_RPM value. The RPM of the motor is read through a light diode/LED pair sensor which will output a logic 1 when there is no obstruction between them. A disk with a hole at the edge is attached to the motor and the light diode and LED are placed on opposite sides of the disk. This means when the sensor reads a logic 1 its 1 revolution. Using input capture, the frequency of the sensor input (PA6) was found and multiplied by 60 to give the current_RPM. Input capture works by setting TIM3 to a frequency of 1 Hz and comparing it to PA6 to get its frequency. Then the current_RPM is printed to the LCD in front of "RPM:" which was printed in the beginning of the program. This is done by moving the LCD cursor 4 to the right of home after every LCD print. However, the LCD is only printed to every 20 cycles with no delay afterwards to prevent errors with the input capture. To simulate the wheel going up a hill, a screw was placed in front of the motor which applies friction when screwed in.

This project introduced coding with analog inputs and outputs in C programing along with the input capture feature of a microcontroller and wiring a high number of desired input and output ports of the STM32F446RE circuit board to different sensors. This was done by reasserting the microcontrollers datasheet to learn how to use the different registers of the STM32F446RE circuit board and by reading the textbook to learn their proper application. Due to the lessons above this project has prepared students to begin designing, coding, and wiring more complex projects with analog and the skills used.

Work cited

[1] M. A. Mazidi, S. Chen, and E. Ghaemi, STM32 arm programming for embedded systems using C Language with STM32 Nucleo. Place of publication not identified: Mazidi, 2018.

[2] M. A. Mazidi, S. Naimi, S. Naimi, and S. Chen, ARM assembly language programming & architecture. Middletown, DE?: www.MicroDigitalEd.com, 2016. 2nd Edition

Appendix (Programs)

```
#include "stm32f4xx.h"

#define RS 0x01     /* PA0 mask for LCD reg select */
#define EN 0x02     /* PA1 mask for LCD enable */

void delayMs(int n);

void LCD_print(char data[]);
void LCD_nibble_write(char data, unsigned char control);
void LCD_command(unsigned char command);
void LCD_data(char data);
void LCD_init(void);
void PORTS_init(void);

int static current_RPM = 0;
int static saved_RPM = 0;
int period;
float frequency;

int main(void) {
```

```
		int speed_control = 0; //o=off, 1=on
		uint32_t output = 0;
		uint32_t max = 0xFFF;
		char A[8];
		double power = 0;
		int count = 0;

int last = 0;
int current;

		LCD_init();

		//set up clock

		LCD_print("5RPM:");

		while(1)
		{
			speed_control = ((GPIOB -> IDR)     & 0x1);  //get mode

			if(speed_control == 0)
			{
				saved_RPM = 0;

				ADC1->CR2 |= 0x40000000;        // start a conversion
				while(!(ADC1->SR & 2)) {}       // wait for conv complete
				output = ADC1->DR;              // read conversion result
				power = 3800*((double)output/(double)max);
				if(power < 800) power = 800;
				DAC->DHR12R1 = (uint32_t)power;
			}
			else
			{
				if(saved_RPM == 0) saved_RPM = current_RPM;

				if(saved_RPM < current_RPM) power = power-1;
				if(saved_RPM > current_RPM) power = power+1;
				if(power > 4095) power = 4095;
				if(power < 0) power = 0;
				DAC->DHR12R1 = (uint32_t)power;
			}

			////////////////////////////////////////////////////////////////////////////// get current_RPM
				while (!(TIM3->SR & 2)) {}  /* wait until input edge is captured */
   current = TIM3->CCR1;       /* read captured counter value */
   period = current - last;    /* calculate the period */
```

```
        last = current;
        frequency = 1000.0f / period;
        last = current;

                                if(count == 20)
                                        {
                                        count = 0;
                                        current_RPM = frequency*60;

                                        sprintf (A,"5%d ", current_RPM);
                                        LCD_print(A);
                                        //delayMs(500);
                                        LCD_command(0x02);
                                        LCD_command(0x14);
                                        LCD_command(0x14);
                                        LCD_command(0x14);
                                        LCD_command(0x14);
                                        }
                                        count++;
                        }
}

/* initialize ports then initialize LCD controller */
void LCD_init(void)
{
   PORTS_init();

   delayMs(20);              /* LCD controller reset sequence */
   LCD_nibble_write(0x30, 0);
   delayMs(5);
   LCD_nibble_write(0x30, 0);
   delayMs(1);
   LCD_nibble_write(0x30, 0);
   delayMs(1);

   LCD_nibble_write(0x20, 0);  /* use 4-bit data mode */
   delayMs(1);
   LCD_command(0x28);         /* set 4-bit data, 2-line, 5x7 font */
   LCD_command(0x06);         /* move cursor right */
   LCD_command(0x01);         /* clear screen, move cursor to home */
   LCD_command(0x0F);         /* turn on display, cursor blinking */
}
void PORTS_init(void)
{
   RCC->AHB1ENR =  7;            /* enable GPIOA, GPIOB and GPIOC clock */
```

```
                GPIOC->MODER &= ~0x0000FF00; /* clear pin mode */
                GPIOA->MODER &= ~0x00000F0F; /* clear pin mode */

                GPIOC->MODER =  0x00005500;   /* set pin output mode for LCD PC4-PC7 for
D4-D7*/
                GPIOB->PUPDR =  0x00000001;   /* INPUT: PB0 - mode */
                GPIOA->MODER =  0x0000C305;   /* PA0-RS, PA1-En for LCD | PA4,7
analog*/
                GPIOA->BSRR  =  0x00020000;   /* turn off EN */

                /* setup DAC */
                RCC->APB1ENR |= 1 << 29;
                DAC->CR |= 1;

   /* setup ADC1 */
   RCC->APB2ENR |= 0x00000100;     /* enable ADC1 clock */
   ADC1->CR2 = 0;               /* SW trigger */
   ADC1->SQR3 = 7;               /* conversion sequence starts at ch 7 */
   ADC1->SQR1 = 0;               /* conversion sequence length 1 */
   ADC1->CR2 |= 1;              /* enable ADC1 */
                //---------------------------------------------------------------------------

                        // configure PA6 as input of TIM3 CH1
                        RCC->AHB1ENR |=  1;           /* enable GPIOA clock */
                        GPIOA->MODER &= ~0x00003000;    /* clear pin mode */
                        GPIOA->MODER |=  0x00002000;    /* set pin to alternate function */
                        GPIOA->AFR[0] &= ~0x0F000000;   /* clear pin AF bits */
                        GPIOA->AFR[0] |= 0x02000000;    /* set pin to AF2 for TIM3 CH1 */

                        // configure TIM3 to do input capture with prescaler ...
                        RCC->APB1ENR |= 2;            /* enable TIM3 clock */
                        TIM3->PSC = 16000 - 1;        /* divided by 16000 */
                        TIM3->CCMR1 = 0x41;           /* set CH1 to capture at every edge */
                        TIM3->CCER = 0x0B;            /* enable CH 1 capture both edges */
                        TIM3->CR1 = 1;               /* enable TIM3 */
}

void LCD_nibble_write(char data, unsigned char control)
{
   /* populate data bits */
   GPIOC->BSRR = 0x00F00000;      /* clear data bits */
   GPIOC->BSRR = data & 0xF0;     /* set data bits */

   /* set R/S bit */
   if (control & RS)
      GPIOA->BSRR = RS;
```

```
    else
        GPIOA->BSRR = RS << 16;

    /* pulse E */
    GPIOA->BSRR = EN;
    delayMs(0);
    GPIOA->BSRR = EN << 16;
}

void LCD_command(unsigned char command)
{
    LCD_nibble_write(command & 0xF0, 0);    /* upper nibble first */
    LCD_nibble_write(command << 4, 0);      /* then lower nibble */

    if (command < 4)
        delayMs(2);         /* command 1 and 2 needs up to 1.64ms */
    else
        delayMs(1);         /* all others 40 us */
}

void LCD_data(char data)
{
    LCD_nibble_write(data & 0xF0, RS);      /* upper nibble first */
    LCD_nibble_write(data << 4, RS);        /* then lower nibble */

    delayMs(1);
}

void LCD_print(char data[])
{
        int size = data[0] - 0x30;                  //use first character as size
        for(int i=1;i<size;i++)
        {
                if(data[i] != 0)
                {
                        LCD_data(data[i]);              //print each character
                        delayMs(30);
                }else break;
        }
}


/* 16 MHz SYSCLK */
void delayMs(int n)
{
    int i;
```

```
    /* Configure SysTick */
    SysTick->LOAD = 16000;  /* reload with number of clocks per millisecond */
    SysTick->VAL = 0;       /* clear current value register */
    SysTick->CTRL = 0x5;    /* Enable the timer */

    for(i = 0; i < n; i++) {
       while((SysTick->CTRL & 0x10000) == 0) /* wait until the COUNTFLAG is set */
          { }
    }
    SysTick->CTRL = 0;      /* Stop the timer (Enable = 0) */
}
```