LAB REPORT: EMBEDDED COMPUTING

LAB II: STEPPER MOTOR CONTROL

Tailei WANG, Hao DENG

Electric Vehicle Propulsion and Control (E-PiCo)

ÉCOLE CENTRALE DE NANTES

January 2022

1 Objective

The objective of this lab is to adapt the timer frequency to control a stepper motor speed. The timer is used in synchronization mode.

2 Control of a stepper motor

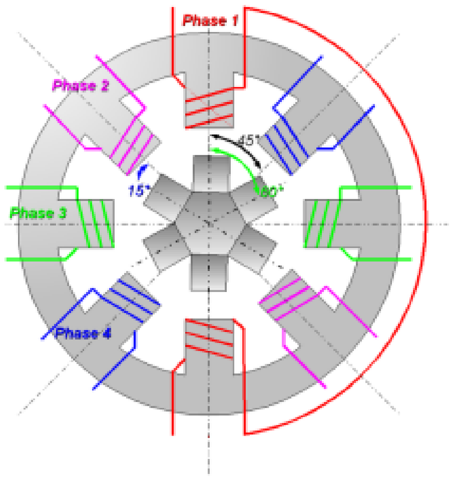

Figure 1: Operating principle of the stepper motor.

The ULN2003 chip is a buffer (Darlington transistor array) that is in charge of the power conversion.

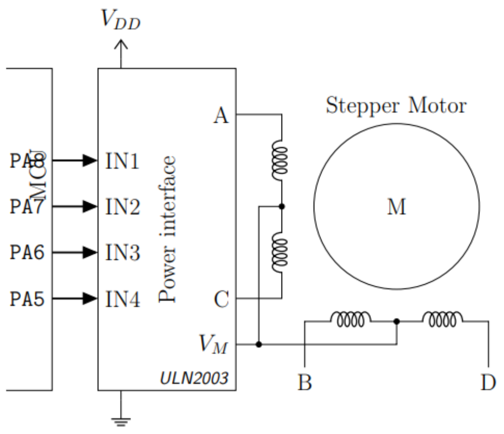

Figure 2: Electronic control interface.

3 Control of a stepper motor

To rotate the stepper motor counter-clockwise, we use a table that stores the entire sequence. The table is defined as follows:

```
const unsigned char seq []={8 ,0 xc ,4 ,6 ,2 ,3 ,1 ,9};
```

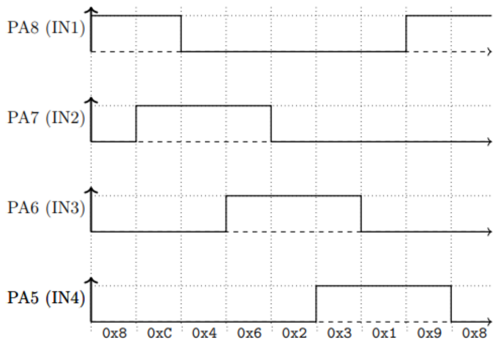

Figure 3: Time sequence for a counter-clockwise rotation.

Question 1 *Write the setup() function to configure outputs.*

The code below shows pin configuration by using the pinMode function.

```c
#include "stm32f3xx.h"
#include "pinAccess.h"
#include "adc.h"

void setup() {
    // OUTPUTs
    for(int i=5; i<9; i++) // PA5 ~ PA8
    {
        pinMode(GPIOA,i,OUTPUT);
    }
}
```

We use in the next two question a direct access to the GPIO, using ODR.

Question 2 *Write the stepCCW() function that updates outputs so that the motor make a single step each time the function is called. Note: a digitalWrite(...) is not appropriate, as the 4 outputs should be updated at the same time. Validate the code with a step by step execution.*

The code below shows the setting of the Output Date Register(ODR). The output is updated everytime the StepCCW function is called.

```c
#include "stm32f3xx.h"
#include "pinAccess.h"
#include "adc.h"

const unsigned char seq[] = {0x8, 0xc, 0x4, 0x6, 0x2, 0x3, 0x1, 0x9};

// Counter-Clockwise Rotation using ODR.
// the 4 outputs are updated at the same time.
void StepCCW(){
    static int num = 0;
    GPIOA->ODR = seq[num] << 5; // we use PA5 ~ PA8 in this lab.
    num ++;
    num = num % 8; // num goes up from 0 to 7 in a cycle.
}
```

Question 3 *Use a timer so that the stepCCW() function is called @10Hz. The synchronization phase should not use a blocking function (active wait for a flag). There prefered way is a function that just look at the timer status (overflow).*

Since the StepCCW function is called at 10Hz, this means that the function is called every 100ms. Therefore, with the internal clock frequency 64 MHz, the prescarler register is set at 64000-1 so the timer have 1ms each tick. Then the ARR register is set at 100-1, meaning that the timer counts 100 ticks so the period is 100ms. The code below give the configuration of the timer and the main function.

```c
void setup() {
    // OUTPUTs
    for(int i=5; i<9; i++) // PA5 ~ PA8
    {
        pinMode(GPIOA,i,OUTPUT);
    }
    // TIM6 - input clock = 64MHz.
    RCC->APB1ENR |= RCC_APB1ENR_TIM6EN;
    __asm("nop");
    // reset peripheral (mandatory!)
    RCC->APB1RSTR |= RCC_APB1RSTR_TIM6RST;
    RCC->APB1RSTR &= ~RCC_APB1RSTR_TIM6RST;
    __asm("nop");

    TIM6->PSC = 64000 - 1;  // the input frequency for the counter (1kHz)
    TIM6->CR1 |= TIM_CR1_CEN;  // enable counter
    TIM6->ARR = 100 -1; // stepCCW() is called @10Hz.
    // setup Button D6/PB1 (see Question 6)
    pinMode(GPIOB, 1, INPUT);
    // setup ADC (see Question 7)
    ADCInit();
}

int main(void) {
    setup();
    // Infinite loop
    while (1) {
        TIM6->SR &= ~TIM_SR_UIF; // reinit overflow flag
        StepCCW(); // established in Q2
        while(! (TIM6->SR & TIM_SR_UIF)); // synchro
    }
}
```

Question 4 *update the code to use the BSRR register of the GPIO. Validate the new solution.*

According to the lecture slides, BSRR register allows to update a single bit.The 0-15 bits of this register is used to bit setting while the 15-31 bits of the register are used to bit resetting. The code below indicates the usage of the BSRR register.

```c
// Counter-Clockwise Rotation using BSRR. BSRR access is atomic.
// the 4 outputs are updated at the same time.
void StepCCW(){
    static int num = 0;
    // BS (Bit Set)
    const unsigned int mask = (seq[num] << 5); // we use PA5 ~ PA8 in this lab.
    // BR (Bit Resit)
    PORTB->BSRR = mask | (~mask & 0xFF) << 16; // BSRR has 32 bits.
    num ++;
    num = num % 8; // num goes up from 0 to 7 in a cycle.
}
```

4 Control of a stepper motor

Question 5 *Write the stepCW() function that works as stepCCW, but in the other direction. The function uses the BSRR GPIO register directly.*

```
// Clockwise Rotation using BSRR.
void StepCW(){
    static int num = 0;
    // BS (Bit Set)
    const unsigned int mask = (seq[num] << 5); // we use PA5 ~ PA8 in this lab.
    // BR (Bit Resit)
    PORTB->BSRR = mask | (~mask & 0xFF) << 16; // BSRR has 32 bits.
    num = num + 7;
    num = num % 8; // num goes down from 7 to 0 in a cycle.
}
```

We now want to mix the two directions.

Question 6 *The stepper motor should turn clockwise when the button (D6) is pressed,and in the other direction when it is released.*

```
void Step(){
    static int num = 0;
    // BS (Bit Set)
    const unsigned int mask = (seq[num] << 5); // we use PA5 ~ PA8 in this lab.
    // BR (Bit Resit)
    PORTB->BSRR = mask | (~mask & 0xFF) << 16; // BSRR has 32 bits.
    // read from D6/PB1.
    int button = digitalRead(GPIOB, 1);

    if(button)
        num = (num+1) % 8; // Counter Clockwise
    else
        num = (num+7) % 8; // Clockwise
}
```

5 Rotation Speed

The rotation speed is directly associated to the frequency of the timer. We use here the potentiometer that is associated to the ADC (ADC1, channel 4 on pin PA3). To use the ADC, a set of files adc.c/h are provided.

Question 7 *Update the timer configuration so that the stepper functions are called from 10Hz (potentiometer set to 0) to 500Hz (potentiometer set to max).*

The potentiometer now acts as a control input to adjust the calling frequency of the StepCCW function. Thus, to manufacture this feature, the prescaler frequency remains and each tick is 1ms. We change the calling frequency by tuning the ARR register value. And the ARR register value is related to the reading of the potentiometer. The corresponding code is given below.

```c
int main(void) {
    setup();
    // Infinite loop
    while (1) {
        TIM6->SR &= ~TIM_SR_UIF; // reinit overflow flag
        Step(); // established in Q6
        int reading = ADCRead();
        TIM6->ARR = (2-100) *reading /4096 + 100 - 1;
        while(! (TIM6->SR & TIM_SR_UIF)); // synchro
    }
}
```

Question 8 *Test your solution: the stepper motor should change its direction of rotation each time button (D6) is pressed (Finite State Machine associated to the button state).*

In order to achieve the feature, a finite state machine is used here. There are four states in the finite state machine: RELEASED, PUSHING,PUSHED,RELEASED.If the button is pushed, the input register is connected to the ground and thus the corresponding bit value will be set to 0. And it enters the state machine with the state becomes pushing. Everytime the button D6 is pressed, it enters the state machine and the direction is toggled by using a steptoggle function.

```c
// FSM associated to the button state.
enum PBState {RELEASED, PUSHING, PUSHED, RELEASING};
enum PBState managePB1(){
    static enum PBState state = RELEASED;
    switch(state) {
        case RELEASED: if((GPIOB->IDR & 0x2) == 0) state = PUSHING; break;
        case PUSHING: state = PUSHED; break;
        case PUSHED: if(GPIOB->IDR & 0x2) state = RELEASING; break;
        case RELEASING: state = RELEASED; break;
    }
    return state;
}
// the stepper motor changes its direction of rotation each time button (D6) is pressed.
void Step_toggle(enum PBState state){
    static int num = 0;
    // BS (Bit Set)
    const unsigned int mask = (seq[num] << 5); // we use PA5 ~ PA8 in this lab.
    // BR (Bit Resit)
    PORTB->BSRR = mask | (~mask & 0xFF) << 16; // BSRR has 32 bits.
    // set a flag of clock-wise.
    static _Bool CW = 1;
    if(state = PUSHING)
        CW = !CW;

    if(!CW)
```

```c
            num = (num+1) % 8; // CCW
        else
            num = (num+7) % 8; // CW
}
// main function
int main(void){
    setup();
    enum PBState state;
    // Infinite loop
    while (1) {
        TIM6->SR &= ~TIM_SR_UIF; // reinit overflow flag
        state = managePB1();
        Step_toggle(state);
        int reading = ADCRead();
        TIM6->ARR = (2-100) *reading /4096 + 100 - 1;
        while(! (TIM6->SR & TIM_SR_UIF)); // synchro
    }
}
```

6 Just a round

There are 64 steps for each round, but there is also a reduction factor of 64. We now want to do just one round. Each time the button D6 is pushed, the stepper performs one round, toggling the direction of rotation.

Question 9 *Update your application to add this constraint.*

In this case, we apply an interrupt to the Timer2. The configuration of Timer2 and interrupt handler are shown below.

```c
void servoInit(){
    // Step1: Pin configuration
    pinAlt(GPIOA, 3, 1); // AF1 for PA3
    pinMode(GPIOB, 1, INPUT); // setup Button D6/PB1
    // Step2: Timer2 configuration
    RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;
    __asm("nop");
    RCC->APB1RSTR |= RCC_APB1RSTR_TIM2RST;
    RCC->APB1RSTR &= ~RCC_APB1RSTR_TIM2RST;
    __asm("nop");

    TIM2->PSC = 64 - 1; //(1MHz)
    TIM2->ARR = 20000 -1; //(period: 20ms)
    // Step3: PWM configuration
    TIM2->CCMR2 &= ~TIM_CCMR2_CC4S_Msk; // channel 4 as output
    TIM2->CCMR2 &= ~TIM_CCMR2_OC4M_Msk;
    TIM2->CCMR2 |= 6 << TIM_CCMR2_OC4M_Pos; // output PWM mode 1
    TIM2->CCMR2 |= TIM_CCMR2_OC4PE; // pre-load register TIM2_CCR4;

    TIM2->CR1 &= ~TIM_CR1_CMS_Msk; // mode 1 // edge aligned mode
    TIM2->CCER |= TIM_CCER_CC4E; // enable
    TIM2->CR1 |= TIM_CR1_CEN; // config reg: enable

    TIM2->CCR4 = 1500 - 1; // 90 degrees
}

void setup(){
    servoInit();
    // init scanning timer
    RCC->APB1ENR |= RCC_APB1ENR_TIM6EN;
    __asm("nop");
    RCC->APB1RSTR |= RCC_APB1RSTR_TIM6RST;
    RCC->APB1RSTR &= ~RCC_APB1RSTR_TIM6RST;

    TIM6->PSC = 64000 - 1;
    TIM6->CR1 |= TIM_CR1_CEN;
    TIM6->ARR = 5-1; // 5ms

    // Enable interrupt for TIM6
    TIM6->DIER |= TIM_DIER_UIE;
    NVIC_EnableIRQ(TIM6_DAC1_IRQn);
}

void servoSet(int setpoint){
    if(setpoint > 180)
        setpoint = 180;
    else if(setpoint < 0)
        setpoint = 0;
    TIM2->CCR4 = 1000*setpoint/180 + 1000 - 1;
}

void TIM6_DAC1_IRQHandler(){
    static int sense = 1;
    static int setpoint = 0;
    int button = digitalRead(GPIOB, 1);
    if(sense){
        setpoint++;
        if(setpoint > 180) setpoint = 180;
        if(setpoint == 180 && button == 0) sense=0;
    }
    else{
        setpoint--;
        if(setpoint < 0) setpoint = 0;
        if(setpoint == 0 && button == 0) sense=1;
    }
    servoSet(setpoint);
    //acknowledge
    TIM6->SR &= ~TIM_SR_UIF;
}

int main(void){
    setup();
    // Infinite loop
    while(1) {
    // empty
    }
}
```