MECHTRON 3TA4 LAB 5 Pre-Lab Report

Name: Qiushuang Li
Student#: 1206650

1) Angular Resolution = $\frac{360°}{\#\ of\ steps}$
   Number of steps = 48
   Angular Resolution = $\frac{360°}{48}$ = 7.5°
2) The mount of time = 50 seconds
3) Half/Full-Stepping Sequence:
   a.) Half-Stepping Sequence:
   Half-step would have 2x48 = 96 steps, according to 48 steps for Full-Stepping
   Time period between two steps = $\frac{50\ seconds}{96}$ = 0.5208 seconds
   b.) Full-Stepping Sequence:
   Time period between two steps = $\frac{50\ seconds}{48}$ = 1.0417 seconds
4) Output Compare Register Value for:
   3a): $\frac{180{,}000{,}000\text{Hz}}{1800-1} \times \frac{50}{96} - 1$ = 52122.28 times, set prescaler = 1800 $-1 = 1799$
   3b): $\frac{180{,}000{,}000\text{Hz}}{3600\text{-}1} \times \frac{50}{48} - 1$ = 52122.28 times, set prescaler = 3600 $-1 = 3599$
5)

```
#include "main.h"

typedef enum {HALF, FULL} STEPS;
typedef enum {FORWARD, BACKWARD} DIRECT;
STEPS StepSize = HALF;
DIRECT Direction = FORWARD;

uint8_t SwitchCount = 0;

void TIM3Config(STEPS);
void GPIOStepConfig(void);
void GPIOPBConfig(void);


int main(void) {
TIM3Config(StepSize);
GPIOStepConfig();
GPIOPBConfig();
STM_EVAL_PBInit(BUTTON_USER, BUTTON_MODE_EXTI);

while(1) {
if (StepSize == FULL) {
switch(SwitchCount) {
case 0:
if (Direction == FORWARD) {
GPIO_SetBits(GPIOD, GPIO_Pin_3);
```

```
GPIO_ResetBits(GPIOD, GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_4);
}
else {
GPIO_SetBits(GPIOD, GPIO_Pin_1);
GPIO_ResetBits(GPIOD, GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_2);
}
break;
case 1:
if (Direction == FORWARD) {
GPIO_SetBits(GPIOD, GPIO_Pin_2);
GPIO_ResetBits(GPIOD, GPIO_Pin_1 | GPIO_Pin_3 | GPIO_Pin_4);
}
else {
GPIO_SetBits(GPIOD, GPIO_Pin_4);
GPIO_ResetBits(GPIOD, GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3);
}
break;
case 2:
if (Direction == FORWARD) {
GPIO_SetBits(GPIOD, GPIO_Pin_4);
GPIO_ResetBits(GPIOD, GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3);
}
else {
GPIO_SetBits(GPIOD, GPIO_Pin_2);
GPIO_ResetBits(GPIOD, GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_1);
}
break;
case 3:
if (Direction == FORWARD) {
GPIO_SetBits(GPIOD, GPIO_Pin_1);
GPIO_ResetBits(GPIOD, GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_4);
}
else {
GPIO_SetBits(GPIOD, GPIO_Pin_3);
GPIO_ResetBits(GPIOD, GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_4);
}
break;
default:
break;
}
}
else {
switch(SwitchCount) {
case 0:
if (Direction == FORWARD) {
GPIO_SetBits(GPIOD, GPIO_Pin_3);
GPIO_ResetBits(GPIOD, GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_4);
}
else {
```

```
GPIO_SetBits(GPIOD, GPIO_Pin_1 | GPIO_Pin_3);
GPIO_ResetBits(GPIOD, GPIO_Pin_4 | GPIO_Pin_2);
}
break;
case 1:
if (Direction == FORWARD) {
GPIO_SetBits(GPIOD, GPIO_Pin_2 | GPIO_Pin_3);
GPIO_ResetBits(GPIOD, GPIO_Pin_1 |  GPIO_Pin_4);
}
else {
GPIO_SetBits(GPIOD, GPIO_Pin_1);
GPIO_ResetBits(GPIOD, GPIO_Pin_4 | GPIO_Pin_2 | GPIO_Pin_3);
}
break;
case 2:
if (Direction == FORWARD) {
GPIO_SetBits(GPIOD, GPIO_Pin_2);
GPIO_ResetBits(GPIOD, GPIO_Pin_1 | GPIO_Pin_4 | GPIO_Pin_3);
}
else {
GPIO_SetBits(GPIOD, GPIO_Pin_1 | GPIO_Pin_4);
GPIO_ResetBits(GPIOD, GPIO_Pin_3 | GPIO_Pin_2);
}
break;
case 3:
if (Direction == FORWARD) {
GPIO_SetBits(GPIOD, GPIO_Pin_2 | GPIO_Pin_4);
GPIO_ResetBits(GPIOD, GPIO_Pin_1 | GPIO_Pin_3);
}
else {
GPIO_SetBits(GPIOD, GPIO_Pin_4);
GPIO_ResetBits(GPIOD, GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3);
}
break;
case 4:
if (Direction == FORWARD) {
GPIO_SetBits(GPIOD, GPIO_Pin_4);
GPIO_ResetBits(GPIOD, GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3);
}
else {
GPIO_SetBits(GPIOD, GPIO_Pin_2 | GPIO_Pin_4);
GPIO_ResetBits(GPIOD, GPIO_Pin_1 | GPIO_Pin_3);
}
break;
case 5:
if (Direction == FORWARD) {
GPIO_SetBits(GPIOD, GPIO_Pin_1 | GPIO_Pin_4);
GPIO_ResetBits(GPIOD, GPIO_Pin_2 | GPIO_Pin_3);
}
```

```
else {
GPIO_SetBits(GPIOD, GPIO_Pin_2);
GPIO_ResetBits(GPIOD, GPIO_Pin_1 | GPIO_Pin_3 | GPIO_Pin_4);
}
break;
case 6:
if (Direction == FORWARD) {
GPIO_SetBits(GPIOD, GPIO_Pin_1);
GPIO_ResetBits(GPIOD, GPIO_Pin_2 | GPIO_Pin_3  | GPIO_Pin_4);
}
else {
GPIO_SetBits(GPIOD, GPIO_Pin_2  | GPIO_Pin_3);
GPIO_ResetBits(GPIOD, GPIO_Pin_1 | GPIO_Pin_4);
}
break;
case 7:
if (Direction == FORWARD) {
GPIO_SetBits(GPIOD, GPIO_Pin_1 | GPIO_Pin_3);
GPIO_ResetBits(GPIOD, GPIO_Pin_2 | GPIO_Pin_4);
}
else {
GPIO_SetBits(GPIOD, GPIO_Pin_3);
GPIO_ResetBits(GPIOD, GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_4);
}
break;
default:
break;
}
}
}
}

void TIM3Config(STEPS Steps) {
TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
NVIC_InitTypeDef NVIC_InitStructure;
TIM_OCInitTypeDef TIM_OCInitStructure;

RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3,ENABLE);

NVIC_InitStructure.NVIC_IRQChannel=TIM3_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=0X00;
NVIC_InitStructure.NVIC_IRQChannelSubPriority=0x01;
NVIC_InitStructure.NVIC_IRQChannelCmd=ENABLE;

NVIC_Init(&NVIC_InitStructure);

TIM_TimeBaseStructure.TIM_Period=65535; //2^16 = 65535
TIM_TimeBaseStructure.TIM_ClockDivision=TIM_CKD_DIV1;  //Set the clock divider to
1, has no effect on timing
```

```
TIM_TimeBaseStructure.TIM_CounterMode=TIM_CounterMode_Up;// set the timer to
count up continuously from 0

switch(Steps) {
case HALF:
TIM_TimeBaseStructure.TIM_Prescaler=3600 - 1;
break;
case FULL:
TIM_TimeBaseStructure.TIM_Prescaler=1800 - 1;
break;
}

TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);


TIM_OCInitStructure.TIM_OCMode=TIM_OCMode_Timing;
TIM_OCInitStructure.TIM_OutputState=TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse=52122;
TIM_OCInitStructure.TIM_OCPolarity=TIM_OCPolarity_High;

TIM_OC1Init(TIM3, &TIM_OCInitStructure);

TIM_OC1PreloadConfig(TIM3, TIM_OCPreload_Disable);

TIM_ITConfig(TIM3, TIM_IT_CC1, ENABLE);

/* TIM3 set to 0 and enable counter */
TIM_SetCounter(TIM3, 0x0000);
TIM_Cmd(TIM3, ENABLE);
}

void GPIOStepConfig(void) {
GPIO_InitTypeDef GPIOInitStructure;

RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);

GPIOInitStructure.GPIO_Pin = GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_4 ;
GPIOInitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIOInitStructure.GPIO_Speed = GPIO_Speed_100MHz ;
GPIOInitStructure.GPIO_OType = GPIO_OType_PP ;
GPIOInitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(GPIOD, &GPIOInitStructure);
}

void GPIOPBConfig(void) {
/**  Create Init Structures  **/
GPIO_InitTypeDef GPIO_InitStructure;
EXTI_InitTypeDef EXTI_InitStructure;
NVIC_InitTypeDef NVIC_InitStructure;
```

```
/**  Clock Enables  **/
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOE, ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);

/**  GPIO Initialization  **/
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_Init(GPIOE, &GPIO_InitStructure);
SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOE, EXTI_PinSource2);

/**  EXTI External Interrupt Handler Initialization  **/
EXTI_InitStructure.EXTI_Line = EXTI_Line2;
EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
EXTI_InitStructure.EXTI_LineCmd = ENABLE;
EXTI_Init(&EXTI_InitStructure);

/**  NVIC Interupt Handler Initialization **/
NVIC_InitStructure.NVIC_IRQChannel = EXTI2_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x00;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x00;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
}

void TIM3_IRQHandler(void) {
if (TIM_GetITStatus(TIM3, TIM_IT_CC1) != RESET) {
TIM_ClearITPendingBit(TIM3, TIM_IT_CC1);

SwitchCount += 1;
if (StepSize == FULL)
SwitchCount %= 4;
else
SwitchCount %= 8;

TIM_SetCounter(TIM3, 0x0000);
}
}

void EXTI0_IRQHandler(void) {

if (Direction == FORWARD)
Direction = BACKWARD;
else
Direction = FORWARD;
EXTI_ClearITPendingBit(USER_BUTTON_EXTI_LINE);
}
```

```
void EXTI2_IRQHandler(void) {
if (EXTI_GetITStatus(EXTI_Line2) != RESET) {
SwitchCount = 0;
if (StepSize == FULL)
StepSize = HALF;
else
StepSize = FULL;
}
EXTI_ClearITPendingBit(EXTI_Line2);
}
```