UNIWERSYTET TECHNOLOGICZNO-PRZYRODNICZY
IM. J. I J. ŚNIADECKICH W BYDGOSZCZY

WYDZIAŁ TELEKOMUNIKACJI, INFORMATYKI I ELEKTROTECHNIKI
ZAKŁAD TECHNIKI CYFROWEJ

UKŁADY CYFROWE I MIKROPROCESORY

PLOTER RYSUJĄCY STEROWANY SILNIKAMI KROKOWYMI

AUTOR:
PIOTR RACHWAŁ
GRUPA I
TRYB STUDIÓW STACJONARNE

KIERUNEK:
INFORMATYKA STOSOWANA
SEMESTR III
ROK AKADEMICKI 2016/2017

PLOTER RYSUJĄCY – PROJEKT

Wykonanie:

Konstrukcja plotera zostanie wykonana z pleksi w taki sposób, aby umożliwić ruchy w osiach „x, y". Do napędu każdej z osi zostanie wykorzystany osobny silnik krokowy z przedłużonym wałem.

Ruch w osi „z" zostanie zablokowany, przedmiot kreślący będzie poruszał się bezpośrednio po kartce papieru bez możliwości podnoszenia i opuszczania.
Sterowanie silnikami odbędzie się za pomocą sterowników Low-Voltage DRV8834.

Rozruch silnika zostanie przeprowadzony z wykorzystaniem tzn. rampy.
(powolne rozpędzanie oraz hamowanie)

Projekt zostanie wykonany na mikrokontrolerze STM32 F072RB Nucleo.
Program realizujący w/w działanie zostanie napisany w środowisku
Workbench for STM32 / CooCox.

Komunikacja z mikrokontrolerem odbędzie się za pomocą portu USART.

Rysowanie:

Przed rozpoczęciem rysowania należy sprawdzić stan przewodów łączących STM32 z silnikami krokowymi. Po sprawdzeniu połączeń należy umieścić konstrukcję plotera na kartce papieru.

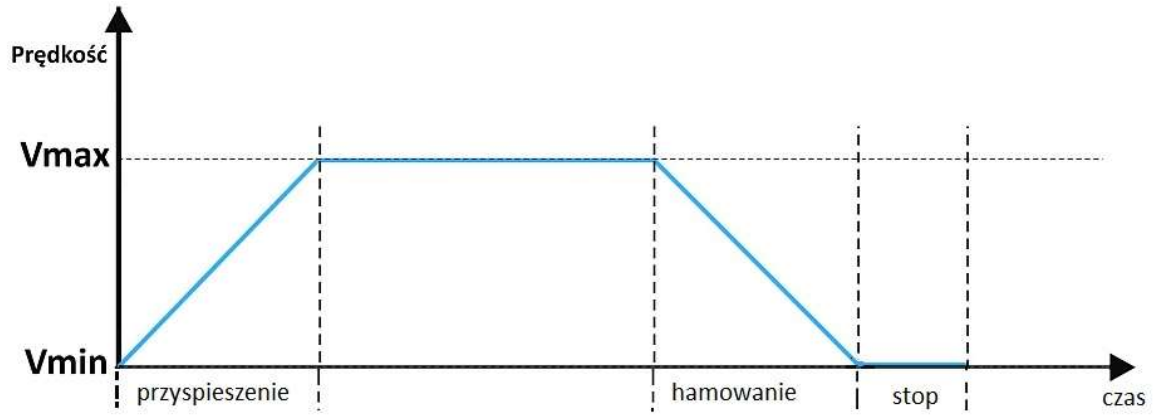
Następnie za pomocą konsoli: HypeTerminal odpowiednio połączyć się z układem.
Po udanym połączeniu należy wydawać odpowiednie polecenia, na które określone silniki krokowe będą reagowały. Np. wysłanie instrukcji: „ruchPrawoX" będzie skutkowało poruszeniem się pierwszego silnika w prawo(liczba wykonanych kroków zostanie statycznie ustalona). Nieprawidłowo wydane polecenia będą ignorowane.

PLOTER RYSUJĄCY – PROTOKÓŁ KOMUNIKACYJNY

Tabela poleceń sterujących:

#	Nazwa	Parametr	Opis
1.	setXSteps	int(step) – ilość kroków	Ustawienie ilości kroków do wykonania przez silnik - oś X.
2.	setXSpeedMin	int(speed) – minimalna prędkość silnika	Ustawienie prędkości minimalnej silnika - oś X.
3.	setXSpeedMax	int(speed) – maksymalna prędkość silnika	Ustawienie prędkości maksymalnej silnika - oś X.
4.	setXAccel	int(time) – czas rozpędzania	Ustawienie czasu rozpędzania. - oś X.
5.	setXDecel	int(time) – czas hamowania	Ustawienie czasu hamowania. - oś X.
6.	setYSteps	int(step) – ilość kroków	Ustawienie ilości kroków do wykonania przez silnik - oś Y.
7.	setYSpeedMin	int(speed) – minimalna prędkość silnika	Ustawienie prędkości minimalnej silnika - oś Y.
8.	setYSpeedMax	int(speed) – maksymalna prędkość silnika	Ustawienie prędkości maksymalnej silnika - oś Y.
9.	setYAccel	int(time) – czas rozpędzania	Ustawienie czasu rozpędzania. - oś Y.
10.	setYDecel	int(time) – czas hamowania	Ustawienie czasu hamowania. - oś Y.
11.	showCommands	brak	Wyświetlenie dostępnych komend.
12.	drawSquare	brak	Rysowanie figury – kwadrat.
13.	axisXmoveLeft	brak	Ruch silnika w osi „x” – kierunek lewo.
14.	axisXmoveRight	brak	Ruch silnika w osi „x” – kierunek prawo.
15.	axisYmoveLeft	brak	Ruch silnika w osi „y” – kierunek lewo.
16.	axisYmoveRight	brak	Ruch silnika w osi „y” – kierunek prawo.

Wykres prędkości obrotowej silnika w funkcji czasu:



Komunikacja z układem odbędzie się za pomocą portu USART z wykorzystaniem programu Hype!Terminal.

Wykrycie komendy nastąpi po pojawieniu się średnika.

W chwili załadowania programu podstawowe parametry silnika zostają ustawione na:
Steps = 100, accel = 3, decel = 3, vMin = 1, Vmax = 10.

W przypadku wysłania nieprawidłowej komendy sterującej program wypisze stosowny komunikat np. „ERROR: Wrong command.”.
W momencie wysłania polecenia zawierającego błędny parametr program wyświetli komunikat. np. „ERROR: Invalid decel time range <1:5>.”

Po weryfikacji polecenia zostanie wyświetlona aktualnie wykonywana przez urządzenie czynność. np. „Engine X running - left.”

Składnia komendy dla polecenia z parametrem: [nazwa]*[separator][parametr]

- nazwa – polecenia zawarte w w/w tabeli (wielkość liter ma znaczenie, wysyłane polecenia muszą być identyczne z tabelą)
- separator – spacja (oddzielenie nazwy polecenia od przyjmowanych parametrów)
- parametr – argument funkcji
- * – miejsce, umożliwiające zakończenie komendy (wyłącznie dla poleceń bez parametru)

Wykonanie:

Deklaracja najważniejszych zmiennych wykorzystywanych w projekcie:

```
char odbieranie[128];
int od_poczatek = 0;
int od_wolny = 0;

char wysylanie[512];
int wy_poczatek = 0;
int wy_wolny = 0;

volatile int times_X_ms = 0;
volatile int stepsX = 100;
float vXMin = 1.0;
float vXMax = 10.0;

int accelXTime = 3;
int decelXTime = 3;

double accelerationX = 0;
double decelerationX = 0;

int stepsXAccel = 0;
int stepsXDecel = 0;
int middleXSteps = 0;

volatile float currentXSpeed = 1.0;
volatile int stepAxisX = 1;
volatile int axisXEnabled = OFF;
```

Definicja stałych używanych w projekcie:

```
#ifndef DEFINE_H_
#define DEFINE_H_

#define axisXpin GPIOC
#define axisXstep GPIO_Pin_1
#define axisXdir GPIO_Pin_0

#define axisYpin GPIOC
#define axisYstep GPIO_Pin_2
#define axisYdir GPIO_Pin_3

#define LEFT 0
#define RIGHT 1

#define OFF 0
#define ON 1

#define minSteps 10
#define maxSteps 230

#define minAccelTime 1
#define maxAccelTime 5

#define minDecelTime 1
#define maxDecelTime 5

#define speedMin 1
#define speedMax 10
#endif
```

W w/w pliku zostały zdefiniowane stałe, aby uprościć możliwość zmiany określonej wartości.

Inicjalizacja USART:

```
void USART2_Init(void) {
    USART_InitTypeDef usart;
    GPIO_InitTypeDef gpio;
    NVIC_InitTypeDef nvic;

    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA, ENABLE);

    GPIO_PinAFConfig(GPIOA, GPIO_PinSource2, GPIO_AF_1);
    GPIO_PinAFConfig(GPIOA, GPIO_PinSource3, GPIO_AF_1);

    gpio.GPIO_Pin = GPIO_Pin_5 | GPIO_Pin_10;
    gpio.GPIO_OType = GPIO_OType_PP;
    gpio.GPIO_Mode = GPIO_Mode_OUT;
    gpio.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &gpio);

    gpio.GPIO_Pin = GPIO_Pin_2;
    gpio.GPIO_Speed = GPIO_Speed_50MHz;
    gpio.GPIO_Mode = GPIO_Mode_AF;
    gpio.GPIO_OType = GPIO_OType_PP;
    gpio.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_Init(GPIOA, &gpio);

    gpio.GPIO_Pin = GPIO_Pin_3;
    gpio.GPIO_Speed = GPIO_Speed_50MHz;
    gpio.GPIO_Mode = GPIO_Mode_AF;
    gpio.GPIO_OType = GPIO_OType_PP;
    gpio.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_Init(GPIOA, &gpio);

    nvic.NVIC_IRQChannel = USART2_IRQn;
    nvic.NVIC_IRQChannelPriority = 0;
    nvic.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&nvic);

    usart.USART_BaudRate = 9600;
    usart.USART_WordLength = USART_WordLength_8b;
    usart.USART_StopBits = USART_StopBits_1;
    usart.USART_Parity = USART_Parity_No;
    usart.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
    usart.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
    USART_Init(USART2, &usart);

    USART_Cmd(USART2, ENABLE);
    USART_ITConfig(USART2, USART_IT_RXNE, ENABLE);

}
```

Inicjalizacja PINÓW odpowiedzialnych za prędkość oraz kierunek pracy silnika:

```
void Pin_AxisXConfig() {
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOC, ENABLE);
    GPIO_InitTypeDef gpio;
    gpio.GPIO_Pin = axisXstep | axisXdir;
    gpio.GPIO_Mode = GPIO_Mode_OUT;
    gpio.GPIO_OType = GPIO_OType_PP;
    gpio.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(axisXpin, &gpio);
}
```

Inicjalizacja kontrolera przerwań NVIC:

```
void Nvic_Timer2_Config() {
    NVIC_InitTypeDef nvic;
    nvic.NVIC_IRQChannel = TIM2_IRQn;
    nvic.NVIC_IRQChannelPriority = 0;
    nvic.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&nvic);
}
```

Inicjalizacja TIMERA generującego impulsy o różnej częstotliwości:

```
void Timer2_Config() {
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
    TIM_TimeBaseInitTypeDef tim;
    TIM_TimeBaseStructInit(&tim);
    tim.TIM_CounterMode = TIM_CounterMode_Up;
    tim.TIM_Prescaler = 48 - 1;
    tim.TIM_Period = 1000 - 1;
    TIM_TimeBaseInit(TIM2, &tim);
    TIM_Cmd(TIM2, ENABLE);
}
```

W/w funkcje wywoływane są w głównej metodzie main.

```
int main(void) {
    USART2_Init();
    Pin_AxisXConfig();
    Pin_Config();
    Nvic_Timer2_Config();
    Timer2_Config();
    basicXSettings();
    SysTick_Config(SystemCoreClock / 1000);

    while (1) {
        usartGetValue();
    }
}
```

Przerwanie USART:

```
void USART2_IRQHandler(void) {
    if (USART_GetITStatus(USART2, USART_IT_RXNE) != RESET) {
        odbieranie[od_wolny] = USART_ReceiveData(USART2);
        od_wolny++;
        if (od_wolny >= 128)
            od_wolny = 0;
    }

    if (USART_GetITStatus(USART2, USART_IT_TXE) != RESET) {
        if (wy_poczatek != wy_wolny) {
            USART_SendData(USART2, wysylanie[wy_poczatek]);
            wy_poczatek++;
            if (wy_poczatek >= 512)
                wy_poczatek = 0;
        } else
            USART_ITConfig(USART2, USART_IT_TXE, DISABLE);
    }
}
```

W chwil wykrycia przerwania od odbierania do bufora odbiorczego zostają dopisywane znaki pobierane z klawiatury. W momencie pojawienia się przerwania od wysyłania oraz gdy buforze wysyłającym znajdują się dane następuje wysłanie znaku i przejście do następnego indeksu. Po wysłaniu wszystkich znaków przerwanie od wysyłania zostaje wyłączone.

Funkcja przepisująca dane z globalnego bufora odbiorczego do bufora pomocniczego:

```
void usartGetValue() {
    if (od_poczatek != od_wolny) {
        static int index = 0;
        static char tab[128];
            if (odbieranie[od_poczatek] != '\n' && odbieranie[od_poczatek] != '\r') {
                if (odbieranie[od_poczatek] != ';') {
                    tab[index] = odbieranie[od_poczatek];
                    index++;
                    if (index >= 128)
                        index = 0;
                } else {
                    checkCommand(tab);
                    index = 0;
                }
            }
            od_poczatek++;
            if (od_poczatek >= 128)
                od_poczatek = 0;
    }
}
```

Zadaniem w/w funkcji jest przepisywanie danych z globalnej tablicy do tablicy pomocniczej. Znaki dodawane są do momentu pojawienia się średnika. W chwili jego wykrycia następuje przejście do funkcji parsującej.

Funkcja sprawdzająca wykryte polecenie sterujące:

```
void checkCommand(char *tablica) {
    if (strncmp(tablica, "setXSteps ", 10) == 0) {
        if ((parametr(tablica) >= minSteps && parametr(tablica) <= maxSteps) && axisXEnabled == OFF) {
            stepsX = parametr(tablica);
            sendString(conkat("INFO: Axis X, steps = ", toString(stepsX), "\r\n"));
        } else
            sendString("ERROR: Invalid steps range <10:230>,\r\n or engine X is currently enabled.\r\n");
    } else if (strncmp(tablica, "setXAccel ", 10) == 0) {
        if ((parametr(tablica) >= minAccelTime && parametr(tablica) <= maxAccelTime) && axisXEnabled == OFF) {
            accelXTime = parametr(tablica);
            sendString(conkat("INFO: Axis X, accel time = ", toString(accelXTime), "\r\n"));
        } else
            sendString("ERROR: Invalid accel time range <1:5>,\r\n or engine X is currently enabled.\r\n");
    } else if (strncmp(tablica, "setXDecel ", 10) == 0) {
        if ((parametr(tablica) >= minDecelTime && parametr(tablica) <= maxDecelTime) && axisXEnabled == OFF) {
            decelXTime = parametr(tablica);
            sendString(conkat("INFO: Axis X, decel time = ", toString(decelXTime), "\r\n"));
        } else
            sendString("ERROR: Invalid decel time range <1:5>,\r\n or engine X is currently enabled.\r\n");
    } else if (strncmp(tablica, "setXSpeedMin ", 13) == 0) {
        if ((parametr(tablica) >= speedMin && parametr(tablica) <= speedMax) && axisXEnabled == OFF) {
            vXMin = parametr(tablica);
            sendString(conkat("INFO: Axis X, speed min = ", toString(vXMin), "\r\n"));
        } else
            sendString("ERROR: Invalid speed min range <1:10>,\r\n or engine X is currently enabled.\r\n");
    }else if (strcmp(tablica, "showCommands") == 0) {
        sendString("\r\nINFORMACJE \r\nsetSteps - ilosc krokow do wykonania.\r\n");
        sendString("setSpeedMin - predkosc minimalna.\r\n");
        sendString("setSpeedMax - predkosc maksymalna.\r\n");
        sendString("setAccel - czas rozpedzania.\r\n");
        sendString("setDecel - czas hamowania.\r\n");
        sendString("showCommands - dostepne polecenia.\r\n");
        sendString("axisXmoveL - ruch os 'x' kierunek lewo.\r\n");
        sendString("axisXmoveR - ruch os 'x' kierunek prawo.\r\n");
    } else
        sendString("ERROR: Wrong command.\r\n");

    memset(tablica, 0, strlen(tablica));
}
```

W w/w funkcja odpowiedzialna jest za sprawdzenie wartości odebranej z klawiatury. Jeśli funkcja posiada parametr jest on ściśle sprawdzany czy znajduje się w danym przedziale. W chwili wykrycia nieznanego polecenia na konsoli zostaje wyświetlony stosowny komunikat.

Funkcje wyświetlające parametry silnika:

```
void basicXSettings(){
    sendString("BASIC X SETTINGS\r\n");
    sendString(conkat("Axis X, speed min = ",toString(vXMin),"\r\n"));
    sendString(conkat("Axis X, speed max = ",toString(vXMax),"\r\n"));
    sendString(conkat("Axis X, steps = ",toString(stepsX),"\r\n"));
    sendString(conkat("Axis X, accel = ",toString(accelXTime),"\r\n"));
    sendString(conkat("Axis X, decel = ",toString(decelXTime),"\r\n"));
}
```

Zadaniem w/w funkcji jest wyświetlenie wartości startowych silnika po wczytaniu programu. Wartości podstawowe zostały zadeklarowane w programie a ich możliwość zmiany jest wyłącznie za pomocą portu USART.

Funkcja wykonywana w momencie wykrycia polecenia: „axisXmoveLeft / axisXmoveRight:

```
void axisXmove(int stepsX, float vXMin, float vXMax,int accelXTime, int decelXTime,  int direction) {
    if (axisXEnabled == OFF) {
        accelerationX = (vXMax - vXMin) / (accelXTime * 1000.0);
        decelerationX = (vXMax - vXMin) / (decelXTime * 1000.0);

        stepsXAccel = ceil(((vXMin * accelXTime) + ((accelerationX * 1000)* (accelXTime * accelXTime)) / 2))
        stepsXDecel = ceil(((vXMin * decelXTime) + ((decelerationX * 1000)* (decelXTime * decelXTime)) / 2))
        middleXSteps = stepsX - stepsXAccel - stepsXDecel;

        if (stepsX >= stepsXAccel + stepsXDecel) {
            if (direction == LEFT){
                sendString("Engine X running. - left\r\n");
                GPIO_SetBits(axisXpin, axisXdir);
            }
            else{
                sendString("Engine X running. - right\r\n");
                GPIO_ResetBits(axisXpin, axisXdir);
            }
            axisXEnabled = ON;
            currentXSpeed = vXMin;
            stepAxisX = 1;
            times_X_ms = 0;
            TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
        } else
            sendString(conkat("ERROR: Minimum ",toString(stepsXAccel + stepsXDecel)," steps.\r\n"));
    } else
        sendString("ERROR: Engine X is currently enabled.\r\n");
}
```

Zadaniem w/w funkcji jest wybór kierunku ruchu silnika oraz obliczenie ilości kroków potrzebnych na rozpędzenie, hamowanie, drogę z prędkością maksymalną. Po wykonaniu obliczeń funkcja załącza przerwanie od Timera generowane co 1ms.

Przerwanie od TIMERA:

```
void TIM2_IRQHandler() {
    if (TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET) {
        times_X_ms++;
        if (stepAxisX <= stepsX) {
            if (times_X_ms > 0 && times_X_ms <= 50) {
                GPIO_SetBits(axisXpin, axisXstep);
            } else {
                GPIO_ResetBits(axisXpin, axisXstep);
            }
                if (stepAxisX < stepsXAccel)
                    currentXSpeed += accelerationX;
                else if (stepAxisX <= (stepsXAccel + middleXSteps))
                    currentXSpeed = 10;
                else {
                    if (currentXSpeed <= 1)
                        currentXSpeed = 1;
                    else
                        currentXSpeed -= decelerationX;
                }
            if (times_X_ms >= (1000 / currentXSpeed)) {
                times_X_ms = 0;
                stepAxisX++;
            }
        } else {
            axisXEnabled = OFF;
            TIM_ITConfig(TIM2, TIM_IT_Update, DISABLE);
        }
        TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
    }
}
```

Przerwanie jest wykonywane do chwili aż wartość obecnego kroku jest mniejsza od kroków do przebycia. Wartość wypełnienia impulsu w tym przypadku jest stała i wynosi 50ms. Jeśli wartość obecnego kroku jest mniejsza od kroków potrzebnych do rozpędzenia zmienna prędkość zostaje inkrementowana. Gdy program wykryje przypadek rozpoczęcia drogi z maksymalną prędkości to wartość obecnej prędkości równa się prędkości maksymalnej. W przeciwnym wypadku rozpoczyna się hamowanie i od prędkości obecnej zostaje odejmowana wartość hamowania

Funkcje odpowiedzialne za konwersję:

```
char *toString(int x) {
    char tab[10];
    char *str = &tab[0];
    sprintf(str, "%d", x);
    return str;
}

char *floatToString(float x){
    char str[33];
    char * p = &str[0];
    sprintf(str, "%0.5f", x);
    return p;
}
```

Zadaniem w/w funkcji jest przekonwertowanie zmiennej typu integer lub float na typ char aby w późniejszym czasie wyświetlić je w terminalu.

Funkcja odpowiedzialna za łączenie znaków:

```
char *conkat(char *a, char *b, char *c){
    char str[120];
    char *p = &str[0];
    sprintf(str,"%s %s %s", a, b, c);
    return  p;
}
```

Zadaniem w/w funkcji jest połączenie trzech zmiennych typu char w jedną tablicę znakową.

Funkcja wyłuskująca parametr z odebranego polecenia:

```
int parametr(char tab[]) {
    char t[10];
    char t2[10];
    sscanf(tab, "%s" "%s" "%s", t, t2, t);
    for(int i = 0; i< strlen(t2); i++){
        if(!isdigit(t2[i]))
            return -1;
    }
    memset(tab, 0, strlen(tab));
    return atoi(t2);
}
```

Zadaniem w/w funkcji jest pobranie wartości znajdującej się po znaku spacji w odebranym poleceniu. Funkcja została także zabezpieczona przed błędnie wprowadzonym parametrem.

Wnioski: Z powodu identycznej implementacji dwóch osi silników w dokumentacji zostały zawarte wyłącznie funkcje przedstawiające inicjalizację oraz obliczenia związane z rozpędzaniem i hamowaniem silnika poruszającego się w osi x.

Wykonany projekt nauczył mnie wiele teoretycznych, praktycznych umiejętności związanych z rozpędzaniem, hamowaniem silnika krokowego(wykorzystanie rampy). Podczas tworzenia projektu napotkałem wiele problemów, z którymi ostatecznie udało mi się rozwiązać. Sądzę, że komunikacja za pomocą USART jest użyteczna i praktyczna ponieważ w prosty sposób nawiązuje komunikację z mikroprocesorem.