Projekt sterownia silnikami prądu stałego przy pomocy STM32 Discovery

Technologia mikroporocesorowa

Paweł Batko, Krzysztof Romanowski

Spis treści

Przedmiot projektu

Przedmiotem projektu było zaprojektowanie oraz wykonanie sterowania dla modułu jezdnego napędzanego dwoma silnikami prądu stałego. Rozwiązanie miało bazować na mikrokontrolerze STM32 Discovery oraz być sterowane z zewnątrz przez port Szeregowy (USART).

Parametry techniczne modułu jezdnego

Parametry silników jakie udało nam się ustalić

- napięcie : 12 V
- natężenie prądu bez dużego obciążenia: 1 A
- natężenie prądu przy dużym obciążeniu: 3,5 A
- zasilanie z 12 V baterii
- brak skrętnych osi – sterowanie na zasadzie modyfikacji momentu obrotowego prawego lub lewego silnika

Użyte komponenty

Mikrokontroler : STM32F100
Zestaw ewaluacyjny: STM32VLDISCOVERY
Mostki: L298N
Stabilizator napięcia: L7805

Na płytce dołączone są Data Sheet'y tych elementów.

Pełna lista elementów:

- DUAL FULL-BRIDGE DRIVER L298 x2
- POSITIVE VOLTAGE REGULATOR L7805 x1
- SCHOTTKY DIODE 3A x8
- STM32VLDISCOVERY x1
- BLUE LED x2
- 100nF CAPACITOR x6
- 470uF CAPACITOR x4
- 0,5OHM RESISTOR x4
- 430OHM RESISTOR x2

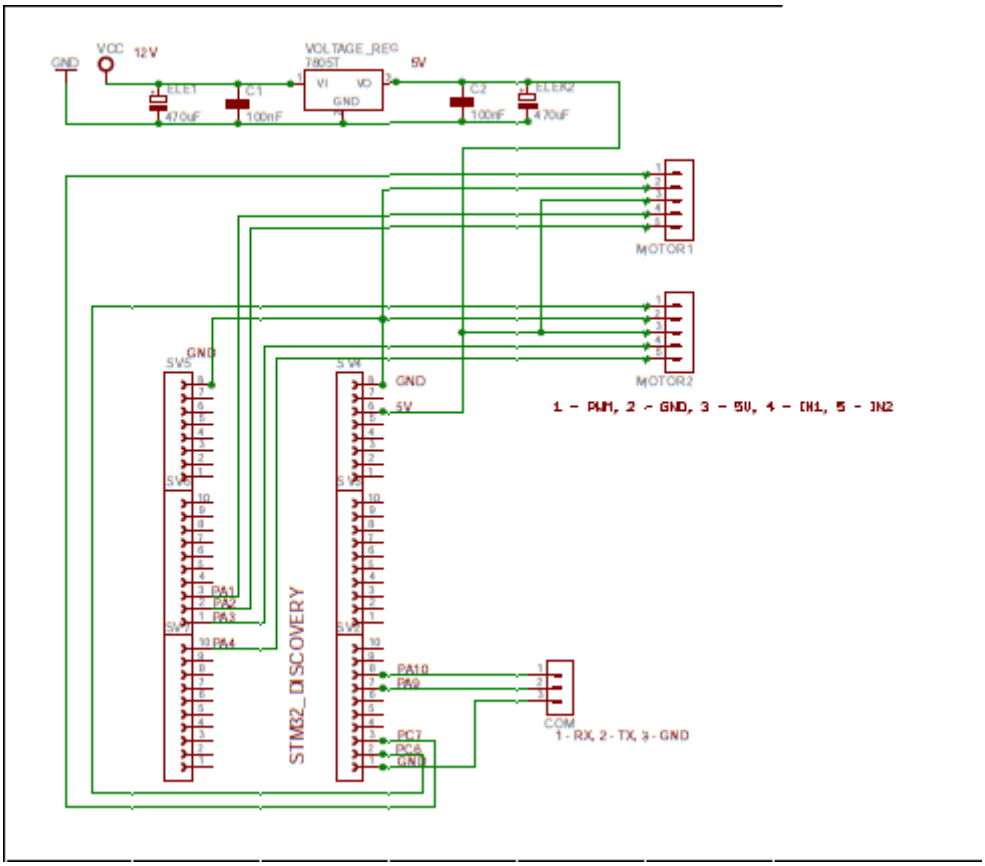
Rozwiązanie

Po analizie problemu doszliśmy do wniosku, że wykonanie jednej płytki z całym układem może być niekorzystne. Podzieliśmy projekt na 3 mniejsze części: Układ zapewniający szukane napięcia oraz logikę, oraz układy sterowania każdym z silników (analogiczne) Ma to szereg zalet:

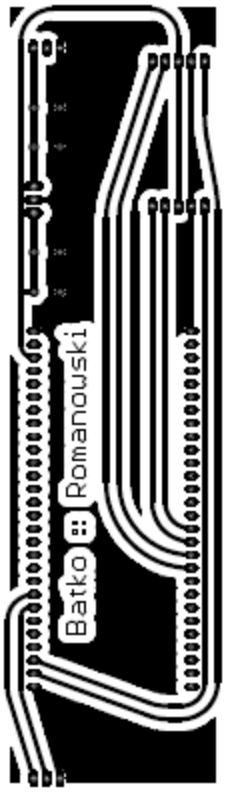
- Możliwość wymiany wadliwych elementów
- Możliwość wymiany całych modułów – np. zasępienie układu sterowania
- Fizyczna separacja nagrzewających się elementów (mostków)
- Łatwiejsze umieszczanie układu w module jezdnym.

Specyfikacja układu zapewniania zasilania oraz logiki

Schemat elektryczny:



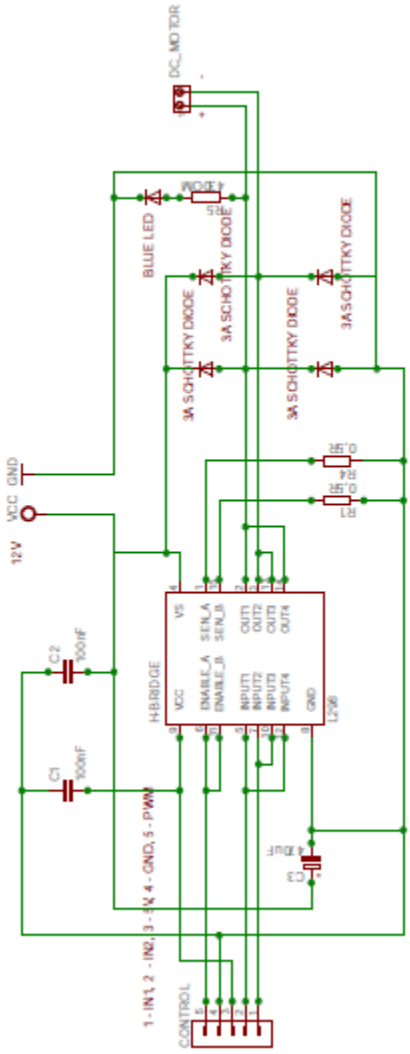
Wygląd płytki:



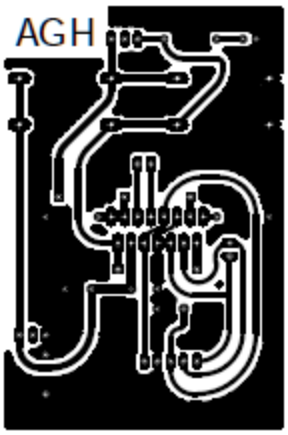
Batko, Romanowski

Specyfikacja moduł sterowania silnikiem.

Schemat elektryczny:



Wygląd na płytce:



Opis protokołu komunikacyjnego

Z układem można komunikować się przy pomocy następującego protokołu:
Wiadomość zawsze się z 3 bajtów:
([LRB][FBS]Value)
gdzie jeśli nie podano inaczej przesyłamy watość danego znaku w kodzie ANSI

- 1 bajt – który silnik sterujemy
  - ✗ L – oznacza lewy silnik
  - ✗ R – oznacza prawy silnik
  - ✗ B – oznacza oba silniki
- 2 bajt – stan pracy silnika
  - ✗ F – do przodu
  - ✗ B – do tył
  - ✗ S – wyłącz silnik
- 3 bajt – procent mocy silnika – wartość od 0 do 99

  lub 1 bajtu H – oznacza zakończenie pracy i wyłączenie silników
  lub dowolnego ciągu bajtów nie dłuższego niż 3 i zakończonego X- pusta komenda.

Po wysłaniu wiadomości układ pozostaje w podanym stanie aż do otrzymania kolejnej przez programowalny czas (sprzętowy watch dog).

Kod źródłowy aplikacji:

```
USART_InitTypeDef USART_InitStructure;
```

Inicjalizjacja WatchDoga

```
void initWatchDog() {
	// Zezwolenie za zappis rejestrow IWDG
	IWDG_WriteAccessCmd(IWDG_WriteAccess_Enable);
	// IWDG taktowany zegarem 40kHz/128 = 312Hz
	IWDG_SetPrescaler(IWDG_Prescaler_128);
	IWDG_SetReload(0xFFF); // Przepelnienie IWDG po okolo 13s
	IWDG_ReloadCounter(); // Przeladowaie IWDG
	IWDG_Enable(); // Wlaczenie IWDG i LSI
}
```

inicjalizacja PWM

```
void initPWM() {

	GPIO_InitTypeDef GPIO_InitStructure;
	TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStruct;
	TIM_OCInitTypeDef TIM_OCInitStruct;

	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC | RCC_APB2Periph_AFIO, ENABLE);

	RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);

	GPIO_StructInit(&GPIO_InitStructure); // Reset init structure

	// Setup Blue & Green LED on STM32-Discovery Board to use PWM.
	GPIO_InitStructure.GPIO_Pin = //GPIO_Pin_8 | GPIO_Pin_9 |
			GPIO_Pin_6 | GPIO_Pin_7;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP; // Alt Function - Push Pull
	GPIO_Init(GPIOC, &GPIO_InitStructure);

	// Map TIM3_CH3 to GPIOC.Pin8, TIM3_CH4 to GPIOC.Pin9
	GPIO_PinRemapConfig(GPIO_FullRemap_TIM3, ENABLE);

	// Let PWM frequency equal 100Hz.
	// Let period equal 1000. Therefore, timer runs from zero to 1000.
	// Gives 0.1Hz resolution.
	// Solving for prescaler gives 240.

	TIM_TimeBaseStructInit(&TIM_TimeBaseInitStruct);
	TIM_TimeBaseInitStruct.TIM_ClockDivision = TIM_CKD_DIV4;
	TIM_TimeBaseInitStruct.TIM_Period = 1000 - 1; // 0..999
	TIM_TimeBaseInitStruct.TIM_Prescaler = 240 - 1; // Div 240
	TIM_TimeBaseInit(TIM3, &TIM_TimeBaseInitStruct);

	TIM_OCStructInit(&TIM_OCInitStruct);
	TIM_OCInitStruct.TIM_OutputState = TIM_OutputState_Enable;
	TIM_OCInitStruct.TIM_OCMode = TIM_OCMode_PWM1;

	// Initial duty cycle equals 0%. Value can range from zero to 1000.
	TIM_OCInitStruct.TIM_Pulse = 0; // 0 .. 1000 (0=Always Off, 1000=Always On)
	TIM_OC1Init(TIM3, &TIM_OCInitStruct); // Channel 3 Blue LED // PC6


	TIM_OCStructInit(&TIM_OCInitStruct);
	TIM_OCInitStruct.TIM_OutputState = TIM_OutputState_Enable;
	TIM_OCInitStruct.TIM_OCMode = TIM_OCMode_PWM1;
	// Initial duty cycle equals 0%. Value can range from zero to 1000.
	TIM_OCInitStruct.TIM_Pulse = 0; // 0 .. 1000 (0=Always Off, 1000=Always On)
	TIM_OC2Init(TIM3, &TIM_OCInitStruct); // PC7


	TIM_Cmd(TIM3, ENABLE);
}
```

Inicjalizacja USART'a

```
void initUSART() {
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_USART1, ENABLE);

	NVIC_InitTypeDef NVIC_InitStructure;

	/////////////////InterruptConfig
	NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
	NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 3;
	NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
	NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
	NVIC_Init(&NVIC_InitStructure);

	GPIO_InitTypeDef structGPIO_Init;

	//kofiguracja PA9 jako tx
	structGPIO_Init.GPIO_Pin = GPIO_Pin_9;
	structGPIO_Init.GPIO_Mode = GPIO_Mode_AF_PP;
	structGPIO_Init.GPIO_Speed = GPIO_Speed_50MHz;

	GPIO_Init(GPIOA, &structGPIO_Init);

	//kofiguracja PA10 jako rx
	structGPIO_Init.GPIO_Pin = GPIO_Pin_10;
	structGPIO_Init.GPIO_Mode = GPIO_Mode_IN_FLOATING;
	structGPIO_Init.GPIO_Speed = GPIO_Speed_50MHz;

	GPIO_Init(GPIOA, &structGPIO_Init);

	//konf usart

	USART_InitTypeDef structUSART_Init;
	structUSART_Init.USART_BaudRate = 19200;
	structUSART_Init.USART_WordLength = USART_WordLength_8b;
	structUSART_Init.USART_StopBits = USART_StopBits_1;
	structUSART_Init.USART_Parity = USART_Parity_No;
	structUSART_Init.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
	structUSART_Init.USART_Mode = USART_Mode_Rx;

	USART_Init(USART1, &structUSART_Init);
	USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
	USART_Cmd(USART1, ENABLE);

}
```

Reszta inicjalizacji

```
void init() {
	//dodki
	STM32vldiscovery_LEDInit(LED3);
	STM32vldiscovery_LEDInit(LED4);
	STM32vldiscovery_PBInit(BUTTON_USER, BUTTON_MODE_GPIO);
	STM32vldiscovery_LEDOff(LED3);
	STM32vldiscovery_LEDOff(LED4);
	RCC->APB2ENR = 0
	// Turn on IO Port A
			| RCC_APB2ENR_IOPAEN
	// Turn on IO Port B
			| RCC_APB2ENR_IOPBEN;
	// set PA0 to input floating, set PA1, PA2, PA3, PA4 to output
	GPIOA->CRL = 0x00011114;

	initPWM();
	initUSART();
	initWatchDog();
}
```

Funkcje sterujące silnikami

```
void motor_left_forward() {
	GPIOA->BSRR = GPIO_Pin_1;
	GPIOA->BRR = GPIO_Pin_2;
}

void motor_left_backward() {
	GPIOA->BRR = GPIO_Pin_1;
	GPIOA->BSRR = GPIO_Pin_2;
}

void motor_left_free() {
	GPIOA->BRR = GPIO_Pin_1;
	GPIOA->BRR = GPIO_Pin_2;
	TIM3->CCR2 = 0;
}

void motor_right_backward() {
	GPIOA->BSRR = GPIO_Pin_3;
	GPIOA->BRR = GPIO_Pin_4;
}

void motor_right_forward() {
	GPIOA->BRR = GPIO_Pin_3;
	GPIOA->BSRR = GPIO_Pin_4;
}

void motor_right_free() {
	GPIOA->BRR = GPIO_Pin_3;
	GPIOA->BRR = GPIO_Pin_4;
	TIM3->CCR1 = 0;
}
```

Ustawianie wartości PWM'ów

```
void set_right_PWMSpeed(uint16_t fillPercentage) {

	if (fillPercentage > 99)
		return;
	if (fillPercentage < 0)
		return;
	uint16_t value = fillPercentage * 10;
	TIM3->CCR1 = value;
}

void set_left_PWMSpeed(uint16_t fillPercentage) {

	if (fillPercentage > 99)
		return;
	if (fillPercentage < 0)
		return;

	uint16_t value = fillPercentage * 10;
	TIM3->CCR2 = value;
}
```

Pobieranie wartości z USART'a

```
char getCharFromUsart() {
	char tmp;

	while (USART_GetFlagStatus(USART1, USART_FLAG_RXNE) == RESET)
		;

	tmp = USART_ReceiveData(USART1);
	return tmp;
}
```

Sterowanie silnikiem

```
void runMotor_left(char mode, char power) {
	set_left_PWMSpeed(power);
	if (mode == 'F') {
		motor_left_forward();
	} else if (mode == 'B') {
		motor_left_backward();
	} else if (mode == 'S') {
		motor_left_free();
	}
}

void runMotor_right(char mode, char power) {
	set_right_PWMSpeed(power);
	if (mode == 'F') {
		motor_right_forward();
	} else if (mode == 'B') {
		motor_right_backward();
	} else if (mode == 'S') {
		motor_right_free();
	}
}
```

Wybór silników

```
void doAction(char motor_select, char motor_mode, char power_perc) {
	if (motor_select == 'L') {
		runMotor_left(motor_mode, power_perc);
	} else if (motor_select == 'R') {
		runMotor_right(motor_mode, power_perc);
	} else if (motor_select == 'B') {
		runMotor_left(motor_mode, power_perc);
		runMotor_right(motor_mode, power_perc);
	}
}
```

Koniec pracy.

```
void finalize() {
	uint16_t pwm;
	uint16_t i = 0;
	pwm = 0;
	set_right_PWMSpeed(pwm);
	set_left_PWMSpeed(pwm);
	while (1) {
		i = i + 1;
		motor_right_free();
		motor_left_free();
	}
}
```

Główna pętla

```
int main(void) {
	init();

	while (run)
		;

	finalize();
}
```

Funkcja obsługii przerwania

```
void USART1_IRQHandler(void) {
		if (run) {
			IWDG_ReloadCounter(); // reload counter

			if (USART_GetITStatus(USART1, USART_IT_RXNE) != RESET) {
				/* Read one byte from the receive data register */

				char this_msg = USART_ReceiveData(USART1);
				uart_msg[uart_counter++] = this_msg;
				if (this_msg == 'X') {
					uart_counter = 0;
				} else if (this_msg == 'E') {
					run = 0;
				} else if (uart_counter >= 3) {
					uart_counter = 0;
					doAction(uart_msg[0], uart_msg[1], uart_msg[2]);
				}
				/* Clear the USART1 Receive interrupt */
				USART_ClearITPendingBit(USART1, USART_IT_RXNE);

			}
		}
	}
```

Wykonanie Projektu:

Projekt realizowany był w domowych warunkach, przy użyciu często mało profesjonalnych narzędzi. Załączamy zdjęcia oraz krótki filmik z pracy nad przygotowaniem płytek.

Efekt finalny: