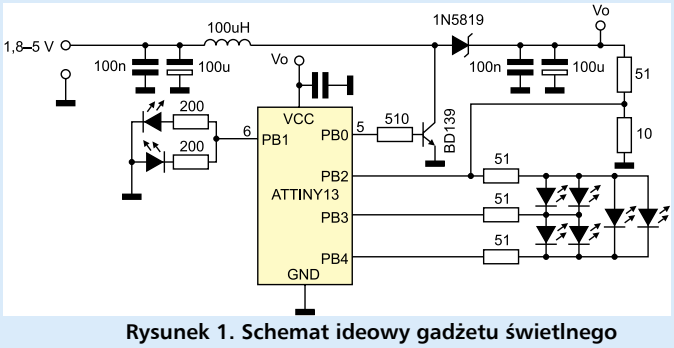
Programowo-sprzętowe przetwornice step-up w przykładach

Wielokrotnie budując układy zasilane bateryjne istnieje konieczność podniesienia napięcia baterii zasilającej, np. do zasilania układów peryferyjnych czy wyświetlacza. Wykonanie przetwornicy w dobie specjalizowanych układów scalonych nie stanowi większego problemu. Jednak jeżeli nie mamy żadnego takiego układu pod ręką i w budowanym układzie mikroprocesorowym istnieje wolny przetwornik A/C oraz układ generatora sygnału PWM, a wymagania nakładane na przetwornice nie są duże, możemy pokusić się o wykonanie przetwornicy za pomocą mikrokontrolera oraz kilku elementów dyskretnych.

Wykonując taki układ w najprostszej wersji musimy liczyć się z tym, że nie będzie on miał funkcjonalności dostępnej w specjalizowanych komponentach np. obwodów zabezpieczenia przeciwzwarciowego czy przed przegrzaniem się. Bardziej zaawansowanego układu nie opłaca się budować, ponieważ koszt zakupu dodatkowych elementów przewyższy koszt wykonania gotowej przetwornicy.

Przetwornica na przykładzie gadżetu świetlnego z ATtiny13

Przetwornica zrealizowana w sposób programowy, przydaje się najczęściej w najprostszych układach elektronicznych, gdzie zastosowanie dodatkowego, zintegrowanego układu przetwornicy jest problematyczne i stanowi dodatkowy problem związany z kosztem lub brakiem miejsca na płytce. Przykładem takiego projektu jest prosty układ realizujący efekt świetlny na diodach LED zasilany z dwóch baterii AAA. Zadaniem układu będzie wyświetlanie sekwencji świetlnych na diodach LED, które zostały zapisane w pamięci stałej. Jeśli chcemy w maksymalnym stopniu wykorzystać energię zgromadzoną w baterii, a planujemy użyć diod o wysokim napięciu przewodzenia (np. niebieskich), konieczne jest zastosowanie przetwornicy podwyższającej. Schemat ideowy takiego gadżetu przedstawiono na **rysunku 1**.



Rysunek 1. Schemat ideowy gadżetu świetlnego z ATtiny13

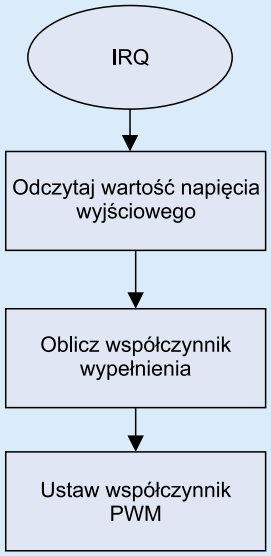
Dodatkowe informacje:
Kody źródłowe oraz wynikowe programu zawierające algorytm sterujący przetwornicą wraz z kompletnym przykładem efektu świetlnego dostępne są pod adresami: *ftp://boff.pl/elektronika_praktyczna/software-stepup-converters/stepup-light.zip*, *ftp://boff.pl/elektronika_praktyczna/software-stepup-converters/stepup_converter.zip* oraz na serwerze *ftp.ep.com.pl* w katalogu zawierającym dodatkowe materiały do artykułów.

Sercem układu jest 8-nóżkowy mikrokontroler ATtiny13 zawierający 1 kB pamięci Flash, 64 B pamięci RAM, a co najważniejsze – przetwornik A/C oraz timer mający 2-kanałowy układ PWM. Do pinu PB1, który również stanowi wyjście kanału #B układu PWM, dołączono dwie diody LED połączone równolegle. Zastosowanie PWM do zasilania diod LED umożliwia płynną regulacje ich jasności.

Do linii portów PB2, PB3 oraz PB4 zostało dołączono 6 diod LED koloru niebieskiego. Sterowanie 6 diodami LED za pomocą 3 wyprowadzeń jest możliwe dzięki traktowaniu linii GPIO jak wyjść trójstanowych. Każda linia może być ustawiona, wyzerowana lub w stanie wysokiej impedancji. Dzięki temu poprzez odpowiednią kombinację poziomów logicznych jest możliwe zaświecenie jednej z sześciu diod. Dodatkowo, stosując multipleksowanie możemy dowolnie sterować wszystkimi diodami jednocześnie.

Przetwornica podwyższającą napięcie zrealizowano w typowy sposób z wykorzystaniem cewki L1 diody D1 oraz tranzystora kluczującego T1. Sterowanie tranzystora T1, odbywa się za pomocą sygnału PWM generowanego przez kanał #A układu PWM. Układ pomiarowy napięcia wyjściowego przetwornicy stanowi dzielnik napięcia złożony z rezystorów R1 (61 kΩ) i R2 (10 kΩ). Jest on dołączony do kanału ADC1 przetwornika A/C. Z uwagi na mała liczbę wyprowadzeń linia PB2 jest wykorzystywana jako wejście przetwornika A/C oraz jako wyjście GPIO sterujące diodami LED. Pomiar napięcia wyjściowego przetwornicy odbywa się w przerwie, podczas której linia PB2 konfigurowana jest jako wejście A/C. Zasilanie mikrokon-

trolera stanowi bezpośrednio wyjście przetwornicy, a więc maksymalne napięcie wyjściowe nie powinno przekraczać maksymalnego napięcia zasilania układu. W tym przypadku napięcie wyjściowe zostało ustalone na 5 V, a prąd maksymalny na 60 mA. Po włączeniu napięcia zasilającego, gdy przetwornica jeszcze nie działa napięcie zasilające mikrokontroler będzie równe napięciu baterii pomniejszone o spadek napięcia na diodzie D1. Minimalne napięcie przy którym układ będzie w stanie się uruchomić, stanowić będzie napięcie przy którym mikrokontroler będzie w stanie rozpocząć wykonywanie programu pomniejszone o spadek napięcia na diodzie D1.



Rysunek 2. Algorytm zamkniętej pętli sterowania

Praktyczne próby wykazały, że układ z mikrokontrolerem ATtiny13, jest w stanie prawidłowo pracować od napięcia około 1,8 V. Mikrokontroler jest taktowany za pomocą sygnału z wewnętrznego generatora RC, którego częstotliwość za pomocą bitów konfigurujących ustawiono na wartość 9,6 MHz. Zatem generator PWM pracujący w trybie 8-bitowym może wygenerować sygnał kluczujący przetwornicę o maksymalnej częstotliwości około 37 kHz.

Obliczenia uproszczone

Znając częstotliwość kluczowania przetwornicy, która jest determinowana przez maksymalną częstotliwość sygnału PWM oraz parametry zastosowanego tranzystora kluczującego T1 i diody D1, należy wyliczyć indukcyjność cewki L1, oraz pojemność kondensatora filtrującego C4. Obliczenia wykonano w sposób uproszczony przy następujących założeniach:

- minimalne napięcie wejściowe: Vin_min=1,8 V,
- nominalne napięcie wyjściowe Vout=5 V,
- maksymalny prąd obciążenia Io=60 mA,
- dopuszczalny poziom tętnień napięcia na wyjściu Vripple=60 mV,
- spadek napięcia na kluczu tranzystorowym T1: Vsat=0,3 V,
- spadek napięcia na diodzie D1: Vf=0,3 V,
- częstotliwość taktowania Fpwm=37 kHz.

Pierwszą czynnością jest wyznaczenie czasu załączenia klucza, do czasu jego wyłączenia:

$$\frac{t_{on}}{t_{off}} = \frac{V_{out} + V_f - V_{in\,min}}{V_{in\,min} - V_{sat}} = \frac{5 + 0{,}3 - 1{,}8}{1{,}8 - 0{,}3} = 2{,}33$$

Kolejną czynnością jest wyznaczenie czasu trwania pojedynczego cyklu na podstawie częstotliwości sygnału PWM, który wynosi

$$t_{on\,max} + t_{off} = \frac{1}{f_{PWM}} = \frac{1}{37\ kHz} \approx 27\ \mu s$$

Następnie, na podstawie powyższych wzorów wyznaczamy czas załączenia klucza T_{on}, oraz czas przewodzenia diody T_{off}:

Listing 1. Obsługa przerwania od timera T0

```
//Define the sampling frequency
#define FREQ_ADC_SAMPLING_RATE 38
//Timer for timeout
static volatile uint8_t wait_timer;
static volatile stimer_t shutdown_timer;
ISR(TIM0_OVF_vect) {
  static uint8_t divider = FREQ_ADC_SAMPLING_RATE;
  if(--divider==0) {
    set_blue_led_output(0);
    start_adc_conv();
    divider = FREQ_ADC_SAMPLING_RATE;
  }
}
```

$$t_{off} = \frac{t_{on\,max} + t_{off}}{\frac{T_{on}}{T_{off}} + 1} = \frac{27\ \mu s}{2{,}33 + 1} = 8{,}1\ \mu s$$

$$T_{on} = t_{on\,max} + t_{off} - T_{off} = 27\ \mu s - 8{,}1\ \mu s = 18{,}9\ \mu s$$

Po wyliczeniu czasów załączenia klucza T_{on} oraz czasu przewodzenia diody T_{off} sprawdzamy warunek czy stosunek czasu załączenia do czasu wyłączenia dla najniższego dopuszczalnego napięcia zasilającego nie przekracza 0,85.

$$\frac{T_{on}}{T_{on\,max} + T_{off}} = \frac{18{,}9\ \mu s}{27\ \mu s} = 0{,}7$$

Otrzymana wartość 0,7 jest mniejsza od 0,85, a więc warunek najdłuższego dopuszczalnego czasu przewodzenia klucza jest spełniony.

Kolejną czynnością jest wyznaczenie maksymalnego prądu szczytowego, który będzie płynął przez cewkę L1 oraz klucz T1, tak abyśmy mogli dobrać odpowiednie parametry elementów.

$$I_{pk} = 2 \cdot I_0 \cdot \left(\frac{t_{on}}{t_{off}} + 1\right) = 2 \cdot 60\ mA \cdot (2{,}33 + 1) = 339\ mA$$

Maksymalny prąd szczytowy płynący przez klucz tranzystorowy, oraz cewkę L1 będzie wynosił około 340mA.

Po wyliczeniu maksymalnego prądu szczytowego, wyliczamy minimalną, dopuszczalną indukcyjność cewki L1, która będzie wynosiła:

$$L_{min} = \left(\frac{V_{in\,min} - V_{sat}}{I_{pk}}\right) \cdot t_{on} = \frac{1{,}8\ V - 0{,}3\ V}{339\ mA} \cdot 18{,}9\ \mu s \approx 71\ \mu H$$

Listing 2. Obsługa przerwania od przetwornika A/C

```
//Macro definition for the adc values
#define ADC_V_IN(x) ((R2_VALUE/( R1_VALUE + R2_VALUE )) * (x))
#define ADC_VALUE(v) ((uint16_t)((ADC_V_IN((v)) * 1024.0)/V_REF))
//ADC conversion finished it will be used to stepup handling
ISR(ADC_vect) {
  static uint8_t pwmv = 0;
  //E signal
  int16_t e = ADC_VALUE(V_STAB) -  get_adc_value();
  //Enable leds again
  set_blue_led_output(blue_led_value);
  bool minus = false;
  if(e<0) { e = -e; minus = true; }
  //Fuzzy logic control loop
  if(e>ADC_VALUE(3.0)) e = 40;
  else if(e>ADC_VALUE(2.5)) e = 30;
  else if(e>ADC_VALUE(2.0)) e = 20;
  else if(e>ADC_VALUE(1.5)) e = 10;
  else if(e>ADC_VALUE(1.0)) e = 6;
  else if(e>ADC_VALUE(0.5)) e = 4;
  else if(e>ADC_VALUE(0.25)) e = 3;
  else if(e>ADC_VALUE(0.125)) e = 2;
  else e = 1;
  //Check and saturation handling
  if(!minus) {
    if(e+(int16_t)pwmv>MAX_PWM_VALUE ) pwmv = MAX_PWM_VALUE;
    else pwmv += e;
  }
  else {
    if((int16_t)pwmv-e<0 ) pwmv = 0;
    else pwmv -= e;
  }
  //Step pwm output
  set_stepup_pwm(pwmv);
}
```

Listing 3. Wyłączenie przetwornicy, uśpienie CPU

```
//Shutdown the CPU disable all perhips
void vc_shutdown(stimer_t value)
{
  cli();
  //Setup timer value
  shutdown_timer = value;
  //Disable blue led hiz
  blue_led_value = 0;
  set_blue_led_output(0);
  //Shutdown the periphs
  periph_shutdown();
  //Emable interrupt
  sei();
  //Wait for timer elapsed
  while( shutdown_timer ) {
    wdt_reset();       //Reset wdt
    sleep();           //go to sleep
    wdt_reset();       //Reset wdt
  }
  //Reset watchdog
  wdt_reset();
  //Enable periph
  periph_wakeup();
}
```

Przyjmijmy, że indukcyjność cewki L1 będzie wynosiła 100 µH a jej dopuszczalny prąd przewodzenia będzie nie mniejszy niż m 350 mA.

Ostatnią czynnością jest wyznaczenie pojemności wyjściowej pojemności filtrującej C4, którą obliczamy w następujący sposób:

$$C_{4\,min} = \frac{I_0}{V_{ripple}} \cdot t_{on} = \frac{60\,mA}{60\,mV} \cdot 18{,}9\,\mu s = 19{,}9\,\mu F$$

Minimalna teoretyczna pojemność filtrująca, aby osiągnąć poziom tętnień zasilania 60 mV wynosi 19,9 µF. Tu wybieramy pojemność równą 100 µF. Kondensator powinien charakteryzować się małym współczynnikiem ESR i najlepiej, gdy będzie to kondensator tantalowy. Jednak w naszym przypadku z uwagi na to, że napięcie tętnień nie ma tak dużego znaczenia, zastosujemy dobry kondensator elektrolityczny o niskim ESR i dodatkowy kondensator filtrujący 100 nF.

Oprogramowanie

Istotnym fragmentem przetwornicy jest oprogramowanie, którego zadaniem jest sterowanie współczynnikiem wypełnienia sygnału PWM kluczującego tranzystor T1 w taki sposób, aby utrzymać napięcie wyjściowe o zadanej wartości. Ogólna zasada sterowania przetwornicą jest typowym algorytmem zamkniętej pętli sterowania, wykonywanej ze stałym cyklem czasowym (**rysunek 2**).

Najpierw jest odczytywana wartość napięcia wyjściowego, a następnie jest obliczany współczynnik wypełnienia PWM, tak aby utrzymać napięcie o żądanej wartości. Następnie, obliczona nastawa PWM jest wpisywana do rejestru kontrolnego sprzętowego generatora PWM. Algorytm powinien być wykonywany ze stałym cyklem, który jest kompromisem pomiędzy obciążeniem procesora wynikającym z jego wykonania, a szybkością reakcji na zmianę obciążenia przetwornicy, a więc utrzymaniem stabilnego napięcia wyjściowego. Jako algorytm sterujący można wykorzystać np. klasyczny regulator PID lub algorytm „Fuzzy Logic". Z uwagi na to, że wykonanie pętli regulatora jest krytyczne czasowo oraz ze względu na niezawodność najlepiej będzie wykonać ją bezpośrednio w procedurze obsługi przerwania. Ponadto, takie rozwiązanie ma zaletę polegającą na tym że algorytm sterowania będzie się wykonywał w tle, niezależnie od właściwego programu.

Należy zwrócić szczególną uwagę na to, że „zawieszenie się" programu może spowodować zwarcie klucza lub utrzymanie sygnału PWM przy wysokim poziomie wypełnienia, co może uszkodzić procesor stanowiący obciążenie przetwornicy lub tranzystor kluczujący. Z tego powodu powinno oprogramowanie powinno być napisane w sposób gwarantujący niezawodność, a mikrokontroler powinien mieć włączony układ watchdog z możliwie jak najkrótszym czasem reakcji. W naszym przykładzie algorytm kontrolny realizowany z cyklem 1 ms zapewnia wystarczającą precyzję regulacji. Do odmierzania cykli wykorzystano przerwanie od przepełnienia układu czasowo – licznikowego T0 (**listing 1**).

Mikrokontroler ma tylko jeden układ czasowo – licznikowy, którego użyto do generowania sygnału PWM, a zatem przerwanie od jego przepełnienia musi być wykorzystane również do innych celów. Ponieważ układ został skonfigurowany w trybie 8-bitowym, jest ono zgłaszane bardzo często – co około 27 µs. Zatem, aby uzyskać okres wyzwalania co około 1 ms, dodatkowo jest ono dzielone programowo przez zmienną divide. Gdy osiągnie ona wartość 0, wówczas diody LED dołączone do portów PB2, PB3, PB4 są wyłączane poprzez ustawienie linii portów w kierunku wejścia, tak aby umożliwić pomiar napięcia wyjściowego przetwornicy za pomocą przetwornika A/C. Następnie jest wywoływana funkcja start_adc_conv(), która powoduje rozpoczęcie pomiaru napięcia wyjściowego.

```
// Start ADC conversion
static inline void start_adc_conv(void)
{
  ADCSRA |= _BV(ADSC);
}
```

Gdy przetwornik A/C zakończy konwersje, zgłaszane jest przerwanie od przetwornika A/C, które stanowi główną część obsługi algorytmu sterowania przetwornicą (**listing 2**).

Jako algorytm sterujący zastosowano regulator z logiką rozmytą, który był najłatwiejszy w implementacji w układzie wyposażonym w tylko 1 kB pamięci Flash. Pierwszą czynnością jest wyznaczenie uchybu, który stanowi różnicę pomiędzy wielkością zmierzoną a wielkością zadaną. Następnie, zależnie od wielkości uchybu, w sposób tablicowy jest ustalana wartość, o którą zostanie zmieniony aktualny współczynnik wypełnienia PWM. Wartość ta jest następnie dodawana do aktualnego współczynnika sygnału PWM, a potem – po wprowadzeniu ograniczeń związanych z maksymalną i minimalną wartością w zakresie 0 do 215, jest wpisywana do rejestru ustalającego współczynnik wypełnienia sygnału w kanale A.

```
static inline void set_stepup_pwm(uint8_t value)
{
  OCR0A = 255 - value;
}
```

Maksymalna wartość wynosząca 215 wynika z opisanego wcześniej maksymalnego czasu włączenia klucza, który nie powinien przekraczać wartości 0,85, co dla 8-bitowego sygnału PWM przekłada się na wspomnianą wcześniej wartość.

Przy sterowaniu efektami świetlnymi przetwornica nie pracuje przez cały czas i jest cyklicznie włączana na okres świecenia diod, po czym następuje uśpienie procesora na czas około 15 sekund. W tym czasie przetwornica jest wyłączana, a napięcie wyjściowe zasilające procesor

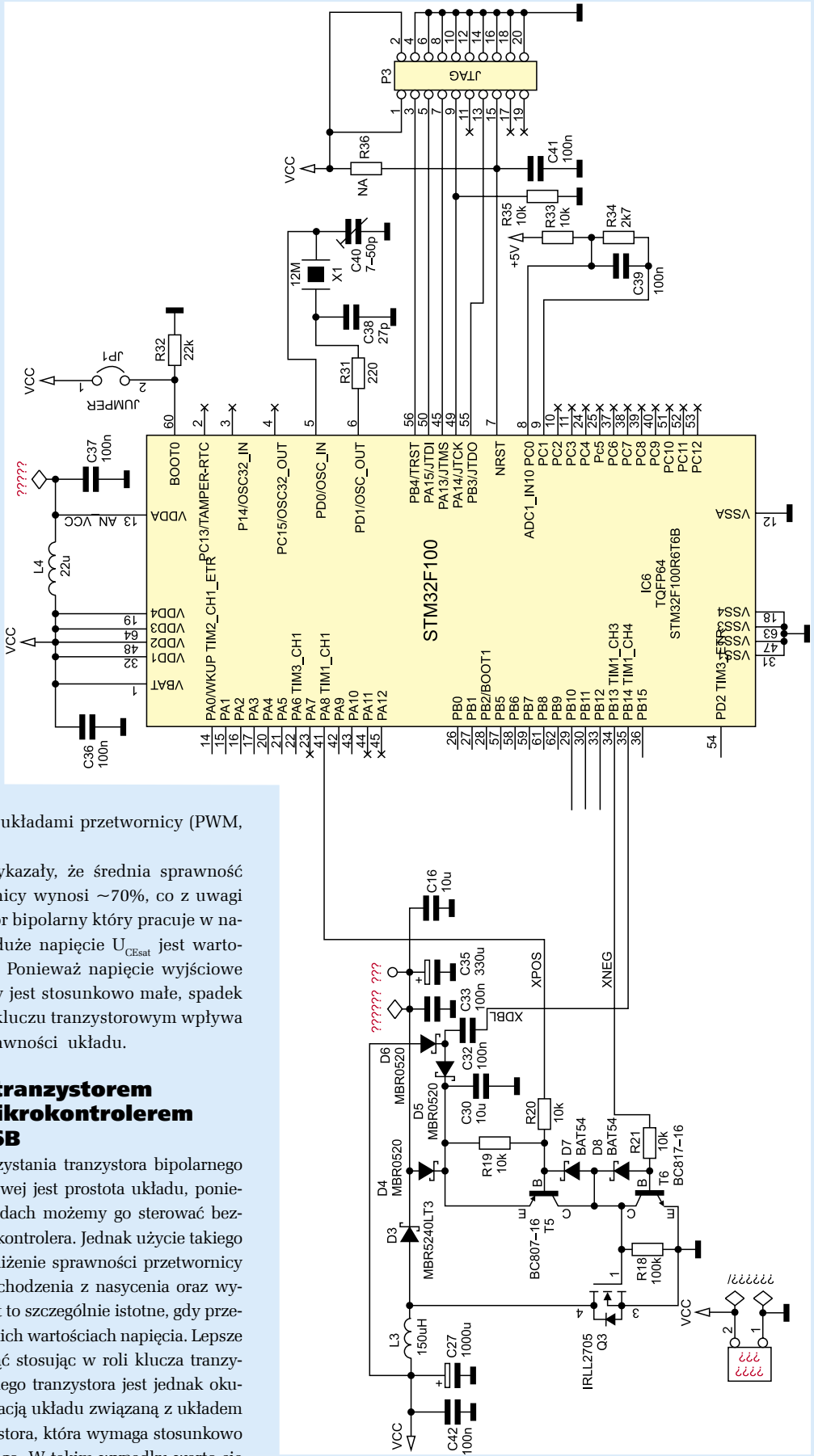
stopniowo obniża się, aż do osiągnięcia napięcia baterii pomniejszonego o spadek na diodzie D1. Usypianie i wybudzanie realizowane jest przez funkcję vc_shutdown(), która jako parametr przyjmuje czas na jaki procesor jest usypiany (**listing 3**).

Czas jest odmierzany za pomocą watchdoga, który może również pełnić rolę układu czasowo – licznikowego. Dzięki temu pobór prądu w stanie spoczynku jest minimalny. Na początku wyłączane są wszystkie przerwania oraz diody dołączone do portów PB2, PB3, PB4. Następnie, wszystkie układy peryferyjne są dezaktywowane oraz ponownie włączane są przerwania. Po odmierzeniu pożądanego odcinka czasu, wszystkie układy peryferyjne są aktywowane, włącznie z układami przetwornicy (PWM, A/C).

Praktyczne próby wykazały, że średnia sprawność tak wykonanej przetwornicy wynosi ~70%, co z uwagi na zastosowany tranzystor bipolarny który pracuje w nasyceniu i posiada dość duże napięcie U_{CEsat} jest wartością całkiem przyzwoitą. Ponieważ napięcie wyjściowe i wejściowe przetwornicy jest stosunkowo małe, spadek napięcia na diodzie D1 i kluczu tranzystorowym wpływa istotnie na obniżenie sprawności układu.

Przetwornica z tranzystorem MOSFET oraz mikrokontrolerem STM32F100R6T6B

Główną zaletą wykorzystania tranzystora bipolarnego w przetwornicy programowej jest prostota układu, ponieważ przy niewielkich prądach możemy go sterować bezpośrednio z portów mikrokontrolera. Jednak użycie takiego tranzystora powoduje obniżenie sprawności przetwornicy z powodu powolnego wychodzenia z nasycenia oraz wysokiego napięcia U_{CEsat}. Jest to szczególnie istotne, gdy przetwornica operuje przy niskich wartościach napięcia. Lepsze rezultaty możemy osiągnąć stosując w roli klucza tranzystor MOSFET. Użycie takiego tranzystora jest jednak okupione dodatkową komplikacją układu związaną z układem sterującym bramką tranzystora, która wymaga stosunkowo dużego napięcia zasilającego. W takim wypadku warto się zastanowić czy nie lepiej będzie użyć gotowego układu przetwornicy, który najczęściej do prawidłowej pracy będzie wymagał dołączenia niewielu elementów zewnętrznych. Jednak, gdy w budowanym układzie mamy do dyspozycji wolny przetwornik A/C oraz układ PWM, można pokusić się o samodzielne wykonanie przetwornicy.



Rysunek 3. Schemat ideowy przetwornicy z mikrokontrolerem STM32F100R6T6B

Mikrokontroler STM32F100R6T6B ma aż 5 układów czasowo – licznikowych, w tym jeden zawierający zaawansowany układ potrafiący generować komplementarny sygnał PWM ze strefą martwą, a więc aż „prosi się" o wykonanie przetwornicy z jego użyciem. Przykład takiej przetwornicy podwyższającej napięcie przedstawiono na **rysunku 3**.

Zaprezentowany układ jest zasilany z baterii AAA i istnieje w nim konieczność dostarczenia stabilnego napięcia o wartości +5 V o wydajności 150 mA do zasilania części analogowej. Z uwagi na stosunkowo dużą wydajność prądową, zdecydowano się na wykorzystanie tranzystora MOSFET, a zatem układ jest zdecydowanie bardziej skomplikowany od poprzedniego. Układ kluczujący stanowi tranzystor N-FET IRLL2805, który charakteryzuje się niewielką rezystancją kanału Rds(on)=55 mΩ, dużym dopuszczalnym prądem drenu Id=3,8 A, oraz co najważniejsze, niewielkim napięciem Vgs(th) = 1…2 V, które zapewnia rezystancję kanału bliską nominalnej już przy napięciu rzędu Vds = 4…5 V. W założeniach przyjęto, że minimalne napięcie zasilające będzie wynosiło 2 V, a więc aby zachować dobre parametry w pobliżu napięcia końcowego, konieczny jest dodatkowy układ sterujący oraz podwyższający napięcie.

Obwód z diodami D5 i D6 oraz kondensatorami C30…C32 stanowi typowy układ podwajający napięcie. Jest on zasilany sygnałem prostokątnym z wyjścia układu PWM1_CH2N układu czasowo-licznikowego TIM1. Zadaniem tego układu jest dostarczenie napięcia bramki w momencie startu przetwornicy. Gdy przetwornica już zostanie uruchomiona i na wyjściu będzie napięcie zbliżone do 5 V, sygnał prostokątny możemy wyłączyć, a napięcie zasilające układ sterownia bramką będzie dostarczane poprzez diodę D4 spolaryzowaną w kierunku przewodzenia.

Układ sterowania bramką stanowi komplementarna para tranzystorów T5-T6 stanowiąca typowy układ *push-pull*, którego zadaniem jest szybkie sterowanie bramką tranzystora Q6, tak aby zminimalizować straty na tranzystorze Q6 powstające w stanach pośrednich. Diody D7 oraz D8 zapobiegają przejściu tranzystorów T5-T6 w stan głębokiego nasycenia wydłużającego czas wyłączenia i stanowią one układ tranzystora schottky'ego. Do sterowania działaniem przetwornicy są konieczne dwa sygnały PWM oznaczone na schemacie XPOS oraz XNEG, komplementarne względem siebie, najlepiej mające przy tym funkcjonalność „strefy martwej", dającą czas na wyłączenie tranzystorów.

Układ STM32F100R6 ma zaawansowany układ czasowo – licznikowy T1 mający wspomnianą funkcjonalność, a więc idealnie nadaje się do realizacji przetwornicy zadania. Pomiar napięcia wyjściowego przetwornicy jest przeprowadzany z wykorzystaniem kanału dziesiątego 12-bitowego przetwornika ADC1, który jest dołączony do wyjścia za pomocą dzielnika składającego się z rezystorów R33 oraz R34. Jeden z rezystorów (R34) nie jest dołączony do masy, tylko do wyzerowanej linii PC1. Dzięki takiemu rozwiązaniu, jeżeli realizujemy programowy wyłącznik urządzenia ustawiając port GPIO w kierunku wejścia, możemy odłączyć dzielnik, tak aby nie pobierał dodatkowo prądu z baterii. Ten sposób podłączenia wiąże się także z drobnym błędem pomiaru napięcia wyjściowego związanym ze spadkiem napięcia na wewnętrznym tranzystorze MOS portu wynoszącym typowo kilkanaście mV.

Układ czasowo-licznikowy został skonfigurowany w trybie 9-bitowego sygnału PWM o częstotliwości 40 kHz. Przy obliczaniu parametrów przetwornicy przyjęto następujące założenia projektowe:

- Minimalne napięcie wejściowe: Vin_min = 2 V.
- Nominalne napięcie wyjściowe Vout = 5 V.
- Maksymalny prąd obciążenia Io = 150 mA.
- Dopuszczalny poziom tętnień napięcia na wyjściu Vripple=15 mV.
- Spadek napięcia na kluczu tranzystorowym Q3: Vds = 0,05 V.
- Spadek napięcia na diodzie D1: Vf = 0,3 V.
- Częstotliwość taktowania Fpwm = 40 kHz.

Obliczenia wykonano w analogiczny sposób, jak dla przetwornicy z poprzedniego przykładu, dlatego pominięto je w tym przykładzie.

Listing 4. Uruchomienie konwersji

```
//Start conversion
inline void stepup_converter::start_conv() {
  stm32::adc_software_start_conv_cmd( ADC1, true );
}
namespace tim {
namespace _internal {
  void stepup_converter_tick_handler()
  {
    if( p_conv_ptr )
    p_conv_ptr->start_conv();
  }
}}
```

Listing 5. Pojedynczy cykl pętli regulatora

```
namespace _internal {
extern "C" {
  void dma1_channel1_isr_vector(void) __attribute__((interrupt));
  void dma1_channel1_isr_vector(void) {
    stm32::dma_clear_flag( DMA1_FLAG_TC1);
    if( p_conv_ptr )
    p_conv_ptr->calculate_pwm_out();
  }
}}
//Calculate PWM output
inline void stepup_converter::calculate_pwm_out() {
  //Max allowed pwm
  static const int MAX_ALLOWED_PWM = 0.75 * MAX_PWM_VALUE;
  m_pwm_value += v2pwm( get_Vzas());
  if( m_pwm_value > MAX_ALLOWED_PWM ) m_pwm_value = MAX_ALLOWED_
PWM;
  else if( m_pwm_value < 0 ) m_pwm_value = 0;
  //Calculate output
  stm32::tim_set_ccr( TIM1, stm32::tim_cc_chn1, m_pwm_value );
}
```

Oprogramowanie

Oprogramowanie napisano w języku C++, a ogólną zasadę jego działania opisano w poprzednim przykładzie przetwornicy. W tym wypadku do sterowania wykorzystano zamiast regulatora „Fuzy-Logic" klasyczny regulator proporcjonalny. Cykl pętli regulatora, podobnie jak poprzednio, ustalono na wartość 1 ms i rozpoczyna się on od uruchomienia konwersji przetwornika A/C w przerwaniu od obsługi timera Systick rdzenia Cortex co zadany odcinek czasu (**listing 4**). Po zakończeniu konwersji i przetransferowaniu danych pomiarowych z przetwornika A/C kanałem DMA jest zgłaszane przerwanie od DMA, które realizuje pojedynczy cykl pętli regulatora. Odpowiednią procedurę pokazano na **listingu 5**.

Właściwa procedura realizowana jest przez metodę stepup_converter::calculate_pwm_out, która na początku, za pomocą metody v2pwm(), na podstawie aktualnego uchybu będącego różnicą pomiędzy wartością zmierzona a zadaną, wyznacza zmianę aktualnego współczynnika wypełnienia sygnału PWM. Następnie jest sprawdzane czy wyznaczony w ten sposób współczynnik wypełnienia mieści się w dopuszczalnym zakresie oraz ewentualnie jest on ograniczany. Po wyznaczeniu aktu-

alnego współczynnika PWM jest on zapisywany do rejestru CCR1, co powoduje zmianę wypełniania sygnału PWM na wyjściach PWM mikrokontrolera. Sama procedura obsługi jest bardzo krótka, a więc bez obaw może stanowić procedurę obsługi przerwania. Zdecydowanie dłuższa jest natomiast procedura inicjalizacji układu czasowo – licznikowego oraz przetwornika A/C, która jest realizowana w konstruktorze klasy stepup_converter::stepup_converter(), którego fragment odpowiedzialny za inicjalizację generowania sygnałów PWM przedstawiono na **listingu 6**.

Inicjalizacja układu rozpoczyna się od dołączenia sygnału zegarowego taktującego timer, a następnie w [2] jest konfigurowana podstawa czasowa, tak aby po zliczeniu do 512 licznik był automatycznie zerowany, co zapewni nam 9-bitową rozdzielczość sygnału PWM. W [3] jest konfigurowany kanał TIM1_CH2N, tak aby na wyjściu PB14 generowany był sygnał PWM o wypełnieniu 50% zasilający układ podwajający napięcie zasilające układ sterujący bramkę tranzystora Q3. W [4] konfigurowane są porty GPIO komplementarnych wyjść PWM, a następnie w [5] jest konfigurowany w roli generatora PWM kanał #1 układu compare – capture. W [6] jest włączany dodatkowy rejestr – cień, który powoduje, że nowa wartość sygnału PWM jest wprowadzana po zakończeniu pełnego cyklu, co zapobiega powstawaniu zniekształcania sygnału PWM w momencie wpisywania nowej wartości współczynnika wypełnienia. Następnie w [7] jest konfigurowany i włączany tryb komplementarny oraz strefa martwa. Na zakończenie, w [8] są uruchamiane wyjścia PWM oraz jest włączany sam układ czasowo – licznikowy.

Praktyczne próby wykazały, że średnia sprawność takiej przetwornicy przy zastosowaniu odpowiedniej cewki jest większa niż 80% i jest to całkiem rezultatem mającym sens. Dalsze zwiększanie sprawności przetwornicy wymagałoby zredukowanie strat powstających na diodzie D3, poprzez zastosowanie dodatkowego tranzystora MOSFET, co znacząco skomplikowałoby układ oraz sposób sterowania. Byłoby to raczej nieopłacalne. Gdy zależy nam na dalszym zwiększaniu sprawności, raczej należy skorzystać z gotowych rozwiązań fabrycznych.

Listing 6. Inicjalizowanie generatora sygnałów PWM

```
//Configure timer in pwm mode
stm32::rcc_periph_clock_cmd(stm32::rcc_clk_apb2, RCC_APB2Periph_
TIM1, true); [1]
//Configure timebase
stm32::tim_timebase_init(TIM1, 0, TIM_CounterMode_Up, MAX_PWM_
VALUE, TIM_CKD_DIV1, 0); [2]
//Configure OC2N output only XNEG PIN
stm32::tim_oc_init(TIM1, stm32::tim_cc_chn2, TIM_OCMode_PWM2, 256,
TIM_OutputState_Disable, TIM_OutputNState_Enable, TIM_OCPolarity_
Low,
TIM_OCNPolarity_Low, TIM_OCIdleState_Reset, TIM_OCNIdleState_
Reset); [3]
//Configure xdbl pin To PWM function
stm32::io_config( XDBL_PORT, XDBL_PIN, stm32::GPIO_MODE_50MHZ,
stm32::GPIO_CNF_ALT_PP );
//Configure the PWM main output
//Configure xpos, xneg pin To PWM function
stm32::io_config(XPOS_PORT, XPOS_PIN, stm32::GPIO_MODE_50MHZ,
stm32::GPIO_CNF_ALT_OD); [4]
stm32::io_config(XNEG_PORT, XNEG_PIN, stm32::GPIO_MODE_50MHZ,
stm32::GPIO_CNF_ALT_PP);
//Configure OC1/IC1N output XPOS, XNEG PIN
stm32::tim_oc_init(TIM1, stm32::tim_cc_chn1, TIM_OCMode_PWM1, 0,
[5]
TIM_OutputState_Enable, TIM_OutputNState_Enable, TIM_OCPolarity_
Low,
TIM_OCNPolarity_High, TIM_OCIdleState_Reset, TIM_OCNIdleState_Reset
);
//Enable preload register
stm32::tim_oc_preload_config( TIM1, stm32::tim_cc_chn1, true ); [6]
//Configure dead time
stm32::tim_bdtr_config( TIM1, PWM_DEAD_TIME, TIM_OSSRState_
Disable|TIM_OSSIState_Disable |TIM_LOCKLevel_1
|TIM_Break_Disable|TIM_BreakPolarity_Low|TIM_AutomaticOutput_
Enable); [7]
//Enable outputs and timer
stm32::tim_ctrl_pwm_outputs( TIM1, true ); [8]
stm32::tim_cmd( TIM1, true );
```

Zakończenie

W niniejszym artykule starałem się wykazać, że mimo iż istnieje wiele tanich i gotowych układów przetwornic, czasem warto skorzystać z realizacji własnej, bezpośrednio wykorzystując do tego celu mikrokontroler mający trochę zasobów sprzętowych i nieco wolnej mocy obliczeniowej. Czasem zastosowanie dodatkowego układu (jak w przykładzie z ATtiny13) znacząco komplikuje układ i jest nieopłacalne. W innych przypadkach, takich jak w przykładzie z mikrokontrolerem STM32F100, układ ma dodatkowe, nieużywane w aplikacji bloki sprzętowe i szkoda było pozostawić je nieużywane.

Lucjan Bryndza, EP