Zaawansowane użycie timerów w STM32F100

Pomiarów częstotliwości z użyciem timerów

W systemach mikroprocesorowych oprócz pomiaru wielkości analogowych np. za pomocą przetwornika A/C, czasem zachodzi konieczność pomiaru częstotliwości, okresu lub wypełnienia. Wtedy z pomocą przychodzą nam układy czasowo-licznikowe, które dostępne są nawet w najprostszych mikrokontrolerach. Mikrokontrolery z rodziny STM32 mają 16-bitowe układy czasowo-licznikowe, które mogą być ze sobą łączone m.in. w trybie bramkowania. Umożliwia to wykonanie dokładnego pomiaru częstotliwości bez konieczności dodatkowych połączeń na zewnątrz układu.

Aby uzyskać odpowiednią dokładność pomiaru, jest konieczne zapewnienie jak najdokładniejszego czasu otwarcia bramki, a więc realizacja w sposób programowy nie jest wskazana. W wypadku prostych mikrokontrolerów realizacja sprzętowa wymaga bramki oraz dodatkowych połączeń na zewnątrz, co w niekiedy bywa kłopotliwe. Mikrokontrolery z rodziny STM32 mają 16-bitowe układy czasowo-licznikowe, które mogą być ze sobą łączone m.in. w trybie bramkowania, co umożliwia realizację dokładnego pomiaru częstotliwości, bez konieczności wykonywania dodatkowych połączeń poza obudową układu scalonego. Do realizacji zadania potrzebować będziemy dwóch układów czasowo-licznikowych, z których jeden będzie służył do odliczania czasu otwarcia bramki, natomiast drugi do zliczania impulsów zewnętrznych.

Najstarsze mikrokontrolery z rodziny STM32 np. F101/F103 mają 2 a w większych wersjach 3 układy czasowo-licznikowe, co często okazywało się niewystarczające. Nowsze układy z rodziny ekonomicznej STM32F100, nawet w najprostszej wersji mają 5 podstawowych oraz jeden zaawansowany układ czasowo-licznikowy, a więc będą idealne dla takich pomiarów z uwagi na dużą liczbę timerów oraz bardzo niską cenę, konkurencyjną w stosunku do najprostszych rozwiązań 8-bitowych.

Wiadomości wstępne

Budowę układów czasowo-licznikowych T2…T5 układów STM32F100 przedstawiono na **rysunku 1**. Standardowy układ czasowo-licznikowy składa się z 16-bitowego licznika głównego CNT oraz 4 układów przechwytująco-porównujących (*Compare-Capture*). Licznik CNT może zliczać w górę lub w dół impulsy wewnętrzne oraz impulsy zewnętrzne podawane na wejście ETR. Dodatkowo, do dyspozycji mamy 16-bitowy preskaler PSC zapewniający podział impulsów zliczanych przez liczbę z zakresu 1…65536. Długość cyklu licznika może być skrócona z wykorzystaniem rejestru ARR. Układ wyzwalania *(Trigger Control)*, zapewnia zaawansowane tryby wyzwalania, z zewnątrz za pomocą wejścia TIMx_CH1, lub z wykorzystaniem innych liczników za pośrednictwem wejść ITR0…ITR3. Układ wewnętrznego wyzwalania umożliwia:

- Bramkowanie licznika za pomocą innego licznika.
- Wyzwolenie licznika za pomocą innego licznika.
- Użycie jednego licznika jako preskalera dla drugiego licznika.

Równoczesne uruchomienie dwóch liczników w wyniku wystąpienia zdarzenia zewnętrznego.

Wejścia wyzwalania ITR0…ITR3 są wewnętrznie połączone z wyjściami wyzwalania TRGO innych liczników w sposób ściśle określony, a więc nie jest to rozwiązanie uniwersalne i w szczególności nie ma możliwości dowolnego łączenia timerów. Sposób połączenia wyjść wyzwalania TRGO z wejściami ITR0…ITR3 umieszczono w **tabeli 1**.

Układy T2…T5 mają 4 bloki przechwytująco-porównujące, które mogą być wykorzystane do generowania sygnału prostokątnego, generowania sygnału PWM, przechwytywania stanu licznika, czy pomiaru szerokości impulsów PWM. Wyjście TRGO timera jest konfigurowalne umożliwiając wybór odpowiedniego sygnału z danego bloku, gdzie możliwości konfiguracji są następujące:

- Bit UG, aktywuje linię TRGO – sygnał uruchomienia układu czasowo-licznikowego CNT_EN aktywuje linię TRGO, co jest użyteczne w wypadku konieczności rozpoczęcia równoczesnego odliczania przez dwa timery.

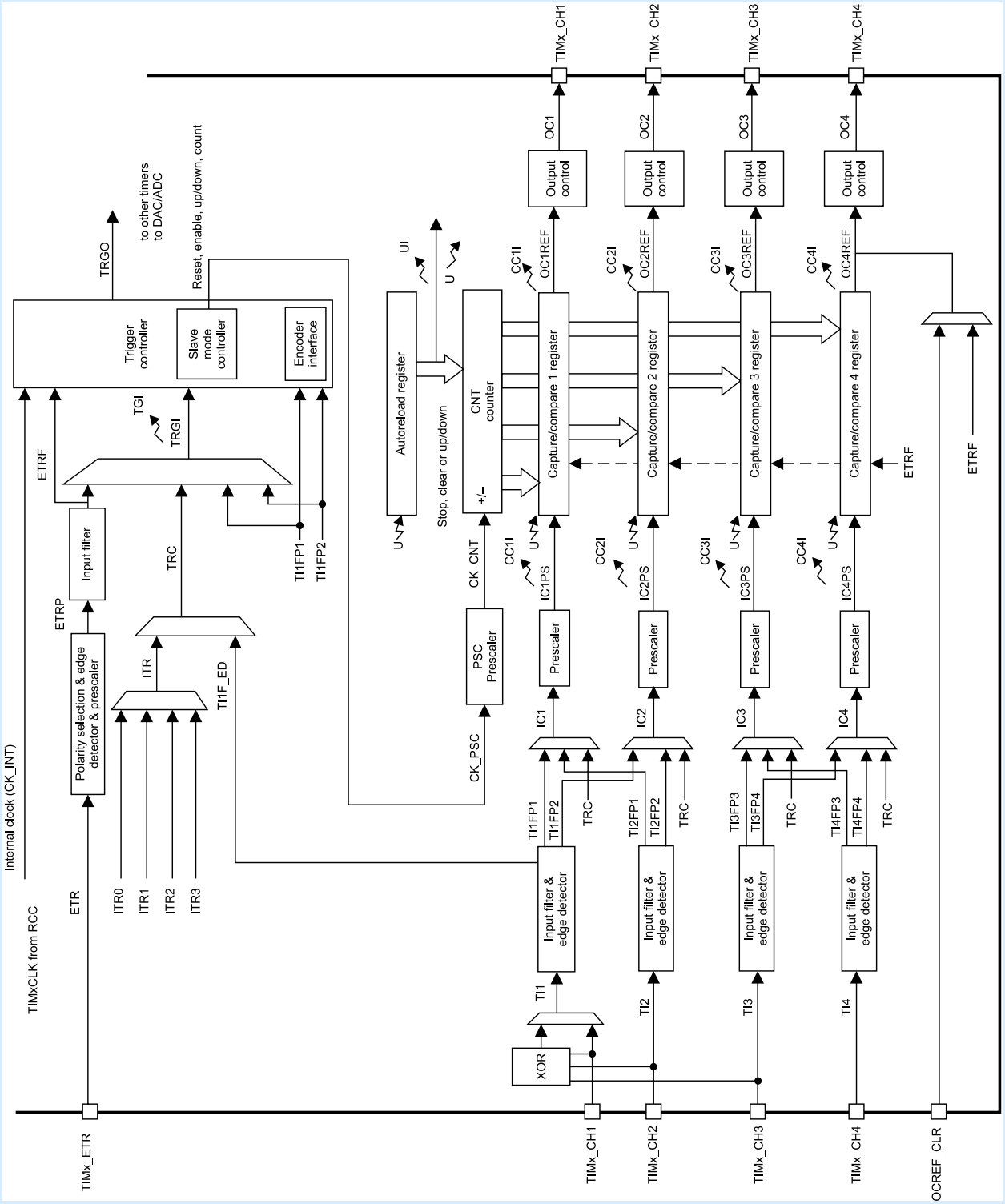
Dodatkowe materiały na CD/FTP:
Kompletny kod źródłowy klasy do pomiaru częstotliwości można pobrać z adresu: *ftp://boff.pl/elektronika_praktyczna/frequency-measure/ep-stm32-tim-freq-measure.tar.gz*. Przykład korzysta z biblioteki standardowej STM32 dla systemu ISIXRTOS, którą można pobrać ze strony domowej projektu.

Tabela 1. Sposoby łączenia wyjść wyzwalania TRGO z wejściami ITR0…ITR3

TRGO	ITR0	ITR1	ITR2	ITR3
TIM1	TIM15	TIM2	TIM3	TIM4
TIM2	TIM1	TIM15	TIM3	TIM4
TIM3	TIM1	TIM2	TIM15	TIM4
TIM4	TIM1	TIM2	TIM3	TIM15

- Zdarzenie doliczenia do wartości maksymalnej (określonej w rejestrze ARR), aktywuje linię TRGO *(update event)*.
- Wyjście wybranego bloku 1…4 porównującego *(Compare)*, odzwierciedla stan linii TRGO.

Aby uniknąć niejednoznaczności pomiaru wynikającego z programowej obsługi bramkowania należy wybrać tryb wyzwalania ze sprzętowym bramkowaniem jednego licznika za pomocą drugiego licznika, wówczas jeden układ będzie służył do odmierzania czasu otwarcia bramki, natomiast drugi układ będzie zliczał impulsy zewnętrzne na wejściu TIMx_ETR. Do odmierzania czasu otwarcia bramki najwygodniej będzie użyć sygnału jednego z kanałów porównujących *(Compare)*, co umożliwi ewentualne zastosowanie pozostałych bloków do innych celów. Zmieniając wartości wpisane do rejestru porównującego CCR będziemy mogli określić czas otwarcia bramki, a tym samym liczbę impulsów zewnętrznych zliczonych w danej jednostce czasu. Maksymalna częstotliwość impulsów zewnętrznych zliczanych przez układ timera jest równa częstotliwości taktowania układu podzielonej przez dwa (*Fmax = Fclk/2*). Ponieważ układy STM32F100 mogą być taktowane jedynie sygnałem o częstotliwości do 24 MHz, maksymalna częstotliwość impulsów zewnętrznych, którą układ będzie w stanie zmierzyć wynosi 12 MHz.



Rysunek 1. Budowa układów czasowo-licznikowych w STM32F100

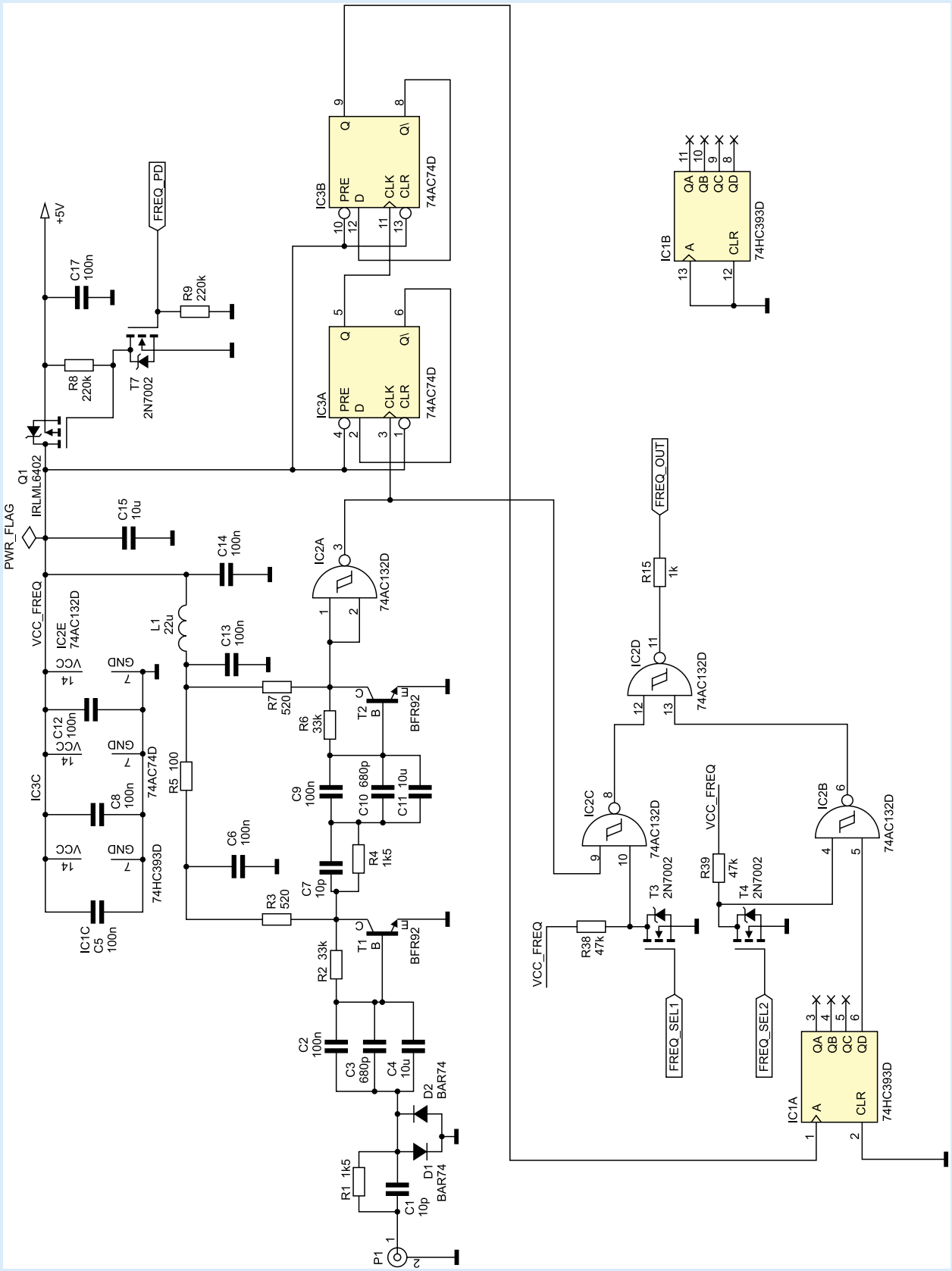
Praktyczny przykład pomiaru częstotliwości

Układ wejściowy

Aby pokazać praktyczny sposób użycia układów czasowo-licznikowych STM32F100 przedstawimy przykład prostego miernika częstotliwości o następujących parametrach:

- częstotliwość sygnału mierzonego: 50 Hz…200 MHz,
- minimalne napięcie wejściowe 40 mVrms.

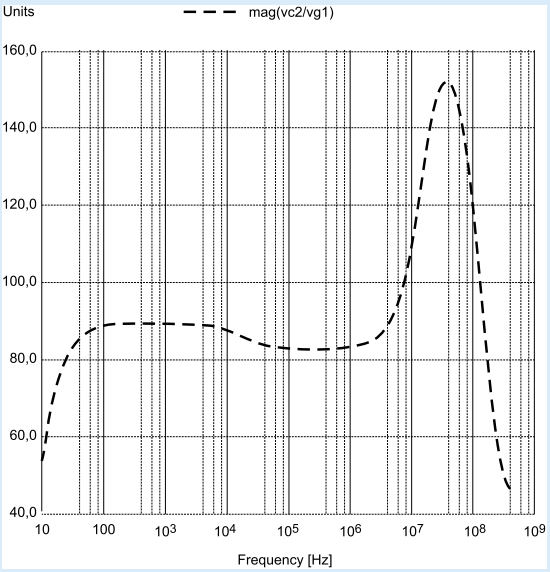
Jak wspomniano, maksymalna częstotliwość impulsów zewnętrznych zliczanych przez wejście ETR timera wynosi 12 MHz, a ponadto ich kształt i amplituda powinny być zgodne ze standardem CMOS. Konieczne jest zatem zbudowanie zewnętrznego układu wejściowego,



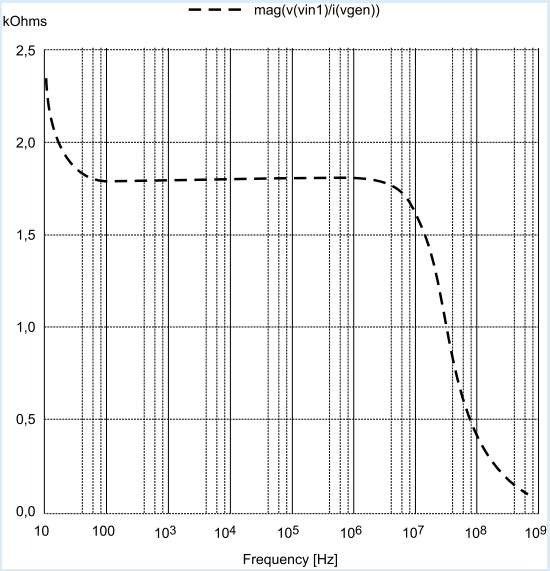
Rysunek 2. Schemat proponowanego rozwiązania obwodów wejściowych

który będzie zawierał wzmacniacz wejściowy oraz dzielnik częstotliwości. Schemat takiego układu wejściowego przedstawiono na **rysunku 2**.

Układ zbudowano bez specjalizowanych układów, z wykorzystaniem powszechnie dostępnych elementów, które możemy znaleźć w przysłowiowej szufladzie. Wzmacniacz wejściowy zrealizowano jako typowy, dwustopniowy wzmacniacz OE ze sprzężeniem pojemnościowym na tranzystorach T1 i T2. Diody D1 oraz D2 są układem ogranicznika napięcia zapobiegającym przechodzeniu tranzystora wejściowego T1 w stan nasycenia. O minimalnej częstotliwości wejściowej wzmacniacza decydują sprzężenia pojemnościowe pomiędzy stopniami. Sprzężenia pojemnościowe stanowią równolegle połączone kondensatory o pojemności 680 pF (NP0), 100 nF (X7R), 10 µF (Y5V), dzięki czemu wypadkowa reaktancja pojemnościowa zachowuje swoje właściwości w szerokim zakresie częstotliwości. O górnym zakresie częstotliwości decydują głównie parametry tranzystorów T1 i T2. Stosując tranzystory BFR92, które charakteryzują się częstotliwością graniczną *Ft=5 GHz*, udało się uzyskać założone pasmo przenoszenia 200 MHz, przy założonej czułości 40 mV. Charakterystykę amplitudową wzmacniacza przedstawiono na **rysunku 3**.



Rysunek 3. Charakterystyka amplitudowo-częstotliwościowa wzmacniacza wejściowego



Rysunek 4. Charakterystyka fazowo-częstotliwościowa wzmacniacza wejściowego

Z uwagi na zastosowanie na wejściu tranzystora bipolarnego, wzmacniacz posiada stosunkowo małą impedancję wejściową, jednak w większości zastosowań okazuje się ona wystarczająca, a wzmacniacz dzięki temu zbiera mniej zakłóceń z otoczenia. Charakterystykę przedstawiającą impedancję wejściową wzmacniacza wejściowego w funkcji częstotliwości przedstawiono na **rysunku 4**.

Impedancja wejściowa wzmacniacza wynosi około 1,8 kΩ dla częstotliwości do około 10 MHz po czym opada, by przy krańcu pasma osiągnąć wartość około 150 Ω. Napięcie z wyjścia wzmacniacza kierowane jest do bramki Schmitta IC2A (74AC132), której zadaniem jest przekształcenie sygnału wyjściowego na sygnał prostokątny. Sygnał ten następnie jest kierowany na układ dzielnika częstotliwości przez 4 zrealizowany za pomocą dwóch przerzutników D układu 74AC74. Układ ten jest jednym z najszybszych układów z tej serii i powinien pracować poprawnie do częstotliwości ponad 200 MHz. Sygnał z wyjścia dzielnika wstępnego IC3 jest podawany następnie do licznika IC1A, który realizuje dodatkowy podział przez 16. Układ IC3 jest już zwykłym układem serii HC, ponieważ częstotliwość na wyjściu dzielnika AC74 nie przekracza 60 MHz. W wyniku tak złożonego układu dzielników uzyskujemy możliwość podziału przez 64, dzięki czemu mikrokontroler bez problemu poradzi sobie z pomiarem, gdyż częstotliwość na wyjściu układu nie będzie przekraczać 4 MHz. Z pozostałych bramek układu IC2 (IC2B, IC2C, IC2D) wykonano układ kombinacyjny umożliwiający, podanie sygnału wyjściowego bezpośrednio z bramki IC2A lub za pośrednictwem dzielnika, co umożliwia zwiększenie dokładności w niższym zakresie pomiarowym. Blok dzielnika i wzmacniacza jest zasilany napięciem 5 V. W stanie spoczynkowym układ pobiera stosunkowo duży prąd, dlatego cały blok może być

Listing 1. Fragment deklaracji klasy *tim_freq_measure*

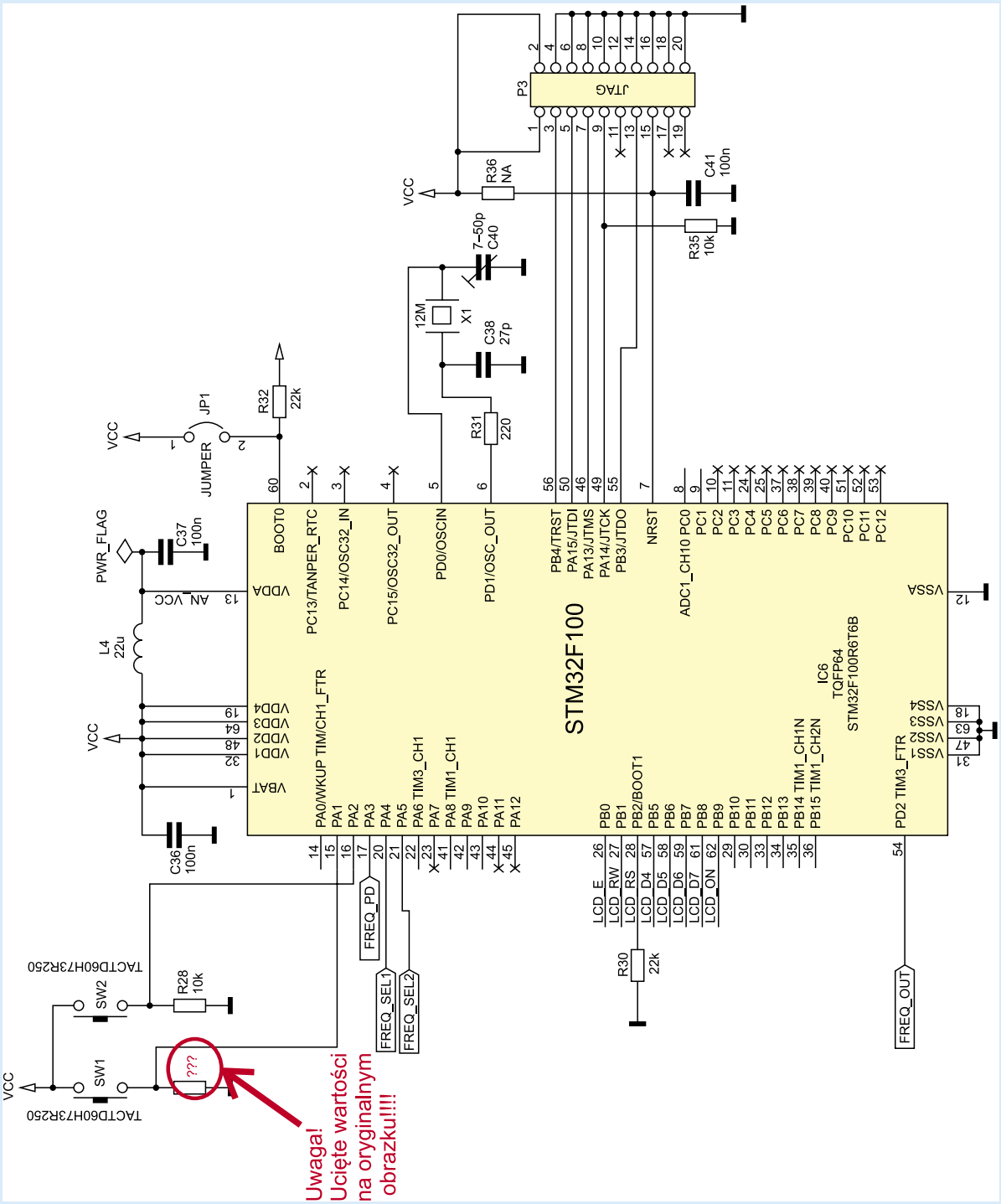
```
class tim_freq_measure : private fnd::noncopyable
{
  friend void _internal::tim2_isr_vector(void);
  friend void _internal::tim3_isr_vector(void);
  friend void _internal::tim1_brk_tim15_isr_
vector(void);
private:
  //Timer base period for prescaler
  static const unsigned long BASE_PERIOD = 1000;
  static const unsigned long GATE_TIME_MS = 250;
  static const unsigned long GATE_NOT_ACTIVE_MS = 5;
private:
  //Handle overflow IRQ
  void handle_overflow();
  //Handle measure finished
  void handle_measure_finished();
public:
  //No measure statement
  static const unsigned long NO_NEW_DATA = 0xFFFFFFFF;
  static const unsigned long RAW_DATA_DIVIDE = SECOND_C/
GATE_TIME_MS;
  /** Construct for measure timer
  * @param tim Used timer
  */
  explicit tim_freq_measure( TIM_TypeDef * tim );
  //Destructor
  ~tim_freq_measure();
  //Frequency
  unsigned long operator()()
  {
    unsigned long value = stm32::atomic_xchg_word( &m_
capture, NO_NEW_DATA );
    if( value != NO_NEW_DATA ) value *= (SECOND_C/GATE_
TIME_MS);
    return  value;
  }
private:  //Temporary
  TIM_TypeDef * const m_tim;   //Timer pointer
  volatile unsigned short m_hi_cnt;    //Hi counter
  volatile unsigned long  m_capture;
};
```

wyłączony programowo dzięki układowi zrealizowanym na tranzystorach Q1 i T7, co może być użyte w celu oszczędzania energii, jeśli układ jest zasilany z baterii. Ponieważ założono, że mikrokontroler STM32F100 jest zasilany bezpośrednio z baterii, tranzystory T3 i T4 pełnią zadanie przesuwnika poziomu napięć. Sposób podłączenia układu wejściowego do mikrokontrolera STM32F100 przedstawiono na **rysunku 5**.

Jest to klasyczna aplikacja mikrokontrolera STM32F100, jedyną różnicą jest zastosowanie w bloku generatora kwarcowego trymera zamiast zwykłego kondensatora. Trymer umożliwia dokładne ustalenie częstotliwości generatora, ponieważ głównie od niego zależy dokładność pomiaru częstotliwości. Aby uzyskać jeszcze lepszą dokładność należałoby zastosować zewnętrzny układ generatora sygnału zegarowego. Wejścia sterujące wyborem podziału częstotliwości FREQ_SEL1 i FREQ_SEL2 oraz wejście sterujące zasilaniem bloku wejściowego FREQ_PD podłączono do portów GPIO, które zostały skonfigurowane w kierunku wyjściowym. Wyjście sygnału FREQ_OUT zostało podłączone do linii PD2, która stanowi wejście sygnału impulsów zewnętrznych ETR układu czasowo-licznikowego TIM3. Do linii portu PB dołączono wyświetlacz LCD 2×16 umożliwiający wyświetlanie zmierzonej częstotliwości.

Oprogramowanie

Ponieważ układ czasowo-licznikowy TIM3 wybraliśmy do zliczania impulsów zewnętrznych, jako podstawę czasu należy wybrać jeden z układów czasowo-licznikowych



Rysunek 5. Sposób podłączenia układu wejściowego do mikrokontrolera STM32F100

zgodnie z tabelą (n). W tym przypadku zdecydowano się wykorzystać układ TIM15, który jest uproszczoną wersją układów ogólnego przeznaczenia pozbawionych, dwóch bloków przechwytująco porównujących CCR3 oraz CCR4. Wybranie tego układu umożliwia wykorzystanie TIM15 do równoczesnego bramkowania układów TIM2 i TIM3.

Oprogramowanie zostało napisane w języku C++ , z wykorzystaniem kompilatora GCC, za bezpośredni pomiar częstotliwości odpowiada klasa *tim_freq_measure*, której fragment deklaracji przedstawiono na **listingu 1**.

Klasa została zaprojektowana w taki sposób aby istniała możliwość pomiaru częstotliwości z wejścia ETR układu TIM2 oraz TIM3. Wybór timera możliwy jest podczas tworzenia obiektu klasy poprzez konsruktor tim_freq_measure(TIM_TypeDef * tim); który jako argument przyjmuje wskaźnik do struktury TIM2, lub TIM3. Utworzenie dwóch obiektów w aplikacji jednego z parametrem TIM2, a drugiego z parametrem TIM3 umożliwia wspólne wykorzystanie TIM15 do bramkowania obu układów. Procedura do pomiaru częstotliwości jest realizowana w sposób całkowicie automatyczny, poprzez procedury obsługi przerwań układów czasowo-licznikowych. Pomiary wykonywane są cyklicznie z czasem otwarcia bramki określonym przez stałą GATE_TIME_MS oraz czasem zamknięcia bramki określonej przez stałą GATE_NOT_ACTIVE_MS. W tym wypadku pomiary wykonywane są z czasem otwarcia bramki 250 ms i 5 ms przerwy pomiędzy pomiarami, co zdaniem autora zapewnia optymalny stosunek dokładności pomiaru (4 Hz)

Listing 2. Inicjalizacja układów czasowo-licznikowych

```
//Timer freq measure
tim_freq_measure::tim_freq_measure( TIM_TypeDef * tim )
  : m_tim(tim), m_hi_cnt(0), m_capture(0)
{
  /* ***************** Configure gate timer (TIM15) ***************** */
  if( !p_tim2 && !p_tim3 )        //Not initialized devs
  {
    //Enable timer
    stm32::rcc_periph_clock_cmd(stm32::rcc_clk_apb2, RCC_APB2Periph_TIM15, true );
    //Configure time base
    stm32::tim_timebase_init(TIM15, config::PCLK2_HZ / BASE_PERIOD - 1, TIM_CounterMode_Up, GATE_TIME_MS +
GATE_NOT_ACTIVE_MS, 0, 0); //[2]
    //Select master slave mode
    stm32::tim_select_master_slave_mode( TIM15, TIM_MasterSlaveMode_Enable );
    //Configure OC1 as gate for slave timer
    stm32::tim_oc_init( TIM15, stm32::tim_cc_chn1, TIM_OCMode_PWM1, GATE_TIME_MS, TIM_OutputState_Disable,
TIM_OutputNState_Disable,
TIM_OCPolarity_Low, TIM_OCNPolarity_Low, TIM_OCIdleState_Reset,
TIM_OCNIdleState_Reset ); //[3]
    //Select trigger output
    stm32::tim_select_output_trigger( TIM15, TIM_TRGOSource_OC1Ref); //[4]
    //Configure NVIC interrupt
    stm32::nvic_set_priority( TIM1_BRK_TIM15_IRQn, IRQ_PRIO, IRQ_SUB );
    stm32::nvic_irq_enable( TIM1_BRK_TIM15_IRQn, true );
    stm32::tim_it_config( TIM15, TIM_IT_CC1, true );
    //Enable the timer
    stm32::tim_cmd( TIM15, true); //[5]
  }
  /* ***************** Configure TIM2 or TIM3 ***************** */
  if( m_tim == TIM2 )     //[6]
  {
    if( p_tim2 ) return;
    //Enable tim2 clock
    stm32::rcc_periph_clock_cmd( stm32::rcc_clk_apb1, RCC_APB1Periph_TIM2, true );
    stm32::io_config( FREQ1_PORT, FREQ1_PIN, stm32::GPIO_MODE_INPUT, stm32::GPIO_CNF_IN_FLOAT );
    p_tim2 = this;
  }
  else if( m_tim == TIM3)
  {
    if( p_tim3 ) return;
    //Enable tim3 clock
    stm32::rcc_periph_clock_cmd( stm32::rcc_clk_apb1, RCC_APB1Periph_TIM3, true );
    stm32::io_config( FREQ2_PORT, FREQ2_PIN, stm32::GPIO_MODE_INPUT, stm32::GPIO_CNF_IN_FLOAT );
    p_tim3 = this;
  }
  else
  {
    return;
  }
  //Initialize the timebase
  stm32::tim_timebase_init(m_tim, 0, TIM_CounterMode_Up, 65535, 0, 0 ); //[7]
  //Select slave mode
  stm32::tim_select_slave_mode(m_tim, TIM_SlaveMode_Gated ); //[8]
  //Select input trigger iówn slave mode
  stm32::tim_select_input_trigger( m_tim, (m_tim==TIM2)?(TIM_TS_ITR1):(TIM_TS_ITR2) );
  //Select external trigger //[9]
  stm32::tim_etr_clock_mode2_config( m_tim, TIM_ExtTRGPSC_OFF, TIM_ExtTRGPolarity_NonInverted, 0 );
  //Configure Capture //[10]
  stm32::tim_it_config( m_tim, TIM_IT_Update, true );
  //Configure NVIC interrupt
  stm32::nvic_set_priority( m_tim==TIM2?TIM2_IRQn:TIM3_IRQn, IRQ_PRIO, IRQ_SUB );
  stm32::nvic_irq_enable( m_tim==TIM2?TIM2_IRQn:TIM3_IRQn, true );
  //enable timer
  stm32::tim_cmd( m_tim, true );
}
```

Listing 3. Metoda tim_freq_measure::handle_overflow()

```
//Handle interrupt
void tim_freq_measure::handle_overflow()
{
  if( stm32::tim_get_it_status(m_tim, TIM_IT_Update ) )
  {
    stm32::tim_clear_it_pending_bit( m_tim, TIM_IT_Update );
    m_hi_cnt++;
  }
}
```

Listing 4. Procedura obsługi przerwania TIM15

```
void tim1_brk_tim15_isr_vector(void)
{
  if( stm32::tim_get_it_status(TIM15, TIM_IT_CC1 ) )
  {
    stm32::tim_clear_it_pending_bit( TIM15, TIM_IT_CC1 );
    if( p_tim2 ) p_tim2->handle_measure_finished();
    if ( p_tim3 ) p_tim3->handle_measure_finished();
  }
}
```

Procedura zeruje flagę zgłoszenia przerwania, a następnie wywołuje metodę handle_measure_finished na rzecz obiektu układu czasowo-licznikowego TIM2 lub/i TIM3 (**listing 5**).

Listing 5. Metoda handle_measure_finished

```
//Handle measure finished
void tim_freq_measure::handle_measure_finished()
{
  uint32_t capture =  (static_cast<uint32_t>(m_hi_cnt)<<16)| stm32::tim_get_counter( m_tim );
  stm32::atomic_try_write_word( &m_capture, capture );
  m_hi_cnt = 0;
  stm32::tim_set_counter( m_tim, 0 );
}
```

Listing 6. Przykład użycia klasy

```
int measure()
{
  dev::tim_freq_measure fm3obj(TIM3);
  //Enable input PWR
  stm32::io_set( FREQ_PORT , FREQ_PWR );
  //Set divide 64
  stm32::io_set( FREQ_PORT , FREQ_SEL1_PIN );
  stm32::io_clr( FREQ_PORT , FREQ_SEL2_PIN );
  for(;;)
  {
    unsigned long curr_f = fm3obj();
    if( curr_f != dev::tim_freq_measure::NO_NEW_DATA )
    {
      //Display frequency
      fnd::tiny_printf(„F=%u”, curr_f * 64);
    }
    //Wait for interrupt
    stm32::wfi();
  }
}
```

do częstotliwości odświeżania wyniku na wyświetlaczu. Pomiar jest realizowany cyklicznie w procedurach obsługi przerwań, natomiast pętla główna programu powinna wywoływać przeciążony unsigned long operator()() w celu odczytania wyniku pomiaru. Jeżeli operator zwraca wartość NO_NEW_DATA, oznacza, że nie ma nowego wyniku pomiaru, w przeciwnym przepadku zwracana jest aktualnie zmierzona częstotliwość. Operator () wykorzystuje funkcję atomic_xchg_word(), co umożliwia odczytanie i zmianę zmiennej m_capture aktualizowanej przez procedurę obsługi przerwania bez konieczności blokowania przerwań, dzięki wykorzystaniu sprzętowych instrukcji rdzenia CORTEX-M3 LDREX oraz STREX. Inicjalizacja układów czasowo-licznikowych realizowana jest w konstruktorze klasy, którego implementację przedstawiono na listingu 2.

Inicjalizacja rozpoczyna się od konfigurowania układu podstawy czasu bramowania, która jest realizowana przez TIM15. Na początku jest włączany sygnał zegarowy dla układu, a następnie w [2] jest konfigurowana podstawa czasu, tak aby cykl pomiarowy był równy czasowi otwarcia bramki powiększonemu o czas przerwy (GATE_TIME_MS + GATE_NOT_ACTIVE_MS). Kolejną czynnością jest ustawienie układu w tryb Master/Slave umożliwiający sterowanie innymi układami peryferyjnymi. Następnie w [3] kanał CCR1 jest konfigurowany w taki sposób, aby na wyjściu porównującym był generowany sygnał o współczynniku wypełnienia (GATE_TIME_MS)/(GATE_TIME_MS + GATE_NOT_ACTIVE_MS). W [4] jako sygnał wyjścia wyzwalania TRGO jest wybierane wyjście układu porównującego CCR1, co spowoduje otwarcie bramki timera podrzędnego przez czas GATE_TIME_MS. W [5] jest włączane przerwanie od układu porównującego CCR1, a na koniec jest włączany układ TIM15. Po zakończeniu inicjalizacji timera podstawy czasu przystępujemy do konfigurowania wybranego timera, którego zadaniem jest zliczanie impulsów zewnętrznych na wejściu ETR. W tym celu w [6] jest włączany sygnał zegarowy dla wybranego układu oraz port GPIO przypisany do wejścia ETR jest konfigurowany w taki sposób, aby pełnił rolę wejścia peryferyjnego. W [7] następuje konfigurowanie układu, tak aby zliczał impulsy w zakresie 0…65535. W [8] jest ustawiana praca układu czasowo-licznikowego w trybie bramkowania oraz jest wybierane odpowiednie wejście ITR układu nadrzędnego, tak aby wskazywało ono na TIM15. W [9] układ czasowo-licznikowy jest konfigurowany, tak aby zliczał zewnętrzne impulsy z wejścia ETR. W [10] uruchamiane są przerwania od przepełnienia licznika wykorzystane do programowego zwiększenie długości licznika do 32 bitów. Obsługa przerwania od przepełnienia licznika TIM2 lub TIM3 jest realizowana przez metodę tim_freq_measure::handle_overflow(), której kod przedstawiono na **listingu 3**.

Dzianie metody sprowadza się do wyzerowania, flagi zgłoszenia przerwania, oraz zwiększenia zmiennej m_hi_cnt, która zawiera starsze 16 bitów licznika realizowanego w sposób programowy. Wartość zmierzonej częstotliwości wpisywana jest do zmiennej m_capture przez procedurę obsługi przerwania układu porównującego CCR1 licznika TIM15. Gdy bramka zostaje zamknięta wywoływana jest procedura obsługi przerwania (**listing 4**).

Działanie tej metody jest trywialne i sprowadza się do odczytania młodszej (sprzętowej) oraz starszej (programowej, zmienna m_hi_cnt) połówki licznika, które po przeliczeniu będą stanowić wartość zmierzonej częstotliwości. Następnie odczytana wartość częstotliwości zapisywana jest do zmiennej m_capture za pomocą funkcji zapisu atomowego stm32::atomic_try_write_word(). Metoda pomiaru częstotliwości jest realizowana w dużej mierze automatycznie stanowiąc niewielkie obciążenie dla procesora. Przykład użycia zaprezentowanej wcześniej klasy przedstawiono na **listingu 6**.

Na początku tworzony jest obiekt klasy pomiaru dla układu TIM3, włączane zasilanie bloku wejściowego oraz włączany dzielnik 1:64, a następnie program wchodzi do pętli nieskończonej, gdzie jest wywoływany operator(), który zwraca wynik pomiaru. Gdy zwróci on wartość różną od dev::tim_freq_measure::NO_NEW_DATA oznacza to, że pomiar częstotliwości zakończył się, w związku z czym jest on wyświetlony, po czym wywoływana jest funkcja stm32::wfi(), która przełącza rdzeń mikrokontrolera w stan uśpienia w oczekiwaniu na nadejście przerwania.

Lucjan Bryndza, EP