32 bity jak najprościej – STM32F0 (2)

Pomiar napięcia i temperatury z uwzględnieniem danych kalibracyjnych

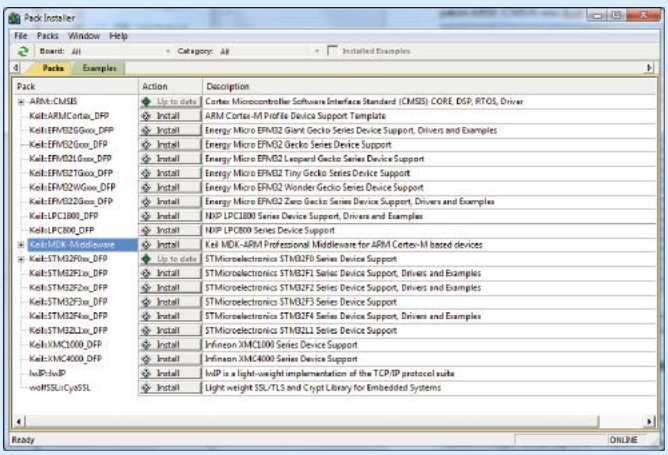
Latem ubiegłego roku przedstawiliśmy serię artykułów poświęconych początkom tworzenia oprogramowania dla mikrokontrolerów 32-bitowych z rdzeniami Cortex na przykładzie prostych projektów działających na płytce STM32F0 Discovery. Pozytywny odzew Czytelników spowodował powstanie drugiej serii artykułów.

Przykłady prezentowane w tej serii mogą być uruchamiane na mikrokontrolerach z rodziny STM32F05x oraz „ekonomicznej" – STM32F030. Strukturę kodu źródłowego nowych przekładów tak zmodyfikowano, aby definicje używane we wszystkich przykładach były zgrupowane w plikach niezależnych od konkretnego projektu. Do kompilowania programów użyto nowej wersji środowiska Keil MDK-ARM – 5.00. Nowe projekty oraz przystosowane do nowej wersji MDK-ARM projekty z poprzedniej serii artykułów są zawarte w materiałach dodatkowych, w pliku *F0tutorial2.zip*.

Przygotowanie środowiska Keil MDK-ARM 5.x

W poprzedniej serii artykułów wprowadzających do programowania mikrokontrolerów STM32F0 korzystaliśmy ze środowiska Keil MDK-ARM w wersji 4.x. Obecnie jest dostępna wersja 5, dlatego kolejne przykłady zostały przygotowane przy jej użyciu.

Instalowanie środowiska Keil w wersji 5.0 jest podobne do znanego z wersji 4.x, jednak została wprowadzona jedna istotna zmiana – pakiet instalacyjny nie zawiera plików związanych z konkretnymi mikrokontrolerami, więc przed rozpoczęciem pracy należy je doinstalować korzystając z przycisku *Pack Installer* (**rysunek 1**) umieszczonego na końcu paska narzędzi. Jego naciśnięcie powoduje otwarcie połączenia z serwerem Keil i wyświetlenie listy dostępnych pakietów. Do podstawowej pracy z STM32F0 należy zainstalować pakiet ARM::CMSIS oraz Keil::STM32F0_DFP.



Rysunek 1. Okno główne instalatora pakietów

Pliki wspólne dla wszystkich projektów

W celu podniesienia czytelności programów i ułatwienia tworzenia kolejnych projektów definicje używane we wszystkich nowych projektach zgrupowano w nowych plikach nagłówkowych. W głównym folderze roboczym utworzono folder *Common*, zawierający wszystkie pliki nagłówkowe.

Jedynym plikiem włączanym do wszystkich programów jest plik *stm32f0discovery.h*. Zawiera on definicje portów dla płytek STM32F0DISCOVERY i STM32F0308DISCO. Włącza on również dwa kolejne pliki:

- stm32f0yy.h, włączający plik stm32f0xx.h i zawierający definicje lokacji w przestrzeni adresowej oraz wartości pól rejestrów mikrokontrolera, które nie zostały zdefiniowane w dostarczonym przez producenta pliku stm32f0xx.h;
- stm32futil.h, zawierający użyteczne definicje do tworzenia pól bitowych oraz deklaracje struktury i procedury inline używanej do inicjowania peryferiali. Zawartość plików pokazano na **listingach 1**…**3**.

Korzystanie z wymienionych plików nagłówkowych w projektach wymaga dodania w opcjach projektu do ścieżki poszukiwania plików nagłówkowych foldera Common (**rysunek 2**).

Rysunek 2. Okno opcji kompilatora

Przykłady

1. Pomiar napięcia i temperatury z uwzględnieniem danych kalibracyjnych

Mikrokontrolery z serii STM32F0 są wyposażone w wewnętrzne źródło napięcia wzorcowego oraz czujnik temperatury. Oba te moduły charakteryzują się znacznym rozrzutem parametrów, dlatego producent podczas testów fabrycznych zapisuje w pamięci stałej mikrokontrolera ich dane kalibracyjne. Na podstawie danych kalibracyjnych można przeliczyć odczyty przetwornika analogowo – cyfrowego na wartość napięcia i temperatury. Źródło napięcia wzorcowego jest dołączone do wejścia przetwornika o numerze 17, a czujnik temperatury do wejścia o numerze 16.

Kalibrowanie pomiaru napięcia

Pod adresem 0x1FFF F7BA jest zapisana 16-bitowa stała VREFINT_CAL, która odpowiada wartości odczytu A/C przy pomiarze napięcia wzorcowego, dla napięcia zasilania 3,3 V i temperatury 30°C. Należy zauważyć, że napięciem odniesienia dla przetwornika A/C jest zawsze napięcie zasilania. Wartość *VREFINT_CAL* może posłużyć do obliczenia wartości napięcia zasilania, która może być następnie użyta do obliczenia wartości zewnętrznych napięć mierzonych przez przetwornik A/C.

Zależność pomiędzy wartością stałej VREFINT_CAL i wartością napięcia wzorcowego V_{REF} wyraża się wzorem:

$$\frac{\text{VREFINT\_CAL}}{4096} = \frac{V_{REF}}{3{,}3\ \text{V}}$$

Listing 1. Plik *stm32f0discovery.h*

```
/*
  STM32F0DISCOVERY & STM32F0308-DISCO board defs
  gbm 11'2013
*/

#include „stm32f0yy.h"
#include „stm32futil.h"

// STM32F0DISCOVERY board conections
#define LED_PORT        GPIOC
#define BLUE_LED_BIT    8
#define GREEN_LED_BIT   9
#define BUTTON_PORT     GPIOA
#define BUTTON_BIT      0
#define BLUE_LED_PWM    TIM3->CCR3
#define GREEN_LED_PWM   TIM3->CCR4
```

W celu uniknięcia zbędnych operacji zmiennopozycyjnych, będziemy prowadzili obliczenia napięć w miliwoltach:

$$V_{DD}[mV] = \frac{\text{VREFINT\_CAL} \cdot 3300}{\text{VREFINT\_ADC}}$$

Taka postać wzoru na napięcie zasilania została użyta w programie przykładowym.

Kalibrowanie pomiaru temperatury

Według dokumentacji producenta, czujnik temperatury zawarty w układzie mikrokontrolera wytwarza napięcie, którego zależność od temperatury jest niemal liniowa. Do kalibrowania czujnika służą dwie 16-bitowe stałe *TS_CAL1* i *TS_CAL2*, zapisane w pamięci stałej, odpowiednio pod adresami 0x1FFF F7B8 i 0x1FFF F7C2. Ich wartości odpowiadają odczytom A/C z pomiaru napięcia

Listing 2. Plik *stm32f0yy.h*

```
/*
        STM32F0 series enhanced defs
        gbm 11'2013
*/
#ifndef __STM32F0YY_H
#define __STM32F0YY_H
#include „stm32f0xx.h"
// STM32F0x register/bit defs not present in stm32f0xx.h file
#define RCC_AHBENR_RSTVAL  (RCC_AHBENR_FLITFEN | RCC_AHBENR_SRAMEN)
#define GPIO_MODER_OUT  1
#define GPIO_MODER_AF  2
#define GPIOA_MODER_SWD  (GPIO_MODER_AF << (14 << 1) | GPIO_MODER_AF << (13 << 1)) // keep SWD pins
#define GPIO_PUPDR_PU  1
#define GPIO_PUPDR_PD  2
#define TIM_CCMR2_OC3M_PWM1  0x0060  // OC3M[2:0] - PWM mode 1
#define TIM_CCMR2_OC4M_PWM1  0x6000  // OC4M[2:0] - PWM mode 1
#define ADC_SMPR_71_5  6
#define ADC_SMPR_239_5  7  // 17 us @ 14 MHz - for TS
// Calibration values stored in ROM
#define VREFINT_CAL  (*(uint16_t *)0x1ffff7ba)
#define TS_CAL1  (*(uint16_t *)0x1ffff7b8)
#define TS_CAL2  (*(uint16_t *)0x1ffff7c2)
#define T_CAL1  30
#define T_CAL2  110
#endif
```

Wartość napięcia wzorcowego naszego egzemplarza mikrokontrolera wynosi:

$$V_{REF} = \frac{\text{VREFINT\_CAL} \cdot 3{,}3\ \text{V}}{4096}$$

Jeżeli wartość odczytaną z ADC podczas pomiaru napięcia wzorcowego oznaczymy jako VREFINT_ADC, to bieżącą wartość napięcia zasilania wyznaczamy z wzoru:

$$\frac{\text{VREFINT\_ADC}}{4096} = \frac{V_{REF}}{V_{DD}}$$

Stąd:

$$V_{DD} = \frac{V_{REF} \cdot 4096}{\text{VREFINT\_ADC}} = \frac{\text{VREFINT\_CAL} \cdot 3{,}3\ \text{V} \cdot 4096}{4096 \cdot \text{VREFINT\_ADC}} = \frac{\text{VREFINT\_CAL} \cdot 3{,}3\ \text{V}}{\text{VREFINT\_ADC}}$$

Listing 3. Plik *stm32futil.h*

```
/*
  some useful defs for STM32F0
  gbm 11'2013
*/
#ifndef STM32FUTIL_H
#define STM32FUTIL_H
#define BF2(b,v)   ((v) << ((b & 0xf) * 2))
#define BF4(b,v)   ((v) << ((b & 7) * 4))
typedef __IO uint32_t * __IO32p;  // short type name
// register init structure and routine

struct init_entry_ {
  volatile uint32_t *loc;
  uint32_t value;
};

static __INLINE void writeregs(const struct init_entry_ *p)
{
  for (; p->loc; p ++) ->loc = p->value;
}
#endif
```

wytwarzanego przez czujnik temperatury przy napięciu zasilania 3,3 V i temperaturach 30°C i 110°C.

Przyjmując, że zależność temperatury od wartości odczytanej z przetwornika analogowo-cyfrowego wyraża się wzorem:

$$T = a \cdot \mathrm{TS\_ADC} + b,$$

Do obliczenia temperatury są potrzebne wartości współczynników a i b, które można wyznaczyć z wartości kalibracyjnych poprzez rozwiązanie układu równań wiążących dwie wzorcowe temperatury z odpowiadającymi im odczytami ADC.

$$T1 = a \cdot \mathrm{TS\_CAL1} + b$$
$$T2 = a \cdot \mathrm{TS\_CAL2} + b$$

Stąd:

$$a = \frac{T2 - T1}{\mathrm{TS\_CAL2} - \mathrm{TS\_CAL1}}$$
$$b = T1 - a \cdot \mathrm{TS\_CAL1}$$

Ponieważ wartości odczytów A/C czujnika temperatury zależą od napięcia zasilania, zależność ta musi być odpowiednio uwzględniona podczas obliczeń. Można to uzyskać np. normalizując wartość odczytaną z ADC tak, aby przekształcić ją na wartość odpowiadającą odczytowi przy napięciu zasilania 3,3 V.

$$\mathrm{TS\_ADC}_{NORM} = \mathrm{TS\_ADC} \cdot \frac{V_{DD}[mV]}{3300} =$$
$$\mathrm{TS\_ADC} \cdot \frac{\mathrm{VREFINT\_CAL} \cdot 3300}{\mathrm{VREFINT\_ADC} \cdot 3300} = \mathrm{TS\_ADC} \cdot \frac{\mathrm{VREFINT\_CAL}}{\mathrm{VREFINT\_ADC}}$$

Listing 4. Pomiar temperatury i przesłanie wyniku za pomocą UART

```
/*
  STM32F0DISCOVERY & STM32F0308-DISCO tutorial
  ADC - calibrated voltage & temperature readout
  gbm, 11'2013
*/

#include „stm32f0discovery.h"
#define SYSCLK_FREQ  HSI_VALUE
#define BAUD_RATE  57600
// PWM constants
#define PWM_FREQ  400  // Hz
#define PWM_STEPS  80
#define PWM_CLK  SYSCLK_FREQ
#define PWM_PRE  (PWM_CLK / PWM_FREQ / PWM_STEPS)
#define LED_MAX  (PWM_STEPS - 1)
#define LED_DIM  1
#define LED_OFF  0

#define AVG_SHIFT 4
void SystemInit(void)
{
  FLASH->ACR = FLASH_ACR_PRFTBE;  // enable prefetch
}
static const struct init_entry_ init_table[] =
{
  // enable peripherals
  {&RCC->APB1ENR, RCC_APB1ENR_TIM3EN},
  {&RCC->APB2ENR, RCC_APB2ENR_USART1EN | RCC_APB2ENR_ADC1EN},
  {&RCC->AHBENR, RCC_AHBENR_GPIOCEN |  RCC_AHBENR_GPIOAEN
   | RCC_AHBENR_FLITFEN | RCC_AHBENR_SRAMEN | RCC_AHBENR_DMA1EN},
    // port setup
    {&GPIOA->AFR[1], BF4(10, 1) | BF4(9, 1)}, // USART pins 10 - RX, 9 - TX
    {&GPIOA->PUPDR, BF2(10, GPIO_PUPDR_PD)},  // pulldn on PA10
    {&GPIOA->MODER, GPIOA_MODER_SWD
     | BF2(10, GPIO_MODER_AF) | BF2(9, GPIO_MODER_AF)},  // UART pins as AF
    {&LED_PORT->MODER, BF2(GREEN_LED_BIT, GPIO_MODER_AF)
     | BF2(BLUE_LED_BIT, GPIO_MODER_AF)},  // set LED pins as AF
    // ADC setup
    {&ADC1->CHSELR, ADC_CHSELR_CHSEL17 | ADC_CHSELR_CHSEL16},  // VREF, TS
    {&ADC1->SMPR, ADC_SMPR_239_5},  // ADC
    {&ADC->CCR, ADC_CCR_TSEN | ADC_CCR_VREFEN},  // ADC
    {&ADC1->CFGR1, ADC_CFGR1_WAIT | ADC_CFGR1_CONT},  // ADC
    // ADC cal
    {&ADC1->CR, ADC_CR_ADCAL},  // ADC
    // USART1 setup
    {(__IO32p)&USART1->BRR, (SYSCLK_FREQ + BAUD_RATE / 2) / BAUD_RATE},
    {&USART1->CR2, USART_CR2_TXINV | USART_CR2_RXINV},  // Invert TX & RX
    {&USART1->CR3, USART_CR3_DMAT},  // enable Tx DMA
    {&USART1->CR1, USART_CR1_TE | USART_CR1_RE | USART_CR1_UE}, // enable
    // DMA init moved to main() to avoid ANSI C warnings
    // PWM timer setup - TIM3
    {(__IO32p)&TIM3->PSC, PWM_PRE - 1},  // prescaler
    {(__IO32p)&TIM3->ARR, PWM_STEPS - 1},  // period
    // blue - CH3, green - CH4
    {(__IO32p)&TIM3->CCMR2, TIM_CCMR2_OC4M_PWM1 | TIM_CCMR2_OC4PE
     | TIM_CCMR2_OC3M_PWM1 | TIM_CCMR2_OC3PE},  // PWM mode 1, buffered preload
    {(__IO32p)&TIM3->CCER, TIM_CCER_CC4E | TIM_CCER_CC3E},  // enable CH3, 4 output
    {(__IO32p)&TIM3->DIER, TIM_DIER_UIE},  // enable update interrupt
    {(__IO32p)&TIM3->CR1, TIM_CR1_ARPE | TIM_CR1_CEN},      // auto reload, enable
    // interrupts and sleep
    {&NVIC->ISER[0], 1 << TIM3_IRQn},  // enable interrupts
    {&SCB->SCR, SCB_SCR_SLEEPONEXIT_Msk},  // sleep while not in handler
    {0, 0}
};
static uint8_t meas[] = „Vdd = 0000 mV, T = +00.0 deg C\r\n";
int main(void)
{
  writeregs(init_table);
  DMA1_Channel2->CMAR = (uint32_t)meas;
  DMA1_Channel2->CPAR = (uint32_t)&USART1->TDR;
  __WFI();  // go to sleep
}

void TIM3_IRQHandler(void)
{
  static uint8_t tdiv = 0;
  static uint8_t blue_target = LED_DIM, green_target = LED_MAX;
  static uint8_t sdiv = 0;
  static uint32_t adc_avg[2];
```

Finalna wartość mierzonej temperatury będzie obliczana z wzoru:

$$T = a \cdot \mathrm{TS\_ADC}_{NORM} + b = a \cdot \mathrm{TS\_ADC}_{NORM} + T1 - a \cdot \mathrm{TS\_CAL1} = a \cdot (\mathrm{TS\_ADC}_{NORM} - \mathrm{TS\_CAL1}) + T1$$

Po podstawieniu wartości współczynników a i b otrzymujemy postać:

$$T = \frac{(T2-T1)\cdot(\mathrm{TS\_ADC}_{NORM} - \mathrm{TS\_CAL1})}{\mathrm{TS\_CAL2} - \mathrm{TS\_CAL1}} + T1,$$

a po wstawieniu wzoru na znormalizowaną wartość odczytu ADC i po redukcji dzieleń:

$$T = \frac{(T2 - T1) \cdot (\mathrm{TS\_ADC} \cdot \mathrm{VREFINT\_CAL} - \mathrm{TS\_CAL1} \cdot \mathrm{VREFINT\_ADC})}{(\mathrm{TS\_CAL2} - \mathrm{TS\_CAL1}) \cdot \mathrm{VREFINT\_ADC}} + T1$$

Program przykładowy

Program demonstracyjny pokazany na **listingu 4** mierzy napięcie zasilania i temperaturę procesora i wysyła je na terminal przez port szeregowy. W celu wyświetlenia wysyłanych danych na komputerze PC należy połączyć pojedynczym przewodem linię PA9 (TXD) mikrokontrolera ze stykiem 2 złącza DB9 interfejsu RS232C. Połączenie masy jest zapewnione przez kabel USB łączący płytkę DISCOVERY z PC, a programowe odwrócenie polaryzacji linii TXD skutecznie zastępuje nadajnik RS232C przy transmisji na małe odległości. Dane są transmitowane z szybkością 57600 b/s. Do komunikacji z płytką można

Listing 4. c.d.

```
  uint32_t pwmval;
  TIM3->SR = ~TIM_SR_UIF;  // clear interrupt flag
  if ((++ tdiv & 3) == 0)
  {
    int i;
    static enum {ADCH_TS, ADCH_VREF} adch = ADCH_TS;
    if (ADC1->ISR & ADC_ISR_EOC)
    {
      uint32_t val = ADC1->DR;     // 0 before first conversion
      if (adc_avg[adch] == 0)
      {
        // initial measure - set
        adc_avg[adch] = val << AVG_SHIFT;
      }
      else
      {
        // low-pass filters
        adc_avg[adch] = adc_avg[adch] + val - (adc_avg[adch] >> AVG_SHIFT);
      }
      if (++ adch > ADCH_VREF) adch = ADCH_TS;
    }
    else if (ADC1->ISR & ADC_ISR_ADRDY)
    {
      // ready for conversion
      ADC1->CR = ADC_CR_ADSTART | ADC_CR_ADEN;  // start cont. conversion
    }
    else if ((ADC1->CR & (ADC_CR_ADCAL | ADC_CR_ADEN)) == 0)
    {
      // calibrated but not enabled yet - enable
      ADC1->CR = ADC_CR_ADEN;
    }
    if (++ sdiv == 100)
    {
      uint32_t Vdd_mV;
      int32_t Temperature_x10, tsign;
      sdiv = 0;
      Vdd_mV = VREFINT_CAL * 3300 / (adc_avg[ADCH_VREF] >> AVG_SHIFT);
      // show results
      for (i = 3; i >= 0; i --)
      {
        meas[6 + i] = Vdd_mV % 10 + ,0';
        Vdd_mV /= 10;
      }
      Temperature_x10 = (T_CAL2 - T_CAL1) * 10 * (((int32_t)adc_avg[ADCH_TS] >> AVG_SHIFT) *
VREFINT_CAL - TS_CAL1 * (int32_t)(adc_avg[ADCH_VREF] >> AVG_SHIFT)) / (((int32_t)TS_CAL2 -
(int32_t)TS_CAL1) * (int32_t)(adc_avg[ADCH_VREF] >> AVG_SHIFT)) + T_CAL1 * 10;
      tsign = ,+';
      if (Temperature_x10 < 0)
      {
        tsign = ,-';
        Temperature_x10 = -Temperature_x10;
      }
      meas[23] = Temperature_x10 % 10 + ,0';
      Temperature_x10 /= 10;
      i = 21;
      do {
        meas[i --] = Temperature_x10 % 10 + ,0';
        Temperature_x10 /= 10;
      } while (Temperature_x10);
      meas[i --] = tsign;
      while (i > 18)
      meas[i --] = , ,;
      // init DMA for string transfer
      DMA1_Channel2->CCR = 0;  // disable
      DMA1_Channel2->CNDTR = sizeof(meas) - 1;  // no. of items
      // increment memory adress, mem->periph, enable
      DMA1_Channel2->CCR = DMA_CCR_MINC | DMA_CCR_DIR | DMA_CCR_EN;
      blue_target ^= LED_MAX ^ LED_DIM;
      green_target ^= LED_MAX ^ LED_DIM;
    }
  }
  if ((pwmval = BLUE_LED_PWM) != blue_target)
  BLUE_LED_PWM = pwmval < blue_target ? pwmval + 1 : pwmval - 1;
  if ((pwmval = GREEN_LED_PWM) != green_target)
  GREEN_LED_PWM = pwmval < green_target ? pwmval + 1 : pwmval - 1;
}
```

użyć dowolnego programu terminala skonfigurowanego na transmisję danych 8-bitowych bez kontroli parzystości z szybkością 57600 b/s, bez synchronizacji.

Program powstał przez modyfikację wcześniejszego przykładu ilustrującego nadawanie danych przez UART z użyciem DMA.

Zaprogramowanie przetwornika A/C

Przetwornik jest skonfigurowany tak, by mierzył on wartości z kanałów 16 i 17. Kolejny pomiar jest wyzwalany przez odczyt wyników poprzedniego – osiąga się to przez ustawienie bitów CONT i WAIT w rejestrze CFGR1. Dzięki temu nie ma potrzeby jawnego wyzwalania pomiarów poprzez zapisy do rejestrów sterujących przetwornika.

Oprogramowanie przetwornika analogowo-cyfrowego stanowi również modyfikację przedstawionego wcześniej przykładu – zostało ono zrealizowane w konwencji automatu zapewniającego zainicjowanie przetwornika i prowadzenie pomiarów bez konieczności oczekiwania programowego. Wszystkie akcje wykonywane są w obsłudze przerwania timera, zgłaszanego z częstotliwością 400 Hz, przy co czwartym wejściu w obsługę przerwania, a więc z częstotliwością 100 Hz. Po zainicjowaniu i początkowej kalibracji przetwornika następuje naprzemienne odczytywanie wyników z A/C i inicjowanie kolejnego pomiaru dla dwóch kanałów, odpowiadających czujnikowi temperatury i źródłu napięcia wzorcowego. Wyniki pomiarów są filtrowane zrealizowanym programowo filtrem dolnoprzepustowym, podobnie jak we wcześniejszym przykładzie dotyczącym przetwornika.

Obliczenia w programie

Zastosowany sposób realizacji obliczeń umożliwia wyeliminowanie operacji zmiennopozycyjnych oraz zmniejszenie liczby operacji dzielenia, które są najwolniejszymi operacjami arytmetycznymi – w przypadku rdzenia Cortex-M0 są one wykonywane przez procedury programowe.

Napięcie zasilania jest obliczane w miliwoltach, a temperatura – w dziesiątych częściach stopnia. Przy zastosowanej w oprogramowaniu postaci wzorów, zarówno do obliczenia wartości napięcia zasilania, jak i temperatury niezbędne jest wykonanie tylko po jednym dzieleniu. Potrzebne do obliczeń napięcia wzorcowego i temperatury stałe i parametry z pamięci ROM mikrokontrolera zostały zdefiniowane jako symbole preprocesora w pliku *stm32f0yy.h*. Wartości napięcia zasilania i temperatury są obliczane i wysyłane do terminala co jedną sekundę. Moment transmisji wyników jest sygnalizowany zmianą stanu diod LED umieszczonych na płytce.

Przygotowanie wyników

Obliczone wartości napięcia i temperatury są zamieniane na postać znakową przy użyciu prostych fragmentów kodu cyfry są wstawiane na odpowiednie pozycje bufora przechowującego cały łańcuch znaków reprezentujący wyniki pomiarów. Następnie jest inicjowana transmisja DMA całego bufora do modułu UART.

Grzegorz Mazur