32 bity jak najprościej (3)

Pierwsze kroki z modułem STM32F0DISCOVERY

Samouczek jest dedykowany szczególnie tym projektantom, którzy stają przed perspektywą zmiany mikrokontrolera z 8-bitowego na nowszy i tańszy 32-bitowy. Wszystkie programy są napisane w języku C. W przykładach nie użyto bibliotek służących do obsługi peryferiali, dzięki czemu kod programów jest krótki i czytelny, a zajętość pamięci znacznie mniejsza, niż w typowych programach demonstracyjnych, udostępnianych przez producentów mikrokontrolerów. Zwrócono również szczególną uwagę na poprawność prezentowanych rozwiązań i sposób zapisu programu ułatwiający optymalizację kodu i wychwytywanie błędów przez kompilator. Przedstawione programy zostały napisane w taki sposób, że nie generują one żadnych ostrzeżeń kompilatora.

Port szeregowy USART

Mikrokontrolery STM32F051 są wyposażone w sterownik transmisji szeregowej USART, który może służyć do transmisji zarówno asynchronicznej, jak i synchronicznej. Użyty w trybie asynchronicznym, może on np. sterować interfejsem RS232. Na płytce DISCOVERY nie ma translatorów poziomów RS232, jednak własności elektryczne i logiczne STM32F05x umożliwiają współpracę z interfejsem RS232 bez użycia translatora, o ile transmisja odbywa się w warunkach laboratoryjnych – na małą odległość i bez zakłóceń. Do połączenia mikrokontrolera z interfejsem RS232C komputera PC potrzebny będzie tylko jeden rezystor o wartości od 47 do 100 kΩ.

Przedstawiony poniżej projekt oprogramowania powstał przez modyfikację projektu BtnBlink. Nosi on nazwę USART1.

Zaprogramowanie modułu USART

Przed zaprogramowaniem modułu USART należy określić parametry transmisji. Wybierzemy szybkość 9600 bit/s, format 8-bitowy bez parzystości, z jednym bitem stopu. Wartość, która posłuży do zaprogramowania szybkości transmisji, zdefiniujemy jako wyrażenie preprocesora. Ponieważ nie użyjemy układu translatora poziomów RS232C, zaprogramujemy odwracanie poziomów napięć na liniach RX i TX.

W celu zaprogramowania modułu USART należy:

- Włączyć moduł poprzez ustawienie bitu USART1EN w rejestrze APB2ENR.
- Włączyć port GPIOA.
- Wybrać funkcję AF 1 (RX i TX USART1) dla wyprowadzeń 10 i 9 portu GPIOA.
- Włączyć ściąganie w dół dla linii RX
- Ustawić linie 10 i 9 portu GPIOA jako AF, pozostawiając funkcje interfejsu SWD dla linii 14 i 13.
- Ustawić podzielnik szybkości transmisji w rejestrze BRR modułu USART1.
- Włączyć odwracanie polaryzacji linii RX i TX – bity TXINV i RXINV rejestru CR2.
- Włączyć przerwanie odbioru danych, zezwolenie na nadawanie i odbiór oraz cały układ USART1 w rejestrze CR1 – musi to być ostatnia czynność podczas programowania rejestrów modułu USART.
- Włączyć przerwanie USART1 w module NVIC – rejestr ISER[0].

Ponieważ w projekcie użyjemy przerwania modułu USART, należy je oprogramować. Jedynym włączonym źródłem tego przerwania będzie odebranie danej. Procedura obsługi przerwania USART1 odczytuje odebrany znak i poddaje go prostemu przetwarzaniu. Małe litery są zamieniane na wielkie i odwrotnie, a odebranie kodu CR powoduje, poprzez ustawienie zmiennej *mchange*, zmianę trybu świecenia diod. Po przetworzeniu znak zostaje odesłany z powrotem.

Podczas wysyłania znaku do PC nie jest sprawdzana gotowość nadajnika USART. W naszym przykładzie nie ma to znaczenia, gdyż przy nadawaniu z komputera PC znaków wprowadzanych przez użytkownika w programie terminala nie ma możliwości przepełnienia bufora nadajnika interfejsu USART mikrokontrolera znakami odsyłanymi zwrotnie. Nie jest to jednak uniwersalny sposób programowej obsługi nadawania, który może być zastosowany w każdej sytuacji. Poprawne nadawanie danych z użyciem przerwań zostanie zaprezentowane w następnym przykładzie.

Połączenie modułu DISCOVERY z komputerem

W celu umożliwienia komunikacji modułu z komputerem PC należy poprowadzić trzy połączenia – masy oraz dwóch linii danych. Ponieważ zasilamy moduł STM32F0 z komputera, masa jest już połączona przez port USB. Do nawiązania transmisji wystarczy więc wykonać połączenia linii danych z komputera do modułu i z modułu do komputera. Wyjście danych mikrokontrolera – linię 9 portu GPIOA – łączymy bezpośrednio z wejściem danych komputera PC – linią 2 złącza DB9. Wejście danych mikrokontrolera – linię 10 portu GPIOA – łączymy przez rezystor szeregowy 100 kΩ z wyjściem danych komputera PC – linią 3 złącza DB9. Podczas wykonywania połączeń

Listing 1. Najprostszy przykład obsługi USART

```
/*
        STM32F0DISCOVERY tutorial
        USART1 + TIM3 PWM blinker with table-driven init and blink mode switching
        gbm, 12'2012
*/

#include „stm32f0xx.h"
//=========================================================================
// defs for STM32F05x chips
#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

typedef __IO uint32_t * __IO32p;
//=========================================================================
// defs for STM32F0DISCOVERY board
#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
//=========================================================================
#define SYSCLK_FREQ        HSI_VALUE

#define BAUD_RATE 9600

#define BLINK_PERIOD       50       // * 10 ms
#define BTN_ACK_PERIOD     50       // * 10 ms

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

static __INLINE void writeregs(const struct init_entry_ *p)
{
    for (; p->loc; p ++)
        *p->loc = p->value;
}
//=========================================================================
void SystemInit(void)
{
    FLASH->ACR = FLASH_ACR_PRFTBE; // enable prefetch
}
//=========================================================================
static const struct init_entry_ init_table[] =
{
    {&RCC->APB2ENR, RCC_APB2ENR_USART1EN}, // USART1 clock enable
    // port setup
    {&RCC->AHBENR, RCC_AHBENR_GPIOCEN |  RCC_AHBENR_GPIOAEN},        // GPIOC, GPIOA
    {&GPIOA->AFR[1], 1 << (2 << 2) | 1 << (1 << 2)},         // set USART pins 10 - RX, 9 - TX
    {&GPIOA->PUPDR, GPIO_PUPDR_PD << (10 << 1)},    // set pulldn on PA10
    {   &GPIOA->MODER, GPIOA_MODER_SWD
        | GPIO_MODER_AF << (10 << 1) | GPIO_MODER_AF << (9 << 1)
    },   // set UART pins as AF
    {   &LED_PORT->MODER, GPIO_MODER_AF << (GREEN_LED_BIT << 1)
        | GPIO_MODER_AF << (BLUE_LED_BIT << 1)
    },   // set LED pins as AF
    {&RCC->AHBENR, RCC_AHBENR_GPIOAEN},    // GPIOA
    // USART1 setup
    {(__IO32p)&USART1->BRR, (SYSCLK_FREQ + BAUD_RATE / 2) / BAUD_RATE},      //
    {&USART1->CR2, USART_CR2_TXINV | USART_CR2_RXINV},      // Invert ext. lines
    {&USART1->CR1, USART_CR1_RXNEIE | USART_CR1_TE | USART_CR1_RE | USART_CR1_UE},    // enable
    // PWM timer setup - TIM3
    {&RCC->APB1ENR, RCC_APB1ENR_TIM3EN},   // 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 << USART1_IRQn | 1 << TIM3_IRQn},    // enable interrupts
    {&SCB->SCR, SCB_SCR_SLEEPONEXIT_Msk},  // sleep while not in handler
    {0, 0}
};
//=========================================================================
int main(void)
{
    writeregs(init_table);
    __WFI();     // go to sleep
}
```

Listing 1. c.d.

```
//=========================================================================
static _Bool mchange = 0;
//=========================================================================
void TIM3_IRQHandler(void)
{
    static uint8_t blink_timer = BLINK_PERIOD;
    static enum {BM_SLOW, BM_FAST, BM_OFF} blink_mode = BM_SLOW;
    static uint8_t bstate = 0;
    static uint8_t blue_led_timer = 0, on_time = 0;
    static const uint8_t blink_periods[] = {100, 50, 1};
    static uint8_t tdiv = 0;
    static uint8_t blue_target = LED_DIM, green_target = LED_DIM;

    uint32_t pwmval;

    TIM3->SR = ~TIM_SR_UIF;        // clear interrupt flag

    if ((++ tdiv & 3) == 0)
    {
        bstate = (bstate << 1 & 0xf) | (BUTTON_PORT->IDR >> BUTTON_BIT & 1);

        if (blue_led_timer)
        {
            if (-- blue_led_timer == 0)
                blue_target = LED_DIM;
        }
        else if (bstate == 1 || mchange)
        {
            // button pressed or CR received - change blink mode
            if (++ blink_mode > BM_OFF)
                blink_mode = BM_SLOW;
            blue_led_timer = BTN_ACK_PERIOD;
            blue_target = LED_MAX;
            USART1->TDR = blink_mode + ,0';
            mchange = 0;
        }

        if (-- blink_timer == 0)
        {
            blink_timer = blink_periods[blink_mode];
            on_time = blink_timer >> 1;
            green_target = LED_DIM;
        }
        else if (blink_timer == on_time)
            green_target = LED_MAX;
    }

    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;
}
//=========================================================================
void USART1_IRQHandler(void)
{
    if (USART1->ISR & USART_ISR_RXNE)      // data received
    {
        uint8_t c;

        c = USART1->RDR;
        if (c == ,\r')
            mchange = 1;
        else if ((c >= ,A' && c <= ,Z') || (c >= ,a' && c <= ,z'))
            c ^= ,A' ^ ,a';
        USART1->TDR = c;
    }
}
//=========================================================================
```

należy zachować szczególną uwagę – błędne podłączenie wyjścia danych mikrokontrolera do niewłaściwej linii złącza DB9 może spowodować trwałe uszkodzenie mikrokontrolera.

Obsługa nadajnika USART z użyciem przerwań

Poprzedni przykład (**listing 1**) pokazuje dość prostą i nienadającą się do szerszego zastosowania metodę nadawania danych przez moduł USART. W kolejnym projekcie zaprezentujemy nadawanie ciągów znaków przy użyciu przerwania zgłaszanego przy gotowości nadajnika. W każdym projekcie urządzenia nadającego dane musimy zadbać o to, by strumień danych nie przekraczał możliwości nadajnika. Nasz program będzie zliczał naciśnięcia przycisku. Po każdym naciśnięciu wartość licznika będzie wysyłana do PC przez interfejs UART w postaci 8-cyfrowej liczby szesnastkowej zakończonej sekwencją przejścia do nowego wiersza CR+LF. Każde naciśnięcie przycisku będzie więc powodowało transmisję 10 bajtów. Przesyłane dane można będzie oglądać przy użyciu programu terminala. Ponieważ z zastosowanego wcześniej algorytmu obsługi przycisku wynika, że można zarejestrować maksymalnie 25 naciśnięć w ciągu sekundy, przy szybkości transmisji 9600 bit/s nie zachodzi obawa nasycenia nadajnika. Aby obsługa transmisji była uniwersalna, zastosujemy dla transmitowanych danych bufor cykliczny, pomimo że nie jest on niezbędny w naszym zastosowaniu, w którym długość przesyłanego bloku danych jest stała. Program powstanie przez modyfikację poprzedniego przykładu użycia USART. Gotowy projekt nosi nazwę USART1cnt.

Procedury obsługi nadawania

Zaczynamy od sekwencji inicjującej USART1. Tym razem nie będziemy używali przerwania odbiornika. Musimy za to oprogramować nadawanie danych przy użyciu przerwania nadajnika.

W programie zdefiniujemy bufor nadawanych znaków *u1_outbuf[]* oraz dwa indeksy – wskaźniki znaków wstawianych i pobieranych z bufora – *u1_obiptr* i *u1_oboptr*. Gdy bufor jest pusty, oba wskaźniki są równe. Przerwanie nadajnika będzie włączane tylko wtedy, gdy w buforze znajdą się dane do nadawania.

Procedura obsługi przerwania USART1 jest wywoływana, gdy interfejs jest gotowy na przyjęcie danej do nadawania i jest włączone przerwanie nadajnika. Procedura najpierw oblicza przyszłą wartość wskaźnika danych pobieranych. Jeżeli po wysłaniu kolejnego znaku bufor zostałby opróżniony, przerwanie nadajnika jest wyłączane. Następnie znak z bufora jest ładowany do rejestru nadajnika oraz jest modyfikowany wskaźnik pobieranych danych.

Procedura wstawienia znaku do bufora *USART1_put* sprawdza, czy w buforze jest miejsce, a następnie wstawia znak do bufora, przesuwa wskaźnik wstawiania i włącza przerwanie nadajnika. W przypadku zapełnienia bufora procedura zwraca wartość 1, sygnalizującą błąd. Procedura nie czeka w takim przypadku na zwolnienie miejsca w buforze.

Procedura wysyłania cyfry szesnastkowej *puthexdigit* zamienia wartość tetrady na jej reprezentację szesnastkową w kodzie ASCII, a następnie wstawia ją do bufora przy użyciu funkcji *USART1_put*.

Procedura obsługi przerwania timera

W procedurze zadeklarowano zmienną *count* – licznik zliczający naciśnięcia przycisku. Procedura obsługi timera w przypadku wykrycia naciśnięcia przycisku inkrementuje licznik i wstawia do bufora nadawczego jego reprezentację znakową, zakończoną sekwencją końca wiersza. Równocześnie jest zaświecana niebieska dioda, sygnalizująca naciśnięcie przycisku (**listing 2**).

Zliczanie czasu i transmisja USART z użyciem DMA

Kolejny program będzie transmitował dane przez USART korzystając z modułu bezpośredniego dostępu do pamięci, czyli bez udziału oprogramowania w transmisji każdego bajtu. Tam razem program będzie pełnił rolę prostego timera, odliczającego czas w sekundach. Naciśnięcie przycisku umieszczonego na płytce będzie powodowało wyzerowanie i restart timera. Co jedną sekundę bieżąca wartość timera będzie transmitowana przez USART – można ją wyświetlić na PC korzystając z programu termi-

Listing 2. Obsługa zliczania przycisku i przesyłania danych przez UART

```
/*
        STM32F0DISCOVERY tutorial
        USART1 interrupt-driven Tx
        gbm, 02'2013
*/

#include „stm32f0xx.h"
//=========================================================================
// defs for STM32F05x chips
#define GPIO_MODER_OUT     1
#define GPIO_MODER_AF      2
// keep SWD pins
#define GPIOA_MODER_SWD    (GPIO_MODER_AF << (14 << 1) | GPIO_MODER_AF << (13 << 1))
#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

typedef __IO uint32_t * __IO32p;
//=========================================================================
// defs for STM32F0DISCOVERY board
#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
//=========================================================================
#define SYSCLK_FREQ        HSI_VALUE

#define BAUD_RATE 9600

#define BTN_ACK_PERIOD     25       // * 10 ms

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

static __INLINE void writeregs(const struct init_entry_ *p)
{
    for (; p->loc; p ++)
        *p->loc = p->value;
}
//=========================================================================
void SystemInit(void)
{
    FLASH->ACR = FLASH_ACR_PRFTBE; // enable prefetch
```

Listing 2. c.d.

```
}
//===========================================================================
static const struct init_entry_ init_table[] =
{
    // clock control
    {&RCC->APB1ENR, RCC_APB1ENR_TIM3EN},
    {&RCC->APB2ENR, RCC_APB2ENR_USART1EN},
    // port setup
    {&RCC->AHBENR, RCC_AHBENR_GPIOCEN |  RCC_AHBENR_GPIOAEN},        // GPIOC, GPIOA
    {&GPIOA->AFR[1], 1 << (2 << 2) | 1 << (1 << 2)},        // set USART pins 10 - RX, 9 - TX
    {&GPIOA->PUPDR, GPIO_PUPDR_PD << (10 << 1)},    // set pulldn on PA10
    {&GPIOA->MODER, GPIOA_MODER_SWD
        | GPIO_MODER_AF << (10 << 1) | GPIO_MODER_AF << (9 << 1)
    },   // set UART pins as AF
    {&LED_PORT->MODER, GPIO_MODER_AF << (GREEN_LED_BIT << 1)
        | GPIO_MODER_AF << (BLUE_LED_BIT << 1)    },        // set LED pins as AF
    {&RCC->AHBENR, RCC_AHBENR_GPIOAEN},    // GPIOA
    // USART1 setup
    {(__IO32p)&USART1->BRR, (SYSCLK_FREQ + BAUD_RATE / 2) / BAUD_RATE},
    {&USART1->CR2, USART_CR2_TXINV | USART_CR2_RXINV},      // Invert ext. lines
    {&USART1->CR1, USART_CR1_TE | USART_CR1_RE | USART_CR1_UE},      // enable
    // 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 << USART1_IRQn | 1 << TIM3_IRQn},    // enable interrupts
    {&SCB->SCR, SCB_SCR_SLEEPONEXIT_Msk},  // sleep while not in handler
    {0, 0}
};
//===========================================================================
int main(void)
{
    writeregs(init_table);
    __WFI();      // go to sleep
}
//===========================================================================
// UART1 output buffer
#define U1OBSIZE  32
static uint8_t u1_outbuf[U1OBSIZE];
static uint8_t u1_obiptr = 0, u1_oboptr = 0;
//===========================================================================
void USART1_IRQHandler(void)
{
    if (USART1->ISR & USART_ISR_TXE)       // ready to send
    {
        uint32_t next_optr = (u1_oboptr + 1) % U1OBSIZE;

        if (u1_obiptr == next_optr)
            USART1->CR1 &= ~USART_CR1_TXEIE; // nothing more to send, disable int
        USART1->TDR = u1_outbuf[u1_oboptr];
        u1_oboptr = next_optr;
    }
}
//===========================================================================
static _Bool UART1_put(uint8_t c)
{
    uint32_t next_iptr = (u1_obiptr + 1) % U1OBSIZE;

    if (next_iptr == u1_oboptr)
        return 1;// buffer full - signal overflow

    u1_outbuf[u1_obiptr] = c;
    u1_obiptr = next_iptr;
    USART1->CR1 |= USART_CR1_TXEIE;
    return 0;    // ok
}
//===========================================================================
static void puthexdigit(uint8_t v)
{
    v &= 0xf;
    UART1_put(v + (v <= 9 ? ,0' : ,A' - 10));
}
//===========================================================================
void TIM3_IRQHandler(void)
{
    static uint8_t bstate = 0;
    static uint8_t blue_led_timer = 0;
    static uint8_t tdiv = 0;
    static uint8_t blue_target = LED_DIM;
    static uint32_t counter = 0;

    uint32_t pwmval;

    TIM3->SR = ~TIM_SR_UIF;       // clear interrupt flag

    if ((++ tdiv & 3) == 0)
    {
        // 100 Hz
        if ((bstate = (bstate << 1 & 0xf) | (BUTTON_PORT->IDR >> BUTTON_BIT & 1)) == 1)
        {
            int i;

            // button pressed - increment counter
            blue_led_timer = BTN_ACK_PERIOD;
            blue_target = LED_MAX;
```

Listing 2. c.d.
```
            counter ++;

            for (i = 7; i >= 0; i --)
                puthexdigit(counter >> (i * 4));
            UART1_put(,\r');
            UART1_put(,\n');
        }

        if (blue_led_timer)
        {
            if (-- blue_led_timer == 0)
                blue_target = LED_DIM;
        }
    }

    if ((pwmval = BLUE_LED_PWM) != blue_target)
        BLUE_LED_PWM = pwmval < blue_target ? pwmval + 1 : pwmval - 1;
}
//======================================================================
```

nala. Program, powstały przez modyfikację poprzedniego przykładu, jest zawarty w projekcie UART1dma.

Oprogramowanie modułu USART i DMA

Moduł USART1 jest inicjowany podobnie jak w poprzednim przykładzie. Wprowadzono jedną istotną zmianę – przed finalnym uaktywnieniem USART w rejestrze CR modułu jest ustawiony bit włączający zgłaszanie żądania DMA przez nadajnik.

Na początku sekwencji inicjującej jest włączany moduł DMA – wymaga to ustawienia odpowiedniego bitu w rejestrze *AHBENR* modułu *RCC*. Sterownik bezpośredniego dostępu do pamięci, zrealizowany w STM32F05x, dysponuje pięcioma kanałami transmisji. Z nadajnikiem USART1 jest związany kanał 2; programując moduł DMA będziemy używali rejestrów tego kanału, dostępnych w postaci struktury poprzez wskaźnik *DMA1_Channel2*.

Do jednorazowego zainicjowania modułu DMA potrzebne są dwa zapisy rejestrów adresów. Ponieważ specyfikowanie adresów jako stałych typów numerycznych generuje ostrzeżenie w ANSI C, zapisy te zostały umieszczone w funkcji *main*. Pełne programowanie modułu DMA do współpracy z nadajnikiem USART1 wymaga łącznie czterech operacji:

- Ustawienia adresu początkowego bufora w pamięci, z którego będą pobierane dane, w rejestrze CMAR.
- Ustawienia adresu rejestru danych nadajnika USART w rejestrze CMAR.
- Ustawienia liczby transmitowanych bajtów w rejestrze CNDTR.
- Ustawienia trybu transmisji oraz uaktywnienia kanału DMA poprzez zapis do rejestru sterującego kanału CCR bitów: inkrementacji adresu pamięci - MINC, kierunku transmisji – z pamięci do modułu peryferyjnego DIR oraz aktywacji kanału EN.

Listing 3. Obsługa DMA oraz UART
```
/*
        STM32F0DISCOVERY tutorial
        USART1 with DMA Tx + TIM3 timer & PWM blinker
        gbm, 02'2013
*/

#include „stm32f0xx.h"
//======================================================================
// STM32F05x
#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

typedef __IO uint32_t * __IO32p;
//======================================================================
// STM32F0DISCOVERY board
#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
//======================================================================
#define SYSCLK_FREQ        HSI_VALUE
#define BAUD_RATE 9600

#define BTN_ACK_PERIOD     25       // * 10 ms
// 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
//======================================================================
struct init_entry_ {
    volatile uint32_t *loc;
    uint32_t value;
};
```

Listing 3. c.d.

```
static __INLINE void writeregs(const struct init_entry_ *p)
{
    for (; p->loc; p ++)
        *p->loc = p->value;
}
//=========================================================================
void SystemInit(void)
{
    FLASH->ACR = FLASH_ACR_PRFTBE; // enable prefetch
}
//=========================================================================
static const struct init_entry_ init_table[] =
{
    // clock enable
    {&RCC->APB1ENR, RCC_APB1ENR_TIM3EN},
    {&RCC->APB2ENR, RCC_APB2ENR_USART1EN},
    {   &RCC->AHBENR, RCC_AHBENR_GPIOCEN |  RCC_AHBENR_GPIOAEN
        | RCC_AHBENR_FLITFEN | RCC_AHBENR_SRAMEN | RCC_AHBENR_DMA1EN
    },
    // port setup
    {&GPIOA->AFR[1], 1 << (2 << 2) | 1 << (1 << 2)}, // USART pins 10 - RX, 9 - TX
    {&GPIOA->PUPDR, GPIO_PUPDR_PD << (10 << 1)},   // pulldn on PA10
    {   &GPIOA->MODER, GPIOA_MODER_SWD
        | GPIO_MODER_AF << (10 << 1) | GPIO_MODER_AF << (9 << 1)
    },   // UART pins as AF
    {   &LED_PORT->MODER, GPIO_MODER_AF << (GREEN_LED_BIT << 1)
        | GPIO_MODER_AF << (BLUE_LED_BIT << 1)
    },   // set LED pins as AF
    // 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 time[8] = „000000\r\n”;
//=========================================================================
int main(void)
{
    writeregs(init_table);
    DMA1_Channel2->CMAR = (uint32_t)time;
    DMA1_Channel2->CPAR = (uint32_t)&USART1->TDR;
    __WFI();     // go to sleep
}
//=========================================================================
void TIM3_IRQHandler(void)
{
    static uint8_t bstate = 0;
    static uint8_t blue_led_timer = 0;
    static uint8_t tdiv = 0;
    static uint8_t blue_target = LED_DIM, green_target = LED_DIM;
    static uint8_t sdiv = 0;

    uint32_t pwmval;

    TIM3->SR = ~TIM_SR_UIF;        // clear interrupt flag

    if ((++ tdiv & 3) == 0)
    {
        // 100 Hz
        int i;
        _Bool upd_time = 0;

        if ((bstate = (bstate << 1 & 0xf) | (BUTTON_PORT->IDR >> BUTTON_BIT & 1)) == 1)
        {
            // button pressed - clear timer
            for (i = 0; i < 6; i ++)
                time[i] = ,0’;
            sdiv = 0;
            upd_time = 1;
            blue_target = LED_MAX;
            blue_led_timer = BTN_ACK_PERIOD;
        }
        else if (blue_led_timer && -- blue_led_timer == 0)
            blue_target = LED_DIM;

        if (++ sdiv == 100)
        {
            sdiv = 0;
            // increment time
            for (i = 5; i >= 0 && ++ time[i] > ,9’; i --)
                time[i] = ,0’;
            upd_time = 1;
            green_target = LED_MAX;
        }
        else if (sdiv == 50)
```

```
Listing 3. c.d.
            green_target = LED_DIM;

        if (upd_time)
        {
            // init DMA for time string transfer
            DMA1_Channel2->CCR = 0;         // disable
            DMA1_Channel2->CNDTR = sizeof(time);    // no. of items
            // increment memory adress, mem->periph, enable
            DMA1_Channel2->CCR = DMA_CCR_MINC | DMA_CCR_DIR | DMA_CCR_EN;
        }
    }

    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;
}
//=========================================================================
```

Z powodu wymagań standardu ANSI języka C zapis rejestrów adresowych kanału DMA następuje w funkcji *main* – umieszczenie go w tablicy inicjowania peryferiali spowodowałoby wygenerowanie przez kompilator ostrzeżeń o użyciu stałych wbrew standardowi.

Moduł DMA po zakończeniu transmisji zachowuje początkowe wartości rejestrów adresowych, ale wymaga zapisu długości bloku danych przed każdą transmisją. Dodatkowo, rejestr długości bloku może być zapisany tylko wówczas, gdy kanał jest nieaktywny, a kanał nie deaktywuje się samoczynnie po zakończeniu transmisji bloku. Z tych powodów w procedurze obsługi przerwania timera, w miejscu, gdzie następuje uaktywnienie kanału i rozpoczęcie transmisji, znalazła się sekwencja trzech instrukcji:

- deaktywacji kanału – zapis 0 do CCR,
- zapisu długości bloku do CNDTR,
- powtórnej aktywacji kanału – zapis słowa sterującego do CCR.

Obsługa przerwania timera

Przerwanie timera generującego przebiegi PWM sterujące diodami jest zgłaszane tak samo, jak we wcześniejszych projektach, z częstotliwością 400 Hz. Jego obsługa obejmuje kilka czynności, wykonywanych z różnymi częstotliwościami:

- płynną modyfikację wypełnień PWM dla diod – 400 Hz;
- reakcję na przycisk zerowania timera – 100 Hz;
- zliczanie czasu w timerze programowym i błyskanie zieloną diodą – 1 Hz.

W celu uniknięcia konwersji wartości timera z postaci binarnej na reprezentację znakową, wartość ta jest przechowywana w postaci znakowej, w tablicy znakowej *timer[]*. Tablica ta, o długości 8 bajtów, zawiera na pozycjach 0…5 sześć cyfr timera, a na pozycjach 6 i 7 – sekwencję końca wiersza, przesyłaną wraz z wartością timera przez łącze szeregowe.

Blok reakcji na naciśnięcie przycisku jest wywoływany, tak jak poprzednio, w co czwartym przerwaniu. W bloku tym jest sprawdzany stan przycisku oraz inkrementowany jest licznik *sdiv* zliczający setne części sekundy. Przy naciśnięciu przycisku zerowany jest timer odliczający sekundy oraz licznik setnych części sekundy. Dodatkowo jest zaświecana niebieska dioda; jej wygaszanie rozpoczyna się po upływie ¼ sekundy po naciśnięciu przycisku. Po osiągnięciu przez licznik setnych wartości 100 jest on zerowany oraz następuje inkrementacja timera programowego. Inkrementacja ta jest wykonywana w prostej pętli *for()*. Zarówno inkrementacja, jak i wyzerowanie licznika kończy się ustawieniem zmiennej logicznej upd_time, co w dalszej części przerwania timera powoduje zainicjowanie transmisji całego bufora timera programowego przez USART. Omawiany przykład pokazano na **listingu 3**.

Grzegorz Mazur
gbm@ii.pw.edu.pl