32 bity jak najprościej – STM32F0 (5)

Przechowywanie danych w pamięci Flash i monitorowanie zasilania

Kolejny przykład dla płytki STM32F0DISCOVERY prezentuje użycie pamięci Flash i monitora napięcia zasilania. Pretekstem do przedstawienia tych bloków mikrokontrolera będzie projekt urządzenia rejestrującego całkowity czas swojej pracy.

Podczas pracy urządzenia następuje zliczanie czasu, a przy wyłączaniu zasilania całkowity czas jego pracy w sekundach jest zapisywany do pamięci Flash. Przy ponownym włączeniu licznik czasu pracy jest inicjowany wartością zapamiętaną w pamięci nieulotnej. Gotowy projekt o nazwie `PVD-Flash` jest zawarty w nowej wersji pliku `F0tutorial.zip`.

Przedstawiony przykład może być uruchomiony na dowolnym mikrokontrolerze rodziny STM32F0 za wyjątkiem serii STM32F030, która nie jest wyposażona w monitor napięcia zasilania.

Organizacja pamięci Flash

Pamięć Flash w mikrokontrolerach STM32F0 jest podzielona na strony o rozmiarze 1 KiB. Najmniejszą jednostką kasowania jest strona, a jednostką zapisu – słowo 16-bitowe. W stanie skasowanym wszystkie bity komórki pamięci mają wartość 1. Do bezpiecznego przechowywania danych w pamięci Flash należy zarezerwować przynajmniej dwie strony, tak, aby podczas kasowania jednej z nich informacja była trwale zapisana na drugiej stronie. Cała pamięć Flash stanowi w STM32F0 jeden blok – podczas wykonywania operacji kasowania lub zapisu żadna komórka pamięci Flash nie może być odczytywana. Próba odczytu pamięci, np. w celu pobrania instrukcji, powoduje wstrzymanie pracy procesora do zakończenia operacji kasowania lub zapisu.

Sposób przechowywania danych

Program demonstracyjny będzie zapisywał do pamięci Flash słowa 32-bitowe zwierające wartość licznika czasu pracy. Aby uniknąć zbędnego kasowania pamięci, powodującego jej zużycie, kolejne wartości licznika będą zapisywane do kolejnych komórek pamięci, a zapis będzie następował podczas wyłączania zasilania urządzenia. Na przechowywanie danych zostaną zarezerwowane dwie kolejne strony pamięci położone poza obszarem zajmowanym przez program. Na każdej stronie można zapisać 256 wartości licznika. Kasowanie strony będzie następowało po jej zapełnieniu i zapisaniu pierwszej wartości na drugiej stronie. W ten sposób będziemy mieli gwarancję, że pamięć będzie zawsze zawierała ważną wartość licznika. W rozwiązaniu produkcyjnym należałoby wzmocnić bezpieczeństwo danych przez dodanie zabezpieczenia przed błędnym zapisem, np. poprzez wprowadzenie sumy kontrolnej; w naszym przykładzie pominiemy to zabezpieczenie.

Kasowanie i zapis pamięci Flash

Operacje na pamięci Flash są wykonywane za pośrednictwem sterownika pamięci, którego rejestry sterujące są reprezentowane w **języku** C przez strukturę `FLASH`. Przed wykonaniem operacji kasowania lub programowania niezbędne jest włączenie modułu `FLASH` w rejestrze `RCC>AHBENR` oraz odblokowanie dostępu do pamięci przez kolejny zapis dwóch kluczy do rejestru `FLASH->KEYR`. Wartości kluczy są zdefiniowane w pliku `stm32f0xx.h` pod nazwami `FLASH_FKEY1` i `FLASH_FKEY2`.

Operacje na pamięci są inicjowane przez zapis rejestru `FLASH->CR`.

W celu skasowania strony należy:

- zapisać do rejestru `FLASH->CR` słowo z ustawionym bitem `PER`;
- zapisać do rejestru `FLASH->AR` adres należący do kasowanej strony;
- zapisać do rejestru `FLASH->CR` słowo z ustawionym bitem `STRT`.

Listing 1. Program demonstracyjny

```
/*
  STM32F0 tutorial
  PVD and Flash data storage demo
  gbm, 04'2014
*/

#include "stm32f0discovery.h"
//=================================================
#define SYSCLK_FREQ         (HSI_VALUE)
#define SYSTICK_FREQ        100
#define BAUD_RATE 115200

#define DFLASH_ADDR         0x08003800
#define dflash16    ((uint16_t *)DFLASH_ADDR)
#define dflash_pg0  ((uint32_t *)DFLASH_ADDR)
#define dflash_pg1  ((uint32_t *)(DFLASH_ADDR + 1024))
//=================================================
void SystemInit(void)
{
}
```

Listing 1. c.d.

```
//=====================================================
static const struct init_entry_ init_table[] =
{
    // enable peripherals
    {&RCC->APB1ENR, RCC_APB1ENR_PWREN},
    {&RCC->APB2ENR, RCC_APB2ENR_USART1EN},
    {&RCC->AHBENR, RCC_AHBENR_GPIOAEN      // UART
        | RCC_AHBENR_DMA1EN | RCC_AHBENR_RSTVAL},
    // port setup
    {&GPIOA->AFR[1], BF4(10, 1) | BF4(9, 1)}, // USART pins 10 - RX, 9 - TX
    {&GPIOA->MODER, GPIOA_MODER_SWD
        | BF2(10, GPIO_MODER_AF) | BF2(9, GPIO_MODER_AF)},  // UART pins as AF
    // Flash unlock
    {&FLASH->KEYR, FLASH_FKEY1},
    {&FLASH->KEYR, FLASH_FKEY2},
    // 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_UE},  // enable
    //SysTick setup
    {&SCB->SHP[1], 0x80c00000},  // PendSV lowest priority, Systick higher
    {&SysTick->LOAD, SYSCLK_FREQ / SYSTICK_FREQ - 1},
    {&SysTick->VAL, 0},
    {&SysTick->CTRL, SysTick_CTRL_CLKSOURCE_Msk | SysTick_CTRL_TICKINT_Msk | SysTick_CTRL_ENABLE_Msk},
    // PVD setup
    {&PWR->CR, PWR_CR_PLSV(5) | PWR_CR_PVDE},  // 2.58 V, enable
    // interrupts and sleep
    {&EXTI->RTSR, 1 << 16},  // PVD
    {&EXTI->IMR, 1 << 16},  // PVD
    {&NVIC->ISER[0], 1 << PVD_IRQn},  // enable interrupts
    {&SCB->SCR, SCB_SCR_SLEEPONEXIT_Msk},  // sleep while not in handler
    {0, 0}
};
//=========================================================
static volatile uint32_t prev_up_time = 0, up_time = 0;
static uint32_t idx = 0;  // flash data index
static uint8_t time[11] = "123456789\r\n";
//=========================================================
int main(void)
{
    writeregs(init_table);
    DMA1_Channel2->CMAR = (uint32_t)time;
    DMA1_Channel2->CPAR = (uint32_t)&USART1->TDR;
    if (dflash_pg0[0] != 0xffffffff && dflash_pg1[0] != 0xffffffff)
    {
    // both pages in use - one should be erased
        FLASH->CR = FLASH_CR_PER;
        // select page to erase
        FLASH->AR = dflash_pg0[0] < dflash_pg1[0] ?
            (uint32_t)dflash_pg0 : (uint32_t)dflash_pg1;
        FLASH->CR = FLASH_CR_STRT;
        FLASH->SR = FLASH_SR_EOP;
    }
    if (dflash_pg1[0] != 0xffffffff)
    idx = 256;  // use page 1
    while (idx < 512 && dflash_pg0[idx] != 0xffffffff)
    prev_up_time = dflash_pg0[idx ++];
    if (idx == 512) idx = 0;
    up_time = prev_up_time;
    __WFI();  // go to sleep
}
//=========================================================
void PVD_IRQHandler(void)
{
    EXTI->PR = 1 << 16;  // clear interrupt rq
    if (up_time != prev_up_time)
    {
        // count changed - write Flash
        FLASH->CR = FLASH_CR_PG;
        dflash16[idx * 2] = up_time;
        FLASH->SR = FLASH_SR_EOP;
        FLASH->CR = FLASH_CR_PG;
        dflash16[idx * 2 + 1] = up_time >> 16;
        FLASH->SR = FLASH_SR_EOP;
        prev_up_time = up_time;
    }
}
//=========================================================
void SysTick_Handler(void)
{
    static uint8_t sdiv = 50;
    if (++ sdiv == SYSTICK_FREQ)
    {
        uint32_t t;
        int i;
        sdiv = 0;
        t = ++ up_time;
        // show
        for (i = 8; i >= 0; i --)
        {
            time[i] = t % 10 + '0';
            t /= 10;
        }
        // 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;
    }
}
```

W celu zapisu słowa 16-bitowego do pamięci należy zapisać do rejestru FLASH->CR słowo z ustawionym bitem PG, a następnie zapisać programowaną wartość słowa 16-bitowego pod adres, pod którym ma ono być umieszczone w pamięci Flash.

Operacja kasowania może trwać do 40 ms, a zapisu słowa – do 60 μs. Ponieważ program jest wykonywany z pamięci Flash, jego wykonanie zostaje wstrzymane na czas operacji – dlatego w programie nie jest sprawdzany bit BSY w rejestrze FLASH->SR. Stan ostatnio zleconej operacji można odczytać z rejestru FLASH->SR po jej zakończeniu. Bit EOP oznacza jej pomyślne zakończenie. Po zakończeniu operacji należy ten bit wyzerować, zapisując do FLASH->SR słowo z ustawionym bitem EOP.

Monitor napięcia zasilania

Monitor napięcia zasilania jest częścią modułu PWR. Jego działanie polega na generowaniu pojedynczego sygnału PVDO, który przyjmuje wartość 0 gdy napięcie zasilania przekracza zaprogramowaną wartość progową, a 1 - gdy napięcie zasilania spadnie poniżej określonej granicy. Komparator napięcia charakteryzuje się histerezą ok. 100 mV.

Dostęp do rejestrów modułu PWR wymaga włączenia w rejestrze RCC->APB1ENR. Monitor zasilania jest programowany poprzez zapis do rejestru PWR->CR. Ustawienie bitu PVDE powoduje jego włączenie, a 3-bitowe pole PLS zawiera zadaną wartość napięcia progowego. Odwzorowanie wartości pola w poziomy napięć jest opisane w dokumencie *STM32F051xx Datasheet*.

Ponieważ na płytce STM32F0DISCOVERY mikrokontroler jest zasilany napięciem ok. 3 V, w przykładzie zaprogramowano detektor na nominalne napięcie progowe 2,58 V.

Aktualną wartość wyjścia PVD można odczytać programowo z bitu PVDO rejestru PWR->CSR. Zmiana tej wartości może również spowodować zgłoszenia przerwania za pośrednictwem modułu EXTI.

Sterownik przerwań EXTI

Sterownik przerwań EXTI pośredniczy w generowaniu sygnałów zgłoszenia przerwań z tych bloków mikrokontrolera, które mogą pracować w stanach głębokiego uśpienia i mogą powodować wybudzenie mikrokontrolera. Jednym z takich bloków jest monitor napięcia zasilania, dlatego uaktywnienie przerwania PVD wymaga zaprogramowania zarówno sterownika EXTI, jaki i głównego sterownika przerwań rdzenia Cortex-M0 – NVIC. Sygnał stanu detektora PVDO jest doprowadzony na wejście EXTI o numerze 16. W celu umożliwienia zgłoszenia przerwania przy spadku napięcia zasilania, sygnalizowanego zmianą stanu PVDO z 0 na 1, należy:

- ustawić bit 16 rejestru zezwolenia na zgłoszenie przerwania od narastającego zbocza sygnału – EXTI->RTSR;
- ustawić bit 16 rejestru zezwolenia na zgłoszenie przerwań z poszczególnych źródeł – EXTI>IMR;
- włączyć przerwanie PVD w sterowniku przerwań NVIC.

Procedura obsługi przerwania PVD musi jawnie skasować zgłoszenie przerwania poprzez zapis do rejestru EXTI->PR słowa z ustawionym bitem 16.

Działanie programu

Plik źródłowy program przedstawiono na **listingu 1**. Program demonstracyjny odmierza czas swojego działania i wysyła go co sekundę w postaci znakowej przez UART, a przy zaniku zachowuje licznik czasu pracy w pamięci Flash.

Do odmierzania czasu użyto timera systemowego SystTick. zaprogramowanego na częstotliwość 100 Hz. Podobnie, jak w kilku poprzednich projektach, moduł UART został zaprogramowany tak, że wyjście TXD (PA9) mikrokontrolera może zostać połączone bezpośrednio z wejściem RXD interfejsu RS232 (styk 2 złącza DB9), bez potrzeby stosowania translatora poziomów. Ponieważ interfejs USB zapewnia połączenia mas, w warunkach doświadczalnych można użyć pojedynczego przewodu. Wartość licznika można wyświetlić na komputerze PC w programie terminala, skonfigurowanym na parametry transmisji 115200, 8, n, 1.

Podczas inicjowania następuje obniżenie priorytetu przerwania SysTick, tak, aby przerwanie od zaniku zasilania powodowało natychmiastowe wywołanie jego obsługi i zapis danych do pamięci Flash. Adres fragmentu pamięci Flash używanego do zapisu danych określa stała DFLASH_ADDR.

Po zaprogramowaniu rejestrów peryferiali program najpierw sprawdza, czy wartości licznika są zapisane na obu stronach pamięci Flash używanych do przechowywania danych. Może to mieć miejsce tylko wtedy, gdy przy poprzednim wyłączeniu wartość licznika została zapisana jako pierwsza na nowej stronie. W takim przypadku program kasuje stronę zawierającą starsze (mniejsze) wartości licznika.

Następnie program sprawdza, która strona zawiera zapisane wartości licznika, po czym odczytuje z tej strony kolejne wartości tak długo, dopóki nie znajdzie pustego słowa (o wartości 0xFFFFFFFF). Zmienna *prev_up_time*, przechowująca wartość czasu zapisaną w pamięci Flash, zostaje zainicjowana ostatnią znalezioną wartością niepustą; jednocześnie zmienna idx zostaje ustawiona tak, by wskazywała ona pierwsze puste słowo pamięci, w którym będzie zapisana nowa wartość licznika. Wartość *prev_up_time* służy do zainicjowania zmiennej *up_time*, która służy jako licznik sekund czasu pracy. Odliczanie czasu następuje w procedurze obsług przerwania timera *SysTick*. Po zliczeniu każdej sekundy następuje konwersja wartości czasu pracy do postaci znakowej i wysłanie go przez UART przy użyciu DMA.

Spadek napięcia zasilania powoduje zgłoszenie przerwania PVD i wywołanie jego obsługi. Jeżeli bieżąca wartość licznika czasu różni się od wartości przechowanej w pamięci Flash, w procedurze obsługi przerwania następuje zapis do pamięci nowej wartości w dwóch porcjach po 16 bitów.

Jeżeli wartość napięcia zasilania wróci do normy, nie spowoduje to żadnych negatywnych skutków. Przy kolejnym spadku nastąpi ewentualny kolejny zapis, o ile licznik czasu pracy zmienił swoją wartość.

Grzegorz Mazur