ISIX-RTOS

Podstawy obsługi wątków

Artykuł poświęcony prostemu systemowi operacyjnemu ISIX-RTOS, jaki opublikowaliśmy w EP3/2010, zainteresował wielu Czytelników. Wracamy więc do tego tematu, pokazując przykład jego praktycznego wykorzystania z zestawem STM32Butterfly – popularną platformą sprzętową propagowaną przez firmę STMIcroelectronics podczas warsztatów STM32TechDays. Pokażemy sposób przygotowania kodu startowego dla mikrokontrolera oraz dwa wątki, które będą wykorzystane do sterowania diodami LED.

Start systemu zrealizowany jest w bibliotece *lib-stm32*, odpowiedzialnej za inicjalizację systemu zgodnie z wymogami ANSI C/C++. Przed wywołaniem globalnych konstruktorów obiektów wywoływana jest funkcja *__external_startup()*, następnie funkcja główna *main()*.

List. 1. Kod funkcji *__external_startup*
```
void __external_startup(void)
{

        //Initialize system perhipheral
        uc_periph_setup();

        //1 bit for preemtion priority
        nvic_priority_group(NVIC_PriorityGroup_1);

        //System priorities
        nvic_set_priority(PendSV_IRQn,1,0x7);

        //System priorities
        nvic_set_priority(SVCall_IRQn,1,0x7);

        //Set timer priority
        nvic_set_priority(SysTick_IRQn,1,0x7);

        //Initialize isix
        isix::isix_init(ISIX_NUM_PRIORITIES);

        //Setup the systick timer
        timer_setup();
}
```

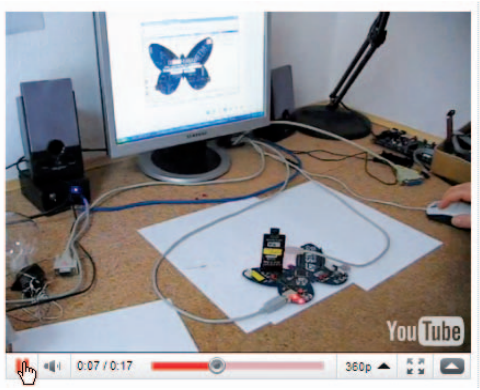
Kod funkcji *__external_startup* pokazano na **list. 1**.

Funkcja u*c_periph_setup()* jest też odpowiedzialna za konfigurację kontrolera pamięci oraz ustawienie pętli PLL mikrokontrolera tak, aby rdzeń był taktowany z maksymalną dozwoloną częstotliwością (72 MHz). Następnie konfigurowany jest kontroler przerwań *NVIC*, w trybie jeden bit priorytetu przerwania oraz trzy bity podpriorytetu. Dodatkowo ustalono priorytet przerwań

List. 2. Główna funkcja *main()*
```
//App main entry point
int main()
{
        //The blinker class
        static app::ledblink led_blinker;
        //The ledkey class
        static app::ledkey led_key;

        //Start the isix scheduler
        isix::isix_start_scheduler();
}
```

Dodatkowe informacje na temat systemu ISIX-RTOS oraz prezentacje multimedialne są dostępne na stronie www.stm32.eu



używanych przez system ISIX-RTOS (*PENDSV, SVC, SYSTICK*) na najniższy możliwy. Kolejna wywoływana funkcja *isix_init* jest odpowiedzialna za inicjalizację systemu, jako argument przyjmuję ona liczbę priorytetów wykorzystywanych przez scheduler. Liczba dostępnych priorytetów jest dowolna (ograniczona wielkością dostępnej pamięci), w praktyce wystarcza najczęściej kilkanaście priorytetów. Następnie wywoływana jest funkcja *timer_setup()*, która jest odpowiedzialna za konfigurację przerwania zegarowego SYSTICK, tak aby było ono generowane z częstotliwością *ISIX_HZ*. Wartość tej częstotliwości w przykładzie ustalono na 1000 Hz. Po wykonaniu funkcji startowej oraz wywołaniu konstruktorów globalnych, wykonywana jest funkcja główna *main()* (**list. 2**).

Najpierw jest tworzony obiekt klasy *ledblink*, której zadaniem jest cykliczne miganie diodą LED1 (patrz dokumentacja zestawu *STM32Butterfly*), następnie obiekt klasy *ledkey*, której zadaniem jest cykliczna zmiana stanu diody LED2 w wyniku naciśnięcia manipulatora joysticka. Następnie jest wywoływana funkcja

List. 3. Implementacja klasy *blinker*
```
/* ------------------------------------------------------
------------ */
//Default constructor, construct base object
ledblink::ledblink():task_base(STACK_SIZE,TASK_PRIO)
{
        //Enable PE in APB2
        RCC->APB2ENR |= RCC_APB2Periph_GPIOE;
        io_config(LED_PORT,LED_PIN,GPIO_MODE_10MHZ,GPIO_
CNF_GPIO_PP);
}


/* ------------------------------------------------------
------------ */
//Main task/thread function
void ledblink::main()
{
        while(true)
        {
                //Enable LED
                io_clr( LED_PORT, LED_PIN );
                //Wait time
                isix::isix_wait( isix::isix_
ms2tick(BLINK_TIME) );
                //Disable LED
                io_set( LED_PORT, LED_PIN );
                //Wait time
                isix::isix_wait( isix::isix_
ms2tick(BLINK_TIME) );
        }
}
```

List. 4. Fragmenty implementacji klasy *ledkey*

```
/Default constructor initialize GPIO and var
ledkey::ledkey():task_base(STACK_SIZE,TASK_PRIO),is_
enabled(false)
{
        //Enable PE in APB2
        RCC->APB2ENR |= RCC_APB2Periph_GPIOE;
        io_config(LED_PORT,LED_PIN,GPIO_MODE_10MHZ,GPIO_
CNF_GPIO_PP);
}

/* ------------------------------------------------------
------------ */
void ledkey::main()
{
        //Last key state
        bool p_state = true;
        //Task/thread main loop
        while(true)
        {
                //Change state on rising edge
                if(io_get(KEY_PORT, KEY_PIN) && !p_
state)
                {
                        is_enabled = !is_enabled;
                }
                //Get previous state
                p_state = io_get(KEY_PORT, KEY_PIN);
                //If enabled change state
                if(is_enabled) io_clr( LED_PORT, LED_PIN
);
                else io_set( LED_PORT, LED_PIN );
                //Wait short time
                isix::isix_wait( isix::isix_
ms2tick(DELAY_TIME) );
        }
}
```

isix_start_scheduler(), która powoduje uruchomienie szeregowania zadań przez system ISIX-RTOS. Obie klasy (*ledblink, ledkey*) dziedziczą z klasy bazowej *isix::task_base*. Każda klasa dziedzicząca z klasy *task_base* musi implementować metodę wirtualną *main()*, która jest wykonywana w oddzielnym wątku tworzonym przez konstruktor klasy *task_base*. Implementację klasy *blinker* odpowiedzialnej za cykliczne miganie diodą LED1 przedstawiono na **list. 3**.

W liście inicjalizacyjnej konstruktora wywoływany jest konstruktor klasy bazowej *task_base()*, który jest odpowiedzialny za tworzenie nowego zadania (wątku). Jako argumenty przyjmuje on rozmiar stosu zadania oraz jego priorytet. W konstruktorze jest także konfigurowany port GPIO PE.14, do którego w zestawie *STM-32Butterfly* podłączono diodę LED1. Wątek realizowany jest w pętli nieskończonej metody wirtualnej *main()*. Wątek zmienia stan LED, a następnie wywołuje funkcję *isix_wait()*, której zadaniem jest uśpienie wątku na czas określony przez stałą *BLINK_TIME*.

Najistotniejsze fragmenty implementacji klasy *ledkey* przedstawiono na **list. 4**.

Podobnie jak poprzednio w liście inicjalizacyjnej jest wywoływany konstruktor klasy bazowej oraz inicjalizowane są porty GPIO. Wątek odpowiedzialny za wspomniane wcześniej zadanie zrealizowany jest przez metodę wirtualną *main()* w pętli nieskończonej. Zmiana stanu diody LED2 następuje w momencie puszczenia manipulatora joysticka (zbocze narastające na wejściu mikrokontrolera). Detekcję zrealizowano w sposób klasyczny przez porównanie bieżącego stanu klawisza ze stanem poprzednim. W momencie, gdy wykryjemy zbocze, jest zmieniany stan zmiennej *is_enabled*, następnie na podstawie stanu zmiennej dioda LED2 jest włączana lub wyłączana. Na koniec cyklu wywoływana jest funkcja *isix_wait()* z argumentem DELAY_TIME (25 ms), która usypia bieżący wątek na krótki czas, tak aby wyeliminować drgania zestyków.

Lucjan Bryndza, EP
lucck@boff.pl