Геращенко А.Ю.
Дк-31мп

Приклад

```
.include    "address_map_arm.s"

.text
.global     _start
_start:
        LDR     R1, =LED_BASE   /* Address of red LEDs. */
        LDR     R2, =SW_BASE    /* Address of switches. */
LOOP:
        LDR     R3, [R2]        /* Read the state of switches. */
        STR     R3, [R1]        /* Display the state on LEDs. */
        B       LOOP
.end
```

Даний код виконує ввімкнення світлодіоду LEDR при ввімкненні відповідного свіча SW.

Спочатку в регістр R1 завантажується базова адреса регістру світлодіодів, в регістр R2 завантажується базова адреса регістру свічів. Потім починає виконуватись цикл LOOP. Команда LDR виконує завантаження в регістр R3 значення, яке знаходиться за адресою, що завантажена в регістр R2. Команда STR записує значення з регістру R3 за адресою з регістру R1. Наступна команда B LOOP виконує безумовний перехід на мітку LOOP і цикл повторюється.

Завдання 1

```
#include "address_map_arm.h"
/* This program demonstrates use of parallel ports in the Computer System
 *
 * It performs the following:
 *  1. displays a rotating pattern on the green LED
 *  2. if a KEY is pressed, uses the SW switches as the pattern
*/
int main(void) {
    /* Declare volatile pointers to I/O registers (volatile means that IO load
     * and store instructions will be used to access these pointer locations,
     * instead of regular memory loads and stores)
    */
    volatile int * LED_ptr       = (int *)LED_BASE; // LED address
    volatile int * SW_switch_ptr = (int *)SW_BASE;  // SW slider switch address
    volatile int * KEY_ptr       = (int *)KEY_BASE; // pushbutton KEY address

    int LED_bits = 0x0F0F0F0F; // pattern for LED lights
    int SW_value, KEY_value;
```

```
    volatile int
        delay_count; // volatile so the C compiler doesn't remove the loop

    while (1) {
        SW_value = *(SW_switch_ptr); // read the SW slider (DIP) switch values

        KEY_value = *(KEY_ptr); // read the pushbutton KEY values
        if (KEY_value != 0)     // check if any KEY was pressed
        {
            /* set pattern using SW values */
            LED_bits = SW_value | (SW_value << 8) | (SW_value << 16) |
                       (SW_value << 24);
            while (*KEY_ptr)
                ; // wait for pushbutton KEY release
        }
        *(LED_ptr) = LED_bits; // light up the LEDs

        /* rotate the pattern shown on the LEDs */
        if (LED_bits & 0x80000000)
            LED_bits = (LED_bits << 1) | 1;
        else
            LED_bits = LED_bits << 1;

        for (delay_count = 350000; delay_count != 0; --delay_count)
            ; // delay loop
    }
}
```

1) Створюємо вказівники LED_ptr, SW_switch_ptr, KEY_ptr для звернення до регістрі, які відповідають за їх стан (ввімкнено/вимкнено).

2) Створюємо змінну LED_bits в якій збережено стан діодів за замовченням.

3) Створюємо змінні SW_value для створення функціоналу з використанням свічів, KEY_value для створення функціоналу з використанням кнопок, delay_count для створення затримки.

4) Заходимо в нескінченний цикл.

Призначаємо змінній SW_value значення стану свічів, яке міститься за адресою в SW_switch_ptr. Призначаємо змінній KEY_value значення стану кнопок, яке міститься за адресою в KEY_ptr.

Далі перевіряється натискання будь-якої кнопки, при натисканні кнопки в регістрі за адресою KEY_ptr буде не 0 значення і виконається умова if(KEY_value != 0). Коли програма зайде під умову if, змінній LED_bits буде призначено значення, отримане після виконання операції бітового «або» над початковим значенням свічів та значенням зі зміщенням на 8, 16 та 24 біти.

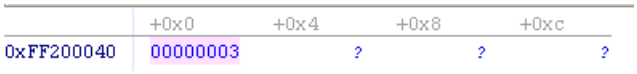
Потім у циклі while(*KEY_ptr) програма очікує поки користувач відпустить кнопку.

Після виходу з if регістру діодів за адресою LED_ptr призначається значення зі змінної LED_bits.

В умові if(LED_bits & 0x80000000) виконується зациклювання старшого біту. Якщо старший біт регістру рівний 1, число зсувається вліво на 1 розряд, 1 зі старшого біту втрачається, бо регістр 32 бітний. Для збереження 1 в молодший розряд виконується операція «або». Якщо умова if не виконується, відбувається зміщення вліво на 1 біт.

Потім описано цикл for, який реалізує затримку в 350000 ітерацій.

Значення в регістрі при ввімкненні перших 2х свічів:



	+0x0	+0x4	+0x8	+0xc
0xFF200040	00000003	?	?	?

Значення в регістрі при натисканні KEY[1]

0xFF200050	00000002	?	?	?

Описана вище програма відображає «бігучий вогник», кількість бігучих вогників можна встановлювати свічами та підтверджувати вибір кнопками.

Завдання 2

Програма interrupt_example демонструє використання переривань з таймером HPS та Interval.

Переривання за таймером HPS вмикає та вимикає світлодіод LEDRG, який підключено до GPIO1.

Переривання за Interval таймером реалізує «бігучий вогник», при кожному натисканні KEY[1] він змінює свій напрям.

```
#include "address_map_arm.h"

void set_A9_IRQ_stack(void);            ////////////////////////exeptions
void config_GIC(void);                  ////////////////////////exeptions
void config_HPS_timer(void);
void config_HPS_GPIO1(void);
void config_interval_timer(void);
void config_KEYs(void);
void enable_A9_interrupts(void);        ////////////////////////exeptions
/* key_dir and pattern are written by interrupt service routines; we have to
 * declare these as volatile to avoid the compiler caching their values in
 * registers */
volatile int tick    = 0; // set to 1 every time the HPS timer expires
```

```c
volatile int key_dir = 0;
volatile int pattern = 0x0F0F0F0F; // pattern for LED lights

int main(void)
{
    volatile int * HPS_GPIO1_ptr = (int *)HPS_GPIO1_BASE; // GPIO1 base address
    volatile int   HPS_timer_LEDG =
        0x01000000; // value to turn on the HPS green light LEDG

    set_A9_IRQ_stack();      // initialize the stack pointer for IRQ mode
    config_GIC();            // configure the general interrupt controller
    config_HPS_timer();      // configure the HPS timer
    config_HPS_GPIO1();      // configure the HPS GPIO1 interface
    config_interval_timer(); // configure Altera interval timer to generate
                             // interrupts
    config_KEYs();           // configure pushbutton KEYs to generate interrupts

    enable_A9_interrupts(); // enable interrupts

    while (1)
    {
        if (tick)
        {
            tick           = 0;
            *HPS_GPIO1_ptr = HPS_timer_LEDG; // turn on/off the green light LEDG
            HPS_timer_LEDG ^= 0x01000000; // toggle the bit that controls
        }
    }
}

/* setup HPS timer */
void config_HPS_timer()
{
    volatile int * HPS_timer_ptr = (int *)HPS_TIMER0_BASE; // timer base address

    *(HPS_timer_ptr + 0x2) = 0; // write to control register to stop timer
    /* set the timer period */
    int counter      = 100000000; // period = 1/(100 MHz) x (100 x 10^6) = 1 sec
    *(HPS_timer_ptr) = counter;   // write to timer load register

    /* write to control register to start timer, with interrupts */
    *(HPS_timer_ptr + 2) = 0b011; // int mask = 0, mode = 1, enable = 1
}

/* setup HPS GPIO1. The GPIO1 port has one green light (LEDG) and one pushbutton
 * KEY connected for the DE1-SoC Computer. The KEY is connected to GPIO1[25],
 * and is not used here. The green LED is connected to GPIO1[24]. */
void config_HPS_GPIO1()
{
    volatile int * HPS_GPIO1_ptr = (int *)HPS_GPIO1_BASE; // GPIO1 base address
```

```
    *(HPS_GPIO1_ptr + 0x1) =
        0x01000000; // write to the data direction register to set
                    // bit 24 (LEDG) to be an output
    // Other possible actions include setting up GPIO1 to use the KEY, including
    // setting the debounce option and causing the KEY to generate an interrupt.
    // We do not use the KEY in this example.
}

/* setup the interval timer interrupts in the FPGA */
void config_interval_timer()
{
    volatile int * interval_timer_ptr =
        (int *)TIMER_BASE; // interal timer base address

    /* set the interval timer period for scrolling the HEX displays */
    int counter                 = 5000000; // 1/(100 MHz) x 5x10^6 = 50 msec
    *(interval_timer_ptr + 0x2) = (counter & 0xFFFF);
    *(interval_timer_ptr + 0x3) = (counter >> 16) & 0xFFFF;

    /* start interval timer, enable its interrupts */
    *(interval_timer_ptr + 1) = 0x7; // STOP = 0, START = 1, CONT = 1, ITO = 1
}

/* setup the KEY interrupts in the FPGA */
void config_KEYs()
{
    volatile int * KEY_ptr = (int *)KEY_BASE; // pushbutton KEY address

    *(KEY_ptr + 2) = 0x2; // enable interrupts for KEY[1]
}
```

```
32      volatile int * HPS_GPIO1_ptr = (int *)HPS_GPIO1_BASE; // GPIO1 base address
33      volatile int   HPS_timer_LEDG =
34          0x01000000; // value to turn on the HPS green light LEDG
```

У 32 рядку створюється вказівник на базову адресу GPIO1 до якого підключено зелений світлодіод.

У рядку 33 створюється змінна для керування станом світлодіоду, яка встановлює 24й біт в 1 для ввімкнення світіння.

```
36      set_A9_IRQ_stack();      // initialize the stack pointer for IRQ mode
37      config_GIC();            // configure the general interrupt controller
38      config_HPS_timer();      // configure the HPS timer
39      config_HPS_GPIO1();      // configure the HPS GPIO1 interface
40      config_interval_timer(); // configure Altera interval timer to generate
41                               // interrupts
42      config_KEYs();           // configure pushbutton KEYs to generate interrupts
43
44      enable_A9_interrupts(); // enable interrupts
```

У 36 рядку функція set_A9_IRQ_stack ініціалізує стек для обробки переривань.

```
/*
 * Initialize the banked stack pointer register for IRQ mode
*/
void set_A9_IRQ_stack(void)
{
    int stack, mode;
    stack = A9_ONCHIP_END - 7; // top of A9 onchip memory, aligned to 8 bytes
    /* change processor to IRQ mode with interrupts disabled */
    mode = INT_DISABLE | IRQ_MODE;
    asm("msr cpsr, %[ps]" : : [ps] "r"(mode));
    /* set banked stack pointer */
    asm("mov sp, %[ps]" : : [ps] "r"(stack));

    /* go back to SVC mode before executing subroutine return! */
    mode = INT_DISABLE | SVC_MODE;
    asm("msr cpsr, %[ps]" : : [ps] "r"(mode));
}
```

mode = INT_DISABLE | IRQ_MODE; Формується режим для процесора, де IRQ біт увімкнутий, а INT біт вимкнутий. Це означає, що переривання дозволені, але у цьому конкретному режимі (IRQ) вони вимкнені.

asm("msr cpsr, %[ps]" : : [ps] "r"(mode)); Використовується асемблерна вбудована функція для запису значення режиму mode у регістр статусу програми (cpsr) процесора ARM.

asm("mov sp, %[ps]" : : [ps] "r"(stack)); Встановлюється значення стеку у регістрі стеку (sp) за допомогою асемблерної інструкції.

mode = INT_DISABLE | SVC_MODE; Знову формується режим для процесора, тепер для повернення до режиму обслуговування (SVC mode).

asm("msr cpsr, %[ps]" : : [ps] "r"(mode)); Повертається до режиму обслуговування за допомогою асемблерної інструкції.

У 37 рядку конфігурується GIC контролер переривань.

```
 * Configure the Generic Interrupt Controller (GIC)
*/
void config_GIC(void)
{
    int address; // used to calculate register addresses

    /* configure the HPS timer interrupt */
    *((int *)0xFFFED8C4) = 0x01000000;
    *((int *)0xFFFED118) = 0x00000080;

    /* configure the FPGA interval timer and KEYs interrupts */
    *((int *)0xFFFED848) = 0x00000101;
    *((int *)0xFFFED108) = 0x00000300;

    // Set Interrupt Priority Mask Register (ICCPMR). Enable interrupts of all
    // priorities
    address           = MPCORE_GIC_CPUIF + ICCPMR;
    *((int *)address) = 0xFFFF;

    // Set CPU Interface Control Register (ICCICR). Enable signaling of
    // interrupts
    address           = MPCORE_GIC_CPUIF + ICCICR;
    *((int *)address) = ENABLE;

    // Configure the Distributor Control Register (ICDDCR) to send pending
    // interrupts to CPUs
    address           = MPCORE_GIC_DIST + ICDDCR;
    *((int *)address) = ENABLE;
}
```

У 38 рядку налаштовується HPS таймер.

```
/* setup HPS timer */
void config_HPS_timer()
{
    volatile int * HPS_timer_ptr = (int *)HPS_TIMER0_BASE; // timer base address

    *(HPS_timer_ptr + 0x2) = 0; // write to control register to stop timer
    /* set the timer period */
    int counter      = 100000000; // period = 1/(100 MHz) x (100 x 10^6) = 1 sec
    *(HPS_timer_ptr) = counter;   // write to timer load register

    /* write to control register to start timer, with interrupts */
    *(HPS_timer_ptr + 2) = 0b011; // int mask = 0, mode = 1, enable = 1
}
```

У відповідні комірки регістрів нульового HPS таймеру вносяться налаштування. Спочатку таймер зупиняється (E встановити в 0), потім записують значення для відліку в Load register, в M записують 1 щоб дати команду використовувати встановлене значення відліку, вмикають таймер

шляхом запису в E 1. В I записується 0 щоб викликати переривання коли лічильник дорахує до 0.

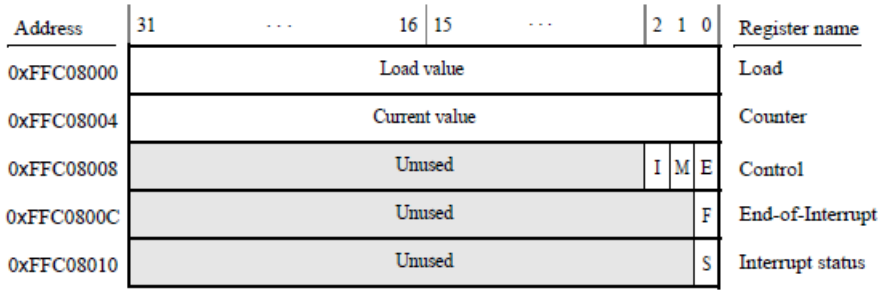


Figure 4. HPS timer port.

У 39 рядку налаштовується GPIO1 для керування світлодіодом.

```
void config_HPS_GPIO1()
{
    volatile int * HPS_GPIO1_ptr = (int *)HPS_GPIO1_BASE; // GPIO1 base address

    *(HPS_GPIO1_ptr + 0x1) =
        0x01000000; // write to the data direction register to set
                    // bit 24 (LEDG) to be an output
    // Other possible actions include setting up GPIO1 to use the KEY, including
    // setting the debounce option and causing the KEY to generate an interrupt.
    // We do not use the KEY in this example.
}
```

У 24й біт регістру Data direction register встановлюється 1 щоб налаштувати порт GPIO1 як вихідний.

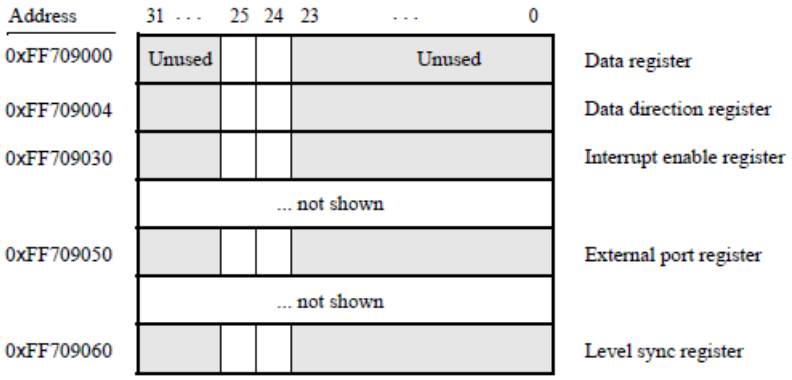


Figure 2. Parallel port GPIO1.

У 40 рядку налаштовується інтервальний таймер FPGA.

```c
/* setup the interval timer interrupts in the FPGA */
void config_interval_timer()
{
    volatile int * interval_timer_ptr =
        (int *)TIMER_BASE; // interal timer base address

    /* set the interval timer period for scrolling the HEX displays */
    int counter                 = 5000000; // 1/(100 MHz) x 5x10^6 = 50 msec
    *(interval_timer_ptr + 0x2) = (counter & 0xFFFF);
    *(interval_timer_ptr + 0x3) = (counter >> 16) & 0xFFFF;

    /* start interval timer, enable its interrupts */
    *(interval_timer_ptr + 1) = 0x7; // STOP = 0, START = 1, CONT = 1, ITO = 1
}
```

Встановлюється значення для відліку таймера у 50 мс після чого генерується переривання. Всі налаштування виконуються шляхом встановлення відповідних значень у біти Interval timer registers.

Address	31 ... 17	16	15 ... 3	2	1	0	
0xFF202000	Not present (interval timer has 16-bit registers)		Unused		RUN	TO	Status register
0xFF202004			Unused	STOP START	CONT	ITO	Control register
0xFF202008			Counter start value (low)				
0xFF20200C			Counter start value (high)				
0xFF202010			Counter snapshot (low)				
0xFF202014			Counter snapshot (high)				

Figure 13. Interval timer registers.

У 42 рядку налаштовуються переривання від кнопки KEY1.

```c
/* setup the KEY interrupts in the FPGA */
void config_KEYs()
{
    volatile int * KEY_ptr = (int *)KEY_BASE; // pushbutton KEY address

    *(KEY_ptr + 2) = 0x2; // enable interrupts for KEY[1]
}
```

У регістр для конфігурування переривань з кнопок записується 0x2, що активує переривання для кнопки KEY1.

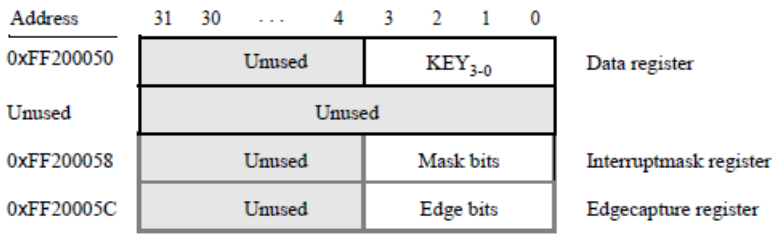


Figure 15. Registers used for interrupts from the pushbutton parallel port.

```
while (1)
{
    if (tick)
    {
        tick           = 0;
        *HPS_GPIO1_ptr = HPS_timer_LEDG; // turn on/off the green light LEDG
        HPS_timer_LEDG ^= 0x01000000; // toggle the bit that controls LEDG
    }
}
```

У нескінченному циклі відбувається миготіння зеленим світлодіодом коли спрацювало HPS переривання.

Обробник HPS переривання:

```
/*****************************************************************************
 * HPS timer0 interrupt service routine
 *
 * This code increments the tick variable, and clears the interrupt
 *****************************************************************************/
void HPS_timer_ISR()
{
    volatile int * HPS_timer_ptr = (int *)HPS_TIMER0_BASE; // HPS timer address

    ++tick; // used by main program

    *(HPS_timer_ptr + 3); // Read timer end of interrupt register to
                          // clear the interrupt
    return;
}
```

++tick; Збільшується значення змінної tick, для підрахунку кількості переривань, що сталися з моменту останнього зчитування цієї змінної.

*(HPS_timer_ptr + 3); Здійснюється читання регістра "end of interrupt" таймера, щоб очистити переривання таймера. Це необхідно для того, щоб переривання не постійно повторювалися після їх обробки.

Обробник переривань інтервального таймеру:

```c
void interval_timer_ISR()
{
    volatile int * interval_timer_ptr = (int *)TIMER_BASE;
    volatile int * LED_ptr            = (int *)LED_BASE; // LED address

    *(interval_timer_ptr) = 0; // Clear the interrupt

    *(LED_ptr) = pattern; // Display pattern on LED

    /* rotate the pattern shown on the LED lights */
    if (key_dir == 0) // for 0 rotate left
        if (pattern & 0x80000000)
            pattern = (pattern << 1) | 1;
        else
            pattern = pattern << 1;
    else // rotate right
        if (pattern & 0x00000001)
        pattern = (pattern >> 1) | 0x80000000;
    else
        pattern = (pattern >> 1) & 0x7FFFFFFF;

    return;
}
```

Цей обробник реалізує «бігучий вогник» вліво або вправо, в залежності від значення key_dir, яке встановлюється обробником переривань кнопки.

Обробник переривань кнопки:

```c
/*****************************************************************************
 * Pushbutton - Interrupt Service Routine
 *
 * This routine toggles the key_dir variable from 0 <-> 1
*****************************************************************************/
void pushbutton_ISR(void)
{
    volatile int * KEY_ptr = (int *)KEY_BASE;
    int            press;

    press          = *(KEY_ptr + 3); // read the pushbutton interrupt register
    *(KEY_ptr + 3) = press;          // Clear the interrupt

    key_dir ^= 1; // Toggle key_dir value

    return;
}
```

Завдання 3

```c
#include "address_map_arm.h"

#define bit_24_pattern 0x01000000
```

```c
/* This program provides a simple example of code for the ARM A9. The
 * program performs the following:
 *  1. starts the ARM A9 private timer
 *  2. loops indefinitely, toggling the green light LEDG when the timer expires
*/
int main(void)
{
    /* Declare volatile pointers to I/O registers (volatile means that the
     * locations will not be cached, even in registers) */
    volatile int * HPS_GPIO1_ptr            = (int *)HPS_GPIO1_BASE;
    volatile int * MPcore_private_timer_ptr = (int *)MPCORE_PRIV_TIMER;

    int HPS_LEDG = bit_24_pattern; // value to turn on the HPS green light LEDG
    int counter  = 900000000;      // timeout = 1/(200 MHz) x 200x10^6 = 1 sec

    *(HPS_GPIO1_ptr + 1) =
        bit_24_pattern; // write to the data direction register to set
                        // bit 24 (LEDG) of GPIO1 to be an output
    *(MPcore_private_timer_ptr)     = counter; // write to timer load register
    *(MPcore_private_timer_ptr + 2) = 0b011;   // mode = 1 (auto), enable = 1

    while (1)
    {
        *HPS_GPIO1_ptr = HPS_LEDG; // turn on/off LEDG
        while (*(MPcore_private_timer_ptr + 3) == 0)
            ;                                  // wait for timer to expire
        *(MPcore_private_timer_ptr + 3) = 1; // reset timer flag bit
        HPS_LEDG ^= bit_24_pattern;          // toggle bit that controls LEDG
    }
}
```

Спочатку оголошуються вказівники на базові адреси рігістрів GPIO1 та регістр приватного таймеру. Потім встановлюється значення для вмикання зеленого світлодіоду та значення кількості відліків таймеру.

Далі конфігурується як output 24 біт регістру керування світлодіодом. Встановлюється режим роботи приватного таймеру.

Address	31 ... 16	15 ... 8	7 ... 3	2	1	0	Register name
0xFFFEC600	Load value						Load
0xFFFEC604	Current value						Counter
0xFFFEC608	Unused	Prescaler	Unused	I	A	E	Control
0xFFFEC60C	Unused					F	Interrupt status

Figure 3. ARM A9 private timer port.

У нескінченному циклі відбувається миготіння зеленим світлодіодом, який підключено до GPIO1. Програма очікує завершення роботи таймеру та перезавантажує його для початку лічби знову. Таким чином реалізовано зміну стану світлодіоду коли таймер дорахував до 0.