

БГУИР

Кафедра ЭВМ

Отчет по лабораторной работе № 1  
Цифровой ввод/вывод. Прерывания  
Вариант №4

Выполнил:  
студент группы 950505 Киреев Ю.В.

Проверил:  
Богдан Е.В.

Минск 2022

## **1 ЦЕЛЬ РАБОТЫ**

Ознакомиться с интегрированной средой разработки CodeComposer Studio. Ознакомиться с основными функциональными возможностями платы MSP-EXP430F5529. Изучить приемы работы с цифровыми выводами.

## 2 ИСХОДНЫЕ ДАННЫЕ

Набор заданий:

В соответствии с вариантом задания написать программу, которая бы включала и выключала заданные светодиоды LED\_A и LED\_B в зависимости от комбинации состояния кнопок S1 и S2.

Нажатие и отжатие кнопок должны обрабатываться корректно:

- одно нажатие должно обрабатываться как только одно нажатие (аналогично с отжатием);
- если было несколько нажатий, ни одно не должно быть пропущено (аналогично с отжатием).

Программа должна быть написана в двух вариантах:

- без использования прерываний;
- с использованием прерываний.

Не допускается подключение к проекту каких-либо файлов, за исключением:

- “msp430.h”;
- библиотек языка C;
- написанных самостоятельно.

Для выполнения работы используется плата MSP-EXP430F5529 и интегрированная среда разработки Code Composer Studio.

Вариант	LED A					LED B				
	Номер LED	Включение		Выключение		Номер LED	Включение		Выключение	
		S1	S2	S1	S2		S1	S2	S1	S2
4	6	U	↑	U	↑	5	↑	-	↓	D

Обозначения:

U - кнопка не нажата;

D - кнопка нажата;

– - любое состояние;

↓ - нажатие кнопки;

↑ - отжатие кнопки.

### 3 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

#### 3.1 Плата MSP-EXP430F5529

Экспериментальная плата MSP-EXP430F5529 разработана на основе микроконтроллера MSP430F5529 компании Texas Instruments. Это серия процессоров для обработки смешанных сигналов со сверхнизким энергопотреблением.

Основные особенности архитектуры:

- 16-разрядная ортогональная RISC архитектура;
- Фон-Неймановская адресная шина общей памяти и шина данных памяти;
- 27 (51) команд + 37 расширенных инструкций (20-бит адрес) + 11 адресных инструкций (20-бит операнды, но ограничения в режимах адресации);
- 7 согласованных способов адресации;
- полный программный доступ к регистрам, включая счетчик команд (PC), регистр состояния (SR), указатель стека (SP);
- одноктактные регистровые операции;
- большой размер регистрового файла, уменьшающий количество обращений к памяти;
- 20-битная шина адреса, 16-битная шина данных;
- генератор констант (6);
- пересылки память-память без промежуточного сохранения в регистре;
- гибкая система тактирования;
- несколько режимов пониженного энергопотребления;
- моментальный переход в активный режим (порядка 6 мкс).

Микроконтроллер обладает следующими характеристиками:

- производительность до 25 MIPS;
- напряжение питания 1,8-3,6 В;
- ток утечки вывода 50 нА;
- потребление в режиме хранения данных 0,1 мкА;
- потребление в режиме часов реального времени 2,5 мкА.

Микроконтроллер включает в свой состав:

- флеш-память 128 Кб, SRAM 8 Кб;
- 80 выводов, 63 линии входа/выхода;
- 4 асинхронных 16-разрядных таймера/счетчика (7,5,3,3 регистров захвата соответственно);
- сторожевой таймер (WDT) и таймер часов реального времени (RTC);
- модуль управления питанием PMM с блоками защиты от падений

- напряжения (BOR) и контроля напряжения питания (SVS);
  - универсальный последовательный коммуникационный интерфейс
- USCI 2
- x UART/LIN/IrDA/SPI + 2 x I2C/SPI;
  - 3 канала DMA;
  - умножитель-накопитель MPY 32 x 32 бита;
  - компаратор;
  - 12 разрядный АЦП (ADC 12A), 16 каналов;
  - полноскоростной USB 2.0 (12Мб/с), до 8 линий в/в со встроенным 3,3 В
  - стабилизатором (питание от 5 В шины, обеспечивает ток 12 мА);
  - интерфейс для измерения линейных и угловых перемещений (SIF);
  - LCD контроллер до 128 сегментов;
  - внутренний генератор частоты с цифровым управлением.

## 1.2 Цифровой ввод-вывод

8-разрядные порты P1, P2, P3,...,P8, PJ управляют выводами контроллера. Выводы программируются либо как I/O, либо как вход/выход периферии. Порты могут объединяться в пары: P1 и P2 = PA, P3 и P4 = PB, P5 и P6 = PC, P7 и P8 = PD. При работе с прерываниями порты в пары не объединяются. Для порта могут быть доступны регистры:

- RxIN – чтение данных с вывода;
- RxOUT – установка значения выхода;
- RxDIR – выбор направления: 0 – вход, 1 – выход;
- RxREN – разрешение подтягивающего резистора;
- RxDS – выбор допустимой силы вывода;
- RxSEL – выбор функции вывода: 0 – I/O, 1 – периферия;
- RxIV – генерирует значение для изменения счетчика команд,
- соответствующее прерыванию с максимальным приоритетом;
- RxIES – выбор направления перепада для генерации запроса на
- прерывание: 0 – по фронту, 1 – по спаду;
- RxIE – разрешение прерывания;
- RxIFG – флаг прерывания.

Адреса соответствующих портов представлены в таблице:

Таблица 1.1 — Адреса портов ввода-вывода

№ порта	1	2	3	4	5	6	7	8	J
База	0200h		0220h		0240h		0260h		0320h
RxIN	0	1	0	1	0	1	0	1	0

PxOUT	2	3	2	3	2	3	2	3	2
PxDIR	4	5	4	5	4	5	4	5	4
PxREN	6	7	6	7	6	7	6	7	6
PxDS	8	9	8	9	8	9	8	9	8
PxSEL	A	B	A	B	A	B	A	B	-
PxIV	E	1E	-	-	-	-	-	-	-
PxIES	18	19	-	-	-	-	-	-	-
PxIE	1A	1B	-	-	-	-	-	-	-
PxIFG	1C	1D	-	-	-	-	-	-	-

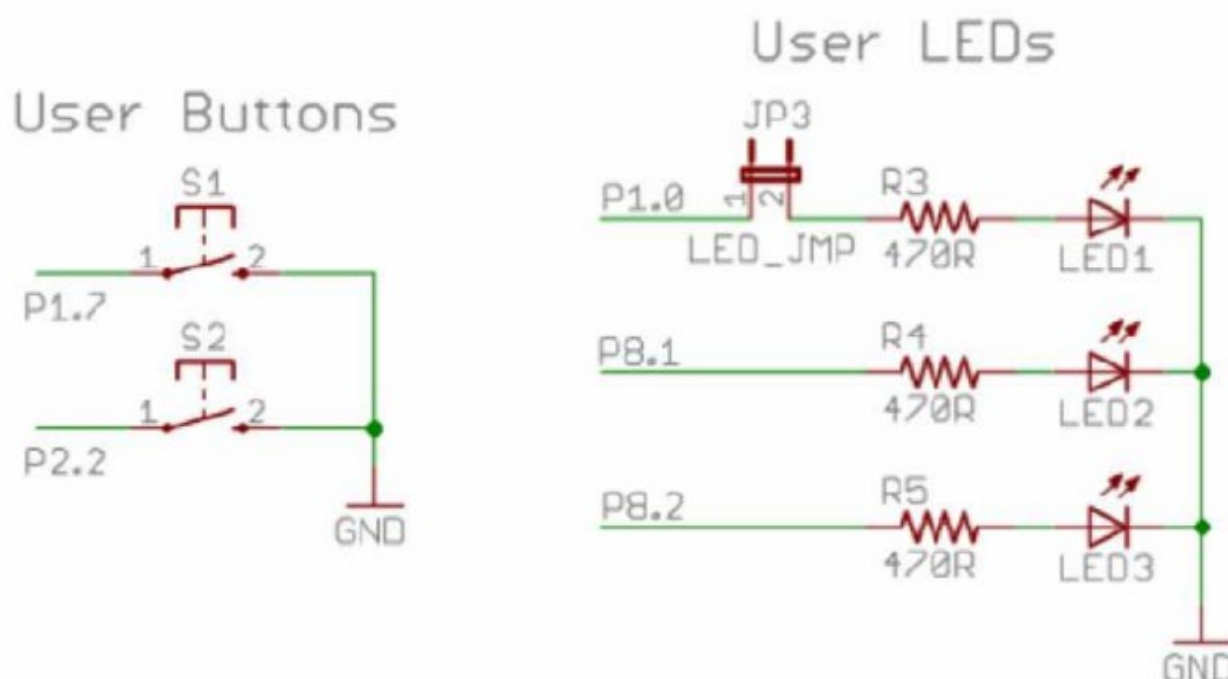


Рис. 1.9 Подключение пользовательских кнопок и светодиодов

Пользователю программно доступны две кнопки S1 и S2, подключенные соответственно к выводу 7 порта 1 и выводу 2 порта 2 (см. рис. 1.9). В дальнейшем такое подключение будем обозначать как P1.7 и P2.2 соответственно. Также программно доступны 8 светодиодов, три из которых (LED1 – LED3, см. рис. 1.9) размещены рядом с кнопками и подключены соответственно к выводам P1.0, P8.1, P8.2. Еще 5 светодиодов (LED4 – LED8) размещаются в блоке сенсорных кнопок и подключены к выводам P1.1 – P1.5 соответственно.

Логика управления выводом на примере порта 1 представлена на рисунке



запускается сторожевой таймер, его следует отключить (иначе через какое-то время сработает сброс).

Константы и определения заданы как для портов, так и для отдельных полей и их значений. Поэтому работа с портами становится максимально удобной для программиста. Так, например, запись `P8DIR |= BIT2`; означает, что в порт `P1DIR`, отвечающий за выбор направления выводов порта 1, заносится новое значение, которое получено логическим ИЛИ его текущего состояния и бита 2. Фактически, это устанавливает бит 2 в заданном порту.

Следует обратить внимание, что при наименовании констант использовались следующие принципы:

- константа, соответствующая биту поля-флага именуется по имени поля, например, полю `CPUOFF` регистра состояния процессора `SR` (бит 4)
- соответствует константа `CPUOFF`;
- константа соответствующая биту `n` в поле `NNN` именуется `NNNn`;
- константа, соответствующая номеру `x` выбранного варианта для поля `NNN` именуется `NNN_x`;
- константа, соответствующая выбранному режиму `zz` для поля `NNN` именуется `NNN__zz`.

Так, например, для 3-битного поля `SELA`, константа, соответствующая 0 биту поля, именована `SELA0`, вариант выбора 0 (`SELA = 000`) именован `SELA_0`, а режим, соответствующий данному варианту именован `SELA__XT1CLK`. В некоторых случаях поля задают делители либо множители, соответствующие степени двойки. Тут надо быть особо внимательным и не спутать похожие мнемоники, например, `NN4` (четвертый бит, т.е. 10000), `NN_4` (четвертый вариант, т.е. 00100), `NN__4` (режим деления на 4, т.е. 00011).

Следует обратить внимание, что регистр `PxOUT` управляет подключением подтягивающего резистора, если вывод сконфигурирован как цифрой I/O, направление — выход, и разрешен подтягивающий резистор. В случае, если вывод сконфигурирован как вывод периферии микроконтроллера, прерывания не генерируются. Отметим также, что после сброса цифровые выводы конфигурируются на вход, кроме того запускается сторожевой таймер в сторожевом режиме.

При написании кода следует учесть несколько моментов. Вначале следует подключить заголовочный файл `mcp430.h`, который в свою очередь подключает файл `mcp430f5529.h`, содержащий необходимые константы в соответствии с архитектурой контроллера. Далее, поскольку после сброса запускается сторожевой таймер, его следует отключить (иначе через какое-то время сработает сброс).

Константы и определения заданы как для портов, так и для отдельных полей и их значений. Поэтому работа с портами становится максимально удобной для программиста. Так, например, запись `P8DIR |= BIT2`; означает, что



в порт P1DIR, отвечающий за выбор направления выводов порта 1, заносится новое значение, которое получено логическим ИЛИ его текущего состояния и бита 2. Фактически, это устанавливает бит 2 в заданном порту. Следует обратить внимание, что при наименовании констант использовались следующие принципы:

- константа, соответствующая биту поля-флага именуется по имени поля,
- например, полю CPUOFF регистра состояния процессора SR (бит 4)
- соответствует константа CPUOFF;
- константа соответствующая биту n в поле NNN именуется NNNn;
- константа, соответствующая номеру x выбранного варианта для поля NNN именуется NNN\_x;
- константа, соответствующая выбранному режиму zz для поля NNN
- именуется NNN\_\_zz.

Так, например, для 3-битного поля SELA, константа, соответствующая 0 биту поля, именована SELA0, вариант выбора 0 (SELA = 000) именован SELA\_0, а режим, соответствующий данному варианту именован SELA\_\_XT1CLK. В некоторых случаях поля задают делители либо множители, соответствующие степени двойки. Тут надо быть особо внимательным и не спутать похожие мнемоники, например, NN4 (четвертый бит, т.е. 10000), NN\_4 (четвертый вариант, т.е. 00100), NN\_\_4 (режим деления на 4, т.е. 00011).

## 4 ВЫПОЛНЕНИЕ РАБОТЫ

### 4.1 Выполнение задания лабораторной работы без прерываний

```
#include <msp430.h>

/**
 * main.c
 */

int main(void) {
    WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer

    P8OUT = 0;

    P8DIR |= BIT1;
    P8DIR |= BIT2;
    P1DIR |= BIT3;
    P1DIR |= BIT2;

    P1REN |= BIT7;
    P1OUT |= BIT7;

    P2REN |= BIT2;
    P2OUT |= BIT2;

    int led1_state = 0;
    int prevButton1Val=0;

    int led2_state = 0;
    int prevButton2Val=0;
    while(1)
    {
        int butt1 = (P1IN & BIT7) == 0;
        int butt2 = (P2IN & BIT2) == 0;

        if(butt1 == 0 && butt2==1 ){
            prevButton1Val=1;
        }

        if ( butt1 ==0  && butt2 == 0 && led1_state==0 && prevButton1Val==1){
            if(led1_state==0){
                P1OUT |= BIT3;
                led1_state=1;
                __delay_cycles(900000);
                prevButton1Val = 0;
            }else{
                prevButton1Val = 0;
            }
        }

        if(butt1==1 && butt2==0){
            prevButton2Val=1;
        }

        if(butt1==0 && butt2==0 && prevButton2Val==1 && led2_state==0){
            prevButton2Val=0;
            P1OUT |= BIT2;
            led2_state=1;
            __delay_cycles(900000);
        }

        if(butt1==1 && butt2==1&& led2_state==1){
            P1OUT &= ~BIT2;
            led2_state=0;
            __delay_cycles(900000);
        }
    }
}
```

```

        if ( butt1 == 0  && butt2 == 0 && led1_state==1 && prevButton1Val==1){
            P1OUT &= ~BIT3;
            led1_state=0;
            prevButton1Val = 0;
            __delay_cycles(900000);
        }
    }
    return 0;
}

```

## 4.2 Выполнение задания лабораторной работы с прерываниями

```

#include <msp430.h>

int butt1_flag = 0;
int butt2_flag = 0;
int led1_state = 0;
int led2_state = 0;
int flag1 = 0;

#pragma vector = PORT1_VECTOR
__interrupt void buttonPush1(void) {
    if (butt1_flag==0){
        butt1_flag=0;
        P1IES |= BIT7;
        if(led1_state==0){
            P1OUT |= BIT2;
            led1_state=1;
        }
        if(butt2_flag==1){
            P1OUT &= ~BIT2;
            led1_state=0;
        }
        __delay_cycles(900000);
    }
    else {
        butt1_flag=1;
        P1IES &= ~BIT7;
    }
    P1IFG=0;
}

#pragma vector = PORT2_VECTOR
__interrupt void buttonPush2(void) {
    flag1 = 1;
    if (butt2_flag==0){
        butt2_flag=1;
        P2IES &= ~BIT2;
    }
    else {
        butt2_flag=0;
        P2IES |= BIT2;
        if(led2_state==0){
            P1OUT |= BIT3;
            led2_state=1;
        }
        else{
            P1OUT &= ~BIT3;
            led2_state=0;
        }
        __delay_cycles(900000);
    }
    P2IFG=0;
}

int main(void) {
    WDTCTL = WDTPW | WDTHOLD;
}

```

```

P1DIR |= BIT3; //
P1DIR |= BIT2;
P8DIR |= BIT1;
P1OUT = 0;
P8OUT=0;

P1REN |= BIT7; //
P1OUT |= BIT7;

P2REN |= BIT2;
P2OUT |= BIT2;
__bis_SR_register(GIE);
P1IE |= BIT7;
P2IE |= BIT2;
P1IES |= BIT7;
P2IES |= BIT2;
P1IFG = 0;
P2IFG = 0;

__no_operation();

return 0;
}

```

## **5 ВЫВОДЫ**

В ходе выполнения лабораторной работы мы ознакомились с интегрированной средой разработки Code Composer Studio. Ознакомились с основными функциональными возможностями платы MSP-EXP430F5529.

Написали программу по управлению цифровым вводом-выводом в соответствии с заданием варианта (с использованием прерываний и без их использования).