БГУИР

Кафедра ЭВМ

Отчет по лабораторной работе № 1

Цифровой ввод/вывод. Прерывания

Вариант №6

Выполнили студенты группы 950501:

Лабецкий А.А.

Ламашко И.С.

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 С;
* написанных самостоятельно.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**
   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);
* однотактные 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:

* PxIN – чтение данных с вывода;
* PxOUT – установка значения выхода;
* PxDIR – выбор направления: 0 – вход, 1 – выход;
* PxREN – разрешение подтягивающего резистора;
* PxDS – выбор допустимой силы вывода;
* PxSEL – выбор функции вывода: 0 – I/O, 1 – периферия;
* PxIV – генерирует значение для изменения счетчика команд,
* соответствующее прерыванию с максимальным приоритетом;
* PxIES – выбор направления перепада для генерации запроса на
* прерывание: 0 – по фронту, 1 – по спаду;
* PxIE – разрешение прерывания;
* PxIFG – 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 |
| PxIN | 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 | - | - | - | - | - | - | - |

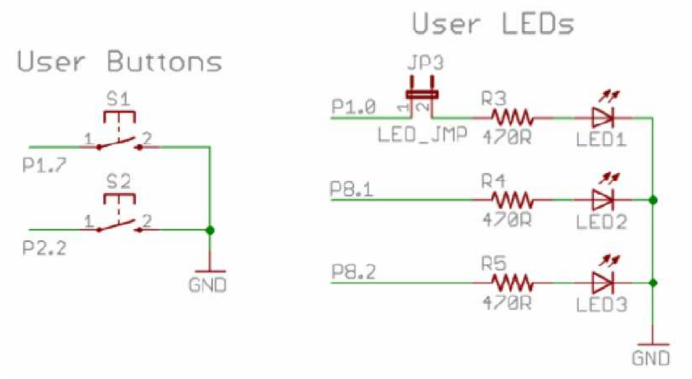


Рис. 1.9 Подключение пользовательских кнопок и светодиодов

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

ниже. Для других портов схемотехника может несколько отличаться, в зависимости от особенностей подключаемой к выводу периферии микроконтроллера.

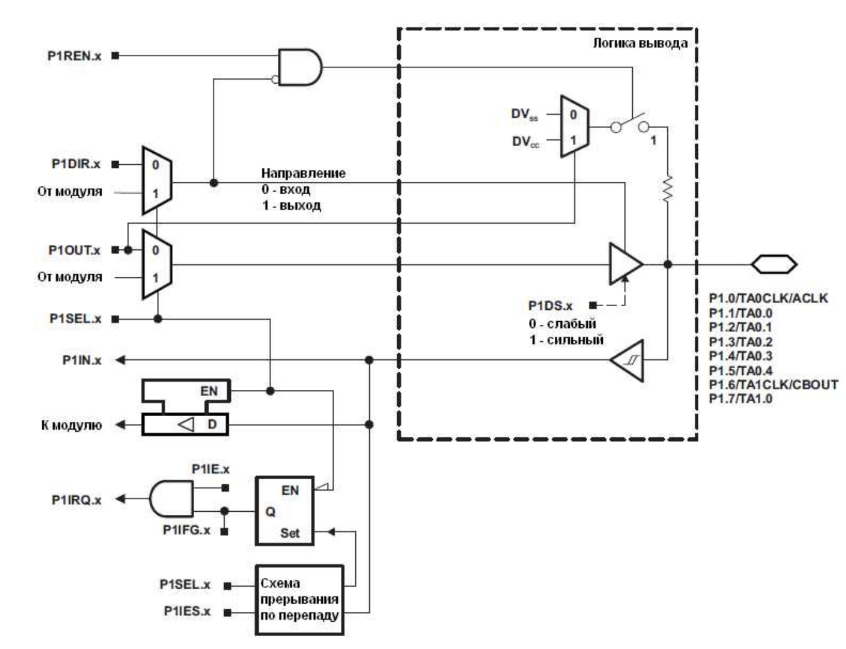


Рис. 1.10 Организация входа-выхода с триггером Шмидта на примере порта 1

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 msp430.h, который в свою очередь подключает файл msp430f5529.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 NNNименуется 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 случая 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 msp430.h, который в свою очередь подключает файл msp430f5529.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 случая 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).

1. **ВЫПОЛНЕНИЕ РАБОТЫ**

**4.1 Выполнение задания лабораторной работы без прерываний**

#include <msp430.h>

#include <stdio.h>

void TurnOnLed(int number);

void TurnOffLed(int number);

int main(void)

{

volatile int i;

// stop watchdog timer

WDTCTL = WDTPW | WDTHOLD;

//set direction for led2 and led3

P8DIR |= BIT2;

P8DIR |= BIT1;

P8OUT = 0;

//set direction for s1, s2

P1DIR = 0;

P1REN = BIT7;

P1OUT= BIT7;

P2DIR = 0;

P2REN = BIT2;

P2OUT = BIT2;

int isPressed = 1;

int isPressed2 = 1;

for(;;)

{

if (((P1IN & BIT7) != BIT7) && ((P2IN & BIT2) == BIT2) && ((P1IN & BIT7) != isPressed))

{

TurnOnLed(2);

}

else if (((P1IN & BIT7) == BIT7) && ((P2IN & BIT2) == BIT2) && ((P1IN & BIT7) != isPressed))

{

TurnOffLed(2);

}

if (((P2IN & BIT2) != BIT2) && ((P2IN & BIT2) != isPressed2))

{

TurnOnLed(3);

}

else if (((P2IN & BIT2) == BIT2) && ((P1IN & BIT7) != BIT7) && ((P2IN & BIT2) != isPressed2))

{

TurnOffLed(3);

}

isPressed = P1IN & BIT7;

isPressed2 = P2IN & BIT2;

\_\_delay\_cycles(10000);

}

}

void TurnOnLed(int number)

{

if(number == 2)

{

P8OUT |= BIT1;

}

else if(number == 3)

{

P8OUT |= BIT2;

}

}

void TurnOffLed(int number)

{

if(number == 2)

{

P8OUT &= ~BIT1;

}

else if(number == 3)

{

P8OUT &= ~BIT2;

}

}

**4.2 Выполнение задания лабораторной работы с прерываниями**

#include <msp430.h>

#include <stdio.h>

void TurnOnLed(int number);

void TurnOffLed(int number);

int main(void)

{

volatile int i;

// stop watchdog timer

WDTCTL = WDTPW | WDTHOLD;

//set direction for led2 and led3

P8DIR |= BIT2;

P8DIR |= BIT1;

P8OUT = 0;

//set direction for s1, s2

P1SEL &= (~BIT7);

P1IES |= (BIT7);

P1IFG &= (~BIT7);

P1IE |= (BIT7);

P1DIR = 0;

P1REN = BIT7;

P1OUT= BIT7;

P2SEL &= (~BIT2);

P2IES |= (BIT2);

P2IFG &= (~BIT2);

P2IE |= (BIT2);

P2DIR = 0;

P2REN = BIT2;

P2OUT = BIT2;

\_\_bis\_SR\_register(LPM4\_bits + GIE); // Enter LPM4 w/interrupt

\_\_no\_operation(); // For debugger

}

void TurnOnLed(int number)

{

if(number == 2)

{

P8OUT |= BIT1;

}

else if(number == 3)

{

P8OUT |= BIT2;

}

}

void TurnOffLed(int number)

{

if(number == 2)

{

P8OUT &= ~BIT1;

}

else if(number == 3)

{

P8OUT &= ~BIT2;

}

}

// Port 1 interrupt service routine

#pragma vector=PORT1\_VECTOR

\_\_interrupt void Port\_1(void)

{

switch(\_\_even\_in\_range(P1IV,16))

{

case 16:

{

\_\_delay\_cycles(8000);//bouncing = 1ms, fr = 8mhz

if (((P2IN & BIT2) == BIT2))

{

P8OUT ^= BIT1;

P1IES ^= BIT7;

}

P1IFG &= ~BIT7;

break;

}// P1.7

}

}

// Port 2 interrupt service routine

#pragma vector=PORT2\_VECTOR

\_\_interrupt void Port\_2(void)

{

switch(\_\_even\_in\_range(P2IV,16))

{

case 6:

{

\_\_delay\_cycles(8000);//bouncing = 1ms, fr = 8mhz

if (((P2IN & BIT2) != BIT2))

{

TurnOnLed(3);

P2IES &= ~BIT2;

}

if (((P2IN & BIT2) == BIT2) && ((P1IN & BIT7) != BIT7))

{

TurnOffLed(3);

P2IES |= BIT2;

}

P2IFG &= ~BIT2;

break;

}// P1.2

}

}

1. **ВЫВОДЫ**

В ходе выполнения лабораторной работы была написана программа для взаимодействия со светодиодами и кнопками с использованием прерываний и без них.