


ЛАБОРАТОРНАЯ РАБОТА №1

ЦИФРОВОЙ ВВОД/ВЫВОД. ПРЕРЫВАНИЯ.

Цель: Ознакомиться с интегрированной средой разработки Code Composer Studio. Ознакомиться с основными функциональными возможностями платы MSP-EXP430F5529. Изучить приемы работы с цифровыми портами ввода/вывода. Ознакомиться с работой подсистемы прерываний

1.1 Основы работы в Code Composer Studio

Code Composer Studio (CCS) – это интегрированная среда разработки (IDE), которая поддерживает микроконтроллеры и встраиваемые процессоры Texas Instruments (TI). CCS включает в себя набор инструментов, используемых для разработки и отладки встраиваемых приложений. Более подробную информацию можно найти на сайте разработчика TI: <http://www.ti.com/tool/CCSTUDIO>. Здесь будут рассмотрены только основные приемы работы (на примере CCS v7)

Для того, чтобы создать новый проект в CCS необходимо выбрать File → New → CCS Project либо Project → New CCS Project... (рисунок 1.1), либо с помощью соответствующей кнопки на панели инструментов .

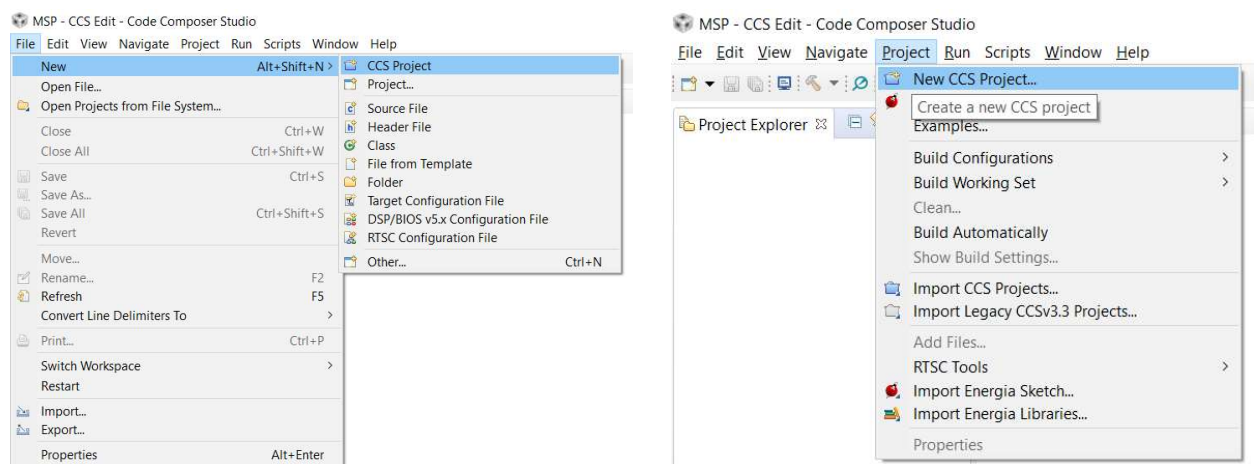


Рисунок 1.1 – Создание нового проекта в CCS

В открывшемся окне необходимо настроить конфигурацию проекта (рисунок 1.2). Для этого требуется:

- выбрать целевое устройство (на лабораторных работах разработка будет производиться для микроконтроллеров MSP430F5529, для упрощения поиска можно сначала выбрать семейство микроконтроллеров);
- задать название проекта;
- при необходимости изменить путь к проекту (нельзя использовать кириллицу);

– выбрать тип шаблона (рекомендуется выбирать пустой проект с “main.c”).

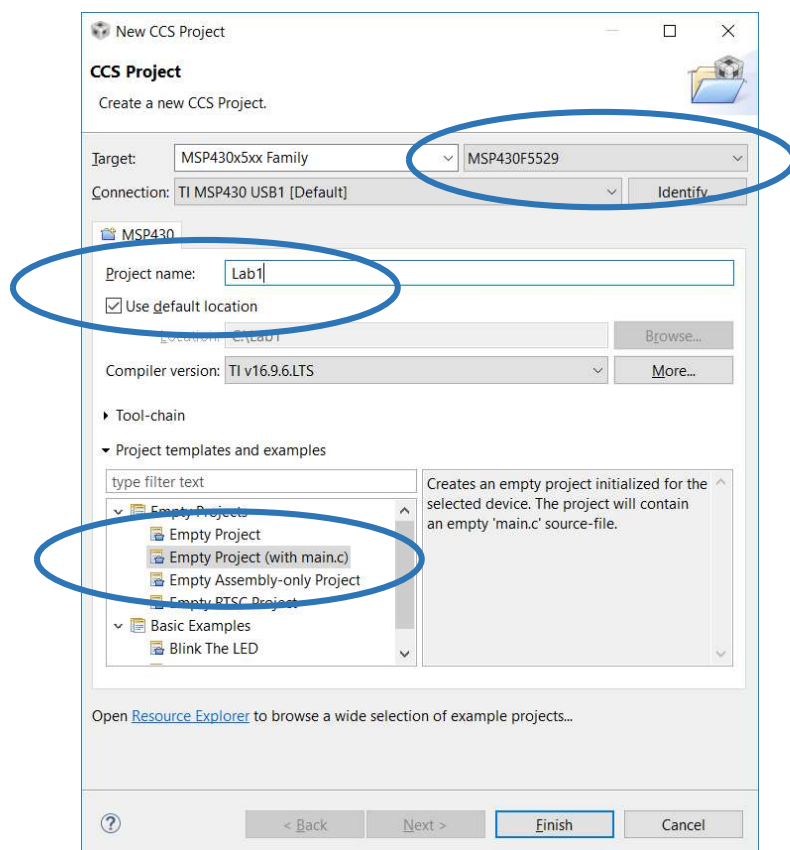


Рисунок 1.2 – Настройка конфигурации нового проекта

После нажатия на кнопку “Finish” будет создан и открыт новый проект (рисунок 1.3).

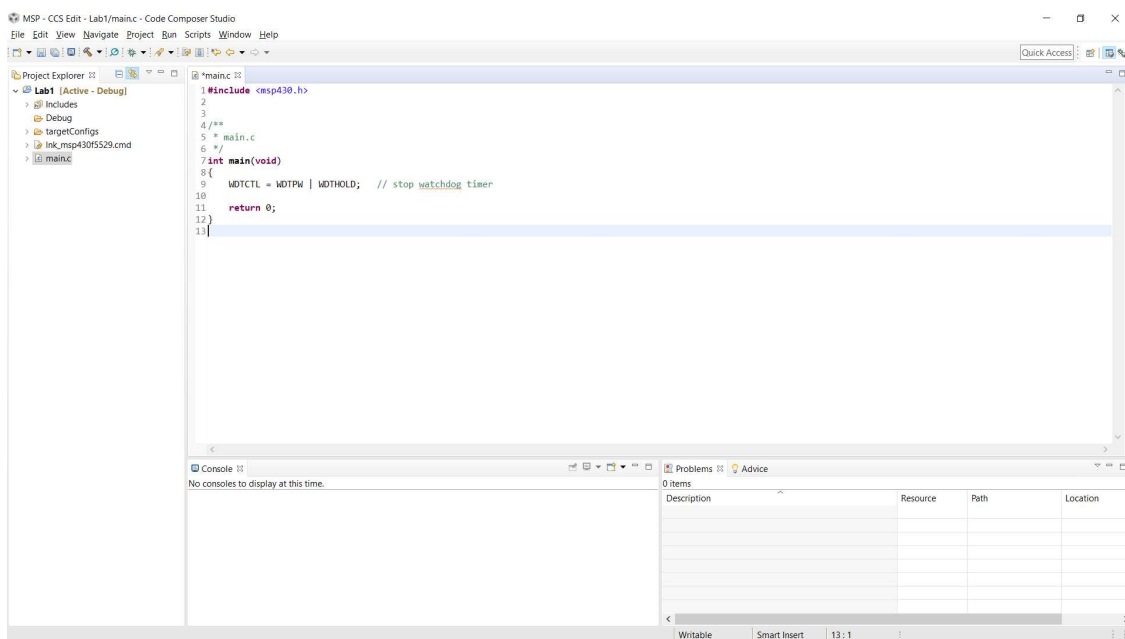



Рисунок 1.3 – Новый проект

Сборка проекта осуществляется с помощью соответствующей кнопки на панели инструментов  либо одним из следующих способов (рисунок 1.4):

- Project → Build Project;
- Используя контекстное меню по щелчку правой кнопкой мыши на папке проекта в поле Project Explorer.

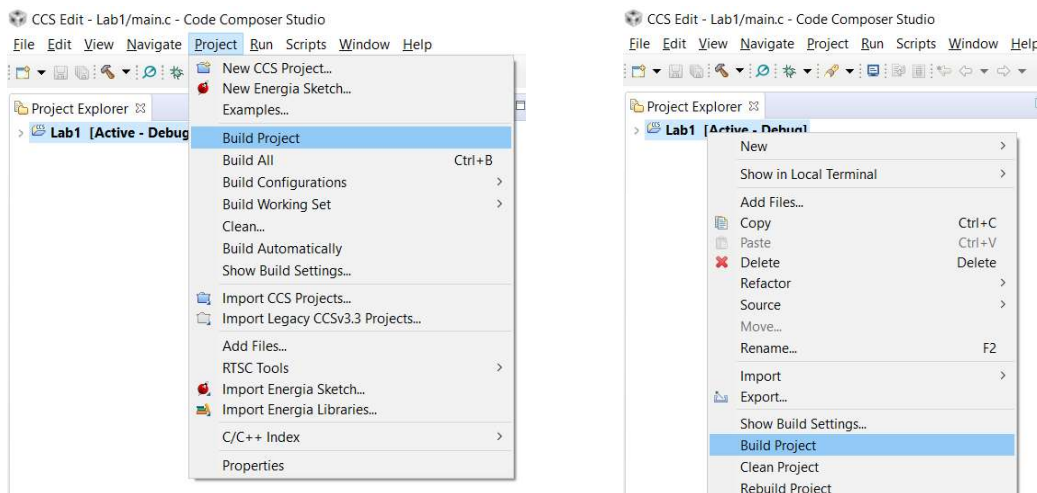


Рисунок 1.4 – Сборка проекта


Запись собранного проекта на микроконтроллер и запуск на выполнение в режиме отладки выполняется вызовом из основного меню среды разработки Run → Debug либо с помощью соответствующей кнопки на панели инструментов  (рисунок 1.5).



Рисунок 1.5 – Запуск проекта в режиме отладки

После прошивки и запуска CCS переходит в режим отладки, открывая соответствующее окно. Продолжить выполнение, войти внутрь вызываемой функции, выйти на уровень выше или остановить выполнение программы можно используя кнопки отладки на панели приложения. Точки останова ставятся в окне отладки слева от кода двойным щелчком. После остановки выполнения среда возвращается в режим редактирования.

Крайне полезной в режиме отладки приложения будет вкладка Registers (рисунок 1.6), где будет отображено содержимое всех регистров микроконтроллера в настоящий момент времени (на точке останова). Если значение в регистре (битах) изменялось, соответствующее поле будет отмечено желтым цветом.

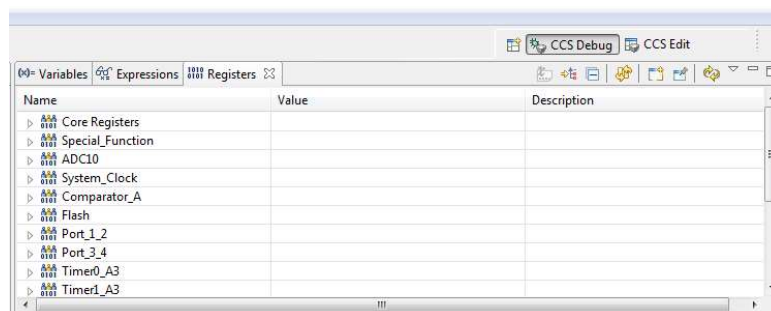


Рисунок 1.6 – Просмотр регистров в режиме отладки

1.2 Плата MSP-EXP430F5529

Подробную информацию и руководства можно найти на сайте компании-разработчика TI: <http://www.ti.com/tool/msp-exp430f5529>. Здесь будут рассмотрены только основные возможности.

Внешний вид экспериментальной платы представлен на рисунке 1.7, а расположение компонентов – на рисунке 1.8.

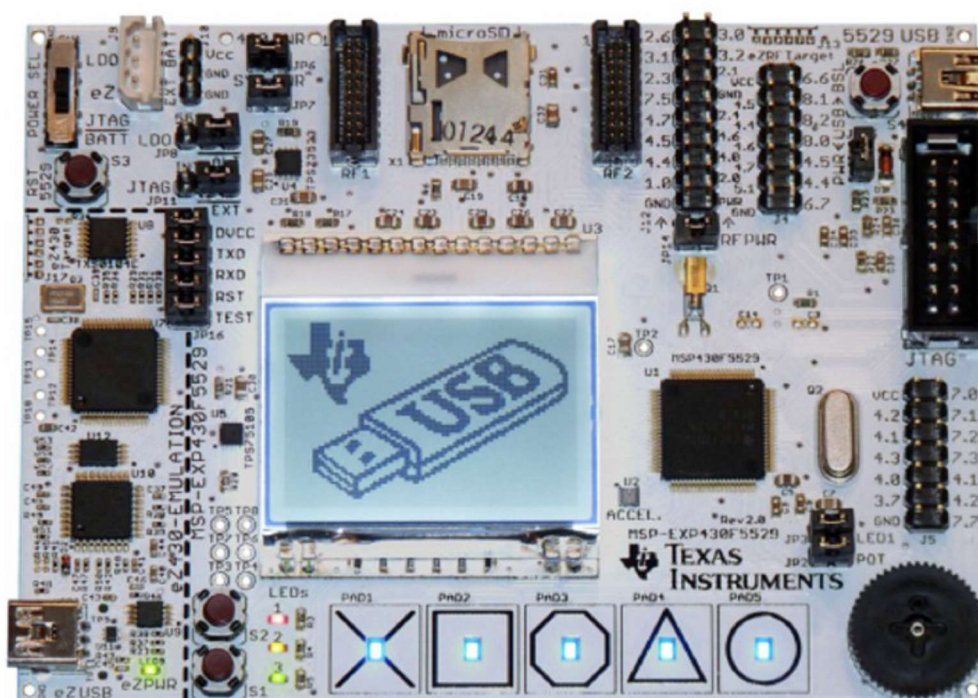


Рисунок 1.7 – Внешний вид экспериментальной платы MSP-EXP430F5529

К основным компонентам, входящим в состав платы относятся:

- микроконтроллер MSP430F5529;
- пользовательские кнопки S1 и S2;
- кнопка сброса S3;
- светодиоды LED1-LED3;
- сенсорные кнопки PAD1-PAD5, совмещенные со светодиодами LED4-LED8;

- колесико для аналогового ввода (потенциометр);
- ЖКИ;
- акселерометр;
- SD-карта;
- разъемы доступа к портам контроллера;
- JTAG-интерфейс;
- USB-интерфейс;
- Интерфейс для беспроводного подключения;
- разъем внешнего питания;
- переключки.

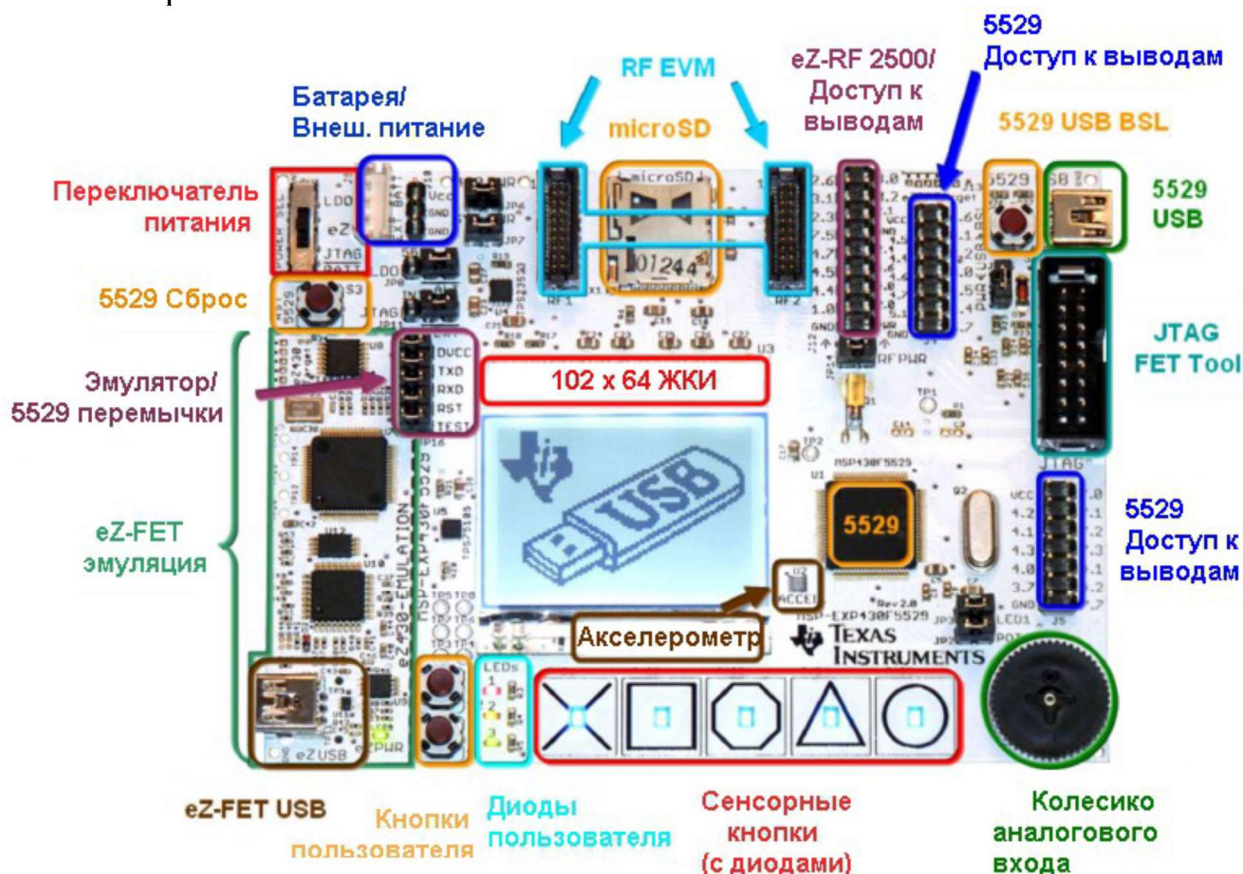


Рисунок 1.8 – Назначение элементов платы MSP-EXP430F5529

Плата MSP-EXP430F5529 подключается к USB-порту ПК через разъем ezUSB платы.

1.3 Микроконтроллер MSP430F5529

MSP430F5529 представляет собой микроконтроллер для обработки смешанных сигналов со сверхнизким энергопотреблением.

К особенностям архитектуры относится:

- 16-тиразрядная RISC архитектура;

- ортогональная архитектура, при которой каждая команда пригодна для каждого режима адресации;
- 27 основных команд + 24 эмулированных;
- 7 согласованных способов адресации;
- полный доступ ко всем регистрам, включая счетчик команд, регистр состояния, и указатель стека;
- генератор шести основных констант;
- одноктактные регистровые операции;
- большой размер регистрового файла, уменьшающий количество обращений к памяти (16 регистров = 4 специальных + 12 РОН);
- 20-битная шина адреса, 16-битная шина данных;
- Фон-Неймановская адресная шина общей памяти и шина данных памяти;
- пересылки память-память без промежуточного сохранения в регистре.

Микроконтроллер обладает следующими характеристиками:

- производительность до 25 MIPS;
- моментальный переход в активный режим (порядка 6 мкс);
- 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:

- FLASH-память 128 Кб;
- SRAM 8 Кб (+2 Кб при выключенном USB);
- 80 выводов, 63 линии входа/выхода;
- единая система управления тактовыми сигналами:
 - система автоматической подстройки частоты (FLL);
 - маломощный/низкочастотный генератор тактового сигнала (VLO)
 - стабилизированный низкочастотный генератор тактового сигнала (REFO)
 - резонатор на 32.768 кГц (XT1);
 - высокочастотный резонатор до 32 МГц (XT2);
- 4 асинхронных 16-разрядных таймера/счетчика:
 - таймер ТА0 с пятью регистрами захвата/сравнения
 - таймер ТА1 с тремя регистрами захвата/сравнения
 - таймер ТА2 с тремя регистрами захвата/сравнения
 - таймер ТВ0 с семью регистрами захвата/сравнения
- сторожевой таймер (WDT) и таймер часов реального времени (RTC);

- гибкая система управления питанием PMM:
 - интегрированный регулятор напряжения (LDO);
 - супервизор и монитор напряжения питания с детектором падения напряжения;
- 2 универсальных последовательных коммуникационных интерфейса (USCI) :
 - порты USCI_A0 и USCI_A1, каждый из которых поддерживает: расширенный UART с автоопределением скорости передачи данных, IrDA, синхронный SPI
 - порты USCI_B0 и USCI_B1, каждый из которых поддерживает: I2C, синхронный SPI;
- 3-канальный контроллер прямого доступа к памяти (DMA);
- 16-канальный, 12-битный АЦП с внутренним источником опорного напряжения, с функциями выборки и хранения и автоматического сканирования;
- 12-канальный аналоговый компаратор;
- 32-битный аппаратный умножитель;
- контроллер полноскоростного USB:
 - интегрированный физический уровень USB;
 - интегрированная система питания через USB на 3.3/1.8 Вольт;
 - интегрированный USB-PLL;
 - восемь оконечных точек ввода, восемь оконечных точек вывода.

Обобщенная архитектура микроконтроллера представлена на рис. 1.9. Элементы архитектуры микроконтроллера будут описаны по мере выполнения лабораторных работ. Более подробную информацию можно найти на <http://www.ti.com/product/msp430f5529>.

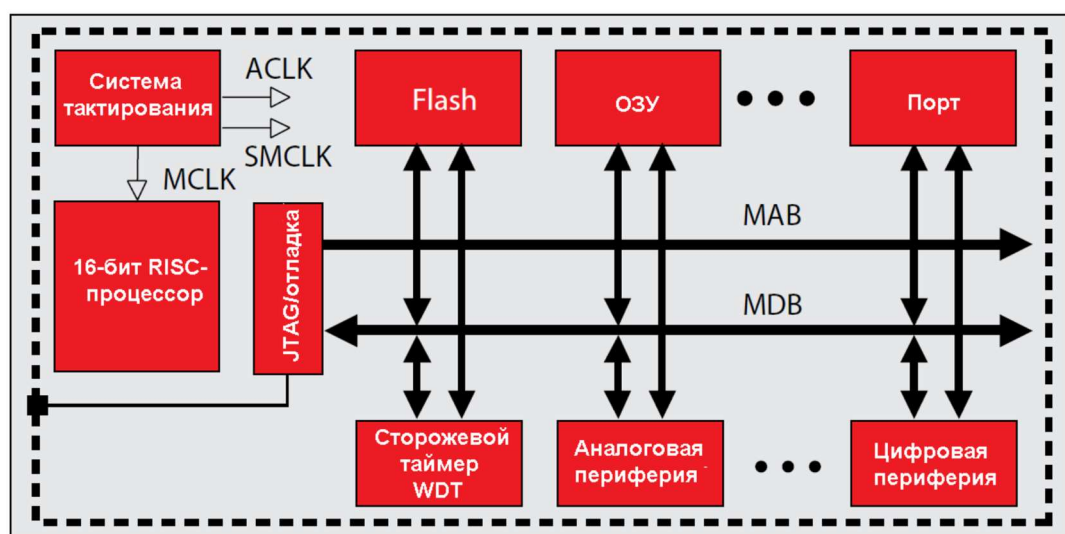


Рисунок 1.9 – Обобщенная архитектура MSP430F5529

1.4 Цифровой ввод/вывод

Как было показано ранее, микроконтроллер имеет 63 линии (ножки, вывода, пина) ввода/вывода, объединенные в 8-миразрядные порты ввода/вывода P1, P2, ... , P8, PJ (для P8 программно доступно 3 бита, для PJ – 4 бита). Каждый из 63 пинов могут работать в режиме цифрового ввода/вывода, порты P1 – P7 могут быть сконфигурированы в режиме входа/выхода внутренней периферии микроконтроллера, пины порта PJ – могут быть сконфигурированы в качестве линий JTAG-интерфейса.

Для удобства работы порты могут объединяться в пары:

- PA – P1 и P2;
- PB – P3 и P4;
- PC – P5 и P6;
- PD – P7 и P8.

Цифровые входы/выходы обладают следующими возможностями:

- независимые индивидуально программируемые входы/выходы;
- любые комбинации входа или выхода;
- индивидуально конфигурируемые прерывания от P1 и P2;
- отдельные регистры данных для входов и выходов.

Цифровые входы/выходы конфигурируются программным обеспечением пользователя с помощью настройки следующих регистров:

- PxIN – каждый бит регистра отражает величину входного сигнала на соответствующей ножке (пине) ввода/вывода, когда она сконфигурирована на функцию ввода/вывода;

- PxOUT – каждый бит в регистре содержит значение, которое будет выведено на соответствующую ножку ввода/вывода, сконфигурированную на функцию ввода/вывода и имеющую направление на вывод;

- PxDIR – каждый бит в регистре позволяет независимо от выбранной для этой ножки функции (ввод/вывод или периферия) выбрать направление соответствующей ножки: 0 – вход, 1 – выход;

- PxREN – каждый бит в регистре включает или отключает подтягивающий резистор соответствующего ножки ввода / вывода.;

- PxDS – каждый бит в регистре осуществляет выбор допустимой выходной силы тока для соответствующей ножки: 0 – пониженная (по умолчанию), 1 – полная;

- PxSEL – каждый бит определяет, как будет использована ножка – в качестве порта ввода/вывода или в качестве функции периферийного модуля: 0 – I/O, 1 – периферия;

- PxIV – генерирует значение для изменения счетчика команд, соответствующее прерыванию с максимальным приоритетом;

- RxIE – каждый бит в регистре разрешает прерывания по соответствующей линии ввода/вывода;
- RxIES – каждый бит в регистре определяет направление перепада для генерации запроса на прерывание: 0 – по фронту, 1 – по спаду;
- RxIFG – каждый бит в регистре соответствует флагу запроса на прерывание по соответствующей линии ввода/вывода.

Логика управления выводом на примере порта 1 представлена на рисунке 1.10. Каждая линия на данной схеме представляет собой 8-мибитную шину. Для других портов схемотехника может несколько отличаться, в зависимости от особенностей подключаемой к выводу периферии микроконтроллера.

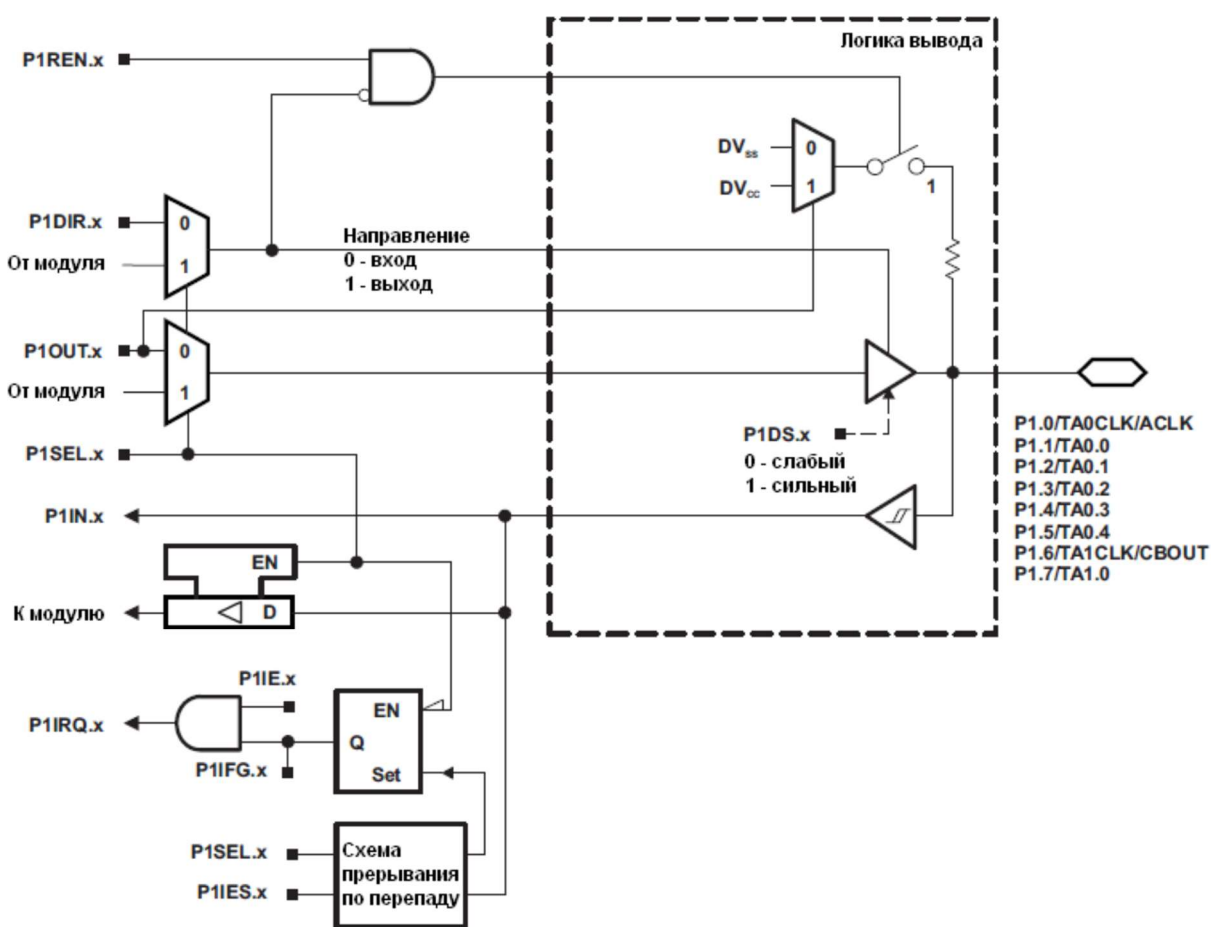


Рисунок 1.10 – Организация ввода/вывода на примере порта 1

Следует обратить внимание, что регистр RxOUT управляет логикой работы подтягивающего резистора, если вывод сконфигурирован как цифрой I/O, направление работы порта – вход, а также разрешен подтягивающий резистор.

Каждому из 8-миразрядных регистров, связанных с портом, соответствует ячейка в памяти контроллера (таблица 1.1).

Таблица 1.1 – Адреса портов ввода/вывода

№ порта	1	2	3	4	5	6	7	8	J
Базовый адрес	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	–	–	–	–	–	–	–

Для упрощения работы с портами ввода вывода будем обозначать их $Px.y$, где x – номер порта, y – номер вывода (например вывод 0 порта один будет обозначаться как $P1.0$). Соответственно, для регистров $PxIN$, $PxOUT$ и т.д. вместо x также подставляется необходимый номер порта.

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). Если при создании проекта был выбран пустой проект с пустой проект с `main.c`, подключение необходимого заголовочного файла, а также отключение сторожевого таймера будет сделано автоматически.

Константы и определения заданы как для портов, так и для отдельных полей и их значений. Поэтому работа с портами становится максимально удобной для программиста. Константы $BIT0 - BIT7$ представляют собой 8-миразрядные числа, в которых все биты нулевые за исключением указанного (например, $BIT0$ означает, что нулевой бит будет равен единицы, остальные биты – нулю). В этом случае, запись $P1DIR |= BIT2$; означает, что в регистр $P1DIR$, 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, при этом остальные биты оставляет неизменными.

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).

1.5 Прерывания

Различают следующие виды прерываний:

– системные немаскируемые (сигнал RST/NMI в режиме NMI, сигнал от сторожевого таймера, сбой тактового генератора, ошибка доступа Flash-памяти);

– пользовательские немаскируемые (сбой напряжения питания от PMM, доступ к несуществующей памяти, события буфера JTAG);

– маскируемые прерывания (генерируются периферийными модулями).

Маскируемые прерывания могут быть отключены (замаскированы) индивидуально или все сразу (бит GIE регистра состояния SR).

Микроконтроллер имеет следующую структуру прерываний:

– векторные прерывания без необходимости опроса;

– векторы прерываний расположены вниз от адреса 0FFFEh.

Для обслуживания прерываний используются подпрограммы обработки прерываний, начальные адреса которых сведены в таблицу векторов прерываний. Таблица векторов прерываний содержат 64 вектора. В каждом векторе должен быть записан 16-битный адрес обработчика прерываний, на который будет передано управление в случае возникновения прерывания.

Бит SYSRIVECT регистра SYSCTL позволяет определить альтернативную таблицу векторов, в старших адресах RAM. По сигналу сброса этот бит автоматически сбрасывается.

Каждый вектор прерывания соответствует одному или нескольким источникам. Вызывается прерывание установкой (как правило, аппаратной) флага запроса на прерывание того или иного устройства.

В том случае, если прерывание берется, флаг требования прерывания сбрасывается автоматически, если он является единственным для данного вектора, в противном случае, этот флаг необходимо сбрасывать программно в обработчике. В этом случае в обработчике необходимо программное тестирование флагов для определения источника, вызвавшего прерывание.

При одновременном возникновении нескольких прерываний управление передается по вектору, имеющему более высокий приоритет. В MSP микроконтроллерах вектора с большим адресом имеют более высокий приоритет.

Кратко рассмотрим, как происходит порядок обработки прерывания. Задержка от возникновения запроса на прерывание до начала выполнения обработчика составляет 6 циклов. При возникновении прерываний выполняются следующие действия:

1. Завершается выполнение текущей инструкции.
2. Счетчик команд PC, указывающий на следующую инструкцию, сохраняется в стеке;
3. Регистр состояния SR сохраняется в стеке.
4. Выбирается прерывание с наивысшим приоритетом.
5. Сбрасывается флаг запроса на прерывания, если данному вектору соответствует единственный источник. Если источников несколько, флаг завтра на прерывание необходимо сбрасывать программно.
6. Все биты регистра состояния SR кроме SCG0 сбрасываются в 0; т.к. бит GIE = 0 маскируемые прерывания запрещаются.
7. В счетчик команд PC загружается содержимое выбранного вектора прерывания.

Во время прерывания счетчик команд и регистр состояния помещаются в стек, как показано на рисунке 1.11. Микроконтроллер эффективно сохраняет полное 20-битное значение PC, добавляя биты 19:16 PC к сохраненному значению SR автоматически в стеке. Когда инструкция RETI выполнена, полный 20-битный ПК восстанавливается, делая возможным возврат из прерывания на любой адрес в диапазоне памяти.

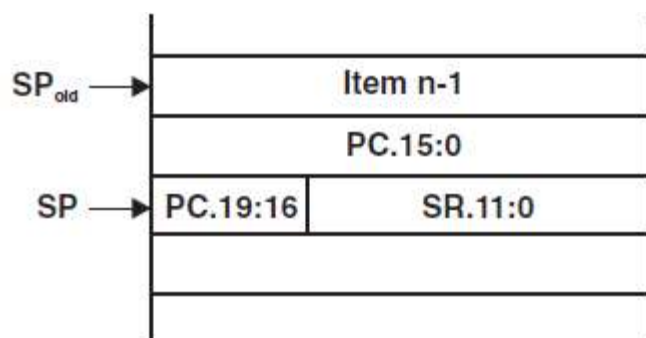


Рисунок 1.11 – Сохранение счетчика команд и регистра состояния в стеке

Из-за конвейерной архитектуры процессора, команда, следующая за EINT (разрешение прерывания), всегда выполняется, даже если запрос на прерывание возник до его разрешения. Если за EINT сразу следует DINT, прерывание, ожидающее обработки может быть не обслужено. Команды, следующие за DINT в этом случае могут сработать некорректно. Аналогичные последствия вызываются альтернативными командами, которые

устанавливают и сразу сбрасывают флаг GIE регистра состояний. Рекомендуется вставлять хотя бы одну команду между EINT и DINT.

Возврат из прерывания выполняется командой RETI, которая выполняется за 5 циклов и загружает из стека SR, PC.

За немаскируемые прерывания отвечают ряд системных регистров (таблица 1.2). Поля данных регистров указаны в таблице 1.3.

Таблица 1.2 – Регистры для работы с немаскируемыми прерываниями

Регистр	Адрес	Назначение
SFRIE1	0100h	Разрешение прерываний
SFRIFG1	0102h	Флаги прерываний
SYSCTL	0180h	Регистр управления
SYSBERRIV	0198h	Генератор вектора ошибок шины
SYSUNIV	019Ah	Генератор вектора пользовательских NMI
SYSSNIV	019Ch	Генератор вектора системных NMI
SYSRSTIV	019Eh	Генератор вектора сброса

Таблица 1.3 – Поля регистров

Регистр	Биты	Поле	Назначение
SFRIE1	7	JMBOUTIE	Разрешение прерываний выхода JTAG
	6	JMBINIE	Разрешение прерываний входа JTAG
	5	ACCVIE	Разрешение прерываний нарушения доступа Flash
	4	NMIIE	Разрешение прерываний вывода NMI
	3	VMAIE	Разрешение прерываний доступа к несуществующей памяти
	1	OFIE	Разрешение прерываний сбоя генератора
	0	WDTIE	Разрешение прерываний сторожевого таймера
SFRIFG1	7	JMBOUTIFG	Флаг прерывания выхода JTAG
	6	JMBINIFG	Флаг прерывания входа JTAG
	4	NMIIFG	Флаг прерывания NMI
	3	VMAIFG	Флаг прерывания доступа к несуществующей памяти
	1	OFIFG	Флаг прерывания сбоя генератора
	0	WDTIFG	Флаг прерывания сторожевого таймера
SYSCTL	0	SYSRIVECT	Вектор прерывания при выходе за пределы RAM (64К или полностью)
SYSUNIV	0-15	SYSUNIV	Вектор пользовательского NMI
SYSSNIV	0-15	SYSSNIV	Вектор системного NMI
SYSRSTIV	0-15	SYSRSTIV	Вектор прерываний сброса
SYSBERRIV	0-15	SYSBSLOFF	Вектор прерываний ошибки системной шины

Пользовательские маскируемые прерывания рассматриваются отдельно при обсуждении соответствующего функционального узла архитектуры микроконтроллера.

Работа с прерываниями достаточно проста. Вначале необходимо разрешить соответствующее прерывание, например, `PIIE |= BIT7;` –

разрешает прерывание по входу 7 вывода порта. Желательно также перед разрешением прерываний, сбросить флаги запроса, чтобы избежать ситуации вызова обработчика до фактического аппаратного возникновения прерывания.

Для разрешения прерываний необходимо воспользоваться функцией записи флага GIE в регистр состояния: `__bis_SR_register(GIE);` (в данном случае запись `SR |= GIE;` использовать нельзя).

Еще одной особенностью запуска в среде отладки Code Composer Studio является необходимость вызова `__no_operation();` перед завершением функции `main()`, если она не использует некоторого цикла. Без этого вызова с завершением функции `main()` завершится и выполнение кода в оболочке.

Собственно обработчик прерывания описывается с использованием директивы `#pragma vector`. Для портов ввода/вывода обработчик выглядит следующим образом:

```
#pragma vector = PORTx_VECTOR
__interrupt void ISR(void)
{...}
```

Где *x* – номер порта, *ISR* – название функции обработчика прерывания.

При работе с прерываниями от портов ввода/вывода следует о следующем:

- *PxIFG* не сбрасывается автоматически при завершении обработчика прерывания;
- запись в регистры *PxOUT*, *PxDIR* и *PxREN* может быть причиной установки значений в регистре *PxIFG*;

1.6 Кнопки и светодиоды на плате MSP-EXP430F5529

Пользователю программно доступны две кнопки *S1* и *S2*, подключенные соответственно к выводу 7 порта 1 и выводу 2 порта 2 (рисунок 1.12). В дальнейшем такое подключение будем обозначать как *P1.7* и *P2.2* соответственно.

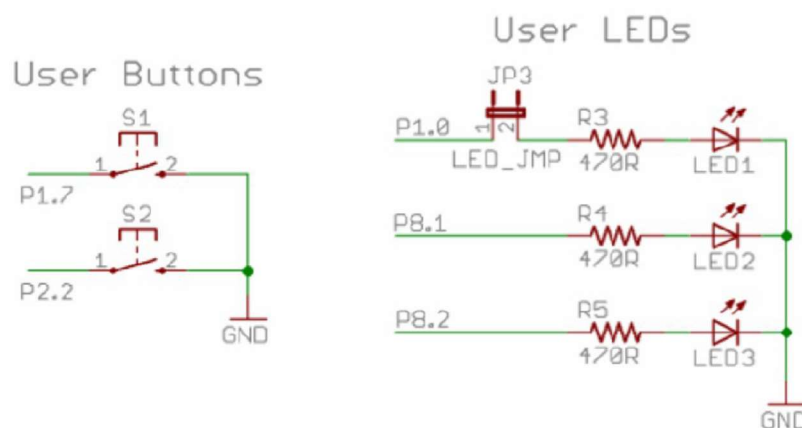


Рисунок 1.12 – Подключение пользовательских кнопок и светодиодов

Также программно доступны 8 светодиодов, три из которых (LED1 – LED3) размещены рядом с кнопками и подключены соответственно к выводам P1.0, P8.1, P8.2 (рисунок 1.12). Еще 5 светодиодов (LED4 – LED8) размещаются в блоке сенсорных кнопок и подключены к выводам P1.1 – P1.5 соответственно (рисунок 1.13).

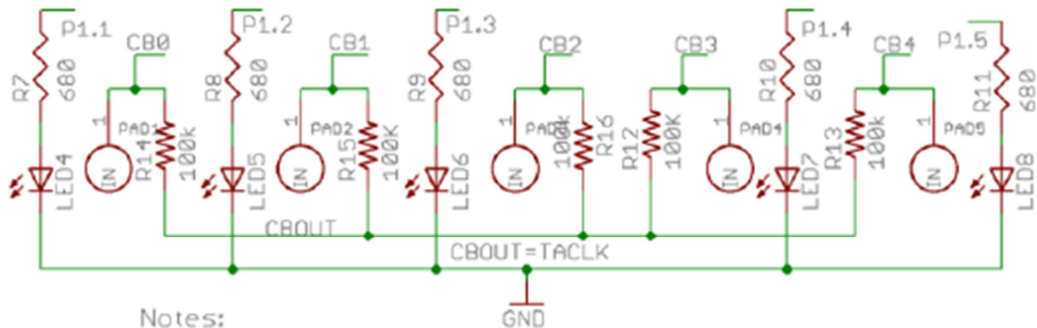


Рисунок 1.13 – Подключение светодиодов в блоке сенсорных кнопок

ВОПРОСЫ К ЛАБОРАТОРНОЙ РАБОТЕ

1. Архитектура MSP430.
2. Что такое пин и порт?
3. Для чего нужны регистры PxIN и PxOUT?
4. Что такое прерывание?
5. Порядок обработки запроса на прерывание в MSP430.

ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

В соответствии с вариантом задания написать программу, которая бы включала и выключала заданный светодиод 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 (как две отдельные программы):

- без использования прерываний;
- с использованием прерываний (не допускается использовать опрос флагов в цикле).

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

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

Варианты заданий на лабораторную работу

Вариант	Номер LED	Включение	Выключение
1	1	По отжатию S2, если S1 не нажата	По отжатию S2, если S1 не нажата
2	4	По отжатию S2, если S1 не нажата	По нажатию S2, если S1 не нажата
3	7	По нажатию S2, если S1 не нажата	По отжатию S2, если S1 не нажата
4	6	По нажатию S2, если S1 не нажата	По нажатию S2, если S1 не нажата
5	8	По отжатию S1, если S2 не нажата	По отжатию S1, если S2 не нажата
6	2	По отжатию S1, если S2 не нажата	По нажатию S1, если S2 не нажата
7	3	По нажатию S1, если S2 не нажата	По отжатию S1, если S2 не нажата
8	5	По нажатию S1, если S2 не нажата	По нажатию S1, если S2 не нажата
9	7	По отжатию S2, если S1 нажата	По отжатию S2, если S1 нажата
10	4	По отжатию S2, если S1 нажата	По нажатию S2, если S1 нажата
11	2	По нажатию S2, если S1 нажата	По отжатию S2, если S1 нажата
12	1	По нажатию S2, если S1 нажата	По нажатию S2, если S1 нажата
13	8	По отжатию S1, если S2 нажата	По отжатию S1, если S2 нажата
14	5	По отжатию S1, если S2 нажата	По нажатию S1, если S2 нажата
15	3	По нажатию S1, если S2 нажата	По отжатию S1, если S2 нажата
16	6	По нажатию S1, если S2 нажата	По нажатию S1, если S2 нажата