Министерство образования и науки Украины
Донбасская государственная машиностроительная академия
Кафедра АПП

Е. И. Донченко

КОНТРОЛЛЕРЫ И ИХ ПРОГРАМНОЕ ОБЕСПЕЧЕНИЕ

Проектирование устройств на ARM7 микроконтроллерах LPC21xx фирмы Philips

Конспект лекций

(для студентов специальности 151
дневной и заочной форм обучения)

Утверждено
на заседании ученого совета
Протокол № 5 от 3.01.18

Краматорск 2018

УДК 681.3.06 (075.8)

Конспект лекций по дисциплине «Контроллеры и их ПО» (для студентов специальности 151 дневной и заочной форм обучения) / Сост. Е. И. Донченко. – Краматорск: ДГМА, 2018. – 96 с.

Рассматриваются особенности архитектуры, программирования и применения ARM7 микроконтроллеров фирмы Philips. Приведены данные для самостоятельного изучения и практического освоения курса «Контроллеры и их ПО»

Составитель Е. И. Донченко, ст. преп.

Отв. за выпуск Е. И. Донченко, ст. преп.

СОДЕРЖАНИЕ

1 СТРУКТУРА МИКРОКОНТРОЛЛЕРА LPC21XX

Микроконтроллеры LPC21xx основаны на 32-х разрядном ядре ARM7 и имеют структуру (рис.1.1) типичную для контроллеров на этом ядре. К ядру микроконтроллера, функционирующему на частоте 60 МГц через локальную шину ARM7 Local Bus, подключены оперативные запоминающее устройство (ОЗУ) и встроенная Flash память. Особенностью Flash памяти микроконтроллеров LPC21xx является функционирование на частоте ядра, в отличие от принятого у конкурентов снижения частоты в два раза.

Программный код может быть размещен и выполнен как в Flash памяти так и в ОЗУ. Но при распределении памяти необходимо помнить, что время чтения для ОЗУ и Flash одинаково, а время записи розниться на порядок. Кроме того запись во Flash память предваряется стиранием страницы, которой принадлежит выбранная группа записываемых данных, т.е. необходимо переписать полностью целую страницу. При этом следует помнить, что паспортное число циклов перезаписи Flash памяти не превышает 1млн. (реально не более 100000), а число циклов перезаписи ОЗУ не ограниченно. С другой стороны, объем ОЗУ исчисляется десятками кбайт, в то время как объем Flash памяти составляет сотни кбайт. И конечно, при выключении питания содержимое Flash памяти не изменяется, а ОЗУ полностью очищается.

По умолчанию, программный код размещается и исполняется в Flash памяти. С учетом вышеприведенных сведений исполнять программный код из ОЗУ имеет смысл только при временной загрузке программных модулей из внешнего источника, например для учебных целей.

Для реализации обработки прерываний к ядру ARM7 через Advanced High – performance Bus (AHB) шину подключается векторный контроллер прерываний, позволяющий расширить возможности обработки прерываний. К этой же шине через мост AHB-APB, подключена Advanced Peripheral Bus (APB) – шина периферийного оборудования, работающая на частоте в 4 раза меньшей частоты ядра.

К шине APB подключаются периферийные устройства, такие как GPIO – цифровой ввод-вывод, АЦП, ЦАП, таймеры-счетчики и т.д. Все они работают на частоте в 4 раза меньше чем частота ядра.

На рисунке 1.1 показана типовая схема включения микроконтроллера.

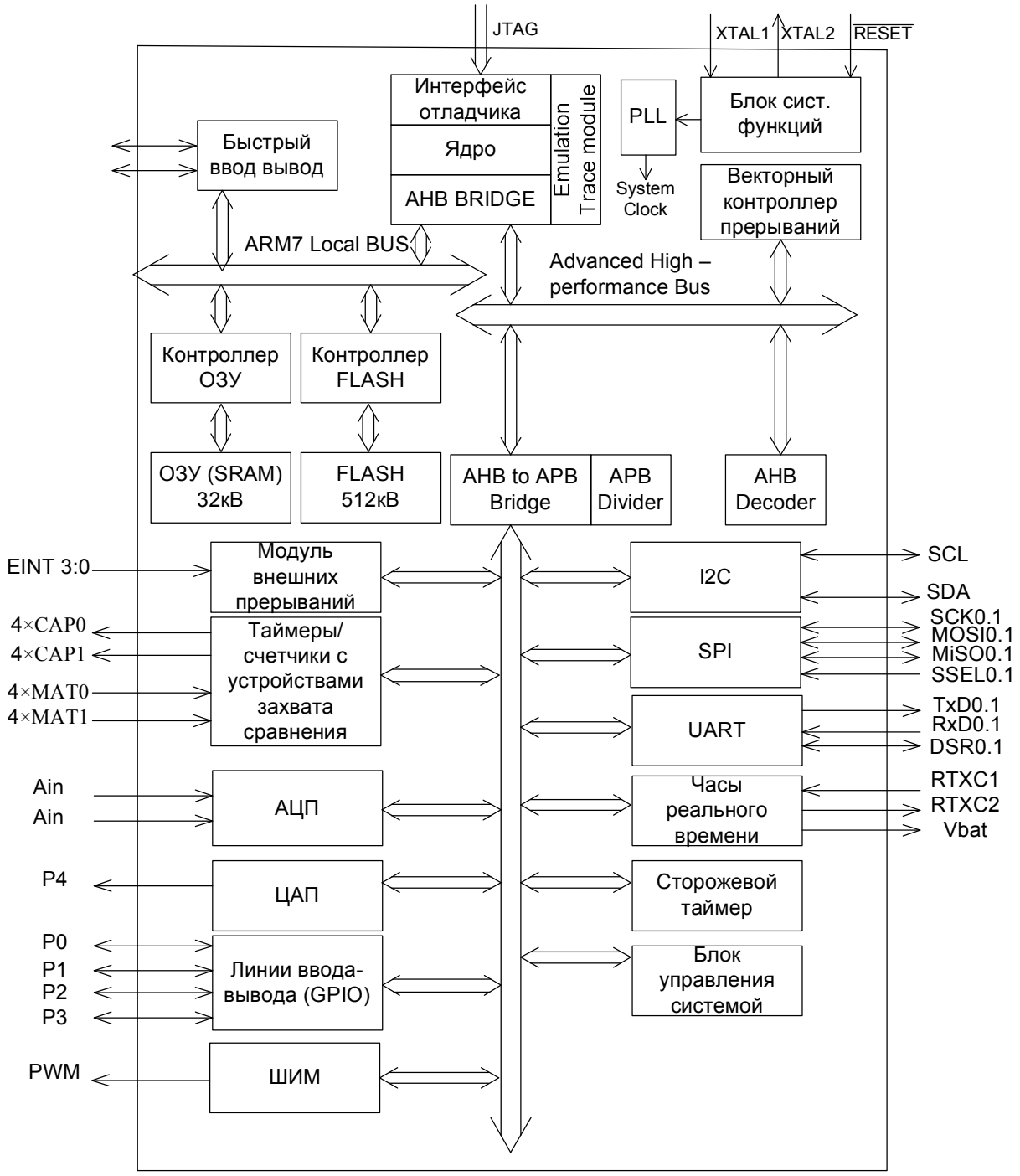


Рисунок 1.1 – Структурная схема микроконтроллера LPC2138

Отличительные особенности МК LPC 2138

Встроенные статическое ОЗУ 32 кбайт и флэш-память 512 кбайт. 128-разр.Интерфейс/ускоритель допускает работу на частоте 60 МГц. Внутрисистемно/ внутриприкладное программирование (ISP/IAP) с помощью встроенного программного обеспечения в загрузочном секторе. Стирание одного сектора или всей флэш-памяти за 400 мс, программирование 256 байт за 1 мс.

Встроенные интерфейсы для реально-временной эмуляции и трассировки со встроенным программным обеспечением RealMonitor и высоко быстродействующим выполнением трассируемых инструкций.

Один 10-разр. ЦАП организует аналоговый регулируемый выход (LPC2138). Два 32-разр. таймеров-счетчиков внешних событий (с четырьмя каналами захвата и четырьмя каналами сравнения в каждом таймере), блок ШИМ (шесть выходов) и сторожевой таймер. Маломощные часы реального времени с отдельным питанием и входом синхронизации 32 кГц.

Несколько последовательных интерфейсов, в т.ч. два УАПП (16C550), две быстродействующих шины I2C (400 кбит/с), SPI и SSP с буферизацией и переменной длиной данных.

Векторизованный контроллер прерываний с конфигурируемыми приоритетами векторных адресов.

Восприятие уровней напряжения до 4,75В на всех линиях ввода-вывода в корпусе LQFP64. До 9 внешних каналов прерывания с настраиваемой чувствительностью к фронтам или уровням. Максимальная тактовая частота ЦПУ 60 МГц генерируется программируемой схемой синтезатора частоты с ФАПЧ (время установления 100мкс). Встроенный генератор работает с внешним кварцевым резонатором в диапазоне от 1 МГц до 30 МГц, а с внешним генератором до 50 МГц.

Экономичные режимы работы, в т.ч. холостой ход (Idle) и снижение мощности (Power-down).

Раздельное включение/отключение периферийных функций, в т.ч. их синхронизация, для оптимизации энергопотребления.

Возобновление работы в режиме снижения мощности (Power-down) по внешнему прерыванию или сигналу детектора снижения питания (BOD).

Встроенные схемы сброса при подаче питания и детектора снижения питания:

Рабочее напряжение ЦПУ составляет 3.0…3.6В (3.3В ± 10 %) с совместимостью с 5В-ыми уровнями на линиях ввода-вывода.

Электрические характеристики контроллеров LPC213x

Uпит = 3,3В

Порты МК допускают подачу напряжения до 5В.

Для подключения внешней логики к выходам МК используются специальные микросхемы рассчитанные на входное U=3,3 В

Линии питания МК должны быть шунтированы блокировочными конденсаторами ,также для синхронизации работы МК на пины XTAL и RTXC подключаются кварцевые резонаторы.

На рисунке 1.2 представлена электрическая схема подключения МК.

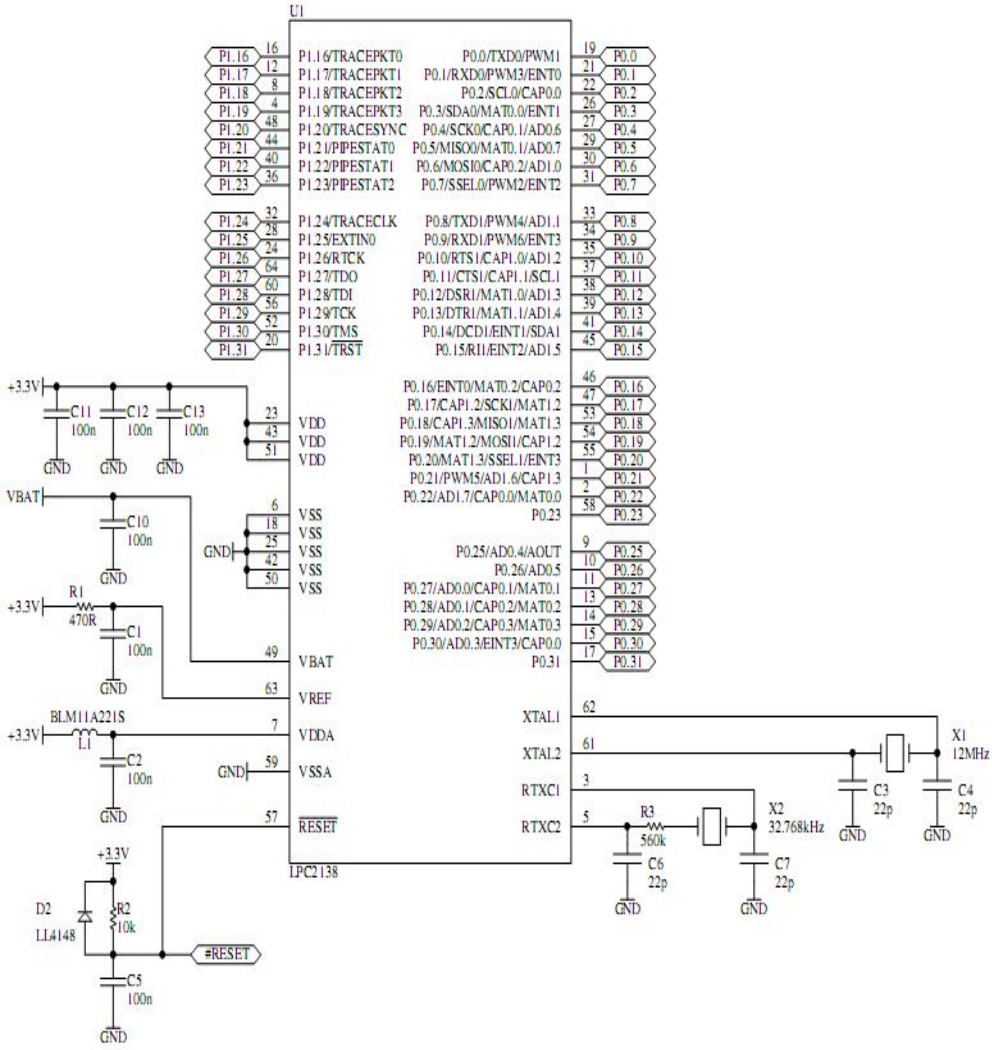


Рисунок 1.2– Схема МК модуля LPC2138

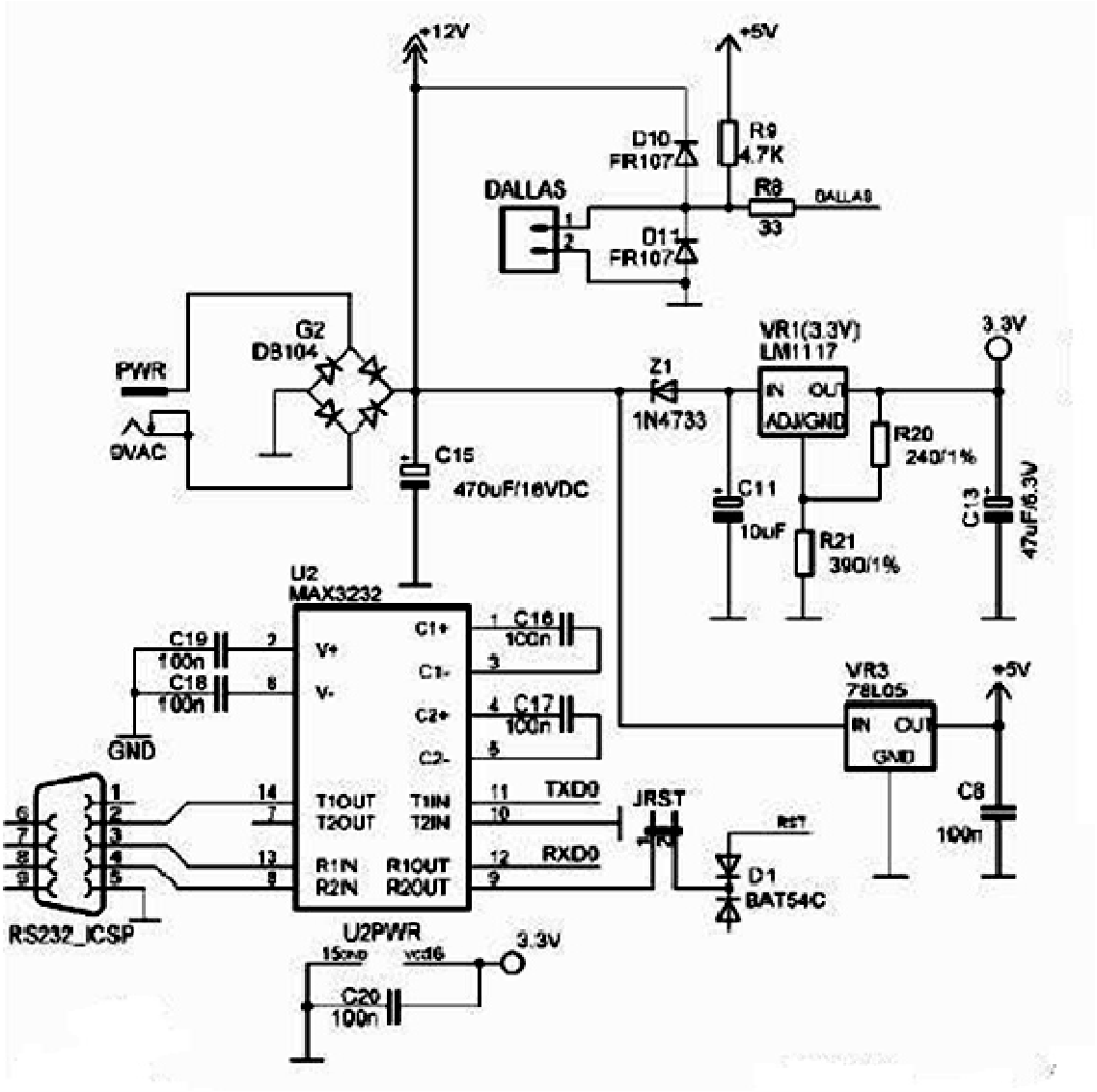


Рисунок 1.2.1 – Схема блока питания и программирования МК LPC2138

Для питания модуля МК можно использовать схему выпрямителя, представленную на рисунке 1.3

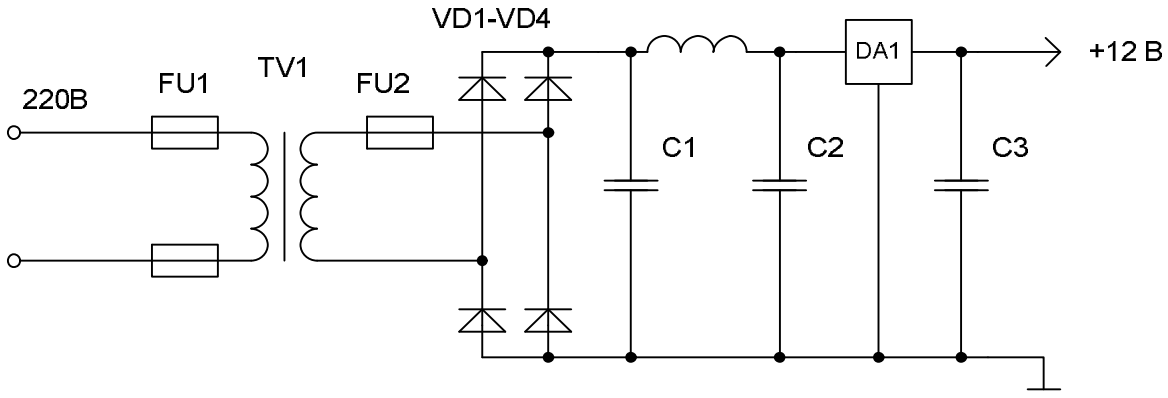


Рисунок 1.3 – Схема выпрямителя

Для получения выпрямленного напряжения заданной амплитуды, можно использовать схему двухполупериодного выпрямителя. Который состоит из трансформатора TF который понижает амплитуду напряжения сети до 12В, на входе и выходе трансформатора стоят плавкие предохранители FU защищающие трансформатор от токов КЗ, т.к. на выходе трансформатора напряжение синусоидальное, а нам необходимо для питания МК выпрямленное, для этого на выходе трансформатора стоят 4 диода которые выпрямляют напряжение, для устранения скачков тока стоит фильтр состоящий из: катушки LC и конденсаторов C1 и C2, микросхема DA1 выполняет функцию стабилизатора напряжения на выходе из выпрямителя, конденсатор C3 поддерживает динамическую мощность на выходе из DA1.

В настоящее время так же широко используются импульсные источники электропитания.

2 ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

2.1 Использование пакета Keil µVision

Использование ARM7 микроконтроллеров для целей обучения обусловлено, в числе прочих факторов, наличием совершенного программного обеспечения, автоматизирующего процесс написания программного кода. Стандартом де-факто для программирования микроконтроллеров в настоящее время является язык Си, а одной из наиболее известных программ, поддерживающих Си для микроконтроллеров ARM – Keil µVision. Рекомендуется использовать версию не ниже 3.1.

После запуска приложения на экране появляется, как правило, последний редактируемый проект, или то, что Keil µVision им считает. Пунктом меню Project -> Close Project необходимо закрыть последний редактируемый проект.

Вызовом пункта меню Project -> New Project вызывается мастер создания проектов. При этом требуется указать имя файла проекта и его размещение. Рекомендуется заранее создать каталог, куда будут размещаться файлы проекта (рисунок 2.1)

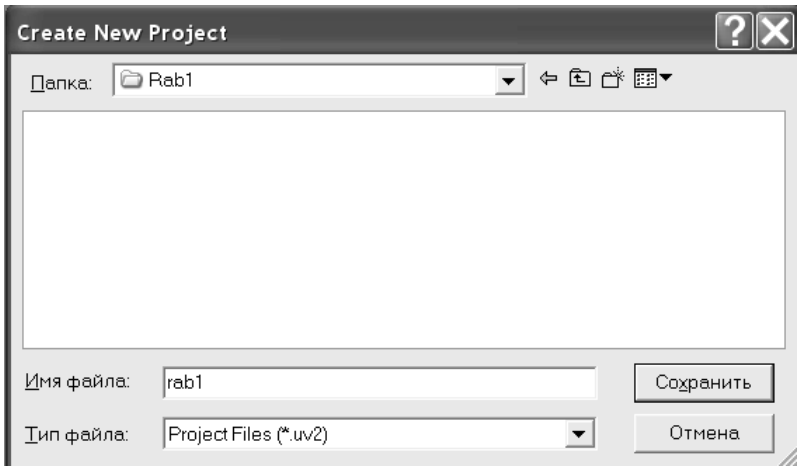


Рисунок 2.1 - Создание файла проекта

В появившемся окне среди фирм производителей принимается фирма Philips, среди ее изделий выбирается требуемый микроконтроллер, рекомендуется LPC2138. В правой части окна можно ознакомиться с характеристиками выбранного микроконтроллера (рисунок 2.2).

Мастер создания проекта предложит создать startup файл в проекте, без которого запуск и конфигурирование микроконтроллера пользователь должен будет выполнить вручную. Именно наличие этого файла позволяет выполнить программирование LPC21xx даже начинающему пользователю.

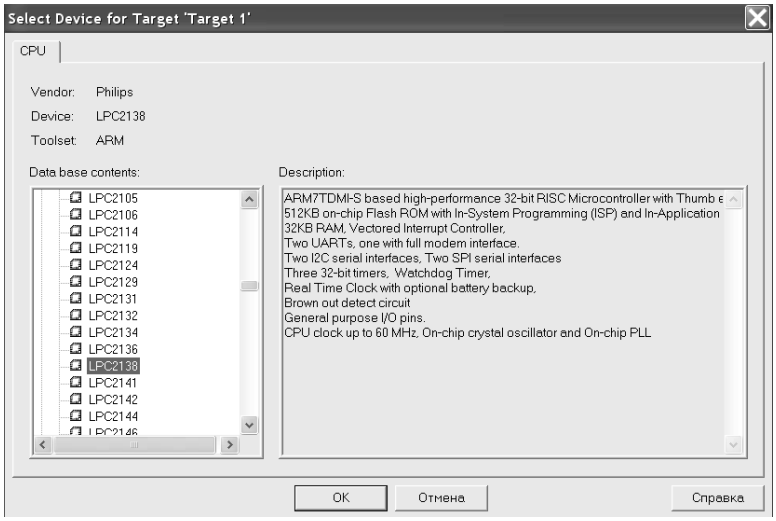


Рисунок 2.2 – Выбор микроконтроллера в проекте

В правой части проекта должна открыться панель рабочего пространства проекта, как показано на рисунке 2.3. Если панели не появилось, ее необходимо визуализировать нажатием на соответствующую кнопку.

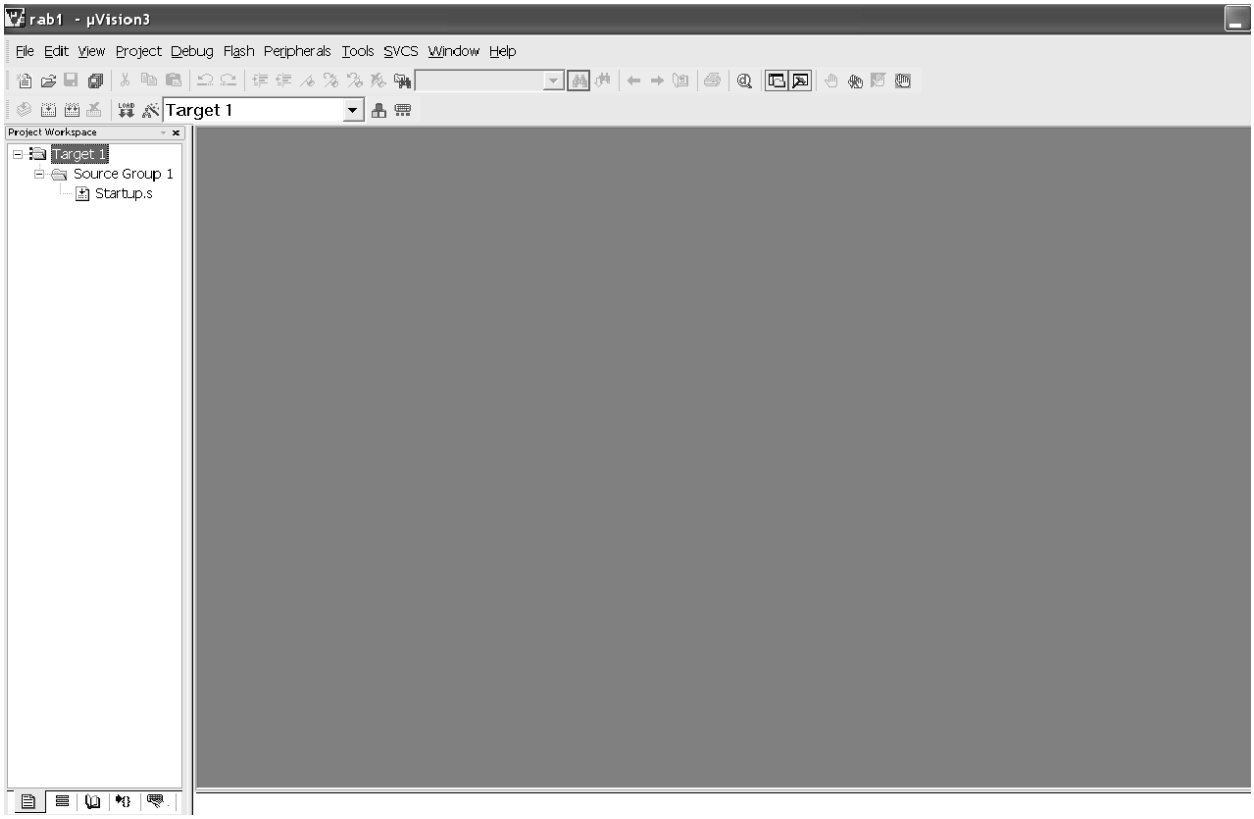


Рисунок 2.3 – Рабочее пространство проекта

Кроме startup.s в проект должен быть включен еще один или более файлов на языке Си или ассемблере. Для этого их можно предварительно создать файлы с любым именем и расширением .c в каталоге проекта. Можно воспользоваться встроенным редактором (кнопка «New»). Создан-

ный файл подключается к проекту из панели Project Workspace, для чего рекомендуется создать правой кнопкой мыши новую группу (New Group) и дать ей имя используемого языка программирования – C. Щелчок левой кнопкой мыши по созданной группе позволит добавить в нее созданные файлы.

Первой строкой во вновь созданном файле следует подключить файл описания адресов регистров периферийных устройств, так как по умолчанию они компилятору не известны (рисунок 2.4)

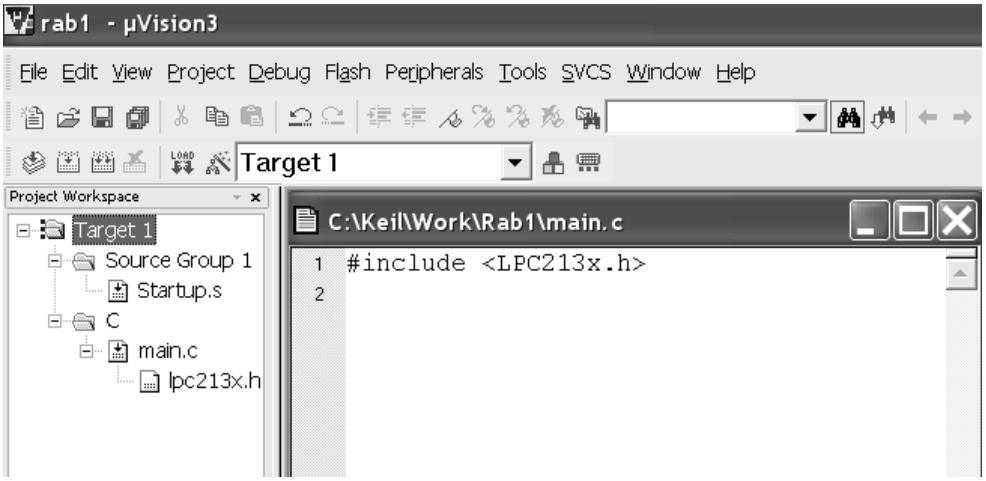


Рисунок 2.4 – Подключение файла описания периферийных устройств

Аналогично подключаются программные модули библиотек или разработанных ранее процедур и функций, необходимых в проекте. Они добавляются в программный код на этапе компиляции.

Для получения программного кода в формате, который в дальнейшем может быть использован для программирования микроконтроллера или симулирования в среде моделирования необходимо разрешить вывод в HEX формате. Для этого нажатием на кнопку “Options for Target” вызывается диалоговое окно свойств проекта, в котором во вкладке OUTPUT (рисунок 2.5) должен быть установлен чекбокс “Create HEX file”.

Окно свойств проекта так же позволяет установить частоту применяемого кварцевого резонатора, что важно при симуляции программы средствами Keil µVision.

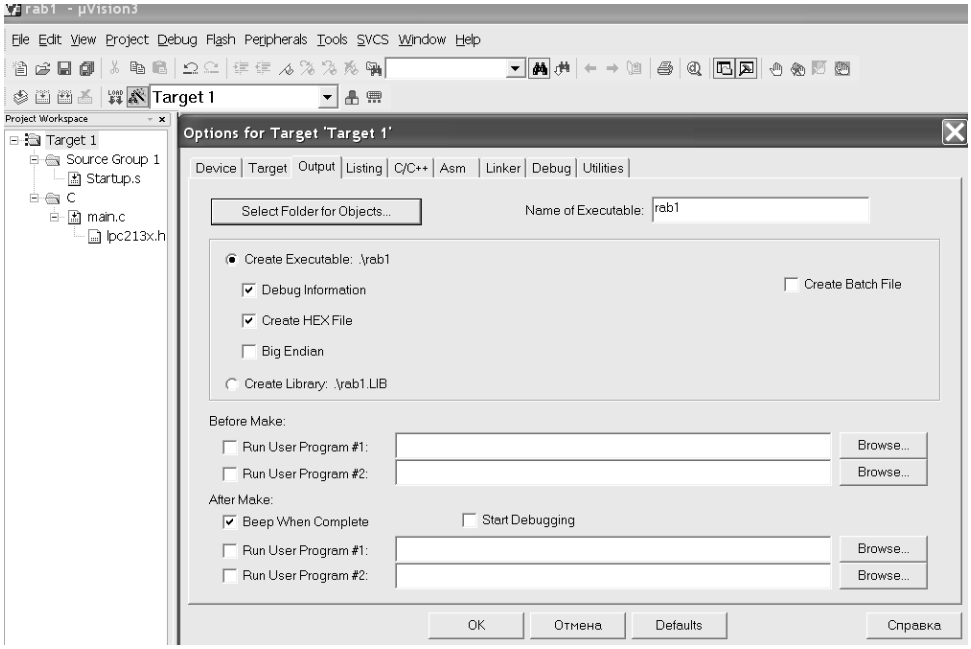


Рисунок 2.5 – Окно свойств проекта, выбор выходных файлов

При написании программного модуля необходимо помнить, что по умолчанию выполняется функция с именем main(), она содержит вызовы остальных процедур и функций.

2.2 Использование программного пакета Proteus

Разработанный программный код можно выполнить при помощи реального микроконтроллера или же при помощи среды симуляции. Программный пакет Proteus позволяет симулировать работу программного кода на микроконтроллере, в том числе с использованием реальных периферийных устройств.

Программный пакет Proteus состоит из программы симулятора ISIS и программы разводки печатных плат ARES, которая может быть использована при практическом изготовлении микроконтроллерных устройств.

На рисунке 2.6 показан фрагмент рабочего стола симулятора ISIS программного пакета Proteus.

Разработка проекта начинается с добавления необходимых элементов. Для этого нажатием на кнопку P вызывается меню Pick Device, показанное на рисунке 2.7. В окне Keywords вводятся полные или сокращенные наименования элементов, что позволяет выполнить их быстрый поиск. Без ключевых слов навигация осуществляется при помощи подменю.

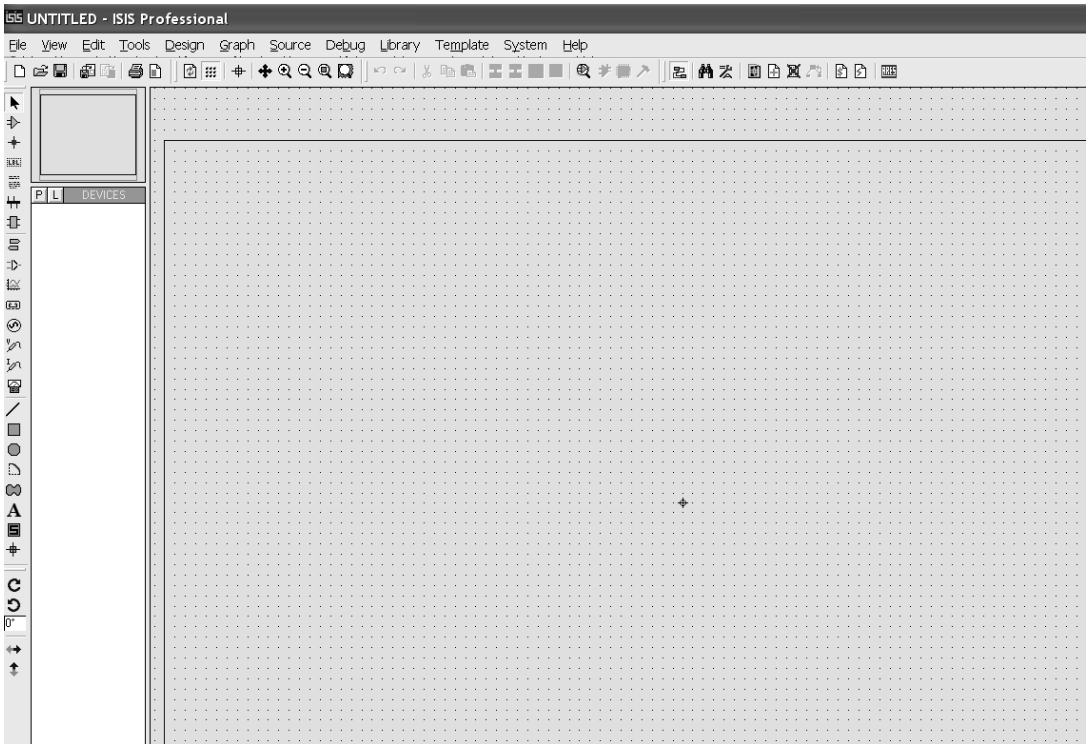


Рисунок 2.6 – Рабочий стол симулятора ISIS программного пакета Proteus

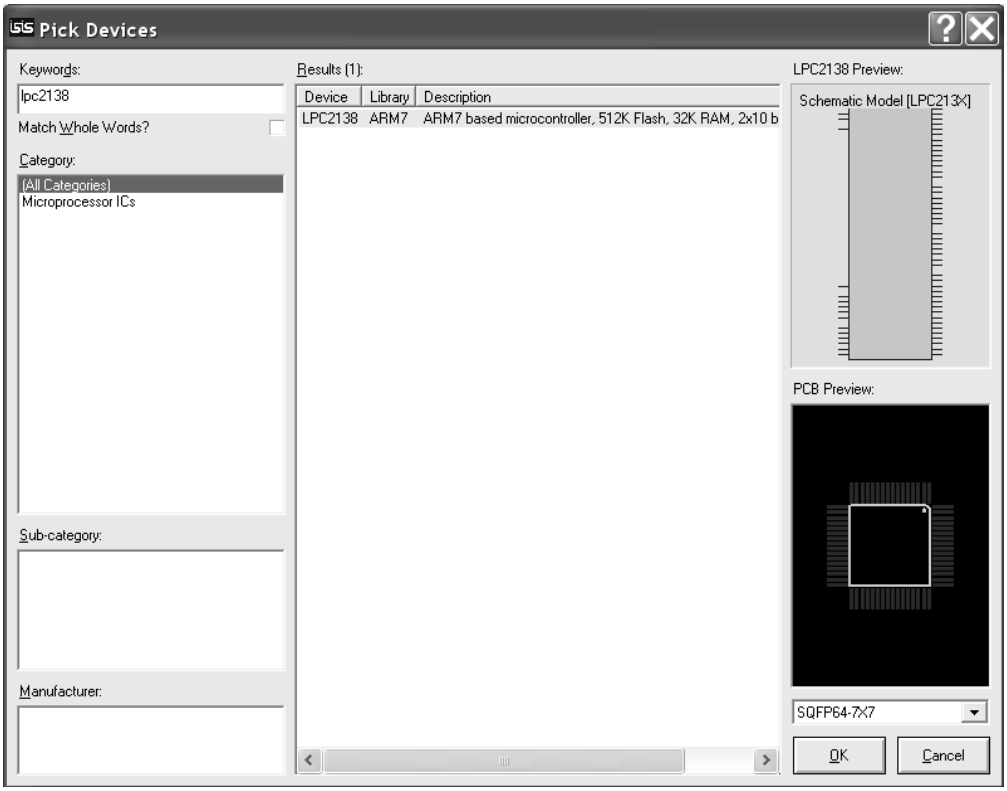


Рисунок 2.7 – Меню Pick Device симулятора ISIS программного пакета Proteus

3 ОРГАНИЗАЦИЯ ДИСКРЕТНОГО ВВОДА-ВЫВОДА GPIO

3.1 Блок подключения пинов микроконтроллера

Ввиду ограничения у существующих микроконтроллеров количества контактов, используемых для связи периферийных модулей с внешним окружением, каждому контакту (пину) микроконтроллера присвоено до четырех альтернативных функций.

Например пин P0.0 микроконтроллера LPC2138 имеет в качестве базовой функции дискретный ввод-вывод (GPIO), а в качестве альтернативных – вход последовательного приемопередатчика TxD и выход широтно-импульсного модулятора (ШИМ) PWM1.

Управление режимом пина осуществляется при помощи регистров выбора функции пина PINSEL0-PINSEL2. Каждые два бита этих регистров влияют на функцию соответствующего пина. Например, биты D0-D1 регистра PINSEL0 влияют на функции пина P0.0, биты D2-D3 регистра PINSEL0 влияют на функции пина P0.1, и т.д. (таблица 3.1)

Структура регистра Pinsel:

-16 пинов 32 бита, по 2 бита на пин.

Таблица 3.1 Режимы регистра PINSEL0

Биты \ Код	0 0 (0)	0 1 (1)	1 0 (2)	1 1 (3)
D01 D00	GPIO	TxD1	PWM1	Резерв
D03 D02	GPIO	RxD1	PWM3	EINT0
D05 D04	GPIO	SCL0(I2C0)	CAP0.0	Резерв
D07 D06	GPIO	SDA0(I2C0)	MAT0.0	EINT1
D09 D08	GPIO	SCLK0(SPI0)	CAP0.1	AD0.6
D11 D10	GPIO	MISO0 (SPI0)	MAT0.1	AD0.7
D13 D12	GPIO	MOSI0 (SPI0)	CAP0.2	Резерв
D15 D14	GPIO	SSEL0(SPI0)	PWM2	EINT2
D17 D16	GPIO	TxD2	PWM4	Резерв
D19 D18	GPIO	RxD2	PWM6	EINT3
D21 D20	GPIO	RTS(UART1)	CAP1.0	AD1.2
D23 D22	GPIO	CTS(UART1)	CAP1.1	SCL1(I2C1)
D25 D24	GPIO	DSR(UART1)	MAT1.0	AD1.3
D27 D26	GPIO	DTR(UART1)	MAT1.1	AD1.4
D29 D28	GPIO	DCD(UART1)	EINT1	SDA1(I2C1)
D31 D30	GPIO	RI(UART1)	EINT2	AD1.5

Программирование регистров выбора функции пина удобно выполнять при помощи операций сдвига.

Для этого используем следующую формулу:

$PINSEL\ \theta = \sum_{i=0}^{15} Fi(1 << 2i)$, \\для пинов P0.0 – P0.15 порта P0,

где Fi – Функция i-го пина

$PINSEL1 = \sum_{i=16}^{31} Fi(1 << 2(i-16))$, \\для пинов P0.16 – 0.31 порта P0

$PINSEL2 = \sum_{i=16}^{31} Fi(1 << 2(i-16))$ \\для пинов P1.16 – 1.31 порта P1

Например, пусть необходимо сконфигурировать пины Port0 таким образом, что бы P0.1 функционировал в режиме EINT0, а P0.7 в режиме PWM2:

$PINSEL\ \theta = 3 << (1 \cdot 2) + (2 << 7 \cdot 2)$, \\P0.1 в EINT0 + P0.7 в PWM2

Остальные пины будут находиться в режиме GPIO.

3.2 Программирование GPIO

Цифровой ввод-вывод данных представляет собой систему с произвольным назначением функций пин микроконтроллера для выполнения ввода или вывода дискретных сигналов. На рисунке 3.1 показана внутренняя структурная схема подключения пина P0.0, иллюстрирующая конфигурирование отдельного пина в различных режимах работы.

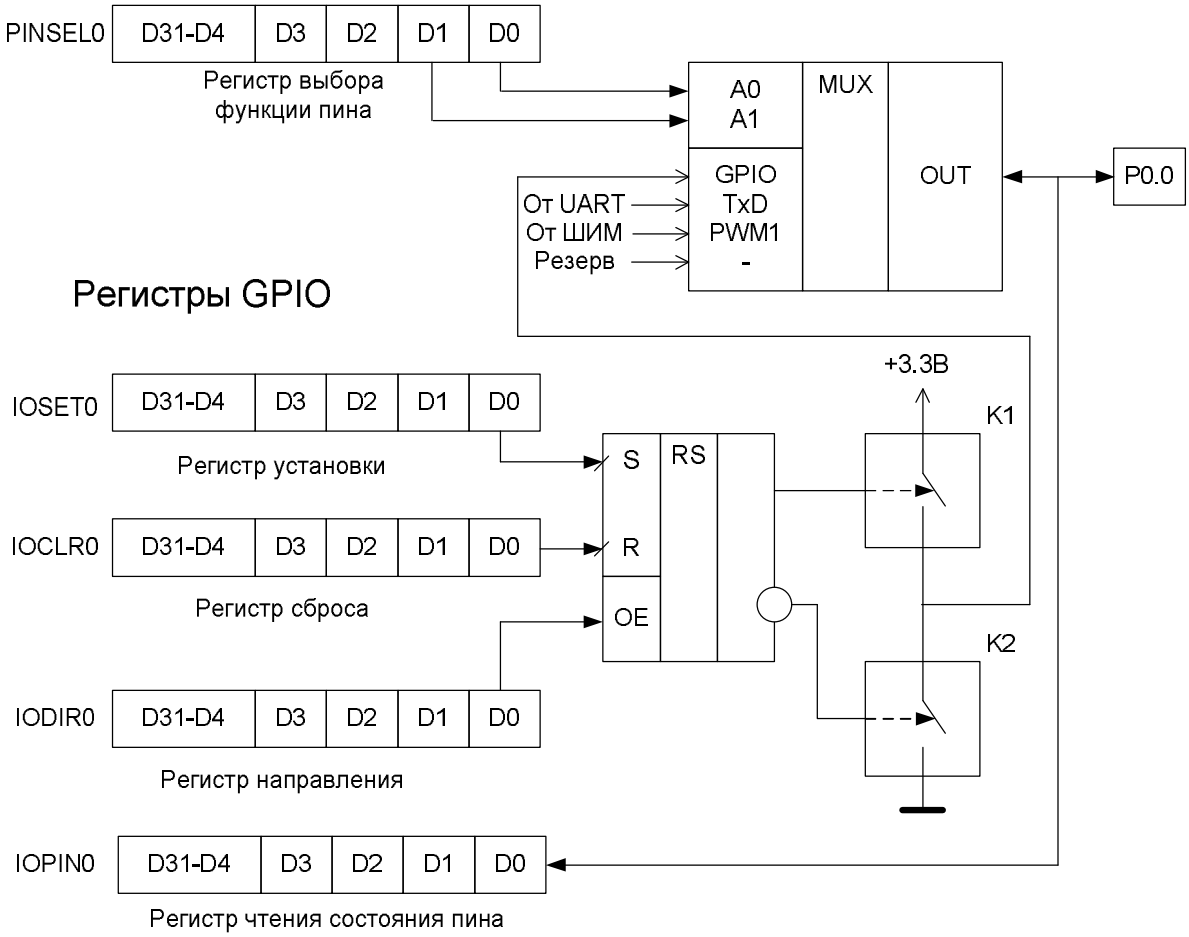


Рисунок 3.1 – Внутренняя структурная схема подключения пина P0.0 микроконтроллера LPC2138

Режим работы определяется состоянием регистра PINSEL0, по умолчанию это режим GPIO. При этом пин P0.0 микроконтроллера подключается через мультиплексор MUX к транзисторным ключам К1 и К2, управляемых от RS триггера. Запись единичного значения в соответствующий бит регистра IOSET0 переводит RS триггер в единичное состояние, при этом ключ К1 отпирается, а К2 закрыт – на выход поступает напряжении 3.3В – сигнал логической единицы.

Регистр IOCLR0 выполняет сброс RS триггера, при этом закрывается ключ К1 и отрывается ключ К2, замыкая соответствующий пин на землю и выдавая на соответствующий пин микроконтроллера сигнал логического нуля.

Соответствие пин определяется по порядковому номеру пина – P0.0 соответствует D0, P0.1 – D1, P0.2 – D2 и т.д. Порт P1 относиться к регистрам IOSET1, IOCLR1, IODIR1 и IOPIN1 соответственно.

Ключи активны, если соответствующий бит регистра IODIR0 находится в единичном состоянии, разрешая передачу данных. В противном случае, GPIO работает в режиме приема. Но в любом случае, состояние пина микроконтроллера, минуя мультиплексор, отображается в соответ-

ствующем бите регистре IOPIN0. По умолчанию, пины микроконтроллера находятся в режиме ввода GPIO.

Регистры IOSET и IOCLR устанавливают или сбрасывают только те RS триггеры в соответствующих которым разрядах записаны лог. единицы, позиции с лог. нулями не изменяются. Например, команда для установки пинов P0.1 и P0.5 в единичное состояние на языке Си можно записать как:

$$IOSET0 = (1 << 1) + (1 << 5); \qquad \text{// SET P0.1 + P0.5}$$

В следующем машинном цикле содержимое регистров IOSET и IOCLR очищается, поэтому переключение состояния RS триггеров происходит при каждой новой записи в них.

3.3 Пример программирования GPIO

Как правило, при помощи GPIO реализуют программы управления логическими функциями или микропрограммные автоматы. Простейшие примеры предусматривают ожидания ввода требуемой комбинации внешних сигналов через пины микроконтроллера сконфигурированные в режиме ввода и вывод требуемой комбинации сигналов через пины микроконтроллера, сконфигурированные в режим выхода.

Рассмотрим пример управление насосной станцией водоочистных сооружений с помощью контроллера:

К пину P0.0 подключен верхний датчик уровня ямы, а к пину P0.1 нижний датчик уровня. При срабатывании верхнего датчика на пин P0.3 поступает логическая единица, которая закрывает базу полевого транзистора, который служит ключом (он пропускает ток через реле), ток пройдя через реле, замкнет его контакт и запустит ЭД привода насоса.

При срабатывании нижнего датчика (установка P0.1 в единицу), пин P0.3 очищается, т.е. на базу транзистора не проходит ток и ЭД выключается.

На рисунке 3.2 представлена функциональная схема управления насосной станцией водоочистных сооружений, с помощью контроллера.

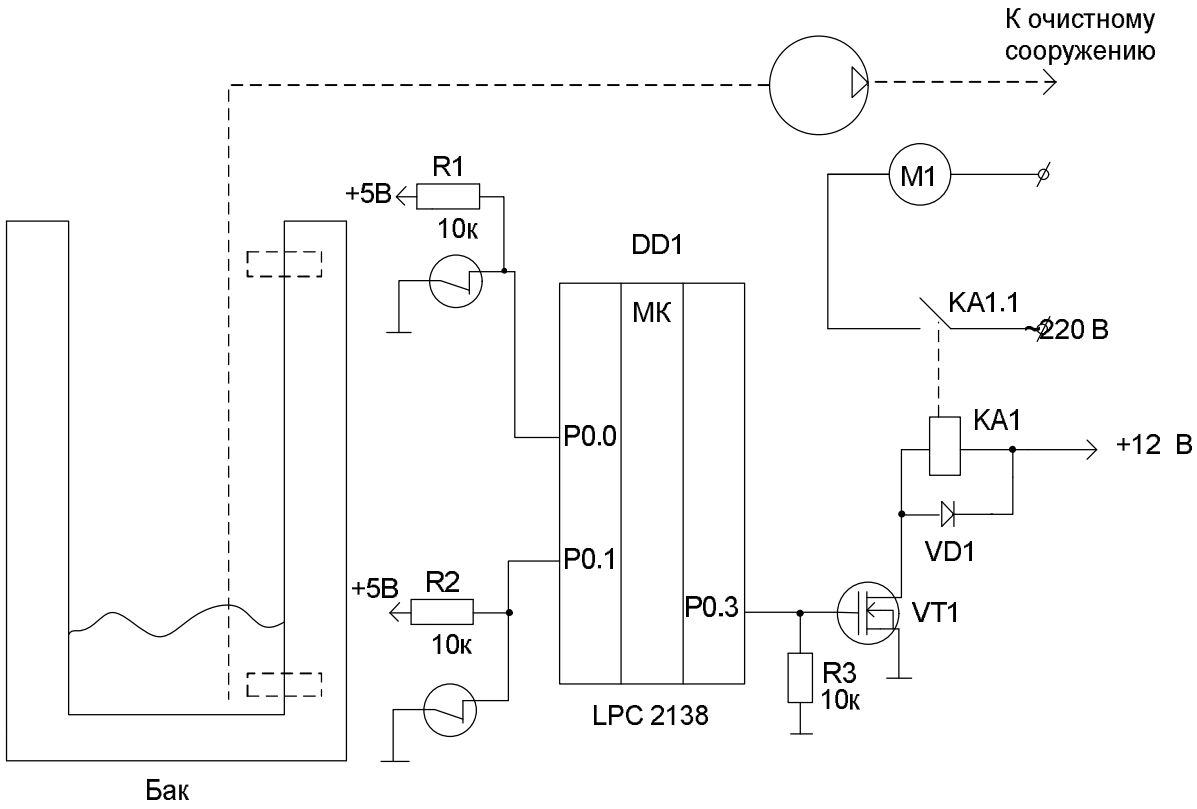


Рисунок 3.2 – Функциональная схема СУ насосная станция водоочистных сооружений

На рисунке 3.3 представлена модель станции водоочистных сооружений, с помощью контроллера в среде Proteus.

.

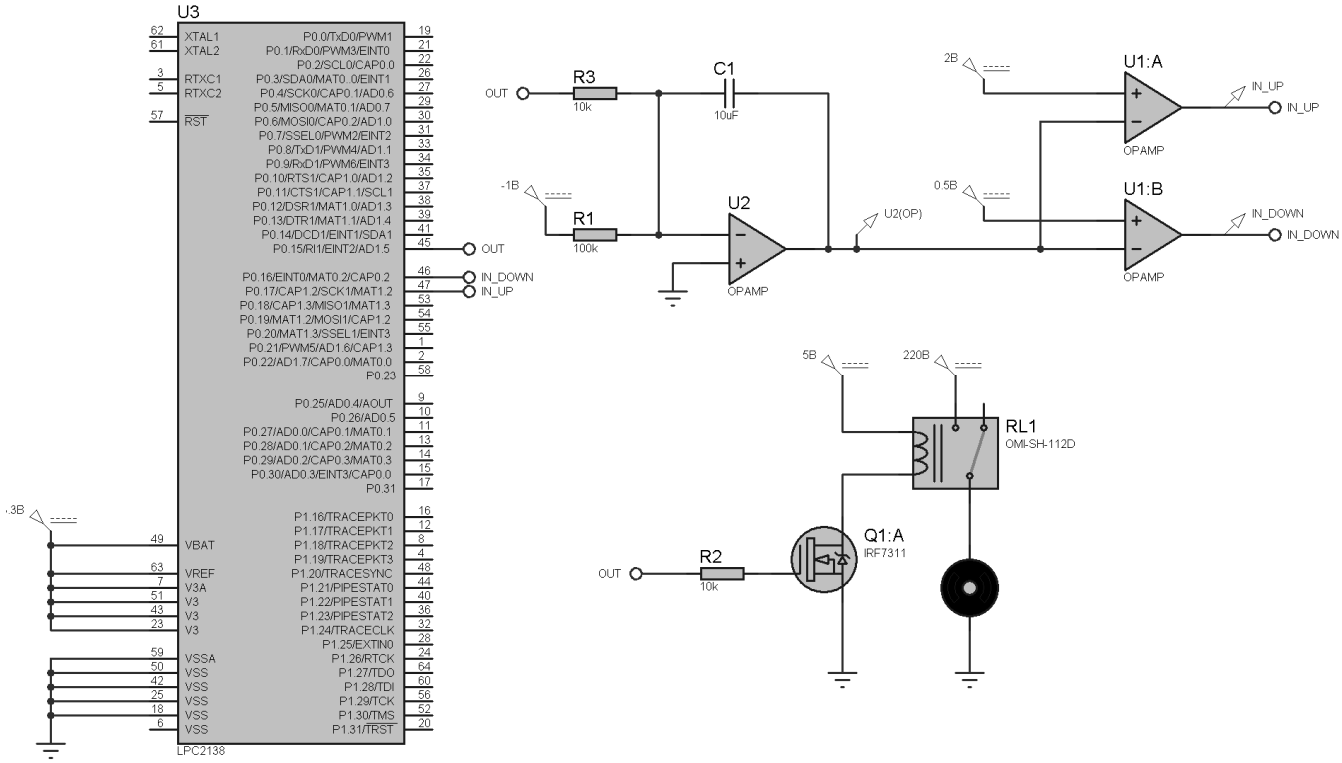


Рисунок3.3 – Модель СУ станции водоочистных сооружений в среде Proteus

На модели стенда показан МК LPC 2138 и подключенные к нему датчики и привод насоса очистных сооружений, а именно:

– RL2 и RL3, верхний и нижний датчики ямы сточных вод,
– Q1,транзистор открывания питания ЭД,
– RL1,реле напряжения для запуска ЭД,
– Motor, привод насоса очистных сооружений.

Пример программы, управления ЭД насоса:

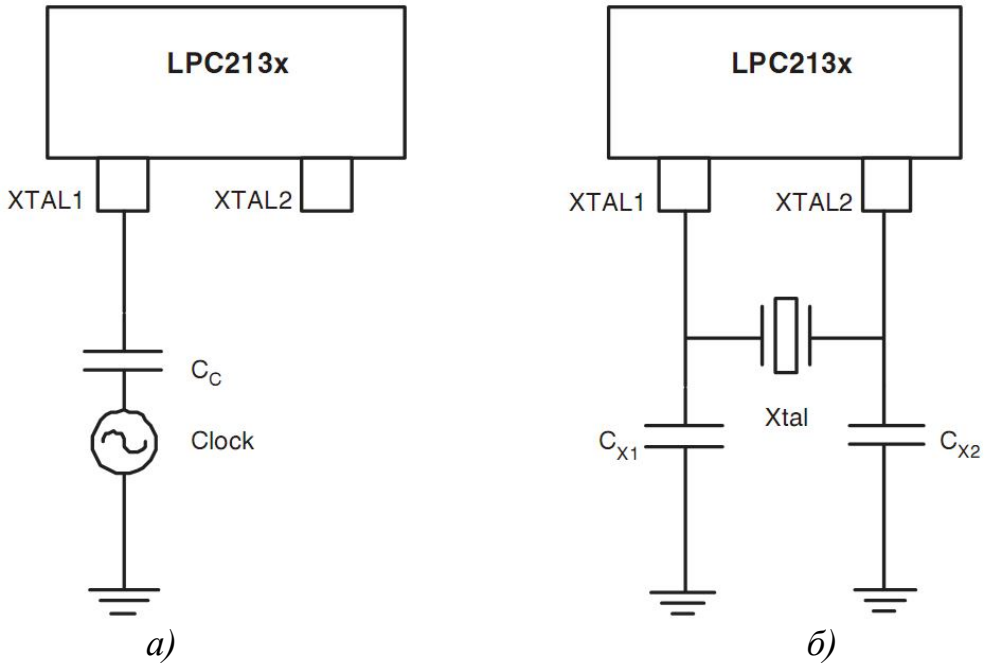
```
#include <LPC213x.h>
int main(void)
{
IODIR0 = (1<<3);          // Установить пин P0.3 выходом
                          // Единичное значение бита рег.IODIR0 соответствует работе
                          // пина порта 0 в режиме выхода
IOCLR0 = (1<<3);          //Очищение бита для предотвращения самопроизвольного включения насоса
While (1){                //зацикливание программы в бесконечный цикл
While (IOPIN0&(1<<0));    //условие по которому пока не сработает верхний датчик насос не включиться, если условие выполнено то программа выполнит следующее
IOSET = (1<<3);           //установить бит 3 порта 0 в «единицу» для запуска ЭД
While(IOPIN0&(1<<1));     //условие по которому пока не сработает нижний датчик насос будет работать, если условие не выполнено то программа //выполнит следующее
IOCLR0 = (1<<3);          //очистить бит 3 порта 0, тем самым выключить ЭД насоса
        }
}
```

Предложенная программа реализует самый простой вид управления, основанный на линейном графе. Это решение не является оптимальным и используется только в качестве учебного примера для начинающих.

4 БЛОК СИСТЕМНЫХ ФУНКЦИЙ

4.1 Кварцевый генератор

В качестве источника синхросигнала ARM7 микроконтроллеры LPC21xx могут использовать импульсный сигнал со скважностью равной двум и частотой от 1 до 50МГц, подключенный к внешнему входу XTAL1 (рисунок 4.1 а)).



а) *б)*

а – внешнего; б – внутреннего
Рисунок 4.1 – Схема подключения генератора

Внутреннюю синхронизацию может обеспечить применение кварцевого резонатора на частоту 1 – 30 МГц, включенного по схеме рис.4.1б. Если использована встроенная система ФАПЧ (PLL system), то частота кварцевого резонатора ограничена диапазоном 10 – 25МГц, для стабильной работы кварцевого резонатора необходимо на вход подключить 2 конденсатора как на рисунке 4.1б для предотвращения запуска резонатора на высших гармониках.

Диапазон 10 – 25МГц для кварцевого резонатора обязателен так же и при необходимости внутрисхемного программирования по последовательному порту (ISP) – это самый простой способ программирования LPC21xx при минимальном требуемом оборудовании.

По умолчанию используется режим работы со встроенным ФАПЧ, его структура представлена ниже (рисунок 4.2).

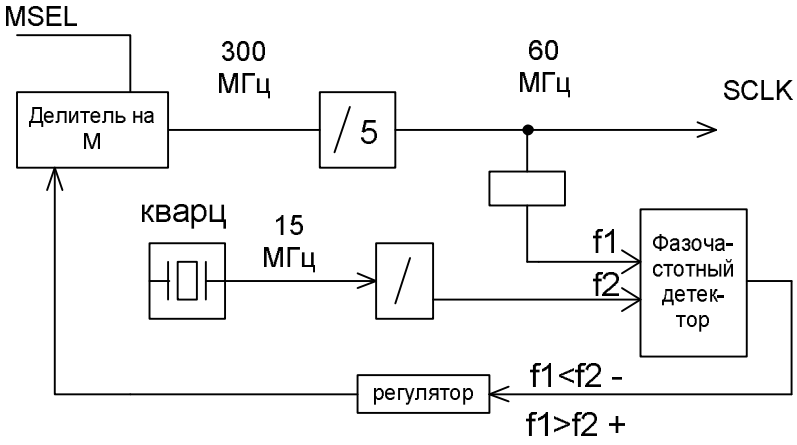


Рисунок 4.2 – Структура ФАПЧ

Он программируется рядом регистров PLL, но наиболее оптимально управлять режимами ФАПЧ непосредственно из Keil µVision. Вкладка Configuration Wizard файла startup.s позволяет включить или выключить PLL, а также указать значения множителя MSEL и делителя PSEL, определяющие частотные настройки системы (рисунок 4.3).

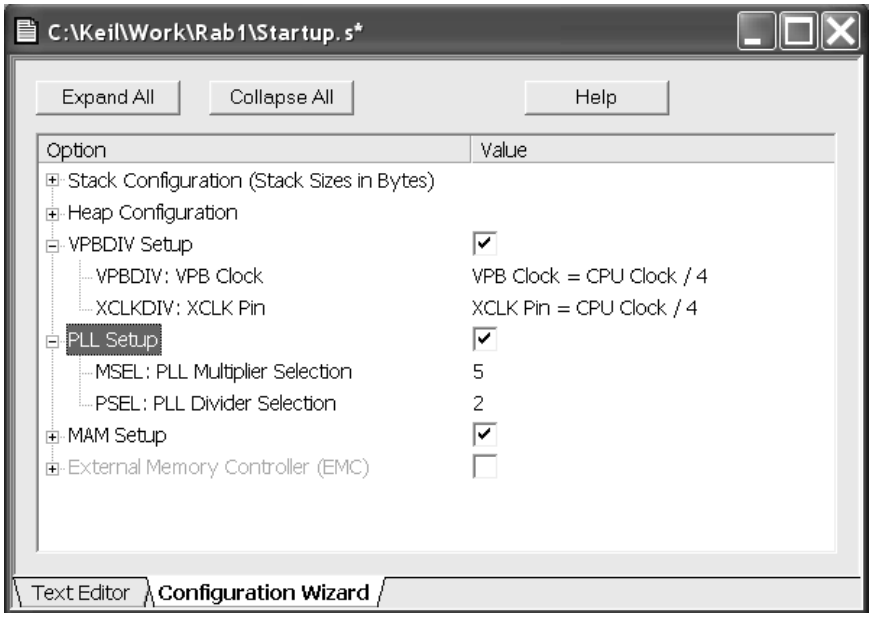


Рисунок 4.3 – Настройка коэффициентов MSEL и PSEL

Коэффициент MSEL определяет частоту ядра, которая рассчитывается по формуле (4.1) :

$$CCLK = MSEL \cdot f_{\hat{e}\hat{a}\grave{a}\eth\ddot{o}\grave{a}}; \qquad (4.1)$$

где $f_{кварца}$ – частота кварцевого резонатора, 10..25МГц.

Так как предельная частота ядра CCLK для LPC2138 не должна превышать 60МГЦ, то допустимое значение MSEL лежит в диапазоне 1..6 в зависимости от частоты кварцевого резонатора.

Коэффициент PSEL определяет частоту встроенного PLL генератора, (4.2) :

$$f_{PLL} = CCLK \cdot 2 \cdot PSEL; \quad (4.2)$$

Допустимое значение частоты встроенного генератора PLL находиться в диапазоне 156…320 МГц, тогда при CCLK=60МГц значение PSEL будет равно двум.

4.2 APB делитель

Частота работы периферийной шины PCLK определяется частотой системного синхросигнала CCLK и коэффициентом деления APB DIVIDER, как показано на рисунке 4.4:

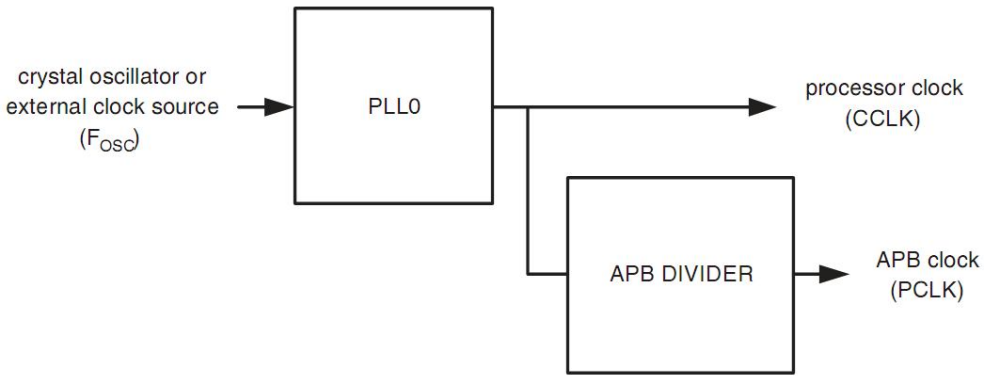


Рисунок 4.4 – Структура получения частот CCLK и PCLK

Значения PCLK могут быть изменены во вкладке Configuration Wizard файла startup.s, показанной на рисунке 4.1, необходимо установить чекбокс VPBDIV Setup и выбрать одно из трех возможных значений для пункта VPBDIV:VPB Clock. Эти значения соответствуют CCLK, CCLK/2 и CCLK/4. Последнее значение является коэффициентом деления по умолчанию.

Таким образом, если значения делителя APB не изменялось, то при частоте системного синхросигнала CCLK=60МГц частота периферийного синхросигнала PCLK=15МГЦ.

4.3 Управление питанием

LPC21xx поддерживают два режима энергосбережения: Idle mode и Power-down mode. В Idle mode процессор прекращает выполнение инструкций и ожидает событий прерывания или сброса, при которых вновь возобновляет работу. Периферийные системы продолжают функционировать и должны генерировать прерывание, если необходима процессорная обработка полученных данных. Переход в Idle mode выполняется установкой в единичное состояние бита 0 регистра PCON, например, при помощи команды:

```
PCON|=1;          // Перейти в режим Idle mode
```

В Power-down mode останавливается работа кварцевого резонатора или прохождение внешних импульсов синхронизации. Энергопотребление микроконтроллера становится практически равным нулю. Выход из Power-down mode выполняется по сигналу внешнего прерывания. Переход в Power-down mode выполняется установкой в единичное состояние бита 1 регистра PCON, например, при помощи команды:

```
PCON|=2;          // Перейти в режим Power-down mode
```

Для сокращения энергопотребления микроконтроллера неиспользуемые периферийные устройства могут быть отключены при помощи регистра управления питанием периферийных устройств PCONP (стр33. [1]). По умолчанию практически все биты этого регистра установлены в единичное состояние, что соответствует рабочему режиму периферийных устройств.

4.4 Сброс

Сброс – процедура сопровождающаяся очисткой регистров МК и запуском программы нулевого адреса.

Процедура сброса микроконтроллеров ARM7 инициируется внешними или внутренними источниками. При этом контроллер выполняет начальную установку регистров периферийных устройств и начинает выполнение программы по адресу 0000 [1] стр. 43 – 45. На рисунке 4.5 представлена функциональная схема внутренней логики сброса.

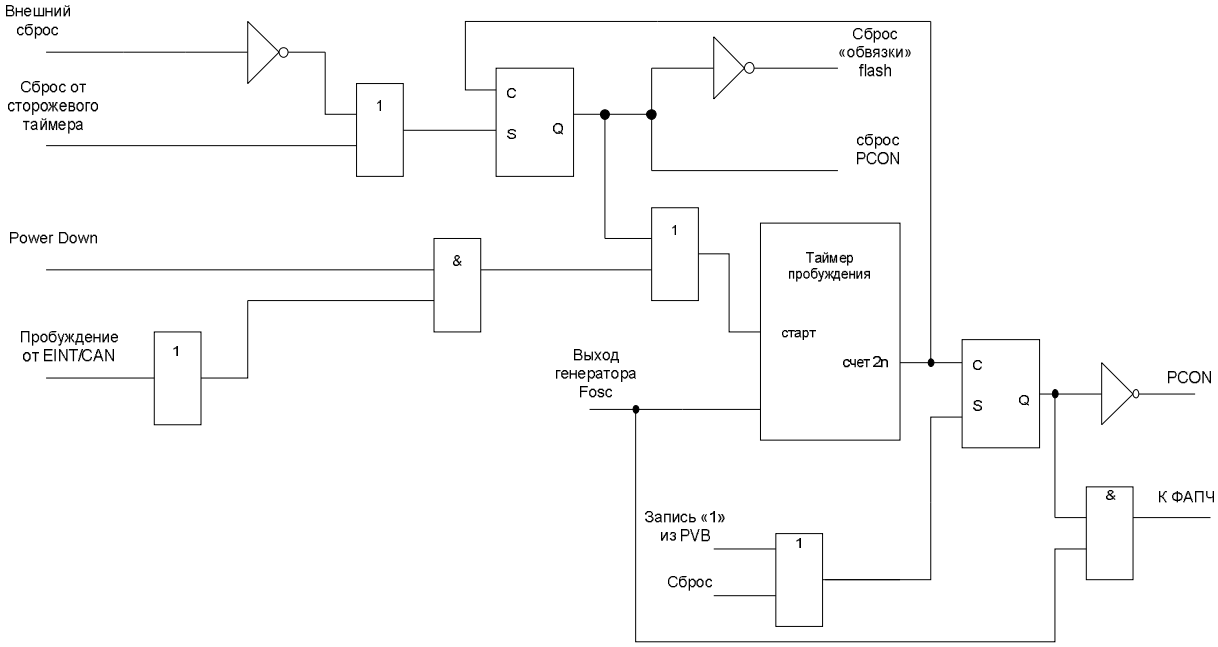


Рисунок 4.5 – Функциональная схема внутренней логики сброса

Всего существуют пять источников сброса:

– RESET пин контроллера

– Watchdog Reset

– Power-On Reset (POR)

– Brown-Out-Detector (BOD) Reset

Перепрограммирование

RESET пин микроконтроллера является внешним источником сброса, остальные – внутренними.

Особенности внешнего сброса:

– Триггер Шмидта во входной цепи

– Фильтр импульсных помех

– Минимальная длительность внешнего сброса 10мС при включении питания и 300нС в остальных режимах

– Совместно с P0.14 вход RESET используется для включения внутрисхемного программирования

Схема внешнего RESET представлена на рисунке 4.6

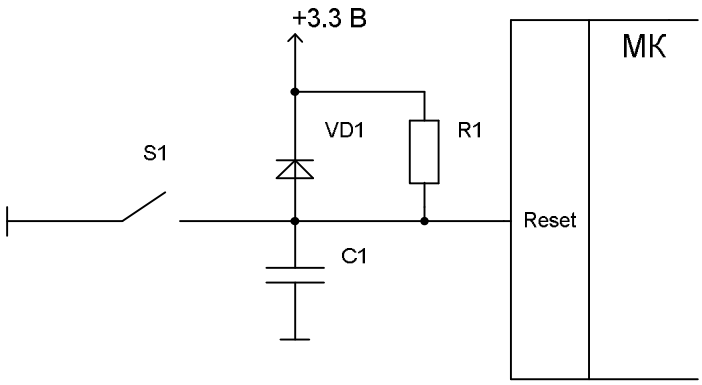


Рисунок 4.6 – Схема подключения внешнего RESET

R1 и C1 формируют принудительный сигнал сброса при включении, VD1 применяется для быстрого разряда конденсатора при включении питания.

Watchdog Reset обеспечивает сброс контроллера при зависании выполняемой программы, обеспечивая перезагрузку контроллера при отсутствии обращения к его feed регистру в течении некоторого временного интервала.

Power-On Reset обеспечивает сброс контроллера при достижении напряжением питания минимального рабочего значения при включении, предназначен для предотвращения хаотического функционирования контроллера при низком напряжении питания.

Brown-Out-Detector (BOD) Reset выполняет сброс контроллера при опускании напряжения питания до значения меньшего 2.6В, предотвращает хаотическое функционирование контроллера при низком напряжении питания.

Перепрограммирование также является сбросом, т.к. при нем все регистры МК устанавливаются в «ноль», для реализации перепрограммирования используется пин P0.14 который включает схему программатора, рисунок 4.7

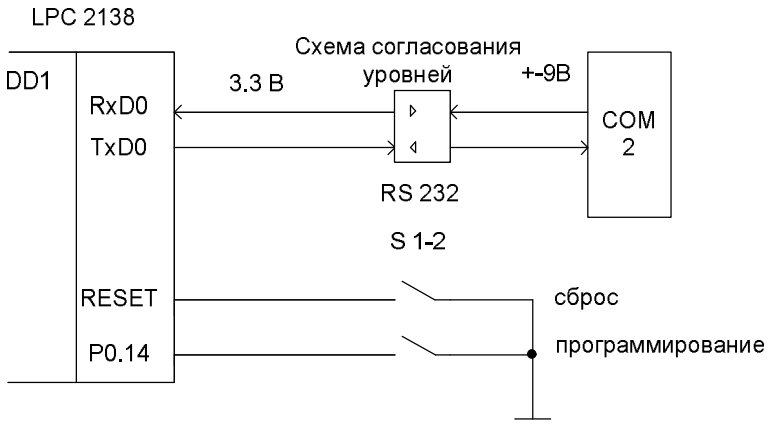


Рисунок 4.7 – схема подключения МК для программирования

Источник сброса может быть определен при помощи регистра идентификации источника сброса, структура которого показана на рисунке 4.8.

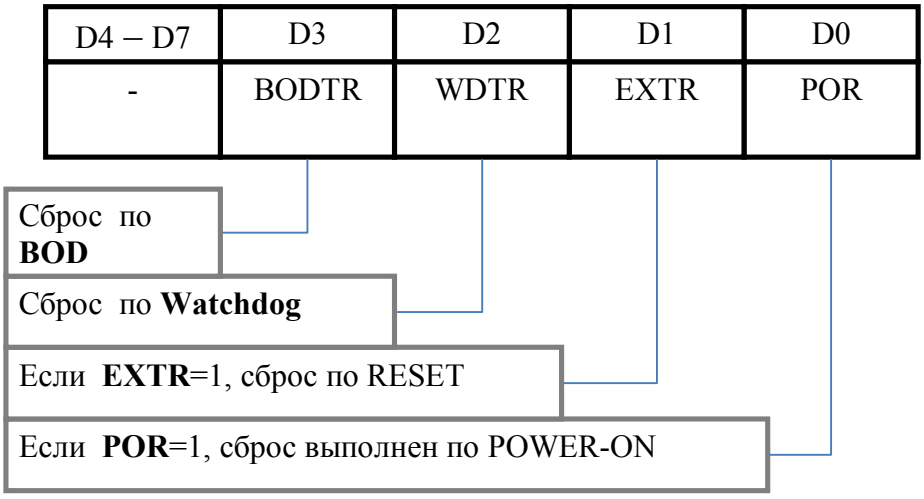


Рисунок 4.8 – Регистр идентификации источника сброса RSID

Для получения значения источника сброса следует анализировать значение отдельных бит, например используя команду логического умножения с маской тестируемых бит:

if (*RSID*& 02){....}; // Если выполнен сброс по RESET выполнить операции {…}.

4.5 Программирование сторожевого таймера

Сторожевой таймер (WatchDog Timer или WDT) представляет собой устройство, контролирующее работоспособность программной среды микроконтроллера и выполняющий его сброс в случае “повисания” программы. Особенностями WDT, применяемого в ARM7 микроконтроллере LPC21xx:

– Выполняет сброс контроллера, если WDT не перезагружается при помощи feed последовательности (например, при зависании программы).

– Наличие режима отладки, при котором сброс не выполняется, но возможно прерывание.

– Некорректная feed последовательность вызывает сброс или прерывание, если оно разрешено.

– Устанавливается программно доступный флаг WDTR при сбросе от WDT.

– В основе WDT находиться программируемый 32-разрядный таймер с внутренним предварительным делителем.

На рисунке 4.9 показана функциональная схема работы WDT.

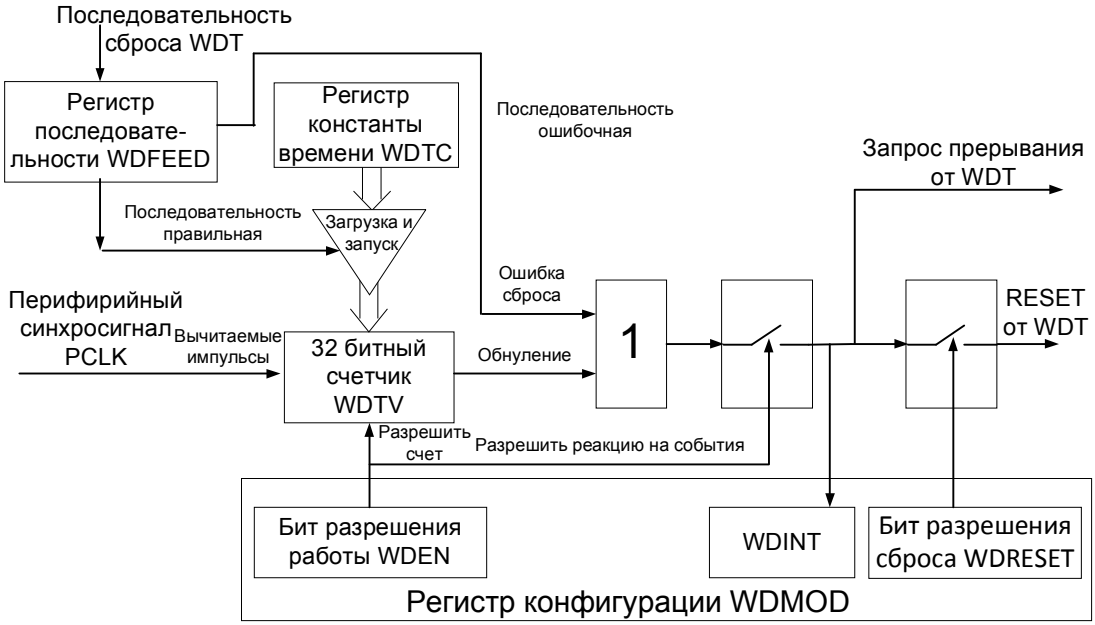


Рисунок 4.9 – Функциональная схема работы WDT

Импульсы периферийного синхросигнала PCLK через делитель на четыре вычитают из 32-битного счетчика WDTV единицу в каждом импульсе, т.е. декрементируют его. Работа счетчика разрешена, если установлен бит WDEN регистра конфигурации сторожевого таймера WDMOD, структура которого показана на рисунке 4.10.

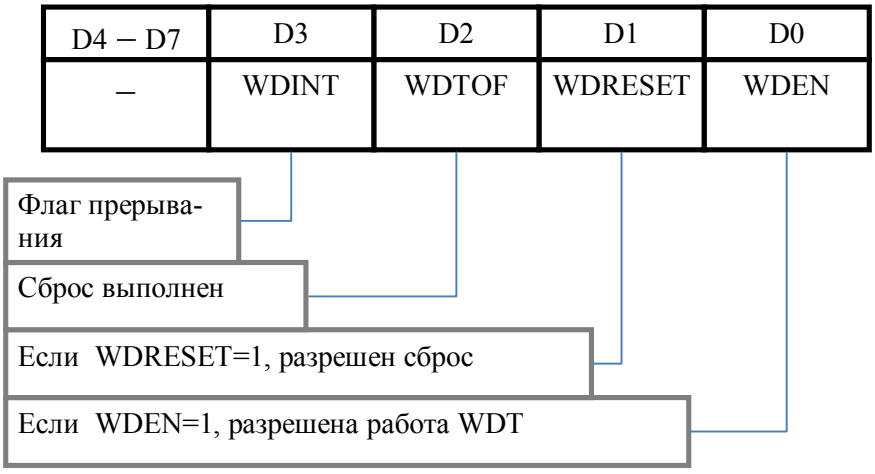


Рисунок 4.10 – Регистр конфигурации сторожевого таймера WDMOD

Для запуска WDT необходимо загрузить в регистр константы таймера WDTC число, являющееся периодом срабатывания сторожевого таймера и запустить WTC при помощи правильной FEED последовательности:

WDFEED=0x55;
WDFEED=0xAA.

Для выполнения декремента счетчика WDTV сторожевого таймера бит WDEN регистра конфигурации WDMOD должен быть установлен. Для разрешения сброса контроллера при обнулении регистра WDTV должен быть установлен флаг WDRESET. Например:

WDMOD = (1<<2); // Разрешить работу + сброс по WDT.

Если после разрешения работы и сброса содержимое WDTV будет очищено или будет выполнена неверная FEED последовательность (в WDFEED будут записаны числа, отличные от 0x55 и 0xAA) то произойдет сброс микроконтроллера, при этом бит WDTOF будет установлен. Значение бита WDTOF может быть использовано программой для обнаружения факта зависания и срабатывания сторожевого таймера.

Интервал задержки сторожевого таймера определяется регистром константы таймера WDTC по формуле 4.3:

$$T_{WDT} = T_{PCLK} \cdot WDTC \tag{4.3}$$

где $WDTC$ – содержимое регистра константы сторожевого таймера, $256..2^{32}$,

T_{PCLK} - период периферийного синхросигнала, обычно 1/15000000 с.

Если в WDTC записывается число меньшее 256, автоматически загружается 256. Если требуемый период задержки сторожевого таймера T_{WDT} задан, то требуемое значение WDTC определяется по формуле 4.4:

$$WDTC = T_{WDT} \cdot f_{PCLK} \tag{4.4}$$

где f_{PCLK} - частота периферийного синхросигнала, обычно 15000000 Гц.

Запись FEED последовательности должна выполняться в главном программном цикле микроконтроллера, который должен выполняться за время меньшее, чем период сброса. Невыполнение этого условия приведет к обнулению регистра WDTV и, при установленном флаге WDRESET, перезагрузке микроконтроллера.

Пример программы, демонстрирующей работу сторожевого таймера:

```
// Спустя 1 с после нажатия на кнопку выполняется сброс по WDT.
#include <LPC213x.h>
int main(void)
 {
unsigned int var;
unsigned int var1;
unsigned int var2;
```

```
IODIR0=(1<<2);              //Установить пин P0.0 в  выходом
IOSET0=(1<<2);              //Потушить светодиод P0.0
IODIR1=(1<<4);              //Установить пины P1.16-  выходом
while (IOPIN0&(1<<1));      //Ожидать нажатия на кнопку
IOCLR0=(1<<4);              //Зажечь светодиод
var=40;                     //Вывод на индикаторы номера варианта = 40
var1=var/10;                //Выполним двоично-десятичное преобразование

var2=var-var1*10;
var=(var1<<20)+(var2<<16);
IOCLR1=(1<<4);              //Очистим индикатор
IOSET1=var;                 //Выведем номер варианта на индикатор
                            //Период времени срабатывания T=1c

WDTC=15000000;
WDMOD=(1<<2);               //Разрешить работу WDT и сброс по его обнулению
                            //D0 - WDEN     - Разрешение работы сторожевого таймера
                            //D1 - WDRESET - Разрешен сброс от сторожевого таймера
                            //D2 - WDTOF - Флаг обнуления (окончания счета) сторожевого таймера
                            //D3 - WDINT - Флаг прерывания от сторожевого таймера
WDFEED=(1<<2)+(1<<4);
WDFEED=(1<<4);              //Выполнить последовательность запуска WDT

                            //Последовательность верна только для Proteus.
                            //При симуляции в Keil  - верно AA затем 55.
                            //Если нужно продемонстрировать немедленный сброс
                            //строчки ниже раскомментируются
                            //WDFEED=(1<<4);
                            //Выполнить неверную последовательность WDT,
WDFEED=(1<<4);              //если нужен немедленный сброс
while (1) {};               //Бесконечный цикл
return 0;
}
```

На рисунке 4.11 показана модель стенда в среде Proteus, иллюстрирующая работу WDT.

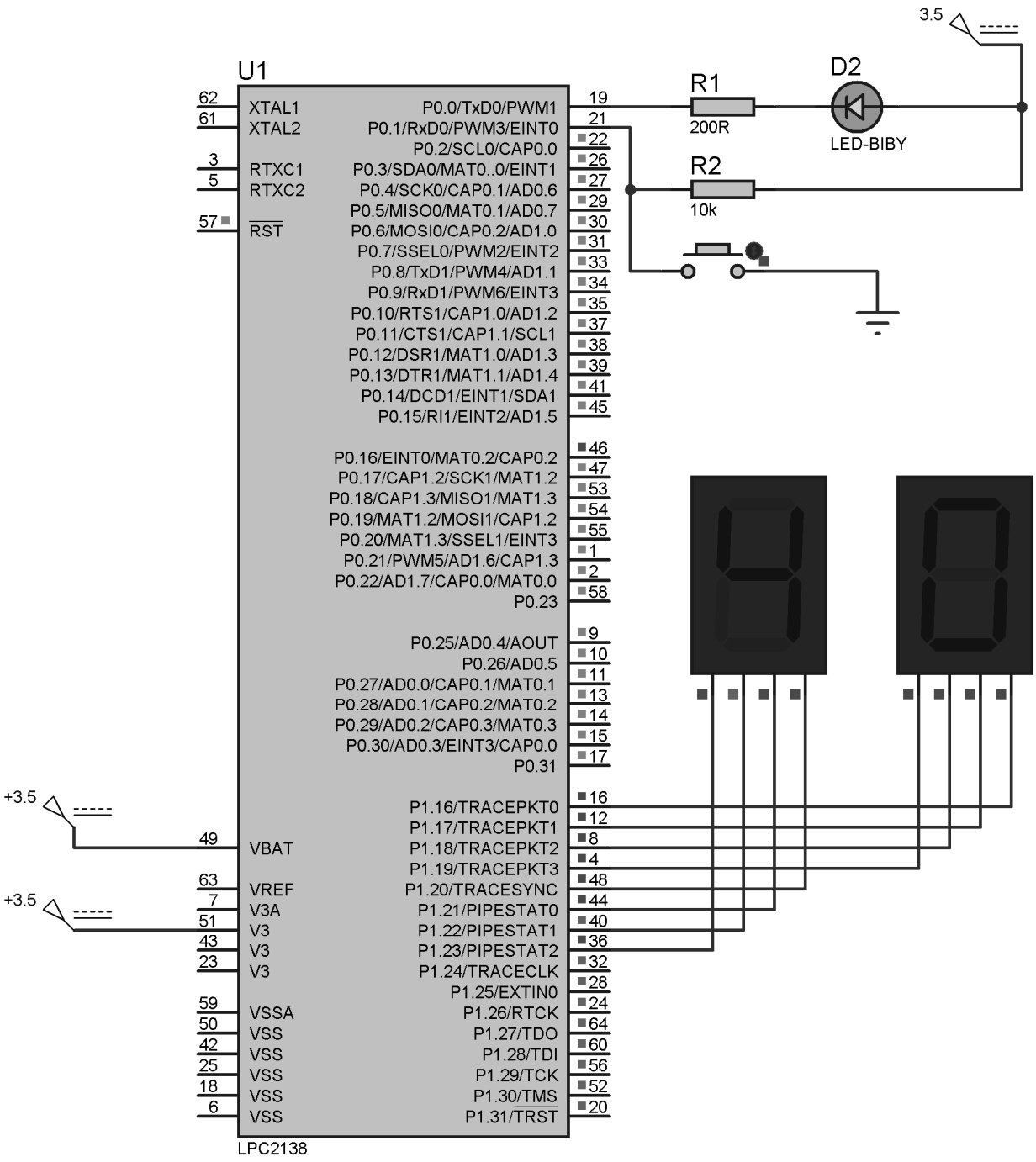


Рисунок 4.11 –Модель стенда, иллюстрирующая работу WDT в микроконтроллере LPC2138

При нажатии на кнопку на 7-сегментных индикаторах загорается номер варианта, и выполнение продолжается в бесконечном цикле. Спустя одну секунду модельного времени происходит сброс микроконтроллера. Если раскомментировать строки с неверной FEED последовательностью, сброс выполняется немедленно.

5 ПРЕРЫВАНИЯ

5.1 Прерывания от внешнего источника

Прерывания от внешнего источника относятся к блоку системных функций, но их программирование и использование требует изучения работы контроллера прерываний. Поэтому в разделе будут изложены принципы программирования внешних прерываний с использованием готовых примеров программирования контроллера прерываний, сущность которых будет раскрыта позднее.

Особенности прерываний от внешнего источника:

- Используется быстрое FIQ или обычное IRQ прерывание.
- Входы прерывания EINT0-EINT1 подключены к пинам P0,P1.
- Возможны прерывания по фронту, срезу или уровню внешнего источника
- При возврате из прерывания соответствующий бит регистра EXTINT должен быть очищен, иначе прерывание будет вызвано вновь.

Регистр флагов внешнего прерывания EXTINT, структура которого показана на рисунке 5.1, предназначен для определения источника прерывания и выхода из него.

Рисунок 5.1 – Регистр флагов внешнего прерывания EXTINT

При чтении флага определяется источник прерывания, например:

if (EXINT&1) IOSET0=1; // При прерывании EINT0 зажечь светодиод на P0.0.

При записи флаг единицы соответствующее прерывание сбрасывается:

EXINT = 1 ; // Сбросить прерывание от EINT0.

Формат остальных регистров внешних прерываний аналогичен представленному на рисунке 5.1, регистры отличаются только функциональностью:

INTWAKE – регистр разрешения пробуждения микроконтроллера в режимах Idle mode и Power-down mode. Если соответствующие биты, уста-

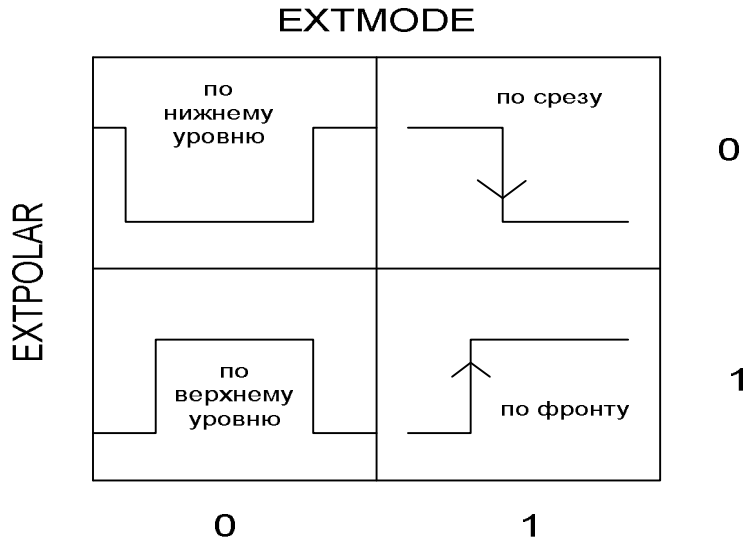
новлены, то при возникновении события на соответствующем пине произойдет пробуждение микроконтроллера.

EXTMODE – регистр режима прерывания. Лог. 0 в соответствующем разряде разрешает для него прерывание по уровню (т.е. прерывание будет происходить все время пока на пине присутствует активный уровень), лог.1 разрешает прерывание по фронту или срезу сигнала на соответствующем пине.

EXTPOLAR – Регистр полярности внешнего сигнала. Лог.0 в соответствующем разряде разрешает прерывание по срезу или по низкому уровню, в зависимости от соответствующего бита в регистре EXTMODE; Лог. 1 разрешает прерывание по фронту или высокому уровню в зависимости от соответствующего бита в регистре EXTMODE.

Необходимо помнить, что по умолчанию функции внешних прерываний на пинах не активны, для их активации соответствующий код должен быть записан в регистры PINSEL0-PINSEL2. В таблице 5.1 показан принцип выбора функций прерывания.

Таблица 5.1 – Таблица выбора функций



$EXTMODE = (1 << 0)$
$EXTPOLAR = (1 << 0)$, судя из таблицы 5.1 событие произойдет по фронту для прерывания EINT0.

На рисунке 5.2 показана модель стенда, используемая для примера.

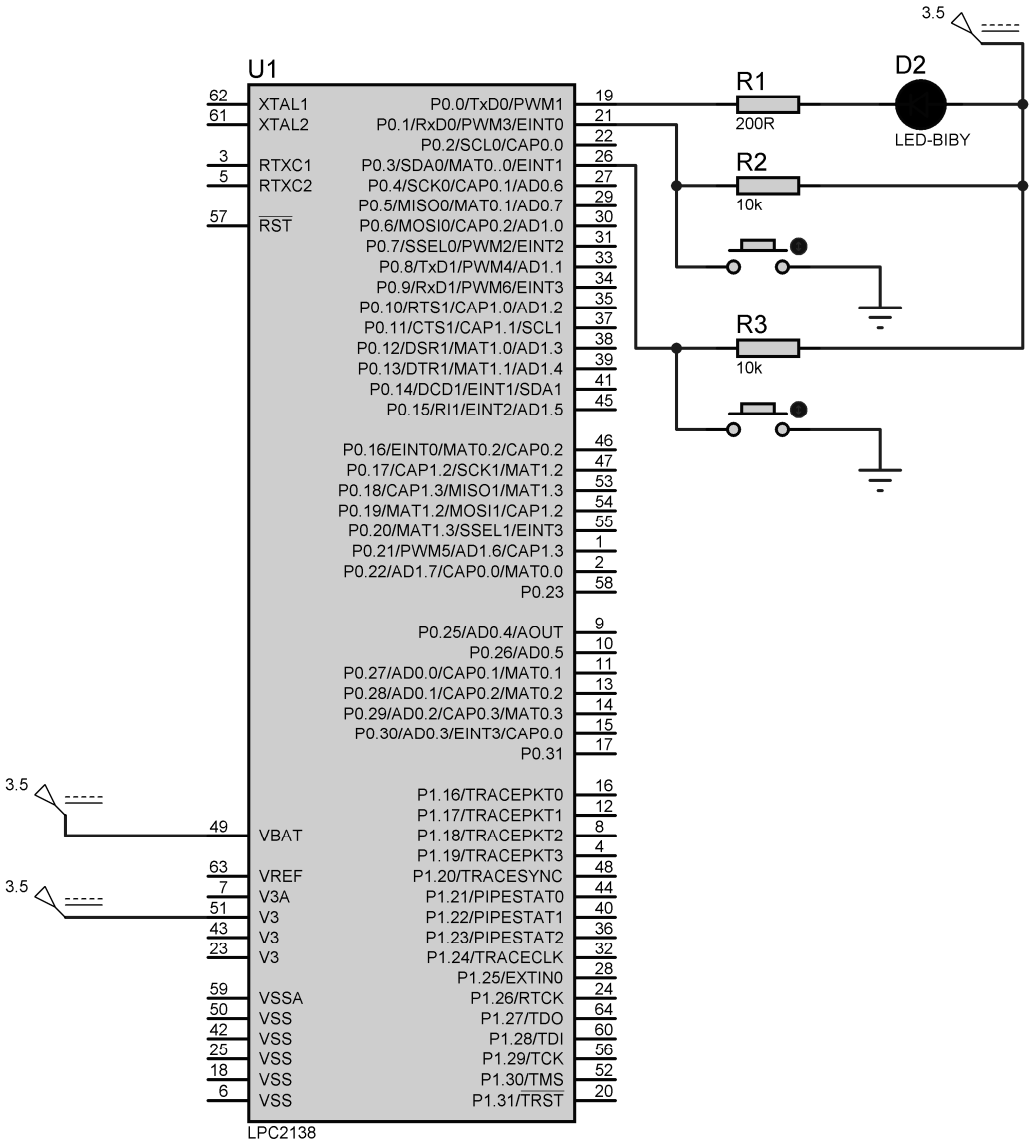


Рисунок 5.2 – Модель стенда, иллюстрирующая работу внешних прерываний EINT на микроконтроллере LPC2138

Текст программы:

// Демонстрационный пример работы FIQ прерывания от внешнего источника

```
#include <LPC213x.h>
void main(void) {
PINSEL0=(1<<1)+(1<<3);          // Установить пин P0.1 в  EINT0     +
                                P0.3 в  EINT1
                                //D0, D1 - Режимы пина 0: 00 - GPIO,
                                01 - TxD (UART0), //10 - PWM1, 11 - не
                                используется (принято GPIO)
                                //D3, D2 - Режимы пина 1: 00 - GPIO,
                                01 - RxD (UART0), //10 - PWM3, 11 -
                                EINT0 (принято EINT0)
```

```
                        //D5, D4 - Режимы пина 2: 00 - GPIO, 01 - SCL0(I2C0), //10 - CAP0.1, 11 - не используется (принято GPIO)
                        //D7, D6 - Режимы пина 3: 00 - GPIO, 01 - SDA0(I2C0), //10 - MAT0.0, 11 - EINT1 (принято EINT1)
                        // Остальные пины удобно смотреть на схеме.
IODIR0 =(1<<1);         // Установить пин P0.0  выходом
IOSET0 =(1<<1);         // Потушить светодиод P0.0
EXTMODE=(1<<0) + (1<<1); // Выбрать для EINT0 +  EINT1 прерывание по гребню
                        // D0 = 0 Прерывание EINT0 по уровню водного сигнала
                        // D0 = 1 Прерывание EINT0 по фронту или срезу сиг//нала, в зависимости от EXTPOLAR
                        // D1 аналогично для EINT1, D2 аналогично для EINT2, //D3 аналогично для EINT3
                        // Сменить комментарии, если нужно выбрать другой //режим
EXTPOLAR=(1<<0);        // Выбрать для EINT0, EINT1 прерывание по срезу
EXTPOLAR=(1<<0)+(1<<1); // Выбрать для EINT0, EINT1 прерывание по фронту
                        // D0 = 0 Прерывание EINT0 по срезу или низкому уровню, в зависимости от EXTMODE
                        // D0 = 1 Прерывание EINT0 по фронту или высокому уровню, в зависимости от EXTMODE
                        // D1 аналогично для EINT1, D2 аналогично для EINT2, D3 аналогично для EINT3
VICIntSelect=(1<<14)+(1<<15);   // Включить FIQ для EINT0, EINT1 прерываний
                        // Регистр выбора типа прерывания, если бит = 1 - FIQ, //если бит =0 - IRQ.
VICIntEnable=(1<<14)+(1<<15);   // Разрешить  прерывания EINT0 + EINT1
                        // Регистр разрешения прерывания, если записать бит = 1 - разрешено,
                        // если бит =0 соответствующее прерывание не изменяется
VICIntEnClr=(1<<0);     // Запретить  прерывания, если необходимо
                        // Регистр запрещения прерываний, если записать бит=1 запрещено,
```

```
                                    // если бит = 0 - не изменяется (т.е. остается, как было)
                                    // Назначение бит регистров:
                                    //D0 - WDT,            D1 - Не исп.
                                    D2 - ARMCore1   D3 - ARMCore2
                                    //D4 - Timer0          D5 - Timer1
                                    D6 - UART0           D7 - UART1
                                    //D8 - PWM0            D9 - I2C0
                                    D10- SPI0            D11- SPI1
                                    //D12- PLL                  D13- RTC
                                    D14- EINT0           D15- EINT1
                                    //D16- EINT2           D17- EINT3
                                    D18- AD0             D19- I2C1
                                    //D20- BOD             D21- I2C1
                                    D22- AD1             D23- Не исп.

while (1) {};                       // Бесконечный цикл
}
void FIQ_Handler(void) __fiq        //Быстрое прерывание
{
 if (EXTINT & (1<<1))
 IOCLR0=(1<<4);                     // Зажечь светодиод
                                    // EXTINT - регистр внешних прерываний , биты = 1 соответствуют произошедшему прерыванию
                                    // D0 - EINT0, D1 - EINT1, D2 - EINT2, D3 - EINT3
if (EXTINT & (1<<0)+(1<<1))
IOSET0 = (1<<0);                    // Потушить светодиод
EXTINT=(1<<2)+(1<<0);               // Сбросить флаги внешнего прерывания //EXINT
                                    // При записи 1 в бит EXINT соответствующее внешнее прерывание сбрасывается
                                    // Если не сбросить - вызов прерывания повториться!
}
```

На рисунке 5.3 показана модель стенда, используемая для примера.

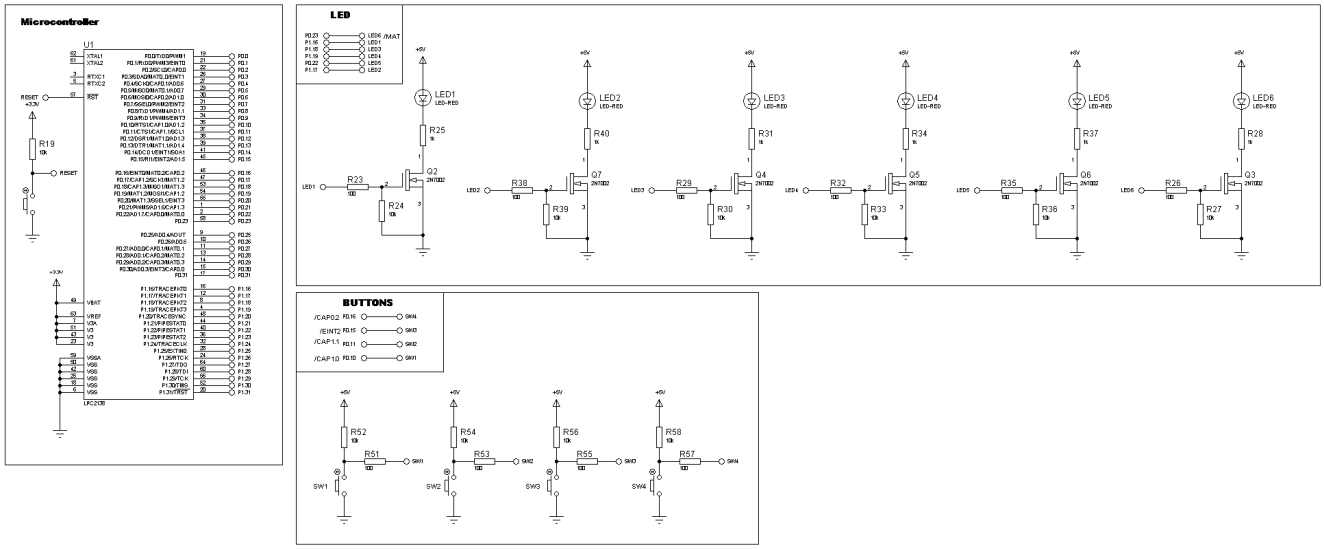


Рисунок 5.3 – Модель стенда для иллюстрации работы внешних прерываний FIQ на микроконтроллере LPC2138

Текст программы на языке C++ для FIQ прерывания:

```
#include <LPC213x.h>
void main(void) {
PINSEL0=(1<<1)+(1<<3);      // Установить пин P0.1 в EINT0 + P0.3 в
EINT1
                            //D0, D1 - Режимы пина 0: 00 - GPIO,
                            01 - TxD (UART0), 10 - //PWM1, 11 - не используется (принято GPIO)
                            //D3, D2 - Режимы пина 1: 00 - GPIO,
                            01 - RxD (UART0), 10 - //PWM3, 11 - EINT0 (принято EINT0)
                            //D5, D4 - Режимы пина 2: 00 - GPIO,
                            01 - SCL0(I2C0), 10 - //CAP0.1, 11 - не используется (принято GPIO)
                            //D7, D6 - Режимы пина 3: 00 - GPIO,
                            01 - SDA0 (I2C0), 10 - //MAT0.0, 11 - EINT1 (принято EINT1)
                            // Остальные пины удобно смотреть на схеме.
IODIR0 =(1<<0);             // Установить пин P0.0 выходом
IOSET0 =(1<<0);             // Потушить светодиод P0.0
EXTMODE=(1<<0) + (1<<1); // Выбрать для EINT0 + EINT1 прерывание
по гребню
                            // D0 = 0 Прерывание EINT0 по уровню водного сигнала
                            // D0 = 1 Прерывание EINT0 по фронту или срезу сигнала, в зависимости от EXTPOLAR
```

```
                                    // D1 аналогично для EINT1, D2 аналогично для EINT2, D3 //аналогично для EINT3
                                    // Сменить комментарии, если нужно выбрать другой режим
EXTPOLAR=(1<<0);                    // Выбрать для EINT0, EINT1 прерывание по срезу
EXTPOLAR=(1<<0)+(1<<1);             // Выбрать для EINT0, EINT1 прерывание по //фронту
                                    // D0 = 0 Прерывание EINT0 по срезу или низкому уровню, в зависимости от EXTMODE
                                    // D0 = 1 Прерывание EINT0 по фронту или высокому уровню, в зависимости от EXTMODE
                                    // D1 аналогично для EINT1, D2 аналогично для EINT2, D3 аналогично для EINT3
VICIntSelect=(1<<14)+(1<<15);       // Включить FIQ для EINT0, EINT1 прерываний

                                    // Регистр выбора типа прерывания, если бит = 1 - FIQ, если  бит =0 - IRQ.
VICIntEnable=(1<<14)+(1<<15);       // Разрешить  прерывания EINT0 + EINT1

                                    // Регистр разрешения прерывания, если записать бит = 1 - разрешено,
                                    // если бит =0 соответствующее прерывание не изменяется
VICIntEnClr=(1<<0);                 // Запретить  прерывания, если необходимо

                                    // Регистр запрещения прерываний, если записать бит=1 - запрещено,
                                    // если бит = 0 - не изменяется (т.е. остается, как было)
                                    // Назначение бит регистров:
                                    //D0 - WDT,          D1 - Не исп.
                                    //D2 - ARMCore1      D3 - ARMCore2
                                    //D4 - Timer0        D5 - Timer1
                                    //D6 - UART0         D7- ART1
                                    //D8 - PWM0          D9 - I2C0
                                    //D10- SPI0          D11- SPI1
                                    //D12- PLL           D13-RTC
                                    //D14- EINT0         D15- EINT1
                                    //D16- EINT2         D17- EINT3
```

```
                                  //D18- AD0          D19- I2C1
                                  //D20- BOD          D21- I2C1
                                  //D22- AD1          D23- Не исп.

while (1) {};                     // Бесконечный цикл
}
void FIQ_Handler(void) __fiq      //Быстрое прерывание
{
 if  (EXTINT & (1<<0))
IOCLR0 = (1<<4);                  // Зажечь светодиод
                                  // EXTINT - регистр внешних прерываний, биты = 1 соответствуют произошедшему прерыванию
                                  // D0 - EINT0, D1 - EINT1, D2 - EINT2, D3 - EINT3
if  (EXTINT & (1<<1))
IOSET0 = (1<<4);                  // Потушить светодиод
EXTINT=(1<<0)+(1<<1);             // Сбросить флаги внешнего прерывания EXINT
                                  // При записи 1 в бит EXINT соответствующее внешнее прерывание сбрасывается
                                  // Если не сбросить - вызов прерывания повториться!
}
```

Текст программы на языке C++ для невекторного IRQ прерывания:

```
#include <LPC213x.h>
void IRQ_Handler(void) __irq;
void main(void) {
                                  // Инициализация портов ввода-вывода, светодиод в начальный момент погашен

PINSEL0 = (1<<(1*2))+(+(1<<(3*2));   // Установить пин P0.1 в  EINT0 + P0.3 в  //EINT1
IODIR0 =(1<<0)+(1<<1);            // Установить пин P0.0  выходом
IOSET0 =(1<<0)+(1<<1);            // Потушить светодиод P0.0
                                  // Инициализация блока внешнего прерывания
EXTMODE=(1<<0 )+ (1<<1);          // Выбрать для EINT0 +  EINT1 прерывание по гребню //(EXTMODE=3)
```

```
EXTPOLAR=(1<<0) +(1<<1);          // Выбрать для EINT0, EINT1 прерывание по фронту (EXTPOLAR=3)
                                  // Инициализация контроллера прерывания
VICDefVectAddr = (unsigned)IRQ_Handler;   // Задать адрес невекторного прерывания
VICIntEnable=(1<<14)+(1<<15);     // Разрешить прерывания для EINT0 и EINT1 (см. //таб5.3)
while (1) {};                     // Бесконечный цикл, может быть заменен на другие, необходимые действия.
}
void IRQ_Handler(void) __irq
{
if (EXTINT & (1<<0))   IOCLR0=(1<<0)+(1<<1);
                                  //Нажата кнопка 1, зажечь светодиод
if (EXTINT & (1<<1))   IOSET0=(1<<0)+(1<<1);
                                  //Нажата кнопка 2,потушить светодиод
 EXTINT=(1<<0)+(1<<1);            //Сбросить флаги внешнего прерывания
        VICVectAddr =(1<<0)+(1<<1);   //Сбросить контроллер прерываний
}
```

5.2 Программирование контроллера прерываний

Кроме внешних прерываний существуют и другие источники. Все периферийные устройства могут вызывать прерывания и имеют собственный флаг прерывания. В таблице 5.1 приведены источники и события прерываний.

Таблица 5.1 – Источники и события прерываний

Блок	Флаги	Источник
WDT	Watchdog Interrupt (WDINT)	Сторожевой таймер
-	Reserved for Software Interrupts only	Программные прерывания
ARM Core	Embedded ICE, DbgCommRx	Отладка ядра прием
ARM Core	Embedded ICE, DbgCommTX	Отладка ядра передача
TIMER 0	Match 0 - 3 (MR0, MR1, MR2, MR3) Capture 0 - 3 (CR0, CR1, CR2, CR3)	Таймер 0. Прерывание при совпадении Таймер 0. Прерывание при захвате
TIMER 1	Match 0 - 3 (MR0, MR1, MR2, MR3) Capture 0 - 3 (CR0, CR1, CR2, CR3)	Таймер 1. Прерывание при совпадении Таймер 1. Прерывание при захвате
UART0	Rx Line Status (RLS) Transmit Holding Register Empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI)	Изменение статуса линии приемника UART0 Опустошение регистра передатчика UART0 Доступны принятые данные UART0 Предельное время приема последовательности
UART1	Rx Line Status (RLS) Transmit Holding Register Empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI) Modem Status Interrupt (MSI)U1	Изменение статуса линии приемника UART1 Опустошение регистра передатчика UART1 Доступны принятые данные UART1 Предельное время приема последовательности Изменение статуса модема
PWM0	Match 0 - 6 (MR0, MR1, MR2, MR3, MR4, MR5)	Совпадение в каналах ШИМ
l^2C0	SI (state change)	Изменение состояния в l^2C0

Продолжение таблицы 5.1

SPI0	SPI Interrupt Flag (SPIF) Mode Fault (MODF)	Прерывание в SPI Ошибка приема
SPI1 (SSP)	TX FIFO at least half empty (TXRIS) Rx FIFO at least half full (RXRIS) Receive Timeout condition (RTRIS) Receive overrun (RORRIS)	Опустошение FIFOпередатчика Заполнение FIFO приемника Предельное время приема последовательности Ошибка кадра приемника
PLL	PLL Lock (PLOCK)	Захват ФАПЧ
RTC	Counter Increment (RTCCIF) Alarm (RTCALF)	Приращение счетчика таймера реального времени Звонок таймера реального времени
System Control	External Interrupt 0 (EINT0) External Interrupt 1 (EINT1) External Interrupt 2 (EINT2) External Interrupt 3 (EINT3)	Прерывание от внешнего источника EINT0 Прерывание от внешнего источника EINT1 Прерывание от внешнего источника EINT2 Прерывание от внешнего источника EINT3
ADC0	A/D Converter 0 end of conversion	Окончание преобразования в АЦП0
I2C1	SI (state change)	Изменение состояния в I^2C1
BOD	Brown Out detect	Недопустимое падение напряжения
ADC1	A/D Converter 1 end of conversion	Окончание преобразования в АЦП1

Контроллер прерываний управляется при помощи регистров, наименование, описание, режим доступа из программы и значение при сбросе показано в таблице 5.2. Для режима доступа приняты сокращения: RO – только чтение, WO – только запись, R/W – разрешена как чтение, так и запись.

Таблица 5.2 – Регистры контроллера прерываний

Наименование регистра	Описание	Доступ	Знач. при сбросе
VICIRQStatus	Регистр статуса IRQ прерывания. Биты соответствуют источнику IRQ прерывания.	RO	0
VICFlQStatus	Регистр статуса FIQ прерывания. Биты соответствуют источнику FIQ прерывания.	RO	0
VICRawlntr	Регистр статуса любого прерывания. Биты соответствуют источнику прерывания.	RO	0
VICIntSelect	Регистр переключения режима прерывания. Биты соответствуют режиму. 1 - FIQ , 0 - IRQ прерывание.	R/W	0
VICIntEnable	Регистр разрешения прерывания. Запись 1 - разрешает прерывание, 0 – не изменяет разрешение.	R/W	0
VICIntEnClr	Регистр запрещения прерывания. Запись 1 - запрещает прерывание, 0 – не изменяет разрешение .	WO	0
VICSoftlnt	Регистр программных прерываний. Регистр позволяет только разрешить до 32 прерываний от различных периферийных устройств.	R/W	0
VICSoft-lntClear	Регистр очистки программных прерываний. Регистр позволяет только запретить программные прерывания.	WO	0
VICProtection	Регистр разрешения доступа.	R/W	0
VICVectAddr	Регистр вектора адресов. Во время IRQ подпрограмма обработки прерывания может использовать этот адрес в качестве адреса перехода.	R/W	0
VICDefVec-tAddr	Адрес вектора прерывания по умолчанию. Этот регистр содержит адрес процедуры обработки IRQ прерывания для режима не векторного IRQ прерывания.	R/W	0
VICVectAddr0 - VICVectAddr15	Регистры векторов адреса IRQ прерывания. Содержат адреса процедур обработки для режима векторного IRQ прерывания (ISRs).	R/W	0
VICVectCntl0-VICVec-tCntl15	Регистры управления слотами векторных прерываний.	R/W	0

Каждый источник прерывания имеет свою позицию в структуре регистров управления прерываниями, одинаковую для всех регистров, как показано в таблице 5.3.

Программирование регистров контроллера прерываний соответствует записи значений бит, соответствующих периферийному устройству, в позицию выбираемую по таблице 5.3. Например, для разрешения прерываний EINT0 и EINT2 необходимо выполнить строку программы на языке СИ:

VICIntEnable = (1<<14) + (1<<16); // Разрешить прерывания EINT0 и EINT2

Остальные биты регистра разрешения прерывания не изменяться. Для запрещения прерывания необходимо выполнить аналогичную операцию с регистром VICIntEnClr:

VICIntClr = (1<<14) + (1<<16); // Запретить прерывания EINT0 и EINT2

В противоположность регистрам разрешения и запрещения, регистр переключения режима прерывания VICIntSelect имеет значение, которое было записано в него с учетом, как единиц, так и нулей. При этом единицы означают периферийные устройства, для обслуживания которых используются быстрые прерывания. Обычно, в качестве быстрых прерываний рекомендуется использовать не более одного источника.

Регистры статуса позволяют определить периферийное устройство, вызвавшее быстрое (FIQ) или обычное (IRQ) прерывание.

Таблица 5.3 – Значение бит регистров и номера прерываний

Бит/номер	Наименование	Описание
0	WDT	Сторожевой таймер
1	-	Не используется
2	ARMCore0	Отладка ядра прием
3	ARMCore1	Отладка ядра передача
4	TIMER0	События в таймере 0
5	TIMER1	События в таймере 1
6	UART0	События в UART0
7	UART1	События в UART1
8	PWM0	Совпадение в каналах ШИМ
9	I^2C0	Изменение состояния в I^2C0
10	SPI0	События SPI0
11	SPI1/SSP	События SPI1
12	PLL	Захват ФАПЧ
13	RTC	Таймер реального времени
14	EINT0	Прерывание от внешнего источника EINT0
15	EINT1	Прерывание от внешнего источника EINT1
16	EINT2	Прерывание от внешнего источника EINT2
17	EINT3	Прерывание от внешнего источника EINT3

Продолжение таблицы 5.3

18	ADO	Окончание преобразования в АЦП0
19	I^2C1	Изменение состояния в I^2C1
20	BOD	Недопустимое падение напряжения
21	AD1	Окончание преобразования в АЦП1
22-31	-	Не используется

5.3 Программирование быстрых (FIQ) прерываний

Особенности быстрого (FIQ) прерывания:
– Режим работы – FIQ (сохранение R7-R14).
– Адрес вектора прерывания 0x0000001C.
– Приоритет прерывания – 3.
– Рекомендуется только один источник.

Благодаря автоматическому сохранению регистров R7-R14 и более высокому приоритету по сравнению с IRQ прерываниями время выполнения быстрого прерывания в несколько раз меньше чем IRQ. Рекомендация применения одного источника связана с необходимостью сокращения времени обработки, которое в случае нескольких источников увеличивается на величину, необходимую для определения источника и обработки переходов к соответствующим ветвям подпрограммы обработки прерываний.

Демонстрационный пример программирования быстрых прерываний приведен в разделе 5.1 (программирование прерываний от внешнего источника). В этом примере не соблюдена рекомендация использовать только один источник прерываний, т.к. требуемое время обработки сигналов от кнопок более 20мс.

Для реализации FIQ прерывания в стартовом модуле программы необходимо выполнить инициализацию контроллера прерываний, для чего программируются два регистра:

$$VICIntSelect = (1 << 14) + (1 << 15);$$

Команда включить FIQ для EINT0, EINT1 прерываний (см. таб.5.3), в этом регистре если будет бит установлен в единицу то для соответствующего прерывания установлен режим FIQ, если бит сброшен в ноль - режим IRQ.

$$VICIntEnable = (1 << 14) + (1 << 15);$$

Команда разрешает прерывания для EINT0 и EINT1, если бит установлен в единицу то соответствующее прерывание разрешается, если бит сброшен в ноль разрешение соответствующего прерывания не изменяется.

Процедура FIQ_Handler(void) имеет зарезервированное Keil □Vision для FIQ прерываний имя, описываемое в файле startup.s. Она содержит процедуру обработки FIQ прерывания. В конце этой процедуры необходимо выполнить сброс регистра флагов прерываний периферийного устройства, для чего в регистр флагов нужно записать логические единицы в соответствующих разрядах. Например, для EINT:

EXTINT=(1<<8); // Очистить все биты без разбора

Команда выполнит очистку регистра флагов прерываний от внешнего источника.

5.4 Программирование невекторного прерывания IRQ

Особенности невекторного прерывания IRQ:
– Режим работы – IRQ (сохранение R13-R14).
– Адрес вектора прерывания 0x00000018.
– Приоритет прерывания – 4.
– Источник прерывания определяется регистром статуса VICIRQStatus.
– Адрес процедуры обработки помещается в регистр VICDefVectAddr.
– При выходе из процедуры обработки регистр вектора VICVectAddr должен быть сброшен записью нулевого значения.

Регистры R7-R12 не сохраняются автоматически, как это происходит в режиме FIQ. Из-за этого программа выполняет их сохранение в программный стек, и время вызова IRQ процедуры увеличивается.

Приведем пример программной реализации невекторного прерывания IRQ. Для этого используем модель микроконтроллера, показанную на рисунке 5.2 и соответственно, в качестве внешнего устройства используем EINT:

```
#include <LPC213x.h>
void IRQ_Handler(void) __irq;
void main(void) {
                                        // Инициализация портов ввода-вывода,
                                        светодиод в начальный момент пога-
                                        шен

PINSEL0=(1<<(1*2))+(1<<(3*2));   // Установить пин P0.1 в  EINT0     +
                                        P0.3 в  //EINT1
IODIR0 =(1<<(1*2))+(1<<(3*2));   // Установить пин P0.0  выходом
IOSET0 =(1<<(1*2))+(1<<(3*2));   // Потушить светодиод P0.0
```

```
                                    // Инициализация блока внешнего прерывания
EXTMODE=(1<<0) + (1<<1);            //Выбрать для EINT0 + EINT1 прерывание по //гребню (EXTMODE=3)
EXTPOLAR=(1<<0) +(1<<1);            // Выбрать для EINT0, EINT1 прерывание по //фронту (EXTPOLAR=3)
                                    // Инициализация контроллера прерывания
VICDefVectAddr = (unsigned)IRQ_Handler;     // Задать адрес невекторного прерывания
VICIntEnable=(1<<14)+(1<<15);       //Разрешить прерывания для EINT0 и EINT1 (см. таб. 5.3)
while (1) {};                       //Бесконечный цикл, может быть заменен на другие необходимые действия
}
void IRQ_Handler(void) __irq
{
if (EXTINT & (1<<0))   IOCLR0=(1<<(1*2))+(1<<(3*2));
                                                    // Нажата кнопка 1, зажечь светодиод
 if (EXTINT & (1<<1))   IOSET0=(1<<(1*2))+(1<<(3*2));
                                                    // Нажата кнопка 2, потушить светодиод
EXTINT=(1<<0)+(1<<1); //                            //Сбросить флаги внешнего прерывания
VICVectAddr = (1<<(1*2))+(1<<(3*2));                // Сбросить контроллер прерываний
}
```

Программирование EINT аналогично приведенному в разделе 5.1 примеру, отличие заключается в программировании регистров контроллера прерываний и оформлении подпрограммы обработки прерывания:

– По умолчанию регистр VICIntSelect, управляющий режимом IRQ или FIQ прерываний равен нулю, что соответствует режиму IRQ, следовательно, в примере не модифицируется и не рассматривается.

– Команда VICDefVectAddr = (unsigned)IRQ_Handler задает адрес невекторного прерывания, при этом выполняется присваивание регистру VICDefVectAddr константы, рассчитанной компилятором СИ на основе фактического адреса процедуры обработки прерывания IRQ_Handler.

– Для разрешения последующих вызовов процедуры обработки прерывания IRQ_Handler регистр адреса вектора VICVectAddr в конце процедуры обнуляется.

Недостатком невекторного прерывания является необходимость при обработке процедуры прерывания определять источник, что при большом их количестве увеличивает время обработки.

5.5 Программирование векторного прерывания IRQ

Проблема потери времени при анализе источника прерывания устраняется векторизацией переходов для каждого из задействованных источников прерываний. Особенности векторного прерывания IRQ:

– Режим работы – IRQ (сохранение R13-R14).

– Адрес вектора прерывания 0x00000018.

– Приоритет прерывания – 4.

– Каждый источник прерывания связан с собственной процедурой обработки.

– Контроллер предоставляет 15 слотов, управляемых регистрами VICVectCntl0-15.

– Слот 0 имеет наибольший приоритет, слот 15 – наименьший.

– Адрес процедуры обработки помещается в один из регистров VICVectAddr0-15.

– При выходе из процедуры обработки регистр вектора VICVectAddr должен быть сброшен записью нулевого значения.

Регистры управления слотами векторного прерывания VICVectCntl0-VICVectCntl15 имеют структуру, показанную на рисунке 5.2 и предназначена для связывания любого из 15 слотов с одним из прерываний от периферийных устройств. Номера периферийных устройств для этого регистра определяются по первой колонке таблицы 5.4 и соответствуют номеру бита периферийного устройства в регистрах контроллера прерываний. Что бы слот стал активным, также должен быть установлен и пятый бит регистра управления слотом векторного прерывания VICVectCntl.

<table>
<tr><td>D0</td><td colspan="2" rowspan="5">Номер вектора прерывания или программного прерывания присвоенный источнику прерывания. Следует использовать номер из таб 5.3, значение из диапазона 1-32.</td></tr>
<tr><td>D1</td></tr>
<tr><td>D2</td></tr>
<tr><td>D3</td></tr>
<tr><td>D4</td></tr>
<tr><td>D5</td><td>CR1I</td><td>Разрешение прерывания для слота</td></tr>
</table>

Рисунок 5.4 – Структура регистров управления слотом векторного прерывания

Регистры VICVectAddr0- VICVectAddr15 содержат адрес используемой для слота (и соответственно, для выбранного периферийного устройства) подпрограммы обработки прерывания.

Приведем пример программной реализации векторного прерывания IRQ. При этом также используем модель микроконтроллера, показанную на рисунке 5.2:

```
#include <LPC213x.h>
void IRQ_EINT0 (void) __irq;          // Предварительное описание процедуры обработки //прерывания EINT0
void IRQ_EINT1 (void) __irq;          // Предварительное описание процедуры обработки //прерывания EINT1

void main(void) {
                                      // Инициализация портов ввода-вывода
PINSEL0=(1<<(1*2))+(1<<(3*2));        // Установить пин P0.1 в EINT0 + P0.3 в //EINT1
IODIR0 =(1<<(1*2))+(1<<(3*2));        // Установить пин P0.0 выходом
IOSET0 =(1<<(1*2))+(1<<(3*2));        // Потушить светодиод P0.0
                                      // Инициализация внешнего прерывания
EXTPOLAR=(1<<0);                      //Выбрать для EINT0,EINT1 прерывание по срезу
                                      // Инициализация контроллера прерывания
VICVectCntl0 =(1<<(5+14));            // Разрешить Слот 0. и задать номер прерывания //для IENT0
VICVectCntl1 =(1<<(5+15));            // Разрешить Слот 1 и задать номер прерывания //для IENT1
                                      // биты D0-D4 - номер источника прерывания
                                      //00 - WDT, 01 - Не исп.
                                      02 - ARMCore1, 03 - ARMCore2
                                      //04 - Timer0      05 - Timer1
                                      //06 - UART0       07 - UART1
                                      //08 - PWM0        09 - I2C0
                                      //10 - SPI0        11 - SPI1
                                      //12 – PLL         13 - RTC
                                      //14 - EINT0       15 - EINT1
                                      //16 - EINT2       17 - EINT3
                                      //18 - AD0         19 - I2C1
                                      //20 – BOD         21- I2C1
                                      //22 - AD1         23 - Не исп.
                                      // бит D5-Разрешение прерывания для данного слота. Если бит=1 – слот активен, прерывание разрешено
VICIntSelect=(1<<0);                  // Не включить FIQ прерываний (не обязателен, т.к. уже равен нулю)
VICVectAddr0    =(unsigned)IRQ_EINT0;
                                      //Задать адрес прерывания от EINT0
VICVectAddr1    = (unsigned)IRQ_EINT1;
```

```
                                        //Задать адрес прерывания для EINT0
VICIntEnable = (1<<14)+(1<<15);         //Разрешить прерывания для EINT0 //и EINT1 (см. таб5.3)

while (1) {}                            // Бесконечный цикл, может быть заменен операторами

}
void IRQ_EINT0(void) __arm __irq        // Вектор для прерывания от EINT0

{
IOCLR0=(1<<(1*2))+(1<<(3*2));           // Зажечь светодиод
EXTINT=(1<<0);                          // Сбросить флаг внешнего прерывания EINT0
        VICVectAddr=(1<<(1*2))+(1<<(3*2));  // Сбросить адрес контроллера прерываний

}
void IRQ_EINT1(void) __arm __irq        // Вектор для прерывания от
E                                       INT1
{
IOSET0=(1<<(1*2))+(1<<(3*2));           // Потушить светодиод
EXTINT=(1<<1);                          // Сбросить флаг внешнего прерывания EINT1

VICVectAddr =(1<<(1*2))+(1<<(3*2));     // Сбросить адрес контроллера прерываний

}
```

Недостатком приведенного примера является то, что вне зависимости от номера слота (и соответственно приоритета прерывания) вызванная ранее и не завершившая работу подпрограмма обработки вектора прерывания не позволит выполнить запуск других векторов IRQ прерывания. FIQ прерывание, имея более высокий приоритет и отдельный режим работы в состоянии выполнить вложенное прерывание.

5.6 Программирование вложенных IRQ прерываний

Особенности вложенных прерываний IRQ:
– Режим работы – IRQ (сохранение R13-R14).
– Адрес вектора прерывания 0x00000018.
– Приоритет прерывания – 4.
– Для разрешения вложенных прерываний необходимо сохранить регистры R13, R14 и разрешить прерывания IRQ, сбросив флаг I.

– После разрешения можно выполнять длительные действия в теле процедуры обработки прерывания.

– В конце процедуры обработки следует запретить прерывания IRQ, установив флаг I и восстановить значение регистров R13, R14.

– При выходе из процедуры обработки регистр вектора VICVectAddr должен быть сброшен записью нулевого значения.

Используя модель на рисунке 5.1, покажем пример программы, использующей вложенные IRQ прерывания:

```
#include <LPC213x.h>
void IRQ_EINT0(void) __irq;
void IRQ_EINT1(void) __irq;
                                        // Макросы для разрешения запрещения вложенных прерываний
                                        // Содержат ассемблерные вставки для сохранения восстановления регистров и переключения режима
#define IENABLE                         // Вход во вложенное прерывание

asm { MRS    LR, SPSR    };             // Копирование SPSR_irq в LR

asm { STMFD  SP!, {LR}};                // Сохранение SPSR_irq

asm { MSR    CPSR_c, #0x1F };           // Разрешение IRQ (Sys Mode)

asm { STMFD  SP!, {LR}};                // Сохранение
#define IDISABLE                        //Выход из вложенного прерывания

asm { LDMFD  SP!, {LR}}                 // Восстановление LR
asm { MSR    CPSR_c, #0x92 }            // Запрещение IRQ (IRQ Mode)
asm { LDMFD  SP!, {LR}}                 // Восстановление SPSR_irq в LR
asm { MSR    SPSR_cxsf, LR }            // Копирование LR в SPSR_irq
void main(void) {

                                        // Инициализация портов ввода-вывода
PINSEL0=(1<<(1*2))+(1<<(3*2));          // Установить пин P0.1 в EINT0 + P0.3 в //EINT1

IODIR0 =(1<<(1*2))+(1<<(3*2));          // Установить пин P0.0 выходом
IOSET0 =(1<<(1*2))+(1<<(3*2));          // Потушить светодиод P0.0

                                    // Инициализация внешнего прерывания
```

```
EXTMODE=(1<<0)+(1<<1);          // Выбрать для EINT0, EINT1 прерывание по //гребню
EXTPOLAR=(1<<0);                //Выбрать для EINT0,EINT1 прерывание по срезу

                                // Инициализация контроллера прерывания
VICIntSelect=(1<<0);            // Не включить FIQ прерываний
VICVectCntl0=(1<<(1*2))+(1<<(3*2));   // Разрешить Слот 0. + Задать номер прерывания //IENT0
VICVectCntl1=(1<<(1*2))+(1<<(3*3));;  // Разрешить Слот 1. + Задать номер прерывания //IENT1
VICVectAddr0 = (unsigned)IRQ_EINT0;   // Задать адрес прерывания от EINT0
VICVectAddr1 = (unsigned)IRQ_EINT1;   // Задать адрес прерывания от EINT0
VICIntEnable =(1<<(1*2))+(1<<(3*2))+ (1<<(1*2))+(1<<(3*3));
                                // Разрешить  прерывания EINT0 + EINT1
while (1) {};                   // Бесконечный цикл
}
void IRQ_EINT0(void)  __arm__irq
                                // Процедура с разрешением вложенности
{
EXTINT=(1<<1);                  // Сбросить флаг внешнего прерывания  EINT0
                                // Если не сбросить флаг, вложенное прерывание тут же вызовется повторно по этому же источнику!
IENABLE;                        //Разрешим вложенные прерывания
                                // Выполняем вызов макроса. Можно просто скопировать коды ассемблерной вставки.
IOCLR0=(1<<(1*2))+(1<<(3*2));   // Зажечь светодиод
while (!(IOPIN0&(1<<2))) {};    // Ожидать отпускания кнопки
                                //Цикл сделан для демонстрации вложенности!
                                // При этом возникает зацикливание программы //в прерывании до отпускания кнопки
                                // Что в реальной программе делать нельзя!
```

```
IDISABLE;                                  // Запретим вложенные прерывания
VICVectAddr =(1<<(1*2))+(1<<(3*2));        // Сбросить контроллер прерываний
}
void IRQ_EINT1(void) __arm __irq           // Процедура без разрешения вложенности
{
IOSET0=(1<<(1*2))+(1<<(3*2));              // Потушить светодиод
EXTINT=(1<<2);                             // Сбросить флаг внешнего прерывания EINT1
VICVectAddr=(1<<(1*2))+(1<<(3*2));         // Сбросить контроллер прерываний
}
```

Отличием программы, приведенной в примере, является использование ассемблерных вставок, понимание работы которых изложено в особенностях программирования вложенных IRQ прерываний. Ассемблерные вставки оформлены в виде макрокоманд, но их текст может быть и просто скопирован в соответствующее место программы.

Процедура обработки прерывания IRQ_EINT0(void) иллюстрирует пример медленно выполняющегося прерывания. Для этого внутрь процедуры помещен цикл ожидания отпускания кнопки, на нажатие которой настроено это прерывание. Если нажать первую кнопку и не отпускать ее, то светодиод загорается, показывая, что прерывание EINT0 выполнено. Использование макроса IENABLE заставляет микроконтроллер считать прерывание IRQ законченным, сохраняя при этом состояние регистров R13-R14. Если нажать вторую кнопку, то будет вызвана процедура IRQ_EINT1(void) и светодиод потухнет. Отпускание кнопки 1 вызовет процедуру IDISABLE, которая возвратит контроллер в состояние IRQ и восстановит содержимое регистров R13-R14. Прерывание завершиться в обычном порядке.

Применение цикла ожидания внутри прерывания обычно недопустимо, но данный пример носит учебный характер и иллюстрирует длительную процедуру обработки, размещенную внутри прерывания.

5.7 Использование программных прерываний (SWI)

Программные прерывания позволяют выполнять пользовательские функции в привилегированном режиме, что исключает их прерывание.

Для использования программных прерываний необходимо подключить в проект файл SWI_VEC.S, содержащий ассемблерный код, обеспечивающий выполнение программных прерываний.

Пример программы, использующий SWI прерывания:

```
#include <LPC213x.h>
                                        // Пользовательские функции
                                        // myfunc1 возвращает результат
                                        деления двух операндов
int myfunc1 (int i1, long i2) __swi (8)  {
  return (i1 / i2);
}
                                        // myfunc1 возвращает результат
                                        умножения на 16 для операнда
int myfunc2 (int i1) __swi (9)  {
        return (i1<<4);
}
int res;
void main (void)  {
  res = myfunc1 (10, 2);                // Вызвать SWI функции
  res += myfunc2 (res);
  while (1);                            // Зациклить программу
}
```

6 ПРОГРАММИРОВАНИЕ ТАЙМЕРОВ/СЧЕТЧИКОВ

6.1 Особенности таймеров-счетчиков

Особенности таймеров общего назначения T/C0 и T/C1 [1] Стр.188:
– В основе находиться 32-битный таймер/счетчик с 32-х битным прескалером.
– Выбирается режим счетчика импульсов или измерения времени.
– Четыре 32-битных модуля захвата, с возможностью генерации прерывания.
– Четыре 32-битных модуля сравнения с генерацией одного из четырех типов внешних сигналов.
Таймеры-счетчики функционируют в следующих режимах:
– Внутренний таймер для подсчета внутренних событий.
– Демодулятор длительности импульсов.
– Свободно запускаемый таймер.
– Счетчик внешних событий.
Регистры управления таймерами-счетчиками приведены в таблице 6.1

Таблица 6.1 – Регистры управления таймерами-счетчиками

Общее	Описание	Доступ	При сбросе	Имя регистра	Имя регистра
IR	Регистр прерывания. При чтении его биты соответствуют источнику прерывания, при записи 1 выполняется сброс прерывания.	R/W	0	T0IR	T1IR
TCR	Регистр управления таймером/счетчиком. С его помощью таймер/счетчик может быть сброшен или остановлен.	R/W	0	T0TCR	T1TCR
TC	32х битный счетный регистр таймера/счетчика.	R/W	0	T0TC	T1TC
PR	Регистр прескалера. Содержит число, при равенстве которому регистр содержимого прескалера PC сбрасывается.	R/W	0	T0PR	T1PR
PC	Регистр счетчика прескаллера. Увеличивается каждый такт периферийного синхросигнала. При совпадении с PR сбрасывается и генерирует (если разрешено) импульс приращения для TC.	R/W	0	T0PC	T1PC

Продолжение таблицы 6.1

MCR	Регистр управления сравнением. MCR используется для управления прерываниями от T/C и сброса счетчика TC при совпадении, если это разрешено.	R/W	0	T0MCR	T1MCR
MR0	Регистр совпадения 0. MR0 сравнивается со значением TC и если соотв. бит в MCR установлен, сбрасывает TC, останавливает счет или генерирует прерывание.	R/W	0	T0MR0	T1MR0
MR1	Регистр совпадения 1. Функции аналогичны MR0.	R/W	0	T0MR1	T1MR1
MR2	Регистр совпадения 2. Функции аналогичны MR0.	R/W	0	T0MR2	T1MR2
MR3	Регистр совпадения 3. Функции аналогичны MR0.	R/W	0	T0MR3	T1MR3
CCR	Регистр управления захватом. Регистр CCR управляет типом события захвата и разрешает генерацию соответствующего прерывания.	R/W	0	T0CCR	T1CCR
CR0	Регистр захвата 0. В регистр CR0 загружается значение из регистра TC в при возникновении события на входе CAP0.0 или CAP1.0 в зависимости от номера таймера/счетчика.	RO	0	T0CR0	T1CR0
CR1	Регистр захвата 1. аналогичен CR0, но событие контролирует на входах CAP0.1 или CAP1.1	RO	0	T0CR1	T1CR1
CR2	Регистр захвата 2. аналогичен CR0, но событие контролирует на входах CAP0.2 или CAP1.2	RO	0	T0CR2	T1CR2
CR3	Регистр захвата 3. аналогичен CR0, но событие контролирует на входах CAP0.3 или CAP1.3	RO	0	T0CR2	T1CR2
EMR	Регистр внешнего совпадения. Регистр EMR управляет событиями на пинах MAT0.0-3 или MAT1.0-3 соответственно таймеру/счетчику.	R/W	0	T0EMR	T1EMR
CTCR	Регистр управления источником счетных импульсов. Регистр CTCR переключает между режимами таймера и счетчика и в режиме счетчика выбирает источник и тип сигнала.	R/W	0	T0CTCR	T1CTCR

6.2 Программирование входной цепи таймеров-счетчиков

Регистры управления T/C T0TCR, T1TCR разрешают работу или сбрасывают соответствующий таймер/счетчик. Его структура показана на рисунке 6.1. Для разрешения работы бит 0 должен быть установлен, установка бита 1 приводит к сбросу счетных регистров таймера/счетчика. Последующий счет возможен только после записи в бит 1 логического нуля:

```
T0TCR=0; // Таймер/счетчик 0 остановлен
T0TCR=2; // Очистить T0TC и T0PC
T0TCR=1; // Запуск таймера
```

D0	Enable	Разрешение работы TC
D1	Reset	Сброс таймера и прескалера

Рисунок 6.1 – Структура регистров управления T/C T0TCR, T1TCR

Регистры содержимого таймеров/счетчиков TC0 и TC1 используются в режиме счетчика для суммирования числа событий на внешнем пине CAP микроконтроллера, а в режиме таймера – для суммирования импульсов периферийного синхросигнала PCLK, т.е. для измерения временных интервалов.

Регистры управления источником счетных импульсов CTCR0 и CTCR1 определяют режим работы – таймер или счетчик в зависимости от выбранного источника счетных импульсов. На рисунке 6.2 показана структура регистров управления источником счетных импульсов.

D0	mode0	D1,D0 - поле режима TC 0 0 (0) - таймер по гребню PCLK 0 1 (1) - счетчик по фронту CAP 1 0 (2) - счетчик по срезу CAP 1 1 (3) - счетчик по обоим CAP
D1	mode1	
D2	inpsel0	D3,D2 - поле CAP источника 0 0 (0) - CAP0.0 , CAP1.0 0 1 (1) - CAP0.1 , CAP1.1 1 0 (2) - CAP0.2 , CAP1.2 1 1 (3) - CAP0.3 , CAP1.3
D3	inpsel1	

Рисунок 6.2 – Структура регистров управления источником счетных импульсов CTCR0 и CTCR1

Первые два бита (D0-D1) определяют режим работы – таймер или счетчик, последующие биты (D2-D3) – источник счетных импульсов для режима счетчика. В качестве источника счетных импульсов используют один из пинов микроконтроллера, сконфигурированных в режим CAP0.0 - CAP0.1 (для T/C0). Например:

$CTCR0 = 0$ // Установить режим таймера для T/C0.

$CTCR0 = ((1+2) << 2)$ // Установить режим счетчика по фронту от CAP0.2 для T/C0.

В режиме таймера T/C выполняет суммирование импульсов периферийного синхросигнала PCLK. При этом возможно выполнить предварительное деление PCLK при помощи прескалера. Прескалер представляет собой счетчик T0PC или T1PC, в зависимости от используемого T/C, суммирующий импульсы PCLK. Значение регистра сравнивается с регистром прескалера T0PR (для T/C0) и в случае равенства выполняется сброс счетчика прескалера T0PC, а на селектор входов подается импульс приращения для счетного регистра T0TC.

Принцип работы входной цепи таймера счетчика показан на рисунке 6.3.

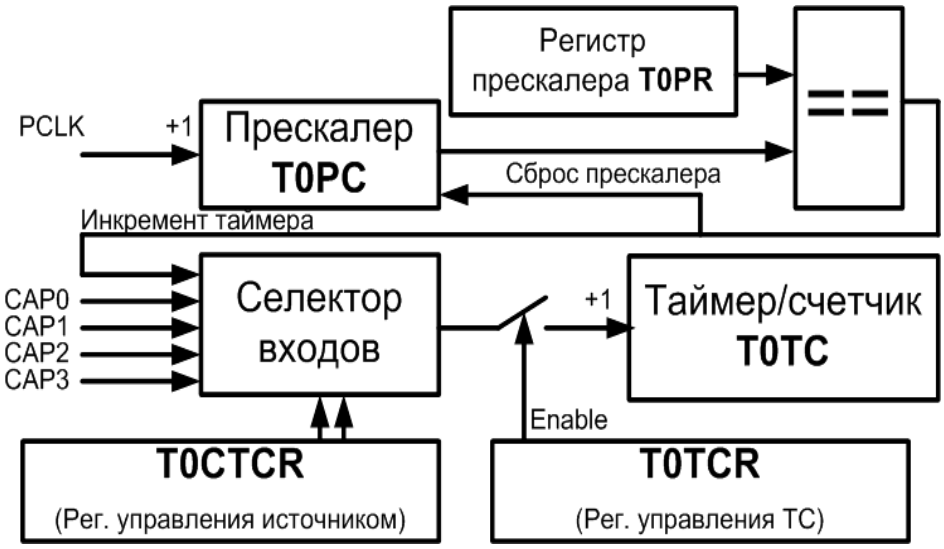


Рисунок 6.3 – Структура цепи подачи счетных импульсов на T0TC

Рассмотрим работу T/C0 в качестве счетчика. При нажатии на кнопку (модель показана на рисунке 6.4),прямоугольные импульсы поступают на вход CAP0.0, сконфигурированный на пине P0.2. Суммированное число нажатий отображается на светодиодной матрице, младший разряд счетного регистра выводиться на светодиод. Программа обработки:

```
#include <LPC213x.h>
void main(void) {
PINSEL0=(1<<3)+(1<<1);          // Установить пин P0.6 в  CAP0
IODIR0 =(1<<1);                 // Установить пин P0.0  выходом
```

```
IODIR1 =(1<<8)&(1<<4);          // Установить пин P1.16-19  выходом
IOSET0 =(1<<1);                 // Потушить светодиод P0.0
T0PR=1;                         // Задать предварительный делитель
                                счетчика T0
T0CTCR=(1<<1);                  // Выбрать для TC0 источник счетных
                                импульсов CAP0.0 по срезу
                                // Регистр управления источником
                                счетных импульсов таймера/счетчика
                                T0
                                // D1=0  D0=0 таймер по импульсу
                                PCLK  D1=0 D0=1  счетчик по фронту
                                CAP
                                // D1=1  D0=0 счетчик по срезу
                                CAP  D1=1 D0=1  счетчик по фронту
                                и срезу CAP
                                // D3 D2 - определяют источник CAP,
                                для таймера T0 CAP0.x,
                                // для таймера T1 CAP1.x, x=0..3
                                // D3=0 D2=0 CAP0.0  D3=0 D2=1
                                CAP0.1  D3=1 D2=0 CAP0.2 D3=1
                                D2=1 CAP0.3

T0TCR=2;                        // Сбрасывает T0
T0TCR=1;                        // Разрешаем работу
                                // Регистр управления тайме-
                                ром/счетчиком T0
                                // D0 = 1 Разрешить работу TC  D1 =
                                1  Сбросить T0 //и прескаллер пока
                                D1=1.

while (1) {

IOSET0=T0TC&(1<<1);             // Вывод младшего бита T/C на свето-
диод
IOCLR0=(T0TC&(1<<1))^(1<<2);
IOSET1=(T0TC&(1<<4))<<16));     // Вывод младшей тетрады T/C на ин-
дикатор
IOCLR1=(T0TC&(1<<4))<<16));
};                              // Бесконечный цикл
}
```

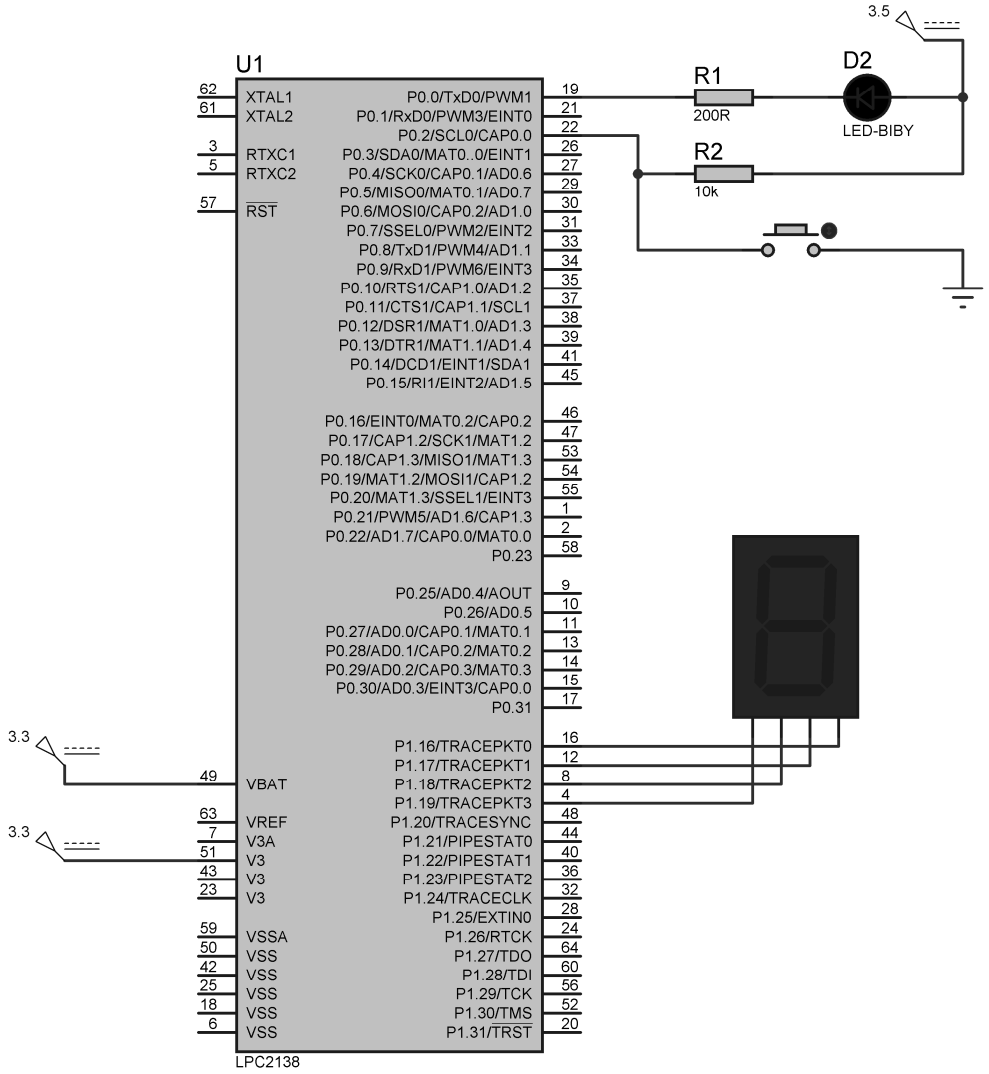


Рисунок 6.4 – Модель стенда, иллюстрирующая работу счетчика внешних импульсов

Рассмотрим работу T/C0 в качестве таймера. Для этого используем модель, показанную на рисунке 6.4, только без использования внешнего входа счетных импульсов CAP0.0. Таймер программируется на подсчет импульсов периферийного синхросигнала с использованием предварительного делителя таким образом, что бы обеспечить выдержку интервала 1 секунда. Вспомогательная переменная i обеспечивает подсчет количества интервалов с последующим выводом на светодиодную матрицу:

```
#include <LPC213x.h>
void main(void) {
int port0;                          // Вспомогательная переменная
int I;                              // Счетчик импульсов таймера
PINSEL0=(1<<3);                     // Установить пин P0.6 в  CAP0
IODIR0 =(1<<0);                     // Установить пин P0.0  выходом
IODIR1 =(1<<4)+(1<<2);              // Установить пин P1.16-19  выходом
```

```
IOSET0 =(1<<0);              // Потушить светодиод P0.0
T1CTCR=(1<<0);               // Выбрать для TC1 режим таймера
                             // Регистр управления источником счетных импульсов таймера/счетчика T0
                             // D1=0 D0=0 таймер по импульсу PCLK D1=0 D0=1 счетчик по фронту CAP
                             // D1=1 D0=0 счетчик по срезу CAP D1=1 D0=1 счетчик по фронту и срезу CAP
                             // D3 D2 - определяют источник CAP, для таймера T1 CAP1.x, x=0..3
                             // D3=0 D2=0 CAP1.0 D3=0 D2=1 CAP1.1 D3=1 D2=0 CAP1.2 D3=1 D2=1 //CAP1.3
                             // Настройка таймера/счетчика на требуемый интервал:
                             // T1PR * T1TC = 1c * 15000000 Гц
                             // 250 * 60000 = 15000000 - распределение между прескалером и счетчиком произвольно

T1PR=250;                    // Предварительный делитель таймера 1
T1TCR=1;                     // Разрешаем работу TC1
while (1) {
 if (T1TC>60000)             // Проверка содержимого таймера на окончание периода
   {
T1TC=0;                      // Обнулить таймер T1
I=I+1;                       // Увеличить счетчик интервалов
port0=IOPIN0;                // Считать значения с порта P0
IOSET0=port0^(1<<0);         // Инверсия вывода на светодиод
IOCLR0=port0;
IOSET1=((I&(1<<4))<<16);     // Вывод младшей тетрады счетчика на индикатор
IOCLR1=(((I&(1<<4))^(1<<4))<<16);
}
}                            // Бесконечный цикл
}
```

6.3 Программирование схемы сравнения

Схема сравнения каждого T/C содержит две части – внутреннего управления и внешнего, и управляется соответственно регистрами управления сравнения T0MCR, T1MCR и регистрами внешнего совпадения T0EMR, T1EMR.

Регистры управления сравнением T0MCR, T1MCR предназначены для операций останова, сброса счетного регистра при совпадении его значения с содержимым одного из четырех регистров сравнения T0MR0 - T0MR3 для T/C0 (T1MR0-T1MR0 для T/C1), а так же инициации прерывания от соответствующего таймера/счетчика. Структура регистров управления сравнением показана на рисунке 6.5.

D0	MROI	Флаг разрешения прерывания от канала сравнения 0
D1	MR0R	Флаг разрешения сброса T/C от канала сравнения 0
D2	MR0S	Флаг разрешения останова счета T/C от канала срав-
D3	MR1I	Флаг разрешения прерывания от канала сравнения 1
D4	MR1R	Флаг разрешения сброса T/C от канала сравнения 1
D5	MR1S	Флаг разрешения останова счета T/C от канала срав-
D6	MR2I	Флаг разрешения прерывания от канала сравнения 2
D7	MR2R	Флаг разрешения сброса T/C от канала сравнения 2
D8	MR2S	Флаг разрешения останова счета T/C от канала срав-
D9	MR3I	Флаг разрешения прерывания от канала сравнения 3
D10	MR3R	Флаг разрешения сброса T/C от канала сравнения 3

Рисунок 6.5 – Структура регистров управления сравнением T0MCR, T1MCR

Регистры внешнего совпадения T0EMR, T1EMR управляют изменением состояния внешних выходов MAT0-MAT3 при возникновении совпадения значения счетного регистра с одним из четырех регистров сравнения T0MR0 - T0MR3 для T/C0 (T1MR0-T1MR0 для T/C1). При этом происходит сброс, установка или изменение состояния пина, сконфигурированного как соответствующий MAT выход. Структура регистров внешнего совпадения показана на рисунке 6.6.

D0	EM0	Бит отражает состояние выхода MAT0.0, MAT1.0
D1	EM1	Бит отражает состояние выхода MAT0.1, MAT1.1
D2	EM2	Бит отражает состояние выхода MAT0.2, MAT1.2
D3	EM3	Бит отражает состояние выхода MAT0.3, MAT1.3
D5,D4	EMC0	Биты управления режимом внешнего сравнения канала 0
D7,D6	EMC1	Биты управления режимом внешнего сравнения канала 1
D9,D8	EMC2	Биты управления режимом внешнего сравнения канала 2
D11,D10	EMC3	Биты управления режимом внешнего сравнения канала 3 Для всех четырех каналов: 0 0 (0) – Нет управления выходом 0 1 (1) – Сброс пина при событии 1 0 (2) – Установка пина при событии 1 1 (3) – Изменение состояния пина при событии

Рисунок 6.6 – Структура регистров внешнего совпадения T0EMR, T1EMR

Структурная схема, иллюстрирующая работу схемы сравнения, показана на рисунке 6.7. Схема показывает работу T/C0 с регистром сравнения T0MR3. При равенстве их значений генерируется сигнал события, которое при включенных битах MR3S, MR3R, MR3I регистра управления сравнением T0 выполняет соответственно останов, сброс счетного регистра T0TC, а также инициирует прерывание от T/C0. Это же событие, при различных значениях бит EMC3 регистра T0EMR выполняет сброс, или установку или изменение состояния пина, на котором выбран MAT0.3.

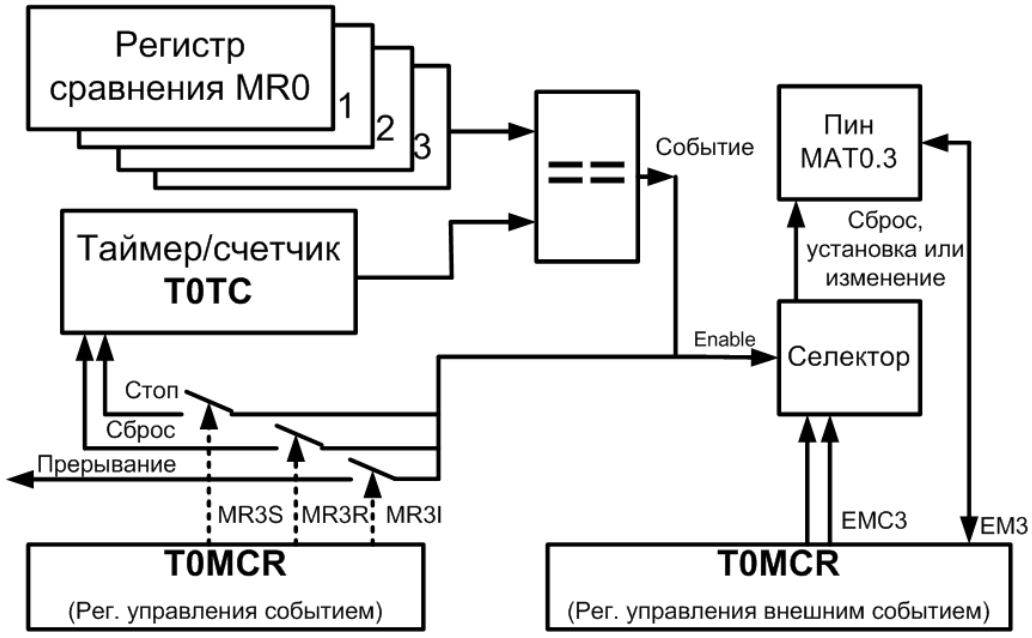


Рисунок 6. – Структура схемы сравнения

Пример программы, показывающей работу схемы сравнения с внешним выходом. Для наглядности, таймер/счетчик T0 суммирует импульсы

поступающие с CAP0.0. Используются два канала сравнения - MR1 - для сброса TC0 при достижении числа 09 и канал MR0 - для формирования сигнала на внешнем выходе MAT0.0 при совпадении с числом 04:

```
#include <LPC213x.h>
void main(void) {
PINSEL0=(1<<2)+(1<<3);      // Установить пин P0.2 в CAP0.0 + пин P0.3 в MAT0
IODIR0 =(1<<0)+(1<<2);      // Установить пин P0.0 , P0.4 выходом для подключения светодиодов
IODIR1 =(1<<4)+(1<<2);      // Установить пин P1.16-19 выходом для подключения 7-сег. индикатора
T0PR=1;                     // Предварительный делитель счетчика 0 = 1.
T0CTCR=(1<<0);              // Выбрать для TC0 источник счетных импульсов с CAP0 по //срезу
T0MCR=(1<<4);               // Разрешаем сброс TC0 при совпадении его с MR1
                            // Регистр управления сравнением. Управляет самим TC0 и прерыванием от схемы сравнения.
                            //D0-Разрешение прерывания от MR0, D1-Разрешение сброса TC0 от MR0, D2-//Разрешение останова TC0 от MR0
                            //D3-Разрешение прерывания от MR1, D4-Разрешение сброса TC0 от MR1, D5-//Разрешение останова TC0 от MR1
                            //D6-Разрешение прерывания от MR2, D7-Разрешение сброса TC0 от MR2, D8-//Разрешение останова TC0 от MR2
                            //D9-Разрешение прерывания от MR3, D10-Разрешение сброса TC0 от MR3, D11-//Разрешение останова TC0 от MR3

T0MR1=(1<<4)+(1<<0);        // Задаем значение регистра сравнения T0MR1, например равным 09
T0EMR=(1<<2);               // Разрешаем изменение MAT0.0 при совпадении TC0 с T0MR0
                            // Регистр управления внешним сравнением. Управляет пинами, сконфигурированными как MAT.
                            // D0- Отражает состояние MAT0.0 D1- Отражает состояние MAT0.1
                            // D2- Отражает состояние MAT0.2 D3- Отражает состояние MAT0.3
```

```
                                // D5 D4 - биты конфигурации управления пином, сконфигурированным как MAT0.0 при событии MR0
                                // D5=0 D4=0 - Нет управления пином 5=0 D4=1 - Сброс MAT0.0 при событии совпадения
                                // D5=1 D4=0 - Установка MAT0.0 при событии D5=1 D4=1 - Инверсия MAT0.0 при событии совпадения
                                // D7 D6 - биты конфигурации управления пином, сконфигурированным как MAT0.1 при событии MR1
                                // D9 D8 - биты конфигурации управления пином, сконфигурированным как MAT0.2 при событии MR2
                                // D11 D10 - биты конфигурации управления пином, сконфигурированным как MAT0.3 //при событии MR3

T0MR0=(1<<2);                   // Задаем значение регистра сравнения T0MR0, например равным 04
T0TCR=2;                        // Сбрасываем T/C0
T0TCR=1;                        // Разрешаем работу T/C0
while (1) {
IOSET0=T0TC&(1<<0);             // Вывод младшего бита T/C на светодиод
IOCLR0=(T0TC&(1<<0))^(1<<0);
IOSET1=((T0TC&(1<<4))<<16); //Вывод младшей тетрады T/C на индик.
IOCLR1=((T0TC&(1<<4))^(1<<4))<<16);
        };                      // Бесконечный цикл
}
```

Схема модели стенда, иллюстрирующая работу схемы сравнения показана на рис.6.8

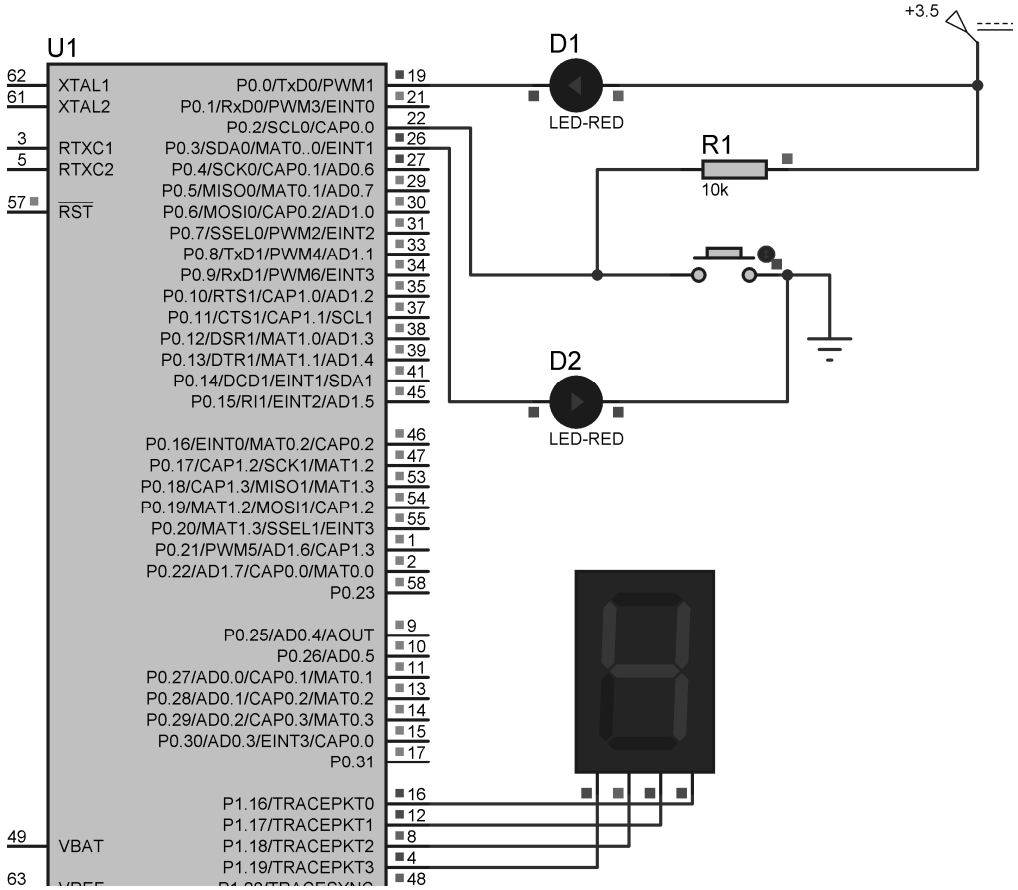


Рисунок 6.8 – Модель стенда, иллюстрирующая работу схемы сравнения

6.4 Программирование прерывания от схемы захвата

Помимо порождения внутренних и внешних событий схема сравнения может инициировать прерывания. Для этого должны быть установлены биты MR0I-MR3I регистра T0MCR (для T/C0) или регистра T1MCR (для T/C1).

Разрешение и порядок выполнения прерывания зависит от конфигурирования контроллера прерываний. Из регистров таймеров счетчиков к контроллеру прерываний относится регистры прерываний T0IR, T1IR.

Регистры прерываний T0IR, T1IR предназначены для определения источника прерываний в процедуре обработки, а также для сброса этого прерывания в конце ее выполнения. Как и во всех периферийных устройствах, не выполнение сброса приводит к повторному вызову прерывания. Структура регистра показана на рисунке 6.9.

D0	MROI	Флаг прерывания сравнения 0
D1	MR1I	Флаг прерывания сравнения 1
D2	MR2I	Флаг прерывания сравнения 2
D3	MR3I	Флаг прерывания сравнения 3
D4	CR0I	Флаг прерывания захвата 0
D5	CR1I	Флаг прерывания захвата 1
D6	CR2I	Флаг прерывания захвата 2
D7	CR3I	Флаг прерывания захвата 3

Рисунок 6.9 – Структура регистра прерываний таймеров/счетчиков T0IR, T1IR

Рассмотрим пример программирования FIQ прерывания от схемы сравнения:

// Программа демонстрирует работу прерывания от модуля сравнения (CAP0.1)

// Для простоты использовано быстрое (FIQ) прерывание

// Нажатие кнопки имитирует поступление счетных импульсов (для наглядности)

// Канал MR1 выполняет сброс, для ограничения максимального значения в счетчике

// Канал MR0 настроен на прерывание.

```
#include <LPC213x.h>
int main(void) {
PINSEL0=(1<<2);              // Установить пин P0.2 в  CAP0
IODIR0 =(1<<0)+(1<<2);       // Установить пин P0.0 , P0.4 выходом
IODIR1 =((1<<4)<<16);        // Установить пин P1.16-19  выходом
T0PR=1;
T0CTCR=(1<<0);               // Выбрать для TC0 источник счетных им-
                             пульсов с CAP0 по срезу
T0MCR=(1<<4)+(1<<0);         // Разрешаем сброс TC0 при совпадении его с
                             T0MR1
                             //+прерывание при его совпадении с T0MR0
T0MR0=(1<<2);                // Задаем значение регистра сравнения
                             T0MR0, например равным 04
T0MR1=(1<<4)+(1<<0);         // Задаем значение регистра сравнения
                             T0MR1, например равным 09
VICIntSelect = (1<<4);       // Включить FIQ для прерывания от T/C0
VICIntEnable = (1<<4);       // Разрешить  прерывания для от T/C0
T0TCR=2;                     // Сбрасываем счетный регистр и прескалер
                             T/C0
T0TCR=1;                     // Разрешаем работу T/C0
```

```
while (1) {
IOSET0=T0TC&(1<<0);              // Вывод младшего бита T/C на светодиод L1
IOCLR0=(T0TC&(1<<0))^(1<<0);
IOSET1=(T0TC&(1<<4))<<16;        // Вывод младшей тетрады T/C на индикатор
IOCLR1=((T0TC&(1<<4))^(1<<4))<<16;
 };                               // Бесконечный цикл вывода информации на индикацию
return 0;
}
void FIQ_Handler(void) __fiq
{
if (T0IR&(1<<0)) {               // Проверяем на совпадение с флагом прерывания от MR0
                                 // Можно и не проверять - в примере их больше нет!
if (IOPIN0&(1<<2))               // Инвертировать свечение светодиода L2
IOCLR0=(1<<2);
else
IOSET0=(1<<2);
T0IR=(1<<2);
                                 // Сбросить  флаг прерывания таймера TC0 от модуля сравнения MR0
}
                                 // T0IR - Регистр прерываний таймера счетчика TC0
                                 // D0 - Флаг прерывания от MR0  D1 - Флаг прерывания от MR1
                                 // D2 - Флаг прерывания от MR2  D3 - Флаг прерывания от MR3
                                 // D4 - Флаг прерывания от CAP0 D5 - Флаг прерывания от CAP1
                                 // D6 - Флаг прерывания от CAP2 D6 - Флаг прерывания от CAP3
                                 // При чтении биты указывают на совершение прерывания,
                                 // При записи выполняется сброс прерывания
VICVectAddr =(1<<4)+(1<<4);      // Сбросить контроллер прерываний
}
```

Модель, используемая программой показана на рисунке 6.8.

6.5 Программирование схемы захвата

Схема захвата, в противоположность схеме сравнения, не порождает внешние события, а фиксирует момент их возникновения. При появлении события на пинах контроллера, сконфигурированных как входы CAP0-CAP3, выполняется сохранение значения счетного регистра T/C в регистрах захвата CR0-CR3. Возможные события – фронт или срез импульса на внешнем входе, при необходимости формируется запрос на прерывание.

Режим работы схемы захвата определяется регистром управления захватом T0CCR и T1CCR для T/C0 и T/C1 соответственно. Структура регистра захвата показана на рисунке 6.10.

D0	CAP0RE	Флаг разрешения записи в T0CR0 T/C по фронту сигнала на CAP0.0
D1	CAP0FE	Флаг разрешения записи в T0CR0 T/C по срезу сигнала на CAP0.0
D2	CAP0I	Флаг записи в T0CR0 T/C и прерывания по сигналу на CAP0.0
D3	CAP1RE	Флаг разрешения записи в T0CR1 T/C по фронту сигнала на CAP0.1
D4	CAP1FE	Флаг разрешения записи в T0CR1 T/C по срезу сигнала на CAP0.1
D5	CAP1I	Флаг записи в T0CR1 T/C и прерывания по сигналу на CAP0.1
D6	CAP2RE	Флаг разрешения записи в T0CR2 T/C по фронту сигнала на CAP0.2
D7	CAP2FE	Флаг разрешения записи в T0CR2 T/C по срезу сигнала на CAP0.2
D8	CAP2I	Флаг записи в T0CR2 T/C и прерывания по сигналу на CAP0.2
D9	CAP3RE	Флаг разрешения записи в T0CR3 T/C по фронту сигнала на CAP0.3
D10	CAP3FE	Флаг разрешения записи в T0CR3 T/C по срезу сигнала на CAP0.3
D11	CAP3I	Флаг записи в T0CR3 T/C и прерывания по сигналу на CAP0.3

Рисунок 6.10 – Структура регистра захвата T0CCR, T1CCR

Функционирование схемы захвата иллюстрирует рисунок 6.11.

Рисунок 6.11 – Структура схемы захвата

Рассмотрим программу, иллюстрирующую программирование схемы захвата. Так как захват значения счетного регистра таймера/счетчика требует последующей обработки полученного значения, событие захвата привязано к FIQ прерыванию:

– Программа демонстрирует работу прерывания от модуля захвата, вход на CAP0.0

– Таймер TC0 выполняет отсчет времени, индикация по прерыванию захвата.

– Для простоты использовано быстрое (FIQ) прерывание

– Нажатие кнопки имитирует поступление импульса захвата, на индикатор выводится

– значения таймера. Предделитель настроен на изменение значения таймера через 0.1с

```
#include <LPC213x.h>
void main(void) {
PINSEL0=(1<<1);              // Установить пин P0.2 в  CAP0.0
IODIR1 = ((1<<4)<<16);       // Установить пины P1.16-23  выходом
T0PR=1500000;                // Предварительный делитель таймера нас
                             троим на изменение
                             // состояния таймера 0.1 раз в секунду (для
                             частоты pclk=15 МГц)
T0CTCR=(1<<0);               // Выбрать для TC0 источник счетных им-
                             пульсов PCLK
T0CCR=(1<<0)+(1<<1);
                             //Разрешить захват по срезу
                             CAP0.0+прерывание по событию захвата
```

```
                                // T0CR - Регистр управления захватом TC0
                                // D0 - Разрешение записи в регистр захвата
                                T0CCR0 по фронту на пине,
                                //  сконфигурированном как CAP0.0
                                // D1 - Разрешение записи в регистр захвата
                                T0CCR0 по
                                //срезу на пине,
                                // сконфигурированном как CAP0.0
                                // D2 - Разрешение прерывания при событии
                                захвата в регистр //T0CCR0
VICIntSelect = (1<<4);          // Включить FIQ для T0 прерывания
VICIntEnable    = (1<<4);       // Разрешить  прерывания для T0
T0TCR=2;                        // Сбрасываем T/C0
T0TCR=1;                        // Разрешаем работу T/C0
while (1) {};                   // Бесконечный цикл
}
void FIQ_Handler(void) __fiq
{
int V1,V2;                      // Вспомогательные переменные для 2-10
                                преобразования
if (T0IR&(1<<4)) {              // Проверяем на совпадение с флагом преры-
                                вания от CAP0.0
                                // T0IR - Регистр прерываний таймера счет-
                                чика TC0
                                // D0 - Флаг прерывания от MR0  D1 - Флаг
                                прерывания от MR1
                                // D2 - Флаг прерывания от MR2  D3 - Флаг
                                прерывания от MR3
                                // D4 - Флаг прерывания от CAP0 D5 - Флаг
                                прерывания от CAP1
                                // D6 - Флаг прерывания от CAP2 D6 - Флаг
                                прерывания от CAP3
                                // При чтении биты указывают на совершение
                                прерывания,
                                // При записи выполняется сброс прерывания
IOCLR1=(1<<4)+(1<<2);           // Тушим 7-сегментный индикатор
                                // Выводим содержимое регистра захвата на
                                7-сегментный индикатор
V1=T0CR0/10;                    // В десятичном виде
V2=T0CR0-V1*10;
IOSET1 = (V1<<20)+(V2<<16);
                                // Десятки выводятся в старший индикатор,
                                единицы - в младший
T0IR=(1<<4);
```

```
                                    // Сбросить  флаг прерывания таймера
                                    TC0 от модуля захвата CAP0
}                                   // Конец проверки условия прерывания
                                    от CAP0
}
                                    // Конец процедуры обработки преры-
                                    вания от T/C0
```

Модель, иллюстрирующая работу программы захвата, показана на рисунке 6.12.

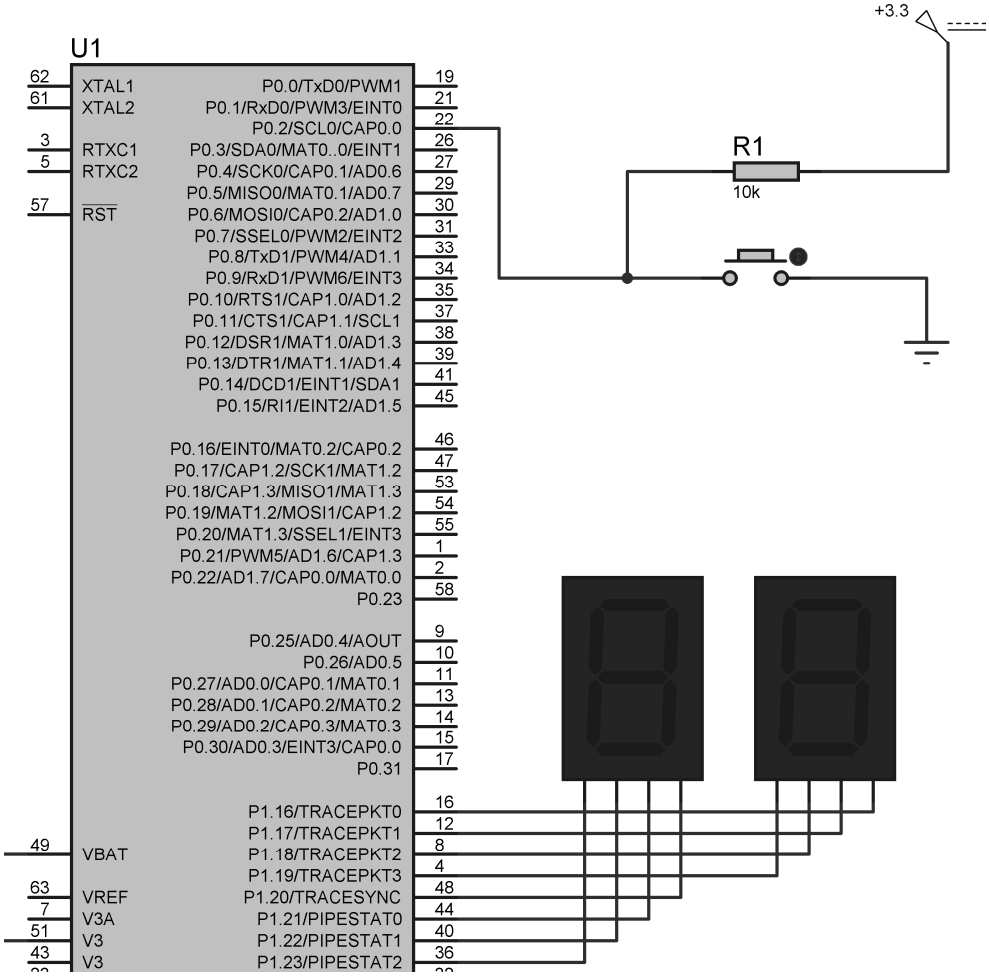


Рисунок 6.12 – Модель стенда, иллюстрирующая работу схемы захвата

7 ПРОГРАММИРОВАНИЕ АЦП, ЦАП и ШИМ

7.1 Особенности АЦП

Аналого-цифровой преобразователь (АЦП) – периферийное устройство микроконтроллера, обеспечивающее преобразование величины входного напряжения в пропорциональный ему код.

Особенности АЦП [1] Стр.199-210:

– В контроллере размещены два 10-разрядных АЦП последовательного приближения с временем преобразования более 2.44 мкс.

– Частота квитирования преобразователя настраивается встроенным предделителем и не должна превышать 4.5МГц.

– Максимальная частота выборок: 410 выборок в секунду на каждом АЦП.

– Каждый АЦП оснащен входным мультиплексором на 6 (AD0) и на 8 (AD1) входов.

– Возможно отключение питания АЦП.

– Возможность подачи сигнала запуска преобразования АЦП от различных источников.

– Входное напряжение АЦП составляет 0..Vref, где опорное напряжение Vref обычно принимают равным 3В.

7.2 Электрические характеристики АЦП

Каждый из двух модулей АЦП AD0 или AD1 подключены одновременно только к одному из внешних пинов микроконтроллера, сконфигурированных как входы AD0.1-AD0.4, AD0.6-AD0.7 (для AD0) и AD1.0-AD1.7 (для AD1). Входное измеряемое напряжение на этих пинах не должно превышать 3.3В, максимальное допустимое входное напряжение составляет 5В. При подключении внешних источников напряжения необходимо учитывать, что их выходное сопротивление не должно превышать 40кОм.

Во входной цепи каждого пина находится П-образный RC фильтр с частотой среза около 4.5 МГц.

Верхний диапазон измеряемого входного напряжения АЦП ограничено величиной опорного напряжения на пине V_{ref}. Это напряжение должно быть стабилизировано на уровне ±1мВ (для V_{ref}=3В), при этом условии АЦП обеспечивает паспортную точность измерений.

Питание модулей АЦП выполнено через отдельные пины V_{dda} и V_{ssa}, которые необходимо подключить к отдельной линии питания и изолировать от возможных наводок или помех. Пины питания должны быть шунтированы керамическими конденсаторами номиналом 0.1мкФ вблизи корпуса микроконтроллера.

7.3 Регистры АЦП

Перечень регистров модуля АЦП, их функциональное назначение и режим доступа показаны в таблице 7.1.

Таблица 7.1 – Регистры управления АЦП

Общее имя	Описание	Доступ	При сбросе	Имя регистра	Имя регистра
ADCR	Регистр управления. Выбирает режим работы АЦП.	R/W	1	AD0CR	AD1CR
ADGDR	Регистр данных. Содержит бит готовности и результат преобразования.	R/W	-	AD0GDR	AD1GDR
ADGSR	Регистр общего запуска.	WO	0	ADGSR	

Регистры управления АЦП AD0CR, AD1CR ([1] стр.213) предназначены для управления входным мультиплексором АЦП (биты D0-D7), управления частотой квантования последовательного приближения (биты D8-D15), количество квантов преобразования (разрядностью полученных данных), событийным или непрерывным режимом преобразования, а также позволяют полностью отключить модуль АЦП. Структура и описание регистра управления показана в таблице 7.2.

АЦП функционирует в режиме непрерывного или событийного запуска в зависимости от бита BURST регистров управления. В режиме непрерывного запуска биты CLKS задают разрядность преобразования и соответственно полное время единичного преобразования. Так, если делитель CLKDIV обеспечивает тактовую частоту преобразования одного бита 4.5МГц, а CLKS=0, то количество разрядов результата преобразования равно 10, а частота преобразования АЦП составит 450кГц. При уменьшении разрядности частота преобразования возрастает, например для CLKS=7 составит 1.5 МГц.

В режиме событийного запуска анализируется состояние одного из пинов микроконтроллера, заданного битами START. При возникновении события, заданного битом EDGE выполняется запуск АЦП преобразования.

Таблица 7.2 – Структура регистров управления AD0CR, AD1CR

Биты	Символ	Назначение
D0-D7	SEL	Выбор пина для входа АЦП. D0 – AD0.0, D1 – AD0.1, D2 – AD0.2 и т.д. Для AD1CR соотв. D0 – AD1.0. Установленный младший пин имеет приоритет перед старшими. Если SEL=0 то принимается AD0.0.
D8-D15	CLKDIV	Коэффициент деления тактовой частоты. Результат деления PCLK на (CLKDIV+1) = частота преобразования одного бита не должен превышать 4.5 МГц.
D16	BURST	Непрерывный (BURST=1) или событийный (BURST=0) режим преобразования.
D17-D19	CLKS	CLKS. Число разрядов в BURST режиме. CLKS=0 – 10 разрядов, 1 – 09 разрядов, 2- 08 разрядов, и т.д. до CLKS=7 – 3 разряда.
D20	-	Резерв
D21	PDN	Включить (PDN=1) или выключить (PDN=0) АЦП.
D22-23	-	Резерв
D24-26	START	Задает условия старта при BURST=0. Если START=0 – Нет старта. START=1 - Немедленный старт. START=2 – Старт по сигналу EDGE на пине P0.16. START=3 – Старт по сигналу EDGE на пине P0.22. START=4 – Старт по EDGE на MAT0.1. START=4 – Старт по EDGE на MAT0.3. START=6 – Старт по EDGE на MAT1.0. START=7 – Старт по EDGE на MAT1.1.
D27	EDGE	Стартовый сигнал. EDGE=1 – По Фронту. EDGE=0 – По срезу
D27-28	-	Резерв

Результат преобразования размещается в регистре данных AD0GDR для AD0 и AD1GDR для AD1. Структура и описание регистров данных показана в таблице 7.3. Биты 6-15 содержат результат преобразования. При использовании режима BURST с уменьшением разрядности младшие разряды RESULT становятся незначащими. Бит OVERUN устанавливается когда значение предыдущего преобразования не было считано и уничтожено текущим результатом. Бит готовности DONE может быть использован для программного контроля готовности результата АЦП преобразования.

Таблица 7.3 – Структура регистров данных AD0GDR и AD1GDR

Биты	Символ	Назначение
D0-D5	-	Резерв
D6-D15	RESULT	Содержит двоичное представление от результата деления Vвх/Vref. Например при Vвх=3В – 0x3ff, при Vвх=0В – 0x00.
D16-23	-	Резерв
D24-26	CHN	Содержит номер канала результата последнего преобразования. 0 соответствует каналу 0, 1 – первому каналу и т.д.
D27-29	-	Резерв
D30	OVERUN	Если OVERUN =1 то предпоследний результат преобразования был перезаписан без считывания (т.е. уничтожен)
D31	DONE	DONE=1 если результат готов. DONE сбрасывается в 0 после чтения или если произошла запись в AD0CR (AD1CR).

Значение RESULT определяется по формуле (7.1):

$$RESULT = 0x3FF\frac{V_{âõ}}{V_{ref}} \qquad (7.1)$$

где $V_{вх}$ - напряжение на подключенном к модулю АЦП входе;

V_{ref} - опорное напряжение, поданное на пин Vref.

Значение RESULT лежит в диапазоне 0..0x3FF.

При необходимости обеспечения одновременно двумя каналами АЦП используется регистр общего старта ADGSR. Структура регистра, приведенная в таблице 7.4, напоминает структуру регистров управления, но позволяет управлять двумя модулями АЦП одновременно.

Таблица 7.4 – Структура регистра общего старта ADGSR

Биты	Символ	Назначение
D00-15	-	Резерв
D16	BURST	Непрерывный (BURST=1) или событийный (BURST=0) режим преобразования.
D17-23	-	Резерв
D24-26	START	Задает условия старта при BURST=0. (См. таб. 7.2)
D27	EDGE	Стартовый сигнал. EDGE=1 – По Фронту. EDGE=0 – По срезу
D27-28	-	Резерв

7.3 Пример программирования АЦП

На рисунке 7.1 показан фрагмент модели, иллюстрирующей использование АЦП в микроконтроллерах LPC21xx.

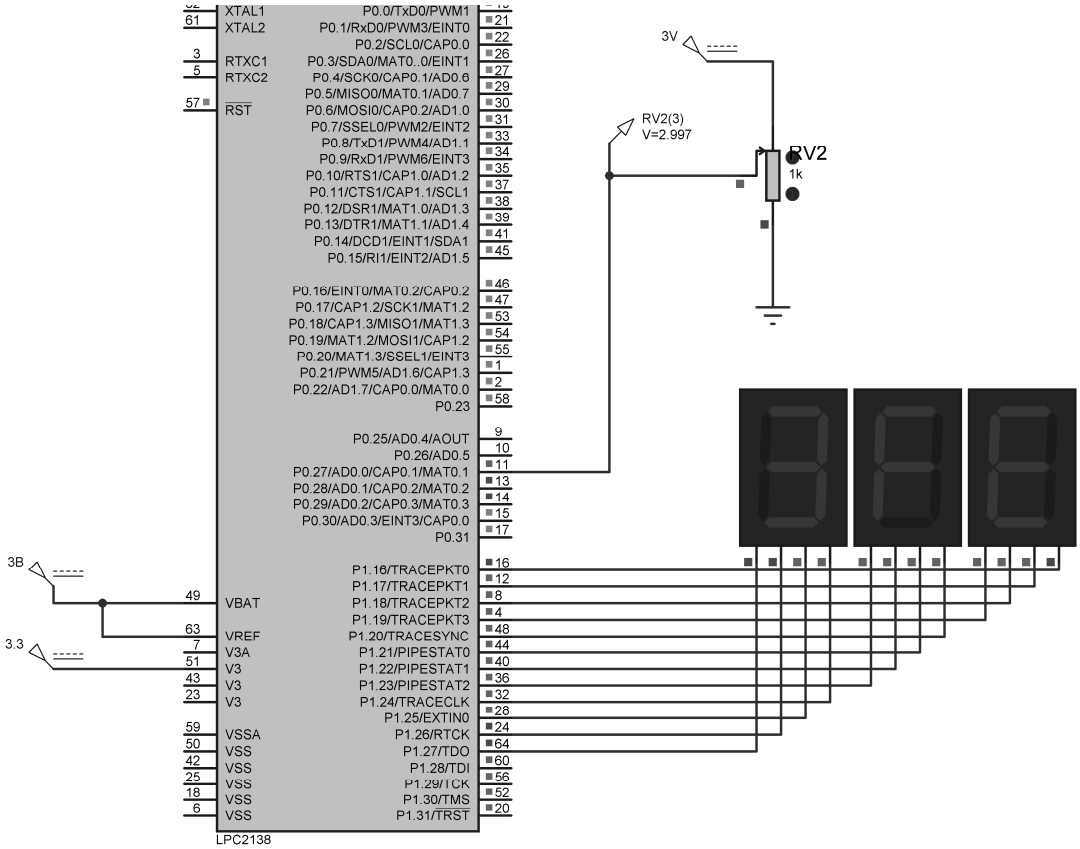


Рисунок 7.1 – Модель стенда, иллюстрирующая работу АЦП

Измеряемое напряжение подается на вход пин P0.27, сконфигурированный как AD0.0. Результат преобразования выводится на светодиодный знакосинтезирующий индикатор.

```
// Программа демонстрации возможностей АЦП
// Данные со входа ADC0 в конце преобразования передаются на семесигментные
// индикаторы по прерыванию.
#include <LPC213x.h>
#include <stdlib.h>
void IRQ_ADC(void) __irq;
void main(void) {
                                        // fкв=10 МГц   PCLK=15 МГЦ
                                        // (CLKDIV+1) = PCLK/4.5=3.33
                                        Принимаем CLKDIV=3,
                                        //тогда fпр=15/4=3.75 МГц.
IODIR1=(((1<<4)+(1<<2))<<16);           // Вывод на 7 сег. индикаторы
```

```
PINSEL1=((1<<1)<<22);                    // Установить пин P0.27 в AD0.0
AD0CR = (1<<1) + (1<<8) + (2<<16) + (2<<21);
// Выбор входа AD0.0 + CLKDIV=3 + BURST=1 + PDN=1
// Регистр управления АЦП
// D0-D7 выбор входного пина. D8-D15 - Частота преобразования CLKDIV
// D16 - BURST - Непрерывный (1) или событийный (0) старт АЦП
// D17-D19 - CLKS - Число тиков (разрядность) в BURST=1 режиме
// D21 - PDN - Включение питания АЦП
// D24-D26 - Режим событийного старта: 0 - Нет старта
// 1 - Немедленный старт  2 - Старт по EDGE P0.16 3 - Старт по EDGE P0.22
// 4 - Старт по EDGE MAT0.1        5 - Старт по EDGE MAT0.3
// 6 - Старт по EDGE MAT1.0 7 - Старт по EDGE MAT1.1
// D27 - EDGE.   EDGE =1 - По фронту.  EDGE = 0 - По срезу

VICVectCntl0 = (1<<(5+18));
// Разрешить Слот 0 + Задать номер прерывания AD0
// биты D0-D4 - номер источника прерывания
//00 - WDT,       01 - Не исп.      02 - ARMCore1  03 - ARMCore2
//04 - Timer0     05 - Timer1       06 - UART0     07 - UART1
//08 - PWM0       09 - I2C0         10 - SPI0      11 - SPI1
//12 - PLL        13 - RTC          14 - EINT0     15 - EINT1
//16 - EINT2      17 - EINT3        18 - AD0       19 - I2C1
//20 - BOD        21- I2C1          22 - AD1       23 - Не исп.
// бит D5-Разрешение прерывания для данного слота. Бит=1 - прерывание разрешено

VICIntSelect=(1<<0);                     // Запретить FIQ прерываний
VICVectAddr0=(unsigned)IRQ_ADC;          // Задать адрес прерывания от ADC0
VICIntEnable=(1<<18);                    // Разрешить AD0 прерывание
while (1) {};                            // Бесконечный цикл
}
void IRQ_ADC(void) __irq                 // Векторное прерывание
{       int ADC;                         // Считанное значение из АЦП
        ADC=AD0DR&(1<<6);                // Считываем значащие данные из АЦП
                                         // D6-D15 - Данные АЦП   D24-D26 - Номер кА //нала последнего преобразования
                                         // D30 - OVERUN - Флаг перезаписи несчитанных данных
                                         // D31 - DONE - Флаг готовности результата. //Сбрасывается при чтении AS0GDR или при записи в AD0CR
```

```
    ADC=(ADC>>6);              // Смещаем данные в разрядной сетке
до нуля
    IOCLR1=((1<<4)<<16);       // Очистить индикаторы
    IOSET1=(ADC<<16);          // Вывести значение из АЦП на индика-
цию
    VICVectAddr =(1<<0);       // Сбросить контроллер прерываний
}
```

7.5 Программирование ЦАП

Цифро-аналоговый преобразователь (ЦАП) – периферийное устройство микроконтроллера, обеспечивающее преобразование кода в значение напряжения на выходе AUOT.

Особенности ЦАП [1] Стр.219-220:

– 10 битный цифро-аналоговый преобразователь.

– Буферизированный выход по напряжению.

– Наличие функция управления питанием ЦАП.

– Переключение режимов скорость или энергопотребления.

– Выходное напряжение АЦП изменяется в диапазоне от 0 до Vref.

Модуль ЦАП управляется при помощи регистра DAC, структура которого показана в таблице 7.5.

Таблица 7.5 – Структура регистра ЦАП DAC

Биты	Символ	Назначение
D0-D5	-	Резерв
D6-D15	VALUE	Содержит двоичное представление выводимого напряжения. Например при VALUE=0x3FF напряжение на VOUT=VREF.
D16	BIAS	Флаг быстродействия. При BIAS=0 время установки выходного напряжения 1мкС и ток потребления ЦАП 0.7мА; При BIAS=1 время установки 2.5мкС и ток потребления ЦАП 0.350мА.
D16-31	-	Резерв

Для расчета выходного напряжения используется формула (7.2):

$$V_{out} = \frac{VALUE}{1024} \cdot V_{ref} \qquad (7.2)$$

Возможности модуля ЦАП ограничены количеством каналов, но его наличие позволяет облегчить разработку специализированных устройств, например, как приведенный на рисунке 7.2 программируемый генератор синусоидальных сигналов.

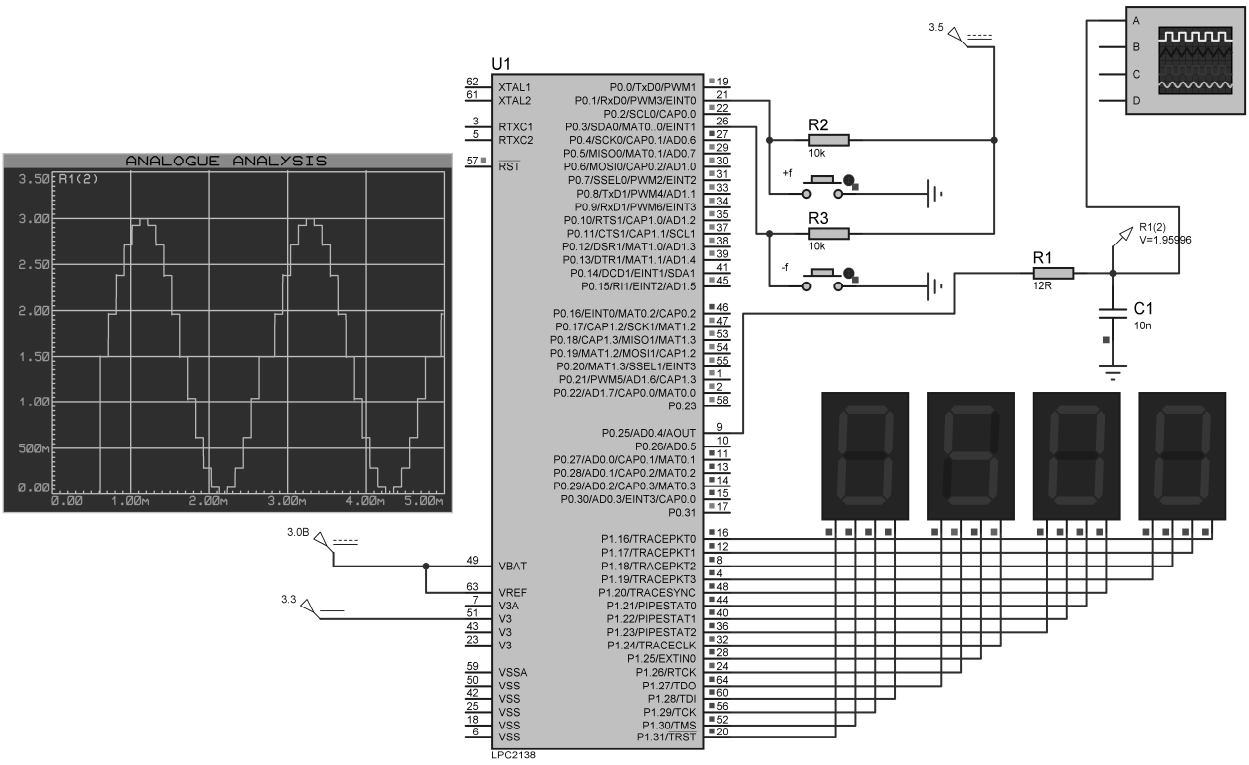


Рисунок 7.2 – Модель стенда генератора синусоидального сигнала

```
//Демонстрационный пример генератора синусоидального сигнала
// С управлением по частоте, регулировки завязаны на  векторные IRQ прерывания
// от внешнего источника. Период квантования привязан на TC0
#include <LPC213x.h>
#include <math.h>
static int f_max = 1000 ;          // Максимальное рабочее значение частоты
static int f;                      // Рабочее значение частоты
static int df;                     // Изменение частоты
static int n_max;                  // Число квантов в периоде
static int n;                      // Номер кванта  в периоде
void IRQ_EINT0(void) __irq;
void IRQ_EINT1(void) __irq;
void IRQ_TC0(void)   __irq;
void ind( int f );                 // Визуализация значения частоты на индикатор
void main(void) {
f=f_max/2;                         // Инициализация рабочей частоты
                                   // Пусть частота приращения таймера с учетом предделителя = 10000 Гц
n_max=(10000/f);                   // Определяем число квантов в периоде рабочей частоты
```

```
                                // Инициализация портов ввода-вывода
PINSEL0=(1<<0)+(1<<2);          // Установить пин P0.1 в EINT0 , P0.3 в
EINT1
PINSEL1=(1<<4)+(1<<3);          // Установить пин P0.25 в AOUT
IODIR1 =(1<<4)+(1<<8);          // Установить пин P1.16-P1.31 выходом
                                // Настройка таймера T0
T0CTCR=(1<<0);                  // Выбрать для TC0 режим таймера
                                // Регистр управления источником счетных
                                импульсов таймера/счетчика T0
                                // Кварц 10МГц, Умножение 6 Деление 1, то-
                                гда периферийный синхросигнал 15 МГц
T0PR=14;                        // Предварительный делитель таймера 0

T0MR0=100;

T0MCR=(1<<3);                   // Разрешаем сброс TC0 и прерывание при
                                совпадении его с MR0
                                // Регистр управления сравнением. Управляет
                                самим TC0 и прерыванием от схемы сравне-
                                ния.
                                // D0-Разрешение прерывания от MR0 D1-
                                //Разрешение сброса TC0 от MR0 D2-
                                Разрешение останова

T0TCR=1;                        // Разрешаем работу TC0

                                // Инициализация внешнего прерывания
EXTMODE=(1<<2)+(1<<0);          // Прерывание по гребню:
EXTPOLAR=(1<<0);                // Выбрать для EINT0, EINT1 прерывание по
срезу

                                // Инициализация контроллера прерывания
VICVectCntl0 =((1<<0)+14);      // Разрешить Слот 0 + Задать номер прерыва-
ния IENT0
VICVectCntl1 =((1<<1)+15);      // Разрешить Слот 1 + Задать номер прерыва-
ния IENT1
VICVectCntl2 = ((1<<2)+14);     // Разрешить Слот 2 + Задать номер
прерывания T0
                                // биты D0-D4 - номер источника прерывания
                                //00 - WDT, 01 - Не исп. 02 - ARMCore1 03
                                //04 - Timer0 05 - Timer1 06 - UART0
                                07 - UART1
                                //08 - PWM0 09 - I2C0 10 - SPI0 11 -
                                SPI1
```

```
                                //12 - PLL    13 - RTC    14 - EINT0        15 -
                                EINT1
                                //16 - EINT2   17 - EINT3       18 - AD0    19 -
                                I2C1
                                //20 - BOD    21- I2C1    22 - AD1    23    -    He
                                исп.
                                // бит D5-Разрешение прерывания для данно-
                                го слота. Бит=1 - прерывание разрешено
VICIntSelect=(1<<0);                    //Не включить FIQ преры-
                                        ваний
VICVectAddr0 = (unsigned)IRQ_EINT0;     //Задать адрес прерывания
от EINT0
VICVectAddr1 = (unsigned)IRQ_EINT1;     //Задать адрес прерывания
от EINT0
VICVectAddr2 = (unsigned)IRQ_TC0;       //Задать адрес прерывания
от EINT0
VICIntEnable=(1<<14)+(1<<15)+(1<<4);    //Разрешить прерывания
                                        EINT0 + EINT1 //+ T0
while (1) {
ind(f);  };                             //Бесконечный цикл
}

void ind( int i )                       //Визуализация значения
                                        частоты на индикатор
{
int var1, var2, var3;
var1=i/1000;
i=i-var1*1000;
var2=i/100;
i=i-var2*100;
var3=i/10;
i=i-var3*10;
i=i+(var3<<4)+(var2<<8)+(var1<<12);
i=i<<16;
IOSET1=i;
IOCLR1=i^((1<<8)+(1<<4));
}
void IRQ_EINT0(void) __arm __irq
{
df=f/10;      //Изменить девиацию частоты
f=f+df;                                 //Изменить частоту
if (f>f_max) f=f_max;                   //Проверка границ
n_max=(10000/f);                        //Определяем число квантов в пе-
                                        риоде
```

```
  EXTINT=(1<<2);                                  //Сбросить флаг внешнего
                                                  прерывания EINT0
        VICVectAddr= (1<<0);                      //Сбросить контроллер пре-
                                                  рываний
}

void IRQ_EINT1(void) __arm __irq
{
df=f/11;                                  //Изменить девиацию частоты
f=f-df;                                   // Изменить частоту
if (f<1) f=1;                             // Проверка границ
n_max=(10000/f);                          // Определяем число квантов в периоде
EXTINT=(1<<2);                            // Сбросить флаг внешнего прерывания
VICVectAddr =(1<<0);                      // Сбросить контроллер прерываний
}


void IRQ_TC0(void) __arm __irq
{ double a;
  a=(1<<8)+(1<<4)*sin(2*3.141592*n/n_max);    //Расчет значения синус.
                                              функции
DACR=a;
n=n+1;
if (n>n_max) n=0;
        T0IR=(1<<0);                          //Сбросить флаг прерыва-
                                              ния таймера //TC0 от моду-
                                              ля сравнения MAT0
VICVectAddr= (1<<0);                          //Сбросить контроллер пре-
                                              рываний

}
```

7.6 Особенности ШИМ

Широтно Импульсный Модулятор – периферийное устройство микроконтроллера, обеспечивающее возможность модуляции необходимой ширины импульса для выполнения конкретной задачи.

Особенности ШИМ микроконтроллера LPC 2138:

1. Семь регистров совпадения позволяют сформировать на выходах ШИМ до 6 отдельных управляемых перепадов, или до трех двойных перепадов, или комбинировать эти виды сигналов.
2. Внешний выход для каждого регистра совпадения обладает следующими функциональными возможностями:

– сброс при совпадении
– установка при совпадении
– переключение уровня
– сохранение текущего состояния при совпадении

3. Поддержка одного управляемого перепада и/или двойного управляемого перепада на выходах ШИМ.
4. Период импульса и его ширина могут быть равны любому количеству периодов счетных импульсов таймера.
5. Двойной управляемый перепад на выходах ШИМ может быть запрограммирован

Чтобы получить положительные или отрицательные импульсы выходной последовательности.

6. Модуль ШИМ может использовать как стандартный таймер, если режим ШИМ не разрешен.
7. 32- битный таймер/счетчик с программируемым 32-битным предделителем.

7.7 Структура и общее описание модуля ШИМ

Модуль ШИМ реализован на основе стандартного таймера и наследует все его особенности. Таймер ШИМ предназначен для подсчета периода периферийных тактовых импульсов, и имеет возможность генерировать прерывания или выполнять другие задачи в случаях, когда его значение достигает заданных величин, определяемых содержимым семи регистров совпадения ШИМ. Модуль ШИМ также включает в себя 4 входа захвата, позволяющие захватывать текущее значение таймера при поступлении заданных изменений входных сигналов, а также генерировать прерывание, когда происходит захват. Как и все перечисленные функции, функция ШИМ базируется на использовании регистров совпадения. Блок-схема модуля ШИМ представлена на рисунке 7.3.

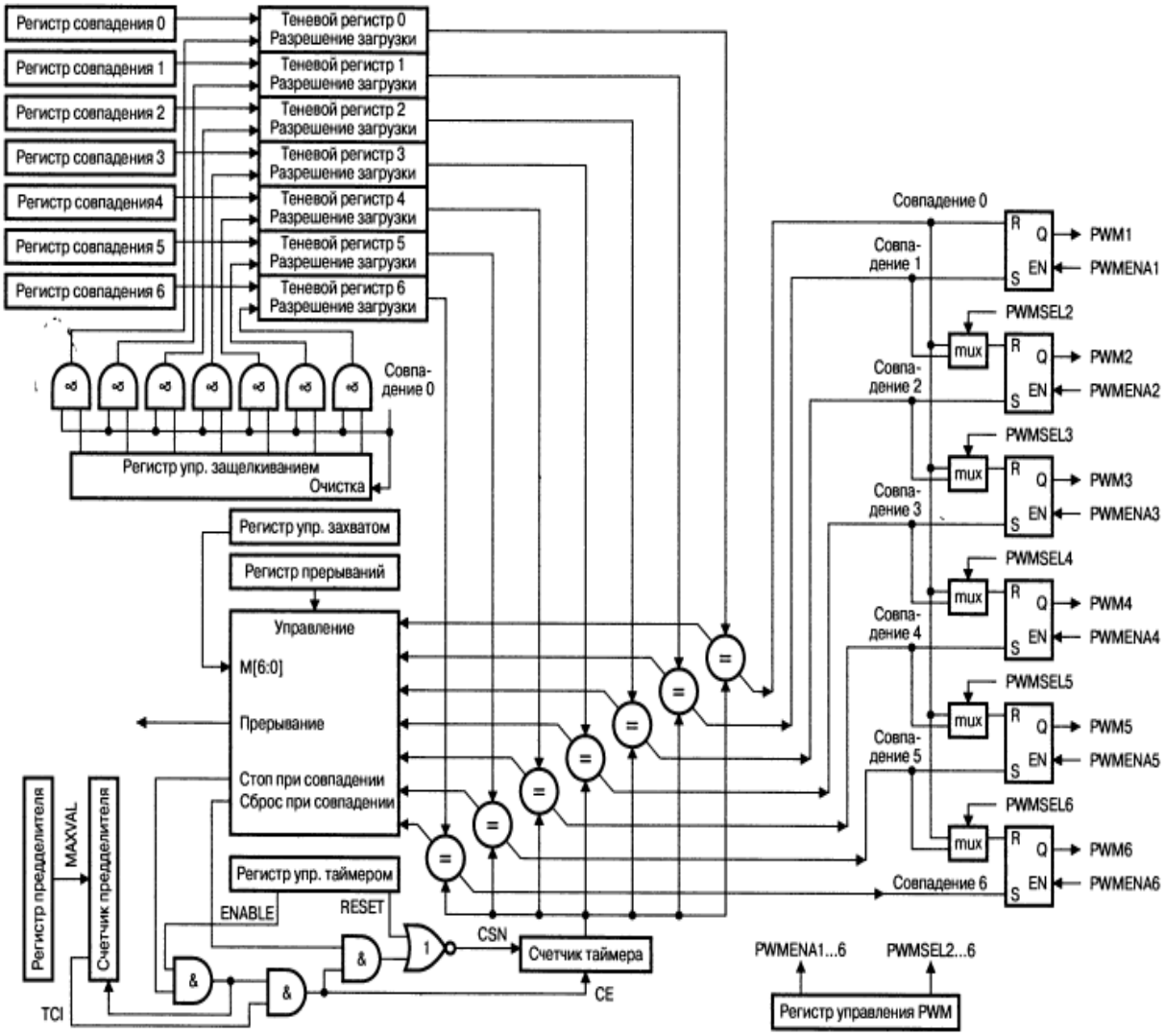


Рисунок 7.3 – Блок-схема модуля ШИМ

7.8 Особенности реализации ШИМ на LPC 2138

1. В основе ШИМ лежит 32 битный Т/С с 32 битным прескайлером.
2. Шесть модулей ШИМ с управлением по срезу (вар. 1 на диаграмме 7.1), или 3 модуля с управлением по фронту и срезу (вар. 2 на диаграмме 7.1).
3. Семь 32-ух битных модуля сравнения (MR0 – MR6)
4. Семь 32-ух битных теневых регистра защелки

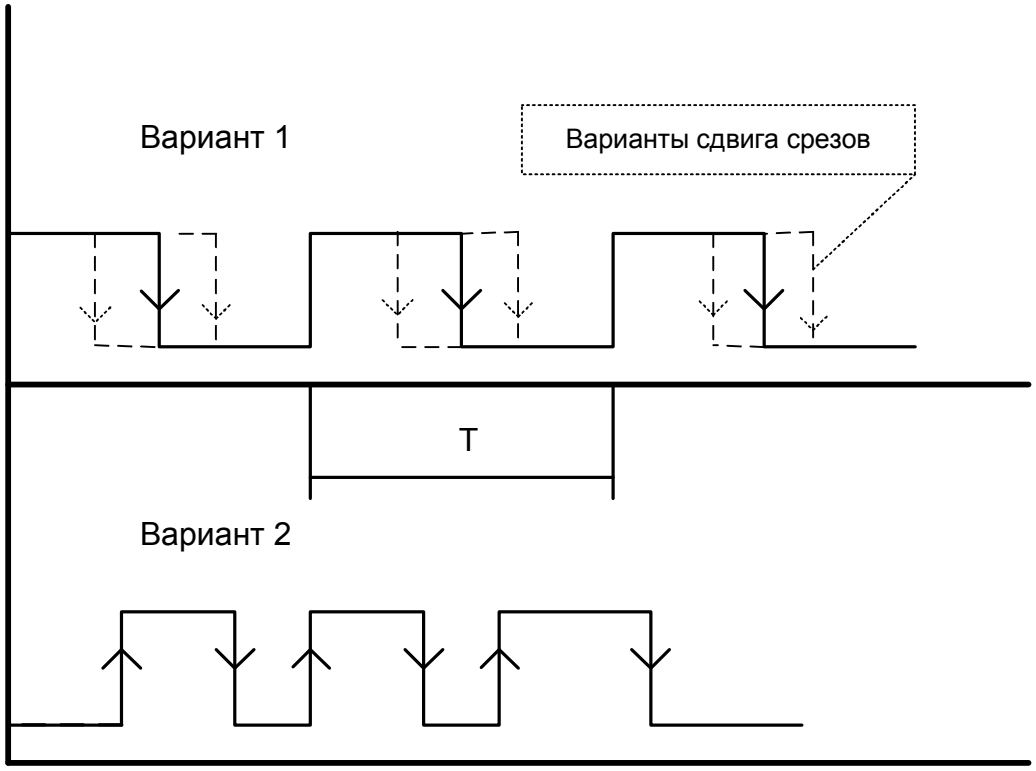


Диаграмма 7.1 – Варианты срабатывания ШИМ

7.9 Регистры управления ШИМ

В таблице 7.6 показаны регистры управления ШИМ.

Таблица 7.6 – Регистры управления ШИМ

Название	Описание
PWMIR	Регистр прерываний от ШИМ. Регистр PWMIR может быть записан, чтобы сбросить прерывания, и может быть прочитан, чтобы определить, запросы от каких источников прерывания поступили.
PWMTCR	Регистр управления ШИМ. Регистр PWMTCR используется, чтобы управлять функциями счетчика таймера.
PWMTR	Счетчик предделителя ШИМ 32-битный регистр PWMTR инкрементируется до значения, содержащегося в регистре PWMPR.
PWMPR	Регистр предделителя ШИМ. Регистр PWMTC инкрементируется каждые (PR+1) циклов pclk
PWMTC	Счетчик таймера ШИМ. 32-битный регистр PWMTC инкрементируется через каждые (PR+1) циклов pclk, где PR – значение регистра PWMTCR
PWMMCR	Регистр управления совпадения PWM.
PWMLER	Регистр разрешения защелкивания ШИМ. Разрешает использовать новые значения совпадения ШИМ.

Для включения ШИМ необходимо использовать следующую команду:

$PWMTCR = (1 << 0) + (1 << 3);$ //1<<0 – включить таймер
//1<<3 – включить ШИМ

7.10 Расчет разрешающей способности ШИМ

1. Определить частоту приращения T/C ≤ 15 МГц
2. Определить период следования импульсов ШИМ (частоту) для большего количества случаев F=20 кГц
3. Рассчитываем дискретность ШИМ по формуле:

$D = \frac{fòñ}{føèì}$, max разрешающая способность для рекомендуемых параметров составит не более D=750

7.11 Использование ШИМ для управления электроприводом

Для применения ШИМ в качестве управляющего модуля ЭД необходимо:

1. Подключить МК к микросхеме серии IR2138, которая является усилителем, она служит для сопряжения МК с транзисторами, которые работают в качестве ключей.

2. Транзисторы подключить к выходам Ho и Lo, т.е. обеспечить реверс ЭД.

3. Соединить коллектор и эмиттер транзисторов, а эмиттер первого подключить к питающему напряжению ЭД.

4. Якорную обмотку ЭД подсоединить к коллектору транзистора VT1.

5. В цепь баз транзисторов поставить токоограничительные резисторы, для обеспечения токов срабатывания транзисторов.

6. Параллельно питанию микросхемы IR2138 Vcc и Vss поставить конденсатор, который будет поддерживать динамическую мощность микросхемы.

7. Подключить выходы PWM1 и PWM2 МК LPC2138 к входам HIN и LIN микросхемы.

8. Далее необходимо разработать программное обеспечение МК для управления ЭД.

На рисунке 7.4 представлена схема подключения ЭД к МК LPC2138.

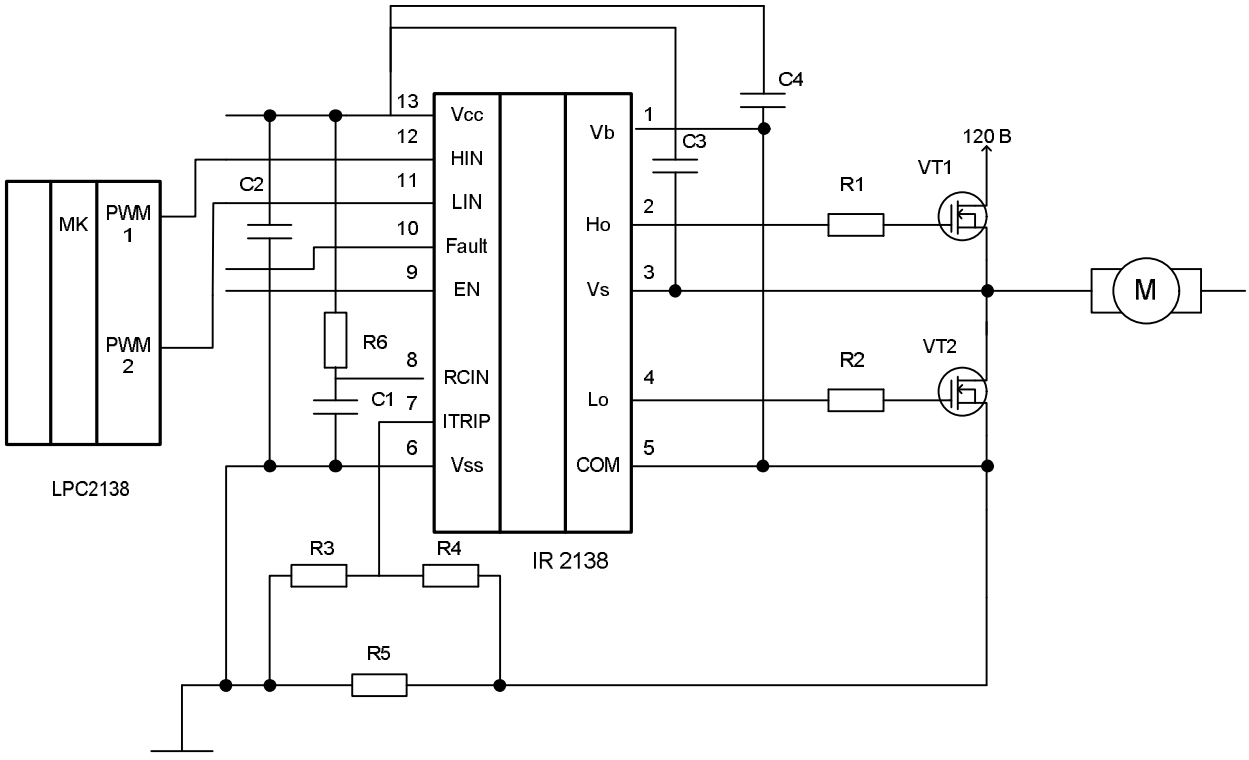


Рисунок 7.4 – Схема подключения ЭД к МК LPC2138

7.12 Пример программирования ШИМ

На рисунке 7.5 представлена модель стенда, иллюстрирующего работу ШИМ, при нажатии кнопок меняется яркость свечения светодиодов на PWM0, PWM1.

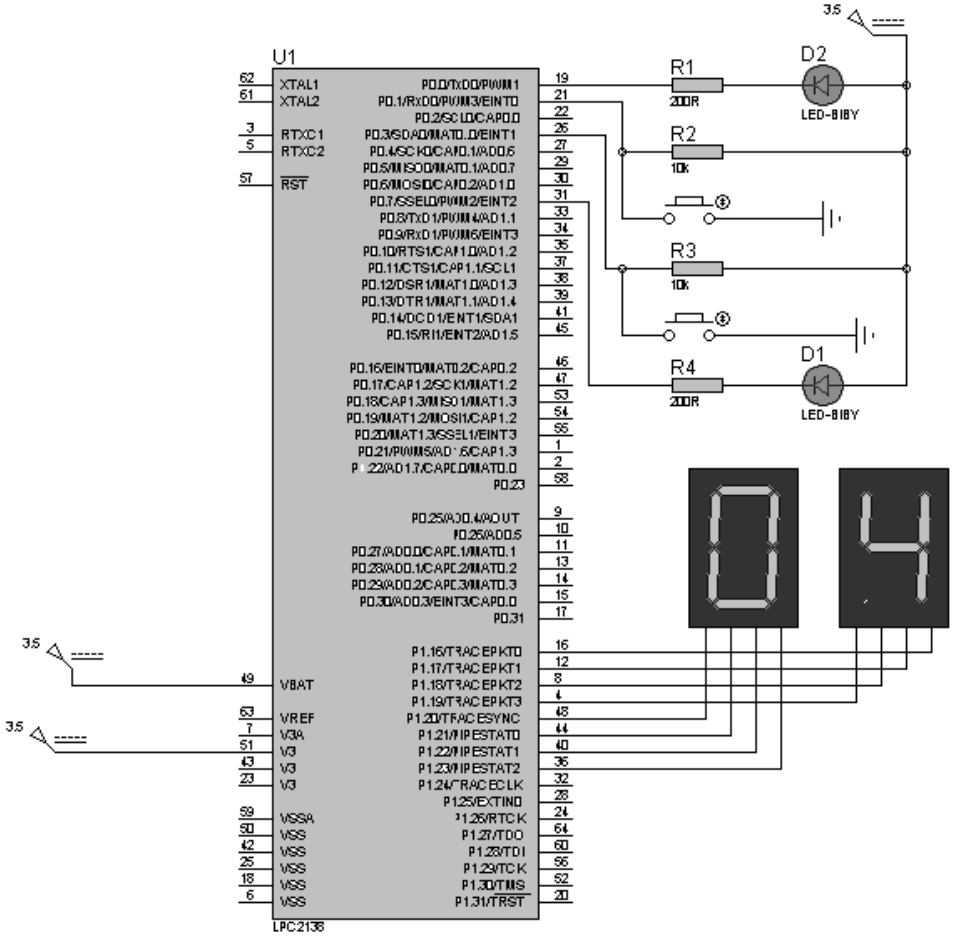


Рисунок 7.5 – Модель стенда

Программа на языке С:

// При нажатии кнопок меняется яркость свечения светодиодов на PWM0, PWM1

// Кнопки привязаны к внешним прерываниям

```
#include <LPC213x.h>
#include <stdlib.h>
int kod;                          //Текущее значение для ШИМ
int main(void) {
unsigned int maxkod ;             //Максимальное значение для ШИМ
```

```
div_t res;                          //Переменная для деления
maxkod=10;                          //Задаем максимальное значение
                                    //длительности периода ШИМ в тактах
PCLK
res=div(maxkod,2);                  //Задаем текущее значение     для ШИМ
                                    канала 1
kod=res.quot;                       // Выделить Частное из результата де-
                                    ления

IODIR1=(1<<4);                      // Вывод на 7 сег. индикаторы
PINSEL0=(1<<1)+(1<<2)+(1<<4)+(1<<0);
                                    // Установить пин P0.0 в PWM1 + P0.1 в
                                    EINT0 + //P0.3 в  EINT1 + P0.3 в PWM2
EXTMODE=(1<<2)+(1<<0);              // Выбрать для EINT0, EINT1 прерыва-
                                    ние по //гребню
EXTPOLAR=(1<<0);                    // Выбрать для EINT0, EINT1 прерыва-
ние по срезу
VICIntSelect=(1<<8);                // Включить FIQ для EINT0, EINT1
прерываний
VICIntEnable=(1<<16);               // Разрешить FIQ для EINT0, EINT1
прерываний

PWMPR=(1<<4);                       // Загрузка предделителя ШИМ
PWMPCR=(1<<4)+(1<<2);               // Разрешение выхода PWM1  и PWM2 ,
                                    ШИМ по срезу
                                    // Регистр управления выходами ШИМ
                                    // D0, D1 - Резерв, D2 - Режим ШИМ
                                    для PWM2, //0 - Single 1 - Double
                                    // D2 - Режим ШИМ для PWM3, D3 -
                                    Режим //ШИМ для PWM4, D4 - Режим
                                    ШИМ для PWM5
                                    // D5 - Режим ШИМ для PWM,       D7,
                                    D8 – Резерв
                                    // D9 - Разрешение выхода PWM1, D9 –
                                    Разрешение выхода PWM2, D10 - Раз-
                                    решение выхода //PWM3
                                    // D11- Разрешение выхода PWM4, D12-
                                    //Разрешение выхода PWM5, D13 - Раз-
                                    решение выхода PWM6

PWMMR0=maxkod;                      // Период таймера  в тактах
                                    // Регистр сравнения канала 0 ШИМ
PWMMR1=kod;                         // Значение срабатывания PWM1
PWMMR2=maxkod-kod;                  // Значение срабатывания PWM2
```

```
PWMLER=(1<<0)+(1<<1)+(1<<2);  // Разрешаем теневые регистры-
                              защелки каналов //0 + 1 + 2
                              // Регистр разрешения теневых реги-
                              стров-защелок
                              // Разрешает загрузку каналов
                              PWMMR0-PWMMR6
                              // D0 - Разрешить загрузку MR0, D1 -
                              Разрешить загрузку MR1, D2 - Разре-
                              шить загрузку MR2
                              // D3 - Разрешить загрузку MR3, D4 -
                              Разрешить загрузку MR4, D5 - Разре-
                              шить загрузку MR5
                              // D6 - Разрешить загрузку MR6, нуле-
                              вое значение бита защелкивает послед-
                              нее введенное значение MR
PWMMCR=(1<<2);                // Сброс таймера при совпадении с
PWMMR0
                              // Регистр управления сравнением. Поз-
                              воляет управлять таймером ШИМ и
                              прерываниями.
                              // D0 - Разрешение прерывания от MR0
                              D1 – Разрешение сброса от MR0  D2 -
                              Разрешение стопа
                              // D3 - Разрешение прерывания от MR1
                              D4 - Разрешение сброса от MR1  D5 -
                              Разрешение стопа от MR1
                              // D6 - Разрешение прерывания от MR2
                              D7 - Разрешение сброса от MR2  D8 -
                              Разрешение стопа от MR2
                              // D9 - Разрешение прерывания от MR3
                              D10- Разрешение сброса от MR3  D11-
                              Разрешение стопа от MR3
                              // D12- Разрешение прерывания от MR4
                              D13- Разрешение сброса от MR4  D14-
                              Разрешение стопа от MR4
                              // D15- Разрешение прерывания от MR5
                              D16- Разрешение сброса от MR5  D17-
                              Разрешение стопа от MR5
                              // D18- Разрешение прерывания от MR6
                              D19- Разрешение сброса от MR6  D20-
                              Разрешение стопа от MR6

PWMTCR=(1<<2);                // Сброс TC и предделителя
PWMTCR=(1<<1)+(1<<2);         // Разрешение  TC и предделителя  +
```

```
                                    //разрешение ШИМ
                                    //Регистр управления ШИМ
                                    // D0 - Разрешение таймера ШИМ   D1 -
                                    Сброс ШИМ   D3 - Не исп. D4

while (1) {};                       // Бесконечный цикл
return 0;
}


void FIQ_Handler(void) __fiq
{

unsigned int maxkod;                //Максимальное значение для ШИМ
maxkod=10;                          // Задаем максимальное значение
if  (EXTINT & (1<<0))  {            // Прерывание от кнопки 1
kod = kod -1;                       // Уменьшаем яркость светодиода 1
if (kod<0) kod=0; }                 // Ограничить kod снизу

 if  (EXTINT & (1<<1))  {           // Прерывание от кнопки 2
kod = kod +1;                       // Уменьшаем яркость светодиода 1
if (kod>maxkod) kod=maxkod;}        // Ограничить kod сверху
PWMMR1=kod;                         // Значение срабатывания PWM1
PWMMR2=maxkod-kod;                  // Значение срабатывания PWM2
PWMLER=(1<<2)+(1<<0);               // Разрешаем теневые регистры-
защелки IOSET1=kod<<16;             // Вывод кода ШИМ на индикатор
IOCLR1=(kod^(1<<4))<<16;
EXTINT=(1<<2);                      // Сбросить флаги внешнего прерыва-
                                    ния
```

Литература

УДК 681.3.06 (075.8)

ББК 22. 151. 3
Б 12

Рецензенти:

?

Рекомендовано
Міністерством освіти і науки України
(лист № 1.4/18-Г-1083 від)

Донченко Е. И.
Б 12 Контроллеры и их програмное обеспечение : конспект лекций / С. О.Донченко Е. И. . – Краматорськ : ДДМА 2018. – 97 с.
ISBN

УДК 681.3.06
(075.8)
ББК 22.151.3

ISBN ?