Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Кафедра «Промышленная электроника»

Ю. В. Крышнев, А. Г. Баранов, А. С. Храмов

ЦИФРОВЫЕ СИГНАЛЬНЫЕ ПРОЦЕССОРЫ

ЛАБОРАТОРНЫЙ ПРАКТИКУМ
по одноименному курсу для студентов специальности 1-36 04 02 «Промышленная электроника» дневной и заочной форм обучения
В 4 частях
Часть 1

Гомель 2010

УДК 621.382.049.77(075.8)
ББК 32.859я73
К85

Рекомендовано научно-методическим советом
факультета автоматизированных и информационных систем
ГГТУ им. П. О. Сухого
(*протокол № 7 от 19.02.2008 г.*)

Рецензент: зав. каф. «Информационные технологии» ГГТУ им. П. О. Сухого
канд. техн. наук *К. С. Курочка*

Крышнев, Ю. В.

К85 Цифровые сигнальные процессоры : лаборатор. практикум по одноим. курсу для студентов специальности 1-36 04 02 «Промышленная электроника» днев. и заоч. форм обучения. В 4 ч. Ч. 1 / Ю. В. Крышнев, А. Г. Баранов, А. С. Храмов. – Гомель : ГГТУ им. П. О. Сухого, 2010. – 48 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: http://lib.gstu.local. – Загл. с титул. экрана.

Содержит необходимые сведения для освоения теоретического материала и практического закрепления знаний по курсу «Цифровые сигнальные процессоры». Включает три лабораторные работы для изучения архитектуры, периферии и программирования цифровых сигнальных процессоров TMS320F2812, а также среды разработки Code Composer Studio.

Для студентов специальности 1-36 04 02 «Промышленная электроника» дневной и заочной форм обучения.

УДК 621.382.049.77(075.8)
ББК 32.859я73

Лабораторная работа № 1
Аппаратные и программные средства отладки eZDSP F2812 и Code Composer Studio

Цель работы: ознакомиться с аппаратной и программной частью лабораторного оборудования, научиться создавать и отлаживать простейшие программы для цифрового сигнального процессора TMS320F2812.

Теоретические сведения

Аппаратная часть стенда состоит из платы эмулятора и платы расширения, а программная – из среды разработки Code Composer Studio.

На рис. 1.1 представлена плата эмулятора eZDSP320F2812 с платой расширения. На плате эмулятора размещен сигнальный процессор TMS320F2812 и схема, обеспечивающая его работу – стабилизатор напряжения (+3.3В для периферии и +1.9В для ядра), схема JTAG-эмулятора, внешнее ОЗУ. Плата эмулятора соединена с платой расширения Zwickau Adapterboard, расположенной под ней (рис. 1.1). На плате расширения располагаются светодиоды, переключатели, кнопки, потенциометры, ЦАП, микросхема FLASH-памяти, датчик температуры, микросхемы интерфейсов RS-232 и CAN, звуковой пьезоэлектрический преобразователь. Связь эмулятора и ПЭВМ осуществляется через LPT-порт.

Code Composer Studio (CCS) – интегрированная среда разработки для цифровых сигнальных процессоров фирмы Texas Instruments. После загрузки программы открывается окно, разделенное на две вертикальные части. В левой части (узкой) отображается окно проекта, в правой (широкой) – рабочая область (рис. 1.2). Любой проект (Project), кроме файла с выполняемой программой на языке Си или Ассемблер, содержит командные файлы, библиотеки, линкеры. Данные файлы необходимы для преобразования исходного текста программ в машинный код, который затем загружается в процессор. Файл управления линкером сопоставляет логические сектора (область данных, программ) и физическую память сигнального процессора.

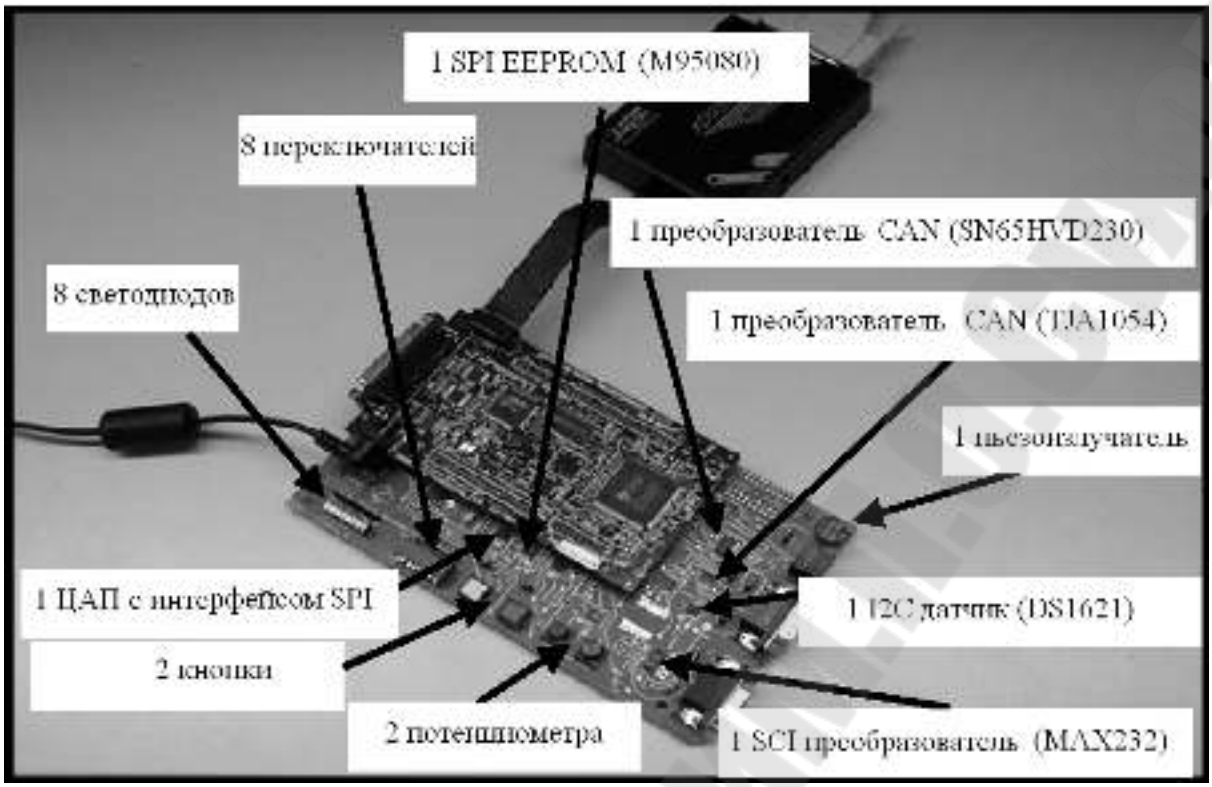


Рис. 1.1. Плата эмулятора eZDSP и плата расширения Zwickau Adapterboard

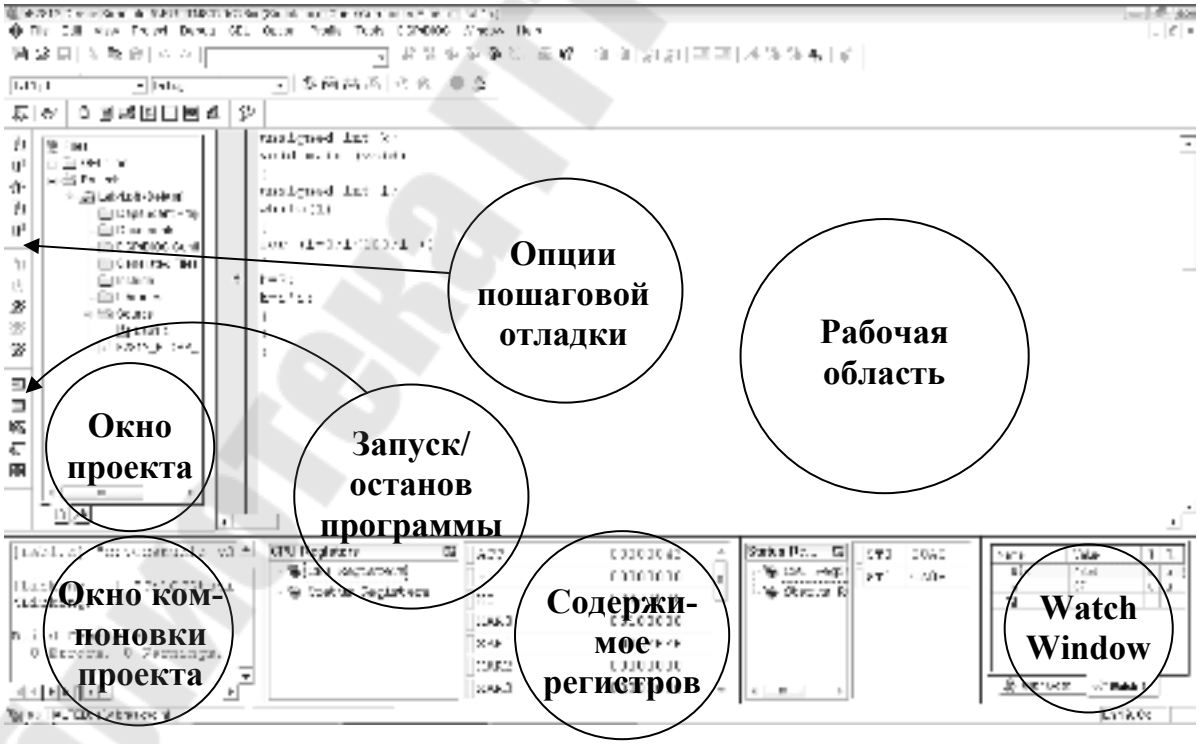


Рис. 1.2. Вид рабочего окна Code Composer Studio

Процедура линковки объединяет один и более объектные файлы (*.obj) в выходной файл (*.out), который содержит машинные коды, адреса и отладочную информацию. Проект может содержать более одного файла программ и подпрограмм, причем допускается написание программ и их частей на различных языках (Си, Си++, Макроассемблер). Все это позволяет максимально эффективно использовать имеющиеся наработки в последующих проектах. Несколько проектов могут объединяться в рабочей области, но компилироваться и отлаживаться будет только один – текущий проект. При выполнении лабораторных работ не рекомендуется открывать сразу несколько проектов.

Ход работы

1. Создание нового проекта.

Через вкладку меню Project → New создаем новый проект. В поле Project Name записываем название проекта «Lab1». В поле Location указывается путь, по которому будет находиться проект – E:\DspUser\Lab1. В поле Project Type выбираем Executable (.out), а в поле Target – TMS320C28XX. В случае если плата эмулятора программно не подключена к ПЭВМ, осуществляем подключение через вкладку меню Debug → Connect.

2. Создание файла программы.

Через вкладку меню File → New → Sourse File создаем файл. Содержимое файла отображается в рабочей области программы. Далее заносим (копируем) в него тестовую программу, представленную на рис. 1.3. Затем необходимо сохранить в папке для проекта, выбранной в п. 1, написанную программу под именем «lab1.c» через вкладку меню File → Save as: «lab1.c».

```
unsigned int k;
void main (void)
{
unsigned int i;
while(1)
{
for (i=0;i<100;i++)
k=i*i;
}
}
```

Рис. 1.3. Тестовая программа

3. Добавление файлов в проект.

Сохраненный файл еще не является частью проекта и при компиляции проекта не будет учтен. Его необходимо добавить в проект через вкладки меню Project → Add files to Project, где указывается имя файла: «lab1.c». После этого имя данного файла появится в разделе «Source» окна проекта. Кроме файла программы, через вкладку меню Project → Add files to Project необходимо добавить в проект файл управления линкером и библиотеки:

C:\tidcs\c28\dsp281x\v100\DSP281x_common\cmd\F2812_EzDSP_RAM_lnk.cmd
C:\CCStudio_v3.3\C2000\cgtools\lib\rts2800_ml.lib

Рекомендуется здесь и далее в аналогичных случаях для исключения ошибок полные пути к файлам копировать в диалоговое окно «Add files to Project».

4. Компиляция программы.

Компилируя программу, мы проверяем её на наличие синтаксических ошибок. Для этого выбираем вкладку меню Project → Compile File (горячие клавиши Ctrl+F7) или иконку  . В случае удачной компиляции в статусной строке будет выдано сообщение об отсутствии ошибок:

«Compile Complete,
0 Errors, 0 Warnings, 0 Remarks.»

5. Добавление библиотек и создание стека.

Подключаем Си-библиотеки:
Project → Build Options → Linker → Libraries → Search Path:
C:\CCStudio_v3.3\C2000\cgtools\lib
Project → Build Options → Linker → Libraries → Search Path Linker → Incl. Libraries: rts2800_ml.lib

Задаем глубину стека 0x400:
Project → Build Options → Linker → Basic → Stack Size (-stack): 0x400 (здесь и далее в формате записи числовых значений префикс «0x» означает hex-формат).

6. Компоновка и загрузка выходного файла в отладочный модуль eZDSP.

Для компоновки выбираем Project → Build (горячая клавиша F7) или  . Открывается окно, в котором указывается наличие или отсутствие ошибок, предупреждений и замечаний. Результатом компоновки будет файл в hex-коде, содержащий коды программ и необходимую отладочную информацию. Далее необходимо загрузить созданный файл в эмулятор-отладчик через вкладку меню: File→Load Program→Debug\lab1.out. После этого тестовая программа загрузится в память ЦСП, расположенного на плате эмулятора. Функцию загрузки готового out-файла в память ЦСП можно настроить автоматически, включив опцию Load Program After Build через меню Option→Customize→Program/Project/CIO.

Замечания:

1) Имя выходного файла генерируется по **имени проекта**, а не имени файла программы.

2) При модификации программы необходимо **каждый раз** компоновать и загружать программу в память процессора.

7. Сброс сигнального процессора и запуск программы на выполнение.

Для правильного выполнения программы необходимо произвести сброс процессора и его перезапуск. Для этого последовательно выполняются директивы Debug → Reset CPU (горячие клавиши Ctrl+R) и Debug → Restart (горячие клавиши Ctrl+Shift+F5) .

Затем с помощью директивы Debug → Go Main (горячая клавиша Ctrl+M) устанавливаем программный счетчик на точку «`void main (void)`» основной программы. Директива Go Main позволяет выполнить начальную установку процессора, генерируемую компилятором языка Си. Положение программного счетчика отображается желтой стрелкой у левого края рабочей области.

Запуск программы на выполнение может осуществляться в пошаговом и в автоматическом режимах.

Запуск программы в автоматическом режиме осуществляется через вкладку меню Debug → Run (горячая клавиша F5) или с помощью иконки  . При этом в строке статуса внизу рабочей области индицируется зелёная лампочка и появляется надпись Running. Остановить выполнение программы можно через вкладку меню Debug → Halt (горячие клавиши Shift+F5) или  .

Пошаговая отладка осуществляется с помощью Debug → Step Into или горячей клавиши F11. Нажимая функциональную клавишу F11, понаблюдайте за ходом выполнения программы.

8. Просмотр переменных.

Для просмотра переменных используется вкладка меню View → Watch Window или  . В колонке Name задается имя переменной, в нашем случае там могут быть записаны переменные i и k (для добавления переменной можно, выделив ее, воспользоваться в контекстном меню опцией Add to Watch Window). В колонке Value отображаются сами переменные, причем после каждой их модификации вручную или исполняемой программой они выделяются красным цветом в течение одного такта отладки. В колонке Type отображается тип переменной (целый, вещественный и т.д.), а в колонке Radix – задается формат представления чисел (десятичный, двоичный, шестнадцатеричный и пр.). В окне Watch Window также можно просматривать также содержимое всех программно доступных регистров ЦСП. Для этого необходимо навести мышь на область окна, щелкнуть правой кнопкой мыши, из открывшегося списка выбрать «Add Globals to Watch», и поставить отметку напротив той группы регистров ЦСП, которую следует просматривать в ходе выполнения программы.

9. Использование точки останова (Breakpoint).

Точки останова (Breakpoint) в Code Composer Studio используются для удобства отладки программ. Для установки Breakpoint устанавливаем курсор на строку, на которой должно остановиться выполнение программы, щелкаем на  , строка выделяется красной точкой. Запускаем программу (F5) и видим, что её выполнение остановилось на выделенной строке (желтая стрелка). Чтобы снять все Breakpoint, необходимо щелкнуть на иконку  . При достижении точки Breakpoint в автоматическом режиме (Run) обновление переменных в окне Watch происходит дискретно с остановкой программы (далее требуется перезапуск).

Измените исходную программу (рис. 1.3) так, как показано на рис. 1.4.

```
unsigned int k;
void main (void)
{
unsigned int i;
while(1)
{
for (i=0;i<100;i++)
{
k=2;
k=i*i;
}
}
}
```

Рис. 1.4. Измененная тестовая программа

10. Режим Animate.

Режим Animate используется для отслеживания содержимого регистров и переменных. Для его запуска необходимо нажать Debug → Animate (горячие клавиши Shift+F5) или  . Скорость анимации устанавливается в окне Option → Customize → Debug Properties → Animate Speed. Установите скорость анимации, равную 1 сек.

11. Просмотр содержимого регистров.

Содержимое регистров процессора можно отследить, выбрав вкладку меню View → Registers → CPU Register и View → Registers → Status Register. Чтобы изменить значение регистра, необходимо дважды щелкнуть по его содержимому и ввести новое значение.

Задание для самостоятельной работы

1. Установите Breakpoint на строки 7, 9 и 10 измененной тестовой программы, показанной на рис. 1.4. Перезапускайте программу в автоматическом режиме (Run) после каждой остановки в выполнении программы на точках останова. Проследите изменение переменных i и k в окне Watch. Объясните полученные результаты.

2. Не снимая Breakpoint, запустите программу в режиме Animate. Проследите выполнение программы, содержимого окна Watch (переменные i и k). Объясните полученные результаты.

3. Открыв окна CPU Register и Status Register, проследите изменение содержимого регистров при выполнении программы в режиме Animate. Объясните полученные результаты.

4. Не останавливая выполнение программы, откройте окно ассемблированного кода программы, полученного в результате компоновки (Window → Disassembly). Объясните полученные результаты. Вернитесь в окно исходного кода программы, написанной на языке Си (Window → Lab1.c). Остановите выполнение программы. Снимите все Breakpoint.

Содержание отчета:

Отчет должен содержать цель работы, краткие теоретические сведения по теме работы, тексты тестовых программ, результаты их выполнения, выводы.

Контрольные вопросы:

1. Структура платы эмулятора eZDSP320F2812 и платы расширения Zwickau Adapterboard.
2. Среда разработки Code Composer Studio. Создание, компиляция, компоновка и загрузка проекта.
3. Среда разработки Code Composer Studio. Режимы запуска программ.
4. Среда разработки Code Composer Studio. Просмотр переменных.
5. Среда разработки Code Composer Studio. Использование точки останова (Breakpoint).
6. Среда разработки Code Composer Studio. Просмотр содержимого регистров ЦСП.
7. Каким образом осуществить отладку исследуемой программы в пошаговом режиме:
   - без исследования ассемблерного кода;
   - с исследованием ассемблерного кода;
   - с одновременным исследованием исходного текста на Си и соответствующего ему ассемблерного кода.

Лабораторная работа № 2
Изучение карты памяти, структуры цифровых портов ввода/ вывода и системы тактирования ЦСП TMS320F2812

Цель работы: изучить карту памяти, структуру цифровых портов ввода-вывода и системы тактирования ЦСП, научиться выполнять настройку данных модулей, вводить и выводить цифровые сигналы.

Теоретические сведения

Структурная схема ЦСП семейства C28x приведена на рис. 2.1. Память и все периферийные модули объединены системой шин по модифицированной гарвардской архитектуре.

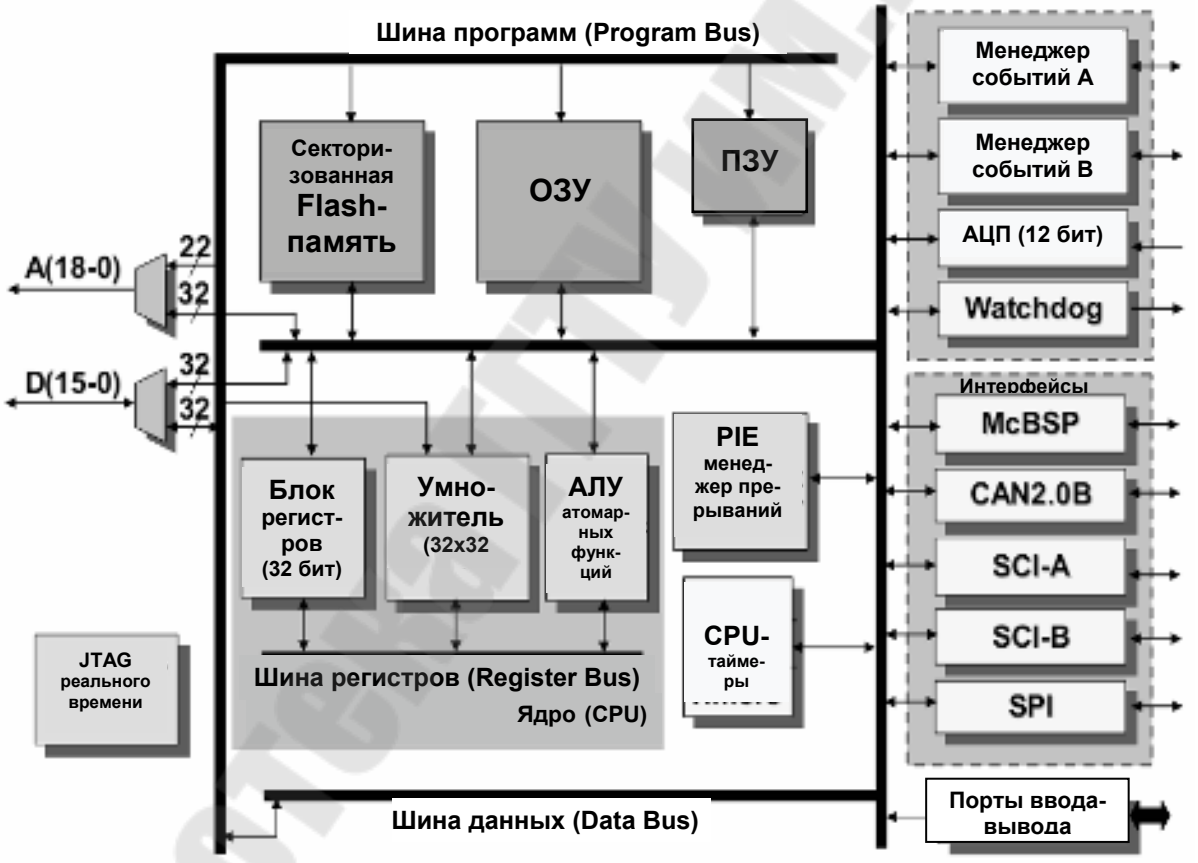


Рис. 2.1. Структурная схема ЦСП семейства C28x

Карта адресного пространства ЦСП C28x показана на рис. 2.2. В качестве памяти данных используется исключительно ОЗУ быстрого доступа (Single Access RAM – SARAM, доступ возможен в течение каждого машинного цикла) общим объёмом 18 Кслов, состоящее из нескольких банков – M0, M1 (2x1K), L0, L1 (2x4K) и H0 (8K). Каждый

банк отображается и на память программ, и на память данных, т.е. эту память можно использовать и в качестве памяти программ, и в качестве памяти данных. В сигнальных процессорах F2812 объём встроенной флэш-памяти составляет 128 Кслов (4 сектора по 8К и 6 секторов по 16К).

В процессоре C28x, помимо центрального процессорного устройства (CPU) имеется внутренняя периферия (АЦП, таймеры, встроенные интерфейсы и пр.), которая также располагается на кристалле. ЦСП семейства C28x содержат регистры модуля центрального процессора (регистры CPU), необходимые для арифметической/логической обработки данных и три сегмента (фрейма) регистров встроенной периферии, предназначенных для управления режимами и хранения данных внутренних периферийных устройств. Эти регистры расположены прямо в адресном пространстве памяти, т.е. доступны не только как регистры с именами, но и как ячейки памяти с определенными адресами (см. рис. 2.2).

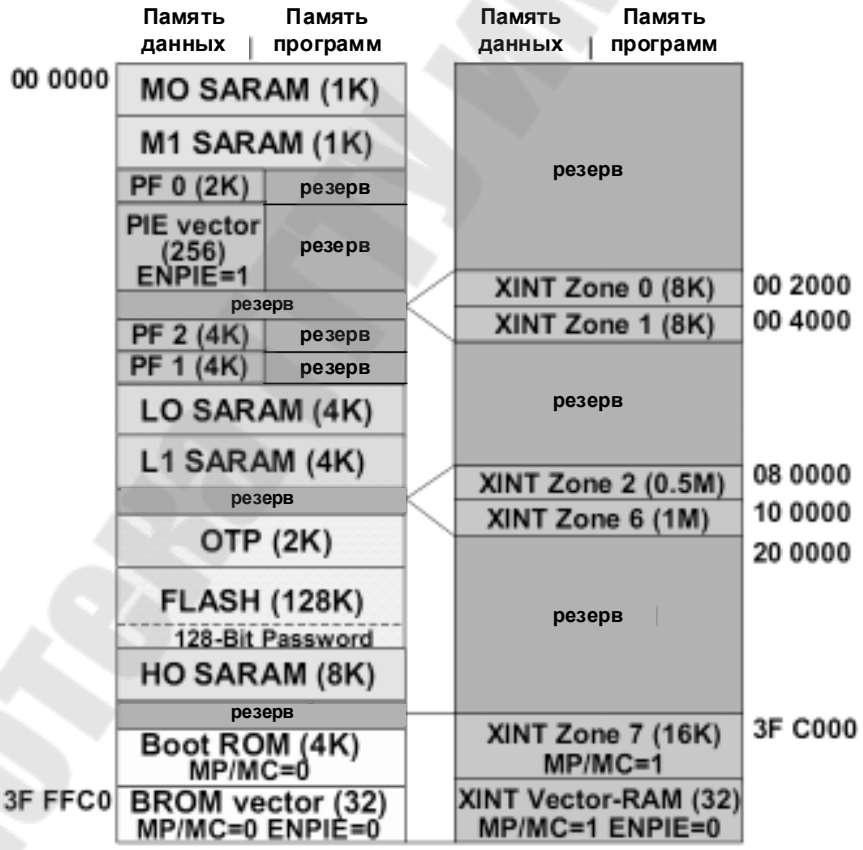


Рис. 2.2. Карта адресного пространства ЦСП семейства C28x

Peripheral Frame 0 (PF0, объем 2K, адреса 0x00 0800… 0x00 0FFF) – включает в себя регистры внешнего интерфейса памяти

XINTF, модуля расширения прерываний PIE, модуля Flash-памяти, модуля таймеров ядра, модуля ключа защиты CSM;

Peripheral Frame 2 (PF2, объем 4K, адреса 0x00 6000…0x00 6FFF) – включает в себя регистры интерфейса eCAN;

Peripheral Frame 1 (PF1, объем 4K, адреса 0x00 7000… 0x00 7FFF) – включает в себя регистры модуля управления системой, модуля ввода-вывода GPIO, модуля менеджеров событий EVA/EVB, модуля последовательного интерфейса McBSP, модуля последовательного интерфейса SCI, модуля последовательного интерфейса SPI, модуля АЦП.

Все программно доступные цифровые выводы сгруппированы в порты GPIOA, GPIOB, GPIOD, GPIOE, GPIOF, GPIOG. GPIO (general purpose input/output) означает «линии ввода/вывода общего назначения».

Периферия имеет выводы, использующиеся для ввода/вывода сигналов. Чтобы не загромождать ЦСП дополнительными выводами, используют мультиплексирование: на одном выводе могут быть реализованы две и более различные функции, которые выбираются программным путем. На рис. 2.3 приведены основные и альтернативные функции портов.

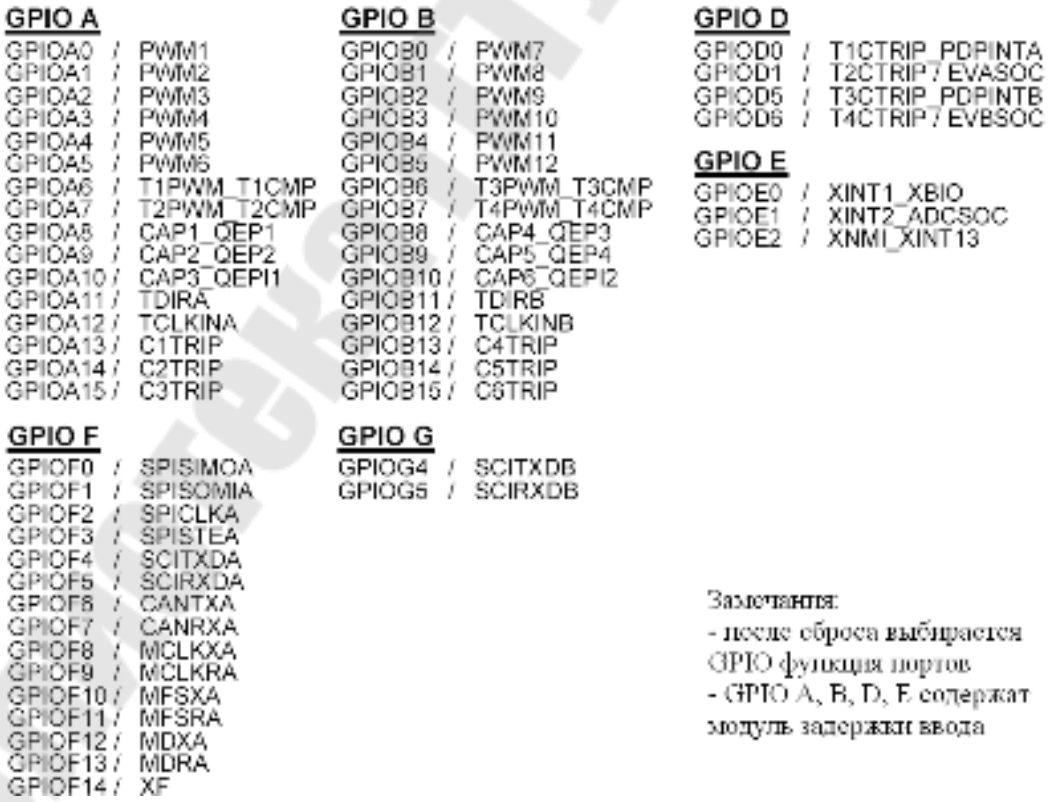


Рис. 2.3. Основные и альтернативные функции портов ввода-вывода F2812

Все 6 портов управляются своими мультиплексирующими регистрами (GPxMUX, здесь и далее в именах регистров, относящихся к портам, x – имя порта). Если бит сброшен в 0, то данный вывод работает как обычная линия порта; установкой бита в 1 выбирается альтернативная функция, т.е. линия порта (pin) подключается к соответствующему периферийному устройству (см. рис. 2.4). К регистрам данных относятся:

- регистры GPxDAT (в них непосредственно хранятся данные порта, которые могут быть изменены записью в данные регистры);
- регистры GPxSET (служат для установки соответствующих разрядов порта);
- регистры GPxCLEAR (служат для сброса соответствующих разрядов порта);
- регистры GPxTOGGLE (служат для переключения соответствующих разрядов порта в альтернативное логическое состояние).

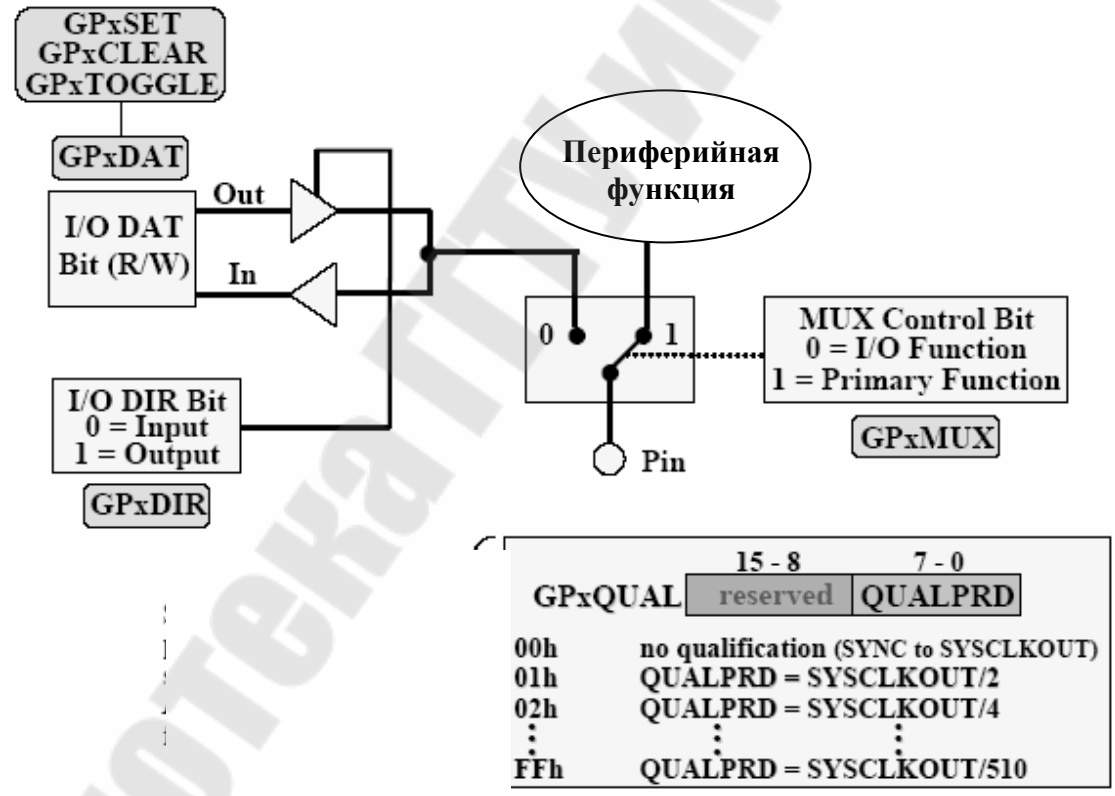
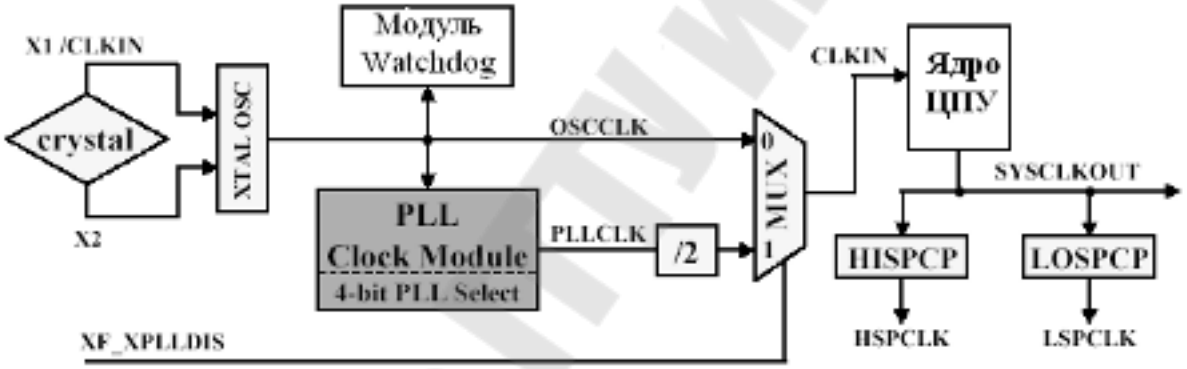


Рис. 2.4. Функциональная схема линии GPIO

Каждый вывод порта может работать на ввод или вывод. Направление передачи данных задается битами в регистрах GPxDIR: 0 – линия работает на ввод, 1 – на вывод данных. При работе портов A, B, D и E на ввод для увеличения помехозащищенности можно на-

строить функцию задержки ввода (Input Qualification feature) установкой битов 7-0 в соответствующем регистре GPxQUAL. После этого импульсы на линиях соответствующих портов, имеющие максимальную длительность от 2 (GPxQUAL=0x01) до 510 (GPxQUAL=0xFF) периодов частоты тактирования ядра SYSCLKOUT, не будут распознаваться ядром процессора.

Перед началом работы необходимо настроить модуль тактирования. Как и многие современные процессоры, ЦСП семейства C28x используют внешний резонатор или генератор с частотой более низкой, чем максимальная тактовая. Это позволяет уменьшить влияние электромагнитных помех, понизить требования к разводке печатной платы и повысить надежность работы схемы. Частота тактового генератора OSCCLK, расположенного на плате эмулятора, составляет 30 МГц. Максимальная тактовая частота процессора 150 МГц получается путем умножения внешней частоты на 5 (OSCCLK*10/2, см. рис. 2.5). Коэффициент деления задается программно в регистре PLLCR.



PLLCR биты 15:4 зарезервированы

DIV3	DIV2	DIV1	DIV0	Частота (CLKIN)
0	0	0	0	OSCCLK x 1 / 2 (no PLL)
0	0	0	1	OSCCLK x 1 / 2
0	0	1	0	OSCCLK x 2 / 2
0	0	1	1	OSCCLK x 3 / 2
0	1	0	0	OSCCLK x 4 / 2
0	1	0	1	OSCCLK x 5 / 2
0	1	1	0	OSCCLK x 6 / 2
0	1	1	1	OSCCLK x 7 / 2
1	0	0	0	OSCCLK x 8 / 2
1	0	0	1	OSCCLK x 9 / 2
1	0	1	0	OSCCLK x 10 / 2

Рис. 2.5. Тактовый модуль и регистр PLLCR

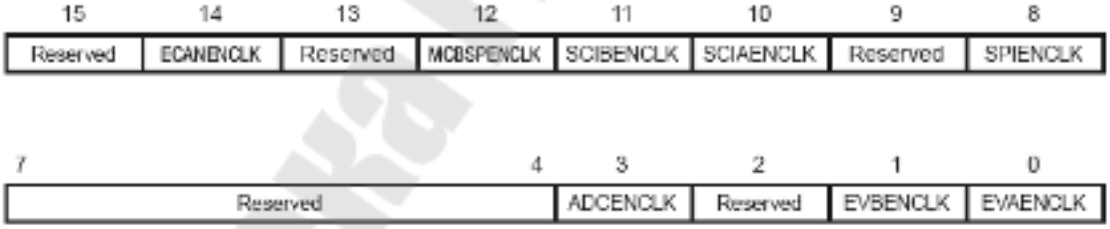
Помимо формирователя тактовой частоты ядра, тактовый модуль содержит предделители: низкоскоростной LOSPCP и высокоскоростной HISPCP (рис. 2.6), которые используются для тактирования периферийных устройств.

Известно, что потребляемая мощность микропроцессорных элементов возрастает пропорционально тактовой частоте. Снижая тактовую частоту, можно снизить энергопотребление и определенного модуля, и всего ЦСП в целом. Коэффициенты деления задаются программно в регистрах LOSPCP и HISPCP.

H/LSPCLK2	H/LSPCLK1	H/LSPCLK0	Тактовая частота периферии
0	0	0	SYSCLKOUT / 1
0	0	1	SYSCLKOUT / 2
0	1	0	SYSCLKOUT / 4
0	1	1	SYSCLKOUT / 6
1	0	0	SYSCLKOUT / 8
1	0	1	SYSCLKOUT / 10
1	1	0	SYSCLKOUT / 12
1	1	1	SYSCLKOUT / 14

Рис. 2.6. Формат регистров HISPCP и LOSPCP

В ЦСП семейства C28x реализована возможность не только изменять, но и полностью запрещать/разрешать тактирование различных периферийных модулей при помощи регистра PCLKCR (рис. 2.7).



EVAENCLK	**Разрешение HSPCLK в EVA**
EVBENCLK	**Разрешение HSPCLK в EVB**
ADCENCLK	**Разрешение HSPCLK в АЦП (ADC)**
SPIAENCLK	**Разрешение LSPCLK в SPI**
SCIAENCLK	**Разрешение LSPCLK в SCI-A**
SCIBENCLK	**Разрешение LSPCLK в SCI-B**
MCBSPENCLK	**Разрешение LSPCLK в McBSP**
ECANENCLK	**Разрешение тактирования eCAN**

Рис. 2.7. Формат регистра PCLKCR: 1 – разрешить тактирование периферийного модуля; 0 – запретить.

Сторожевой таймер (Watchdog timer или WDT, рис. 2.8) предназначен для предотвращения «зависания» ядра ЦСП и реализован на основе суммирующего счетчика WDCNTR, который тактируется через счетчик-делитель от внешнего генератора и при переполнении вырабатывает сигнал прерывания WDINT либо сброса ядра ЦСП WDRST (задается программно). WDT работает независимо от ядра ЦСП и должен сбрасываться программно для предотвращения сброса процессора.

Сторожевой таймер защищен ключом «101», и, в случае записи в биты WDCHK 2–0 регистра WDCR (рис. 2.9) любой иной кодовой комбинации, в следующем машинном цикле происходит генерация выходного сигнала (такого же, как и при переполнении). Для предотвращения сброса ЦСП необходимо периодически программно сбрасывать счетчик WDCNTR при помощи записи последовательности кодов «0x55 + 0xAA» в специальный регистр WDKEY.

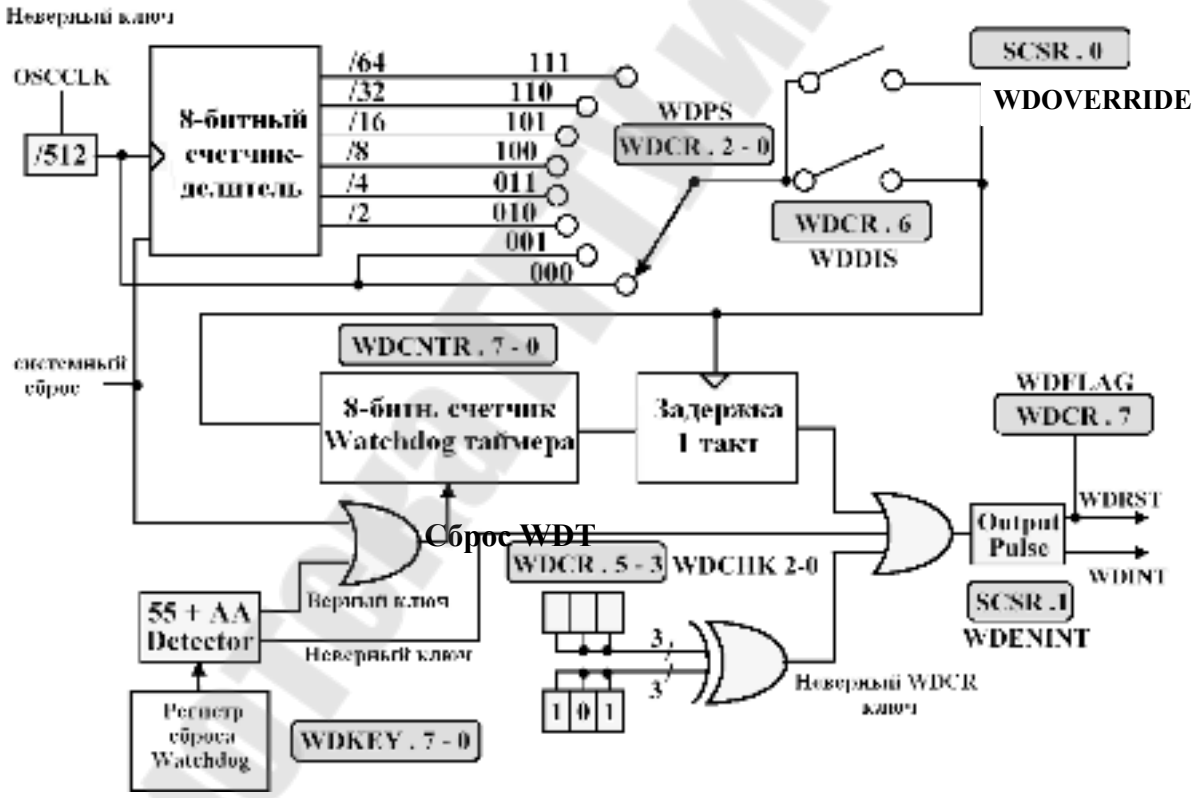


Рис. 2.8. Функциональная схема сторожевого таймера

Рис. 2.9. Формат регистра управления сторожевого таймера WDCR

Бит WDFLAG используется для того, чтобы указать источник сброса: обычный сброс (WDFLAG=0) или сброс по сторожевому таймеру (WDFLAG=1). Биты WDPS 2–0 позволяют выбрать необходимый коэффициент деления частоты для работы сторожевого таймера. Бит WDDIS служит для запрета работы (блокировки) WDT.

Регистр управления и состояния (System Control and Status Register – SCSR, рис. 2.10) управляет сбросом сторожевого таймера. Бит WDINTS отражает текущее состояние сигнала WDINT. Бит WDENINT – выбор режима действия от WDT (сброс или прерывание). Если WDENINT=0 – разрешен сигнал сброса WDRST, если WDENINT=1 – разрешен сигнал прерывания WDINT.

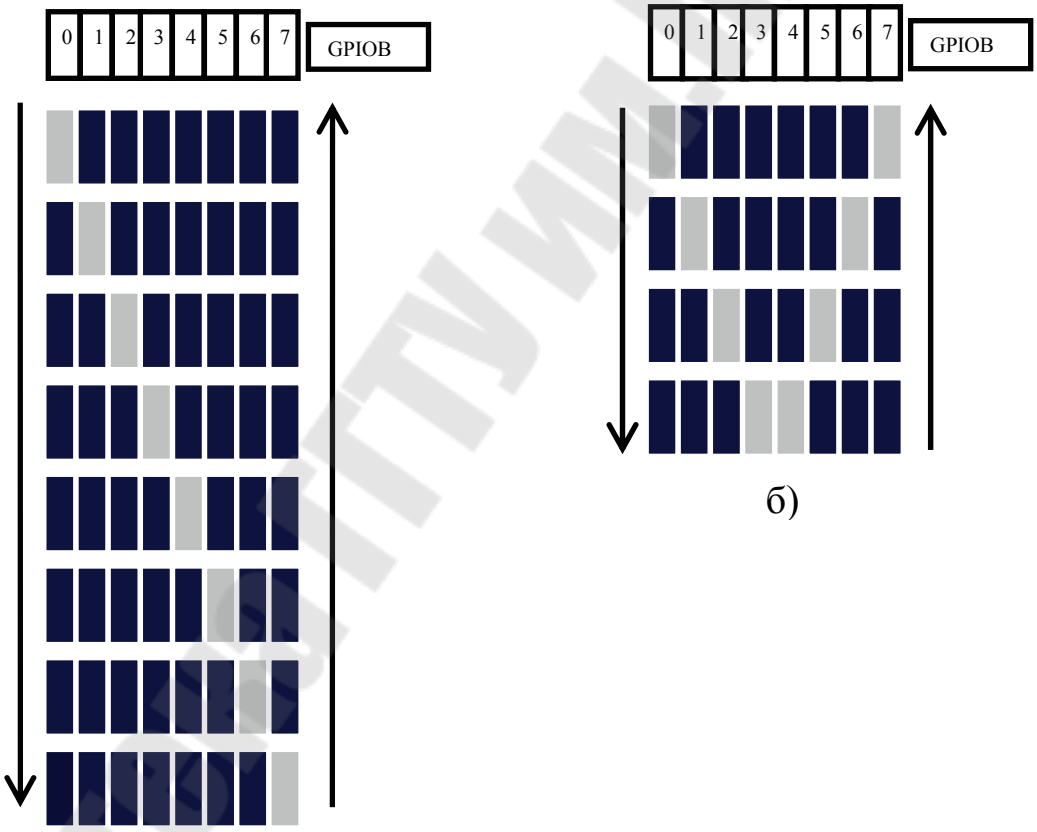
Если бит WDOVERRIDE установлен в 1, то пользователь может изменять состояние сторожевого таймера, т.е. запрещать или разрешать его работу с помощью бита WDDIS в регистре управления WDCR. Особенностью является то, что пользователь может **только сбросить бит WDOVERRIDE, записав в него «1», установить бит программно нельзя**.

Рис. 2.10. Формат регистра управления и состояния SCSR

Ход работы

Часть I

В части I лабораторной работы необходимо разработать программу «бегущие огни», задавая направление движения: сначала – от края к краю (см. рис. 2.11, а), а потом, зажигая по два светодиода, – от периферии к центру (см. рис. 2.11, б). Светодиоды подключены к линиям порта GPIOB7 – GPIOB0 (1 – горит, 0 – погашен), а линии порта GPIOB15 – GPIOB8 соединены с ключами (1 – ключ замкнут, 0 – разомкнут).



а) б)

Рис. 2.11. Направление движения светодиодов: а) – слева направо и наоборот; б) – от периферии к центру и наоборот.

1. Создание нового проекта.

В Code Composer Studio создаем новый проект Lab2.pjt. В поле Project Name записываем название проекта «Lab2». В поле Location указывается путь, по которому будет находиться проект – E:\DspUser\Lab2.

1.1. Для упрощения работы файл под именем «_lab2_.c» с заготовкой текста программы имеется на диске в папке «E:\DspUser\Templates». Знаками «?» в исходном тексте программы отмечены значения, которые необходимо будет заменить на требуемые для правильной работы программы (рис. 2.12). Копируем данный файл в папку проекта E:\DspUser\Lab2 и переименовываем в «Lab2.c». Добавляем в проект и открываем в окне редактора данный файл: Project → Add files to Project.

1.2. Добавляем в проект управляющий файл линкера, командные файлы, библиотеки, необходимые внешние программные модули (рекомендуется для исключения ошибок пути к файлам, указанные ниже, копировать в диалоговое окно «Add files to Project»):

C:\tidcs\c28\dsp281x\v100\DSP281x_common\cmd\F2812_EzDSP_RAM_lnk.cmd
C:\tidcs\c28\dsp281x\v100\DSP281x_headers\cmd\DSP281x_Headers_nonBIOS.cmd
C:\CCStudio_v3.3\C2000\cgtools\lib\rts2800_ml.lib
C:\tidcs\c28\dsp281x\v100\DSP281x_headers\source\DSP281x_GlobalVariableDefs.c

2. Настройка параметров проекта.

2.1. Включаем в проект заголовочные файлы, для этого выбираем Project → Build Options, в закладке Compiler выбираем Preprocessor и в поле Include Search Path (-i) вводим:

C:\tidcs\C28\dsp281x\v100\DSP281x_headers\include;..\include

2.2. Добавление библиотек и создание стека.

Подключаем Си-библиотеки:

Project → Build Options → Linker → Libraries → Search Path: C:\CCStudio_v3.3\C2000\cgtools\lib

Project → Build Options → Linker → Libraries → Search Path Linker → Incl. Libraries: rts2800_ml.lib

Задаем глубину стека 0x400:

Project → Build Options → Linker → Basic → Stack Size (-stack): 0x400

```
//###########################################################################
//    Имя файла:  _Lab2_.c
//
//    Описание:   С помощью 8 светодиодов, подключенных к линиям
//    порта GPIOB0 - GPIOB7, формируются "бегущие огни".
//    Направление движения справа - налево и наоборот
//###########################################################################

#include "DSP281x_Device.h" // Включение заголовочного файла

void delay_loop(long);
void Gpio_select(void);
void InitSystem(void);

void main(void)
{
     unsigned int i;
```

```
      unsigned int LED[8]= {0x0001,0x0002,0x0004,0x0008,
                            0x0010,0x0020,0x0040,0x0080};

      InitSystem();           // Инициализация регистров ЦСП
      Gpio_select();          // Инициализация линий ввода/вывода

while(1)
      {
          for(i=0;i<14;i++)
            {
       if(i<7)      GpioDataRegs.GPBDAT.all = LED[i];
       else   GpioDataRegs.GPBDAT.all = LED[14-i];
       delay_loop(?);
        }
      }
}

//###########################################################################
//    Подпрограмма:     delay_loop
//
//    Описание:   Формирование временной задержки горения светодиодов
//###########################################################################

void delay_loop(long end)
{
      long i;
      for (i = 0; i < end; i++);
      EALLOW; // Сброс сторожевого таймера
      //SysCtrlRegs.WDKEY = 0x?;
      //SysCtrlRegs.WDKEY = 0x?;
      EDIS;
}

//###########################################################################
//    Подпрограмма:     Gpio_select
//
//    Описание:   Настройка линий порта GPIO B15-8 на ввод, а линий
//    B7-0  на вывод. Настройка всех линий портов A, D, F, E, G на
//    ввод. Запрещение работы входного ограничителя
//###########################################################################

void Gpio_select(void)
{
      EALLOW;
    GpioMuxRegs.GPAMUX.all = 0x?; // Настройка линий ввода/вывода на
    GpioMuxRegs.GPBMUX.all = 0x?; // работу в качестве портов
    GpioMuxRegs.GPDMUX.all = 0x?;
    GpioMuxRegs.GPFMUX.all = 0x?;
    GpioMuxRegs.GPEMUX.all = 0x?;
    GpioMuxRegs.GPGMUX.all = 0x?;
    GpioMuxRegs.GPADIR.all = 0x?; // Настройка портов A, D, E, F, G
                                  // на ввод
    GpioMuxRegs.GPBDIR.all = 0x?; // Настройка линий 15-8 на ввод,
    GpioMuxRegs.GPDDIR.all = 0x?; // а линий 7-0 на вывод
    GpioMuxRegs.GPEDIR.all = 0x?;
    GpioMuxRegs.GPFDIR.all = 0x?;
    GpioMuxRegs.GPGDIR.all = 0x?;

    GpioMuxRegs.GPAQUAL.all = 0x?; // Запрещение входного ограничителя
    GpioMuxRegs.GPBQUAL.all = 0x?;
    GpioMuxRegs.GPDQUAL.all = 0x?;
    GpioMuxRegs.GPEQUAL.all = 0x?;
    EDIS;
}
```

6

7

5

```
//###########################################################################
//     Подпрограмма:     InitSystem
//
//     Описание:   Настройка сторожевого таймера на работу,
//           предделитель = 1. Выработка сброса сторожевым
//           таймером. Внутренняя частота работы ЦСП 150 МГц.
//           Запись в предделитель высокоскоростного таймера
//           коэффициента деления 2, а в предделитель
//           низкоскоростного таймера - 4. Запрещение работы
//           периферийных устройств
//###########################################################################

void InitSystem(void)
{
     EALLOW;
     SysCtrlRegs.WDCR= 0x?;         // Разрешение работы сторожевого
                                    // таймера, предделитель = 64
                                    // или запрещение работы  сторо-
                                    // жевого таймера, предделитель = 1

     SysCtrlRegs.SCSR = ?;          // Конфигурирование сброса ЦСП от WDT

     SysCtrlRegs.PLLCR.bit.DIV = ?;// Настройка блока умножения частоты
     SysCtrlRegs.HISPCP.all = 0x?;// Задание значений высокоскоростного
     SysCtrlRegs.LOSPCP.all = 0x?; // и низкоскоростного предделителей

     SysCtrlRegs.PCLKCR.bit.EVAENCLK=?;  // Запрещение работы
     SysCtrlRegs.PCLKCR.bit.EVBENCLK=?;  // периферийных устройств
     SysCtrlRegs.PCLKCR.bit.SCIAENCLK=?;
     SysCtrlRegs.PCLKCR.bit.SCIBENCLK=?;
     SysCtrlRegs.PCLKCR.bit.MCBSPENCLK=?;
     SysCtrlRegs.PCLKCR.bit.SPIENCLK=?;
     SysCtrlRegs.PCLKCR.bit.ECANENCLK=?;
     SysCtrlRegs.PCLKCR.bit.ADCENCLK=?;
     EDIS;
}
```

Рис. 2.12. Заготовка текста программы «бегущие огни»

3. Инициализация системы.

Открываем файл Lab2.c и переходим к подпрограмме «`InitSystem()`».

3.1. Присваивая необходимое значение регистру WDCR (см. рис. 2.9), запрещаем работу сторожевого таймера (WDDIS=0), сбрасываем бит WDFLAG, задаем коэффициент деления частоты тактирования Watchdog-таймера, равный 1 (WDPS0, WDPS1, WDPS2=0). Указанные значения нужно внести в область 1 программы (см. рис. 2.12).

3.2. Программируем регистр SCSR (см. рис. 2.10) на сброс процессора при переполнении Watchdog-таймера. В бит WDENINT записываем «0», в бит WDOVERRIDE также записываем «0» для того, чтобы оставить его в единичном состоянии (см. область 2 на рис. 2.12).

3.3. В регистр PLLCR (см. рис. 2.5) заносим код, необходимый для преобразования частоты кварцевого резонатора 30 МГц во внутреннюю частоту SYSCLKOUT работы процессора 150 МГц (см. область 3 на рис. 2.12).

3.4. Заносим в высокоскоростной предделитель HISPCP (см. рис. 2.6) коэффициент деления 2, а в низкоскоростной (LOSPCP) – 4 (см. область 3 на рис. 2.12).

3.5. Установкой-сбросом бит в регистре PCLKCR (см. рис. 2.7) запрещаем работу всех периферийных устройств (см. область 4 на рис. 2.12).

4. Настройка портов.

Переходим к подпрограмме «`Gpio_select()`».

4.1. Установкой-сбросом бит в регистрах GPxMUX настраиваем все выводы на работу в качестве портов (см. область 5 на рис. 2.12).

4.2. Установкой-сбросом бит в регистрах GPxDIR настраиваем все линии портов A, D, E, F, G на ввод (см. область 5 на рис. 2.12). В порту GPIOB настраиваем линии порта GPIOB15 – GPIOB8 на ввод, а GPIOB7 – GPIOB0 на вывод (см. область 5 на рис. 2.12).

4.3. Сбрасываем все биты регистров GPxQUAL портов A, B, D, E в ноль (см. область 5 на рис. 2.12).

5. Расчет и задание временной задержки

В строке «`delay_loop()`» (см. область 6 на рис. 2.12) в скобках указываем число n – параметр задержки. Длительность задержки будет пропорциональна данному числу и рассчитывается по формуле (в секундах):

$$t_{зд} = \frac{6 \cdot n}{150 \cdot 10^6}.$$

6. Компиляция, компоновка и загрузка выходного файла в отладочный модуль ЦСП.

6.1. Компилируем программу: Project → Compile File. При наличии ошибок, исправляем их.

6.2. Компонуем проект: Project → Build. При наличии ошибок, исправляем их.

6.3. Загружаем выходной файл: File → Load Program → Debug\ lab2.out (в случае, если данная функция сконфигурирована для выполнения автоматически, выполнять данный пункт не нужно).

7. Тестирование программы.

7.1. Сбрасываем ЦСП: Debug → Reset CPU, Debug → Restart.

7.2. Устанавливаем программный счетчик на точку «`void main (void)`» основной программы: Debug → Go main.

7.3. Запускаем программу: Debug → Run. Проверяем правильность работы, наблюдая за светодиодами.

Имеется возможность отслеживать правильность выполнения логики программы, просматривая в отдельном окне, как изменяется содержимое выводов порта GPIOB. Для этого необходимо:

- остановить выполнение программы;
- установить точку останова на строку программы, отмеченную знаком «●»;
- выделить мышью в любом месте текста программы название «GpioDataRegs.GPBDAT», нажать правую кнопку мыши и в открывшемся окне выбрать «Add to Watch Window»;
- щелкнуть по значку «+» напротив имени «GpioDataRegs.GPBDAT» в окне Watch Window, далее выбрать «bit», после чего в данном окне будут выведены в столбик двоичные состояния разрядов порта GPIOB (рис. 2.13);
- исключить из программы (закомментировать) задержку, вставив символы «//» в начало строки `delay_loop(?);`
- сохранить проект (Project → Save), выполнить компиляцию и линковку измененного проекта;
- запустив программу в режиме «Animate», наблюдать последовательность изменения логических уровней на линиях порта GPIOB.

8. Работа со сторожевым таймером.

8.1. Останавливаем выполнение программы.

8.2. В подпрограмме «`InitSystem()`» изменением содержимого регистра WDCR разрешаем работу сторожевого таймера (см. область 1 на рис. 2.12). В битовом поле WDPS этого регистра задаем коэффициент деления программного предделителя равным 64 (см. рис. 2.8, 2.9).

8.3. В подпрограмме временной задержки «`delay_loop()`» удаляем символы «//» перед двумя строками программы, в которых должна производиться запись в регистр WDKEY (см. область 7 на рис. 2.12), и записываем кодовую комбинацию для сброса Watchdog-таймера: `SysCtrlRegs.WDKEY = 0x55`, `SysCtrlRegs.WDKEY = 0xAA`.

Расчетное время до формирования импульса сброса /WDRST от Watchdog-таймера в выбранном режиме составит:

$$t_{WDRST} = \frac{512 \cdot 64 \cdot 256}{30 \cdot 10^6} = 0{,}28\ c.$$

Для того чтобы сброс WDT происходил раньше (т.е. сброса ЦСП не происходило), нужно обеспечить соотношение $t_{зд} \le t_{WDRST}$.

8.4. Компилируем, компонуем проект и тестируем программу.

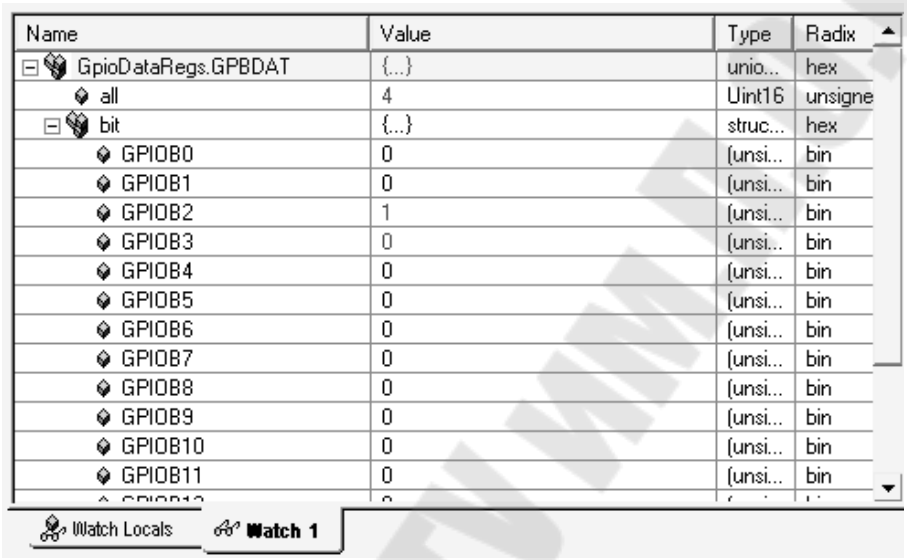


Рис. 2.13. Просмотр состояния двоичных разрядов порта GPIOB в окне Watch Window

9. Работа с двумя светодиодами.

9.1. Копируем содержимое файла «Lab2.c» в новый файл, который называем «Lab2a.c».

9.2. Преобразуем программу Lab2a.c так, чтобы одновременно зажигались два светодиода в направлении от периферии к центру (см. рис. 2.11, б).

9.3. Добавляем в проект «Lab2a.c» и удаляем «Lab2.c» (щелкаем правой клавишей мышки и выбираем Remove from project).

9.4. Компилируем, компонуем проект и тестируем программу.

Часть II

В части II лабораторной работы необходимо положение ключей отражать на светодиодах (ключ замкнут – светодиод горит, разомкнут – погашен), а также положением ключей задавать длительность свечения светодиодов.

1. Отображение состояния ключей на светодиодах.

1.1. Копируем содержимое файла «Lab2a.c» в новый файл, который называем «Lab2b.c».

1.2. Преобразуем программу Lab2b.c. Состояние (двоичный код), установленный ключами, должен индицироваться на светодиодах. С учетом того, что для подключения и светодиодов, и ключей используется один и тот же порт (GPIOB), состояние ключей можно индицировать, введя в основную программу команду циклического сдвига содержимого порта GPIOB на 8 бит:

```
GpioDataRegs.GPBDAT.all = GpioDataRegs.GPBDAT.all >> 8
```

Удаляем таблицу состояния светодиодов, а подпрограммы «`InitSystem()`» и «`Gpio_select()`» оставляем без изменений.

1.3. Добавляем в проект «Lab2b.c» и удаляем «Lab2a.c».

1.4. Компилируем, компонуем проект и тестируем программу.

2. Формирование длительности свечения светодиодов в зависимости от состояния ключей.

2.1. Копируем содержимое файла «Lab2.c» в новый файл, который называем «Lab2c.c».

2.2. Преобразуем программу Lab2c.c. Задаем длительность задержки при переключении светодиодов согласно рис. 2.11, а, равную (0,1*N+0,001) сек, где N – код, заданный переключателями на линиях B15–B8. Подпрограммы «`InitSystem()`» и «`Gpio_select()`» оставляем без изменений.

2.3. Добавляем в проект «Lab2c.c» и удаляем «Lab2b.c».

2.4. Компилируем, компонуем проект и тестируем программу.

Содержание отчета:

Отчет должен содержать цель работы, краткие теоретические сведения по теме работы, тексты исследуемых программ, результаты их выполнения, выводы.

Контрольные вопросы:

1. Структурная схема ЦСП семейства C28x.

2. Карта адресного пространства ЦСП С28х.
3. Структура портов ввода/вывода, режимы работы, регистры управления.
4. Вывод простейших управляющих сигналов через порты TMS320F2812.
5. Система тактирования TMS320F2812. Структура, особенности, назначение регистров высокоскоростного и низкоскоростного предделителей.
6. Watchdog timer: назначение, принцип действия, особенности.
7. Какие изменения необходимо внести в исходный текст, чтобы программа «бегущие огни» запускалась по нажатию кнопки, присоединенной к линии порта GPIOD1, а останавливалась – по нажатию кнопки, присоединенной к линии порта GPIOD6 («0» – соответствующая кнопка нажата, «1» – кнопка отжата)? В исходном состоянии обе кнопки отжаты.
8. Модифицируйте исходную программу таким образом, чтобы происходило движение одного «погашенного» светодиода среди остальных «горящих».

Лабораторная работа № 3
Исследование системы прерываний и таймеров ядра ЦСП семейства С28х

Цель работы: изучить систему прерываний процессоров семейства С28х, а также таймеры ядра ЦСП. Научиться создавать программы, использующие прерывания.

Теоретические сведения

Система прерываний ядра процессора С28х содержит 16 линий прерываний (рис. 3.1). Два из них – немаскируемые (RS, NMI). Пользователь не может запретить данные прерывания. Остальные 14 прерываний – маскируемые, т.е. пользователь может разрешить/запретить прерывания от них программно соответственно установкой /сбросом соответствующих бит в регистре IER (Interrupt Enable Register). Регистр IFR (Interrupt Flag Register) – регистр флагов прерываний. При обнаружении прерывания соответствующий бит регистра IFR защелкивается в единичном состоянии, а после обслуживания прерывания – сбрасывается. Так же, когда аппаратное прерывание обслужено, или когда выполнена инструкция INTR, сбрасывается соответствующий бит регистра IER.

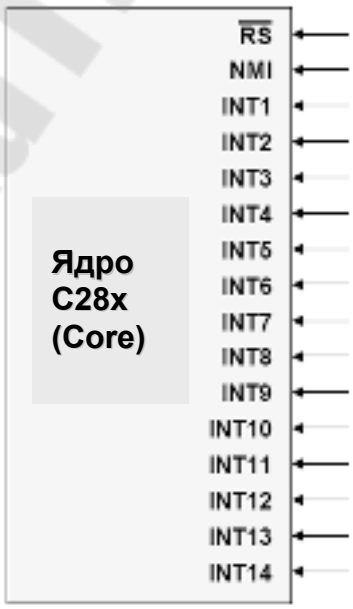


Рис. 3.1. Линии прерываний ядра С28х

Общая структура прерываний ядра C28x показана на рис. 3.2, а форматы регистров IER и IFR – на рис. 3.3. Следует отметить, что программная установка одного из битов в регистре IFR приведет к обработке данного прерывания ядра таким же образом, как и в случае возникновения соответствующего прерывания. INTM – младший бит в статусном регистре ST1, служит для общего разрешения/запрета маскируемых прерываний ядра.

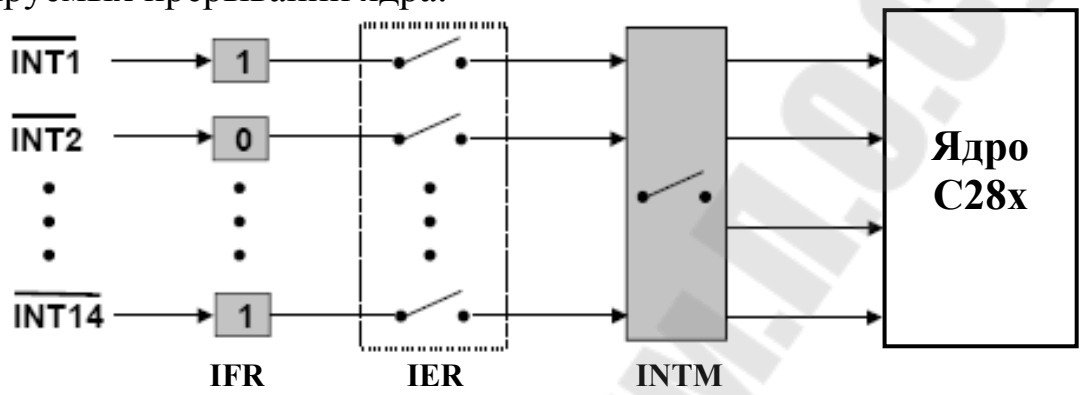
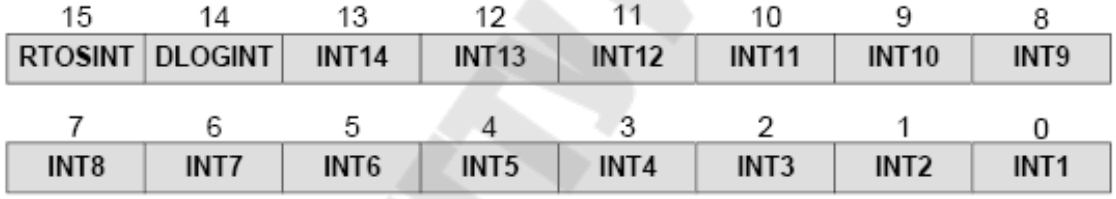
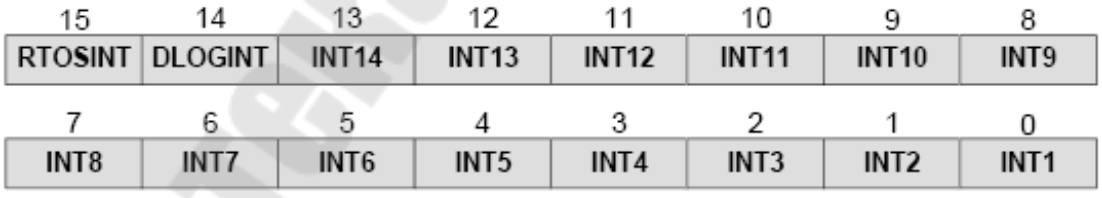


Рис. 3.2. Общая структура прерываний ядра C28x



Прерывание разрешено: $\mathbf{IER_{Bit}=1}$
Прерывание запрещено: $\mathbf{IER_{Bit}=0}$



Прерывание ожидает разрешения: $\mathbf{IFR_{Bit}=1}$
Прерывание отсутствует: $\mathbf{IFR_{Bit}=0}$

Рис. 3.3. Форматы регистров IER и IFR

Все 16 прерываний однозначно связаны в памяти с таблицей векторов прерываний ядра BROM vector (см. рис. 2.2), содержащей 22-битные адреса (на каждое прерывание – 16 бит в ячейке с млад-

шим адресом и 6 бит в ячейке со старшим адресом), по которым должны располагаться стартовые адреса подпрограмм обработки соответствующих прерываний. Всего данная область памяти содержит 32 таких вектора, т.к. к 16 прерываниям ядра добавлены прерывания DLOGINT, RTOSINT, Illegal (недопустимая инструкция), 12 программных прерываний USER1…USER12 и один резервный вектор (Reserved).

Подача активного сигнала на вход сброса (на вывод /RS) вызовет его сброс и перезапуск программы с начального адреса. Сброс отличается от остальных прерываний тем, что программа затем не возвращается в исходную точку, а регистры сбрасываются в начальное состояние. Сброс может происходить как от внешнего источника, так и от сторожевого таймера (WDT). Сброс внешних схем при сбросе ядра процессора от WDT осуществляется через вывод /RS, который является двунаправленным.

Особенностью ЦСП семейства C28x является возможность запуска программ как из внутренней памяти (микроконтроллерный режим), так и из внешней (микропроцессорный режим). Режим определяется состоянием вывода XMP/MC (см. рис. 2.2). Соответственно, после сброса программа переходит либо к начальному адресу 0x3F FFC0 внутренней памяти (при XMP/MC=0), либо к тому же адресу внешней памяти (при XMP/MC=1), а режим запоминается с помощью флага XMP/MC в регистре XINTCNF2, который может быть впоследствии обработан программно.

После сброса в режиме микроконтроллера запускается служебная программа Bootloader, которая анализирует выводы порта GPIOF (GPIOF2, GPIOF3, GPIOF4 и GPIOF12) и, исходя из комбинации сигналов на них, выполняет один из переходов, перечисленных в табл. 3.1 и условно показанных на рис 3.4.

В отладочном стенде ezDSP2812, используемом в лабораторных работах, программа загружается в H0 SARAM (ОЗУ), а прочие возможные режимы загрузки могут быть заданы при помощи переключателей (джамперов). Следует отметить, что память программ и данных имеют единое адресное пространство, программа может выполняться как из ОЗУ, так и из FLASH, или ПЗУ (OTP).

Таблица 3.1. Режимы запуска программы Bootloader

Выводы GPIO				Режим запуска
F4	F12	F3	F2	
1	x	x	x	Передать управление FLASH-памяти по адресу 0x3F 7FF6
0	0	1	0	Передать управление H0 SARAM-памяти по адресу 0x3F 8000
0	0	0	1	Передать управление OTP-памяти по адресу 0x3D 7800
0	1	x	x	Загрузить программу из внешнего EEPROM во внутреннюю память через SPI-порт
0	0	1	1	Загрузить программу во внутреннюю память через SCI-A порт
0	0	0	0	Загрузить программу во внутреннюю память через параллельный порт GPIOB

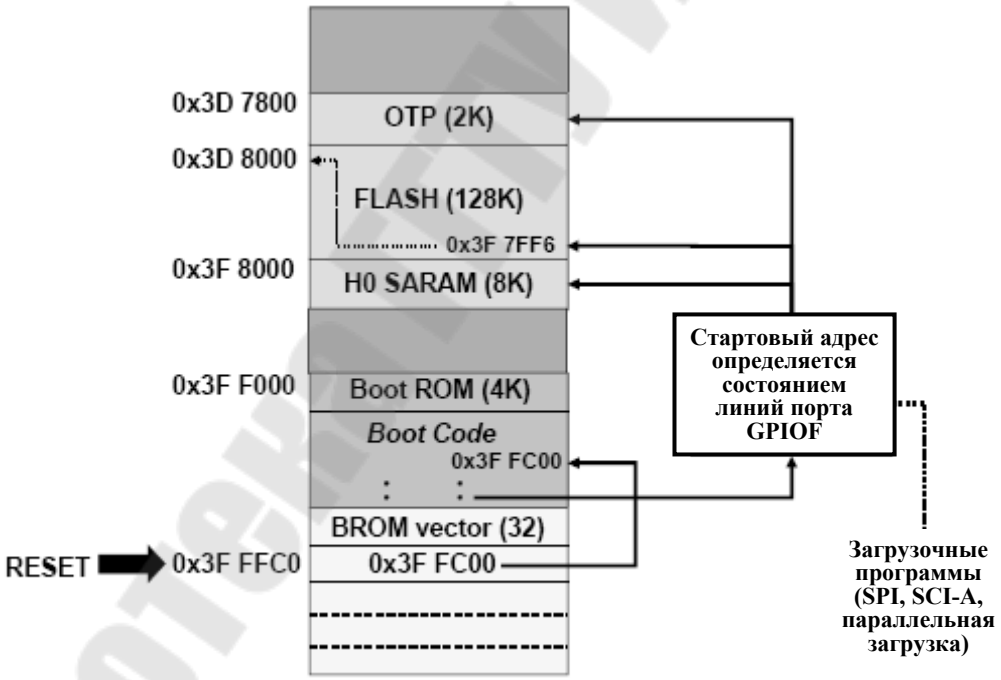


Рис. 3.4. Режимы запуска программы Bootloader

ЦСП имеет большое количество источников прерываний – 96, но только 16 линий прерываний ядра. Для возможности обслуживания всех прерываний для линий INT1…INT12 применяется мультиплексирование (рис. 3.5). Т.к. программный поиск конкретного прерывания в линии при обработке программно занял бы длительное время, то при-

меняется специальный аппаратный модуль – Peripheral Interrupt Expansion (PIE), или расширитель прерываний периферии. При работе с PIE происходит перенос области векторов: каждому из 96 прерываний соответствует свой 32-битный адрес в адресном пространстве – таблице векторов прерываний расширителя (PIE vector, адреса 0x00 D000…0x00 DFFF, см. рис. 2.2 и 3.9). Прерывания сгруппированы по 8 источников на линию (см. рис. 3.8). Для разрешения/запрета каждого прерывания используются биты в регистрах PIEIERx (x может принимать значения от 1 до 12), для индикации прерываний – биты в регистрах PIEIFRx. Соответствующий бит в регистре подтверждения PIEACK (активный уровень – 0) определяет номер активного прерывания для ядра CPU внутри группы. В результате каждая группа мультиплексируется в одно из прерываний ядра INT1…INT12 (рис. 3.6).

Форматы регистров модуля расширителя прерываний PIEIERx и PIEIFRx показаны на рис. 3.7, данные регистры содержат по 8 информационных бит, т.е. по числу прерываний в группе. В регистре PIECTRL биты 15-1 (PIEVECT) показывают адрес в пределах таблицы векторов PIE vector, из которой был извлечен вектор. Младший значащий бит игнорируется и показываются биты адреса от 1 до 15 (т.е. только четные адреса), что позволяет при чтении из регистра однозначно определить, какое прерывание генерировалось. ENPIE – бит разрешения извлечения векторов из таблицы PIE-контроллера. Если ENPIE=1, все вектора извлекаются из таблицы векторов PIE vector, а если ENPIE=0, PIE-контролер запрещен, и вектора извлекаются из таблицы CPU-векторов (BROM vector, см. рис. 2.2).

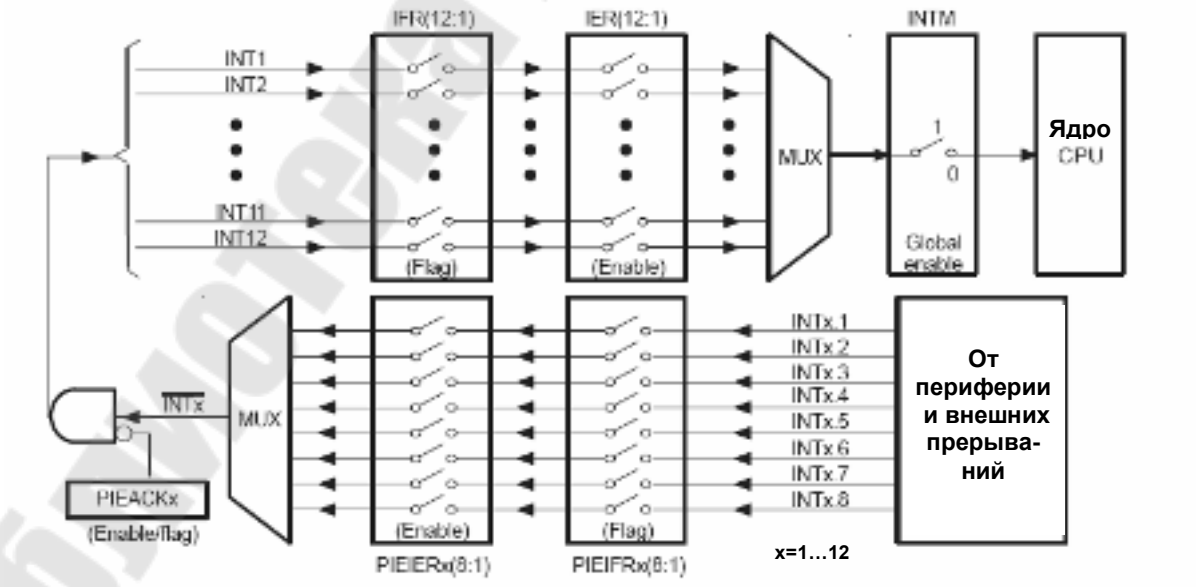


Рис. 3.5. Логика работы регистров PIEIFRx, PIEIERx, PIEACKx

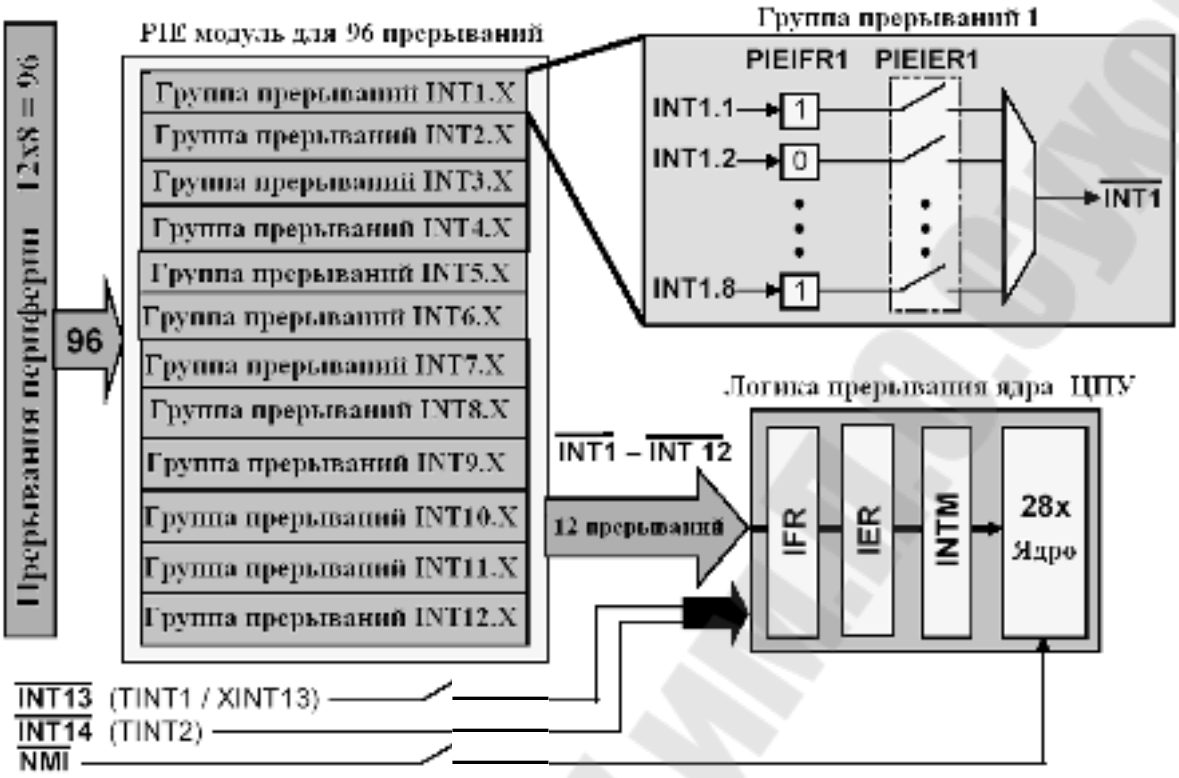


Рис. 3.6. Общая структура расширения прерываний ЦСП C28x

Рис. 3.7. Формат регистров модуля расширителя прерываний ЦСП C28x

Таблица векторов прерываний приведена на рис. 3.8.
Примеры векторов прерываний:
- прерывание от встроенного АЦП (ADCINT) – INT1.6;
- прерывание от CPU-таймера 0 (TINT0) – INT1.7.

приоритет

	INTx.8	INTx.7	INTx.6	INTx.5	INTx.4	INTx.3	INTx.2	INTx.1
INT1	WAKEINT	TINT0	ADCINT	XINT2	XINT1		PDPINTB	PDPINTA
INT2		T1OFINT	T1UFINT	T1CINT	T1PINT	CMP3INT	CMP2INT	CMP1INT
INT3		CAPINT3	CAPINT2	CAPINT1	T2OFINT	T2UFINT	T2CINT	T2PINT
INT4		T3OFINT	T3UFINT	T3CINT	T3PINT	CMP6INT	CMP5INT	CMP4INT
INT5		CAPINT6	CAPINT5	CAPINT4	T4OFINT	T4UFINT	T4CINT	T4PINT
INT6			MXINT	MRINT			SPITXINTA	SPIRXINTA
INT7								
INT8								
INT9			ECAN1INT	ECAN0INT	SCITXINTB	SCIRXINTB	SCITXINTA	SCIRXINTA
INT10								
INT11								
INT12								

Рис. 3.8. Таблица источников прерываний в PIE

PIE Vector Mapping (ENPIE = 1)

Vector name	PIE vector address	PIE vector Description
Not used	0x00 0D00	Reset Vector Never Fetched Here
INT1	0x00 0D02	INT1 re-mapped below
.....	.....	..... re-mapped below
INT12	0x00 0D18	INT12 re-mapped below
INT13	0x00 0D1A	XINT1 Interrupt Vector
INT14	0x00 0D1C	Timer2 - RTOS Vector
Datalog	0x00 0D1D	Data logging vector
.....	.....	.....
USER11	0x00 0D3E	User defined TRAP
INT1.1	0x00 0D40	PIEINT1.1 interrupt vector
.....	.....	.....
INT1.8	0x00 0D4E	PIEINT1.8 interrupt vector
.....	.....	.....
INT12.1	0x00 0DF0	PIEINT12.1 interrupt vector
.....	.....	.....
INT12.8	0x00 0DFE	PIEINT12.8 interrupt vector

Рис. 3.9. Таблица векторов прерываний при ENPIE=1: вектор 0x00 0D00 не активен, первичные вектора прерывания ядра по адресам 0x00 0D02 по 0x00 0D1C переадресовываются в область 0x00 0D40…0x00 0DFE с учетом конкретного источника прерывания периферии; вектор извлекается за 9 шагов (машинных циклов ЦСП).

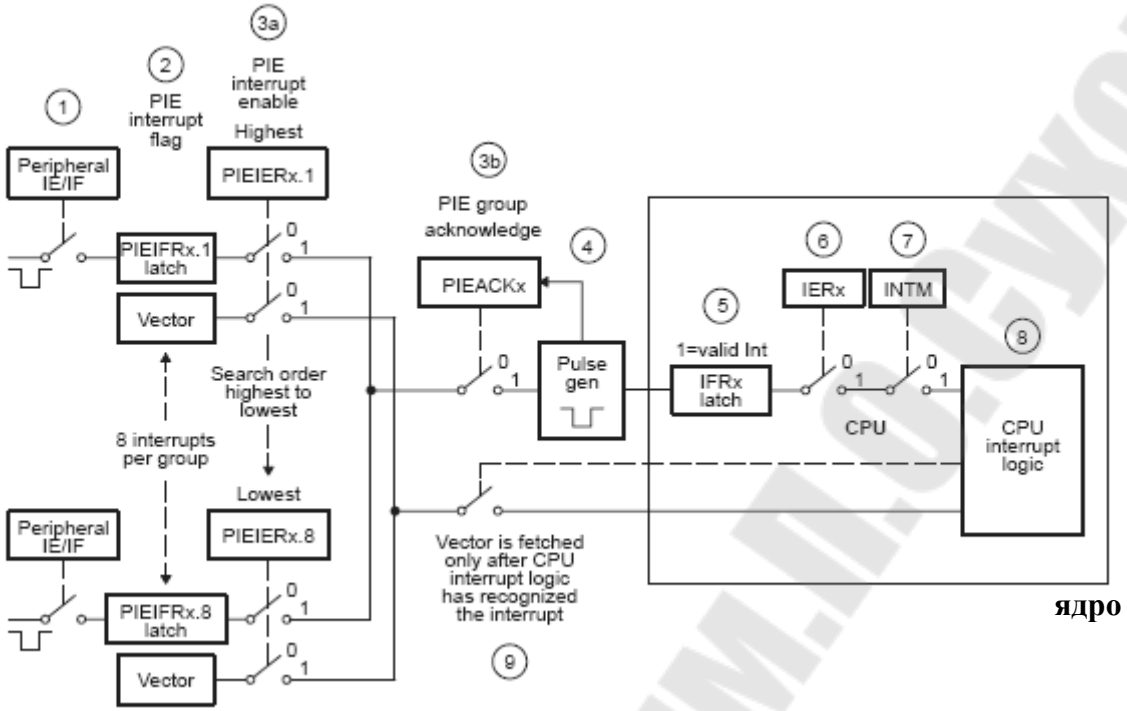


Рис. 3.10. Полная процедура обработки прерываний при ENPIE=1: шаг 1 – генерация прерывания от периферии; шаг 2 – установка флага PIEIFRx.y = 1; шаг 3а – проверка одновременного наличия двух условий: PIEIERx.y = 1 и PIEACKx=0; шаг 3б – установка в «1» бита PIEACKx для подтверждения прерывания от группы x; шаг 4 – формирование импульса прерывания по линии INTx на ядро (PIEACKx продолжает оставаться в единичном состоянии и требует программного сброса для возможности приема прерывания ядром по линии INTx в дальнейшем); шаг 5 – установка флага IFRx = 1; шаг 6 – проверка условия IERx = 1; шаг 7 – проверка условия INTM = 1, подготовка адреса возврата и данных к сохранению в стеке; шаг 8 – процессор определяет адрес вектора прерывания в области PIE Vector Mapping (адреса с 0x00 0D02 по 0x00 0D1C); шаг 9 – процессор определяет первичный адрес вектора прерывания в области PIE Vector Mapping с учетом текущего значения регистров PIEIER и PIEIFR (адреса с 0x00 0D40 по 0x00 0DFE).

В сигнальных процессорах семейства C28x имеется три 32-битных таймера ядра (CPU timers) с одинаковой структурой. Схема одного таймера приведена на рис. 3.11. Работа таймера разрешается сбросом бита TCR.4. Таймер имеет 16-битный предварительный делитель (прескалер) PSCH: PSC, который формирует счетный импульс вычитания из основного 32-битного счетчика TIMH: TIM. По достижении счетчиком TIMH: TIM нуля формируется сигнал прерывания

/TINT, поступающий на ядро. 16-битный регистр TDDRH: TDDR используется для перезагрузки прескалера таймера. Регистр PRDH: PRD содержит значение основного 32-битного счетчика, перезагружаемое в него при очередном переопустошении. Данная каскадная структура позволяет получить очень широкий диапазон коэффициентов деления частоты: от 2^2 до 2^{48}.

Таймеры 1 и 2, как правило, используются для операционной системы реального времени «DSP/BIOS», в то время как таймер 0 используется для пользовательских приложений. Данные таймеры интегрированы в ЦСП, не следует их путать с таймерами менеджеров событий (EvA и EvB). Необходимо отметить, что после сброса разрешается работа всех трех таймеров ядра.

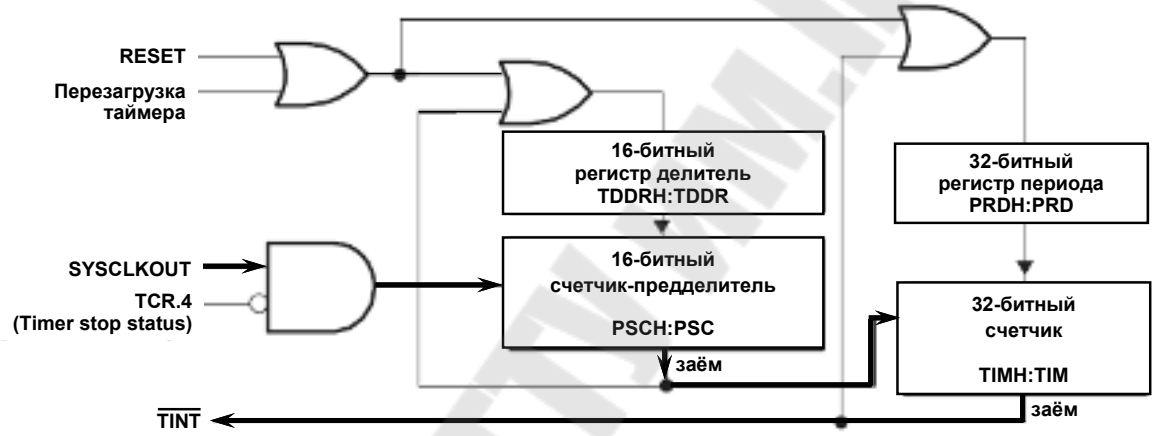


Рис. 3.11. Схема таймера ядра ЦСП семейства C28x (жирной линией показано прохождение тактового сигнала с делением частоты на выход)

Полный набор программно доступных регистров, относящихся к таймерам ядра, показан на рис. 3.12.

Счетчик-предделитель TIMERxPSC и регистр-делитель TIMERxTDDR программно доступны как один 16-разрядный регистр TIMERxTPR (см. рис. 3.13).

Аналогично, регистры TIMERxPSCH и TIMERxTDDRH, программно представляют собой единый 16-битный регистр TIMERxTPRH.

Address	Register	Name
0x0000 0C00	TIMER0TIM	Timer 0, Counter Register Low
0x0000 0C01	TIMER0TIMH	Timer 0, Counter Register High
0x0000 0C02	TIMER0PRD	Timer 0, Period Register Low
0x0000 0C03	TIMER0PRDH	Timer 0, Period Register High
0x0000 0C04	TIMER0TCR	Timer 0, Control Register
0x0000 0C06	TIMER0TPR	Timer 0, Prescaler Register
0x0000 0C07	TIMER0TPRH	Timer 0, Prescaler Register High
0x0000 0C08	TIMER1TIM	Timer 1, Counter Register Low
0x0000 0C09	TIMER1TIMH	Timer 1, Counter Register High
0x0000 0C0A	TIMER1PRD	Timer 1, Period Register Low
0x0000 0C0B	TIMER1PRDH	Timer 1, Period Register High
0x0000 0C0C	TIMER1TCR	Timer 1, Control Register
0x0000 0C0D	TIMER1TPR	Timer 1, Prescaler Register
0x0000 0C0F	TIMER1TPRH	Timer 1, Prescaler Register High
0x0000 0C10 to 0C17	Timer 2 Registers ; same layout as above	

Рис. 3.12. Регистры таймеров ядра ЦСП семейства C28x

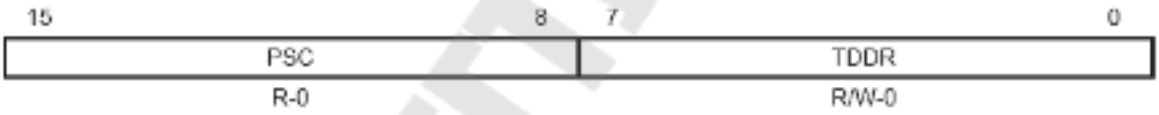


Рис. 3.13. Формат регистров-прескалеров TIMERxTPR (x = 0, 1, 2)

Функции управления каждым из таймеров реализованы в регистрах управления TIMERxTCR, формат которых показан на рис. 3.14.

Сигналы прерываний, формируемые CPU-таймерами, связаны с прерываниями ядра, как показано на рис. 3.15. Прерывание таймера 0 происходит через PIE, а таймеров 1 и 2 – попадают напрямую на ядро через соответствующие линии.

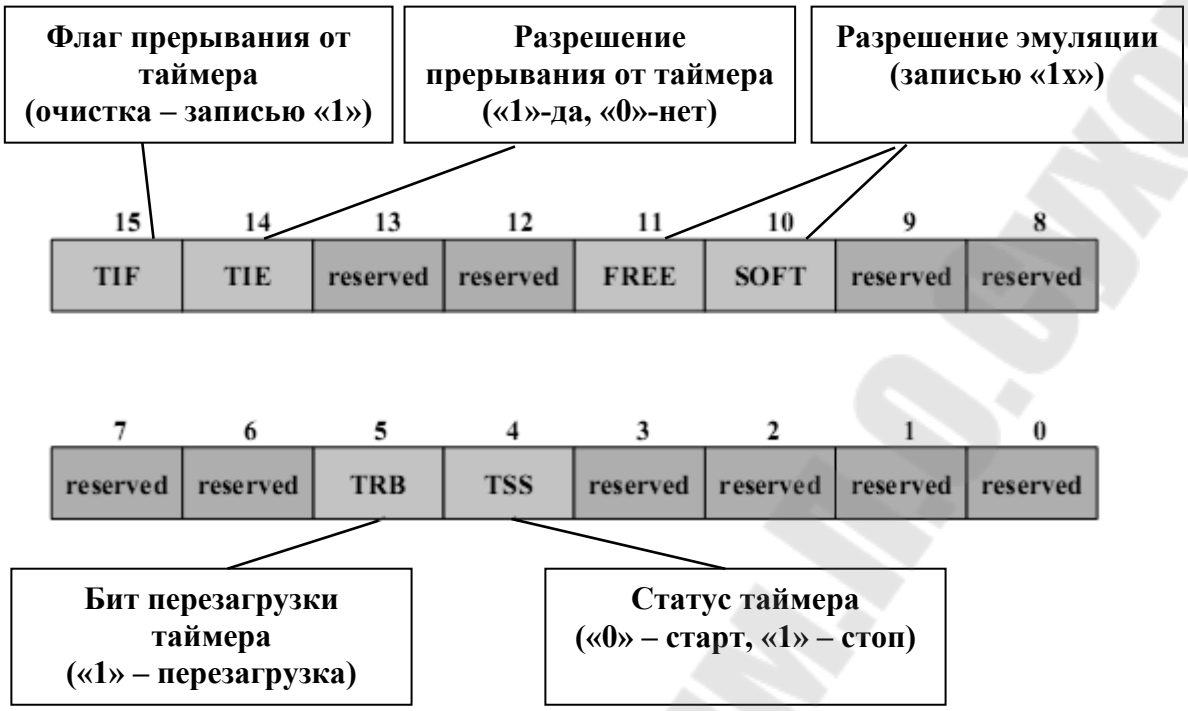


Рис. 3.14. Формат регистров управления таймерами ядра (TIMERxTCR)

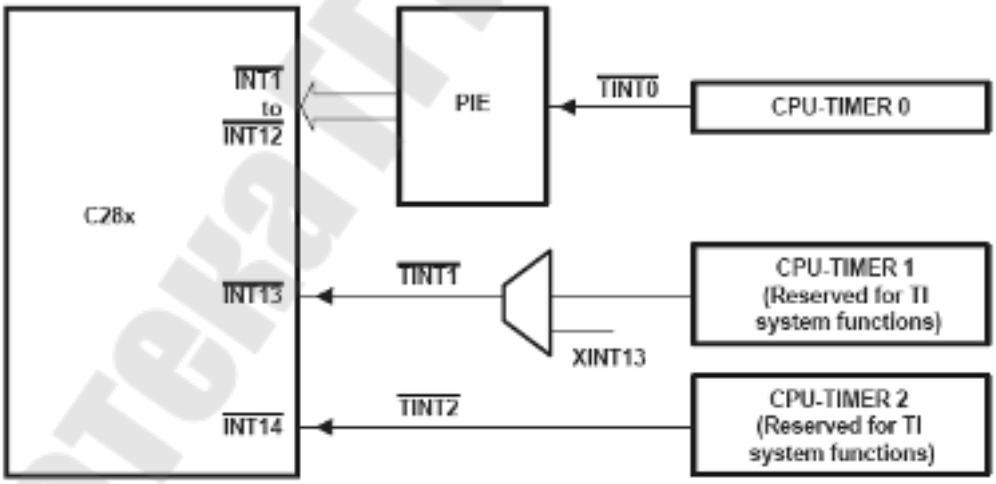


Рис. 3.15. Сигналы прерываний от CPU-таймеров

Ход работы

Использование таймера позволяет более эффективно использовать ресурсы процессора. Наиболее простые задачи для таймера – вы-

зывать на выполнение периодические задачи либо инкрементировать глобальную переменную. Данная переменная будет пропорциональна машинному времени, прошедшему с момента запуска таймера.

Для выполнения данной работы следует использовать файл с программой из предыдущей работы. Но вместо программной задержки в этот раз будем использовать аппаратную задержку, формируемую таймером 0 ядра ЦСП. Данный таймер использует систему расширения прерываний (PIE).

1. Создание нового проекта.

В Code Composer Studio создаем новый проект Lab3.pjt. В поле Project Name записываем название проекта «Lab3». В поле Location указывается путь, по которому будет находиться проект – E:\DspUser\Lab3.

1.1. Открываем файл с программой формирования бегущих огней из лабораторной работы №2 Lab2.c и сохраняем его под именем Lab3.c. Затем добавляем файл Lab3.c в проект: Project → Add files to Project.

Структура исходного текста программы показана на рис. 3.16.

1.2. Добавляем в проект управляющий файл линкера, командные файлы, библиотеки, необходимые внешние программные модули (рекомендуется для исключения ошибок пути к файлам, указанные ниже, копировать в диалоговое окно «Add files to Project»):

C:\tidcs\c28\dsp281x\v100\DSP281x_common\cmd\F2812_EzDSP_RAM_lnk.cmd
C:\tidcs\c28\dsp281x\v100\DSP281x_headers\cmd\DSP281x_Headers_nonBIOS.cmd
C:\CCStudio_v3.3\C2000\cgtools\lib\rts2800_ml.lib
C:\tidcs\c28\dsp281x\v100\DSP281x_headers\source\DSP281x_GlobalVariableDefs.c
C:\tidcs\c28\dsp281x\v100\DSP281x_common\source\DSP281x_PieCtrl.c
C:\tidcs\c28\dsp281x\v100\DSP281x_common\source\DSP281x_PieVect.c
C:\tidcs\c28\dsp281x\v100\DSP281x_common\source\DSP281x_DefaultIsr.c
C:\tidcs\c28\dsp281x\v100\DSP281x_common\source\DSP281x_CpuTimers.c

2. Настройка параметров проекта.

2.1. Включаем в проект заголовочные файлы, для этого выбираем Project → Build Options**,** в закладке Compiler выбираем Preprocessor и в поле Include Search Path (-i) вводим:

C:\tidcs\C28\dsp281x\v100\DSP281x_headers\include;..\include

2.2. Добавление библиотек и создание стека.

Подключаем Си-библиотеки:

Project → Build Options → Linker → Libraries → Search Path: C:\CCStudio_v3.3\C2000\cgtools\lib
Project → Build Options → Linker → Libraries → Search Path Linker → Incl. Libraries: rts2800_ml.lib

Задаем глубину стека 0x400:
Project → Build Options → Linker → Basic → Stack Size (-stack): 0x400

```
//###########################################################################
//     Имя файла:  Lab3.c
//
//     Описание:   С помощью 8 светодиодов, подключенных к линиям
//           порта GPIOB0 - GPIOB7, формируются "бегущие огни".
//           Направление движения справа - налево и наоборот
//###########################################################################

#include "DSP281x_Device.h" // Включение заголовочного файла

void delay_loop(long);
void Gpio_select(void);
void InitSystem(void);

void main(void)
{
      unsigned int i;
      unsigned int LED[8]= {0x0001,0x0002,0x0004,0x0008,
                            0x0010,0x0020,0x0040,0x0080};

      InitSystem();           // Инициализация регистров ЦСП
      Gpio_select();          // Инициализация линий ввода/вывода

while(1)
      {
          for(i=0;i<14;i++)
            {
           if(i<7)      GpioDataRegs.GPBDAT.all = LED[i];
           else   GpioDataRegs.GPBDAT.all = LED[14-i];
           delay_loop(?);
       }
     }
}


//###########################################################################
//     Подпрограмма:     delay_loop
//
//     Описание:   Формирование временной задержки горения светодиодов
//###########################################################################

void delay_loop(long end)
{
      long i;
      for (i = 0; i < end; i++);
      EALLOW; // Сброс сторожевого таймера
      SysCtrlRegs.WDKEY = 0x?;
      SysCtrlRegs.WDKEY = 0x?;
      EDIS;
}
```

```
//###########################################################################
//    Подпрограмма:     Gpio_select
//
//    Описание:   Настройка линий порта GPIO B15-8 на ввод, а линий B7-0
//          на вывод. Настройка всех линий портов A, D, F, E, G на
//          ввод. Запрещение работы входного ограничителя
//###########################################################################

void Gpio_select(void)
{
      EALLOW;
    GpioMuxRegs.GPAMUX.all = 0x?; // Настройка линий ввода/вывода на
    GpioMuxRegs.GPBMUX.all = 0x?; // работу в качестве портов
    GpioMuxRegs.GPDMUX.all = 0x?;
    GpioMuxRegs.GPFMUX.all = 0x?;
    GpioMuxRegs.GPEMUX.all = 0x?;
    GpioMuxRegs.GPGMUX.all = 0x?;

    GpioMuxRegs.GPADIR.all = 0x?; // Настройка портов A, D, E, F, G на ввод
    GpioMuxRegs.GPBDIR.all = 0x?; // Настройка линий 15-8 на ввод,
    GpioMuxRegs.GPDDIR.all = 0x?; // а линий 7-0 на вывод
    GpioMuxRegs.GPEDIR.all = 0x?;
    GpioMuxRegs.GPFDIR.all = 0x?;
    GpioMuxRegs.GPGDIR.all = 0x?;

    GpioMuxRegs.GPAQUAL.all = 0x?; // Запрещение входного ограничителя
    GpioMuxRegs.GPBQUAL.all = 0x?;
    GpioMuxRegs.GPDQUAL.all = 0x?;
    GpioMuxRegs.GPEQUAL.all = 0x?;
    EDIS;
}

//###########################################################################
//    Подпрограмма:     InitSystem
//
//    Описание:   Настройка сторожевого таймера на работу,
//          предделитель = 1. Выработка сброса сторожевым
//          таймером. Внутренняя частота работы ЦСП 150 МГц.
//          Запись в предделитель высокоскоростного таймера
//          коэффициента деления 2, а в предделитель
//          низкоскоростного таймера - 4. Запрещение работы
//          периферийных устройств
//###########################################################################

void InitSystem(void)
{
      EALLOW;
      SysCtrlRegs.WDCR= 0x?;        // Разрешение работы сторожевого
                                    // таймера, предделитель = 64
                                    // или запрещение работы  сторо-
                                    // жевого таймера, предделитель = 1

      SysCtrlRegs.SCSR = ?;         // Конфигурирование сброса ЦСП от WDT

      SysCtrlRegs.PLLCR.bit.DIV = ?;// Настройка блока умножения частоты
      SysCtrlRegs.HISPCP.all = 0x?; //  Задание значений высокоскоростного
      SysCtrlRegs.LOSPCP.all = 0x?; // и низкоскоростного предделителей
```

6

```
    SysCtrlRegs.PCLKCR.bit.EVAENCLK=?;  // Запрещение работы
    SysCtrlRegs.PCLKCR.bit.EVBENCLK=?;  // периферийных устройств
    SysCtrlRegs.PCLKCR.bit.SCIAENCLK=?;
    SysCtrlRegs.PCLKCR.bit.SCIBENCLK=?;
    SysCtrlRegs.PCLKCR.bit.MCBSPENCLK=?;
    SysCtrlRegs.PCLKCR.bit.SPIENCLK=?;
    SysCtrlRegs.PCLKCR.bit.ECANENCLK=?;
    SysCtrlRegs.PCLKCR.bit.ADCENCLK=?;
    EDIS;
}
```

Рис. 3.16. Структура исходного текста программы

3. Редактирование программы.

3.1. В программе Lab3.c объявляем подпрограмму обработки прерываний от CPU-таймера 0:

```
interrupt void cpu_timer0_isr(void);
```

Место вставки – область 1 на рис. 3.16.

Далее в основной программе добавляем вызов подпрограммы:

```
InitPieCtrl();
```

Место вставки – область 2 на рис. 3.16.

Данная подпрограмма описана в файле DSP281x_PieCtrl.c, добавленном в проект. Она позволяет очистить флаги и запретить все прерывания, что удобно при написании программ. Просмотреть и проанализировать содержимое данного файла можно, открыв его в другом окне редактора CCS. Аналогичным образом рекомендуется далее ознакомиться и с программами, хранящимися в прочих программных модулях, подключенных к проекту (файлы *.c).

3.2. Непосредственно после вызова подпрограммы «`InitPieCtrl();`» добавляем еще один вызов подпрограммы:

```
InitPieVectTable();
```

Данная подпрограмма инициализирует область векторов PIE в начальное состояние. Она использует предварительно заданную таблицу прерываний «PieVectTableInit()», которая определена в файле DSP281x_PieVect.c, и копирует эту таблицу в глобальную переменную «PieVectTable», которая связана с областью памяти процессора PIE area. Также, для использования подпрограммы `InitPieVectTable`, в проект добавлен файл DSP281x_DefaultIsr.c, который добавляет в проект подпрограммы обработки прерываний.

3.3. Необходимо переопределить имя подпрограммы обработки прерываний от CPU-таймера 0 на нашу подпрограмму. Для этого в основную программу сразу после вызова подпрограммы «`InitPieVectTable();`» добавляем вызов следующих подпрограмм:

```
EALLOW;
PieVectTable.TINT0 = &cpu_timer0_isr;
EDIS;
```

Здесь `EALLOW` и `EDIS` – подпрограммы, используемые соответственно для разрешения и запрета доступа к системным регистрам процессора, а «`cpu_timer0_isr`» – имя подпрограммы обработки прерываний, описанной в программе ранее.

3.4. Инициализируем CPU-таймер 0. В основную программу необходимо добавить вызов подпрограммы (место вставки – сразу после команд, указанных в п. 3.3):

```
InitCpuTimers();
```

Для возможности работы этой подпрограммы в проект добавлен файл DSP281x_CpuTimers.c. После этого таймер будет инициализирован и остановлен.

3.5. Необходимо настроить CPU Timer0 для генерации временных интервалов в 50 мс при тактовой частоте 150 МГц. Для этого сразу после вызова подпрограммы «`InitCpuTimers();`» добавляем вызов подпрограммы:

```
ConfigCpuTimer(&CpuTimer0, 150, 50000);
```

Выполнение данной подпрограммы реализует файл DSP281x_CpuTimers.c.

3.6. Настраиваем прерывания от CPU-таймера 0. Необходимо настроить три уровня прерывания.

Первый уровень – модуль PIE, группа PIEIER1 (т.к. прерывания от CPU-таймера 0 относятся именно к данной группе, см. рис. 3.8):

```
PieCtrlRegs.PIEIER1.bit.INTx7 = 1;
```

Второй уровень – разрешение прерываний линии 1 ядра ЦПУ. Для этого необходимо настроить регистр IER (см. рис. 3.2 и 3.3).

```
IER = 1;
```

Третий уровень – разрешить глобальные прерывания. Следует разрешить глобальные прерывания, добавив в программу команды:

```
EINT;
ERTM;
```

Вызов данных подпрограммы следует произвести сразу вслед за конфигурацией таймера, произведенной в п. 3.6.

3.7. Затем необходимо добавить команду запуска CPU Timer0:

```
CpuTimer0Regs.TCR.bit.TSS = 0;
```

3.8. Сразу после основной программы (область 3 на рис. 3.16) необходимо добавить подпрограмму обработки прерываний от CPU-таймера 0 «cpu_timer0_isr». Подпрограмму и обращение к ней мы уже внесли в программу. Теперь необходимо написать саму подпрограмму. Подпрограмма будет иметь общий вид:

```
interrupt void cpu_timer0_isr(void)
{
…
```

```
(текст подпрограммы)
…
}
```

Подпрограмма должна выполнять следующие действия:

- инкрементировать глобальную переменную «CpuTimer0.InterruptCount», описываемую (начальное значение = 0) в подключаемом файле DSP281x_CpuTimers.c. Данная переменная будет показывать истечение интервала времени 50 мс с момента запуска таймера;
- сбросить в «0» бит 1 регистра PIEACK, что необходимо для разрешения последующих прерываний. Это действие выполняет команда:

```
PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
```

3.9. После настройки таймера и прерываний корректируем основную программу. Для этого удаляем (либо добавляем в начале строки признак комментария «//») вызов подпрограммы «`delay_loop(?);`» (см. область 4 на рис. 3.16). Также удаляем объявление этой подпрограммы (см. область 5 на рис. 3.16), и заключаем в скобки комментария саму подпрограмму «`void delay_loop(long end)`». Скобки комментария открываются парой символов «/*» и закрываются парой символов «*/».

Затем необходимо программно реализовать цикл ожидания для формирования задержки длительностью 150 мс, с учетом того, что переменная «CpuTimer0.InterruptCount» однократно инкрементируется в подпрограмме «`interrupt void cpu_timer0_isr(void)`» каждые 50 мс. Место вставки цикла ожидания – область 4 на рис. 3.16. После цикла ожидания необходимо сбросить переменную «CpuTimer0.InterruptCount» в 0.

3.10. Разрешаем работу Watchdog timer (WDT). См. область 6 на рис. 3.16.

3.11. Добавляем обслуживание WDT. Для этого необходимо последовательно записать в регистр WDKEY коды «0x55» и «0xAA», аналогично тому, как это выполнялось в программе к лабораторной работе № 2 (см. область 7 на рис. 3.16). Отличием является то, что

данную процедуру вместе с макросами `EALLOW` и `EDIS` необходимо перенести внутрь подпрограммы обработки прерывания от CPU-таймера 0 «`interrupt void cpu_timer0_isr(void)`». Поскольку прерывание от таймера происходит циклично с периодом 50 мс, сброс ЦСП от WDT не происходит.

4. Компиляция, компоновка и загрузка выходного файла в отладочный модуль ЦСП.

4.1. Компилируем программу: Project → Compile File. При наличии ошибок, исправляем их.

4.2. Компонуем проект: Project → Build. При наличии ошибок, исправляем их.

4.3. Загружаем выходной файл: File → Load Program → Debug\ lab3.out (в случае, если данная функция сконфигурирована для выполнения автоматически, выполнять данный пункт не нужно).

Содержание отчета:

Отчет должен содержать цель работы, краткие теоретические сведения по теме работы, тексты исследуемых программ, результаты их выполнения, выводы.

Контрольные вопросы:

1. Маскируемые и немаскируемые прерывания.
2. Сброс ЦСП.
3. Обработка прерываний ядра ЦСП семейства С28х.
4. Назначение и структура модуля расширения прерываний (PIE) ЦСП семейства С28х.
5. Расположение векторов прерываний ЦСП семейства С28х в режимах ENPIE=0 и ENPIE=1.
6. Процедура обработки прерываний при ENPIE=1.
7. Назначение и структура CPU-таймеров ЦСП семейства С28х.
8. Регистры CPU-таймеров ЦСП семейства С28х.
9. Какие изменения необходимо внести в исходный текст, чтобы программа «бегущие огни» выполнялась с движением «горящих» светодиодов:
   - в 1,7 раз быстрее?
   - в 2,45 раз медленнее?

10. Какие изменения необходимо внести в исходный текст, чтобы программа «бегущие огни» выполнялась:

- с замедлением движения «горящих» светодиодов в 2 раза на каждом шаге движения?

- с ускорением движения «горящих» светодиодов в 3 раза на каждом шаге движения?

Содержание

Крышнев Юрий Викторович
Баранов Алексей Геннадьевич
Храмов Александр Сергеевич

ЦИФРОВЫЕ СИГНАЛЬНЫЕ ПРОЦЕССОРЫ

Лабораторный практикум
по одноименному курсу для студентов специальности 1-36 04 02 «Промышленная электроника» дневной и заочной форм обучения
В 4 частях
Часть 1

Подписано в печать 22.01.10.
Формат 60x84/16. Бумага офсетная. Гарнитура «Таймс».
Ризография. Усл. печ. л. 2,79. Уч.-изд. л. 3,04.
Изд. № 48.
E-mail: ic@gstu.by
http://www.gstu.by

Отпечатано на цифровом дуплекаторе
с макета оригинала авторского для внутреннего использования.
Учреждение образования «Гомельский государственный
технический университет имени П. О. Сухого».
246746, г. Гомель, пр. Октября, 48.