

**Министерство образования Республики Беларусь**

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

**Кафедра «Промышленная электроника»**

# **АППАРАТУРА ЦИФРОВОЙ ОБРАБОТКИ СИГНАЛОВ**

**ЛАБОРАТОРНЫЙ ПРАКТИКУМ**

**для студентов специальности 1-36 04 02**

**«Промышленная электроника», специализации  
1-36 04 02 01 «Микроэлектронные и микропроцессорные  
управляющие и информационные устройства»  
дневной и заочной форм обучения**

**Гомель 2012**

УДК 621.382.049.77(075.8)  
ББК 32.859я73  
А76

*Рекомендовано научно-методическим советом  
факультета автоматизированных и информационных систем  
ГГТУ им. П. О. Сухого  
(протокол № 12 от 25.06.2012 г.)*

Авторы: *Ю. В. Крышнев, А. С. Храмов, В. Н. Гарбуз, О. А. Елисеева*

Рецензент: зав. каф. «Автоматизированный электропривод» ГГТУ им. П. О. Сухого  
канд. техн. наук, доц. *В. С. Захаренко*

**А76** **Аппаратура** цифровой обработки сигналов : лаборатор. практикум для студентов специальности 1-36 04 02 «Промышленная электроника», специализации 1-36 04 02 01 «Микроэлектронные и микропроцессорные управляющие и информационные устройства» днев. и заоч. форм обучения / Ю. В. Крышнев [и др.] – Гомель : ГГТУ им. П. О. Сухого, 2012. – 125 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://alis.gstu.by/StartEK/>. – Загл. с титул. экрана.

Включает шесть лабораторных работ по изучению и программированию основных функциональных модулей цифрового сигнального процессора TMS320F2812.

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

УДК 621.382.049.77(075.8)  
ББК 32.859я73

© Учреждение образования «Гомельский государственный технический университет имени П. О. Сухого», 2012

## **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.

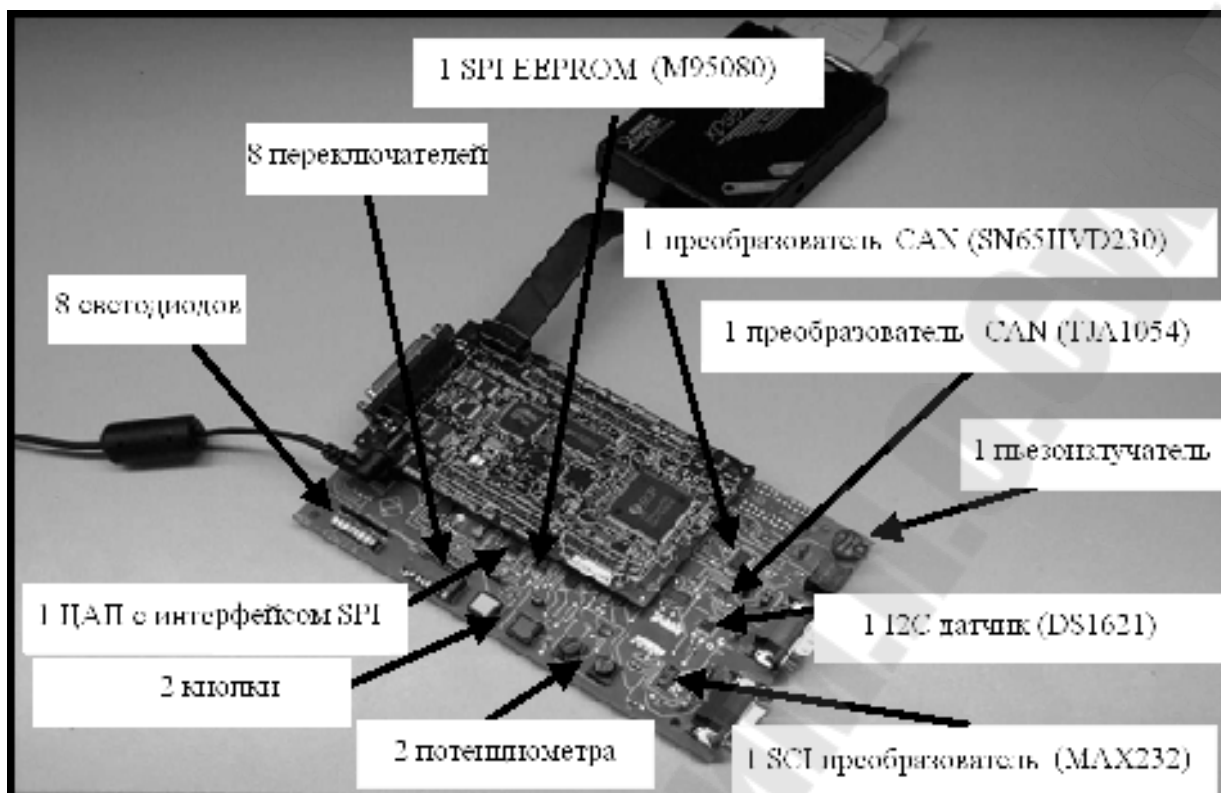


Рис. 1.1. Плата эмулятора eZDSP и плата расширения Zwickau Adapterboard

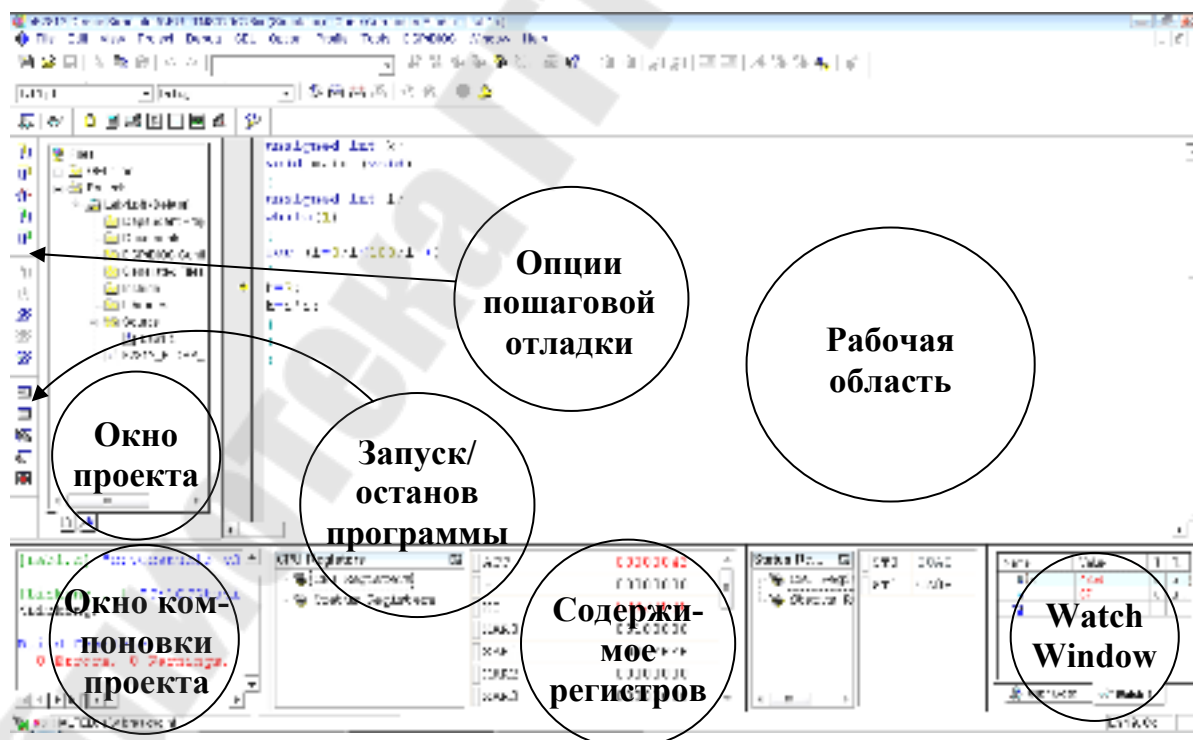


Рис. 1.2. Вид рабочего окна Code Composer Studio

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++, Макроасемблер). 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 → Source 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:\tides\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, наблюдайте 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 асемблированного 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.

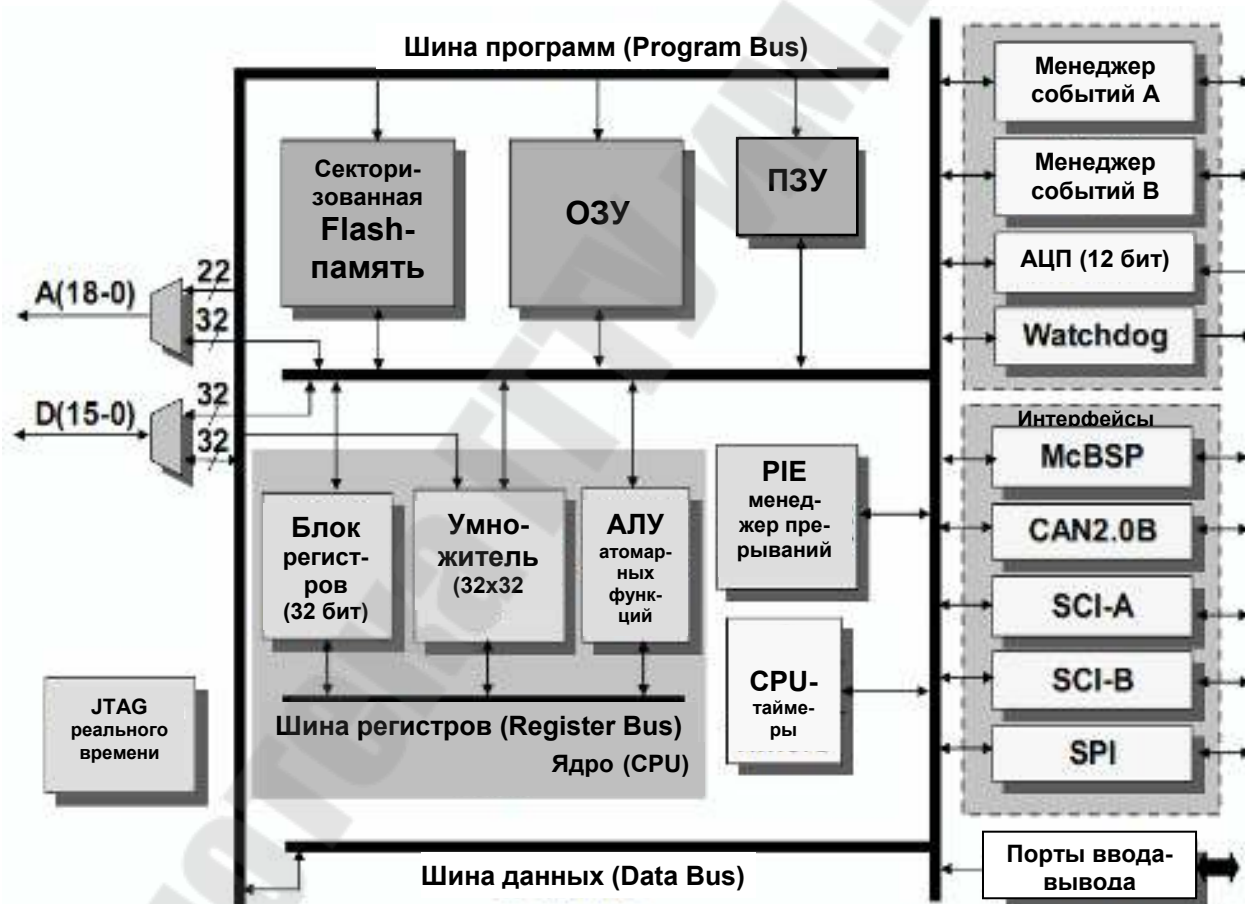


Рис. 2.1. Структурная схема ЦСП семейства C28x

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 объемом 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).

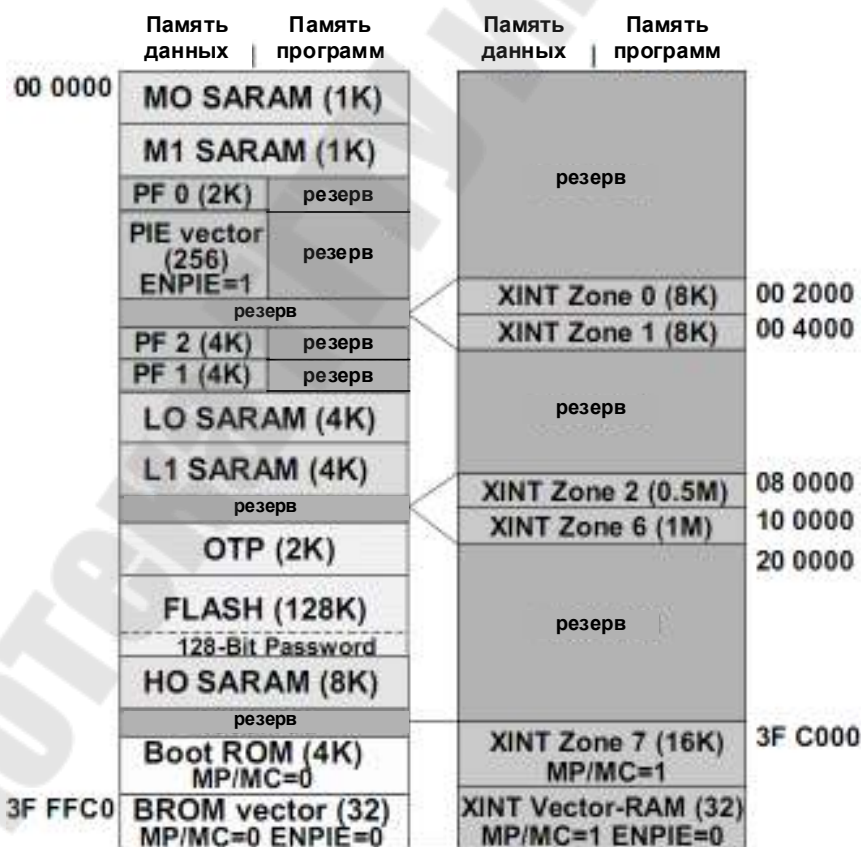


Рис. 2.2. Карта адресного пространства ЦСП семейства C28x

Peripheral Frame 0 (PF0, объем 2К, 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 4К, адреса 0x00 6000...0x00 6FFF) – включает в себя регистры интерфейса eCAN;

Peripheral Frame 1 (PF1, объем 4К, 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 выходы сгруппированы в порты GPIOA, GPIOB, GPIOD, GPIOE, GPIOF, GPIOG. GPIO (general purpose input/output) означает «линии ввода/вывода общего назначения».

Периферия имеет выходы, 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.

<b><u>GPIO A</u></b>	<b><u>GPIO B</u></b>	<b><u>GPIOD</u></b>
GPIOA0 / PWM1	GPIOB0 / PWM7	GPIOD0 / T1CTRIP_PDPINTA
GPIOA1 / PWM2	GPIOB1 / PWM8	GPIOD1 / T2CTRIP7_EVASOC
GPIOA2 / PWM3	GPIOB2 / PWM9	GPIOD5 / T3CTRIP_PDPINTB
GPIOA3 / PWM4	GPIOB3 / PWM10	GPIOD6 / T4CTRIP7_EVBSOC
GPIOA4 / PWM5	GPIOB4 / PWM11	
GPIOA5 / PWM6	GPIOB5 / PWM12	<b><u>GPIO E</u></b>
GPIOA6 / T1PWM_T1CMP	GPIOB6 / T3PWM_T3CMP	GPIOE0 / XINT1_XBIO
GPIOA7 / T2PWM_T2CMP	GPIOB7 / T4PWM_T4CMP	GPIOE1 / XINT2_ADCSOC
GPIOA8 / CAP1_QEP1	GPIOB8 / CAP4_QEP3	GPIOE2 / XNMI_XINT13
GPIOA9 / CAP2_QEP2	GPIOB9 / CAP5_QEP4	
GPIOA10 / CAP3_QEP11	GPIOB10 / CAP6_QEP12	
GPIOA11 / TDIRA	GPIOB11 / TDIRB	
GPIOA12 / TCLKINA	GPIOB12 / TCLKINB	
GPIOA13 / C1TRIP	GPIOB13 / C4TRIP	
GPIOA14 / C2TRIP	GPIOB14 / C5TRIP	
GPIOA15 / C3TRIP	GPIOB15 / C6TRIP	
<b><u>GPIO F</u></b>	<b><u>GPIO G</u></b>	
GPIOF0 / SPISIMOA	GPIOG4 / SCITXDB	
GPIOF1 / SPISOMIA	GPIOG5 / SCIRXDB	
GPIOF2 / SPICLKA		
GPIOF3 / SPISTEA		
GPIOF4 / SCITXDA		
GPIOF5 / SCIRXDA		
GPIOF6 / CANTXA		
GPIOF7 / CANRXA		
GPIOF8 / MCLKXA		
GPIOF9 / MCLKRA		
GPIOF10 / MFSXA		
GPIOF11 / MFSRA		
GPIOF12 / MDXA		
GPIOF13 / MDRA		
GPIOF14 / XF		

Замечания:

- после сброса выбирается GPIO функция портов
- GPIO A, B, D, E содержат модуль задержки ввода

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).

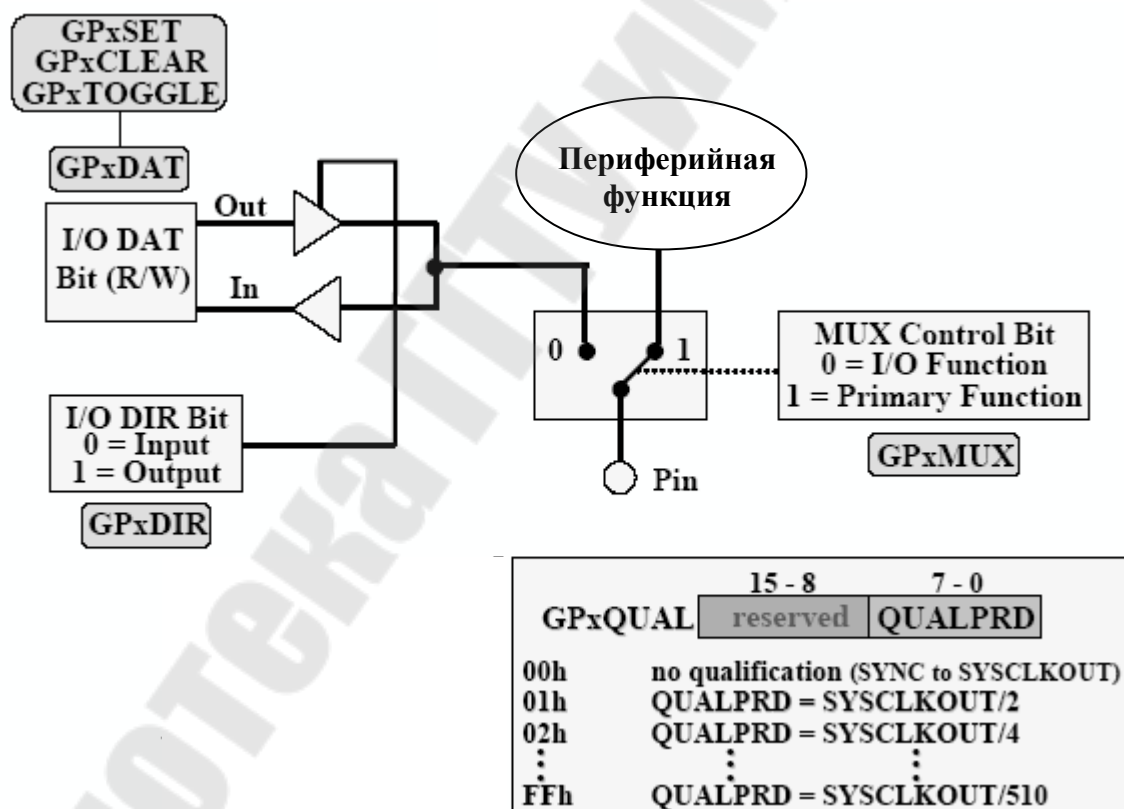


Рис. 2.4. Функциональная схема линии GPIO

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 А, В, D и Е 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.

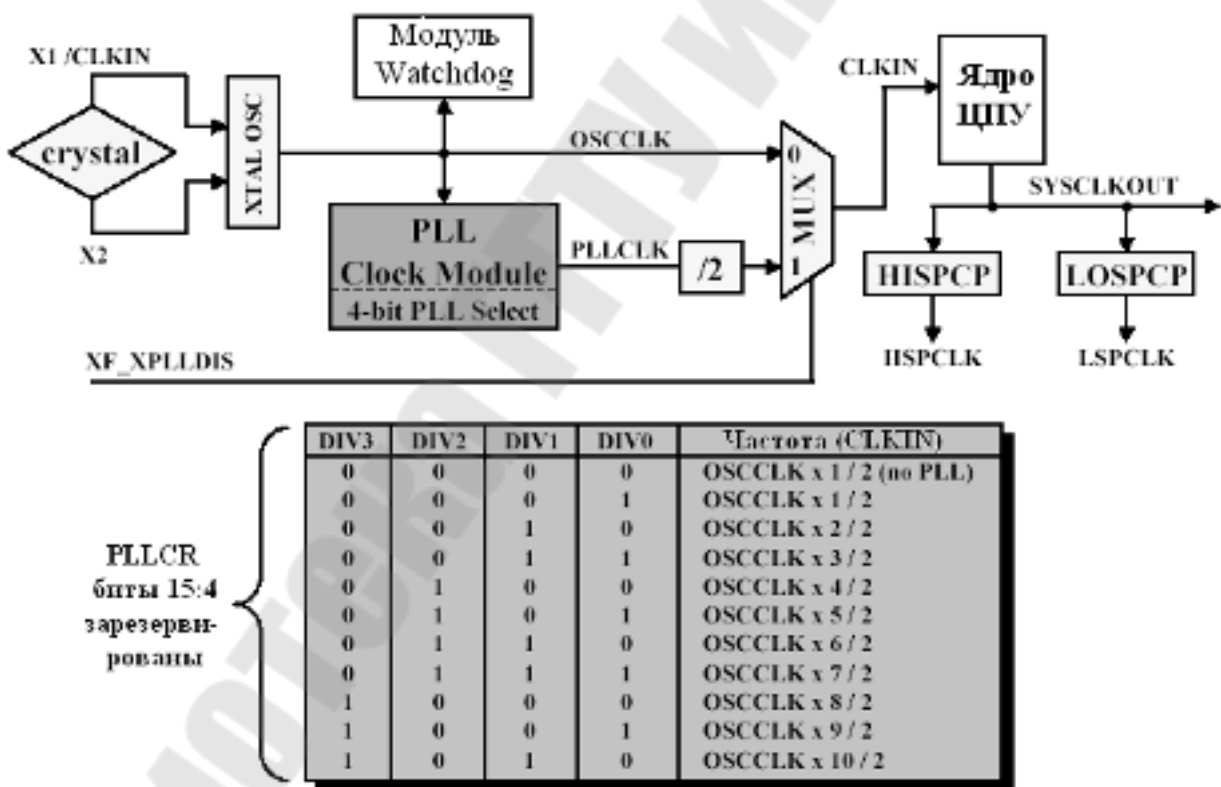


Рис. 2.5. Тактовый модуль и регистр PLLCR

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 LOSPCR и HISPCR.

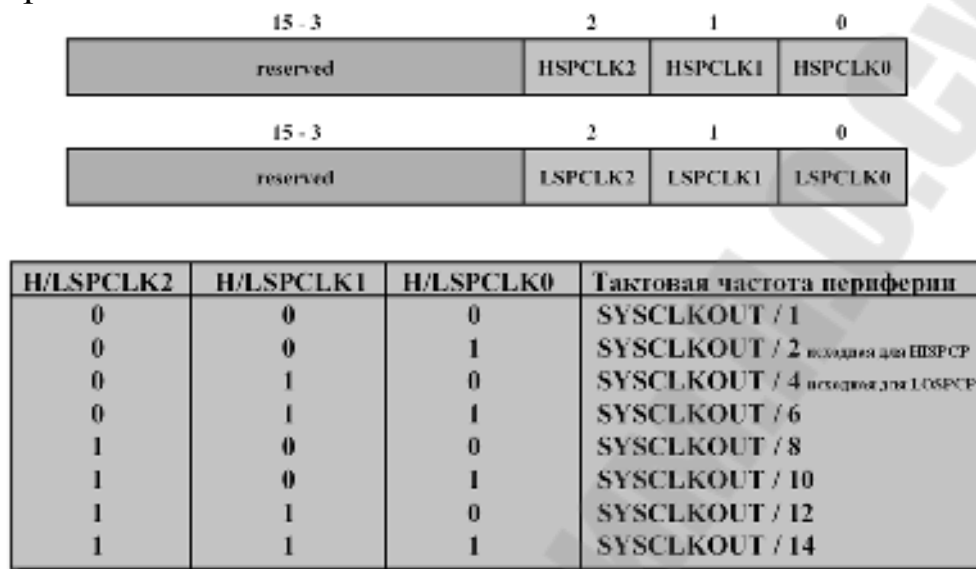
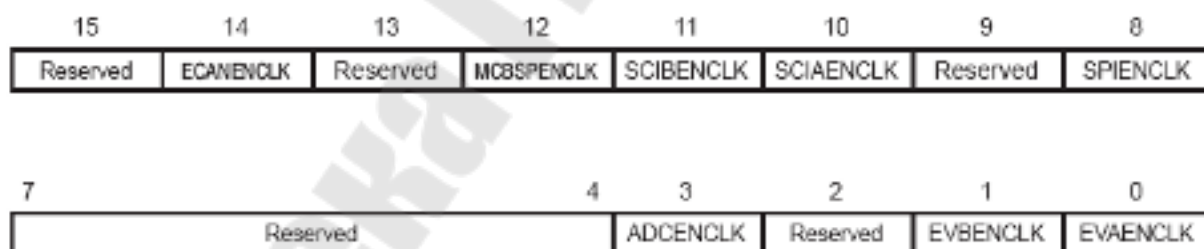


Рис. 2.6. Формат регистров HISPCR и LOSPCR

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.

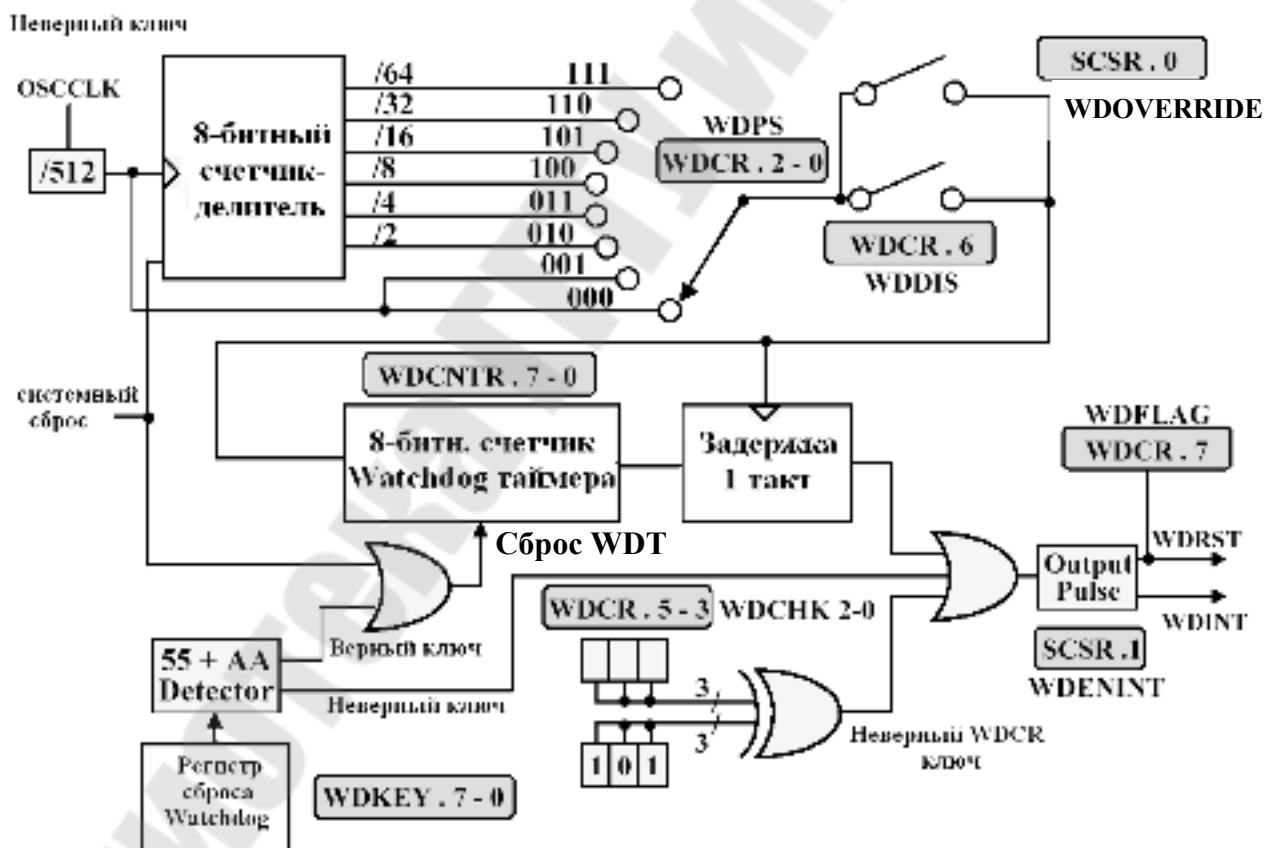


Рис. 2.8. Функциональная схема сторожевого таймера

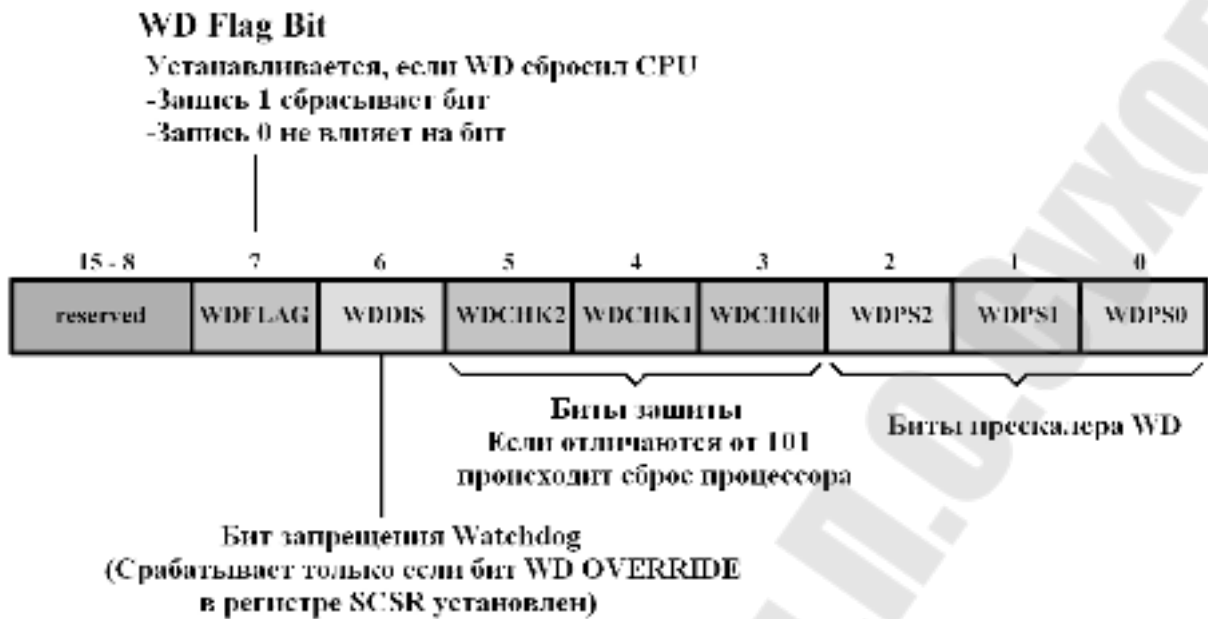


Рис. 2.9. Формат регистра управления сторожевого таймера WDCR

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).

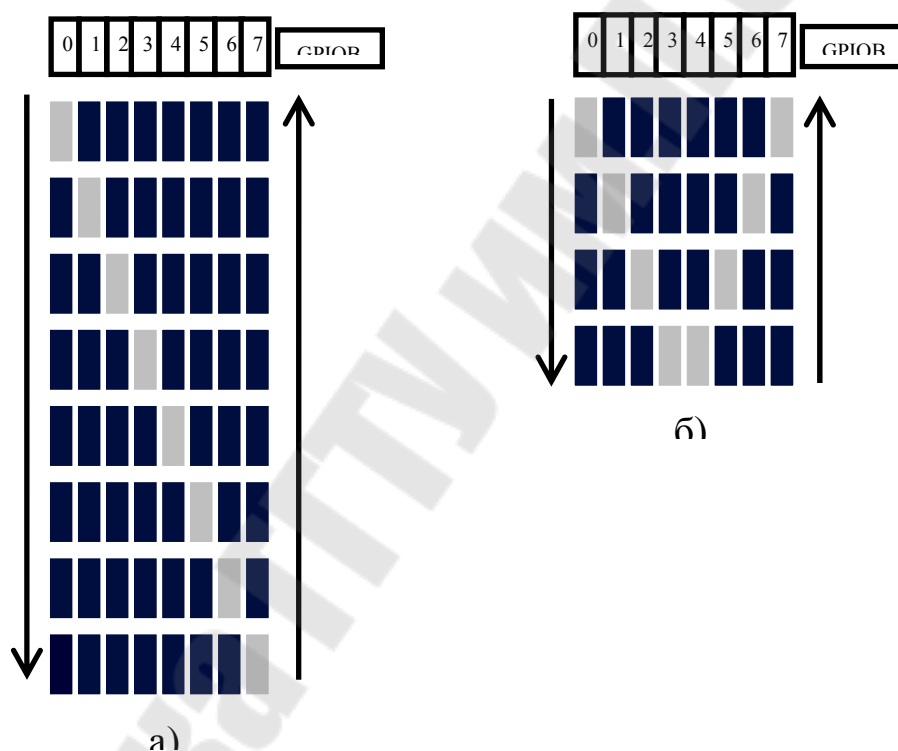


Рис. 2.11. Направление движения светодиодов: а) – слева направо и наоборот; б) – от периферии к центру и наоборот

#### 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 требу-

мые 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;
}

#####
// 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 ЦП 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 HISPSP (см. рис. 2.6) коэффициент деления 2, а в низкоскоростной (LOSPSP) – 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 выходы 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).

4.4. 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. Компоуем 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:

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



Для того чтобы сброс WDT происходил раньше (т.е. сброса ЦСП не происходило), нужно обеспечить соотношение  $t_{30} \leq t_{WDRST}$ .

8.4. Компилируем, компоуем проект и тестируем программу.

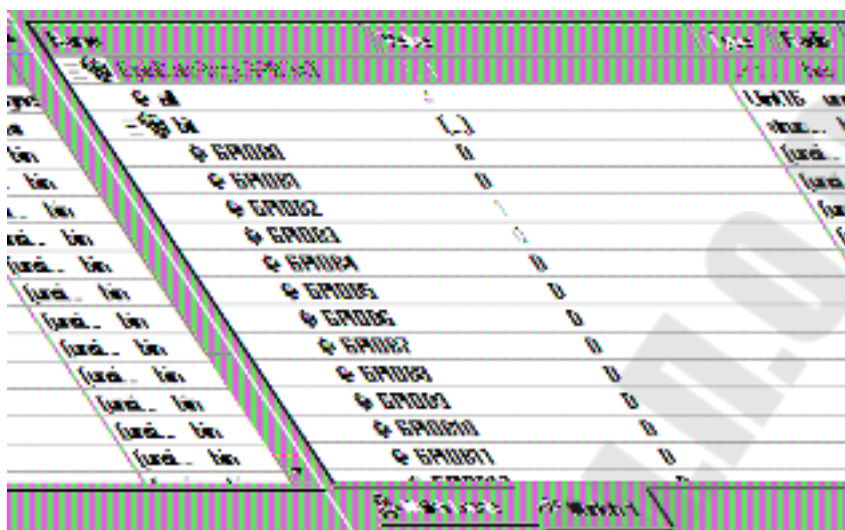


Рис. 2.13. Просмотр состояния двоичных разрядов порта GPIOB в окне Watch Window

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, компоуем 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 индцироваться на светодиодах. С

учетом того, что для подключения и светодиодов, и ключей используется один и тот же порт (GPIOB), состояние ключей можно индексировать, 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, компоуем 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 «Lab2с.c».

2.2. Преобразуем программу Lab2с.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 В15–В8. Подпрограммы «InitSystem()» и «Gpio\_select()» оставляем без изменений.

2.3. Добавляем в проект «Lab2с.c» и удаляем «Lab2b.c».

2.4. Компилируем, компоуем 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 C28х.
2. Карта адресного пространства ЦСП C28х.
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 C28x

**Цель работы:** изучить систему прерываний процессоров семейства C28x, а также таймеры ядра ЦСП. Научиться создавать программы, использующие прерывания.

#### Теоретические сведения

Система прерываний ядра процессора C28x 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.

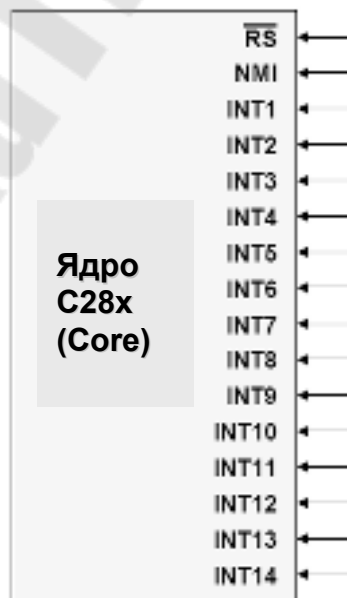


Рис. 3.1. Линии прерываний ядра C28x

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.

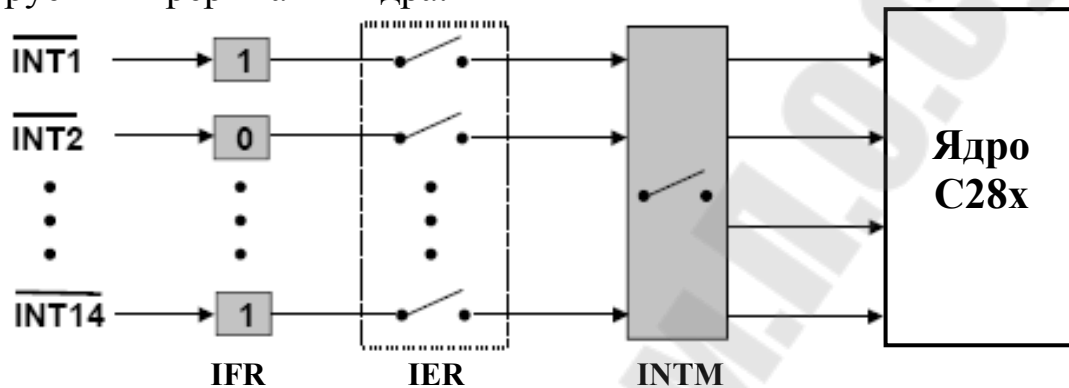


Рис. 3.2. Общая структура прерываний ядра C28x

15	14	13	12	11	10	9	8
RTOSINT	DLOGINT	INT14	INT13	INT12	INT11	INT10	INT9
7	6	5	4	3	2	1	0
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1

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

15	14	13	12	11	10	9	8
RTOSINT	DLOGINT	INT14	INT13	INT12	INT11	INT10	INT9
7	6	5	4	3	2	1	0
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1

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

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

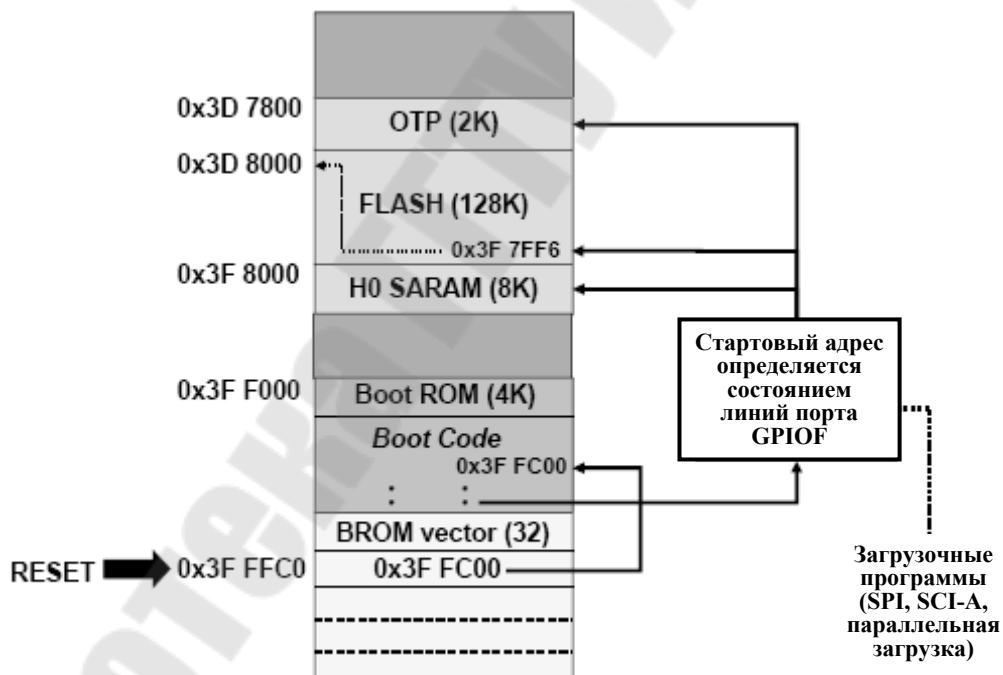


Рис. 3.4. Режимы запуска программы Bootloader

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 PIEIER<sub>x</sub> (x может принимать значения от 1 до 12), для индикации прерываний – биты в регистрах PIEIFR<sub>x</sub>. 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 PIEIER<sub>x</sub> и PIEIFR<sub>x</sub> 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).

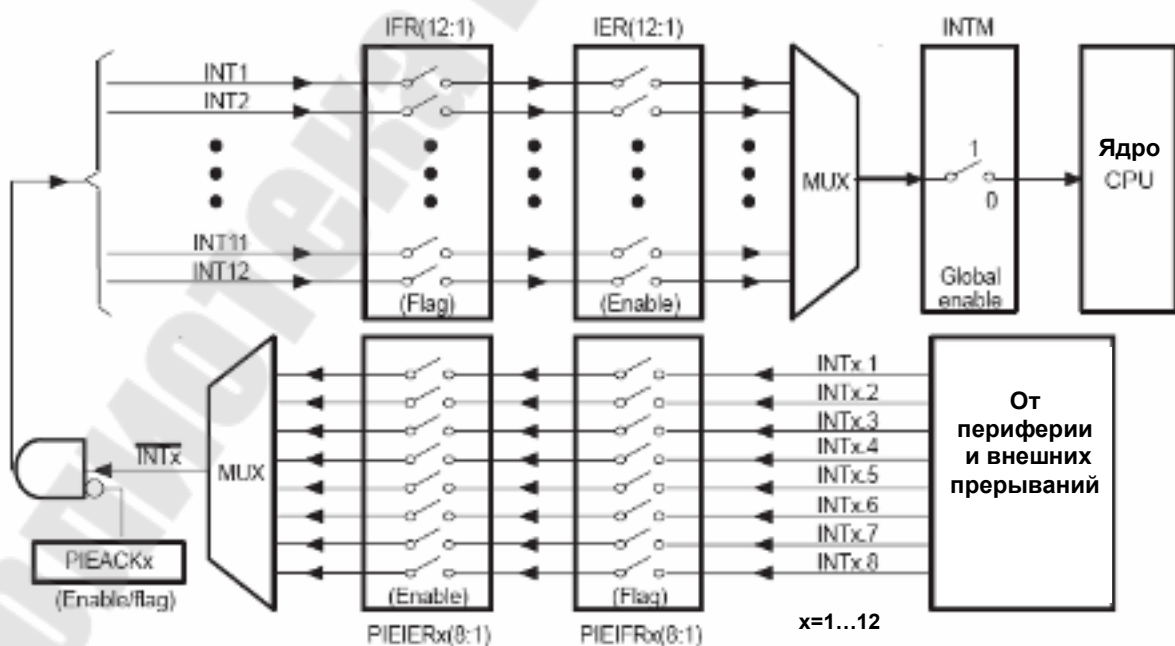


Рис. 3.5. Логика работы регистров PIEIFR<sub>x</sub>, PIEIER<sub>x</sub>, PIEACK<sub>x</sub>



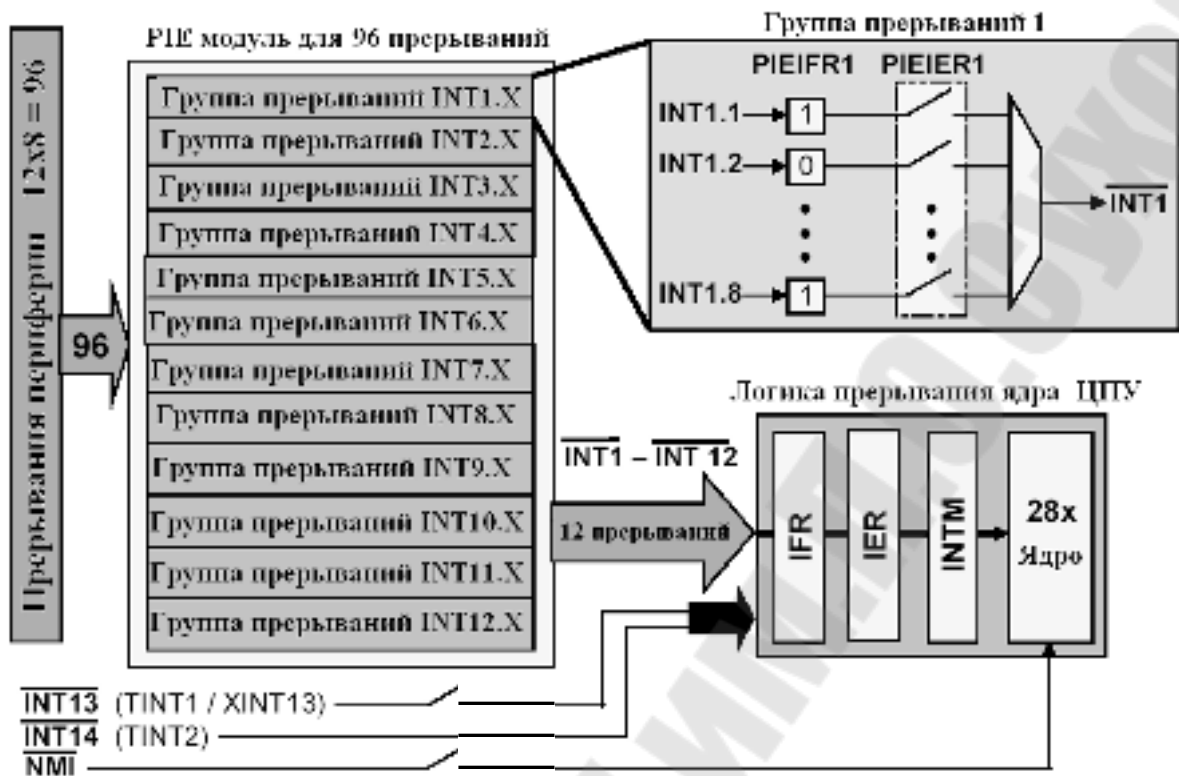


Рис. 3.6. Общая структура расширения прерываний ЦСП C28x



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.

	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).

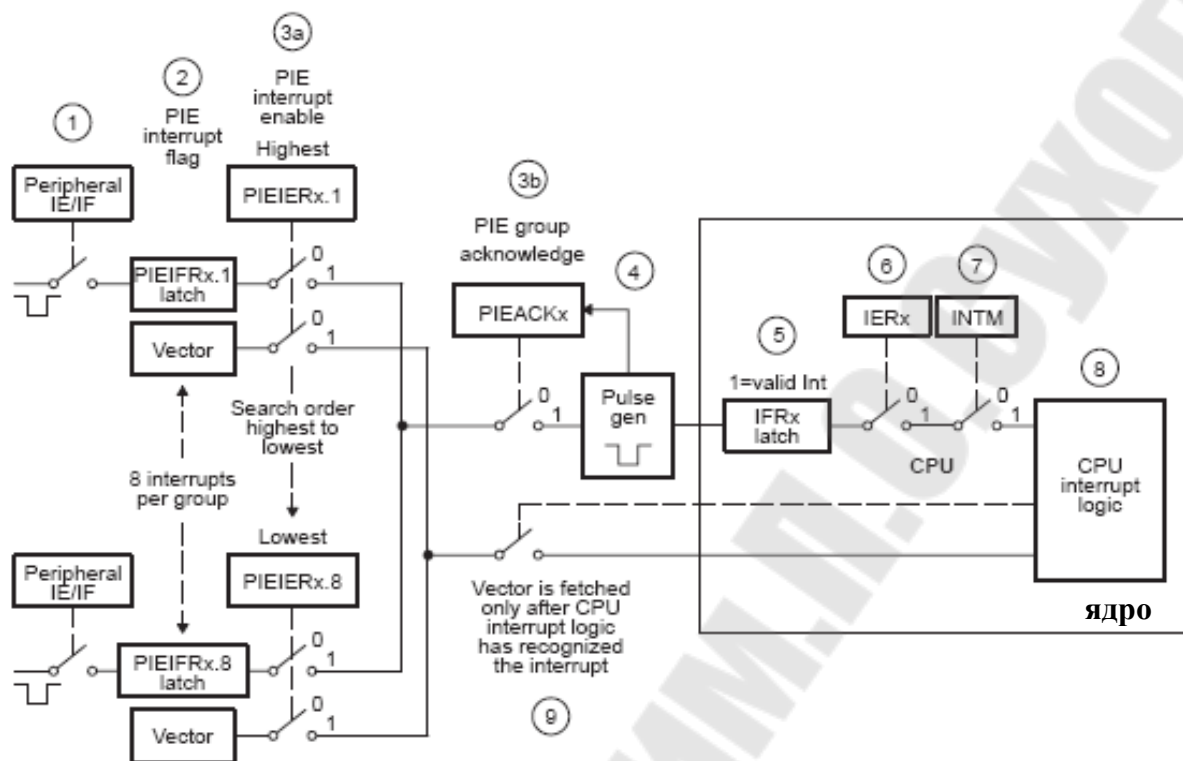


Рис. 3.10. Полная процедура обработки прерываний при  $ENPIE=1$ : шаг 1 – генерация прерывания от периферии; шаг 2 – установка флага  $PIEIFR_{x.y} = 1$ ; шаг 3а – проверка одновременного наличия двух условий:  $PIEIER_{x.y} = 1$  и  $PIEACK_x=0$ ; шаг 3б – установка в «1» бита  $PIEACK_x$  для подтверждения прерывания от группы  $x$ ; шаг 4 – формирование импульса прерывания по линии  $INT_x$  на ядро ( $PIEACK_x$  продолжает оставаться в единичном состоянии и требует программного сброса для возможности приема прерывания ядром по линии  $INT_x$  в дальнейшем); шаг 5 – установка флага  $IFR_x = 1$ ; шаг 6 – проверка условия  $IER_x = 1$ ; 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 дости-

жении 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.

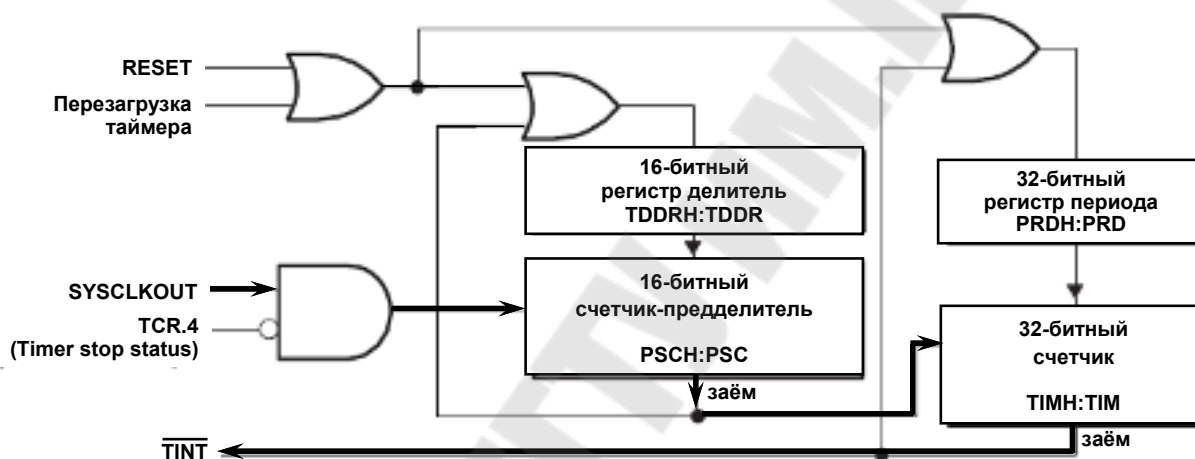


Рис. 3.11. Схема таймера ядра ЦСП семейства C28x (жирной линией показано прохождение тактового сигнала с делением частоты на выход)

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

Счетчик-предделитель  $TIMER_xPSC$  и регистр-делитель  $TIMER_xTDDR$  программно доступны как один 16-разрядный регистр  $TIMER_xTPR$  (см. рис. 3.13).

Аналогично, регистры  $TIMER_xPSCH$  и  $TIMER_xTDDRH$ , программно представляют собой единый 16-битный регистр  $TIMER_xTPRH$ .

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.

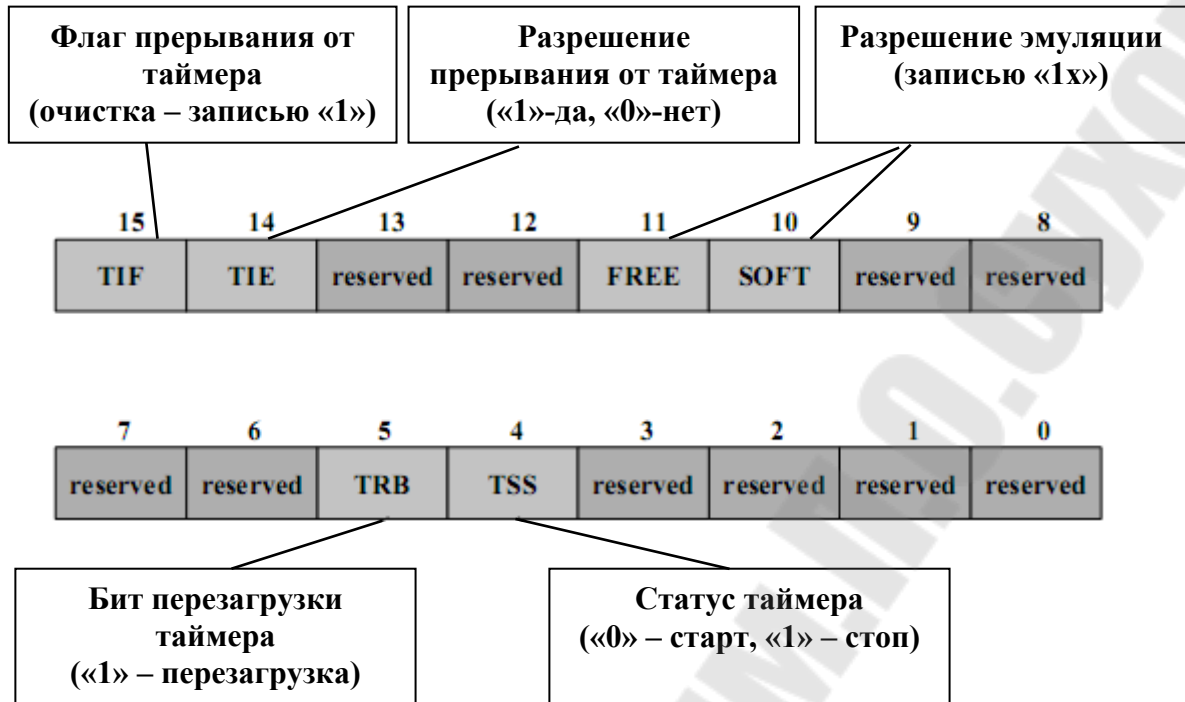


Рис. 3.14. Формат регистров управления таймерами ядра (TIMERxTCR)

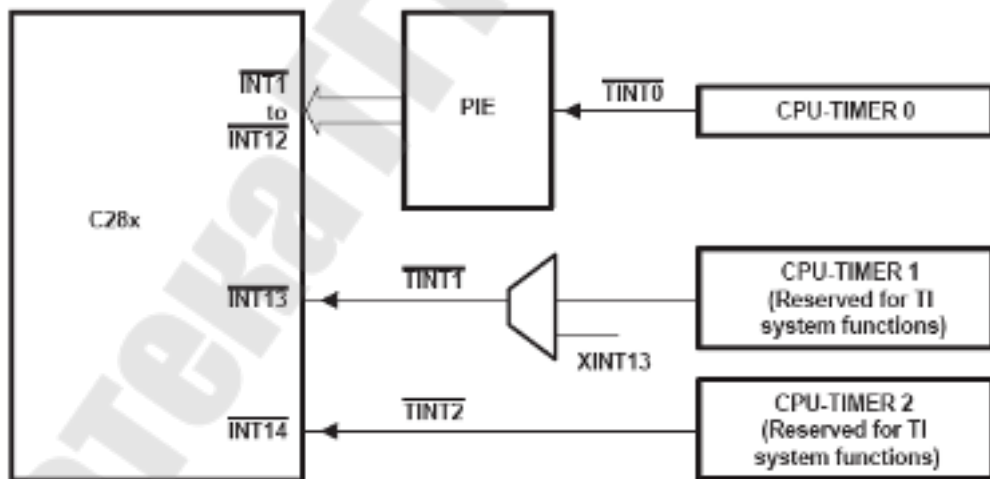


Рис. 3.15. Сигналы прерываний от CPU-таймеров

### 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:\tides\c28\dsp281x\v100\DSP281x_common\cmd\F2812_EzDSP_RAM_Ink.cmd  
C:\tides\c28\dsp281x\v100\DSP281x_headers\cmd\DSP281x_Headers_nonBIOS.cmd  
C:\CCStudio_v3.3\C2000\cgtools\lib\rts2800_ml.lib  
C:\tides\c28\dsp281x\v100\DSP281x_headers\source\DSP281x_GlobalVariableDefs.c  
C:\tides\c28\dsp281x\v100\DSP281x_common\source\DSP281x_PieCtrl.c  
C:\tides\c28\dsp281x\v100\DSP281x_common\source\DSP281x_PieVect.c  
C:\tides\c28\dsp281x\v100\DSP281x_common\source\DSP281x_DefaultIsr.c  
C:\tides\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:\tides\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); ← 5
void Gpio_select(void);
void InitSystem(void);
void main(void) ← 1
{
    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 ← 2

    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(?); ← 4
        }
    }
} ← 3

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

void delay_loop(long end)
{
    long i;
    for (i = 0; i < end; i++); ← 7
    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,
//             делитель = 1. Выработка сброса сторожевым
//             таймером. Внутренняя частота работы ЦСП 150 МГц.
//             Запись в делитель высокоскоростного таймера
//             коэффициента деления 2, а в делитель
//             низкоскоростного таймера - 4. Запрещение работы
//             периферийных устройств
//#####

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

```

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 WDKKEY 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. Компоуем 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 C28х.
4. Назначение и структура модуля расширения прерываний (PIE) ЦСП семейства C28х.
5. Расположение векторов прерываний ЦСП семейства C28х в режимах ENPIE=0 и ENPIE=1.
6. Процедура обработки прерываний при ENPIE=1.
7. Назначение и структура CPU-таймеров ЦСП семейства C28х.
8. Регистры CPU-таймеров ЦСП семейства C28х.
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?

## Лабораторная работа № 4 Исследование модуля Менеджера Событий

**Цель работы:** изучить аппаратные и программные возможности Менеджера Событий (Event Manager) DSP TMS320F2812.

### Теоретические сведения

#### Структура Менеджера Событий

Менеджер Событий (EV) включает в себя таймеры общего назначения (GP), устройства сравнения/ШИМ, устройства захвата, схему квадратурного анализа (QEP).

В сигнальном процессоре TMS320F2812 имеется два Менеджера Событий (EVA и EVB), которые выполняют аналогичные функции. EVA и EVB имеют идентичные регистры, расположенные по разным адресам.

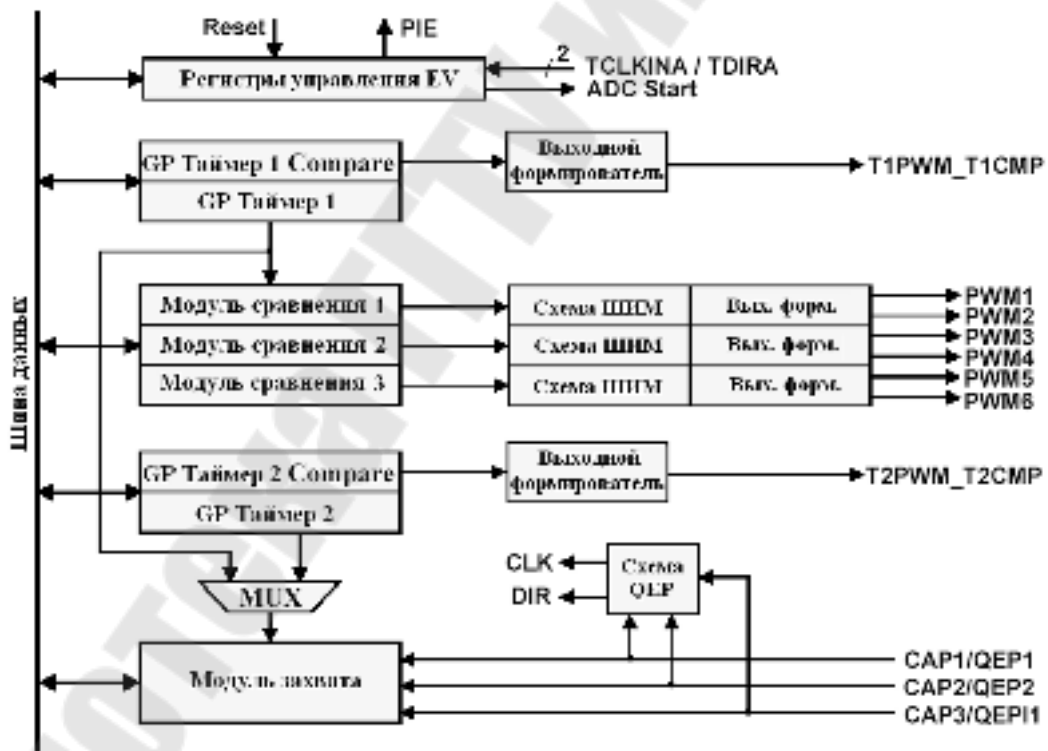


Рис. 4.1. Менеджер Событий

Каждый Менеджер Событий управляется своей собственной логикой. Она может запрашивать прерывания. Менеджер Событий позволяет запускать встроенный либо внешний аналого-цифровой преобразователь. Для запуска внешнего АЦП на выводах EVASOC или



EVBSOC (которые мультиплексируются с сигналами  $T2CTIP$  и  $T4CTIP$ ), вырабатывается строб начала преобразования (SOC). На рис. 4.1 представлена функциональная схема модуля Менеджера Событий (EVA).

Рассмотрим блоки EVM подробнее.

### Таймеры общего назначения

В каждом модуле EVM имеется по два GP (General Purpose) таймера общего назначения. В отличие от таймеров CPU, которые имеют разрядность 32-бита, таймеры Менеджера Событий являются независимыми 16-разрядными устройствами, с расширенной системой ввода/вывода. Они могут работать независимо друг от друга или быть синхронизированными.

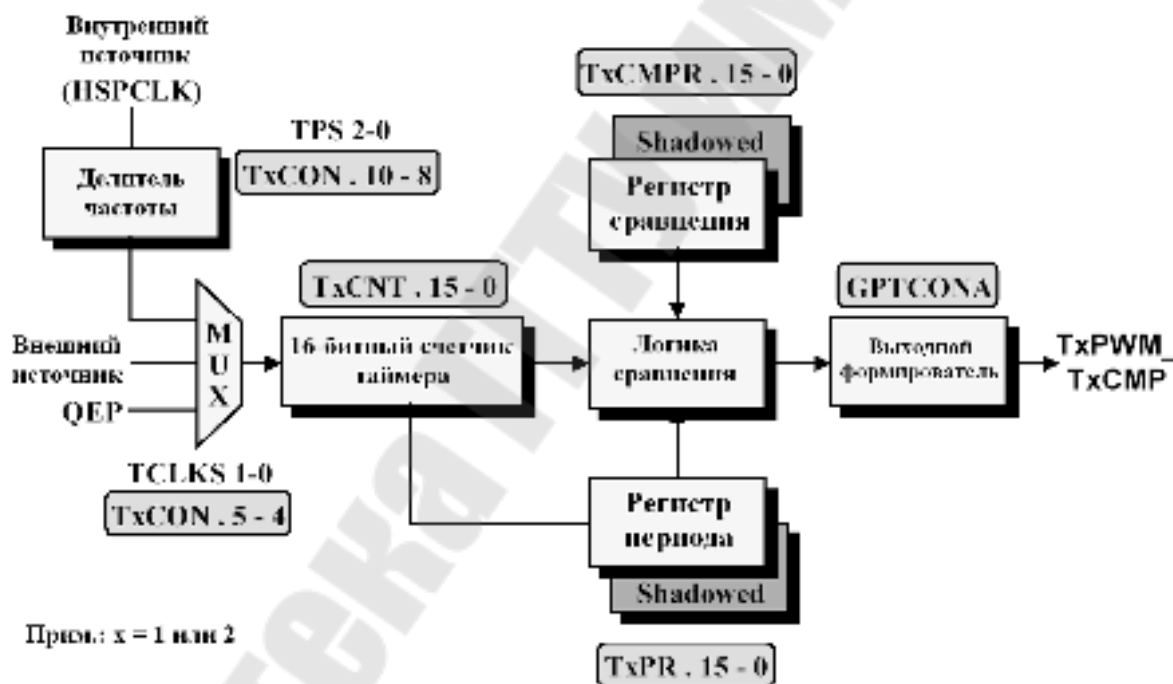


Рис. 4.2. Таймер общего назначения модуля EVM

Центральным блоком GP таймера является блок сравнения. Здесь происходит сравнение значения 16-битного счетчика (TxCNT) с двумя другими регистрами: регистром сравнения (TxCMPR) и регистром периода (TxPR). Если значения счетчика и регистра сравнения равны, то выходной формирователь устанавливает в «1» выходной сигнал (TxPWM). При достижении счетчиком значения регистра периода TxPWM сбрасывается.

Особенностью DSP TMS320F2812 является наличие буферов регистров TxCMPR и TxPR которые позволяют обновлять значения по заранее заданным событиям:

- а) достижение GP таймером-счетчиком нуля;
- б) достижение GP таймером-счетчиком значения, равного значению в регистре периода;
- в) немедленная загрузка после записи в буфер.

Наличие буферов позволяет записывать новые значения в регистры в любой момент времени, не дожидаясь окончания цикла. Загрузка значения из буфера в регистр периода происходит только при достижении GP таймером-счетчиком нуля.

Источником тактирования счетчика может являться: внешний сигнал (TCLKIN), тактовые импульсы от схемы квадратурного анализа (QEP) или тактовый сигнал от высокоскоростного предделителя (HSPCLK).

В регистрах EVAIFRA, EVAIFRB, EVBIFRA, EVBIFRB имеются биты, отражающие флаги прерывания GP таймеров. Каждый из 4-х GP таймеров может вырабатывать прерывание на следующие события:

- а) достижение GP таймером-счетчиком нуля 0000h (TxUFINT);
- б) достижение максимального значения FFFFh (TxOFINT);
- в) достижение заданного значения сравнения (TxCINT);
- г) достижение значения, равного значению в регистре периода (TxPINT).

Каждый GP таймер может работать в одном из 4-х режимов.

1) *СТОП/Хранение*. В этом режиме GP таймер останавливается и удерживает текущее значение, при этом таймер-счетчик, выходы сравнения и значение предделителя остаются без изменения. Если же остановить таймер, просто запретив его работу, то счетчик будет сброшен и значение предделителя установится в x/1.

2) *Непрерывный режим счета вверх*. В этом режиме значение счетчика увеличивается до тех пор, пока не достигнет значения, равного значению в регистре периода (рис. 4.3). После этого счетчик сбрасывается в ноль и начинает считать сначала. При этом вырабатывается флаг прерывания, который остается установленным в течение одного такта. Если флаг не был маскирован, то вырабатывается запрос прерывания от периферийного устройства.

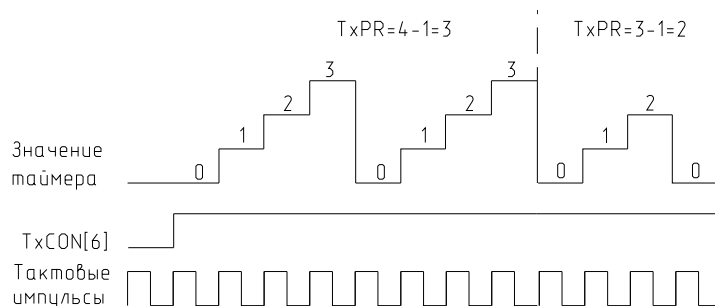


Рис. 4.3. Режим непрерывного счета вверх GP таймера

Продолжительность периода равна  $(TxPR)+1$ , за исключением первого периода, который может быть произвольным, так как начальное значение в счетчике может быть любым от 0000h до FFFFh.

Если исходное значение больше, чем значение в регистре периода, то таймер досчитает до FFFFh, сбросится в ноль и продолжит работать так, как если бы исходное значение было равно нулю. При достижении счетчиком значения, равного значению в регистре периода, устанавливается флаг прерывания по периоду, происходит сброс в ноль и устанавливается флаг прерывания по нулю.

Если исходное значение в счетчике находится между нулем и значением в регистре периода, то таймер сначала досчитает до значения в регистре периода, а затем будет работать так, как был запрограммирован.

3) *Управляемый режим счета вверх/вниз.* В этом режиме направление счета зависит от состояния входа TDIRA/B: вверх, если сигнал на TDIRA/B высокого уровня; вниз – низкого.

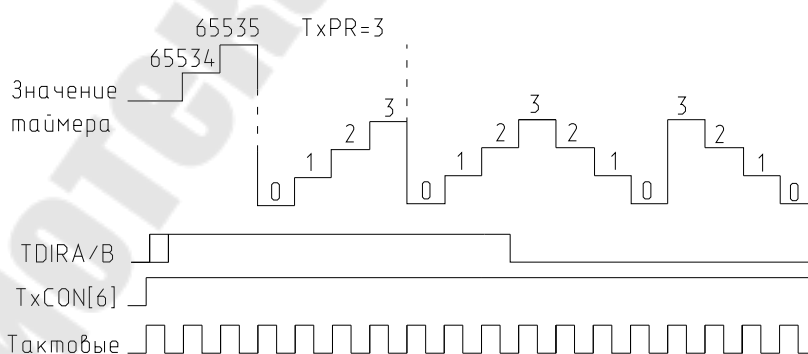


Рис. 4.4. Режим управляемого счета вверх/вниз GP таймера

4) *Непрерывный режим счета вверх/вниз.* В отличие от предыдущего режима, направление счета изменяется при достижении нуля

или значения в регистре периода. Продолжительность периода в этом режиме равна  $2 \cdot (T_{xPR})$ .

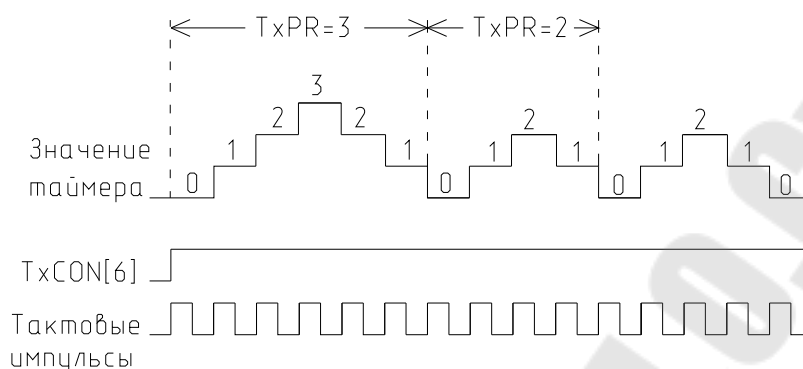


Рис. 4.5. Непрерывный режим счета вверх/вниз

**Устройство захвата (Capture Unit)** предназначено для определения временных параметров внешних сигналов. Значение выбранного GP таймера захватывается и запоминается в 2-уровневом стеке FIFO, когда на соответствующих выводах фиксируется заданный перепад уровней. Устройство захвата состоит из 3-х цепей CAP<sub>x</sub> (x=1, 2 или 3 для EVA; x=4, 5 или 6 для EVB).

Устройство захвата обладает следующими особенностями:

- 1) имеется один 16-разрядный регистр управления захватом (CAPCON<sub>x</sub>);
- 2) имеется один 16-разрядный регистр статуса FIFO (CAPFIFO<sub>x</sub>);
- 3) в качестве тактирования можно использовать любой GP таймер;
- 4) все входы синхронизируются таймерами CPU;
- 5) пользователь сам устанавливает, по какому уровню осуществлять захват;
- 6) имеется 3 маскируемых флага прерывания.

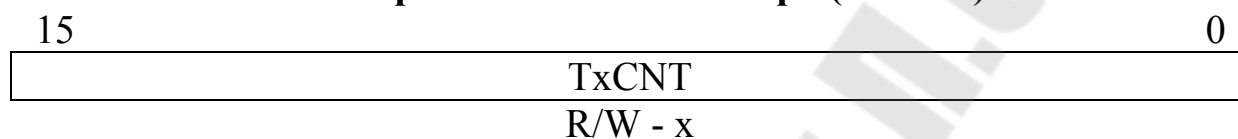
Входы CAP 1/2 и CAP 4/5 также могут быть использованы как входы схемы квадратурного анализа.

**Устройство сравнения.** В каждом EVM предусмотрено по 3 устройства сравнения (Compare Unit). Эти устройства используют GP таймер 1 в качестве синхронизатора, и могут вырабатывать до 6 выходных сигналов сравнения (ШИМ-сигналов). Все 6 выходов работают независимо друг от друга. Регистры сравнения дублируются, позволяя фиксировать изменения ширины импульсов. Они позволяют снизить до минимума программную загрузку ядра при операциях измерений длительности, периодических выборок и генерации сигналов ШИМ.

Схема квадратурного анализа используется для подключения энкодера – оптического преобразователя направления и скорости вращения. Выходными сигналами энкодера являются два сигнала типа меандр, по частоте и фазовым сдвигам которых можно определить направление и скорость вращения. Схема QEP по этим сигналам формирует два сигнала: логический сигнал направления вращения (DIR) и частотный сигнал скорости вращения (CLK)

### Формат регистров модуля Менеджера Событий

#### Регистр счетчика GP таймера (TxCNT)



Биты	Название	Описание
15:0	TxCNT	Мгновенное значение таймера-счетчика 1

Рис. 4.6. Формат регистра счетчика GP таймера.

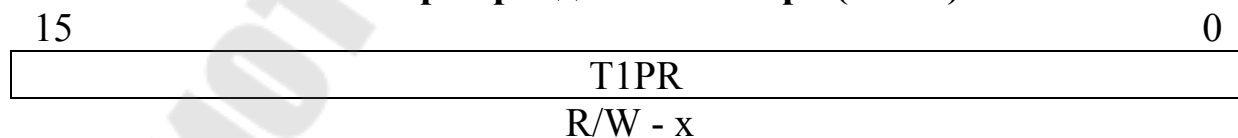
#### Регистр сравнения GP таймера (TxCMPR)



Биты	Название	Описание
15:0	TxCMPR	Хранимое значение сравнения таймера-счетчика 1

Рис. 4.7. Формат регистра сравнения GP таймера

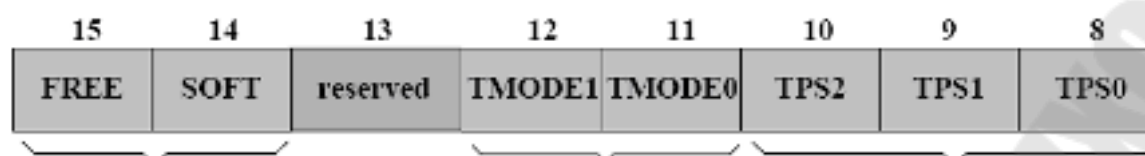
#### Регистр периода GP таймера (TxPR)



Биты	Название	Описание
15:0	TxPR	Хранимое значение периода таймера-счетчика 1

Рис. 4.8. Формат регистра периода GP таймера

## Старший байт:



### Биты управления эмулятором:

- 00 Мгновенная остановка
- 01 Остановка в конце периода
- 1x Работа без остановки

### Прескалер таймера:

- |          |            |
|----------|------------|
| 000: ÷ 1 | 100: ÷ 16  |
| 001: ÷ 2 | 101: ÷ 32  |
| 010: ÷ 4 | 110: ÷ 64  |
| 011: ÷ 8 | 111: ÷ 128 |

### Режим работы:

- 00 Стоп/ Хранение
- 01 Непрерывный счет вверх/ вниз
- 10 Непрерывный счет вверх
- 11 Управляемый счет вверх/ вниз

## Младший байт:

### Запуск таймера:

- 0 Останов таймера (сброс счетчика и предельного)
- 1 Запуск таймера

### Разрешение схемы сравнения:

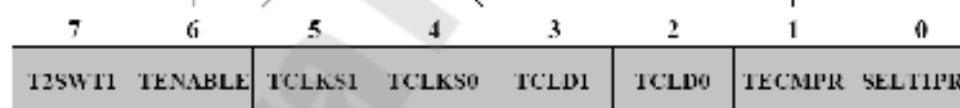
- 0 Запретить
- 1 Разрешить

### Источник тактирования:

- 00 Внутренний (ISPCLK)
- 01 Внешний (TCLKIN)
- 10 Резервный
- 11 От схемы QEP

### Выбор регистра сравнения:

- 0 Свой регистр сравнения
- 1 Регистр сравнения Таймера1



### Запуск вместе с Таймером 1

- 0 Использовать свой бит запуска (TENABLE)
- 1 Использовать TENABLE Таймера 1

### Перезагрузка регистра сравнения:

- 00 Когда счетчик равен нулю
- 01 Когда счетчик равен нулю или регистру периода
- 10 Периодически
- 11 Резервные

Рис. 4.9. Регистры управления таймером (TxCON)

## Старший байт:



## Младший байт:

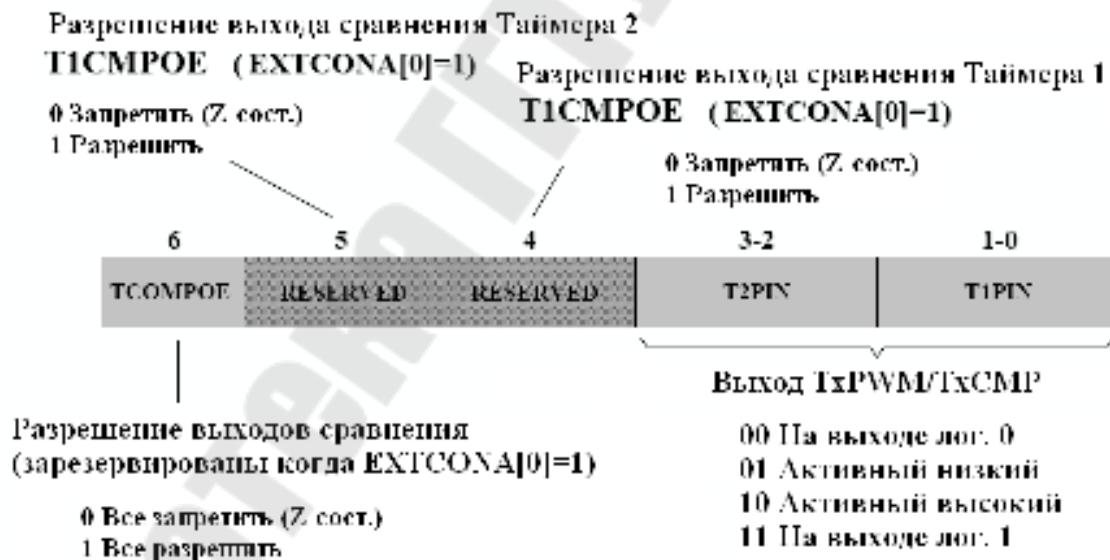
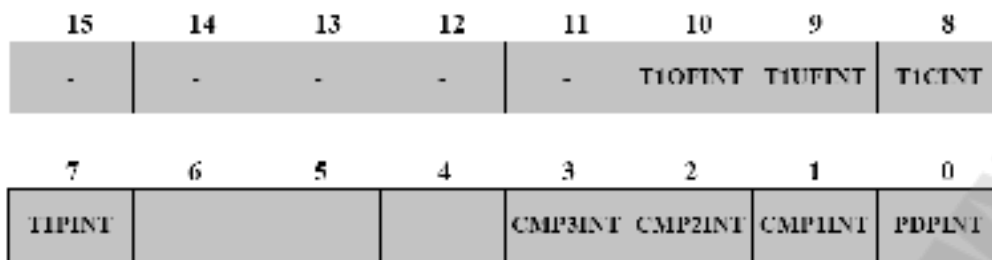
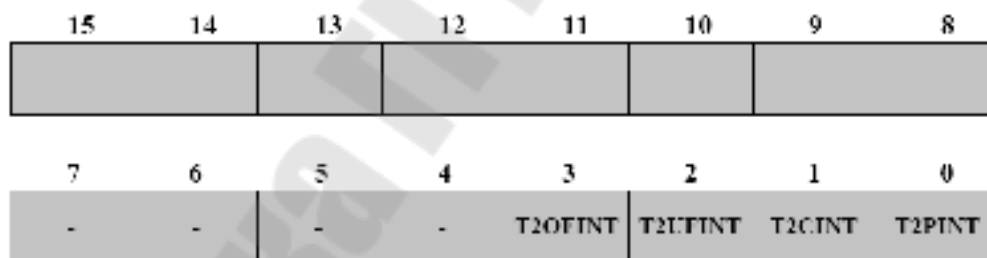


Рис. 4.10. Регистр А управления GP таймером (GPTCONA)



Разрешение прерываний:	Бит	Назначение:
0 Запретить прерывания	10:	Максимальное значение Таймера 1
1 Разрешить прерывания	9:	Минимальное значение Таймера 1
	8:	Таймер 1 равен регистру сравнения
	7:	Таймер 1 равен регистру периода
	3:	Прерывание от Блока Сравнения 1
	2:	Прерывание от Блока Сравнения 2
	1:	Прерывание от Блока Сравнения 3
	0:	Защита силового преобразователя

Рис. 4.11. Формат регистра А маски прерываний (EVAIMRA)



Разрешение прерываний:	Бит	Назначение
0 запретить прерывание	3:	Максимальное значение Таймера 2
1 разрешить прерывание	2:	Минимальное значение Таймера 2
	1:	Таймер 2 равен регистру сравнения
	0:	Таймер 2 равен регистру периода

Рис. 4.12. Формат регистра В маски прерываний (EVAIMRB)





Разрешение прерываний:	Бит	Назначение
0 - запретить прерывание	2:	От модуля Захвата 3
1 - разрешить прерывание	1:	От модуля Захвата 2
	0:	От модуля Захвата 1

Рис. 4.13. Формат регистра С маски прерываний (EVAIMRC)

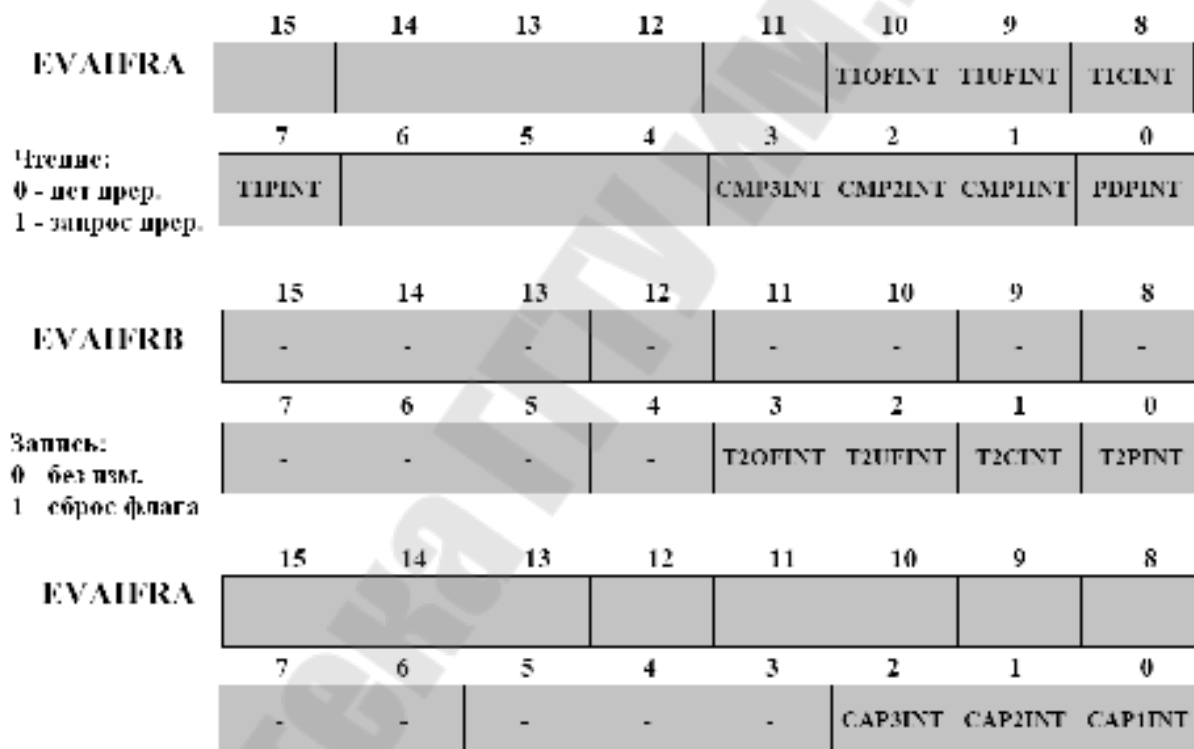


Рис. 4.14. Формат регистров флагов прерываний

### Прерывания Менеджера Событий

Если от какого-либо периферийного устройства пришло прерывание, то в регистрах EVxIFRA, EVxIFRB или EVxIFRC (x=A или B) устанавливается соответствующий флаг.

Прерывания Менеджера Событий могут быть индивидуально разрешены/ запрещены с помощью маскирования прерываний в регистрах EVxIMRA, EVxIMRB или EVxIMRC. Бит, установленный в 1 разрешает (не маскирует) прерывания, сброшенный в 0 – запрещает (маскирует). Если установлены биты флага и маскирования прерывания, то в модуль PIE (Peripheral Interrupt Expansion) отправляется запрос на прерывание. Логика PIE записывает все запросы прерывания и вырабатывает прерывание CPU.

При получении от INT 1, 2, 3, 4 или 5 запроса прерывания, устанавливается соответствующий бит в регистре флага прерывания CPU (IFR). Если соответствующий бит регистра маскирования прерываний (IER) установлен и сброшен бит INTM, то CPU признает прерывание. Далее CPU завершает выполнение текущей инструкции и переходит на адрес вектора прерываний соответственно INT 1.у, 2.у, 3.у, 4.у, 5.у в таблице вектора PIE. В это время сбрасывается бит IFR и устанавливается бит INTM, запрещая очередное прерывание.

Вывод и прерывание PDP (power drive protect) предназначены для защиты внешнего силового преобразователя от перегрузок, в т.ч. от коротких замыканий. Функция PDP позволяет защитить внешние силовые устройства при системных сбоях.

### **Примеры использования менеджера событий.**

Менеджер событий предназначен для формирования управляющих сигналов и для определения временных характеристик внешних информационных сигналов. Наличие специализированного аппаратного модуля позволяет разгрузить ЦПУ для других задач. Одно из применений TMS320F2812 - системы электропривода. Для формирования ШИМ – сигналов специальной формы (например, синусоидального) используют один из двух методов.

Первый метод предполагает вычисление функции синуса, которая входит в библиотеку «math.lib». Для этого в проект включается заголовочный файл «math.h», который позволяет обратиться к библиотеке математических функций. Но функция синуса в библиотеке «math.lib» – это функция с плавающей запятой и для её реализации на процессоре F2812, работающего с фиксированной запятой, потребуется много времени.

Второй метод основан на работе с таблицей, в которую предварительно занесены рассчитанные значения функции. Этот метод еще на-

зывают «доступ к поисковой таблице» («Lookup Table Access»). Электронные устройства контроля используют такие таблицы не только для вычисления тригонометрических функций, но и для определения параметров управления. Доступ ячейкам таблицы осуществляется быстро, всего за несколько тактов можно определить требуемое значение функции. Увеличить точность вычисления функций можно, применив аппроксимацию. Таблица может быть как одномерной (выходная величина зависит от одного параметра), так и многомерной (выходная величина - функция двух и более переменных). Чем больше число ячеек таблицы, тем точнее аппроксимация исходной функции. В ПЗУ процессора TMS320F2812 уже размещена таблица синусоидальной функции. Таблица использует дробно-целочисленный формат с фиксированной запятой IQ, или Q. Таблица представлена в формате Q30. Это означает, что 30 младших бит представляют дробную часть числа, один – целую и старший – знак числа.

## **Порядок выполнения лабораторной работы**

### **Часть I. Формирование сигналов звуковой частоты при помощи прямоугольных импульсов**

В работе необходимо сгенерировать звуковую гамму (звуковой ряд) используя динамик, соединенный с выходом T1PWM. В первой части лабораторной работы звучание организуется при помощи прямоугольных импульсов, генерируемых таймером Менеджера Событий. Каждая нота будет воспроизводиться 500 мс. Загрузка новой ноты будет производиться по прерыванию таймера ЦПУ 0. Период таймера ЦПУ – 50 мс. В подпрограмме обслуживания прерываний от ЦПУ таймера 0 необходимо производить сброс Watchdog и загружать новую ноту. Обратите внимание на период между прерываниями таймера, периодом Watchdog и периодом перезагрузки новой ноты.

Таблица. Соответствие частот музыкальным нотам

Нота	C <sup>1</sup>	D	E	F	G	A	H	C <sup>2</sup>
Частота, Гц	264	297	330	352	396	440	495	528

1. Создание нового проекта.

В Code Composer Studio создаем новый проект Lab5.pjt. Открываем файл Lab4.c и сохраняем его E:\C281x\Labs\Lab5\lab5.c.

Добавляем в проект файлы:

C:\tidcs\c28\dsp281x\v100\DSP281x\_headers\source\DSP281x\_GlobalVariableDefs.c

C:\tidcs\c28\dsp281x\v100\DSP281x\_common\cmd\F2812\_EzDSP\_RAM\_lnk.cmd;

C:\tidcs\c28\dsp281x\v100\DSP281x\_headers\cmd\F2812\_Headers\_nonBIOS.cmd;

C:\ti\c2000\cgtools\lib rts2800\_ml.lib;

C:\tidcs\c28\dsp281x\v100\DSP281x\_common\source:  
DSP281x\_CpuTimers.c, DSP281x\_PieCtrl.c, DSP281x\_PieVect.c,  
DSP281x\_DefaultIsr.c.

---

```
#####
```

```
// Имя файла: lab4_с
```

```
//
```

```
// Описание: Выработка прямоугольных импульсов. Воспроизведение
```

```
// гаммы через динамик, подключенный к выводу T1PWM (GPIO A6). Длительность звучания каждой ноты 0,5 с.
```

```
// Выработка прерываний CPU - таймером 0 каждые 50 мс.
```

```
// Сброс сторожевого таймера каждые 0,2 с.
```

```
#####
```

```
#include "DSP281x_Device.h"
```

```
void Gpio_select(void);
```

```
void InitSystem(void);
```

```
interrupt void cpu_timer0_isr(void); // Подпрограмма обработки прерываний от // CPU - таймера 0
```

```
void main(void)
```

```
{
```

```
    unsigned int i;
```

```
    unsigned long time_stamp; // Временная фиксация
```

```
    int frequency[8]={2219,1973,?,?,?,?};
```

```
    InitSystem(); // Инициализация регистров ЦСП
```

```

    Gpio_select();          // Инициализация линий ввода/вывода
    InitPieCtrl();         // Вызов подпрограммы инициализации
блока PIE
    InitPieVectTable(); // Вызов подпрограммы инициализации таб-
лицы
                                // векторов блока PIE

    EALLOW;
    PieVectTable.TINT0 = &cpu_timer0_isr;
    EDIS;

    InitCpuTimers();

    // Настройка CPU - таймера 0: частота 150 МГц, период преры-
вания 50 мс
    ConfigCpuTimer(&CpuTimer0, 150, 50000);

    // Разрешение прерывания от CPU - таймера 0, группа 1, прерыва-
ние 7
    PieCtrlRegs.PIEIER1.bit.INTx7 = 1;

    IER = 1;                // Разрешение прерываний линии 1 (INT1)
    EINT;                   // Общее разрешение прерываний INTM
    ERTM;                   // Разрешение общих прерываний в режи-
ме
                                // реального времени

    EALLOW;
    // Настройка модуля Менеджера Событий А
    EvaRegs.GPTCONA.bit.TCMPOE = ?; // Управление логикой
сравнения
                                // T1PWM /
T2PWM
    // Полярность выходов сравнения таймера 1: активный уровень -
низкий
    EvaRegs.GPTCONA.bit.T1PIN = ?;
    // Режим счета вверх таймера-счетчика модуля Менеджера Со-
бытий
    EvaRegs.T1CON.all = 0x????;

    CpuTimer0Regs.TCR.bit.TSS = ?; // Запуск CPU - таймера 0

```

```

EDIS;

i = 0;
time_stamp = 0;

while(1)
{
    if ((CpuTimer0.InterruptCount%4)==0)// Деление значения
счетчика // на 4 без
остатка
    {
        EALLOW;
        SysCtrlRegs.WDKEY = 0xAA;
        EDIS;
    }
    if ((CpuTimer0.InterruptCount - time_stamp)>10)
    {
        time_stamp = CpuTimer0.InterruptCount;
        if(i<7) EvaRegs.T1PR = frequency[i++];
        else EvaRegs.T1PR = frequency[14-i++];

        EvaRegs.T1CMPR = EvaRegs.T1PR/2;
        EvaRegs.T1CON.bit.TENABLE = 1; // Разрешение
работы //
таймера модуля EVA
        if (i>=14) i=0;
    }
}

#####
// 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 = ?;           // Настройка линий
ввода/вывода на
    GpioMuxRegs.GPBMUX.all = 0x0;       // работу в качестве пор-
ТОВ
    GpioMuxRegs.GPDMUX.all = 0x0;
    GpioMuxRegs.GPFMUX.all = 0x0;
    GpioMuxRegs.GPEMUX.all = 0x0;
    GpioMuxRegs.GPGMUX.all = 0x0;

    GpioMuxRegs.GPADIR.all = 0x0;       // Настройка портов A, D,
E, F, G на ввод
    GpioMuxRegs.GPBDIR.all = 0x00FF; // Настройка линий порта B 15-
8 на ввод,
    GpioMuxRegs.GPDDIR.all = 0x0;      // а линий 7-0 на вывод
    GpioMuxRegs.GPEDIR.all = 0x0;
    GpioMuxRegs.GPFDIR.all = 0x0;
    GpioMuxRegs.GPGDIR.all = 0x0;

    GpioMuxRegs.GPAQUAL.all = 0x0;     // Запрещение входного
ограничителя
    GpioMuxRegs.GPBQUAL.all = 0x0;
    GpioMuxRegs.GPDQUAL.all = 0x0;
    GpioMuxRegs.GPEQUAL.all = 0x0;
    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= 0x00AF;           // Разрешение работы сторо-
рожевого таймера,
предделитель = 64
// 0x00E8 - запреше-
ние работы сторо-
// жевого таймера,
предделитель = 1

    SysCtrlRegs.SCSR = 0;             // Выработка сброса WDT

    SysCtrlRegs.PLLCR.bit.DIV = 10; // Настройка блока умножения
частоты
    SysCtrlRegs.HISPCP.all = 0x1;    // Задание значения преддели-
теля высо-
    SysCtrlRegs.LOSPCP.all = 0x2;    // коскоростного и низкоскоро-
стного
// таймеров

    // Разрешение / запрещение тактирования периферийных устройств
    SysCtrlRegs.PCLKCR.bit.EVAENCLK = ?;
    SysCtrlRegs.PCLKCR.bit.EVBENCLK = 0;
    SysCtrlRegs.PCLKCR.bit.SCIAENCLK = 0;
    SysCtrlRegs.PCLKCR.bit.SCIBENCLK = 0;
    SysCtrlRegs.PCLKCR.bit.MCBSPENCLK = 0;
    SysCtrlRegs.PCLKCR.bit.SPIENCLK = 0;
    SysCtrlRegs.PCLKCR.bit.ECANENCLK = 0;
    SysCtrlRegs.PCLKCR.bit.ADCENCLK = 0;

    EDIS;
}

#####
// Подпрограмма: cpu_timer0_isr
//
// Описание: Обработка прерывания по CPU - таймеру 0
#####

interrupt void cpu_timer0_isr(void)

```



```

{
    CpuTimer0.InterruptCount++;

    EALLOW;
    SysCtrlRegs.WDKEY = 0x55;           // Сброс сторожевого
таймера
    EDIS;

    // Подтверждение прерывания от группы прерываний 1
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}

```

Рис. 4.15. Структура текста программы

2. Настройка параметров проекта, компоновка проекта и загрузка выходного файла.

Включаем в проект заголовочные файлы: 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*

Задаем глубину стека: Project → Build Options → Linker → Stack Size: 0x400.

Компонуем проект: Project → Build.

Загружаем выходной файл: File → Load Program → Debug\lab5.out.

3. Преобразование файла Lab5.c.

Удаляем те части программы, которые не будут использоваться данной лабораторной работе: массив LED[8].

Переходим к подпрограмме «Gpio\_select()». Настраиваем линию 6 порта GPIOA на вывод функции T1PWM (GPAMUX).

В подпрограмме «InitSystem» разрешаем работу Менеджера Событий А.

Внутри главной подпрограммы записываем:

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

Настраиваем таймер 1 Менеджера Событий на выработку модулированного сигнала PWM:

Бит «TCMPOE» устанавливаем в 1, задаем “активный низкий” уровень (GPTCONA);

В регистре T1CON устанавливаем: режим счета вверх (TMODE), делитель 128 (TPS), запрещаем работу таймера (TENABLE), выбираем внутреннюю синхронизацию (TCLKS), разрешаем сравнение (TECMPR);

Определяем значения регистра периода (T1PR) для разных частот и заносим их в массив `int frequency [8] = {?,?,?,?,?,?,?,?}`. Исходя из формулы  $T1\_PWM\_Freq = 150MHz / (HISPCP * TPS * T1PR)$  для первой ноты в регистр периода необходимо занести:  $T1PR = 150MHz / (2 * 128 * 264 \text{ Гц}) = 2219$ , для второй ноты  $T1PR = 150MHz / (2 * 128 * 297 \text{ Гц}) = 1973$  и т.д.

Так как сторожевой таймер будет сбрасываться быстрее (каждые 200 мс), чем прозвучит следующая нота (через 500 мс), то задаем условия:

```
if ((CpuTimer0.InterruptCount%4) == 0)
SysCtrlRegs.WDKEY = 0xAA
```

Разбиваем период воспроизведения нот на 10 периодов по 50 мс, для этого записываем условие: `if ( (CpuTimer0.InterruptCount – time_stamp) > 10)`.

После чего переменной «time\_stamp» присваиваем текущее значение `CpuTimer0.InterruptCount`, загружаем код следующей ноты в T1PR, а в регистр T1CMPR заносим значение, равное T1PR/2. Разрешаем работу таймера.

---

```
#####
//  Имя файла:      _lab5A_.c
//
//  Описание: Выработка синусоидальных колебаний на выводе
T1PWM
//              (GPIO A6). Выработка прерываний CPU - таймером 0
//              каждые 50 мс. Сброс сторожевого таймера каждые 0,2 с.
#####

#include "DSP281x_Device.h"
#include "IQmathLib.h"                // Включение математиче-
ских библиотек
#pragma DATA_SECTION(sine_table,"IQmathTables");
_iq30 sine_table[512];
```

```

void Gpio_select(void);
void InitSystem(void);
interrupt void T1_Compare_isr(void); // Подпрограмма обработки
прерываний
// таймера сравне-
ния 1 EVA
int Count = 0;

void main(void)
{
    InitSystem(); // Инициализация регистров ЦСП
    Gpio_select(); // Инициализация линий ввода/вывода
    InitPieCtrl(); // Инициализации блока PIE
    InitPieVectTable(); // Инициализация таблицы векторов блока
    PIE

    EALLOW;
    PieVectTable.T1CINT = &T1_Compare_isr;
    EDIS;

    // Разрешение прерывания от таймера сравнения 1, группа 2,
    прерывание 5
    PieCtrlRegs.PIEIER2.bit.INTx5 = 1;

    IER = 2; // Разрешение прерываний линии 2 (INT2)
    EINT; // Общее разрешение прерываний INTM
    ERTM; // Разрешение общих прерываний в режи-
    ме
    // реального времени

    // Настройка модуля Менеджера Событий А

    EvaRegs.GPTCONA.bit.TCMPOE = ?; // Управление логикой
    сравнения
    // T1PWM /
    T2PWM
    // Полярность выходов сравнения GP таймера 1: активный уро-
    вень - низкий
    EvaRegs.GPTCONA.bit.T1PIN = ?;

```

```

    EvaRegs.T1CON.bit.FREE = ?;           // Биты управления эму-
лятором:
    EvaRegs.T1CON.bit.SOFT = ?;          // мгновенная остановка
    EvaRegs.T1CON.bit.TMODE = ?;        // Непрерывный счет вверх
    EvaRegs.T1CON.bit.TPS = ?;          // Предделитель = 1 : 75 МГц
    EvaRegs.T1CON.bit.TENABLE = ?;      // Запрещение работы GP тай-
мера 1
    EvaRegs.T1CON.bit.TCLKS10 = ?;      // Внутренний источник
тактирования
    EvaRegs.T1CON.bit.TCLD10 = ?; // Перезагрузка GP таймера
// при достижении
нуля
    EvaRegs.T1CON.bit.TECMPR = ?;       // Разрешение режима
сравнения

    EvaRegs.T1PR = 1500;                // Регистр периода GP
таймера

    EvaRegs.T1CMPR = EvaRegs.T1PR/2;

    EvaRegs.EVAIMRA.bit.T1CINT = ?;     // Разрешение прерывания
при
// достижении зна-
чения сравнения
    EvaRegs.T1CON.bit.TENABLE = ?;     // Разрешение работы GP
таймера 1

    while(1)
    {
        EALLOW;
        SysCtrlRegs.WDKEY = 0xAA;
        EDIS;
    }
}

#####
// Подпрограмма: T1_Compare_isr
//
// Описание: Обработка прерывания по GP таймеру 1
#####

```

```

interrupt void T1_Compare_isr(void)
{
    static int index=0;

    Count++;

    EALLOW;
    SysCtrlRegs.WDKEY = 0x55;
    EDIS;

    EvaRegs.T1CMPR      =      EvaRegs.T1PR      -
    _IQsat(_IQ30mpy(sine_table[index]+_IQ30(0.9999),EvaRegs.T1PR/2),Eva
    aRegs.T1PR,0);

    index +=4;          // Использование каждого 4-го значения
таблицы
    if (index >511) index = 0;

    // Сброс флага прерывания GP таймера сравнения
    EvaRegs.EVAIFRA.bit.T1CINT = 1;

    // Подтверждение прерывания от группы прерываний 2
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP2;
}

#####
// 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 = ?; // Настройка линий вво-
да/вывода на
    // Настройка линии порта GPIO A6 на работу в качестве специ-
альной

```

```

// функции (T1PWM) - вывод прямоугольных импульсов

GpioMuxRegs.GPBMUX.all = 0x0; // работу в качестве портов
GpioMuxRegs.GPDMUX.all = 0x0;
GpioMuxRegs.GPFMUX.all = 0x0;
GpioMuxRegs.GPEMUX.all = 0x0;
GpioMuxRegs.GPGMUX.all = 0x0;

GpioMuxRegs.GPADIR.all = 0x0; // Настройка портов A, D, E, F,
G на ввод
    GpioMuxRegs.GPBDIR.all = 0x00FF; // Настройка линий 15-8
на ввод, а линий
    GpioMuxRegs.GPDDIR.all = 0x0; // 7-0 на вывод
    GpioMuxRegs.GPEDIR.all = 0x0;
    GpioMuxRegs.GPFDIR.all = 0x0;
    GpioMuxRegs.GPGDIR.all = 0x0;

GpioMuxRegs.GPAQUAL.all = 0x0; // Запрещение входного огра-
нителя
GpioMuxRegs.GPBQUAL.all = 0x0;
GpioMuxRegs.GPDQUAL.all = 0x0;
GpioMuxRegs.GPEQUAL.all = 0x0;
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= 0x00AF;           // Разрешение работы сторожевого таймера, предделитель = 64
    // 0x00E8 - запрещение работы сторожевого таймера,
    // предделитель = 1
    SysCtrlRegs.SCSR = 0;               // Выработка сброса WDT
    SysCtrlRegs.PLLCR.bit.DIV = 10;    // Настройка блока умножения частоты
    SysCtrlRegs.HISPCP.all = 0x1;      // Задание значения предделителя высокого-
    SysCtrlRegs.LOSPCP.all = 0x2;      // коскоростного и низкоскоростного таймеров
    SysCtrlRegs.PCLKCR.bit.EVAENCLK=?; // Запрещение работы периферийных устройств
    SysCtrlRegs.PCLKCR.bit.EVBENCLK=0;
    SysCtrlRegs.PCLKCR.bit.SCIAENCLK=0;
    SysCtrlRegs.PCLKCR.bit.SCIBENCLK=0;
    SysCtrlRegs.PCLKCR.bit.MCBSPENCLK=0;
    SysCtrlRegs.PCLKCR.bit.SPIENCLK=0;
    SysCtrlRegs.PCLKCR.bit.ECANENCLK=0;
    SysCtrlRegs.PCLKCR.bit.ADCENCLK=0;
    EDIS;
}

```

Рис. 4.16. Структура текста программы

#### 4. Тестирование программы.

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

Переходим к главной подпрограмме: Debug → Go main.

Запускаем программу: Debug → Run.

### **Часть II. Формирование сигналов звуковой частоты при помощи синусоидальных импульсов**

Во второй части работы необходимо вывести синусоидальный ШИМ - сигнал. Отметим, что музыкальная нота – это гармоническое синусоидальное колебание определенной частоты, поэтому, изучив принцип получения синусоидальных колебаний, можно получить чистоту звучания нот, близкую к реальности. В таблице представлено соответствие частот и нот музыкального звукоряда.

Для расчета частоты синусоидальных колебаний используется формула:

$$f_{sin} = \frac{f_{ШИМ}}{N_{ШИМ}} \quad (4.1)$$

где  $f_{ШИМ}$  - частоты модулированных импульсов;  
 $N_{ШИМ}$  - число модулированных импульсов за период.

### 1. Создание нового проекта.

#### 1.1 Создаем новый проект Lab5A.pjt.

1.2 Добавляем в проект файлы (см. п. 1.2 части I), кроме файла *DSP281x\_CpuTimers.c*, и вместо *lab5.c* загружаем *lab5A.c*.

2. Настройка параметров проекта, компоновка проекта и загрузка выходного файла (аналогично п.п. 2.1 – 2.4 части I данной лабораторной работы). В закладке Compiler выбираем Preprocessor и в поле Include Search Path (-i) вводим:

```
C:\tidcs\C28\dsp281x\v100\DSP281x_headers\include;  
..\include; C:\tidcs\C28\IQmath\cIQmath\include
```

### 3. Преобразование файла Lab5A.c.

Удаляем те строки программы, которые касаются работы таймера 0 ядра процессора F2812: `ConfigCpuTimer(&CpuTimer0, 150, 50000), InitCpuTimers(), CpuTimer0Regs.TCR.bit.TSS = 0`, переменные *i* и *time\_stamp*, массив *frequency* [8].

Переименовываем подпрограмму “*cpu\_timer0\_isr()*” в “*T1\_Compare\_isr*” и в ней добавляем строку: `PieCtrlRegs.PIEACK.all = PIEACK_GROUP2`, устанавливаем бит T1CINT регистра EVAIFRA в 1.

Добавляем инструкцию, разрешающую в модуле расширения прерываний PIE прерывание от GP таймера 1: `PieCtrlRegs.PIEIER2.bit.INTx5=1` и устанавливаем разрешение прерывания ядра процессора INT2: `IER = 2`.

Настраиваем таймер 1 Менеджера Событий (аналогично пп. 3.5.1 – 3.5.2 части I): записываем коэффициент деления равный 1, устанавливаем бит T1CINT регистра EVAIMRA в 1.



В регистр периода (T1PR) заносим значение, рассчитанное по формуле:

$$f_{PWM} = \frac{f_{CPU}}{T1PR \cdot TPS_{T1} \cdot HISCP} \quad (4.2)$$

где  $f_{PWM} = 50 \text{ кГц}$ ,  $f_{CPU} = 150 \text{ МГц}$ ,  $HISCP = 2$ ,  $TPS = 1$ .

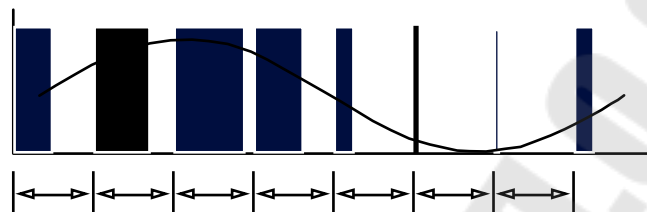


Рис. 4.17. Принцип формирования синусоидального сигнала широтно-импульсной модуляцией

#### 4. Тестирование программы.

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

Переходим к главной подпрограмме: Debug → Go main.

Запускаем программу: Debug → Run.

Устанавливаем точку останова на строчке:

```
PieCtrlRegs.PIEACK.all = PIEACK_GROUP2.
```

#### 5. Включение библиотеки “IQ-Math” в проект.

Добавляем в начальную часть текста программы строки:

```
#include "IQmathLib.h",
 IQ30 sine_table[512],
#pragma DATA_SECTION(sine_table, "IQmathTables");
```

и добавляем файл Lab5A.cmd.

#### 6. Расчет необходимого значения регистра сравнения.

Учитывая, что разность значений регистра периода и регистра сравнения определяет ширину выходного модулированного импульса (см. рис. 4.15), а значения функции синуса изменяются от минус 1 до плюс 1, рассчитываем значение регистра сравнения по формуле:

$$T1CMPR = T1PR - \text{IQ30mpy}(\text{sine\_table}[\text{index}] + \text{IQ30}(0.9999), T1PR/2) \quad (4.3)$$

где функция  $\text{IQ30mpy}(a, b)$  – умножение чисел  $a$  и  $b$  в формате  $\text{IQ30}$ ;  $\text{\_IQ30}(0.9999)$  – оттранслированное значение единицы.

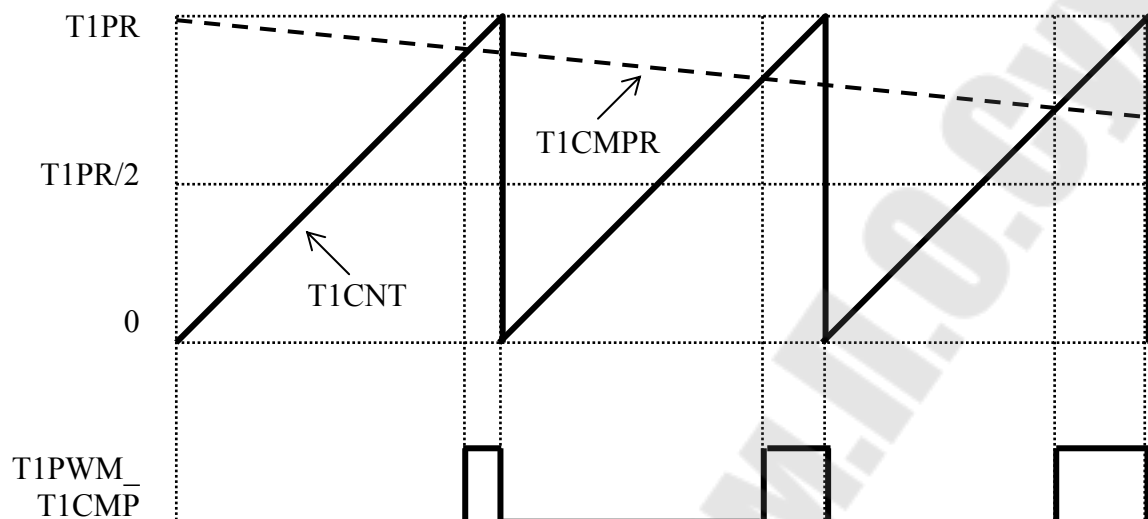


Рис. 4.18. Изменение ширины импульса при ШИМ по синусоидальному закону

Чтобы избежать переполнения, применим функцию насыщения  $\text{\_IQsat}(x, \text{max}, \text{min})$ . Тогда окончательно записываем в подпрограмму  $\text{T1\_Compare\_isr}$  формулу для расчета необходимого значения регистра сравнения:

```
EvaRegs.T1CMPR = EvaRegs.T1PR -
_IQsat(_IQ30mpy(sine_table[index]+_IQ30(0.9999),
EvaRegs.T1PR/2),EvaRegs.T1PR,0) .
```

#### 7. Расчет частоты синусоидальных колебаний.

Для этого используем формулу (1), задаемся  $f_{\text{ШИМ}} = 50$  кГц,  $N_{\text{ШИМ}} = 128$ , т.е. из таблицы значений синуса выбираем каждое четвертое число, добавляем строку:

```
index += 4.
```

Тогда  $f_{\text{sin}} = 390,6$  Гц.

Компонуем проект: Project → Rebuild All.

### **Содержание отчета:**

Отчет должен содержать цель работы, краткие теоретические сведения по теме работы, рисунки, поясняющий принцип формирования ШИМ – сигналов, тексты исследуемых программ, результаты их выполнения, выводы.

### **Контрольные вопросы:**

1. Назначение и структура Менеджера Событий DSP TMS320F2812.
2. Структура таймеров Менеджера Событий, их отличие от таймеров ЦПУ по структуре и назначению.
3. Режимы работы таймеров Менеджера Событий.
  1. Регистры Менеджера Событий.
  2. ШИМ – сигнал: определение, формирование с помощью Менеджера Событий.
  3. Прерывания Менеджера Событий.

## Лабораторная работа № 5 Исследование модуля АЦП

**Цель работы:** изучить структуру встроенного АЦП, режимы его работы. Научится создавать простейшие программы с использованием АЦП.

### Теоретические сведения

#### Структура модуля АЦП

Модуль АЦП содержит ядро АЦП, два устройства выборки – хранения, аналоговый мультиплексор, мультиплексор УВХ, мультиплексор результата и автоматический секвенсер (устройство управления работой АЦП и мультиплексоров). 12- битный АЦП имеет 16 мультиплексированных входов. АЦП может работать либо в каскадном, либо в двухканальном режимах.

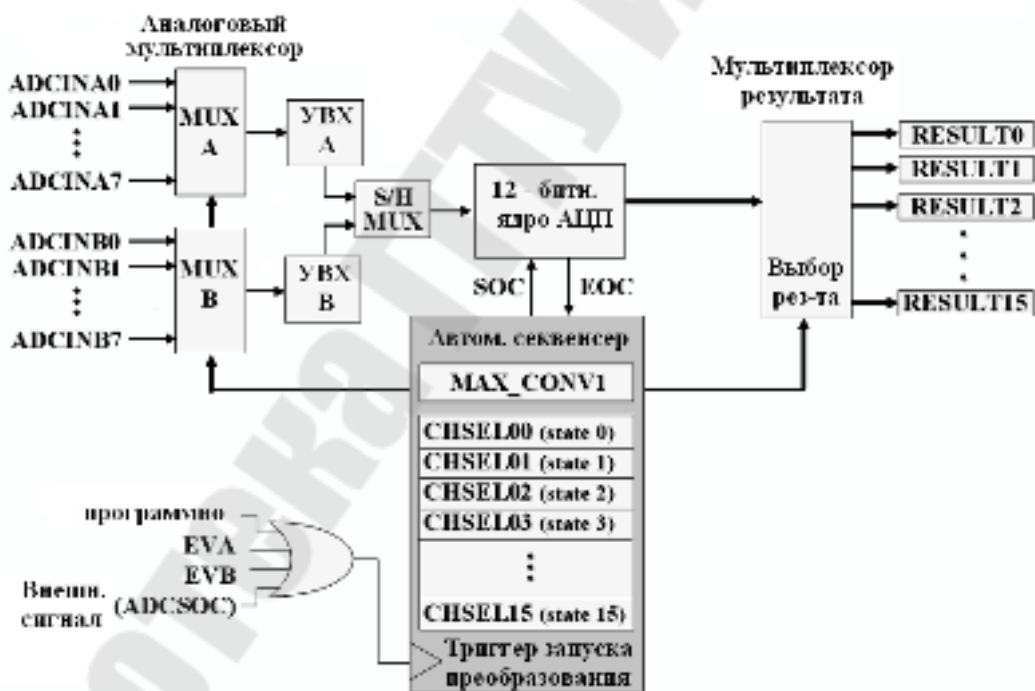


Рис. 5.1. Блок-схема модуля АЦП в каскадном режиме

Структура АЦП в *каскадном режиме* представлена на рис. 5.1. Как видно из рисунка, в этом режиме работой модуля управляет один автоматический секвенсер. Перед запуском необходимо задать число преобразований (“MAX\_CONV1”) и номер канала, который будет

преобразован на каждом шаге (“CHSELxx”). Результат преобразования на каждом шаге сохраняется в соответствующий регистр (“RESULT0” to “RESULT15”).

Можно задать два режима захвата сигналов – одновременный и последовательный. В первом случае два УВХ работают в параллель, т.е. выборка и захват происходят одновременно. При этом за один шаг осуществляется преобразование двух каналов различных групп с одинаковым кодом (например, ADCINA3 и ADCINB3). В последовательном режиме сигнал с любого входа может быть преобразован на любом шаге. Запуск преобразования может осуществляться программно, от внешнего источника или от менеджеров событий А или В. Запуск от Менеджера Событий осуществляется аппаратно, без использования прерываний, что позволяет очень точно задавать интервал преобразования. Прерывания от АЦП для обработки результатов могут быть сконфигурированы либо после каждого преобразования, либо по окончании преобразования последовательности.

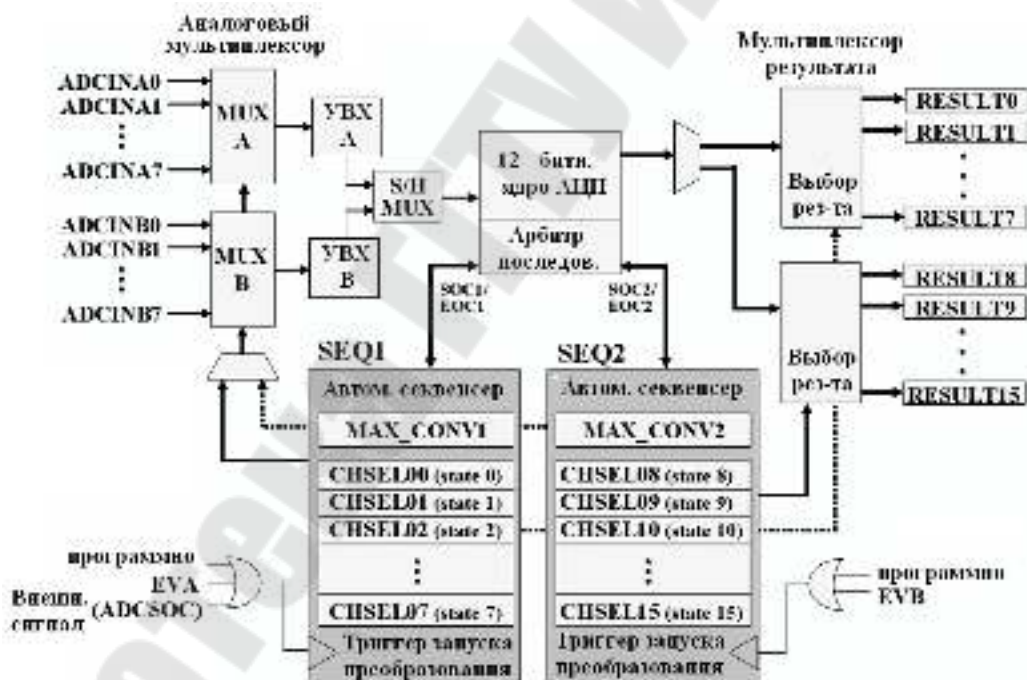


Рис. 5.2. Блок-схема модуля АЦП в двухканальном режиме

В *двухканальном режиме* автоматический секвенсер разделяется на две независимые части (“SEQ1” и “SEQ2”) со своими настройками и сигналами запуска. Входные каналы задаются в регистрах CHSEL00 .. CHSEL07 для последовательности SEQ1 и CHSEL08 .. CHSEL15

для последовательности SEQ2, результаты преобразования сохраняются в регистрах RESULT0 .. RESULT7 и RESULT8 .. RESULT15 соответственно. Для любой из двух последовательностей может быть задан любой из 16 входных каналов. Данный режим позволяет получить фактически два независимых АЦП, со своими регистрами управления и сигналами запуска. Арбитр последовательности используется в случае одновременного появления сигналов запуска от двух последовательностей. В таком случае приоритет имеет SEQ1, преобразование SEQ2 будет задержано до окончания SEQ1.

### Время преобразования и система тактирования АЦП.

Рассмотрим настройку тактирования АЦП на примере. Тактирование АЦП осуществляется от высокоскоростного прескалера периферии HSPCLK (см. рис. 5.3).

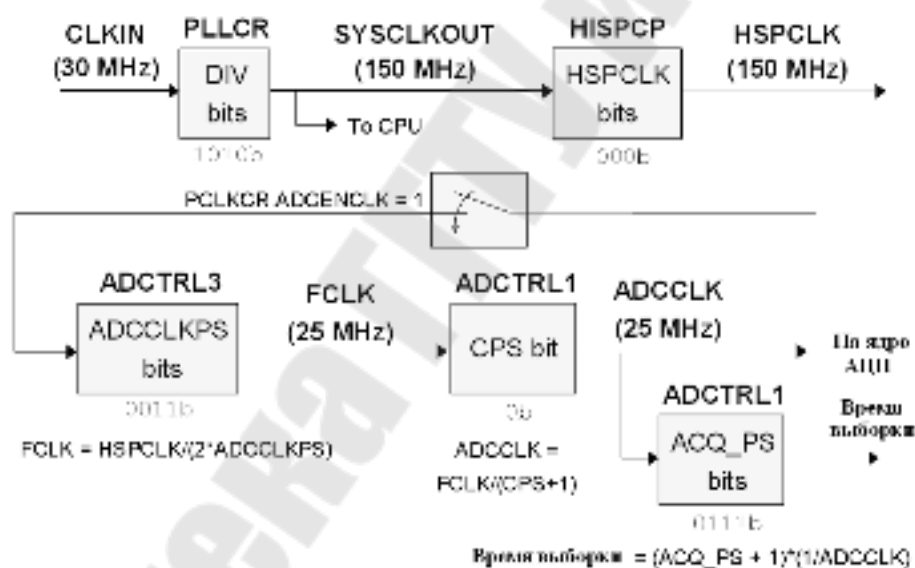


Рис. 5.3. Пример конфигурации модуля тактирования АЦП

Максимальная частота на выходе HSPCLK составляет 150 МГц. Максимальная частота тактирования АЦП FCLK согласно документации составляет 25 МГц. ADCCLKPS служит для формирования требуемой частоты FCLK из HSPCLK. Бит CPS позволяет при необходимости понизить частоту в два раза. Полученная ADCCLK подается на ядро АЦП и на устройство выборки-хранения. Битами ACQ\_PS можно задать необходимое время захвата сигнала. За это

время сигнал на УВХ должен установиться с заданной точностью. Время захвата зависит от сопротивления источника сигнала, характеристик сигнала, требуемой точности и рассчитывается для каждого случая отдельно. В лабораторных работах сигнал подается с потенциометра, высокая точность не требуется, поэтому время выборки может быть задано любое.

Минимальное время преобразования для первого канала в последовательности составляет 200 мкс, для последующих - 80 мкс.

### 3. Формат регистров модуля АЦП



Рис. 5.4. Регистр 1 управления АЦП (ADCTRL1)

### Старший байт

Запуск АЦП от EVB:

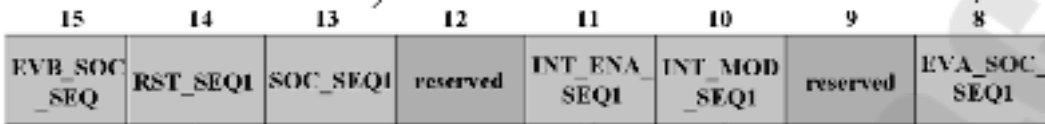
0 - нет запуска от EVB  
1 - запуск АЦП от EVB  
(только в каскадном режиме)

Запуск преобразования SEQ1:

0 - сброс сигнала запуска  
1 - прогр. запуск преобр. SEQ1

Запуск АЦП от EVA:

0 - нет запуска от EVA  
1 - разрешить запуск SEQ1 от EVA



Сброс SEQ1:

0 - нет действий  
1 - сброс кофайтера в исходное состояние

Прерывания SEQ1:

0 - запретить прерывания  
1 - разрешить прерывания

Режим прерываний кофайтера:

0 - прерывания после каждого преобразования  
1 - прерывания после каждого второго преобразования

### Младший байт

Запуск АЦП от внешн. сигнала:

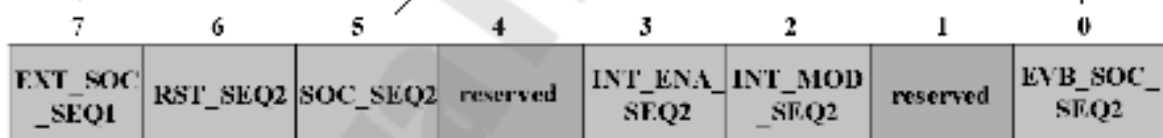
0 - нет действий  
1 - запуск АЦП от пина ADCSOC

Запуск преобразования:

0 - нет запуска  
1 - прогр. запуск преобр. SEQ2  
(только в двухканальном режиме)

Запуск АЦП от EVB:

0 - нет запуска  
1 - запуск SEQ2 от EVB



Сброс SEQ2:

0 - нет действий  
1 - сброс SEQ2 в исходное состояние

Разрешение прерываний SEQ2:

0 - прерывания запрещены  
1 - прерывания разрешены

Режим прерываний SEQ2:

0 - прерывания после каждого преобразования  
1 - прерывания после каждого второго преобразования

Рис. 5.5 Регистр 2 управления АЦП (ADCTRL2)





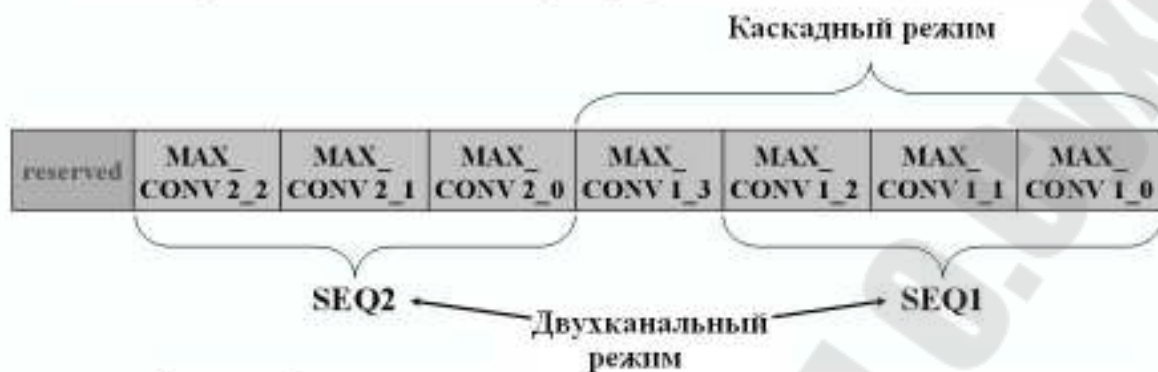
Рис. 5.6. Регистр 3 управления АЦП (ADCTRL3)

Делитель частоты ядра. Периферийная частота процессора, HSPCLK, делится на  $2^{*ADCCLKPS[3:0]}$ , за исключением, когда  $ADCCLKPS[3:0] = 0000$ , в этом случае HSPCLK не делится. Полученная частота дополнительно делится на  $ADCTRL1[7]+1$ .

ADCCLKPS [3:0]	Делитель частоты ядра	ADCLK (частота тактирования АЦП)
0000	0	$HSPCLK / (ADCTRL1[7] + 1)$
0001	1	$HSPCLK / [2 * (ADCTRL1[7] + 1)]$
0010	2	$HSPCLK / [4 * (ADCTRL1[7] + 1)]$
0011	3	$HSPCLK / [6 * (ADCTRL1[7] + 1)]$
0100	4	$HSPCLK / [8 * (ADCTRL1[7] + 1)]$
0101	5	$HSPCLK / [10 * (ADCTRL1[7] + 1)]$
0110	6	$HSPCLK / [12 * (ADCTRL1[7] + 1)]$
0111	7	$HSPCLK / [14 * (ADCTRL1[7] + 1)]$
1000	8	$HSPCLK / [16 * (ADCTRL1[7] + 1)]$
1001	9	$HSPCLK / [18 * (ADCTRL1[7] + 1)]$
1010	10	$HSPCLK / [20 * (ADCTRL1[7] + 1)]$
1011	11	$HSPCLK / [22 * (ADCTRL1[7] + 1)]$
1100	12	$HSPCLK / [24 * (ADCTRL1[7] + 1)]$
1101	13	$HSPCLK / [26 * (ADCTRL1[7] + 1)]$
1110	14	$HSPCLK / [28 * (ADCTRL1[7] + 1)]$
1111	15	$HSPCLK / [30 * (ADCTRL1[7] + 1)]$

## Регистр числа каналов для преобразования (ADCMAXCONV)

- ◆ Биты определяют число каналов, преобразуемых за один цикл (двоичн. код + 1)



- ◆ Автопреобразование начинается с начального канала и до последнего, если другое не определено в управляющих регистрах

	SEQ1	SEQ2	Cascaded
Начальный	CONV00	CONV08	CONV00
Последний	CONV07	CONV15	CONV15

Рис. 5.7. Формат регистра ADCMAXCONV

Биты 15-12	Биты 11-8	Биты 7-4	Биты 3-0	
CONV03	CONV02	CONV01	CONV00	ADCCHSELSEQ1
CONV07	CONV06	CONV05	CONV04	ADCCHSELSEQ2
CONV11	CONV10	CONV09	CONV08	ADCCHSELSEQ3
CONV15	CONV14	CONV13	CONV12	ADCCHSELSEQ4

Рис. 5.8. Регистр статуса автопоследовательности (ADCASEQASR)

Каждый набор из 4-х бит CONVnn, выбирает один из 16 аналоговых входов АЦП для последовательного автоматического преобразования.

Значение CONVnn	Выбираемый канал АЦП
0000	ADCINA0
0001	ADCINA1
0010	ADCINA2
0011	ADCINA3
0100	ADCINA4
0101	ADCINA5
0110	ADCINA6
0111	ADCINA7
1000	ADCINB0
1001	ADCINB1
1010	ADCINB2
1011	ADCINB3
1100	ADCINB4
1101	ADCINB5
1110	ADCINB6
1111	ADCINB7

### Буферный регистр результата АЦП (ADCRESULTn)

15	14	13	12	11	10	9	8
D11	D10	D9	D8	D7	D6	D5	D4
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
7	6	5	4	3	2	1	0
D3	D2	D1	D0	Reserved	Reserved	Reserved	Reserved
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Рис. 5.9. Формат регистра ADCRESULTn

В каскадном режиме начиная с регистра ADCRESULT8 по регистр ADCRESULT15 содержат результаты преобразования с девятого по шестнадцатый. Значения всех регистров ADCRESULTn выровнены по левому краю.

## Порядок выполнения лабораторной работы № 5

### Часть I

#### 1. Создание проекта.

1.1 В Code Composer Studio создаем новый проект Lab6.pjt. Копируем из папки c:\tidcs файл lab6.c в папку с созданным проектом. Добавляем lab6.c в проект.

1.2 Добавляем в проект следующие файлы:

*C:\tidcs\c28\dsp281x\v100\DSP281x\_headers\source\DSP281x\_GlobalVariableDefs.c*

*C:\tidcs\c28\dsp281x\v100\DSP281x\_common\cmd\F2812\_EzDSP\_RAM\_lnk.cmd*

*C:\tidcs\c28\dsp281x\v100\DSP281x\_headers\cmd\F2812\_Headers\_nonBIOS.cmd*

*C:\ti\c2000\cgtools\lib\rts2800\_ml.lib*

*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\_Adc.c*

*C:\tidcs\c28\dsp281x\v100\DSP281x\_common\source\DSP281x\_usDelay.asm*

1.3 Включаем в проект заголовочные файлы: 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.*

1.4 Задаем глубину стека: Project → Build Options → Linker → Stack Size: 0x400.

#### 2. Инициализация системы (подпрограмма “InitSystem”).

2.1 Разрешаем работу сторожевого таймера (см. лаб.раб. №2), устанавливаем коэффициентом деления 64 (регистр WDCR), сбрасываем сторожевой таймер (регистр SCSR).

2.2 Настраиваем ЦСП на частоту 150 МГц (регистр PLLCR), в предделитель высокоскоростного таймера заносим 2.

2.3 Разрешаем тактирование модуля АЦП и Менеджера Событий (регистр PCLKCR).

---

```

#####
//  Имя файла:      _lab6_.c
//
//  Описание: Преобразование напряжений каналов ADCINA0
//             и ADCINB0. Диапазон напряжений потенциометров,
//             соединенных с каналами АЦП, от 0 до 3 В. Запуск АЦП
//             от Менеджера Событий А. Результаты читаются по пре-
//             рыванию и отображаются на светодиодах (GPIO B7-B0)
#####

#include "DSP281x_Device.h"

void Gpio_select(void);
void InitSystem(void);
void show_ADC(int result);
interrupt void adc_isr(void); // Подпрограмма обработки прерывания
АЦП

int Voltage_A0;                // Глобальные переменные
int Voltage_B0;

void main(void)
{
    unsigned long i;
    InitSystem();              // Инициализация регистров ЦСП
    Gpio_select();            // Инициализация линий ввода/вывода
    InitPieCtrl();            // Инициализации блока PIE
    InitPieVectTable();       // Инициализация таблицы векторов блока
PIE
    InitAdc();                 // Инициализация модуля АЦП

    EALLOW;
    PieVectTable.ADCINT = &adc_isr;
    EDIS;

    // Разрешение прерывания от АЦП: группа 1, прерывание 6
    PieCtrlRegs.PIEIER1.bit.INTx6 = 1;

```

```

IER = 2;           // Разрешение прерываний линии 2 (INT2)
EINT;             // Разрешение общих прерываний INTM
ERTM;            // Разрешение общих прерываний в режи-
ме
// реального времени

// Настройка модуля АЦП

EALLOW;
SysCtrlRegs.PCLKCR.bit.ADCENCLK=1; // Разрешение такти-
рования АЦП
EDIS;

AdcRegs.ADCTRL1.bit.SEQ_CASC = 0; // Двухпоследова-
тельный режим
AdcRegs.ADCTRL1.bit.CONT_RUN = 0; // Режим
СТАРТ/СТОП
AdcRegs.ADCTRL1.bit.CPS = 0; // Пределитель = 1

AdcRegs.ADCMAXCONV.all = 0x0001; // Количество преоб-
разований 2

// Первое преобразование - канал ADCINA0
AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0x0;

// Второе преобразование - канал ADCINB0
AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 0x8;

// Разрешение начала преобразования от EVA
AdcRegs.ADCTRL2.bit.EVA_SOC_SEQ1 = 1;

// Разрешение запроса прерывания от SEQ1
AdcRegs.ADCTRL2.bit.INT_ENA_SEQ1 = 1;

// Коэффициент деления частоты = 4
AdcRegs.ADCTRL3.bit.ADCCLKPS = 2;

// Настройка модуля Менеджера Событий А
EALLOW;

```

```
    SysCtrlRegs.PCLKCR.bit.EVAENCLK=1;    // Разрешение такти-
рования EVA
    EDIS;
```

```
    EvaRegs.GPTCONA.bit.TCMPOE = 1;    // Управление логикой
сравнения
                                                    // T1PWM /
T2PWM
```

```
    // Полярность выходов сравнения GP таймера 1: активный уро-
вень - низкий
```

```
    EvaRegs.GPTCONA.bit.T1PIN = 1;
```

```
    // Разрешение выработки строба начала преобразования в EVA
```

```
    EvaRegs.GPTCONA.bit.T1TOADC = 2;
```

```
    EvaRegs.T1CON.bit.FREE = 0;    // Биты управления эму-
лятором:
```

```
    EvaRegs.T1CON.bit.SOFT = 0;    // мгновенная остановка
```

```
    EvaRegs.T1CON.bit.TMODE = 2;    // Непрерывный счет вверх
```

```
    EvaRegs.T1CON.bit.TPS = 7;    // Предделитель = 128
```

```
    EvaRegs.T1CON.bit.TENABLE = 1; // Разрешение работы GP тай-
мера 1
```

```
    EvaRegs.T1CON.bit.TCLKS10 = 0;    // Внутренний источник
тактирования
```

```
    EvaRegs.T1CON.bit.TCLD10 = 0; // Перегрузка GP таймера
// при достижении
нуля
```

```
    EvaRegs.T1CON.bit.TECMPR = 0;    // Запрещение режима
сравнения
```

```
    EvaRegs.T1PR = 58594;    // Регистр периода GP таймера
1
```

```
    while(1)
```

```
    {
        for(i=0;i<1500000;i++)
```

```
        {
            EALLOW;
```

```
            SysCtrlRegs.WDKEY = 0xAA;
```

```
            EDIS;
```

```

        show_ADC(Voltage_A0>>8);    // Отражение результата
на светодиодах
    }
    for(i=0;i<1500000;i++)
    {
        EALLOW;
        SysCtrlRegs.WDKEY = 0xAA;
        EDIS;
        show_ADC(Voltage_B0>>8);    // Отражение результата на
светодиодах
    }
}
}

```

```

#####
// Подпрограмма: show_ADC
//
// Описание: Отображение диапазона входного напряжения
//           на светодиодах
#####

```

```

void show_ADC(int result)
{
    switch(result)
    {
        case 0 : GpioDataRegs.GPBDAT.all=0x0000;break;
        case 1 : GpioDataRegs.GPBDAT.all=0x0001;break;
    }
    result>>=1;
    switch(result)
    {
        case 1 : GpioDataRegs.GPBDAT.all=0x0003;break;
        case 2 : GpioDataRegs.GPBDAT.all=0x0007;break;
        case 3 : GpioDataRegs.GPBDAT.all=0x000F;break;
        case 4 : GpioDataRegs.GPBDAT.all=0x001F;break;
        case 5 : GpioDataRegs.GPBDAT.all=0x003F;break;
        case 6 : GpioDataRegs.GPBDAT.all=0x007F;break;
        case 7 : GpioDataRegs.GPBDAT.all=0x00FF;break;
    }
}
}

```



```

#####
// Подпрограмма: adc_isr
//
// Описание: Обработка прерывания модуля АЦП
#####

interrupt void adc_isr(void)
{
    EALLOW;
    SysCtrlRegs.WDKEY = 0x55;
    EDIS;

    Voltage_A0 = AdcRegs.ADCRESULT0>>4;
    Voltage_B0 = AdcRegs.ADCRESULT1>>4;

    // Подготовка АЦП к следующему преобразованию
    AdcRegs.ADCTRL2.bit.RST_SEQ1 = 1; // Сброс SEQ1
    AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1; // Сброс бита
INT SEQ1
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1; //
Подтверждение // преры-
вания в блоке PIE
}

#####
// 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 = 0x0; // Настройка линий вво-
да/вывода на
    GpioMuxRegs.GPBMUX.all = 0x0; // работу в качестве портов
    GpioMuxRegs.GPDMUX.all = 0x0;
    GpioMuxRegs.GPFMUX.all = 0x0;
}

```

```

GpioMuxRegs.GPEMUX.all = 0x0;
GpioMuxRegs.GPGMUX.all = 0x0;

GpioMuxRegs.GPADIR.all = 0x0;           // Настройка портов A, D,
E, F, G на ввод
    GpioMuxRegs.GPBDIR.all = 0x00FF;    // Настройка линий 15-8
на ввод, а линий
    GpioMuxRegs.GPDDIR.all = 0x0;       // 7-0 на вывод
    GpioMuxRegs.GPEDIR.all = 0x0;
    GpioMuxRegs.GPFDIR.all = 0x0;
    GpioMuxRegs.GPGDIR.all = 0x0;

GpioMuxRegs.GPAQUAL.all = 0x0; //Запрещение входного ограни-
чителя
GpioMuxRegs.GPBQUAL.all = 0x0;
GpioMuxRegs.GPDQUAL.all = 0x0;
GpioMuxRegs.GPEQUAL.all = 0x0;
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= 0x00AF;    // Разрешение работы сто
                                // рожеевого таймера, предделитель = 64
                                // 0x00E8 - запрещение работы сторо-
                                // жеевого таймера, предделитель = 1

```

```

SysCtrlRegs.SCSR = 0; // Выработка сброса WDT

SysCtrlRegs.PLLCR.bit.DIV = 10; // Настройка блока
// умножения частоты
SysCtrlRegs.HISPCP.all = 0x1; // Задание значения преддели-
теля ВЫСО-
SysCtrlRegs.LOSPCP.all = 0x2; // коскоростного и низкоскорос-
ТНОГО
// таймеров

SysCtrlRegs.PCLKCR.bit.EVAENCLK=0; // Запрещение рабо-
ты периферий-
SysCtrlRegs.PCLKCR.bit.EVBENCLK=0; // ных устройств
SysCtrlRegs.PCLKCR.bit.SCIAENCLK=0;
SysCtrlRegs.PCLKCR.bit.SCIBENCLK=0;
SysCtrlRegs.PCLKCR.bit.MCBSPENCLK=0;
SysCtrlRegs.PCLKCR.bit.SPIENCLK=0;
SysCtrlRegs.PCLKCR.bit.ECANENCLK=0;
SysCtrlRegs.PCLKCR.bit.ADCENCLK=0;
EDIS;
}

```

Рис. 5.10. Текст программы

### 3. Инициализация портов (подпрограмма “Gpio\_select”).

3.1 Настраиваем все выходы на работу в качестве портов (регистр GPxMUX) (см. лаб.раб. №2).

3.2. Настраиваем порты A, D, E, F, G на ввод (регистр GPxDIR).

3.3 Настраиваем линии порта GPIOB15 – GPIOB8 на ввод, а GPIOB7 – GPIOB0 на вывод.

3.4 Сбрасываем биты регистров GPxQUAL в ноль.

### 4. Инициализация модуля Менеджера Событий.

4.1 Запрещаем выходы сравнения и выбираем полярность выходов сравнения принудительный низкий уровень (регистр GPTCONA) (см. лаб.раб. №3).

4.2 В регистре T1CON выбираем: непрерывный режим счета вверх, синхронизация от внутреннего источника, запрещаем режим сравнения, разрешаем работу GP таймера 1, задаем коэффициент деления 128.

4.3 Разрешаем запуск АЦП от Менеджера Событий А (регистр GPTCONA).

4.4 Рассчитываем значение периода (T1PR), зная, что  $f_{PWM} = 10$  Гц:

$$f_{PWM} = \frac{f_{CPU}}{T1PR \cdot TPS_{T1} \cdot HISCP}$$

## 5. Инициализация модуля АЦП.

5.1 Задаем: двух последовательный режим, запрещаем непрерывный режим, коэффициент деления 1 (регистр ADCTRL1).

5.2 Записываем количество преобразований 2 (регистр ADCMAXCONV).

5.3 Выбираем каналы ADCIN\_A0 и ADCIN\_B0 (регистр ADCCHSELSEQ1).

5.4 Разрешаем преобразования от Менеджера Событий А, разрешаем прерывания АЦП в конце каждой последовательности (регистр ADCTRL2)

5.3 Задаем делитель частоты высокоскоростного таймера равный 4.

## 6. Настройка прерываний.

6.1 Указываем адрес вектора прерываний (регистр ADCINT).

6.2 Разрешаем прерывание 6 группы 1 (регистр PIEIER1).

6.3 Вызываем подпрограммы инициализации: модуля прерывания периферийных устройств "InitPieCtrl()", вектора прерывания периферийных устройств "InitPieVectTable()", модуля АЦП "InitAdc()".

## 7. Получение результатов преобразования АЦП (подпрограмма "adc\_isr()").

7.1 Считываем результаты преобразования из регистров ADCRESULT0 и ADCRESULT1 и сохраняем их соответственно в переменные Voltage\_A0 и Voltage\_B0. Так как данные в регистрах ADCRESULTn выровнены к левому краю, необходимо произвести сдвиг данных на четыре разряда вправо.

7.2 Подготавливаем АЦП к следующему преобразованию: сбрасываем последовательность SEQ1 (регистр ADCTRL2), сбрасываем бит прерывания SEQ1 (регистр ADCST), подтверждаем прерывания (регистр PIEACK).

## **8. Вывод показаний АЦП на светодиодные индикаторы**

8.1. В основной подпрограмме main() написать программу вывода показаний данных из АЦП на линейку светодиодных индикаторов. Для этого необходимо использовать подпрограмму show\_ADC(result), которая отображает четыре младших бита параметра result на светодиодных индикаторах в виде светящейся линейки. Вывод показаний с каждого канала осуществлять поочередно через 1 секунду. В программе также необходимо сбрасывать сторожевой таймер процессора.

8.2. Компилируем проект: Project → Build.

8.3 Загружаем выходной файл: File → Load Program → Debug\lab6.out.

8.4. Запускаем программу на выполнение Debug → Run.

8.5. Контролируем правильное выполнение программы, изменяя потенциометрами напряжение, подаваемое на входы АЦП и наблюдая за линейкой светодиодных индикаторов.

## **9. Задание для самостоятельной работы**

Написать программу «бегущий огонь», аналогичную разработанной в лабораторной работе № 2. Скорость движения «бегущего огня» задавать потенциометром канала ADCIN\_A0.

### **Содержание отчета:**

Отчет должен содержать цель работы, краткие теоретические сведения по теме работы, рисунки, поясняющий структуру и режимы работы АЦП, тексты исследуемых программ, результаты их выполнения, выводы.

### **Контрольные вопросы:**

1. Встроенный АЦП DSP TMS320F2812: структура, параметры, режимы работы.
2. Каскадный режим работы АЦП: особенности, источники запуска преобразования.
3. Двухканальный режим работы АЦП: особенности, источники запуска преобразования.
4. Система тактирования АЦП.
5. Регистры АЦП.

## **Лабораторная работа № 6**

### **Исследование встроенного CAN-интерфейса DSP TMS320F2812**

**Цель работы:** изучить характеристики встроенного CAN-интерфейса DSP TMS320F2812.

#### **Теоретические сведения**

### **1. Особенности CAN-интерфейса DSP TMS320F2812**

Модуль eCAN имеет следующие особенности:

- 1) Протокол версии 2.0В, полностью совместимый с CAN;
- 2) Поддержка скоростей передачи данных до 1 Мбит/с;
- 3) 32 почтовых ящика, каждый со следующими свойствами:
  - конфигурация на прием или передачу;
  - стандартный или расширенный идентификатор;
  - фильтр с программируемой маской при приеме;
  - поддержка кадров удаленного запроса данных;
  - поддержка 0-8 байтов данных;
  - возможность установки 32-разрядной временной метки на принятом и переданном сообщениях;
  - защита от приема нового сообщения;
  - программирование приоритета передающего сообщения.
  - использование программируемой системы прерываний с двумя уровнями прерываний;
  - программируемое прерывание по окончании времени передачи или приема.
- 4) Режим пониженного энергопотребления;
- 5) Программируемый выход из режима пониженного энергопотребления при появлении активности на шине;
- 6) Автоматический ответ на удаленный запрос данных;
- 7) Автоматический повтор передачи при потере арбитража или в случае возникновения ошибки;
- 8) 32-разрядный счетчик временной метки, синхронизированный на определенное сообщение;
- 9) Режим самопроверки:
  - работа в режиме петли; прием своего собственного сообщения, что устраняет потребность в другом узле для определения бита признака.

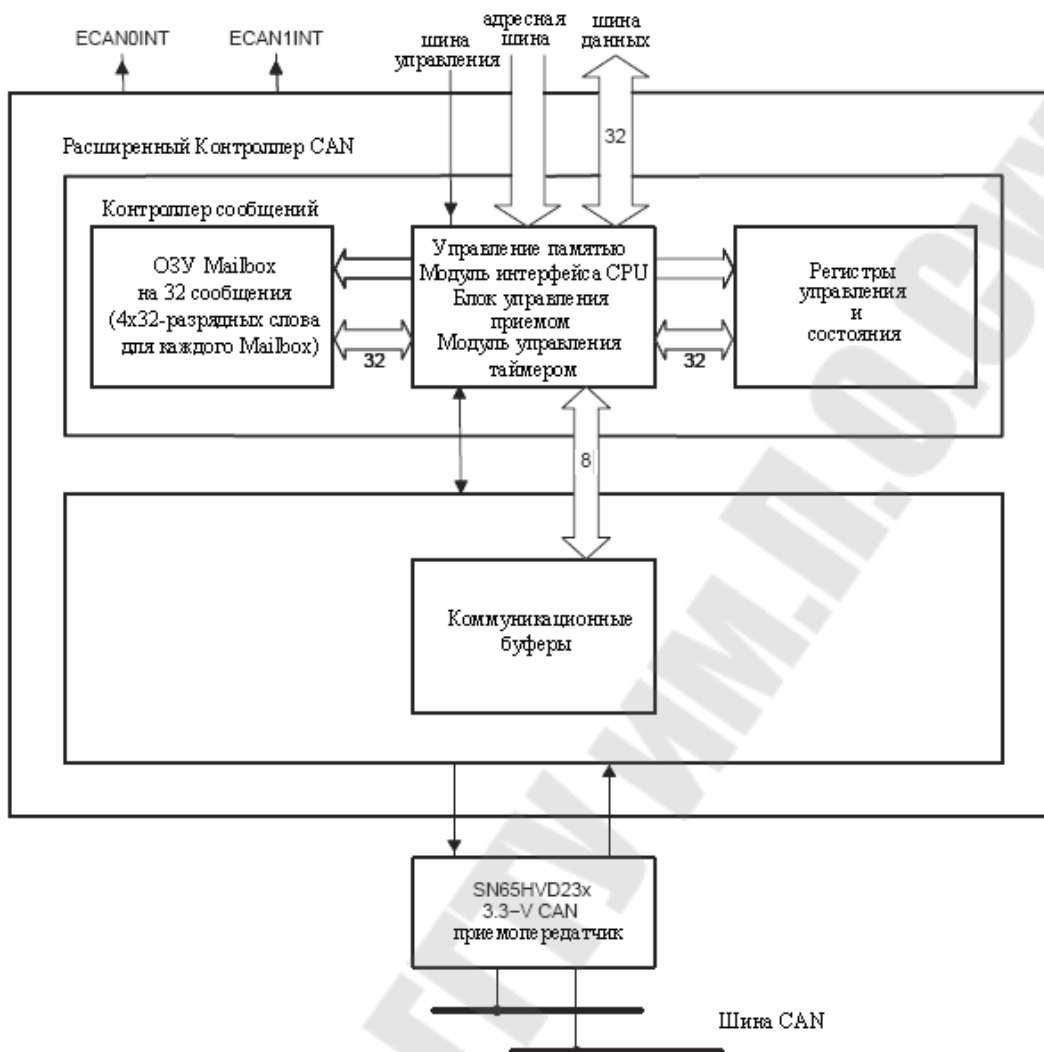


Рис. 6.1. Блок-схема CAN-интерфейса в ЦСП TMS320F2812

## 2. Совместимость eCAN с другими CAN-модулями

eCAN-модуль идентичен "Высокопроизводительному CAN-контроллеру (HECC)", используемому в микроконтроллерах серии TMS470 фирмы Texas Instruments, с некоторыми незначительными изменениями. Также модуль eCAN представлен более широко (увеличенное число Mailbox с индивидуальными приемными масками, временные метки и т.д.) в сравнении с CAN-модулем ЦСП семейства 240x. Поэтому программы, написанные для CAN-модуля 240x, не могут применяться к eCAN 320x. Однако eCAN обладает той же самой структурой размещения разрядов регистров и функциональными возможностями, как и CAN серии 240x (для регистров, существующих в

обоих устройствах), т.е. большинство регистров и битов выполняют идентичные функции на двух платформах. Это упрощает перемещение программ, написанных на языке C++.

### ***2.1. Модуль и сеть CAN***

В локальной контроллерной сети (CAN) используется последовательный протокол ведущей передачи, который эффективно поддерживает распределенное управление в реальном времени с сверхвысоким уровнем безопасности и скоростью передачи до 1 Мбит/с. CAN-шина идеальна для приложений, работающих в зашумленных средах, в автомобильных и других промышленных областях, которые требуют надежной коммуникации.

Расположенные по приоритетам сообщения, до 8 байтов данных в длину, посылаются на ведущую шину, использующую арбитражный протокол и механизм обнаружения ошибок для высокого уровня целостности данных.

### ***2.2. Протоколы CAN***

CAN поддерживает четыре типа различных типов кадров для передачи:

- кадры, передающие данные от передающего узла к принимающему;
- удаленные кадры, переданные узлом с целью запроса кадра данных с таким же идентификатором;
- кадры паузы, обеспечивающие дополнительную задержку между предшествующим и последующим кадром данных или удаленными кадрами;
- кадры об ошибках, передающиеся любым узлом на шине, обнаружившим ошибку.

Кроме того, в версии CAN 2.0В определены два вида формата сообщений, которые отличаются длиной полей идентификатора: стандартный кадр с 11-разрядным идентификатором и расширенный кадр с 29-разрядным идентификатором.

Сообщения CAN со стандартными кадрами данных содержат от 44 до 108 битов, а сообщения CAN с расширенными кадрами данных содержат от 64 до 128 битов. Кроме того, могут быть добавлены до 23 битов данных в стандартном кадре и до 28 данных в расширенном



кадре данных, в зависимости от кодировки потока данных. Полная максимальная длина сообщения – 131 бит со стандартным кадром и 156 бит с расширенным кадром.

Поля битов, находящиеся в стандартных/расширенных кадрах данных, расположенные как показано на рисунке 6.2, включают в себя следующее:

- бит начала кадра SOF;
- поле арбитража, содержащее идентификатор и тип посылаемого сообщения (11-разрядный идентификатор + бит RTR стандартного формата кадра или 29-разрядный идентификатор + бит SRR + бит IDE + бит RTR расширенного формата кадра);
- управляющее поле, содержащее размер данных (6 бит);
- данные (до 8 байтов);
- поле циклического контроля избыточности (CRC) – 16 бит;
- 2 бита подтверждения;
- 7 битов конца кадра.



Рис. 6.2. Структура кадра сообщения стандартного формата

В TMS320x28xx CAN-контроллеры предоставляют центральному процессору полные функциональные возможности, с помощью CAN-протоколов версии 2.0B. CAN-контроллер минимизирует загрузку CPU при передаче и увеличивает возможности CAN.

Архитектура CAN-модуля, показанная на рисунке 6.3, состоит из ядра протоколов (СРК) и контроллера сообщений.

СРК выполняет 2 функции. Первая – декодирование всех сообщений, появляющихся на шине согласно протоколам CAN, и передача этих сообщений в буфер приема. Вторая функция – передача сообщений на шину, согласно протоколу CAN.

Контроллер сообщений решает, должно ли принятое в СРК сообщение быть сохранено для использования CPU, или нет. При инициализации CPU передает контроллеру сообщений идентификаторы

всех сообщений, которые будут использоваться. Контроллер сообщений также отвечает за передачу следующего сообщения из СРК согласно приоритету сообщений.

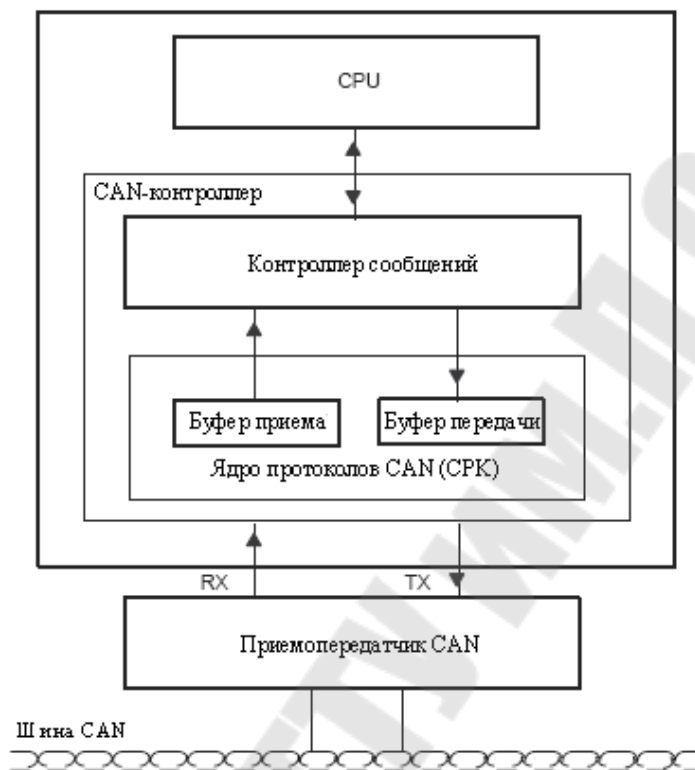


Рис. 6.3. Архитектура модуля eCAN

### 2.3. Контроллеры eCAN

eCAN-модуль состоит из:

- 1) ядра протоколов CAN;
- 2) контроллера сообщений, содержащего:

- модуль управления памятью, включая интерфейс CPU, блок управления приемом (фильтрация) и модуль управления таймером;
- ОЗУ Mailbox, позволяющая хранить до 32 сообщений;
- регистры управления и состояния.

После приема сообщения СРК определяет блок управления приемом в контроллере сообщений, который решает, сохранять принятое сообщение в одну из 32 ячеек ОЗУ Mailbox или нет. Блок управления анализирует идентификатор и сравнивает с масками Mailbox для определения номера Mailbox, в котором будет сохранено со-

общение. При совпадении маски и идентификатора принятое сообщение сохраняется в соответствующий Mailbox. Если блок управления приемом не обнаружил совпадения маски и идентификатора, то сообщение игнорируется.

Сообщение состоит из 11- или 29-разрядного идентификатора, поля управления и поля данных до 8 байт.

При необходимости передачи сообщения контроллер сообщений передает сообщение в буфер передачи СРК, который осуществит передачу в следующем неактивном состоянии шины. Когда необходимо передать несколько сообщений, контроллер сообщений сначала передает в буфер передачи СРК сообщение с самым высоким приоритетом, а после его отправки – следующее по убыванию приоритета сообщение. Если два Mailbox имеют одинаковые приоритеты, то сначала передается сообщение из Mailbox с более высоким номером.

Модуль управления таймером включает в себя счетчик временной метки и привязывает эту метку ко всем полученным и переданным сообщениям. Прерывание происходит, когда сообщение не принято или передано в течение некоторого ограниченного времени. Особенность ограничения времени на прием/передачу доступна только в модуле CAN.

Для инициализации переданных данных бит запроса передачи должен быть установлен в соответствующем регистре управления. Тогда процедура передачи и обработки сообщений выполняется без привлечения CPU. Если Mailbox ориентирован на прием, то CPU легко читает его регистры, используя команды разрешения чтения. Mailbox формирует прерывание CPU после каждого успешного приема/передачи сообщения.

### ***2.3.1. Режим стандартного контроллера CAN (SCC)***

Режим SCC – это упрощенный режим функциональных возможностей eCAN. В этом режиме доступны только 16 Mailbox (от 0 до 15). Особенность ограничения времени недоступна и число доступа к маскам для приема сообщений сокращено. Этот режим устанавливается по умолчанию. Переход из режима SCC в режим полного eCAN осуществляется с помощью бита SCB (CANMC.13).

### ***2.3.2. Карта памяти***

eCAN-модуль имеет два банка адресов, отражаемых в памяти TMS320x28xx. Первый банк используется для обращения к регистрам

управления, регистрам состояния, приемным маскам, временной метке и ограничению времени исследования сообщения. Доступ к регистрам управления и состояния ограничен 32-разрядной шиной. К приемным маскам, регистрам временной метки и регистрам ограничения времени можно обратиться 8-, 16- и 32-разрядными шинами. Каждый из этих двух банков памяти, которые показаны на рисунке 6.4, использует 512 байтов адресного пространства. Аналогично выглядит карта памяти eCAN-B, изображенная на рисунке 6.5.

Регистры управления и состояния eCAN-A

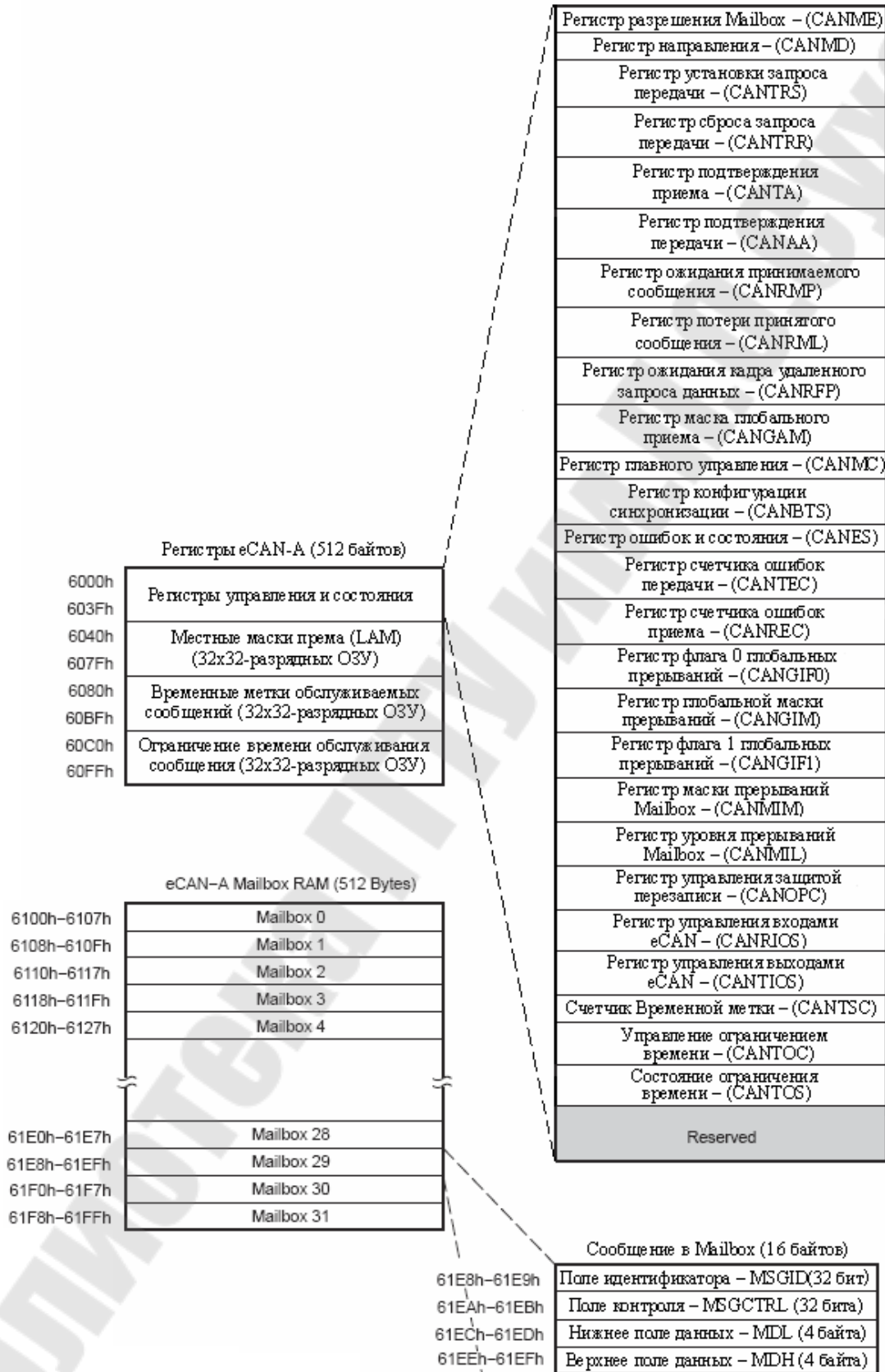


Рис. 6.4. Карта памяти eCAN-A

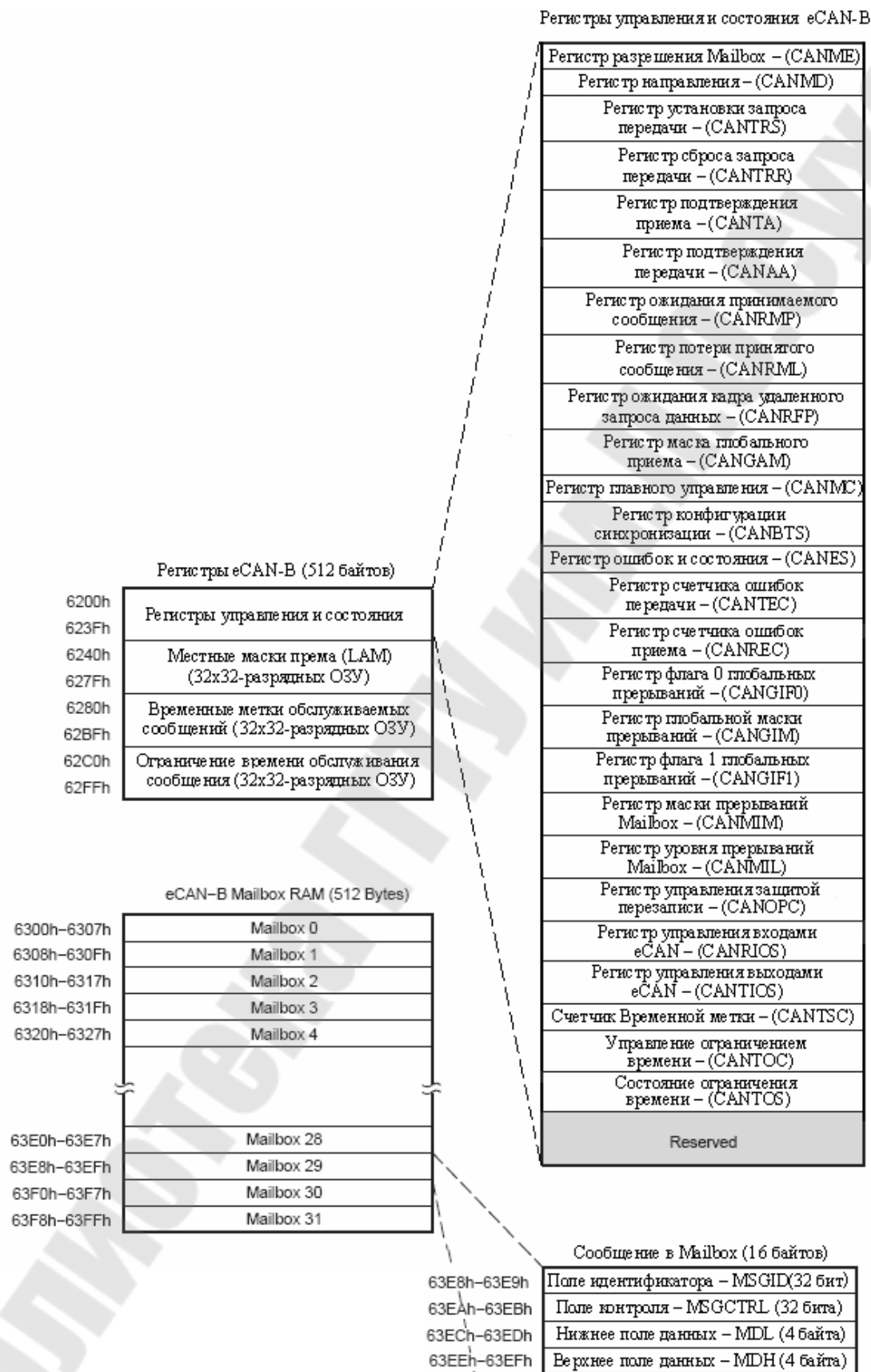


Рис. 6.5. Карта памяти eCAN-B

Хранение сообщения осуществляется оперативной памятью, к которой может обратиться CAN-контроллер или CPU. CPU при помощи CAN-контроллера может превратить различные Mailbox в ОЗУ или в дополнительную память. Дополнительная память может использоваться для хранения различных элементов, необходимых для фильтрации принятых сообщений и обработки прерываний.

Модуль Mailbox eCAN представляет собой 32 почтовых ящика, в каждом из которых содержится до 8 байт данных, до 29 битов идентификатора и несколько служебных битов сообщения. Каждый Mailbox может быть ориентирован как на прием, так и на передачу сообщения. В eCAN каждый Mailbox имеет свою индивидуальную маску.

### 2.3.3. Регистры управления и состояния eCAN

Регистры eCAN, перечисленные в таблице 6.1, используются CPU для конфигурации и управления контроллером CAN при работе с сообщениями.

Таблица 6.1. Регистры eCAN

Название регистра	Адреса eCAN-A	Адреса eCAN-B	Разрядность (x32)	Описание
CANME	0x6000	0x6200	1	Регистр разрешения Mailbox
CANMD	0x6002	0x6202	1	Регистр направления
CANTRS	0x6004	0x6204	1	Регистр установки запроса передачи
CANTRR	0x6006	0x6206	1	Регистр сброса запроса передачи
CANTA	0x6008	0x6208	1	Регистр подтверждения приема
CANAA	0x600A	0x620A	1	Регистр подтверждения передачи
CANRMP	0x600C	0x620C	1	Регистр ожидания принимаемого сообщения
CANRML	0x600E	0x620E	1	Регистр потери принятого сообщения
CANRFP	0x6010	0x6210	1	Регистр ожидания кадра удаленного запроса данных

Название регистра	Адреса eCAN-A	Адреса eCAN-B	Разрядность (x32)	Описание
CANGAM	0x6012	0x6212	1	Регистр маска глобального приема
CANMC	0x6014	0x6214	1	Регистр главного управления
CANBTS	0x6016	0x6216	1	Регистр конфигурации синхронизации
CANES	0x6018	0x6218	1	Регистр ошибок и состояния
CANTEC	0x601A	0x621A	1	Регистр счетчика ошибок передачи
CANREC	0x601C	0x621C	1	Регистр счетчика ошибок приема
CANGIF0	0x601E	0x621E	1	Регистр флага 0 глобальных прерываний
CANGIM	0x6020	0x6220	1	Регистр глобальной маски прерываний
CANGIF1	0x6022	0x6222	1	Регистр флага 1 глобальных прерываний
CANMIM	0x6024	0x6224	1	Регистр маски прерываний Mailbox
CANMIL	0x6026	0x6226	1	Регистр уровня прерываний Mailbox
CANOPC	0x6028	0x6228	1	Регистр управления защитой перезаписи
CANRIOS	0x602A	0x622A	1	Регистр управления входами eCAN
CANTIOS	0x602C	0x622C	1	Регистр управления выходами eCAN
CANTSC	0x602E	0x622E	1	Счетчик временной метки (reserved в режиме SCC)
CANTOC	0x6030	0x6230	1	Управление ограничением времени (reserved в режиме SCC)
CANTOS	0x6032	0x6232	1	Состояние ограничения времени (reserved в режиме SCC)



## 2.4. Работа с сообщениями

eCAN-модуль имеет 32 различных Mailbox, каждый из которых имеет свою индивидуальную приемную маску и может быть настроен на прием или передачу.

Сообщение Mailbox содержит:

- 29-разрядный идентификатор сообщения;
- регистр управления сообщением;
- до 8 байтов данных;
- 29-разрядная приемная маска;
- 32-разрядная временная метка;
- 32-разрядное значение ограничения времени.

Соответствующие биты регистров состояния и управления позволяют управлять сообщениями.

## 2.5. Mailbox

Mailbox – это область оперативной памяти, где сообщения CAN сохраняются после того, как они приняты, или прежде, чем они будут отправлены.

CPU может использовать области оперативной памяти Mailbox, не используемые для хранения сообщений, как обычную память.

Каждый Mailbox содержит:

- 1) Идентификатор сообщения:
  - 29-разрядов для расширенного идентификатора;
  - 11-разрядов для стандартного идентификатора;
- 2) Бит расширения идентификатора, IDE (MSGID,31);
- 3) Бит установки приемной маски, AME (MSGID,30);
- 4) Бит автоматического ответа, AAM (MSGID,29);
- 5) Уровень приоритетной передачи, TPL (MSGCTRL,12-8);
- 6) Бит удаленного запроса данных, RTR (MSGCTRL,4);
- 7) Код длины данных, DLC (MSGCTRL,3-0);
- 8) Поле данных до 8 байтов.

Каждый из Mailbox может работать в режиме одного из четырех типов сообщения (см. таблицу 6.2). Прием и передача сообщений необходимы для обмена данными между передающим узлом и многочисленными узлами-приемниками, тогда как сообщения запроса и подтверждения используются для поддержания взаимосвязи на шине.

Таблица 6.2. Конфигурация сообщений

Виды сообщений	Регистр направления Mailbox (CANMD)	Бит установки режима автоматического ответа (AAM)	Бит установки запроса данных (RTR)
Передача сообщения	0	0	0
Прием сообщения	1	0	0
Сообщение-запрос	1	0	1
Сообщение-ответ	0	1	0

Таблица 6.3 показывает расположение байтов Mailbox eCAN-A в ОЗУ.

Таблица 6.3. Расположение байтов Mailbox eCAN-A в ОЗУ

Mailbox	MSGID MIDL-MIDH	MSGCTRL MCF-Rsvd	MDL MDL_L- MDL_H	MDH MDH_L- MDH_H
0	6100-6101h	6102-6103h	6104-6105h	6106-6107h
1	6108-6109h	610A-610Bh	610C-610Dh	610E-610Fh
2	6110-6111h	6112-6113h	6114-6115h	6116-6117h
3	6118-6119h	611A-611Bh	611C-611Dh	611E-611Fh
4	6120-6121h	6122-6123h	6124-6125h	6126-6127h
5	6128-6129h	612A-612Bh	612C-612Dh	612E-612Fh
6	6130-6131h	6132-6133h	6134-6135h	6136-6137h
7	6138-6139h	613A-613Bh	613C-613Dh	613E-613Fh
8	6140-6141h	6142-6143h	6144-6145h	6146-6147h
9	6148-6149h	614A-614Bh	614C-614Dh	614E-614Fh
10	6150-6151h	6152-6153h	6154-6155h	6156-6157h
11	6158-6159h	615A-615Bh	615C-615Dh	615E-615Fh
12	6160-6161h	6162-6163h	6164-6165h	6166-6167h
13	6168-6169h	616A-616Bh	616C-616Dh	616E-616Fh
14	6170-6171h	6172-6173h	6174-6175h	6176-6177h
15	6178-6179h	617A-617Bh	617C-617Dh	617E-617Fh

Mailbox	MSGID MIDL-MIDH	MSGCTRL MCF-Rsvd	MDL MDL_L- MDL_H	MDH MDH_L- MDH_H
16	6180-6181h	6182-6183h	6184-6185h	6186-6187h
17	6188-6189h	618A-618Bh	618C-618Dh	618E-618Fh
18	6190-6191h	6192-6193h	6194-6195h	6196-6197h
19	6198-6199h	619A-619Bh	619C-619Dh	619E-619Fh
20	61AO-61A1h	61A2-61A3h	61A4-61A5h	61A6-61A7h
21	61A8-61A9h	61AA-61ABh	61AC-61ADh	61AE-61AF
22	61BO-61B1h	61B2-61B3h	61B4-61B5h	61B6-61B7h
23	61B8-61B9h	61BA-61BBh	61BC-61BDh	61BE-61BFh
24	61CO-61C1h	61C2-61C3h	61C4-61C5h	61C6-61C7h
25	61C8-61C9h	61CA-61CBh	61CC-61CDh	61CE-61CFh
26	61DO-61D1h	61D2-61D3h	61D4-61D5h	61D6-61D7h
27	61D8-61D9h	61DA-61DBh	61DC-61DDh	61DE-61DF
28	61EO-61E1h	61E2-61E3h	61E4-61E5h	61E6-61E7h
29	61E8-61E9h	61EA-61EBh	61EC-61EDh	61EE-61EFh
30	61FO-61F1h	61F2-61F3h	61F4-61F5h	61F6-61F7h
31	61F8-61F9h	61FA-61FBh	61FC-61FDh	61FE-61FFh

### 2.5.1. Передача в Mailbox

CPU хранит данные, которые передаются в настроенный на передачу сообщения Mailbox. После присоединения к данным идентификатора посылается запрос в ОЗУ, был ли бит TRS установлен, разрешается ли работа Mailbox, при этом устанавливается бит передачи ME.n (n – номер Mailbox).

Если более одного Mailbox готовы к передаче и более чем один соответствующий бит TRS установлен, сообщения посылаются по очереди в порядке уменьшения, начиная с Mailbox с самым высоким приоритетом.

В режиме SCC приоритет передающего Mailbox зависит от номера Mailbox. Самый высокий номер Mailbox (=15) обладает самым высоким приоритетом передачи.

В eCAN приоритет передачи Mailbox зависит от состояния регистра управления сообщением (MSGCTRL) в поле TPL. Сначала передает Mailbox с самым высоким значением поля TPL. Только когда два Mailbox имеют одинаковые значения TPL-полей, сначала передает Mailbox с высшим номером.

Если во время передачи будет допущена ошибка из-за потери арбитража или другая ошибка, то передача сообщения будет произведена вновь. Перед повтором передачи CAN-модуль проверяет, нет ли запросов от других узлов, и затем разрешает передачу Mailbox с самым высоким приоритетом.

### ***2.5.2. Прием в Mailbox***

Идентификатор каждого принятого сообщения сравнивается с идентификатором, содержащимся на каждом Mailbox, используя маску. Когда обнаружено совпадение, полученный идентификатор, биты управления и байты данных записываются на определенный ОЗУ адрес. В то же время формируется бит ожидания приема сообщения RMP[n] (RMP.31-0), устанавливается и происходит прерывание, если оно разрешено. Если сообщение идентификатора не произошло, то сообщение не сохраняется [5].

При приеме сообщения контроллер сообщения начинает искать соответствующий идентификатор в Mailbox, начиная с самого высокого номера. Mailbox 15 в SCC-режиме имеет самый высокий номер и, следовательно, высший приоритет. Mailbox 31 имеет самый высокий номер при выключенном режиме SCC и имеет наивысший приоритет в eCAN-модуле.

Регистр RMP[n] (RMP.31-0) должен быть сброшен CPU после чтения данных. Если принимается второе сообщение для одного Mailbox и бит ожидания приема сообщения уже установлен, то устанавливается бит RML[n] (RML.31-0) и сообщение задерживается. Хранимое сообщение записывается поверх с новыми данными, если сброшен бит защиты наложения записи OPC[n] (OPC.31-0), иначе проверяется следующий Mailbox.

Если Mailbox настроен на прием и бит RTR установлен для этого, Mailbox может передать кадр удаленного запроса данных. Как только запрос передан, бит TRS Mailbox eCAN сбрасывается.

### ***2.5.3. Операции CAN-модуля в нормальном режиме***

Если CAN-модуль работает в нормальном режиме (не в режиме самопроверки), то должен быть как минимум еще один CAN-модуль в сети, настроенный на такую же скорость передачи информации в битах. Второй CAN-модуль может быть настроен на прием сообщений

от передающего узла, но должен быть настроен на ту же самую скорость передачи информации в битах. Это необходимо, т.к. передающий CAN-модуль видит, что есть как минимум один узел в CAN-сети, чтобы подтвердить правильную передачу переданного сообщения в сеть. Согласно CAN-протоколу любой узел, который получил сообщение, присылает подтверждение (если программно не отключено), независимо от того, был ли он настроен на получение именно этого сообщения, или нет.

Необходимость второго узла отсутствует в режиме самопроверки (STM). В этом режиме передающий узел формирует свой собственный сигнал. Единственное требование – узел должен быть настроен на любую рабочую скорость передачи информации. Т.о. регистры синхронизации не должны содержать значения, не актуальные для CAN-протокола.

## **Порядок выполнения лабораторной работы**

### **1. Настройка отправки сообщения**

1) Создание нового проекта.

1.1) В Code Composer Studio создать новый проект Lab9.pjt. Открыть файл Lab9.c и сохранить его в E:\C281x\Labs\Lab9\lab9.c.

1.2) Добавить в проект файлы:

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

C:\tidcs\c28\dsp281x\v100\DSP281x\_headers\cmd\F2812\_Headers\_nonBIOS.cmd

C:\tidcs\c28\dsp281x\v100\DSP281x\_common\cmd\F2812\_EzDSP\_RAM\_lnk.cmd

C:\ti\c2000\cgtoolslib\rts2800\_ml.lib

2) Настройка параметров проекта, компоновка проекта и загрузка выходного файла.

2.1) Включить в проект заголовочные файлы: Project → Build Options, в закладке Compiler выбрать Preprocessor и в поле Include Search Path (-i) ввести:

C:\tidcs\C28\dsp281x\v100\DSP281x\_headers\include; ..\include

2.2) Задать глубину стека: Project → Build Options → Linker → Stack Size: 0x400.

2.3) Закрывать Build Options, кликнув ОК.

3) Преобразование файла Lab9.c.

3.1) Настроить в основной программе цикл while( ) таким образом, чтобы каждая отправка осуществлялась через 1 секунду. Сделать прерывание «CPU core timer 0» через каждые 50 мс для увеличения значения «CpuTimer0.InterruptCount».

3.2) Создать новую структуру в основной программе «ECanaShadow»:

```
struct ECAN_REGS ECanaShadow;
```

3.3) В подпрограмме «Gpio\_select( )» настроить периферийные функции CANTxA и CANRxА для работы с CAN-модулем:

```
GpioMuxRegs.GPFMUX.bit.CANTXA_GPIOF6 = 1;
```

```
GpioMuxRegs.GPFMUX.bit.CANRXA_GPIOF7 = 1;
```

3.4) В конце кода программы создать подпрограмму InitCan( ), с помощью которой осуществить следующие шаги:

3.4.1) В регистрах «ECanaRegs.CANRIOС» и «ECanaRegs.CANRIOС» установить биты «TXFUNC» и «RXFUNC».

3.4.2) Включить режим HECC модуля CAN (регистр «ECanaRegs.CANМС»).

3.4.3) Для получения доступа к регистрам времени установить бит «CCR» регистра «ECanaRegs.CANМС».

3.4.4) Для передачи запроса инициализации CAN установить флаг «CCE» регистра «ECanaRegs.CANES».

3.4.5) Установить параметры «BRP», «TSEG1» и «TSEG2» регистра «ECanaRegs.CANBTC» таким образом, чтобы скорость передачи была 100 кбит/с.

3.4.6) После установки параметров регистра «ECanaRegs.CANBTC» запретить к нему доступ - очистить бит CCR регистра «ECanaRegs.CANМС».

3.4.7) Отключить все mailboxes, кроме того, который отправляет сообщение, установкой в 0 полей регистра «ECanaRegs.CANME».

3.5) Для подготовки mailbox#5 к отправке сообщения сделать следующее:

3.5.1) Установить идентификатор сообщения 0x10000000 (бит “IDE” регистра “ECanaMboxes.MBOX5.MSGID”). Также установить бит IDE регистра “ECanaMboxes.MBOX1.MSGID” в 1.

3.5.2) Для установления mailbox#5 передающим, сбросить бит “MD5” регистра “ECanaRegs.CANMD”. Так как мы не имеем доступа к регистру “ECanaRegs.CANMD”, то сделать это следующим образом:  
ECanaShadow.CANMD.all = ECanaRegs.CANMD.all;  
ECanaShadow.CANMD.bit.MD5 = 0;  
ECanaRegs.CANMD.all = ECanaShadow.CANMD.all;

Активировать mailbox#5:

ECanaShadow.CANME.all = ECanaRegs.CANME.all;  
ECanaShadow.CANME.bit.ME5 = 1;  
ECanaRegs.CANME.all = ECanaShadow.CANME.all;

3.8.3) Установить длину сообщения равной 1 (бит DLC регистра “ECanaMboxes.MBOX5.MSGCTRL”).

4) Присоединение байта данных и отправка.

Теперь надо организовать периодическую загрузку байта данных в mailbox и отправку с помощью цикла while(1). Для этого:

4.1) Из входных портов GPIO-Port B (с 15 по 8 бит) загрузить байт данных в регистр “ECanaMboxes.MBOX5.MDL.byte.BYTE0”.

4.2) Послать запрос передачи mailbox#5. Для этого в регистре “ECanaShadow.CANTRS” установить бит TRS5=1, а все остальные 0. Затем загрузить все это в регистр “ECanaRegs.CANTRS”.

4.3) Как только придет запрос на отправку сообщения, флаг “ECanaRegs.CANTA.bit.TA5” будет установлен в 1.

4.4) Установить бит “ECanaRegs.CANTA.bit.TA5” в исходное состояние:

ECanaShadow.CANTA.all = 0;  
ECanaShadow.CANTA.bit.TA5 = 1;  
ECanaRegs.CANTA.all = ECanaShadow.CANTA.all;

5) Тестирование программы.

5.1) Сбросить ЦСП: Debug → Reset CPU, Debug → Restart.

5.2) Перейти к главной подпрограмме: Debug → Go main.

5.3) Запустить программу: Debug → Run.

## 2. Настройка приема сообщения

1) Создание нового проекта.

1.1) В Code Composer Studio создать новый проект Lab10.pjt. Открыть файл Lab10.c и сохранить его в E:\C281x\Labs\Lab10\lab10.c.

1.2) Добавить в проект файлы:

C:\tidcs\c28\dsp281x\v100\DSP281x\_headers\source\DSP281x\_GlobalVariableDefs.c

C:\tidcs\c28\dsp281x\v100\DSP281x\_headers\source\DSP281x\_GlobalVariableDefs.c

C:\tidcs\c28\dsp281x\v100\DSP281x\_headers\cmd\F2812\_Headers\_nonBIOS.cmd

C:\tidcs\c28\dsp281x\v100\DSP281x\_common\cmd\F2812\_EzDSP\_RAM\_ink.cmd

C:\ti\c2000\cgtools\lib\rts2800\_ml.lib

2) Настройка параметров проекта, компоновка проекта и загрузка выходного файла.

2.1) Включить в проект заголовочные файлы: Project → Build Options, в закладке Compiler выбрать Preprocessor и в поле Include Search Path (-i) ввести:

C:\tidcs\C28\dsp281x\v100\DSP281x\_headers\include; ..\include

2.2) Задать глубину стека: Project → Build Options → Linker → Stack Size: 0x400.

2.3) Закрывать Build Options, кликнув ОК.

3) Преобразование файла Lab9.c.

3.1) Открыть файл Lab10.c

Удалить те части программы, которые не будут использоваться в данной лабораторной работе: подпрограмму “cpu\_timer0\_isr()” и массив LED[8].

3.2) Также от начала основной программы до цикла while(1) удалить все вызовы подпрограмм InitSystem( ) и GpioSelect( ).

Задать новые инструкции для сторожевого таймера:

```
EALLOW;
```

```
SysCtrlRegs.WDKEY = 0x55;
```

```
SysCtrlRegs.WDKEY = 0xAA;
```

```
EDIS;
```

3.3) Создать новую структуру в основной программе “ECanaShadow”:

```
struct ECAN_REGS ECanaShadow;
```

3.4) Перейти к подпрограмме «Gpio\_select( )». Настроить периферийные функции CANTxA и CANRxA:

```
GpioMuxRegs.GPFMUX.bit.CANTXA_GPIOF6 = 1;
```

```
GpioMuxRegs.GPFMUX.bit.CANRXA_GPIOF7 = 1;
```



3.5) В конце кода программы создать подпрограмму InitCan( ), с помощью которой осуществить следующие шаги:

3.5.1) В регистрах “ECanaRegs.CANTIOС” и “ECanaRegs.CANRIOС” установить биты “TXFUNC” и “RXFUNC”.

3.5.2) Включить режим HECC модуля CAN (регистр “ECanaRegs.CANMC”).

3.5.3) Для получения доступа к регистрам времени установить бит “CCR” регистра “ECanaRegs.CANMC”.

3.5.4) Для передачи запроса инициализации CAN установить флаг “CCE” регистра “ECanaRegs.CANES”.

3.5.5) Установить параметры “BRP”, “TSEG1” и “TSEG2” регистра “ECanaRegs.CANBTC”.

3.5.6) После установления параметров регистра “ECanaRegs.CANBTC” запретить к нему доступ - очистить бит CCR регистра “ECanaRegs.CANMC”.

3.5.7) Отключить все mailboxes кроме того, который отправляет сообщение, установкой в 0 полей регистра “ECanaRegs.CANME”.

3.6) Для подготовки mailbox#1 к приему сообщения сделать следующее:

3.6.1) Установить идентификатор сообщения 0x10000000 (бит “IDE” регистра “ECanaMboxes.MBOX1.MSGID”).

3.6.2) Для установления mailbox#1 принимающим установить бит “MD1” регистра “ECanaRegs.CANMD”. Используя буферные регистры «ECanaShadow.CANMD.all» осуществить запись в регистр “ECanaRegs.CANMD”:

```
ECanaShadow.CANMD.all = ECanaRegs.CANMD.all;
```

```
ECanaShadow.CANMD.bit.MD1 = 1;
```

```
ECanaRegs.CANMD.all = ECanaShadow.CANMD.all;
```

Активировать mailbox#5:

```
ECanaShadow.CANME.all = ECanaRegs.CANME.all;
```

```
ECanaShadow.CANME.bit.ME1 = 1;
```

```
ECanaRegs.CANME.all = ECanaShadow.CANME.all;
```

4) Организация цикла опроса mailbox#1

4.1) Организовать цикл опроса и приема сообщения. Когда сообщение придет, бит “RMP1” регистра “ECanaRegs.CANRMP” станет равным 1.

С помощью цикла do-while организовать ожидание RMP1=1.

Примечание 1. Рекомендуется скопировать регистры “ECanaRegs.CANRMP” в буферные “ECanaShadow.CANRMP”.

Примечание 2. В цикле ожидания не забывать активировать сторожевой таймер.

4.2) Как только бит RMP1 станет равным 1 следует отправить нулевой байт на порты GPIO-B7...B0:

```
GpioDataRegs.GPBDAT.all = ECanaMboxes.MBOX1.MDL.byte.BYTE0;
```

4.3) Сбросить бит RMP1, записав в него «1»:

```
ECanaShadow.CANRMP.bit.RMP1 = 1;
```

```
ECanaRegs.CANRMP.all = ECanaShadow.CANRMP.all;
```

5) Тестирование программы.

5.1) Сбросить ЦСП: Debug → Reset CPU, Debug → Restart.

5.2) Перейти к главной подпрограмме: Debug → Go main.

5.3) Запустить программу: Debug → Run. [6]

### 3. Инициализация нового узла в сети CAN

Для правильного приема/передачи сообщения необходимо сначала запустить программу приема узлом сообщения л/р 10 (узел отправляет на шину сигнал о своем присутствии), и только затем запустить программу передачи сообщения л/р 9. При подключении к сети нового принимающего узла необходимо до подключения перевести в режим останова передающий узел. После подключения нового принимающего узла запустить программу приема (л/р 10), и только потом произвести перезапуск передающего узла (л/р 9).

#### Листинг передачи сообщения

```
// FILE: Lab9.c
// TITLE:DSP28 CAN Передача
//CPU Timer0 ISR каждые 50 мс
//Watchdog активен, работает в ISR и в главном цикле
//CAN-сообщение: 1 Байт (состояние GPIO B15-B8) каждую 1 секунду
// Скорость 100KBPS
// Идентификатор : 0x1000 0000
// Mailbox #5
#include "DSP281x_Device.h" // Включение заголовочного файла

void Gpio_select(void);
void SpeedUpRevA(void);
```

```

void InitSystem(void);
void InitCan();
interrupt void cpu_timer0_isr(void);
// Программа обработки прерывания Timer 0

void main(void)
{
struct ECAN_REGS ECanaShadow;

InitSystem();           // Инициализация регистров ЦСП

Gpio_select();         // Инициализация линий ввода/вывода
InitPieCtrl();// Подключение PIE-модуля (DSP281x_PieCtrl.c)

InitPieVectTable(); // Подключение PIE-вектора (DSP281x_PieVect.c )

// отобразим PIE – вход для таймера прерываний Timer 0
EALLOW; // Необходимо для записи EALLOW в защищенные реги-
стры
    PieVectTable.TINT0 = &cpu_timer0_isr;
    EDIS; // Необходимо для отмены записи EALLOW в защищенные
регистры

InitCpuTimers();

// Настроить CPU-Timer 0 на прерывание каждые 50 мс:
// 150MHz частота CPU, период прерывания 50000 мкс
    ConfigCpuTimer(&CpuTimer0, 150, 50000);

    // Допуск TINT0 в PIE: Группа 1 прерывание 7
    PieCtrlRegs.PIEIER1.bit.INTx7 = 1;

// Допуск CPU к INT1 связанному с CPU-Timer 0:
    IER = 1;

// Разрешение глобальным прерываниям и более приоритетным собы-
тиям отладки в реальном времени:
    EINT; // Разрешение глобального прерывания INTM

```

```
ERTM; // Разрешение глобального прерывания в реальном времени
DBGM
```

```
InitCan();
```

```
/* Запись в поле MSGID */
```

```
ECanaMboxes.MBOX5.MSGID.all = 0x????????;
```

```
ECanaMboxes.MBOX5.MSGID.bit.IDE = ?; // Установка идентификатора сообщения
```

```
/* Установка Mailbox 5 передающим */
```

```
ECanaShadow.CANMD.all = ECanaRegs.CANMD.all;
```

```
ECanaShadow.CANMD.bit.MD5 = ?;
```

```
ECanaRegs.CANMD.all = ECanaShadow.CANMD.all;
```

```
/* Активация Mailbox 5 */
```

```
ECanaShadow.CANME.all = ECanaRegs.CANME.all;
```

```
ECanaShadow.CANME.bit.ME5 = ?;
```

```
ECanaRegs.CANME.all = ECanaShadow.CANME.all;
```

```
/* Установка длины сообщения равной 1 */
```

```
ECanaMboxes.MBOX5.MSGCTRL.bit.DLC = ?;
```

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

```
while(1)
```

```
{
```

```
while(CpuTimer0.InterruptCount < 20)
```

```
{ // ожидание Timer 0
```

```
EAALLOW;
```

```
SysCtrlRegs.WDKEY = 0xAA; // и передача watchdog #2
```

```
EDIS;
```

```
}
```

```
CpuTimer0.InterruptCount = 0; // обнулить таймер прерываний
```

```
ECanaMboxes.MBOX5.MDL.byte.BYTE0 = (GpioDataRegs.GPBDAT.all >> 8);
```

```
ECanaShadow.CANTRS.all = ?;
```

```
ECanaShadow.CANTRS.bit.TRS5 = ?; // Посыл запроса передачи Mailbox 5
```

```
ECanaRegs.CANTRS.all = ECanaShadow.CANTRS.all;
```

```
while(ECanaRegs.CANTA.bit.TA5 == 0 ) {} // Когда бит TA5 установлен..
```

```
    ECanaShadow.CANTA.all = ?;  
    ECanaShadow.CANTA.bit.TA5 = ?;           // Установка бита TA5 в  
исходное состояние  
    ECanaRegs.CANTA.all = ECanaShadow.CANTA.all;  
    }  
}
```

```
void Gpio_select(void)
```

```
{  
    EALLOW;  
    GpioMuxRegs.GPAMUX.all = 0x?; // Настройка линий ввода/  
вывода на работу в качестве портов  
    GpioMuxRegs.GPBMUX.all = 0x?;  
    GpioMuxRegs.GPDMUX.all = 0x?;  
    GpioMuxRegs.GPFMUX.all = 0x?;  
    GpioMuxRegs.GPFMUX.bit.CANTXA_GPIOF6 = ?; // Настройка  
периферийных функций CANTXA и  
    GpioMuxRegs.GPFMUX.bit.CANRXA_GPIOF7 = ?; // CANRXA  
для работы с CAN-модулем  
    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, а линий 7-0 на вывод  
    GpioMuxRegs.GPDDIR.all = 0x?;  
    GpioMuxRegs.GPEDIR.all = 0x?;  
    GpioMuxRegs.GPFDIR.all = 0x?;  
    GpioMuxRegs.GPGDIR.all = 0x?;  
  
    GpioMuxRegs.GPAQUAL.all = 0x?; // Запрещение входного огра-  
нителя  
    GpioMuxRegs.GPBQUAL.all = 0x?;  
    GpioMuxRegs.GPDQUAL.all = 0x?;  
    GpioMuxRegs.GPEQUAL.all = 0x?;  
    EDIS;
```

```

}

void InitSystem(void)
{
    EALLOW;
    SysCtrlRegs.WDCR= 0x00AF;      // Работа сторожевого таймера
    // 0x00E8  запрещение работы сторожевого таймера, преддели-
    тель = 1
    // 0x00AF  Разрешение работы сторожевого таймера, предделитель
    = 64
    SysCtrlRegs.SCSR = 0;          // Выработка сброса WDT
    SysCtrlRegs.PLLCR.bit.DIV = 10; // Настройка блока умножения
    частоты
    SysCtrlRegs.HISPCP.all = 0x1; // Задание значения предделителя вы-
    сокоскоростного таймера
    SysCtrlRegs.LOSPCP.all = 0x2; // Задание значения предделителя
    низкоскоростного таймера
    // Запрещение работы периферийных устройств кроме eCAN
    SysCtrlRegs.PCLKCR.bit.EVAENCLK=0;
    SysCtrlRegs.PCLKCR.bit.EVBENCLK=0;
    SysCtrlRegs.PCLKCR.bit.SCIAENCLK=0;
    SysCtrlRegs.PCLKCR.bit.SCIBENCLK=0;
    SysCtrlRegs.PCLKCR.bit.MCBSPENCLK=0;
    SysCtrlRegs.PCLKCR.bit.SPIENCLK=0;
    SysCtrlRegs.PCLKCR.bit.ECANENCLK=1;
    SysCtrlRegs.PCLKCR.bit.ADCENCLK=0;
    EDIS;
}
interrupt void cpu_timer0_isr(void)
{
    CpuTimer0.InterruptCount++; // Сообщение watchdog-таймеру о
    каждом прерывании Timer 0
    EALLOW;
    SysCtrlRegs.WDKEY = 0x55;      // Сообщение таймеру watchdog
    #1
    EDIS;
    // Это прерывание произошло от прерываний группы 1
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}

```

```

void InitCan(void)
{
    asm(" EALLOW");
    /* Настроить линии TX и RX на передачу используя регистры eCAN*/
    ECanaRegs.CANTIOC.bit.TXFUNC = ?;
    ECanaRegs.CANRIOC.bit.RXFUNC = ?;
    /* Настроить eCAN на режим HECC - (доступ к чтению mailboxes 16 -
    31) */
    // В режиме HECC разрешена особенность временного ограничения
    ECanaRegs.CANMC.bit.SCB = ?;
    /* Настроить параметры тактовой синхронизации */
    ECanaRegs.CANMC.bit.CCR = ? ;      // Установка CCR для получе-
    ния доступа к регистрам времени
    while(ECanaRegs.CANES.bit.CCE != 1 ) {} // Когда CCE (флаг для
    передачи запроса инициализации CAN)установлен..
    ECanaRegs.CANBTC.bit.BRPREG = ??; // Установка параметров
    BRP,TSEG1 и TSEG2
    ECanaRegs.CANBTC.bit.TSEG2REG = ?; // таким образом, чтобы
    скорость передачи
    ECanaRegs.CANBTC.bit.TSEG1REG = ??; // была 100кбит/с

    ECanaRegs.CANMC.bit.CCR = ? ;      // Сброс CCR регистра
    CANMC для запрета доступа к регистру CANBTC
    while(ECanaRegs.CANES.bit.CCE == !0 ) {} // Когда CCE сбро-
    шен..
}
/* Отключить все Mailboxes */
//=====
// Конец программы.
//=====

```

### Листинг приема сообщения

```

// FILE:   Lab10.c
// TITLE:  DSP28 CAN Прием , Mailbox 1
//Идентификатор 0x10 000 000 ; Скорость 100 KBPS
//Байт данных 1 из CAN – кадр будет скопирован из 8 LED's (GPIOB7
- B0)
// Watchdog активен, работает в ISR и в главном цикле

```

```

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

void Gpio_select(void);
void InitSystem(void);
void InitCan(void);

void main(void)

{
struct ECAN_REGS ECanaShadow;

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

/* Запись в поле MSGID - MBX номер записан как его MSGID */
ECanaMboxes.MBOX1.MSGID.all = 0x????????; // Установка идентификатора сообщения
ECanaMboxes.MBOX1.MSGID.bit.IDE = ?;

/* Установка Mailbox 1 принимающим */
ECanaShadow.CANMD.all = ECanaRegs.CANMD.all;
ECanaShadow.CANMD.bit.MD1 = ?;
ECanaRegs.CANMD.all = ECanaShadow.CANMD.all;

/* Активация Mailbox 1 */
ECanaShadow.CANME.all = ECanaRegs.CANME.all;
ECanaShadow.CANME.bit.ME1 = ?;
ECanaRegs.CANME.all = ECanaShadow.CANME.all;
while(1)
{
do
{
ECanaShadow.CANRMP.all = ECanaRegs.CANRMP.all;
EALLOW;
SysCtrlRegs.WDKEY = 0x55; // Сообщение таймеру watchdog #1
SysCtrlRegs.WDKEY = 0xAA; // Сообщение таймеру watchdog #2

```



```

EDIS;
}
while(ECanaShadow.CANRMP.bit.RMP1 != 1 );    // Когда RMP1 ус-
тановлен..

```

```

    GpioDataRegs.GPBDAT.all = ECanaM-
boxes.MBOX1.MDL.byte.BYTE0;
    ECanaShadow.CANRMP.bit.RMP1 = ?;
    ECanaRegs.CANRMP.all = ECanaShadow.CANRMP.all;
    // Clear RMP1 bit and start again
}
}

```

```

void Gpio_select(void)
{
EALLOW;
GpioMuxRegs.GPAMUX.all = 0x?;    // Настройка линий вво-
да/вывода на работу в качестве портов
    GpioMuxRegs.GPBMUX.all = 0x?;
    GpioMuxRegs.GPDMUX.all = 0x?;
    GpioMuxRegs.GPFMUX.all = 0x?;
    GpioMuxRegs.GPEMUX.all = 0x?;
    GpioMuxRegs.GPGMUX.all = 0x?;
    GpioMuxRegs.GPFMUX.bit.CANTXA_GPIOF6 = ?; // Настройка
периферийных функций CANTxA и CANRxА
    GpioMuxRegs.GPFMUX.bit.CANRXA_GPIOF7 = ?;

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

    GpioMuxRegs.GPAQUAL.all = 0x?; // Запрещение входного огра-
нителя

```

```

    GpioMuxRegs.GPBQUAL.all = 0x?;
    GpioMuxRegs.GPDQUAL.all = 0x?;
    GpioMuxRegs.GPEQUAL.all = 0x?;
    EDIS;
}

void InitSystem(void)
{
    EALLOW;
    SysCtrlRegs.WDCR= 0x00AF;          // Работа сторожевого таймера
    // 0x00E8 запрещение работы сторожевого таймера, преддели-
    тель = 1
    // 0x00AF Разрешение работы сторожевого таймера, предделитель
    = 64

    SysCtrlRegs.SCSR = 0;              // Выработка сброса WDT

    SysCtrlRegs.PLLCR.bit.DIV = 10;    // Настройка блока умножения
    частоты
    SysCtrlRegs.HISPCP.all = 0x1; // Задание значения предделителя вы-
    сокоскоростного таймера
    SysCtrlRegs.LOSPCP.all = 0x2; // Задание значения предделителя
    низкоскоростного таймера

    // Запрещение работы периферийных устройств кроме eCAN
    SysCtrlRegs.PCLKCR.bit.EVAENCLK=0;
    SysCtrlRegs.PCLKCR.bit.EVBENCLK=0;
    SysCtrlRegs.PCLKCR.bit.SCIAENCLK=0;
    SysCtrlRegs.PCLKCR.bit.SCIBENCLK=0;
    SysCtrlRegs.PCLKCR.bit.MCBSPENCLK=0;
    SysCtrlRegs.PCLKCR.bit.SPIENCLK=0;
    SysCtrlRegs.PCLKCR.bit.ECANENCLK=1;
    SysCtrlRegs.PCLKCR.bit.ADCENCLK=0;
    EDIS;
}

void InitCan(void)
{
    asm(" EALLOW");
    /* Настроить линии TX и RX на передачу используя регистры eCAN*/

```

```

ECanaRegs.CANTIOC.bit.TXFUNC = ?;
ECanaRegs.CANRIOC.bit.RXFUNC = ?;

/* Настроить eCAN на режим HECC - (доступ к чтению mailboxes 16 -
31) */
// В режиме HECC разрешена особенность временного ограничения
ECanaRegs.CANMC.bit.SCB = ?;

/* Настроить параметры тактовой синхронизации */
ECanaRegs.CANMC.bit.CCR = ? ;    // Установка CCR для получе-
ния доступа к регистрам времени
    while(ECanaRegs.CANES.bit.CCE != 1 ) {} // Когда CCE (флаг для
передачи запроса инициализации CAN) установлен..
    ECanaRegs.CANBTC.bit.BRPREG = ??;
    ECanaRegs.CANBTC.bit.TSEG2REG = ?;
    ECanaRegs.CANBTC.bit.TSEG1REG = ??;

    ECanaRegs.CANMC.bit.CCR = ? ;    // Сброс бита CCR регист-
ра CANMC для запрета доступа к регистру CANBTC
    while(ECanaRegs.CANES.bit.CCE == !0 ) {} // Когда CCE сбро-
шен..

/* Отключить все Mailboxes */

    ECanaRegs.CANME.all = ?; // Отключить все Mailboxes кроме того
который отправляет сообщение
asm(" EDIS");
}
//=====
// Конец программы.
//=====

```

### Содержание отчета:

Отчет должен содержать цель работы, краткие теоретические сведения по теме работы, рисунки, тексты исследуемых программ пе-

передачи и приема данных через CAN-интерфейс, результаты их выполнения, выводы.

**Контрольные вопросы:**

1. Особенности CAN-интерфейса DSP TMS320F2812.
2. Модуль и сеть CAN.
3. Протоколы CAN.
4. Контроллеры eCAN.
5. Карта памяти eCAN.
6. Регистры управления и состояния eCAN.
7. Использование Mailbox для приема и передачи сообщений.

## Содержание

	стр
Лабораторная работа № 1. Аппаратные и программные средства отладки eZDSP F2812 и Code Composer Studio .....	3
Лабораторная работа № 2. Изучение карты памяти, структуры цифровых портов ввода/ вывода и системы тактирования ЦСП TMS320F2812.....	11
Лабораторная работа № 3. Исследование системы прерываний и таймеров ядра ЦСП семейства C28x .....	28
Лабораторная работа № 4. Исследование модуля Менеджера Событий.....	48
Лабораторная работа № 5. Исследование модуля АЦП.....	76
Лабораторная работа № 6. Исследование встроенного CAN-интерфейса DSP TMS320F2812.....	95

**Крышнев Юрий Викторович  
Храмов Александр Сергеевич  
Гарбуз Вячеслав Николаевич  
Елисеева Ольга Александровна**

## **АППАРАТУРА ЦИФРОВОЙ ОБРАБОТКИ СИГНАЛОВ**

**Лабораторный практикум  
для студентов специальности 1-36 04 02  
«Промышленная электроника», специализации  
1-36 04 02 01 «Микроэлектронные и микропроцессорные  
управляющие и информационные устройства»  
дневной и заочной форм обучения**

Подписано к размещению в электронную библиотеку  
ГГТУ им. П. О. Сухого в качестве электронного  
учебно-методического документа 15.11.12.

Рег. № 41Е.

<http://www.gstu.by>