

Министерство образования Республики Беларусь

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

Кафедра «Промышленная электроника»

ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА ОБРАБОТКИ СИГНАЛОВ

ПРАКТИКУМ

**по выполнению лабораторных работ
по одноименной дисциплине для студентов
специальности 1-53 01 07 «Информационные
технологии и управление в технических системах»
дневной формы обучения**

Электронный аналог печатного издания

Гомель 2021

УДК 004.383.3:004.31(075.8)
ББК 32.859я73
И72

*Рекомендовано к изданию научно-методическим советом
факультета автоматизированных и информационных систем
ГГТУ им. П. О. Сухого
(протокол № 4 от 28.12.2020 г.)*

Составители: *Ю. В. Крышнёв, В. А. Хананов*

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

Инструментальные средства обработки сигналов : практикум по выполнению лаборатор. работ по одноим. дисциплине для студентов специальности 1-53 01 07 «Информационные технологии и управление в технических системах» днев. формы обучения / сост.: Ю. В. Крышнёв, В. А. Хананов. – Гомель : ГГТУ им. П. О. Сухого, 2021. – 117 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: <https://elib.gstu.by>. – Загл. с титул. экрана.

И72

ISBN 978-985-535-468-1.

Содержит семь лабораторных работ с порядком их выполнения, основными теоретическими сведениями, заданиями для самостоятельной работы и контрольными вопросами.

Для студентов специальности 1-53 01 07 «Информационные технологии и управление в технических системах» дневной формы обучения.

УДК 004.383.3:004.31(075.8)
ББК 32.859я73

ISBN 978-985-535-468-1

© Крышнёв Ю. В., Хананов В. А.,
составление, 2021
© Учреждение образования «Гомельский
государственный технический университет
имени П. О. Сухого», 2021

ВВЕДЕНИЕ

На сегодняшний день представлено большое количество литературы, в которой рассмотрены современные инструментальные средства обработки сигналов (ИСОС) и изложены известные базовые алгоритмы цифровой обработки сигналов (ЦОС), однако реализация даже простейших алгоритмов на базе микропроцессоров и элементов с параллельной логикой может вызвать трудности у разработчиков начального уровня квалификации. Это объясняется тем, что реализация алгоритма ЦОС тесно связана с устройством обработки сигналов, т. е. для реализации этого алгоритма необходимо знание архитектуры устройства обработки сигналов (микропроцессора, программируемой логической интегральной схемы и т. п.).

Наиболее распространенными инструментальными средствами цифровой обработки аналоговых сигналов в реальном масштабе времени являются отладочные платы (Evaluation board) на основе цифровых сигнальных процессоров, программируемых интегральных схем ПЛИС (CPLD, Complex programmable logic device и FPGA, Field-programmable gate array), ARM-процессоров (Advanced RISC Machine). Изучение приемов работы с такими ИСОС является важным этапом подготовки инженеров по информационным технологиям и управлению.

Лабораторная работа № 1

Изучение отладочного модуля для цифрового сигнального процессора семейства C28x

Цель работы

Изучить отладочное оборудование для цифрового сигнального процессора семейства C28x фирмы Texas Instruments – отладочную плату eZDSP320F2812 с платой расширения Zwickau Adapterboard. Освоить интегрированную среду разработки Code Composer Studio, научиться создавать и отлаживать простейшие программы для цифрового сигнального процессора TMS320F2812.

Основные 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 отладочная плата (Evaluation board) eZDSP320F2812 с платой расширения. На плате эмулятора размещен сигнальный процессор TMS320F2812 и схема, обеспечивающая его работу – стабилизатор напряжения (+3,3 В для периферии и +1,9 В 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.

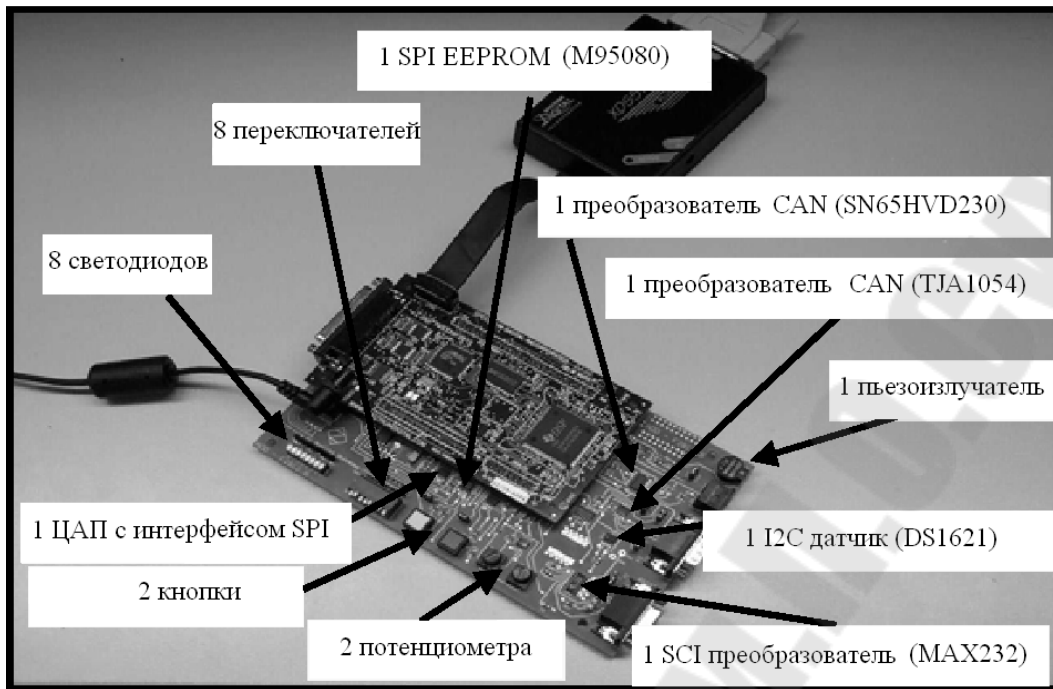


Рис. 1.1. Плата эмулятора eZDSP и плата расширения Zwickau Adapterboard

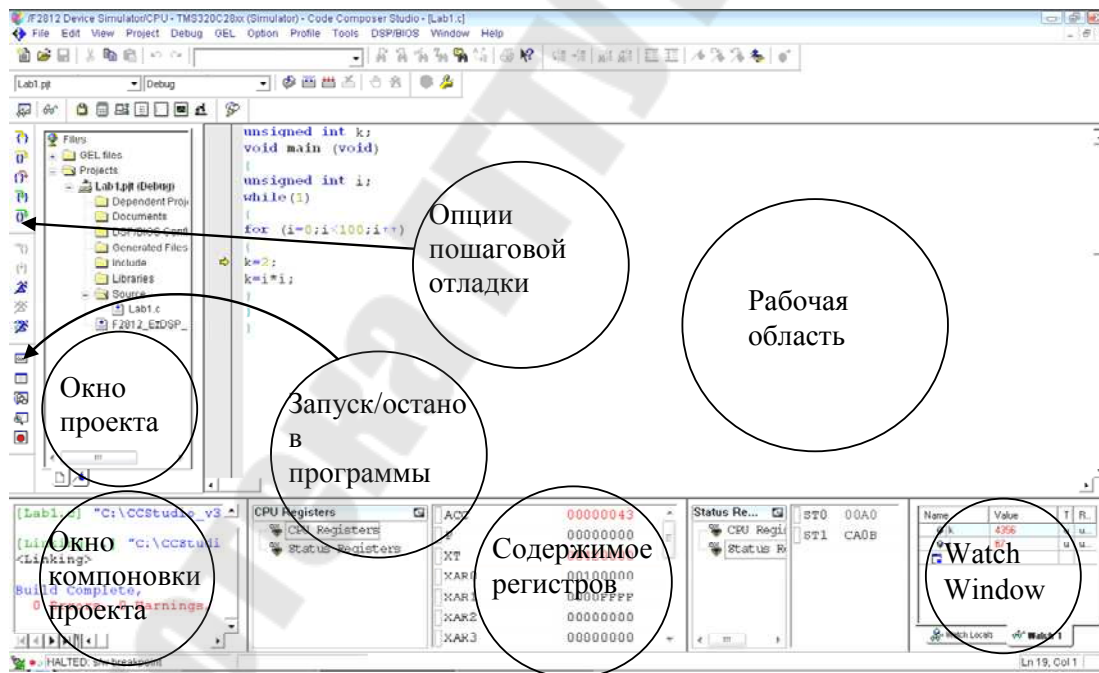


Рис. 1.2. Вид рабочего окна Code Composer Studio

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

Порядок выполнения 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 нужно добавить в проект файл управления линкером и библиотеки:

```
C:\tides\c28\dsp281x\v100\DSP281x_common\cmd\F2812_EzDSP_
RAM_Ink.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 ее 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.

Примечания:



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 следует 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 выполняется через вкладку меню Debug → Run (горячая клавиша F5) или с помощью иконки . При этом в строке статуса внизу рабочей области индицируется зеленая 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 цифрового сигнального процессора (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 ее 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 с.

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, следует дважды щелкнуть по его содержимому и ввести новое значение.

Задание для самостоятельной работы

В ходе выполнения задания для самостоятельной работы требуется:

1) установить Breakpoint на строки 7, 9 и 10 измененной тестовой программы, показанной на рис. 1.4. Перезапустить программу в автоматическом режиме (Run) после каждой остановки в выполнении программы на точках останова. Проследить изменение переменных *i* и *k* в окне Watch. Объяснить полученные результаты;

2) не снимая Breakpoint, запустить программу в режиме Animate. Следить за выполнением программы и содержимым окна Watch (переменные *i* и *k*). Объяснить полученные результаты;

3) открыв окна CPU Register и Status Register, проследить изменение содержимого регистров при выполнении программы в режиме Animate. Объяснить полученные результаты;

4) не останавливая выполнение программы, открыть окно асемблированного кода программы, полученного в результате компоновки (Window → Disassembly). Объяснить полученные результаты. Вернуться в окно исходного кода программы, написанной на языке Си (Window → Lab1.c). Остановить выполнение программы. Снять 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. Укажите, 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

Исследование систем управления и ввода/вывода цифрового сигнального процессора семейства C28x

Цель работы

Ознакомиться с системой управления и синхронизации (Control System) ЦСП семейства C28x, изучить аппаратную организацию и приемы работы с цифровыми портами ввода/вывода.

Основные теоретические сведения

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

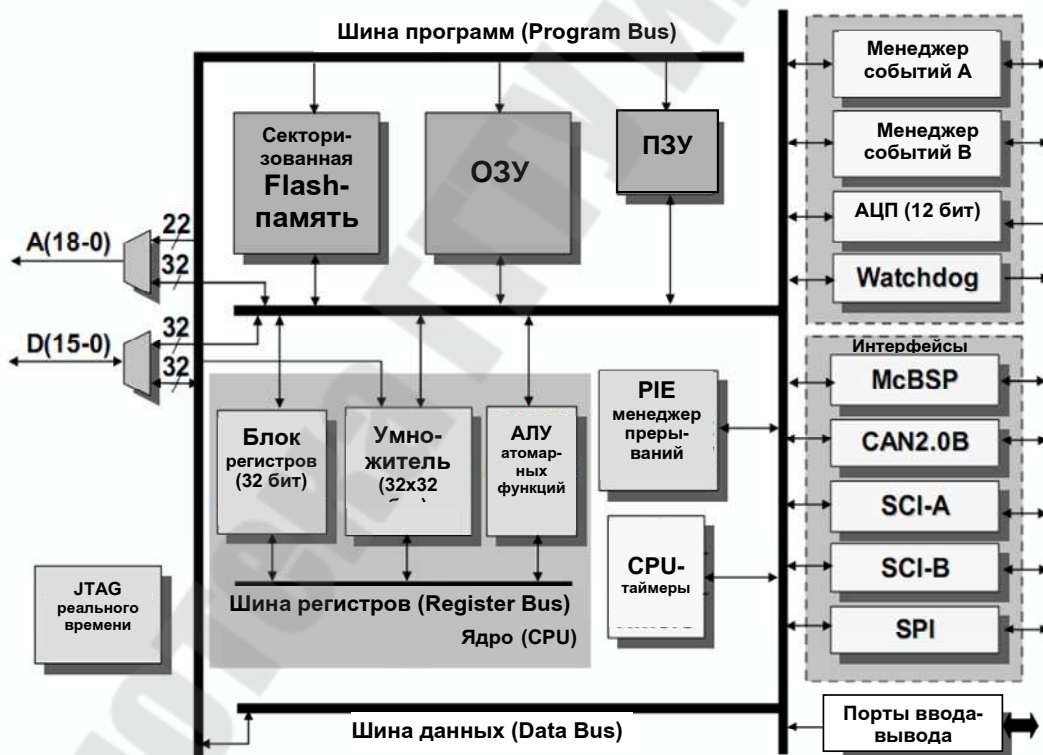


Рис. 2.1. Структурная схема ЦСП семейства C28x

Карта адресного пространства ЦСП C28x показана на рис. 2.2. В качестве памяти данных используется исключительно оперативно-запоминающее устройство (ОЗУ) быстрого доступа (Single Access RAM – SARAM, доступ возможен в течение каждого машинного цик-

ла) 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 объем встроенной флэш-памяти составляет 128 Кслов (4 сектора – по 8К и 6 секторов – по 16К).

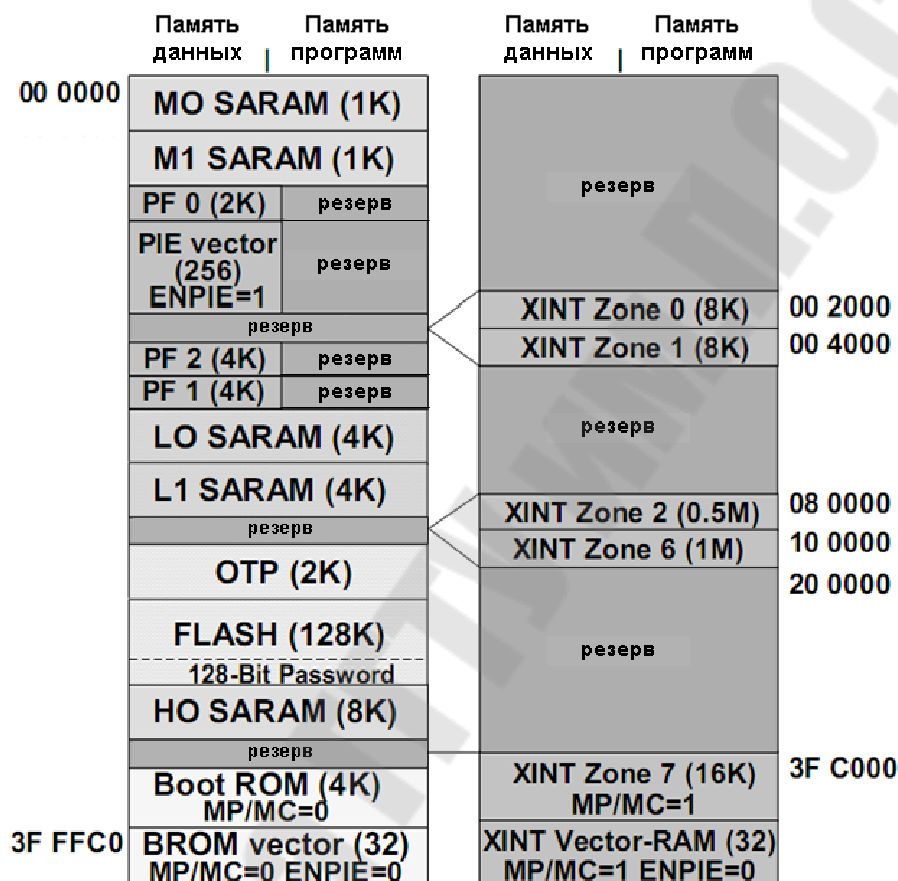


Рис. 2.2. Карта адресного пространства ЦСП семейства C28x

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 (рис. 2.2):

- 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.

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 (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).

GPIO A

GPIOA0 / PWM1
 GPIOA1 / PWM2
 GPIOA2 / PWM3
 GPIOA3 / PWM4
 GPIOA4 / PWM5
 GPIOA5 / PWM6
 GPIOA6 / T1PWM_T1CMP
 GPIOA7 / T2PWM_T2CMP
 GPIOA8 / CAP1_QEP1
 GPIOA9 / CAP2_QEP2
 GPIOA10 / CAP3_QEP1
 GPIOA11 / TDIRA
 GPIOA12 / TCLKINA
 GPIOA13 / C1TRIP
 GPIOA14 / C2TRIP
 GPIOA15 / C3TRIP

GPIO F

GPIOF0 / SPISIMOA
 GPIOF1 / SPISOMIA
 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

GPIO B

GPIOB0 / PWM7
 GPIOB1 / PWM8
 GPIOB2 / PWM9
 GPIOB3 / PWM10
 GPIOB4 / PWM11
 GPIOB5 / PWM12
 GPIOB6 / T3PWM_T3CMP
 GPIOB7 / T4PWM_T4CMP
 GPIOB8 / CAP4_QEP3
 GPIOB9 / CAP5_QEP4
 GPIOB10 / CAP6_QEP2
 GPIOB11 / TDIRB
 GPIOB12 / TCLKINB
 GPIOB13 / C4TRIP
 GPIOB14 / C5TRIP
 GPIOB15 / C6TRIP

GPIO G

GPIOG4 / SCITXDB
 GPIOG5 / SCIRXDB

GPIO D

GPIOD0 / T1CTRIP_PDPINTA
 GPIOD1 / T2CTRIP7_EVASOC
 GPIOD5 / T3CTRIP_PDPINTB
 GPIOD6 / T4CTRIP7_EVBSOC

GPIO E

GPIOE0 / XINT1_XBIO
 GPIOE1 / XINT2_ADCSOC
 GPIOE2 / XNMI_XINT13

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

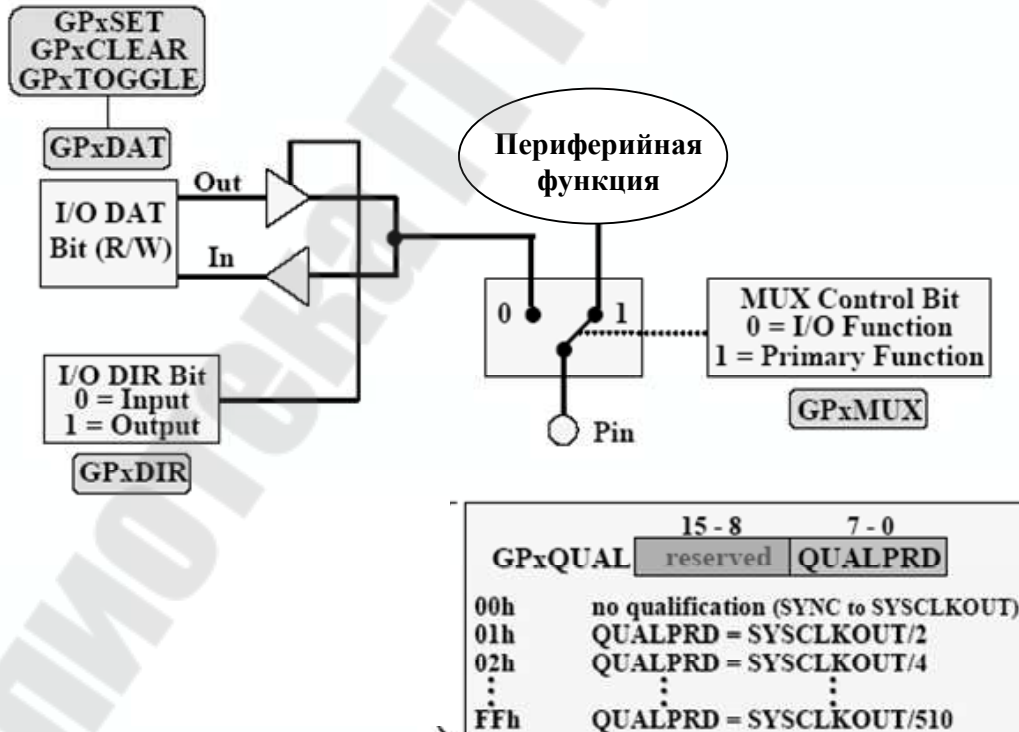


Рис. 2.4. Функциональная схема линии GPIO

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 настроить 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, рис. 2.5). Коэффициент деления задается программно в регистре PLLCR.

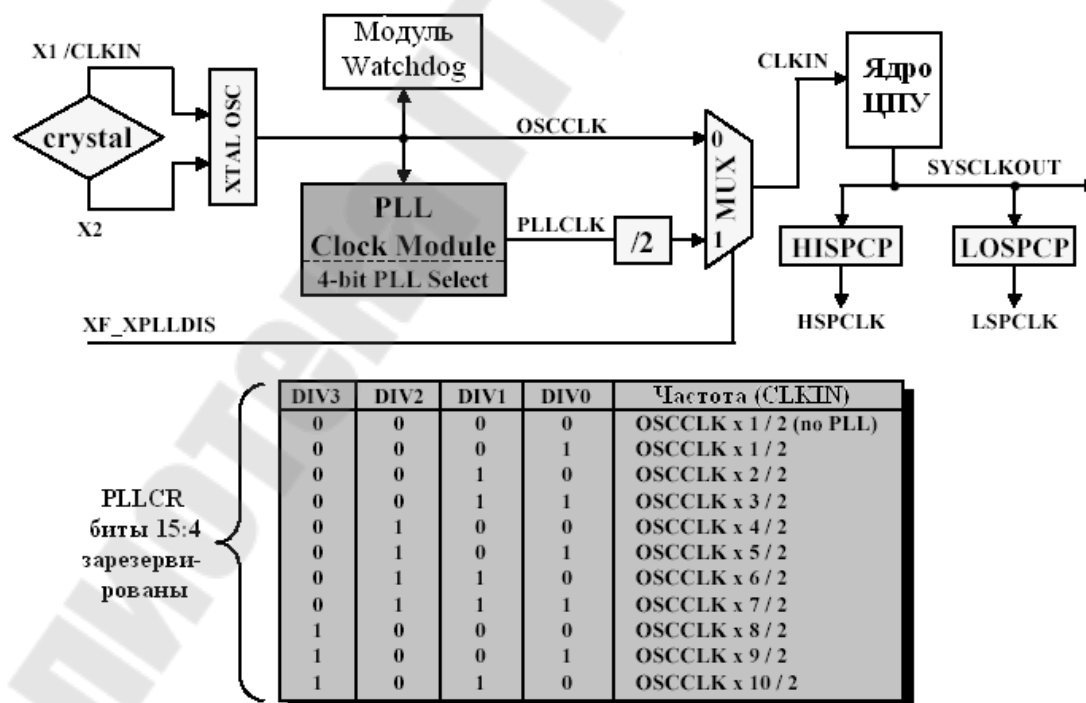


Рис. 2.5. Тактовый модуль и регистр PLLCR

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 LOSPCP и HISPCP.

15 - 3	2	1	0
reserved	HSPCLK2	HSPCLK1	HSPCLK0

15 - 3	2	1	0
reserved	LSPCLK2	LSPCLK1	LSPCLK0

H/LSPCLK2	H/LSPCLK1	H/LSPCLK0	Тактовая частота периферии
0	0	0	SYSCCLKOUT / 1
0	0	1	SYSCCLKOUT / 2 <small>исходная для HISPCP</small>
0	1	0	SYSCCLKOUT / 4 <small>исходная для LOSPCP</small>
0	1	1	SYSCCLKOUT / 6
1	0	0	SYSCCLKOUT / 8
1	0	1	SYSCCLKOUT / 10
1	1	0	SYSCCLKOUT / 12
1	1	1	SYSCCLKOUT / 14

Рис. 2.6. Формат регистров HISPCP и LOSPCP

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

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.

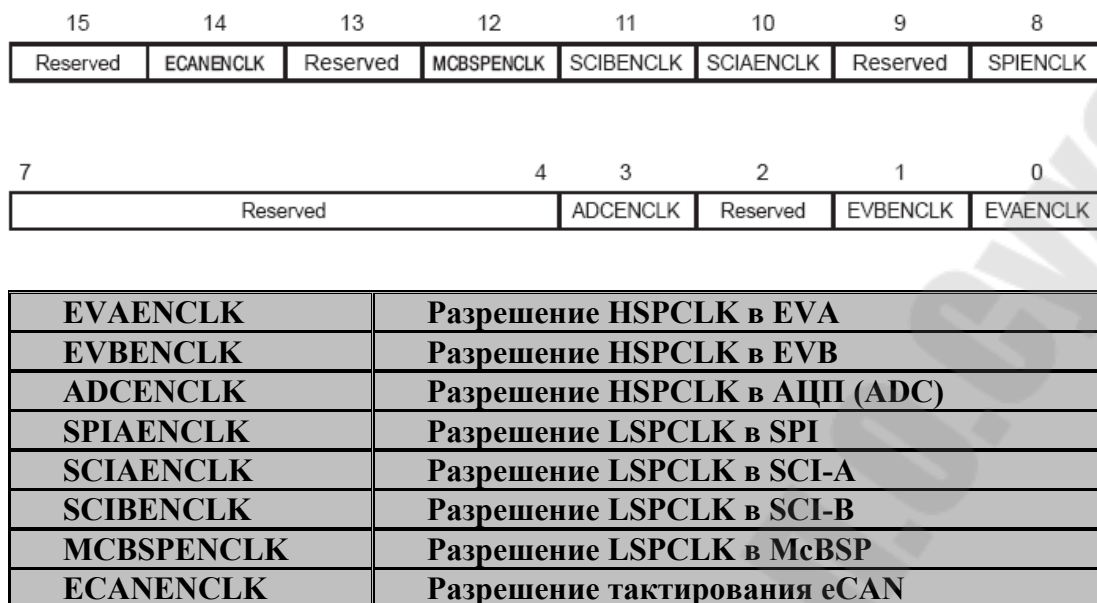


Рис. 2.7. Формат регистра PCLKCR:

1 – разрешить тактирование периферийного модуля; 0 – запретить



Рис. 2.8. Формат регистра управления сторожевого таймера WDCR

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).

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.

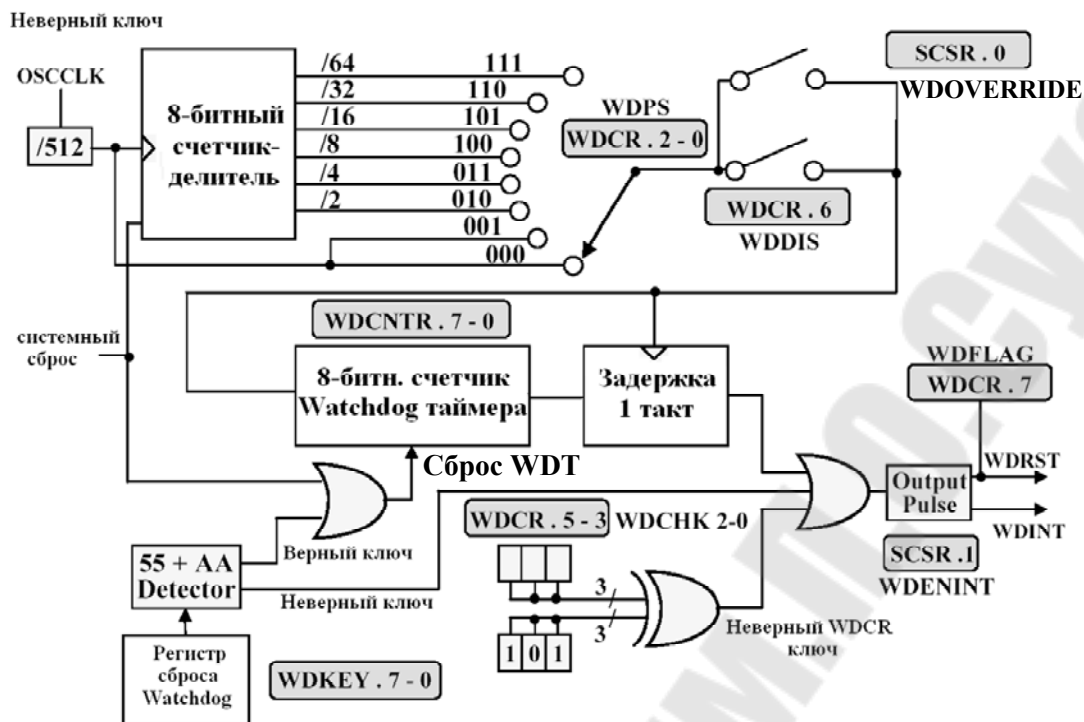


Рис. 2.9. Функциональная схема сторожевого таймера

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.



Рис. 2.10. Формат регистра управления и состояния SCSR

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.

Порядок выполнения работы

Часть I

В части I лабораторной работы необходимо разработать программу «бегущие огни», задавая направление движения: сначала – от края к краю (рис. 2.11, а), а потом, зажигая по два светодиода, – от периферии к центру (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).

1. Создание нового проекта.

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

1.1. Для упрощения работы файл под именем «_lab2.c» с заготовкой текста программы имеется на диске в папке «E:\DspUser\Templates». Знаками «?» в исходном тексте программы отмечены значения, которые необходимо будет заменить на требуемые для правильной работы программы (рис. 2.12). Копируем данный файл в папку проекта E:\DspUser\Lab2 и переименовываем в «Lab2.c». Добавляем в проект и открываем в окне редактора данный файл: Project → Add files to Project.

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

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

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).

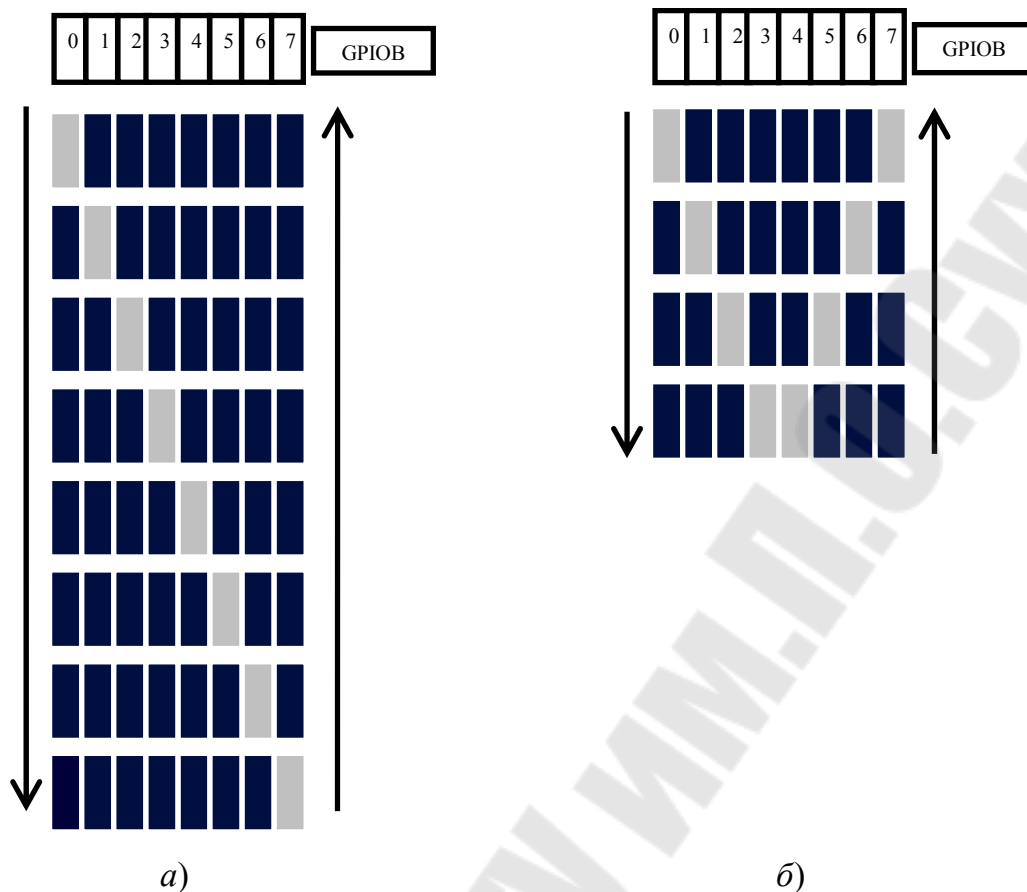


Рис. 2.11. Направление движения светодиодов:
 а – слева направо и наоборот; б – от периферии к центру
 и наоборот

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?;
        //SvsCtrlRegs.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
    //#####

```

Рис. 2.12. Заготовка текста программы «бегущие огни»
(продолжение и окончание см. на с. 23 и 24)

```

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-
    // жевое таймера, предделитель = 1

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

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

```

Рис. 2.12. Продолжение (начало см. на с. 22, окончание – на с. 24)

```

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;
}

```

4

Рис. 2.12. Окончание (начало см. на с. 22 и 23)

3. Инициализация системы.

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

3.1. Присваивая необходимое значение регистру WDCR (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 (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 (область 2 на рис. 2.12).

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

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

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

4. Настройка портов.

Переходим к подпрограмме «Gpio_select()».

4.1. Установкой-сбросом бит в регистрах GPxMUX настраиваем все выводы на работу в качестве портов (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 (область 5 на рис. 2.12). В порту GPIOB настраиваем линии порта GPIOB15 – GPIOB8 на ввод, а GPIOB7 – GPIOB0 на вывод (область 5 на рис. 2.12).

4.4. Сбрасываем все биты регистров GPxQUAL портов А, В, D, Е в ноль (область 5 на рис. 2.12).

5. Расчет и задание временной задержки.

В строке «`delay_loop()`» (область 6 на рис. 2.12) в скобках указываем число n – параметр задержки. Длительность задержки $t_{зд}$ будет пропорциональна данному числу и рассчитывается по формуле, с:

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

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.

Name	Value	Type	Radix
GpioDataRegs.GPBDAT	{...}	unio...	hex
all	4	Uint16	unsigne
bit	{...}	struc...	hex
GPIOB0	0	(unsi...	bin
GPIOB1	0	(unsi...	bin
GPIOB2	1	(unsi...	bin
GPIOB3	0	(unsi...	bin
GPIOB4	0	(unsi...	bin
GPIOB5	0	(unsi...	bin
GPIOB6	0	(unsi...	bin
GPIOB7	0	(unsi...	bin
GPIOB8	0	(unsi...	bin
GPIOB9	0	(unsi...	bin
GPIOB10	0	(unsi...	bin
GPIOB11	0	(unsi...	bin

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

8. Работа со сторожевым таймером.

8.1. Останавливаем выполнение программы.

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

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

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

8.4. Компилируем, компонуем проект и тестируем программу.

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 (рис. 2.11, б).

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

9.4. Компилируем, компонуем проект и тестируем программу.

Часть II

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

1. Отображение состояния ключей на светодиодах.

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

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

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

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

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

1.4. Компилируем, компонуем проект и тестируем программу.

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

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

2.2. Преобразуем программу Lab2c.c. Задаем длительность задержки при переключении светодиодов согласно рис. 2.11, а, равную

($0,1*N+0,001$) с, где N – код, заданный переключателями на линиях В15–В8. Подпрограммы «InitSystem()» и «Gpio_select()» оставляем без изменений.

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

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. Укажите, 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

Формирование функций времени на основе сигнального процессора семейства C28х. Модуль расширения прерываний

Цель работы

Изучить методы формирования функций времени ЦСП TMS320F2812 с помощью таймеров, а также систему прерываний ЦСП. Освоить программирование типовых процедур, использующих прерывания и временные задержки.

Основные теоретические сведения

Система прерываний ядра процессора C28х 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.

Общая структура прерываний ядра C28х 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.

Все 16 прерываний однозначно связаны в памяти с таблицей векторов прерываний ядра BR0M vector (рис. 2.2), содержащей 22-битные адреса (на каждое прерывание – 16 бит в ячейке с младшим 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, так как 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.

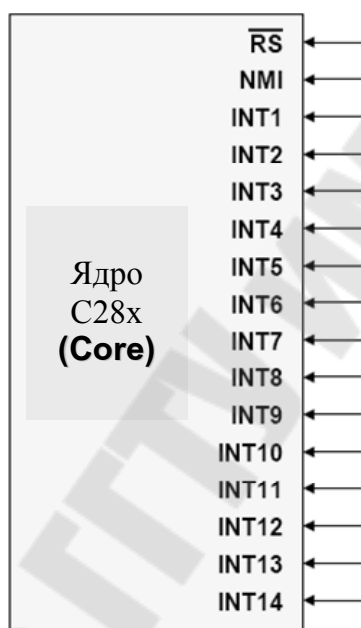


Рис. 3.1. Линии прерываний ядра C28x

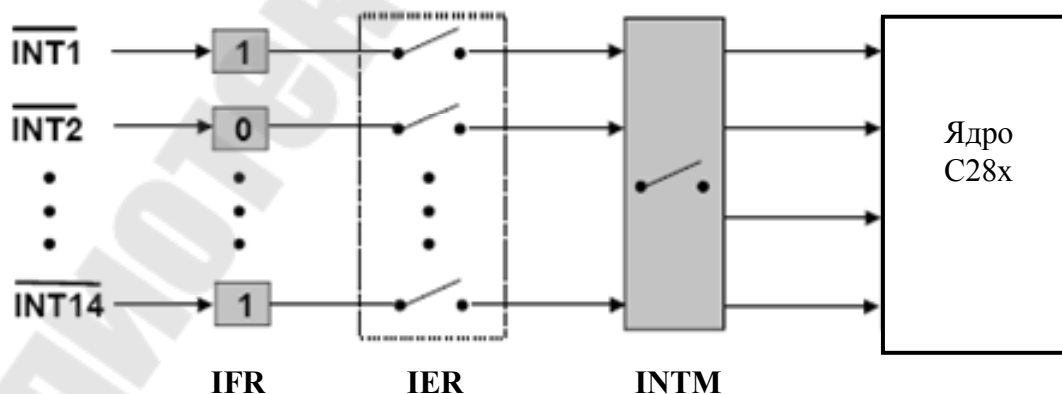


Рис. 3.2. Общая структура прерываний ядра C28x

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: $I\mathbf{E}R_{\text{Bit}}=1$
 Прерывание запрещено: $I\mathbf{E}R_{\text{Bit}}=0$

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: $I\mathbf{F}R_{\text{Bit}}=1$
 Прерывание отсутствует: $I\mathbf{F}R_{\text{Bit}}=0$

Рис. 3.3. Форматы регистров IER и IFR

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

После сброса в режиме микроконтроллера запускается служебная программа Bootloader, которая анализирует выходы 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 (ОТР).

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

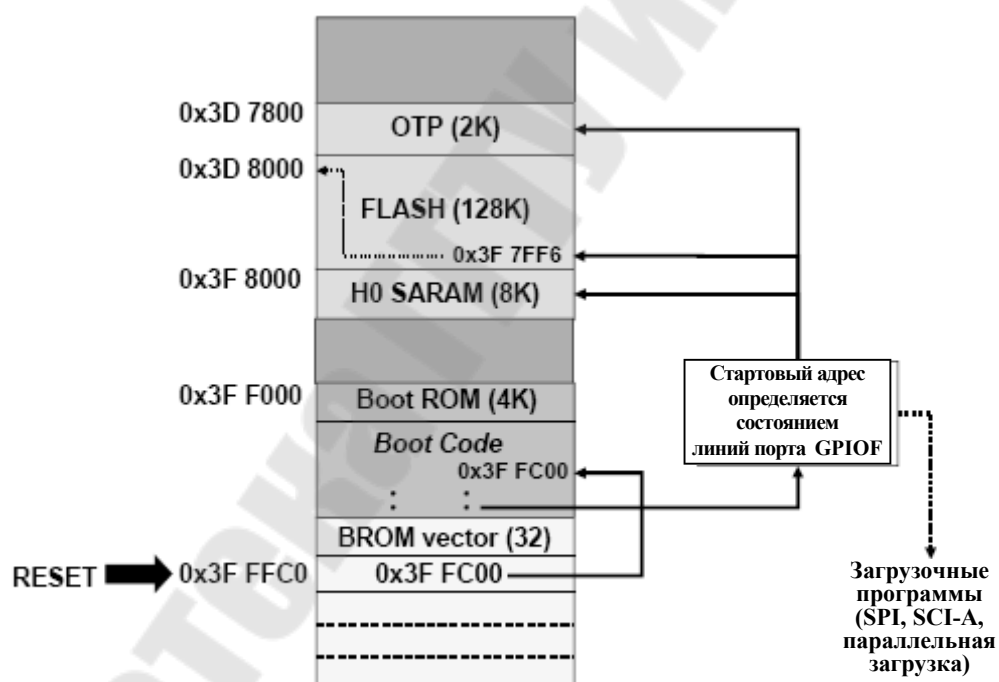


Рис. 3.4. Режимы запуска программы Bootloader

Цифровой сигнальный процессор 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). Так как программный поиск конкретного прерывания в линии при обработке программно занял бы длительное время, то применяется 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, рис. 2.2 и 3.9).

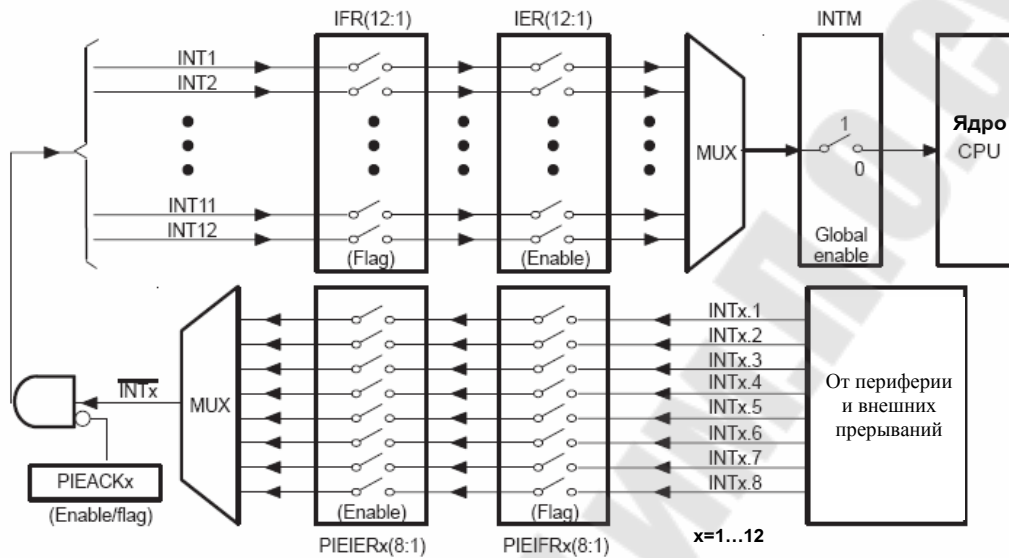


Рис. 3.5. Логика работы регистров PIEIFRx, PIEIERx, PIEACKx

Форматы регистров модуля расширителя прерываний PIEIERx и PIEIFRx показаны на рис. 3.7, данные регистры содержат по 8 информационных бит, т. е. по числу прерываний в группе. В регистре PIECTRL биты 15–1 (PIEVECT) показывают адрес в пределах таблицы векторов PIE vector, из которой был извлечен вектор. Младший значащий бит игнорируется и показываются биты адреса от 1 до 15 (т. е. только четные адреса), что позволяет при чтении из регистра однозначно определить, какое прерывание генерировалось.

ENPIE – бит разрешения извлечения векторов из таблицы PIE-контроллера. Если ENPIE=1, все векторы извлекаются из таблицы векторов PIE vector, а если ENPIE=0, PIE-контролер запрещен, и векторы извлекаются из таблицы CPU-векторов (BROM vector, рис. 2.2).

Прерывания сгруппированы по 8 источников на линию (рис. 3.8). Для разрешения/запрета каждого прерывания используются биты в регистрах PIEIERx (x может принимать значения от 1 до 12), для индикации прерываний – биты в регистрах PIEIFRx. Соответствующий бит в регистре подтверждения PIEACK (активный уровень – 0) определяет номер активного прерывания для ядра CPU внутри группы. В результате каждая группа мультиплексируется в одно из прерываний ядра INT1–INT12 (рис. 3.6).

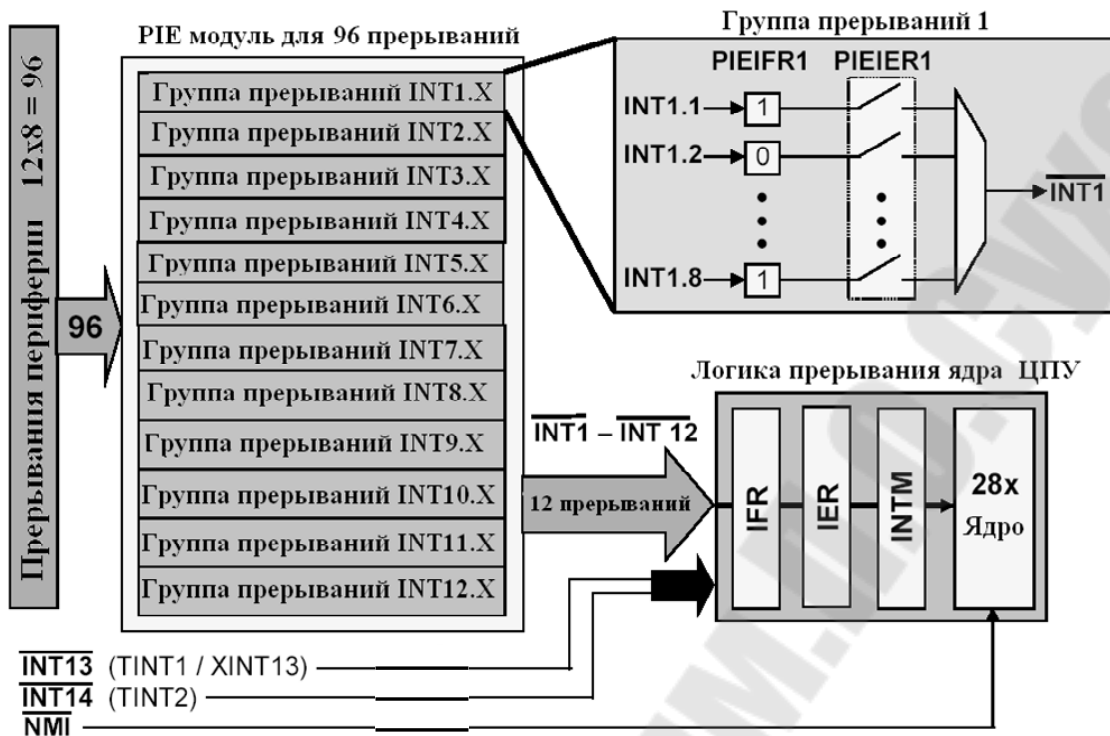


Рис. 3.6. Общая структура расширения прерываний ЦСП C28x

Регистры PIE

PIEIFR_x (x = 1 to 12)

15 - 8	7	6	5	4	3	2	1	0
reserved	INT _x .8	INT _x .7	INT _x .6	INT _x .5	INT _x .4	INT _x .3	INT _x .2	INT _x .1

PIEIER_x (x = 1 to 12)

15 - 8	7	6	5	4	3	2	1	0
reserved	INT _x .8	INT _x .7	INT _x .6	INT _x .5	INT _x .4	INT _x .3	INT _x .2	INT _x .1

Регистр запроса прерывания (PIEACK)

15 - 12	11	10	9	8	7	6	5	4	3	2	1	0
reserved	PIEACK _x											

PIECTRL

15 - 1	0
PIEVECT	ENPIE

Рис. 3.7. Формат регистров модуля расширителя прерываний ЦСП C28x

Таблица векторов прерываний приведена на рис. 3.8.

Примеры векторов прерываний:

- прерывание от встроенного АЦП (ADCINT) – INT1.6;
- прерывание от CPU-таймера 0 (TINT0) – INT1.7.

← приоритет

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

Рис. 3.8. Таблица источников прерываний в PIE

PIE Vector Mapping (ENPIE = 1)

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

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

Полная процедура обработки прерываний при ENPIE=1 показана на рис. 3.10.

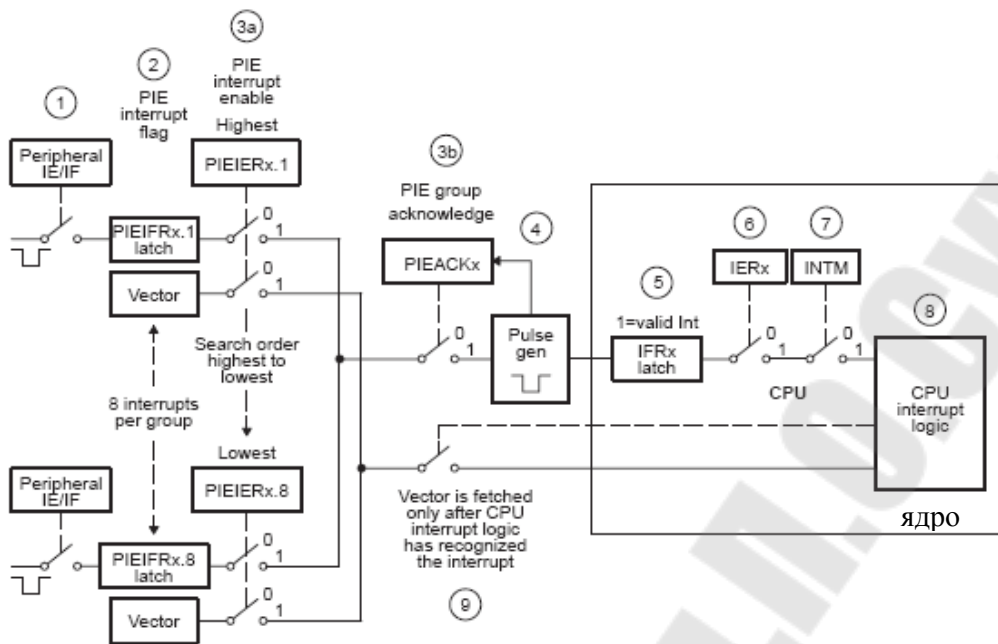


Рис. 3.10. Полная процедура обработки прерываний при ENPIE=1: шаг 1 – генерация прерывания от периферии; шаг 2 – установка флага PIEIFR_{x.y}=1; шаг 3а – проверка одновременного наличия двух условий: PIEIER_{x.y} = 1 и PIEACK_x=0; шаг 3б – установка в «1» бита PIEACK_x для подтверждения прерывания от 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 имеются три 32-битных таймера ядра (CPU timers) с одинаковой структурой. Схема одного таймера приведена на рис. 3.11. Работа таймера разрешается сбросом бита TCR.4. Таймер имеет 16-битный предварительный делитель (прескалер) PSCH: PSC, который формирует счетный импульс вычитания из основного 32-битного счетчика TIMH: TIM. По достижении счетчиком TIMH: TIM нуля формируется сигнал прерывания /TINT, поступающий на ядро. 16-битный регистр TDDRH: TDDR используется для перезагрузки прескалера таймера. Регистр PRDH: PRD содержит значение основного 32-битного счетчика, перезагружаемое в него при очередном переопус-

тошении. 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 применяется 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.

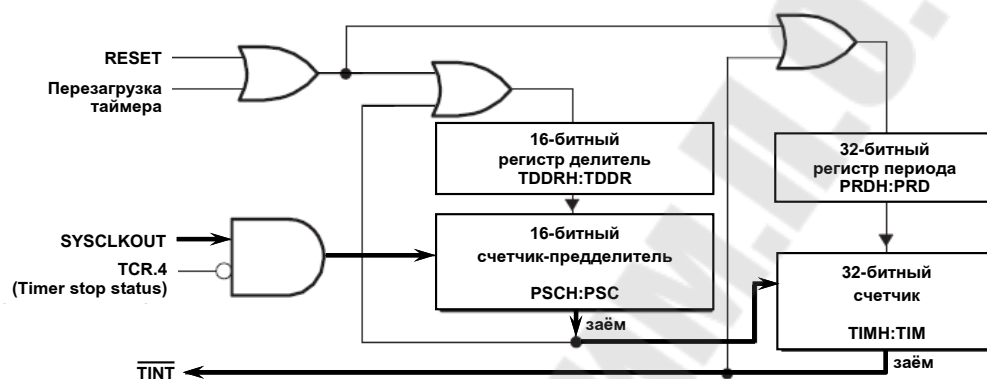


Рис. 3.11. Схема таймера ядра ЦСП семейства C28x (жирной линией показано прохождение тактового сигнала с делением частоты на выход)

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

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

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

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

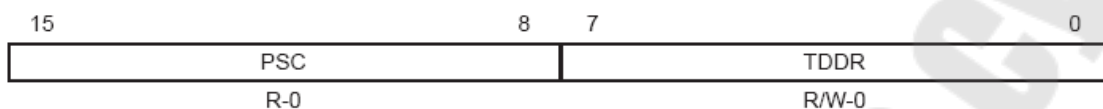


Рис. 3.13. Формат регистров-пределителей $TIMERxTPR$ ($x=0,1,2$)

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

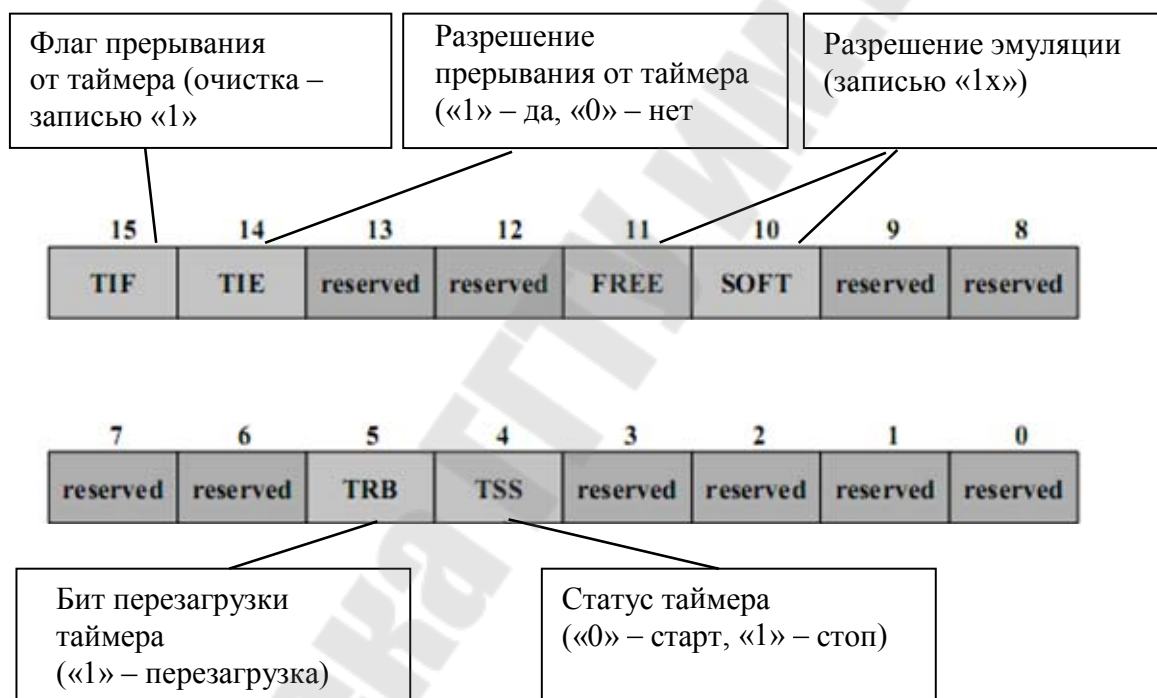


Рис. 3.14. Формат регистров управления таймерами ядра ($TIMERxTCR$)

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

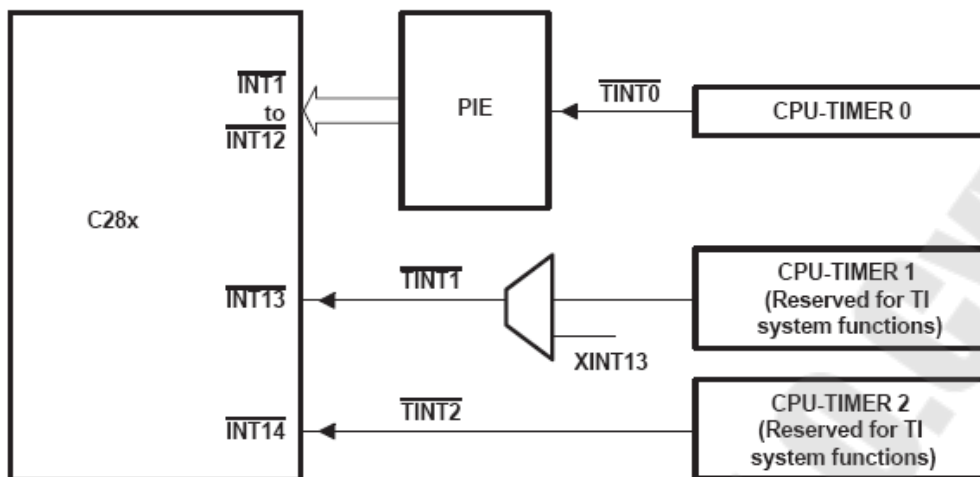


Рис. 3.15. Сигналы прерываний от CPU-таймеров

Порядок выполнения работы

Использование таймера позволяет более эффективно использовать ресурсы процессора. Наиболее простые задачи для таймера – вызывать на выполнение периодические задачи либо инкрементировать глобальную переменную. Данная переменная будет пропорциональна машинному времени, прошедшему с момента запуска таймера.

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

1. Создание нового проекта.

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

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

Структура исходного текста программы показана на рис. 3.16.

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

```

C:\tidcs\c28\dsp281x\v100\DSP281x_common\cmd\F2812_EzDSP_RAM_Ink.cmd
C:\tidcs\c28\dsp281x\v100\DSP281x_headers\cmd\DSP281x_Headers_nonBIOS.cmd
C:\CCStudio_v3.3\C2000\cgtools\lib\rts2800_ml.lib
C:\tidcs\c28\dsp281x\v100\DSP281x_headers\source\DSP281x_GlobalVariableDefs.c
C:\tidcs\c28\dsp281x\v100\DSP281x_common\source\DSP281x_PieCtrl.c
C:\tidcs\c28\dsp281x\v100\DSP281x_common\source\DSP281x_PieVect.c
C:\tidcs\c28\dsp281x\v100\DSP281x_common\source\DSP281x_DefaultIsr.c
C:\tidcs\c28\dsp281x\v100\DSP281x_common\source\DSP281x_CpuTimers.c

```

2. Настройка параметров проекта.

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

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

2.2. Добавление библиотек и создание стека.

Подключаем Си-библиотеки:

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

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

Задаем глубину стека 0x400:

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

```

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

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

void delay_loop(long); ← 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
}

```

Рис. 3.16. Структура исходного текста программы
(продолжение и окончание см. на с. 41 и 42)


```

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;
}

//#####

```

Рис. 3.16. Продолжение (начало см. на с. 40, окончание – на с. 42)

```

// Подпрограмма:      InitSystem
//
// Описание:  Настройка сторожевого таймера на работу,
//            предделитель = 1. Выработка сброса сторожевым
//            таймером. Внутренняя частота работы ЦСП 150 МГц.
//            Запись в предделитель высокоскоростного таймера
//            коэффициента деления 2, а в предделитель
//            низкоскоростного таймера - 4. Запрещение работы
//            периферийных устройств
//#####
void InitSystem(void)
{
    EALLOW;
    SysCtrlRegs.WDCR= 0x?; // Разрешение работы сторожевого
                          // таймера, предделитель = 64
                          // или запрещение работы сторо-
                          // жевого таймера, предделитель = 1

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

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

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

```

Рис. 3.16. Окончание (начало см. на с. 40 и 41)

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 следует 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 сра-

зу 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 (так как прерывания от CPU-таймера 0 относятся именно к данной группе, рис. 3.8):

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

Второй уровень – разрешение прерываний линии 1 ядра ЦПУ. Для этого необходимо настроить регистр IER (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 нужно 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 под-

ключаемом 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(?) ;» (область 4 на рис. 3.16). Также удаляем объявление этой подпрограммы (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) (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 (область 7 на рис. 3.16). Отличием является то, что данную процедуру вместе с макросами EALLOW и EDIS нужно перенести внутрь подпрограммы обработки прерывания от CPU-таймера 0. Поскольку прерывание от таймера «interrupt void cpu_timer0_isr(void)» 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. Компилируем 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 C28x.
4. Назначение и структура модуля расширения прерываний (PIE) ЦСП семейства C28x.
5. Расположение векторов прерываний ЦСП семейства C28x в режимах ENPIE=0 и ENPIE=1.
6. Процедура обработки прерываний при ENPIE=1.
7. Назначение и структура CPU-таймеров ЦСП семейства C28x.
8. Регистры CPU-таймеров ЦСП семейства C28x.
9. Укажите, 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. Поясните, 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

Формирование сигналов широтно-импульсной модуляции на основе сигнального процессора семейства C28x

Цель работы

Изучить принципы формирования сигналов широтно-импульсной модуляции на основе ЦСП TMS320F2812, организацию и программирование встроенной периферии, предназначенной для реализации сигналов специальной формы и ШИМ-модулированных сигналов.

Основные теоретические сведения

1. 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.

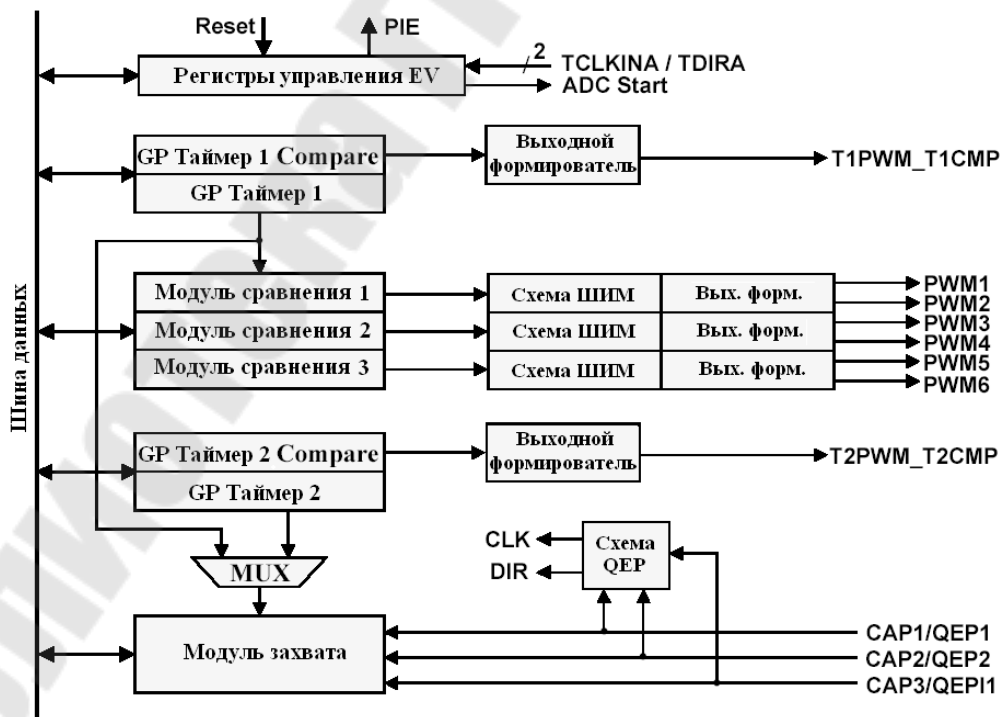


Рис. 4.1. Менеджер Событий

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 T2CTRIIP и T4CTRIIP) вырабатывается строб начала преобразования (SOC).

На рис. 4.1 представлена функциональная схема модуля Менеджера Событий (EVA).

Рассмотрим блоки EVM подробнее.

2. 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.

Таймер общего назначения модуля EVM дан на рис. 4.2.

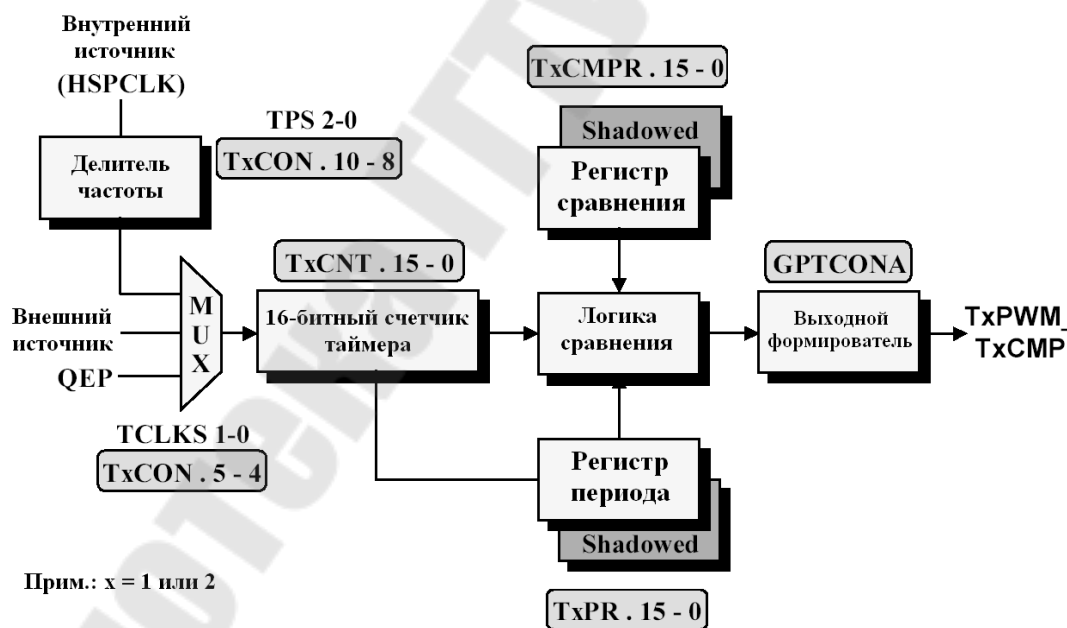


Рис. 4.2. Таймер общего назначения модуля EVM

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 GP таймеров может вырабатывать прерывание на следующие события при достижении:

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

Каждый GP таймер может работать в одном из 4 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.

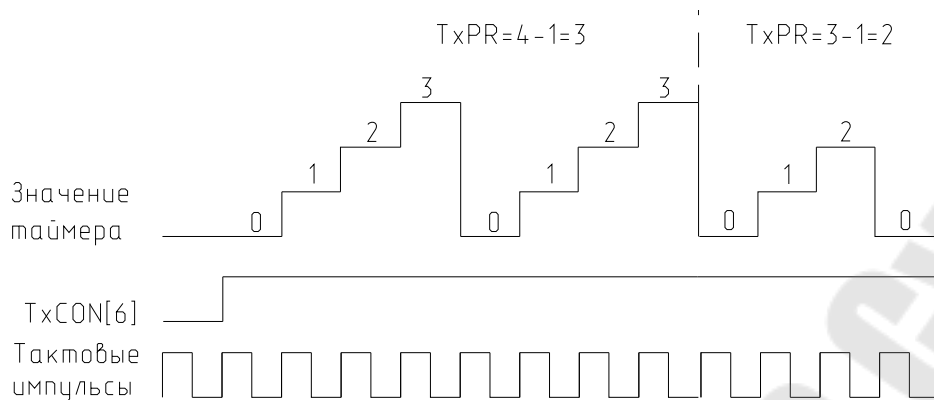


Рис. 4.3. Режим непрерывного счета вверх GP таймера

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»* (рис. 4.4). В этом режиме направление счета зависит от состояния входа TDIRA/B: вверх, если сигнал на TDIRA/B высокого уровня; вниз – если низкого.

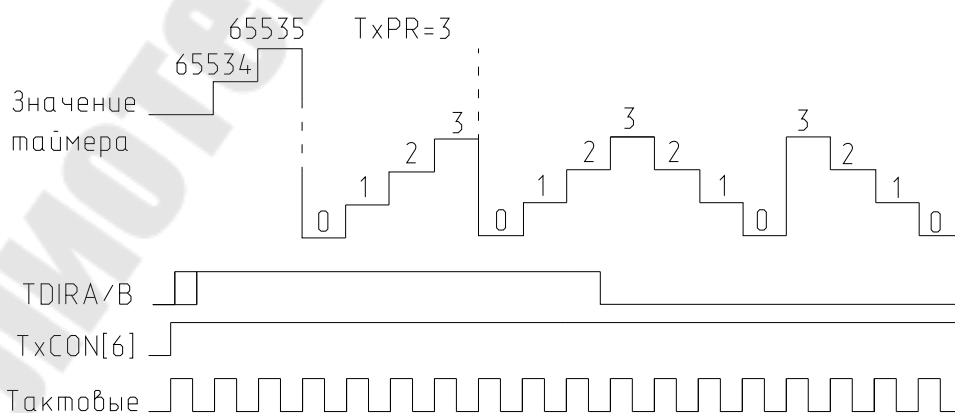


Рис. 4.4. Режим управляемого счета «вверх/вниз» GP таймера

4. *Непрерывный режим счета «вверх/вниз»* (рис. 4.5). В отличие от предыдущего режима, направление счета изменяется при достижении нуля или значения в регистре периода. Продолжительность периода в этом режиме равна $2 \cdot (T_{xPR})$.

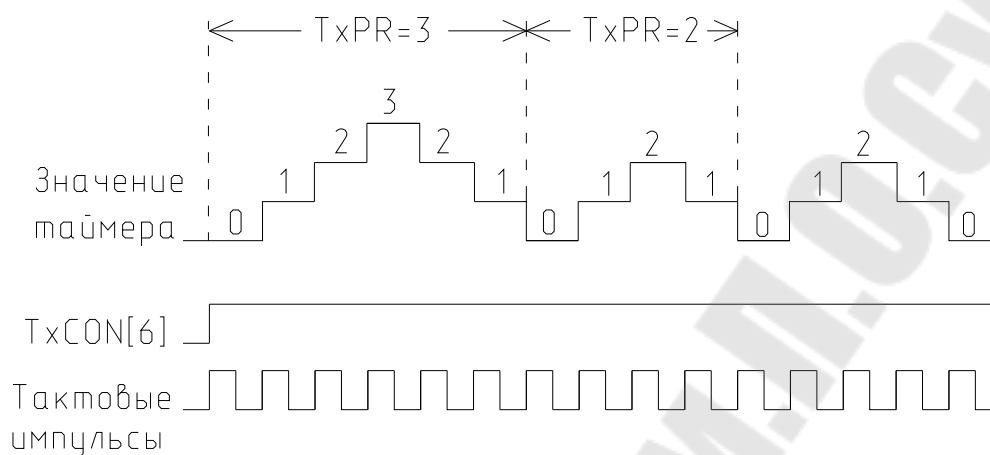


Рис. 4.5. Непрерывный режим счета «вверх/вниз»

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 цепей CAP_x (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_x);
- 2) есть один 16-разрядный регистр статуса FIFO (CAPFIFO_x);
- 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 ми-

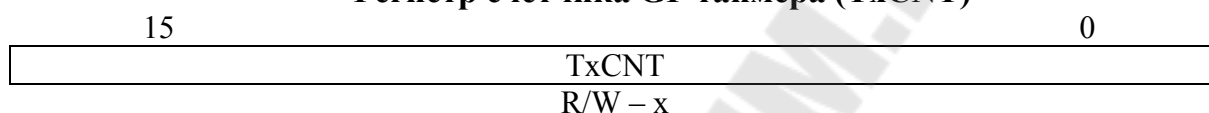
нимума программную загрузку ядра при операциях измерений длительности, периодических выборок и генерации сигналов ШИМ.

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)

3. Формат регистров модуля Менеджера Событий

Формат регистра счетчика GP таймера дан на рис. 4.6.

Регистр счетчика GP таймера (TxCNT)

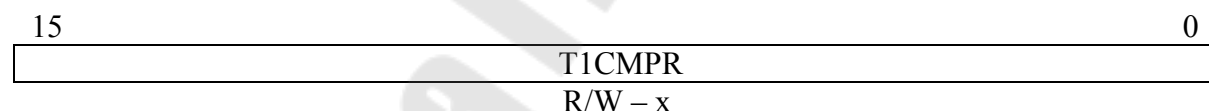


Биты	Название	Описание
15:0	TxCNT	Мгновенное значение таймера-счетчика 1

Рис. 4.6. Формат регистра счетчика GP таймера

Формат регистра сравнения GP таймера приведен на рис. 4.7.

Регистр сравнения GP таймера (TxCMPR)

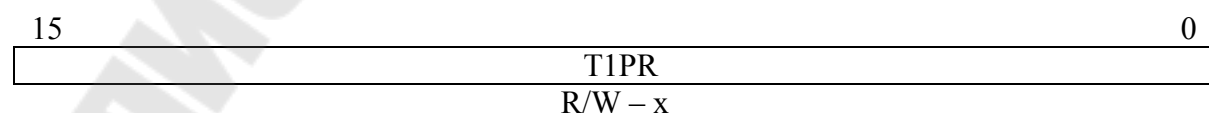


Биты	Название	Описание
15:0	T1CMPR	Хранимое значение сравнения таймера-счетчика 1

Рис. 4.7. Формат регистра сравнения GP таймера

Формат регистра периода GP таймера показан на рис. 4.8.

Регистр периода GP таймера (TxPR)

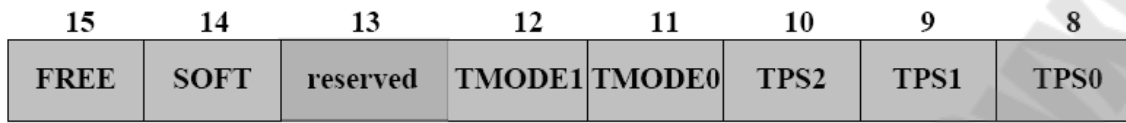


Биты	Название	Описание
15:0	T1PR	Хранимое значение периода таймера-счетчика 1

Рис. 4.8. Формат регистра периода GP таймера

Регистры управления таймером (TxCON) представлены на рис. 4.9.

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: Разрешение схемы сравнения:

Запуск таймера:

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

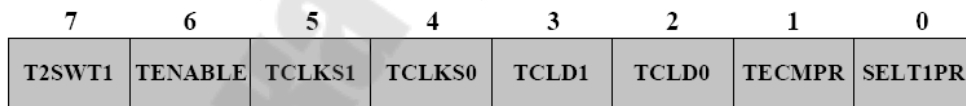
- 0 Запретить
- 1 Разрешить

Источник тактирования:

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

Выбор регистра сравнения:

- 0 Свой регистр сравнения
- 1 Регистр сравнения Таймера 1



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 Немедленно
- 11 Резервные

Рис. 4.9. Регистры управления таймером (TxCON)

Регистр А управления GP таймером (GPTCONA) приведен на рис. 4.10.

Старший байт:

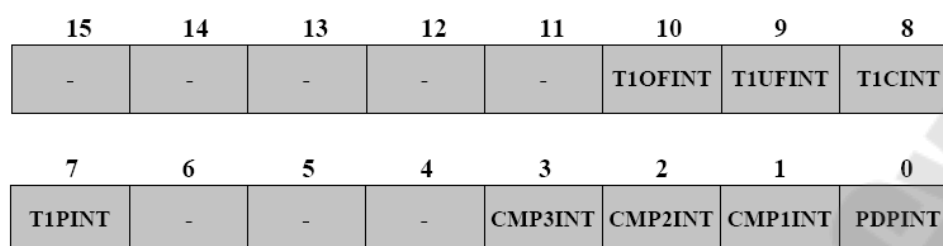


Младший байт:



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

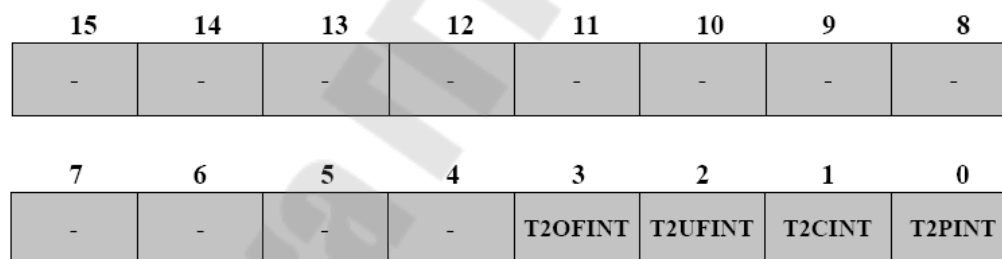
Формат регистра А маски прерываний (EVAIMRA) дан на рис. 4.11.



Разрешение прерываний:	Бит	Назначение:
0 Запретить прерывания		
1 Разрешить прерывания		
	10:	Максимальное значение Таймера 1
	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)

Формат регистра В маски прерываний (EVAIMRB) представлен на рис. 4.12.



Разрешение прерываний:	Бит	Назначение
0 - запретить прерывание		
1 - разрешить прерывание		
	3:	Максимальное значение Таймера 2
	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)

Формат регистра С маски прерываний (EVAIMRC) показан на рис. 4.13.



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

Формат регистров флагов прерываний приведен на рис. 4.14.

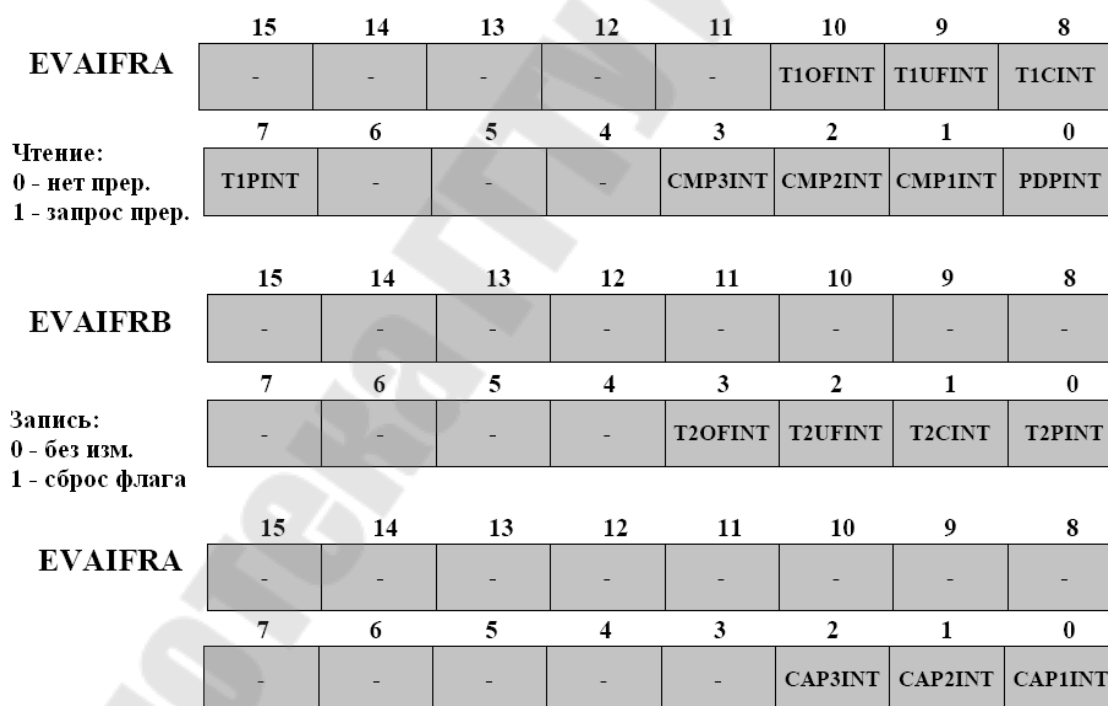


Рис. 4.14. Формат регистров флагов прерываний

4. 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 том числе от коротких замыканий. Функция PDP позволяет защитить внешние силовые устройства при системных сбоях.

5. 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 ее 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 называют «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 «зашиита» таблица синуса. Таблица использует дробно-целочисленный формат с фиксированной запятой IQ, или Q. Представлена она в формате Q30. Это означает, что 30 младших бит представляют дробную часть числа, один – целую и старший – знак числа.

Порядок выполнения 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 следует производить сброс Watchdog и загружать новую ноту. Обратите внимание на период между прерываниями таймера, периодом Watchdog и периодом перезагрузки новой ноты.

Соответствие частот музыкальным нотам приведено в табл. 4.1.

Таблица 4.1

Соответствие частот музыкальным нотам

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

1. Создание нового проекта.

В Code Composer Studio создаем новый проект Lab4.pjt. Открываем файл Lab4.c и сохраняем его E:\C281x\Labs\Lab4\lab4.c.

Добавляем в проект файлы:

```
C:\CCStudio_v3.3\C2000\cgtools\lib\rts2800_ml.lib
C:\tidcs\c28\dsp281x\v100\DSP281x_common\source\DSP281x_CpuTimers.c
C:\tidcs\c28\dsp281x\v100\DSP281x_common\source\DSP281x_DefaultIsr.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_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\DSP281x_Headers_nonBIOS.cmd
```

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\lab4.out.

3. Преобразование файла Lab4.c.

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

Переходим к подпрограмме «Gpio_select()». Настраиваем линию 6 порта GPIOA на вывод функции T1PWM (GRAMUX).

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 «TCMP0E» 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 (TECMR);

Определяем значения регистра периода (T1PR) для разных частот и заносим их в массив `int frequency [8] = {?, ?, ?, ?, ?, ?, ?, ?}`. Исходя из формулы

$$T1_PWM_Freq = \frac{150 \text{ МГц}}{HISPCP \cdot TPS \cdot T1PR} \quad (4.1)$$

для первой ноты в регистр периода необходимо занести: $T1PR = 150 \text{ МГц} / (2 \cdot 128 \cdot 264 \text{ Гц}) = 2219$, для второй ноты $T1PR = 150 \text{ МГц} / (2 \cdot 128 \cdot 297 \text{ Гц}) = 1973$ 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.

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 табл. 4.1 представлено соответствие частот и нот музыкального звукоряда.

Для расчета частоты синусоидальных колебаний используется формула

$$f_{\sin} = \frac{f_{PWM}}{N_{PWM}}, \quad (4.2)$$

где f_{PWM} – частоты модулированных импульсов; N_{PWM} – число модулированных импульсов за период.

1. Создание нового проекта.

1.1. Создаем новый проект Lab4A.pjt.

1.2. Добавляем в проект файлы (п. 1.2 части I), кроме файла DSP281x_CpuTimers.c, и вместо lab4.c загружаем lab4A.c.

2. Настройка параметров проекта, компоновка проекта и загрузка выходного файла (аналогично 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 Lab4A.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

$$f_{PWM} = \frac{f_{CPU}}{T1PR \cdot TPS_{T1} \cdot HISCP}, \quad (4.3)$$

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

Принцип формирования синусоидального сигнала широтно-импульсной модуляцией приведен на рис. 4.15.

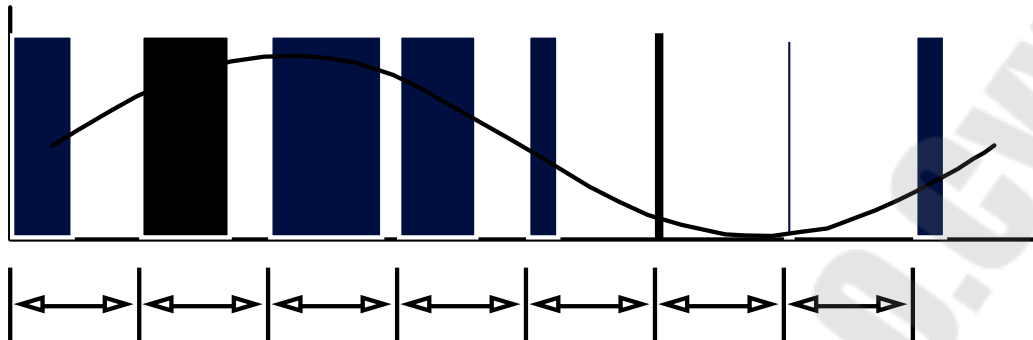


Рис. 4.15. Принцип формирования синусоидального сигнала широтно-импульсной модуляцией

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 «Q-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 Lab4A.cmd.

6. Расчет необходимого значения регистра сравнения.

Учитывая, что разность значений регистра периода и регистра сравнения определяет ширину выходного модулированного импульса (рис. 4.16), а значения функции синуса изменяются от -1 до $+1$, рассчитываем значение регистра сравнения по формуле:

$$T1CMPR = T1PR - _IQ30\text{mpy}(\text{sine_table}[\text{index}] + _IQ30(0.9999), T1PR/2), \quad (4.4)$$

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

Изменение ширины импульса при ШИМ по синусоидальному закону представлено на рис. 4.16.

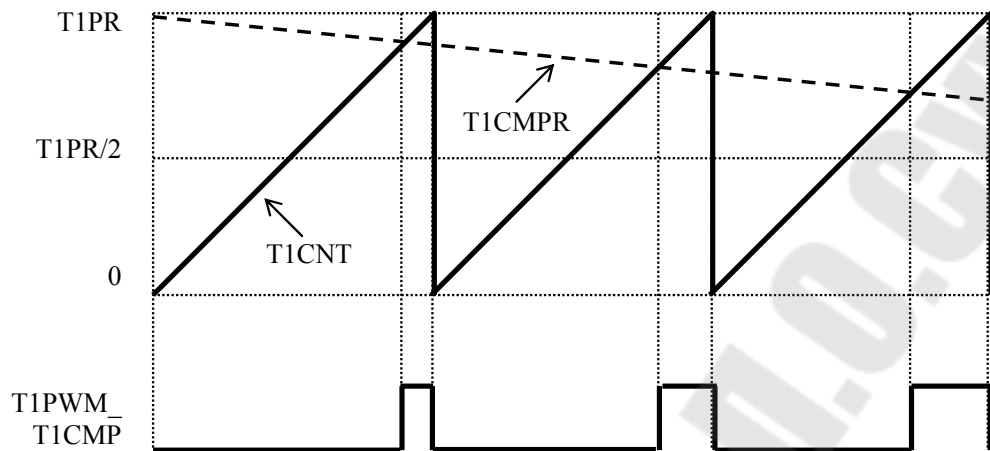


Рис. 4.16. Изменение ширины импульса при ШИМ по синусоидальному закону

Чтобы избежать переполнения, применим функцию насыщения `_IQsat(x, max, min)`. Тогда окончательно записываем в подпрограмму `T1_Compare_isr` формулу для расчета необходимого значения регистра сравнения:

$$\text{EvaRegs.T1CMPR} = \text{EvaRegs.T1PR} - \text{IQsat}(\text{_IQ30mpy}(\text{sine_table}[\text{index}] + \text{_IQ30}(0.9999), \text{EvaRegs.T1PR}/2), \text{EvaRegs.T1PR}, 0).$$

7. Расчет частоты синусоидальных колебаний.

Для этого используем формулу (4.2), задаемся $f_{PWM} = 50$ кГц, $N_{PWM} = 128$, т. е. из таблицы значений синуса выбираем каждое четвертое число, добавляем строку:

```
index += 4.
```

Тогда $f_{sin} = 390,6$ 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.
4. Регистры Менеджера Событий.
5. ШИМ-сигнал: определение, формирование с помощью Менеджера Событий.
6. Прерывания Менеджера Событий.

Лабораторная работа № 5

Ввод и масштабирование аналоговых сигналов в процессорах семейства C28x

Цель работы

Изучить структуру встроенного АЦП цифрового сигнального процессора TMS320F2812, режимы его работы. Научиться разрабатывать простейшие программы с использованием АЦП.

Основные теоретические сведения

1. 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.

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.

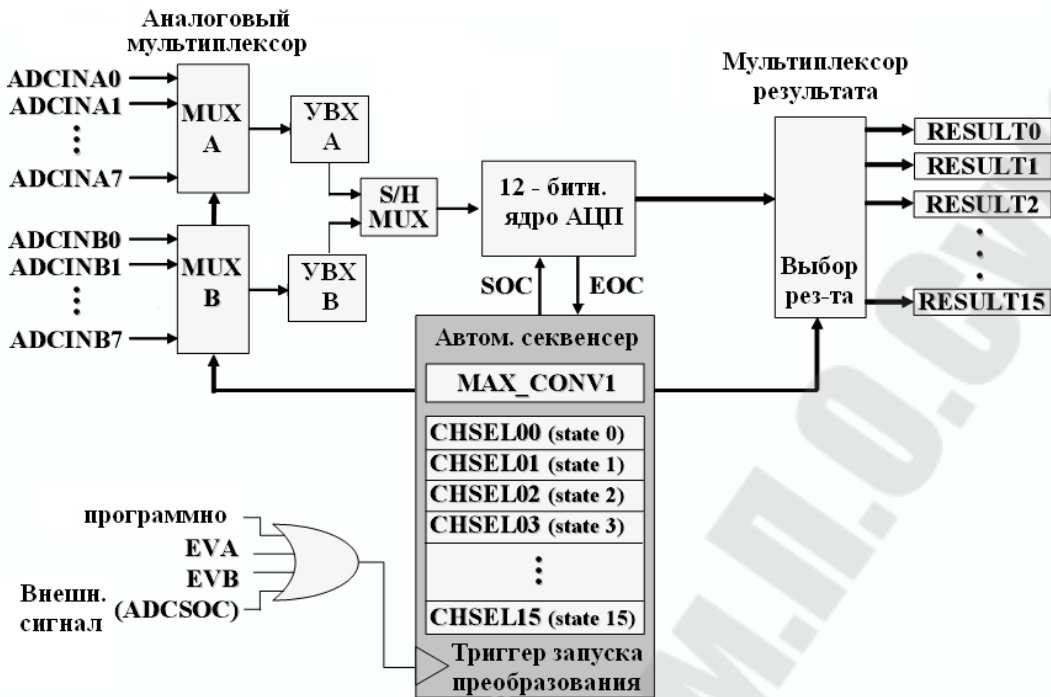


Рис. 5.1. Блок-схема модуля АЦП в каскадном режиме

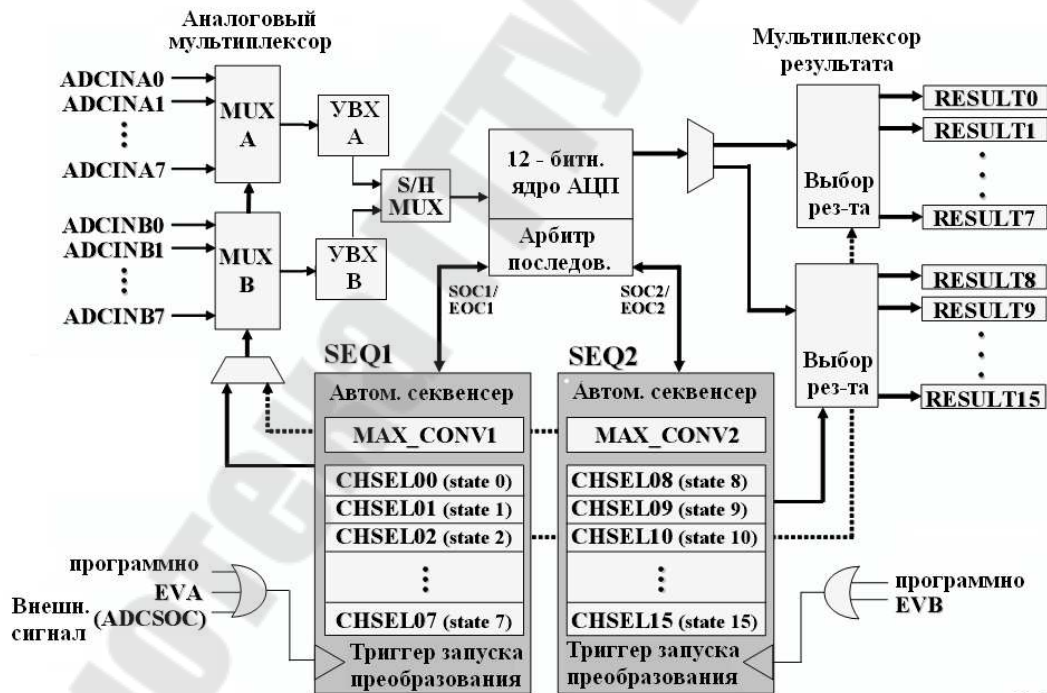


Рис. 5.2. Блок-схема модуля АЦП в двухканальном режиме

В двухканальном режиме (рис. 5.2) автоматический секвенсер разделяется на две независимые части («SEQ1» и «SEQ2») со своими настройками и сигналами запуска. Входные каналы задается в регистрах CHSEL00...CHSEL07 для последовательности SEQ1

и CHSEL08...CHSEL15 для последовательности SEQ2, результаты преобразования сохраняется 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.

2. Время преобразования и система тактирования АЦП

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

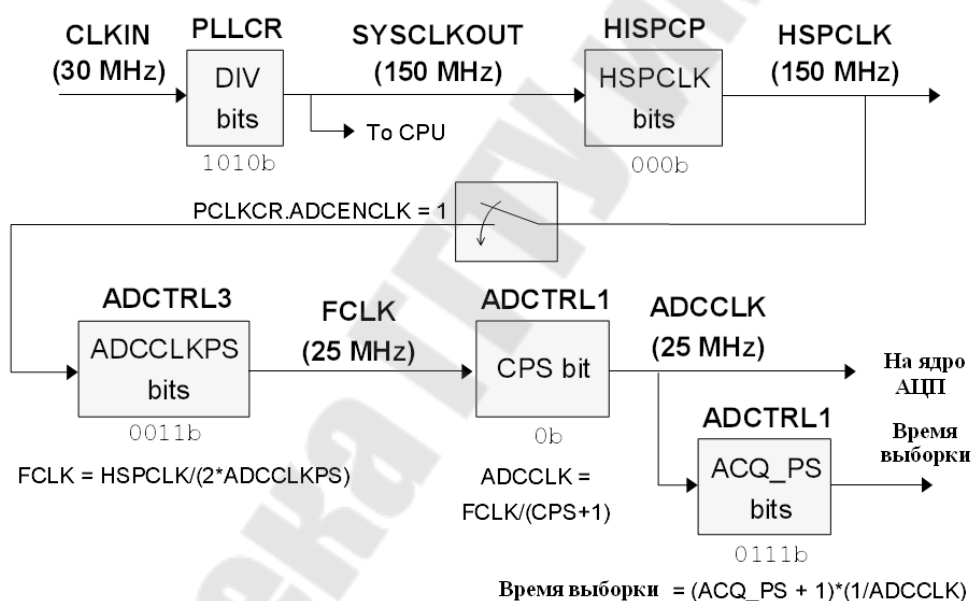


Рис. 5.3. Пример конфигурации модуля тактирования АЦП

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

Формат регистров модуля АЦП представлен ниже:

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	$HSPCLK/[30*(ADCTRL1[7] + 1)]$

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 $ADCTRL1[7]+1$.

Регистры 1–3 управления АЦП представлены на рис. 5.4–5.6.

Старший байт

Сброс АЦП

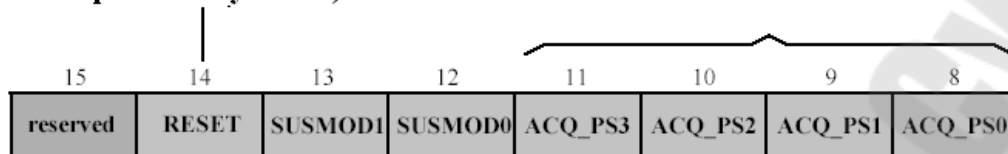
0 - не влияет

1 - сброс АЦП (после сброса авт. уст. в 0)

Время выборки:

$$t = ACQ_PS0..3 + 1$$

Время зависит от тактовой частоты ADCCLK



Биты управления эмулятором:

00 - Не ост. АЦП при ост. эмулятора

01 - остановка после преобр. последов.

10 - остановка после преобр. канала

11 - немедленная остановка

Младший байт

Перезапуск:

0 - остановка после преобр. посл-ти

1 - непрерывное преобразование

Режим:

0 - двухканальный

1 - каскадный



Прескалер:

0 - CLK/1

1 - CLK/2

Перезагрузка конвейера

(исп-ся при непрерывном преобр.)

0 - перезагрузка по дост. MAX_CONVn

1 - перезагрузка по дост. последнего зн-я

Рис. 5.4. Регистр 1 управления АЦП (ADCTRL1)

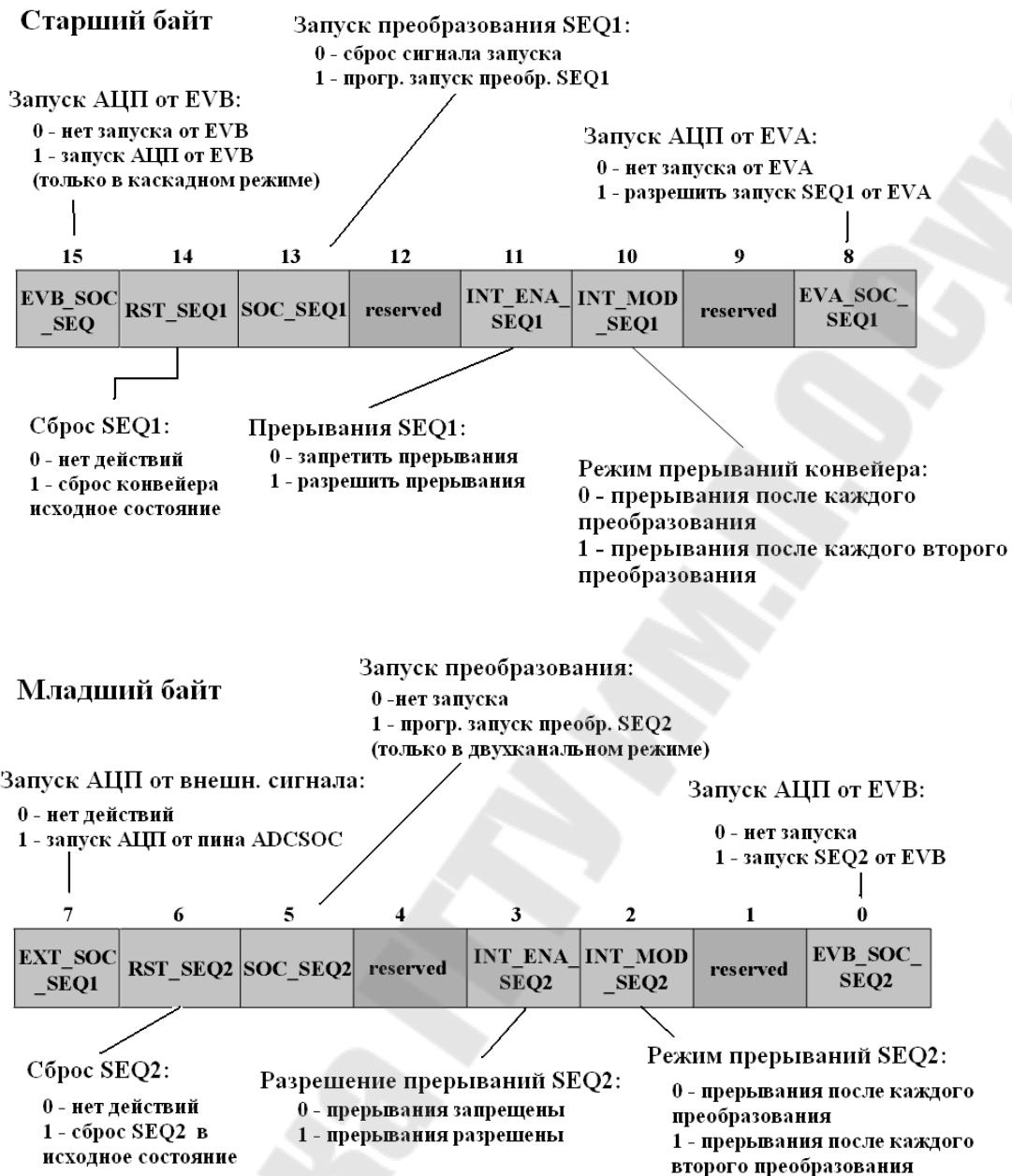


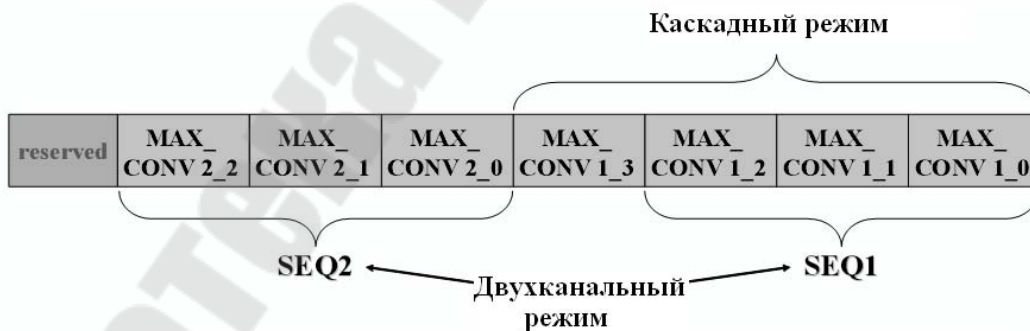
Рис. 5.5. Регистр 2 управления АЦП (ADCTRL2)



Рис. 5.6. Регистр 3 управления АЦП (ADCTRL3)

Формат регистра регистра числа каналов для преобразования ADCMAXCONV приведен на рис. 5.7.

- ◆ 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 регистра числа каналов для преобразования ADCMAXCONV

Регистр статуса автопоследовательности (ADCASEQASR) показан на рис. 5.8.

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 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

Формат регистра ADCRESULTn представлен на рис. 5.9.

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 работы

1. Создание проекта.

1.1. В Code Composer Studio создаем новый проект Lab5.pjt. Копируем из папки c:\tides файл lab5.c в папку с созданным проектом. Добавляем lab5.c в проект.

1.2. Добавляем в проект следующие файлы:

```
C:\tides\c28\dsp281x\v100\DSP281x_headers\source\DSP281x_GlobalVariableDefs.c
C:\tides\c28\dsp281x\v100\DSP281x_common\cmd\F2812_EzDSP_RAM_Ink.cmd
C:\tides\c28\dsp281x\v100\DSP281x_headers\cmd\F2812_Headers_nonBIOS.cmd
C:\ti\c2000\cgtools\lib\rts2800_ml.lib
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_Adc.c
C:\tides\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:\tides\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 (лабораторная работа № 2), устанавливаем коэффициент 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).

3. Инициализация портов (подпрограмма «Gpio_select()»).

3.1. Настраиваем все выходы на работу в качестве портов (регистр GPxMUX) (лабораторная работа № 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) (лабораторная работа № 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 двухпоследовательный режим, запрещаем непрерывный режим, коэффициент деления 1 (регистр ADCTRL1).

5.2. Записываем количество преобразований 2 (регистр AD-SMAXCONV).

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)

5.5. 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 с. В программе также следует 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
**Изучение отладочного модуля
для ПЛИС Spartan-3E Starter Kit.
Реализация логических функций**

Цель работы

Ознакомиться со структурой отладочного модуля Spartan-3E Starter Kit, научиться создавать и отлаживать на его основе простейшие программы, реализующие функции комбинационных логических элементов.

Основные теоретические сведения

Аппаратная часть стенда состоит из отладочной платы Spartan-3E Starter Kit, а программная – из САПР WebPACK ISE.

Внешний вид платы изображен на рис. 6.1. Эта плата имеет много встроенных, уже готовых к использованию компонентов. На плате установлена ПЛИС семейства Spartan-3E xc3s500e. Широкие возможности платы обеспечиваются наличием большого количества интерфейсов ввода/вывода на плате и интегрированного USB-JTAG загрузчика-отладчика. Отладочная плата позволяет разрабатывать и отлаживать сложные проекты с наименьшей затратой времени.

Отладочная плата Spartan-3E Starter Kit имеет следующие компоненты:

- ПЛИС XC3S500E Spartan-3E FPGA;
- память Platform Flash PROM объемом 4 Мбит;
- ПЛИС XC2C64A CoolRunner CPLD;
- память DDR SDRAM объемом 64 Мбайт;
- память NOR Flash объемом 64 Мбайт;
- память SPI serial Flash объемом 16 Мбит;
- двухстрочный жидкокристаллический индикатор (LCD);
- порт PS/2;
- порт VGA;
- интерфейс Ethernet;
- два порта RS-232;
- порт USB;
- кварцевый резонатор частотой 50 МГц;
- память EEPROM;
- дополнительный разъем FX2 фирмы Hirose;

- три дополнительных шестипиновых разъема фирмы Digilent;
- четырехканальный ЦАП;
- двухканальный АЦП с программируемым предусилителем;
- вращающаяся нажимаемая кнопка;
- восемь светодиодов;
- четыре кнопки;
- четыре переключателя;
- разъем для подключения внешнего генератора;
- восьмипиновый сокет для дополнительного генератора.

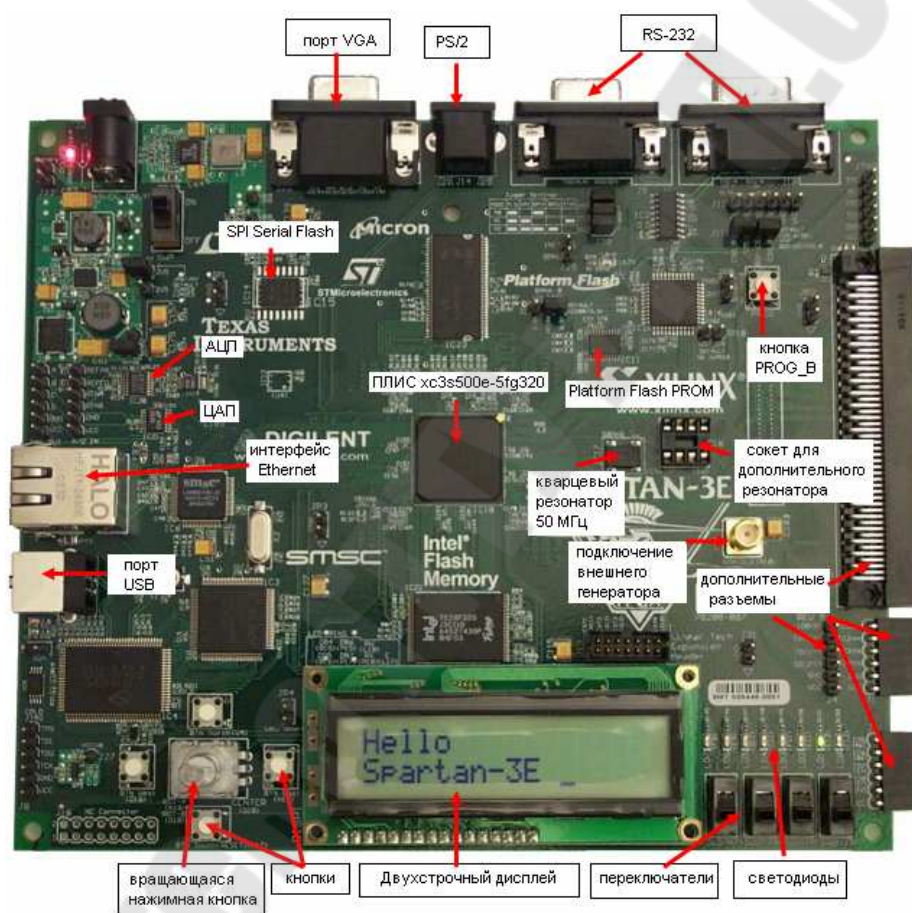


Рис. 6.1. Отладочная плата Spartan-3E Starter Kit

Программным обеспечением для проектирования систем на базе ПЛИС является WebPACK ISE. В результате его запуска будет открыт «Навигатор проекта» («Project Navigator») основная программа САПР WebPACK ISE. «Навигатор проекта» позволяет упорядочить файлы с исходным описанием проектируемого устройства, тестовыми модулями, модулями временных и топологических ограничений, а также предоставляет возможность простого доступа ко всем процес-

сам, необходимым при проектировании цифрового устройства на базе ПЛИС с архитектурами FPGA и CPLD. На рис. 6.2 показан внешний вид основного окна «Навигатора проекта» и его компоненты.

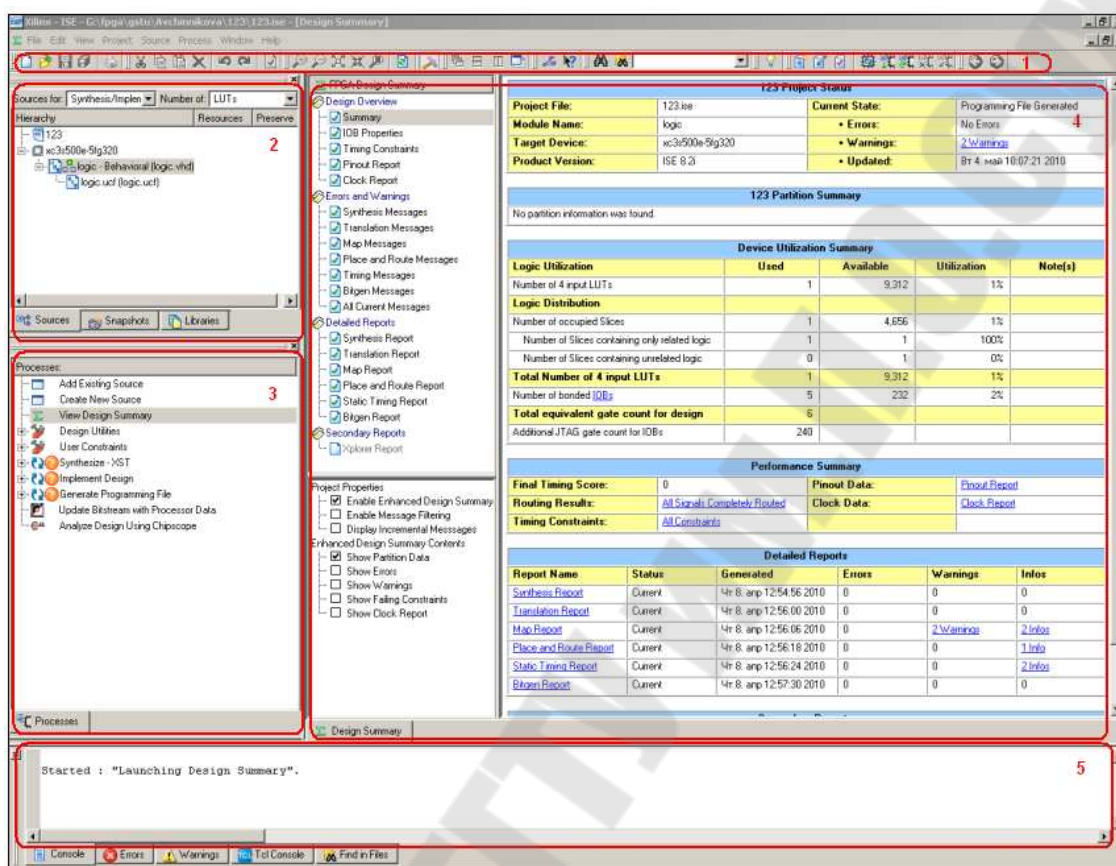


Рис. 6.2. Внешний вид основного окна «Навигатора проекта»: 1 – инструментальная панель («Toolbar»); 2 – окно описания проекта («Source window»); 3 – окно процессов («Process window»); 4 – рабочий стол («Workspace»); 5 – окно отчетов («Transcript window»)

В процессе разработки цифрового устройства на базе ПЛИС в общем случае можно выделить следующие этапы:

- создание нового проекта, включающее выбор семейства и типа ПЛИС, а также средства синтеза;
- подготовка описания устройства в схемотехнической, текстовой или алгоритмической форме;
- синтез устройства;
- функциональное моделирование и тестирование;
- размещение и трассировка проекта в кристалле;
- временное моделирование;
- программирование ПЛИС (загрузка проекта в кристалл).

Каждому из этих этапов соответствует определенный набор процессов, к которым можно получить доступ из «Навигатора проектов».

Порядок выполнения работы

В ходе лабораторной работы необходимо разработать программу на языке Verilog, загрузить ее в ПЛИС и проверить правильность работы. Программа должна вычислять заданное булево выражение для четырех переменных А, В, С, D. При проверке правильности работы следует задавать значения переменных с помощью переключателей, а результат выводить на светодиод (рис. 6.3).

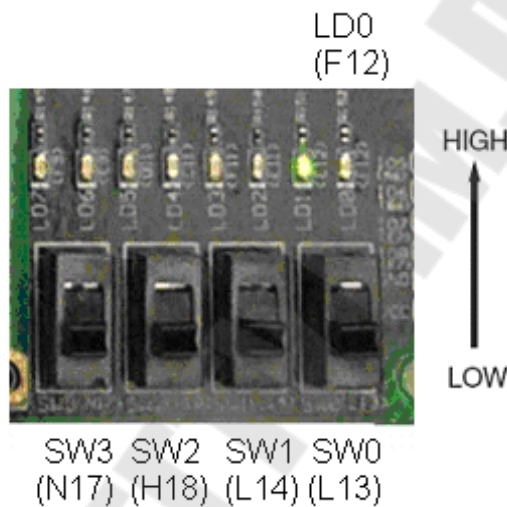


Рис. 6.3. Переключатели и светодиоды

1. Запуск САПР WebPACK ISE.

Для запуска САПР WebPACK ISE требуется выбирать в главном меню операционной системы пункта «ПУСК» → «Программы» → «ISE Design Suite 14.7» → «Project Navigator».

2. Создание нового проекта.

Для создания нового проекта необходимо проделать ряд шагов:

2.1. Выбрать в меню пункт «File» → «New Project...», после чего будет запущен мастер нового проекта «New Project Wizard» (рис. 6.4).

2.2. Ввести или выбрать местоположение создаваемого проекта в поле «Project Location».

2.3. Убедиться, что в качестве типа основного файла описания устройства («Top-Level Source Type») выбран «HDL».

Нажать кнопку «Next >» для перехода к странице выбора кристалла.

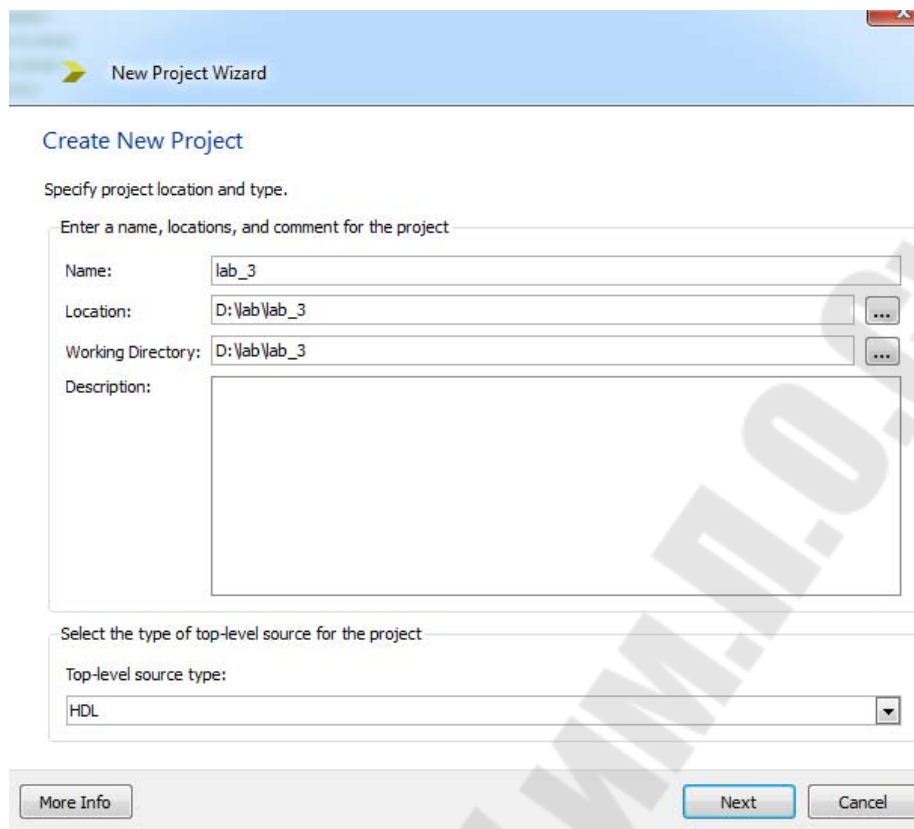


Рис. 6.4. Мастер нового проекта «New Project Wizard»

2.4. Заполнить свойства проектируемого устройства, как показано на рис. 6.5:

- категория устройства («Product Category»): All;
- семейство («Family»): Spartan-3E;
- устройство («Device»): XC3S500E;
- исполнение («Package»): FG320;
- градация по быстродействию («Speed»): -4;
- инструмент синтеза («Synthesis Tool»): XST (VHDL/Verilog);
- симулятор («Simulator»): ISim (VHDL/Verilog).

2.5. Убедиться, что установлена опция «Enable Enhanced Design Summary». В остальных полях необходимо оставить значения «по умолчанию».

2.6. Нажать кнопку «Next >» для перехода к странице создания заготовки основного файла с описанием устройства. После создания этого файла процесс создания проекта будет закончен.

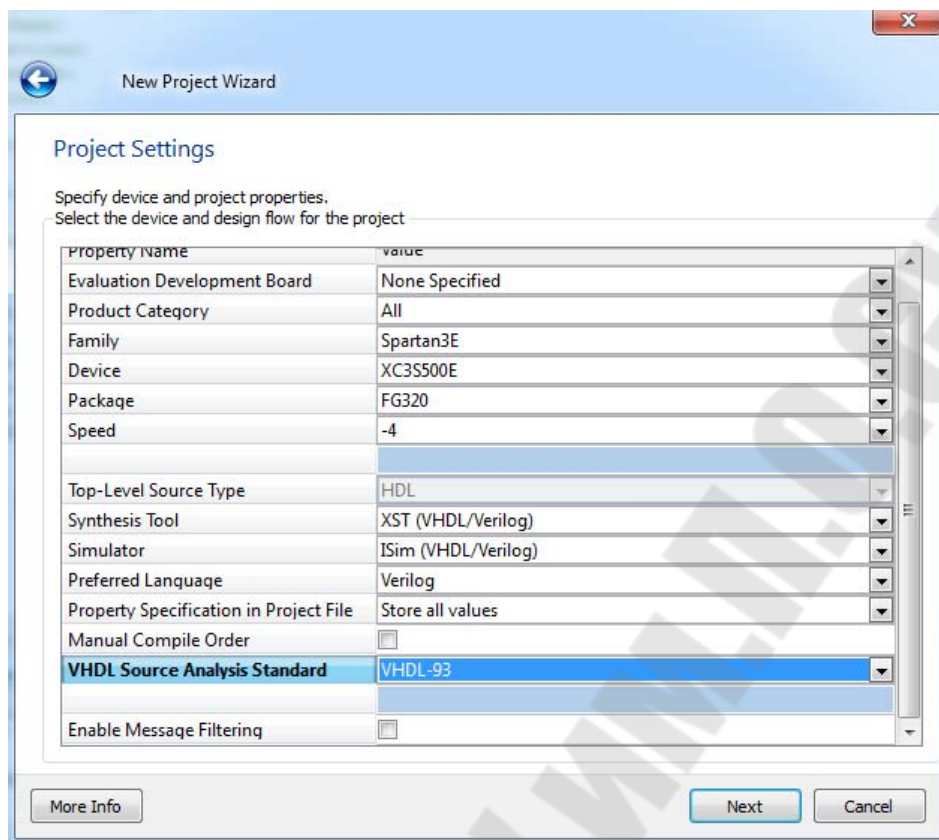


Рис. 6.5. Окно свойств проектируемого устройства

3. Создание описания устройства.

Проект может включать от одного до нескольких файлов с описанием проектируемого цифрового устройства. Описания можно создавать в виде электрических принципиальных схем, в виде HDL-описания на языке VHDL или Verilog или диаграмм состояний и переходов между ними. Кроме того, допускаются смешанные способы описания, представляющие собой сочетание перечисленных форм. В данном случае остановимся на создании HDL-описания.

Для создания HDL-описания устройства на языке VHDL необходимо выполнить следующие шаги:

3.1. В окне мастера нового проекта нажать кнопку «New Source» (рис. 6.5).

3.2. Выбрать «VHDL Module» в качестве типа исходного файла (рис. 6.6).

3.3. Ввести имя файла в поле «File name».

3.4. Убедиться, что установлена опция «Add to project».

3.5. Нажать кнопку «Next >».

3.6. В открывшемся окне установить свойства портов ввода-вывода проектируемого устройства, как показано на рис. 6.7.

3.7. Нажать кнопку «Next >». В новом окне проверить свойства создаваемого файла с описанием устройства, после чего нажать кнопку «Finish».

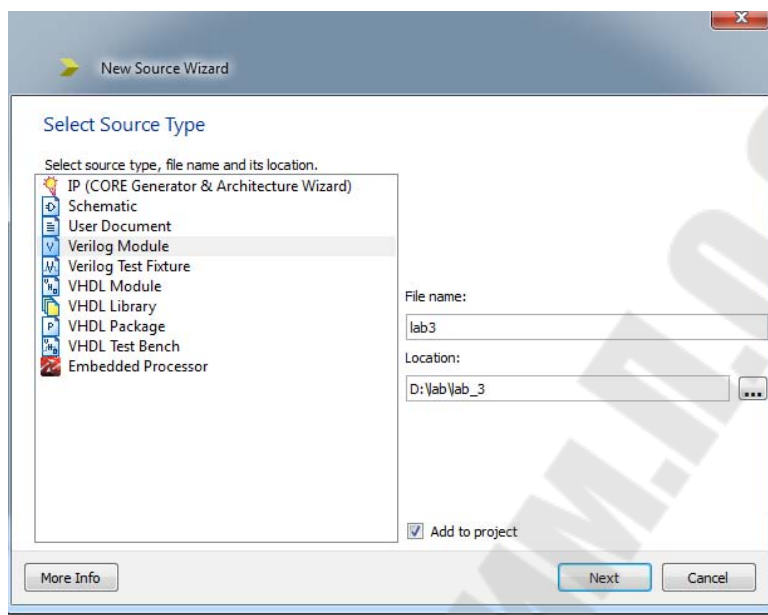


Рис. 6.6. Выбор типа исходного файла

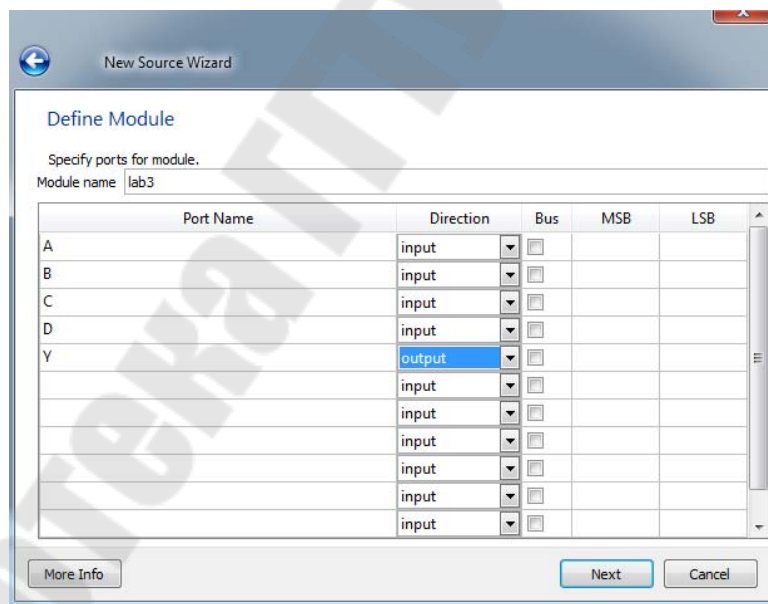


Рис. 6.7. Настройка портов ввода/вывода

3.8. В следующих двух окнах нажать кнопку «Next >». Появится окно с резюме о новом проекте (рис. 6.8). Если все свойства соответствуют требованиям, предъявляемым к проектируемому устройству, нажать кнопку «Finish».

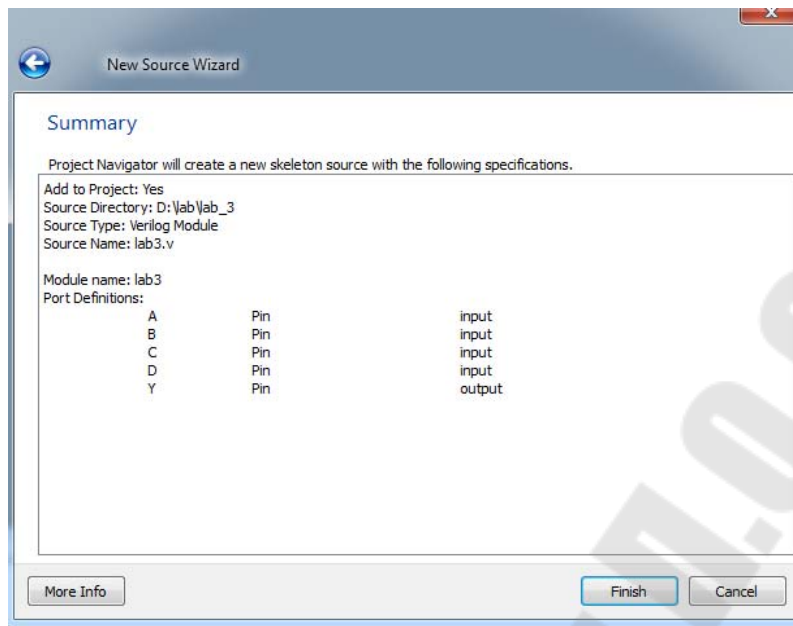


Рис. 6.8. Проверка свойств нового проекта

Следующим шагом является создание архитектуры проектируемого устройства.

Создадим устройство, которое вычисляет булево выражение: $(A \oplus \overline{B}) \wedge C \vee \overline{D}$.

Для этого необходимо выполнить следующие шаги:

1. Поместить курсор после ключевого слова `begin` в разделе `architecture`.

2. Ввести строку, которая описывает заданное булево выражение на языке VHDL:


```
process (A, B, C, D)
begin
    Y <= (A and (not B)) or (C xor D);
end process;
```

Для этого воспользуемся таблицей логических операций (табл. 6.1).

Таблица 6.1

Таблица логических операций

Условное обозначение	Выполняемая функция	Запись на языке Verilog
$X \wedge Y$	Логическое И	<code>X & Y</code>
$X \vee Y$	Логическое ИЛИ	<code>X Y</code>
$X \oplus Y$	Исключающее ИЛИ	<code>X ^ Y</code>
\overline{X}	Инверсия	<code>~X</code>

3. Сохранить файл, выбрав в меню пункт «File» → «Save» или нажав на кнопку  на инструментальной панели.

После выполнения всех действий окончательное описание устройства будет выглядеть, как показано ниже:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
---- Uncomment the following library declaration if in-
stantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
entity lab1 is
    Port ( A : in  STD_LOGIC;
          B : in  STD_LOGIC;
          C : in  STD_LOGIC;
          D : in  STD_LOGIC;
          Y : out  STD_LOGIC);
end lab1;
architecture Behavioral of lab1 is
begin
process (A,B,C,D)
begin
Y <= (A and (not B)) or (C xor D);
end process;
end Behavioral;
```

После окончания редактирования HDL-описания устройства необходимо проверить файл на наличие синтаксических ошибок. Для этого необходимо сделать ряд шагов:



1. Удостовериться, что в выпадающем меню «Sources for:» в окне описания проекта выбран режим «Synthesis/Implementation».

2. Выбрать в этом же окне файл с описанием устройства, в окне процессов будут отображены процессы, доступные для этого устройства и режима

3. Нажать символ «+» и раскрыть группу процессов «Synthesize-XST».

4. Выполнить двойной щелчок мышью по пункту «Check Syntax».

Сообщения обо всех найденных ошибках отображаются в разделе «Console» окна отчетов. Свидетельством отсутствия ошибок будет

символ  перед названием выполненного процесса. Если в результате работы процесса были обнаружены ошибки, то перед его именем появится символ  (рис. 6.9).

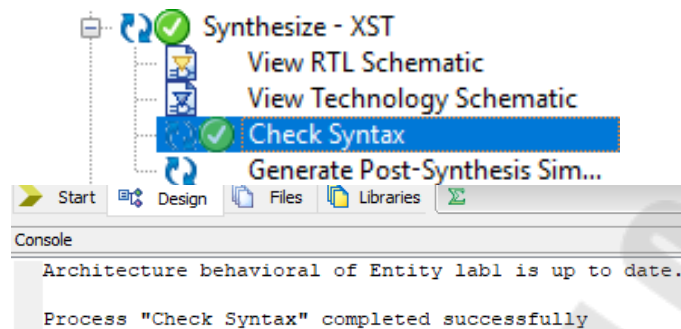


Рис. 6.9. Окно процессов

4.1. Назначение выводов с помощью утилиты PlanAhead 14.7.

Далее необходимо поставить в однозначное соответствие линии портов проектируемого устройства выводам микросхемы ПЛИС. Для этого необходимо выполним следующие шаги:

4.1.1. Выберем в главном меню операционной системы пункт «Пуск» → «Программы» → «PlanAhead (32 bit)».


4.1.2. Для начала работы выберем пункт меню «Open Project» (рис. 6.10).



Рис. 6.10. Окно редактора PlanAhead 14.7

4.1.3. С помощью проводника выбрать файл проекта «Lab1.xise» и нажать кнопку «ОК».

4.1.4. В окне «Import ISE Project As» указать имя проекта, ввести или выбрать местоположение создаваемого проекта в поле «Project Location». Убедиться, что установлена опция «Create project subdirectory». Нажать клавишу «ОК» и дождаться завершения импорта проекта.

4.1.5. В окне «Flow Navigator» из списка «Implementation» (рис. 6.11) выбрать пункт меню «Run Implementation» . По окончании выполнения процесса в окне «Implementation Completed» выбрать пункт меню «Open Implemented Design» и нажать кнопку «ОК».

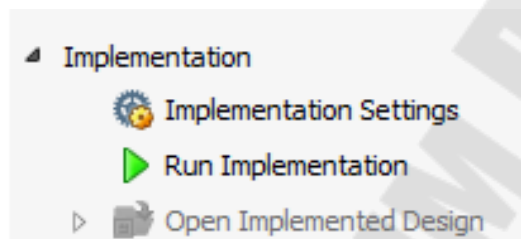


Рис. 6.11. Пункты меню «Implementation»

4.1.6. Проверить наличие схемы расположения выводов ПЛИС, а при ее отсутствии в окне программы из пункта меню «Layout» выбрать «I/O Planning» (рис. 6.12).

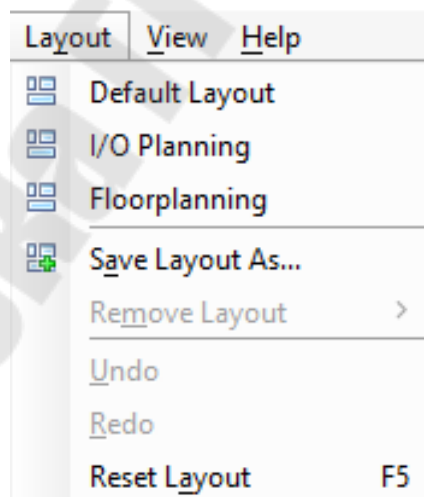


Рис. 6.12. Всплывающее меню «Layout»

4.1.7. В окне списка объектов «I/O Ports» открыть папку «Scalar Ports» (рис. 6.13), ввести местоположение линий портов в ПЛИС, как это показано ниже:

- вход А сопоставить с выводом N17;
- вход В сопоставить с выводом H18;
- вход С сопоставить с выводом L14;
- вход D сопоставить с выводом L13;
- выход Y сопоставить с выводом F12.

Схема расположения выводов должна получиться такой, как показано на рис. 6.14.

I/O Ports											
Name	Direction	Neg Diff Pair	Site	Fixed	Bank	I/O Std	Vcco	Vref	Drive Stre...	Slew Type	Pull Type
All ports (5)											
Scalar ports (5)											
A	Input		N17	<input checked="" type="checkbox"/>		1 default (LVCMOS25)	2.500				NONE
B	Input		H18	<input checked="" type="checkbox"/>		1 default (LVCMOS25)	2.500				NONE
C	Input		L14	<input checked="" type="checkbox"/>		1 default (LVCMOS25)	2.500				NONE
D	Input		L13	<input checked="" type="checkbox"/>		1 default (LVCMOS25)	2.500				NONE
Y	Output		F12	<input checked="" type="checkbox"/>		0 default (LVCMOS25)	2.500		12	SLOW	NONE

Рис. 6.13. Список объектов проекта

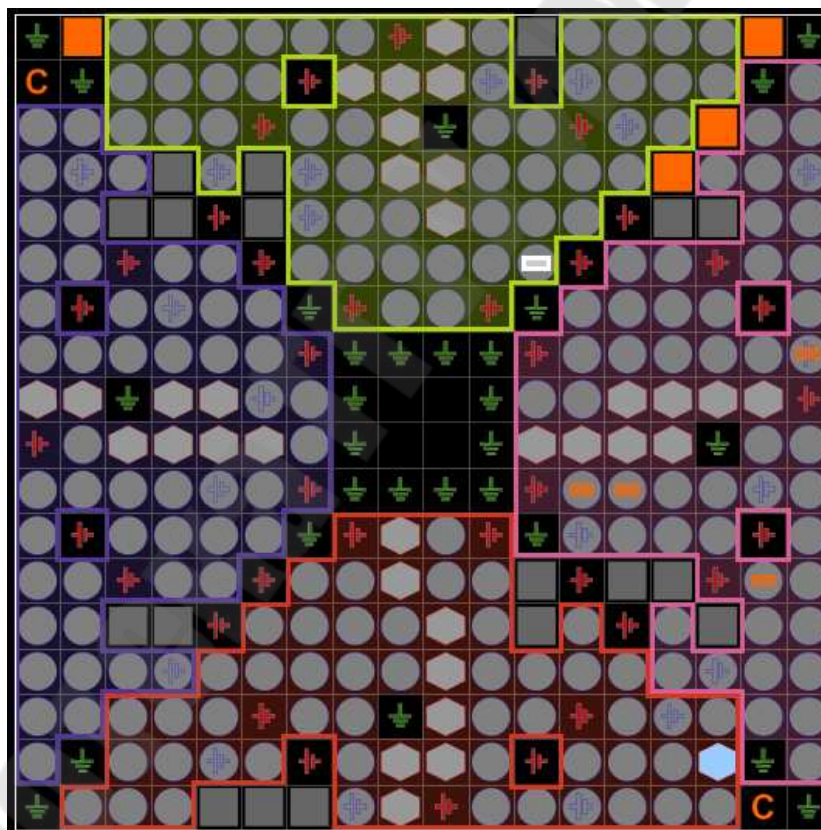


Рис. 6.14. Схема расположения выводов ПЛИС

4.1.8. Выбрать пункт меню File → Save

4.1.9. Закрывать PlanAhead 14.7.

4.2. Ручное назначение выводов.

Для установки в однозначное соответствие линии портов проектируемого устройства выводам микросхемы ПЛИС необходимо выполнить следующие действия:

4.2.1. Нажать правой кнопкой мыши по окну ресурсов проекта, в всплывающем меню выбрать пункт «New Source».

4.2.2. В окне создания ресурсов проекта (рис. 6.15) выбрать пункт «Implementation Constraints File», указать имя и ввести/указать место расположения файла, нажать клавишу «ОК».

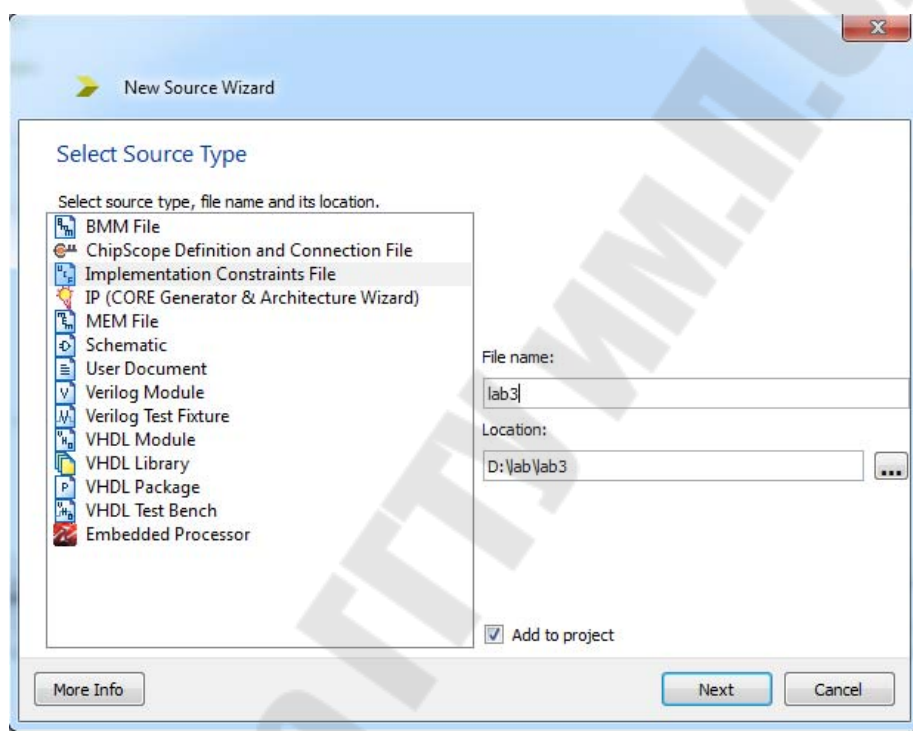


Рис. 6.15. Окно создания ресурсов проекта


4.2.3. Убедиться в наличии файла «.ucf» в окне ресурсов проекта.

4.2.4. Двойным нажатием на «.ucf» открыть окно редактора.

4.2.5. Описать установку в однозначное соответствие линии портов проектируемого устройства, как показано на рис. 6.16.

```
1 NET "A" LOC = "N17" | IOSTANDARD = LVCMOS33;  
2 NET "B" LOC = "H18" | IOSTANDARD = LVCMOS33;  
3 NET "C" LOC = "L14" | IOSTANDARD = LVCMOS33;  
4 NET "D" LOC = "L13" | IOSTANDARD = LVCMOS33;
```

Рис. 6.16. Конфигурация линии портов

4.2.6. Сохранить внесенные изменения нажав на кнопку сохранения ().

5. Размещение устройства в кристалле.

Процесс размещения устройства на кристалле включает следующие шаги:

5.1. Выбрать в окне описания проекта файл lab1.

5.2. Открыть окно резюме проекта, дважды щелкнув по процессу «View Design Summary» в окне процессов.

5.3. Запустить процесс реализации проекта, дважды щелкнув по процессу «Implement Design» в окне процессов.

Если процесс реализации проекта прошел без ошибок и предупреждений, то это будет отмечено зеленым символом перед именем процесса (рис. 6.17). Если же произошел сбой на одном из этапов работы, то этот этап можно идентифицировать по красному (признак ошибки) или желтому (признак предупреждения) символу перед именем процесса «Implement Design», а также перед именем сбойного подпроцесса.

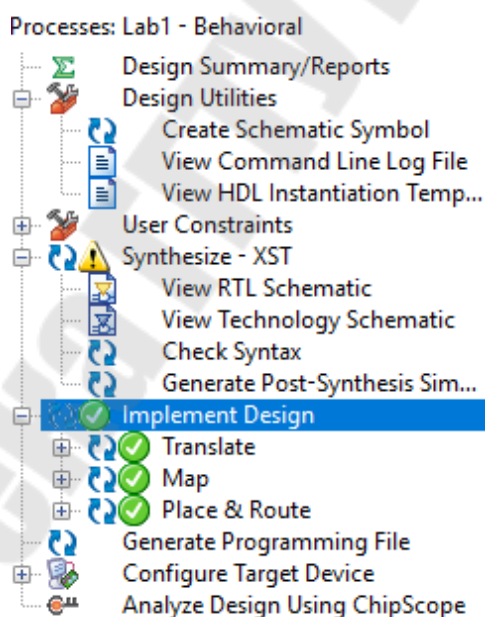


Рис. 6.17. Результат безошибочной реализации проекта

6. Загрузка конфигурации в Spartan 3.

Для загрузки конфигурации в ПЛИС необходимо проделать следующие шаги:

6.1. Выбрать в окне описания проекта файл lab1.

6.2. В окне процессов дважды щелкнуть по пункту меню «Generate Programming File», чтобы выполнить группу процессов. Дождаться завершения процедуры, убедиться в правильности работы программы (рис. 6.18).

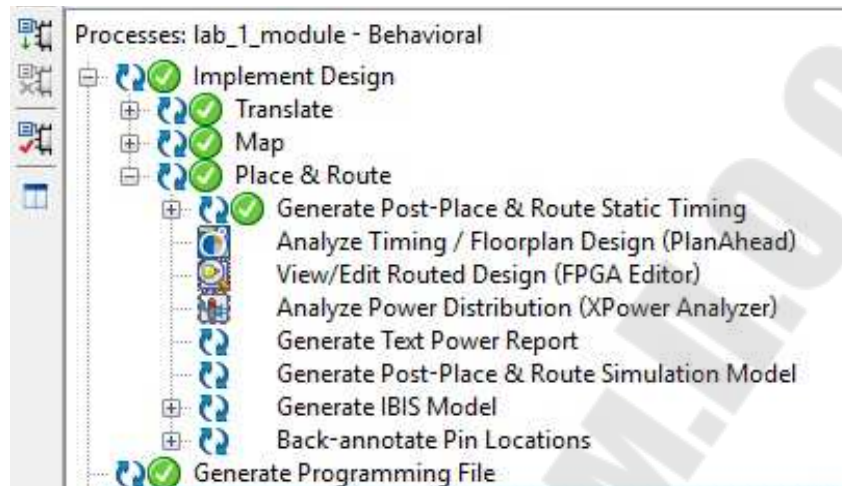


Рис. 6.18. Результат выполнения процедуры «Generate Programming File»

6.3. В окне процессов щелкнуть по символу «+», чтобы раскрыть группу процессов «Configure Target Device»

6.4. Дважды щелкнуть по процессу «Configure Target Device», или выбрать пункт всплывающего меню «Run» (по нажатию правой кнопки мыши). Будет запущена программа iMPACT и откроется окно выбора режима конфигурирования.

6.5. В открывшемся окне выбрать режим конфигурирования с использованием периферийного сканирования «Configure devices using Boundary-Scan (JTAG)».

6.6. Убедиться, что в выпадающем списке выбран пункт «Automatically connect to a cable and identify Boundary-Scan chain».

6.7. Нажать кнопку «Finish».

Если появится сообщение о том, что найдено несколько устройств, нажать «ОК» для продолжения работы. После этого будет сформирован канал периферийного сканирования в соответствии со стандартом JTAG и произойдет автоматическое определение устройства, после чего появится основное окно программы iMPACT.

6.8. В открывшемся диалоговом окне по выбору пункта «File»-> «New Project» выбрать файл битовой последовательности lab1.bit для загрузки в устройство xc3s500e, после чего нажать кнопку «Open».

Если появится предупреждение (Warning), то необходимо нажать «ОК».

6.9. Снова откроется окно выбора файла битовой последовательности. Поскольку в цепочке JTAG в нашем случае только один ПЛИС, следует нажать «Bypass», чтобы пропустить загрузку в другие устройства.

6.10. Правой кнопкой мыши щелкнуть по устройству xc3s500e и выбрать в выпадающем меню пункт «Program...». Откроется диалоговое окно определения настроек программирования устройства (Programming Properties) (рис. 6.19).

6.11. Нажать «ОК» для начала программирования. Об успешном окончании программирования будет свидетельствовать появившееся в главном окне iMPACT сообщение «Program Succeeded».

Если появится предупреждение (Warning), то нажать «ОК».

6.12. Закрывать программу iMPACT, не сохраняя проект.

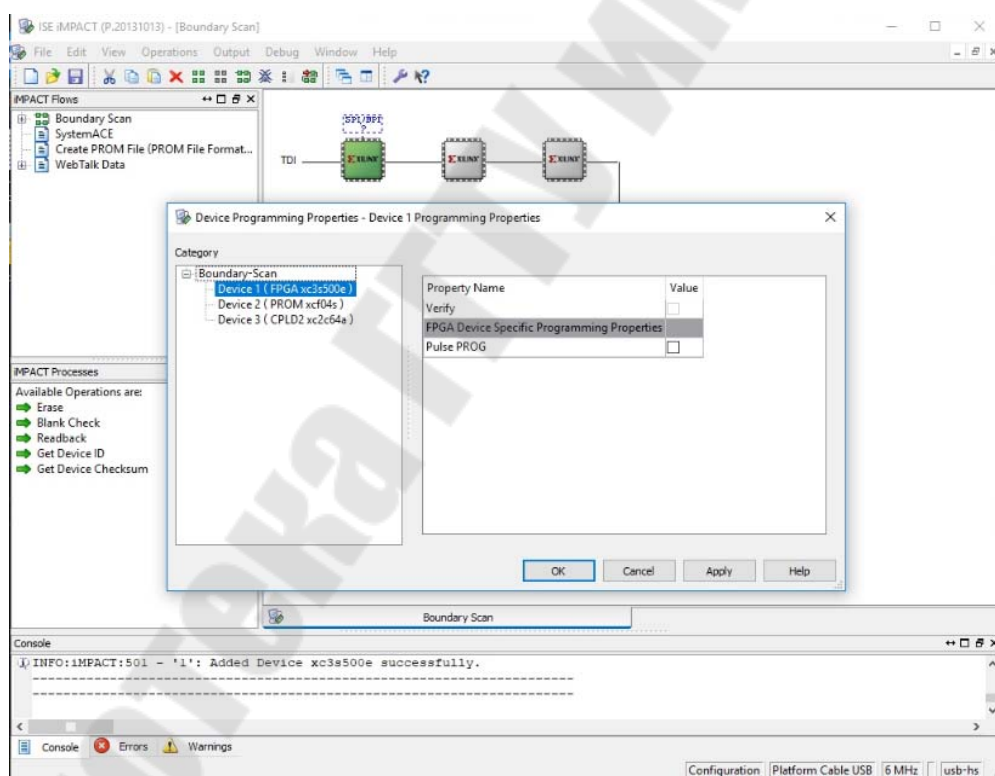


Рис. 6.19. Диалоговое окно определения настроек программирования устройства

7. Проверка правильности работы устройства.

Изменяя положения переключателей N17, N18, L14 и L13, требуется провести тестирование разработанного устройства, наблюдая за светодиодом F12. Полученные результаты занести в табл. 6.2.

Пример таблицы истинности

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

Задание для самостоятельной работы

В ходе выполнения задания для самостоятельной работы требуется:

- 1) составить таблицу истинности для булева выражения, соответствующего номеру варианта согласно табл. 6.3;
- 2) разработать описание устройства, вычисляющего заданное булево выражение;
- 3) поставить в соответствие линии портов устройства выводам микросхемы;
- 4) разместить разработанное устройство в кристалле;
- 5) загрузить полученную конфигурацию в ПЛИС;
- 6) проверить правильность работы программы.

Исходные данные

Номер варианта	Булево выражение	Номер варианта	Булево выражение
1	$(\overline{A \oplus \overline{B}}) \wedge C \vee \overline{D}$	16	$(\overline{\overline{\overline{A \vee B \wedge C}}}) \oplus \overline{D}$
2	$A \wedge C \oplus \overline{\overline{B \vee D}}$	17	$\overline{A} \wedge \overline{B} \vee \overline{C \oplus D}$

Номер варианта	Булево выражение	Номер варианта	Булево выражение
3	$B \wedge A \oplus \overline{D} \vee C$	18	$D \oplus A \oplus \overline{B} \vee C$
4	$(A \oplus \overline{D}) \vee \overline{C} \wedge \overline{D}$	19	$\overline{\overline{\overline{D} \wedge C \wedge B \wedge A}}$
5	$A \wedge \overline{B} \oplus \overline{C} \wedge D$	20	$(D \oplus B) \vee \overline{A} \wedge \overline{C}$
6	$\overline{A} \wedge B \oplus C \vee D$	21	$\overline{(A \oplus B) \wedge (D \vee C)}$
7	$\overline{(A \oplus D)} \vee (C \wedge \overline{B})$	22	$\overline{(A \wedge B \oplus C)} \vee D$
8	$D \oplus C \wedge \overline{A} \vee \overline{B}$	23	$\overline{A \oplus \overline{B} \oplus \overline{C} \wedge D}$
9	$(A \wedge \overline{C} \oplus \overline{B}) \vee D$	24	$\overline{(A \oplus \overline{B})} \vee (C \oplus D)$
10	$(B \oplus \overline{D}) \vee (C \wedge A)$	25	$\overline{(A \vee B)} \oplus (\overline{C} \wedge \overline{D})$
11	$A \wedge B \vee \overline{D} \oplus \overline{C}$	26	$\overline{B \wedge C \oplus \overline{D} \wedge A}$
12	$D \oplus \overline{C} \wedge \overline{B} \vee \overline{A}$	27	$\overline{(\overline{A} \vee C)} \oplus (\overline{B} \vee \overline{D})$
13	$(D \oplus C) \vee (\overline{\overline{A} \wedge \overline{B}})$	28	$\overline{A \oplus B \wedge C \oplus D}$
14	$\overline{(A \oplus \overline{B})} \wedge (\overline{C \oplus D})$	29	$(A \wedge \overline{C} \oplus \overline{D}) \vee B$
15	$\overline{A \oplus \overline{B} \oplus \overline{C} \wedge D}$	30	$\overline{(\overline{A \oplus \overline{B} \oplus D})} \wedge C$

Содержание отчета

Отчет должен содержать цель работы, краткие теоретические сведения по теме работы, таблицу истинности булева выражения, текст разработанной конфигурации, выводы.

Контрольные вопросы

1. Поясните, что такое ПЛИС и FPGA.
2. Раскройте понятие HDL.
3. Укажите, каким образом производится описание комбинационной логики на языке описания цифровых устройств VHDL.
4. Поясните, почему в промышленности текстовое описание цифровых схем вытеснило схемотехническое.
5. Поясните, какая часть цифровой схемы описывается в блоке «architecture Behavioral».
6. Укажите, какая часть цифровой схемы описывается в блоке «entity».

Лабораторная работа № 7

Реализация типовых функций элементов цифровой электроники на отладочном модуле Spartan-3E Starter Kit

Цель работы

Научиться описывать последовательностные устройства различными методами на основе отладочного модуля Spartan-3E Starter Kit, а также создавать тестовые модули для проверки правильности работы программ.

Основные теоретические сведения

Рассмотрим различные стили описания одного и того же поведения на примере дешифратора, который имеет два входных полюса и четыре выходных (рис. 7.1).

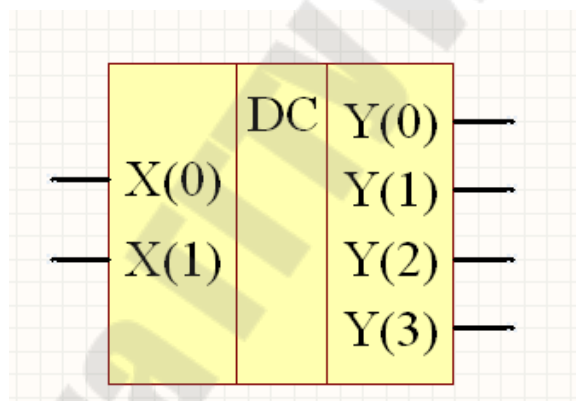


Рис. 7.1. Дешифратор

На рис. 7.1 слева и справа указаны имена входных и выходных полюсов, употребляемых в VHDL-описании. Функции дешифратора имеют следующий вид:

$$y_0 = \overline{x_0} \wedge \overline{x_1};$$

$$y_1 = \overline{x_0} \wedge x_1;$$

$$y_2 = x_0 \wedge \overline{x_1};$$

$$y_3 = x_0 \wedge x_1.$$

Интерфейс дешифратора:

```
entity Decoder is
port (X: in Bit_vector (0 to 1);
      Y: out Bit_vector (0 to 3));
end Decoder;
```

Представим различные стили описания функционирования дешифратора: структурное описание, описание в виде потока данных, процедурное описание. Заметим, что при спецификации цифровых систем допустим смешанный стиль, что весьма удобно при проектировании.

Структурное описание архитектурного тела в виде схемы в базе инверторов (Inverter) и двухвходовых элементов И (AND_Gate) имеет вид:

```
architecture structure of Decoder is
signal S: bit_vector (0 to 1);
component AND_Gate
port (A, B: in Bit; D: out Bit);
end component;
component Inverter
port (A: in Bit; B: out Bit);
end component;
begin
  Inv1: Inverter port map (A=>x(0), B=>s(0));
  Inv2: Inverter port map (A=>x(1), B=>s(1));
  A1:AND_Gate port map (A=>s(0), B=>s(1), D=>y(0));
  A2:AND_Gate port map (A=>s(0), B=>x(1), D=>y(1));
  A3:AND_Gate port map (A=>x(0), B=>s(1), D=>y(2));
  A4:AND_Gate port map (A=>x(0), B=>x(1), D=>y(3));
end structure;
```

Описание поведения дешифратора в виде потока данных (data flow):

```
architecture Data_flow of Decoder is
begin
  y(0) <= not x(0) and not x(1);
  y(1) <= not x(0) and x(1);
  y(2) <= x(0) and not x(1);
  y(3) <= x(0) and x(1);
end Data_flow;
```


Процедурное описание дешифратора выглядит следующим образом:

```
architecture Procedural of Decoder is
  signal s: bit_vector (0 to 3);
begin
  process (x)
  case x is
    when "00"=>s<="1000";
    when "01"=>s<="0100";
    when "10"=>s<="0010";
    when "11"=>s<="0001";
  end case;
  end process;
  y<=s;
end Procedural;
```

Смешанное описание, использующее элементы структурного описания и элементы описания «поток данных», приведено ниже:

```
architecture Mixed of Decoder is
  component Inverter
    port (A: in Bit; B: out Bit);
  end component;
  signal S: bit_vector (0 to 1);
begin
  Inv1: Inverter port map (A=>x(0), B=>s(0));
  Inv2: Inverter port map (A=>x(1), B=>s(1));
  process (s, x)
  begin
    y(0) <= s(0) and s(1);
    y(1) <= s(0) and x(1);
    y(2) <= x(0) and s(1);
    y(3) <= x(0) and x(1);
  end process;
end Mixed;
```

В данном примере структурное описание и описание типа «поток данных» весьма схожи, так как происходит замена логических элементов (компонентов) логическими выражениями. Однако для более сложных схем такая замена может быть совершенно неочевидной.

Порядок выполнения работы

В ходе лабораторной работы необходимо разработать программу на языке VHDL, которая описывает заданное последовательностное устройство; создать набор тестовых воздействий для проверки правильности работы проектируемого устройства; получить временные диаграммы выходных сигналов; загрузить разработанную программу в ПЛИС, а также проверить правильность ее работы. При проверке правильности работы следует задавать значения переменных с помощью переключателей, а результат выводить на светодиоды.

1. Создание нового проекта и описания устройства.

В качестве примера рассмотрим создание описания микросхемы K561ИЕ19 в виде потока данных.

Микросхема K561ИЕ19 (рис. 7.2) – пятиразрядный синхронный счетчик по схеме Джонсона. От каждого триггера счетчика сделан инверсный выход $\overline{Q0}–\overline{Q4}$. Счетчик имеет пять входов предварительной записи (установки) S0–S4, тактовый вход C, вход последовательных данных D, а также вход сброса R. Входами S0–S4 можно воспользоваться, если подать сигнал разрешения установки (высокий уровень) на вход SE.

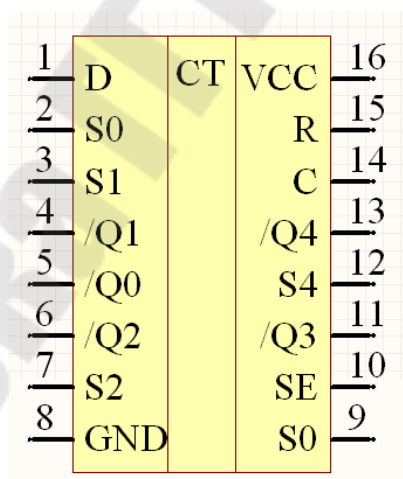


Рис. 7.2. Условно-графическое обозначение микросхемы K561ИЕ19

Создадим новый проект и присвоим ему имя «lab2».

Создадим HDL-описание устройства на языке VHDL, для этого необходимо выполнить следующие шаги:

- 1.1. Выберем «VHDL Module» в качестве типа исходного файла.
- 1.2. В поле «File name» введем «lab2».

1.3. Установим свойства портов ввода/вывода проектируемого устройства, как показано на рис. 7.3.

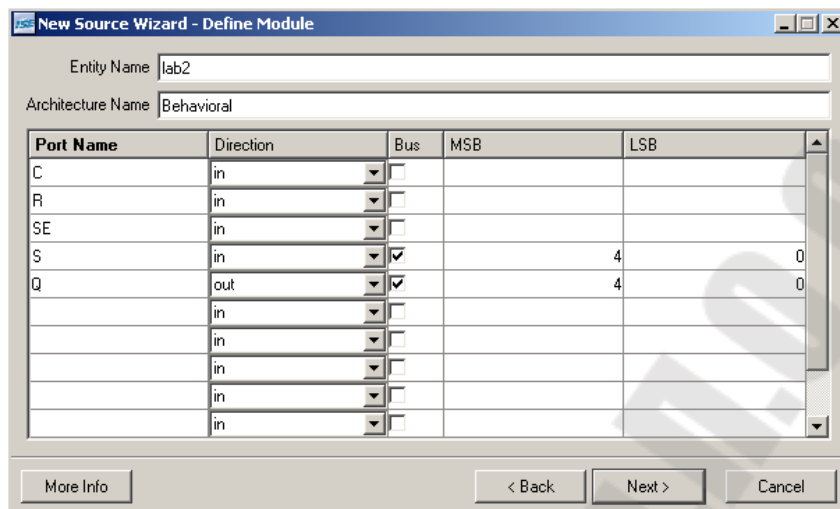


Рис. 7.3. Настройка портов ввода/вывода

Вход D используется для преобразования рассматриваемого счетчика в делитель частоты на целое число, поэтому его в данной лабораторной работе не будем рассматривать.

В результате получим файл описания устройства с определенным разделом entity и незаполненным разделом architecture.

1.4. Опишем дополнительный сигнал Q_temp, для этого перед словом begin введем следующую строку:

```
signal Q_temp : STD_LOGIC_vector(5 downto 0);
```

1.5. Введем процесс counter, который описывает работу микросхемы К561ИЕ19:

```
counter: process(C,R,SE)
begin
  if (SE = '1')then
    Q_temp(4 downto 0)<=S(4 downto 0);
  elsif ( R = '1') then
    Q_temp <= (others => '0');
  elsif ( C'event and C = '1') then
    Q_temp(4 downto 1)<=Q_temp(3 downto 0) ;
    Q_temp(0) <= not(Q_temp(5) XOR Q_temp(4));
  end if;
end process counter;
```

1.6. Присвоим выходному сигналу Q значения сигнала Q_temp, для этого после текста процесса counter введем следующую строку:

```
Q(4 downto 0) <= Q_temp(4 downto 0);
```

После выполнения всех действий окончательное описание устройства будет выглядеть, как показано ниже:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
---- Uncomment the following library declaration if in-
stantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
entity lab2 is
    Port(C : in  STD_LOGIC;
          R : in  STD_LOGIC;
          SE : in  STD_LOGIC;
          S : in  STD_LOGIC_VECTOR (4 downto 0);
          Q : out STD_LOGIC_VECTOR (4 downto 0));
end lab2;

architecture Behavioral of lab2 is
    signal Q_temp: STD_LOGIC_vector(5 downto 0);
begin
    counter: process (C,R,SE)
    begin
        if (SE = '1')then
            Q_temp(4 downto 0)<=S(4 downto 0);
        elsif ( R = '1') then
            Q_temp <= (others => '0');
        elsif ( C'event and C ='1') then
            Q_temp(4 downto 1) <= Q_temp(3 downto 0) ;
            Q_temp(0) <= not(Q_temp(5) XOR Q_temp(4));
        end if;
    end process counter;
    Q(4 downto 0) <= Q_temp(4 downto 0);
end Behavioral;
```

1.7. Сохраним файл.

1.8. Проверим его на наличие синтаксических ошибок.

2. Создание набора тестовых воздействий.

Этап функционального моделирования (Simulate Behavioral VHDL Model) позволяет выполнить предварительную верификацию

проекта. На этой стадии отсутствует информация о значениях задержек распространения сигналов, поэтому при функциональном моделировании можно обнаружить только логические и синтаксические ошибки в описании разрабатываемого устройства. Для функционального моделирования проекта применяется библиотека UniSim Library, элементы которой имеют единичные задержки.

При функциональном моделировании создается набор тестовых воздействий (Test Bench), по реакции на которые оценивается правильность работы проектируемого устройства. Для создания набора тестовых воздействий необходимо проделать следующие шаги:

2.1. Выбрать в окне описания проекта файл lab2.

2.2. Для создания нового набора тестовых воздействий выбрать пункт меню «Project» → «New Source».

2.3. В открывшемся окне выбрать тип исходного файла «Test Bench WaveForm» и ввести имя для создаваемого набора test1 в поле «File Name» (рис. 7.4).

2.4. Нажать кнопку «Next >».

2.5. В открывшемся окне привязок (Associate Source) выбрать файл с описанием устройства, к которому будет привязан набор тестовых воздействий. Поскольку в нашем проекте всего один файл, то он уже выбран для создания связи.

2.6. Нажать кнопку «Next >».

2.7. В окне резюме проверить соответствие файла с исходным описанием устройства и привязанного к нему набора тестовых воздействий, после чего нажать кнопку «Finish».

2.8. В окне инициализации системы синхронизации (Initialize Timing) следует настроить временные параметры тактового сигнала.

Зададимся следующими требованиями к разрабатываемому счетчику:

– счетчик должен работать корректно при частоте входного сигнала 50 МГц;

– значения сигналов на входах сброса R и разрешения параллельной загрузки SE должны быть установлены не позднее чем за 5 нс до установления высокого уровня на тактовом входе C;

– значение выходных сигналов на шине Q должно быть установлено не позднее чем через 5 нс после установления высокого уровня на тактовом входе C.

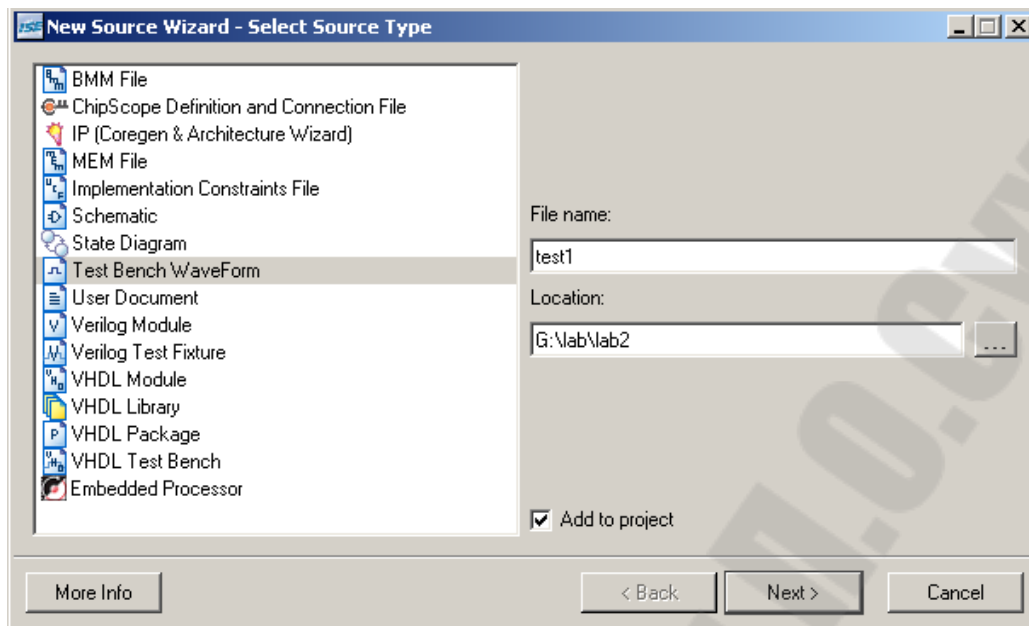


Рис. 7.4. Выбор типа исходного файла

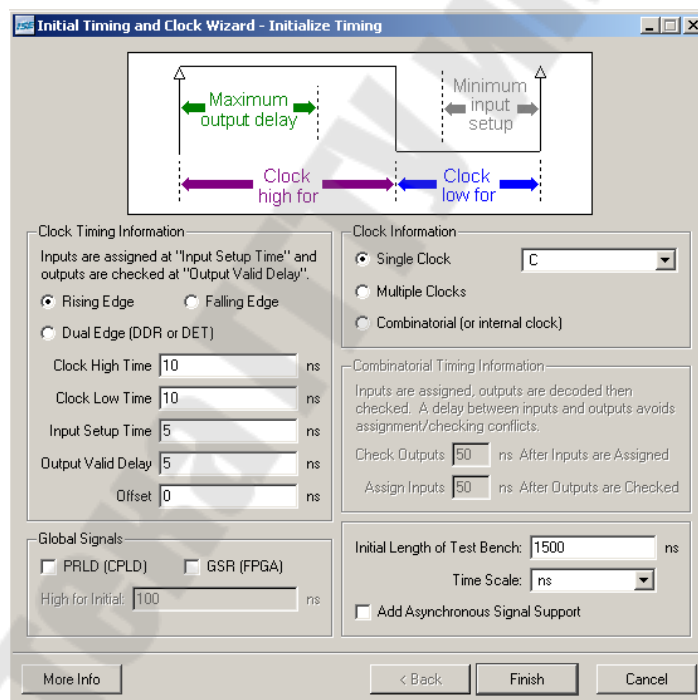


Рис. 7.5. Настройка системы синхронизации

Для реализации этих требований необходимо заполнить поля настройки системы синхронизации следующим образом (рис. 7.5):

- длительность высокого уровня тактового сигнала (Clock Time High): 10 нс;
- длительность низкого уровня тактового сигнала (Clock Time Low): 10 нс;

- задержка до установления входного сигнала (Input Setup Time): 5 нс;
- задержка до установления выходного сигнала (Output Valid Delay): 5 нс;
- сдвиг (Offset): 0 нс;
- глобальный сигнал синхронизации (Global Signal): GSR (FPGA);
- длительность моделирования (Initial Length of Test Bench): 1500 нс.

Значения остальных параметров оставить без изменений.

2.9. Нажать кнопку «Finish» для окончания процесса настройки системы синхронизации.

На временной диаграмме (рис. 7.6) голубым цветом отмечены зоны, равные по длительности заданному времени установления значения входного сигнала. Смоделируем параллельную загрузку данных:

- для создания временных диаграмм изменения сигнала на каждой из линий шины требуется щелкнуть по символу «+» перед именем шины (в нашем случае S);
- загрузим состояние «00011», для этого щелкнем мышью по голубой зоне в районе отметки 530 нс для установки высокого уровня сигнала S(3);
- щелкнем мышью по голубой зоне в районе отметки 670 нс для установки низкого уровня сигнала S(3). Таким образом получим импульс на линии S(3);
- проделаем аналогичные операции для линии S(4);
- на линии разрешения параллельной загрузки SE сформируем импульс в том же временном диапазоне.

Для моделирования работы входа сброса R сформируем импульс вида $0 \rightarrow 1 \rightarrow 0$ в районе отметки 110 нс и установим высокий уровень в районе отметки 1090 нс.

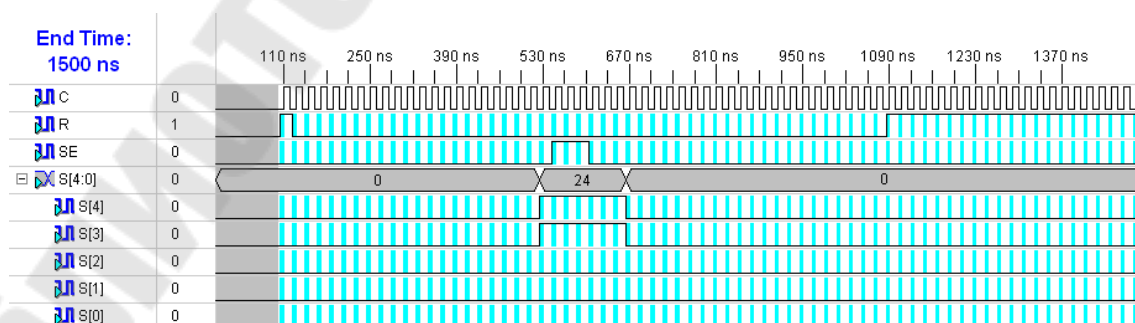


Рис. 7.6. Временная диаграмма набора тестовых воздействий

Временная диаграмма (Waveform) с заданными параметрами входных сигналов должна выглядеть, как показано на рис. 7.6. Серая зона в начале диаграммы соответствует параметру «Сдвиг».

При установке глобального сигнала синхронизации к значению, заданному в поле «Offset», автоматически добавляется 100 нс, что и отражено на диаграмме.

2.10. Сохранить временную диаграмму.

2.11. В выпадающем меню «Sources for:» в окне описания проекта выбрать режим «Behavioral Simulation» (рис. 7.7).

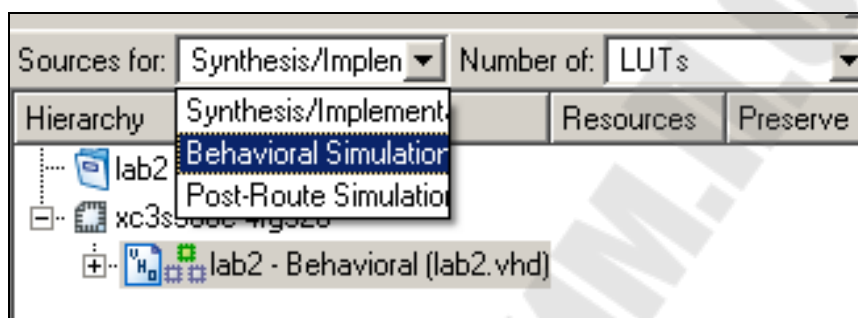


Рис. 7.7. Выбор режима функционального моделирования

3. Создание тестового модуля.

Создание набора ожидаемых значений выходных сигналов завершает процесс создания набора тестовых воздействий. Эта операция преобразует набор тестовых воздействий в полноценный тестовый модуль. Целью создания такого модуля является получение возможности сравнения ожидаемых значений выходных сигналов со значениями, полученными в результате моделирования проектируемого устройства. По результатам такого моделирования делается заключение о правильности функционирования реализованного алгоритма.

Для создания тестового модуля необходимо выполнить следующие шаги:

3.1. Убедиться, что выбран режим «Behavioral Simulation» в выпадающем меню «Sources for:» в окне описания проекта.

3.2. В окне описания проекта выбрать файл с тестовым модулем test1.

3.3. В окне процессов развернуть группу «Xilinx ISE Simulator» и выполнить двойной щелчок по процессу «Generate Expected Simulation Results». Запустится процесс моделирования работы проектируемого устройства.

3.4. В ответ на запрос диалогового окна «Expected Results» ответить «Yes», и будет выведена временная диаграмма с рассчитанными значениями выходных сигналов (рис. 7.8).

3.5. Для шины существует возможность просмотра временных диаграмм изменения сигнала на каждой из линий шины. Для этого следует щелкнуть по символу «+» перед именем шины (в нашем случае Q).

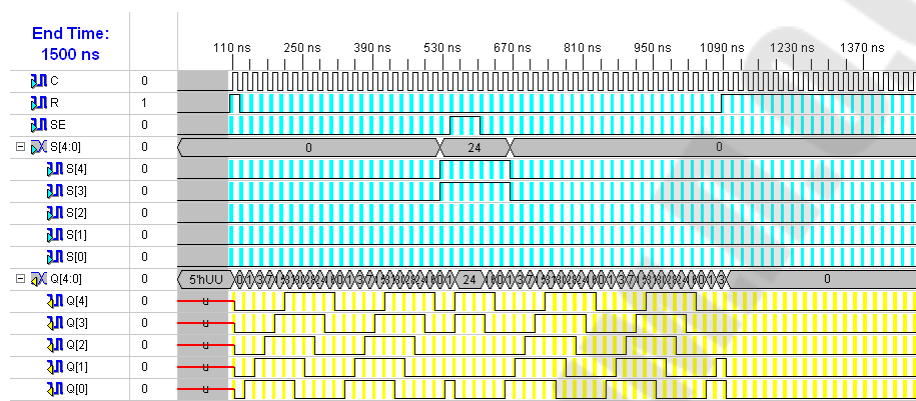


Рис. 7.8. Тестовый модуль

3.6. Начиная с версии ISE 11.1, Xilinx больше не поддерживает Test Bench Waveform Editor. Когда проект, в котором есть test bench waveform (TBW), обновляется до версии 11.1, то TBW будет автоматически преобразован в HDL test bench и добавлен в проект. Xilinx рекомендует для новых проектов использовать тесты (HDL test benches), написанные на основе HDL.

Представим пример такого HDL описания тестового воздействия:

```

-----
-----
-- Company:
-- Engineer:
--
-- Create Date:    23:03:47 09/25/2018
-- Design Name:
-- Module Name:    E:/DSPUSER/fpga /xilinx_project/LAB_2_GSTU/Test.vhd
-- Project Name:   LAB_2_GSTU
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: lab2
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created

```

```

-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic
and
-- std_logic_vector for the ports of the unit under test.  Xilinx recom-
mends
-- that these types always be used for the top-level I/O of a design in or-
der
-- to guarantee that the testbench will bind correctly to the post-
implementation
-- simulation model.
-----
-----
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY Test IS
END Test;

ARCHITECTURE behavior OF Test IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT lab2
    PORT(
        C : IN  std_logic;
        R : IN  std_logic;
        SE : IN  std_logic;
        S : IN  std_logic_vector(4 downto 0);
        Q : OUT std_logic_vector(4 downto 0)
    );
    END COMPONENT;

    --Inputs
    signal C : std_logic := '0';
    signal R : std_logic := '0';
    signal SE : std_logic := '0';
    signal S : std_logic_vector(4 downto 0) := "00101"; --(others => '0');

    --Outputs
    signal Q : std_logic_vector(4 downto 0) ;
    -- No clocks detected in port list. Replace <clock> below with
    -- appropriate port name

    -- constant <clock>_period : time := 10 ns;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: lab2 PORT MAP (
        C => C,
        R => R,
        SE => SE,
        S => S,

```

```

        Q => Q
    );

    -- Clock process definitions
    -- <clock>_process :process
    -- begin
    --     <clock> <= '0';
    --     wait for <clock>_period/2;
    --     <clock> <= '1';
    --     wait for <clock>_period/2;
    -- end process;

    -- Stimulus process
    stim_proc: process
    begin
        wait for 100 ns;
        C <= not C;
    end process;

    stim_proc_SE: process
    begin
        wait for 150 ns;
        SE <= '1';
        wait for 40 ns;
        SE <= '0';

        wait;
    end process;

    stim_proc_R: process
    begin
        wait for 0 ns;
        R <= '1';
        wait for 10 ns;
        R <= '0';

        wait;
    end process;

END;
```

На рис. 7.9 показан способ перехода в режим симуляции. Для создания файла тестовых воздействий Test1.vhd необходимо правой кнопкой мыши вызвать выпадающее меню, выбрать «New source...» -> «VHDL test bench». Указать в качестве названия файла Test1.vhd и ввести в него вышеприведенное описание тестовых воздействий.

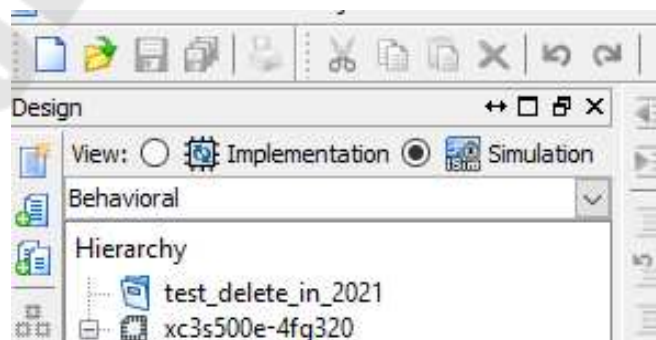


Рис. 7.9. Режим симуляции

4. Функциональное моделирование.

Получить информацию о работе устройства можно при помощи функционального моделирования. Этот процесс похож на процесс создания тестового модуля, но с той лишь разницей, что значения выходных сигналов не задаются, а только рассчитываются. Процесс функционального моделирования включает следующие шаги:

4.1. Убедиться, что выбран режим «Simulation» в выпадающем меню «Sources for:» в окне описания проекта и указан файл с тестовым модулем test1.

4.2. В окне процессов развернуть группу «Xilinx ISE Simulator» и выполнить двойной щелчок по процессу «Simulate Behavioral Model».

Результаты моделирования выводятся в окне «Simulation» и доступны только для просмотра или печати (рис. 7.10).

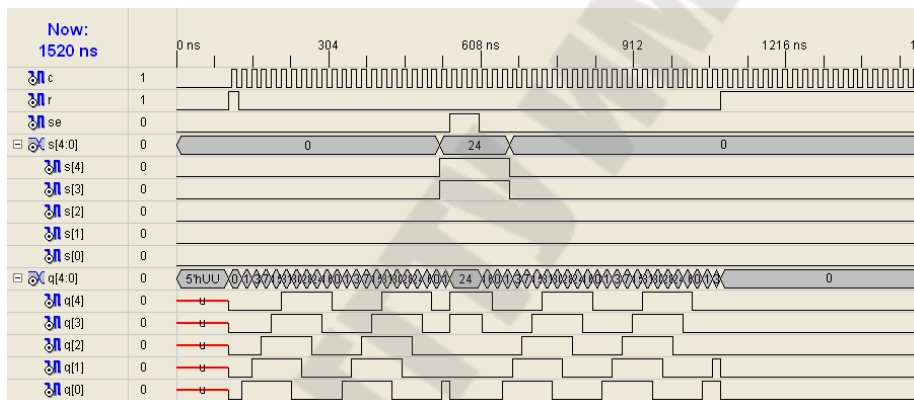


Рис. 7.10. Результаты функционального моделирования

5. Проверка правильности работы устройства.

Для проверки правильности работы устройства внесем некоторые изменения в текст программы:

5.1. Из-за нехватки переключателей при разрешении параллельной загрузки данных всегда будем загружать «11000», для этого удалим строку, описывающую вектор входных переменных S:

```
entity lab2 is
  Port ( C : in  STD_LOGIC;
        R : in  STD_LOGIC;
        SE : in  STD_LOGIC;
        Q : out  STD_LOGIC_VECTOR (4 downto 0));
end lab2;
```

и добавим сигнал S:

```
signal S : STD_LOGIC_vector(4 downto 0) := B'11000';
```

5.3. Для того чтобы можно было видеть изменения выходного сигнала, выводимого на светодиоды, добавим процесс CLK, с помощью которого происходит деление частоты C, равной 50 МГц, на 25000000 с целью получения частоты C_new, равной 2 Гц:

```
CLK: process(c)
begin
  if c'event and c='1' then-
    --divide 50MHz by 25000000 to form 2Hz pulses
    if int_count=25000000 then
      int_count <= 0;
      C_new <= '1';
    else
      int_count <= int_count + 1;
      C_new <= '0';
    end if;
  end if;
end process CLK;
```

Затем введем два дополнительных сигнала int_count и C_new:

```
signal int_count : integer range 0 to 25000000 :=0;
signal C_new : std_logic;
```

Кроме того, в процессе counter изменим название сигнала C на C_new.

После выполнения всех действий окончательное описание устройства будет выглядеть, как показано ниже:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
---- Uncomment the following library declaration if in-
stantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
entity lab2 is
  Port ( C : in STD_LOGIC;
         R : in STD_LOGIC;
```

```

        SE : in  STD_LOGIC;
        Q  : out STD_LOGIC_VECTOR (4 downto 0));
end lab2;
architecture Behavioral of lab2 is
signal Q_temp      : STD_LOGIC_vector(5 downto 0);
signal S:STD_LOGIC_vector(4 downto 0) :=B ''11000'';
signal int_count   : integer range 0 to 25000000 :=0;
signal C_new       : std_logic;
begin
CLK: process(c)
begin
    if c'event and c='1' then-
--divide 50MHz by 25000000 to form 2Hz pulses
        if int_count=25000000 then
            int_count <= 0;
            C_new <= '1';
        else
            int_count <= int_count + 1;
            C_new <= '0';
        end if;
    end if;
end process CLK;
counter: process(C_new,R,SE)
begin
if (SE = '1')then
    Q_temp(4 downto 0)<=S(4 downto 0);
elsif ( R = '1') then
    Q_temp <= (others => '0');
    elsif ( C_new'event and C_new ='1') then
        Q_temp(4 downto 1) <= Q_temp(3 downto 0) ;
        Q_temp(0) <= not(Q_temp(5) XOR Q_temp(4));
    end if;
end process counter;
Q(4 downto 0) <= Q_temp(4 downto 0);
end Behavioral;

```

6. Назначение выводов и загрузка конфигурации в Spartan 3.

Далее необходимо поставить в однозначное соответствие линии портов проектируемого устройства выводам микросхемы ПЛИС.

6.1. В редакторе PACE в окне списка объектов проекта («Design Object List») ввести в поле «Loc» местоположение линий портов в ПЛИС, как это показано ниже:

- вход С сопоставить с выводом С9;
- выход Q<0> сопоставить с выводом F12;

- выход Q<1> сопоставить с выводом E12;
- выход Q<2> сопоставить с выводом E11;
- выход Q<3> сопоставить с выводом F11;
- выход Q<4> сопоставить с выводом C11;
- вход R сопоставить с выводом № 17;
- вход SE сопоставить с выводом № 18.

6.2. Запустить процесс реализации проекта.

6.3. Удостовериться, что все выводы микросхемы правильно поставлены в соответствие портам проектируемого устройства.

6.4. Запустить программу iMPACT.

6.5. Загрузить в устройство xc3s500e файл битовой последовательности lab2.bit.

Результатом правильной работы устройства будет:

а) при установленных в «0» переключателях № 17 и № 18: последовательное включение и выключение светодиодов, соответствующее временным диаграммам, приведенным на рис. 7.9;

б) при установленном в «1» переключателе № 17: все светодиоды погашены;

в) при установленном в «1» переключателе № 18: установка светодиодов в положение «11000».

Задание для самостоятельной работы

В ходе выполнения задания для самостоятельной работы требуется:

- 1) разработать описание устройства, соответствующего номеру варианта, и поставить в соответствие линии портов устройства выводам микросхемы в соответствии с табл. 7.1;
- 2) создать набор тестовых воздействий;
- 3) создать тестовый модуль;
- 4) провести функциональное моделирование;
- 5) разместить разработанное устройство в кристалле;
- 6) загрузить полученную конфигурацию в ПЛИС;
- 7) проверить правильность работы программы.

Варианты заданий для самостоятельной работы

№	УГО микросхемы	Примечания
1		К155ТМ8 – четыре D-триггера, имеющие общие входы синхронного сброса /R и тактового запуска С, а также имеются выходы Q и /Q. Информацию от параллельных входов данных D1–D4 можно загрузить, если на вход /R подать напряжение высокого уровня.
2		К555ИР8 – 8-разрядный сдвиговый регистр с последовательным входом и параллельными выходами. Регистр имеет асинхронный сброс (вход /R) и два входа для последовательных данных DSa и DSb (логика И). Поданные через эти входы данные сдвигаются на одну позицию вправо согласно каждому положительному перепаду импульса, пришедшего на тактовый вход С
3		К155ИЕ4 – 4-разрядный двоичный счетчик-делитель на 2, на 6 и на 12. Чтобы построить счетчик с модулем деления 12, требуется соединить делители на 2 и на 6, замкнув выводы 12 и 1. На вход /C0 дается входная частота f, на выходе Q3 получается последовательность прямоугольных импульсов с частотой f/12. Тактовые запускающие перепады – отрицательные
4		К155ТМ2 – два независимых D триггера. У каждого триггера есть входы D, /S и /R, а также комплементарные выходы Q и /Q. Входы /S и /R – асинхронные. Сигнал от входа D передается на выходы Q и /Q по положительному перепаду импульса на тактовом входе С

№	УГО микросхемы	Примечания
5		<p>К561ИР2 – два независимых 4-разрядных регистра сдвига. Данные в регистр вводятся через последовательный вход D. Регистр имеет вход тактовых импульсов С, причем данные принимаются от входа D первого триггера и сдвигаются на один такт вправо после каждого положительного тактового перепада на входе С. Сброс в нуль данных на выходе Q регистра получится, если на вход асинхронного сброса R подать напряжение высокого логического уровня</p>
6		<p>К531ИЕ14 – асинхронный счетчик пульсаций. Состояния счетчика меняются по отрицательному перепаду тактового импульса. Двоично-десятичную выходную последовательность можно получить, если тактовые импульсы подать на вход /C0 и соединить выводы 5 и 6. Сигналом R='0' запрещается работа всем входам счетчика, а на всех выходах появляется напряжение низкого уровня. Когда на вход разрешения параллельной загрузки /PE подано напряжение низкого уровня, действие тактовых входов запрещается. Данные, присутствующие на входах D0–D3, загружаются параллельно в триггеры счетчика</p>
7		<p>К531ТВ11 – два JK-триггера. Оба триггера микросхемы имеют по две общие цепи управления: тактовый вход /С и вход сброса /R. Входы J и K могут работать, если на асинхронных входах /S и /R присутствуют напряжения высокого уровня</p>

№	УГО микросхемы	Примечания
8		<p>K555IP16 – 4-разрядный сдвиговый регистр. Если на входе /PE присутствует напряжение высокого уровня, данные загружаются в регистр от параллельных входов D0–D3 синхронно с отрицательным перепадом на тактовом входе /C. Напряжение низкого уровня на входе /PE вызывает загрузку данных от последовательного входа SI. Цифровое слово сдвигается вправо от Q0 к Q1 далее к Q2 и Q3 синхронно с каждым отрицательным перепадом на тактовом входе /C</p>
9		<p>K155IE5 – 4-разрядный асинхронный счетчик пульсаций. Счетчик имеет две части: делитель на 2 (выход Q0; тактовый вход /C0) и делитель на восемь (выходы Q1–Q3; тактовый вход /C1). Входы синхронного сброса R1 и R2 (логика И) запрещают действие импульсов по обоим тактовым входам. Импульс, поданный на вход R, дает сброс данных по всем триггерам одновременно</p>

Содержание отчета

Отчет должен содержать цель работы, краткие теоретические сведения по теме работы, условное графическое изображение исследуемой микросхемы и краткое описание ее работы, текст разработанной программы, изображение тестового модуля, результат функционального моделирования, выводы.

Контрольные вопросы

1. Дайте определение понятию «последовательностная схема».
2. Укажите, каким образом можно описать шифратор при помощи оператора case.
3. Поясните, что значит запись «Q_temp(4 downto 1) <= Q_temp(3 downto 0)».
4. Укажите, каким образом можно понизить частоту встроенного генератора с 50 МГц до 5 Гц.
5. Поясните, каким образом описывается регистр на языке VHDL.

Литература

1. Айфичер, Э. Цифровая обработка сигналов: практический подход / Э. Айфичер, Б. Джервис. – 2-е изд. – М. : Вильямс, 2004. – 992 с.
2. Поляков, А. К. Языки VHDL и VERILOG в проектировании цифровой аппаратуры / А. К. Поляков. – М. : СОЛОН-Пресс, 2003. – 320 с. : ил.
3. Лайонс, Р. Цифровая обработка сигналов / Р. Лайонс. – 2-е изд. – М. : Бином-Пресс, 2006. – 652 с.
4. Калабеков, Б. А. Микропроцессоры и их применение в системах передачи и обработки сигналов : учеб. пособие для вузов / Б. А. Калабеков. – М. : Радио и связь, 1988. – 368 с.
5. Рабинер, Л. Р. Теория и применение цифровой обработки сигналов / Л. Р. Рабинер, Б. Голд ; пер. с англ. ; под ред. Ю. Н. Александрова. – М. : Мир, 1978. – 637 с.
6. Шевкопляс, Б. В. Микропроцессорные структуры. Инженерные решения : справочник / Б. В. Шевкопляс. – 2-е изд., перераб. и доп. – М. : Радио и связь, 1990. – 512 с.
7. Сергиенко, А. Б. Цифровая обработка сигналов : учеб. пособие для вузов / А. Б. Сергиенко. – СПб. : Питер, 2007. – 603 с.
8. Бибило, П. Н. Основы языка VHDL : учеб. пособие / П. Н. Бибило. – 5-е изд. – М. : ЛИБРОКОМ, 2012. – 328 с.
9. Яне, Б. Цифровая обработка изображений / Б. Яне. – М. : Техносфера, 2007. – 584 с.
10. Гутников, В. С. Фильтрация измерительных сигналов / В. С. Гутников. – Л. : Энергоатомиздат, 1990. – 190 с.
11. Шостак, А. С. Прием и обработка сигналов : курс лекций / А. С. Шостак. – Томск : Том. гос. ун-т систем упр. и радиоэлектроники, 2012. – Ч. 2. – 87 с.
12. Щетинин, Ю. И. Анализ и обработка сигналов в среде MATLAB : учеб. пособие / Ю. И. Щетинин. – Новосибирск : НГТУ, 2011. – 115 с.
13. Оппенгейм, А. Цифровая обработка сигналов / А. Оппенгейм, Р. Шафер ; пер. С. Ф. Боева. – 3-е изд., испр. – М. : Техносфера, 2012. – 1048 с. – (Мир радиоэлектроники).
14. Meyer-Baese, U. Digital Signal Processing with Field Programmable Arrays / U. Meyer-Baese (3th ed.). – Florida State University, College of Engineering : Springer, 2016. – 788 p.

15. Стешенко, В. Б. ПЛИС фирмы Altera: проектирование устройств обработки сигналов / В. Б. Стешенко. – М. : ДОДЭКА, 2000. – 128 с.
16. Бродин, В. Б. Системы на микроконтроллерах и БИС программируемой логики / В. Б. Бродин, А. В. Калинин. – М. : ЭКОМ, 2002. – 400 с.
17. Потехин, Д. С. Разработка систем цифровой обработки сигналов на базе ПЛИС / Д. С. Потехин, И. Е. Тарасов. – М. : Горячая линия – Телеком, 2007. – 248 с.
18. Солонина, А. И. Алгоритмы и процессоры цифровой обработки сигналов / А. И. Солонина, Д. А. Улахович, Л. А. Яковлев. – СПб. : БХВ-Петербург, 2001. – 464 с.
19. Вальпа, О. Разработка устройств на основе цифровых сигнальных процессоров фирмы Analog Devices с использованием Visual DSP++ / О. Вальпа. – М. : Горячая линия – Телеком, 2007. – 446 с.
20. Введение в цифровую фильтрацию / под ред. Р. Богнера, А. Константинодиса. – пер. с англ., под ред Л. И. Филлипова – М. : Мир, 1976. – 216 с.
21. Марпл, С. Л. Цифровой спектральный анализ и его приложения / С. Л. Марпл. – М. : Мир, 1990. – 236 с.
22. Сато, Ю. Обработка сигналов. Первое знакомство / Ю. Сато. – М. : ДОДЭКА, 2003. – 174 с.
23. Техническая документация. ADSP-2181 DSP Microcomputer. Data Sheet. – Rev. В. – Analog Devices Inc.
24. Техническая документация. ADSP-2100 Family User's Manual. – Edition 3. – Analog Devices Inc.
25. Техническая документация. TMS320F28x Family User's Manual. – Edition 1. – Texas Instruments Inc. – 9 pt.
26. Шпак, Ю. А. Программирование на языке С для AVR и PIC микроконтроллеров / Ю. А. Шпак. – К. : МК-Пресс ; СПб. : КОРОНА-ВЕК, 2011. – 544 с.
27. Применение цифровой обработки сигналов / под ред. Э. Опенгейма. – М. : Мир. – 1980. – 552 с.
28. Марков. С. Цифровые сигнальные процессоры / С. Марков. – М. : Микроарт. – 1996. – Кн. 1. – 144 с.
29. Цифровые сигнальные процессоры фирмы Zilog и их применение // CHIPNEWS. – 1997. – № 2 (11).
30. Блаттер, К. Вейвлет-анализ. Основы теории / К. Блаттер. – М. : Техносфера, 2006. – 279 с.

31. Цифровые процессоры обработки сигналов : справочник / А. Г. Остапенко [и др.] ; под ред. А. Г. Остапенко. – М. : Радио и связь, 1994. – 264 с.

32. Гонсалес, Р. Цифровая обработка изображений : пер. с англ. / Р. Гонсалес, Р. Вудс. – М. : Техносфера. – 2006. – 1072 с.

33. Liptak, B. G. Process Control and Optimization / B. G. Liptak. – 4th ed. // Instrument Engineers' Handbook 2 / B. G. Liptak. – CRC Press, 2006. – 2368 p.

Содержание

Введение.....	3
<i>Лабораторная работа № 1</i>	
Изучение отладочного модуля для цифрового сигнального процессора семейства C28х.....	4
<i>Лабораторная работа № 2</i>	
Исследование систем управления и ввода/вывода цифрового сигнального процессора семейства C28х.....	12
<i>Лабораторная работа № 3</i>	
Формирование функций времени на основе сигнального процессора семейства C28х. Модуль расширения прерываний.....	29
<i>Лабораторная работа № 4</i>	
Формирование сигналов широтно-импульсной модуляции на основе сигнального процессора семейства C28х.....	47
<i>Лабораторная работа № 5</i>	
Ввод и масштабирование аналоговых сигналов в процессорах семейства C28х.....	65
<i>Лабораторная работа № 6</i>	
Изучение отладочного модуля для ПЛИС Spartan-3E Starter Kit. Реализация логических функций.....	77
<i>Лабораторная работа № 7</i>	
Реализация типовых функций элементов цифровой электроники на отладочном модуле Spartan-3E Starter Kit.....	95
Литература.....	115

Учебное электронное издание комбинированного распространения

Учебное издание

ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА ОБРАБОТКИ СИГНАЛОВ

**Практикум
по выполнению лабораторных работ
по одноименной дисциплине для студентов
специальности 1-53 01 07 «Информационные
технологии и управление в технических системах»
дневной формы обучения**

Составители: **Крышнёв Юрий Викторович
Хананов Валентин Андреевич**

Электронный аналог печатного издания

Редактор *Т. Н. Мисюрова*
Компьютерная верстка *Н. Б. Козловская*

Подписано в печать .21.

Формат 60x84/16. Бумага офсетная. Гарнитура «Таймс».

Ризография. Усл. печ. л. 6,97. Уч.-изд. л. 7,68.

Изд. № 4.

<http://www.gstu.by>

Издатель и полиграфическое исполнение
Гомельский государственный
технический университет имени П. О. Сухого.
Свидетельство о гос. регистрации в качестве издателя
печатных изданий за № 1/273 от 04.04.2014 г.
пр. Октября, 48, 246746, г. Гомель