



Министерство образования Республики Беларусь

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

Кафедра «Промышленная электроника»

АППАРАТУРА ЦИФРОВОЙ ОБРАБОТКИ СИГНАЛОВ

ПРАКТИКУМ

**по выполнению лабораторных работ
для студентов специальности 1-36 04 02
«Промышленная электроника»
дневной и заочной форм обучения**

Гомель 2022

УДК 004.383.3:004.31(075.8)
ББК 32.859я73
А76

*Рекомендовано 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. «Информационные технологии» ГГТУ им. П. О. Сухого
канд. экон. наук, доц. *К. С Курочка*

Аппаратура цифровой обработки сигналов : практикум по выполнению лаборатор. работ для студентов специальности 1-36 04 02 «Промышленная электроника» днев. и заоч. форм обучения / сост.: Ю. В. Крышнёв, В. А. Хананов. – Гомель : ГГТУ им. П. О. Сухого, 2022. – 203 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.

Содержит 12 лабораторных работ с порядком их выполнения, основными теоретическими сведениями, заданиями для самостоятельной работы и контрольными вопросами.

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

**УДК 004.383.3:004.31(075.8)
ББК 32.859я73**

© Учреждение образования «Гомельский государственный технический университет имени П. О. Сухого», 2022

Введение

Приобретение навыков проектирования, анализа и отладки систем, реализованных на основе современной аппаратуры цифровой обработки сигналов (АЦОС) является одной из ключевых задач в подготовке инженеров по радиоэлектронике.

Важнейшим аспектом в современной инженерной деятельности является умение пользоваться отладочными модулями с соответствующим программным обеспечением, которые принципиально необходимы при проектировании преобразователей информации, цифровых фильтров, устройств обнаружения объектов, контроля сложных технологических процессов и т.п.

В практикуме для выполнения лабораторных работ рассмотрена структура и программное обеспечение отладочного оборудования для программируемых логических интегральных схем (ПЛИС) и цифровых сигнальных процессоров (ЦСП).

Лабораторная работа № 1

Реализация комбинационных цифровых устройств на основе отладочной платы Spartan-3E Starter Kit

Цель работы

Ознакомиться с аппаратной и программной частью лабораторного оборудования, научиться создавать и отлаживать простейшие программы для отладочной платы Spartan-3E Starter Kit.

Основные теоретические сведения

1. Преимущества использования языков HDL

Проектирование на языке HDL имеет много преимуществ по сравнению с традиционным проектом на основе схемного описания:

Проекты могут быть выполнены на очень абстрактном уровне при помощи HDL. Проектировщики могут выполнить описание на уровне RTL, не выбирая определенную технологию изготовления. Логические инструментальные средства синтеза могут автоматически преобразовать проект в любую технологию изготовления. Если появляется новая технология, то специалисты не должны перепроектировать всю схему целиком. Они просто вводят в логический инструмент синтеза новое описание RTL-уровня и создают новый netlist, в соответствии с новой технологией изготовления микросхем. Логический инструмент синтеза оптимизирует схему для новой технологии по занимаемой на кристалле площади и по временным задержкам.

Если проекты пользователя выполнены на языке HDL, то функциональная проверка проекта может быть сделана непосредственно в цикле проекта, на ранней стадии. Выполняя проверку проекта уже на уровне RTL, проектировщики могут оптимизировать и изменять описание RTL до тех пор, пока оно не начнет выполнять требуемые для проекта функции. Большинство ошибок в проектах устраняется именно на этом этапе. Это значительно сокращает время проектирования, потому что при устранении ошибок нет необходимости многократно выполнять

достаточно большой объем работ по размещению проекта на кристалле на уровне вентиляей.

Проектирование на языке HDL очень похоже на программирование. Текстовое описание с комментариями – это более простой способ разработки и отладки схемы. Текстовое описание выглядит более компактным, по сравнению со схемными решениями, представленными на уровне примитивов – триггеров и вентиляей. Даже для проектов средней сложности, схемы, выполненные на уровне триггеров и вентиляей, довольно трудны в сопровождении и дальнейшей модернизации.

2. Соглашения об именах Verilog

Для Verilog приняты следующие соглашения об именах, применяемых пользователями в своих проектах:

- Verilog отличает строчные и прописные буквы.
- Два знака «//» используются для строчного комментария.
- Набор знаков «/*» используется для начала блочного комментария, а «*/» – для конца комментария.
- В именах можно использовать цифры, буквы, символ подчеркивания «_» и символ «\$».
- Имена могут начинаться с цифры, буквы или символа подчеркивания «_».
- В именах нельзя использовать символ пробела.

Зарезервированные слова Verilog приведены в табл. 1.1.

Сигналы или переменные могут быть представлены следующими логическими уровнями:

- 0: ноль, логический низкий уровень, «ложно», «земля» (zero, logic low, false, ground);
- 1: единица, логический высокий уровень, «истина», «питание» (one, logic high, power);
- X: неизвестное значение (unknown);
- Z: высокий импеданс, неподключенный сигнал, третье состояние (high impedance, unconnected, tri-state).

Таблица 1.1

Зарезервированные слова Verilog

always	endmodule	medium	reg	tranif0
and	endprimitive	module	release	tranif1
assign	endspecify	nand	repeat	tri
attribute	endtable	negedge	rnmos	tri0
begin	endtask	nmos	rpmos	tri1
buf	event	nor	rtran	triand
bufif0	for	not	rtranif0	trior
bufif1	force	notif0	rtranif1	trireg
case	forever	notif1	scalared	unsigned
casex	fork	or	signed	vectored
casez	function	output	small	wait
cmos	highz0	parameter	specify	wand
deassign	highz1	pmos	specparam	weak0
default	if	posedge	strength	weak1
defparam	ifnone	primitive	strong0	while
disable	initial	pull0	strong1	wire
edge	inout	pull1	supply0 s	wor
else	input	pulldown	upply1	xnor
end	integer	pullup	table	xor
endattribute	join	remos	task	
endcase	large	real	time	
endfunction	macromodule	tran		
realtime				

3. Операторы

Операторы могут быть трех типов: унитарные, бинарные и тернарные. Унитарные операторы производят действия над единственным операндом. Бинарные операнды производят действия над двумя операндами. Тернарные операторы имеют два оператора, которые делают выбор для трех операндов. Примеры этих операндов:

$a = \sim b$; // \sim унитарный оператор. b – операнд

$a = b \&\& c$; // $\&\&$ бинарный оператор. b и c – операнды

$a = b ? c : d$; // $?$: тернарный оператор. b , c и d – операнды

В языках VHDL и Verilog используются следующие арифметические и битовые операторы:

- арифметические (Arithmetic);
- конкатенации (объединения) (Concentration);
- повторения (Replication);

- условные (Conditional);
- равенства (Equality);
- логические побитовые (Logical Bit-wise);
- логического сравнения (Logical Comparison);
- свертки (Reduction);
- отношения (Relational);
- сдвига (Shift);
- унитарные арифметические (Unary Arithmetic (Sign)).

4. Операнды (Operands)

Есть несколько типов операндов, которые могут быть определены в выражениях. Самый простой тип – это цепь или регистр, и, если при этом указывается на полную разрядность цепи или регистра, то можно приводить только название цепи или регистра. В этом случае все биты, составляющие цепь или выходы регистра, используются как операнд.

Как цепи, так и регистры могут иметь несколько разрядов. При этом, если используется только единственный бит из вектора цепи или регистра, то необходимо использовать операнд выбора бита. Если при обращении к группе смежных битов в векторной цепи или в регистре используется только часть битов, то необходимо применить операнд выбора части битов из вектора.

На элемент памяти можно сослаться как на операнд. Конкатенация других операндов (включая вложенные конкатенации) может быть определена как операнд. Запрос функции – это тоже операнд.

5. Этапы ведения проекта

Типовой проект ведется следующим образом (рис. 1.1). Проект начинается с проработки технического задания на проектирование и создания спецификаций проекта. Далее выполняется поведенческое описание проекта. На этом этапе моделируется то, как проект будет взаимодействовать с внешней средой. После того как этот этап выполнен, разработчики переходят к одному из наиболее существенных этапов – RTL-описанию. На этом этапе производится моделирование работы устройства на уровне передачи информации между регистрами.

Далее производят компиляцию проекта под конкретную технологию, проверяют временные соотношения сигналов и производят окончательную трассировку на физическом уровне.



Рис. 1.1. Этапы проведения разработки

6. Модули

Модуль – это основной стандартный блок при описании на языке Verilog. Модуль может быть как элементом библиотеки самого нижнего уровня, так и блоком, содержащим несколько субблоков низшего уровня. Как правило, субблоки группируются в модули для того, чтобы обеспечить общие функциональные возможности, которые многократно используются во многих местах проекта. Если проводить аналогию с Си++, то понятие модуль в Verilog можно соотнести с понятием класс в Си++.

Если рассматривать с точки зрения проекта верхнего уровня, то модуль обеспечивает требуемые функциональные возможности через интерфейс портов (вводы и выводы), но, при этом, вся «начинка» модуля и ее функционирование будет скрыто для проекта верхнего уровня. Это позволяет проектировщику гибко менять внутреннюю организацию модуля, не затрагивая остальную часть дизайна. Если провести аналогию со схемным описанием проекта, то схема тоже может быть выполнена как иерархическая структура. На верхнем уровне иерархии схема представляет собой набор квадратиков, представляющих схемы более низкого уровня, и провода, связывающие эти квадратики. Каждый модуль нижнего уровня имеет место для подключения проводов – разъем или контакт на схеме. В Verilog’е такое место, аналогичное разъему на схеме, называется порт.

В примере ниже показано описание модуля. В языке Verilog модуль объявляется при помощи ключевого слова `module`. Соответствующее ключевое слово `endmodule` должно быть написано в конце определения модуля. Каждый модуль должен иметь имя – `module_name`, которое является идентификатором для модуля, и список входов и выходов модуля – `module_terminal_list`.

```
module <module_name> (<module_terminal_list>);  
...  
<module internals>  
...  
...  
endmodule
```

Определение модуля всегда начинается с ключевого слова `module`. Название модуля, список портов модуля, объявления портов и дополнительные параметры должны быть выбраны в начале определения модуля. Список портов и объявления портов имеют место только в том случае, если модуль имеет какие-либо порты. Порты предназначены для того, чтобы взаимодействовать с внешними сигналами или другими модулями.

У модуля есть пять компонентов:

- объявления переменных (`variable declarations`);
- операторы прохождения данных (`dataflow statements`);
- установленные компоненты модулей (инстансы) более низкого уровня (`instantiation of lower modules`);
-

- блоки с поведенческим описанием (behavioral blocks)
- и задачи или функции (tasks or functions).

Эти компоненты могут быть расположены в любом порядке и в любом месте в описании модуля. Слово `endmodule` должно всегда стоять в конце определения модуля. Все компоненты, кроме ключевого слова – `module`, названия модуля и `endmodule`, являются дополнительными и могут быть расположены в любом месте в описании модуля, в соответствии с тем, как это нужно разработчику при описании модуля. Verilog позволяет выполнить несколько описаний модулей в одном файле. В таком файле модули могут быть расположены в любом порядке.

Модуль обеспечивает шаблон, из которого можно создать фактические компоненты, устанавливаемые в проект. Компилятор производит вызов файла модуля, и создается новый объект – уникальный компонент из шаблона модуля. Каждый объект имеет свое собственное уникальное имя, свои переменные, параметры и интерфейс ввода/вывода. Процесс создания объектов от шаблона модуля называют установкой (instantiation) компонента, и сами объекты называют компонентами (instance).

```
// Define the top-level module called ripple carry
// counter. It instantiates 4 T-flipflops.
module ripple_carry_counter(q, clk, reset);
    output [3:0]q; //I/O signals and vector declarations
                  //will be explained later.
    input clk, reset; //I/O signals will be explained later.
                    //Four instances of the module T_FF
                    //are created. Each has a unique
                    //name.Each instance is passed
                    //a set of signals. each instance
                    //is a copy of the module T_FF
    T_FF tff0(q[0],clk, reset);
    T_FF tff1(q[1],q[0], reset);
    T_FF tff2(q[2],q[1], reset);
    T_FF tff3(q[3],q[2], reset);
Endmodule

module T_FF(q, clk, reset);
//Declarations to be explained later
    output q;
    input clk, reset;
    wire d;
    D_FF dff0(q, d, clk, reset); // Instantiate D_FF.
                                //Call it dff0.
    not n1(d, q); // not gate is a Verilog primitive.
Endmodule
```

В языке Verilog нет возможности делать вложенные описания модулей. Одно определение модуля не может содержать другое определение модуля в пределах определений `module` и `endmodule`. Вместо этого описание одного модуля может включать установленные компоненты от других модулей. Важно не путать описание модуля и установленные компоненты – копии этого модуля. Определение модуля просто описывает то, как модуль будет работать, его внутреннюю организацию и его интерфейс.

Для того чтобы использовать модуль в проекте, необходимо выполнить установку компонента – копии этого модуля и в самом начале файла необходимо вставить директиву препроцессора ``include` с именем того файла, в котором находится описание устанавливаемого модуля.

Синтаксис: ``include имя_файла`. Например, ``include ripple carry.v`.

7. Порты

Порты обеспечивают интерфейс, с помощью которого модуль может общаться с внешней средой. Например, входы ввода/вывода сигналов из микросхемы представляют собой ее порты. Среда может взаимодействовать с модулем только через порты модуля. Внутренняя организация модуля недоступна извне. Это обеспечивает проектировщику очень большие возможности, потому что внутренняя организация модуля может быть изменена, но это не затрагивает внешнюю, по отношению к модулю, среду до тех пор, пока интерфейс модуля не будет изменен.

Порты модуля определяются следующим образом: после имени модуля в круглых скобках приводится список портов модуля так, как показано здесь:

```
module name ( port_list );
```

8. Список портов

Определение модуля содержит список портов. Если модуль не обменивается сигналами со средой, то у него нет списка портов.

Порт в списке может быть выражен следующим образом:

- идентификатор;
- единственный бит, выбранный из вектора, объявленного в пределах модуля;

– группа битов, выбранных из вектора, объявленного в пределах модуля;

– конкатенация любого из вышеупомянутых (конкатенация – процесс объединения нескольких однобитовых или многобитовых операндов в один большой битовый вектор. Для получения дополнительной информации см. раздел «Оператор конкатенации»).

Каждый порт в списке должен быть объявлен как ввод, вывод или двунаправленный порт при помощи ключевых слов `input`, `output` или `inout`.

```
module fulladd4(sum, c_out, a, b, c_in);  
//Module with a list of ports  
module Top;  
// No list of ports, top-level module in simulation
```

На модуль `fulladd4` входные сигналы поступают на порты `a`, `b` и `c_in`, а выходные сигналы, формируемые модулем, выдаются из него через порты `sum` и `c_out`. Модуль `Top` – модуль верхнего уровня в моделировании, и к нему нет необходимости передавать сигналы или получать их извне. Таким образом, этот модуль не имеет списка портов.

Стили описания – функциональное (поведенческое) и структурное

`Verilog` является языком, который позволяет выполнять как функциональное или поведенческое, так и структурное описание.

9. Структурные описания

Элементами структурного описания в языке `Verilog` являются различные логические вентили, специфические библиотечные компоненты и компоненты, определяемые пользователем. Само структурное описание как таковое представляет собой набор компонентов, установленных в проект и связанных проводами. В простейшем случае структурное описание может рассматриваться как простой `netlist`, в котором представлены установленные в проекте вентили, порты которых связаны проводами между собой. Однако, в отличие от списка связей (`netlist`), цепи в структурном описании могут быть также представлены в виде произвольных выражений, которые описывают то, как будет функционировать та или иная цепь. Такое

назначение на цепь называют непрерывным назначением. Непрерывные назначения – удобное средство для связи между простыми списками связей (netlist) и функциональными описаниями.

Структурное описание Verilog может содержать различные иерархические конструкции и конструкции уровня вентилей, а также определения модулей, установленные компоненты и подключения из netlist.

10. Функциональное описание

Функциональные элементы описания в языке Verilog – это, в основном, объявления функций, задачи – tasks и блоки – always. Эти элементы описывают функцию схемы, именно то, как она работает, но не описывают ее физическую сущность или размещение на кристалле. Выбор вентилей и компонентов оставляют полностью на усмотрение того программного инструмента, который будет проводить компиляцию проекта.

Разработчик может создать описания с функциональными конструкциями Verilog. Эти конструкции могут появиться в пределах функций или блоков (always). Функции подразумевают только комбинационную логику, а блоки (always) могут подразумевать как комбинационную логику, так и триггеры, регистры или счетчики.

11. Назначения (Assignments)

Назначение – основной механизм для того, чтобы получить значения для цепи и для регистра. Есть две канонических формы назначения:

- непрерывное назначение, которое определяет значения на цепь;
- процедурное назначение, которое определяет значения на регистры.

Назначение состоит из двух частей, которые отделены символом равенства (=). Правая сторона может быть любым выражением, которое вычисляется до некоторого определенного значения. Левая сторона указывает переменную, на которую должно быть сделано назначение, вычисленное в соответствии с тем, что записано на правой стороне. Левая сторона может принять одну из следующих форм в зависимости от того, является ли назначение непрерывным

(continuous assignment) или процедурным (procedural assignment) (табл. 1.2).

Таблица 1.2

Разрешенные формы назначений для левой стороны

Тип назначения	Выражение на левой стороне
Непрерывное назначение	<ul style="list-style-type: none"> – цепь (net)(vector или scalar) – определенный бит, выбранный из векторной цепи (constant bit select of a vector net) – определенная группа битов, выбранных из векторной цепи (constant part select of a vector net) – комбинация любых вариантов из трех перечисленных
Процедурное назначение	<ul style="list-style-type: none"> – регистр (register) (vector или scalar) – определенный бит, выбранный из регистра (bit select of a vector register) – определенная группа битов, выбранных из регистра (constant part select of a vector register) – элемент памяти (memory element) – комбинация любых вариантов из четырех перечисленных

12. Непрерывные назначения

Непрерывные назначения управляют значениями цепи, вектора и скаляра. Значение слова «непрерывное» – в том, что назначение происходит всякий раз, когда моделирование заставляет значение правой стороны изменяться. Непрерывные назначения обеспечивают способ моделировать комбинационную логику, не определяя взаимосвязь вентилях логики. Вместо этого модель определяет логическое выражение, которое управляет цепью. Выражение на правой стороне непрерывного назначения ничем не ограничено. Оно может даже содержать ссылку к функции. Таким образом, результат оператора выбора – case, условного оператора – if или другой процедурной конструкции может управлять цепью.

13. Назначение, выполняемое при объявлении цепи

Назначение, выполняемые при объявлении цепи, позволяет поместить непрерывное назначение на цепь в том же самом утверждении, которое объявляет эту цепь. Следующий пример для непрерывного назначения при <net_declaration>:

```
wire (strong1, pull0) mynet = enable;
```

Примечание. Поскольку цепь может быть объявлена только однажды, только одно назначение на эту цепь может быть сделано при объявлении именно этой конкретной цепи. Это отличается от обычного непрерывного назначения, так как обычно одна цепь может получить многократные непрерывные назначения.

14. Оператор непрерывного назначения – assign

Утверждение `<continuous_assign>` помещает непрерывное назначение на цепь, которая была предварительно объявлена, или явно, в соответствии с объявлением цепи, или неявно при использовании ее названия в списке цепей для примитива – вентиля, определяемого пользователем примитива, или устанавливаемого в проект экземпляра модуля. Вот пример непрерывного назначения на цепь, которая была предварительно объявлена:

```
assign (strong1, pull0) mynet = enable;
```

Назначения на цепях являются непрерывными и автоматическими. Это означает, что всякий раз, когда операнд, находящийся с правой стороны выражения, меняет свое значение в течение моделирования, то все, что находится с правой стороны выражения, пересчитывается и назначается на левую сторону выражения.

Вот пример использования непрерывного назначения для того, чтобы промоделировать сумматор на четыре бита с переносом. Отметьте, что назначение не могло быть определено непосредственно в объявлении цепей, потому что оно требует конкатенации в левой части выражения:

```
module adder (sum_out, carry_out, carry_in, ina, inb);  
  output [3:0]sum_out;  
  input [3:0]ina, inb;  
  output carry_out;  
  input carry_in;  
  wire carry_out, carry_in;  
  wire[3:0] sum_out, ina, inb;  
  assign {carry_out, sum_out} = ina + inb + carry_in;  
endmodule
```

Назначение на цепь при объявлении и утверждении непрерывного назначения покажем на примере с одной 16-разрядной выходной шиной. В нем производится выбор между одной из четырех входных шин, и выбранная шина подключается к выходной шине:

```
module select_bus (busout, bus0, bus1, bus2, bus3, enable, s);
  parameter n=16;
  parameter Zee=16'bz;
  output [1:n ] busout;
  input [1:n] bus0, bus1, bus2, bus3;
  input enable;
  input [1:2] s;
  tri [1:n] data; // net declaration.
  tri [1:n] busout t=enable ? data : Zee; // net declaration
  //with continuous assignment.

  assign // assignment statement with
  data = (s==0) ? bus0 : Zee, // 4 continuous assignments.
  data = (s==1) ? bus1 : Zee,
  data = (s==2) ? bus2 : Zee,
  data = (s==3) ? bus3 : Zee;
endmodule
```

Имеется входная переменная *s*, значение которой проверяется при назначении цепи. И в соответствии с тем, какое значение принимает *s*, на шину *data* коммутируются данные от одной из четырех входных шин.

Для выходной шины при установке данных сделано непрерывное назначение в объявлении цепи: если установлен сигнал разрешения – *enable*, то содержание данных назначено на выходную шину, если же сигнал разрешения не принимает ни одного из значений, использованных для сравнения (0, 1, 2, 3, 4), то на выходную шину будет назначено значение *Zee*.

15. Задержки (delays)

Задержка (delay), заданная при непрерывном назначении, определяет продолжительность времени между изменением значения, находящегося справа от операнда, и назначением, сделанным на левую сторону операнда. Если то, что находится слева от операнда, представляет собой скалярную цепь, то задержка будет представлять собой такую же задержку, какая имеет место в примитивах – вентилях. То есть производится задержка того сигнала, который приходит на вентиль, причем можно дать различные задержки для нарастающего фронта сигнала, спадающего фронта сигнала и перехода сигнала в высокий импеданс.

Если то, что находится слева от операнда, представляет собой векторную цепь, то можно применить до трех различных задержек. Следующие правила определяют, какая из задержек будет выполняться при назначении:

- если правая сторона была ненулевой и становится нулем, то используется задержка для спадающего фронта сигнала;
- если правая сторона становится z , то используется задержка перехода сигнала в высокий импеданс;
- для остальных случаев используется задержка для нарастающего фронта сигнала;
- если производится объявление цепи, то непосредственно в этом же объявлении можно назначить и задержки на эту цепь. Такое назначение задержки отличается от того случая, который имеет место в объявлении задержки при непрерывном назначении на цепь. Значение задержки может быть применено в объявлении цепи так:

```
wire #10 wireA;
```

Эту форму записи применяют для того, чтобы показать, что в цепи, называемой `wireA`, имеется задержка в течение десяти единиц времени, прежде чем сигнал, передаваемый по цепи `wireA`, пройдет через эту цепь и поступит для моделирования во все утверждения, где участвует данный сигнал. А в том случае, когда задержка делается при непрерывном назначении в объявлении, она представляет собой только часть непрерывного назначения и не является задержкой всей цепи. Тогда она не будет добавлена к задержке других драйверов на этой цепи. Более того, эта задержка не добавляется, если производится назначение, приводящее к расширению векторной цепи (то есть добавляются цепи, которые не входили в первоначальное определение с ключевым словом `vector`); задержки на положительный фронт и задержки на отрицательный фронт не будут применены к индивидуальным битам, если назначение включено в объявления этих цепей.

Описание лабораторного стенда

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

Внешний вид платы изображен на рис. 1.2. 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 нажимающаяся кнопка;
- восемь светодиодов;
- четыре кнопки;
- четыре переключателя;
- разъем для подключения внешнего генератора;
- восьмипиновый сокет для дополнительного генератора.

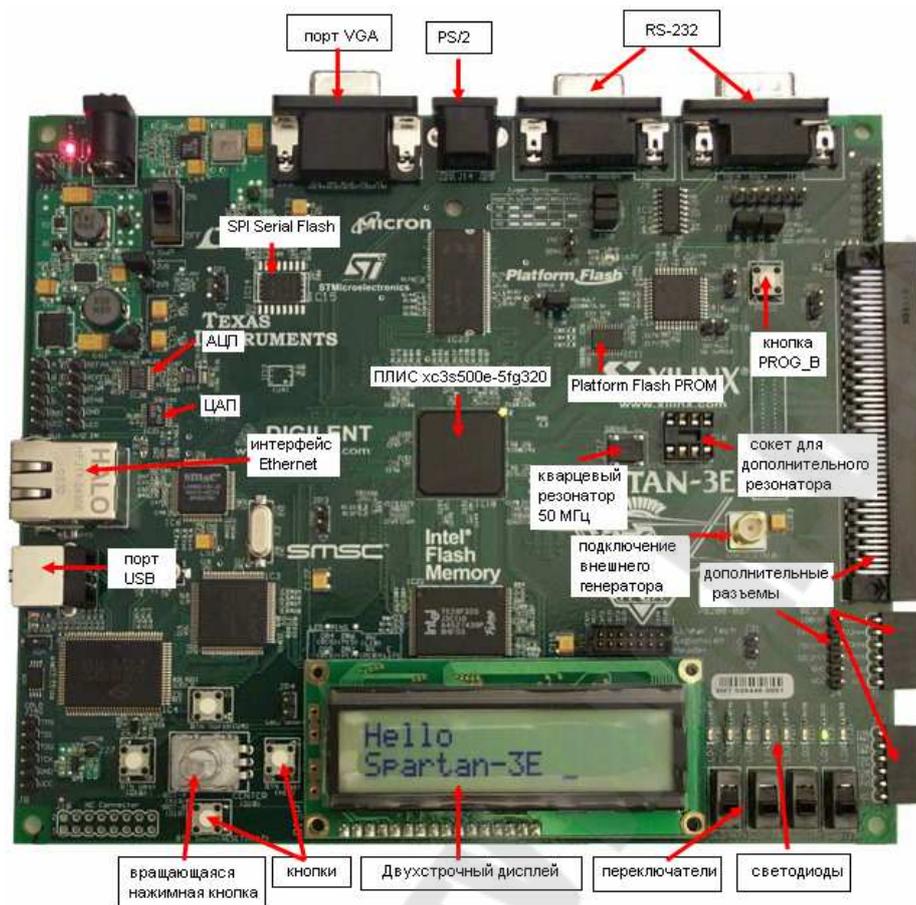


Рис. 1.2. Отладочная плата Spartan-3E Starter Kit

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 процессам, необходимым при проектировании цифрового устройства на базе ПЛИС с архитектурами FPGA и CPLD. На рис. 1.3 показан внешний вид основного окна «Навигатора проекта» и его компоненты.

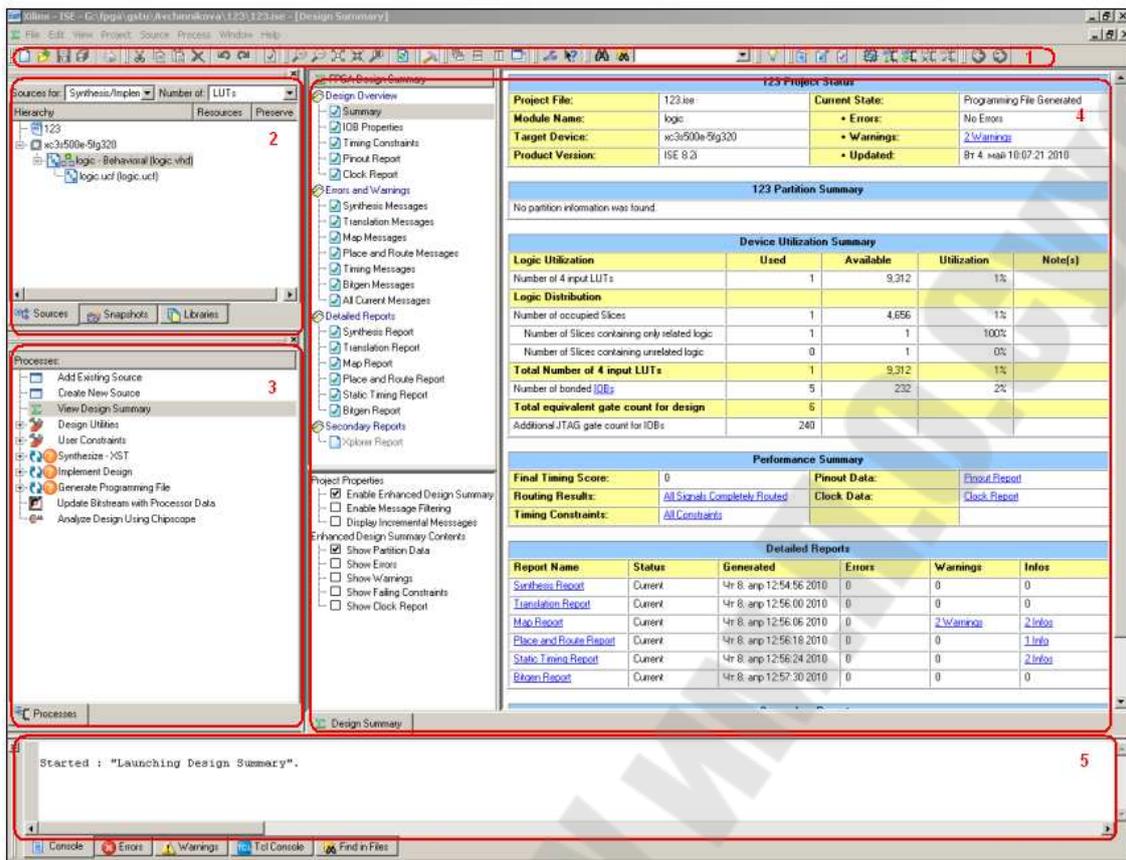


Рис. 1.3. Внешний вид основного окна «Навигатора проекта»:
 1 – инструментальная панель («Toolbar»); 2 – окно описания проекта («Source window»); 3 – окно процессов («Process window»);
 4 – рабочий стол («Workspace»); 5 – окно отчетов («Transcript window»)

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. 1.4).

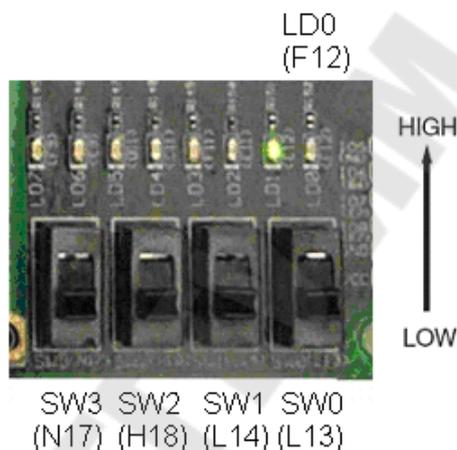


Рис. 1.4. Переключатели и светодиоды

1. Запуск САПР WebPACK ISE

Для запуска САПР WebPACK ISE требуется выбрать 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:

1) Выбрать в меню пункт «File» → «New Project...», после чего будет запущен мастер нового проекта «New Project Wizard» (рис. 1.5);

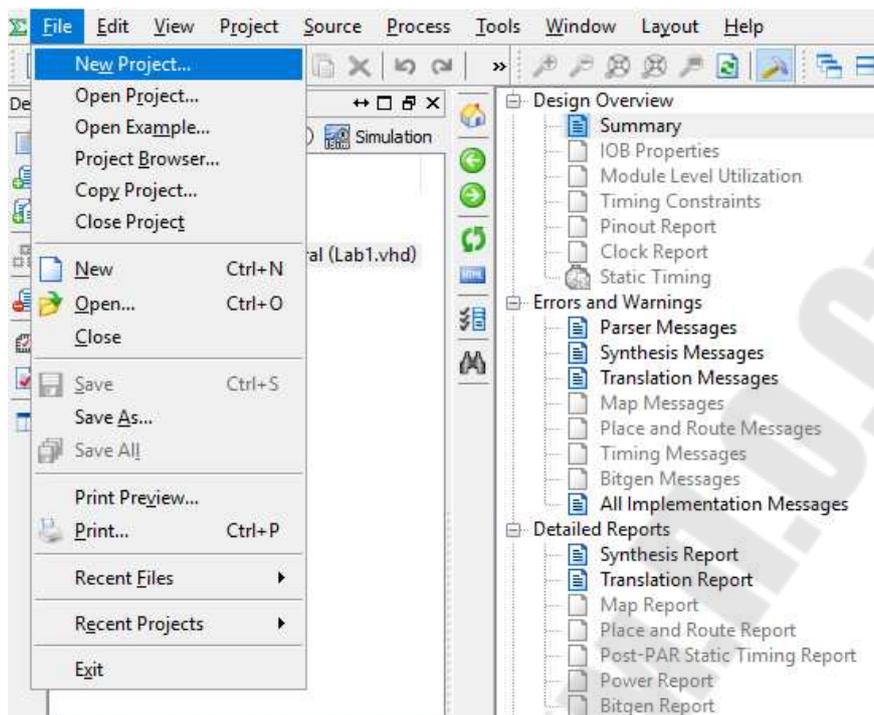


Рис. 1.5. Создание нового проекта

2) Ввести или выбрать местоположение создаваемого проекта в поле «Project Location»;

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;

4) Заполнить свойства проектируемого устройства, как показано на рис.1.6:

- 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);

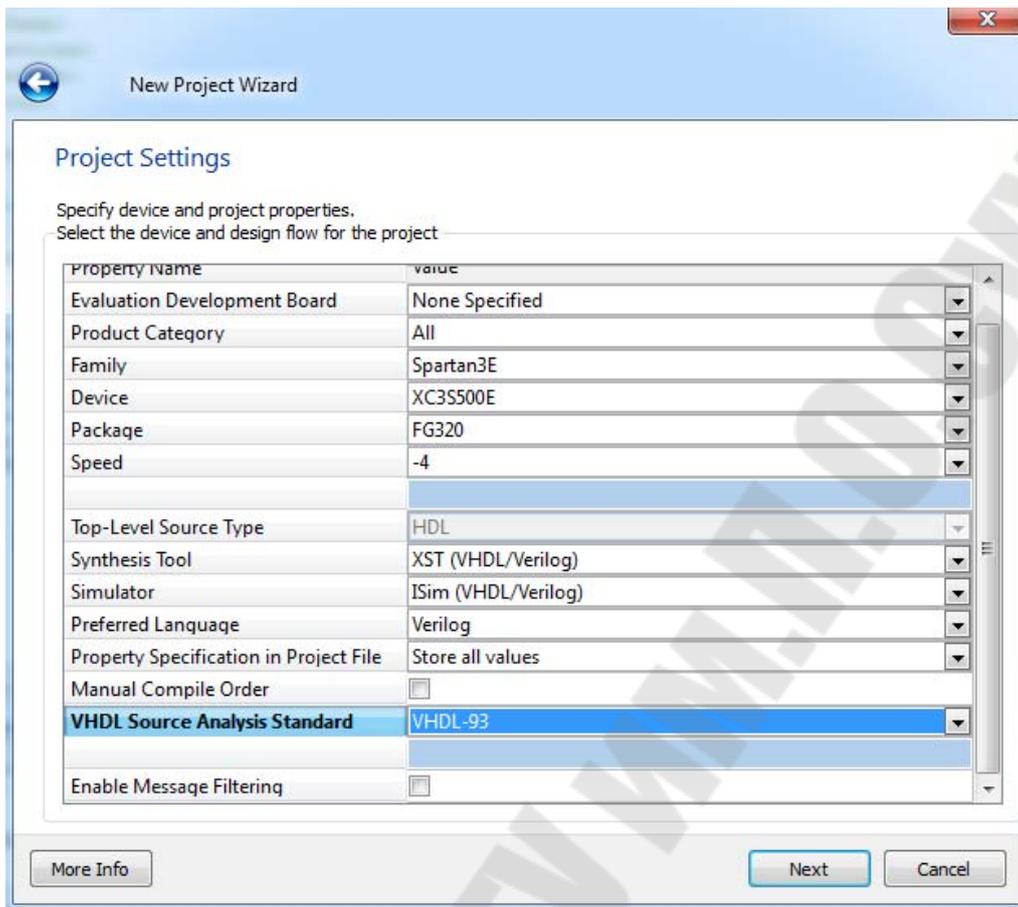


Рис. 1.6. Окно свойств проектируемого устройства

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

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

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 Verilog необходимо выполнить следующие шаги:

- 1) В окне мастера нового проекта нажать кнопку «New Source» (рис. 1.7);
- 2) Выбрать «Verilog Module» в качестве типа исходного файла (рис. 1.7);
- 3) Ввести имя файла в поле «File name»;

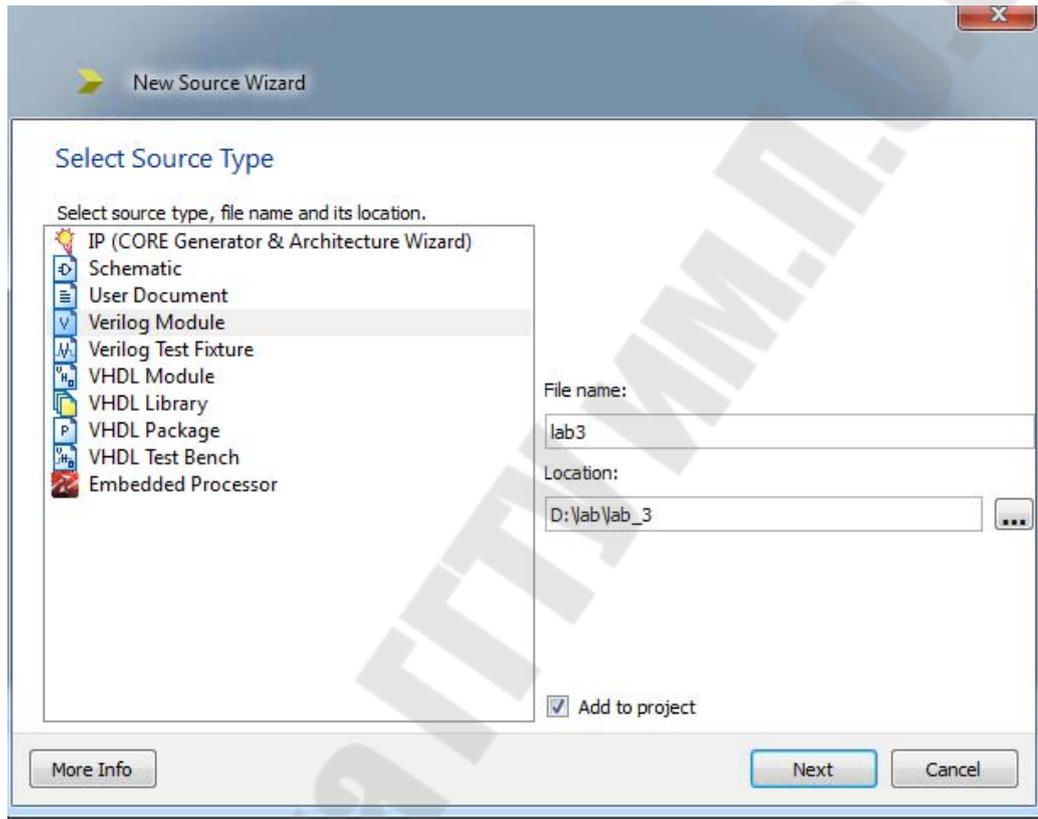


Рис. 1.7. Выбор типа исходного файла

- 4) Убедиться, что установлена опция «Add to project»;
- 5) Нажать кнопку «Next >»;
- 6) В открывшемся окне установить свойства портов ввода-вывода проектируемого устройства, как показано на рис. 1.8;
- 7) Нажать кнопку «Next >». В новом окне проверить свойства создаваемого файла с описанием устройства, после чего нажать кнопку «Finish»;
- 8) В следующих двух окнах нажать кнопку «Next >». Появится окно с резюме о новом проекте (Рис. 1.9). Если все свойства

соответствуют требованиям, предъявляемым к проектируемому устройству, нажать кнопку «Finish».

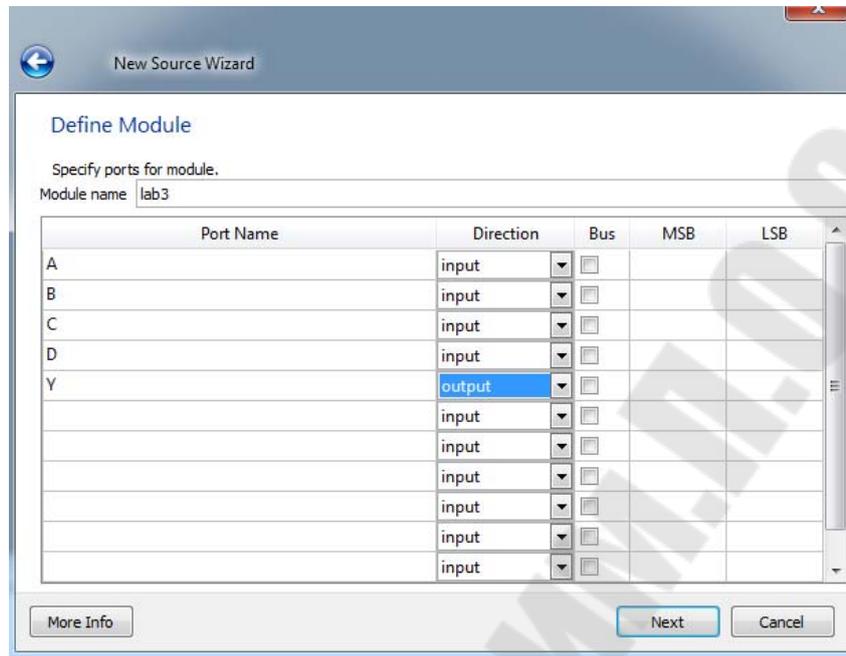


Рис. 1.8. Настройка портов ввода-вывода

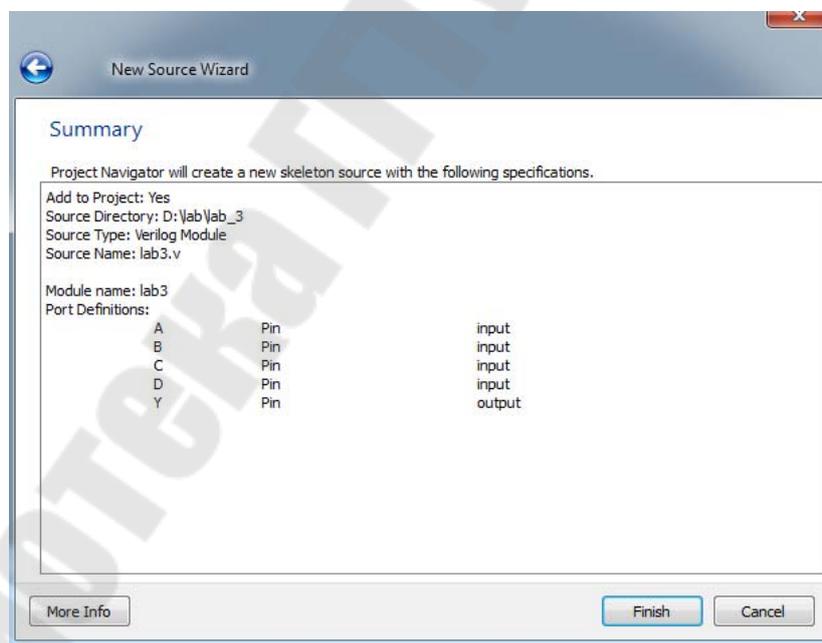


Рис. 1.9. Проверка свойств нового проекта

Следующим шагом является создание архитектуры проектируемого устройства.

Создадим устройство, которое вычисляет следующее 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 Verilog:

```
assign Y=(A^(~ B))&C|(~ D);
```

Для этого воспользуемся таблицей логических операций (табл. 1.3):

Таблица 1.3

Таблица логических операций

Условное обозначение	Выполняемая функция	Запись на языке Verilog
$X \wedge Y$	Логическое И	$X \& Y$
$X \vee Y$	Логическое ИЛИ	$X \parallel Y$
$X \oplus Y$	Исключающее ИЛИ	$X \wedge Y$
\overline{X}	Инверсия	$\sim X$

3) Сохранить файл, выбрав в меню пункт «File» → «Save» или нажав на кнопку  на инструментальной панели (рис. 1.10).

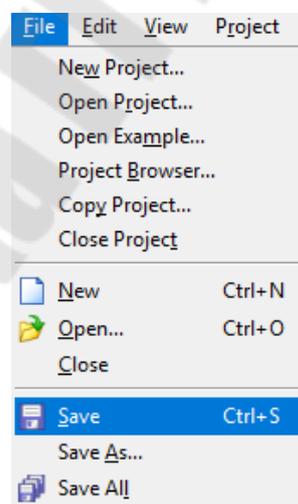


Рис. 1.10. Сохранение файла архитектуры проектируемого устройства

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

```

`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:      20:30:15 10/30/2018
// Design Name:
// Module Name:      lab1
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
module nat1(
    input A,
    input B,
    input C,
    input D,
    output Y
);

assign Y=(A^(~ B))&C|(~ D);

endmodule

```

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. 1.11).

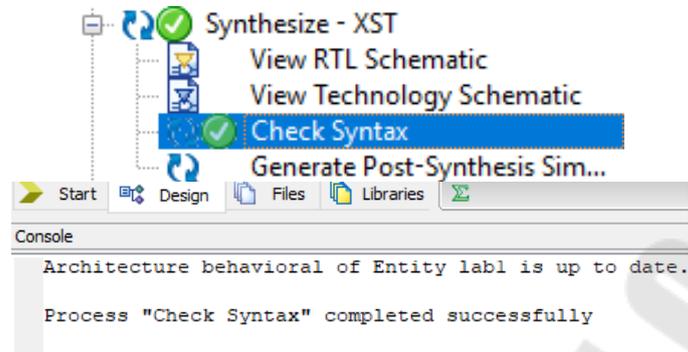


Рис. 1.11. Окно процессов

3.1. 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:

- 1) Выберем в главном меню операционной системы пункт «Пуск» → «Программы» → «PlanAhead (32 bit)».
- 2) Для начала работы выберем пункт меню «Open Project» (рис. 1.12).

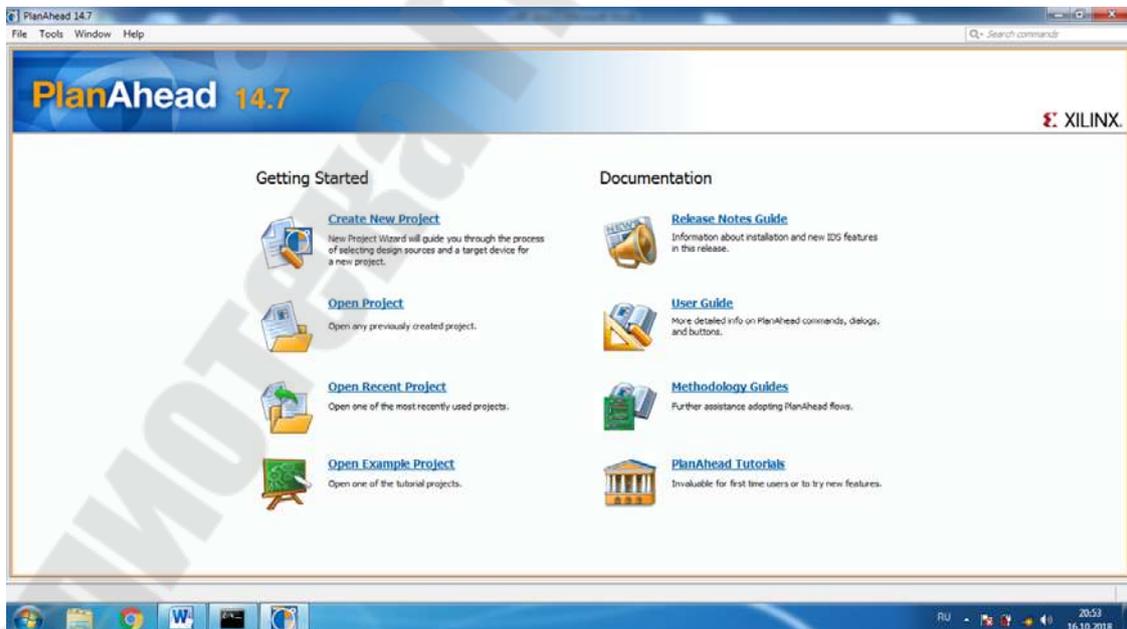


Рис. 1.12. Окно редактора PlanAhead 14.7

3) С помощью проводника выбрать файл проекта «Lab1.xise» и нажать кнопку «ОК» (рис. 1.13).

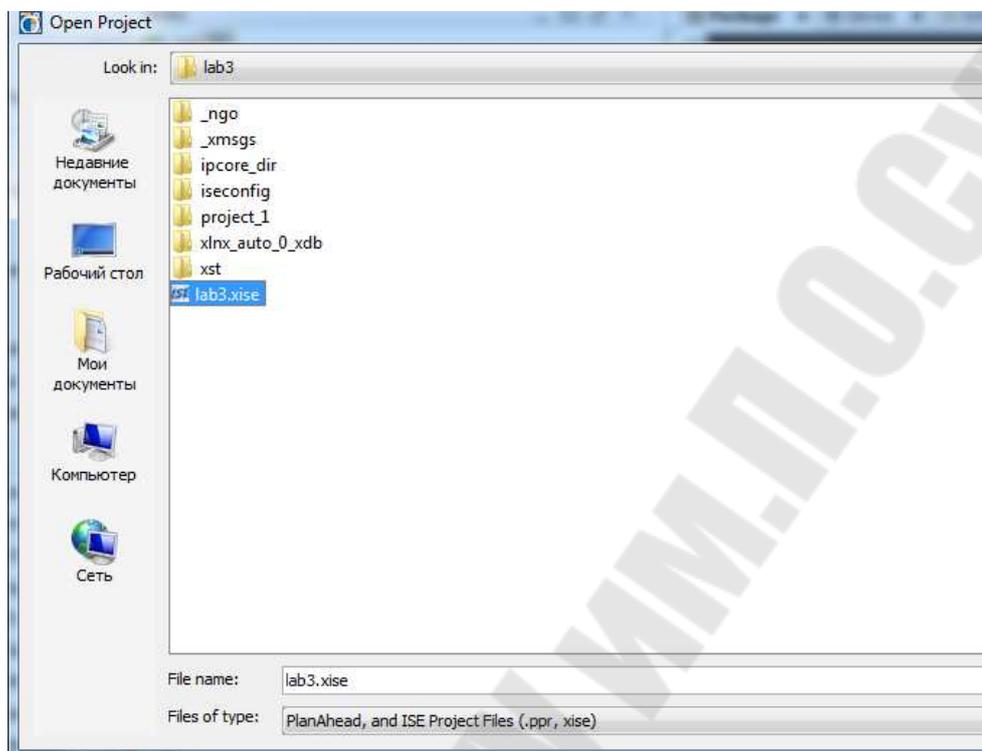


Рис. 1.13. Окно «Open Project»

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.

5) В окне «Flow Navigator» из списка «Implementation» (Рис.1.14) выбрать пункт меню «Run Implementation» . По окончании выполнения процесса в окне «Implementation Completed» (Рис.1.15.) выбрать пункт меню «Open Implemented Design» и нажать кнопку «ОК».

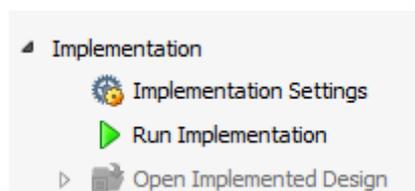


Рис. 1.14. Пункты меню «Implementation»

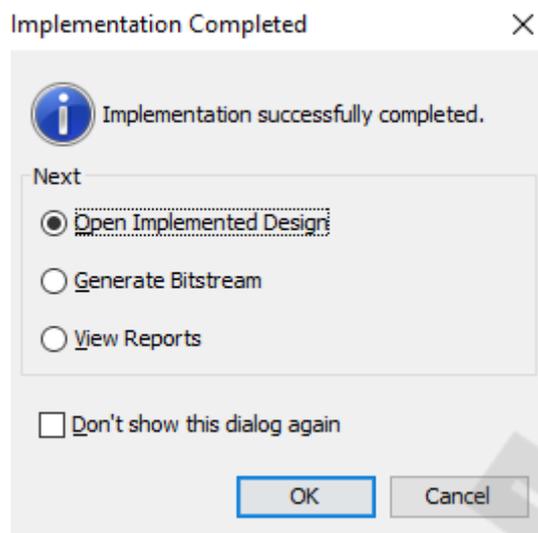


Рис. 1.15. Окно «Implementation Completed»

6) Проверить наличие схемы расположения выводов ПЛИС (Рис.1.19), а при ее отсутствии в окне программы из пункта меню «Layout» выбрать «I/O Planing» (рис.1.16).

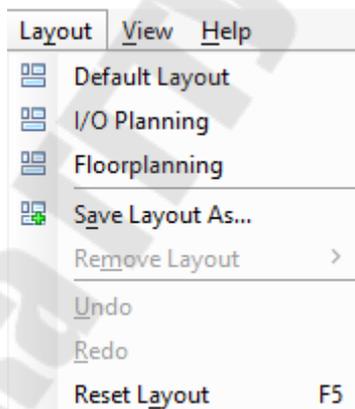


Рис. 1.16. Всплывающее меню «Layout»

7) В окне списка объектов «I/O Ports» открыть папку «Scalar Ports» (рис.1.17), 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.1.18.

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. 1.17. Список объектов проекта

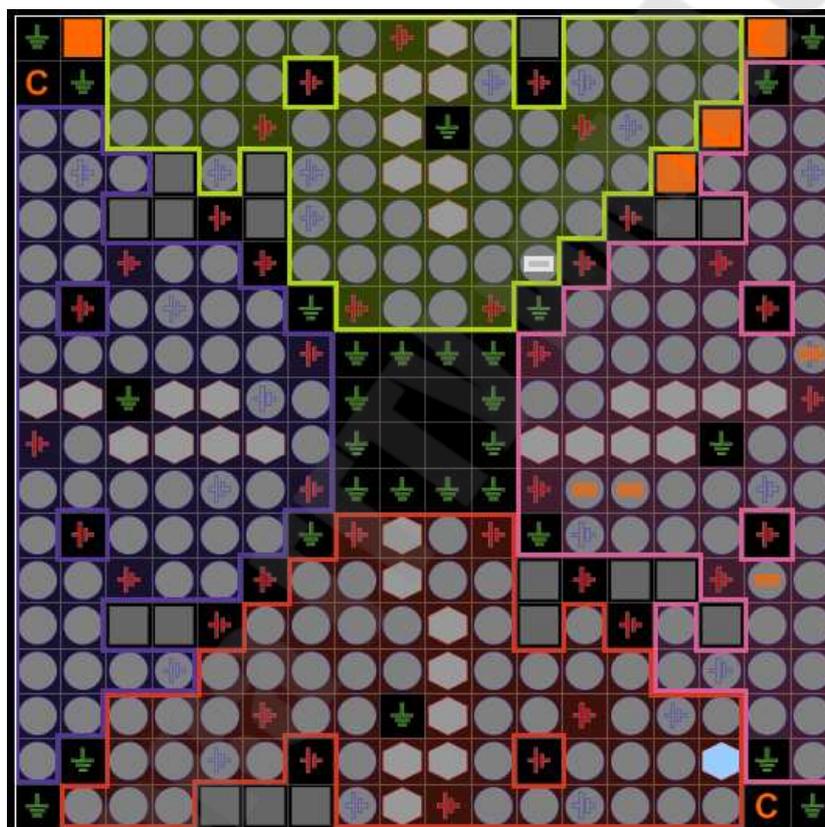


Рис. 1.18. Схема расположения выводов ПЛИС

- 8) Выбрать пункт меню File → Save
- 9) Закрыть PlanAhead 14.7.

3.2. Ручное назначение выводов

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

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

2) В окне создания ресурсов проекта (Рис.1.19) выбрать пункт «Implementation Constraints File», указать имя и ввести/указать место расположения файла, нажать клавишу «ОК».

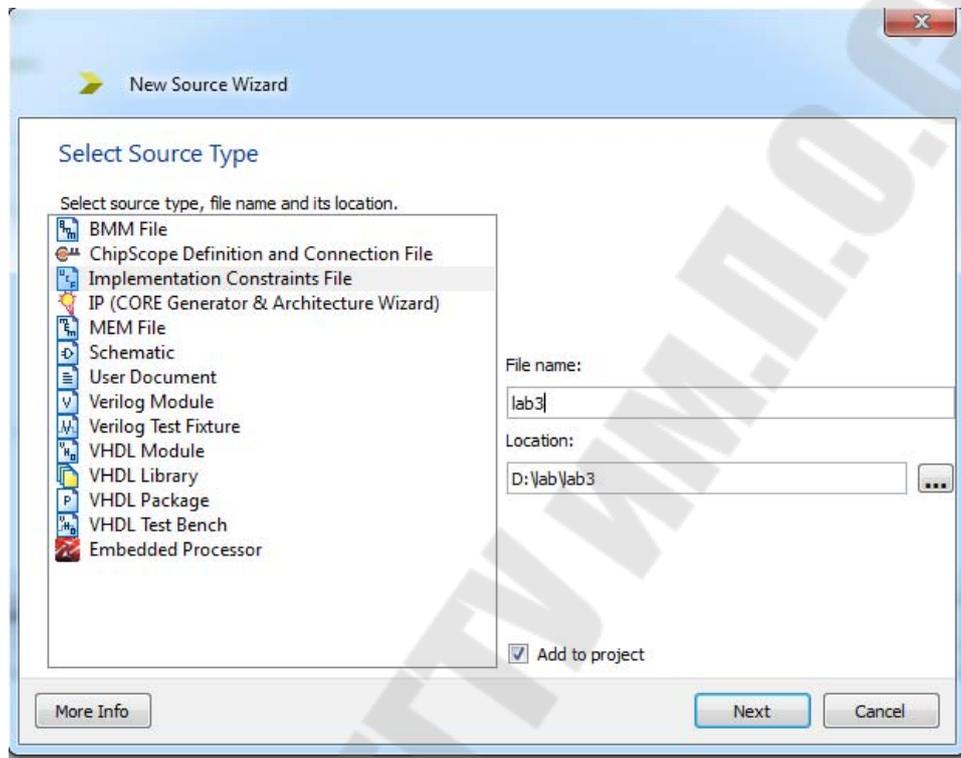


Рис. 1.19. Окно создания ресурсов проекта

- 3) Убедиться в наличии файла «.ucf» в окне ресурсов проекта.
- 4) Двойным нажатием на «.ucf» открыть окно редактора.
- 5) Описать установку в однозначное соответствие линии портов проектируемого устройства как показано на рис. 1.20.

```
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. 1.20. Конфигурация линии портов

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

4. Размещение устройства в кристалле

Процесс размещения устройства на кристалле включает следующие шаги:

- 1) Выбрать в окне описания проекта файл lab1.
- 2) Открыть окно резюме проекта, дважды щелкнув по процессу «View Design Summary» в окне процессов.
- 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. 1.21). 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.

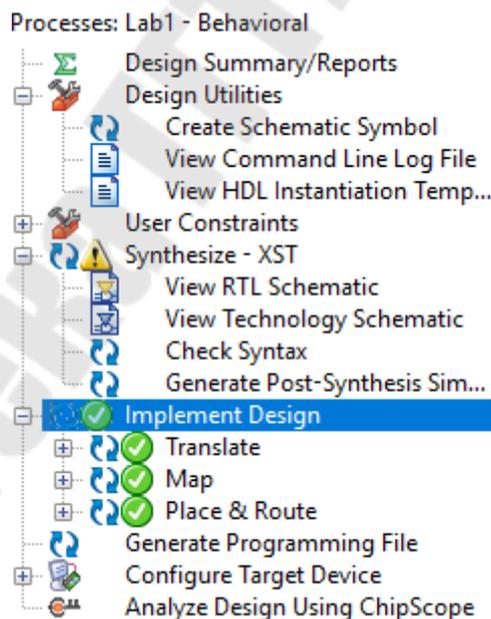


Рис. 1.21. Результат безошибочной реализации проекта

5. Загрузка конфигурации в Spartan 3

Для загрузки конфигурации в ПЛИС необходимо проделать следующие шаги:

- 1) Выбрать в окне описания проекта файл lab1.
- 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. 1.22).

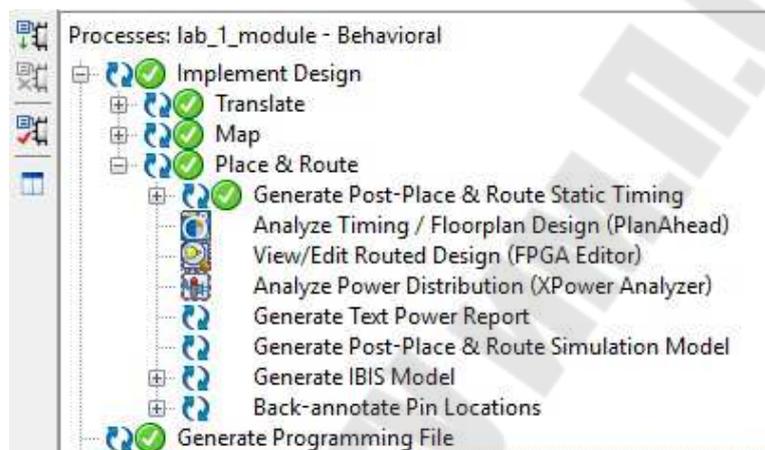


Рис. 1.22. Результат выполнения процедуры «Generate Programming File»

- 3) В окне процессов щелкнуть по символу «+», чтобы раскрыть группу процессов «Configure Target Device»
- 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 (рис. 1.23).

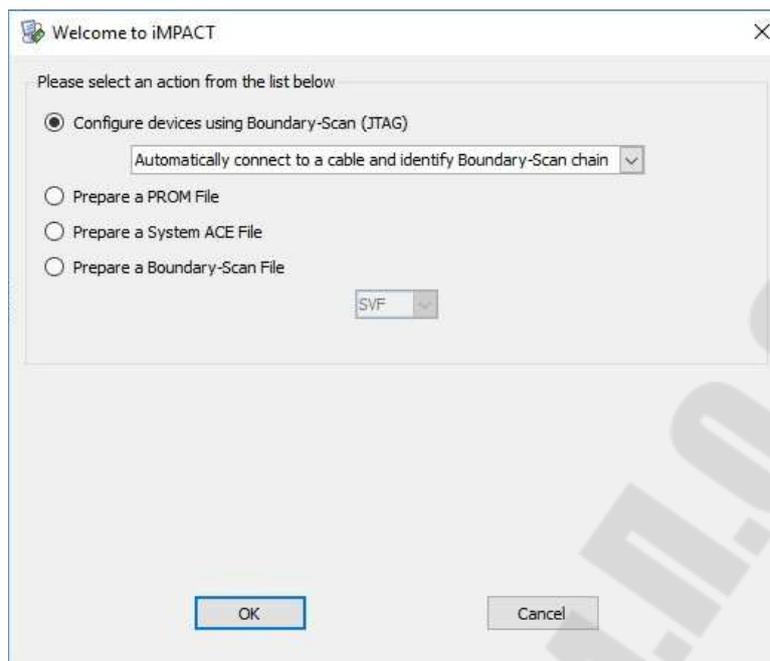


Рис. 1.23. Выбор режима работы iMPACT

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

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

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.

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».

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

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. 1.24).

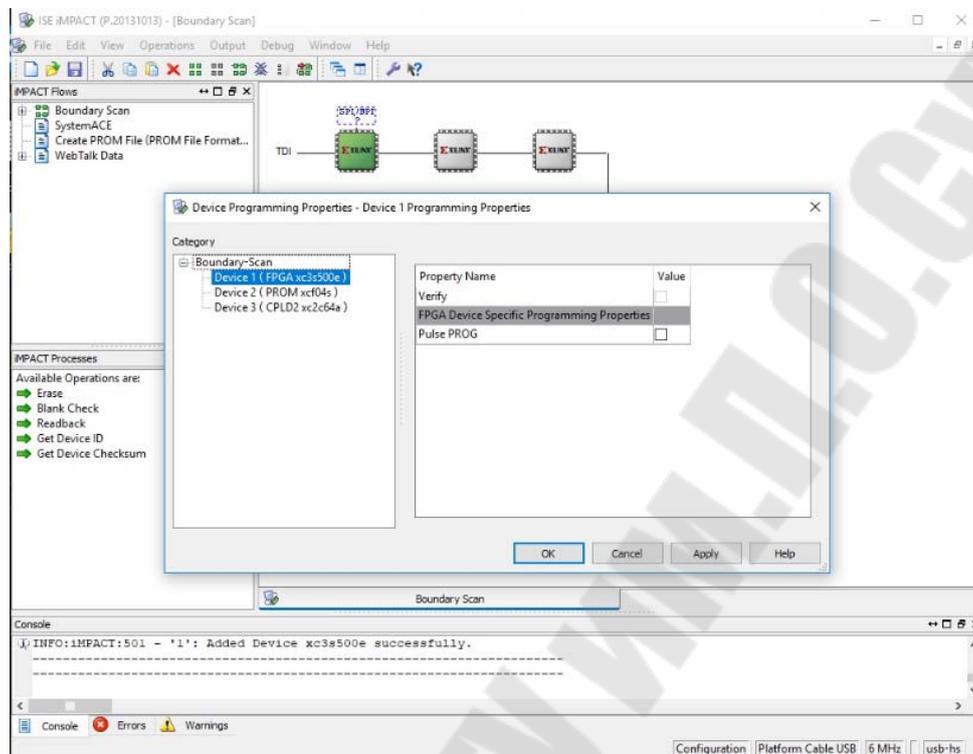


Рис. 1.24. Диалоговое окно определения настроек программирования устройства

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».

12) Закрывать программу iMPACT, не сохраняя проект.

7. Проверка правильности работы устройства

Изменяя положения переключателей N17, H18, L14 и L13, требуется провести тестирование разработанного устройства, наблюдая за светодиодом F12. Полученные результаты занести в табл. истинности.

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 лабораторной работы требуется:

- 1) составить таблицу истинности для булева выражения, соответствующего номеру варианта (смотри табл. 1.3);
- 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	$(\overline{A \oplus B}) \wedge C \vee \overline{D}$	16	$(\overline{\overline{A \vee B \wedge C}}) \oplus \overline{D}$
2	$A \wedge \overline{C} \oplus \overline{B} \vee \overline{D}$	17	$\overline{A} \wedge \overline{B} \vee \overline{C} \oplus \overline{D}$
3	$B \wedge A \oplus \overline{D} \vee \overline{C}$	18	$\overline{D} \oplus A \oplus \overline{B} \vee \overline{C}$
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	$(\overline{D \oplus B}) \vee \overline{A} \wedge \overline{C}$
6	$\overline{A} \wedge \overline{B} \oplus C \vee \overline{D}$	21	$(\overline{A \oplus B}) \wedge (\overline{D} \vee C)$
7	$(\overline{A \oplus D}) \vee (C \wedge \overline{B})$	22	$(\overline{A} \wedge \overline{B} \oplus C) \vee D$
8	$\overline{D} \oplus C \wedge \overline{A} \vee \overline{B}$	23	$\overline{A \oplus B} \oplus \overline{C} \wedge \overline{D}$
9	$(A \wedge \overline{C} \oplus \overline{B}) \vee D$	24	$(\overline{A} \oplus \overline{B}) \vee (C \oplus D)$
10	$(\overline{B} \oplus \overline{D}) \vee (C \wedge A)$	25	$(\overline{A} \vee \overline{B}) \oplus (\overline{C} \wedge \overline{D})$
11	$A \wedge B \vee \overline{D} \oplus \overline{C}$	26	$\overline{B} \wedge C \oplus \overline{\overline{D} \wedge \overline{A}}$
12	$\overline{D} \oplus C \wedge \overline{B} \vee \overline{A}$	27	$(\overline{\overline{A} \vee C}) \oplus (\overline{B} \vee \overline{D})$
13	$(\overline{D} \oplus C) \vee (\overline{\overline{A} \wedge \overline{B}})$	28	$\overline{A} \oplus B \wedge \overline{C} \oplus \overline{D}$
14	$(\overline{\overline{A \oplus B}}) \wedge (\overline{\overline{C \oplus D}})$	29	$(A \wedge \overline{C} \oplus \overline{D}) \vee B$
15	$\overline{\overline{\overline{A \oplus B} \oplus \overline{C} \wedge \overline{D}}}$	30	$(\overline{\overline{\overline{A \oplus B} \oplus D}}) \wedge C$

Содержание отчета

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

Контрольные вопросы

1. Какие преимущества использования языков HDL?
2. Перечислите основные этапы ведения проекта.
5. Для чего применяются модули в языке Verilog?
6. Для чего применяются порты?
7. Приведите пример непрерывных назначений.
6. Приведите пример назначения, выполняемого при объявлении цепи.

Лабораторная работа № 2

Реализация последовательностных цифровых устройств на основе отладочной платы Spartan-3E Starter Kit

Цель работы

Научиться описывать последовательностные устройства различными методами, а также создавать тестовые модули для проверки правильности работы программ.

Основные теоретические сведения

Последовательностные цифровые устройства (ПЦУ) характеризуются тем, что выходные сигналы зависят не только от текущих значений входных сигналов, но и от последовательности значений входных сигналов, поступивших на входы в предшествующие моменты времени. Последовательностные цифровые устройства на языке Verilog описываются при помощи процедурных назначений (Procedural Assignments). Рассмотрим их подробно.

1. Процедурные назначения (Procedural Assignments)

И выражения непрерывных назначений чаще преобразуются в комбинаторную схему, которая непрерывно управляет цепью. В отличие от непрерывных, процедурные назначения помещают входные значения в *регистры*, причем эта запись данных производится только в *определенные моменты времени*.

Процедурные назначения происходят в пределах процедур типа **always**, **initial**, **task** и **function** и их можно трактовать как защелкнутые – «triggered» – назначения. Защелкивание происходит тогда, когда процесс выполнения в моделировании достигает конкретного назначения, находящегося в пределах данной процедуры. Условные утверждения могут управлять процессом назначения: разрешать или не разрешать выполнять блоки назначений. Контроль событий (**events**), управления задержки (**delay**), условные операторы (**if**), операторы выбора (**case**) и утверждения выполнения цикла (**loop**) – всех их можно использовать для того,

чтобы иметь возможность управлять процессом выполнения блоков назначений.

Левая сторона процедурного оператора назначения может содержать только следующие типы данных:

- переменные типа **reg**;
- выбор бита из переменной типа **reg**;
- выборы части битов из переменной типа **reg** (их положение должно быть описано как константы);
- целые числа;
- конкатенации предыдущих типов данных.

Компилятор HDL назначает младший бит на левой стороне в соответствии со значением младшего бита на правой стороне. Если число битов на правой стороне больше, чем число битов на левой стороне, то со старшими битами на правой стороне не производится никаких действий. Если число битов на левой стороне больше, чем число битов на правой стороне, то старшие биты на правой стороне дополняются нулями. Компилятор HDL позволяет выполнять многократные процедурные назначения.

2. Инициализация переменных (Initial)

Инициализация является процедурным назначением и производится только в пределах утверждения `initial`. Это утверждение начинается с ключевого слова `initial`. Утверждение, которое следует за ключевым словом, должно быть оператором назначения, назначающим определенное значение на выход `reg`:

```
module shift;  
  reg [3:0] start;  
  initial  
  begin  
    start = 1;  
  end  
endmodule
```

3. Блоки always

Блоки `always` могут быть синтезированы как триггеры-защелки или триггеры, срабатывающие по фронту сигнала, но они могут быть выполнены и как комбинационная логика. Все сигналы, входящие в блоки `always`, должны быть определены как регистры и триггеры.

Блок `always` может содержать логику, выходная информация которой защелкивается на триггерах при изменении уровня сигнала или на переднем/заднем фронте сигнала.

Синтаксис блока `always` показан ниже:

```
always @ ( event-expression [or event-expression*] )
begin
... statements ...
end
```

4. Выражение события – event

Выражение события (`event`) объявляет то, что в схеме присутствуют переключающиеся сигналы и осуществляется контроль управления по событиям, вызванным переключением сигналов. Слово `or` объединяет в группы несколько таких сигналов. В языке Verilog определено следующее: если происходят события, указанные в выражениях для переключающихся сигналов, то весь блок выполняется, и это приводит к повторному вычислению всех переменных, входящих в конструкцию `always`. Еще раз необходимо отметить, что весь блок выполняется даже и в том случае, если только один из переключающихся сигналов в группе имеет событие по переключению – **event**. Однако если выражение для события не происходит синхронно с теми событиями, которые входят в описание конструкции `event`, то компилятор HDL их игнорирует:

```
always @ ( a or b or c )
begin
    f = a & b & c
end
```

В примере выше `a`, `b`, и `c` используются как асинхронные переключающиеся сигналы.

Если какой-нибудь из сигналов изменится, то симулятор повторно моделирует блок `always` и вновь вычисляет значение `f`. Однако на самом деле в этом примере компилятор HDL не выполнит эти цепи как триггеры, потому что они не синхронны. Но, тем не менее, чтобы получить такую функцию, необходимо указать все переменные, которые используются в блоке `always` как триггеры. Если такое описание не будет сделано, то компилятор HDL выдаст предупреждающее сообщение.

В редакции Verilog 2001 в качестве выражения для события (event) может быть использован символ «*» – повторная обработка конструкции always проводится от изменения всех сигналов.

Для синхронных триггеров применяются следующие блоки always:

```
//По переднему фронту синхрочастоты
always @ ( posedge event ) begin
... statements ...
end

//По заднему фронту синхрочастоты
always @ ( negedge event ) begin
... statements ...
end

//По фронту синхрочастоты или
//по сигналу асинхронной установки/сброса
always @ (posedge CLOCK or negedge reset) begin
if (!reset) begin
... statements ...
end
else begin
... statements ...
end
end

//По фронту синхрочастоты или по двум сигналам
// асинхронной установки/сброса,
always @(posedge CLOCK or posedge event1 or negedge
event2)
begin
if (event1) begin
statements ...
end
else if (!event2) begin
statements ...
end
else begin
statements ...
end
end
```

Когда выражение события не содержит posedge или negedge, то обычно генерируются не регистры, а комбинационная логика, хотя могут быть сгенерированы проходные триггеры-защелки (flowthrough).

5. Процедурные утверждения назначения – *assign* и *deassign*

Процедурные утверждения для назначений *assign* и *deassign* позволяют сделать непрерывные назначения на регистры для тех промежутков времени, которыми управляют эти назначения. Назначение процедурного оператора назначения *assign* отменяет процедурные назначения, заданные на этот регистр до данного момента времени, и производит новые назначения. Процедурное утверждение *deassign* отменяет заданное непрерывное назначение на регистр. Процедурные утверждения *assign* и *deassign* позволяют, например, провести моделирование асинхронных входов сброса/установки (*clear/preset*) на D-триггере, тактируемом по фронту сигнала. В этом триггере синхрочастота будет запрещена, когда сигналы сброса или установки будут находиться в активном состоянии.

Покажем использование назначений *assign* и *deassign* в поведенческом описании триггера D-типа с входами сброса/установки:

```
module dff(q,d,clear,preset,clock);
    output q;
    input d,clear,preset,clock;
    reg q;

    always @(clear or preset) // Назначения для
        if(!clear)           // выполнения асинхронного
            assign q=0;       // сброса / установки
        else if(!preset)
            assign q=1;       // Назначения для записи
        else                  // входных данных,
            deassign q;       // производящееся по
    always @(posedge clock) // переднему фронту clock
        q=d;                 //- событие @(posedge clock)
endmodule
```

Если вход сброса или вход установки будет установлен в низкий уровень, то на выходе *q* непрерывно будет выдаваться соответствующее постоянное значение, и положительные фронты синхрочастоты не будут управлять выходом *q*. Когда и вход сброса, и вход установки находятся в высоком уровне, тогда на выход *q* будет действовать назначение *deassign*.

Если ключевое слово назначения *assign* будет применено к регистру, для которого уже есть процедурное непрерывное назначение, то это новое процедурное непрерывное назначение

автоматически произведет отмену ранее сделанного на регистр назначения.

6. Утверждения *if...else*

Утверждения *if...else* позволяют выполнять блок утверждений в зависимости от результата проверки на истинность или ложность выражений, определяющих логику выбора.

В соответствии со значением этих выражений и производится обработка блока утверждений, входящих в конструкцию *if...else*.

Синтаксис утверждения *if...else* показан ниже:

```
if ( expr )
  begin
    ... statements ...
  end
else
  begin
    ... statements ...
  end
```

Условный оператор состоит из ключевого слова *if* и сопровождается выражением в круглых скобках. Условный оператор *if* сопровождается утверждением или блоком утверждений, которые начинаются со слова *begin* и заканчиваются словом *end*. Если значение выражения является отличным от нуля, то выражение истинно (*true*), и блок утверждений, который следует в этой конструкции, выполняется. Если значение выражения ноль, то выражение ложно (*false*), и блок утверждений, который следует в этой конструкции, не выполняется.

Дополнительное утверждение *else* может следовать за оператором *if*. Если выражение, следующее за *if*, ложно, то выполняется утверждение или блок утверждений, следующий за *else*.

Утверждения *if...else* могут привести к синтезу регистра. Регистры синтезируются в том случае, когда вы не устанавливаете значение этого же регистра *reg* во всех ветвлениях условной конструкции. Компилятор HDL синтезирует логику мультиплексора (или подобную логику выбора) от одного оператора *if*. Условное выражение в операторе *if* синтезируется как сигнал управления на мультиплексор, который определяет выбор сигналов, проходящих через мультиплексор. Например, утверждения в примере ниже

создают логику мультиплексора, которая управляет входом с и переключает входы a или b на выход переменной x:

```
if (c)
    x = a;
else
    x = b;
```

Мультиплексор с четырьмя входами описывается с применением вложенных конструкций if...else:

```
if (select[1])
    begin
        if (select[0]) out = in[3];
        else out = in[2];
    end
else
    begin
        if (select[0]) out = in[1];
        else out = in[0];
    end
end
```

7. Утверждения case

Функции, выполняемые утверждением case, подобны тем, что выполняются в утверждениях if...else. Утверждение case позволяет выполнить многократные ветвления цепей в логике на значениях выражения. Утверждение case состоит из ключевого слова case, сопровождаемого выражением в круглых скобках, кроме того, в конструкцию входят одно или более значений для case (и утверждения, с ним связанные, которые будут выполняться), сопровождаемые ключевым словом endcase. Значения для case состоят из выражения (обычно это простые константы) или списка выражений, отделенных запятыми и сопровождаемых двоеточием «:».

При выполнении выражения case производится сравнение значения выражения, следующего после ключевого слова case, по очереди с каждым из вариантов значений, находящихся в конструкции case. Когда выражение равно значению в одном из вариантов в конструкции, то это условие оценивается как истина (true). В каждом из вариантов значений case может быть записано несколько выражений, отделенных друг от друга запятыми. Когда используются несколько выражений, то условие будет истинным в том случае, если любое из выражений в теме case соответствует

выражению после ключевого слова case. Первое же значение, которое будет оцениваться как истина в case, определит выбор пути. Все последующие значения для case игнорируются, даже если они истинны. Если ни одно из значений, находящихся в case, не выбрано, то никаких действий не будет выполнено.

Можно определить выбор значения в case по умолчанию (default) и соответственно приписать ему выражения, которые будут использоваться, когда не выбрано никакое другое значение в case:

```
input [1:0] a;
always @(a or w or x or y or z) begin
  case (a)
    2'b11: b = w ;
    2'b10: b = x ;
    2'b01: b = y ;
    2'b00: b = z ;
  endcase
end
```

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 K155TM2 (рис. 2.1).

K155TM2 – 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 (табл. 2.1).

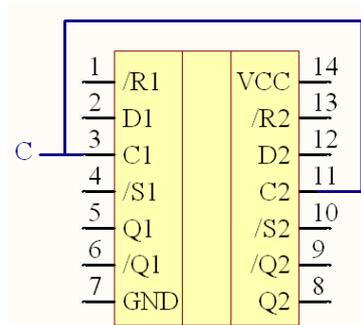


Рис. 2.1. УГО микросхемы К155ТМ2

Таблица 2.1

Таблица истинности К155ТМ2

Режим работы	Вход				Выход	
	S <input type="checkbox"/>	R <input type="checkbox"/>	C	D	Q	Q <input type="checkbox"/>
Асинхронная установка	0	1	x	x	1	0
Асинхронный сброс	1	0	x	x	0	1
Неопределенность	0	0	x	x	1	1
Загрузка 1	1	1	↑	1	1	0
Загрузка 0	1	1	↑	0	0	1

Создадим новый проект и присвоим ему имя «lab2».

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

- 1) Выберем «Verilog Module» в качестве типа исходного файла.
- 2) В поле «File name» введем «lab2».
- 3) Установим свойства портов ввода-вывода проектируемого устройства, как показано на рис. 2.2.

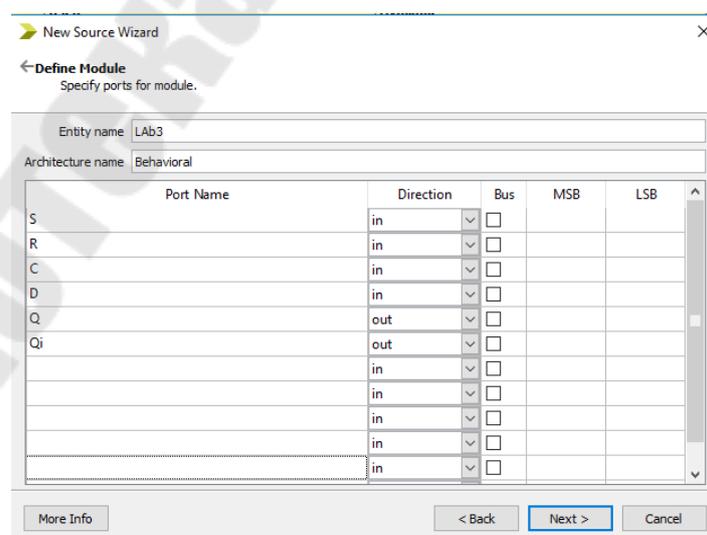


Рис. 2.2. Настройка портов ввода-вывода

В результате получим файл описания устройства с незаполненным блоком module.

4) Чувствительность блока always будет зависеть от трех входов: «C», «S», «R». При этом входы «S», «R» будут запускать блок по спаду сигнала, т.к. асинхронный сброс и установка происходят по негативному уровню.

```
always @(posedge C or negedge S or negedge R)
```

5) Опишем работу микросхемы K155TM2:

```
module Lab2(  
    input S,  
    input R,  
    input C,  
    input D,  
    output reg Q,  
    output reg Qi  
);  
  
always @(posedge C or negedge S or negedge R)  
begin  
    if(S==0 & R==0)  
        begin  
            Q=1'b1;  
            Qi=1'b1;  
        end  
    else if (~S)  
        begin  
            Q=1'b1;  
            Qi=1'b0;  
        end  
    else if (~R)  
        begin  
            Q=1'b0;  
            Qi=1'b1;  
        end  
    else  
        begin  
            Q=D;  
            Qi=!D;  
        end  
end  
  
endmodule
```

6) Сохраните файл.

7) Проверьте его на наличие синтаксических ошибок.

2. Создание набора тестовых воздействий

Этап функционального моделирования (Simulate Behavioral Verilog 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..

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:

- 1) Выбрать в окне описания проекта файл lab2.
- 2) Для создания нового набора тестовых воздействий выбрать пункт меню «Project» → «New Source».
- 3) В открывшемся окне выбрать тип исходного файла «Verilog Test Fixture» и ввести имя для создаваемого набора test1 в поле «File Name».
- 4) Нажать кнопку «Next >».
- 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.
- 6) Нажать кнопку «Next >».
- 7) Перейти в режим поведенческого моделирования так, как показано на рис. 2.3.

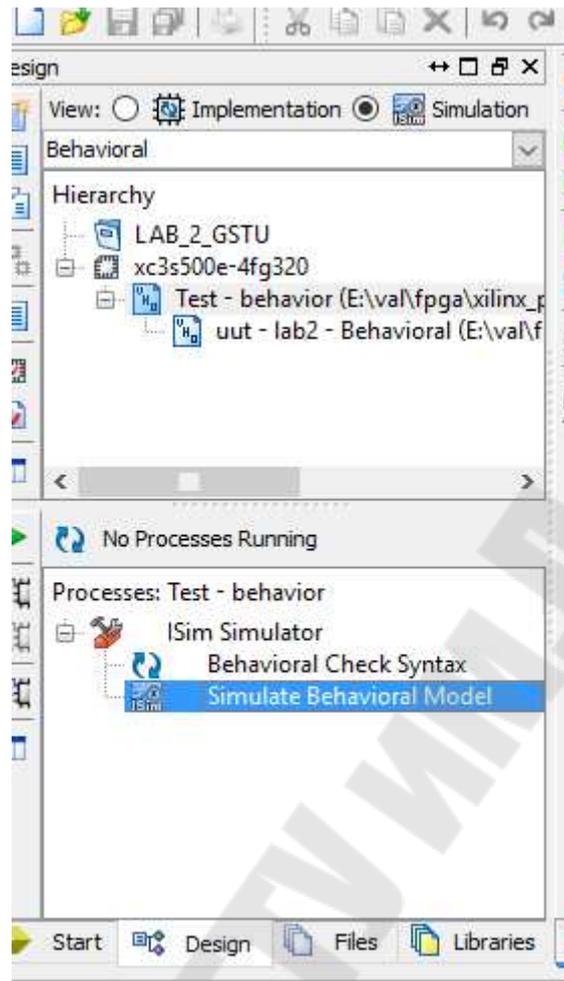


Рис. 2.3. Выбор режима поведенческого моделирования

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

9) В появившемся файле Test1 (Initialize Timing) необходимо указать поведение тестовых воздействий, для проведения симуляции.

Поведенческие модели в Verilog содержат процедурные операторы, которые управляют симуляцией и манипулируют переменными типов данных. Все эти утверждения содержатся в процедурах. Во время моделирования поведенческой модели все потоки, определенные операторами «always» и «initial», начинаются вместе во время моделирования «ноль». Первоначальные операторы выполняются один раз, а операторы always выполняются многократно. В этой модели регистровые переменные a и b инициализируются в двоичные 1 и 0 соответственно во время моделирования «ноль»:

```

module Lab2Verilog;
  // Inputs
  reg S;
  reg R;
  reg C;
  reg D;
  // Outputs
  wire Q;
  wire Qi;
  // Instantiate the Unit Under Test (UUT)
  Lab2Verilog uut (
    .S(S),
    .R(R),
    .C(C),
    .D(D),
    .Q(Q),
    .Qi(Qi)
  );
  initial begin
    // Initialize Inputs
    S = 0;
    R = 0;
    C = 0;
    D = 0;
  end
  always
    #10 C = !C;
  always
    #25 D = !D;

  initial begin
    #30 S=1'b1;
    #20 S=1'b0;
    R=1'b1;
    #20 S=1'b1;
    #130 R=1'b0;
  end
endmodule

```

Сигналы «S», «R», «C», «D» описываются типом reg, выходы Q, Qi – типом wire. Оператор initial содержит последовательный блок операторов, в котором проводятся инициализация нулевым значением входов «S», «R», «C», «D». В блоке always задана смена сигнала «C» с периодом 10 нс, сигнала D – 25нс. После чего проводится однократное переключение сигналов «S» и «R» в блоке initial.

3. Создание тестового модуля

Создание набора ожидаемых значений выходных сигналов завершает процесс создания набора тестовых воздействий. Эта

4. Проверка правильности работы устройства

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

1) 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:

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

```
module Lab2(  
    input S,  
    input R,  
    input C,  
    input D,  
    output reg Q,  
    output reg Qi  
);  
integer int_count=0;  
reg C_new;  
always @(posedge C)  
begin  
    if (int_count==25000000)  
        begin  
            int_count <= 0;  
            C_new <= 1;  
        end  
    else  
        begin  
            int_count <= int_count + 1;  
            C_new <= 0;  
        end  
end  
always @(posedge C_new or negedge S or negedge R)  
begin  
    if(S==0 & R==0)  
        begin  
            Q=1'b1;  
            Qi=1'b1;  
        end  
    else if (S==0)  
        begin  
            Q=1'b1;  
            Qi=1'b0;  
        end  
    else if(R==0)  
        begin  
            Q=1'b0;  
            Qi=1'b1;  
        end  
end
```

```

else
  begin
    Q=D;
    Qi=!D;
  end
endendmodule

```

5. Назначение выводов и загрузка конфигурации в Spartan 3

Далее необходимо поставить в однозначное соответствие линии портов проектируемого устройства выводам микросхемы ПЛИС, для этого необходимо:

1) Указать адреса входных сигналов так, как показано на рис. 2.5.

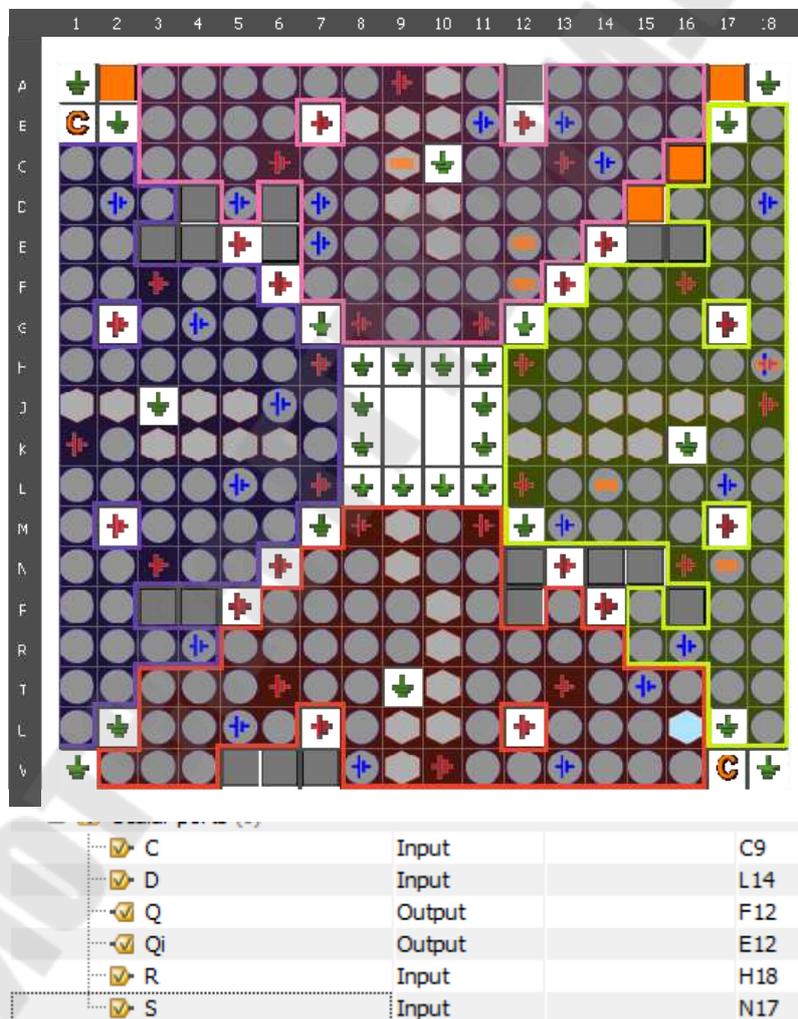


Рис. 2.5. Схема расположения выводов ПЛИС

- 2) Запустить процесс реализации проекта.
- 3) Удостовериться, что все выводы микросхемы правильно поставлены в соответствие портам проектируемого устройства.
- 4) Запустить программу iMPACT.
- 5) Загрузить в устройство xc3s500e файл битовой последовательности lab2.bit.

Результатом правильной работы устройства будет соответствие свечения светодиодов табл. 2.1.

Задание для самостоятельной работы

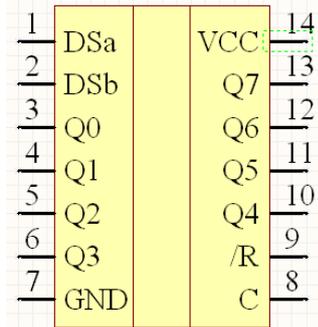
В ходе лабораторной работы требуется:

- 1) повторить разработку D-триггера, провести поведенческое моделирование;
- 2) разработать описание устройства, соответствующего номеру варианта, и поставить в соответствие линии портов устройства выводам микросхемы в соответствии с табл. 2.2;

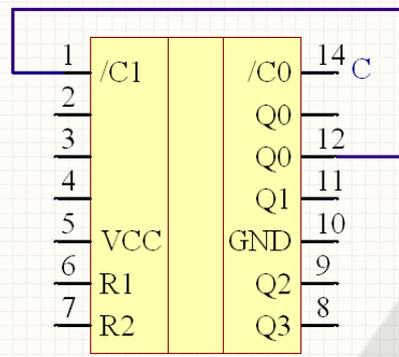
Таблица 2.2

Варианты заданий для самостоятельной работы

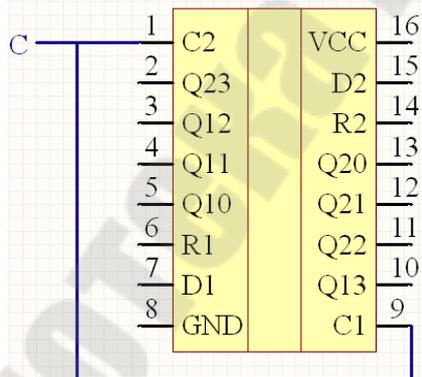
УГО микросхемы	Примечания
	<p>K155TM8 – четыре D-триггера, имеющие общие входы синхронного сброса /R и тактового запуска C, а также имеются выходы Q и /Q. Информацию от параллельных входов данных D1-D4 можно загрузить, если на вход /R подать напряжение высокого уровня.</p>



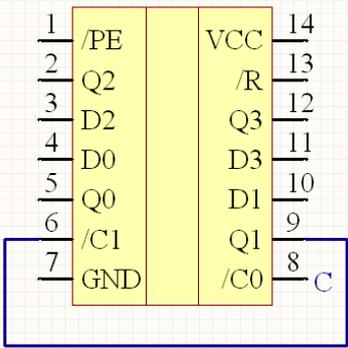
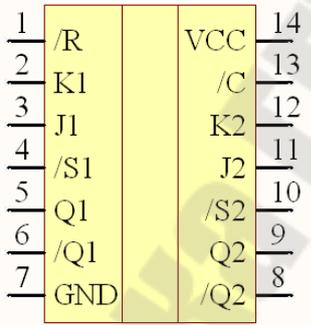
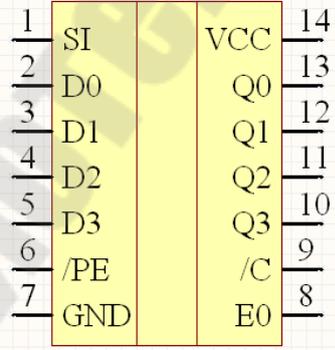
K555IP8 – восьмиразрядный сдвиговый регистр с последовательным входом и параллельными выходами. Регистр имеет асинхронный сброс (вход /R) и два входа для последовательных данных DSa и DSb (логика И). Поданные через эти входы данные сдвигаются на одну позицию вправо согласно каждому положительному перепаду импульса, пришедшего на тактовый вход C.

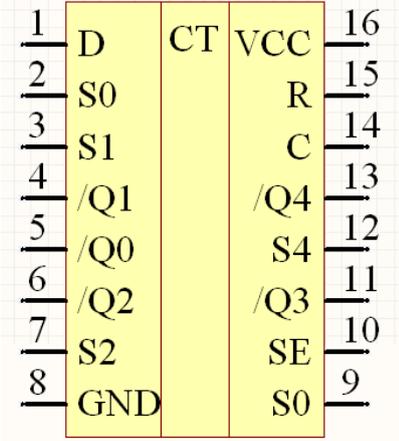
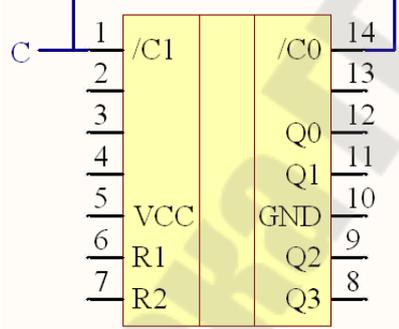


K155IE4 – четырехразрядный двоичный счетчик-делитель на 2, на 6 и на 12. Чтобы построить счетчик с модулем деления 12, требуется соединить делители на 2 и на 6, замкнув выводы 12 и 1. На вход /C0 дается входная частота f , на выходе Q3 получается последовательность прямоугольных импульсов с частотой $f/12$. Тактовые запускающие перепады – отрицательные.



K561IP2 – два независимых четырехразрядных регистра сдвига. Данные в регистр вводятся через последовательный вход D. Регистр имеет вход тактовых импульсов C, причем данные принимаются от входа D первого триггера и сдвигаются на один такт вправо после каждого положительного тактового перепада на входе C. Сброс в нуль данных на выходе Q регистра получится, если на вход асинхронного сброса R подать напряжение высокого логического уровня.

	 <p>Pinout diagram for K531IE14 showing pins 1-7 on the left and 14-8 on the right. Pin 14 is VCC, pin 8 is GND. Pins 2-7 are labeled /PE, Q2, D2, D0, Q0, /C1, GND. Pins 13-9 are labeled /R, Q3, D3, D1, Q1, /C0. Pin 13 is VCC, pin 8 is GND. Pin 14 is VCC, pin 8 is GND. Pin 13 is /R, pin 12 is Q3, pin 11 is D3, pin 10 is D1, pin 9 is Q1, pin 8 is /C0.</p>	<p>K531IE14 – 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 выходы 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>
	 <p>Pinout diagram for K531TB11 showing pins 1-7 on the left and 14-8 on the right. Pin 14 is VCC, pin 8 is GND. Pins 2-7 are labeled /R, K1, J1, /S1, Q1, /Q1, GND. Pins 13-9 are labeled /C, K2, J2, /S2, Q2, /Q2. Pin 13 is VCC, pin 8 is GND. Pin 14 is VCC, pin 13 is /C, pin 12 is K2, pin 11 is J2, pin 10 is /S2, pin 9 is Q2, pin 8 is /Q2.</p>	<p>K531TB11 – два JK триггера. Оба триггера микросхемы имеют по две общие цепи управления: тактовый вход /C и вход сброса /R. Входы J и K могут работать, если на асинхронных входах /S и /R присутствуют напряжения высокого уровня.</p>
	 <p>Pinout diagram for K555IP16 showing pins 1-7 on the left and 14-8 on the right. Pin 14 is VCC, pin 8 is GND. Pins 2-7 are labeled SI, D0, D1, D2, D3, /PE, GND. Pins 13-9 are labeled Q0, Q1, Q2, Q3, /C, E0. Pin 14 is VCC, pin 8 is GND. Pin 13 is Q0, pin 12 is Q1, pin 11 is Q2, pin 10 is Q3, pin 9 is /C, pin 8 is E0.</p>	<p>K555IP16 – четырехразрядный 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</p>

		<p>SI. Цифровое слово сдвигается вправо от Q0 к Q1 далее к Q2 и Q3 синхронно с каждым отрицательным перепадом на тактовом входе /C.</p>																																								
	 <p>Pinout diagram for K561IE19:</p> <table border="1"> <tr><td>1</td><td>D</td><td>CT</td><td>VCC</td><td>16</td></tr> <tr><td>2</td><td>S0</td><td></td><td>R</td><td>15</td></tr> <tr><td>3</td><td>S1</td><td></td><td>C</td><td>14</td></tr> <tr><td>4</td><td>/Q1</td><td></td><td>/Q4</td><td>13</td></tr> <tr><td>5</td><td>/Q0</td><td></td><td>S4</td><td>12</td></tr> <tr><td>6</td><td>/Q2</td><td></td><td>/Q3</td><td>11</td></tr> <tr><td>7</td><td>S2</td><td></td><td>SE</td><td>10</td></tr> <tr><td>8</td><td>GND</td><td></td><td>S0</td><td>9</td></tr> </table>	1	D	CT	VCC	16	2	S0		R	15	3	S1		C	14	4	/Q1		/Q4	13	5	/Q0		S4	12	6	/Q2		/Q3	11	7	S2		SE	10	8	GND		S0	9	<p>Микросхема K561IE19 – пятиразрядный синхронный счетчик по схеме Джонсона. От каждого триггера счетчика сделан инверсный выход $\overline{Q0} \text{—} \overline{Q4}$. 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.</p>
1	D	CT	VCC	16																																						
2	S0		R	15																																						
3	S1		C	14																																						
4	/Q1		/Q4	13																																						
5	/Q0		S4	12																																						
6	/Q2		/Q3	11																																						
7	S2		SE	10																																						
8	GND		S0	9																																						
	 <p>Pinout diagram for K155IE5:</p> <table border="1"> <tr><td>1</td><td>/C1</td><td></td><td>/C0</td><td>14</td></tr> <tr><td>2</td><td></td><td></td><td>Q0</td><td>13</td></tr> <tr><td>3</td><td></td><td></td><td>Q1</td><td>12</td></tr> <tr><td>4</td><td></td><td></td><td>Q2</td><td>11</td></tr> <tr><td>5</td><td>VCC</td><td></td><td>GND</td><td>10</td></tr> <tr><td>6</td><td>R1</td><td></td><td>Q3</td><td>9</td></tr> <tr><td>7</td><td>R2</td><td></td><td></td><td>8</td></tr> </table>	1	/C1		/C0	14	2			Q0	13	3			Q1	12	4			Q2	11	5	VCC		GND	10	6	R1		Q3	9	7	R2			8	<p>K155IE5 – четырехразрядный 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>					
1	/C1		/C0	14																																						
2			Q0	13																																						
3			Q1	12																																						
4			Q2	11																																						
5	VCC		GND	10																																						
6	R1		Q3	9																																						
7	R2			8																																						

- 3) создать набор тестовых воздействий;
- 4) создать тестовый модуль;
- 5) провести функциональное моделирование;
- 6) разместить разработанное устройство в кристалле;
- 7) загрузить полученную конфигурацию в ПЛИС;
- 8) проверить правильность работы программы.

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. Приведите пример процедурных назначений.
2. Каким образом в языке Verilog выполняется инициализация переменных?
3. Для чего предназначены блоки always?
4. Что может выступать в качестве выражения события (event)?
5. Для чего и каким образом применяются утверждения case?

Исследование среды симуляции ModelSim

Цель работы

Ознакомиться с интерфейсом среды симуляции Modelsim, научиться создавать проект, проводить компиляцию, создавать TCL-макросы и выполнять симуляцию.

Основные теоретические сведения

Среда моделирования ModelSim предназначена для проверки работоспособности проекта, описанного на одном из языков описания аппаратуры (HDL). Она включает в себя средства создания проекта, создания и редактирования исходных файлов проекта, компилятор, моделирующую программу и средства визуализации результатов моделирования (графический редактор и пр.). ModelSim поддерживает работу с тестовыми файлами на HDL (test-bench) или на языке Tcl (командный язык для создания управляющих файлов).

1. Отличительные особенности пакета

Полная поддержка всех основных стандартов языков VHDL и Verilog и их расширений: IEEE VHDL Std 1076–1987 и IEEE VHDL Std 1076–1993, IEEE 1164-1993 Standard Multivalued Logic System for VHDL Interoperability, IEEE 1076.2-1996 Standard VHDL Mathematical Packages, IEEE VITAL Std 1076.4-1995, VITAL 2000 и IEEE Verilog Std 1364-1995. Полная совместимость со спецификациями 1.0 – 3.0 стандартного формата описания задержек SDF (Standard Delay Format) обеспечивает возможность обратной аннотации временных параметров. Кроме того, пакет ModelSim удовлетворяет требованиям VCD (Value Change Dump) по формированию стандартных выходных векторов для VHDL и Verilog и PLI (Programming Language Interface).

Единое ядро моделирования пакета SKS (Single Kernel Simulation), обеспечивающее возможность полной отладки «смешанных» проектов, которые одновременно содержат модули, написанные на VHDL и Verilog. Для реализации этого режима предусмотрена специальная лицензия (редакция пакета

ModelSim SE/PLUS), разрешающая совместное VHDL- и Verilog-моделирование.

Поддержка библиотек всех ведущих фирм-изготовителей как программируемых логических интегральных схем (ПЛИС) семейств FPGA (Field Programmable Gate Array) и CPLD (Complex Programmable Logic Device), так и ASIC (Application-Specific Integrated Circuit), предоставляющая разработчику широкие возможности сравнения различных платформ и выбора оптимальной для реализации проектируемой системы. Сертифицированные изготовителями библиотеки обеспечивают максимальную достоверность результатов моделирования.

Высокая скорость компиляции и моделирования (полнофункциональных версий), обеспечивающая минимальное время отладки систем различного уровня сложности. Одним из главных факторов, повышающих производительность, является использование принципа оптимизированной прямой компиляции. В соответствии с этим принципом исходные VHDL- или Verilog-описания компилируются в машинно-независимый объектный код, исполняемый на любой поддерживаемой платформе.

Открытая архитектура программных средств ModelSim, обеспечивающая тесную интеграцию с пакетами САПР «третьих» фирм. Пользователь может выполнять этапы моделирования фактически в рамках основной системы проектирования, в среде которой осуществляется процесс разработки устройства. Средства управления пользовательским интерфейсом Tcl (Tool command language) и Tk (Tool kit) предоставляют возможность организации прямого доступа к моделирующему ядру ModelSim, загрузки информации о выполнении процесса моделирования и его результатов в среду используемой САПР и управления работой системы ModelSim через интерфейс применяемых средств проектирования. Возможен также и противоположный метод интеграции с программным обеспечением других фирм, когда в качестве основной системы используется ModelSim, интерфейсная оболочка которой адаптируется для управления выбранным пакетом САПР.

Наличие защищенного режима компиляции моделей, гарантирующего выполнение требований, предъявляемых к охране интеллектуальной собственности, к которой относятся коммерчески распространяемые модули (IP Core). В обычном режиме разрешается

полная отладка моделей с доступом к исходному коду и внутренней структуре объекта. Если интеллектуальные продукты распространяются производителем в скомпилированном виде без передачи исходного кода, внутренней структуры и переменных, то следует использовать защищенный режим компиляции моделей.

При этом в процессе моделирования отображается состояние только внешних (интерфейсных) сигналов объектов интеллектуальной собственности, а контроль поведения их внутренних сигналов и процессов недоступен пользователю.

Расширенные отладочные возможности пакета, позволяющие пользователю не только быстро отыскать и идентифицировать ошибки, но и сразу же устранить причины их возникновения. После обнаружения ошибки достаточно перейти из режима отладки в режим редактирования исходного кода, внести соответствующие изменения в текст описания и после сохранения файла выполнить повторную компиляцию данного модуля. Все перечисленные операции производятся в процессе текущего сеанса работы системы моделирования и требуют минимальных временных затрат. Динамическое обновление окон системы обеспечивает возможность быстрого и легкого перемещения по базе данных проекта.

Наличие встроенного индикатора активности кода, не только повышающего эффективность отладки проекта, но и позволяющего быстро создавать более полные и надежные тестовые последовательности. Этот инструмент предоставляет возможность проследить строки исходного кода, которые не «активизировались» в процессе моделирования, и вывести в графической форме соответствующий отчет обо всех файлах проекта. Индикатор активности кода может быть использован как на уровне отдельного блока, так и для всей системы в целом.

Использование встроенного анализатора производительности, позволяющего повысить скорость моделирования за счет обнаружения в проекте и последующего устранения факторов, оказывающих отрицательное влияние на быстродействие этого процесса. С помощью этого инструмента можно получить информацию о библиотечных элементах, обработка которых требует значительных временных затрат, фрагментах исходного кода, написанных нерационально с точки зрения скорости его исполнения, неиспользуемых сигналах в выводимых списках, избыточном коде в тестовых последовательностях. Исключение перечисленных

элементов из проекта позволяет резко снизить общее время моделирования.

Возможность работы в различных режимах, в том числе и пакетном. Разработав и отладив некоторый сценарий моделирования в интерактивном режиме, можно оформить его для последующего использования в виде пакетного командного файла.

Доступный для разработчика визуальный пользовательский интерфейс и наличие подробной справочной системы, сокращающие время освоения пакета моделирования. Средства управления пользовательским интерфейсом Tcl и Tk позволяют выполнить настройку его элементов (панелей кнопок, меню) в соответствии с требованиями каждого конкретного пользователя.

Порядок выполнения работы

1. Запустите программу ModelSim с помощью команды Пуск – Все программы – Modelsim PE 10.4 – Modelsim.
2. Закройте окно IMPORTANT Information с помощью кнопки Close. Откроется основное рабочее окно среды моделирования (рис. 3.1):

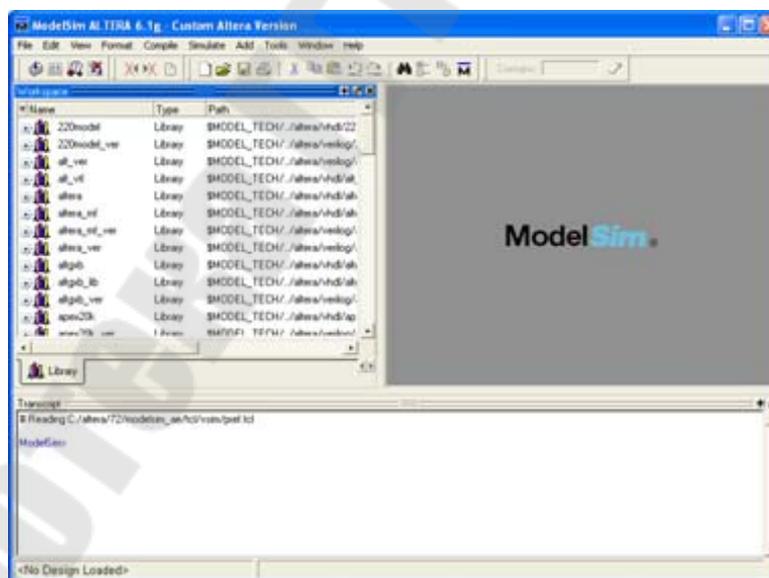


Рис. 3.1. Основное рабочее окно среды моделирования

3. Создайте новый проект. Для этого в меню File выберите команду New – Project.. Откроется диалоговое окно Create Project:

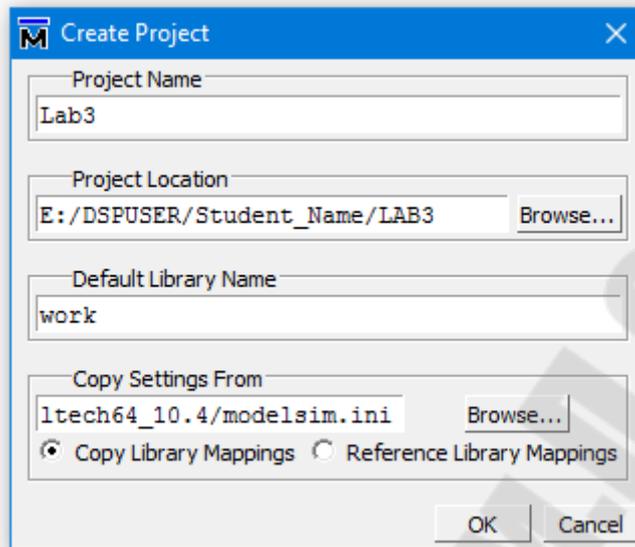


Рис. 3.2. Вид диалогового окна Create Project

Укажите в нем имя создаваемого проекта Lab3, рабочую директорию проекта. Имя текущей библиотеки проекта оставьте по умолчанию (work). Нажмите ОК.

4. В открывшемся диалоговом окне выберите команду добавления существующих исходных файлов в проект – Add Existing File.

5. В открывшемся диалоговом окне Add file to Project укажите файл лабораторной работы №2. Нажмите кнопку Открыть. Подтвердите выбор командой ОК. Закройте окно Add items to the Project.

6. В окне рабочей области проекта (Workspace, закладка Project) появится выбранный файл. Двойное нажатие левой кнопки мыши на данном файле позволяет открыть текстовый редактор для просмотра и редактирования исходного файла:

7. Скомпилируйте проект. Для этого выберите в меню Compile команду Compile All. Информация о результате компиляции появится в окне сообщений (Transcript). Если компиляция завершилась успешно, изменится вид значка статуса возле имени файла.

8. Теперь перейдите в режим моделирования. Для этого в меню Simulate выберите команду Start Simulation. В открывшемся диалоговом окне Start Simulation укажите файл верхнего уровня иерархии для моделирования (файл LAB2 из рабочей библиотеки work) (рис.3.3).

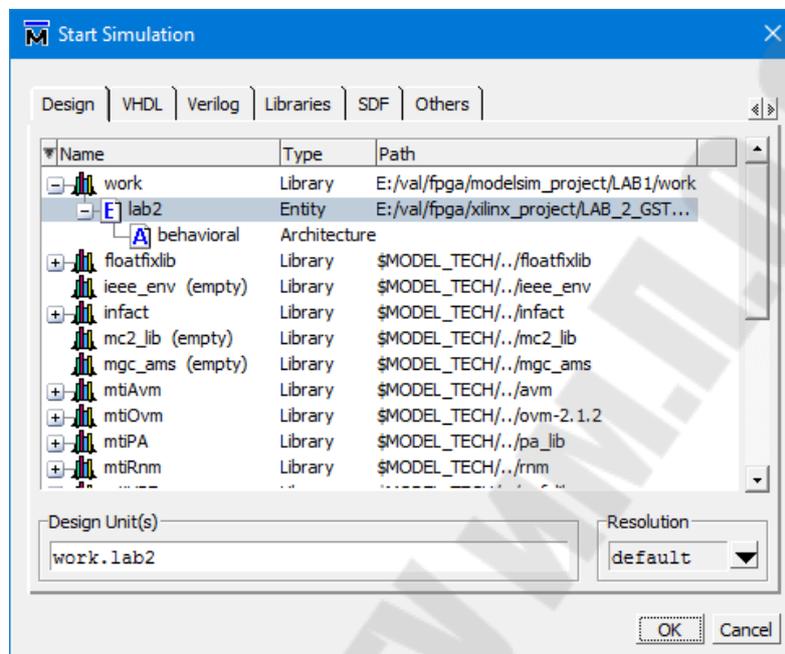


Рис. 3.3. Окно Start Simulation

Отключите опцию Enable optimization. Укажите шаг моделирования (Resolution) равный ps (одна пикосекунда). Нажмите ОК.

9. Внешний вид среды ModelSim изменился. Она переключилась в режим моделирования. В окне рабочей области проекта (Workspace) появилась закладка sim (список доступных для моделирования объектов – исходный файл, входящие в его состав функции и процессы). Открылось окно Objects – доступные для просмотра объекты (входные, выходные и внутренние сигналы).

10. Сделайте окно Object активным (нажмите левой кнопкой мыши на заглавии окна). В меню Add выберите команду To Wave – All items in region. Данная команда позволяет открыть графическое окно и добавить в него для просмотра все сигналы, присутствующие в данном модуле. (рис 3.4).

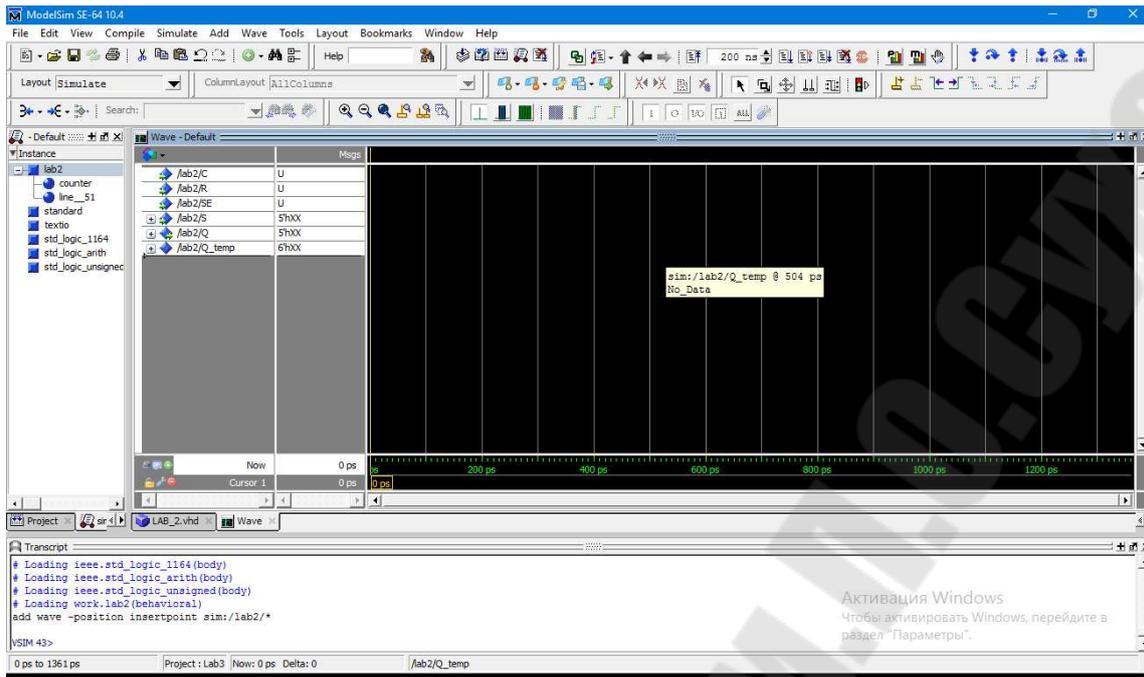


Рис. 3.4. Вид окна Wave

11. Для того чтобы приступить к моделированию, необходимо создать файл с входными тестовыми сигналами (test-bench). Для этого перейдите в окне рабочей области проекта (Workspace) к закладке Project. В меню File выберите команду New, тип файла .do. Сохраните файл под именем Stim.do (на языке Tcl). В окне текстового редактора откроется данный файл.

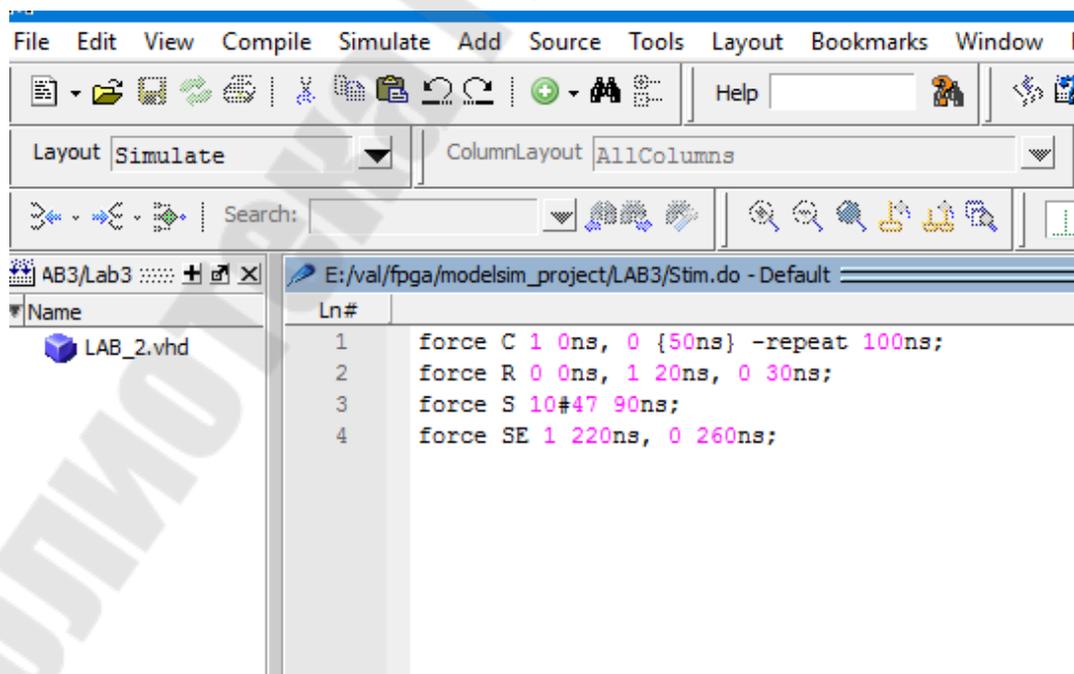


Рис. 3.5. Вид окна редактора файла Stim.do

Рассмотрим подробнее формат команды формирования входных воздействий:

1) `force clr 1 0ns, 0 50ns`; – данная команда говорит о том, что сигналу `clr` необходимо присвоить значение логической «1» на 0 наносекунде, а затем – логического «0» на 50 наносекунде. Последнее значение будет сохраняться до конца времени моделирования.

2) `force clk 0 0ns, 1 {50ns} –repeat 100ns`; – данная команда говорит о том, что сигналу `clk` необходимо присвоить значение логического «0» на 0 наносекунде, а затем – логической «1» на 50 наносекунде. Данная последовательность будет периодически повторяться с периодом 100 наносекунд до конца времени моделирования. Обратите внимание – значение времени перед опцией `–repeat` обязательно указывается в фигурных скобках.

3) `force ina 10#0 0ns, 10#2 30ns`; – данная команда присваивает следующие значения входному сигналу `ina` : на 0 наносекунде – 0 (в десятичном формате), на 30 наносекунде – 2 (в десятичном формате). Последнее значение будет сохраняться до конца времени моделирования. Первая цифра перед знаком `#` указывает на формат числа (2 – двоичный формат, 10 – десятичный формат, 16 – шестнадцатичный формат), а цифра за знаком – его значение.

12. Для подключения тестового файла к проекту в меню `Tools` выберите команду `TCL – Execute Macro...` и укажите файл `Stim.do`.

13. Запуск моделирующей программы осуществляется из меню `Simulate` командой `Run – Run – All`. Результат моделирования будет отображаться в графическом окне (рис. 3.6):

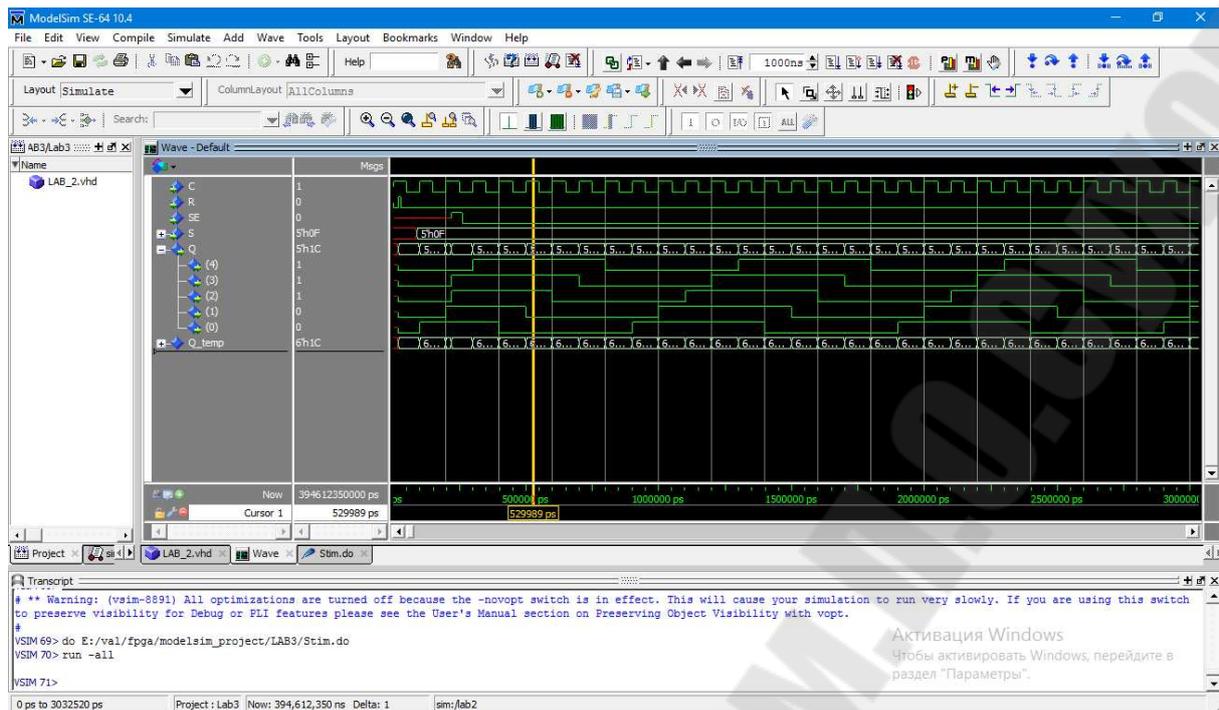


Рис 3.6. Результаты моделирования

Проверьте правильность работы устройства по результатам моделирования.

14. Управление моделированием может осуществляться с помощью команды `run` из окна сообщений. Формат команды:

`run 100` – выполнение 100 шагов моделирования;

`run 1000ns` – выполнение моделирования до отметки времени, отстоящей на 1000 наносекунд от текущей позиции;

`run @ 300` – выполнение моделирования до отметки времени, отстоящей на 300 шагов моделирования от нулевой позиции;

`run @ 1500ns` – выполнение моделирования до отметки времени, отстоящей на 1500 наносекунд от нулевой позиции;

`restart -force` – сброс результатов моделирования.

Обратите внимание, что для повторного моделирования после сброса результатов необходимо снова подключить к проекту файл с входными тестовыми сигналами.

14. Для выхода из режима моделирования в меню `Simulate` выберите команду `End Simulation`.

Задание для самостоятельной работы

В ходе лабораторной работы требуется:

- 1) создать проект в программе моделирования ModelSim;
- 2) подключить файл описания, разработанный в лабораторной работе №2;
- 3) разработать файла TCL-макросов Stim.do, необходимый для проведения моделирования;
- 4) провести моделирование в программе ModelSim;

Содержание отчета

Отчет должен содержать цель работы, содержание разработанного файла TCL-макросов Stim.do, результаты моделирования, выводы.

Контрольные вопросы

1. Для чего предназначена программа ModelSim?
2. Каким образом запустить режим моделирования?
3. Что отображается в окне Wave?
4. Что описывается на языке Tcl?
5. Какой формат команды формирования входных воздействий?
6. Расшифруйте макрос: force ina 10#1 0ns, 10#5 50ns.

Лабораторная работа № 4

Реализация конечных автоматов на основе отладочной платы Spartan-3E Starter Kit

Цель работы

Ознакомиться с понятием «конечный автомат». Получить навыки в проектировании автоматов Мура. Разработать и проверить работоспособность конечного автомата при помощи отладочной платы Spartan-3E Starter Kit.

Основные теоретические сведения

Конечный автомат – это общее название последовательностных схем. Слово «тактируемый» указывает на тот факт, что элементы памяти в конечном автомате (триггеры) имеют тактовый вход. Слово «синхронный» означает, что все триггеры используют один и тот же тактовый сигнал. Состояние такого конечного автомата изменяется только в момент времени, когда в тактовом сигнале происходит переключающий переход или, как говорят, на очередном «такте».

На рис. 4.1 приведена общая структура тактируемого синхронного конечного автомата.

Память состояний представляет собой набор из n триггеров, в которых хранится текущее состояние автомата. Всего имеется 2^n различных состояний. Все триггеры подключены к общему источнику тактового сигнала, который позволяет им изменять состояние на каждом такте тактового сигнала.

Следующее состояние конечного автомата определяется логикой переходов и является функцией текущего состояния и входного воздействия. Выходные сигналы определяются выходной логикой и также зависят от текущего состояния и входного воздействия. Оба блока логики являются строго комбинационными схемами. Последовательностная схема, выход которой зависит как от состояния, так и от входа, называется автоматом Мили (рис. 4.1, б). В некоторых приложениях выход зависит только от состояния. Такая схема называется автоматом Мура (рис. 4.1, а).

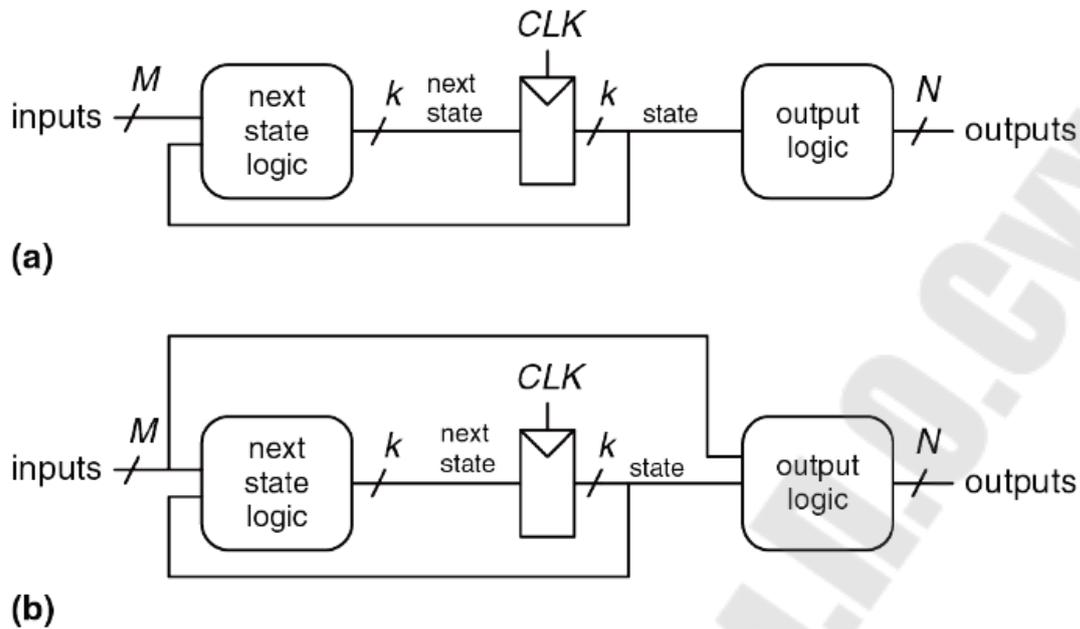


Рис. 4.1. Структура тактируемого синхронного конечного автомата:
a – автомат Мура; *b* – автомат Мили

Очевидно, что единственное различие между этими двумя моделями конечных автоматов заключается в том, как вырабатываются выходные сигналы. На практике многие конечные автоматы могут иметь выходы типа Мили и выходы типа Мура, т.е. иметь смешанную структуру.

Порядок выполнения работы

1. Постановка задачи

1. В качестве примера разработаем конечный автомат для простейшей задачи – при нажатии на кнопку на 1 секунду загорается светодиод, и через 1 секунду гаснет. Данная задача имеет множество решений, но применение принципа конечного автомата позволит получить наиболее масштабируемый результат, который можно будет видоизменить и использовать для решения других задач.

2. Запишем алгоритм решения задачи в виде кода на языке Си (такое решение более лаконично, чем блок-схемы):

```

while(1)
{
    if(button)
    {
        led = 1;
        delay();
        led = 0;
        delay();
    }
}

```

3. Следующим этапом необходимо выделить состояния, в которых находится система.

- основное состояние можно охарактеризовать так: система ничего не делает, только опрашивает кнопку. Назовем его "IDLE" («режим ожидания»)

- при нажатии на кнопку система переходит состояние: «светодиод горит» ("LED_ON"). В этом состоянии система должна находиться в течение 1 секунды.

- согласно условия, после секундного свечения светодиода необходимо его принудительно выключить – состояние «светодиод выключен» ("LED_OFF").

4. Система находится в трех состояниях: IDLE, LED_ON и LED_OFF. Переход из состояния IDLE в состояние LED_ON осуществляется внешним событием – нажатием кнопки. В состоянии LED_ON система пребывает определенное время – пока не сработает таймер. Запуск таймера осуществляется вместе с переходом из состояния IDLE в состояние LED_ON, также запуск осуществляется переходом из состояния LED_ON в состояние LED_OFF.

5. Изобразим вышенаписанное в виде графа переходов (рис. 4.3.).

Эллипсы показывают состояния системы, векторы – переход между ними. Над векторами указаны условия перехода (если ничего не указано, то переход безусловный).

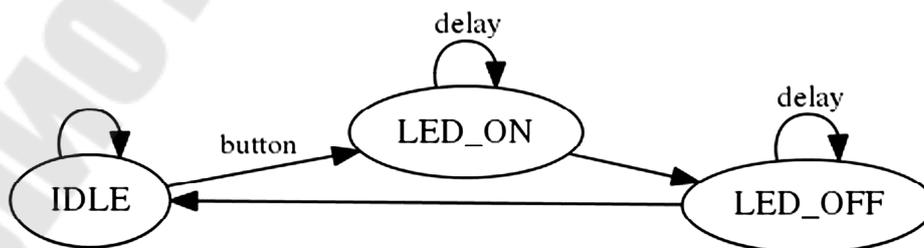


Рис. 4.3. Граф переходов

2. Реализация автомата на Verilog

6. Введем два регистра, хранящие состояние системы. Один регистр (STATE), будет хранить текущее состояние, другой (NEXT_STATE) – следующее.

7. Реализацию блока логики переходов между состояниями и блока смены состояния выполним при помощи двух операторов always.

8. Для описания логики переходов часто применяют оператор case:

```
always @*
case(state) // Выбор текущего состояния
IDLE:
  if(button) // Если кнопка нажата,
    next_state = LED_ON; // то переходим в состояние LED_ON,
  else // иначе
    next_state = IDLE; // остаемся в состоянии IDLE.
LED_ON:
  if(timer_overflow) // Если таймер переполнен,
    next_state = LED_OFF; // то переходим в состояние LED_OFF,
  else // иначе
    next_state = LED_ON; // остаемся в состоянии LED_ON.
LED_OFF:
  if(timer_overflow)
    next_state = IDLE;
  else
    next_state = LED_OFF;
default: // Для всех остальных, неописанных
  next_state = IDLE; // состояний, переходим в IDLE
endcase
```

Смысл данного блока в том, чтобы для каждого состояния (state) определить следующее состояние (next_state). Следует отметить, что в списке чувствительности always не указаны сигналы, а это значит, что во время синтеза будет создано комбинационное устройство. При описании переходов избегайте двусмысленности, должны быть отражены все варианты, каждому if должен соответствовать свой else. На случай непредвиденный, если в результате сбоя, переменная state примет какое-либо не описанное значение, в ветви default предусмотрим переход в начальное состояние.

9. Второй блок always, предназначенный для смены состояний, должен выполняться синхронно с остальной частью схемы, в наших простых примерах, для задания главного тактового сигнала используем внешний генератор на 50МГц. Помимо синхроимпульсов в список чувствительности always поместим сигнал reset –

глобальный сброс, чтобы после подачи питания устройство начало свою работу в состоянии IDLE:

```
always @(posedge reset or posedge clk_50MHz_i)
if(reset)
    state <= IDLE;
else
    state <= next_state;
```

10. Для формирования сигнала глобального сброса после включения питания воспользуемся сдвиговым регистром, разрядность которого определяет количество тактов, в течение которых произойдет сброс:

```
reg reset;
reg [3:0]rst_delay = 0;
always @(posedge clk_50MHz_i)
    rst_delay <= { rst_delay[2:0], 1'b1 };

always @*
    reset = ~rst_delay[3];
```

В результате на линии reset, после подачи питания устанавливается единица, а после трех периодов тактового генератора (задержка реализована на сдвиговом регистре rst_delay) устанавливается ноль. Получившийся сигнал рекомендуется использовать во всех модулях нашего проекта для перевода элементов с памятью (регистров, триггеров) в начальное значение.

11. За понятиями "IDLE", "LED_ON", "LED_OFF" нужно закрепить определенное числовое значение. Применим параметр localparam, т.к. его нельзя переопределить извне:

```
localparam IDLE = 2'd0;
localparam LED_ON = 2'd1;
localparam LED_OFF = 2'd2;
```

12. Из состояния IDLE в состояние LED_ON система выходит при логической единице на линии button. Ниже на диаграмме (рис. 4.4) показано, что значение next_state обновляется сразу как появился внешний сигнал, а значение state обновляется синхронно с тактовым генератором.

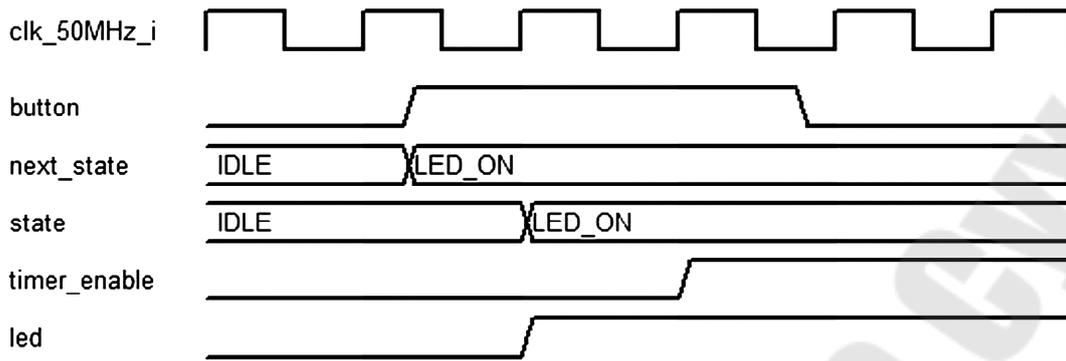


Рис. 4.4. Диаграмма переходов

13. Линия `timer_overflow` генерируется таймером и сигнализирует о переполнении. Логическая единица на линии `timer_overflow` обеспечивает переход из `LED_ON` в `LED_OFF`, а из `LED_OFF` в `IDLE`.

14. Описание схемы управления светодиодом:

```
reg led;
always @(posedge clk_50MHz_i)
if(state == LED_ON)
    led <= 1'b1;
else led <= 1'b0;
assign led_out = led;
```

15. Дополним временную диаграмму переходов (рис. 4.5):

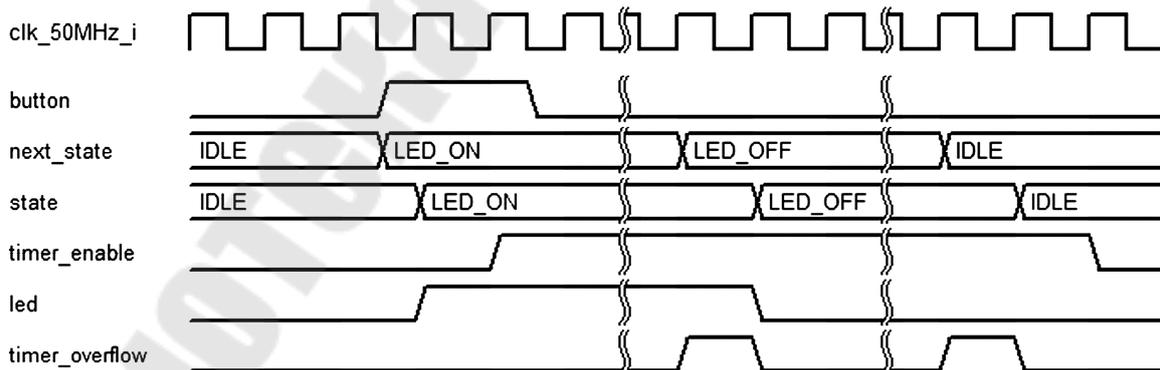


Рис. 4.5. Полная диаграмма переходов

16. При переходе в состояния LED_ON и LED_OFF нам нужно запустить таймер. Опишем функционал таймера:

– на вход его должны поступать синхриимпульсы, по которым счетчик будет менять свое значение;

– у таймера должен быть вход глобального сброса, для того, чтобы при подаче питания, он начал считать с нуля;

– у таймера должен быть выход timer_overflow, который мы использовали в переходах автомата;

– у таймера должен быть вход, разрешающий работу.

17. Опираясь только на требования, создадим пустой модуль таймера:

```
module timer
(
    input clk_i, rst_i, enable_i,
    output reg overflow_o
);
// здесь будет код модуля
endmodule
```

И в основной модуль вставим его экземпляр:

```
timer timer_inst
(
    .clk_i(clk_50MHz_i),
    .rst_i(reset),
    .enable_i(timer_enable),
    .overflow_o(timer_overflow)
);
```

Итоговое описание выглядит следующим образом:

```
module Spartan_3E(
    input clk_50MHz_i, // Генератор
    input sw_in, // Тактовая кнопка
    output led_out // Светодиод
);
// Сигнал глобального сброса
reg reset;
reg [3:0]rst_delay = 0;
always @(posedge clk_50MHz_i)
    rst_delay <= { rst_delay[2:0], 1'b1 };
always @*
    reset = ~rst_delay[3];
// Сигнал с кнопки
wire button = sw_i;
```

```

// Цепи взаимодействия с таймером
wire timer_overflow;
reg timer_enable;

// Регистры для хранения текущего и следующего состояния
reg[1:0] state, next_state;

// Описание состояний
localparam IDLE = 2'd0;
localparam LED_ON = 2'd1;
localparam LED_OFF = 2'd2;

// Логика переходов
always @*
case(state) // Выбор текущего состояния
IDLE:
    if(button) // Если кнопка нажата,
        next_state = LED_ON; // то переходим в состояние LED_ON,
    else // иначе
        next_state = IDLE; // остаемся в состоянии IDLE.
LED_ON:
    if(timer_overflow) // Если таймер переполнен,
        next_state = LED_OFF; // то переходим в состояние LED_OFF,
    else // иначе
        next_state = LED_ON; // остаемся в состоянии LED_ON.
LED_OFF:
    if(timer_overflow)
        next_state = IDLE;
    else
        next_state = LED_OFF;
default: // Для всех остальных, неописанных
    next_state = IDLE; // состояний, переходим в IDLE
endcase

// Смена состояния.
always @(posedge reset or posedge clk_50MHz_i)
if(reset)
    state <= IDLE;
else
    state <= next_state;

// Управляем светодиодом
assign led_o[0] = (state == LED_ON)? 1'b1 : 1'b0;

// Управляем таймером
always @(posedge clk_50MHz_i)
if((state == LED_ON) | (state == LED_OFF))
    timer_enable <= 1;
else timer_enable <= 0;

// Экземпляр таймера
timer timer_inst
(
    .clk_i(clk_50MHz_i),
    .rst_i(reset),
    .enable_i(timer_enable),
    .overflow_o(timer_overflow)
);

endmodule

```

18. Провести моделирование проекта в любой среде симуляции. Получить результаты, аналогичные диаграммам рис. 4.4 и рис. 4.5

19. Скомпилировать проект. Провести загрузку конфигурации в ПЛИС. В качестве проверки работоспособности провести тест: после нажатия кнопки светодиод должен загореться и потухнуть. При длительном удержании кнопки светодиод должен мерцать.

Задание для самостоятельной работы

В ходе лабораторной работы требуется:

1) Доработать таймер, разработанный при выполнении лабораторной работы №2, так чтобы он соответствовал требованиям п.16.

2) Получить у преподавателя и реализовать усложненный конечный автомат с большим числом состояний.

Примеры задач:

- время свечения должно быть отлично от времени затухания;
- при повторном нажатии кнопки загорается второй светодиод;
- время свечения зависит от количества нажатий и т.п.

Содержание отчета

Отчет должен содержать цель работы, краткие теоретические сведения по теме работы, текст разработанной конфигурации, результаты моделирования, выводы.

Контрольные вопросы

1. Что такое конечный автомат?
2. Особенности конечного автомата Мили.
3. Особенности конечного автомата Мура.
4. Для чего нужен блок «состояние системы»?
5. Для чего нужен блок «логика переходов»?

Реализация аналого-цифрового преобразования на основе отладочной платы Spartan-3E Starter Kit

Цель работы

Разработать описание модуля для работы с АЦП и масштабирующим усилителем, реализовать его на ПЛИС, а также изучить технические характеристики микросхем АЦП.

Основные теоретические сведения

Узел АЦП, установленный на отладочной плате Spartan-3E Starter Kit, включает в себя входной разъем J7, двухканальный программируемый предварительный усилитель и двухканальный АЦП (рис. 5.1).

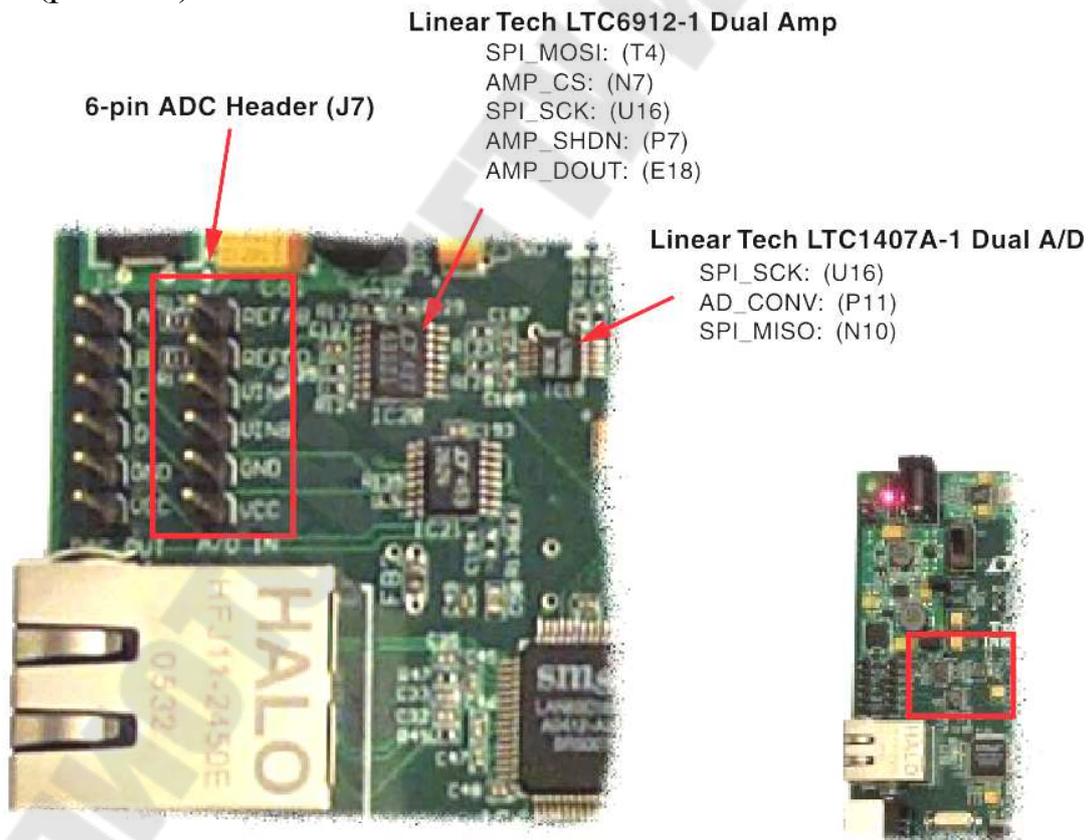


Рис. 5.1. Узел АЦП Spartan-3E Starter Kit

Структурная схема этого узла и его сопряжения с основной ПЛИС XC3S500E отладочного модуля Xilinx Spartan-3E Starter Board показана на рис.5.2.

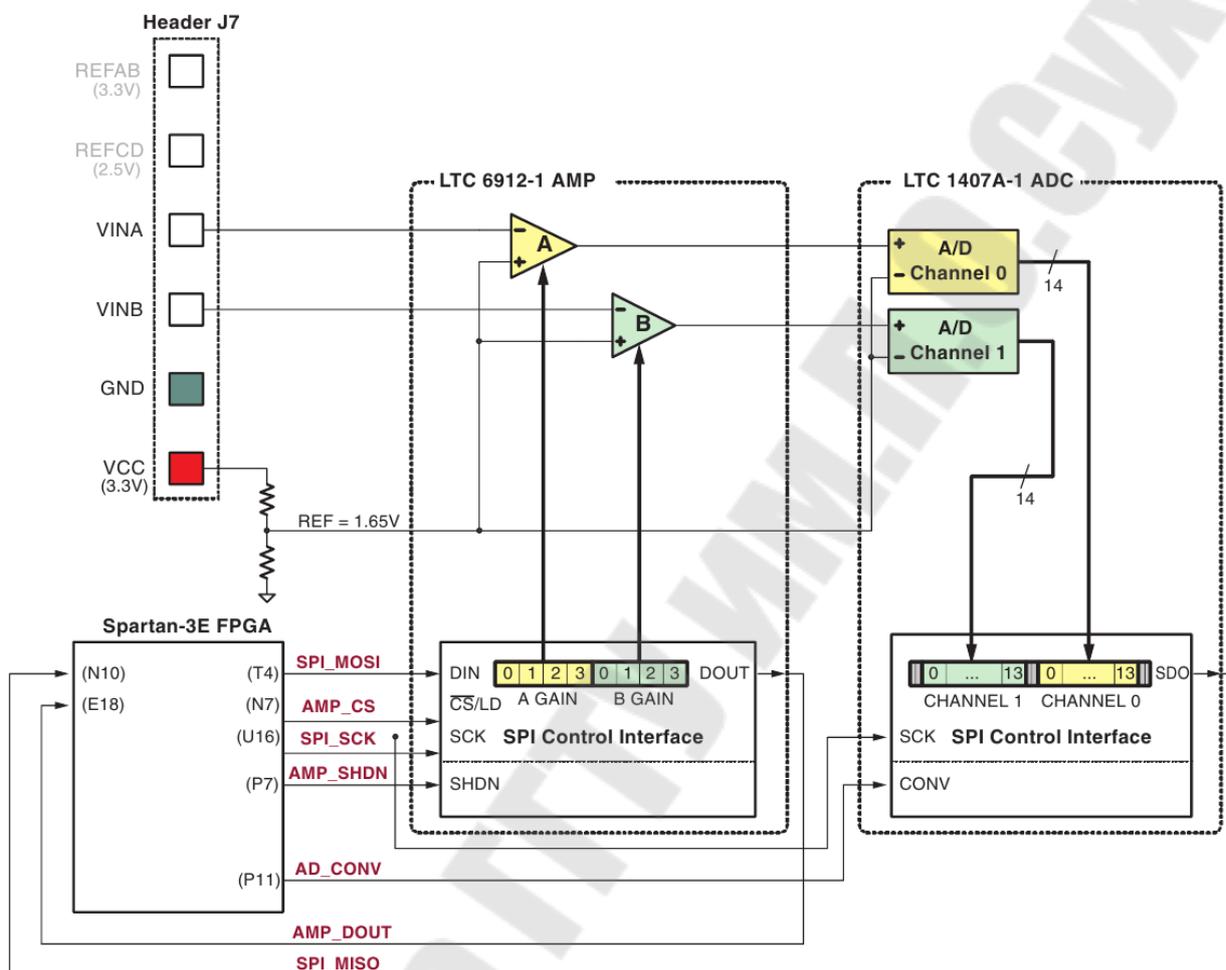


Рис. 5.2. Структурная схема узла АЦП и его сопряжения с основной ПЛИС XC3S500E отладочного модуля Xilinx Spartan-3E Starter Board

Аналоговые сигналы подаются на контакты VINA и VINB входного разъема, с которого поступают на входы соответствующих каналов предварительного усилителя, выполненного на основе ИС LTC6912-1 фирмы Linear Technology. Данный усилитель выполняет функцию масштабирования входных сигналов. Регулировка коэффициента усиления осуществляется с помощью интерфейса SPI. АЦП реализован на базе ИС LTC1407A-1, выпускаемой этой же фирмой. АЦП выполняет преобразование аналоговых сигналов в 14-разрядный двоичный код. Результат преобразования транслируется в ПЛИС XC3S500E через последовательный интерфейс SPI.

Аналоговая схема преобразует аналоговое напряжение, подаваемое на VINA и VINB, в 14-разрядный двоичный код, выраженный уравнением:

$$D[13:0] = GAIN \cdot \frac{(V_{IN} - 1,65)}{1,25} \cdot 8192, \quad (5.1)$$

где

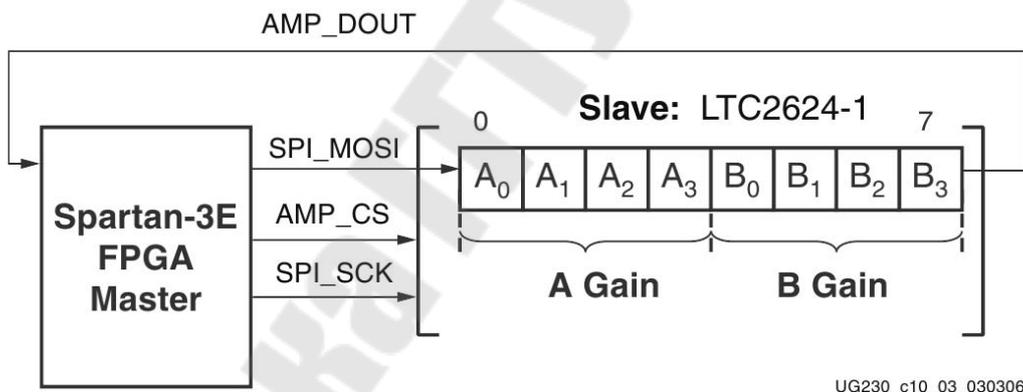
$D[13:0]$ – 14-разрядный код преобразованного аналогового сигнала;

$GAIN$ – коэффициент текущей настройки, загружаемый в программируемый предварительный усилитель;

V_{IN} – напряжение на входах VINA и VINB.

Усиление программируется в диапазоне от -1 до -100. Результат преобразования транслируется в ПЛИС XC3S500E через последовательный интерфейс SPI.

На рис. 5.3 показан протокол связи для взаимодействия с усилителем на основе SPI. Коэффициент усиления для каждого усилителя передается в виде 8-битного командного слова, состоящего из двух 4-битных полей. Самый старший бит, B3, отправляется первым.



UG230_c10_03_030306

Рис. 5.3. Протокол связи для взаимодействия с LTC6912-1

В табл. 5.1 представлены сигналы интерфейса SPI между ПЛИС и усилителем. Сигналы SPI_MISO и SPI_SCK используются совместно с другими устройствами на шине SPI. Сигнал AMP_CS это сигнал выбора входа предусилителя.

Таблица 5.1

Сигналы интерфейса предварительного усилителя

Сигнал	Выходы ПЛИС	Направление	Описание
SPI_MOSI	T4	FPGA→AD	Последовательные данные: Master Output, Slave Input.
AMP_CS	N7	FPGA→AMP	Выбор микросхемы низким уровнем. Коэффициент усиления усилителя устанавливается при возвращении сигнала высокого уровня.
SPI_SCK	U16	FPGA→AMP	Тактовый сигнал
AMP_SHDN	P7	FPGA→AMP	Выключение и сброс высоким уровнем
AMP_DOUT	E18	FPGA←AMP	Последовательные данные. Повторяет предыдущие настройки усиления усилителя.

Транзакция шины SPI начинается, когда ПЛИС XC3S500E устанавливает низкий уровень AMP_CS (рис. 5.4). Усилитель захватывает последовательные данные на SPI_MOSI по переднему фронту тактового сигнала SPI_SCK и предоставляет последовательные данные на AMP_DOUT по спаду SPI_SCK.

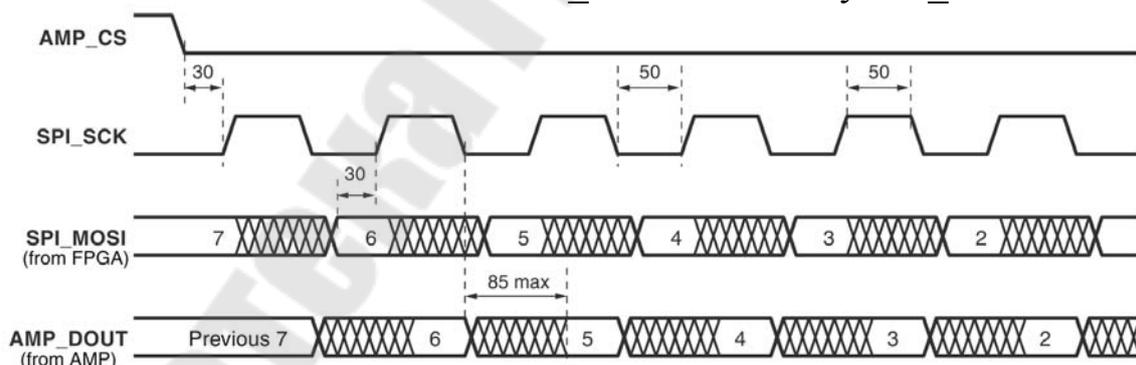


Рис. 5.4. Временные диаграммы шины SPI при связи с усилителем

Реализация модуля AMP_LTC6912 на языке VHDL:

```
architecture Behavioral of AMP_LTC6912 is
    signal gain: STD_LOGIC_VECTOR(7 downto 0) := "00010001";
begin
    ST1_AMPLIFIER_AMP_CS: process(spi_clk)
    begin
        if (spi_clk'event and spi_clk = '0') then -- falling edge
            if (rst = '0') then
```

```

        amp_cs <= '1';
    elsif (interstate = 1) then
        case (n_bit_amp) is
            when 1 to 8 =>
                amp_cs <= '0';
            when others =>
                amp_cs <= '1';
            end case;
        else
            amp_cs <= '1';
        end if;
    end if;
end process;

ST1_AMPLIFIER_MOSI: process(spi_clk)
begin
    if (spi_clk'event and spi_clk = '0') then
        if (rst = '0') then
            n_bit_amp <= 0; -- counter up to 8
            gain <= "00010001";
        elsif (interstate = 1) then
            case (n_bit_amp) is
                when 0 to 7 =>
                    spi_mosi <= gain(7);
                    gain <= gain(6 downto 0) & '0';
                    n_bit_amp <= n_bit_amp + 1;
                    when 9 =>
                        n_bit_amp <= 9;
                    when others =>
                        n_bit_amp <= n_bit_amp + 1;
                end case;
            else
                n_bit_amp <= 0; gain <= "00010001";
            end if;
        end if;
    end process;
end Behavioral;

```

Оба аналоговых входа двухканального АЦП LTC1407A-1 дискретизируются одновременно при подаче сигнала AD_CONV. В табл. 5.2 перечислены сигналы интерфейса SPI между ПЛИС и АЦП. Сигналы SPI_MOSI, SPI_MOSO и SPI_SCK используются совместно с другими устройствами на шине SPI. Сигнал DAC_CS является входом выбора ЦАП. Сигнал DAC_CLR является асинхронным входом сброса ЦАП.

Таблица 5.2

Сигналы интерфейса АЦП

Сигнал	Выходы ПЛИС	Направление	Описание
SPI_SCK	U16	FPGA→ADC	Тактовый сигнал
AD_CONV	P11	FPGA→ADC	Выключение и сброс высоким уровнем
SPI_MISO	N10	FPGA←ADC	Последовательные данные: Master Input, Slave Output. Цифровое представление выборочных аналоговых значений в виде двух 14-разрядных двоичных значений.

Протокол связи для взаимодействия шины SPI с АЦП приведен на рис. 5.5.

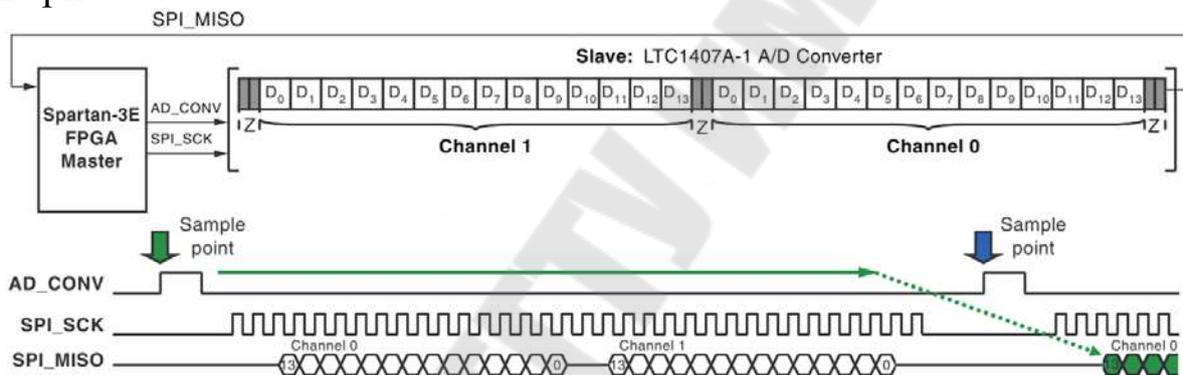


Рис. 5.5. Протокол связи для взаимодействия с LTC1407A-1

Когда уровень сигнала AD_CONV становится высоким, АЦП одновременно выполняет выборку двух аналоговых каналов. Результат этого преобразования предоставляется с задержкой в одну выборку.

На рис. 5.6 показаны временные диаграммы шины SPI при связи с АЦП. Сигнал AD_CONV – это активация выбора SPI. Нужно обязательно обеспечить достаточное количество тактов SPI_SCK, чтобы АЦП оставил сигнал SPI_MISO в высокоимпедансном состоянии. В противном случае АЦП будет блокировать связь с другими периферийными устройствами SPI. Требуется использовать последовательно 34 цикла тактирования. АЦП сообщает о своих выходных данных за два такта до и после каждой 14-битной передачи данных.

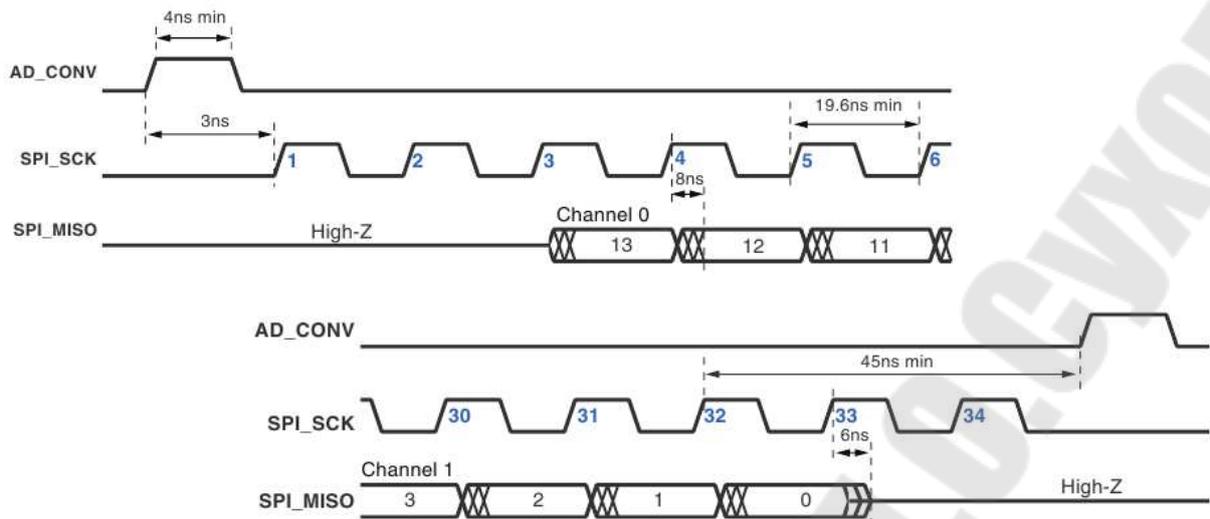


Рис. 5.6. Временные диаграммы шины SPI при связи с АЦП

Шина SPI является общей для других устройств на плате. Очень важно, чтобы другие устройства были отключены, когда ПЛИС взаимодействует с усилителем или АЦП.

В табл. 5.3 приведены сигналы и логические значения, необходимые для отключения других устройств. Несмотря на то, что PROM StrataFlash является параллельным устройством, его младший бит данных используется совместно с сигналом SPI_MISO. Platform Flash PROM может быть включена только случае, если ПЛИС настроена для конфигурации в режиме Master Serial.

Таблица 5.3

Отключение других устройств на шине SPI

Сигнал	Отключенное устройство	Значение для отключения
SPI_SS_B	SPI Serial Flash	1
AMP_CS	Программируемый предусилитель	1
DAC_CS	ЦАП	1
SF_CE0	StrataFlash Parallel Flash PROM	1
FPGA_INIT_B	Platform Flash PROM	0

Реализация модуля ADC_LTC1407A на языке VHDL:

```
architecture Behavioral of ADC_LTC1407A is
    signal adc_1_temp: STD_LOGIC_VECTOR(13 downto 0);
begin
    ST2_ADC: process(spi_clk)
    begin
```

```

if (spi_clk'event and spi_clk = '0') then
  if (rst = '0') then
    n_bit_adc <= 0;
    adc_1_temp <= "0000000000000000";
  elsif (interstate = 2) then
    if (n_bit_adc = 34) then
      n_bit_adc <= 0;
    else
      if (n_bit_adc > 0) then
        n_bit_adc <= n_bit_adc + 1;
      end if;
      case (n_bit_adc) is
        when 0 =>
          ad_conv <= '1';
          n_bit_adc <= n_bit_adc + 1;
        when 1 to 2 =>
          ad_conv <= '0';
        when 3 to 16 =>
          adc_1_temp <= adc_1_temp(12 downto 0) & spi_miso;
        when 17 =>
          adc_1_temp <= not adc_1_temp;
        when 18 =>
          if (adc_1_temp(13) = '0') then
            adc_1_data <= adc_1_temp + "0000000000000001";
            sign_adc_1_data <= '1';
          else
            adc_1_data <= not adc_1_temp;
            sign_adc_1_data <= '0';
          end if;
        when others =>
          ad_conv <= '0';
        end case;
      end if;
    else
      ad_conv <= '0';
    end if;
  end if;
end process;
end Behavioral;

```

Порядок выполнения работы

1. На примере представленных VHDL модулей работы с АЦП и масштабирующим усилителем разработать их описание на языке Verilog.

2. Подключить выходы регистра `adc_1_data` к светодиодам LD0–LD7, изображенным на рис. 5.7. Организовать конечный автомат, который изменяет выводимую на светодиоды информацию следующим образом:

– при нижнем положении переключателя N17 светодиоды отображают младшие 8 бит 14-битного регистра `adc_1_data`;

– при верхнем положении переключателя N17 светодиоды отображают старшие 6 бит 14-битного регистра `adc_1_data`;

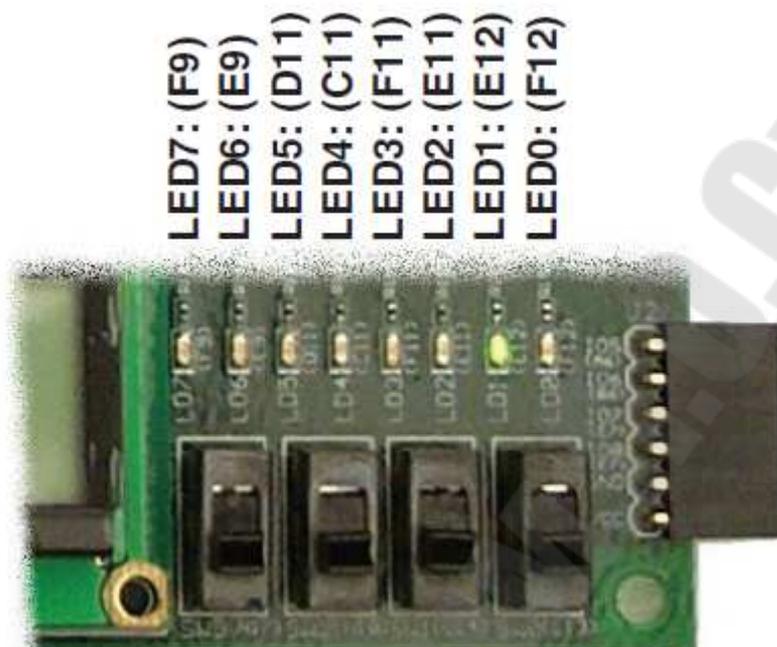


Рис. 5.7. Группа светодиодов LD0–LD7

3. Подключить к разъему J7 магазин сопротивлений с подключенным к нему вольтметром В7-37/2. Показания вольтметра $U_{д,В}$ примем за действительные значения измерения.

4. Переключатели 1 и 2 образуют с переключателем 3 – делитель напряжения. Таким образом, изменяя их положение требуется подать на вход V_{inA} разные значения входного напряжения (значение напряжения $U_{д,В}$ контролируется вольтметром). Результаты занести в табл. 5.4.

5. На основании формулы 5.1 рассчитать измеренное значение напряжения АЦП $U_{АЦП,В}$. Рассчитать абсолютные и относительные погрешности измерения.

Таблица 5.4

Результаты эксперимента

$U_{д}, В$	Код АЦП	$U_{АЦП}, В$	$\Delta, В$	$\delta, \%$
0,1				
0,5				
1				
1,5				
2				
2,5				
3				

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, результаты измерений, выводы.

Контрольные вопросы

1. Что такое АЦП?
2. Какую функцию выполняет программируемый предварительный усилитель?
3. Опишите протокол связи для взаимодействия с усилителем на основе SPI?
4. Каким образом с помощью интерфейса SPI настроить АЦП?
5. Опишите протокол связи для взаимодействия шины SPI с АЦП.

Лабораторная работа № 6

Реализация цифро-аналогового преобразования на основе отладочной платы Spartan-3E Starter Kit

Цель работы

Разработать описание модуля для работы с ЦАП, реализовать его на ПЛИС, а также изучить технические характеристики микросхем ЦАП.

Основные теоретические сведения

В состав узла ЦАП входит ИС LTC2624 фирмы Linear Technology и выходной разъем J5. ЦАП и разъем J5 расположены над разъемом Ethernet RJ-45, как показано на рис. 6.1.

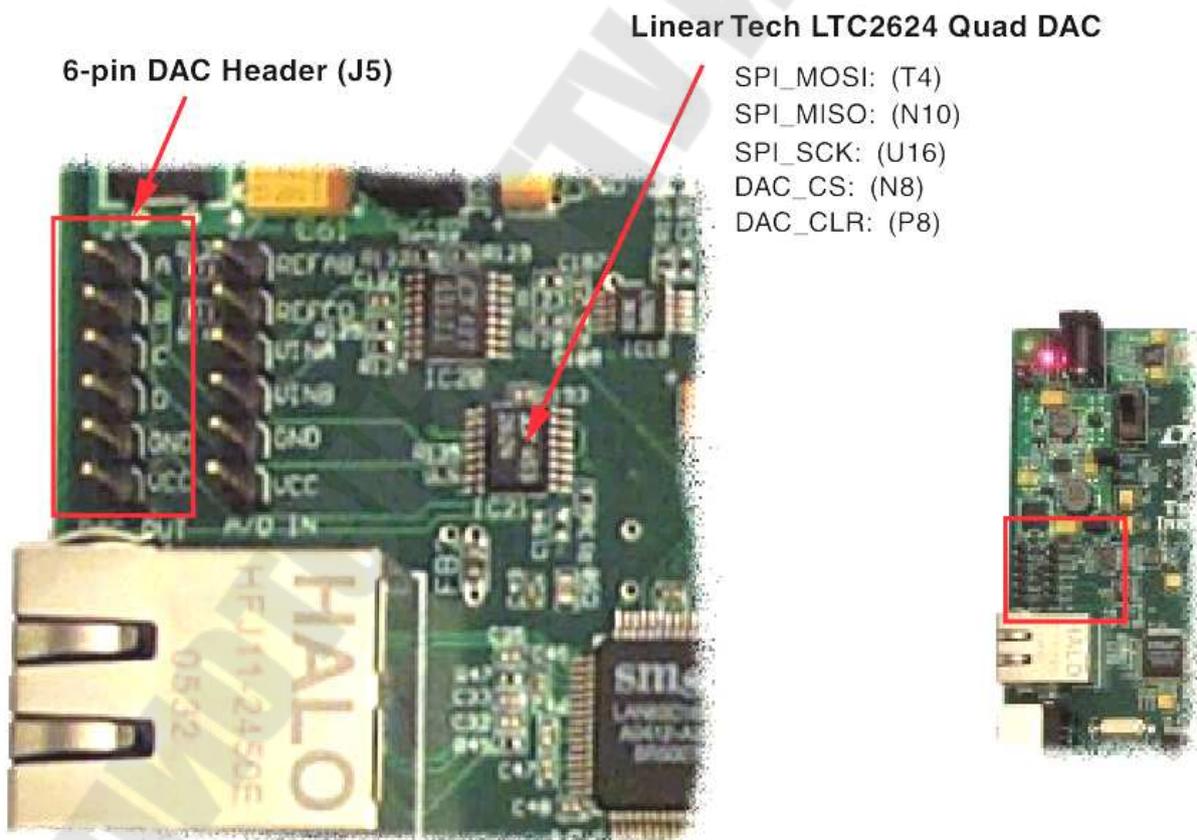


Рис. 6.1. Узел ЦАП Spartan-3E Starter Kit

Микросхема LTC2624 содержит четырехканальный ЦАП с 12-разрядным разрешением. Коммутация цифровых сигналов, формируемых в ПЛИС, на входы ЦАП осуществляется с помощью интерфейса SPI. При этом может использоваться как 24-, так и 32-разрядный протокол передачи данных. Сформированные аналоговые сигналы, уровень которых соответствует значениям входного 12-разрядного двоичного кода без знака, поступают на контакты А-Д выходного разъема.

Рис. 6.2 демонстрирует структурную схему узла ЦАП и его сопряжение с основной ПЛИС XC3S500E отладочного модуля Xilinx Spartan-3E Starter Board.

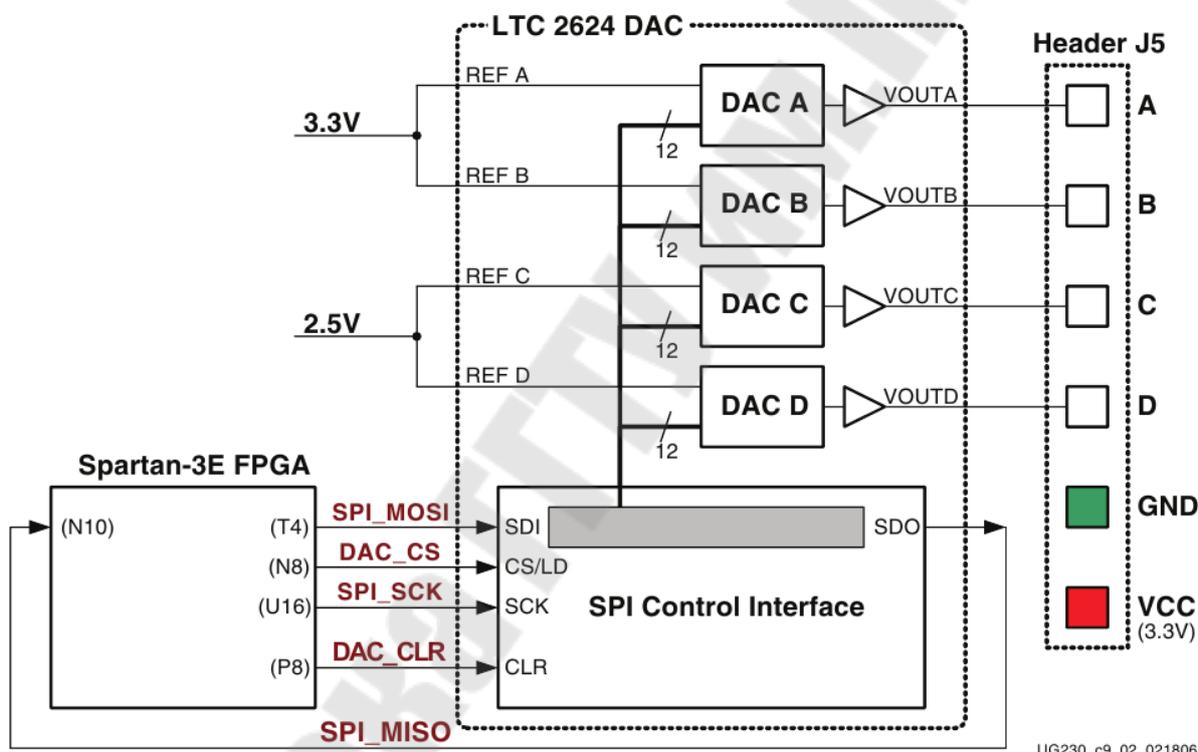


Рис. 6.2. Структурная схема узла ЦАП и его сопряжения с основной ПЛИС XC3S500E отладочного модуля Xilinx Spartan-3E Starter Board

В табл. 6.1 представлены сигналы интерфейса между ПЛИС и ЦАП. DAC_CS – это сигнал выбора входа ЦАП. Сигнал DAC_CLR является асинхронным входом сброса ЦАП.

Сигналы интерфейса ЦАП

Сигнал	Выходы ПЛИС	Направление	Описание
SPI_MOSI	T4	FPGA→DAC	Последовательные данные: Master Output, Slave Input
DAC_CS	N8	FPGA→DAC	Цифроаналоговое преобразование начинается, когда сигнал возвращается в состояние единицы.
SPI_SCK	U16	FPGA→DAC	Тактовый сигнал
DAC_CLR	P8	FPGA→DAC	Асинхронный вход сброса, активный-низкий
SPI_MISO	N10	FPGA←DAC	Последовательные данные: Master Input, Slave Output.

На рис. 6.3 показаны временные диаграммы шины SPI при работе с ЦАП. Каждый бит передается или принимается относительно тактового сигнала SPI_SCK. Шина поддерживает тактовую частоту до 50 МГц.

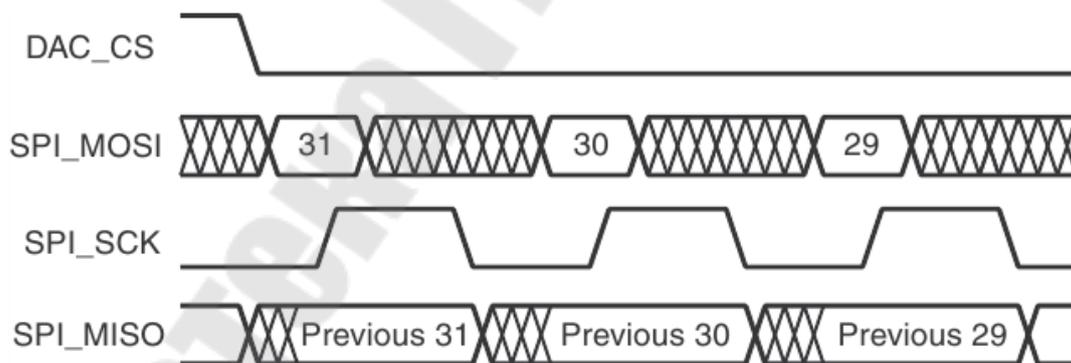


Рис. 6.3. Временные диаграммы шины SPI при работе с ЦАП

Последовательный вывод данных из ЦАП используется для каскадного подключения нескольких ЦАП. В таб. 6.2 приведены сигналы и логические значения, необходимые для отключения других устройств шины SPI.

Таблица 6.2

Отключение других устройств на шине SPI

Сигнал	Отключенное устройство	Значение для отключения устройства
SPI_SS_B	SPI Serial Flash	1
AMP_CS	Программируемый предварительный усилитель	1
AD_CONV	АЦП	0
SG_CE0	StrataFlash Parallel Flash PROM	1
FPGA_INIT_B	Platform Flash PROM	0

После перехода сигнала DAC_CS в низкий уровень, ПЛИС предает данные по SPI_MOSI. LTC2624 захватывает входные данные SPI_MOSI по переднему фронту SPI_SCK. Данные должны быть действительны минимум 4 нс относительно нарастающего фронта тактового сигнала. ЦАП LTC2624 передает свои данные по сигналу SPI_MISO по спаду SPI_SCK. ПЛИС фиксирует эти данные на следующем нарастающем фронте SPI_SCK. ПЛИС должна прочитать первое значение SPI_MISO на первом нарастающем фронте SPI_SCK после того, как уровень сигнала DAC_CS станет низким. В противном случае теряется 31 бит. После передачи всех 32 битов данных ПЛИС завершает транзакцию шины SPI, возвращая сигнал DAC_CS в высокое состояние. Высокий фронт запускает процесс цифроаналогового преобразования.

На рис. 6.4 показан протокол связи, необходимый для взаимодействия с LTC2624. ЦАП поддерживает как 24-битный, так и 32-битный протокол. Внутри ЦАП интерфейс SPI образован 32-битным регистром сдвига. Каждое 32-битное командное слово состоит из команды, адреса, за которым следуют данные. Когда новая команда входит в ЦАП, предыдущее 32-битное командное слово возвращается ведущему.

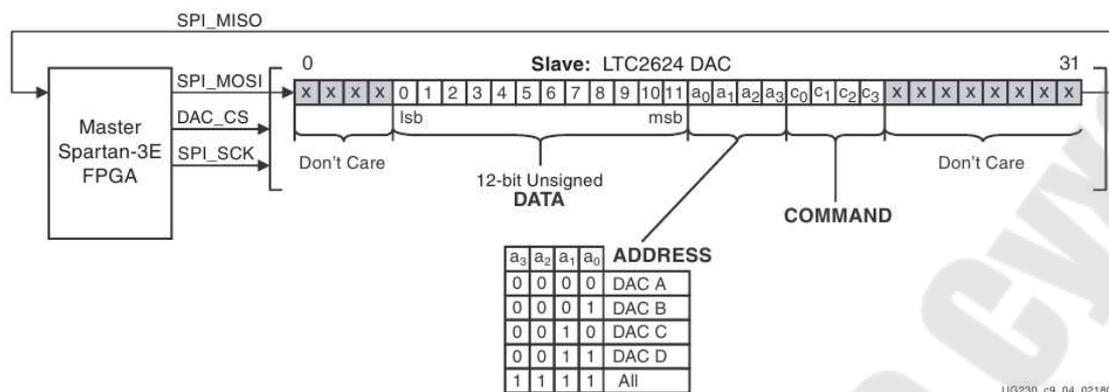


Рис. 6.4. Протокол связи для взаимодействия с LTC2624

Сначала ПЛИС отправляет восемь незначимых битов, а затем 4-битную команду. Наиболее часто используемая команда – COMMAND [3:0] = «0011», которая обновляет выбранный выход ЦАП с указанным значением данных. Следуя этой команде, ПЛИС выбирает один или несколько выходных каналов ЦАП через 4-битное поле адреса. После поля адреса ПЛИС отправляет 12-разрядное значение без знака данных, которое ЦАП преобразует в аналоговое значение на выбранных входах. Дальше четыре дополнительных незначимых бита дополняют 32-битное командное слово.

Напряжение на конкретном выходе описывается уравнением 6.1. Опорное напряжение $V_{\text{REFERENCE}}$ разное на четырех выходах ЦАП. Каналы А и В используют опорное напряжение 3,3 В, а каналы С и D используют 2,5 В. Опорные напряжения имеют допуск $\pm 5\%$, поэтому могут быть небольшие отклонения в выходном напряжении.

$$V_{\text{OUT}} = \frac{D[11:0]}{4,096} \cdot V_{\text{REFERENCE}}, \quad (6.1)$$

где V_{OUT} – выходное напряжение; $D[11:0]$ – 12-разрядное беззнаковое значение, записанное в ЦАП; $V_{\text{REFERENCE}}$ – опорное напряжение.

Уравнение 6.2 используется для выходов А и В. Опорное напряжение равняется 3,3 В $\pm 5\%$

$$V_{\text{OUTA}} = \frac{D[11:0]}{4,096} \cdot (3,3 \pm 5\%). \quad (6.2)$$

Уравнение 6.3 используется для выходов С и D. Опорное напряжение этих выводов равно 2,5 В $\pm 5\%$

$$V_{\text{OUTA}} = \frac{D[11:0]}{4,096} \cdot (2,5 \pm 5\%). \quad (6.3)$$

Реализация модуля DAC_LTC26224:

```

architecture Behavioral of DAC_LTC26224 is
    signal mux_data: STD_LOGIC_VECTOR(11 downto 0);
    signal const_com: STD_LOGIC_VECTOR(3 downto 0) := "0011";
begin
    ST3_DAC: process (spi_clk) begin
        if (spi_clk'event and spi_clk = '0') then
            if (rst = '0') then n_bit_dac <= 0;
            elsif (interstate = 3) then
                if (n_bit_dac = 25) then
                    n_bit_dac <= 0; dac_cs <= '1';
                else
                    n_bit_dac <= n_bit_dac + 1;
                    if (n_bit_dac = 1) then dac_cs <= '0'; end if;
                    case (n_bit_dac) is
                        when 0 to 3 =>
                            spi_mosi <= const_com(3);
                            const_com <= const_com(2 downto 0) & '0';
                        when 4 to 7 => spi_mosi <= '0';
                        when others =>
                            spi_mosi <= mux_data(11);
                            mux_data <= mux_data(10 downto 0) & '0';
                    end case;
                end if;
            else
                dac_cs <= '1'; const_com <= "0011";
                if (sign_adc_1_data = '0') then
                    mux_data <= ("100000000000" - adc_1_cut);
                else
                    mux_data <= ("100000000000" + adc_1_cut);
                end if;
            end if;
        end if;
    end process;
end Behavioral;

```

Порядок выполнения работы

1. На примере представленных VHDL модулей работы с микросхемой ЦАП требуется самостоятельно разработать его описание на языке Verilog.

2. Разработать конечный автомат, который выполняет следующие функции:

– напряжение порта А микросхемы ЦАП линейно возрастает если переключатели N17 и N18 в верхнем положении;

– напряжение порта А микросхемы ЦАП линейно убывает, если переключатель N17 в нижнем положении, а переключатель N18 – в верхнем;

– напряжение порта А микросхемы ЦАП не изменяет своего значения при любом положении переключателя N17, если переключатель N18 в нижнем положении.

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.

Контрольные вопросы

1. Что такое ЦАП?
2. Опишите протокол связи для взаимодействия с ЦАП LTC2624 на основе SPI.
3. Какое опорное напряжение имеют каналы А, В, С и D?

Лабораторная работа № 7

Аппаратные и программные средства отладки eZDSP F2812 и Code Composer Studio

Цель работы

Изучить отладочное оборудование для цифрового сигнального процессора семейства C28x фирмы Texas Instruments – отладочную плату eZDSP320F2812 с платой расширения Zwickau Adapterboard. Изучить 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. 7.1 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. 7.1). 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. 7.2). 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.

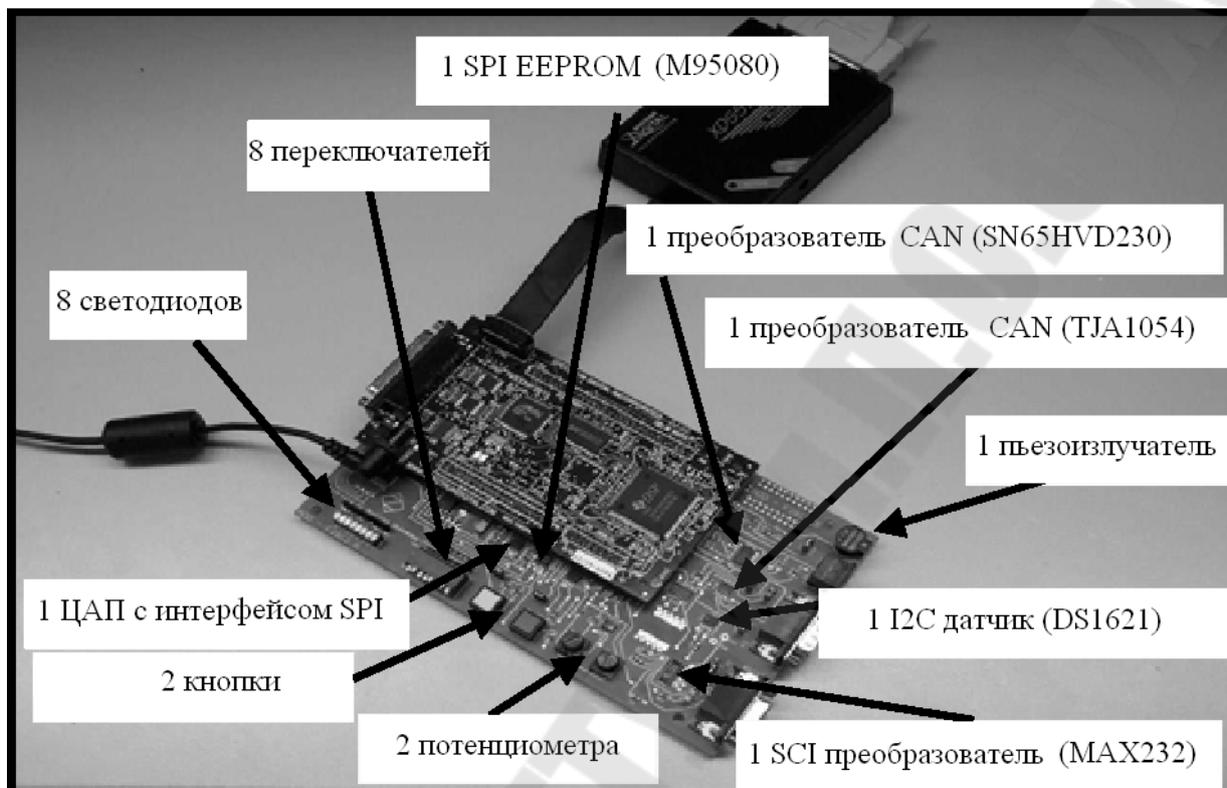


Рис. 7.1. Плата эмулятора eZDSP и плата расширения Zwickau Adapterboard

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

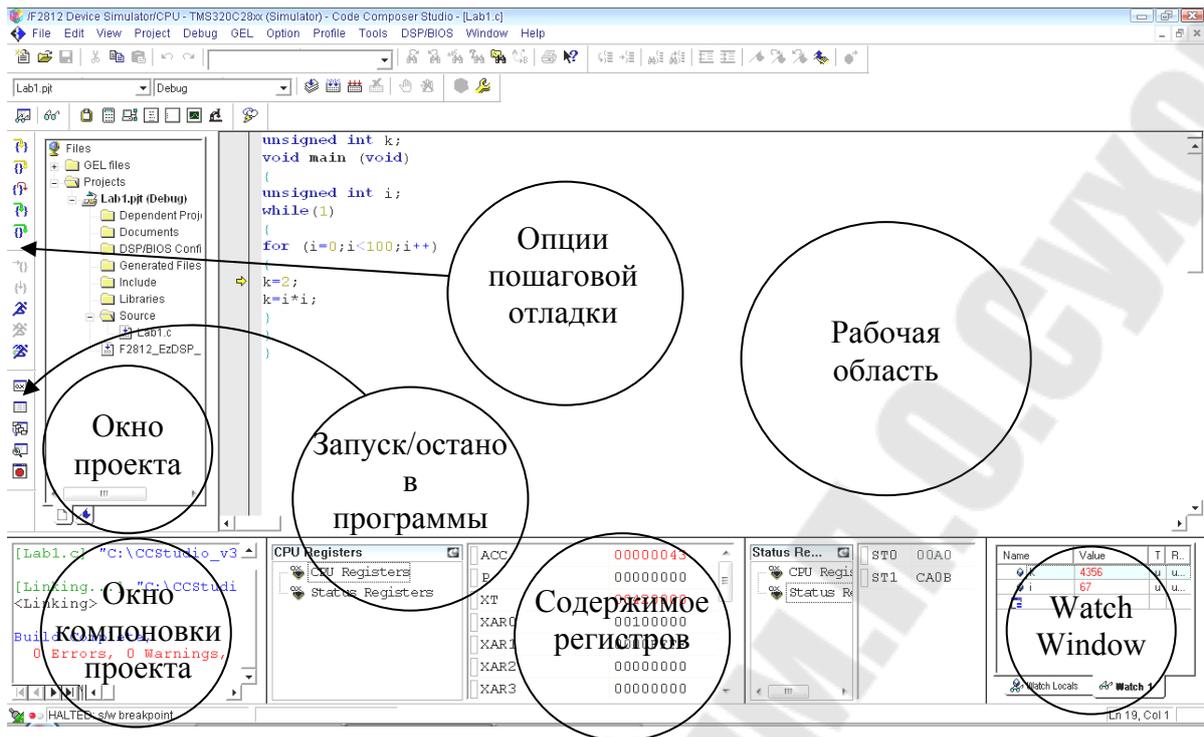


Рис. 7.2. Вид рабочего окна Code Composer Studio

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.7.3. 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.7 .3. Тестовая программа

3. Добавление файлов в проект.

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

```

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

```

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

4. Компиляция программы.

Компилируя программу, мы проверяем 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.

Замечания:

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

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

7. Сброс сигнального процессора и запуск программы на выполнение.

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

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

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

Запуск программы в автоматическом режиме осуществляется через вкладку меню Debug → Run (горячая клавиша F5) или с помощью иконки . При этом в строке статуса внизу рабочей области индицируется 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 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.7.3) так, как показано на рис. 7.4.

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

Рис. 7.4. Измененная тестовая программа

10. Режим Animate.

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

11. Просмотр содержимого регистров.

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

Задание для самостоятельной работы

1. Установите Breakpoint на строки 7, 9 и 10 измененной тестовой программы, показанной на рис.7.4. Перезапускайте программу в автоматическом режиме (Run) после каждой остановки в выполнении программы на точках останова. Проследите изменение переменных *i* и *k* в окне Watch. Объясните полученные результаты.

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

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

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

Содержание отчета

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

Контрольные вопросы:

1. Структура платы эмулятора eZDSP320F2812 и платы расширения Zwickau Adapterboard.
2. Среда разработки Code Composer Studio. Создание, компиляция, компоновка и загрузка проекта.
3. Среда разработки Code Composer Studio. Режимы запуска программ.
4. Среда разработки Code Composer Studio. Просмотр переменных.
5. Среда разработки Code Composer Studio. Использование точки останова (Breakpoint).
6. Среда разработки Code Composer Studio. Просмотр содержимого регистров ЦСП.

7. Каким образом осуществить отладку исследуемой программы в пошаговом режиме:

- без исследования ассемблерного кода;
- с исследованием ассемблерного кода;
- с одновременным исследованием исходного текста на Си и соответствующего ему ассемблерного кода.

Изучение карты памяти, структуры цифровых портов ввода/вывода и системы тактирования ЦСП TMS320F2812

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. 8.1. Память и все периферийные модули объединены системой шин по модифицированной гарвардской архитектуре.

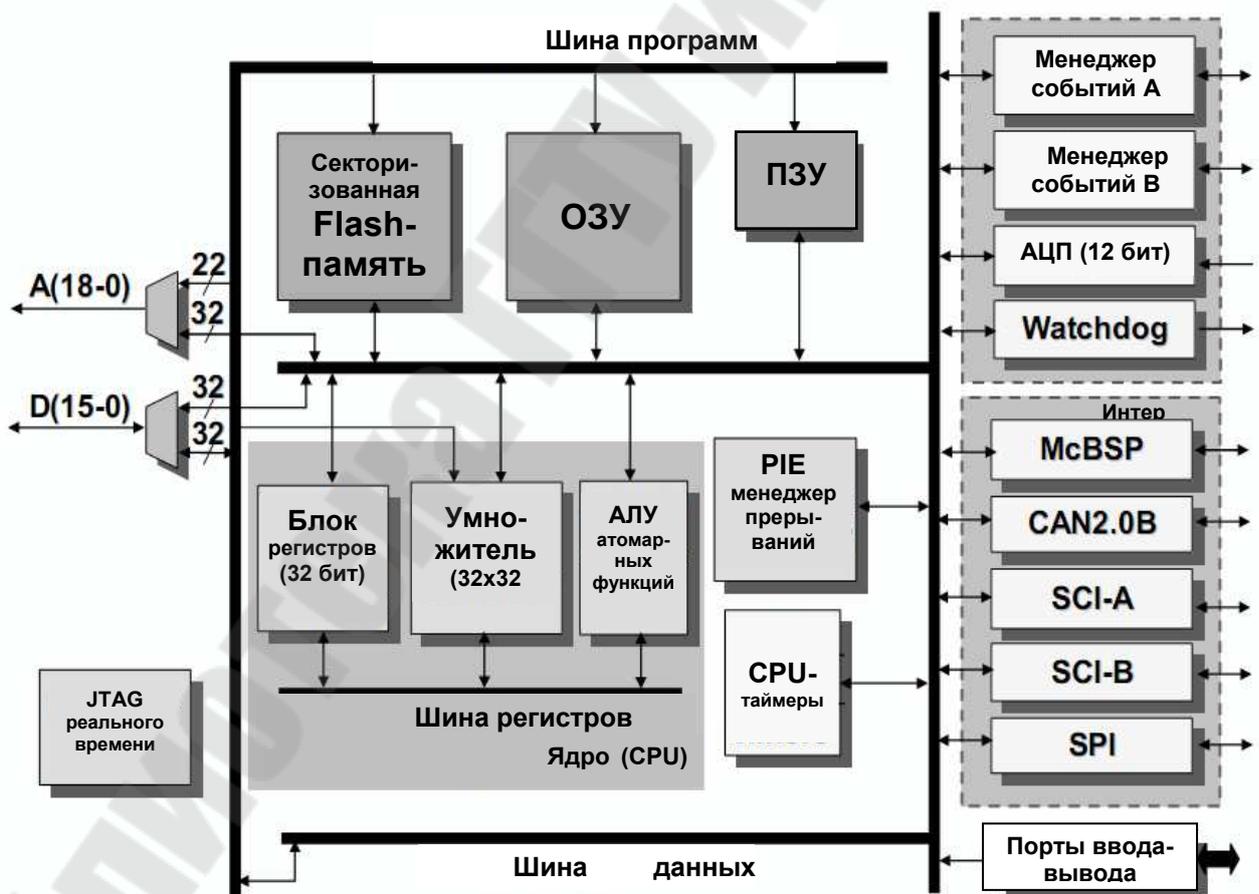


Рис. 8.1. Структурная схема ЦСП семейства C28x

Карта адресного пространства ЦСП C28x показана на рис. 8.2. В качестве памяти данных используется исключительно ОЗУ быстрого доступа (Single Access RAM – SARAM, доступ возможен в течение каждого машинного цикла) общим объемом 18 Кслов, состоящее из нескольких банков – M0, M1 (2x1К), L0, L1 (2x4К) и H0 (8К). 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).

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. 8.2).

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.

- 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. 8.3. Основные и альтернативные функции портов ввода-вывода F2812

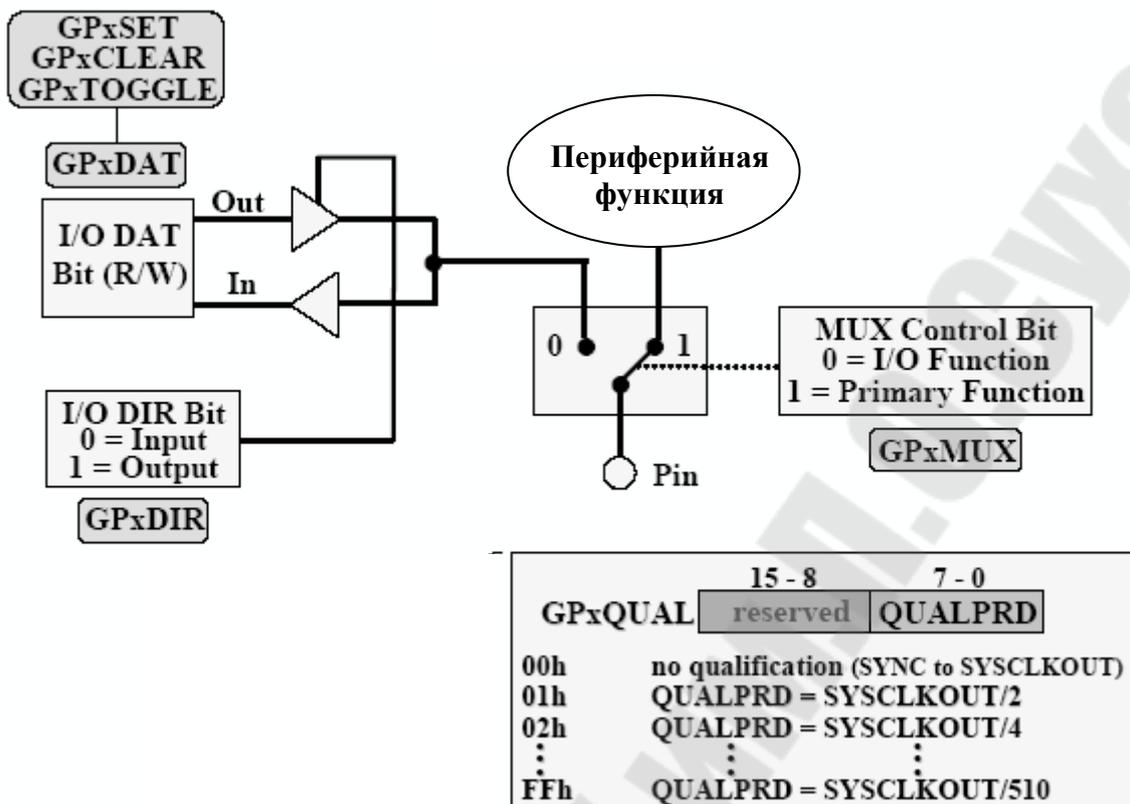


Рис. 8.4. Функциональная схема линии GPIO

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 E 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 SYSCCLKOUT, 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 ($OSCCLK \cdot 10/2$, см. рис. 8.5). Коэффициент деления задается программно в регистре PLLCR.

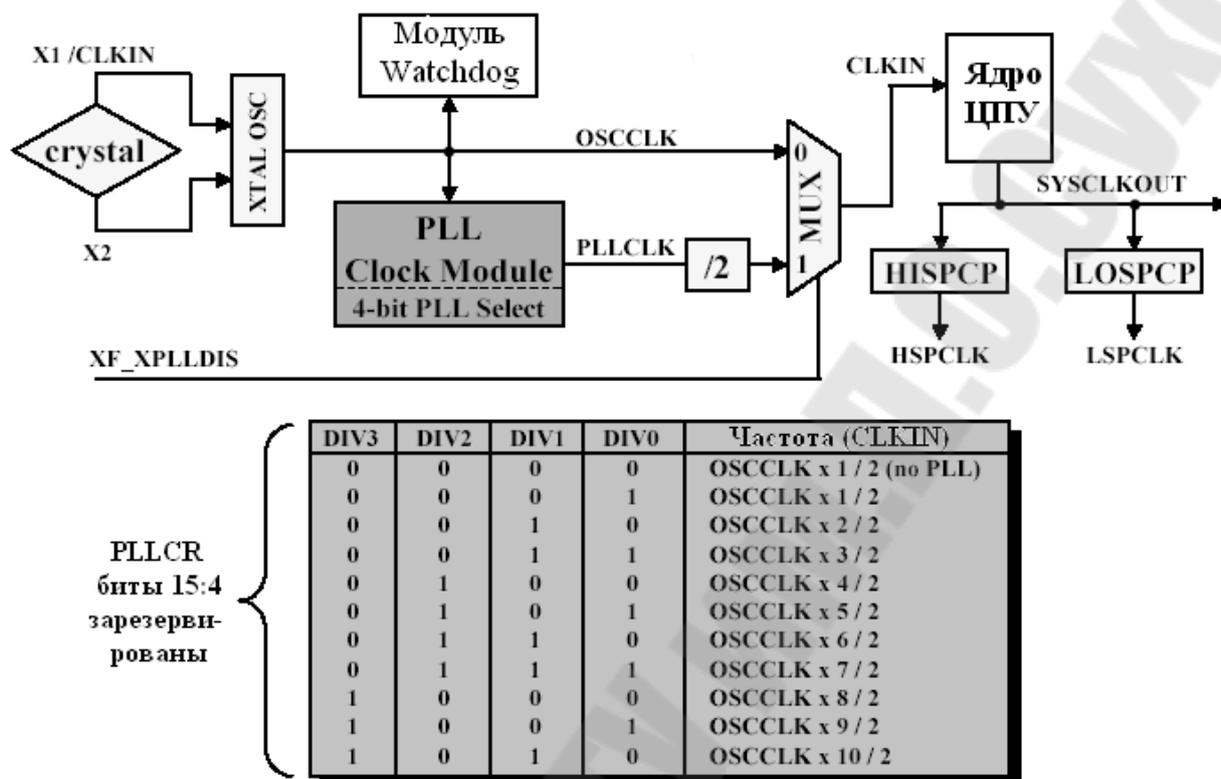
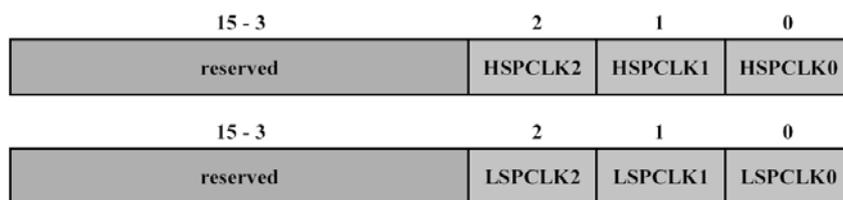


Рис. 8.5. Тактовый модуль и регистр PLLCR

Помимо формирователя тактовой частоты ядра, тактовый модуль содержит предделители: низкоскоростной LOSPCP и высокоскоростной HSPCP (рис. 8.6), 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 HSPCP.

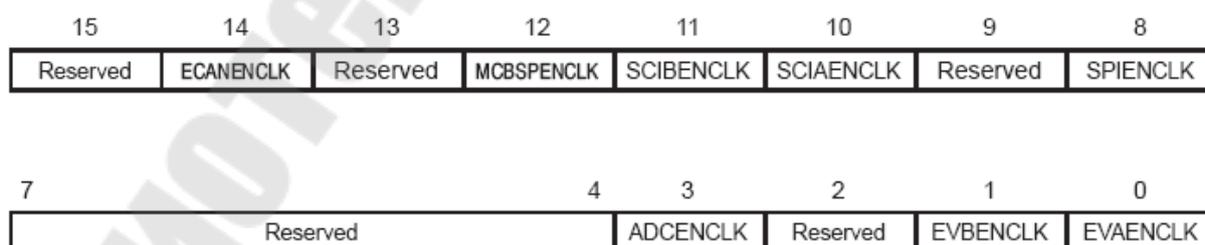


H/LSPCLK2	H/LSPCLK1	H/LSPCLK0	Тактовая частота периферии
0	0	0	SYSCCLKOUT / 1
0	0	1	SYSCCLKOUT / 2 исходная для HSPSCP
0	1	0	SYSCCLKOUT / 4 исходная для LSPSCP
0	1	1	SYSCCLKOUT / 6
1	0	0	SYSCCLKOUT / 8
1	0	1	SYSCCLKOUT / 10
1	1	0	SYSCCLKOUT / 12
1	1	1	SYSCCLKOUT / 14

Рис. 8.6. Формат регистров HSPSCP и LSPSCP

В ЦСП семейства C28x реализована возможность не только изменять, но и полностью запрещать/разрешать тактирование различных периферийных модулей при помощи регистра PCLKCR (рис. 8.7).

Сторожевой таймер (Watchdog timer или WDT, рис. 8.8) 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.



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

Рис. 8.7. Формат регистра PCLKCR: 1 – разрешить тактирование периферийного модуля; 0 – запретить.



Рис. 8.8. Формат регистра управления сторожевого таймера WDCR

Сторожевой таймер защищен ключом «101», и, в случае записи в биты WDCHK 2–0 регистра WDCR (рис. 8.9) любой иной кодовой комбинации, в следующем машинном цикле происходит генерация выходного сигнала (такого же, как и при переполнении).

Неверный ключ

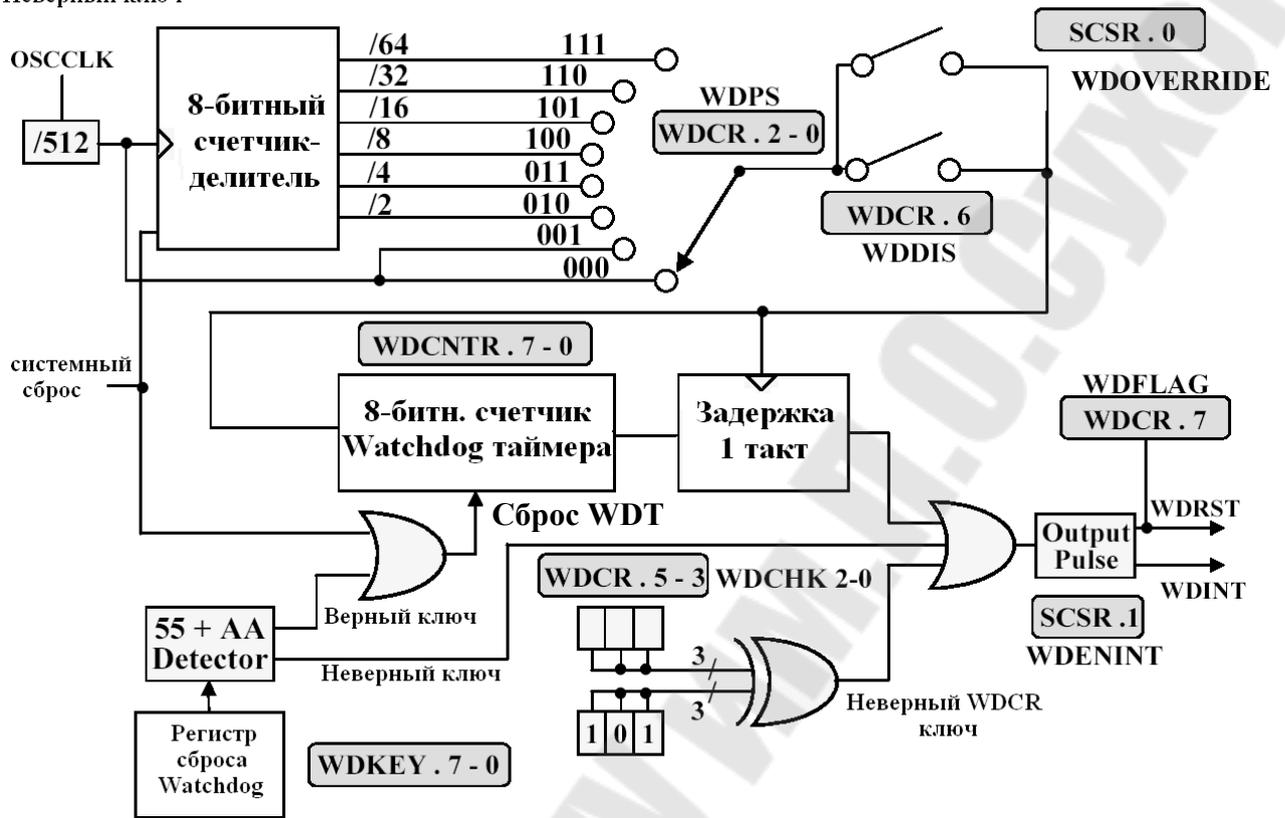


Рис. 8.9. Функциональная схема сторожевого таймера

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

Регистр управления и состояния (System Control and Status Register – SCSR, рис.8.10) управляет сбросом сторожевого таймера. Бит WDINTS отражает текущее состояние сигнала WDINT. Бит WDENINT – выбор режима действия от WDT (сброс или прерывание). Если WDENINT=0 – разрешен сигнал сброса WDRST, если WDENINT=1 – разрешен сигнал прерывания WDINT.

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



Рис. 8.10. Формат регистра управления и состояния SCSR

Порядок выполнения работы

Часть I

В части I лабораторной работы необходимо разработать программу «бегущие огни», задавая направление движения: сначала – от края к краю (рис. 8.11, а), а потом, зажигая по два светодиода, – от периферии к центру (рис. 8.11, 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. 8.12). 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_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
```

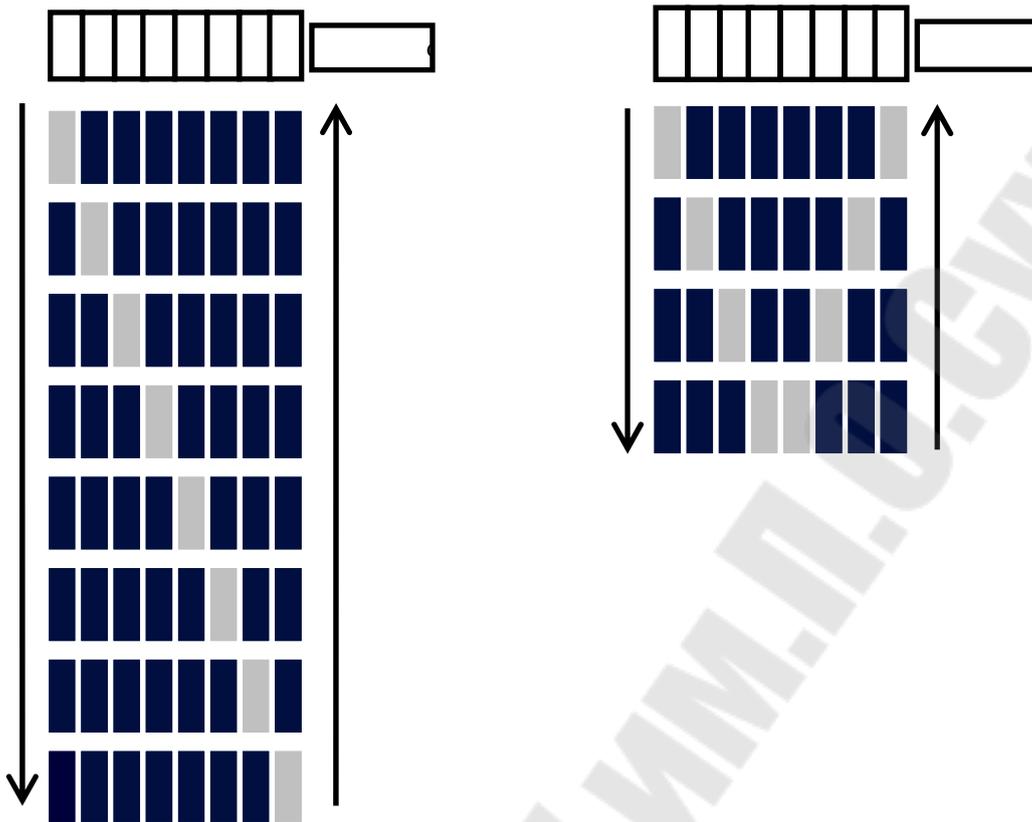


Рис. 8.11. Направление движения светодиодов:
а – слева направо и наоборот;
б – от периферии к центру и наоборот.

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:\tides\C28\dsp281x\v100\DSP281x_headers\include;..\include

2.2. Добавление библиотек и создание стека.

Подключаем Си-библиотеки:

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

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

Задаем глубину стека 0x400:

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

```

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

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

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

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

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

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

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

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

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

```

6

7

```

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

    GpioMuxRegs.GPAQUAL.all = 0x?; // Запрещение входного
ограничителя
    GpioMuxRegs.GPBQUAL.all = 0x?;
    GpioMuxRegs.GPDQUAL.all = 0x?;
    GpioMuxRegs.GPEQUAL.all = 0x?;
    EDIS;
}
}
#####
##
// Подпрограмма: InitSystem
//
// Описание: Настройка сторожевого таймера на работу,
// предделитель = 1. Выработка сброса сторожевым
// таймером. Внутренняя частота работы ЦСП 150 МГц.
// Запись в предделитель высокоскоростного таймера
// коэффициента деления 2, а в предделитель
// низкоскоростного таймера - 4. Запрещение работы
// периферийных устройств
//#####
##
void InitSystem(void)
{
    EALLOW;
    SysCtrlRegs.WDCR = 0x?; // Разрешение работы сторожевого
// таймера, предделитель = 64
//или запрещение работы сторо-
//жевого таймера, предделитель = 1
    SysCtrlRegs.SCSR = ?; // Конфигурирование сброса ЦСП от
WDT
    SysCtrlRegs.PLLCR.bit.DIV = ?; // Настройка блока умножения
частоты
    SysCtrlRegs.HISPCP.all = 0x?; // Задание значений
высокоскоростного
    SysCtrlRegs.LOSPCP.all = 0x?; // и низкоскоростного предделителей
    SysCtrlRegs.PCLKCR.bit.EVAENCLK=?; // Запрещение работы
    SysCtrlRegs.PCLKCR.bit.EVBENCLK=?; // периферийных устройств
    SysCtrlRegs.PCLKCR.bit.SCIBENCLK=?;
    SysCtrlRegs.PCLKCR.bit.MCBSPENCLK=?;
    SysCtrlRegs.PCLKCR.bit.SPIENCLK=?;
    SysCtrlRegs.PCLKCR.bit.ECANENCLK=?;
    SysCtrlRegs.PCLKCR.bit.ADCENCLK=?;
    EDIS;
}
}

```

Рис. 8.12. Заготовка текста программы «бегущие огни»

3. Инициализация системы.

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

3.1. Присваивая необходимое значение регистру WDCR (рис. 8.9), 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. 8.12).

3.2. Программируем регистр SCSR (см. рис.8.10) на сброс процессора при переполнении Watchdog-таймера. В бит WDENINT записываем «0», в бит WDOVERRIDE также записываем «0» для того, чтобы оставить его в единичном состоянии (см. область 2 на рис. 8.12).

3.3. В регистр PLLCR (см. рис.8.5) заносим код, необходимый для преобразования частоты кварцевого резонатора 30 МГц во внутреннюю частоту SYSCLKOUT работы процессора 150 МГц (см. область 3 на рис.8.12).

3.4. Заносим в высокоскоростной предделитель HISPCP (см. рис.8.6) коэффициент деления 2, а в низкоскоростной (LOSPCP) – 4 (см. область 3 на рис.8.12).

3.5. Установкой-сбросом бит в регистре PCLKCR (рис. 8.7) запрещаем работу всех периферийных устройств (см. область 4 на рис. 8.12).

4. Настройка портов.

Переходим к подпрограмме «Gpio_select()».

4.1. Установкой-сбросом бит в регистрах GPxMUX настраиваем все выходы на работу в качестве портов (см. область 5 на рис. 8.12).

4.2. Установкой-сбросом бит в регистрах GPxDIR настраиваем все линии портов A, D, E, F, G на ввод (см. область 5 на рис.8.12). В порту GPIOB настраиваем линии порта GPIOB15 – GPIOB8 на ввод, а GPIOB7 – GPIOB0 на вывод (см. область 5 на рис.8.12).

4.4. Сбрасываем все биты регистров GPxQUAL портов A, B, D, E в ноль (см. область 5 на рис. 8.12).

5. Расчет и задание временной задержки

В строке «delay_loop()» (см. область 6 на рис. 8.12) в скобках указываем число n – параметр задержки. Длительность задержки будет пропорциональна данному числу и рассчитывается по формуле (в секундах):

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

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

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

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

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

7. Тестирование программы.

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

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

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

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

- остановить выполнение программы;
- установить точку останова на строку программы, отмеченную знаком «●»;
- выделить мышью в любом месте текста программы название «GpioDataRegs.GPBDAT», нажать правую кнопку мыши и в открывшемся окне выбрать «Add to Watch Window»;
- щелкнуть по значку «+» напротив имени «GpioDataRegs.GPBDAT» в окне Watch Window, далее выбрать «bit», после чего в данном окне будут выведены в столбик двоичные состояния разрядов порта GPIOB (рис.8.13);
- исключить из программы (закомментировать) задержку, вставив символы «//» в начало строки `delay_loop(?)`;
- сохранить проект (Project → Save), выполнить компиляцию и линковку измененного проекта;
- запустив программу в режиме «Animate», наблюдать последовательность изменения логических уровней на линиях порта GPIOB.

8. Работа со сторожевым таймером.

8.1. Останавливаем выполнение программы.

8.2. В подпрограмме «InitSystem()» изменением содержимого регистра WDCR разрешаем работу сторожевого таймера (см. область 1 на рис.8.12). В битовом поле WDPS этого регистра задаем коэффициент деления программного предделителя равным 64 (см. рис. 8.8, 8.9).

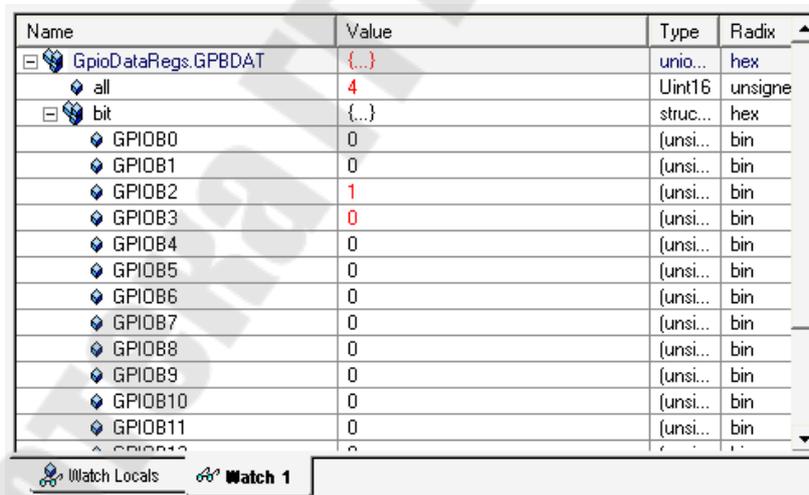
8.3. В подпрограмме временной задержки «delay_loop()» удаляем символы «//» перед двумя строками программы, в которых должна производиться запись в регистр WDKEY (см. область 7 на рис.8.12), и записываем кодовую комбинацию для сброса Watchdog-таймера: SysCtrlRegs.WDKEY = 0x55, SysCtrlRegs.WDKEY = 0xAA.

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

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

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

8.4. Компилируем, компоуем проект и тестируем программу.



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. 8.13. Просмотр состояния двоичных разрядов порта GPIOB в окне Watch Window

9. Работа с двумя светодиодами.

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

9.2. Преобразуем программу Lab2a.c так, чтобы одновременно зажигались два светодиода в направлении от периферии к центру (см. рис. 8.11, б).

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

9.4. Компилируем, компоуем проект и тестируем программу.

Часть II

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

1. Отображение состояния ключей на светодиодах.

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

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

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

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

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

1.4. Компилируем, компоуем проект и тестируем программу.

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

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

2.2. Преобразуем программу Lab2c.c. Задаем длительность задержки при переключении светодиодов согласно рис.8.11, а, равную $(0,1*N+0,001)$ сек, где N – код, заданный переключателями на линиях В15–В8. Подпрограммы «InitSystem()» и «Gpio_select()» оставляем без изменений.

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

2.4. Компилируем, компоуем проект и тестируем программу.

Содержание отчета

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

Контрольные вопросы

1. Структурная схема ЦСП семейства C28х.
2. Карта адресного пространства ЦСП C28х.
3. Структура портов ввода/вывода, режимы работы, регистры управления.
4. Вывод простейших управляющих сигналов через порты TMS320F2812.
5. Система тактирования TMS320F2812. Структура, особенности, назначение регистров высокоскоростного и низкоскоростного предделителей.
6. Watchdog timer: назначение, принцип действия, особенности.
7. Какие изменения необходимо внести в исходный текст, чтобы программа «бегущие огни» запускалась по нажатию кнопки, присоединенной к линии порта GPIOD1, а останавливалась – по нажатию кнопки, присоединенной к линии порта GPIOD6 («0» – соответствующая кнопка нажата, «1» – кнопка отжата)? В исходном состоянии обе кнопки отжаты.
8. Модифицируйте исходную программу таким образом, чтобы происходило движение одного «погашенного» светодиода среди остальных «горящих».

Исследование системы прерываний и таймеров ядра ЦСП семейства C28x

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.

Теоретические сведения

Система прерываний ядра процессора C28x содержит 16 линий прерываний (рис. 9.1). 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 C28x показана на рис. 9.2, а форматы регистров IER и IFR – на рис. 9.3. Следует отметить, что программная установка одного из битов в регистре IFR приведет к обработке данного прерывания ядра таким же образом, как и в случае возникновения соответствующего прерывания. INTM – младший бит в статусном регистре ST1, служит для общего разрешения/запрета маскируемых прерываний ядра.

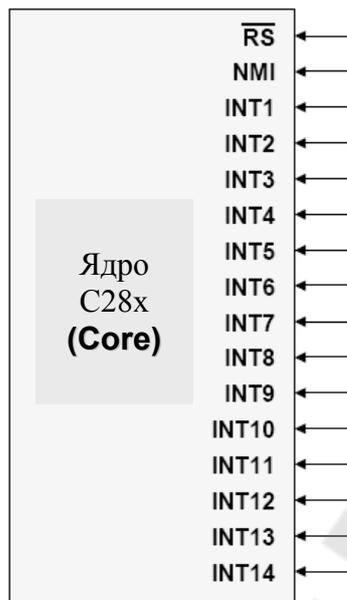


Рис. 9.1. Линии прерываний ядра C28x

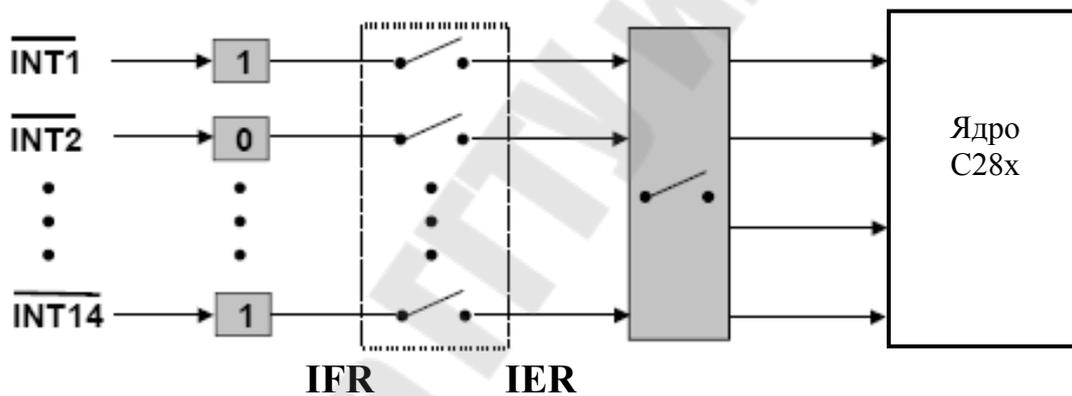


Рис. 9.2. Общая структура прерываний ядра C28x

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

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

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

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

Рис. 9.3. Форматы регистров IER и IFR

Все 16 прерываний однозначно связаны в памяти с таблицей векторов прерываний ядра BROM vector (рис. 9.2), 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, т.к. к 16 прерываниям ядра добавлены прерывания DLOGINT, RTOSINT, Illegal (недопустимая инструкция), 12 программных прерываний USER1...USER12 и один резервный вектор (Reserved).

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

Особенностью ЦСП семейства C28x является возможность запуска программ как из внутренней памяти (микроконтроллерный режим), так и из внешней (микропроцессорный режим). Режим определяется состоянием вывода XMP/MC (рис. 9.2). Соответственно, после сброса программа переходит либо к начальному адресу 0x3FFF0 внутренней памяти (при XMP/MC=0), либо к тому же адресу внешней памяти (при XMP/MC=1), а режим запоминается с помощью флага XMP/MC в регистре XINTCNF2, который может быть впоследствии обработан программно.

После сброса в режиме микроконтроллера запускается служебная программа Bootloader, которая анализирует выводы порта GPIOF (GPIOF2, GPIOF3, GPIOF4 и GPIOF12) и, исходя из

комбинации сигналов на них, выполняет один из переходов, перечисленных в табл. 9.1 и условно показанных на рис. 9.4.

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

Таблица 9.1

Режимы запуска программы Bootloader

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

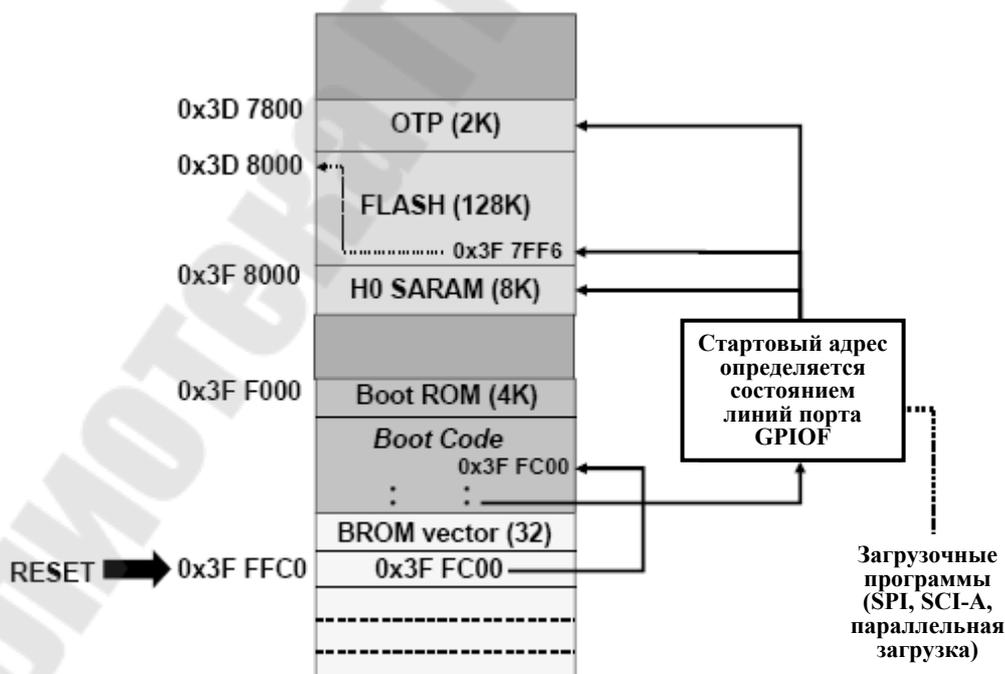


Рис. 9.4. Режимы запуска программы Bootloader

ЦСП имеет большое количество источников прерываний – 96, но только 16 линий прерываний ядра. Для возможности обслуживания всех прерываний для линий INT1...INT12 применяется мультиплексирование (рис. 9.5). Т.к. 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 – табл.векторов прерываний расширителя (PIE vector, адреса 0x00 D000...0x00 DFFF, см. рис. 8.2 и 3.9).

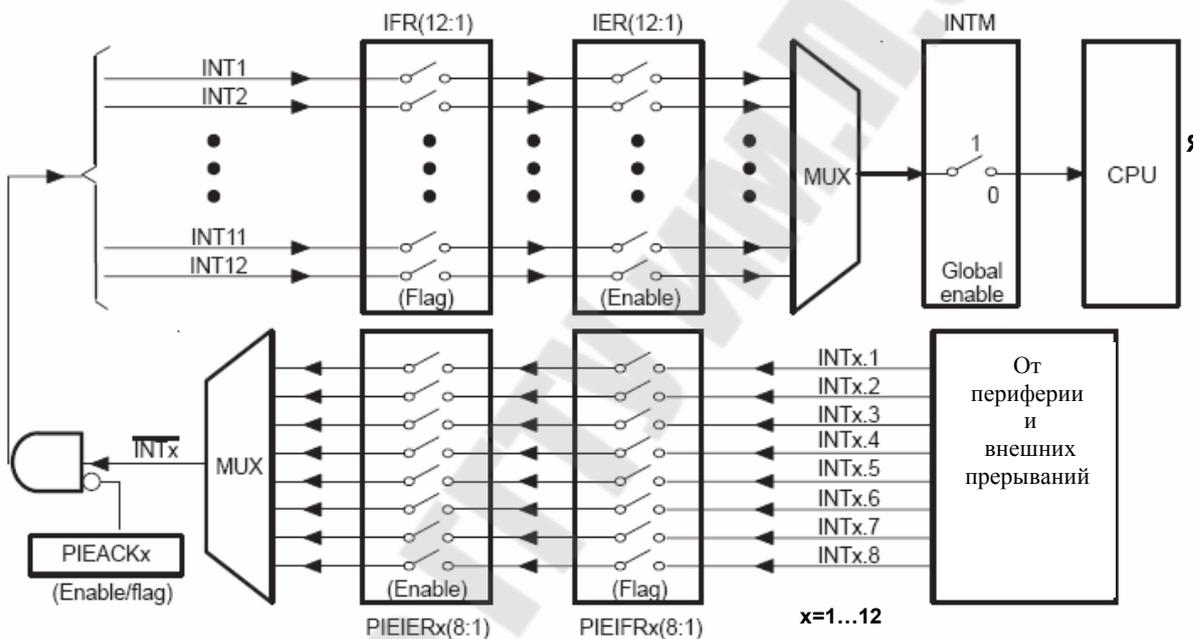


Рис. 9.5. Логика работы регистров PIEIFRx, PIEIERx, PIEACKx

Форматы регистров модуля расширителя прерываний PIEIERx и PIEIFRx показаны на рис. 9.7, данные регистры содержат по 8 информационных бит, т.е. по числу прерываний в группе. В регистре PIECTRL биты 15-1 (PIEVECT) показывают адрес в пределах таблицы векторов PIE vector, из которой был извлечен вектор. Младший значащий бит игнорируется и показываются биты адреса от 1 до 15 (т.е. только четные адреса), что позволяет при чтении из регистра однозначно определить, какое прерывание генерировалось.

ENPIE – бит разрешения извлечения векторов из таблицы PIE-контроллера. Если ENPIE=1, все вектора извлекаются из таблицы векторов PIE vector, а если ENPIE=0, PIE-контролер запрещен, и

вектора извлекаются из таблицы CPU-векторов (BROM vector, см. рис. 8.2).

Прерывания сгруппированы по 8 источников на линию (см. рис. 8.8). 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. 9.6).

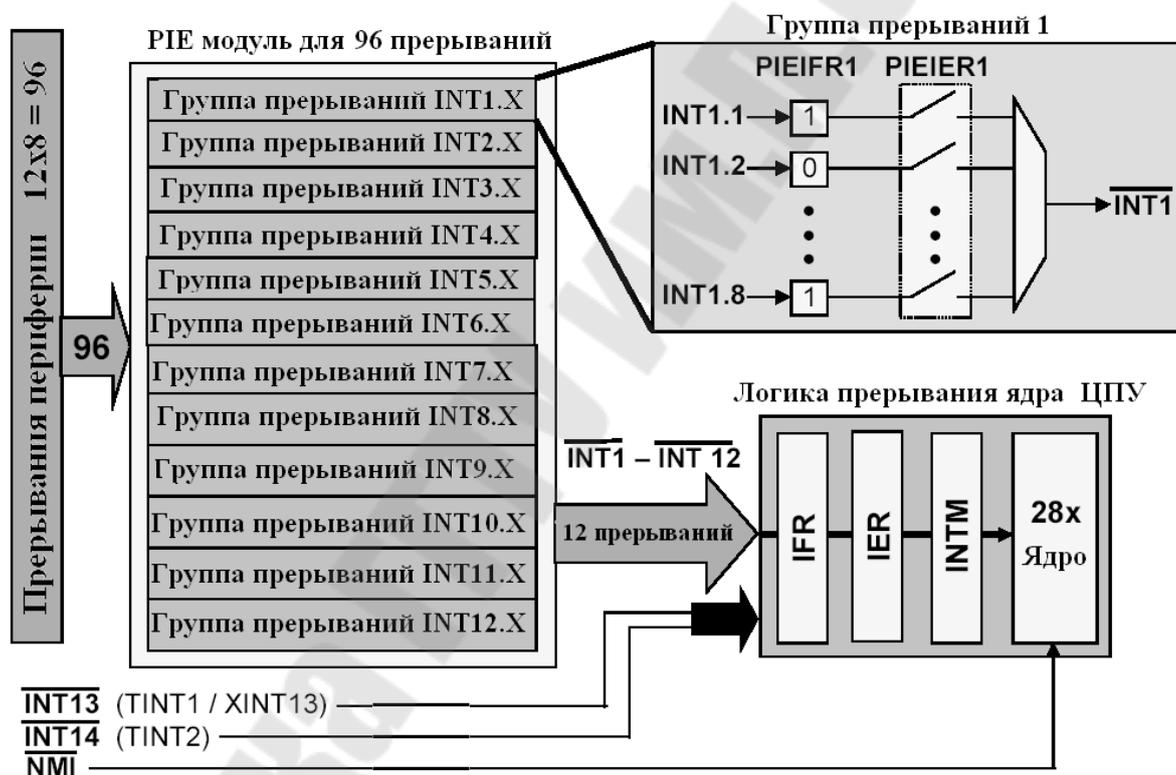


Рис. 9.6. Общая структура расширения прерываний ЦСП С28х

Регистры PIE



Рис. 9.7. Формат регистров модуля расширителя прерываний ЦСП C28x

Табл.векторов прерываний приведена на рис. 9.8.

Примеры векторов прерываний:

- прерывание от встроенного АЦП (ADCINT) – INT1.6;

- прерывание от CPU-таймера 0 (TINT0) – INT1.7.

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

Рис. 9.8. 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. 9.9. 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).

В сигнальных процессорах семейства C28x имеется три 32-битных таймера ядра (CPU timers) с одинаковой структурой. Схема одного таймера приведена на рис. 9.11. 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 TDDR: TDDR используется для перезагрузки прескалера таймера. Регистр PRDH: PRD содержит значение основного 32-битного счетчика, перезагружаемое в него при очередном переопустошении. Данная каскадная структура позволяет получить очень широкий диапазон коэффициентов деления частоты: от 2^2 до 2^{48} .

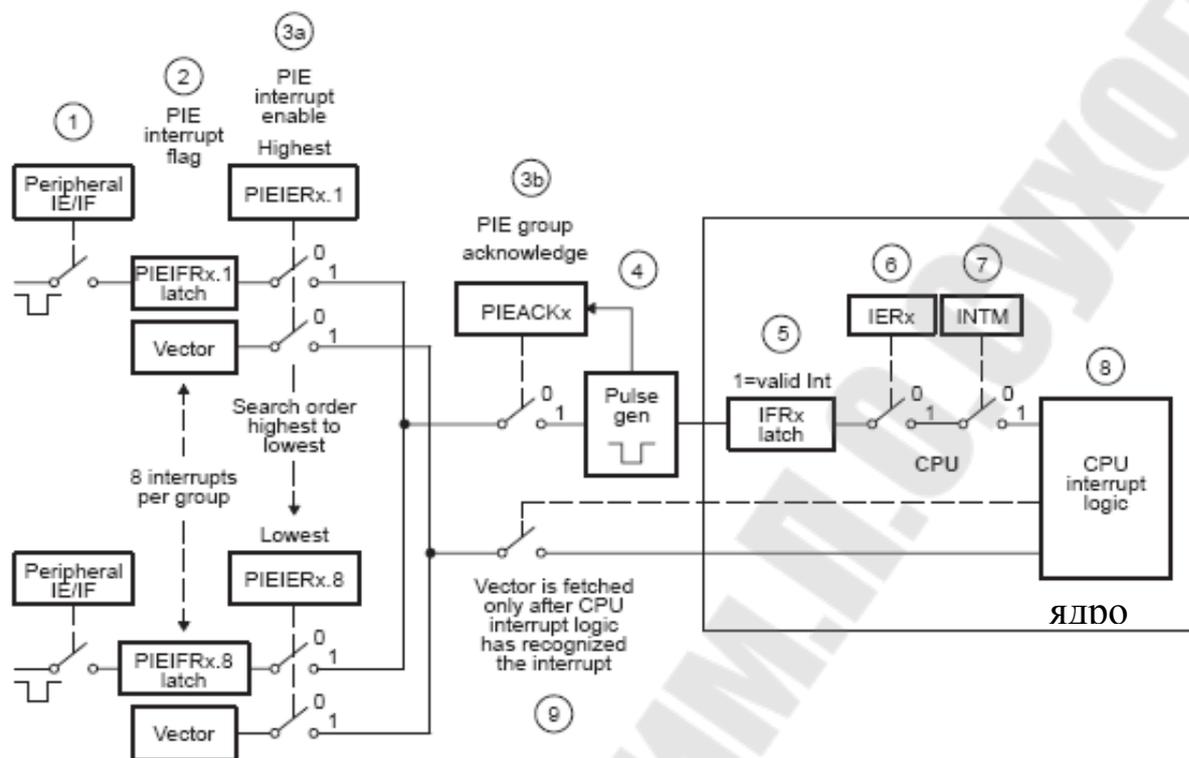


Рис. 9.10. Полная процедура обработки прерываний при $ENPIE=1$: шаг 1 – генерация прерывания от периферии; шаг 2 – установка флага $PIEIFR_{x.y} = 1$; шаг 3а – проверка одновременного наличия двух условий: $PIEIER_{x.y} = 1$ и $PIEACK_x = 0$; шаг 3б – установка в «1» бита $PIEACK_x$ для подтверждения 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 $0x00\ 0DFE$).

Таймеры 1 и 2, как правило, используются для операционной системы реального времени «DSP/BIOS», в то время как таймер 0 используется для пользовательских приложений. Данные таймеры интегрированы в ЦСП, не следует их путать с таймерами менеджеров

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. 9.12. Регистры таймеров ядра ЦСП семейства C28x

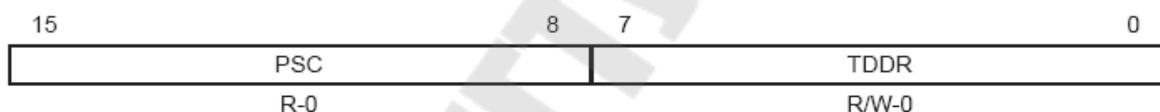


Рис. 9.13. Формат регистров-прескалеров TIMERxTPR (x = 0, 1, 2)

Функции управления каждым из таймеров реализованы в регистрах управления TIMERxTCR, формат которых показан на рис. 9.14.

Сигналы прерываний, формируемые CPU-таймерами, связаны с прерываниями ядра, как показано на рис. 9.15. Прерывание таймера 0 происходит через PIE, а таймеров 1 и 2 – попадают напрямую на ядро через соответствующие линии.

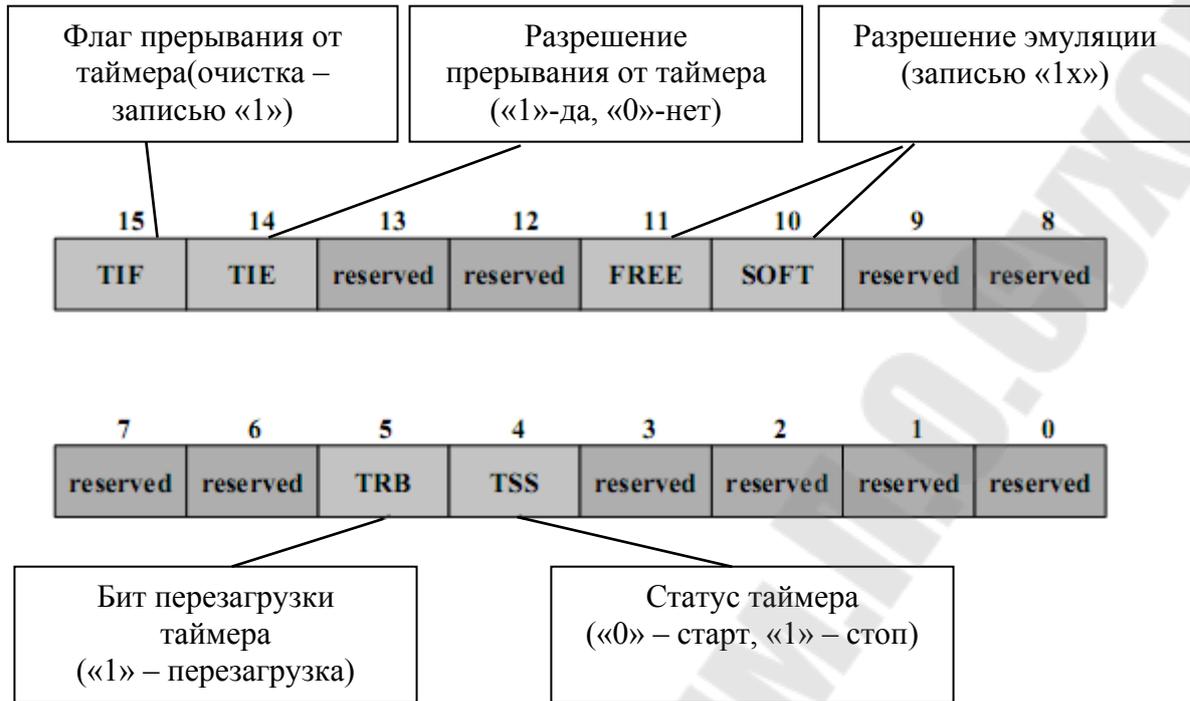


Рис. 9.14. Формат регистров управления таймерами ядра (TIMERxTCR)

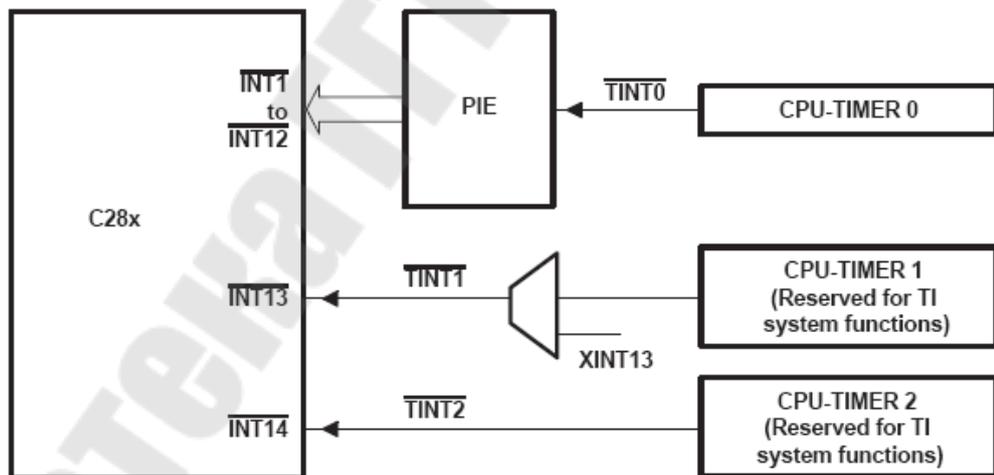


Рис. 9.15. Сигналы прерываний от CPU-таймеров

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.8.16.

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

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

2. Настройка параметров проекта.

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

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

2.2. Добавление библиотек и создание стека.

Подключаем Си-библиотеки:

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

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

Задаем глубину стека 0x400:

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

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

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

void delay_loop(long); ← 5
void Gpio_select(void);
void InitSystem(void); ← 1

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 ← 2

    while(1)
    {
        for(i=0;i<14;i++)
        {
            if(i<7)      GpioDataRegs.GPBDAT.all = LED[i];
            else         GpioDataRegs.GPBDAT.all = LED[14-i];
            delay_loop(?); ← 4
        }
    }
} ← 3

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

void delay_loop(long end)
{
```

```

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

#####
// Подпрограмма:   Gpio_select
//
// Описание:   Настройка линий порта GPIO В15-8 на ввод, а
//             линий В7-0
//             на вывод. Настройка всех линий портов А, D, F, Е, 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 А, D, Е, F, G
на ввод
    GpioMuxRegs.GPBDIR.all = 0x?; // Настройка линий 15-8 на ввод,
    GpioMuxRegs.GPDDIR.all = 0x?; // а линий 7-0 на вывод
    GpioMuxRegs.GPEDIR.all = 0x?;
    GpioMuxRegs.GPFDIR.all = 0x?;
    GpioMuxRegs.GPGDIR.all = 0x?;

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

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

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

```

7

6

```

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

        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. 8.16. Структура исходного текста программы

3. Редактирование программы.

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

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

Место вставки – область 1 на рис. 9.16.

Далее в основной программе добавляем вызов подпрограммы:

```
InitPieCtrl();
```

Место вставки – область 2 на рис. 9.16.

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 табл. прерываний «PieVectTableInit()», которая определена в файле DSP281x_PieVect.c, и копирует эту табл. в глобальную переменную «PieVectTable», которая связана с областью памяти процессора PIE area. Также, для использования подпрограммы InitPieVectTable, в проект добавлен файл DSP281x_DefaultIsr.c, который добавляет в проект подпрограммы обработки прерываний.

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

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

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

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

```
InitCpuTimers();
```

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

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

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

Выполнение данной подпрограммы реализует файл DSP281x_CpuTimers.c.

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

Первый уровень – модуль PIE, группа PIEIER1 (т.к. прерывания от CPU-таймера 0 относятся именно к данной группе, рис.8.8):

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

Второй уровень – разрешение прерываний линии 1 ядра ЦПУ. Для этого необходимо настроить регистр IER (рис. 8.2 и 3.3).

```
IER = 1;
```

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

```
EINT;  
ERTM;
```

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

3.7. Затем необходимо добавить команду запуска CPU Timer0:

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

3.8. Сразу после основной программы (область 3 на рис. 9.16) необходимо добавить подпрограмму обработки прерываний от CPU-таймера 0 «cpu_timer0_isr». Подпрограмму и обращение к ней мы уже внесли в программу. Теперь необходимо написать саму подпрограмму. Подпрограмма будет иметь общий вид:

```
interrupt void cpu_timer0_isr(void)  
{  
...  
(текст подпрограммы)  
...  
}
```

Подпрограмма должна выполнять следующие действия:

- инкрементировать глобальную переменную «CpuTimer0.InterruptCount», описываемую (начальное значение = 0) в подключаемом файле DSP281x_CpuTimers.c. Данная переменная будет показывать истечение интервала времени 50 мс с момента запуска таймера;

- сбросить в «0» бит 1 регистра PIEACK, что необходимо для разрешения последующих прерываний. Это действие выполняет команда:

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

3.9. После настройки таймера и прерываний корректируем основную программу. Для этого удаляем (либо добавляем в начале строки признак комментария «//») вызов подпрограммы «delay_loop(?)» (см. область 4 на рис. 9.16). Также удаляем объявление этой подпрограммы (см. область 5 на рис. 9.16), 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. 9.16. После цикла ожидания необходимо сбросить переменную «CpuTimer0.InterruptCount» в 0.

3.10. Разрешаем работу Watchdog timer (WDT). См. область 6 на рис. 9.16.

3.11. Добавляем обслуживание WDT. Для этого необходимо последовательно записать в регистр WDKKEY коды «0x55» и «0xAA», аналогично тому, как это выполнялось в программе к лабораторной работе № 2 (см. область 7 на рис. 9.16). Отличием является то, что данную процедуру вместе с макросами EALLOW и EDIS необходимо 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. Компоуем проект: Project → Build. При наличии ошибок, исправляем их.

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

Содержание отчета

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

Контрольные вопросы

1. Маскируемые и немаскируемые прерывания.
2. Сброс ЦСП.
3. Обработка прерываний ядра ЦСП семейства C28х.
4. Назначение и структура модуля расширения прерываний (PIE) ЦСП семейства C28х.
5. Расположение векторов прерываний ЦСП семейства C28х в режимах ENPIE=0 и ENPIE=1.
6. Процедура обработки прерываний при ENPIE=1.
7. Назначение и структура CPU-таймеров ЦСП семейства C28х.
8. Регистры CPU-таймеров ЦСП семейства C28х.
9. Какие изменения необходимо внести в исходный текст, чтобы программа «бегущие огни» выполнялась с движением «горящих» светодиодов:
 - в 1,7 раз быстрее?
 - в 2,45 раз медленнее?
10. Какие изменения необходимо внести в исходный текст, чтобы программа «бегущие огни» выполнялась:
 - с замедлением движения «горящих» светодиодов в 2 раза на каждом шаге движения?
 - с ускорением движения «горящих» светодиодов в 3 раза на каждом шаге движения?

Исследование модуля Менеджера Событий ЦСП TMS320F2812

Цель работы

Изучить аппаратные и программные возможности Менеджера Событий (Event Manager) DSP TMS320F2812.

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.

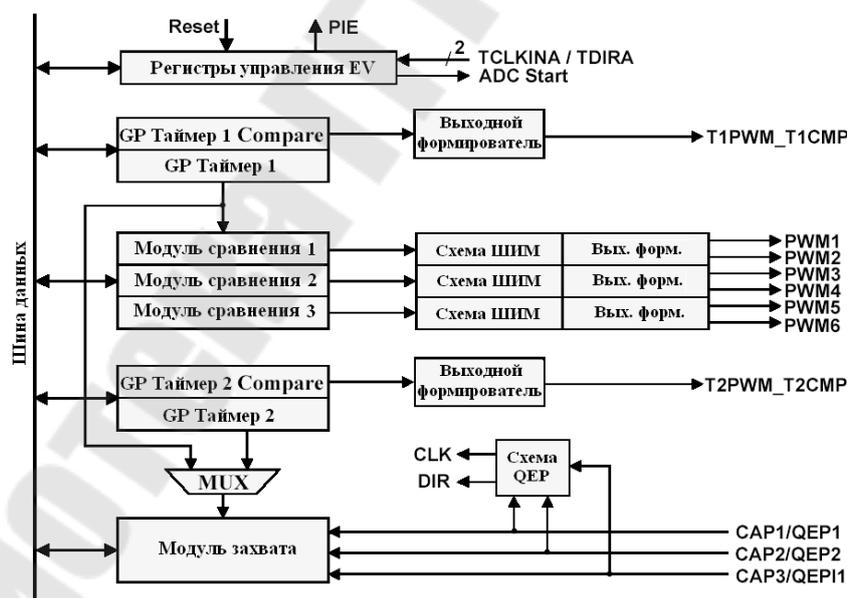


Рис. 10.1. Менеджер Событий

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

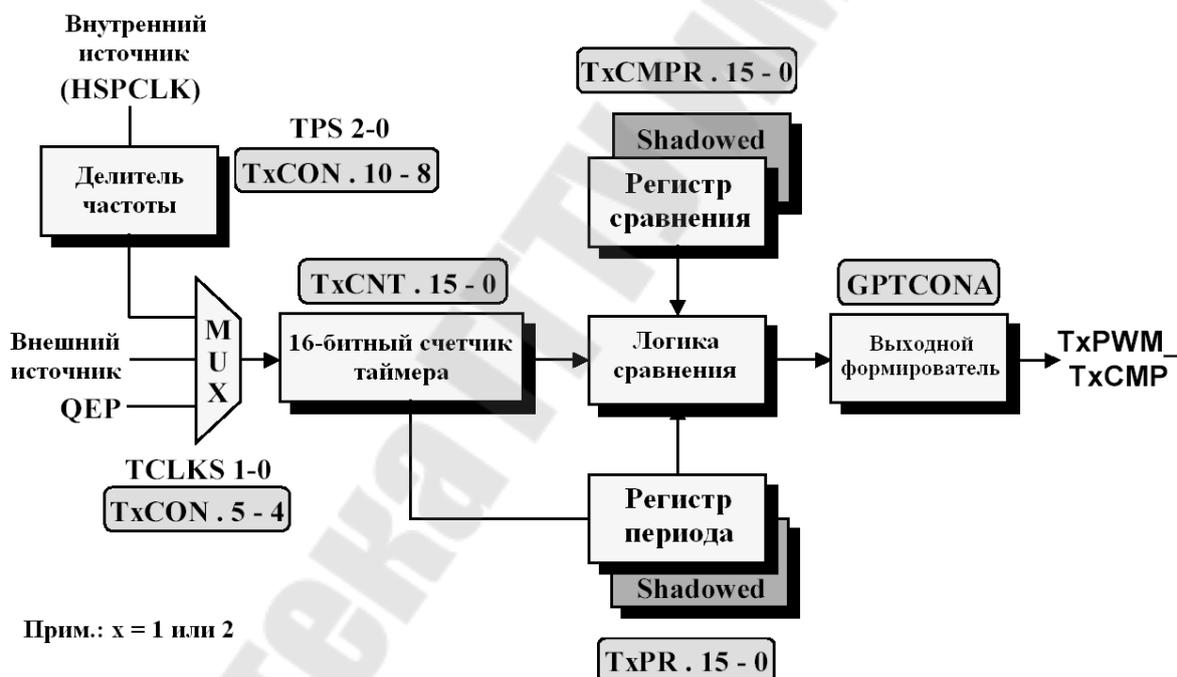
или EVBSOC (которые мультиплексируются с сигналами $\overline{T2CTrip}$ и $\overline{T4CTrip}$), вырабатывается строб начала преобразования (SOC).

На рис. 10.1 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.



Прим.: $x = 1$ или 2

Рис. 10.2. Таймер общего назначения модуля EVM

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

выходной сигнал (TxPWM). При достижении счетчиком значения регистра периода TxPWM сбрасывается.

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

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

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

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

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

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

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

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

2) *Непрерывный режим счета вверх*. В этом режиме значение счетчика увеличивается до тех пор, пока не достигнет значения, равного значению в регистре периода (рис. 10.3). 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.

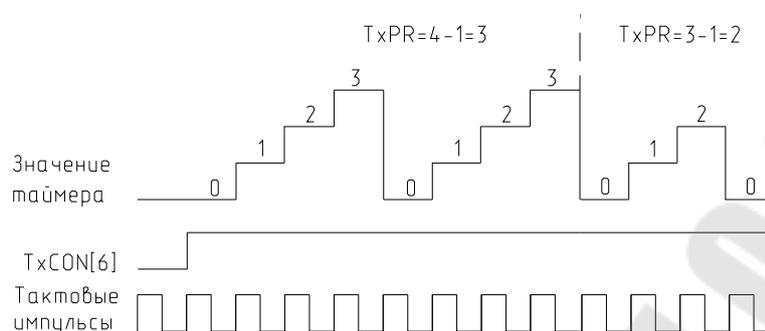


Рис. 10.3. Режим непрерывного счета вверх GP таймера

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

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

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

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

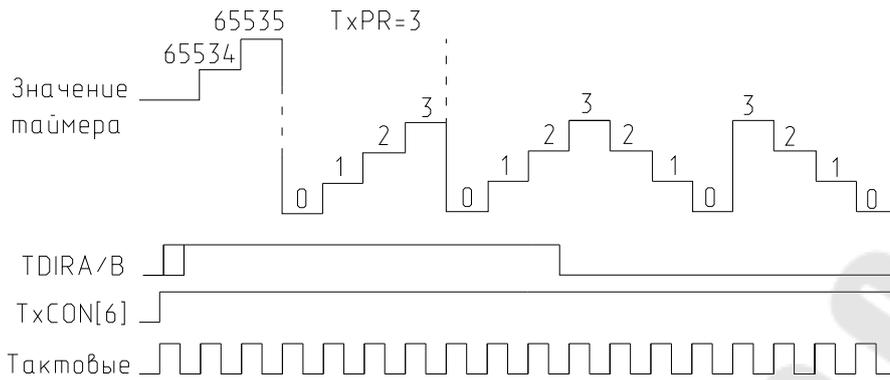


Рис. 10.4. Режим управляемого счета вверх/вниз GP таймера

4) *Непрерывный режим счета вверх/вниз.* В отличие от предыдущего режима, направление счета изменяется при достижении нуля или значения в регистре периода. Продолжительность периода в этом режиме равна $2 \cdot (TxPR)$.

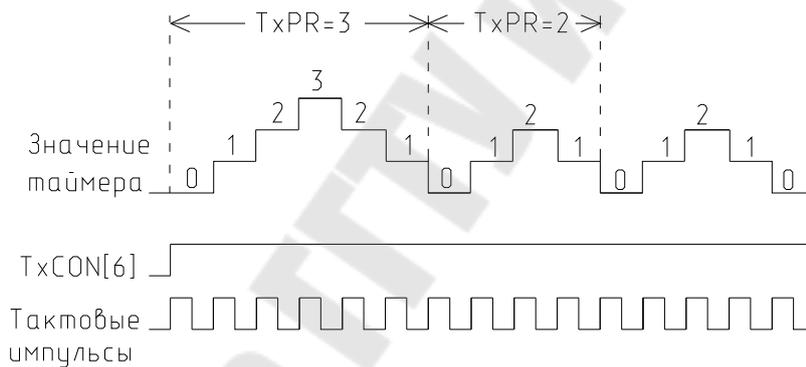


Рис. 10.5. Непрерывный режим счета вверх/вниз

Устройство захвата (Capture Unit) предназначено для определения временных параметров внешних сигналов. Значение выбранного GP таймера захватывается и запоминается в 2-уровневом стеке FIFO, когда на соответствующих выводах фиксируется заданный перепад уровней. Устройство захвата состоит из 3-х цепей CAP_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) имеется 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 минимума программную загрузку ядра при операциях измерений длительности, периодических выборок и генерации сигналов ШИМ.

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

3. Формат регистров модуля Менеджера Событий

На рис. 10.6 изображен формат регистра счетчика GP таймера.

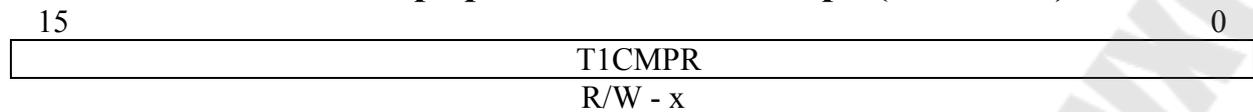


Биты	Название	Описание
15:0	TxCNT	Мгновенное значение таймера-счетчика 1

Рис. 10.6. Формат регистра счетчика GP таймера.

На рис. 10.7 изображен формат регистра сравнения GP таймера

Регистр сравнения GP таймера (TxCMPR)

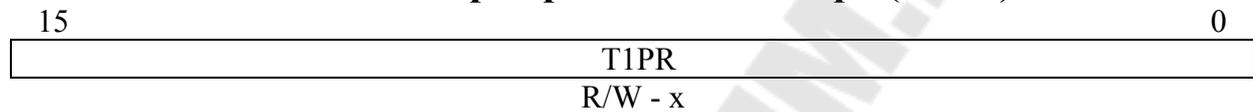


Биты	Название	Описание
15:0	TxCMPR	Хранимое значение сравнения таймера-счетчика 1

Рис. 10.7. Формат регистра сравнения GP таймера

На рис. 10.8. изображен формат регистра периода GP таймера

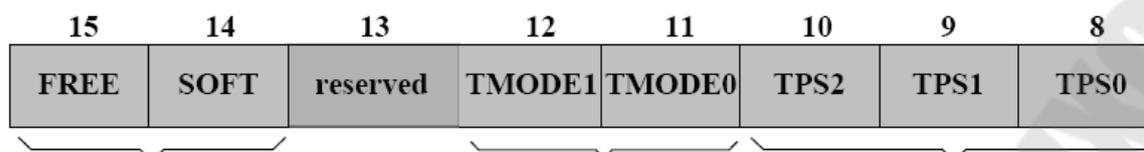
Регистр периода GP таймера (TxPR)



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

Рис. 10.8. Формат регистра периода GP таймера

Старший байт:



Биты управления эмулятором:

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

Прескалер таймера:

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

Режим работы:

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

Младший байт:

Разрешение схемы сравнения:

- 0 Запретить
- 1 Разрешить

Запуск таймера:

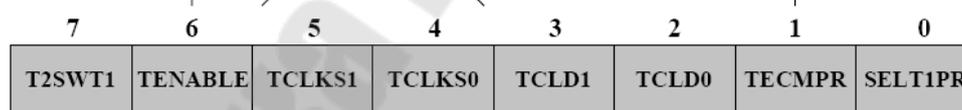
- 0 Останов таймера (сброс счетчика и делителя)
- 1 Запуск таймера

Источник тактирования:

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

Выбор регистра сравнения:

- 0 Свой регистр сравнения
- 1 Регистр сравнения Таймера1



Запуск вместе с Таймером 1

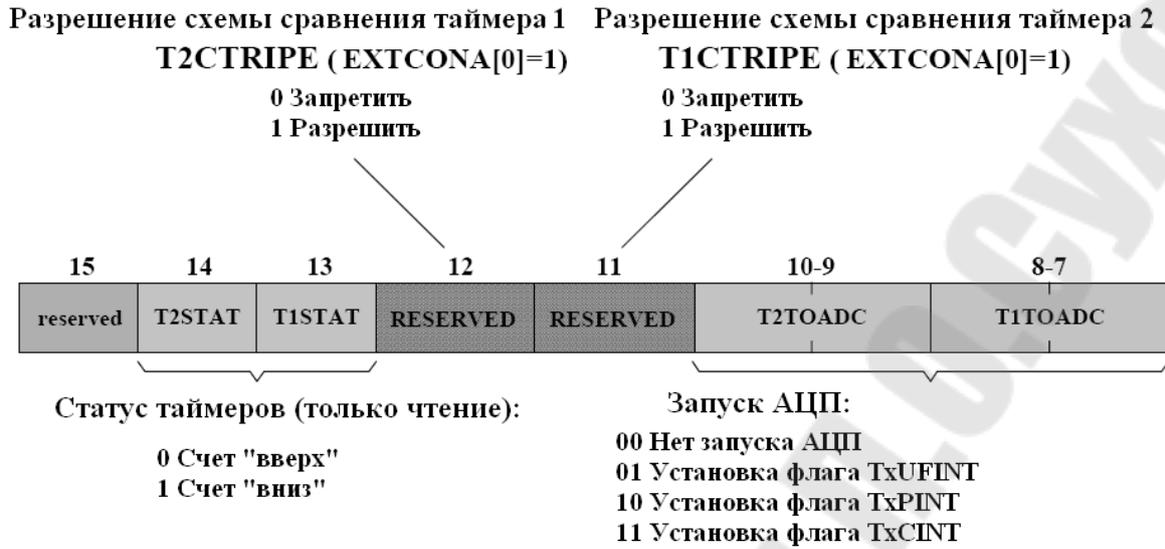
- 0 Использовать свой бит запуска (TENABLE)
- 1 Использовать TENABLE Таймера 1

Перезагрузка регистра сравнения:

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

Рис. 10.9 Регистры управления таймером (TxCON)

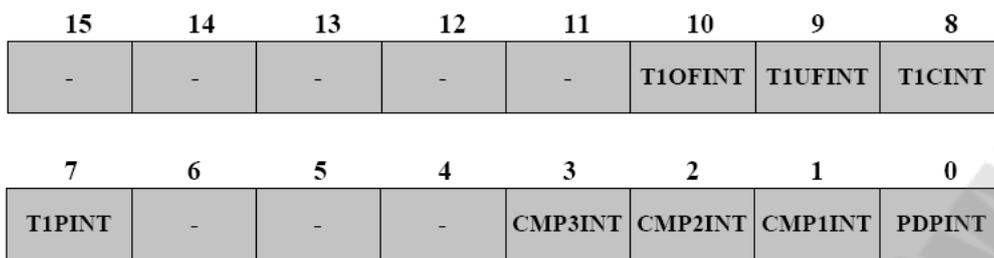
Старший байт:



Младший байт:

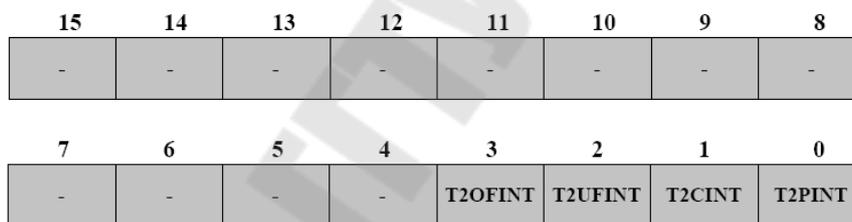


Рис. 10.10. Регистр А управления GP таймером (GPTCONA)



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

Рис. 10.11. Формат регистра А маски прерываний (EVAIMRA)



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

Рис. 10.12. Формат регистра В маски прерываний (EVAIMRB)

15	14	13	12	11	10	9	8
-	-	-	-	-	-	-	-
7	6	5	4	3	2	1	0
-	-	-	-	-	CAP3INT	CAP2INT	CAP1INT

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

Рис. 10.13. Формат регистра С маски прерываний (EVAIMRC)

EVAIFRA	15	14	13	12	11	10	9	8
	-	-	-	-	-	T1OFINT	T1UFINT	T1CINT
Чтение: 0 - нет прер. 1 - запрос прер.	7	6	5	4	3	2	1	0
	T1PINT	-	-	-	CMP3INT	CMP2INT	CMP1INT	PDPINT
EVAIFRB	15	14	13	12	11	10	9	8
	-	-	-	-	-	-	-	-
Запись: 0 - без изм. 1 - сброс флага	7	6	5	4	3	2	1	0
	-	-	-	-	T2OFINT	T2UFINT	T2CINT	T2PINT
EVAIFRA	15	14	13	12	11	10	9	8
	-	-	-	-	-	-	-	-
	7	6	5	4	3	2	1	0
	-	-	-	-	-	CAP3INT	CAP2INT	CAP1INT

Рис. 10.14. 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 т.ч. 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. Таблица представлена 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 необходимо производить сброс Watchdog и загружать новую ноту. Обратите внимание на период между прерываниями таймера, периодом Watchdog и периодом перезагрузки новой ноты.

Таблица 10.1

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:\tides\c28\dsp281x\v100\DSP281x_common\source\DSP281x_CpuTimers.c
C:\tides\c28\dsp281x\v100\DSP281x_common\source\DSP281x_DefaultIsr.c
C:\tides\c28\dsp281x\v100\DSP281x_common\source\DSP281x_PieCtrl.c
C:\tides\c28\dsp281x\v100\DSP281x_common\source\DSP281x_PieVect.c
C:\tides\c28\dsp281x\v100\DSP281x_headers\source\DSP281x_GlobalVariableDefs.c
C:\tides\c28\dsp281x\v100\DSP281x_common\cmd\F2812_EzDSP_RAM_lnk.cmd
C:\tides\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:\tides\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 данной лабораторной работе: массив LED[8].

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

В подпрограмме «InitSystem» разрешаем работу Менеджера Событий А.

Внутри главной подпрограммы записываем:

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

Настраиваем таймер 1 Менеджера Событий на выработку модулированного сигнала PWM:

Бит «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

$$T1_PWM_Freq = \frac{150\text{ MHz}}{HISPCP \cdot TPS \cdot T1PR} \quad (1)$$

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

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

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

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

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

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. представлено соответствие частот и нот музыкального звукоряда.

Для расчета частоты синусоидальных колебаний используется формула:

$$f_{\sin} = \frac{f_{PWM}}{N_{PWM}} \quad (2)$$

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 (см. 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.п. 2.1 – 2.4 части I данной лабораторной работы). В закладке Compiler выбираем Preprocessor и в поле Include Search Path (-i) вводим:

```
C:\tids\C28\dsp281x\v100\DSP281x_headers\include;..\include;
C:\tids\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 ПИЕ 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:

$$f_{PWM} = \frac{f_{CPU}}{T1PR \cdot TPS_{T1} \cdot HISCP} \quad (3)$$

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

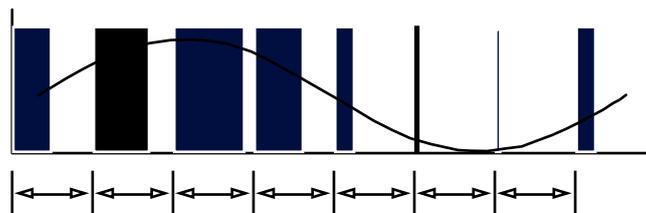


Рис. 10.15. Принцип формирования синусоидального сигнала широтно-импульсной модуляцией

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 “IQ-Math” в проект.

Добавляем в начальную часть текста программы строки:

```
#include "IQmathLib.h",
    _iq30_sine_table[512],
#pragma DATA_SECTION(sine_table, "IQmathTables");
```

и добавляем файл Lab4A.cmd.

6. Расчет необходимого значения регистра сравнения.

Учитывая, что разность значений регистра периода и регистра сравнения определяет ширину выходного модулированного импульса (см. рис.9.16), а значения функции синуса изменяются от минус 1 до плюс 1, рассчитываем значение регистра сравнения по формуле:

$$T1CMPR = T1PR - _IQ30mpy(\text{sine_table}[\text{index}] + _IQ30(0.9999), T1PR/2), \quad (4)$$

где функция $_IQ30mpy(a, b)$ – умножение чисел a и b в формате $_IQ30$; $_IQ30(0.9999)$ – оттранслированное значение единицы.

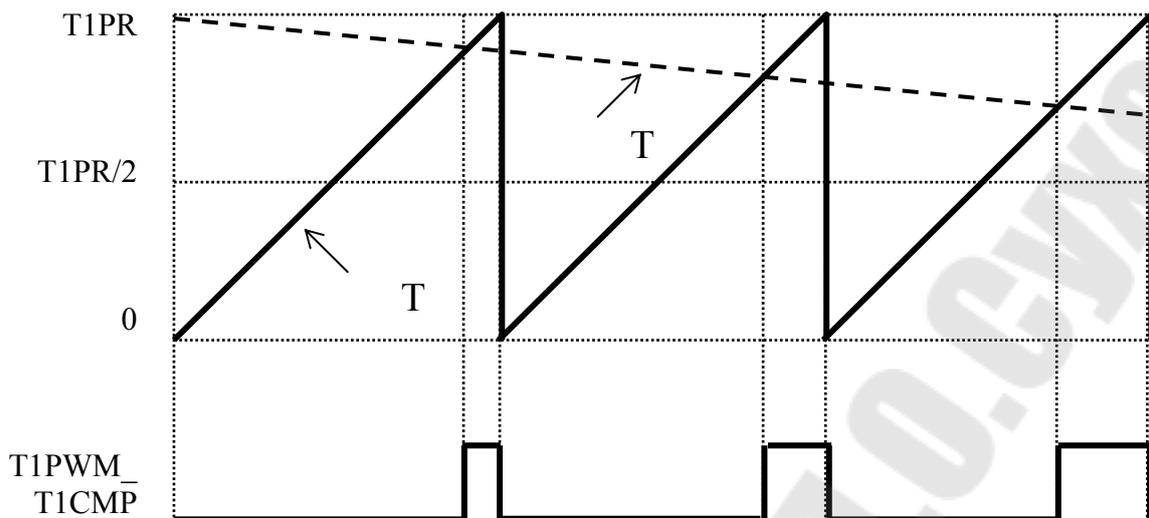


Рис. 10.16. Изменение ширины импульса при ШИМ по синусоидальному закону

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:

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

7. Расчет частоты синусоидальных колебаний.

Для этого используем формулу (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.

Исследование модуля АЦП ЦСП TMS320F2812

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. 11.1. 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.

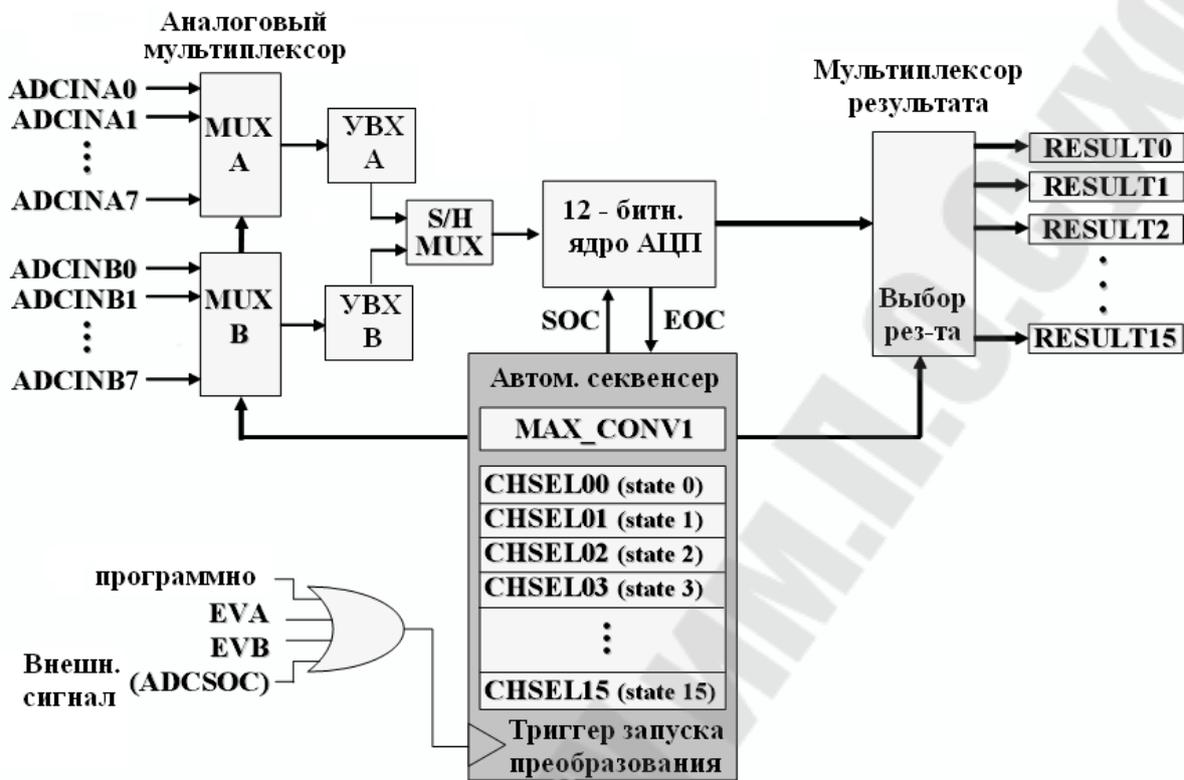


Рис. 11.1. Блок-схема модуля АЦП в каскадном режиме

В *двухканальном режиме* автоматический секвенсер разделяется на две независимые части («SEQ1» и «SEQ2») со своими настройками и сигналами запуска. Входные каналы задаются в регистрах CHSEL00 .. CHSEL07 для последовательности SEQ1 и CHSEL08 .. CHSEL15 для последовательности SEQ2, результаты преобразования сохраняются в регистрах RESULT0 .. RESULT7 и RESULT8 .. RESULT15 соответственно. Для любой из двух последовательностей может быть задан любой из 16 входных каналов. Данный режим позволяет получить фактически два независимых АЦП, со своими регистрами управления и сигналами запуска. Арбитр последовательности используется в случае одновременного появления сигналов запуска от двух последовательностей. В таком случае приоритет имеет SEQ1, преобразование SEQ2 будет задержано до окончания SEQ1.

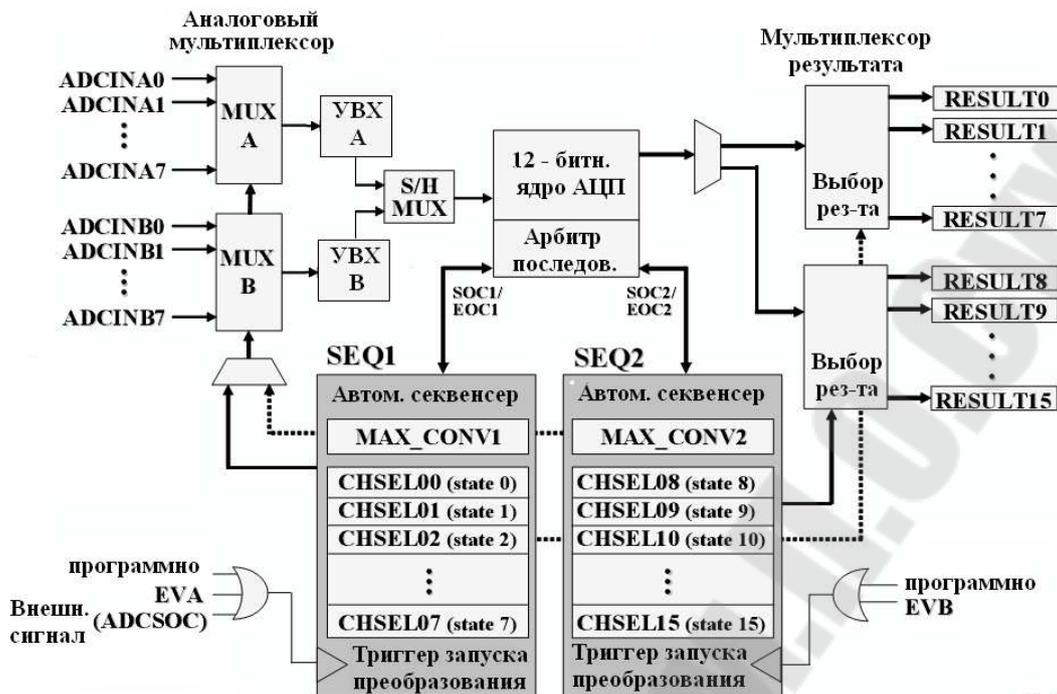


Рис. 11.2. Блок-схема модуля АЦП в двухканальном режиме

2. Время преобразования и система тактирования АЦП

Рассмотрим настройку тактирования АЦП на примере. Тактирование АЦП осуществляется от высокоскоростного прескалера периферии HSPCLK (рис. 11.3).

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_PC можно задать необходимое время захвата сигнала. За это время сигнал на UBХ 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.

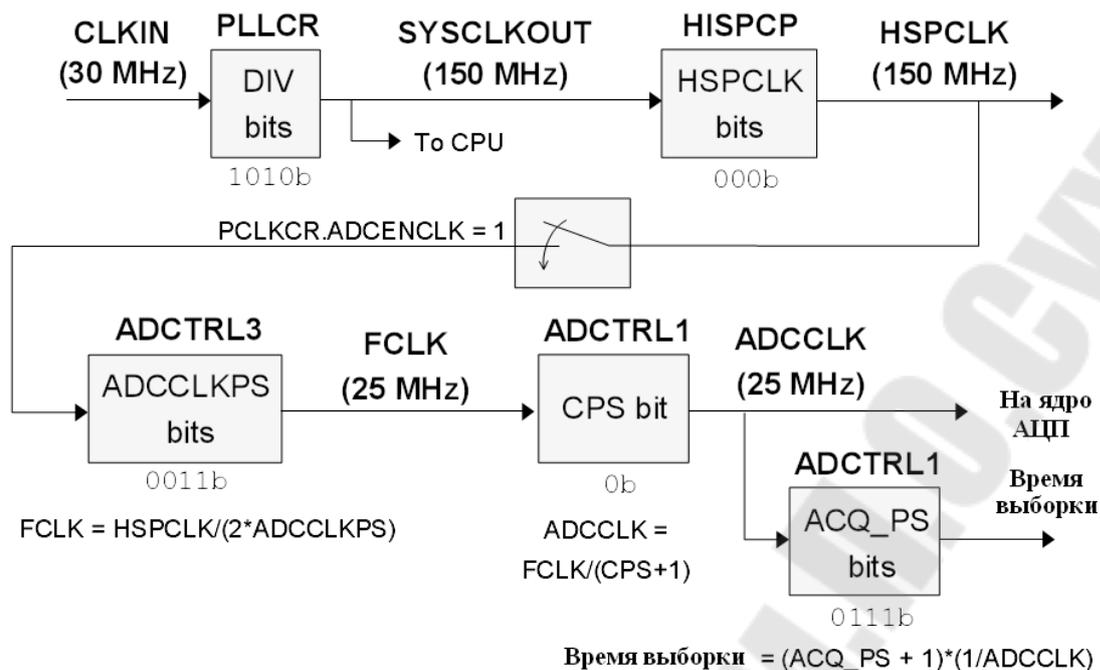


Рис. 11.3. Пример конфигурации модуля тактирования АЦП

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

3. Формат регистров модуля АЦП

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

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

Старший байт

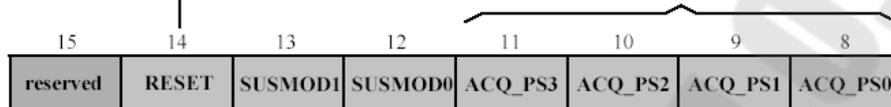
Сброс АЦП

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

Время выборки:

$$t = \text{ACQ_PS0..3} + 1$$

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

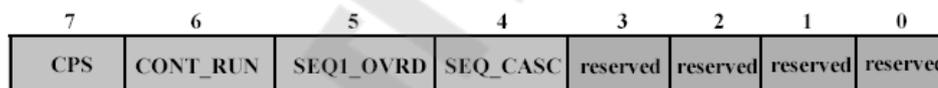
Младший байт

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. 11.4. Регистр 1 управления АЦП (ADCTRL1)

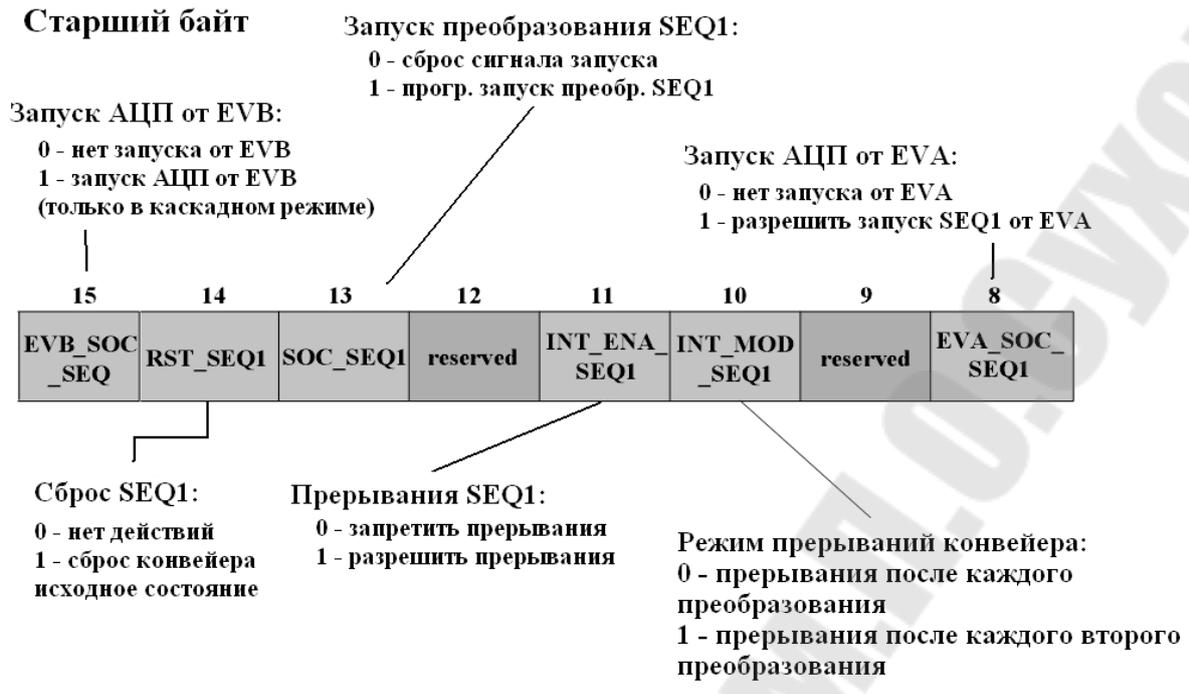
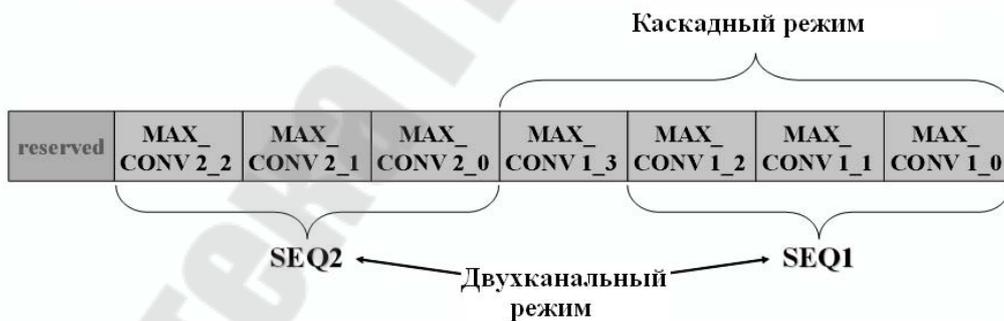


Рис. 11.5. Регистр 2 управления АЦП (ADCTRL2)



Рис. 11.6. Регистр 3 управления АЦП (ADCTRL3)

◆ 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. 11.7. Формат регистра регистра числа каналов для преобразования ADCMAXCONV

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

Рис. 11.8 Регистр статуса автопоследовательности (ADCASEQASR)

Каждый набор из 4-х бит CONV_{nn}, выбирает один из 16 аналоговых входов АЦП для последовательного автоматического преобразования.

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

Буферный регистр результата АЦП (ADCRESULTn)

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

Рис. 11.9. 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 (см. лаб. раб. № 8), устанавливаем коэффициентом деления 64 (регистр WDCR), сбрасываем сторожевой таймер (регистр SCSR).

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

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

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

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

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

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

3.4. Сбрасываем биты регистров GPxQUAL в ноль.

4. Инициализация модуля Менеджера Событий.

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

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

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

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

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

5. Инициализация модуля АЦП.

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

5.2. Записываем количество преобразований 2 (регистр ADCMAXCONV).

5.3. Выбираем каналы ADCIN_A0 и ADCIN_B0 (регистр ADCCHSELSEQ1).

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

5.5. 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 секунду. В программе также необходимо сбрасывать сторожевой таймер процессора.

8.2. Компонуем 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 № 8. 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.

Исследование встроенного CAN-интерфейса DSP TMS320F2812

Цель работы

Изучить характеристики встроенного CAN-интерфейса ЦСП TMS320F2812.

Теоретические сведения

1. Особенности CAN-интерфейса DSP TMS320F2812

Модуль eCAN имеет следующие особенности:

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

9) Режим самопроверки:
 - работа в режиме петли; прием своего собственного сообщения, что устраняет потребность в другом узле для определения бита признака.

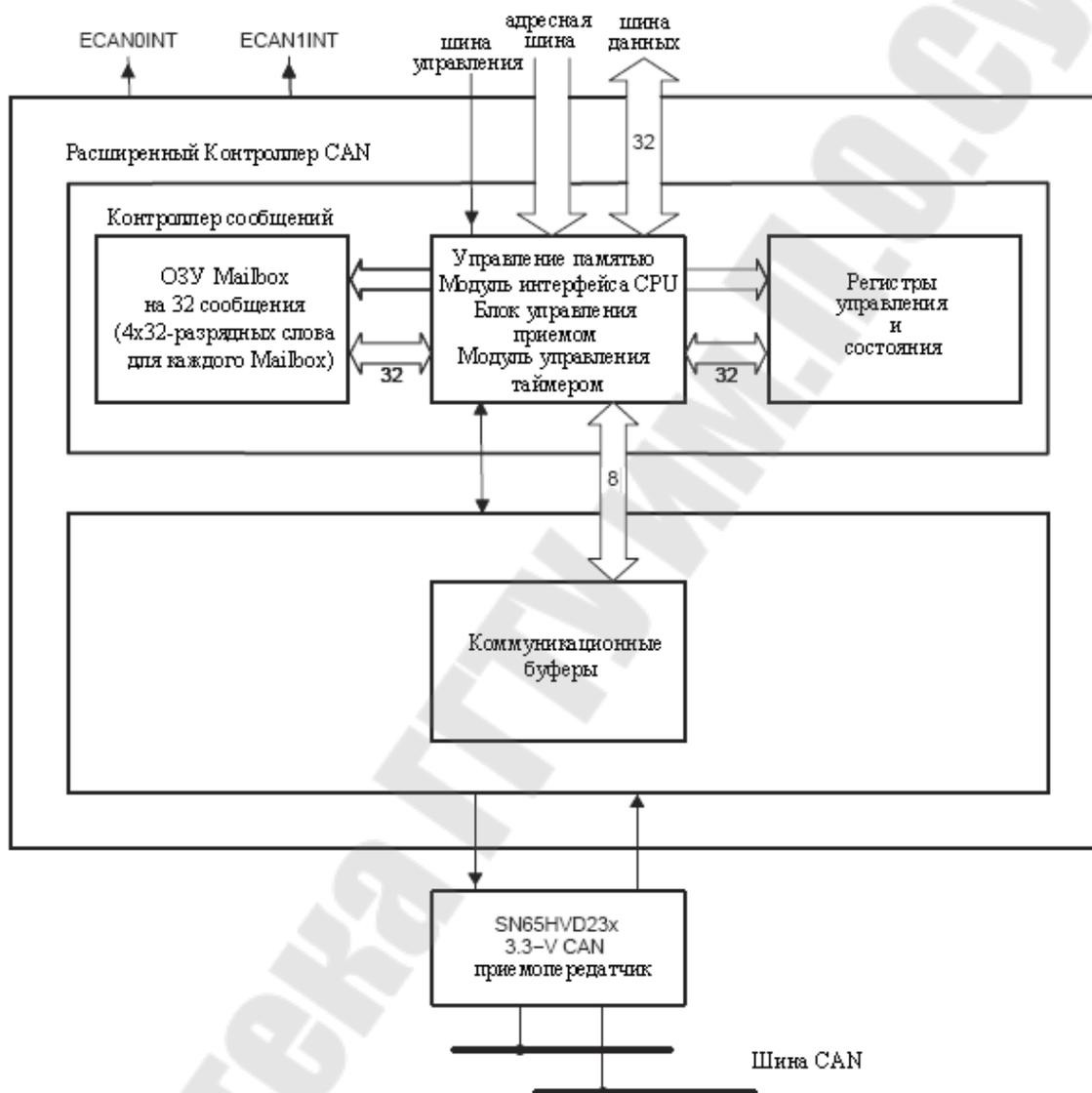


Рис. 12.1. Блок-схема CAN-интерфейса в ЦСП TMS320F2812

2. Совместимость eCAN с другими CAN-модулями

eCAN-модуль идентичен "Высокопроизводительному CAN-контроллеру (HECC)", используемому в микроконтроллерах серии TMS470 фирмы Texas Instruments, с некоторыми незначительными изменениями. Также модуль eCAN представлен более широко

(увеличенное число Mailbox с индивидуальными приемными масками, временные метки и т.д.) в сравнении с CAN-модулем ЦСП семейства 240х. Поэтому программы, написанные для CAN-модуля 240х, не могут применяться к eCAN 320х. Однако eCAN обладает той же самой структурой размещения разрядов регистров и функциональными возможностями, как и CAN серии 240х (для регистров, существующих в обоих устройствах), т.е. большинство регистров и битов выполняют идентичные функции на двух платформах. Это упрощает перемещение программ, написанных на языке C++.

2.1. Модуль и сеть CAN

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

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

2.2. Протоколы CAN

CAN поддерживает четыре типа различных типов кадров для передачи:

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

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

Сообщения CAN со стандартными кадрами данных содержат от 44 до 108 битов, а сообщения CAN с расширенными кадрами данных содержат от 64 до 128 битов. Кроме того, могут быть добавлены до 23 битов данных в стандартном кадре и до 28 данных в расширенном кадре данных, в зависимости от кодировки потока данных. Полная максимальная длина сообщения – 131 бит со стандартным кадром и 156 бит с расширенным кадром.

Поля битов, находящиеся в стандартных/расширенных кадрах данных, расположенные как показано на рис. 12.2, включают в себя следующее:

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



Рис. 12.2. Структура кадра сообщения стандартного формата

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

Архитектура CAN-модуля, показанная на рис. 12.3, состоит из ядра протоколов (СРК) и контроллера сообщений.

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

Контроллер сообщений решает, должно ли принятое в СРК сообщение быть сохранено для использования CPU, или нет. При инициализации CPU передает контроллеру сообщений идентификаторы всех сообщений, которые будут использоваться. Контроллер сообщений также отвечает за передачу следующего сообщения из СРК согласно приоритету сообщений.

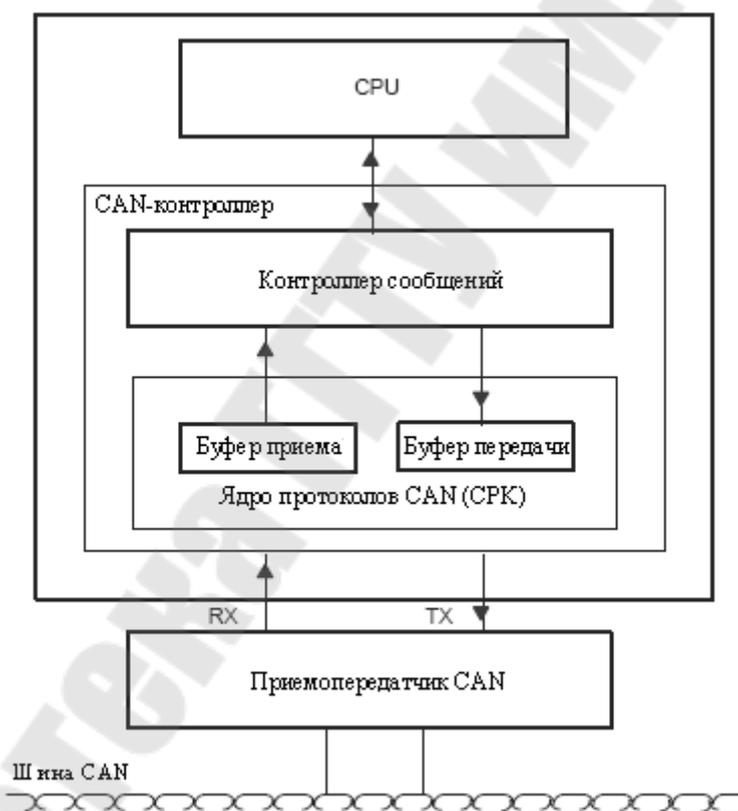


Рис. 12.3. Архитектура модуля eCAN

2.3. Контроллеры eCAN

eCAN-модуль состоит из:

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

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

После приема сообщения СРК определяет блок управления приемом в контроллере сообщений, который решает, сохранять принятое сообщение в одну из 32 ячеек ОЗУ Mailbox или нет. Блок управления анализирует идентификатор и сравнивает с масками Mailbox для определения номера Mailbox, в котором будет сохранено сообщение. При совпадении маски и идентификатора принятое сообщение сохраняется в соответствующий Mailbox. Если блок управления приемом не обнаружил совпадения маски и идентификатора, то сообщение игнорируется.

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

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

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

Для инициализации переданных данных бит запроса передачи должен быть установлен в соответствующем регистре управления. Тогда процедура передачи и обработки сообщений выполняется без привлечения CPU. Если Mailbox ориентирован на прием, то CPU

легко читает его регистры, используя команды разрешения чтения. Mailbox формирует прерывание CPU после каждого успешного приема/передачи сообщения.

2.3.1. Режим стандартного контроллера CAN (SCC)

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

2.3.2. Карта памяти

eCAN-модуль имеет два банка адресов, отражаемых в памяти TMS320x28xx. Первый банк используется для обращения к регистрам управления, регистрам состояния, приемным маскам, временной метке и ограничению времени исследования сообщения. Доступ к регистрам управления и состояния ограничен 32-разрядной шиной. К приемным маскам, регистрам временной метки и регистрам ограничения времени можно обратиться 8-, 16- и 32-разрядными шинами. Каждый из этих двух банков памяти, которые показаны на рис. 12.4, использует 512 байтов адресного пространства. Аналогично выглядит карта памяти eCAN-B, изображенная на рис. 12.5.

Регистры управления и состояния eCAN-A

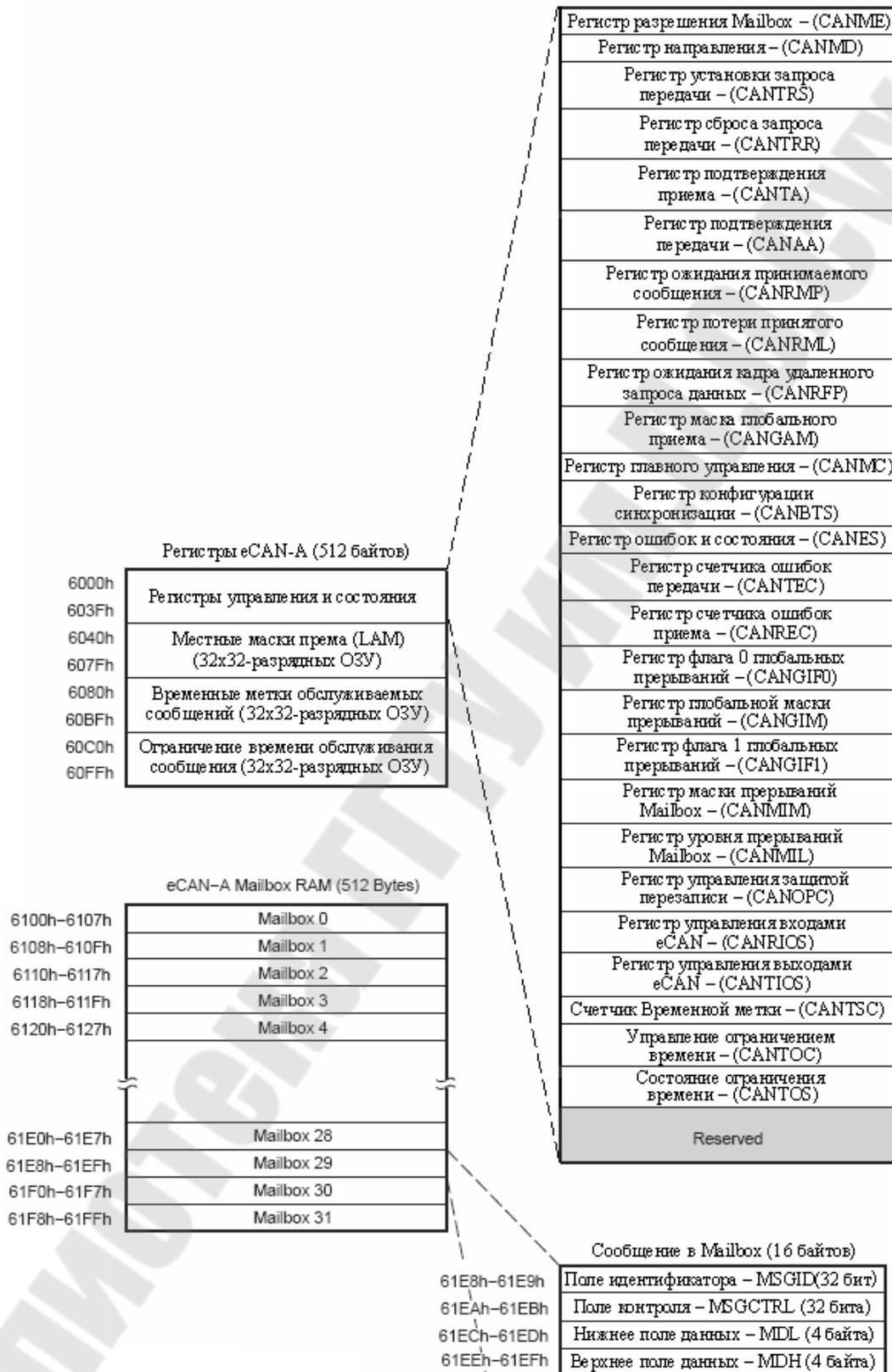


Рис. 12.4. Карта памяти eCAN-A

Регистры управления и состояния eCAN-B

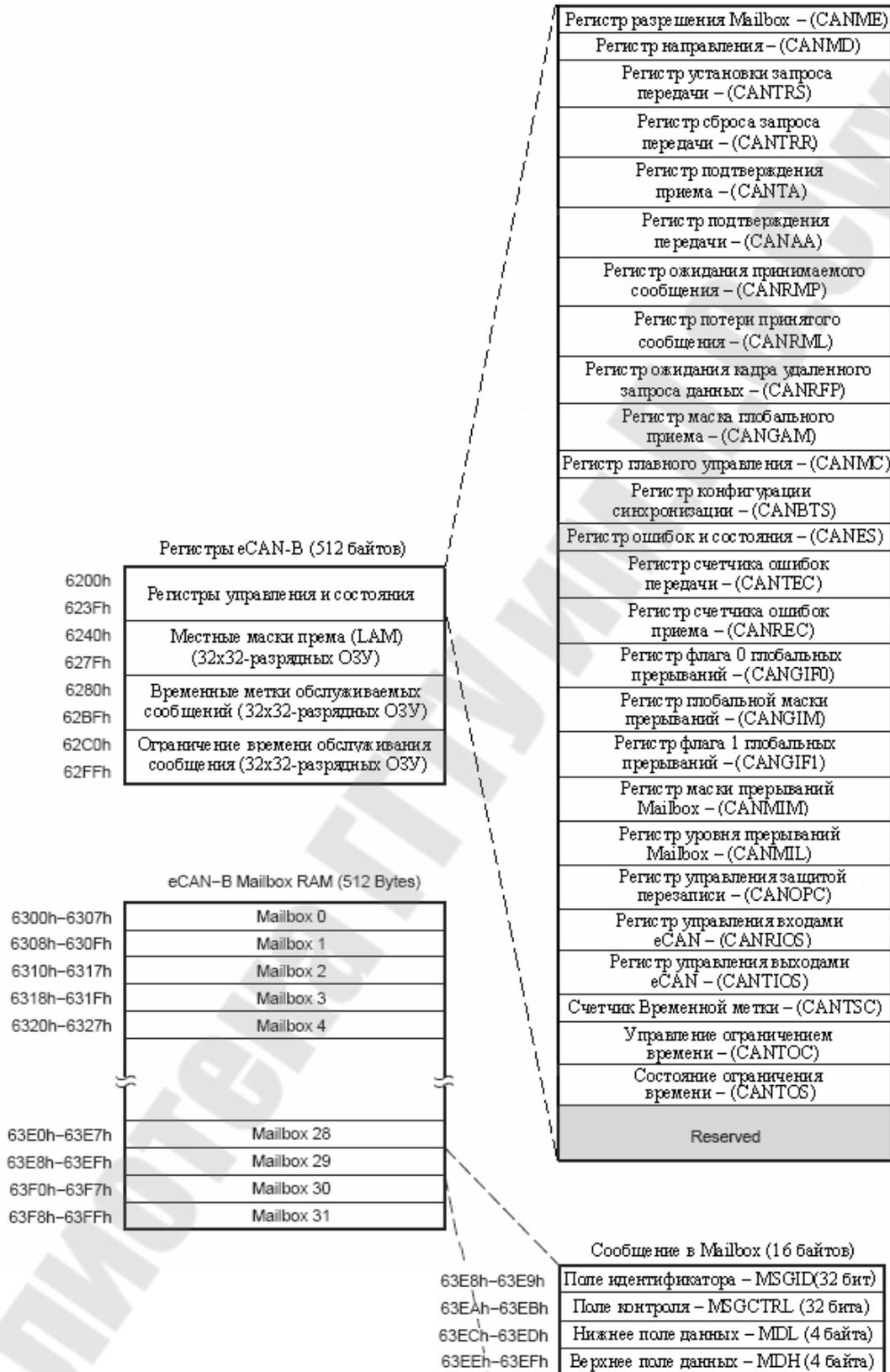


Рис. 12.5. Карта памяти eCAN-B

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

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

2.3.3. Регистры управления и состояния eCAN

Регистры eCAN, перечисленные в табл. 12.1, используются CPU для конфигурации и управления контроллером CAN при работе с сообщениями.

Таблица 12.1

Регистры eCAN

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

Окончание табл. 12.1

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

2.4. Работа с сообщениями

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

Сообщение Mailbox содержит:

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

Соответствующие биты регистров состояния и управления позволяют управлять сообщениями.

2.5. Mailbox

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

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

Каждый Mailbox содержит:

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

Каждый из Mailbox может работать в режиме одного из четырех типов сообщения (табл. 12.2). Прием и передача сообщений необходимы для обмена данными между передающим узлом и многочисленными узлами-приемниками, тогда как сообщения запроса и подтверждения используются для поддержания взаимосвязи на шине.

Таблица 12.2

Конфигурация сообщений

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

Табл. 12.3 показывает расположение байтов Mailbox eCAN-A в ОЗУ.

Таблица 12.3

Расположение байтов Mailbox eCAN-A в ОЗУ

Mailbox	MSGID	MSGCTRL	MDL	MDH
	MIDL-MIDH	MCF-Rsvd	MDL L-MDL H	MDH L-MDH H
0	6100-6101h	6102-6103h	6104-6105h	6106-6107h
1	6108-6109h	610A-610Bh	610C-610Dh	610E-610Fh
2	6110-6111h	6112-6113h	6114-6115h	6116-6117h
3	6118-6119h	611A-611Bh	611C-611Dh	611E-611Fh
4	6120-6121h	6122-6123h	6124-6125h	6126-6127h
5	6128-6129h	612A-612Bh	612C-612Dh	612E-612Fh
6	6130-6131h	6132-6133h	6134-6135h	6136-6137h
7	6138-6139h	613A-613Bh	613C-613Dh	613E-613Fh
8	6140-6141h	6142-6143h	6144-6145h	6146-6147h
9	6148-6149h	614A-614Bh	614C-614Dh	614E-614Fh
10	6150-6151h	6152-6153h	6154-6155h	6156-6157h
11	6158-6159h	615A-615Bh	615C-615Dh	615E-615Fh
12	6160-6161h	6162-6163h	6164-6165h	6166-6167h
13	6168-6169h	616A-616Bh	616C-616Dh	616E-616Fh
14	6170-6171h	6172-6173h	6174-6175h	6176-6177h
15	6178-6179h	617A-617Bh	617C-617Dh	617E-617Fh
16	6180-6181h	6182-6183h	6184-6185h	6186-6187h
17	6188-6189h	618A-618Bh	618C-618Dh	618E-618Fh
18	6190-6191h	6192-6193h	6194-6195h	6196-6197h
19	6198-6199h	619A-619Bh	619C-619Dh	619E-619Fh
20	61A0-61A1h	61A2-61A3h	61A4-61A5h	61A6-61A7h
21	61A8-61A9h	61AA-61ABh	61AC-61ADh	61AE-61AFh
22	61B0-61B1h	61B2-61B3h	61B4-61B5h	61B6-61B7h
23	61B8-61B9h	61BA-61BBh	61BC-61BDh	61BE-61BFh
24	61C0-61C1h	61C2-61C3h	61C4-61C5h	61C6-61C7h
25	61C8-61C9h	61CA-61CBh	61CC-61CDh	61CE-61CFh

26	61D0-61D1h	61D2-61D3h	61D4-61D5h	61D6-61D7h
27	61D8-61D9h	61DA-61DBh	61DC-61DDh	61DE-61DF
28	61E0-61E1h	61E2-61E3h	61E4-61E5h	61E6-61E7h
29	61E8-61E9h	61EA-61EBh	61EC-61EDh	61EE-61EFh
30	61F0-61F1h	61F2-61F3h	61F4-61F5h	61F6-61F7h
31	61F8-61F9h	61FA-61FBh	61FC-61FDh	61FE-61FFh

2.5.1. Передача в Mailbox

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

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

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

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

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

2.5.2. Прием в Mailbox

Идентификатор каждого принятого сообщения сравнивается с идентификатором, содержащимся на каждом Mailbox, используя маску. Когда обнаружено совпадение, полученный идентификатор, биты управления и байты данных записываются на определенный ОЗУ адрес. В то же время формируется бит ожидания приема

сообщения RMP[n] (RMP.31-0), устанавливается и происходит прерывание, если оно разрешено. Если сообщение идентификатора не произошло, то сообщение не сохраняется [5].

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

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

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

2.5.3. Операции CAN-модуля в нормальном режиме

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

Необходимость второго узла отсутствует в режиме самопроверки (STM). В этом режиме передающий узел формирует свой собственный сигнал. Единственное требование – узел должен быть настроен на любую рабочую скорость передачи информации.

Т.о. регистры синхронизации не должны содержать значения, не актуальные для CAN-протокола.

Порядок выполнения лабораторной работы

1. Настройка отправки сообщения

1. Создание нового проекта.

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

1.2. Добавить в проект файлы:

```
C:\tides\c28\dsp281x\v100\DSP281x_headers\source\DSP281x_GlobalVariableDefs.c
C:\tides\c28\dsp281x\v100\DSP281x_common\source\DSP281x_PieCtrl.c
C:\tides\c28\dsp281x\v100\DSP281x_common\source\DSP281x_PieVect.c
C:\tides\c28\dsp281x\v100\DSP281x_common\source\DSP281x_DefaultIsr.c
C:\tides\c28\dsp281x\v100\DSP281x_common\source\DSP281x_CpuTimers.c
C:\tides\c28\dsp281x\v100\DSP281x_headers\cmd\F2812_Headers_nonBIOS.cmd
C:\tides\c28\dsp281x\v100\DSP281x_common\cmd\F2812_EzDSP_RAM_Ink.cmd
C:\ti\c2000\cgtoolslib\rts2800_ml.lib
```

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

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

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

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

2.3. Закрывать Build Options, кликнув ОК.

3. Преобразование файла Lab9.c.

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

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

```
struct ECAN_REGS ECanaShadow;
```

3.3. В подпрограмме «Gpio_select()» настроить периферийные функции CANTxA и CANRxA для работы с CAN-модулем:

```
GpioMuxRegs.GPFMUX.bit.CANTXA_GPIOF6 = 1;  
GpioMuxRegs.GPFMUX.bit.CANRXA_GPIOF7 = 1;
```

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

3.4.1. В регистрах “ECanaRegs.CANTIOC” и “ECanaRegs.CANRIOC” установить биты “TXFUNC” и “RXFUNC”.

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

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

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

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

3.4.6. После установки параметров регистра “ECanaRegs.CANBTC” запретить к нему доступ - очистить бит CCR регистра “ECanaRegs.CANMC”.

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

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

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

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

```
ECanaShadow.CANMD.all = ECanaRegs.CANMD.all;  
ECanaShadow.CANMD.bit.MD5 = 0;  
ECanaRegs.CANMD.all = ECanaShadow.CANMD.all;  
Активировать mailbox#5:  
ECanaShadow.CANME.all = ECanaRegs.CANME.all;  
ECanaShadow.CANME.bit.ME5 = 1;  
ECanaRegs.CANME.all = ECanaShadow.CANME.all;
```

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

4. Присоединение байта данных и отправка.

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

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

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

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

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

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

5. Тестирование программы.

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

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

5.3. Запустить программу: Debug → Run.

2. Настройка приема сообщения

1. Создание нового проекта.

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

1.2. Добавить в проект файлы:

```
C:\ti\c28\dsp281x\v100\DSP281x_headers\source\DSP281x_GlobalVariableDefs.c
C:\ti\c28\dsp281x\v100\DSP281x_headers\source\DSP281x_GlobalVariableDefs.c
C:\ti\c28\dsp281x\v100\DSP281x_headers\cmd\F2812_Headers_nonBIOS.cmd
C:\ti\c28\dsp281x\v100\DSP281x_common\cmd\F2812_EzDSP_RAM_Ink.cmd
C:\ti\c2000\cgtools\lib\rts2800_ml.lib
```

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

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

```
C:\ti\c28\dsp281x\v100\DSP281x_headers\include;..\include
```

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

2.3. Закрыть Build Options, кликнув ОК.

3. Преобразование файла Lab9.c.

3.1. Открыть файл Lab10.c

Удалить те части программы, которые не будут использоваться в данной лабораторной работе: подпрограмму «cpu_timer0_isr()» и массив LED[8].

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

Задать новые инструкции для сторожевого таймера:

```
EALLOW;  
SysCtrlRegs.WDKEY = 0x55;  
SysCtrlRegs.WDKEY = 0xAA;  
EDIS;
```

3.3. Создать новую структуру в основной программе «ECanaShadow»:

```
struct ECAN_REGS ECanaShadow;
```

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

```
GpioMuxRegs.GPFMUX.bit.CANTXA_GPIOF6 = 1;  
GpioMuxRegs.GPFMUX.bit.CANRXA_GPIOF7 = 1;
```

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

3.5.1. В регистрах “ECanaRegs.CANTIOC” и “ECanaRegs.CANRIOC” установить биты “TXFUNC” и “RXFUNC”.

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

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

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

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

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

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

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

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

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

```
ECanaShadow.CANMD.all = ECanaRegs.CANMD.all;  
ECanaShadow.CANMD.bit.MD1 = 1;  
ECanaRegs.CANMD.all = ECanaShadow.CANMD.all;  
Активировать mailbox#5:  
ECanaShadow.CANME.all = ECanaRegs.CANME.all;  
ECanaShadow.CANME.bit.ME1 = 1;  
ECanaRegs.CANME.all = ECanaShadow.CANME.all;
```

4. Организация цикла опроса mailbox#1

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

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

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

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

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

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

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

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

5. Тестирование программы.

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

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

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

3. Инициализация нового узла в сети CAN

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

Листинг передачи сообщения

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

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

void main(void)
{
struct ECAN_REGS ECanaShadow;

InitSystem(); // Инициализация регистров ЦСП

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

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

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

InitCpuTimers();

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

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

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

// Разрешение глобальным прерываниям и более приоритетным событиям
отладки в реальном времени:
EINT; // Разрешение глобального прерывания INTM
ERTM; // Разрешение глобального прерывания в реальном времени
DBGM

InitCan();
/* Запись в поле MSGID */
ECanaMboxes.MBOX5.MSGID.all = 0x????????;
ECanaMboxes.MBOX5.MSGID.bit.IDE = ?; // Установка идентификатора
сообщения

/* Установка Mailbox 5 передающим */
ECanaShadow.CANMD.all = ECanaRegs.CANMD.all;
ECanaShadow.CANMD.bit.MD5 = ?;
ECanaRegs.CANMD.all = ECanaShadow.CANMD.all;
```

```

/* Активация Mailbox 5 */
ECanaShadow.CANME.all = ECanaRegs.CANME.all;
ECanaShadow.CANME.bit.ME5 = ?;
ECanaRegs.CANME.all = ECanaShadow.CANME.all;

/* Установка длины сообщения равной 1 */
ECanaMboxes.MBOX5.MSGCTRL.bit.DLC = ?;
CpuTimer0Regs.TCR.bit.TSS = 0;
while(1)
{
while(CpuTimer0.InterruptCount < 20)
{
// ожидание Timer 0
EALLOW;
SysCtrlRegs.WDKEY = 0xAA; // и передача watchdog #2

EDIS;
}
CpuTimer0.InterruptCount = 0; // обнулить таймер прерываний
ECanaMboxes.MBOX5.MDL.byte.BYTE0 = (GpioDataRegs.GPBDAT.all>>8 )
;

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

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

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

void Gpio_select(void)
{
EALLOW;
GpioMuxRegs.GPAMUX.all = 0x?; // Настройка линий ввода/вывода на
работу в качестве портов
GpioMuxRegs.GPBMUX.all = 0x?;
GpioMuxRegs.GPDMUX.all = 0x?;
GpioMuxRegs.GPFMUX.all = 0x?;
GpioMuxRegs.GPFMUX.bit.CANTXA_GPIOF6 = ?; // Настройка
периферийных функций CANTXA и
GpioMuxRegs.GPFMUX.bit.CANRXA_GPIOF7 = ?; // CANRXA для
работы с CAN-модулем
GpioMuxRegs.GPEMUX.all = 0x?;
GpioMuxRegs.GPGMUX.all = 0x?;
GpioMuxRegs.GPADIR.all = 0x?; // Настройка портов A, D, E, F, G
на ввод
GpioMuxRegs.GPBDIR.all = 0x????; // Настройка линий 15-8 на
ввод, а линий 7-0 на вывод
GpioMuxRegs.GPDDIR.all = 0x?;
GpioMuxRegs.GPEDIR.all = 0x?;
GpioMuxRegs.GPFDIR.all = 0x?;
GpioMuxRegs.GPGDIR.all = 0x?;

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

```

```

void InitSystem(void)
{
    EALLOW;
    SysCtrlRegs.WDCR= 0x00AF;          // Работа сторожевого таймера
    // 0x00E8      запрещение работы сторожевого таймера,
предделитель = 1
    // 0x00AF      Разрешение работы сторожевого таймера, предделитель =
64
    SysCtrlRegs.SCSR = 0;              // Выработка сброса WDT
    SysCtrlRegs.PLLCR.bit.DIV = 10;    // Настройка блока умножения
частоты
    SysCtrlRegs.HISPCP.all = 0x1;      // Задание значения предделителя
высокоскоростного таймера
    SysCtrlRegs.LOSPCP.all = 0x2;      // Задание значения предделителя
низкоскоростного таймера
    // Запрещение работы периферийных устройств кроме eCAN
    SysCtrlRegs.PCLKCR.bit.EVAENCLK=0;
    SysCtrlRegs.PCLKCR.bit.EVBENCLK=0;
    SysCtrlRegs.PCLKCR.bit.SCIAENCLK=0;
    SysCtrlRegs.PCLKCR.bit.SCIBENCLK=0;
    SysCtrlRegs.PCLKCR.bit.MCBSPENCLK=0;
    SysCtrlRegs.PCLKCR.bit.SPIENCLK=0;
    SysCtrlRegs.PCLKCR.bit.ECANENCLK=1;
    SysCtrlRegs.PCLKCR.bit.ADCENCLK=0;
    EDIS;
}
interrupt void cpu_timer0_isr(void)
{
    CpuTimer0.InterruptCount++;        // Сообщение watchdog-таймеру о
каждом прерывании Timer 0
    EALLOW;
    SysCtrlRegs.WDKEY = 0x55;          // Сообщение таймеру watchdog #1
    EDIS;
    // Это прерывание произошло от прерываний группы 1
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}
void InitCan(void)
{
    asm(" EALLOW");
    /* Настроить линии TX и RX на передачу используя регистры eCAN*/
    ECanaRegs.CANTIOC.bit.TXFUNC = ?;
    ECanaRegs.CANRIOC.bit.RXFUNC = ?;
    /* Настроить eCAN на режим HECC - (доступ к чтению mailboxes 16 - 31)
*/
    // В режиме HECC разрешена особенность временного ограничения
    ECanaRegs.CANMC.bit.SCB = ?;
    /* Настроить параметры тактовой синхронизации */
    ECanaRegs.CANMC.bit.CCR = ? ;      // Установка CCR для
получения доступа к регистрам времени
    while(ECanaRegs.CANES.bit.CCE != 1 ) {} // Когда CCE (флаг для
передачи запроса инициализации CAN) установлен..
    ECanaRegs.CANBTC.bit.BRPREG = ??;  // Установка параметров
BRP, TSEG1 и TSEG2
    ECanaRegs.CANBTC.bit.TSEG2REG = ?; // таким образом, чтобы
скорость передачи
    ECanaRegs.CANBTC.bit.TSEG1REG = ??; // была 100кбит/с

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

```

```
//=====
```

Листинг приема сообщения

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

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

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

void main(void)
{
    struct ECAN_REGS ECanaShadow;

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

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

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

    /* Активация Mailbox 1 */
    ECanaShadow.CANME.all = ECanaRegs.CANME.all;
    ECanaShadow.CANME.bit.ME1 = ?;
    ECanaRegs.CANME.all = ECanaShadow.CANME.all;
    while(1)
    {
        do
        {
            ECanaShadow.CANRMP.all = ECanaRegs.CANRMP.all;
EALLOW;
            SysCtrlRegs.WDKEY = 0x55; // Сообщение таймеру watchdog #1
            SysCtrlRegs.WDKEY = 0xAA; // Сообщение таймеру watchdog #2
EDIS;
        }
        while(ECanaShadow.CANRMP.bit.RMP1 != 1 ); // Когда RMP1
установлен..

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

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

```

    GpioMuxRegs.GPBMUX.all = 0x?;
    GpioMuxRegs.GPDMUX.all = 0x?;
    GpioMuxRegs.GPFMUX.all = 0x?;
    GpioMuxRegs.GPEMUX.all = 0x?;
    GpioMuxRegs.GPGMUX.all = 0x?;
    GpioMuxRegs.GPFMUX.bit.CANTXA_GPIOF6 = ?; // Настройка
периферийных функций CANTxA и CANRxA
    GpioMuxRegs.GPFMUX.bit.CANRXA_GPIOF7 = ?;

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

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

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

    SysCtrlRegs.SCSR = 0; // Выработка сброса WDT

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

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

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

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

    /* Настроить eCAN на режим HECC - (доступ к чтению mailboxes 16 - 31)
*/

```

```

// В режиме HECC разрешена особенность временного ограничения
ECanaRegs.CANMC.bit.SCB = ?;

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

ECanaRegs.CANMC.bit.CCR = ? ; // Сброс бита CCR
регистра CANMC для запрета доступа к регистру CANBTC
while(ECanaRegs.CANES.bit.CCE == !0 ) {} // Когда CCE сброшен..

/* Отключить все Mailboxes */

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

```

Содержание отчета

Отчет должен содержать цель работы, краткие теоретические сведения по теме работы, рисунки, тексты исследуемых программ передачи и приема данных через CAN-интерфейс, результаты их выполнения, выводы.

Контрольные вопросы

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

Литература

1. Соловьев, В. В. Основы языка проектирования цифровой аппаратуры Verilog : учеб. пособие для студентов] / В. В. Соловьев. – М. : Горячая линия-Телеком, 2016. – 205 с.
2. Бибило, П. Н. Основы языка VHDL : учеб. пособие / П. Н. Бибило. – 5-е изд. – М. : Книжный дом «ЛИБРОКОМ», 2012. – 328 с.
3. Айфичер, Э. Цифровая обработка сигналов: практический подход / Э. Айфичер, Б. Джервис. – 2-е изд. – 2004. – 992 с.
4. Поляков, А. К. Языки VHDL и VERILOG в проектировании цифровой аппаратуры / А. К. Поляков. – М. : СОЛОН-Пресс, 2003. – 320 с.
5. Лайонс, Р. Цифровая обработка сигналов / Р. Лайонс. – 2-е изд. – М., 2006. – 652 с.
6. 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.
7. Рабинер, Л. Р. Теория и применение цифровой обработки сигналов ; пер. с англ. / Л. Р. Рабинер, Б. Голд ; под ред. Ю. Н. Александрова. – М. : Мир, 1978. – 637 с.
8. Шевкопляс, Б. В. Микропроцессорные структуры. Инженерные решения : Справочник / Б. В. Шевкопляс – 2-е изд., перераб. и доп. – М.: Радио и связь, 1990. – 512 с.
9. Сергиенко, А. Б. Цифровая обработка сигналов: учеб. пособие для вузов / А. Б. Сергиенко. – СПб. : Питер, 2007. – 603 с.
10. Яне, Б. Цифровая обработка изображений / Б. Яне. – М. : Техносфера, 2007. – 584 с.
11. Гутников, В. С. Фильтрация измерительных сигналов / В. С. Гутников. – Л. : Энергоатомиздат, 1990. – 190 с.
12. Шостак, А. С. Прием и обработка сигналов [Электронный ресурс]: курс лекций / А. С. Шостак. – Томск : Томский государственный университет систем управления и радиоэлектроники, 2012. – Ч.2. – 87 с. – Режим доступа: <http://biblioclub.ru/index.php?page=book&id=208720>.
13. Щетинин, Ю. И. Анализ и обработка сигналов в среде MATLAB : учеб. пособие / Ю.И. Щетинин. – Новосибирск: НГТУ, 2011. – 115 с. – Режим доступа: <http://biblioclub.ru/index.php?page=book&id=229142>.

14. Оппенгейм, А. Цифровая обработка сигналов / А.Оппенгейм, Р. Шафер: пер-к С.Ф. Боев. – 3-е изд., испр. – Москва: Техносфера, 2012. – 1048 с. – (Мир радиоэлектроники). – URL: <http://biblioclub.ru/index.php?page=book&id=233730> (09.11.2017).
15. Meyer-Baese, U. Digital Signal Processing with Field Programmable Arrays (3th ed.). – Springer, 2016. – Florida State University, College of Engineering. – 788 p.
16. Стешенко, В. Б. ПЛИС фирмы Altera: проектирование устройств обработки сигналов. – М.: ДОДЭКА, 2000. – 128 с.
17. Бродин, В. Б. Системы на микроконтроллерах и БИС программируемой логики / В. Б. Бродин, А. В. Калинин. – М. : Изд-во ЭКОМ, 2002. – 400 с.
18. Потехин, Д. С. Разработка систем цифровой обработки сигналов на базе ПЛИС / Д. С. Потехин, И. Е. Тарасов. – М. : Горячая линия – Телеком, 2007. – 248 с.
19. Солонина, А. И. Алгоритмы и процессоры цифровой обработки сигналов / А. И. Солонина, Д. А. Улахович, Л. А. Яковлев. – СПб: БХВ-Петербург, 2001. – 464 с.
20. Вальпа, О. – Разработка устройств на основе цифровых сигнальных процессоров фирмы Analog Devices с использованием Visual DSP++. – Горячая линия-Телеком, 2007. – 446 с.
21. Введение в цифровую фильтрацию / Под ред. Р. Богнера и А. Константинодиса. – пер. с англ., под. ред Л. И. Филлипова – М. : Мир, 1976. – 216 с.
22. Марпл, С. Л. Цифровой спектральный анализ и его приложения / С. Л. Марпл. – М. : Мир, 1990. – 236 с.
23. Сато Ю. Обработка сигналов. Первое знакомство. – М. : ДОДЭКА, 2003. – 174 с.
24. ADSP-2181 DSP Microcomputer. Data Sheet. Rev.B., Analog Devices Inc.
25. ADSP-2100 Family User's Manual. Edition 3, Analog Devices Inc.
26. TMS320F28x Family User's Manual. Edition 1, Texas Instruments Inc. – в 9-ти ч.
27. Шпак, Ю. А. Программирование на языке С для AVR и PIC микроконтроллеров / Ю. А. Шпак. – К.: МК-Пресс, СПб.: КОРОНА-ВЕК, 2011.
28. Применение цифровой обработки сигналов / Под ред. Э. Оппенгейма. – М. : Мир. – 1980. – 552 с.

29. Марков. С. Цифровые сигнальные процессоры. Книга 1. – М. : Микроарт. – 1996.

30. Цифровые сигнальные процессоры фирмы Zilog и их применение. CHIPNEWS. – 1997. – № 2(11).

31. Блаттер, К. Вейвлет-анализ. Основы теории / К. Блаттер. – М. – Техносфера. – 2006. – 279 с.

32. Цифровые процессоры обработки сигналов : справочник / А. Г. Остапенко, С. И. Лавлинский, А. В. Сушков и др. ; под ред. А. Г. Остапенко. – М. : Радио и связь, 1994. – 264 с.

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

34. Liptak, В. G. Process Control and Optimization. Instrument Engineers' Handbook 2 (4th ed.). – CRC Press – 2006.

35. Verbauwhede, I., Schaumont, P., Piguet, C., Kienhuis, B. Architectures and Design techniques for energy efficient embedded DSP and multimedia processing (PDF). – rijndael.ece.vt.edu. – Retrieved 2014-06-11.

36. Bogdanowicz, A. IEEE Milestones Honor Three. The Institute. IEEE. – Retrieved 2012-03-02.

Содержание

Лабораторная работа № 1	
Реализация комбинационных цифровых устройств на основе отладочной платы Spartan-3E Starter Kit	4
Лабораторная работа № 2	
Реализация последовательностных цифровых устройств на основе отладочной платы Spartan-3E Starter Kit	39
Лабораторная работа № 3	
Исследование среды симуляции ModelSim.....	60
Лабораторная работа № 4	
Реализация конечных автоматов на основе отладочной платы Spartan-3E Starter Kit	70
Лабораторная работа № 5	
Реализация аналого-цифрового преобразования на основе отладочной платы Spartan-3E Starter Kit	79
Лабораторная работа № 6	
Реализация цифроаналогового преобразователя на основе отладочной платы Spartan-3E Starter Kit	89
Лабораторная работа № 7	
Изучение отладочного модуля для цифрового сигнального процессора семейства C28x.....	96
Лабораторная работа № 8	
Исследование систем управления и ввода/вывода цифрового сигнального процессора семейства C28x	105
Лабораторная работа № 9	
Формирование функций времени на основе сигнального процессора семейства C28x. Модуль расширения прерываний.....	123
Лабораторная работа № 10	
Формирование сигналов широтно-импульсной модуляции на основе сигнального процессора семейства C28x.....	143
Лабораторная работа № 11	
Ввод и масштабирование аналоговых сигналов в процессорах семейства C28x	162
Лабораторная работа № 12	
Исследование встроенного CAN-интерфейса DSP TMS320F2812	174
Литература.....	200

АППАРАТУРА ЦИФРОВОЙ ОБРАБОТКИ СИГНАЛОВ

**Практикум
по выполнению лабораторных работ
для студентов специальности 1-36 04 02
«Промышленная электроника»
дневной и заочной форм обучения**

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

Подписано к размещению в электронную библиотеку
ГГТУ им. П. О. Сухого в качестве электронного
учебно-методического документа 31.10.22.

Пер. № 9Е.
<http://www.gstu.by>