Разработка проекта микроконтроллера 8051s на основе IP-ядер корпорации Microsemi.

Часть 5. Создание собственных устройств для шины APB. Увеличение устойчивости проекта на ПЛИС Microsemi к однократным сбоям

Дмитрий ИОФФЕ
support@actel.ru
Андрей МАКСИМОВ
maksimov@actel.ru

Это пятая статья из цикла, посвященного применению микропроцессорного IP-ядра 8051s для ПЛИС фирмы Microsemi. В первой части статьи [1] было рассмотрено построение аппаратной части системы на основе 8051s с использованием IP-ядер, поставляемых в составе САПР Libero. Во второй части [2] приведено описание ядра 8051s для программиста. В третьей части [3] мы написали и отладили простейшую программу для микроконтроллера. В четвертой части [4] была рассмотрена работа с прерываниями и подключение сторожевого таймера.
Сегодня мы будем учиться создавать собственные устройства для шины APB. Кроме того, мы познакомимся с методами повышения радиационной стойкости проектов на ПЛИС корпорации Microsemi за счет использования специальных атрибутов для синтезатора Synplify.

Введение

Каким бы широким ни был выбор готовых IP-ядер, все равно рано или поздно потребуется что-то такое, чего до нас никто еще не делал. Собственно, это и есть работа инженера-разработчика. «Ученые изучают то, что уже есть; инженеры создают то, чего никогда не было» (А. Эйнштейн) [5]. Попробуем на простом примере разобраться, как можно создать свое устройство для шины APB. Для начала попытаемся заменить одно из периферийных устройств в нашем учебном проекте. Первый кандидат на замену — универсальный модуль ввода/вывода CoreGPIO. Для работы с двумя светодиодами и одним микропереключателем сложный настраиваемый 32-разрядный блок явно избыточен.

Порядок работы предлагается следующий:

- ознакомление с шиной APB;
- создание на языке описания аппаратуры устройства, которое будет опрашивать микропереключатель и управлять двумя светодиодами;
- включение этого устройства в проект вместо модуля GPIO;
- внесение изменений в программу для микроконтроллера;
- генерация проекта;
- отладка проекта в среде Modelsim;
- прошивка проекта в ПЛИС и констатация факта, что снаружи все осталось как было.

Шина APB

Шина APB — это часть семейства шин AMBA 3 фирмы ARM. Она представляет собой универсальный интерфейс для подключения относительно низкоскоростных и мало потребляющих устройств. Описание ее работы приведено в [6, 7]. APB подключается к высокопроизводительной части шины AMBA — AHB — через специальный мост или к микроконтроллерному ядру, такому как 8051s или CoreABC.

Передача данных по шине APB (табл. 1) состоит из двух фаз: фазы адреса и фазы данных. Фаза адреса всегда занимает один такт шины, а фаза данных может содержать состояния ожидания и длиться несколько тактов.

Простейший цикл записи без состояний ожидания происходит следующим образом (рис. 1а). В фазе адреса по нарастающему фронту тактового сигнала PCLK ведущее устройство устанавливает следующие сигналы:

- адрес ведомого устройства PADDR;
- сигнал записи PWRITE (активный уровень — высокий);
- сигнал выбора устройства PSEL (активный уровень — высокий);
- данные для записи PWDATA.

Состояния этих сигналов сохраняются и в фазе данных. По второму фронту тактового сигнала устанавливается сигнал PENABLE (активный уровень — высокий). Это означает начало фазы записи данных. До следующего такта ведомое устройство должно установить сигнал

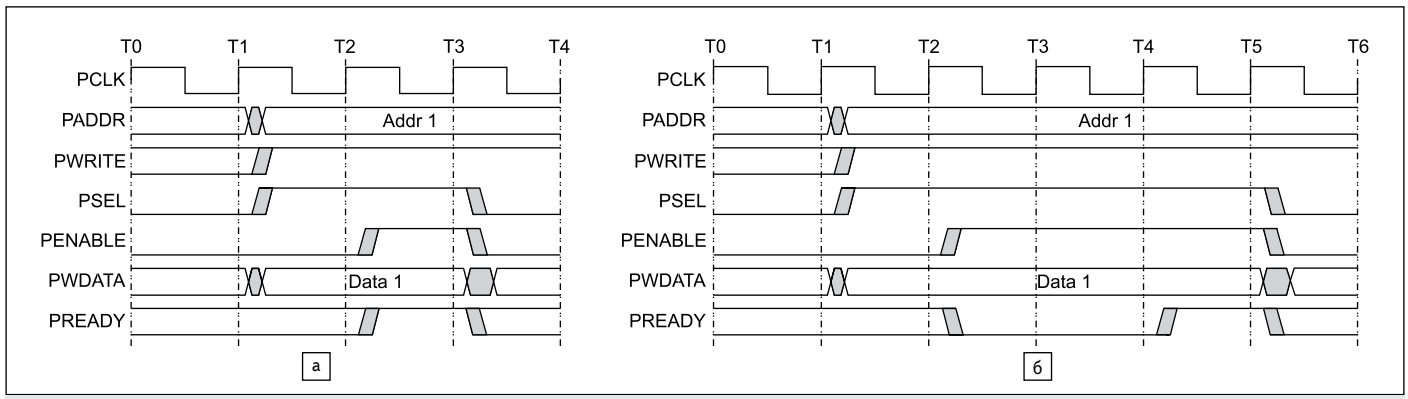

Рис. 1. Временная диаграмма цикла записи данных: а) без состояний ожидания; а) с состояниями ожидания

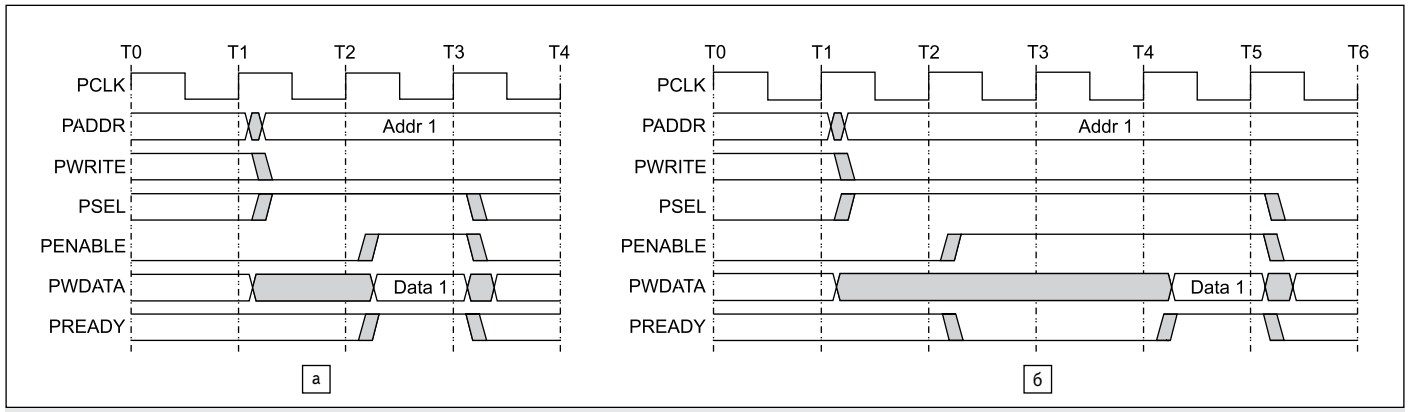

Рис. 2. Временная диаграмма цикла чтения данных: а) без состояний ожидания; б) с состояниями ожидания

PREADY (активный уровень — высокий) и принять передаваемые данные. Получив сигнал PREADY, ведущее устройство по третьему такту снимает сигнал PENABLE. Сигнал выбора PSEL при этом также снимается, даже если следующее обращение будет происходить к тому же самому устройству. На этом цикл записи заканчивается.

Периферийное устройство может задержать окончание цикла записи (рис. 1б). Для этого оно должно при активном сигнале PENABLE снять сигнал PREADY и не устанавливать его до тех пор, пока не закончит прием данных. В таком случае цикл записи закончится по первому фронту тактового сигнала, на котором будет обнаружен активный уровень PREADY.

Временные диаграммы циклов чтения без состояний ожидания (рис. 2а) и с ними (рис. 2б) выглядят аналогично. Первое отличие состоит в том, что в фазе адреса сигнал PWRITE имеет низкий уровень. В этом случае при активном уровне сигнала PENABLE ведомое устройство должно выставить данные на линиях чтения PRDATA, сопровождая их активным уровнем сигнала PREADY. Если периферийное устройство хочет задержать цикл чтения, оно должно снять сигнал PREADY при высоком уровне сигнала PENABLE. Тогда ведущее устройство перейдет в состояние ожидания до тех пор, пока не получит активного уровня сигнала PREADY.

Таблица 1. Сигналы шины APB

Имя сигнала	Описание
PCLK	Тактовая частота. Все действия на шине APB происходят по нарастающему фронту PCLK.
PRESETn	Сброс. Активный уровень — низкий. Обычно этот сигнал подключается непосредственно к сигналу сброса системной шины.
PADDR	Адрес. Может иметь разрядность до 32 бит. Поступает от ведущего устройства, которым может быть микроконтроллерное IP-ядро или мост к шине AHB.
PSELx	Выбор. Ведущее устройство вырабатывает этот сигнал отдельно для каждого периферийного устройства. Он означает, что ведомое устройство выбрано и запрашивается передача данных.
PENABLE	Разрешение. Этот сигнал активен во втором и последующих циклах передачи данных на шине APB.
PWRITE	Направление передачи данных. Высокий уровень означает запись данных из шины в периферийное устройство, низкий — чтение данных из периферийного устройства.
PWDATA	Данные для записи в периферийное устройство. Разрядность — до 32 бит.
PREADY	Сигнал готовности периферийного устройства. Ведомое устройство может использовать этот сигнал для увеличения времени цикла передачи на шине APB.
PRDATA	Данные, читаемые из периферийного устройства. Разрядность — до 32 бит.
PSLVERR	Сигнал ошибки передачи. Периферийные устройства APB, как существующие, так и вновь разрабатываемые, не требуют поддержки этого сигнала. Если периферийное устройство не формирует этот сигнал, то на соответствующий вход ведущего устройства надо подать низкий уровень.

Создание собственного модуля ввода/вывода

После всего сказанного мы можем приступить к написанию кода нашего модуля ввода/вывода на языке VHDL.

Замечание: мы не можем обеспечить нашему модулю участие в механизме автоматического назначения адресов периферийных устройств, так как корпорация Microsemi не документирует работу данного механизма. А потому просто посмотрим, какое смещение относительно базового адреса шины APB выделено в нашем проекте модулю GPIO. Для этого щелкнем правой кнопкой мыши по изображению IP-ядра CoreAPB3 на холсте нашего проекта и в выпавшем контекстном меню выберем пункт **Modify Memory Map**. Появится окно с таким же именем (рис. 3).

Мы видим, что модулю CoreGPIO присвоено смещение 0x00000100. Воспользуемся этим смещением для нашего нового модуля. Если по ходу работы с проектом распределение адресов автоматически изменится, мы всегда сможем откорректировать его в том же окне. Как это сделать, описано в первой части статьи [1].

Назовем наш модуль myio. Его код на языке VHDL прост и очевиден (листинг 1), более или менее необходимые комментарии приводятся прямо в его тексте.

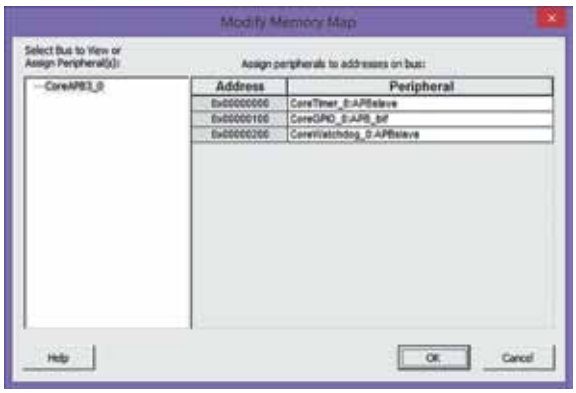

Рис. 3. Окно Modify Memory Map

```
library IEEE;
use IEEE.std_logic_1164.all;

entity myio is
   port (
         --сигналы шины APB:
         PRESETN: in std_logic;
         PCLK: in std_logic;
         PENABLE: in std_logic;
         PSEL: in std_logic;
         PREADY: out std_logic;
         PADDR: in std_logic_vector(31 downto 0);
         PWRITE: in std_logic;
         PWDATA: in std_logic_vector(31 downto 0);
         PRDATA: out std_logic_vector(31 downto 0);
         --линии ввода-вывода:
         myio_in: in std_logic;
         myio_out: out std_logic_vector(1 downto 0)
   );
end myio;

architecture myio_0 of myio is
   constant myaddr: std_logic_vector := --адрес на шине
      ('0','0','0','0','0','0','0','0',
      '0','0','0','0','0','0','0','0',
      '0','0','0','0','0','0','0','1',
      '0','0','0','0','0','0','0','0');

begin
--выходные данные — управление светодиодами:
myio_out_l:
   process(PRESETN,PCLK,PSEL,PADDR,PWRITE,PENABLE,PWDATA)
   begin
      if PRESETN = '0' then myio_out(1 downto 0) <= (others => '0');
      elsif rising_edge(PCLK) then
         if PSEL = '1' and PADDR = myaddr and PWRITE = '1' and PENABLE = '1'
         then myio_out <= PWDATA(1 downto 0);
         end if;
      end if;
   end process myio_out_l;

--входные данные — чтение микропереключателя:
 PRDATA(0) <= myio_in;
 PRDATA(31 downto 1) <= (others => '0');

--сигнал готовности — модуль всегда готов, обмен без тактов ожидания:
   PREADY <= '1';

end myio_0;
```

Листинг 1. Код самодельного модуля ввода/вывода

Включение модуля в проект

После создания файла с кодом нашего модуля его имя отобразится в поле **Design Explorer** среды Libero (рис. 4).

Для начала проверим синтаксис нашего кода. Щелкнем правой кнопкой мыши по имени myio и выберем из выпавшего меню пункт **Check HDL file**. Если в поле протокола в нижней части окна появятся сообщения об ошибках — находим их и устраняем. Если нам скажут, что все в порядке, то еще раз щелкнем правой кнопкой мыши по myio в поле **Design Explorer** и выберем из выпавшего меню пункт **Create Symbol**. При этом могут появиться новые сообщения об ошибках, которых не было после первой проверки.

Рис. 4. Имя вновь созданного модуля myio в поле Design Explorer

Теперь удалим с холста модуль CoreGPIO, еще раз щелкнем правой кнопкой мыши по имени myio и выберем из выпавшего меню пункт **Instantiate in System8051s**. На холсте появится символ модуля myio.

Соединим его с остальным проектом (порядок действий произвольный, их последовательность, показанная ниже, дана для примера).

Для начала сделаем наиболее очевидные соединения — подключим выводы myio к отдельным сигналам и линиям ввода/вывода (табл. 2).

Таблица 2. Первая часть подключений

Вывод модуля myio	Сигнал в проекте
myio_in	Вывод проекта GPIO_IN. (После удаления CoreGPIO он остался в одиночестве на серых полях холста.)
myio_out	Выводы проекта GPIO_OUT. (Их также можно найти на полях холста.)
PCLK	PCLK
PRESETN	PRESETn

Метод подключения обычный: щелкаем левой кнопкой мыши по имени входа на символе, затем, одновременно нажимая на клавишу **Ctrl**, — по имени вывода проекта, выбираем из выпавшего меню пункт **Connect**.

Теперь найдем на изображении IP-ядра CoreAPB3 пустой слот S1, оставшийся после удаления модуля CoreGPIO. Щелкнем по нему правой кнопкой мыши и выберем из выпавшего меню **пункт Expose Bus Interface Pins…** Появится окно **Pins to Expose** (рис. 5). Установим флажки рядом со всеми именами сигналов и нажмем **OK**. На линии CoreAPB3 рядом со слотом S1 появятся порты для подключения сигналов. К сожалению, они могут наложиться на другие элементы схемы и чтение их имен окажется серьезной творческой задачей.

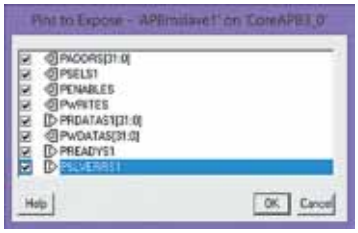

Рис. 5. Окно Pins to Expose

Выполним подключения в соответствии с данными, представленными в таблице 3.

Таблица 3. Вторая часть подключений

Сигналы слота S1	Выводы модуля myio или другое назначение
PADDRS[31..0]	PADDR[31..0]
PWDATAS[31..0]	PWDATA[31..0]
PRDATAS[31..0]	PRDATA[31..0]
PSELS1	PSEL
PENABLES	PENABLE
PWRITES	PWRITE
PREADYS1	PREADY
PSLVERRS1	Tie Low

Внесение изменений в программу для микроконтроллера

В тексте программы для микроконтроллера у нас есть ссылка на определение базового адреса IP-ядра CoreGPIO — COREGPIO_0:

```
#define GPIO_addr APB_base + COREGPIO_0
```

Пока этот блок был в составе нашего проекта, среда Libero автоматически вставляла его в текст файла *System8051s_hw_platform.h*. Теперь мы лишились этого сервиса, и константу COREGPIO_0 придется прописать в программе вручную:

```
#define COREGPIO_0 0x00000100U
```

Кроме того, все обращения к модулю myio производятся по одному адресу, базовому. Отразим этот факт в программе:

```
__xdata at GPIO_addr + 0x00 unsigned char dataout;
__xdata at GPIO_addr + 0x00 unsigned char datain;
```

Теперь наша программа будет выглядеть так (листинг 2):

```
#include "..\System8051s_hw_platform.h"

#define APB_base 0xF000U //начало адресного пространства APB

/*Работа с GPIO:*/
#define COREGPIO_0 0x00000100U
#define GPIO_addr APB_base + COREGPIO_0 //адрес блока GPIO
__xdata at GPIO_addr + 0x00 unsigned char dataout;
__xdata at GPIO_addr + 0x00 unsigned char datain;
#define jumper 0x01 //вход опроса переключателя
#define LED_0 0x01 //нулевой светодиод — следит за джампером
#define LED_1 0x02 //первый светодиод — моргает по прерываниям
unsigned char inreg;
unsigned char outreg;
unsigned char temp;

/*Работа с таймером*/
#define TimerAddr APB_base + CORETIMER_0 //адрес таймера
__xdata at TimerAddr unsigned char TimerLoad;
__xdata at TimerAddr + 0x08 unsigned char TimerControl;
__xdata at TimerAddr + 0x0C unsigned char TimerPrescale;
__xdata at TimerAddr + 0x10 unsigned char TimerIntClr;
__xdata at TimerAddr + 0x14 unsigned char TimerRIS;

/*Работа со сторожевым таймером*/
#define WatchdogAddr APB_base + COREWATCHDOG_0 //адрес таймера
__xdata at WatchdogAddr unsigned char WatchdogLoad;
__xdata at WatchdogAddr + 0x08 unsigned char WatchdogControl;
__xdata at WatchdogAddr + 0x0C unsigned char WatchdogRefresh;

/*X-регистры для записи в старшие 24 разряда шины APB*/
__data at 0x9A unsigned char XWB1;
__data at 0x9B unsigned char XWB2;
__data at 0x9C unsigned char XWB3;

void main ()
{
   outreg = 0x00;
//Инициализация таймера:
   XWB3 = 0;
/* XWB2 = 0x2D;
   XWB1 = 0xC6;
   TimerLoad = 0xC0; //записали 32-разрядное число*/
   XWB2 = 0x00;
   XWB1 = 0x00;
   TimerLoad = 0xC0; //записали 32-разрядное число
   TimerPrescale = 0;
   TimerControl = 0x03;
//Инициализация сторожевого таймера:
   XWB3 = 0;
   XWB2 = 0x00;
   XWB1 = 0x00;
   WatchdogLoad = 0xFF;
   WatchdogControl = 0x10;
//Основной цикл программы:
   for (;;) {
   //Опрос флага прерывания:
      if (TimerRIS == 1)
         {temp = outreg & LED_1;
         if (temp == 0) outreg = outreg | LED_1;
         else outreg = outreg & ~LED_1;
         dataout = outreg;
         WatchdogRefresh = 0x00;
         TimerIntClr = 0; /*сброс прерывания*/};
   //Отслеживание джампера:
       inreg = datain;
      if ((inreg & jumper) == 0)
         outreg = outreg & ~LED_0;
      else
         outreg = outreg | LED_0;
      dataout = outreg;
   }
}
```

Листинг 2. Исправленная программа для микроконтроллера

Далее компилируем программу и получаем из файла IHX код модуля ПЗУ на VHDL, как описано в [3].

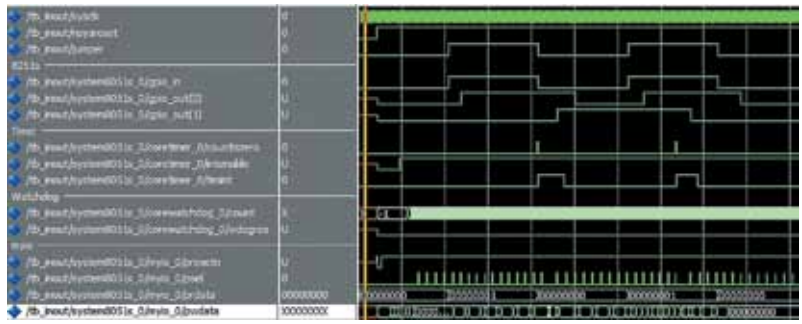

Рис. 6. Результат моделирования проекта

Генерация проекта

Проверяем проект: выбираем пункт меню **SmartDesign → Check Design Rules** среды Libero. Если найдены ошибки, то исправляем их, а затем выбираем пункт меню **SmartDesign → Generate Design**. Дождемся сообщения об успешной генерации проекта.

Теперь можно отлаживать проект.

Отладка проекта в среде Modelsim

Сеанс моделирования происходит так же, как и раньше [3]. И полученные при моделировании временные диаграммы выглядят так же, как и прежде (рис. 6).

Выход GPIO_OUT_0 повторяет с некоторой задержкой сигнал от микропереключателя, а выход GPIO_OUT_1 переключается по сигналу таймера, также с некоторой задержкой. При этом сторожевой таймер обновляется своевременно и поэтому не мешает работать.

Теперь компилируем проект и загружаем его в ПЛИС. Все работает, как работало в предыдущей части статьи, то есть замена «фирменного» IP-ядра на модуль собственной разработки прошла успешно.

Шина APB устроена достаточно просто, и создание подключаемых к ней периферийных устройств доступно даже начинающим разработчикам. Это обеспечивает, в частности, большую гибкость при разработке систем на основе микроконтроллерного IP-ядра 8051s.

Увеличение устойчивости проекта на ПЛИС Microsemi к однократным сбоям

Широко известны радиационно-стойкие ПЛИС, производство которых перешло к корпорации Microsemi с приобретением компании Actel. Это семейства микросхем RTAX-S/SL, RTAX-DSP, RTSX-SU и RT ProASIC3. При их изготовлении используются специальные технологические и топологические методы, позволяющие добиться высокой устойчивости микросхем к различным радиационным эффектам: старению и деградации от суммарной дозы полученной радиации, а также сбоев и защелкиваний при попадании тяжелых высокоэнергетических частиц.

Непредсказуемые, а потому наиболее опасные проблемы вызваны так называемыми случайными воздействиями (англ. Single Event Effects, SEE), которые происходят, когда в кристалл интегральной схемы попадают высокоэнергетические частицы (космические лучи, протоны, электроны, альфа-частицы, термические нейтроны и т. д.) [8]. Проходя сквозь объем полупроводника, они оставляют за собой трек из свободных носителей заряда. Это приводит к генерации электронно-дырочных пар в подзатворном окисле обычных КМОП-схем. Возникший при этом импульс тока может переводить соседние логические элементы в противоположное состояние. Такого рода эффекты называются однократными сбоями (Single-event Upset, SEU), и возникают они в триггерах и ячейках статической памяти.

При помощи специальных приемов защиты проекта в ПЛИС на уровне резервирования логических элементов можно намного уменьшить вероятность проявления негативных радиационных эффектов на конечное устройство. Описанные ниже возможности поддерживаются синтезатором Synplify, начиная с версии 5.31.

Разработчику предлагается три техники реализации логики и последовательных элементов для проектов повышенной надежности [9]. Они называются C-C, TMR и TMR_CC.

В технике C-C вместо триггеров и защелок для реализации элементов памяти применяются комбинационные ячейки с обратными связями. Например, вместо D-триггера DF1 используется примитив DFP1, состоящий из двух комбинационных модулей.

TMR (triple-module-redundancy) — троирование. Это метод реализации триггеров. Каждый триггер создается в трех экземплярах, соединенных по выходу мажоритарной схемой «два из трех». То есть если в результате сбоя у них на выходах сигналы оказываются разными, то результирующим пра-

вильным сигналом считается тот, который присутствует на двух выходах.

TMR_CC — это вариант аппаратного троирования, при котором каждый из трех триггеров состоит из комбинационных ячеек с обратной связью.

Отметим, что в однократно программируемых ПЛИС семейств RTAX-S и RTSX корпорации Microsemi уже имеется аппаратное троирование триггеров, поэтому троирование на пользовательском уровне при помощи синтезатора не требуется.

Управление синтезатором

Для задания техники реализации высоконадежных регистров в ПЛИС корпорации Microsemi в синтезаторе Synplify предусмотрен атрибут syn_radhardlevel. Его можно применить к модулю, архитектуре или отдельному триггеру (подразумеваемому, «inferred»-триггеру в VHDL). При необходимости разработчик может применить его глобально, на верхнем уровне модуля или архитектуры, а затем избирательно отменять его действие в разных частях проекта.

Атрибут syn_radhardlevel может принимать следующие значения:

- none — использовать стандартный синтез;
- cc — использовать реализацию C-C;
- tmr — использовать реализацию TMR;
- tmr_cc — использовать реализацию TMR_CC.

Для модулей и архитектур можно устанавливать атрибут syn_radhardlevel как через панель атрибутов в окне SCOPE пакета Synplify, так и в исходном коде. Однако для отдельных регистров его можно задать только в исходном коде.

Отметим, что атрибут syn_radhardlevel применяется только на том уровне проекта, где он задан, и не влияет на регистры нижних уровней.

Работа со сбоезащищенным проектом строится следующим образом [10].

1. Добавляем в наш проект файлы макросов для той техники реализации защищенных блоков, которую мы будем использовать в нашем проекте (табл. 4). Эти файлы расположены в каталоге установки Synplify в подкаталоге /lib/actel. Для устройств ProASIC3/3E добавлять эти файлы в проект не нужно.

Таблица 4. Макросы для реализации радстойких блоков

Техника реализации	Макрофайл для Verilog	Макрофайл для VHDL
cc	cc.v	cc.vhd
tmr	tmr.v	tmr.vhd
tmr_cc	tmr_cc.v	tmr_cc.vhd

2. Чтобы установить атрибут syn_radhardlevel глобально или по умолчанию, надо сделать следующее:
   - задать нужное значение атрибута в исходном файле для всего модуля. В следующем примере (табл. 5) показано, как задать для модуля module_b синтез всех регистров по технике tmr;
   - убедиться, если требуется, что соответствующий макрофайл Actel (шаг 1) упоминается в проекте первым.
3. Установить в исходном коде атрибут syn_radhardlevel для отдельных регистров. На уровне регистров можно перекрыть установленное по умолчанию значение атрибута или вообще отменить его, установив значение none. В следующем примере (табл. 5) показано, как установить атрибут syn_radhardlevel для регистра bl_int равным tmr.

Таблица 5. Пример задания атрибута syn_radhardlevel для модуля и для регистра

VHDL	Verilog
для модуля	
library synplify; use synplify.attributes.all; attribute syn_radhardlevel of behav: architecture is "tmr";	module module_b (a, b, sub, clk, rst) /*synthesis syn_radhardlevel="tmr"*/;
для регистра	
library synplify; use synplify.attributes.all; attribute syn_radhardlevel of bl_int: signal is "tmr"	reg [15:0] a1_int, b1_int /*synthesis syn_radhardlevel = "tmr"*/;

Заключение

В цикле из пяти статей, начиная с [1] и заканчивая этой статьей, мы познакомились с микропроцессорным IP-ядром 8051s корпорации Microsemi и особенностями его применения в ПЛИС для тяжелых условий эксплуатации, включая техники защиты проекта от сбоев под воздействием радиационных факторов.

Рассмотренное в рамках этого цикла микропроцессорное ядро является бесплатным и может быть интегрировано в практически любые семейства ПЛИС корпорации Microsemi. Ядро имеет полное описание, что вместе с данным курсом позволит разработчикам быстро и эффективно создать, настроить и запустить свой собственный проект.

Помимо IP-ядра 8051s в каталоге среды проектирования Libero можно найти более сотни различных интерфейсных и функциональных модулей, готовых к применению в пользовательских проектах, что может значительно сократить время разработки. При этом практически все IP-ядра предоставляются в открытом коде Verilog или VHDL. Установочный пакет среды разработки Libero можно свободно скачать с сайта www.microsemi.com, а при возникновении любых вопросов обратиться в службу технической поддержки официального представительства корпорации Microsemi SoC в России.

Авторы цикла искренне надеются, что дали читателям достаточно подробные инструкции для начала работы и сняли у специалистов, ранее не работавших с платформой Microsemi, большинство начальных вопросов.

Файлы учебного проекта из этой части статьи можно найти в Интернете [11].

Литература

1. Иоффе Д., Максимов А. Разработка проекта микроконтроллера 8051s на основе IP-ядер корпорации Microsemi // Компоненты и технологии. 2014. № 1.
2. Иоффе Д., Максимов А. Разработка проекта микроконтроллера 8051s на основе IP-ядер корпорации Microsemi. Часть 2. IP-ядро 8051s для программиста // Компоненты и технологии. 2014. № 3.
3. Иоффе Д., Максимов А. Разработка проекта микроконтроллера 8051s на основе IP-ядер корпорации Microsemi. Часть 3. Первая программа для микроконтроллера // Компоненты и технологии. 2014. № 5.
4. Иоффе Д., Максимов А. Разработка проекта микроконтроллера 8051s на основе IP-ядер корпорации Microsemi. Часть 4. Использование прерываний // Компоненты и технологии. 2014. № 6.
5. http://chemy.ksc.ru/vyskazyvaniya-o-nauke
6. Application Note AC333. Connecting User Logic to the SmartFusion Microcontroller Subsystem — www.microsemi.com
7. AMBA3 APB Protocol v1.0. Specification — www.arm.com
8. Воздействие радиации на интегральные схемы // http://ru.wikipedia.org/wiki/
9. Application Note AC139. Using Synplify to Design in Microsemi Radiation-Hardened FPGAs — www.microsemi.com
10. Working with Radhard Designs. Раздел справочной системы пакета Synplify.
11. http://www.actel.ru/catalog/8051p5_source.zip