Клуб экспертов по ПЛИС

После того, как был напечатан предыдущий отчет «Клуба экспертов» («КиТ» № 1`2007, стр. 10-15), организатор собраний Клуба Иосиф Каршенбойм обратился к участникам телеконференции по поводу новой темы. На этом заседании «Клуба экспертов» будут обсуждаться три темы, которые представляются довольно интересными. Все они связаны с разработками проектов ПЛИС. Надеемся, что уважаемые читатели примут участие в обсуждении данных вопросов.

На данном заседании присутствовали

- *Сергей Демура* — главный инженер компании «Новомар», sd@sovintel.ru
- *Денис Шехалев* — инженер научно-производственной фирмы «Микран», diod2003@list.ru
- *Иосиф Каршенбойм* — менеджер отдела технической поддержки фирмы «Элтех», iosifk@eltech.spb.ru

Повестка дня

В повестке дня есть только один вопрос. Он был задан в письме, пришедшем к нам от Павла Ядвичука. Далее мы приводим выдержки из текста его письма:

«Прочел вашу статью «Заседание Клуба экспертов» в «КиТ» № 1 '2007. Не называю себя экспертом, но есть предложение по теме. В основном занимаюсь HDL-дизайном, поэтому предложения могу вносить только в эту узенькую веточку. А предложение такое — просматривая ветки форумов на телесистемах и электронике, увидел странную закономерность, проявляющуюся у 90% участников, от монстров до начинающих — нет понимания в использовании синтезируемых/несинтезируемых языковых конструкций. Сплошь и рядом все наступают на грабли несинтезируемости в RTL-коде и, с другой стороны, ваяют километры синтезируемого кода в тестбенчах, когда можно обойтись 2 строчками несинтезируемых конструкций. Думаю, разъяснения по этому вопросу не будут лишними.

С наилучшими пожеланиями
Павел Владимирович Ядвичук, г. Одесса,
ООО «Телекарт-Прибор»»

Итак, обобщая, тему обсуждения можно сформулировать так (извините за «казенный» язык, но ведь это же «заседание», следовательно, и терминология должна быть соответствующая): «**Перечень организационно-технических мероприятий по ускорению проектирования в ПЛИС с применением новейших технологических приемов**».

Чтобы ответить Павлу, мы обратились к коллегам и предложили им высказаться. Далее вы познакомитесь с двумя предложениями по этому поводу. Но если вы хотите добавить свои высказывания или замечания, то помните, что Клуб открыт для всех!

Иосиф Каршенбойм: Слово предоставляется главному инженеру фирмы «Новомар» — Сергею Демуре. Мы знаем, что фирма «Новомар» заслужила право быть Экспертом в области проектирования на ПЛИС, мало того, эта фирма является консультантом по применению микросхем Xilinx. На рис. 1 показана часть платы, содержащей микросхемы ПЛИС. Сейчас Сергей Дмитриевич поделится с нами своими взглядами на процесс проектирования.

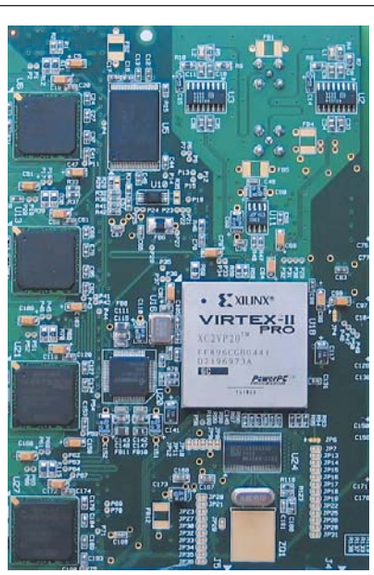

Рис. 1. Вот такого уровня изделия и выпускает фирма «Новомар»

Сергей Демура: Эта статья предназначена для тех, кто только начинает работать со сложными и большими по объему проектами, имея в своем распоряжении базовые пакеты ПО для разработки ПЛИС. Я попробую дать некоторые советы общего плана из опыта разработок нашей компании, стараясь по возможности избегать технологической привязанности к ПЛИС какой-либо конкретной фирмы.

В начале любого проекта всегда рассматриваются два возможных варианта, происходящих из допустимых экономических затрат и технологических возможностей компании:

- *Вариант 1.* Технологических и экономических ограничений нет. Для реализации проекта нет ограничений при выборе объема ресурсов, количества выводов, типа корпуса и стоимости ПЛИС.
- *Вариант 2.* Есть технологические или экономические ограничения (сложность печатной платы или стоимость ПЛИС большого объема). Проект должен быть реализован на нескольких ПЛИС меньшего объема или в определенном типе корпуса.

Начинаем, как обычно, с построения структурной и функциональной схем. С первым вариантом все просто — выбираем ПЛИС необходимого объема в удобном для нас корпусе, подключаем периферию, формируем внешние интерфейсы, разбиваем схему ПЛИС на функциональные блоки и прорабатываем их интерфейсы связи. Со вторым сложнее. Сразу возникают вопросы: как поделить схему, как организовать интерфейсы между ПЛИС? К сожалению, однозначных ответов на эти вопросы нет. В первую очередь необходимо рассмотреть количество и типы стандартов портов ввода/вывода для подключения внешних микросхем или интерфейсов. Допустим, у нас есть стандарты PCI 3.3V, LVDS25 и SSTL18. Это три разных стандарта, отличающиеся напряжениями питания, и каждая группа сигналов должна быть расположена в своем банке ввода/вывода со своим питанием. Ко всему прочему некоторые ПЛИС имеют ограничения или на общее количество дифференциальных пар, или на количество портов вывода в одном банке (как правило, это связано с количеством выводов питания банков ввода/вывода и зависит от типа корпуса). Перед выбором конкретных типов ПЛИС рекомендую внимательно проверить эти ограничения. Особое внимание обратите на ПЛИС в корпусах PQ и TQ. Как правило, они имеют ограничение на объем

внутренних логических ресурсов и количество портов по отношению к микросхемам в корпусах BGA. При экономических или технологических ограничениях задача распределения общих логических ресурсов проекта и портов ввода/вывода смешанных стандартов сильно усложняется. Для связи ПЛИС между собой, возможно, придется пользоваться оставшимися выводами смешанных стандартов. Облегчить жизнь позволяет использование последовательных интерфейсов, но при высокой скорости интерфейса возрастают сложность организации ПЛИС и требования к PCB, это тоже необходимо учитывать при организации проекта.

Как поделить проект между разработчиками? Практика показывает: каждый должен делать функционально законченную часть схемы или ПЛИС целиком, а не какие-то простые модули, зачастую являющиеся частью других, более сложных модулей. Моделирование своей части проекта на начальных этапах каждый должен также выполнять сам, это позволяет избавиться от синтаксических и грубых схемотехнических ошибок еще на стадии разработки элементарных модулей, иногда это позволяет даже переосмыслить алгоритм работы функционального блока. Не старайтесь экономить общее время разработки проектов, имеющих экономические или технологические ограничения, привязывая сигналы к номерам выводов ПЛИС на PCB до черновой проработки схемы ПЛИС. На сложных проектах это может привести к неоправданным потерям времени, связанным с переделками PCB — не всегда выбранное расположение портов позволяет достигнуть частотных требований схемы ПЛИС или совместимости выводов по стандартам. Такой подход является оправданным решением в том случае, когда вы четко знаете, что у вас есть большой частотный запас внутри ПЛИС и стандарты портов однотипны.

Одновременно с началом разработки можно начинать создание сценария для проверки функциональности всего устройства целиком — от входного потока данных до выходных и управляющих интерфейсов. Желательно, чтобы в него были включены как самостоятельно созданные модули, так и модели внешних компонентов (это зависит от возможностей вашего базового симулятора). Лучше, когда этим занимается специально выделенный человек, еще лучше, когда он больше склонен к программированию, чем к схемотехнике. Такой подход обеспечивает дополнительную проверку модулей, разработанных другими участниками проекта, и экономит время на верификацию устройства в целом. Бесплатные модели внешних компонентов (как правило, это память или очень простые контроллеры) рекомендую искать в Интернете на сайтах производителей, большинство производителей выкладывает симуляционные модели бесплатно. На крайний случай есть сайт **http://www.freemodelfoundry.com** (но это без гарантии полной работоспособности и достоверности). Для разработчиков, использующих в своих проектах модули SODIM (DIMM), рекомендую брать модели отдельных микросхем, установленных на SODIM (можно поискать на **http://www.micron.com**), и самостоятельно построить модель модуля целиком. Для красоты можно еще включить в модель и EEPROM (SPD) — функциональные схемы модулей приводятся производителями. А вот что касается всевозможных контроллеров, то здесь вопрос сложный. Есть простой способ — эмуляция. Можно эмулировать выходной поток данных и интерфейс связи до момента, когда это избавляет от грубых ошибок. В принципе, все, что для этого необходимо — это описание контроллера и временные диаграммы его работы. Здесь все зависит от знаний и опыта, ну еще, может быть, от фантазии и здравого смысла. Единственный совет: старайтесь вводить в тестбенч недопустимые состояния на шинах связи с внешними компонентами или недопустимые кодовые комбинации в потоке данных — это избавляет от многих неприятностей. Предположим, что какой-то интерфейс находится в недопустимом состоянии из-за неконтакта на плате или сбоя в линии передачи данных, а скажем, указатель поворота автомобиля, вместо того чтобы показать направо, указывает налево. Результат такой ошибки может оказаться плачевным. При разработке проектов рекомендую включать в исходные тексты задержки, вносимые схемой, как минимум, на портах ввода/вывода модулей. При синтезе они не учитываются, а при моделировании могут значительно облегчить жизнь и помогут быстрее найти и устранить ошибки. При создании тестбенча старайтесь не использовать синтезируемые конструкции — это уменьшит время симуляции. Зачем заставлять симулятор строить дополнительное устройство, являющееся генератором тестовой последовательности, то есть фактически включать в тестбенч еще одну ПЛИС?

Я не буду вдаваться в особенности языковых конструкций и обсуждать, какой язык лучше — VHDL или VERILOG. На самом деле каждый разработчик имеет какое-то предпочтение, но знать хотя бы основы другого языка — полезно. Что касается стилей написания — ответ однозначен: чем проще, тем лучше. Старайтесь по возможности избегать неоправданно длинных и сложных языковых оборотов, важна не красота написания (сложность), а качество и надежность синтезированной схемы. И не забывайте о необходимости написания комментариев в текстах — это сильно помогает при модификациях устройства в дальнейшем. При создании схемы никогда не забывайте, что вы не пишете программу для уже существующего процессора, а создаете свое собственное устройство и требования к нему — не длина исходного текста, а минимум затраченных ресурсов ПЛИС и максимум быстродействия. При создании тестбенча — наоборот, старайтесь создать программу, требующую минимальных процессорных затрат времени, а не разработать схему какого-то абстрактного генератора кода.

При синтезе проекта рекомендую всегда анализировать полученную технологическую схему. Не всегда синтезаторы делают именно то, что хотели получить вы, написав конкретную строку кода. Для проектов с высокой тактовой частотой это может оказаться очень важным. Не стесняйтесь включать в исходные тексты технологические примитивы компонентов — иногда это единственный способ заставить синтезатор сделать то, что необходимо вам, а не то, что умеет синтезатор.

Много технических решений для элементарных частей проекта можно найти на сайтах производителей ПЛИС, но не стоит забывать, что это, как правило, не готовые и уже реализованные проекты, а чаще иллюстрации возможностей конкретного семейства ПЛИС. Возможные реализации можно посмотреть и на сайте **http://www.opencores.org**. С одной стороны, анализ того, что кем-то уже сделано, может сильно облегчить работу. С другой стороны, время, требующееся на доработку или переработку этих проектов, может превысить время собственной разработки «с нуля», поэтому советую обращаться с этими проектами с осторожностью. Также рекомендую обязательно тестировать бесплатные IP-core производителей ПЛИС, не всегда их функционирование во всех режимах работы соответствует описанию.

На отладочном образце по возможности старайтесь использовать ПЛИС большего объема, чем те, которые запланированы в рабочем проекте (естественно, в случае совпадения выводов и корпусов). Это позволит, с одной стороны, быстрее начать отладку, так как ускорится процесс трассировки, а с другой стороны — позволит пользоваться инструментальными отладчиками. Иногда использование инструментального отладчика — единственный простой способ выловить хитрую ошибку, проявляющуюся один раз за час работы или один раз на 100 Гбайт входного потока данных.

Последнее замечание. Как показала практика при разработке проектов на встроенных в ПЛИС процессорах, вести проект удобнее программисту. Аппаратчика в этом случае надо подключать при начальном построении структуры проекта ПЛИС и для создания своих, не входящих в состав пакетов функциональных модулей.

Иосиф Каршенбойм: Второй «сюжет» в заседаний этого клуба также посвящен теме облегчения труда разработчика проектов в FPGA, и в том числе облегчения отладки проектов. Этот раздел начался с письма от Дениса Шехалева, с которым мы уже несколько лет переписываемся и обмениваемся мнениями по поводу ведения проектов.

```
//
// Finite State Machine  : generated by psm generator version 1.11
// Author                : Shekhalev Denis (des00)
// License               : Free for use
//

`timescale 10ns/1ns

module rx_ctrl_x1 (
 input clk, reset, sclr,
 input     [6:0] br,
 output reg [31:0] outr,
 output reg [31:0] outc
 );

reg  [3:0] pc,    pc_next;
reg  [3:0] pc_ret, pc_ret_next;

always @(posedge clk or posedge reset) begin
 if (reset) begin
   pc       <= 4'd0;
   pc_ret  <= 4'd0;
   end
 else if (sclr) begin
   pc       <= 4'd0;
   pc_ret  <= 4'd0;
   end
 else begin
   pc       <= pc_next;
   pc_ret  <= pc_ret_next;
 end
end

wire [3:0] pc_p1;
assign pc_p1 = pc + 1'b1;

always @(*) begin
 pc_next      = 4'd0;
 pc_ret_next = pc_ret;
 case (pc)   // synthesis parallel_case
  4'd0 : begin
    pc_next   = pc_p1;
    end
  ...здесь описаны состояния автомата с 4'd1 по 4'd4 ...
  4'd5 : begin
    if     ( br[1])      begin pc_next = 4'd9; end
    else if ( br[2])     begin pc_next = 4'd10; end
    else                 begin pc_next = pc_p1; end
    end
  ...здесь описаны все состояния автомата с 4'd6 по 4'd10 ...
  4'd11 : begin
    pc_next = 4'd4;
    end
  default : begin end
 endcase
end

always @(*) begin
 case (pc)   // synthesis parallel_case
    4'd0 : outc =  32'd0;
    4'd1 : outc =  32'd1;
    4'd2 : outc =  32'd4;
   ...здесь описаны все остальные комбинационные выходы автомата Мура...
   default : outc = 32'd0;
 endcase
end

always @(posedge clk or posedge reset) begin
 if (reset)
   outr <= 32'd0;
 else
 if (sclr)
   outr <= 32'd0;
 else
   case (pc_next)   // synthesis parallel_case
    4'd0 : outr <=  32'd0;
    4'd1 : outr <=  32'd1;
    4'd2 : outr <=  32'd4;
   ...здесь описаны все остальные регистровые выходы автомата Мили...

    default : outr <= 32'd0;
   endcase
end

endmodule


// всего текст с комментариями и пустыми строками занимает
135 строк (И. К.)
```

Рис. 2. Часть описания автомата на языке Verilog. Этот файл — первый результат работы программного инструмента Programable State Machine generator

```
wire [6:0] rx_ctrl_x1_br;
wire [31:0] rx_ctrl_x1_outr;
wire [31:0] rx_ctrl_x1_outc;

assign rx_ctrl_x1_br[0] = 1'b0;
assign rx_ctrl_x1_br[1] = 1'b0;
assign rx_ctrl_x1_br[2] = 1'b0;
assign rx_ctrl_x1_br[3] = 1'b0;
assign rx_ctrl_x1_br[4] = 1'b0;
assign rx_ctrl_x1_br[5] = 1'b0;
assign rx_ctrl_x1_br[6] = 1'b0;


rx_ctrl_x1 rx_ctrl_x1_inst (
 .clk(clk),
 .reset(reset),
 .sclr(sclr),
 .br(rx_ctrl_x1_br),
 .outr(rx_ctrl_x1_outr),
 .outc(rx_ctrl_x1_outc)
 );


// всего текст с комментариями и пустыми строками занимает
21 строку (И. К.)
```

Рис. 3. Шаблон вставки модуля конечного автомата на языке Verilog. Этот файл — второй результат работы программного инструмента Programable State Machine generator

```
constant   SetRxEn          ,0x01
constant   ClrRxEn          ,0x02
constant   RxWbEn           ,0x04
constant   PauseReq         ,0x08
constant   FifoWrRq         ,0x10
constant   DRamRUpdate      ,0x100

namebranch   RxPacketDone,         br0
namebranch   RxPacketOvf,          br1
namebranch   RxPacketMustSkip,     br2
namebranch   FifoFull,             br3
namebranch   WUpdateAck,           br4
namebranch   WUpdateErr,           br5
namebranch   nPauseNeed,           nbr6

reset :
 nop, 0
 nop, SetRxEn
 nop, RxWbEn
 nop, 0
idle :
 if RxPacketDone nop else wait, 0
decode :
 if RxPacketOvf jump SetPauseReq elif RxPacketMustSkip jump
SetRxEnable else nop, 0
ram_update :
 if WUpdateErr jump SetPauseReq elif WUpdateAck nop else wait,
DRamRUpdate
Write2Fifo:
 if FifoFull wait else nop,  FifoWrRq
TestForPause :
 if nPauseNeed jump SetRxEnable else nop, 0
SetPauseReq :
 nop,  PauseReq
SetRxEnable :
 nop,  RxWbEn
 jump idle, 0


// всего текст с пустыми строками занимает 35 строк (И. К.)
```

Рис. 4. Вот так выглядит текст задания, по которому программный инструмент генерирует описание автомата, приведенное на рис. 2

Вот часть письма от Дениса:

«…хочу сделать пополнение в вашу копилку тулзов :)

Сделал небольшую тулзовину для себя: генератор микропрограммных расширенных КА (название собственное) на языке Python. ***Потратил на это 2 дня, с учетом того, что до этого в Питоне ни ногой :)*** *благодаря его широким возможностям по обработке строк*

```
#-------------------------------------------------------------------
-
# global constants and dictionarys
#-------------------------------------------------------------------
-
version = 1.11

command            = ['if', 'nop', 'wait', 'jump', 'ret', 'call']
unary_command  = ['nop', 'wait', 'ret']
binary_command  = ['jump', 'call']

.....
label                = {}
constant           = {}
brrename           = {}
addr_max          = [0]
branch_idx_max  = [0]
#-------------------------------------------------------------------
-
# Verilog codes for fsm with sequental coding
#-------------------------------------------------------------------
-
verilog_code = '''
//
// Finite State Machine  : generated by psm generator version %(ver-
sion)s
// Author                : Shekhalev Denis (des00)
// License               : Free for use
//

`timescale 10ns/1ns

module %(name)s (
 input clk,    reset, sclr,
 input         [%(br_width_m1)i:0] br,
 output reg   [31:0] outr,
 output reg   [31:0] outc
 );

reg  [%(width_m1)i:0] pc,       pc_next;
reg  [%(width_m1)i:0] pc_ret,  pc_ret_next;

always @(posedge clk or posedge reset) begin
 if (reset) begin
   pc       <= %(width)i\'d0;
   pc_ret  <= %(width)i\'d0;
   end
 else if (sclr) begin
   pc       <= %(width)i\'d0;
   pc_ret  <= %(width)i\'d0;
   end
 else begin
   pc       <= pc_next;
   pc_ret  <= pc_ret_next;
 end
end
...здесь следует дальнейшее описание шаблона генерации...
endmodule
'''
```

Рис. 5. Вот так выглядит часть текста программного инструмента Programable State Machine generator

и его спискам все получилось намного проще и быстрее чем на C, C++. В отличие от чистого МПА в этом можно описывать обработку всех условий в 1 строке микропрограммы и выполняться это будет за 1 состояние КА. Поддерживаемые команды: nop, wait, jump, call, ret.

На очереди возможность выбора способа кодирования КА, пока только линейный. ***Высылаю вам сам скрипт, файл ассемблера и верилог-файл.*** *Скрипт «весит» в десятки раз меньше, чем он же скомпилированный (3 мегабайта), чтобы его запустить, нужно иметь интерпретатор Питона версии не ниже 2.4.*

Если у вас его нет, то могу выслать скомпилированный скрипт. Но думаю, что вам лучше скачать интерпретатор (10 мегабайт), освоение Питона дает интересные результаты.

Полагаю, вам будет интересно посмотреть на эту тулзу :)».

Конечно, освоение еще одного языка программирования нельзя назвать «ходом по облегчению жизни разработчика». Но ведь в теме Клуба цель заседания сформулирована так — «не легче, а лучше». Итак, давайте рассмотрим первую фразу письма, выделенную жирным шрифтом. Само по себе, без реальных примеров, такое суждение пока ничего нам не говорит. Ну так давайте рассмотрим и вторую фразу. Что касается Питона, то у меня (И. К.) его освоение еще впереди. Поэтому привожу здесь части файлов, присланные Денисом, а также ссылки на ресурсы в Сети.

Начнем с того, что нам «возвращает» программный инструмент, сделанный Денисом, — Programable State Machine generator.

Программный инструмент создает два файла. Первый файл, показанный на рис. 2, представляет собой часть описания автомата на языке Verilog. Второй файл (рис. 3), создаваемый программным инструментом, — это файл шаблона для вставки полученного модуля конечного автомата в проект более высокого уровня. Этот файл также выполнен для проекта на языке Verilog.

Это как раз и будет результат работы программного инструмента, сделанного Денисом.

Ну а теперь посмотрим, что же было на «входе», для того чтобы любой пользователь смог получить описание данного автомата. Текст, приведенный на рис. 4, и является как раз тем самым заданием, по которому программный инструмент генерирует описание автомата. Арифметический итог выглядит так: 135+21 строка делим на 35 строк, получаем сокращение количества строк описания в 4,4 раза! Причем без ошибок! И по отлаженному шаблону. А для использования модуля достаточно воспользоваться известным приемом Copy/Paste. А где же они, эти «волшебные пузырьки», в которых «все дело»… Конечно, все дело в файле, часть которого приведена на рис. 5.

Как мы видим на рис. 5, текст шаблона весьма похож на Verilog. Ну, а как с арифметикой на этот раз? Здесь надо сразу признать, что инструмент занимает 906 строк кода. Но ведь это же инструмент, а при разработке инструмента арифметический подход уже не годится. Ведь инструмент делается лучшим инженером фирмы, а пользуются им все сотрудники. Вот отсюда и получается ощутимая экономия.

Тех, кто хочет познакомиться с описанием языка Питон, могу адресовать на сайт **http://www.w3.org/1999/xhtml**, где находится раздел «Python, Материал из Википедии — свободной энциклопедии». Также может быть интересен и сайт **http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd**, где представлены материалы по языку MyHDL. Автор данного языка применил Питон для проектирования аппаратуры. Проект ведется на MyHDL, а результат транслируется в язык группы HDL — Verilog.

Иосиф Каршенбойм: Итак, коллеги, мы рассмотрели все вопросы, стоявшие на повестке дня. Еще раз хочу отметить, что Клуб Экспертов — это не учебник по «применению». Эксперты высказывают свои мнения по тем вопросам, которые им задавали. Конечно, стиль изложения почти «телеграфный». Но он дает возможность понять, в какую сторону надо двигаться, чтобы достичь уровня Экспертов. На этом заседание Клуба Экспертов объявляю закрытым. Вопросы — по почте. Предложения по поводу следующего заседания — также по почте. Копию протокола, с вашего согласия, я направляю редактору журнала.

Заседание закрыто. ■