Построение Ethernet-контроллера на ПЛИС

Алексей ПАШИНОВ

Введение

Идея построить собственный Ethernet-контроллер появилась в результате необходимости найти замену микросхеме Wiznet w5300. Данная микросхема — это однокристальный Ethernet-контроллер на 10/100 Мбит/с, аппаратно реализующий стек протоколов TCP/IP: TCP, UDP, ICMP, IPv4, ARP, IGMP, PPPoE. Но у таких микросхем есть существенный недостаток, связанный с ее заменой на какой-нибудь аналог, ведь в случае снятия микросхемы с производства или невозможности иметь к ней доступ возникнет множество проблем. Во-первых, для них нет общего стандарта, в отличие от тех же микросхем физического уровня (PHY), поэтому у всех производителей они существенно отличаются друг от друга. Следовательно, при переходе на новый контроллер придется заново изучать особенности его работы и переписывать пользовательский уровень, который обеспечивает связь с контроллером и его управление. Во-вторых, рынок предоставляет довольно ограниченный выбор таких микросхем, в отличие от тех же PHY, что также может создать трудности в поисках нового решения. Таким образом, появилась идея спроектировать собственный Ethernet-контроллер, который исключит данные недостатки.

Физический уровень было решено реализовать с помощью отдельной микросхемы трансивера (PHY), а вышестоящие протоколы полностью описать на ПЛИС. Как такового выбора, что использовать — ПЛИС или микроконтроллер, не возникало. Проект, к которому добавлялся Ethernet, был описан на ПЛИС, где оставалось достаточно места, чтобы добавить Ethernet, поэтому применение ПЛИС стало очевидным решением. Если же в качестве микросхемы управления на плате используется микроконтроллер (МК), то отдельно ставить ПЛИС для Ethernet нет никакого смысла (хотя можно обдумать вариант, где для таких целей можно использовать небольшие по размерам, объемам, потребляемой мощности, а также ценам ПЛИС фирмы Lattice, чтобы, например, разгрузить работу МК).

Итак, получим довольно гибкий проект: трансивер — решение легко заменимое, как было сказано выше, в силу того что оно выполнено по единому стандарту (останется только его сконфигурировать); проект, созданный на программируемой логике, можно без труда переносить между различными ПЛИС (в данном случае будут использоваться разные IP-ядра от Xilinx, которые поддерживаются сериями Spartan и Virtex, а потому проект можно переносить между этими сериями).

Технические требования, предъявляемые к будущему контроллеру

В отличие от микросхемы Wiznet, будущий контроллер должен поддерживать скорость 100/1000 Мбит/с; в качестве протокола обмена будет использоваться UDP, а для установления и проверки связи — ARP-протокол; реализована функция Hot Plug. Иными словами, если во время работы подключить к контроллеру устройство, действующее в другом режиме, контроллер должен автоматически переконфигурироваться.

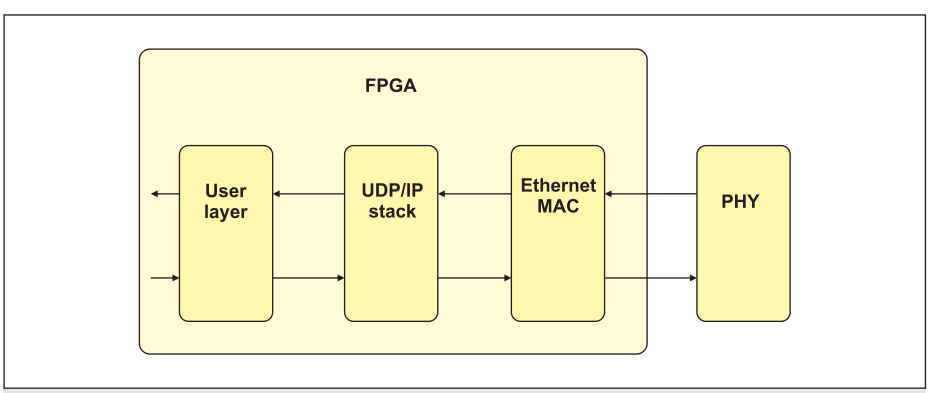

Рис. 1. Блок-схема проекта Ethernet-контроллера

Описание процесса разработки

Описание процесса построения Ethernet-контроллера будет представлено на примере имеющейся отладочной платы sp-605, на которой стоит ПЛИС серии Spartan 6 и микросхема физического уровня Marvell 88e1111. Структура будущего контроллера приведена на рис. 1.

Как видно на рисунке, контроллер состоит из четырех основных частей. PHY (Physical layer) — трансивер, взятый в качестве отдельной микросхемы для реализации физического уровня. Далее все последующие уровни описываются непосредственно в ПЛИС. Ethernet MAC — блок, реализующий канальный уровень. Блок UDP/IP аппаратно реализует сетевой и транспортный уровни. В блоке User layer описана работа с информацией, поступающей из блока UDP/IP, а также выставление данных на этот блок.

PHY

Для реализации физического уровня используется гигабитный конфигурируемый трансивер Marvell 88e1111. Он поддерживает несколько интерфейсов обмена данными с MAC-уровнем (GMII/MII, RGMII, SGMII, TBI, RTBI). Выбор пал на GMII, поскольку он позволяет передавать данные на трех скоростях: 10/100 Мбит/с (режим MII) и 1 Гбит/с (режим GMII). Также микросхема имеет интерфейс для конфигурации — MDIO (Management Data Input/Output), который позволяет управлять регистрами трансивера. Подробно о его функционале и конфигурации будет рассказано дальше.

Ethernet MAC

Для реализации канального уровня используется IP-ядро, бесплатно предоставляемое Xilinx, — Tri-mode Ethernet MAC v4.6. Данное ядро поддерживается сериями Virtex-4, 5, 6 и Spartan-3,3A, 3E, 6.

На рис. 2 приведена блок-схема этого ядра, состоящего из двух блоков: Ethernet MAC core и Ethernet MAC FIFO.

Ethernet MAC core посредством блока GMII/MII interface реализует прием и передачу данных в формате GMII/MII, связывая канальный и физический уровни. Блок Core выполняет функции канального уровня:

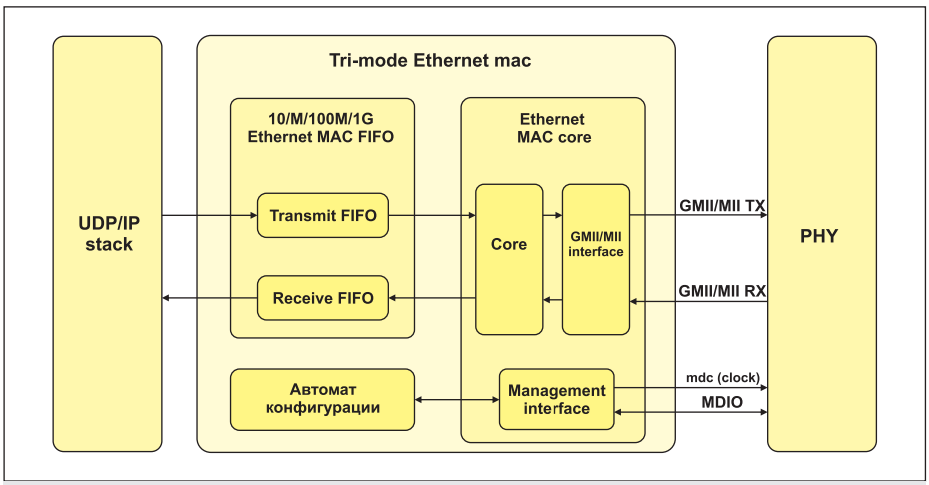

Рис. 2. Блок-схема IP-ядра Tri-mode Ethernet MAC v4.6

добавление/удаление преамбулы в начале поля кадра, добавление/удаление битов заполнения, проверка контрольной суммы кадра при приеме данных и формирование контрольной суммы при отправке. Management interface служит для конфигурации MAC-ядра, а также для доступа к регистрам PHY.

Ethernet MAC FIFO — буфер памяти типа FIFO. Обеспечивает связь с сетевым уровнем и необходим для синхронизации блоков Tri-mode Ethernet MAC и UDP/IP. Далее будет подробно разобрана потребность в такой синхронизации.

На рисунке можно заметить еще один блок, который называется «Автомат конфигурации» и не входит в предоставляемое IP-ядро, а описывается пользователем, если есть необходимость в настройке параметров трансивера и MAC-ядра. Остановимся на этом подробней.

Конфигурация

В требованиях к контроллеру были указаны автоматическая конфигурация и реализация функции Hot Plug. В первом требовании главную роль играет такой параметр трансивера, как автосогласование. Задача автосогласования состоит в том, чтобы найти способ обмена данными между двумя сетевыми адаптерами, подключенными к одному UTP-каналу, независимо от того, принадлежат ли они к одной версии Ethernet и работают ли в одном режиме. То есть в результате автопереговоров устройства связываются и автоматически выставляют нужные параметры для обмена данными (технология передачи, режим дуплекса). Данные о результатах автосогласования хранятся в статусных регистрах трансивера.

Трансивер Marvell 88E1111 имеет конфигурационные выводы, которые можно применить для аппаратной конфигурации трансивера. На плате sp-605 аппаратная конфигурация используется, но эти настройки нельзя поменять на плате (единственный способ — конфигурировать программно). Трансивер таким методом сконфигурирован на работу режима GMII, и включен необходимый для наших требований режим автосогласования. Что касается остальных аппаратных настроек, таких как включение сберегающего режима, отключение генерации выходного тактового сигнала 125 МГц, указание физического адреса трансивера, их тоже оставляем как есть. Соответственно, с помощью отладочной платы можно воспользоваться аппаратной конфигурацией трансивера и не конфигурировать его программно.

Переходим к конфигурации ядра Tri-mode Ethernet MAC. Главная задача — сконфигурировать MAC-ядро на ту же скорость, на какую согласован трансивер. Для этого необходимо сначала считать статусные регистры трансивера, в которых хранится информация о скорости его работы, настроенной в результате автосогласования, и на основе полученных данных переконфигурировать MAC-ядро.

Так мы постепенно подошли к реализации функции Hot Plug. Для этого MAC должен постоянно считывать регистры трансивера и, если содержание регистров, в которых хранится информация о скорости, изменилось, на основе новых данных переконфигурировать ядро Tri-mode Ethernet MAC. Ниже приведен VHDL-код, описывающий процесс конфигурации.

```
counter_process : process(gtx_clk)
begin
if(rising_edge(gtx_clk)) then
if(counter_1 = 1_250_000) then
   counter_2 <= counter_2 + 1;
   counter_1 <= 0;
   clk_state <= '1';
else
counter_1 <= counter_1 + 1;
clk_state <= '0';
end if;
end if;
end process;


Host_interface_state : process(gtx_clk)
begin
if(rising_edge(gtx_clk)) then
  if (reset = '1') then
    state <= state_1;
  else
    case(state) is
  when state_1 =>
    if (clk_state = '1') then
      state <= state_2;
    end if;
  when state_2 =>
    if(clk_state = '1') then
      state <= state_3;
    end if;
when state_3 =>
if(mdio_buf_read(1)= host_rd_data(15) and mdio_buf_read(0)=
host_rd_data(14)) then
state <= state_2;
      else
        state <= state_4;
      end if;
  when state_4 =>
        state <= state_2;
  end case;
end if;
end if;
end process;

Host_interface : process(gtx_clk)
begin
if(rising_edge(gtx_clk)) then
  if (reset = '1') then
    host_opcode          <= "10";
    host_addr            <= "0000000000";
    host_wr_data         <= x"00000000";
    host_miim_sel        <= '1';
  else
    case(state)          is
    when state_1 =>
      host_opcode        <= "01";
      host_addr          <= "1101000000";
      host_wr_data       <= x"0000007F";
      host_miim_sel      <= '0';
    when state_2 =>
      if(clk_state = '1') then
        host_opcode      <= "10";
        host_addr        <= "0011110001";

        host_miim_sel    <= '1';
        host_req         <= '1';
      else
        host_opcode(0)   <= '1';
        host_opcode(1)   <= '1';
        host_miim_sel    <= '1';
        host_req         <= '0';
        host_addr        <= "0000000000";
        host_wr_data     <= x"00000000";
      end if;
    when state_3 =>
      if(mdio_buf_read(1) /= host_rd_data(15) or mdio_buf_
read(0) /= host_rd_data(14)) then
        mdio_buf_read(1 downto 0) <= host_rd_data(15 downto 14);
      end if;
    when state_4 =>
      host_opcode          <= "01";
      host_addr            <= "1100000000";
      host_wr_data (29 downto 0) <= "00000000000000000000000000
000000";
      host_wr_data (30)    <= mdio_buf_read(0);
      host_wr_data (31)    <= mdio_buf_read(1);
      host_miim_sel        <= '0';
    end case;
  end if;
end if;
end process
```

Процесс counter_process описывает выработку сигнала clk_state, который с периодичностью 0,01 с выставляет единицу на один такт. По этому сигналу происходит переход между состояниями автомата конфигурации.

Сам автомат разбит на два процесса. В Host_interface_state описана логика переходов, а процесс Host_interface описывает регистр текущего состояния, используемый для хранения кода текущего состояния.

В состоянии state_1 происходит включение интерфейса MDIO посредством записи в регистр по адресу "1101000000" значения x "0000007F". В состоянии state_2 MAC-ядро считывает регистр трансивера по адресу

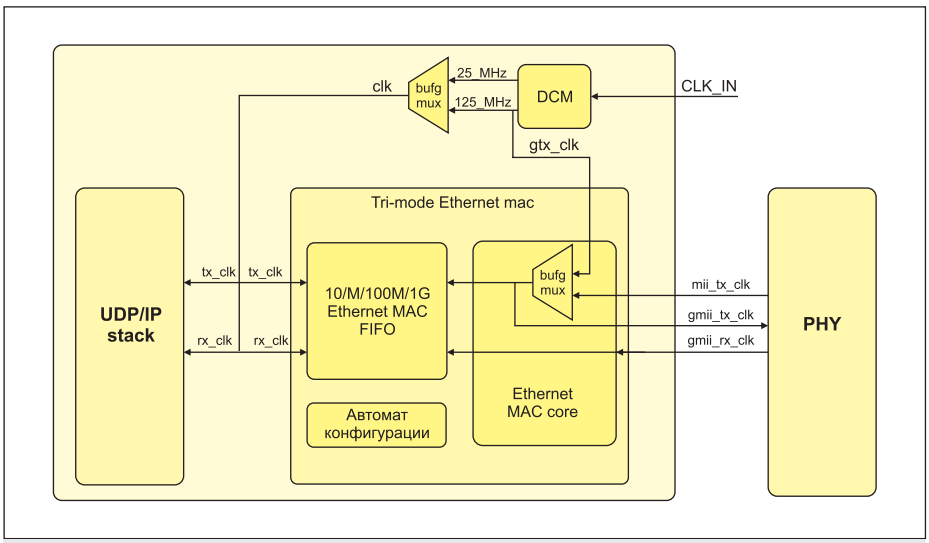

Рис. 3. Блок-схема тактирующих сигналов

"0011110001". Биты 15 и 14 этого регистра содержат информацию о скорости согласования. Если 15 бит равен 1, а 14 — 0, то выбран режим 1 Гбит/с, если 15 бит равен 0, а 14 — 1, то 100 Мбит/с. Переходим к состоянию state_3. По умолчанию MAC-ядро сконфигурировано на работу в режиме 1 Гбит/с, что соответствует записи в регистр по адресу "1100000000" в биты 31 и 30 значений 1 и 0 соответственно. Поэтому создаем вектор mdio_buf_read (1 downto 0) = 10. Теперь сравниваем считанные значения со значением вектора mdio_buf_read. Если они совпадают, следовательно, трансивер сконфигурирован на 1 Гбит/с, как и MAC, по умолчанию, поэтому настройки MAC-ядра оставляем без изменений и возвращаемся в state_2. Если значения не равны, переходим в состояние state_4, в котором конфигурируем MAC-ядро на новую скорость, соответствующую скорости трансивера. Затем снова возвращаемся в state_2 для дальнейшего опроса статусного регистра PHY.

Разводка тактовых сигналов

Теперь, когда настроены трансивер и ядро Tri-mode Ethernet MAC, осталось разобраться с сигналами тактирования. На рис. 3 приведена подробная карта тактовых сигналов, используемых в проекте. DCM (Digital Clock Manager) вырабатывает два тактовых сигнала: 25 МГц — для работы на скорости 100 Мбит/с, и 125 МГц — для работы на скорости 1 Гбит/с. Эти сигналы подаются на вход мультиплексора bufgmux, который управляется сигналом S (на рисунке не показан). Сигнал S формируется в зависимости от поступивших данных о согласовании с PHY (по аналогии с конфигурацией MAC-ядра). Ядро Tri-mode Ethernet MAC также имеет глобальный мультиплексор, если ядро конфигурируется на 100 Мбит/с, то выбирается сигнал mii_tx_clk (25 МГц), идущий с PHY, если ядро сконфигурировалось на 1 Гбит/с, то выбирается сигнал gtx_clk (125 МГц), поступающий с DCM. Сигнал gmii_tx_clk, идущий в PHY, необходим для синхронизации PHY и MAC, а также для выработки gmii_rx_clk в 125 МГц, поскольку внешний тактовый сигнал для PHY только 25 МГц. Следует обратить внимание на то, что блок Ethernet MAC core тактируется сигналами, идущими с трансивера (за исключение линии TX в режиме 1 Гбит/с), а блок UDP/IP — внутренним тактовым сигналом, вырабатываемым с DCM. IP-ядро имеет встроенный буфер памяти типа FIFO, позволяющий синхронизировать работу между блоками Tri-mode Ethernet MAC и UDP/IP. Нужно только правильно подать тактовые сигналы на входы чтения и записи данного блока.

На блок-схеме не показан сигнал, тактирующий автомат конфигурации. Его можно выбрать на усмотрение разработчика. К примеру, могут использоваться сигналы gtx_clk или clk.

Таким образом, получаем полностью описанные физический и канальный уровни.

UDP/IP-блок

Блок UDP/IP описывает протоколы IP, UDP и ARP. Он был взят в качестве готового проекта с ресурса opencores, авторами которого являются Питер Фолл и Адриан Фиргольски. Структурная схема блока представлена на рис. 4.

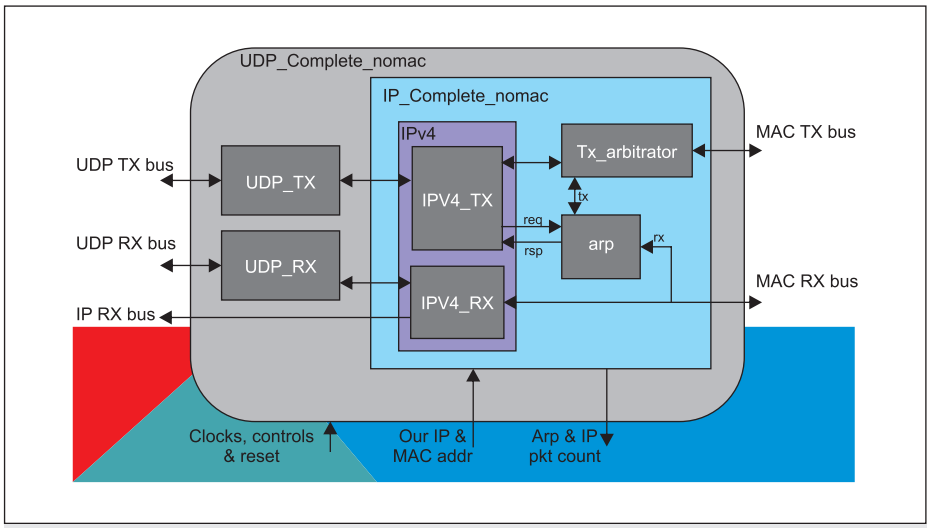

Рис. 4. Блок-схема UDP/IP-блока

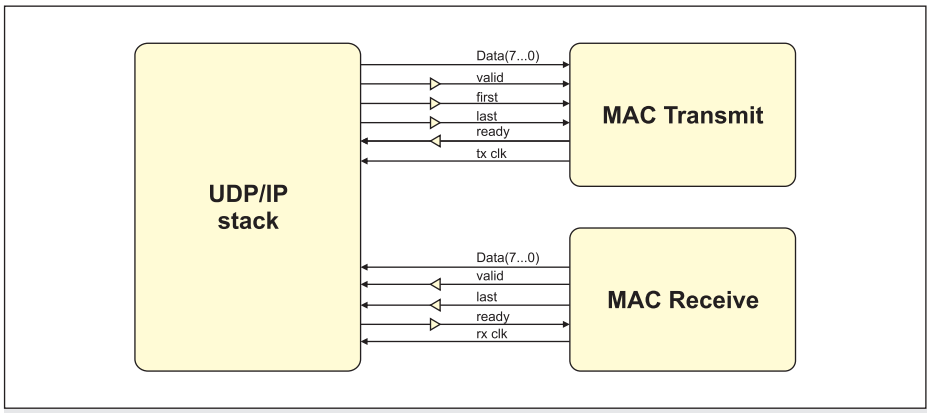

Рис. 5. Соединение блоков Tri-mode Ethernet MAC и UDP/IP

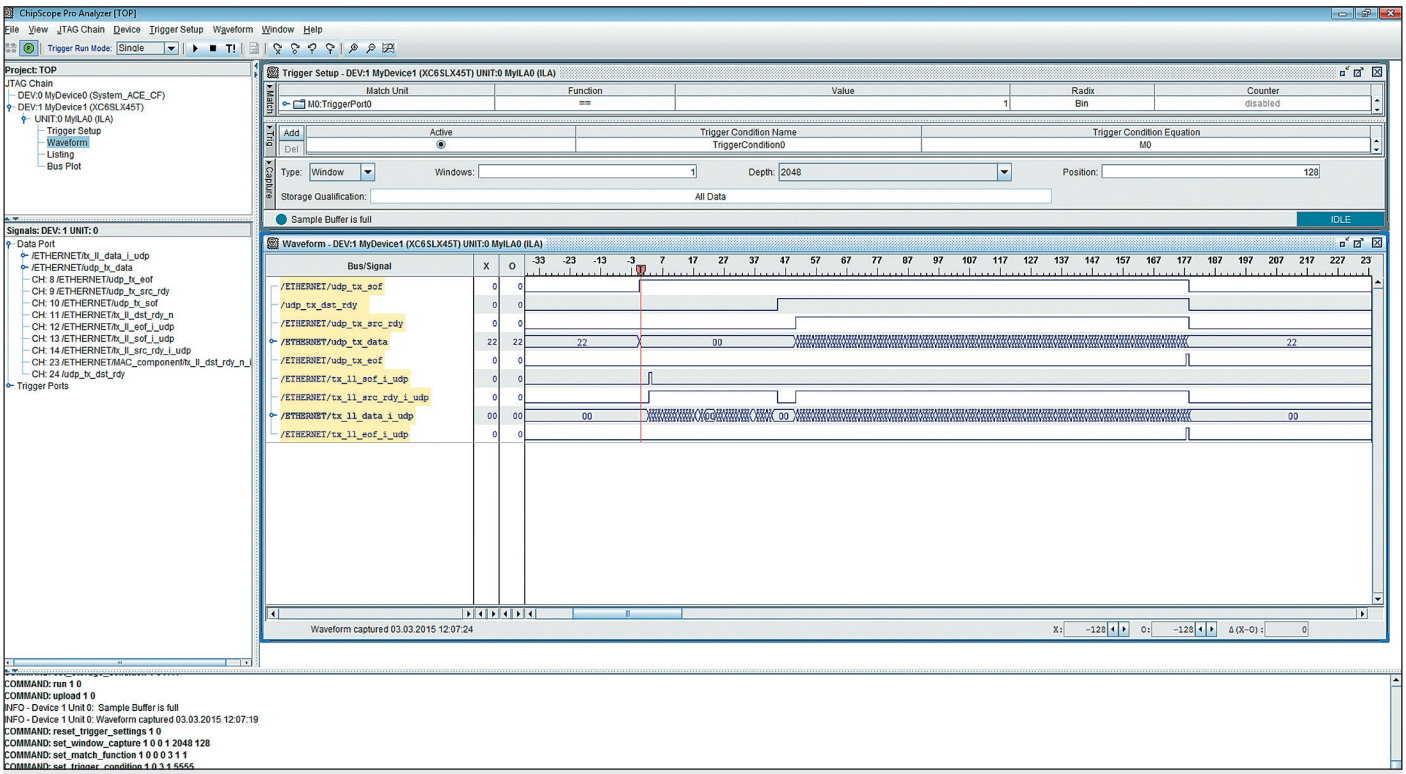

Рис. 6. Выставление данных на блок UDP/IP

Это готовый проект, аппаратно описывающий протоколы IP, UDP и ARP. Интерфейс, связывающий данный блок с канальным уровнем, полностью совместим с интерфейсом MAC-ядра. Но есть одно замечание: управляющие сигналы (valid, first, last и ready) необходимо инвертировать, как показано на рис. 5.

В данном блоке описываются автоматы, реализующие инкапсуляцию пакетов IP и UDP при передаче и деинкапсуляцию при приеме. Также большая часть проекта уделена описанию протокола ARP. В целом можно не вдаваться в подробности этого блока, а использовать его как «черный ящик».

Пользовательский уровень

Работа с блоком UDP/IP

Выставление передаваемых данных на блок UDP/IP показано на рис. 6. Работает это следующим образом. Сначала выставляется сигнал udp_tx_sof (рис. 6), сообщающий о начале передачи. Одновременно должна быть выставлена информация, содержащая IP-адрес получателя, адреса портов получателя и отправителя и длина передаваемых данных (на рисунке не показаны). После этого в самом блоке UDP/IP выставляется сигнал tx_ll_src_rdy_i_udp, и на линии данных tx_ll_data_i_udp начинает формироваться заголовок udp-пакета. Как только заголовок сформирован, UDP/IP сообщает о своей готовности принимать данные, выставляя сигнал udp_tx_dst_rdy. Далее, как только пришел сигнал от этого блока о разрешении передавать данные, выставляется сигнал начала передачи данных udp_tx_src_rdy и сами данные, которые затем передаем. При передаче последнего байта необходимо выставить сигнал конца передачи — udp_tx_eof. По завершении передачи все управляющие сигналы сбрасываются.

Проблема синхронизации контроллера и некой системы, к которой подключен данный контроллер

Имеем готовый Ethernet-контроллер, который можно назвать упрощенной альтернативой контроллеру Wiznet (без реализации нескольких протоколов — TCP, ICMP, IGMP, PPPoE, зато поддерживающий режим 1 Гбит/с). Важной задачей является совместить работу написанного контроллера с остальной логикой устройства. Как мы помним, контроллер может работать на частотах 2,5, 25 и 125 МГц в зависимости от того, на какой режим работы он настроен. Но система может работать на скорости, отличной от вышеприведенных. Возникает проблема синхронизации при выставлении данных описанному Ethernet-контроллеру. Для того чтобы решить проблему синхронизации, можно использовать два двухпортовых блока памяти типа RAM, в которых чтение и запись происходят под управлением разных тактовых сигналов. С одной стороны подводится сигнал тактирования, с которым работает система, с другой — сигнал тактирования контроллера. Также блок RAM может работать с данными, чья ширина для чтения и для записи не совпадает. Контроллер, к примеру, работает с данными, ширина которых 8 бит, и если в системе ширина данных 16 бит, то применение блока RAM легко решает проблему. Вместо RAM можно использовать буфер FIFO, кому что по душе, главное — следить, чтобы эти элементы не переполнялись.

Заключение

В силу понятных причин в данной статье опущены многие подробности этапов разработки и множество нюансов, с которыми в ходе данного процесса сталкивается инженер. Но, тем не менее, она позволяет создать общую картину проекта, дает понимание, из каких частей он состоит и каким образом они могут быть реализованы.