Разработка контроллера протокола MIL-STD-1553B на ПЛИС. Часть 3

В предыдущей, второй части статьи автор начал рассматривать HDL-код проекта на ПЛИС, который описывает контроллер авиационного протокола MIL-STD-1553B. Из всех модулей были рассмотрены передатчик (Transmitter.v) и приемник (Receiver.v). Были приведены временные диаграммы для лучшего понимания кода модулей.
Модули передатчика и приемника обеспечивают декодирование и кодирование слов стандарта MIL-STD-1553B соответственно. Пакетной организацией этих слов занимается модуль RT_control.v. Именно ему и посвящена эта часть статьи.

Дмитрий ДАЙНЕКО
dyne@micran.ru

Модуль RT_control.v

Перейдем к описанию модуля, реализующего алгоритм работы оконечного устройства. Что это значит? Вспомним, что оконечное устройство должно правильно реагировать на командные слова — принимать или отсылать информационные слова и организовывать ответное слово.

Рассмотрим интерфейс ввода/вывода этого модуля:

```
module RT_control (

  input clk,
  input reset,

  //receiver
  input rx_done,
  input [15:0] rx_data,
  input rx_cd,
  input p_error,

  //transmitter
  output tx_ready,
  output [15:0] tx_data,
  output tx_cd,
  input tx_busy,

  //memory dev3
  input [4:0] addr_rd_dev3,
  input clk_rd_dev3,
  output [15:0] out_data_dev3,
  output busy_dev3,

  //memory dev5
  input [15:0] in_data_dev5,
  input [4:0] addr_wr_dev5,
  input clk_wr_dev5,
  input we_dev5,
  output busy_dev5
    );
```

Под комментарием //receiver представлены сигналы взаимодействия с модулем приемника Receiver.v. Сигнал rx_done — это короткий импульс шириной в один период clk16. Этот импульс сообщает модулю RT_control.v о том, что принято какое-то слово. Сигнал rx_data[15:0] — параллельные данные, принятые от контроллера канала. Сигнал rx_cd показывает, какое именно слово принято — командное (0) или информационное (1). Наконец, p_error устанавливается в «лог. 1» в случае ошибки паритета в принятом слове.

Под текстом комментария //transmitter объявлены сигналы взаимодействия с модулем Transmitter.v. На шине tx_data[15:0] выставляются данные, которые необходимо передать на контроллер канала. С помощью сигнала tx_cd можно указать, что именно оконечное устройство будет передавать — ответное (0) или информационное слова (1). Для передачи следует выработать импульс tx_ready, ширина которого должна быть не менее одного периода clk16. Процесс передачи сопровожден «лог. 1» на сигнале tx_busy.

Под комментариями //memory dev3 и //memory dev5 приведены интерфейсы внутренней памяти для устройств с субадресами 3 и 5 соответственно. Назовем их просто: device3 и device5. Как и в любом интерфейсе памяти, здесь присутствуют шина адреса (addr_rd_dev3, addr_wr_dev5), шина данных (data_rd_dev3, data_wr_dev5) и тактовый сигнал (clk_rd_dev3, clk_wr_dev5). Было решено, что устройство device3 будет принимать информацию от контроллера канала, поэтому интерфейс memory dev3 служит для извлечения принятой информации на остальную периферию (Other Logic). Устройство device5 будет отсылать на контроллер канала ту информацию, которую загрузит во внутреннюю память Other Logic. Поэтому в интерфейсе для device5 присутствует еще и сигнал разрешения записи в память (we_dev5). Для мониторинга процессов приема и передачи пакетов служат сигналы busy_dev3 и busy_dev5 соответственно.

На рис. 17 представлены временные диаграммы интерфейса ввода/вывода этого модуля.

Перейдем к рассмотрению тела модуля. Адреса и субадреса опишем в виде параметров:

```
parameter [4:0] ADDRESS = 5'd1;

parameter [4:0] SUBADDR_3 = 5'd3,
                SUBADDR_5 = 5'd5;
```

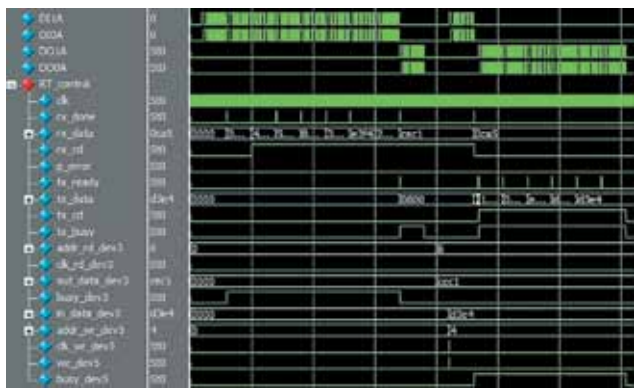

Рис. 17. Временные диаграммы интерфейса ввода/вывода модуля RT_control.v

Вспомним, что в разделе описания протокола были приведены понятия адреса оконечного устройства и субадреса. К магистрали протокола MIL-STD-1553B могут быть подключены, помимо нашего, какие-то другие устройства. Поэтому именно наше устройство будет откликаться только тогда, когда в принятом командном слове поле ADDR RT будет иметь значение ADDRESS = 5'd1.

Далее нам необходимо подключить к проекту два подмодуля, каждый из которых будет выполнять нужные действия, соответствующие его субадресу:

```
defparam device3.ADDRESS = ADDRESS;
device3 device3 (
   .clk (clk),
   .reset (reset),
   .start (dev3),
   .rx_done (rx_done),
   .rx_data (rx_data),
   .p_error (p_error),
   .tx_data (tx_data_dev3),
   .tx_cd (tx_cd_dev3),
   .tx_ready (tx_ready_dev3),
   .addr_rd (addr_rd_dev3),
   .clk_rd (clk_rd_dev3),
   .out_data (out_data_dev3),
   .busy (busy_dev3)
      );

defparam device5.ADDRESS = ADDRESS;
device5 device5 (
   .clk (clk),
   .reset (reset),
   .start (dev5),
   .rx_data (rx_data),
   .p_error (p_error),
   .tx_data (tx_data_dev5),
   .tx_cd (tx_cd_dev5),
   .tx_ready (tx_ready_dev5),
   .addr_wr (addr_wr_dev5),
   .clk_wr (clk_wr_dev5),
   .in_data (in_data_dev5),
   .we (we_dev5),
   .busy (busy_dev5)
      );
```

Интерфейсы ввода/вывода этих подмодулей будут рассмотрены ниже. Следует только добавить, что каждый их этих подмодулей включает в себя память данных (для приема или передачи), а также конечный автомат, который в нужной последовательности принимает/передает информационные слова и ответное слово. Так как в ответном слове необходимо передавать, помимо служебной информации, еще и адрес оконечного устройства, то с помощью ключевого слова defparam каждому из подмодулей передали значение параметра ADDRESS.

К device3.v и device5.v должен подходить какой-нибудь сигнал, сообщающий, что от контроллера канала пришло командное слово именно для этого модуля:

```
wire wr_rd = rx_data[10];

wire dev3 = ((~rx_cd)
             &(rx_data[15:11] == ADDRESS)
             &(rx_data[9:5] == SUBADDR_3)
             &(rx_done)
             &(~wr_rd));

wire dev5 = ((~rx_cd)
             &(rx_data[15:11] == ADDRESS)
             &(rx_data[9:5] == SUBADDR_5)
             &(rx_done)
             &(wr_rd));
```

Для этого за основу автор решил взять импульс окончания приема rx_done от модуля Receiver.v. Для подмодуля device3.v импульс старта dev3 будет вырабатываться одновременно с rx_done при дополнительных условиях:

- Принято именно командное слово (rx_cd = 1'b0).
- Поле ADDR RT содержит именно то, что нам нужно (rx_data[15:11] = ADDRESS).
- Поле SUBADDR имеет значение, соответствующее подмодулю device3.v (rx_data[9:5] = SUBADDR_3).
- Принятое командное слово требует именно передачи информационных слов на оконечное устройство (wr_rd = rx_data[10] = 1'b0).

Для подмодуля device5.v импульс старта dev5 будет вырабатываться одновременно с rx_done при следующих условиях:

- Принято именно командное слово (rx_cd = 1'b0).
- Поле ADDR RT содержит именно то, что нам нужно (rx_data[15:11] = ADDRESS).
- Поле SUBADDR имеет значение, соответствующее подмодулю device5.v (rx_data[9:5] = SUBADDR_5).
- Принятое командное слово требует именно приема информационных слов от оконечного устройства (wr_rd = rx_data[10] = 1'b1).

Нам осталось организовать мультиплексор данных для передатчика Transmitter.v, потому что ему должны выставлять данные для передачи как device3.v, так и device5.v:

```
reg sel = 1'b0;
wire [15:0] tx_data_dev3;
wire tx_cd_dev3;
wire tx_ready_dev3;
wire [15:0] tx_data_dev5;
wire tx_cd_dev5;
wire tx_ready_dev5;

always @ (posedge clk)
begin
   case ({dev3, dev5})
      2'b10:sel <= 1'b0;
      2'b01:sel <= 1'b1;
   endcase
end

assign tx_data = (sel) ? tx_data_dev5 : tx_data_dev3;
assign tx_cd = (sel) ? tx_cd_dev5 : tx_cd_dev3;
assign tx_ready = (sel) ? tx_ready_dev5 : tx_ready_dev3;
```

Управляющим сигналом мультиплексора является sel. В переключении сигнала sel участвуют все те же импульсы старта dev3 и dev5. Другими словами, после прихода соответствующего командного слова к передатчику Transmitter.v подключаются нужные данные и линии управления.

Модуль device3.v

Напомним, что этот подмодуль обеспечивает прием данных от контроллера канала с субадресом 3. Рассмотрим интерфейс ввода/вывода:

```
module device3 (
   input clk,
   input reset,

   input start,

   //rx interface
   input rx_done,
   input [15:0] rx_data,
   input p_error,

   //tx interface
   output reg [15:0] tx_data,
   output reg tx_cd,
   output reg tx_ready,

   //memory interface
   input [4:0] addr_rd,
   input clk_rd,
   output [15:0] out_data,
   output reg busy
      );
```

Мы уже знаем, что импульс start инициирует процесс приема данных и запись их во внутреннюю память ПЛИС. Другими словами, сигнал start запускает конечный автомат.

Под текстом комментария //rx interface приводятся сигналы от модуля Receiver.v. Принятое слово будет находиться на шине rx_data[15:0]. Импульс rx_done в данном конкретном случае будет говорить о приходе очередного информационного слова. При несоблюдении правила паритета в принятом слове на ошибку будет указывать сигнал p_error.

Под текстом комментария //tx interface описаны сигналы для модуля Transmitter.v. В модуле с помощью этих сигналов будет инициирована передача ответного слова на контроллер канала.

Естественно, принятая от контроллера канала информация должна быть доступна остальной логике HDL-проекта на ПЛИС, поэтому под комментарием //memory interface приведен интерфейс обычной памяти (адрес, данные и тактовый сигнал), дополненный сигналом busy, который будет являться индикатором занятости памяти (производится прием и запись информации во внутреннюю память ПЛИС).

Далее рассмотрим тело модуля, которое можно разделить на три части: приведение числа принимаемых слов к «адекватному», двухпортовое ОЗУ с разделенными портами записи и чтения и сам конечный автомат.

Вспомним, в разделе описания протокола было сказано, что в поле N/COM командного слова указывается количество слов, которое необходимо принять. Причем при значении /COM, равном нулю, это означает количество слов, равное 32:

```
reg [4:0] num_word = 5'd0;
reg [4:0] num_word_buf = 5'd0;

always @ (num_word)
   case (num_word)
      5'd0:num_word_buf = 5'd31;
      default:num_word_buf = num_word - 1'b1;
   endcase
```

Также в нашем модуле необходимо подключить память:

```
reg [4:0] addr_wr;
reg clk_wr;
reg we;
wire [15:0] in_data = rx_data;
```

```
mem_dev3 mem_dev3
(
   .data (in_data),
   .read_addr (addr_rd), .write_addr (addr_wr),
   .we (we),
   .read_clock (clk_rd), .write_clock (clk_wr),
   .q (out_data)
);
```

Память для хранения принятой информации представляет собой ОЗУ, у которого шины адреса, шины данных, а также тактовый сигнал разделены — для чтения и для записи. Каждый производитель ПЛИС предоставляет свой вариант HDL-кода такой памяти, чтобы оптимально использовались ресурсы встроенных блоков памяти.

Конечный автомат, реализующий последовательные этапы приема посылки от контроллера канала:

```
parameter IDLE_STATE = 8'd0,
          START_STATE = 8'd1,
          DATA_WAIT_STATE = 8'd2,
          DATA_SAVE_STATE = 8'd3,
          CHECK_NUM_STATE = 8'd4,
          LOAD_OS_STATE = 8'd5,
          SEND_OS_STATE = 8'd6;
```

Состояния автомата:

- IDLE_STATE — ожидание импульса на старт приема информационных слов.
- START_STATE — начало работы.
- DATA_WAIT_STATE — ожидание следующего информационного слова.
- DATA_SAVE_STATE — сохранение принятого информационного слова.
- CHECK_NUM_STATE — проверка количества принятых информационных слов.
- LOAD_OS_STATE — подготовка ответного слова.
- SEND_OS_STATE — отправка ответного слова на контроллер канала.

Теперь приведем структуру нашего конечного автомата:

```
reg [4:0] cnt_word;
reg [5:0] cnt_p;
reg [7:0] cnt;
reg [7:0] STATE;

always @ (posedge clk or posedge start or posedge reset)
begin : state_machine
  if (reset) begin
     <…>
     end
  else if (start) begin
     <…>
  end
  else case (STATE)
     IDLE_STATE:begin
        <…>
     end
     START_STATE:begin
        <…>
     end
     DATA_WAIT_STATE:begin
        <…>
     end
     DATA_SAVE_STATE:begin
        <…>
     end
     CHECK_NUM_STATE:begin
        <…>
     end
     LOAD_OS_STATE:begin
        <…>
     end
     SEND_OS_STATE:begin
        <…>
     end
  endcase
end
```

Как видим, структура автомата — наипростейшая. Помимо самих состояний, здесь присутствуют моменты аппаратного сброса и, если можно так выразиться, предстарт. В регистре STATE[7:0] содержится значение текущего состояния автомата. Количество поступивших информационных слов инкрементируется счетчиком cnt_word[4:0]. Счетчик cnt_p[5:0] будет накапливать количество ошибок паритета в принимаемом пакете. cnt[7:0] — вспомогательный счетчик. Напомним, что эту систему можно было бы оптимизировать, но цель нашей статьи привести не варианты реализации конечных автоматов, а простой пример реализации контроллера протокола для начинающих разработчиков.

Рассмотрим все по порядку.

Аппаратный сброс:

```
if (reset) begin
   STATE <= IDLE_STATE;
   tx_data <= 16'd0;
   addr_wr <= 5'd0;
   we <= 1'b0;
   clk_wr <= 1'b0;
   cnt_word <= 5'd0;
   cnt_p <= 6'd0;
   cnt <= 8'd0;
   tx_ready <= 1'b0;
   busy <= 1'b0; end
```

Поскольку reset — это сигнал сброса, то по стробу этого сигнала происходит установка состояний автомата в начальные значения. Все сигналы сбрасываются в ноль. После окончания аппаратного сброса конечный автомат перейдет в состояние ожидания IDLE_STATE.

Предстарт:

```
else if (start)begin
   STATE <= START_STATE;
   addr_wr <= 5'd0;
   we <= 1'b1;
   clk_wr <= 1'b0;
   cnt_word <= 5'd0;
   cnt_p <= 6'd0;
   cnt <= 16'd0;
   tx_ready <= 1'b0;
   busy <= 1'b1; end
```

После аппаратного сброса автомат будет ожидать сигнала start, который, как было сказано выше, будет означать приход командного слова для device.3. В этом случае поднимается индикатор занятости устройства (busy), сигнал разрешения записи в ОЗУ (we), и по завершении импульса start автомат переходит в состояние START_STATE.

Состояние IDLE_STATE:

```
IDLE_STATE:begin
   STATE <= IDLE_STATE;
      ddr_wr <= 5'd0;
      tx_ready <= 1'b0;
      tx_data <= 16'd0;
      tx_cd <= 1'b0;
      we <= 1'b0;
      busy <= 1'b0;
   end
```

В этом состоянии ожидания (или простоя) автомат находится до и после приема пакета.

Состояние START_STATE:

```
START_STATE:begin
      num_word <= rx_data[4:0];
      STATE <= DATA_WAIT_STATE;
      if (p_error) cnt_p <= cnt_p + 1'b1;
   end
```

В это состояние автомат переходит сразу после приема командного слова. В регистре num_word сохраняется количество информационных слов, которые ожидаются от контроллера канала. В случае ошибки паритета накапливаем счетчик ошибок cnt_p.

Безусловный переход в состояние ожидания информационного слова DATA_WAIT_STATE:

```
DATA_WAIT_STATE:begin
      if (rx_done) STATE <= DATA_SAVE_STATE;
      else STATE <= DATA_WAIT_STATE;
   end
```

Здесь ожидается импульс приема данных rx_done. Происходит условный переход в состояние сохранения принятых данных DATA_SAVE_STATE. Читатель может сказать, что в случае прихода командного слова также выработается импульс rx_done. Но вспомним, что в этом случае выработается импульс start, который переведет автомат в состояние обработки командного слова START_STATE.

Состояние DATA_SAVE_STATE:

```
DATA_SAVE_STATE:begin
      clk_wr <= 1'b1;
      STATE <= CHECK_NUM_STATE;
      if (p_error) cnt_p <= cnt_p + 1'b1;
   end
```

В этом состоянии выполняется тактирование ОЗУ, при этом информационное слово сохраняется во внутренней памяти ПЛИС, а также накапливается количество ошибок паритета при приеме.

Состояние CHECK_NUM_STATE:

```
CHECK_NUM_STATE:begin
   clk_wr <= 1'b0;
   addr_wr <= addr_wr + 1'b1;
   if (cnt_word == num_word_buf) begin
      cnt_word <= 5'd0;
      STATE <= LOAD_OS_STATE; end
   else begin
      cnt_word <= cnt_word + 1'b1;
      STATE <= DATA_WAIT_STATE; end
   end
```

Выше представлено состояние проверки количества принятых информационных слов. Если принято последнее слово (cnt_word == num_word_buf), то осталось передать на контроллер канала ответное слово (переход на LOAD_OS_STATE). Если же счетчик информационных слов не совпадает с указанным числом N/COM в командном слове, то ожидаем следующее информационное слово (переход на DATA_WAIT_STATE).

Состояние LOAD_OS_STATE:

```
LOAD_OS_STATE:begin
    tx_cd <= 1'b0;
    tx_data <= {ADDRESS,lcnt_p,10'd0};
    STATE <= SEND_OS_STATE;
end
```

Здесь осуществляется подготовка ответного слова. Низкий уровень сигнала tx_cd указывает передатчику Transmitter.v, что нужно передать синхросигнал, соответствующий ответному слову. В поле данных ответного слова нужно указать адрес нашего устройства (ADDRESS), а также сигналы статуса. В нашем простом проекте будем в качестве статуса считать только признак ошибки в принятом пакете. Ошибкой будет считаться, если хотя бы одно принятое слово содержало ошибку паритета (счетчик cnt_p не пустой).

Состояние SEND_OS_STATE:

```
SEND_OS_STATE:begin
  tx_ready <= 1'b1;
  if (cnt == 8'd1)begin
    STATE <= IDLE_STATE;
    cnt <= 8'd0; end
  else begin
    STATE <= SEND_OS_STATE;
    cnt <= cnt + 1'b1; end
  end
```

В этом состоянии инициируется отправка ответного слова на контроллер канала. Вспомним, что для этого необходимо выработать импульс tx_ready с длительностью, соответствующей частоте работы модуля Transmitter.v (а это два такта clk32).

Конечный автомат получился хоть и простой, но в случае сбоя (зависания) устройство не утратит работоспособности. Например, если количество принятых слов в итоге окажется меньше указанных в командном слове, автомат зависнет. Но в момент прихода следующего командного слова автомат вернется в работоспособное состояние.

Модуль device5.v

Этот подмодуль должен обеспечивать передачу данных на контроллер канала по запросу с субадресом 5. По интерфейсу ввода/вывода, структуре и функционалу device5.v во многом идентичен подмодулю device3.v. Поэтому рассматривать его полностью не имеет особого смысла, так как внимательный читатель уже сам сможет его реализовать. В помощь приведем только состояния конечного автомата этого подмодуля:

- IDLE_STATE — ожидание автомата.
- START_STATE — сохранение количества информационных слов. Переход на PAUSE_WAIT_STATE.
- PAUSE_WAIT_STATE — отсчитывание того количества тактов, которое соответствует паузе между командным и ответным словом. Переход на LOAD_OS_STATE.
- LOAD_OS_STATE — подготовка ответного слова (адрес устройства, биты статуса). Переход на READ_DATA_STATE.
- SEND_OS_STATE — инициирование отправки ответного слова на контроллер канала (формирование импульса tx_ready). Переход на READ_DATA_SEND.
- READ_DATA_SEND — чтение данных из внутренней ОЗУ. Переход на PREP_DATA_SEND.
- PREP_DATA_SEND — подготовка информационного слова для передачи на контроллер канала. Здесь происходит выставление данных на шину tx_data[15:0], а также необходимый уровень сигнала tx_cd. Переход на SEND_WAIT_DATA.
- SEND_WAIT_STATE — ожидание окончания отправки предыдущего слова на контроллер канала. Переход на CHECK_NUM_STATE.
- CHECK_NUM_STATE — проверка количества отправленных информационных слов. Если не все информационные слова переданы — переход на READ_DATA_SEND. Если же накануне было передано последнее слово — переход на END_WAIT_STATE.
- END_WAIT_STATE — окончание передачи всей информации на контроллер канала.

```
parameter IDLE_STATE = 8'd0,
          START_STATE = 8'd1,
          PAUSE_WAIT_STATE = 8'd2,
          LOAD_OS_STATE = 8'd3,
          SEND_OS_STATE = 8'd4,
          READ_DATA_STATE = 8'd5,
          PREP_DATA_STATE = 8'd6,
          SEND_WAIT_STATE = 8'd7,
          SEND_DATA_STATE = 8'd8,
          CHECK_NUM_STATE = 8'd9,
          END_WAIT_STATE = 8'd10;
```

На этом мы закончили рассмотрение HDL-кода проекта контроллера протокола MIL-STD-1553B. Следующая, заключительная часть статьи будет посвящена моделированию работы нашего контроллера протокола с использованием testbench. По мнению автора, написание тестбенчей — творческое занятие, потому как в тестбенче разрешается использовать несинтезируемые конструкции языка описания аппаратуры, а это открывает широкие возможности для моделирования. ■

Литература

1. ГОСТ Р 52070-2003. Интерфейс магистральный последовательный системы электронных модулей.
2. Дайнеко Д. Реализация CORDIC-алгоритма на ПЛИС // Компоненты и технологии. 2011. № 12.
3. IEEE Standart Verilog Hardware Description Language. 2001.
4. Mentor Graphics. ModelSim Tutorial. May, 2008.
5. Дайнеко Д. Разработка контроллера протокола MIL-STD-1553B на ПЛИС. Ч. 1 // Компоненты и технологии. 2013. № 12.
6. Дайнеко Д. Разработка контроллера протокола MIL-STD-1553B на ПЛИС. Ч. 2 // Компоненты и технологии. 2014. № 1.