FPGA-конструктор

советских микро-ЭВМ

ДВК-1

ДВК-2

ДВК-3

Электроника-60

Краткое руководство по созданию собственных периферийных устройств.

Содержание

1. Введение

Это проект разрабатывался в первую очередь как замена физически и морально устаревшего управляющего оборудования, все еще находящегося в эксплуатации на различных предприятиях. В 80-х — 90-х годах прошлого века контроллеры на основе PDP-11 — совместимого оборудования были весьма популярны и использовались в различных контрольно-измерительных приборах, системах ЧПУ, лабораторных стендах и много где еще. Сейчас это оборудование все еще частично эксплуатируется, и самым узким местом становится именно управляющий контроллер — древняя элементная база уже переработала все мыслимые сроки, и дальнейший ремонт становится весьма затруднительным.

В основе таких управляющих систем обычно находится какая-либо из плат микро-ЭВМ, чаще всего МС1201.02 или Электроника-60. К микро-ЭВМ через шину МПИ подключаются специализированные платы, стыкующие контроллер с управляемым оборудованием — аналоговыми и дискретными каналами ввода-вывода. В контроллер обычно или добавляется набор ПЗУ с управляющими программами, или программы грузятся с перфоленты или гибкого диска.

Предлагаемый, и уже опробованный мной подход предполагает не замену вышедших из строя плат в корзине МПИ, а замену всего контроллера целиком. Для этого придется разработать новую плату, уже на современных электронных компонентах, содержащую в своем составе все необходимые каналы для стыковки с оборудованием — ЦАП, АЦП, опторазвязки дискретных датчиков, реле, и все прочее. Плата эта подключается к портам FPGA, и вся обработка производится контроллером, схемотехнически реализованном уже внутри FPGA. Такой подход имеет массу достоинств. Можно выкинуть из электрошкафа громоздкую корзину МПИ и еще более громоздкий блок питания — новая схема будет компактной, и потреблять всего несколько ватт. Как следствие, контроллер будет совершенно холодным и не потребует систему охлаждающих вентиляторов, противно воющих в электрошкафе и собирающих из воздуха цеха грязь и пыль. Схема станет надежной — она построена на современной элементной базе, имеет минимум разъемных соединений.

В качестве платы FPGA можно использовать любую подходящую отладочную плату — их много продается и в Китае, и у локальных торгашей. Главное, чтобы стоящая на ней FPGA имела достаточно ресурсов для реализации схемы контроллера, и имела достаточное количество внешних портов.

Для реализации полноценного контроллера потребуется создать дополнительные модули, подключаемые к внутренней общей шине wishbone, и осуществляющие взаимодействие с оборудованием. Модули должны иметь тот же самый набор регистров и тот же протокол работы, что и оригинальные интерфейсные платы, работавшие ранее на шине МПИ в оригинальном контроллере. Этот документ дает краткое знакомство с процессом добавления модулей к проекту.

2. Постановка задачи

Процесс создания модуля будем разбирать на конкретном примере. Пример будет сильно упрощенным и избыточным по количеству регистров — он не предназначен для использования на практике, и нужен только как пример.

Предположим, что нам нужно сделать устройство двухканального дискретного ввода-вывода. Снаружи к модулю подключаются два исполнительных устройства, например два реле, и два дискретных датчика, например концевых выключателя. Назовем наш модуль просто и незатейливо — **dio**.

Модуль имеет на общей шине 2 регистра — регистр управления CSR и регистр данных DR. Биты регистра данных отражают состояние дискретных датчиков, а также управляют состоянием исполнительных устройств.

В случае изменения состояния сигнала на любом из входов модуль должен вызывать векторное прерывание.

Присвоим регистру управления адрес 175300, регистру данных адрес 175302. Прерывание будет использовать вектор 310.

Формат регистра CSR:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
									IE	**IRQ**					
									R/W	R/W					

Бит 6 (**IE**) — разрешение прерывания. Этот бит есть в регистре CSR всех стандарных устройств.

Бит 5 (**IRQ**) — признак запроса прерывания. Если наступило событие, приводящее к прерыванию, но самого прерывания еще не произошло, это флаг устанавливается в 1. Запись 1 в этот разряд приводит к сбросу ожидающего прерывания.

Конечно, в данном случае регистр упрощен до предела и вообще не особо нужен. В более сложных устройствах обычно есть бит 15 (BSY) — признак занятости модуля, бит 0 (GO) — триггер запуска команды на выполнение, и бит 7 (DRQ) — признак готовности данных для чтения-записи процессором.

Формат регистра DR:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
												O2	**O1**	**I2**	**I1**
												R/W	R/W	R	R

Биты 3 и 2 (**O2** и **O1**) — управление каждым из каналов дискретного вывода. Записанная в этот регистр 0 или 1 переключает уровень сигнала на выходе, например включает-выключает реле.

Биты 1 и 0 (**I2** и **I1**) отражают состояние соответствующих дискретных входов. Запись в эти разряды игнорируется.

3. Внешние порты модуля

Для стыковки с шиной wishbone модуль должен иметь стандартный набор сигналов этой шины. Кроме того, в список внешних портов включаются 2 линии ввода и 2 линии вывода дискретных данных. Вот заголовок нашего модуля:

```
module dio (
   input                  wb_clk_i,   // тактовая частота шины
   input                  wb_rst_i,   // сброс
   input    [1:0]         wb_adr_i,   // адрес
   input    [15:0]        wb_dat_i,   // входные данные
   output reg [15:0]      wb_dat_o,   // выходные данные
   input                  wb_cyc_i,   // начало цикла шины
   input                  wb_we_i,    // разрешение записи (0 - чтение)
   input                  wb_stb_i,   // строб цикла шины
   input      [1:0]       wb_sel_i,   // выбор байтов для записи
   output reg             wb_ack_o,   // подтверждение выбора устройства

// обработка прерывания
   output reg             irq,        // запрос
   input                  iack,       // подтверждение

// интерфейс ввода-вывода дискретных сигналов
   output reg             do1,        // выходная линия 1
   output reg             do2,        // выходная линия 2
   input                  di1,        // входной сигнал 1
   input                  di2         // входной сигнал 2
 );
```

Из всей адресной шины нам требуются только 2 бита, поскольку в адресном пространстве модуль представлен двумя регистрами (диапазон адресов 175300-175303). Назначение всех сигналов шины и ее протокол можно найти в описании шины wishbone.

4. Фильтрация входных сигналов

Поскольку входные линии изменяют свое состояние асинхронно с тактовым сигналом шины, требуется обязательная фильтрация входных сигналов для предотвращения состояния метастабильности. Сделать это можно разными способами, простейший — цепочка из двух триггеров, образующая сдвиговый регистр. На выходе регистра получается уже отфильтрованный сигнал, пригодный для дальнейшей синхронной обработки. Вот пример схемы такого фильтра:

```
always @ (posedge wb_clk_i) begin
     // вводим каждый сигнал в сдвиговый регистр
     di1_filter[0] <= di1;
     di1_filter[1] <= di1_filter[0];

     di2_filter[0] <= di2;
     di2_filter[1] <= di2_filter[0];
end

// выходы сдвиговых регистров - это отфильтрованные сигналы
assign di1f=di1_filter[1];
assign di2f=di2_filter[1];
```

В реальной жизни, кроме фильтрации состояния метастабильности, скорее всего потребуется добавить в схему подавление дребезга, если на вход канала поступает сигнал от датчика с механическими контактами, например кнопки или концевого выключателя. Но не будем зря усложнять тестовую схему.

5. Формирование ответа на шинную транзакцию

Каждое периферийное устройство, находящееся на общей шине, в ответ на начало транзакции **wb_stb** должно сформировать ответ **wb_ack**, подтверждающий выполнение транзакции со стороны устройства. Вот пример схемы формирователя ответа:

```
wire reply=wb_cyc_i & wb_stb_i & ~wb_ack_o; // сигнал ответа на шинную транзакцию

always @(posedge wb_clk_i or posedge wb_rst_i)
  if (wb_rst_i == 1'b1) wb_ack_o <= 1'b0;   // сбрасываем сигнал подтверждения по rst
  else wb_ack_o <= reply;                   // выводим сигнал ответа на шину
```

6. Блок синхронной обработки и начальный сброс

Всю обработку мы будем делать в едином always-блоке, синхронизированном с тактовым сигналом шины wishbone. В более сложных проектах часть логики, конечно, имеет смысл вынести в отдельные блоки, но в данном случае этого не требуется.

Первое, что обязательно надо сделать — это сбросить все внутренние флаги модуля, а выходные линии **do1** и **do2** установить в начальное состояние, например в 0:

```
always @(posedge wb_clk_i)   begin
    if (wb_rst_i == 1'b1) begin
        //******************
       // сброс системы
        //******************
       interrupt_state <= i_idle; // состояние ожэидания прерывания
       irq <= 1'b0 ;              // снимаем запрос на прерывания
       ie <= 1'b0;                // запрещаем прерывания
       interrupt_trigger <= 1'b1; // сбрасываем триггер запроса прерывания
       di1old <= 1'b0;            // сбрасываем предыдущие значения
       di2old <= 1'b0;            //    входных сигналов
       do1 <= 1'b0;               // устанавливаем в начальное значение
       do2 <= 1'b0;               // выходные сигналы
    end
```

7. Выделение сигналов управления шиной

Нас интересуют 2 сигнала — запрос чтения и запрос записи. Выделяем их из сигналов wishbone:

```
wire bus_strobe = wb_cyc_i & wb_stb_i;          // строб цикла шины
wire bus_read_req = bus_strobe & ~wb_we_i;      // запрос чтения
wire bus_write_req = bus_strobe & wb_we_i;      // запрос записи
```

8. Обработка запросов чтения регистров

При поступлении строба **bus_read_req** следует выставить на выходную шину данных модуля содержимое регистров CSR или DR, в зависимости от состояния лини адреса **wb_adr_i[1]**:

```
if (bus_read_req == 1'b1)   begin
         case (wb_adr_i[1])
           1'b0 : // 175300 - CSR
                    wb_dat_o <= {9'o0, ie, interrupt_trigger,5'o0};
           1'b1 : // 175302 - DR
                    wb_dat_o <= {12'o0, do2, do1, di2f, di1f};
         endcase
end
```

В данном случае мы выставляем на шину только значащие биты. Остальные биты заменяем нулями.

9. Обработка запросов записи регистров.

При поступлении сигнала **bus_write_req** с входной шины данных следует прочитать содержимое значащих бит и записать их в соответствующие регистры:

```
 // запись регистров
 else if (bus_write_req == 1'b1)  begin
   if (wb_sel_i[0] == 1'b1)
   // запись младших байтов
     case (wb_adr_i[1])
     // 175300 - CSR
     1'b0:  begin
         ie <= wb_dat_i[6];   // флаг разрешения прерывания
         if (wb_dat_i[5] == 1'b1) interrupt_trigger <= 1'b0; // сброс запроса прерывания
       end

     // 175302 - DR
     1'b1 : begin
         // запись регистров выходных линий
        do1 <= wb_dat_i[2];
        do2 <= wb_dat_i[3];
       end
     endcase
end
```

В данном случае анализируется только запись младших байтов регистров, поскольку разряды старших байтов не содержат полезной информации.
В более сложных случаях в этом блоке производится анализ некоторых разрядов и запуск обработки команд.

10. Обнаружение условий для прерывания.

По условиям задачи, любое изменение состояния входных линий должно вызывать запрос на векторное прерывание. Для этого в схеме предусмотрен регистр **interrupt_trigger**. Он взводится, когда изменяется любая из входных линий, и сбрасывается после обработки прерывания, или записи 1 в разряд **IRQ** регистра CSR.
Вот схема обнаружения перепадов входных сигналов:

```
reg di1old,di2old;

if ((di1old != di1f) || (di2old != di2f)) interrupt_trigger <= 1'b1;
di1old <= di1f;
di2old <= di2f;
```

Предыдущие состояния входных сигналов сохраняются в регистрах **di1old** и **di2old** и сравниваются с текущим состоянием этих сигналов.

11. Обработка запросов прерывания

Эта схема нужна для установки выходного сигнала **IRQ**, поступающего к контроллеру прерываний, ожидания ответного сигнала **IACK** от этого контроллера, и последующего

снятия запроса **IRQ**. Всю остальную работу, в том числе и формирование правильного адреса вектора, берет на себя контроллер прерываний.
Схему формирователя запроса прерывания можно создавать разными способами, я предпочитаю это делать через машину состояний:

```
parameter[1:0] i_idle = 0;    // ожидание прерывания
parameter[1:0] i_req = 1;     // запрос векторного прерывания
parameter[1:0] i_wait = 2;    // ожидание обработки прерывания со стороны процессора

reg[1:0] interrupt_state;

case (interrupt_state)
   // нет активного прерывания
   i_idle :  begin
       //  Если поднят флаг - переходим в состояние активного прерывания
       if ((ie == 1'b1) & (interrupt_trigger == 1'b1))  begin
               interrupt_state <= i_req ;
               irq <= 1'b1 ;    // запрос на прерывание
        end
            // иначе снимаем запрос на прерывание
        else   irq <= 1'b0 ;
   end
   // Формирование запроса на прерывание
   i_req :
        if (ie == 1'b0)    interrupt_state <= i_idle;     // прерывания запрещены
        else if (iack == 1'b1) begin
           // если получено подтверждение прерывания от процессора
           irq <= 1'b0 ;               // снимаем запрос
           interrupt_trigger <= 1'b0;  // очищаем триггер прерывания
           interrupt_state <= i_wait ; // переходим к ожиданию окончания обработки
        end
   // Ожидание окончания обработки прерывания
   i_wait : if (iack == 1'b0)  interrupt_state <= i_idle ; // ждем снятия сигнала iack
endcase
```

12. Подключение модуля к шине wishbone

Наш модуль готов. Теперь его надо подключить к общей шине — соединительному модулю topboard. Для этого в topboard надо описать несколько дополнительных сигналов:

```
wire dio_stb;          // строб выбора устройства
wire dio_ack;          // сигнал подтверждения транзакции от устройства
wire [15:0] dio_dat;   // локальная выходная шина данных модуля
```

Далее создаем формирователь сигнала выбора устройства по адресам 175300-175303:

```
assign dio_stb = wb_stb & wb_cyc & (wb_adr[15:2] == (16'o175300 >> 2));
```

Подключаем сигнал подтверждения в сборную шину этих сигналов:

```
assign global_ack  = ram_ack | rom_ack | …. | dio_ack;
```

Добавляем локальную выходную шину данных в мультиплексор шин данных:

```
assign wb_mux =
       (ram_stb   ? ram_dat   : 16'o000000)
     | (rom_stb   ? rom_dat   : 16'o000000)
            ……..
     | (dio_stb   ? dio_dat   : 16'o000000)
```

Подключаем модуль к контроллеру прерываний, не забыв увеличить на 1 параметр N:

```
wbc_vic #(.N(10)) vic
    ……….....
(//      UART1-Tx     UART1-Rx  ...     DIO
.ivec({16'o000064, 16'o000060,  ...   16'o000310 }), // векторы
.ireq({irpstx_irq, irpsrx_irq,  ...   dio_irq  }),   // запрос прерывания
.iack({irpstx_iack,irpsrx_iack, ...   dio_iack })    // подтверждение прерывания
```

Вот и вся подготовка. Теперь можно подключать наш модуль к шине:

```
//*********************************
//* Тестовый модуль
//*********************************
dio dig1 (
// шина wishbone
   .wb_clk_i(wb_clk),      // тактовая частота шины
   .wb_rst_i(sys_init),    // сброс
   .wb_adr_i(wb_adr[1:0]), // адрес
   .wb_dat_i(wb_out),      // входные данные
   .wb_dat_o(dio_dat),      // выходные данные
   .wb_cyc_i(wb_cyc),      // начало цикла шины
   .wb_we_i(wb_we),        // разрешение записи (0 - чтение)
   .wb_stb_i(dio_stb),      // строб цикла шины
    .wb_sel_i(wb_sel),      // выбор конкретных байтов для записи - старший, младший или
оба
   .wb_ack_o(dio_ack),      // подтверждение выбора устройства

// обработка прерывания
   .irq(dio_irq),           // запрос
   .iack(dio_iack),         // подтверждение

// линии ввода-вывода
   .di1(di1),
   .di2(di2),
   .do1(do1),
   .do2(do2)
);
```

Сигналы **di1**, **di2**, **do1**, **do2** выводятся на внешние порты модуля topboard, им назначаются свободные ножки FPGA и подключаются к управляемому оборудованию. На этом создание простейшего модуля закончено.

13. Заключение.

Полный исходный текст модуля dio лежит в файле doc/dio.v. Можно подключить его к проекту и поиграться с ним, повесив на внешние порты пару кнопок и светодиодов. Сам по себе этот модуль никакой практической ценности не представляет, он служит иллюстрацией процесса разработки собственных периферийных устройств. Через пару лет я и сам забуду все тонкости этого процесса, и вот тут пригодится данный набор заметок.
Здесь не рассматривается создание модулей с собственными DMA-контроллерами. DMA — это отдельная и не очень приятная тема. Как правило, в управляющих системах этот способ работы с шиной никто не использует, разве только высокоскоростные АЦП и ЦАП могут работать через DMA. В основном DMA применялось в дисковых контроллерах.

Также хотелось бы обратить внимание на необходимость создания опторазвязки со всеми входящими дискретными сигналами. И, кроме того, требуется тщательная фильтрация источника питания платы FPGA и хорошая экранировка контроллера. В оборудовании достаточно мощных импульсных помех, и переключение какого-нибудь силового контактора запросто может привести к зависанию или перезагрузке контроллера. На эту тему написано

много полезной литературы, и я крайне не рекомендую все это игнорировать — уж больно нежной стала современная электроника.

Используя данные заметки, можно создавать не только модули управления технологическим оборудованием, но и стандартную периферию PDP-11. Можно, например, сделать ленточный перфоратор и перфосчитыватель с хранением данных на SD-карте, можно реализовать контроллер магнитной ленты, можно очень много чего сделать. Если в будущем наступит счастливый момент, и ширина шины адреса будет увеличена с 16 до 22 бит, все эти модули (кроме использующих DMA) даже переделывать не придется. Конструктор есть конструктор — бесконечный простор для творчества.