Лабораторная работа №6 Контроллер PS/2 для клавиатуры

Для успешного применения цифровых устройств в реальности важно обеспечить их взаимодействие с внешним миром. Для этого существуют внешние интерфейсы для обмена информацией с другими устройствами и приборами. Внешние интерфейсы имеют различное функциональное назначение и спецификацию. Например, раньше для подключения клавиатуры и мыши использовался интерфейс PS/2. Один с самых простых способов подключения устройств ввода-вывода.

Давайте изучим протокол и реализацию интерфейса PS/2.

Интерфейс PS/2 сточки зрения соединения представляет собой два провода **Clock** и **Data**. По **Clock** передаются синхроимпульсы, а по **Data** предаются данные. На рисунке пример одной транзакции обмена.

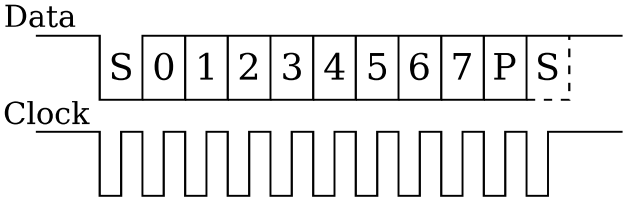

Рис. 6.1: Передача данных по протоколу PS/2

Структура транзакции похожа на UART и состоит из:

1. старт бит – всегда ноль;
2. 8 бит данных;
3. бит четности, равен 1 если количество единиц в данных четно и 0 если нечетно;

4. стоп бит – всегда единица.

Данные на линию выставляются, когда **Clock** равен **1** и считываются, когда **Clock** равен **0**. На практике данные обычно выставляются по положительному фронту и считываются по отрицательному.

Частота сигнала **Clock** примерно 10-16.7кГц. Время от фронта сигнала **Clock** до момента изменения сигнала **Data** не менее 5 микросекунд.

Транзакции разделяются на два вида:

1. От устройства к контроллеру;
2. От контроллера к устройству.

На примере клавиатуры.

1. Клавиатура посылает на контроллер 8 битный код нажатой клавиши;
2. Контроллер посылает на клавиатуру команды управления. Такие как, команды сброса, выключения светодиодов.

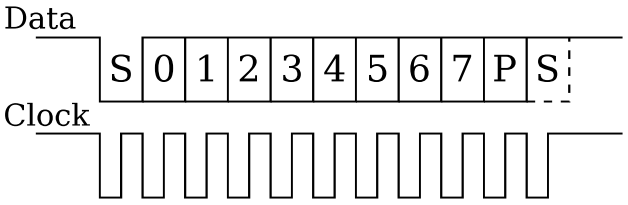

Рис. 6.2: Передача 8 битного пакета данных

При транзакции от устройства (клавиатуры) к контроллеру сигналы на линиях **Clock** и **Data** генерирует устройство. Контроллер выступает в роли приемника считывая данные по отрицательному фронту **Clock**.

При передаче в обратную сторону команд от контроллера к клавиатуре или мыши протокол отличается от описанного выше.

Последовательность обмена другая:

1. контроллер опускает сигнал **Clock** в ноль на время примерно 100 микросекунд;
2. контроллер опускает сигнал **Data** в ноль формируя старт

бит;
3. контроллер отпускает сигнал **Clock** в логическую единицу, клавиатура фиксирует старт бит;
4. далее клавиатура генерирует сигнал **Clock**, а хост контроллер подает передаваемые биты;
5. после того, как контроллер передал все свои биты, включая бит четности и стоп бит, клавиатура посылает последний бит «ноль», который является подтверждением приема.

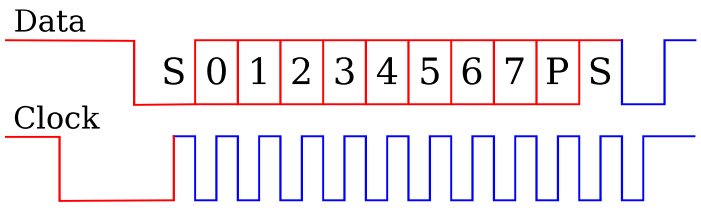

Рис. 6.3: Распределение управления между контроллером и устройством

Поскольку одним сигналом управляют два устройства, то довольно трудно понять, кто в какой момент времени управляет сигналом. По этому, диаграмма нарисована двумя цветами. Красный цвет – сигнал управляется контроллером, а синий – сигнал управляется устройством.

Давайте выясним какие коды же коды передает клавиатура для каждой клавиши.

Ниже приведена таблица кодов для клавиш. Каждой клавише соответствует код генерируемый при нажатии и код генерируемый при деактивации. Коды состоящие из нескольких байтов предаются в нескольких подряд идущих транзакций.

KEY	**MAKE**	**BREAK**	**KEY**	**MAKE**	**BREAK**
A	1C	F0,1C	R ALT	E0,11	E0,F0,11
B	32	F0,32	APPS	E0,2F	E0,F0,2F
C	21	F0,21	ENTER	5A	F0,5A
D	23	F0,23	ESC	76	F0,76
E	24	F0,24	F1	05	F0,05
F	2B	F0,2B	F2	06	F0,06

KEY	MAKE	BREAK	KEY	MAKE	BREAK
G	34	F0,34	F3	04	F0,04
H	33	F0,33	F4	0C	F0,0C
I	43	F0,43	F5	03	F0,03
J	3B	F0,3B	F6	0B	F0,0B
K	42	F0,42	F7	83	F0,83
L	4B	F0,4B	F8	0A	F0,0A
M	3A	F0,3A	F9	01	F0,01
N	31	F0,31	F10	09	F0,09
O	44	F0,44	F11	78	F0,78
P	4D	F0,4D	F12	07	F0,07
Q	15	F0,15	SCROLL	7E	F0,7E
R	2D	F0,2D	[	54	FO,54
S	1B	F0,1B	INSERT	E0,70	E0,F0,70
T	2C	F0,2C	HOME	E0,6C	E0,F0,6C
U	3C	F0,3C	PG UP	E0,7D	E0,F0,7D
V	2A	F0,2A	DELETE	E0,71	E0,F0,71
W	1D	F0,1D	END	E0,69	E0,F0,69
X	22	F0,22	PG DN	E0,7A	E0,F0,7A
Y	35	F0,35	UP	E0,75	E0,F0,75
Z	1A	F0,1A	LEFT	E0,6B	E0,F0,6B
0	45	F0,45	DOWN	E0,72	E0,F0,72
1	16	F0,16	RIGHT	E0,74	E0,F0,74
2	1E	F0,1E	NUM	77	F0,77
3	26	F0,26	KP /	E0,4A	E0,F0,4A
4	25	F0,25	KP *	7C	F0,7C
5	2E	F0,2E	KP -	7B	F0,7B
6	36	F0,36	KP +	79	F0,79
7	3D	F0,3D	KP EN	E0,5A	E0,F0,5A
8	3E	F0,3E	KP .	71	F0,71
9	46	F0,46	KP 0	70	F0,70
‘	0E	F0,0E	KP 1	69	F0,69
-	4E	F0,4E	KP 2	72	F0,72
=	55	F0,55	KP 3	7A	F0,7A
	5D	F0,5D	KP 4	6B	F0,6B
BKSP	66	F0,66	KP 5	73	F0,73
SPACE	29	F0,29	KP 6	74	F0,74

KEY	MAKE	BREAK	KEY	MAKE	BREAK
TAB	0D	F0,0D	KP 7	6C	F0,6C
CAPS	58	F0,58	KP 8	75	F0,75
L SHFT	12	FO,12	KP 9	7D	F0,7D
L CTRL	14	FO,14	]	5B	F0,5B
L GUI	E0,1F	E0,F0,1F	;	4C	F0,4C
L ALT	11	F0,11	’	52	F0,52
R SHFT	59	F0,59	,	41	F0,41
R CTRL	E0,14	E0,F0,14	.	49	F0,49
R GUI	E0,27	E0,F0,27	/	4A	F0,4A

Разработаем простой контроллер для клавиатуры. Контроллер предназначается для работы только в режиме устройство-контроллер.

Для начала определим функционал контроллера и набор сигналов.

Функционал:

1. Считываются данные с линии PS/2 по отрицательному фронту синхросигнала;
2. Проверяется корректность принятых данных. Проверяется стоп бит, бит четности и стартовый бит;
3. Выводит принятые данные на внешнюю шину и формирует сигнал готовности данных.

Набор сигналов:

1. **areset** – асинхронный ресет, активный уровень **1**;
2. **Data –** 8 битная шина данных;
3. **valid_Data** – сигнал готовности данных, равен **1** с момента завершения приема по PS/2 до начала следующей транзакции;
4. **ps2_clk** – тактовая линия PS/2, является внешним сигналом для ПЛИС;
5. **ps2_dat** – сигнальная линия PS/2, является внешним сигналом для ПЛИС.

Примечание. Сигналы **ps2_clk** и **ps2_dat** как внешние сигналы необходимо подключить к соответствующим пинам ПЛИС. Это пины H15(**ps2_clk**) J14(**ps2_dat**), к которым на плате DE1 подключен коннектор PS/2.

```
1  module ps2_keyboard(
2                  input             areset,
3                  input             ps2_clk,
4                  input             ps2_dat,
5                  output  reg       valid_data,
6                  output     [7:0]  data);
```

Листинг 6.1: Описание входов и выходов контроллера

Основой контроллера будет конечный автомат со следующей последовательностью действий:

1. Состояние покоя;
2. Прием стартового бита и его проверка. Если стартовый бит не равен 0 переходим в состояние покоя;
3. Прием данных;
4. Прием бита четности и стопового бита их проверка.
5. При правильных значениях стопового бита и бита четности формирования сигнала готовности данных.
6. Переход в состояние покоя.

```
1  reg [1:0] state;
2
3  localparam IDLE                  = 2'd0;
4  localparam RECEIVE_DATA          = 2'd1;
5  localparam CHECK_PARITY_STOP_BITS = 2'd2;
6
7
8    always @(negedge ps2_clk or posedge areset)
9    if(areset)
10          state <= IDLE;
11   else
12   begin
13          case (state)
14          IDLE:
15                 begin
16                     if(!ps2_dat)
17                            state = RECEIVE_DATA;
18                 end
19          RECEIVE_DATA:
20                 begin
```

```
                        if (count_bit == 8)
                          state =
                           CHECK_PARITY_STOP_BITS;
                end
         CHECK_PARITY_STOP_BITS:
                begin
                   state = IDLE;
                end
        default:
                begin
                  state = IDLE;
                end
        endcase
end
```

Листинг 6.2: Описание конечного автомата контроллера

Конечный автомат имеет три состояния IDLE, RECEIVE_DATA, CHECK_PARITY_STOP_BIT.

IDLE – состояние покоя в котором контроллер ожидает первого отрицательного фронта **ps2_clk**. Переход в состояние RECEIVE_DATA происходит по отрицательному фронту **ps2_clk** если **ps2_dat** равно 0, то есть пришел стартовый бит, иначе остаемся в IDLE.

RECEIVE_DATA – состояние в ходе, которого происходит прием данных и бита четности. Переход в состояние CHECK_PARITY_STOP_BIT происходит при счетчике бит равном 8. Отсчитано 8 бит данных и идет прием бита четности.

Последнее состояние CHECK_PARITY_STOP_BITS длительностью в один период **ps2_clk**. В CHECK_PARITY_STOP_BITS происходит проверка бита паритета и стопового бита.

```
reg  [8:0]  shift_reg;

  assign data = shift_reg[7:0];

  always @(negedge ps2_clk or posedge areset) begin
    if(areset)
            shift_reg <= 9'b0;
          else
```

```
 9                 if(state == RECEIVE_DATA)
10                 shift_reg <=
11                      {ps2_dat, shift_reg[8:1]};
12   end
```

Листинг 6.3: Описание сдвигового регистра

Сдвиговый регистр необходим для приема и хранения данных и бита четности. По этому, разрядность регистра равна 9. По завершению транзакции в восьмом бите хранится бит четности с 7-0 бит данные. Данные непрерывным присваиванием выведены на внешнюю шину модуля.

Если обратиться к началу лабораторной работы то стоит заметить что данные передаются по интерфейсу PS/2 начиная с младшего бита. Логично будет использовать схему работы сдвигового регистра, при которой сдвиг происходит вправо. Таким образом, первый принятый бит окажется в 0 ячейке сдвигового регистра по окончании транзакции.

Запись в сдвиговый регистр происходит по отрицательному фронту **ps2_clk** и состоянию конечного автомата RECEIVE_DATA.

```
 1  reg  [3:0]      count_bit;
 2
 3    always @(negedge ps2_clk or posedge areset) begin
 4      if(areset)
 5             count_bit <= 4'b0;
 6           else
 7                  if(state == RECEIVE_DATA)
 8                  count_bit <= count_bit + 4'b1;
 9                  else
10                  count_bit <= 4'b0;
11    end
```

Листинг 6.4: Описание счетчика принятых бит

Счетчик принятых бит служит для контроля за процессом приема. Инкрементация счетчика происходит только в состоянии RECEIVE_DATA.

```
 1  function parity_calc;
```

```
2   input [7:0] a;
3     parity_calc = ~(a[0] ^ a[1] ^ a[2] ^ a[3] ^
4                     a[4] ^ a[5] ^ a[6] ^ a[7]);
5   endfunction
6
7   always @(negedge ps2_clk or posedge areset) begin
8     if(areset)
9           valid_data <= 1'b0;
10        else
11              if (ps2_dat &&
12                  parity_calc(shift_reg[7:0]) ==
13                      shift_reg[8] &&
14                  state == CHECK_PARITY_STOP_BITS)
15              valid_data <= 1'b1;
16              else
17                valid_data <= 1'b0;
18  end
```

Листинг 6.5: Описание вырабатывания сигнала готовности к передаче

Последним нерассмотренным моментом остался вопрос генерации сигнала готовности данных. Как было сказано ранее сигнал готовности генерируется к конце транзакции в случае успешного приема и равен 1 до начала следующей транзакции. То есть пока конечный автомат в состоянии IDLE.

Условием успешного окончания транзакции является стоповый бит равный 1 и бит четности равный рассчитанному значению.

Генерация сигнала готовности происходит в момент приема стопового бита. По этому для его проверки достаточно убедиться что значение на линии **ps2_dat** равно 1.

Для проверки бита четности необходимо рассчитать четность принятых 8 бит данных и сравнить ее с значением бита четности. Для упрощения читаемости кода используется функция расчета честности для 8 разрядного регистра согласно правилу отрицания побитового XOR регистра. Правило расчета бита паритета можно узнать из стандарта на интерфейс PS/2.

6.1 Задание лабораторной работы:

Ознакомиться с спецификацией на интерфейс PS/2 и представленной реализацией контроллера клавиатуры.

Построить временные диаграммы его работы с помощью САПР Altera Quartus.

Подготовиться к выполнению индивидуального задания с использованием предложенного котроллера на лабораторной работе.

6.2 Пример индивидуального задания

Выводить на семи сегментные индикаторы только коды клавиш с цифрами.

```
module ps2_keyboard(
                input            areset,
                input            ps2_clk,
                input            ps2_dat,
                output  reg      valid_data,
                output     [7:0} data);


reg  [3:0]  count_bit;
reg  [8:0]  shift_reg;

  assign data = shift_reg[7:0];

  function parity_calc;
  input [7:0] a;
    parity_calc = ~(a[0] ^ a[1] ^ a[2] ^ a[3] ^
                    a[4] ^ a[5] ^ a[6] ^ a[7]);
  endfunction

reg [1:0] state;

localparam IDLE                  = 2'd0;
localparam RECEIVE_DATA          = 2'd1;
```

```
localparam CHECK_PARITY_STOP_BITS = 2'd2;


  always @(negedge ps2_clk or posedge areset)
  if(areset)
         state <= IDLE;
  else
  begin
         case (state)
         IDLE:
                begin
                    if(!ps2_dat)
                           state = RECEIVE_DATA;
                end
         RECEIVE_DATA:
                begin
                        if (count_bit == 8)
                          state =
                            CHECK_PARITY_STOP_BITS;
                end
         CHECK_PARITY_STOP_BITS:
                begin
                   state = IDLE;
                end
        default:
                begin
                  state = IDLE;
                end
        endcase
end

  always @(negedge ps2_clk or posedge areset) begin
    if(areset)
           valid_data <= 1'b0;
         else
                if (ps2_dat &&
                    parity_calc(shift_reg[7:0]) ==
                        shift_reg[8] &&
                    state == CHECK_PARITY_STOP_BITS)
                valid_data <= 1'b1;
```

```
                else
                  valid_data <= 1'b0;
  end

  always @(negedge ps2_clk or posedge areset) begin
    if(areset)
            shift_reg <= 9'b0;
          else
                if(state == RECEIVE_DATA)
                shift_reg <=
                        {ps2_dat, shift_reg[8:1]};
  end

  always @(negedge ps2_clk or posedge areset) begin
    if(areset)
            count_bit <= 4'b0;
          else
                if(state == RECEIVE_DATA)
                count_bit <= count_bit + 4'b1;
                else
                count_bit <= 4'b0;
  end

endmodule
```

Листинг 6.6: Пример реализации контроллера PS/2