7

Периферийные устройства

Целью лабораторной работы является **разработка периферийных устройств** и подключение их к ранее реализованной процессорной системы

- Подключение конфигурационных регистров
- Карта памяти
- Светодиоды и семисегментные дисплеи
- Кнопки и переключатели
- Контроллер клавиатуры

Подключение конфигурационных регистров

Помимо процессора и памяти, третьим ключевым элементом вычислительной системы является система ввода\вывода, обеспечивающая обмен информации между ядром вычислительной машины и периферийными устройствами.

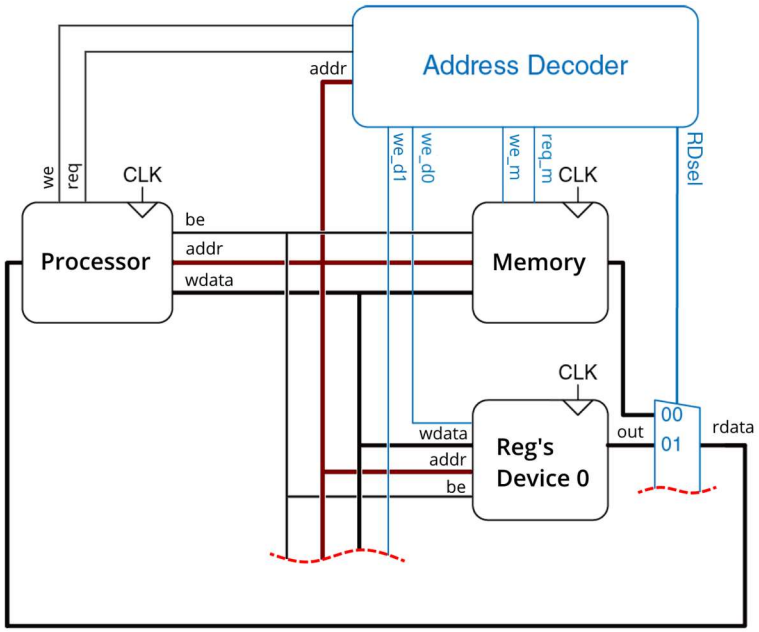
Любое периферийное устройство, со стороны вычислительной машины, представляется набором ячеек памяти (регистров). С помощью чтения и записи этих регистров происходит обмен информации с периферийным устройством, и управление им. Например, датчик температуры может быть реализован самыми разными способами, но для процессора он в любом случае ячейка памяти, из которой он считывает число – температуру.

Система ввода\вывода может быть организована одним из двух способов: с **выделенным адресным пространством** устройств ввода\вывода, или **с совместным адресным пространством**. В первом случае система ввода\вывода имеет отдельную шину для подключения к процессору, во втором – шина для памяти и системы ввода\вывода общая.

Архитектура RISC-V подразумевает использование совместного адресного пространства, это значит, что в лабораторной работе будет использована единая шина для подключения памяти и регистров управления периферийными устройствами. При обращении по одному диапазону адресов процессор будет попадать в память, при обращении по другим – взаимодействовать с регистрами управления\статуса периферийного устройства.

В следующем параграфе будет представлена карта памяти, с указанием конкретных адресов для регистров управления периферийных устройств. Пока что договоримся, что память инструкций и данных занимает 256 слов и располагается по адресам **0x000003FC** – **0x00000000**.

Ниже демонстрируется фрагмент подключения периферийных устройств к общей шине, ранее соединявшей только память и процессор. Обратите внимание, что это только часть системы, тут не изображено подключение к портам памяти для чтения инструкций, и подсистема прерывания. Два сигнала шины **req** (запрос на обращение к памяти) и **we** (сигнал записи) теперь подключаются не напрямую, а через дешифратор адреса (**Address Decoder**), который перенаправляет эти сигналы в блок, соответствующий своему диапазону адресов. Для регистров управления сигнал запроса **req** не требуется. Каждый из подключенных управляющих регистров использует столько бит адреса **addr**, сколько ему требуется. То же самое касается сигналов **be** – они используются по необходимости, и для каких-то устройств могут частично или полностью отсутствовать. В задачу дешифратора адреса также входит управление мультиплексором, объединяющем выходы всех подключенных устройств в шину **rdata**. Разрядность сигнала **RDsel** определяется количеством подключенных устройств.



Чтобы разобраться с разрядностями **addr** и **RDsel** рассмотрим пример. К системе подключено 4 периферийных устройства. *Первое* из них имеет один 8-битный управляющий регистр по адресу **0x6000D350**. *Второе* устройство располагается по адресу **0x6000C120** и имеет один 32-битный регистр управления. *Третье* периферийное устройство имеет два 8-битных регистра по адресам **0x6000B008** и **0x6000B00C**. И наконец, *четвертое* использует семь 32-битных регистров начиная с адреса **0x6000FFF0**.

Для первого устройства вообще не надо подводить линии адреса **addr** и выбора байта **be**, так не надо уточнять к какому из регистров устройства идет обращение – он у него один, и тот 8-битный, то есть при указании конкретного байтового адреса **Address Decoder** поймет, что обращение будет именно к

нему и никак иначе. Он либо сформирует **we_d0** для регистра, если это операция записи, либо переключит мультиплексор в нужное положение, если это операция чтения. Фрагмент **Verilog**-кода реализующего формирование сигнала записи для такого периферийного устройства будет выглядеть так:

```
assign we_d0 = req & we & (addr == 32'h6000D350);
```

Для сигнала записи первого периферийного устройства дешифратор адреса использует весь 32-битный адрес для дешифрации.

Второе периферийное устройство имеет всего один 32-битный регистр управления, поэтому к нему также не нужно подводить линии адреса **addr**, достаточно подключить только линии **be**, для выбора конкретных байт для обращения.

```
assign we_d1 = req & we & ({addr[31:2], 2'b0} == 32'h6000C120);
```

Оператор объединения { } используется только для того, чтобы в правой части не пришлось писать 30-битное число, которое из-за смещения на 2, а не на 4 бита будет иметь иное 16-ричное представление **30'h18003048**.

Третье периферийное устройство использует 2 слова, поэтому, для выбора одного из них достаточно использовать (подать на вход устройства) второй бит адреса **addr[2]**. При этом используются только младшие байты слов, поэтому для их активации достаточно подать только младший из сигналов выбора байта слова **be[0]**.

Так как устройство покрывает 8 байтовых адресов (2 слова), то для дешифрации необходимо игнорировать 3 младших бита адреса:

```
assign we_d2 = req & we & ({addr[31:3], 3'b0} == 32'h6000B008);
```

Если примеры с 16-ричными числами выше сходу не понятны, то попробуйте переводить адреса в двоичную форму.

Четвертое устройство использует 7 32-битных регистров управления. Для адресации регистров внутри него потребуется задействовать 3 бита адреса **addr[4:2]** и все биты выбора байта **be**.

В дешифраторе адреса будет откинуто уже 5 бит адреса.

```
assign we_d3 = req & we & ({addr[31:5], 5'b0} == 32'h6000FFE0);
```

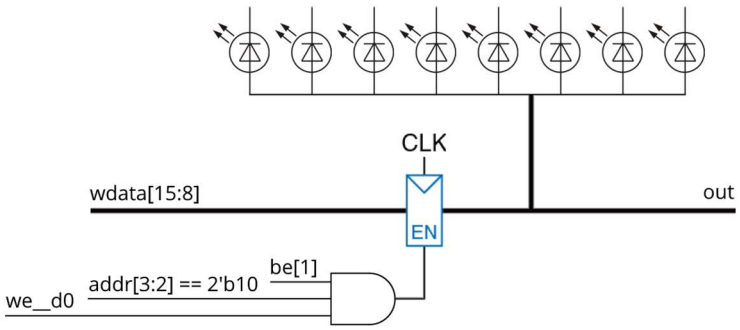
Что же касается сигнала управления мультиплексором **RDsel**, его разрядность для данного примера будет 3, потому что количество подключенных устройств 5 – память и четыре периферийных. Однако, при описании работы мультиплексора можно не задаваться вопросом разрядности **RDsel**, так как устройство можно описать через конструкцию **case** в качестве параметра используя какое-то количество старших бит адреса **addr**, достаточное, чтобы однозначно определить источник данных.

```
always @ (*)
   case (addr[31:n])
      ...
      default: ...;
   endcase
```

Никогда не забывай о конструкции **default**, иначе на выходе мультиплексора появится защелка (**latch**), что может приводить к нежелательным эффектам!

Рассмотрим немного подробнее конкретное подключение регистров (**Reg's Device**) простейшего периферийного устройства – светодиоды, изображенные ниже. Светодиоды – это устройства вывода, с них нельзя считать информацию, поэтому выход чтения подключается к регистру управления. Сам регистр подключен к светодиодам и его содержимое определяет их состояние – горит или не горит.

Допустим, контроллер светодиодов имеет 16 управляющих 8-битных регистров, начинающихся с адреса **0x6000FFF0**. Все регистры что-то делают, зажигают еще какие-то светодиоды, управляют миганием или еще чем-нибудь, но конкретно 9-й регистр, по адресу **0x6000FF9,** управляет представленными ниже светодиодами. Чтобы этот регистр корректно реагировал на команду записи вход разрешения записи **EN** должен срабатывать, когда в единице одновременно: сигнал разрешения записи периферийному блоку **we_d0**, биты адреса **addr[3:2]** соответствуют значению **2'b10** (потому что 9-й регистр **10**01) и выбран именно этот байт **be[1]** (потому что 9-й регистр 10**01**). На вход **wdata** необходимо подать соответствующие 8 бит слова.



В случае реализации устройств ввода линии записи **wdata** не потребуются, так как само устройство должно обновлять содержимое регистра. Например, клавиатура в момент ввода должна сама обновлять содержимое регистра с кодом нажатой клавиши. Процессору нет никакой необходимости писать в этот регистр, ему надо из него только читать, поэтому для данного примера линий **wdata** не будет.

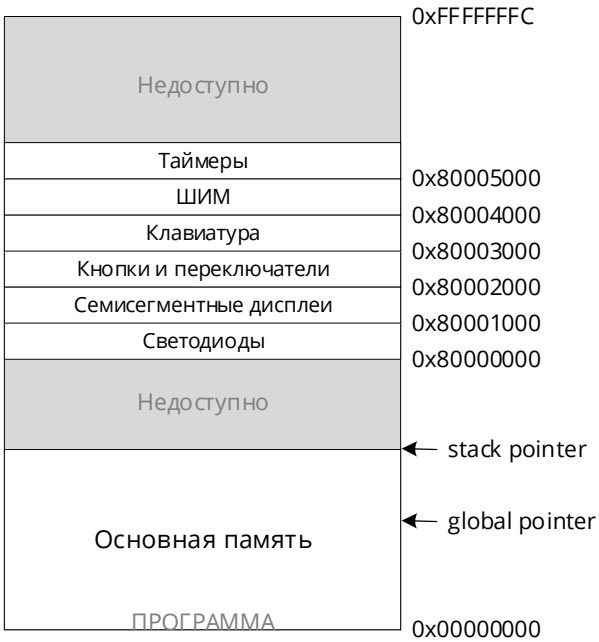
ЗАДАНИЕ: необходимо реализовать одно из устройств ввода (1) и одно устройство вывода (2) и подключить их управляющие регистры к общей шине, для чего потребуется описать дешифратор адреса (3) и мультиплексор на входе процессора (4). Если вычислительная система включает в себя подсистему прерывания, то устройство ввода должны иметь возможность генерировать сигнал прерывания.

Карта памяти

32-битная архитектура RISC-V оперирует 32-битным адресом и может адресовать 2^{32} байт. Часть из них относится к общей памяти, часть – к системе ввода вывода, оставшееся – не используется. Карта памяти, представленная далее, демонстрирует распределение адресов между основной памятью и периферийными устройствами. Распределение адресного пространства определяется разработчиком. В лабораторной работе предлагается придерживаться единой карты, чтобы была возможность обмена программами между студентами и их запуск друг у друга.

Разделение основной памяти на сегменты стека и глобальных данных определяются программным способом – инициализируя вначале программы значения регистров **sp** (**Stack Pointer**) и **gp** (**Global Pointer**) необходимыми значениями. Например, если используется основная память объемом 256 слов, то она будет занимать адреса с **0x00000000** до **0x000003FC**. Разумно загрузить в указатель на стек **sp** старший адрес, а указатель на глобальные данные **gp** поставить на инструкцию выше конца программы.

```
li sp, 0x000003FC
li gp, 0x00000084  # если в программе 32 инструкции
```



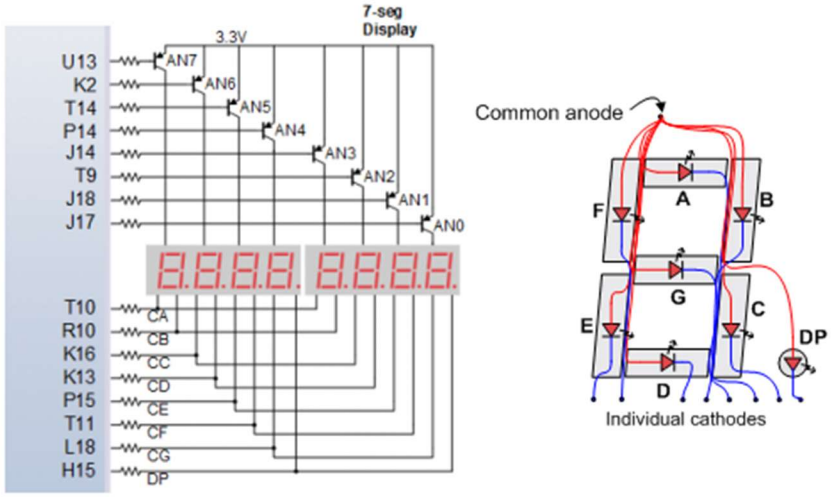
Светодиоды и семисегментные дисплеи

Пример подключения светодиодов рассматривался выше. Стоит лишь отметить, что их на отладочном стенде 16 штук, а их управляющий регистр является полусловом по адресу **0x80000800**.

Семисегментные светодиодные индикаторы предназначены для отображения арабских цифр от 0 до 9. Индикатор называется семисегментным из-за того, что отображаемый символ строится из отдельных семи сегментов (часто бывает, что элементов 8 – еще точка). Внутри корпуса такого индикатора находятся светодиоды, каждый из которых засвечивает свой элемент (сегмент).

Существуют разные способы подключения нескольких семисегментных дисплеев к управляющему устройству. Ниже представлена схема подключения дисплеев к ПЛИС (синий прямоугольник) на отладочном стенде **Nexys A7**. Такая организация имеет минимальное количество подключений, при которой все дисплеи имеют общие сигналы управления выбора сегмента (**CA**, **CB**, **CC**, **CD**,

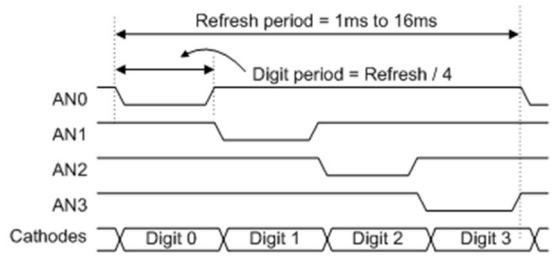
CE, **CF**, **CG**, **DP**) – катоды, и есть сигналы управления анодом (**AN7** – **AN0**), с помощью которых выбирается конкретный дисплей, который будет сейчас светить. Дисплеи на отладочном стенде объединены в группы по четыре штуки, что видно в левой части рисунка. Подключение отдельного дисплея демонстрируется в правой части рисунка.



Эта схема подключения сигналов создает мультиплексированный дисплей, где катодные сигналы являются общими для всех цифр, но они могут освещать только те дисплеи, соответствующий анодный сигнал которых является активным. Это значит, например, что нельзя выводить два разных числа на два разных дисплея одновременно.

Чтобы осветить сегмент, анод должен быть поднят в единицу, а катод – в ноль. Однако, поскольку **Nexys A7** использует транзисторы для подачи достаточного тока в общую анодную точку, анодные значения инвертированы. Следовательно, как для сигналов **AN7** – **AN0**, так и для сигналов **CA** – **CG**/**DP** активным уровнем является 0. Чтобы вывести цифру 7 на третий дисплей необходимо подать на входы **AN3**, **CA**, **CB** и **CC** нули, а на все остальные входы – единицы.

Для того, чтобы видеть разные цифры на разных дисплеях одновременно, достаточно быстрого переключаться между ними, с периодом, приблизительно, 1 миллисекунда. Для человеческого глаза это будет незаметно.



Вашим периферийным устройством будет набор семисегментников на отладочном стенде. Для этого набора необходимо написать контроллер, который будет взаимодействовать с системной шиной и осуществлять управление дисплеями, например, конвертировать двоичные коды процессора в управляющие сигналы для дисплея. Например, когда процессор помещает в управляющий регистр второго дисплея число 7 (00000111), контроллер должен

сформировать для семисегментников сигналы **AN2**, **CA**, **CB** и **CC** как нули, а на все остальные входы подать единицы (этот пример не отражает того факта, что дисплеи при этом должны быстро переключаться).

Для контроллера предусмотрено следующее адресное пространство:

- адреса **0x80001000** – **0x80001007** — под значения, выводимые на конкретный дисплей (допустимые значения **0x00** – **0x0F**);
- адрес **0x80001008** – регистр, отвечающий за включение/выключение отдельных семисегментников (например **0xFF** — все семисегментники горят, **0x00** — все отключены, т. е. каждый бит в этом байте отвечает за конкретный семисегментник);
- адрес **0x80001009** – режим выбора семисегментника. Если значение **0xFF**, то этот режим не используется, если значение **0x00** – **0x07** — выбран конкретный семисегментник. Этот семисегментник должен имитировать анимацию выбора (моргать — включиться с текущем значением на секунду, выключиться на секунду, снова включиться и так циклом). Прочие значения являются запрещенными и не должны записываться;
- адрес **0x8000100A** — сброс контроллера, при котором в регистры **0x00** – **0x07** записываются нули, в **0x08** записывается **0xFF**, в **0x09** записывается **0xFF**.

Необходимо написать контроллер, реализующий подобный функционал, посадить его на системную шину с процессором (после этого адресное пространство контроллера отобразится на внешнюю память, к которой вы получите доступ из процессора) и написать программу, которая продемонстрирует работу вашего контроллера, поочередно запишет данные в регистры значений семисегментников, выключит отдельные семисегментники, включит режим выбора и попереключает его, сбросит контроллер (в программе должны быть предусмотрены паузы, чтобы все за полсекунды не закончилось).

Регистр **0x80001009** главнее регистра **0x80001008**, т. е. если в регистре **0x80001008** семисегментник выключен, а в **0x80001009** этот же семисегментник в режиме выбора, то он будет моргать.

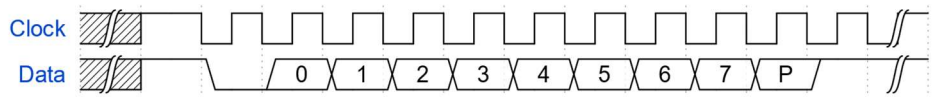
Кнопки и переключатели

Используемый отладочный стенд имеет на своем борту 16 переключателей и 5 кнопок, для которых определяются два адреса в адресном пространстве для управляющих регистров. Два младших байта слова по адресу **0x80002000** для переключателей и младшие 5 бит слова по адресу **0x80002004**. Неиспользуемые биты должны быть равны 0.

Как и говорилось ранее, для некоторых устройств ввода нет необходимости поддерживать возможность записи в управляющий регистр. Так, для кнопок и переключателей регистры вообще не нужны, достаточно подключить сигналы от них к выходному мультиплексору шины (управляемому сигналом **RDsel**).

Контроллер клавиатуры

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



Структура транзакции похожа на UART и состоит из:

1. старт бит – всегда ноль;
2. 8 бит данных (**0** – **7**);
3. бит четности **P**, 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 бывают двух видов: от устройства к контроллеру, и от контроллера к устройству. Например, клавиатура посылает на контроллер 8 битный код нажатой клавиши. В другую сторону – контроллер посылает на клавиатуру команды управления, например команды сброса или выключения светодиодов.

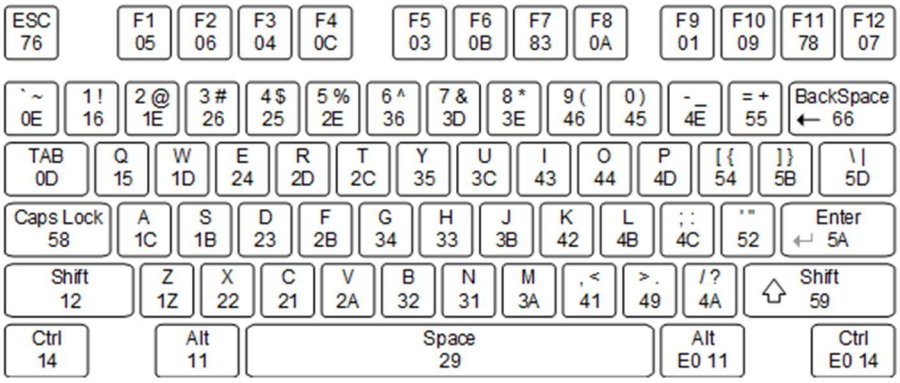
При транзакции от устройства (клавиатуры) к контроллеру сигналы на линиях **Clock** и **Data** генерирует устройство. Контроллер выступает в роли приемника считывая данные по отрицательному фронту **Clock**.

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.

Давайте выясним какие коды же коды передает клавиатура для каждой клавиши. Ниже приведена таблица кодов клавиш. Каждой клавише соответствует код, генерируемый при нажатии (**MAKE**), он изображен на клавише, и код, генерируемый при деактивации (**BREAK**), состоящий из **F0** и кода отжатой клавиши. Коды, состоящие из нескольких байтов, предаются в нескольких подряд идущих транзакций.



Например, если нажать на клавишу **R**, то клавиатура выдаст значение **2D**, а когда клавиша **R** будет отжата, клавиатура пошлет два байта информации: **F0** и сразу следом **2D** – код клавиши **R**.

Разработаем простой контроллер для клавиатуры. Контроллер предназначается для работы только в режиме устройство-контроллер. Для начала определим функционал контроллера и набор сигналов:

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 сброс, активный уровень **1**;
2. **clk_50** – вход тактовой частоты с частотой 50 МГц;
3. **Data** – 8 битная шина данных;
4. **valid_Data** – сигнал готовности данных, равен **1** с момента завершения приема по PS/2 до начала следующей транзакции;
5. **ps2_clk** – тактовая линия PS/2, является внешним сигналом для ПЛИС;
6. **ps2_dat** – сигнальная линия PS/2, является внешним сигналом для ПЛИС.

```
module ps2_keyboard (
    input            areset,
    input            clk_50,

    input            ps2_clk,
    input            ps2_dat,

    output reg       valid_data,
    output     [7:0] data);
```

Прежде всего, необходимо надежно определять нисходящий фронт сигнала **ps2_clk**,так как его качество (крутизна фронтов, зашумленность) может сильно варьироваться в зависимости от клавиатуры и непосредственное тактирование от этого сигнала может вызвать некорректную работу всей схемы в целом.

Для определения нисходящего фронта мы используем вариант схемы для работы с кнопками. Схема представляет из себя 10-битный сдвиговый регистр, в который каждый такт **clk_50** сдвигается текущее значение **ps2_clk**. Схема ожидает момент, когда старшие 5 бит сдвигового регистра заполнены нулями, а младшие 5 бит – единицами. В этот момент на 1 такт возводится сигнал **ps2_clk_negedge**, который используется в остальной схеме.

```
reg [9:0] ps2_clk_detect;
always@(posedge clk_50 or posedge areset) begin
  if (areset)
    ps2_clk_detect <= 10'd0;
  else
    ps2_clk_detect <= {ps2_clk, ps2_clk_detect[9:1]};
end

wire ps2_clk_negedge = &ps2_clk_detect[4:0] &&
                       &(~ps2_clk_detect[9:5]);
```

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.

```
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:  state = IDLE;
      default:                 state = IDLE;
    endcase
   end
```

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.

Сдвиговый регистр, который описан далее, необходим для приема и хранения данных и бита четности. Поэтому, разрядность регистра равна 9. По

завершению транзакции в восьмом бите хранится бит четности с 7-0 бит данных. Данные **data** непрерывным присваиванием выведены на внешнюю шину модуля.

```
reg [8:0] shift_reg;
assign data = shift_reg[7:0];
always @(posedge clk_50 or posedge areset) begin
  if (areset)
    shift_reg <= 9'b0;
  else if (ps2_clk_negedge)
    if (state == RECEIVE_DATA)
      shift_reg <= {ps2_dat, shift_reg[8:1]};
end
```

Если обратиться к началу описания работы контроллера, 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**.

```
reg [3:0] count_bit;
always @ (posedge clk_50 or posedge areset) begin
  if (areset)
    count_bit <= 4'b0;
  else if (ps2_clk_negedge) begin
    if (state == RECEIVE_DATA)
      count_bit <= count_bit + 4'b1;
    else
      count_bit <= 4'b0;
  end
end
```

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

```
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

always @ (posedge clk_50 or posedge areset) begin
  if (areset) valid_data <= 1'b0;
  else if (ps2_clk_negedge)
    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
```

Последним нерассмотренным моментом остался вопрос генерации сигнала готовности данных. Как было сказано ранее сигнал готовности генерируется к концу 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. Поэтому, 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 со значением бита четности. Для упрощения читаемости кода используется функция расчета честности **function parity_calc** для 8 разрядного регистра согласно правилу отрицания побитового XOR регистра. Правило расчета бита паритета можно узнать из стандарта на интерфейс **PS/2**.

В самом простом случае, в качестве управляющего регистра клавиатуры будет выступать 8-битный регистр кода клавиши по адресу **0x80003000**, который должен хранить в себе полученный код клавиши, и обновляться только с приходом сигнала **valid_data**, при этом не реагируя на код отжатия клавиши. Процессору в регистр кода клавиши записывать нельзя, только читать. При этом каждое обновление регистра кода клавиши должно обновлять и регистр **0x80003001**, записывая в него единицу. Процессор тоже должен иметь возможность записывать в этот регистр, таким образом он будет понимать новое значение в регистре кода клавиши или нет.

Считав из регистра **0x80003001** единицу, он поймет, что в регистре кода клавиши новое значение кода. Тогда он его считает, а в **0x80003001** запишет ноль. В следующий раз, когда он прочитает в регистре **0x80003001** единицу, он поймет, что значение кода обновилось.

В случае реализованной подсистемы прерывания регистр **0x80003001** не нужно подключать к системной шине, вместо этого он должен быть подключен к одному из входов запроса прерывания. Сигнал «прерывание обслужено» должно обнулять этот регистр.