Лабораторный практикум

«Проектирование цифровых устройств с помощью Verilog HDL»

Лабораторная работа №3 Секундомер

В прошлых лабораторных работах мы изучили базовые строительные блоки цифровых устройств. Теперь у нас уже достаточно знаний для реализации несложного, но функционально законченного цифрового устройства.

В данной лабораторной работе мы познакомимся с процессом проектирования полноценного цифрового устройства на примере разработки простого секундомера. Мы подробно, поэтапно, рассмотрим процесс проектирования, проиллюстрировав каждый этап графической схемой.

Для эффективного проектирования любого цифрового устройства нужно придерживаться некоторой «канвы» проектирования. Это поможет не запутаться и последовательно разобраться с вопросами, возникающими в ходе проектирования.

Начинать проектирование любого цифрового устройства следует с определения входов и выходов. Нужно понять какие данные будут входными для проектируемого устройства, и какие данные нам надо выработать и подать на выход.

В случае секундомера справедливы такие рассуждения:

Чтобы управлять работой секундомера нам понадобятся два входа: «старт/стоп» и «сброс».

Для отображения времени можно воспользоваться семисегментными индикаторами. Значит, для управления каждым из них понадобится семибитная шина, которая будет выходом нашего устройства.

Для отображения времени выделим 2 индикатора для отображения количества прошедших секунд и 2 индикатора для

отображения количества прошедших десятых и сотых долей секунды.

Значит выходом секундомера будут четыре семибитные шины для управления индикаторами.

В основе секундомера лежит счётчик. Работая, секундомер отсчитывает время, считая количество пришедших импульсов сигнала синхронизации, частота которого заранее известна.

Т.е. нам потребуется сигнал синхронизации со стабильной частотой.

Больше никаких входов и выходов не требуется.

Общая схема на данный момент выглядит так:

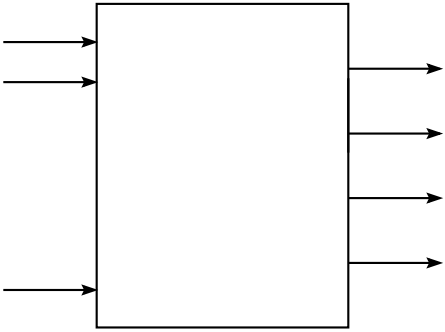

Рис. 1.1: Описание входов и выходов секундомера

Начнём описывать модуль на языке Verilog:

```
 1  module stopwatch (
 2  input start_stop,
 3  input reset,
 4  input clk,
 5  output [6:0] hex0,
 6  output [6:0] hex1,
 7  output [6:0] hex2,
 8  output [6:0] hex3);
 9
10  endmodule;
```

Листинг 1.1: Описание входов и выходов модуля на языке Verilog HDL

Теперь приступим к описанию «внутренностей» модуля.

Чтобы реализовать секундомер, нам необходимо отсчитывать время.

Для отсчёта времени в цифровых устройствах считают количество прошедших импульсов синхронизации (тактов). Так как тактовые импульсы генерируются кварцевым генератором со стабильной, известной нам, частотой, то мы можем рассчитать количество импульсов, которое соответствует заданному времени.

Например, если в устройстве установлен кварцевый генератор на 26 МГц, то одной секунде соответствует 26 миллионов тактовых импульсов, а одной сотой секунды соответствует 260 тысяч тактовых импульсов.

Для того, чтобы отсчитать это количество импульсов подходит единственный из известных нам «строительных блоков» - счётчик:

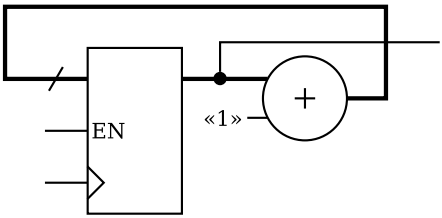

Рис. 1.2: Структура счетчика

Как мы уже говорили, счётчик состоит из регистра и сумматора. Чтобы счётчик циклически отсчитывал одну сотую секунды его необходимо обнулить после того, как он отсчитает 260 тысяч тактовых импульсов. В этот же момент нужно выработать сигнал для остальной схемы, что прошла одна сотая секунды.

Из всех цифровых блоков, которые мы рассмотрели, для реализации задачи сравнения текущего значения счётчика с константой подходит только компаратор. На один из входов компаратора подадим текущее значение счётчика, а на другой вход - константу 260 000.

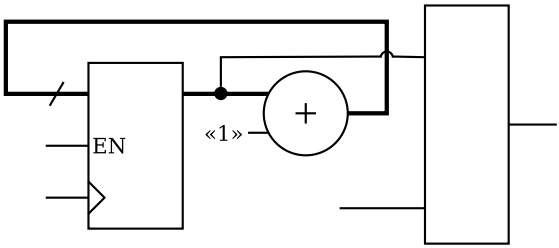

Рис. 1.3: Структура счетчика с компаратором

Пока значения на входах компаратора будут отличаться, на выходе компаратора будет значение «0». Когда значения будут равны, компаратор изменит выход с «0» на «1», это и будет признак того, что прошло 0.01 секунды. Для того, чтобы можно было эффективно использовать сигнал «прошло 0.01с», этот сигнал должен иметь длительность равную 1 такту.

Этот же сигнал мы будет использовать для управления сбросом счётчика.

Итак, счётчик должен после достижения значения 260 000 принять значение «0», но переход должен случиться, как и все остальные переходы, в момент перехода тактового сигнала из «0» в «1».

Сброс, отвечающий таким условиям, называется «синхронный сброс».

Посмотрите, как будет выглядеть на временной диаграмме как будет работать счётчик если выход компаратора, подключить как сигнал синхронного сброса:

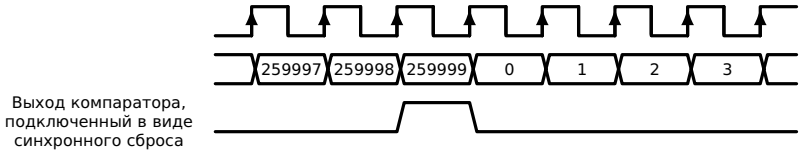

Рис. 1.4: Временная диаграмма работы синхронного счетчика

А так выглядит временная диаграмма, если подключить выход компаратора к входу асинхронного сброса триггера:

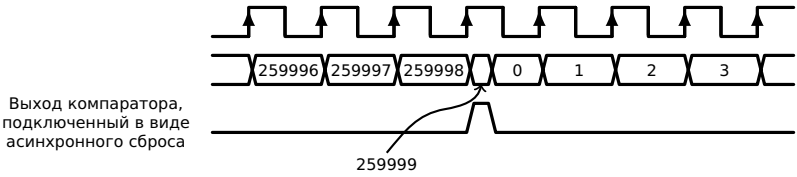

Рис. 1.5: Временная диаграмма работы асинхронного счетчика

Выход компаратора, установившись в единицу, моментально сбросит счётчик и, так как значение счётчика изменилось, а значит, изменился и один из входов компаратора, выход компаратора сразу же перейдет в значение «0».

Обратите внимание, что длительность сигнала с выхода компаратора должна быть равна одному такту. Ведь в дальнейшем нам необходимо будет считать события «прошла одна сотая секунды», а значит подготовить сигнал единичной длительности, который соответствует этому событию (см. лабораторную работу №3).

Сигнал с компаратора, в случае, когда он подключен в виде синхронного сброса, полностью удовлетворяет этому условию, а значит нам не придется в дальнейшем вводить новые фрагменты схемы.

Как реализовать синхронный сброс в цифровом устройстве?

Для этого можно использовать мультиплексор. Схема будет выглядеть следующим образом:

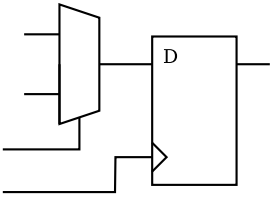

Рис. 1.6: Схема реализации синхронного сброса

Если подключить sync_reset к выходу компаратора, то когда счётчик достигнет порогового значения, выход компаратора изменится и переключит мультиплексор. Теперь на выход мультиплексора будет подаваться «0». Этот сигнал будет

поступать на вход триггера, но запись нового значения произойдет только во время положительного фронта сигнала синхронизации.

Для общего сброса секундомера при нажатии кнопки «сброс» как раз можно воспользоваться входом асинхронного сброса регистра. Ведь при нажатии кнопки «сброс» можно обнулять регистр мгновенно.

Теперь надо выбрать правильный сигнал управления работой счётчика - сигнал разрешения работы (Enable, EN). Ведь счётчик должен начинать считать после нажатия кнопки «старт/стоп», а после её повторного нажатия должен останавливаться.

Для управления работой счётчика можно использовать сигнал, который будет единицей, пока счётчик должен работать и нулём, если отсчёт времени остановлен. Как раз такой сигнал можно подать на вход разрешения работы регистра. Назовём этот сигнал «device_running».

Посмотрите, как выглядит остановка и запуск счётчика в таком случае:

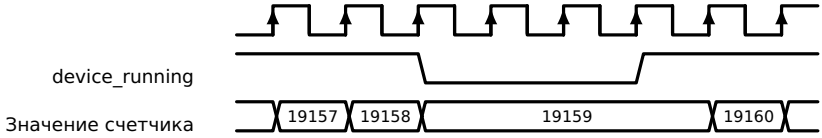

Рис. 1.7: Временная диаграмма работы сигнала device_running

К проектированию и описанию схемы, которая вырабатывала бы сигнал «device_running», мы вернемся позднее.

Стоит обратить внимание на следующий момент: что будет, если счётчик остановить в тот момент времени, когда его значение стало равно 259999?

Взгляните на временную диаграмму:

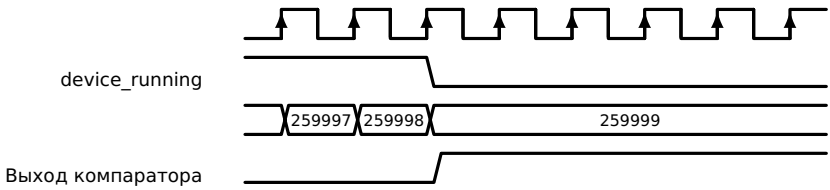

Рис. 1.8: Временная диаграмма работы счетчика

Для того чтобы не допустить такого поведения, можно немного изменить условие, запрещающее работу счётчика. Теперь мы будем дополнительно проверять сигнал с компаратора. И если счётчик в данный момент равен 259999, то запретить его работу будет невозможно.

Для решения этой задачи подойдет вентиль «или». Условие будет таким: «работа разрешена, если сигнал «device_running» равен единице **ИЛИ** когда текущее значение счётчика равно 259999».

Теперь схема счётчика выглядит следующим образом:

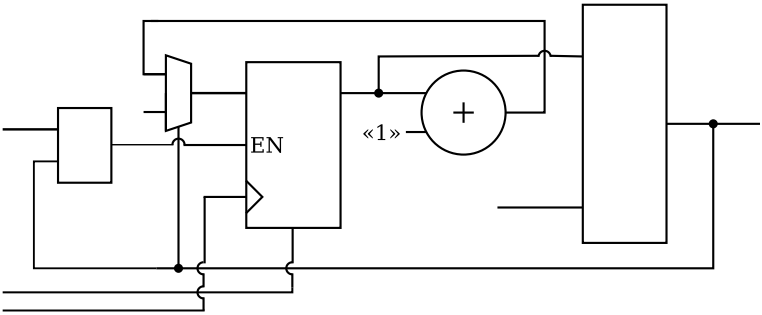

Рис. 1.9: Схема счетчика тысячных долей секунды

Когда мы представили схему в виде набора цифровых блоков, мы можем описать её поведение на языке Verilog:

```
1 //регистр счётчика
2 reg [16:0] pulse_counter = 17'd0;
3
4 //описание компаратора
5 wire hundredth_of_second_passed =
```

```
 6                (pulse_counter == 17'd259999);
 7
 8  //описание счётчика
 9  always @(posedge clk or posedge reset) begin
10     if (reset) pulse_counter <= 0;
11         //асинхронный сброс
12
13     //сигнал разрешения работы счётчика
14     else if (device_running |
15                hundredth_of_second_passed)
16
17         //синхронный сброс
18         if (hundredth_of_second_passed)
19                pulse_counter <= 0;
20
21         //увеличение счётчика на единицу
22         else pulse_counter <= pulse_counter + 1;
23  end;
```

Листинг 1.2: Описание счетчика тактовых импульсов на языке Verilog HDL

Теперь, когда у нас есть счетчик, отсчитывающий такты и сигнализирующий о том, что прошла сотая доля секунды, мы можем отсчитывать сотые доли секунды.

Перед нами встаёт выбор.

Первый вариант – отсчитывать количество прошедших сотых долей секунды единственным счётчиком. Значение этого счётчика мы можем дешифрировать, чтобы выделить из него количество единиц, десятков, сотен и тысяч прошедших долей секунды, чтобы подать эти значения на дешифраторы семисегментных индикаторов:

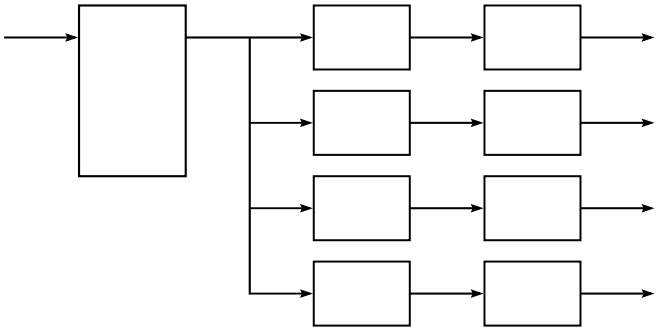

Рис. 1.10: Первый пример реализации секундомера

Второй вариант – использовать отдельные счётчики для сотых долей секунды, десятых долей секунды, целых секунд и десятков секунд.

Т.е. первый счетчик подсчитывает количество прошедших сотых долей секунды от 0 до 9, и, затем обнуляется, вырабатывая сигнал «прошла десятая доля секунды». Следующий счётчик, точно также считает уже десятые доли и вырабатывает сигнал «прошла одна секунда» и так далее.

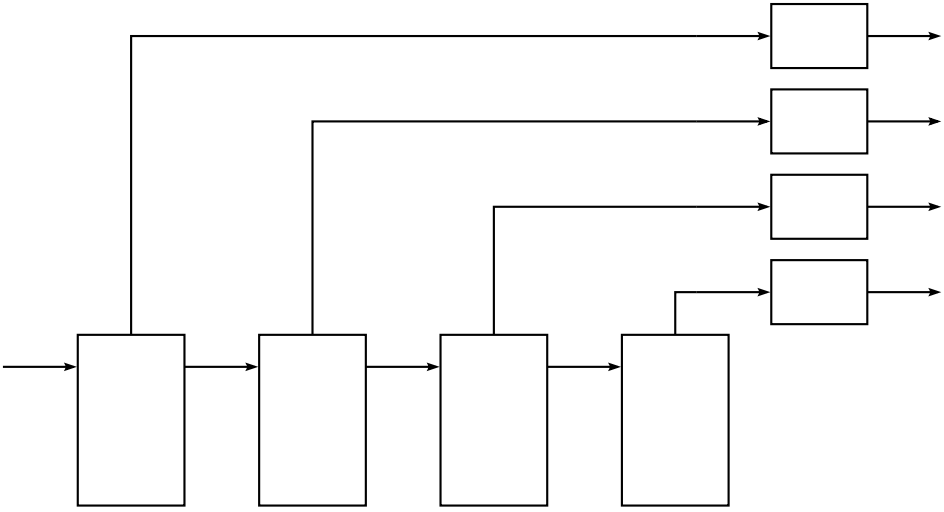

Рис. 1.11: Второй пример реализации секундомера

Второй вариант для нас проще в реализации, компактнее, удобнее и понятнее.

Поэтому выберем именно его.

В качестве счётчиков подойдет уже описанная нами схема для подсчёта тактов, но с небольшими правками.

Счетчики подойдут нам потому, что функция их идентична – подсчёт событий с ограничением диапазона. Изменить нужно будет только разрядность счётчика с 16 на 4 и верхнюю границу счёта с 260000 на 9. Тогда счётчик будет выдавать признак переполнения (достижения границы отсчёта, когда его значение будет становиться) девяткой.

Еще одним моментом, о котором нужно позаботиться – длительность выходного сигнала.

Пока счётчик считал такты, его значение менялось каждый такт. Компаратор просто не мог принять значение 1 более чем на один такт. Теперь ситуация выглядит следующим образом:

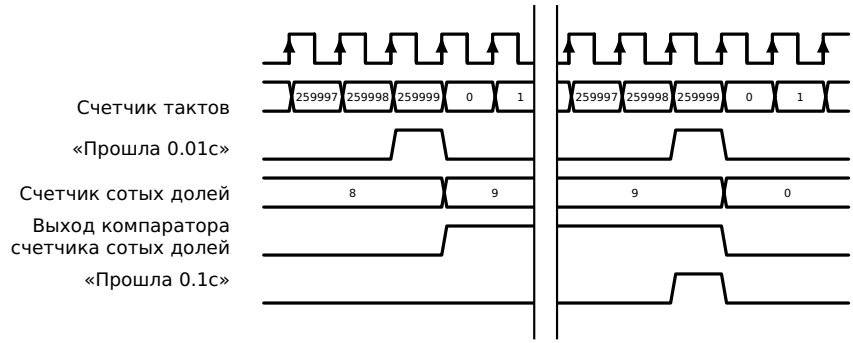

Рис. 1.12: Временная диаграмма работы секундомера

Счетчик будет переключаться каждую 0,01 секунды, 0,1 секунды ,1 секунду или 10 секунд. И выход компаратора будет устанавливаться в 1 на всё время, которое потребуется для переключения счётчика из 9 в ноль. Т.е. в случае счётчика сотых долей секунды потребуется 259999 тактов.

Как выделить из всего времени, пока счётчик имеет значение «9» сигнал длительностью в один такт, который возникает в нужный момент времени? На временной диаграмме этот сигнал отмечен как «прошла 0,1с.»

Сигнал «прошла 0,1с» можно получить из сигналов, представленных на временной диаграмме следующим образом: «прошла 0,1с» правда, когда выход компаратора равен единице **И** «прошла 0,01с».

Схема счётчика практически не изменилась:

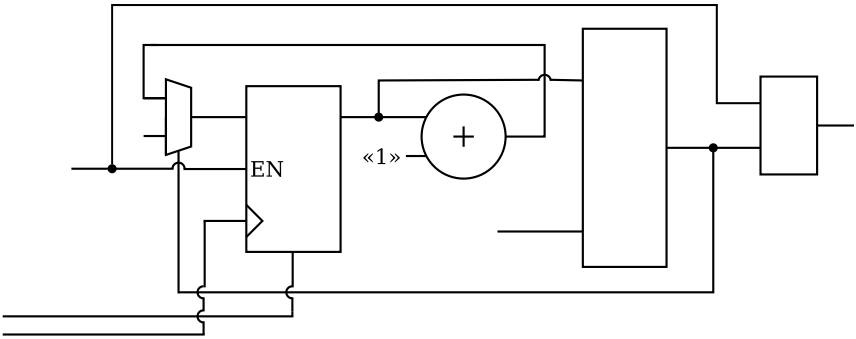

Рис. 1.13: Схема счётчика сотых долей секунды

Скорректируем описание её работы на Verilog:

```
1  //регистр счётчика
2  reg [3:0] hundredth_counter = 4'd0;
3
4  //описание компаратора
5  wire tenth_of_second_passed =
6         ((hundredths_counter == 4'd9) &
7           hundredths_of_second_passed);
8
9  //описание счётчика
10 always @(posedge clk or posedge reset) begin
11    if (reset) hundredths_counter <= 0;
12        //асинхронный сброс
13
14    //сигнал разрешения работы счётчика
15    else if (hundredth_of_second_passed)
16
17        //синхронный сброс
18        if (tenth_of_second_passed)
19               hundredths_counter <= 0;
```

```
20
21          //увеличение счётчика на единицу
22          else hundredths_counter <=
23                  hundredths_counter + 1;
24  end;
```

Листинг 1.3: Описание счетчика сотых долей секунды на языке Verilog HDL

Счётчики десятых долей секунды, целых секунд и десятков секунд устроены абсолютно также. В описаниях изменятся только названия сигналов и регистров. Единственное в чем необходимо быть внимательным – это подключение сигналов. Для правильного подключения надо свериться со схемой, которую мы выбрали ранее.

Теперь вернёмся к вопросам, которые мы отложили ранее.

В нашем устройстве пока нет описания схемы, которая вырабатывает сигнал «device_stopped». Сигнал должен управляться кнопкой, поэтому как мы уже говорили, в лабораторной работе №3 потребуется схема, синхронизирующая сигнал, поступающий с кнопки с внутренним сигналом clk (тактовые импульсы).

Также сразу выделим из всего нажатия признак того, что кнопка была нажата, так, чтобы по длительности этот признак был равен одному такту.

Тогда схема будет абсолютно такой же, как и в лабораторной работе №3 и будет выглядеть следующим образом:

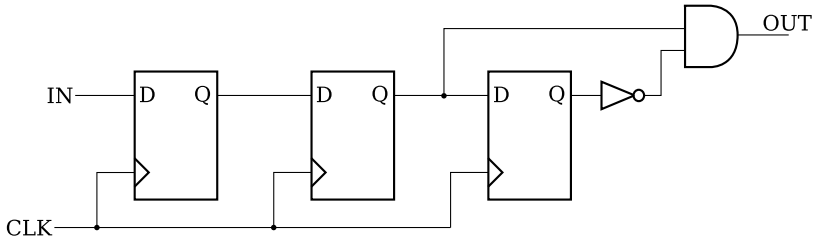

Рис. 1.14: Структура схемы синхронизации сигнала

Поведение такой схемы описывается на языке Verilog следующим образом:

```
1  reg [2:0] button_syncroniser;
2        wire button_was_pressed;
3
4  always @(posedge clk) begin
5     button_syncroniser[0] <= in;
6     button_syncroniser[1] <= button_syncroniser[0];
7     button_syncroniser[2] <= button_syncroniser[1];
8  end;
9
10 assign button_was_pressed <= ~button_syncroniser[2]
11                              & button_syncroniser[1];
```

Листинг 1.4: Описание схемы синхранизации на языке Verilog HDL

Эта схема и её описание подробно рассмотрены в лабораторной работе №3

Теперь нам нужно построить схему, которая по нажатию кнопки переключала бы сигнал «device_stopped» из «0» в «1» и из «1» в «0».

Что нам понадобится? Триггер, чтобы хранить значение «device_stopped». Чтобы менять значение на противоположное надо знать противоположное значение, значит, нужен инвертор. Событие должно случаться по сигналу «button_was_pressed», а значит речь, скорее всего, идет о входе разрешения работы триггера.

Немного подумав над этими вводными, нетрудно составить следующую схему:

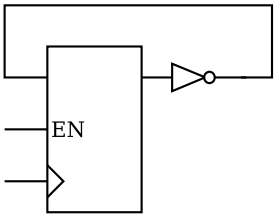

Рис. 1.15: Схема переключения сигнала «device_running»

Временная диаграмма, которая соответствует работе этого устройства:

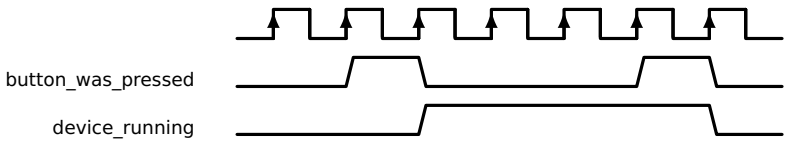

Рис. 1.16: Временная диаграмма переключения сигнала «device_running»

Описание поведения этой схемы на Verilog также не представляет сложности. Выполните его самостоятельно.

Теперь приведем полное описание секундомера (за исключением схемы, вырабатывающей сигнал «device_stopped»), выполненное на языке Verilog:

```
 1 module stopwatch (
 2 input start_stop,
 3 input reset,
 4 input clk,
 5 output [6:0] hex0,  //индикатор сотых долей секунды
 6 output [6:0] hex1,  //десятых долей секунды
 7 output [6:0] hex2,  //секунд
 8 output [6:0] hex3); //десятков секунд
 9
10 // Часть I - синхронизация обработки
11 // нажатия кнопки «Старт/Стоп»
12 reg [2:0] button_syncroniser;
13       wire button_was_pressed;
14
15 always @(posedge clk) begin
16    button_syncroniser[0] <= start_stop;
17    button_syncroniser[1] <= button_syncroniser[0];
18    button_syncroniser[2] <= button_syncroniser[1];
19 end
20
21 assign button_was_pressed = ~button_syncroniser[2]
22                              & button_syncroniser[1];
23
24
25 // Часть II - выработка признака «device_running»
26 // Самостоятельная работа студента!
```

```
reg device_running;



// Часть III - счётчик импульсов
// и признак истечения 0,01 сек
reg [16:0] pulse_counter = 17'd0;
wire hundredth_of_second_passed =
       (pulse_counter == 17'd259999);
always @(posedge clk or posedge reset) begin
   if (reset) pulse_counter <= 0;
       //асинхронный сброс
   else if (  device_running |
              hundredth_of_second_passed)
       if (hundredth_of_second_passed)
              pulse_counter <= 0;
       else pulse_counter <= pulse_counter + 1;
      end


// Часть IV - основные счётчики
reg [3:0] hundredths_counter = 4'd0;
wire tenth_of_second_passed =
       ((hundredths_counter == 4'd9) &
         hundredth_of_second_passed);
always @(posedge clk or posedge reset) begin
   if (reset) hundredths_counter <= 0;
   else if (hundredth_of_second_passed)
       if (tenth_of_second_passed)
              hundredths_counter <= 0;
       else hundredths_counter <=
              hundredths_counter + 1;
      end


reg [3:0] tenths_counter = 4'd0;
wire second_passed = ((tenths_counter == 4'd9) &
                       tenth_of_second_passed);
always @(posedge clk or posedge reset) begin
   if (reset) tenths_counter <= 0;
```

```
   else if (tenth_of_second_passed)
       if (second_passed) tenths_counter <= 0;
       else tenths_counter <= tenths_counter + 1;
      end

reg [3:0] seconds_counter = 4'd0;
wire ten_seconds_passed =
       ((seconds_counter == 4'd9) &
         second_passed);
always @(posedge clk or posedge reset) begin
   if (reset) seconds_counter <= 0;
   else if (second_passed)
       if (ten_seconds_passed) seconds_counter <= 0;
       else seconds_counter <= seconds_counter + 1;
      end

reg [3:0] ten_seconds_counter = 4'd0;
always @(posedge clk or posedge reset) begin
   if (reset) ten_seconds_counter <= 0;
   else if (ten_seconds_passed)
       if (ten_seconds_counter == 4'd9)
              ten_seconds_counter <= 0;
       else ten_seconds_counter <=
              ten_seconds_counter + 1;
      end




// Часть V - дешифраторы для отображения
// содержимого основных регистров
// на семисегментных индикаторах
reg [6:0] decoder_ten_seconds;
always @(*) begin
   case (ten_seconds_counter)
       4'd0: decoder_ten_seconds <= 7'b0000001;
       4'd1: decoder_ten_seconds <= 7'b1001111;
       4'd2: decoder_ten_seconds <= 7'b0010010;
       4'd3: decoder_ten_seconds <= 7'b0000110;
       4'd4: decoder_ten_seconds <= 7'b1001100;
```

```
        4'd5: decoder_ten_seconds <= 7'b0100100;
        4'd6: decoder_ten_seconds <= 7'b0100000;
        4'd7: decoder_ten_seconds <= 7'b0001111;
        4'd8: decoder_ten_seconds <= 7'b0000000;
        4'd9: decoder_ten_seconds <= 7'b0000100;
        default: decoder_ten_seconds <= 7'b1111111;
    endcase;
end
assign hex3 = decoder_ten_seconds;
reg [6:0] decoder_seconds;
always @(*) begin
    case (seconds_counter)
        4'd0: decoder_seconds <= 7'b0000001;
        4'd1: decoder_seconds <= 7'b1001111;
        4'd2: decoder_seconds <= 7'b0010010;
        4'd3: decoder_seconds <= 7'b0000110;
        4'd4: decoder_seconds <= 7'b1001100;
        4'd5: decoder_seconds <= 7'b0100100;
        4'd6: decoder_seconds <= 7'b0100000;
        4'd7: decoder_seconds <= 7'b0001111;
        4'd8: decoder_seconds <= 7'b0000000;
        4'd9: decoder_seconds <= 7'b0000100;
        default: decoder_seconds <= 7'b1111111;
    endcase;
end
assign hex2 = decoder_seconds;

reg [6:0] decoder_tenths;
always @(*) begin
    case (tenths_counter)
        4'd0: decoder_tenths <= 7'b0000000;
        4'd1: decoder_tenths <= 7'b1001111;
        4'd2: decoder_tenths <= 7'b0010010;
        4'd3: decoder_tenths <= 7'b0000110;
        4'd4: decoder_tenths <= 7'b1001100;
        4'd5: decoder_tenths <= 7'b0100100;
        4'd6: decoder_tenths <= 7'b0100000;
        4'd7: decoder_tenths <= 7'b0001111;
        4'd8: decoder_tenths <= 7'b0000000;
        4'd9: decoder_tenths <= 7'b0000100;
```

```
147         default: decoder_tenths <= 7'b1111111;
148     endcase;
149 end
150 assign hex1 = decoder_tenths;
151 
152 reg [6:0] decoder_hundredths;
153 always @(*) begin
154     case (hundredths_counter)
155         4'd0: decoder_hundredths <= 7'b0000000;
156         4'd1: decoder_hundredths <= 7'b1001111;
157         4'd2: decoder_hundredths <= 7'b0010010;
158         4'd3: decoder_hundredths <= 7'b0000110;
159         4'd4: decoder_hundredths <= 7'b1001100;
160         4'd5: decoder_hundredths <= 7'b0100100;
161         4'd6: decoder_hundredths <= 7'b0100000;
162         4'd7: decoder_hundredths <= 7'b0001111;
163         4'd8: decoder_hundredths <= 7'b0000000;
164         4'd9: decoder_hundredths <= 7'b0000100;
165         default: decoder_hundredths <= 7'b1111111;
166     endcase;
167 end
168 assign hex0 = decoder_hundredths;
169 
170 endmodule
```

Листинг 1.5: Описание секундомера на языке Verilog HDL

1.1 Задание лабораторной работы:

Изучить разработку к лабораторной работе.

Самостоятельно выполнить описание схемы, вырабатывающей сигнал «device_stopped».

Выполнить синтез и моделирование работы счётчика.

Продемонстрировать в результатах моделирования фрагменты временных диаграмм, приведенных в заработке.

Изучить работу устройства, реализованного в ПЛИС учебного стенда.

Подготовить ответы на вопросы к защите лабораторной работы.

1.2 Вопросы к защите лабораторной работы

in progress