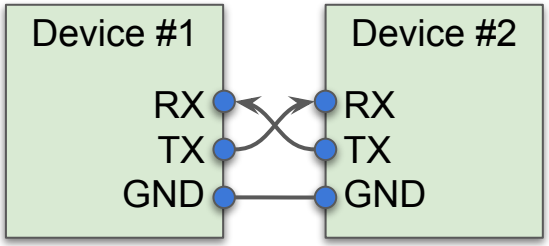
UART. Создание стекового калькулятора

Artem Voronov, Roman Voronov, Rafael Ilyasov

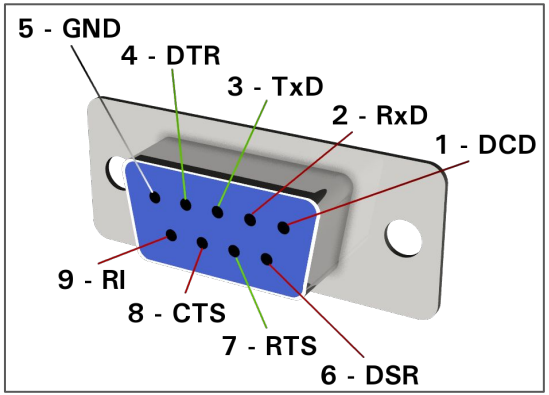
5 March 2022, Innopolis, Russia

UART

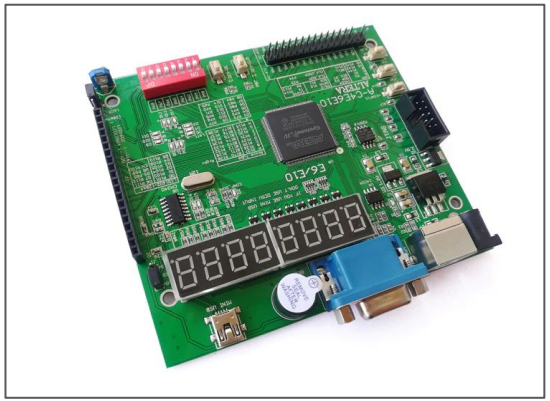
UART - universal asynchronous receiver-transmitter (универсальный асинхронный приёмопередатчик)



RS232

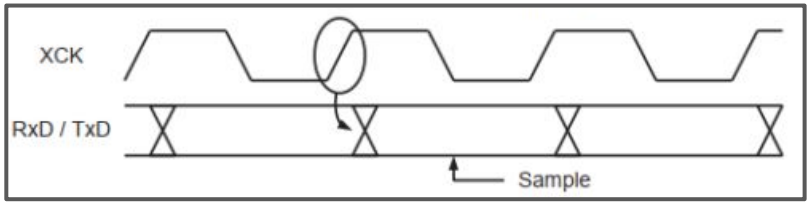


FPGA

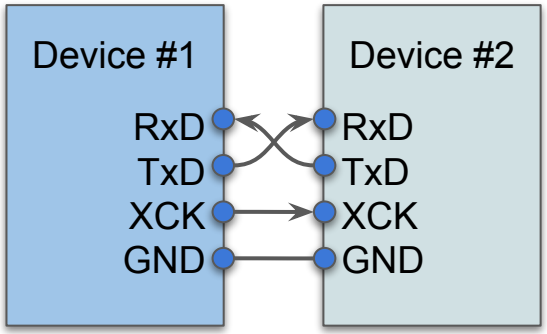


USART

USART - универсальный синхронный/асинхронный приёмо/передатчик

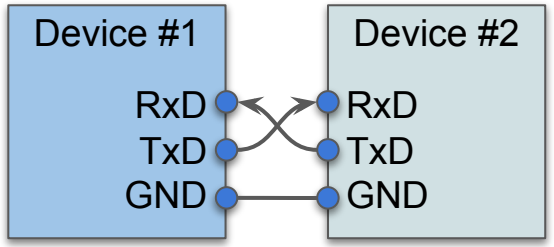


USART в синхронном режиме:



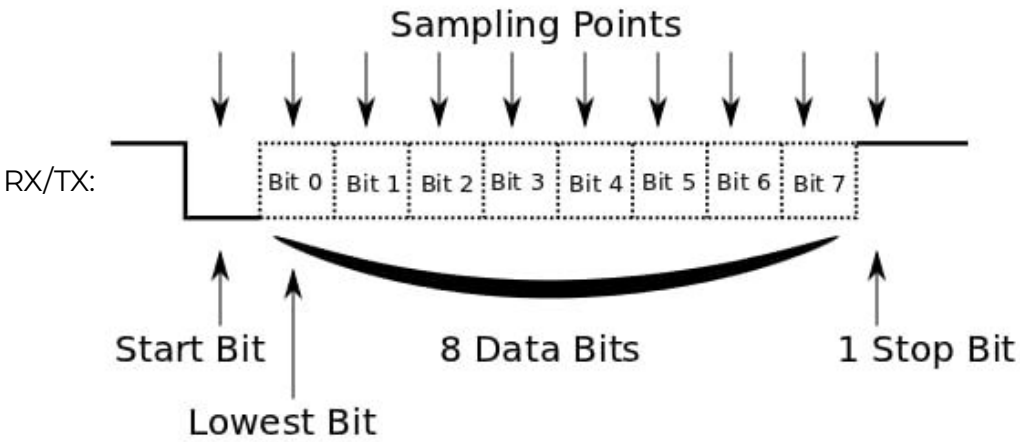
USART MASTER USART SLAVE

USART в асинхронном режиме:



USART MASTER USART SLAVE

Формат кадра UART



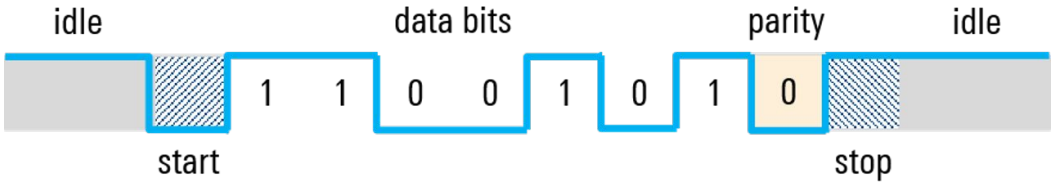
Свойства:

- Состояние IDLE (отсутствие сигнала) - логическая единица.
- Передача слова начинается со стартового бита - логический ноль.
- Младший бит впереди (***little-endian***).
- После передачи одного слова данных идёт стоп бит - логическая единица.

Короткая запись параметров UART:

9600/8-N-1

- Число до черты указывает на скорость UART.
- Первая цифра после обозначает количество бит **данных**, обычно от 5 до 9.
- Буква обозначает наличие и тип бита чётности.
  N (No parity) — без бита чётности;
  E (Even parity) — с битом проверки на чётность,
  O (Odd parity) — с битом проверки на нечётность;
- Последняя цифра обозначает длительность стоп-бита. Встречаются значения 1, 1.5 и 2.



Параметры UART: скорость передачи

- Скорость работы UART измеряется в бодах, а скорость передачи полезных данных — в битах/секунду.

Boud rate	Time required to transfer 1 KiB over UART 8-N-1
9600	**1067 msec.**
19200	533 msec.
38400	267 msec.
57600	178 msec.
115200	88.9 msec.
1 Mbit	10.24 msec.

Скорость передачи данных в битах в секунду:

$$V_{bit} = \frac{d}{d+1+s+p} V_{UART}$$

где:

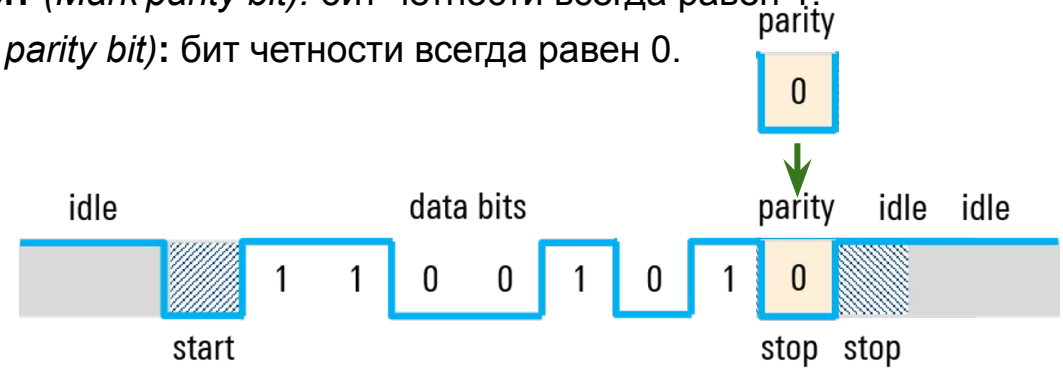
- V_{UART} — скорость UART (например: 9600, 115200), бод;
- d — количество бит данных;
- s — количество стоповых бит;
- p — количество бит четности, $p = 1$ если бит четности присутствует, или $p = 0$ если бит четности отсутствует;
- единица в знаменателе отражает наличие стартового бита.

Параметры UART: бит чётности (**не обязательно**)

- Используется для обнаружения ошибок.

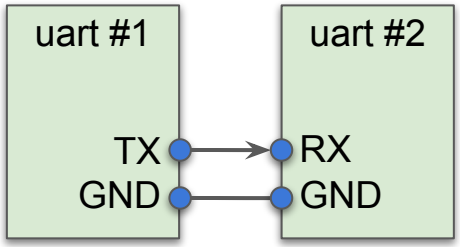
- **N - Отсутствует** *(None)*: без бита чётности.
- **E - Чётное равенство** *(Even parity)*: количество 1ц должно быть **чётно**.
- **O - Нечётное равенство** *(Odd parity)*: количество 1ц должно быть **нечётно**.
- **M - Отмеченный бит** *(Mark parity bit):* бит четности всегда равен 1.
- **S - Пропуск** *(Space parity bit)*: бит четности всегда равен 0.

UART 7-E-1:

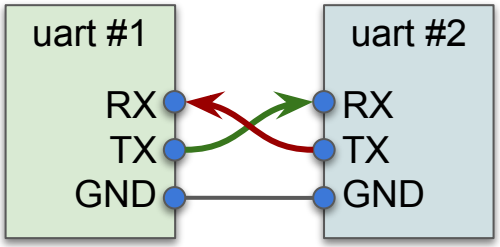


Параметры UART: **Дуплекс/Симплекс/Полудуплекс**

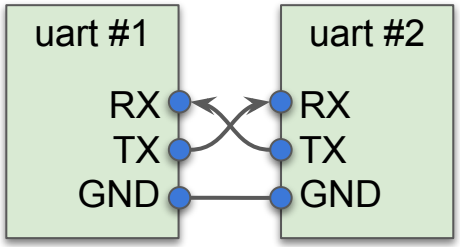
- **Симплекс -** данные передаются только в одном направлении.
- **Полудуплекс** - каждая сторона осуществляет передачу, но только по очереди.
- **Дуплекс** - обе стороны могут передавать одновременно.



Симплекс



Полудуплекс



Дуплекс

Заключение

●**UART** - универсальный асинхронный приёмопередатчик и представляет собой простой двухпроводной протокол для обмена последовательными данными.

●**Асинхронность** означает отсутствие общего тактового сигнала, поэтому на обоих устройствах необходимо настроить одинаковую битовую скорость и параметры кадра.

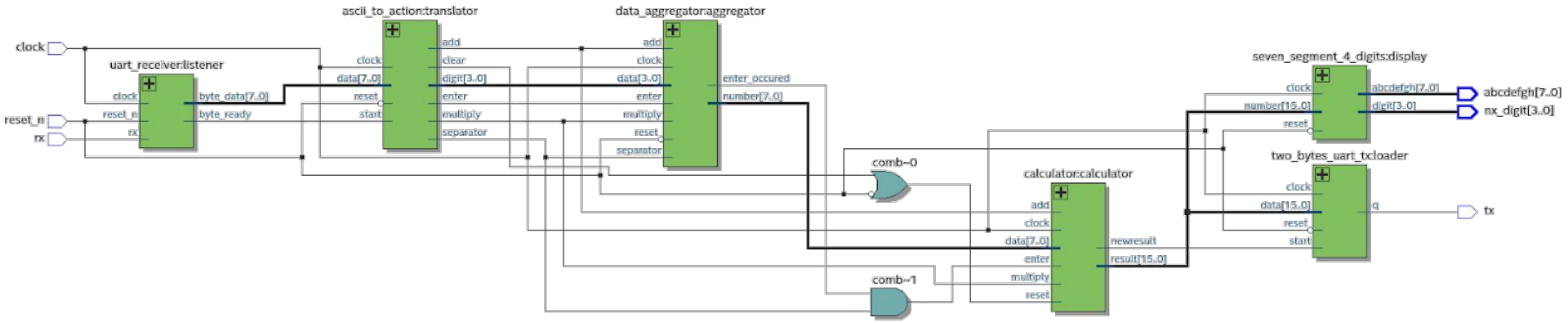
●**Стартовые** и **стоповые биты** используются, чтобы указать, где начинаются и заканчиваются пользовательские данные, или для «кадрирования» данных.

●Необязательный **бит четности** может использоваться для обнаружения однобитовых ошибок.

●UART по-прежнему **является широко используемым** протоколом последовательной передачи данных, но в последние годы был заменен в некоторых областях применения такими интерфейсами, как SPI, I2C, USB и Ethernet.

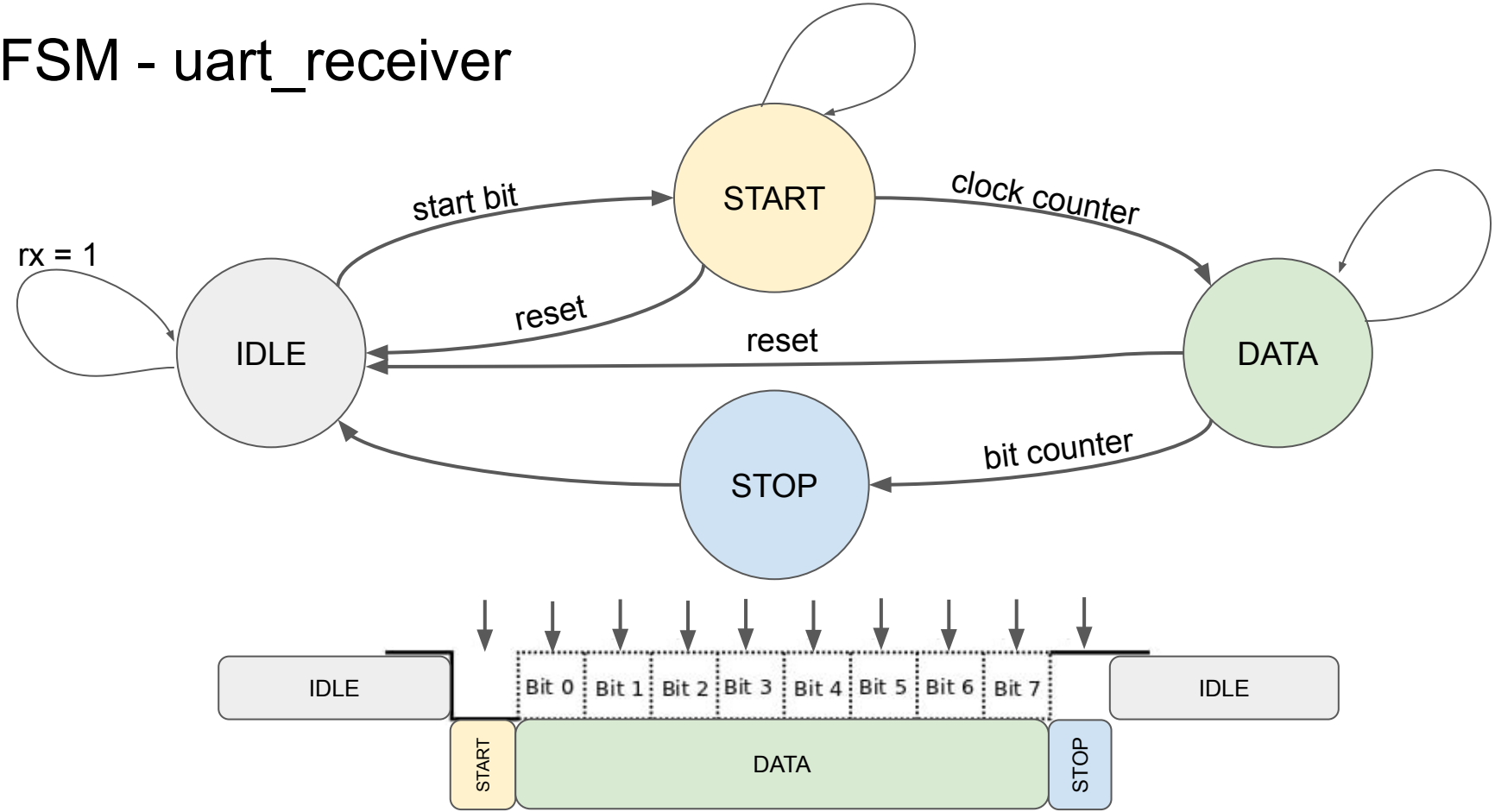
Пример проекта с использованием UART

Calculator RTL viewer:



Код для плат предоставлен:
https://github.com/DigitalDesignSchool/2021dev/tree/main/Innopolis%20University/fpga_calculator_with_uart

FSM - uart_receiver



Создание модуля UART receiver

/uart_reciver.sv

```
1  module UART_receiver
2      input clock,
3      input reset_n,
4      input rx,
5
6      output reg [7:0] byte_data,
7      output           byte_ready
8  );
9    parameter clock_frequency = 50000000;
10   parameter baud_rate       = 9600;
11   parameter clock_cycles_in_bit =
     clock_frequency / baud_rate;
12
13   enum {IDLE, START, DATA, STOP} state;
14   reg [31:0] counter;
15   reg [3:0] bit_count;
16   reg [1:0] rx_filter;
      …
```

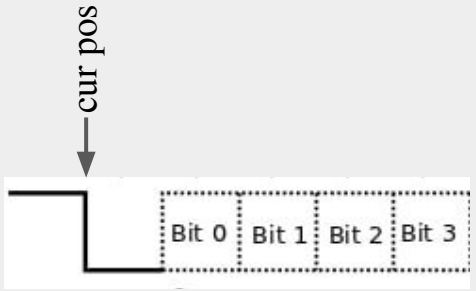
```
18   always @(posedge clock) begin
19
20     if (!reset_n) begin
21       counter <= 0;
22       state <= IDLE;
23       rx_filter<= 0;
24     end else begin
25       rx_filter <= {rx_filter[0], rx};
26
27       case (state)
…          IDLE:  begin … end
37         START: begin … end
…          DATA:  begin … end
54         STOP:  begin … end
…
60         default: state <= IDLE;
61       endcase
62
63     end
   end
endmodule
```

Создание модуля UART receiver: State IDLE

/uart_reciver.sv

```
…   ...
27   rx_filter <= {rx_filter[0], rx};
28   case (state)
29
30     IDLE: begin
31       byte_ready <= 0;
32       if (rx_filter == 2'b10) begin //Edge for start bit
33         state <= START;
34         bit_count <= 0;
34         counter <= 0;
35       end
36     end
…    ...
```

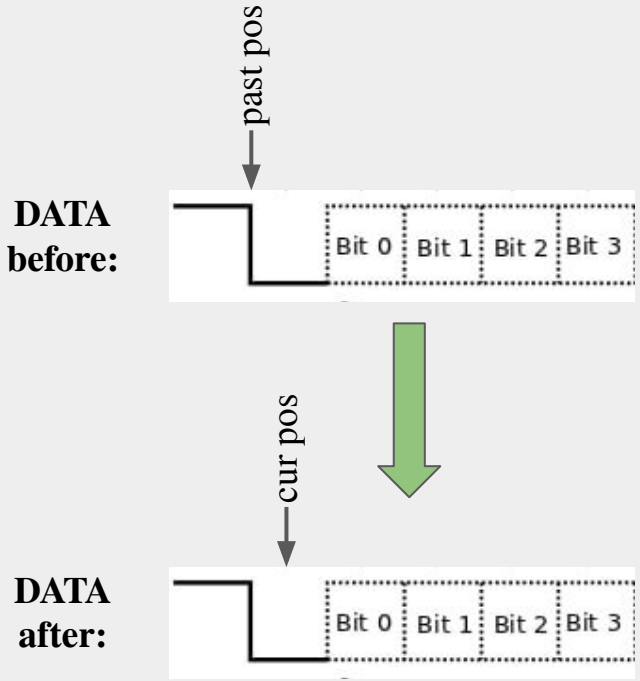
DATA:



Создание модуля UART receiver: State START

/uart_reciver.sv

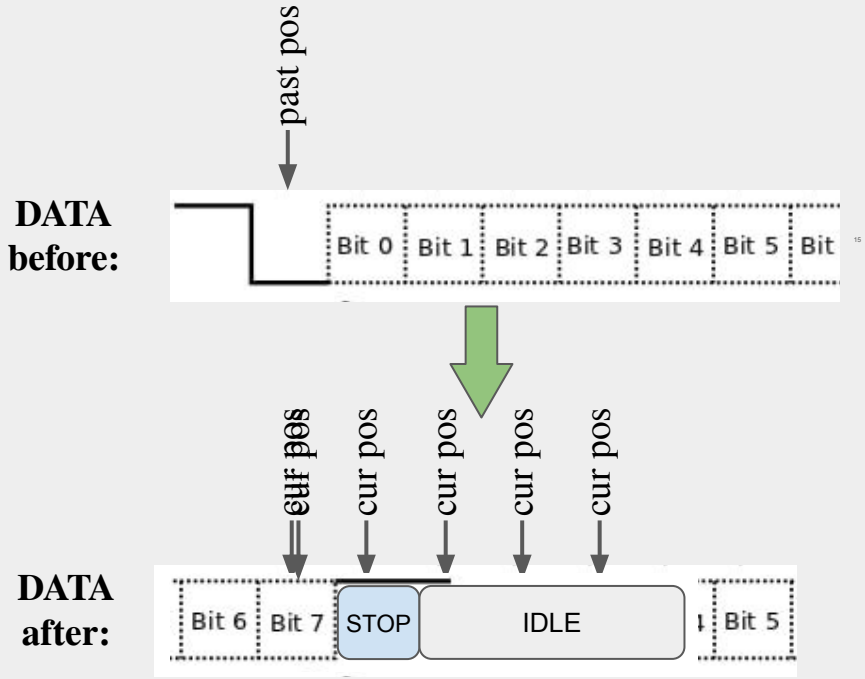
```
…   ...
39
40      START: begin
41        counter <= counter + 1;
42        if ((2 * counter) >= clock_cycles_in_bit)
43        begin  // First shift is a half values
44          counter <= 0;
45          state <= DATA;
46        end
47      end
48
…   ...
```



Создание модуля UART receiver: State DATA

/uart_reciver.sv

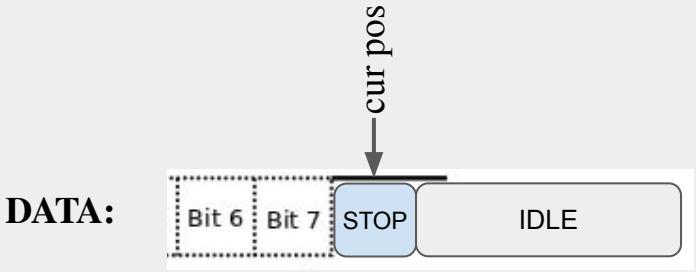
```
…  ...
46
47     DATA: begin
48       counter <= counter + 1;
49       if (counter == clock_cycles_in_bit) begin
50         counter <= 0;
51         bit_count <= bit_count + 1;
52         byte_data <= {rx, byte_data[7:1]}; // LSB
53       end
54       if (bit_count == 4'b1000) state <= STOP;
55     end
56
…  ...
```



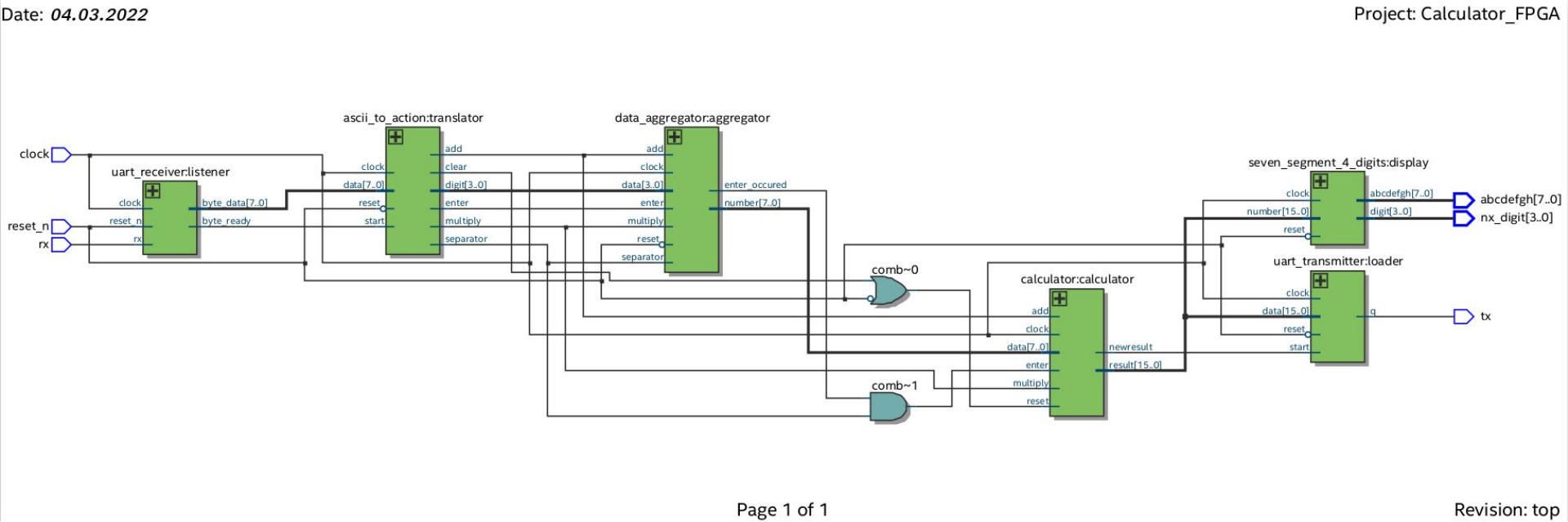
Создание модуля UART receiver: State STOP

/uart_reciver.sv

```
…
      case (state)
        IDLE:  begin … end
        START: begin … end
        DATA:  begin … end
…
56
57        STOP: begin
58          state <= IDLE;
59          byte_ready <= 1;
60        end
61
62        default: state <= IDLE;
63      endcase
64    end
  end
endmodule
```



RTL viewer of project (uart_transmitter):



Создание модуля UART transmitter

/uart_tx.sv

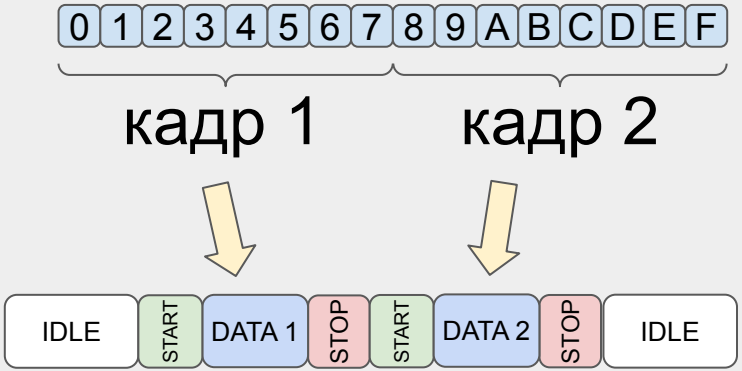
```
1  module two_bytes_uart_tx (
2      input         clock,
3      input         start,
4      input         reset,
5      input [15:0] data,
6
7      output reg q,
8      output     busy
9  );
10   parameter clock_frequency = 50000000;
11   parameter baud_rate       = 9600;
12   parameter clock_cycles_in_bit = clock_frequency /
     baud_rate;
13
14   reg [12:0] cnt;
15   reg [3:0]  bit_num;
16
17   wire bit_start = (cnt == clock_cycles_in_bit);
18   wire idle = (bit_num == 4'hF);
19   assign busy = ~idle;
20
21   reg byte_state;
22   wire [7:0] word = byte_state ? data[7:0]:data[15:8];
```

```
23   always @(posedge clock) begin
24     if (reset) cnt <= 13'b0;
25     else if (start && idle) cnt <= 13'b0;
26     else if (bit_start) cnt <= 13'b0;
27     else cnt <= cnt + 13'b1;
28   end
29
30   always @(posedge clock) begin
31     if (reset) begin
32       bit_num    <= 4'hf;
33       byte_state <= 1'b0;
34       q <= 1'b1;
35     end else if (start && idle) begin
36       bit_num    <= 4'h0;
37       byte_stat <= 1'b0;
38       q <= 1'b1;
39     end else if (bit_start) begin
40       case (bit_num)
…          ...
57       endcase
58     end
59   end
60 endmodule
```

Создание модуля UART transmitter

/uart_tx.sv

```
40    case (bit_num)
41      4'h0: begin bit_num <= 4'h1; q <= 1'b0;    end // start
42      4'h1: begin bit_num <= 4'h2; q <= word[0]; end
43      4'h2: begin bit_num <= 4'h3; q <= word[1]; end
44      4'h3: begin bit_num <= 4'h4; q <= word[2]; end
45      4'h4: begin bit_num <= 4'h5; q <= word[3]; end
46      4'h5: begin bit_num <= 4'h6; q <= word[4]; end
47      4'h6: begin bit_num <= 4'h7; q <= word[5]; end
48      4'h7: begin bit_num <= 4'h8; q <= word[6]; end
49      4'h8: begin bit_num <= 4'h9; q <= word[7]; end
50      4'h9: begin bit_num <= 4'ha; q <= 1'b1;        // finish
51              if (!byte_state) begin
52                byte_state <= 1'b1;
53                bit_num <= 4'h0;
54              end
55            end
56      default: begin bit_num <= 4'hF; end           // Stop
57    endcase
```

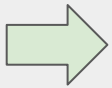


Модуль: ascii_to_action

/ascii_to_action.sv

```
case (data)
        'h30: digit <= 'h0;
        'h31: digit <= 'h1;
        'h32: digit <= 'h2;
        'h33: digit <= 'h3;

        ...

        'h64: digit <= 'hd;
        'h65: digit <= 'he;
        'h66: digit <= 'hf;

        'h2a: multiply  <= 1'b1; // *
        'h2b: add   <= 1'b1;     // +
        'h0a: separator <= 1'b1; // LF
        'h0d: separator <= 1'b1; // CR
        'h20: separator <= 1'b1; // SPACE
        'h52: clear <= 1'b1;     //R (Reset)
        'h72: clear <= 1'b1;     //r (Reset)

        default: error  <= 1'b1;
endcase
```

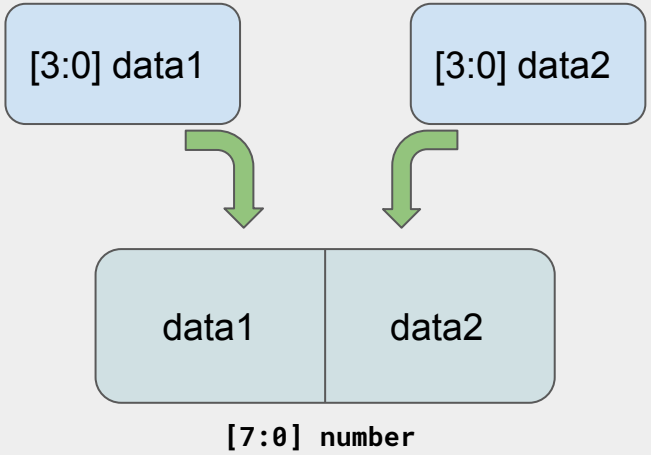
ASCII: *Action:*

0x2B  **‘+’**

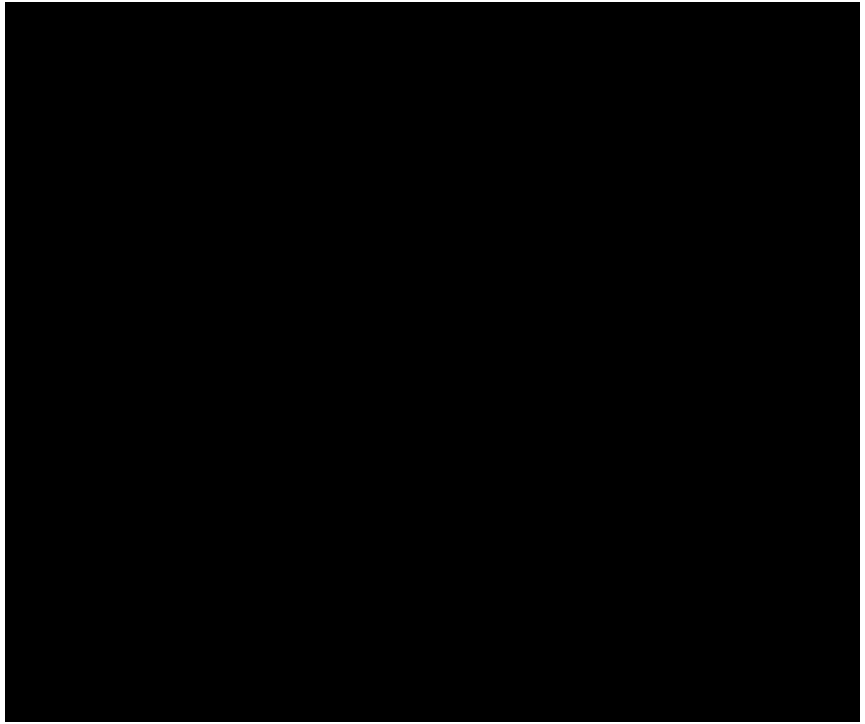
Модуль: data_aggregator

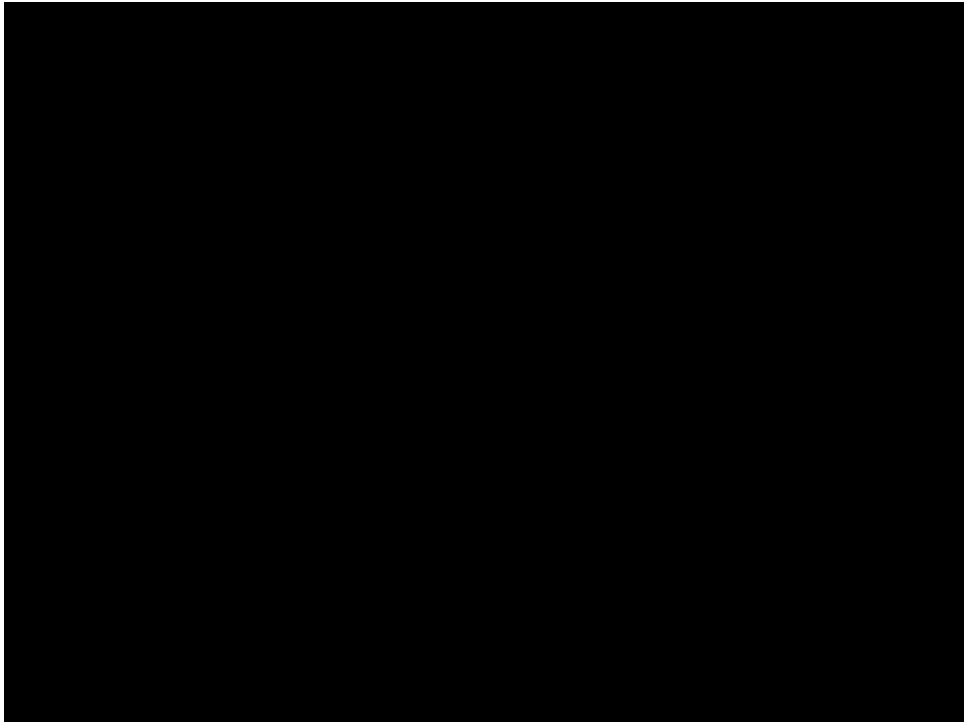
/data_aggregator.sv

```
1  module data_aggregator (
2      input       clock,
3      input       reset,
4      input       enter,
5      input       add,
6      input       multiply,
7      input       separator,
8      input [3:0] data,
9
10     output reg [7:0] number,
11     output reg enter_occured
12 );
13
14   always @(posedge clock) begin
15     if (reset) begin
16       number <= 8'b0;
17       enter_occured <= 1'b0;
18     end else if (enter) begin
19       number <= {number[3:0], data};
20       enter_occured <= 1'b1;
21     end else if (add || multiply) begin
22       enter_occured <= 1'b0;
23     end else if (separator) begin
24       number <= 8'b0;
25     end
26   end
27 endmodule
```



Демонстрация работы проекта:





Инструкция к Задаче №1,2,3 для Linux:

1. Подключить устройство и проверить, что оно определяется правильно, иначе установить CH340.

```
vorart@vorart-T90-TB:~$ ls -la /dev/ttyU*
crw-rw---- 1 root dialout 188, 0 Mar  4 23:52 /dev/ttyUSB0
vorart@vorart-T90-TB:~$
```

2. Установить CuteCom:

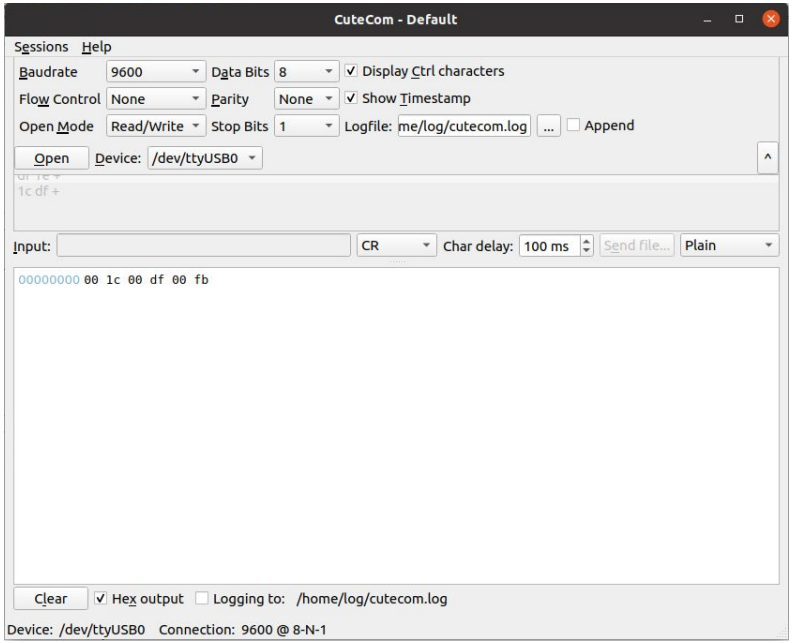
```
$ sudo apt install cutecom
```

3. Запустить программу:

```
$ cutecom
```

4. Выставить настройки для отправки:

Boundrate: 9600
Data bits: 8
Остальные по умолчанию



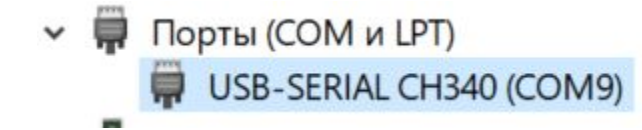
5. Выбрать нужное устройство (Device)
6. В поле Input выбрать CR (После отправки идёт бит с возвратом каретки).
7. В самом низу окна установить флаг Hex output
8. Нажать на кнопку Open для открытия канала связи.

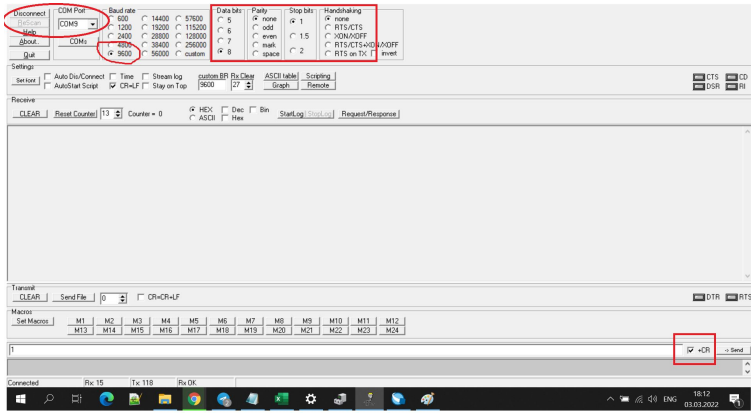
Инструкция к задаче №1,2,3 для Windows:

1. Установить программу Terminal из: https://github.com/DigitalDesignSchool/ce2020labs/tree/master/day_8

2. Подключить кабель в USB разъем пк/ноутбука и посмотреть в диспетчере устройств определился ли виртуальный ComPort подлюченного преобразователя.
   Если же ОС не определила его, то необходимо установить драйвер для микросхемы CH340 из: https://github.com/DigitalDesignSchool/ce2020labs/tree/master/day_8 После чего, должен виртуальный ComPort определиться в ОС.

3. Запускаем программу Terminal, устанавливаем в ней номер COM (в нашем случае 9).



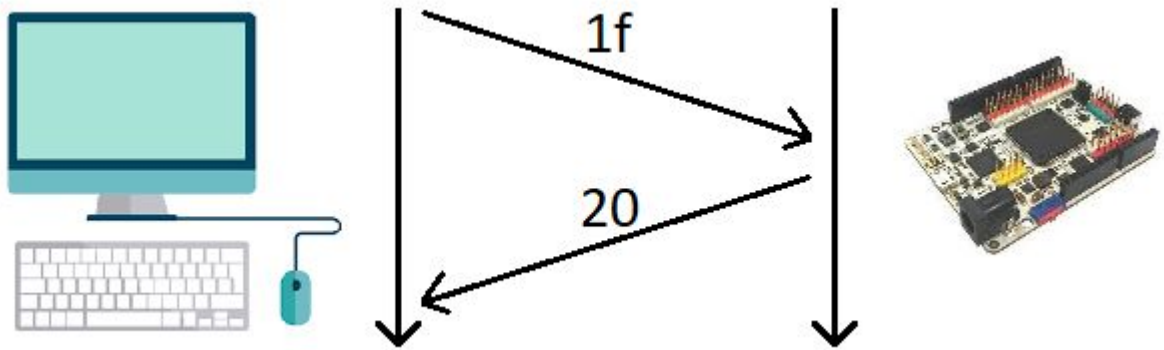


Задача 1 (20 минут)

Через интерфейс UART (9600/8-N-1) сделать обмен сообщениями с FPGA.

Описание: На компьютере вводят число, которое отправляется на fpga, тот прибавляет 1 к числу и отправляет обратно.

Пример:



Обратная польская нотация

Форма записи выражений, в которой операнды расположены перед знаками операций, по другому данный вид записи называют постфиксным.

- 2 3 + => 2 + 3
- 3 2 1 + × => 3 × (2 + 1)
- 2 1 + 3 × => (2 + 1) × 3
- 1 5 2 + 4 × + 3 - => 1 + (5 + 2) × 4 - 3

Преимущества и недостатки

+ Любая формула может быть выражена без скобок, из-за этого постфиксная нотация короче инфиксной;
+ Удобна для вычисления формул в машинах со стеком;
+ Не нужен приоритет операций, в отличие от обычной записи;
  Например, в выражении a+b×c, сначала выполняется умножение, а потом сложение, в обратной польской abc×+ читается однозначно;
- Не удобна в использовании для человека, так как операнды могут располагаться на большом расстоянии: abc×de+e-++ => a + b×c + d+e - e;
- Практически невозможно оптимизировать выражения, так как для работы доступна только верхушка стека;

Расчет выражения используя стек

Обработка входного символа:

a. Если на вход подан операнд, он помещается на вершину стека.
b. Если на вход подан знак операции, то она применяется к последним двум элементам, извлечённым из стека, взятых в порядке добавления. Затем результат выполненной операции кладётся на вершину стека.

Результат вычисления выражения будет лежать на вершине стека.

Пример: 2 3 2 × 5 10 + 10 - + +

На входе операнд, поэтому он помещается на вершину стека.

Пример: 2 3 2 × 5 10 + 10 - + +

На входе операнд, поэтому он помещается на вершину стека.

Пример: 2 3 2 × 5 10 + 10 - + +

На входе операнд, поэтому он помещается на вершину стека.

Пример: 2 3 2 × 5 10 + 10 - + +

На входе оператор, который применяется к двум верхним элементам стека, а результат кладется обратно

$3 \times 2 = 6$

Пример: 2 3 2 × 5 10 + 10 - + +

На входе операнд, поэтому он помещается на вершину стека.

Пример: 2 3 2 × 5 10 + 10 - + +

На входе операнд, поэтому он помещается на вершину стека.

Пример: 2 3 2 × 5 10 ± 10 - + +

На входе оператор, который применяется к двум верхним элементам стека, а результат кладется обратно

5 + 10 = 15

Пример: 2 3 2 × 5 10 + 10 - + +

На входе операнд, поэтому он помещается на вершину стека.

Пример: 2 3 2 × 5 10 + 10 - + +

На входе оператор, который применяется к двум верхним элементам стека, а результат кладется обратно

15 - 10 = 5

Пример: 2 3 2 × 5 10 + 10 - ± +

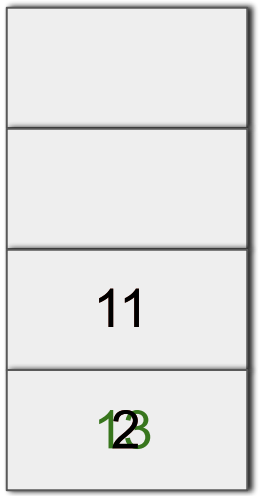
На входе оператор, который применяется к двум верхним элементам стека, а результат кладется обратно

6 + 5 = 11

Пример: 2 3 2 × 5 10 + 10 - + ±

На входе оператор, который применяется к двум верхним элементам стека, а результат кладется обратно

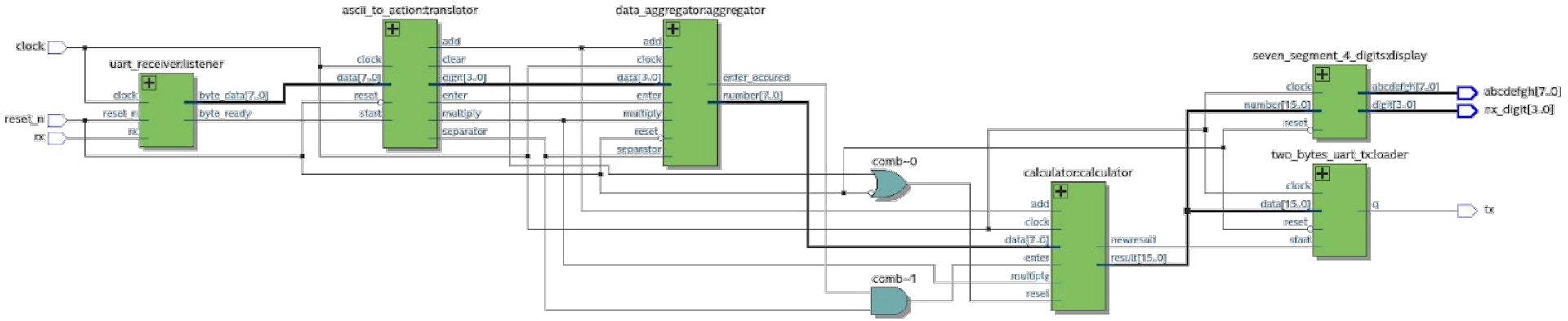
2 + 11 = 13



Ответ: 13

Реализация стекового калькулятора

RTL viewer of project: Calculator



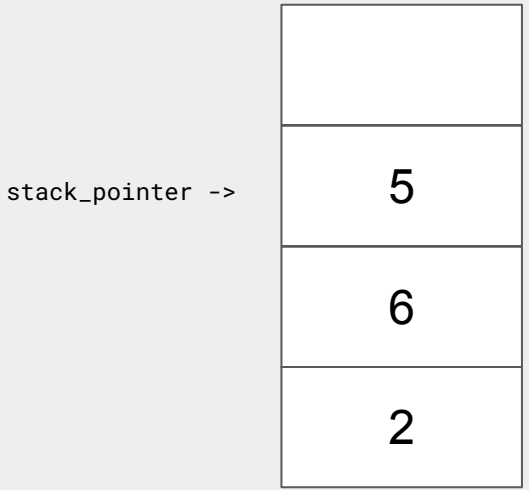
Код для плат предоставлен:
https://github.com/DigitalDesignSchool/2021dev/tree/main/Innopolis%20University/fpga_calculator_with_uart

Создание модуля ALU

```
1  module alu
2  (
3      input  [15:0] a,
4      input  [15:0] b,
5      input         multiply,
6      output [15:0] result,
7      output        overflow
8  );
9
10     wire [16:0] result_add = a + b;
11     wire [31:0] result_mul = a * b;
12
13     assign result   = multiply ?   result_mul [15: 0] : result_add [15:0];
14     assign overflow = multiply ? | result_mul [31:16] : result_add [16];
15
16 endmodule
17
```

Создание модуля stack

```
1  module stack
2  (
3      input  clock,
4      input  reset,
5      input  push,
6      input  pop,
7
8      input  [`word_width - 1:0] write_data,
9      output [`word_width - 1:0] read_data
10 );
11     reg [`word_width - 1:0] stack [0:`stack_size - 1];
12     reg [`stack_pointer_size - 1:0] stack_pointer;
13     assign read_data = stack [stack_pointer];
```

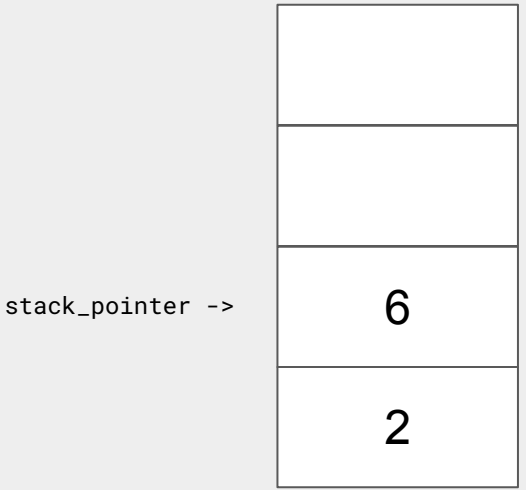


push - добавить элемент в стек
pop - удалить элемент из стека

stack pointer - указатель на верхний элемент

Создание модуля stack

```
14      integer i;
15      always @(posedge clock) begin
16          if (reset) begin
17              stack_pointer <= 0;
18              for (i = 0; i < `stack_size; i = i + 1)
19                  stack [i] <= 0;
20          end else
21          if (push) begin
22              if (stack_pointer == `stack_size - 1) begin
23                  stack [0] <= write_data;
24                  stack_pointer <= 0;
25              end else begin
26                  stack [stack_pointer + 1] <= write_data;
27                  stack_pointer <= stack_pointer + 1;
28              end
29          end else
30          if (pop) begin
31              stack [stack_pointer] <= 0;
32              if (stack_pointer == 0)
33                  stack_pointer <= `stack_size - 1;
34              else
35                  stack_pointer <= stack_pointer - 1;
36          end
37      end
38  endmodule
```



Создание модуля калькулятора

```
1  module calculator
2  (
3      input         clock,
4      input         reset,
5      input         enter,
6      input         add,
7      input         multiply,
8      input  [ 7:0] data,
9      output        newresult,
10     output [15:0] result,
11     output        overflow,
12     output [ 3:0] error
13 );
14     assign error = 0;
```

```
15     reg  [15:0] alu_a;
16     reg  [15:0] alu_b;
17     reg         alu_multiply;
18     wire [15:0] alu_result;
19     wire        alu_overflow;
20
21     alu alu
22     (
23         .a          ( alu_a        ),
24         .b          ( alu_b        ),
25         .multiply   ( alu_multiply ),
26         .result     ( alu_result   ),
27         .overflow   ( alu_overflow )
28     );
```

```
41     reg         stack_push;
42     reg         stack_pop;
43     reg  [15:0] stack_write_data;
44     wire [15:0] stack_read_data;
45
46     stack stack
47     (
48         .clock      ( clock            ),
49         .reset      ( reset            ),
50         .push       ( stack_push       ),
51         .pop        ( stack_pop        ),
52         .write_data ( stack_write_data ),
53         .read_data  ( stack_read_data  )
54     );
```

enter - сигнал ввода числа

add - сигнал сложения элементов

multiply - сигнал умножения элементов

Создание модуля калькулятора

```
63      reg [15:0] oldresult = 0;
64      assign result    = stack_read_data;
65      assign newresult = (result != oldresult) && state == 0;
66
67      always @(*)
68      begin
69          alu_a            = r_alu_a;
70          alu_b            = r_alu_b;
71          alu_multiply     = r_alu_multiply;
72          stack_push       = 0;
73          stack_pop        = 0;
74          stack_write_data = data;
75          next_state       = state;
76
77          case (state)
78          0:
79             if (enter) begin
80                 stack_push       = 1;
81                 stack_write_data = data;
82             end else
83             if (add | multiply) begin
84                 alu_a            = stack_read_data;
85                 alu_multiply     = multiply;
86
87                 stack_pop        = 1;
88                 next_state       = 1;
89             end
```

```
90          1:
91             begin
92                 alu_b            = stack_read_data;
93                 stack_pop        = 1;
94                 next_state       = 2;
95             end
96
97          2:
98             begin
99                 stack_push       = 1;
100                stack_write_data = alu_result;
101                next_state       = 0;
102            end
103         endcase
104     end
```

Машина состояний для работы со стеком

Создание модуля калькулятора

```
105     always @(posedge clock)
106     begin
107         if (reset)
108         begin
109             r_alu_a        <= 0;
110             r_alu_b        <= 0;
111             r_alu_multiply <= 0;
112             state          <= 0;
113             oldresult      <= 0;
 14         end else begin
115             r_alu_a        <= alu_a;
116             r_alu_b        <= alu_b;
117             r_alu_multiply <= alu_multiply;
118             state          <= next_state;
119            oldresult      <= result;
120         end
121     end
122
123 endmodule
```

По положительному фронту клока обновляем все регистры

Задача 2 (25 минут)

Добавить в калькулятор новый оператор вычитания по модулю “-”.

Описание: Для добавления нового оператора нужно модифицировать следующие модули: alu, calculator, ascii_to_action, top.

Пример:

input: 1 5 2 + 4 * + 3 -
output: 1A

input: 5 2 * 5 3 * -
output: 5

Применение UART для отладки кода HDL

Метод отладки	Микроконтроллеры	ПЛИС
вывод требуемых значений на отображающее устройство платы	+	+
пошаговая отладка кода в IDE	+	+ (Microsemi)
онлайн мониторинг значений в IDE	+	-
мониторинг состояний пинов осциллографе, логическом анализаторе	+	+

Задача 3 (30 минут)

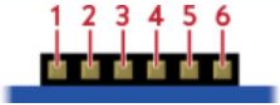
Считать значение с энкодера платы Pmod ENC и вывести считанное значение на семисегментный индикатор и на ComPort компьютера.

1. Скачать файл rotary_encoder.v из https://github.com/DigitalDesignSchool/ce2020labs/tree/master/day_8/common

2. Подключить плату Pmod ENC к плате с ПЛИС, использовать выводы 1,2,5,6.

   На вывод 6 подать 3.3V с платы ПЛИС.

J1 Pinout



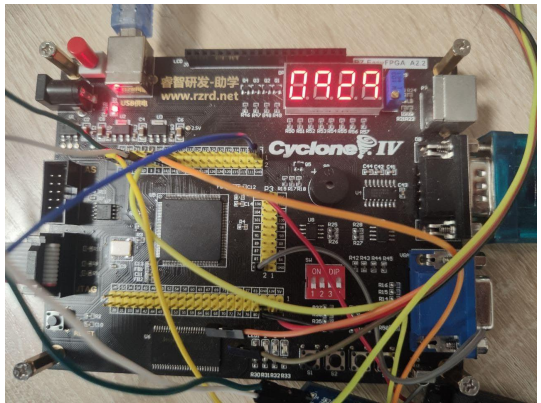
Pin 1	A
Pin 2	B
Pin 3	BTN
Pin 4	SWT
Pin 5	GND
Pin 6	VCC

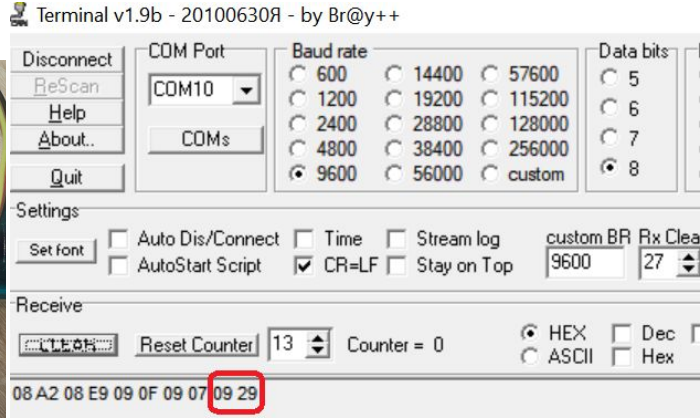
Выполнение задачи №3

3. В Pin Planner среды разработки Quartus назначить номера пинов согласно подключения пинов 1,2 платы Pmod ENC на плату с ПЛИС.

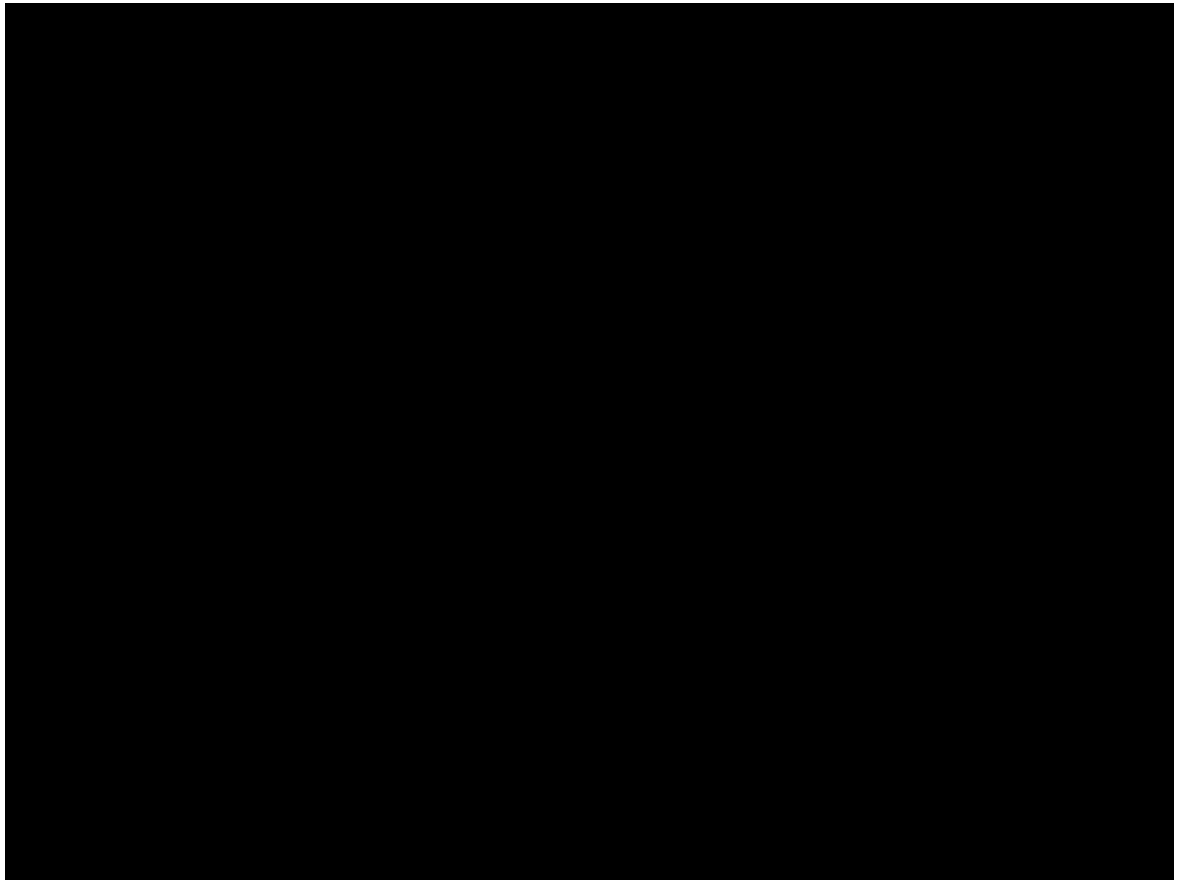
4. Синтезировать измененный проект и прошить плату с ПЛИС.

Результат выполнения Задачи **№3**:
Вращая энкодер,выводится значение с него на семисегментный индикатор и в ComPort ПК.





Выполнение задачи №3



Выполнение задачи №3

```
1
2   module top (
3       input clock,
4       input reset_n,
5       input rx,
6        input encoder_a,
7        input encoder_b,
8
9       output reg       tx,
10      output reg [7:0] abcdefgh,
11      output reg [3:0] nx_digit
12  );
13
14    wire byte_ready;
15    wire [7:0] ascii_data;
16
17    wire [15:0] encoder_value;
18    wire [15:0] prev_encoder_value;
19
20
21   rotary_encoder encoder
22  (
23      .clk        (clock),
24      .reset   (~reset_n),
25      .a       (encoder_a),
26      .b       (encoder_b),
27      .value   (encoder_value)
28  );
29
```

```
57
58    // Prepare and accumulate data
59
60    reg [7:0] number;
61    reg enter_occured;
62
63    always @(posedge clock) begin
64      if (!reset_n) begin
65        number <= 8'b0;
66        enter_occured <= 1'b0;
67      end else if (enter) begin
68        number <= {number[3:0], digit};
69        enter_occured <= 1'b1;
70      end else if (add || multiply || error_ascii) begin
71        enter_occured <= 1'b0;
72      end else if (separator) begin
73        number <= 8'b0;
74      end
75      prev_encoder_value <= encoder_value;
76    end
77
```

```
107
108   two_bytes_uart_tx loader (
109       .clock(clock),
110       .reset(~reset_n),
111       .start( (prev_encoder_value != encoder_value) ),// 1'b1
112       .data (encoder_value),// encoder_value
113
114       .q(tx)
115   );
116
117
```

Дизайн-центр Электроники и Микроэлектроники на базе Университета Иннополис