1) Чем отличается task от function?

Характеристика	Task	Function
Задержки (wait, #, clk)	Может содержать	Выполняется мгновенно, задержки нельзя использовать
Возврат значения	Не имеет возвращаемого значения (для возвращаемых значений использует output аргументы)	Всегда возвращает значение (или void)
Использование в выражениях	Используется как самостоятельная конструкция (процедура)	Может использоваться в выражениях
Вызов других элементов	Может внутри себя содержать другие function и task	Может содержать только другие function

2) Предположим, что в FPGA используются LUT таблицы 4 входа на 1 выход. Нарисуйте, в какую схему синтезируется следующий модуль:

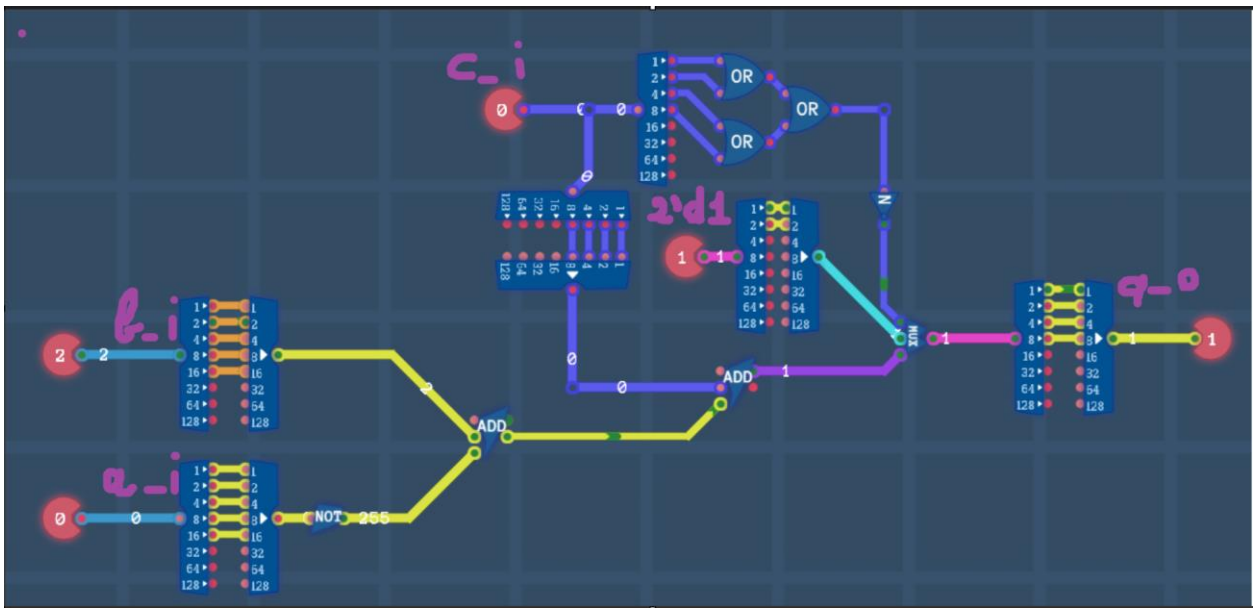
```
module mega_module (
  input [4:0]        a_i,
  input [4:0]        b_i,
  input [3:0]        c_i,

  output logic [3:0] q_o
);
logic compare;

assign compare = a_i < b_i;

always_comb
  begin
    if( c_i == 0 )
      q_o = 2'd1;
    else
      q_o = c_i + compare;
  end

endmodule
```



1) a_i < b_i ⇔
   (a_4 < b_4) ||
   (a_4 == b_4 && a_3 < b_3) ||
   (a_4 == b_4 && a_3 == b_3 && a_2 < b_2) ||
   (a_4 == b_4 && a_3 == b_3 && a_2 == b_2 && a_1 < b_1) ||
   (a_4 == b_4 && a_3 == b_3 && a_2 == b_2 && a_1 == b_1 a_0 < b_0)

   Обозначим LUT для ==(eq_{y}) и <(less_{y}) для каждого бита
   LUT: less_{y}

a_{y}	b_{y}	X	x	q_o
0	0	X	x	0
0	1	X	X	1
1	0	X	X	0
1	1	x	X	0

   LUT: eq_{y}

a_{y}	b_{y}	X	x	q_o
0	0	X	x	1
0	1	X	X	0
1	0	X	X	0
1	1	x	X	1

⇨ a_i < b_i ⇔
less_4 ||
(eq_4 && less_3) ||
(eq_4 && eq_3 && less_2) ||
(eq_4 && eq_3 && eq_2 && less_1) ||
(eq_4 && eq_3 && eq_2 && eq_1 && less_0)

LUT для (less_4 || (eq_4 && less_3)) {comp_0}:

less_4	eq_4	less_3	X	q_o
0	0	0	X	0
0	0	1	X	0
0	1	0	X	0
0	1	1	X	1
1	0	0	X	1
1	0	1	X	1
1	1	0	X	1
1	1	1	X	1

LUT для (eq_4 && eq_3 && less_2) {comp_1}:

eq_4	eq_3	less_2	X	q_o
0	0	0	X	0
0	0	1	X	0
0	1	0	X	0
0	1	1	X	0
1	0	0	X	0
1	0	1	X	0
1	1	0	X	0
1	1	1	X	1

LUT для (eq_4 && eq_3 && eq_2 && less_1) {comp_2}

eq_4	eq_3	less_2	less_1	q_o
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	0	0	0	0
1	0	0	1	0
1	1	1	0	0
1	1	1	1	1

LUT для (eq_4 && eq_3 && eq_2 && eq_1) {comp_3}

eq_4	eq_3	eq_2	eq_1	q_o
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	0	0	0	0
1	0	0	1	0
1	1	1	0	0
1	1	1	1	1

LUT для (eq_4 && eq_3 && eq_2 && eq_1 && less_0) ⇔ (comp_3 && less_0) {comp_4}

comp_3	less_0	x	X	q_o
0	0	X	x	0
0	1	X	X	0
1	0	X	X	0
1	1	x	X	1

LUT для

(less_4 || (eq_4 && less_3) ||

(eq_4 && eq_3 && less_2) ||

(eq_4 && eq_3 && eq_2 && less_1) ||

(eq_4 && eq_3 && eq_2 && eq_1 && less_0)

⇔

comp_0 || comp_1 || comp_2 || comp_4 => comp_a_b

comp_0	comp_1	comp_2	comp_4	q_o
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	0	0	0	1
1	0	0	1	1
1	1	1	0	1
1	1	1	1	1

2) LUT для c_i == 0 {is_zero}

c_0	c_1	c_2	c_4	q_o
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	0	0	0	0
1	0	0	1	0
1	1	1	0	0
1	1	1	1	0

3) LUT для c_i + compare ⇔

q_0=c_0⊕compare

q_1=c_1⊕(c_0&compare)

q_2=c_2⊕(c_1&c_0&compare)

q_3=c_3⊕(c_2&c_1&c_0&compare)

LUT для (c_0⊕compare) {q_0_LUT}

c_0	comp_a_b	x	X	q_o
0	0	X	x	0
0	1	X	X	1
1	0	X	X	1
1	1	x	X	0

LUT для (c_1⊕(c0&compare)) {q_1_LUT}

c_1	c0	comp_a_b	X	q_o
0	0	0	X	0
0	0	1	X	0
0	1	0	X	0
0	1	1	X	1
1	0	0	X	1
1	0	1	X	1
1	1	0	X	1
1	1	1	X	0

LUT для (c_2⊕(c_1&c_0&compare)) {q_2_LUT}

c_2	c_1	c_0	comp_a_b	q_o
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	0	0	0	1
1	0	0	1	1
1	1	1	0	1
1	1	1	1	0

LUT для (c_2&c_1&c_0&compare) {q_3_tmp_LUT}

c_2	c_1	c_0	comp_a_b	q_o
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	0	0	0	0
1	0	0	1	0
1	1	1	0	0
1	1	1	1	1

LUT для c_3⊕(c_2&c_1&c_0&compare) ⇔ c_3⊕ q_3_tmp_LUT : {q_3_LUT}

c_3	q_3_tmp_LUT	x	X	q_o
0	0	X	x	0
0	1	X	X	1
1	0	X	X	1
1	1	x	X	0

4) LUT (мультиплексор) для

```
if (c_i == 0)
   q_o = 2'd1
else
   q_o = c_i + compare
```

⇔

```
if (is_zero)
   q_o_i = {0001}
else
   q_o_i = q_i_LUT
```

LUT для q_o_0

is_zero	q_0_LUT	x	X	q_o_0
0	0	X	x	0
0	1	X	X	1
1	0	X	X	1
1	1	x	X	1

LUT для q_o_1

is_zero	q_1_LUT	x	X	q_o_1
0	0	X	x	0
0	1	X	X	1
1	0	X	X	0
1	1	x	X	0

LUT для q_o_2

is_zero	q_2_LUT	x	X	q_o_2
0	0	X	x	0
0	1	X	X	1
1	0	X	X	0
1	1	x	X	0

LUT для q_o_3

is_zero	q_3_LUT	x	X	q_o_3
0	0	X	x	0
0	1	X	X	1
1	0	X	X	0
1	1	x	X	0

3: Заполните таблицу значений a, b, c, d по окончанию event regions Active и NBA:

	Active	**NBA**
a	7	0
b	19	12
c	0	19
d	12	12

4) Какой будет порядок вывода сообщений в терминал? Поясните почему именно такой:

```
module tb;

bit clk;

always #5 clk <= !clk;

task print0();
  $display("print0");
  @( posedge clk );
  $display("one more print0");
endtask

task print1();
  $display("print1");
  @( posedge clk );
  $display("one more print1");
endtask

task print2();
  $display("print2");
endtask

initial
  begin
    @( posedge clk );
    fork
      print0();
      print1();
    join_none
    print2();
  end

endmodule
```

t=5	или	или	Или…
print0	print1	print2	
print1	print0	print0	
print2	print2	print1	

t=15	или
one more print0	one more print1
one more print1	one more print0

Поскольку все три $display выполняются в разных потоках, стандарт точно не определяет порядок их выполнения

5) На пин FPGA приходит сигнал, который меняется редко (частота изменения меньше 10 Гц). Какая схема внутри FPGA необходима, чтобы начать работать с этим сигналом на системной частоте FPGA (150 МГц)?

Можно использовать двухтактный синхронизатор на D-триггерах

6. Какая ошибка допущена в описании FSM и как её исправить:

```
module some_state_machine (
  input  clk_i,
  input  srst_i,

  input  en_i,
  input  done_i,

  output is_waiting_o
);

enum logic [1:0] {
  IDLE_S = 0,
  WORK_S = 1,
  WAIT_S = 2
} state, next_state;

always_ff @( posedge clk_i )
  if( srst_i )
    state <= IDLE_S;
  else
    state <= next_state;

always_comb
  begin
    case( state )
      IDLE_S :
        begin
          if( en_i)
            next_state = WORK_S;
        end
      WORK_S :
        begin
          if( done_i )
            next_state = WAIT_S;
        end
      WAIT_S : next_state = IDLE_S;
    endcase
  end

assign is_waiting_o = ( state == WAIT_S );

endmodule
```

В always_comb блоке для next_state определены не все ветви переходов. Например, когда state = IDLE_S и en_i = 0, либо state = WORK_S и done_i = 0. В результате создается latch для хранения

предыдущего значения next_state. Для решения можно задать в начале блока next_state = state; или сделать default: next_state = state;

7. Что такое мультиплексор и демультиплексор?

a. Нарисуйте схему мультиплексора 4 → 1 используя логические элементы И, ИЛИ, НЕ

b. Нарисуйте схему демультиплексора 4 → 1**(опечатка 1->4)** используя логические элементы И, ИЛИ, НЕ

Мультиплексор X->1: выбирает один из X входов и передаёт его на выход, управляемый $\log_2(X)$ управляющими линиями

Демультиплексор 1->X: передаёт значение со входа на один из X выходов, управляемый $\log_2(X)$ управляющими линиями

a) Мультиплексор 4->1

b) Демультиплексор 1->4

8) В чем разница между static и automatic (для function и для task)?

Свойство	**static**	**automatic**
По умолчанию для	function	task
Память	одна на весь модуль	каждый вызов своя копия (stack)
Сохранение значения	да	нет
Поддержка рекурсии	нет	да

9. Чем queue (\[$\]) отличается от mailbox?

Свойство	Queue ($[])	Mailbox
Подходит для	Хранения данных в массиве	Producer/consumer FIFO
Синхронизация между потоками	Нет	Есть (блокирующее чтение/запись)
Использование	Один поток, локальное хранение	Межпроцессная передача данных
Размер	Динамический	Может быть ограничен
Методы доступа	push_back, pop_front, индекс	put/get, try_put/try_get

10. Далее следует код, который содержит описание комбинационной петли (не надо использовать комбинационные петли при работе с FPGA). Нарисуйте RTL схему, которая бы полностью соответствовала поведению, которое описано этим кодом. Можно использовать блоки: мультиплексор, демультиплексор, логическое отрицание

```
logic [1:0] a;
logic [1:0] b;

always_comb
  if( b == 2'd0 )
    a = 2'd1;
  else
    if( b == 2'd1 )
      a = 2'd2;
```