



Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное  
учреждение  
высшего образования  
«Московский государственный технический университет  
имени Н.Э. Баумана  
(национальный исследовательский университет)»  
(МГТУ им. Н.Э. Баумана)

---

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ  
ТЕХНИКА

**О Т Ч Е Т**

**по лабораторной работе № 1**

**Название:** Разработка конвейерных устройств

**Дисциплина:** Основы проектирования ЭВМ

**Вариант 10**

Студент

ИУ6-63Б

(Группа)

\_\_\_\_\_  
(Подпись, дата)

В.К. Залыгин

(И.О. Фамилия)

Преподаватель

\_\_\_\_\_  
(Подпись, дата)

С.В. Ибрагимов

(И.О. Фамилия)

Москва, 2025

**Цель работы:** изучение и применение на практике принципов конвейеризации устройств на примере конвейерного сумматора. Лабораторная работа направлена на закрепление практических навыков проектирования цифровых устройств на языке Verilog HDL. В ходе выполнения лабораторной работы осваиваются методики эффективного использования конструкций языка Verilog для описания цифровых схем, а также выполняется разработка проекта конвейерного сумматора и сравнения производительности различных вариантов реализации многоразрядных сумматоров. Студенты получают практический опыт проектирования на ПЛИС Xilinx Virtex-6 с использованием среды Xilinx ISE 14.7 и отладочной платы Xilinx ML605, выполняют анализ отчётов по результатам синтеза и оптимизацию проекта для эффективного использования ресурсов ПЛИС.

### **Выполнение работы**

#### **Задание 1. Исследование работы сумматора с передачей переноса по цепочке замкнутых ключей (CLA)**

В данном задании необходимо создать простой многоразрядный сумматор на основе примитивного Verilog описания и исследовать результаты его синтеза в САПР Xilinx ISE. В первую очередь, используя команды и код, указанные в методическом указании, была получена синтезированная схема технологического уровня, представленная на рисунке 1. Далее с помощью некоторых команд было открыто окно редактора FPGA Editor, который позволяет детально изучить и модифицировать результаты выполнения размещения и трассировки. Схема сумматора для одного блока SLICEL представлена на рисунке 2.

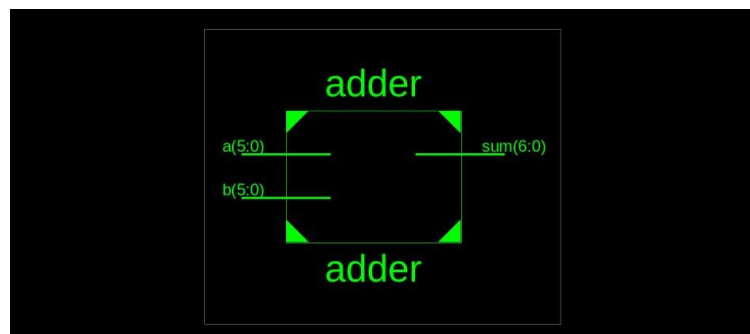


Рисунок 1 – Схема полученного сумматора



связанных с нарушением условий setup/hold, так как сигнал rst не синхронизирован с тактовым сигналом и используется вне основной логики передачи данных. При этом корректная работа сброса обеспечивается за счёт соответствующего использования сброса в описании регистров на Verilog (например, always @(posedge clk or posedge rst)).

Ответ на вопрос «В каких случаях и с помощью какого сигнала происходит сброс модуля cla\_checker?»: сброс модуля cla\_checker осуществляется при поступлении внешнего сигнала rst, который подаётся от кнопки SW10 и имеет высокий активный уровень. Этот сигнал инициирует асинхронный сброс внутренних регистров модуля, таких как счётчики, флаги ошибок и другие состояния. Сброс необходим для инициализации модуля перед началом работы и используется в случае необходимости перезапуска тестирования. Сигнал rst напрямую подключён к входу модуля cla\_checker и обрабатывается в конструкции always @(posedge clk or posedge rst).

Резюмированная информация о выполнении временных ограничений (Score, Timing errors и другие) показана на рисунке 3.

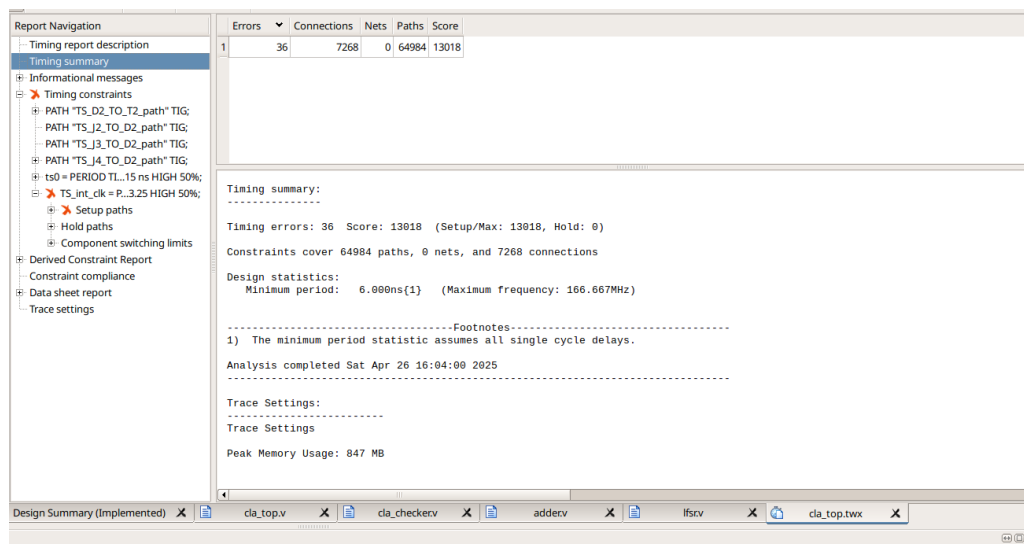


Рисунок 3 - Резюмированная информация о выполнении временных ограничений

Лог сообщений статического временного анализа (Informational messages) показан на рисунке 4.

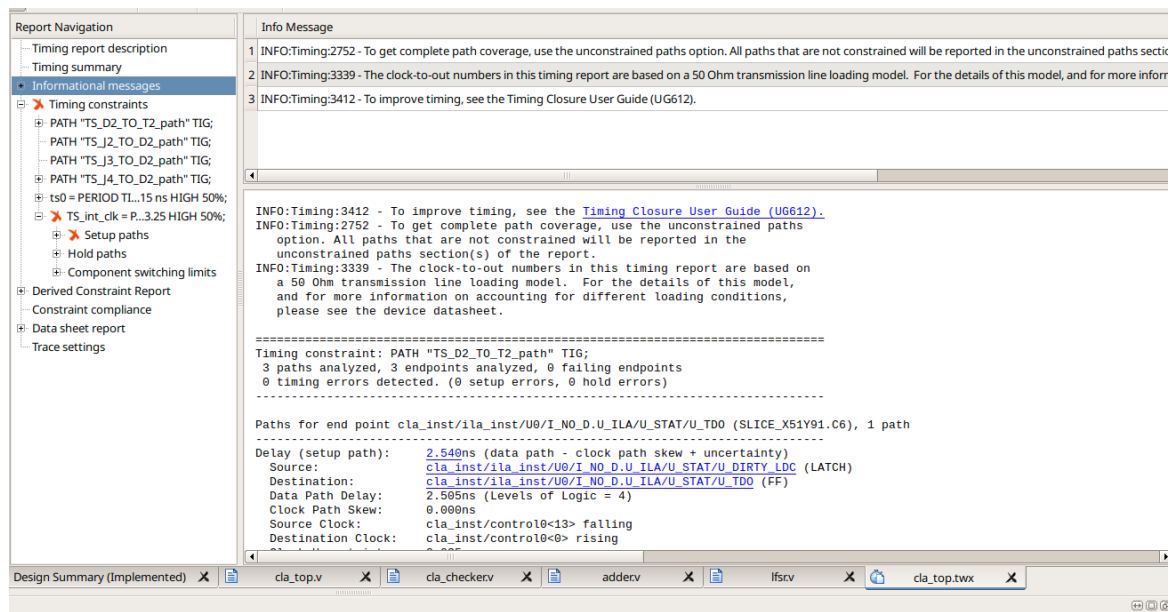


Рисунок 4 - Лог сообщений статического временного анализа

Перечень заданных временных ограничений (Timing Summary) продемонстрирован на рисунке 5.

Report Navigation	Constraint	Paths	Failing Paths	Endpoints	Setup Errors	Hold Errors
Timing report description	1 PATH "TS_D2_TO_T2_path" TIG;	3	0	3	0	0
Timing summary	2 PATH "TS_J2_TO_D2_path" TIG;	0	0	0	0	0
Informational messages	3 PATH "TS_J3_TO_D2_path" TIG;	0	0	0	0	0
Timing constraints	4 PATH "TS_J4_TO_D2_path" TIG;	12	0	2	0	0
PATH "TS_D2_TO_T2_path" TIG;	5 ts0 = PERIOD TIMEGRP "userclk" 15.15 ns HIGH 50%;	0	0	0	0	0
PATH "TS_J2_TO_D2_path" TIG;	6 TS_int_clk = PERIOD TIMEGRP "int_clk" ts0 / 3.25 HIGH 50%;	64969	1372	6566	36	0
PATH "TS_J3_TO_D2_path" TIG;						
PATH "TS_J4_TO_D2_path" TIG;						
ts0 = PERIOD TL...15 ns HIGH 50%;						
TS_int_clk = P...3.25 HIGH 50%;						
Setup paths						
Hold paths						
Component switching limits						
Derived Constraint Report						
Constraint compliance						
Data sheet report						
Trace settings						

Timing constraint: PATH "TS\_D2\_TO\_T2\_path" TIG;  
3 paths analyzed, 3 endpoints analyzed, 0 failing endpoints  
0 timing errors detected. (0 setup errors, 0 hold errors)

Paths for end point cla\_inst/ila\_inst/U0/I\_NO\_D.U\_ILA/U\_STAT/U\_TDO (SLICE\_X51Y91.C6), 1 path

Delay (setup path): 2.540ns (data path - clock path skew + uncertainty)  
Source: cla\_inst/ila\_inst/U0/I\_NO\_D.U\_ILA/U\_STAT/U\_DIRTY\_LDC (LATCH)  
Destination: cla\_inst/ila\_inst/U0/I\_NO\_D.U\_ILA/U\_STAT/U\_TDO (FF)  
Data Path Delay: 2.505ns (Levels of Logic = 4)  
Clock Path Skew: 0.000ns  
Source Clock: cla\_inst/control0<13> falling  
Destination Clock: cla\_inst/control0<0> rising  
Clock Uncertainty: 0.035ns ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE  
Total System Jitter (TSJ): 0.070ns  
Total Input Jitter (TIJ): 0.000ns  
Discrete Jitter (DJ): 0.000ns

Рисунок 5 - Перечень заданных временных ограничений

Перечень критических путей показан на рисунках 6 и 7.

Тип возникшего нарушения показан на рисунке 8.

Перечень сигналов и компонентов, входящий в самую длинную комбинационную цепь (первую из трех), показан на рисунке 9.

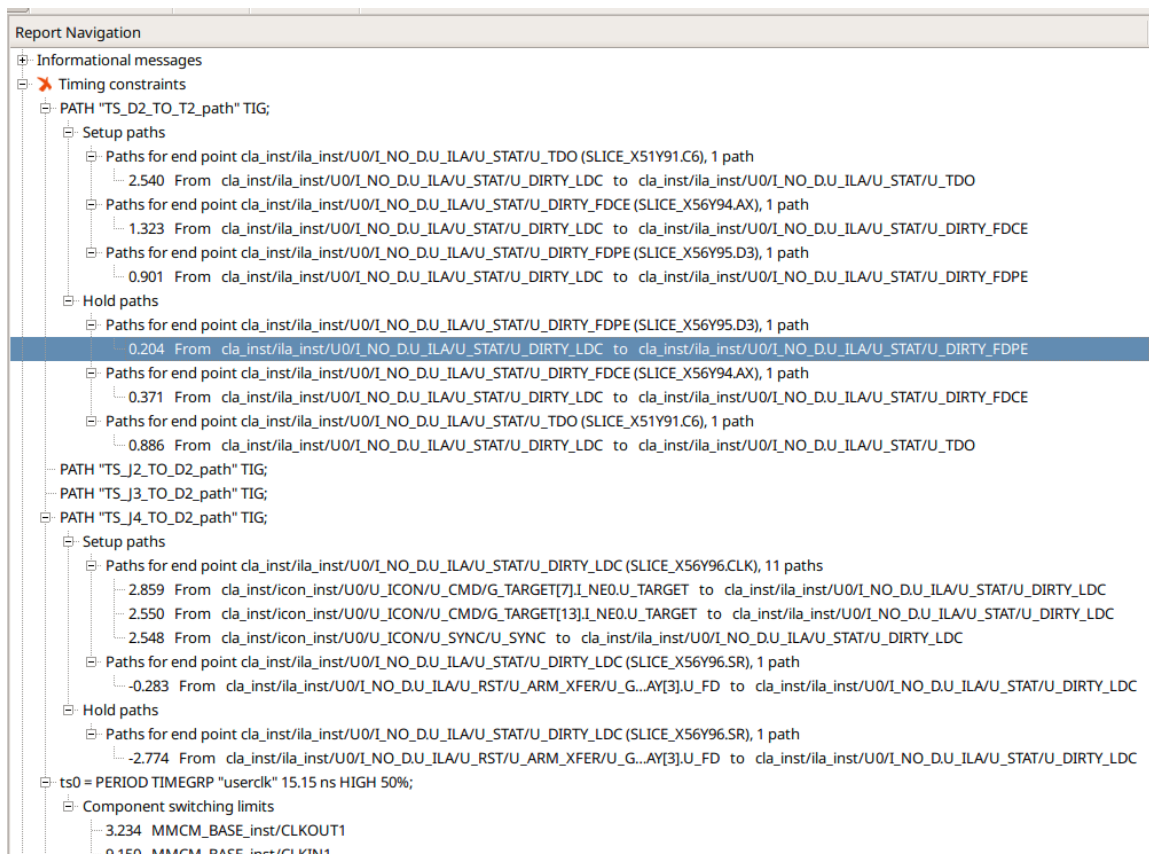


Рисунок 6 – Перечень критических путей

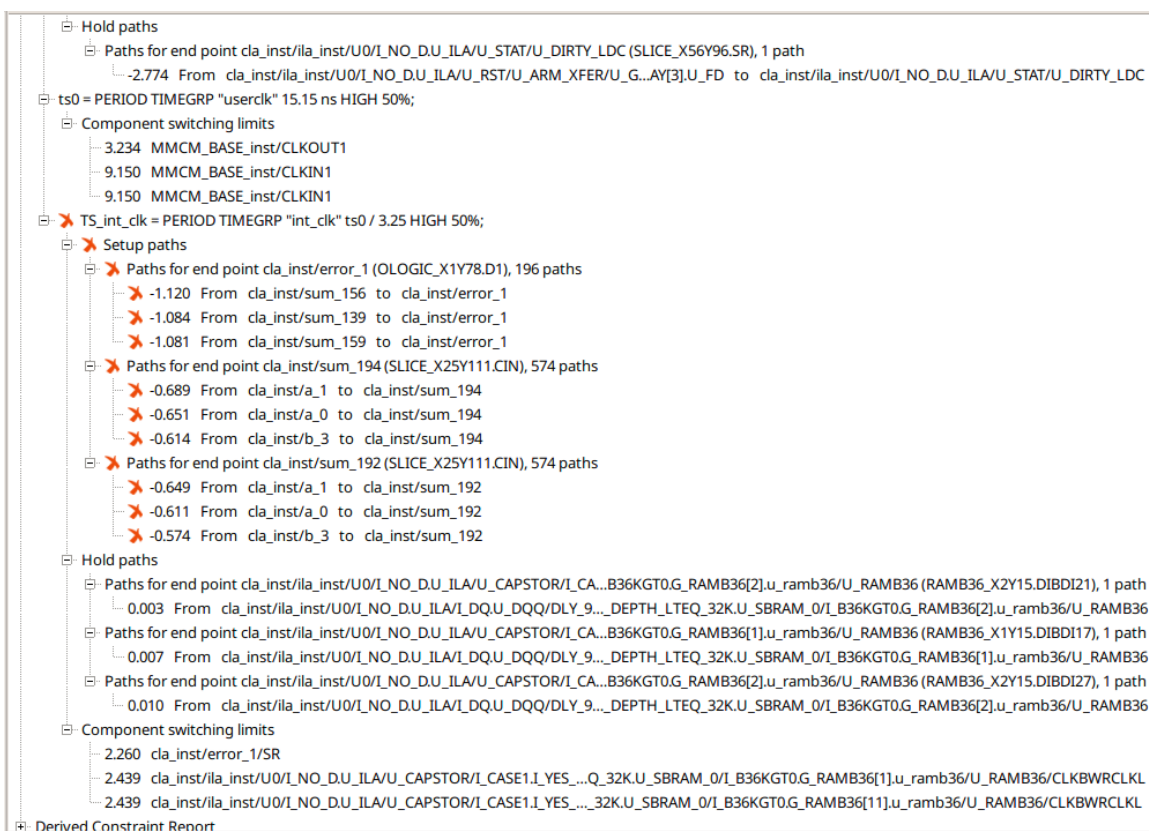


Рисунок 7 - Перечень критических путей



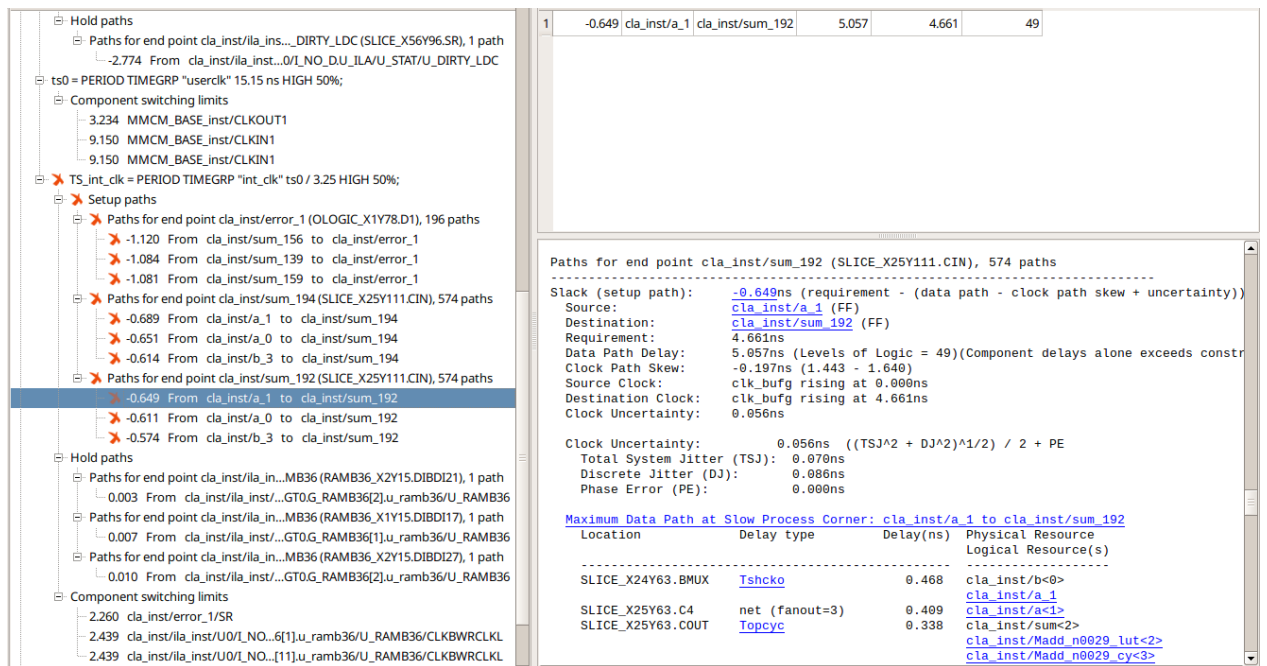


Рисунок 8 – Тип возникших нарушений

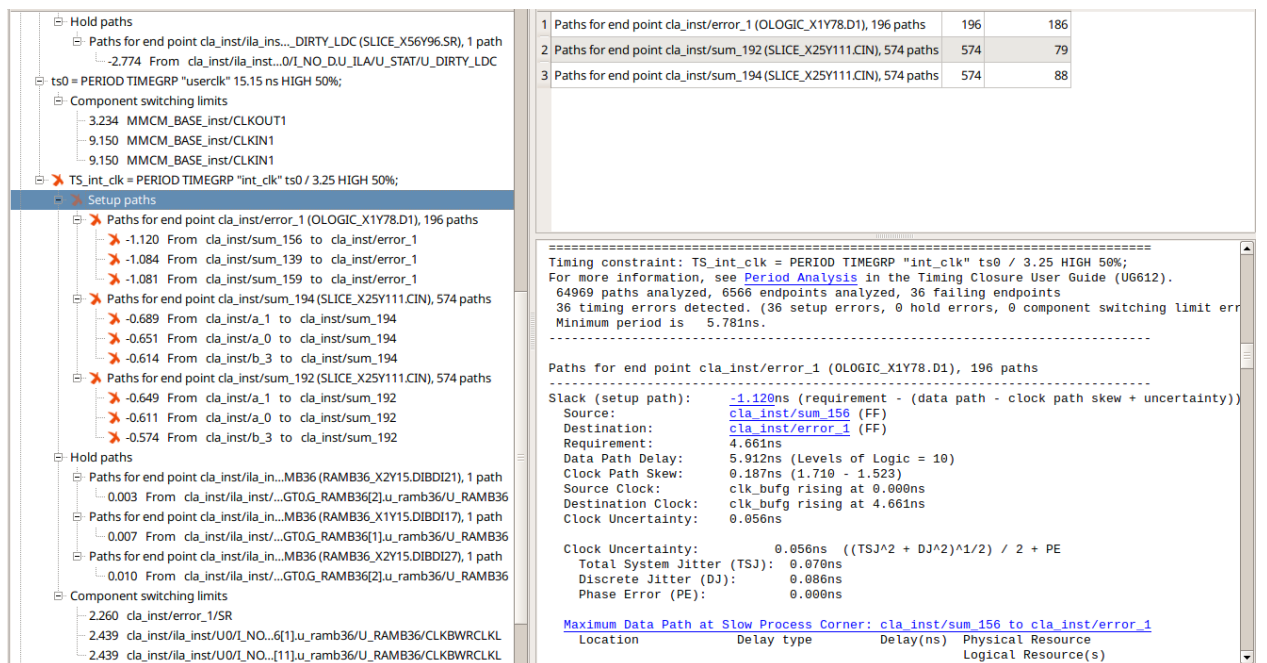


Рисунок 9 – Перечень сигналов и компонентов

Вывод: при выполнении задания были получены следующие параметры (рисунок 10): Slack - X и частота – X. Так же корректность работы была проверена на ПЛИС, «юг» не горел, что означает правильность выполнения (рисунок 11).

Design Overview								
<ul style="list-style-type: none"> <li>Summary</li> <li>IOB Properties</li> <li>Module Level Utilization</li> <li>Timing Constraints</li> <li>Pinout Report</li> <li>Clock Report</li> <li>Static Timing</li> <li>Errors and Warnings <ul style="list-style-type: none"> <li>Parser Messages</li> <li>Synthesis Messages</li> <li>Translation Messages</li> <li>Map Messages</li> <li>Place and Route Messages</li> <li>Timing Messages</li> <li>Bitgen Messages</li> <li>All Implementation Messages</li> </ul> </li> </ul>		Met	Constraint	Check	Worst Case Slack	Best Case Achievable	Timing Errors	Timing Score
1	Yes		<a href="#">TS_int_clk = PERIOD TIMEGRP "int_clk" ts0 / 2.6 HIGH 50%</a>	SETUP HOLD	0.061ns 0.014ns	5.765ns	0 0	0 0
2	Yes		<a href="#">ts0 = PERIOD TIMEGRP "userclk" 15.15 ns HIGH 50%</a>	MINLOWPULSE	9.150ns	6.000ns	0	0
3	Yes		<a href="#">PATH "TS_D2_TO_T2_path" TIG</a>	SETUP		2.234ns		0
4	Yes		<a href="#">PATH "TS_I2_TO_D2_path" TIG</a>					
5	Yes		<a href="#">PATH "TS_I3_TO_D2_path" TIG</a>					
6	Yes		<a href="#">PATH "TS_I4_TO_D2_path" TIG</a>	MAXDELAY		2.605ns		0

Рисунок 10 – Полученные параметры

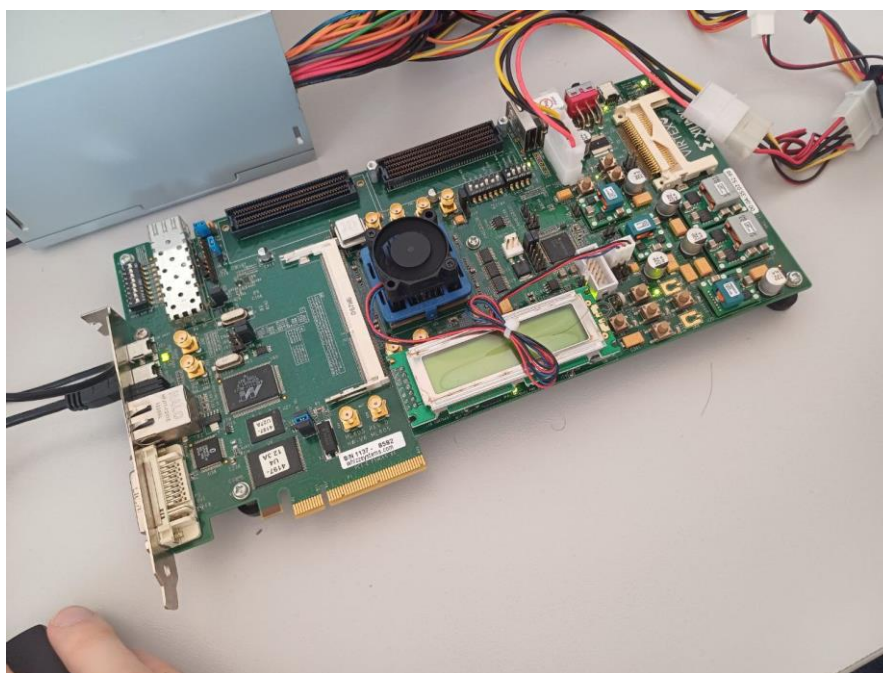


Рисунок 11 – Проверка на ПЛИС

## Задание 2. Генерация ядер логического анализатора для внутрисхемной отладки

Внутрисхемная отладка (In-System Debugging, ISD) позволяет анализировать сигналы и поведение проекта на ПЛИС непосредственно на целевом устройстве, без необходимости использования внешних эмуляторов или моделирования. Перед началом отладки необходимо определить, какие сигналы будут анализироваться и передать их на Интегрированный логический анализатор ИЛА. Триггеры позволяют захватывать данные только при наступлении определенных событий, что значительно упрощает анализ больших объемов данных. В соответствии с заданием были выполнены необходимые действия от создания нового IP ядра до генерации файла прошивки ПЛИС и конфигурацию ПЛИС. Был получен правильный результат



на плате. Также была получена диаграмма корректно работающего устройства, представленная на рисунке 12.

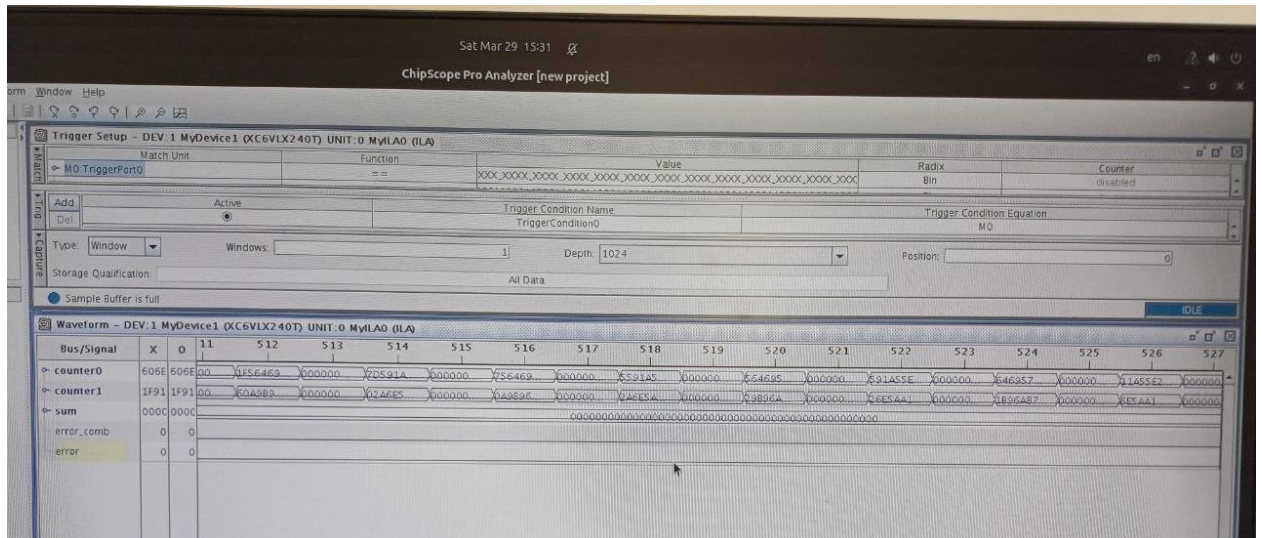


Рисунок 12 – Диаграмма корректно работающего устройства

После была увеличена тактовая частота синхронизации в 1.5 раза с помощью параметров CLKFBOUT\_MULT\_F/CLKOUT1\_DIVIDE, выполнена сборка проекта и прошивка ПЛИС. Для порта TRIG3 был указан в окне Trigger Setup тип триггерного события “1”. Полученная диаграмма некорректно работающего устройства показана на рисунке 13.

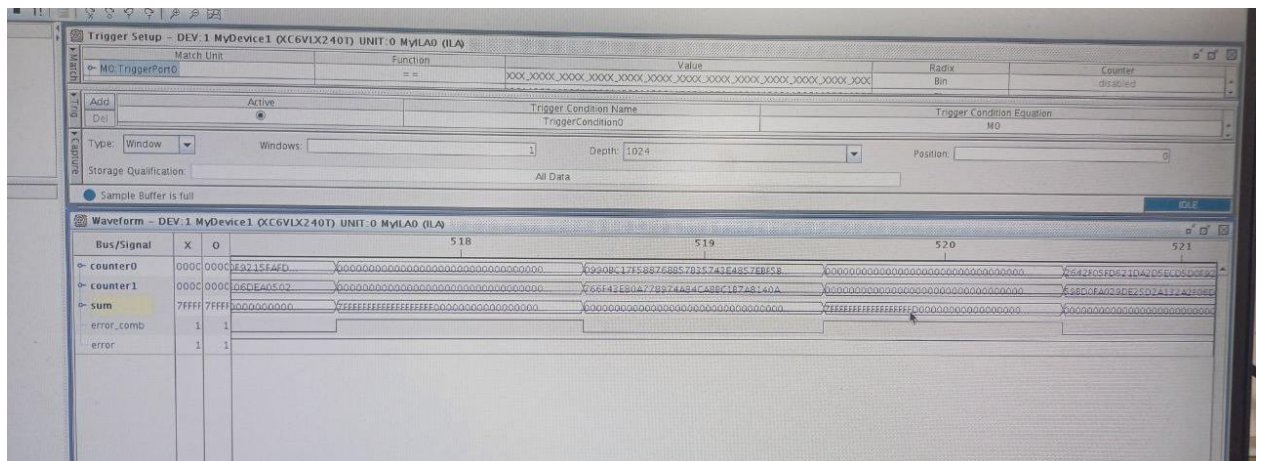


Рисунок 13 – Диаграмма некорректно работающего устройства

### Задание 3. Исследование работы конвейерного сумматора с передачей переноса по цепочке замкнутых ключей (CLA)

#### 3.1 Разработка описания конвейерного сумматора на языке Verilog

Модули, данные в методическом указании и отредактированные в соответствии с вариантом, представлены в листингах 2 и 3. Было выполнено моделирование и проверено, что устройство работает корректно.

#### Листинг 2 – Код cla\_checker\_pipelined

```
//Файл cla_checker_pipelined.v
module cla_checker_pipelined #(
    parameter w = 195
)
(
    input    rstn,
    input    clk,
    input    en,
    output reg error
);

    (* KEEP="TRUE" *) (* DONT_TOUCH="TRUE" *) wire [w-1:0]
counter,sum;
    (* KEEP="TRUE" *) (* DONT_TOUCH="TRUE" *) reg [w-1:0] a,b;
    wire error_comb;
    reg carry_in;
    wire sum_valid;
    //LFSR
    lfsr lfsr_inst(.clk(clk),.rstn(rstn),.en(en),.q(counter));
    // Входной перенос формируется Т триггером
    always @(posedge clk)
        if (!rstn)
            carry_in <= 1'b0;
        else
            carry_in <= ~carry_in;
    // Если carry_in==1, то а и b подаются из lsfr счетчика
    // иначе a=b=0
    always @(posedge clk) begin
        if (!rstn) begin
            a <= {w{1'b0}};
            b <= {w{1'b0}};
        end
        else if (!carry_in) begin
            a <= counter;
            b <= ~counter;
        end else begin
            a <= {w{1'b0}};
            b <= {w{1'b0}};
        end
    end
end
// Сумматор
    pipelined_adder #(
        .w(195),          //
        .s(4)             //
    ) pipelined_adder_inst (
        .clk(clk),
```

```

        .rstn(rstn),
        .op1(a),
        .op2(b),
        .cin(carry_in),
        .valid_op1(~en),
        .valid_op2(~en),
        .res(sum),
        .valid(sum_valid)
    );

    // Сравнение результата суммирования с эталонным значением
    // происходит в следующем такте
    assign error_comb = sum_valid & (sum != {w{1'b0}});
    // Если наблюдается сигнал ошибки, то установить состояние
    // светодиода: ВКЛ
    always @(posedge clk) begin
        if (!rstn) begin
            error <= 1'b0;
        end
        else if (error_comb) begin
            error <= 1'b1;
        end
    end

    wire[35:0] control0;
    icon666 icon_inst (
        .CONTROL0(control0)    // INOUT BUS [35:0]
    );

    ila ila_inst (
        .CONTROL(control0),    // INOUT BUS [35:0]
        .CLK(clk),              // IN
        .TRIG0(a),              // IN BUS [127:0]
        .TRIG1(b),              // IN BUS [127:0]
        .TRIG2(sum),            // IN BUS [127:0]
        .TRIG3(error_comb),     // IN BUS [0:0]
        .TRIG4(error)           // IN BUS [0:0]
    );
endmodule

```

**Листинг 3 – Файл cla\_pipelined**

```

//Файл cla_pipelined.v
module pipelined_adder #(
    parameter w = 195, // Ширина данных
    parameter s = 4     // Количество ступеней конвейера
) (
    input clk,           // Тактовый сигнал
    input rstn,          // Сброс (активен низкий)
    input [w-1:0] op1,   // Операнд 1
    input [w-1:0] op2,   // Операнд 2
    input cin,           // Входной перенос
    input valid_op1,     // Сигнал готовности операнда 1
    input valid_op2,     // Сигнал готовности операнда 2

```

```

output reg [w-1:0] res,    // Результат сложения
output reg valid          // Сигнал готовности результата
);

// Ширина каждой ступени конвейера
localparam [s*32-1:0] stage_widths = {32'd49, 32'd49, 32'd49,
32'd48};

// Макрос для доступа к ширине ступени
`define wth(stage) stage_widths[32*stage+:32]

// Функция для вычисления базового адреса для данной ступени
function integer base;
input integer stage;
begin
base = 0;
for (stage = stage; stage > 0; stage = stage - 1) begin
base = base + stage_widths[32*(stage-1)+:32];
end
end
endfunction

// Функция для получения ширины ступени
function integer width;
input integer stage;
begin
width = stage_widths[32*stage+:32];
end
endfunction

// Регистры для хранения данных на каждой ступени конвейера
reg [w-1:0] stage_reg [0:s-1];
// Комбинационные сигналы для каждой ступени
wire [w-1:0] stage_comb [0:s-1];
// Регистры для хранения операндов на каждой ступени
reg [w-1:0] stage_op1 [0:s-1];
reg [w-1:0] stage_op2 [0:s-1];
// Регистры для сигналов готовности на каждой ступени
reg [s-1:0] valid_reg;
// Регистры для переноса на каждой ступени
reg [s:0] c_reg;
// Комбинационные сигналы для переноса
wire [s:0] c_comb;
// Сигналы переноса из каждой ступени
wire [s-1:0] f;
integer i;
genvar k;

// Инициализация регистров
initial begin
for (i = 0; i < s; i = i + 1) begin
stage_reg[i] <= {w{1'b0}};
valid_reg[i] <= 1'b0;

```

```

        stage_op1[i] <= {w{1'b0}};
        stage_op2[i] <= {w{1'b0}};
        res <= {w{1'b0}};
    end
end

// Загрузка операндов в первую ступень конвейера
always @(*) begin
    stage_op1[0] <= op1; //Операнд 1
    stage_op2[0] <= op2; //Операнд 2
    c_reg[0] <= cin; //Входной перенос
end

// Генерация ступеней конвейера
generate
    for (k = 0; k < s; k = k + 1) begin : adder
        // Сложение на k-ой ступени
        assign {c_comb[k+1], stage_comb[k][base(k)+:`wth(k)],
f[k]} = {1'b0, stage_op1[k][base(k)+:`wth(k)], c_reg[k]} +
{1'b0, stage_op2[k][base(k)+:`wth(k)], c_reg[k]};

        // Тактируемый процесс для k-ой ступени
        always @(posedge clk) begin: stage_reg_inst
            if (~rstn) begin // Сброс
                for (i = 1; i < s; i = i + 1) begin
                    stage_reg[i][base(k)+:`wth(k)] <=
{(`wth(k)){1'b0}};
                end
            end else begin
                // Запись результата в регистр текущей ступени
                stage_reg[0][base(k)+:`wth(k)] <=
stage_comb[0][base(k)+:`wth(k)];
                // Передача данных между ступенями
                for (i = 1; i < s; i = i + 1) begin
                    if (valid_reg[i-1]) begin
                        if (i == k)
                            stage_reg[i][base(k)+:`wth(k)] <=
stage_comb[i][base(k)+:`wth(k)];
                        else
                            stage_reg[i][base(k)+:`wth(k)] <= stage_reg[i-
1][base(k)+:`wth(k)];
                    end
                end
            end
        end
    end
endgenerate

// Тактируемый процесс для управления конвейером и выдачи
результата
always @(posedge clk) begin
    if (~rstn) begin // Сброс
        valid <= 1'b0;
    end
end

```

```

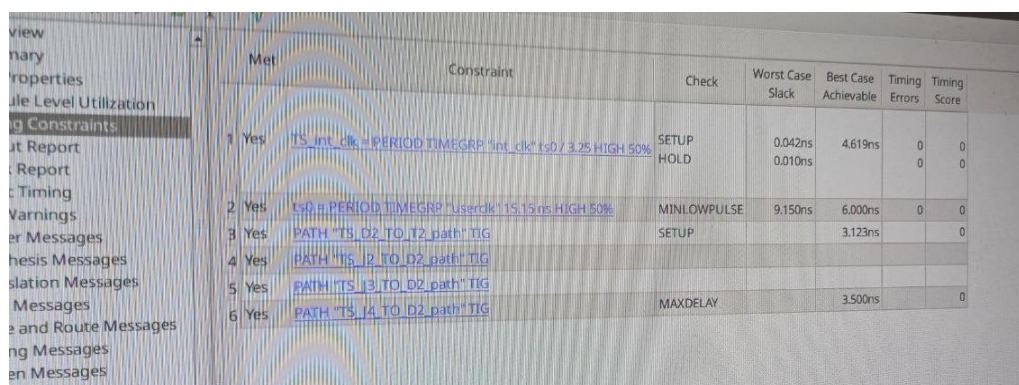
    res <= {w{1'b0}};
    valid_reg <= {s{1'b0}};
    for (i = 1; i < s; i = i + 1) begin
        stage_op1[i] <= {w{1'b0}};
        stage_op2[i] <= {w{1'b0}};
        c_reg[i] <= 1'b0;
    end
end else begin
    valid_reg[0] <= valid_op1 & valid_op2; // Сигнал
готовности для первой ступени
    // Распространение сигнала готовности и переноса по
ступеням конвейера
    for (i = 1; i < s; i = i + 1) begin
        valid_reg[i] <= valid_reg[i-1];
        c_reg[i] <= c_comb[i];
        stage_op1[i] <= stage_op1[i-1];
        stage_op2[i] <= stage_op2[i-1];
    end
    res <= stage_reg[s-1]; // Выдача результата из последней
ступени
    valid <= valid_reg[s-1]; // Сигнал готовности результата
end
end
endmodule

```

Рисунок 14 – Результат моделирования

### 3.2 Оптимизация конвейерного сумматора

Для индивидуального задания части 2 (разрядность устройства) были получены значения  $F\_CLA\_max$  для количества стадий конвейерного сумматора от 2-х до 5-ти, представленные на рисунках 15-18.



Met	Constraint	Check	Worst Case Slack	Best Case Achievable	Timing Errors	Timing Score
1 Yes	<a href="#">TS_int_clk = PERIOD TIMEGRP "int_clk" ts0 / 3.25 HIGH 50%</a>	SETUP HOLD	0.042ns 0.010ns	4.619ns	0 0	0
2 Yes	<a href="#">ts0 &gt; PERIOD TIMEGRP "Userclk" 15.15 ns HIGH 50%</a>	MINLOWPULSE	9.150ns	6.000ns	0	0
3 Yes	<a href="#">PATH "TS_D2 TO T2_path" TIG</a>	SETUP		3.123ns		0
4 Yes	<a href="#">PATH "TS_I2 TO D2_path" TIG</a>					
5 Yes	<a href="#">PATH "TS_I3 TO D2_path" TIG</a>					
6 Yes	<a href="#">PATH "TS_I4 TO D2_path" TIG</a>	MAXDELAY		3.500ns		0

Рисунок 15 – Результат временного анализа для 2-х стадий



Met	Constraint	Check	Worst Case Slack	Best Case Achievable	Timing Errors	Timing Score
1 No	TS_int_clk = PERIOD TIMEGRP "int_clk" ts0 / 3.5 HIGH 50%	SETUP HOLD	-0.271ns 0.010ns	4.599ns	1 0	271 0
2 Yes	ts0 = PERIOD TIMEGRP "userclk" 15.15 ns HIGH 50%	MINLOWPULSE	9.150ns	6.000ns	0	0
3 Yes	PATH "TS_D2_TO_T2_path" TIG	SETUP		2.712ns		0
4 Yes	PATH "TS_J2_TO_D2_path" TIG					
5 Yes	PATH "TS_J3_TO_D2_path" TIG			2.211ns		0
6 Yes	PATH "TS_J4_TO_D2_path" TIG	MAXDELAY				

Рисунок 16 – Результат временного анализа для 3-х стадий

Met	Constraint	Check	Worst Case Slack	Best Case Achievable	Timing Errors	Timing Score
1 Yes	TS_int_clk = PERIOD TIMEGRP "int_clk" ts0 / 3.5 HIGH 50%	SETUP HOLD	0.102ns 0.015ns	4.226ns	0 0	0 0
2 Yes	ts0 = PERIOD TIMEGRP "userclk" 15.15 ns HIGH 50%	MINLOWPULSE	9.150ns	6.000ns	0	0
3 Yes	PATH "TS_D2_TO_T2_path" TIG	SETUP		2.387ns		0
4 Yes	PATH "TS_J2_TO_D2_path" TIG					
5 Yes	PATH "TS_J3_TO_D2_path" TIG			2.306ns		0
6 Yes	PATH "TS_J4_TO_D2_path" TIG	MAXDELAY				

Рисунок 17 – Результат временного анализа для 4-х стадий

Met	Constraint	Check	Worst Case Slack	Best Case Achievable	Timing Errors	Timing Score
1 Yes	TS_int_clk = PERIOD TIMEGRP "int_clk" ts0 / 3.5 HIGH 50%	SETUP HOLD	0.015ns 0.010ns	4.313ns	0 0	0 0
2 Yes	ts0 = PERIOD TIMEGRP "userclk" 15.15 ns HIGH 50%	MINLOWPULSE	9.150ns	6.000ns	0	0
3 Yes	PATH "TS_D2_TO_T2_path" TIG	SETUP		2.319ns		0
4 Yes	PATH "TS_J2_TO_D2_path" TIG					
5 Yes	PATH "TS_J3_TO_D2_path" TIG			2.941ns		0
6 Yes	PATH "TS_J4_TO_D2_path" TIG	MAXDELAY				

Рисунок 18 – Результат временного анализа для 5-х стадий

На основе полученных результатов был построен график зависимости максимальной частоты устройства от количества стадий конвейера  $F\_CLA\_max(\text{Количество стадий конвейера})$  для варианта с латентностью 1 (получена ранее) и для четырех конвейерных вариантов, представленный на рисунке 19.

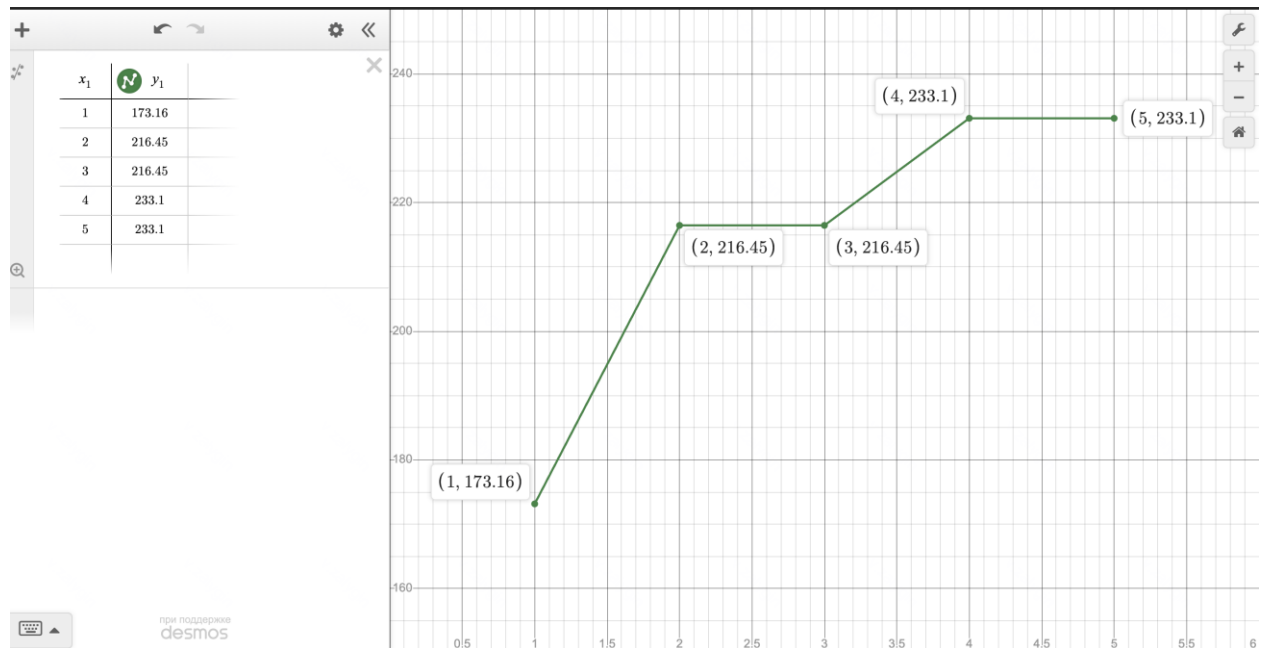


Рисунок 19 - График

В таблице 1 представлены подобранные параметры CLKFBOUT\_MULT\_F, CLKOUT1\_DIVIDE в зависимости от количества ступеней.

Таблица 1 – Параметры делителя частоты

Количество ступеней	CLKFBOUT_MULT_F	CLKOUT1_DIVIDE	Частота, MHz
1	13	5	173.16
2	13	4	216.45
3	13	4	216.45
4	14	4	233.1
5	14	4	233.1

**Вывод:** в ходе выполнения лабораторной работы были выполнены три задания, связанные с исследованием работы сумматора с передачей переноса

по цепочке замкнутых ключей, генерацией ядер логического анализатора для внутрисхемной отладки и исследованием работы конвейерного сумматора с передачей переноса по цепочке замкнутых ключей. Были получены корректные результаты при выполнении всех заданий: правильная работа на ПЛИС, верные параметры, диаграммы, созданные с помощью ChipScope, и график зависимости максимальной частоты устройства от количества стадий конвейера