|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **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**

**Отчет**

**по лабораторной работе № 1**

**Название:** Разработка конвейерных устройств

**Дисциплина:** Основы проектирования ЭВМ

**Вариант 10**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-63Б |  |  | В.К. Залыгин |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | С.В. Ибрагимов |
|  |  |  | (Подпись, дата) | (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.

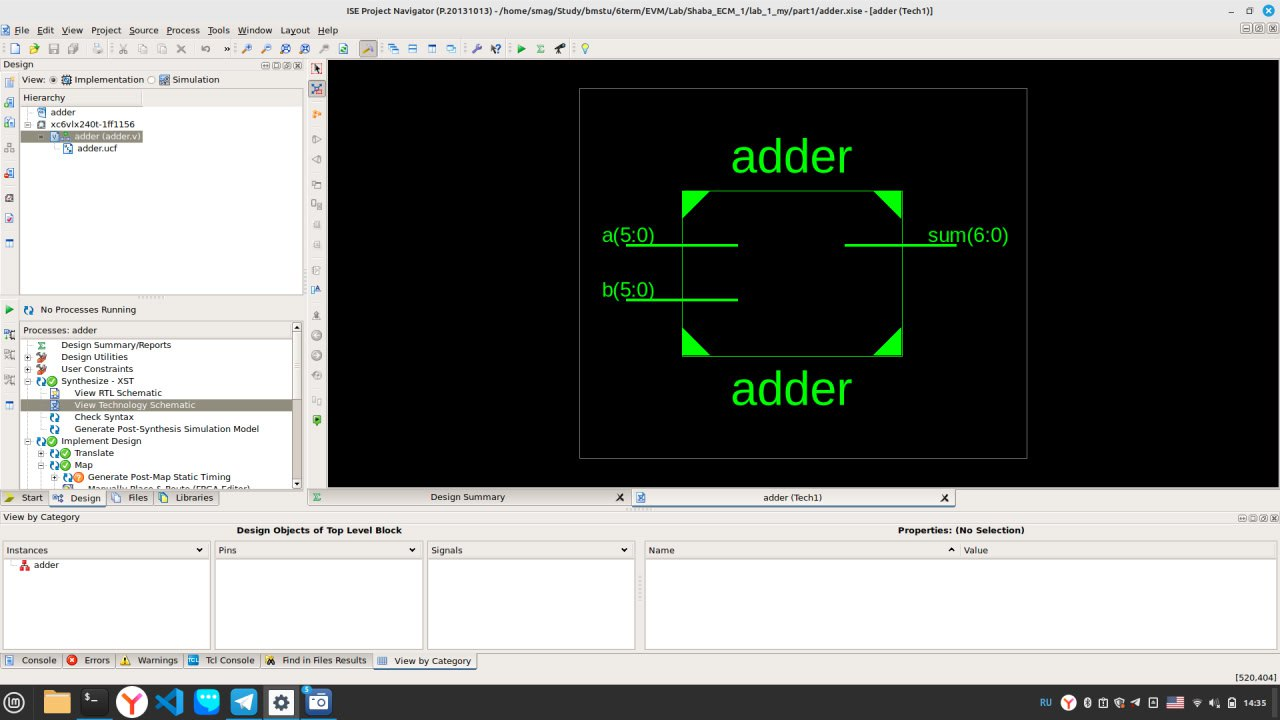


Рисунок 1 – Схема полученного сумматора

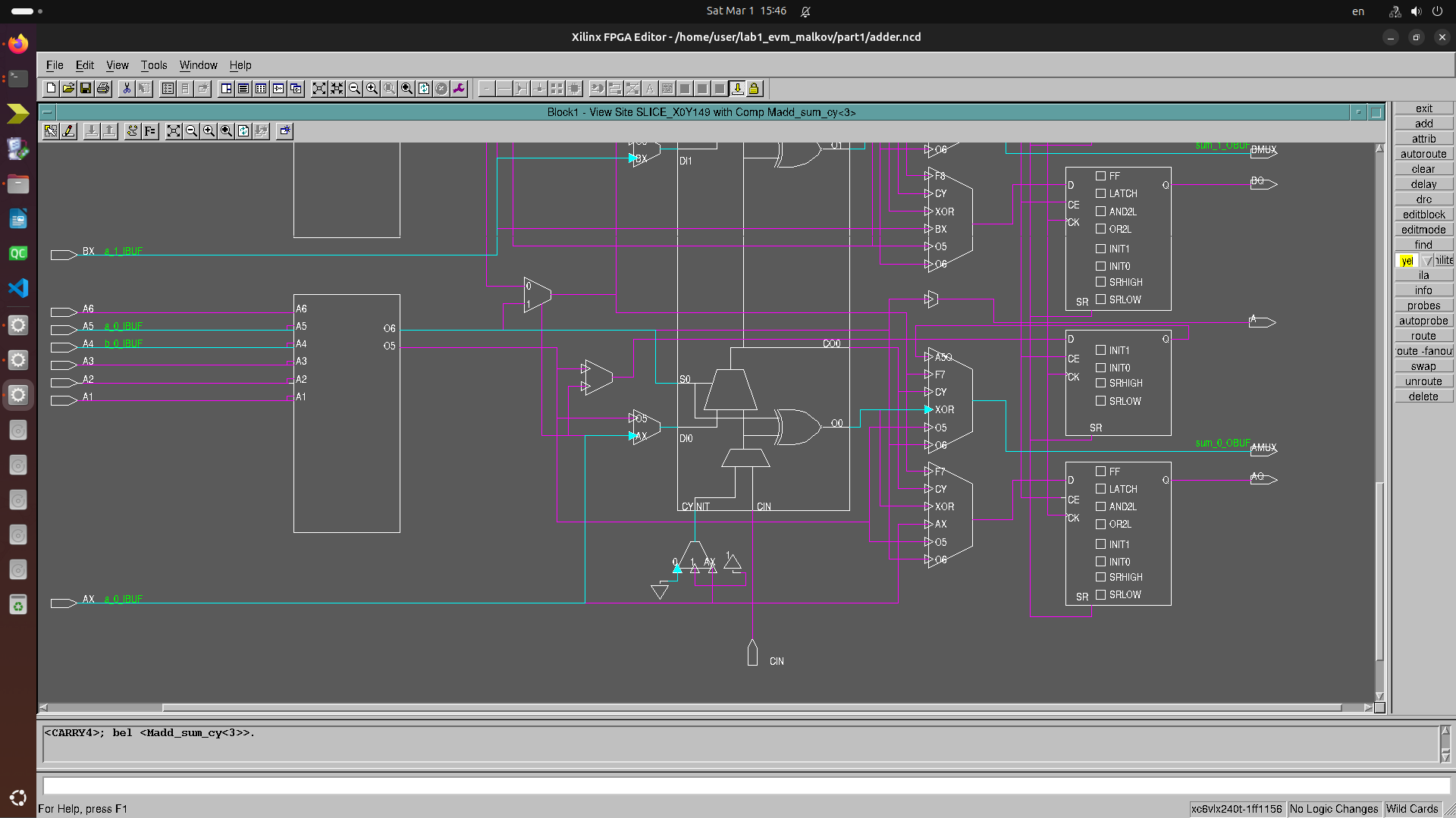


Рисунок 2 – Схема сумматора для одного блока SLICEL

Далее был получен модуль проверки нашего сумматора с заданной по варианту разрядностью (w=195). Код модуля представлен в листинге 1.

Листинг 1 – Код для случайной последовательности чисел

|  |
| --- |
| //Файл lfsr.v  module lfsr (  input clk,  input rstn,  input en,  output reg [194:0] q  );  reg [194:0] next\_q;  always @(posedge clk) begin  if (!rstn) begin  q <= {195{1'b1}}; // Инициализация  end else if (!en) begin  next\_q = q << 1;  next\_q[0] = q[194] ^ q[192] ^ q[191] ^ q[186]; // Обратная связь согласно неприводимому полиному степени 195  q <= next\_q;  end  end  endmodule |

Ответ на вопрос *«Почему в файле part2.ucf задано ограничение NET “rst” TIG, с условием, что мы игнорируем результаты анализа тайминов для сигнала RST?»*: сигнал rst представляет собой внешний асинхронный сигнал сброса от кнопки SW10. В связи с этим в файле part2.ucf задано ограничение NET "rst" TIG;, указывающее синтезатору игнорировать временной анализ для данного сигнала. Это предотвращает появление ложных ошибок тайминга, 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.

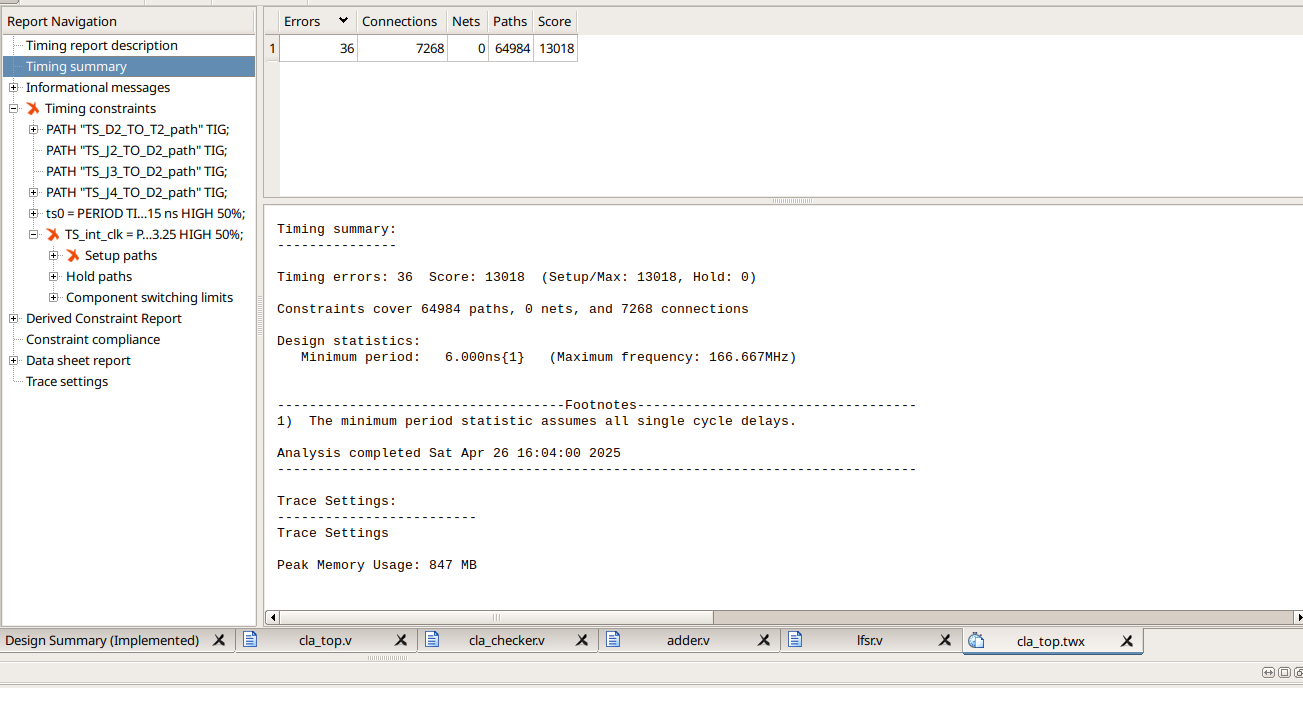


Рисунок 3 - Резюмированная информация о выполнении временных ограничений

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

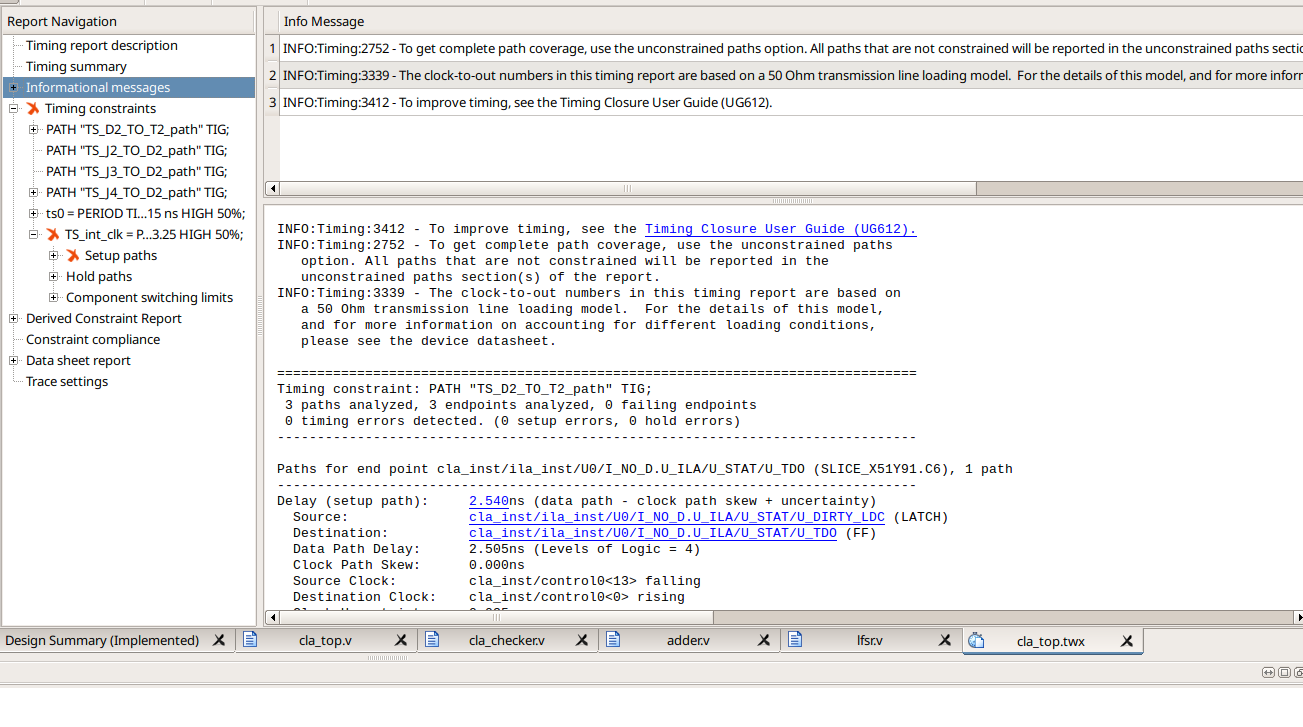


Рисунок 4 - Лог сообщений статического временного анализа

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

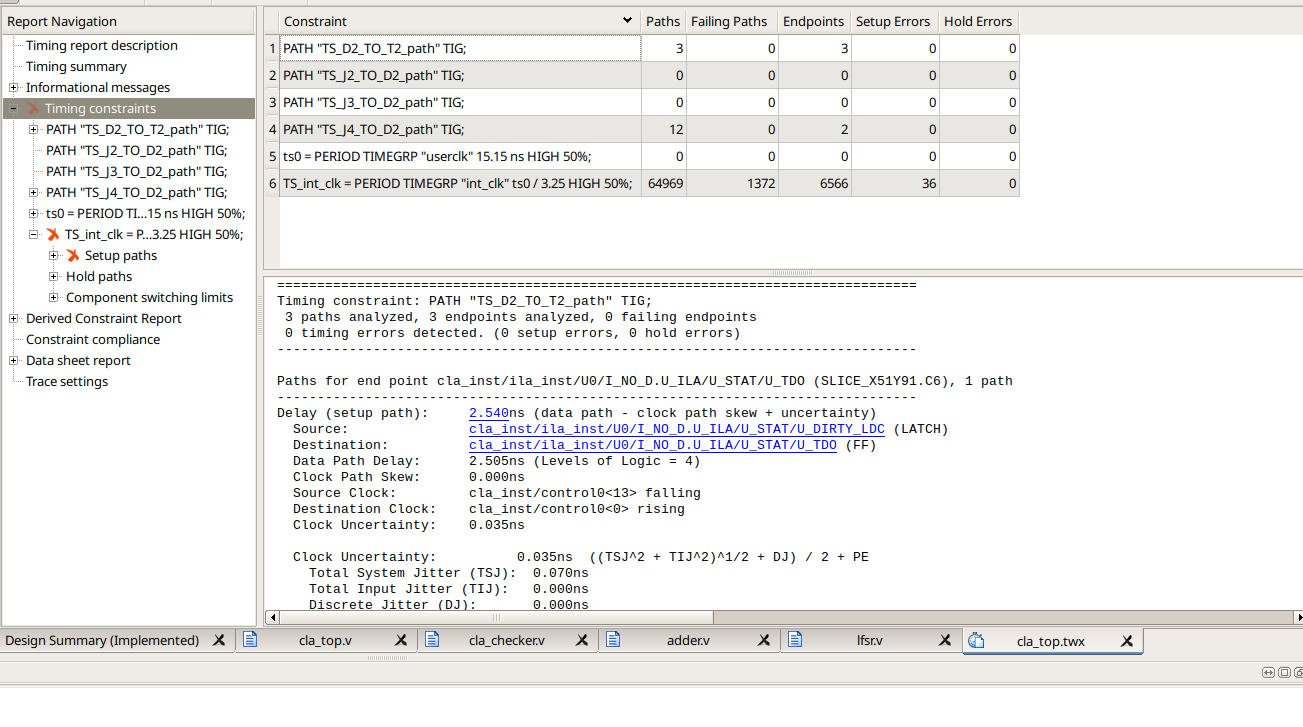


Рисунок 5 - Перечень заданных временных ограничений

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.

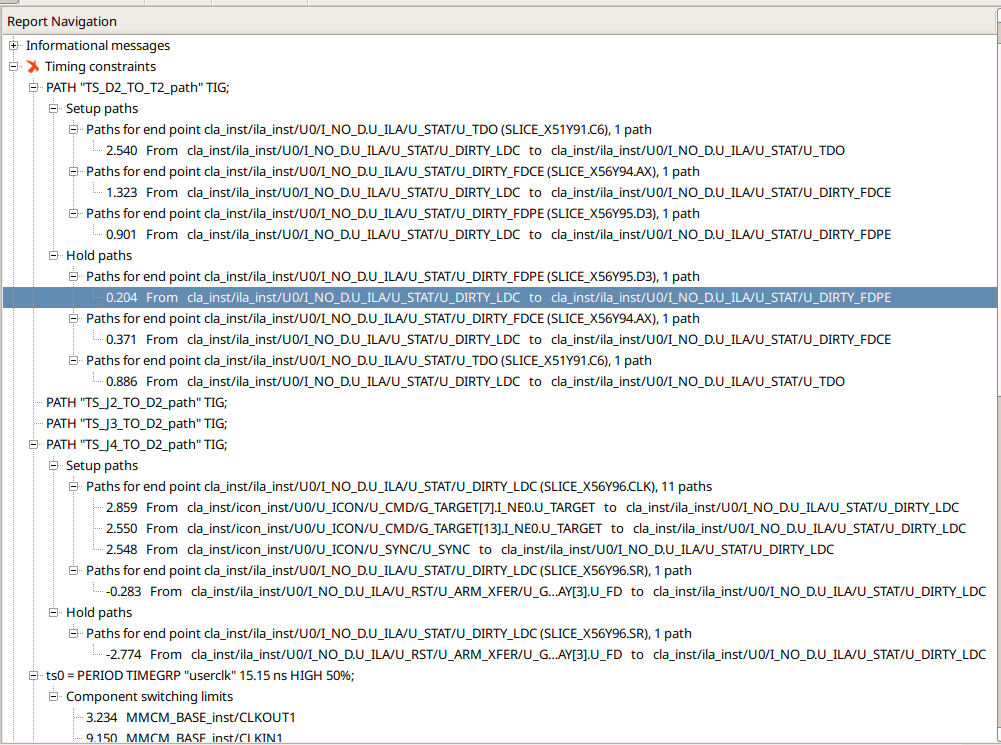


Рисунок 6 – Перечень критических путей

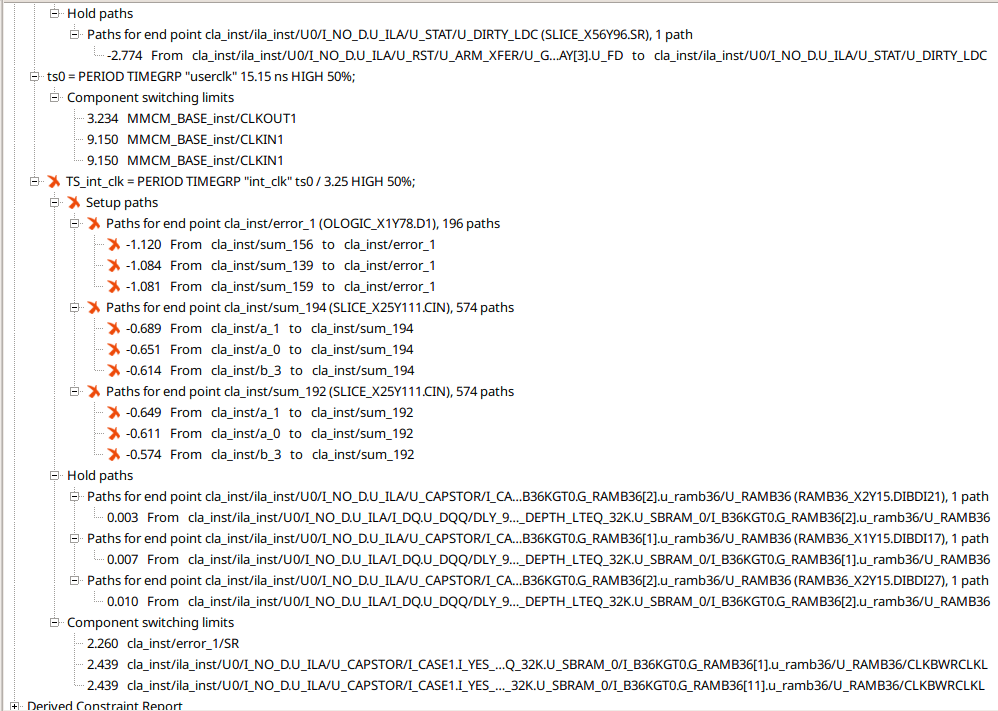


Рисунок 7 - Перечень критических путей

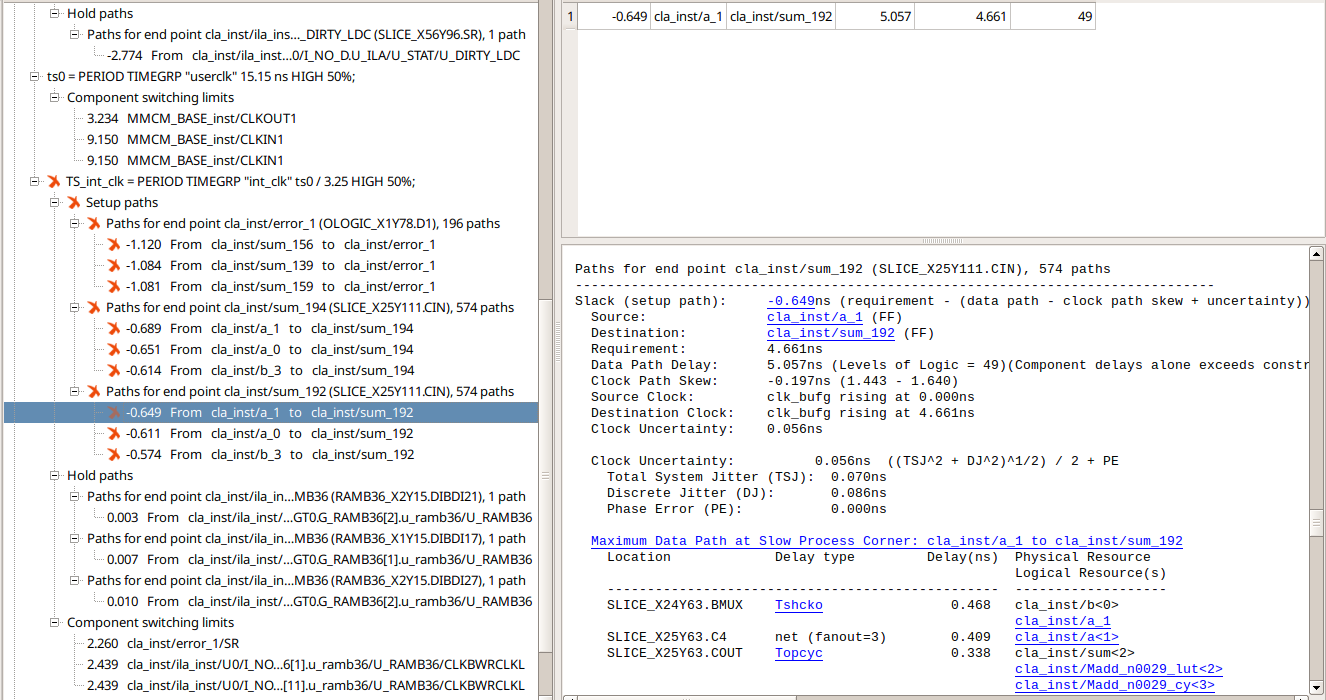


Рисунок 8 – Тип возникших нарушений

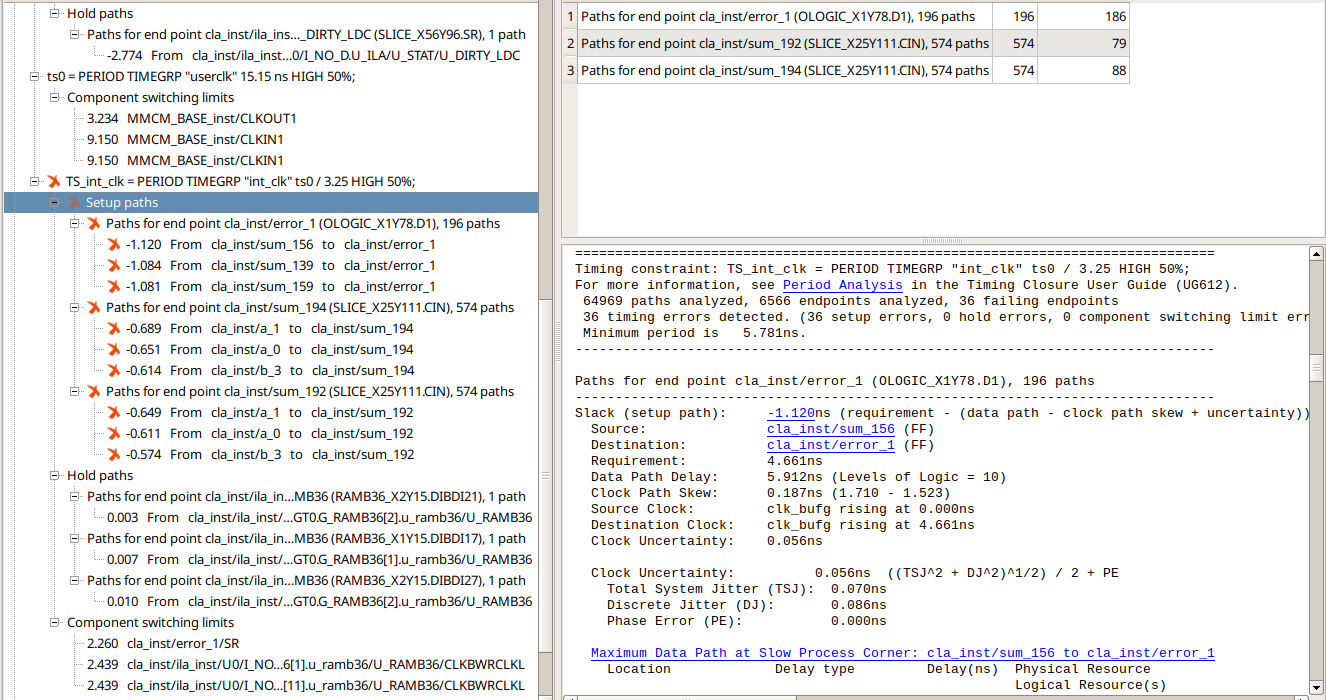


Рисунок 9 – Перечень сигналов и компонентов

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

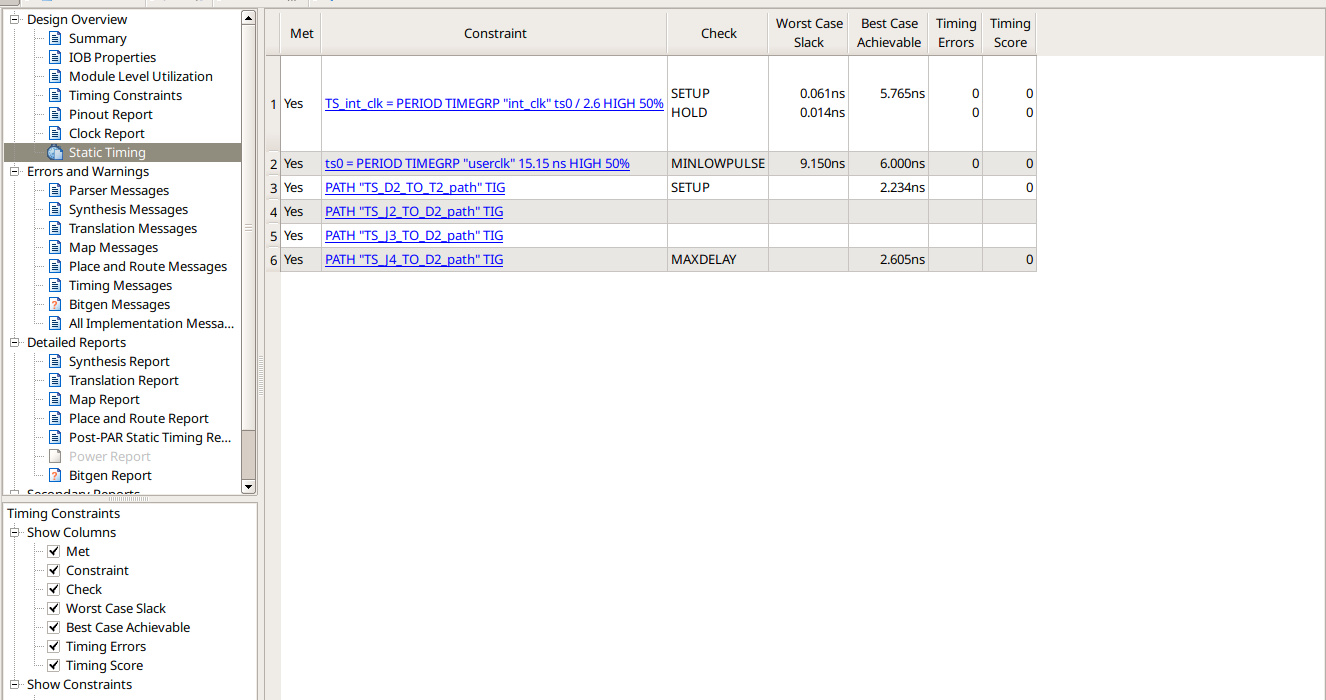


Рисунок 10 – Полученные параметры

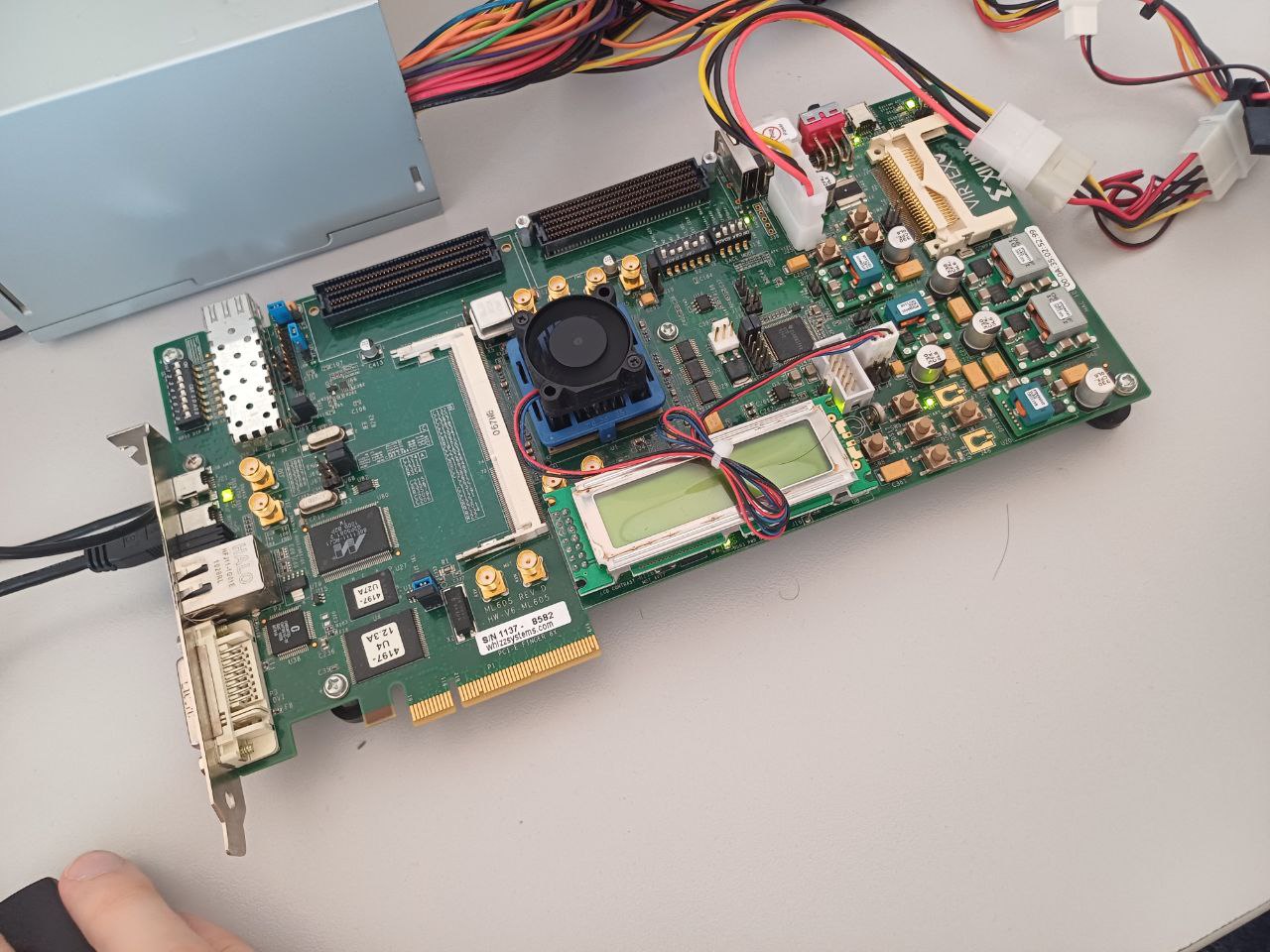


Рисунок 11 – Проверка на ПЛИС

**Задание 2. Генерация ядер логического анализатора для внутрирсхемной 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 ILA. 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.

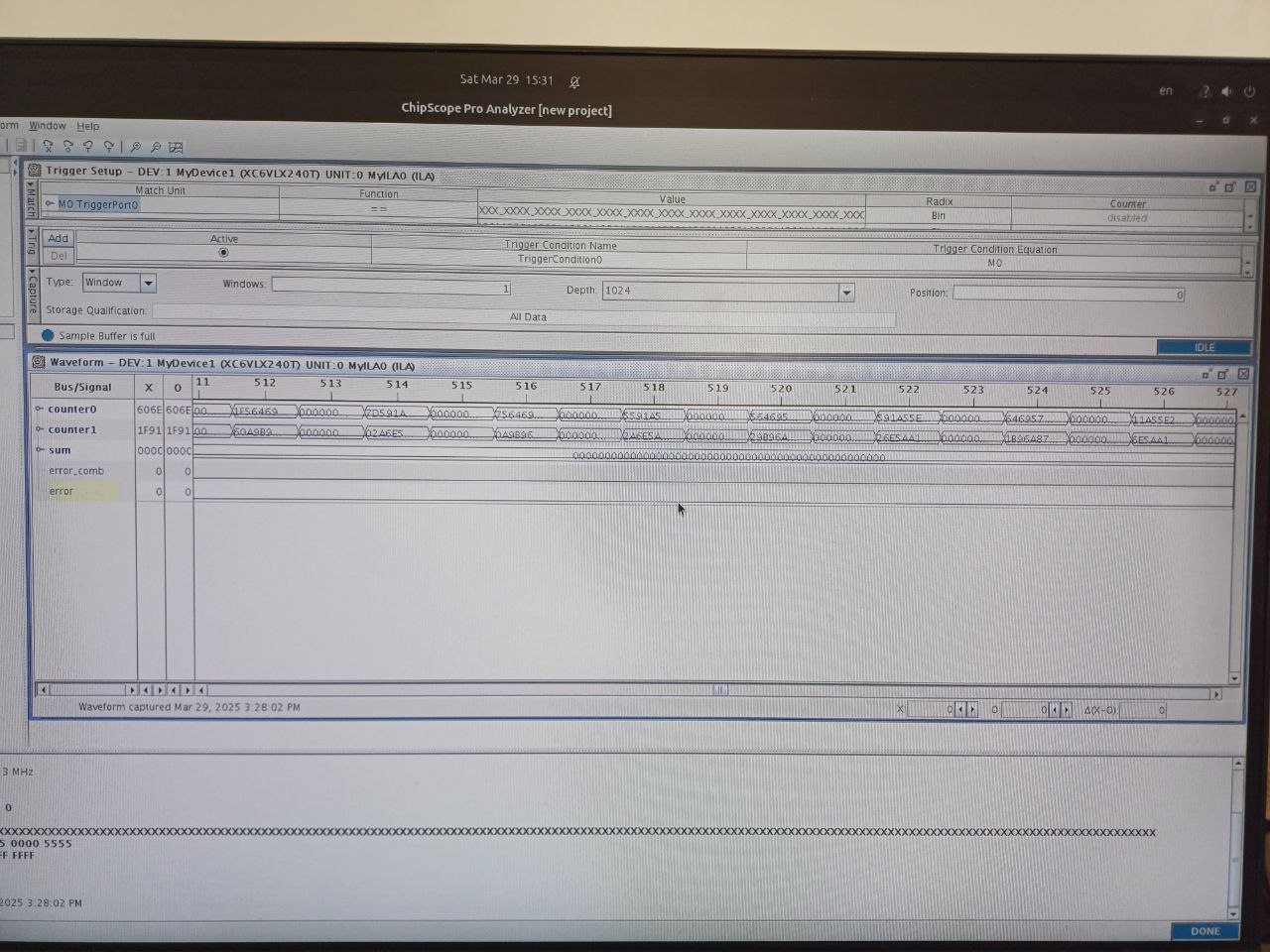


Рисунок 12 – Диаграмма корректно работающего устройства

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.

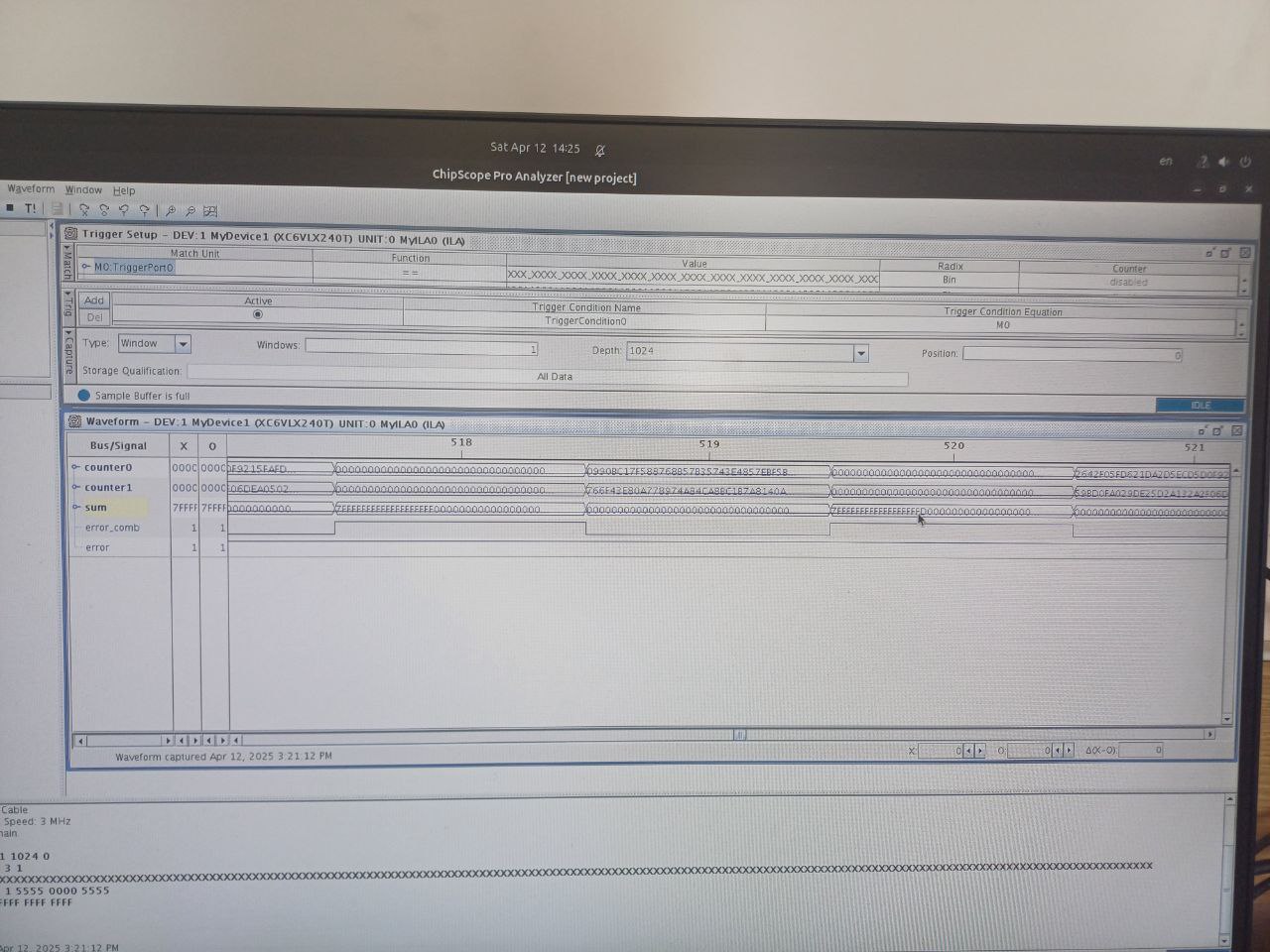


Рисунок 13 – Диаграмма некорректно работающего устройства

**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 T триггером  always @(posedge clk)  if (!rstn)  carry\_in <= 1'b0;  else  carry\_in <= ~carry\_in;  // Если carry\_in==1, то a 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  // Cумматор  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;  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.

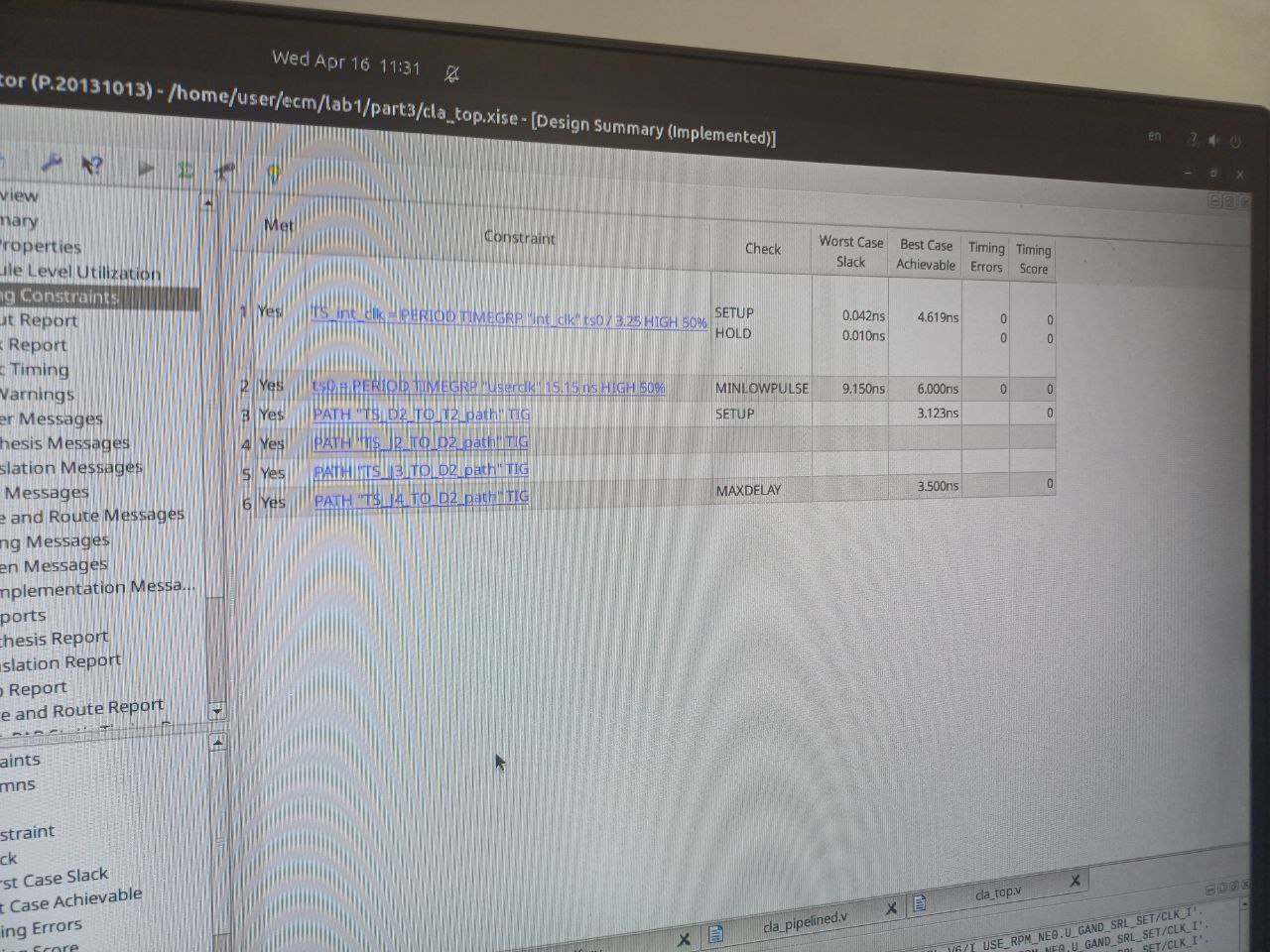


Рисунок 15 – Результат временного анализа для 2-х стадий

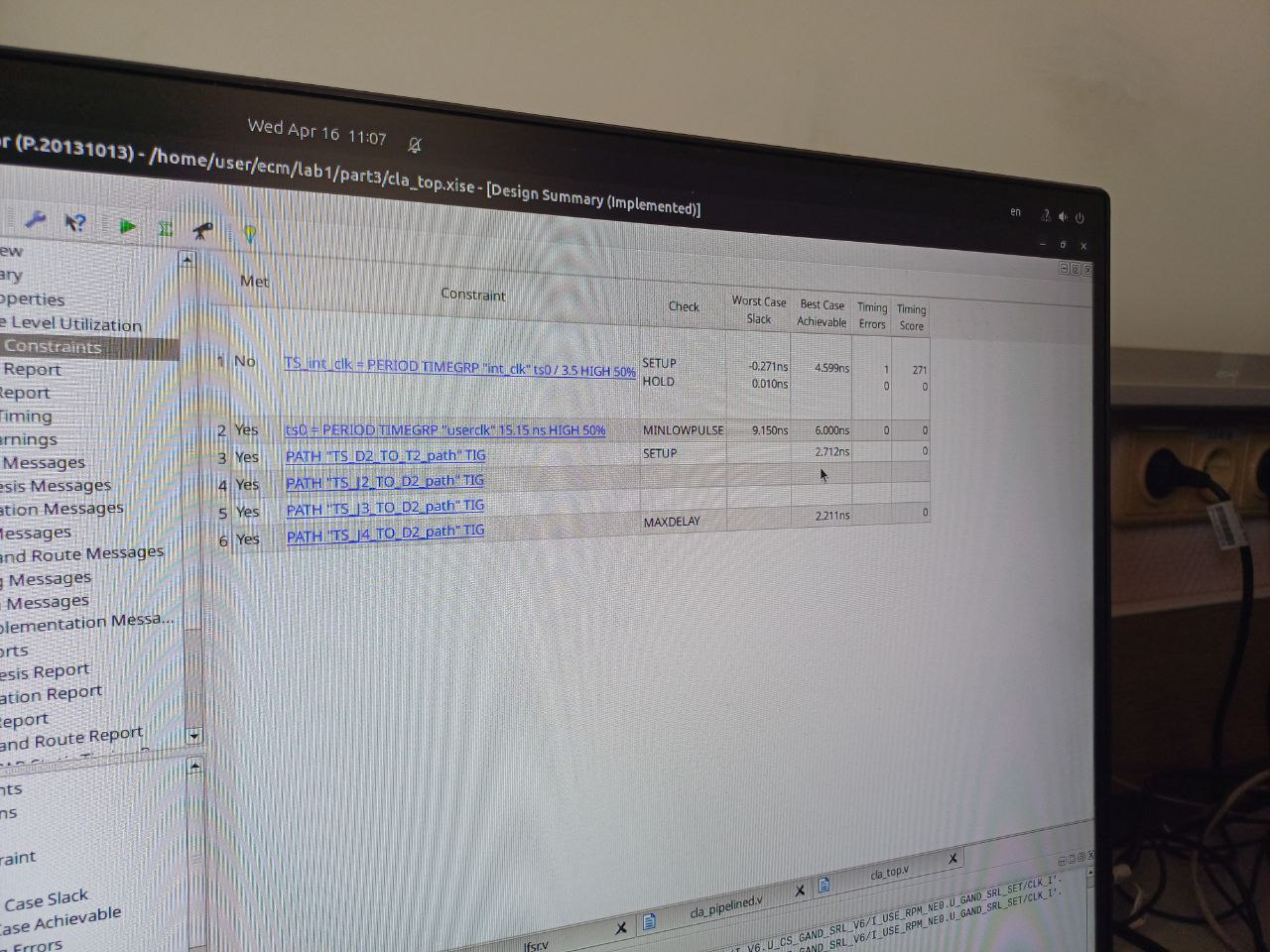


Рисунок 16 – Результат временного анализа для 3-х стадий

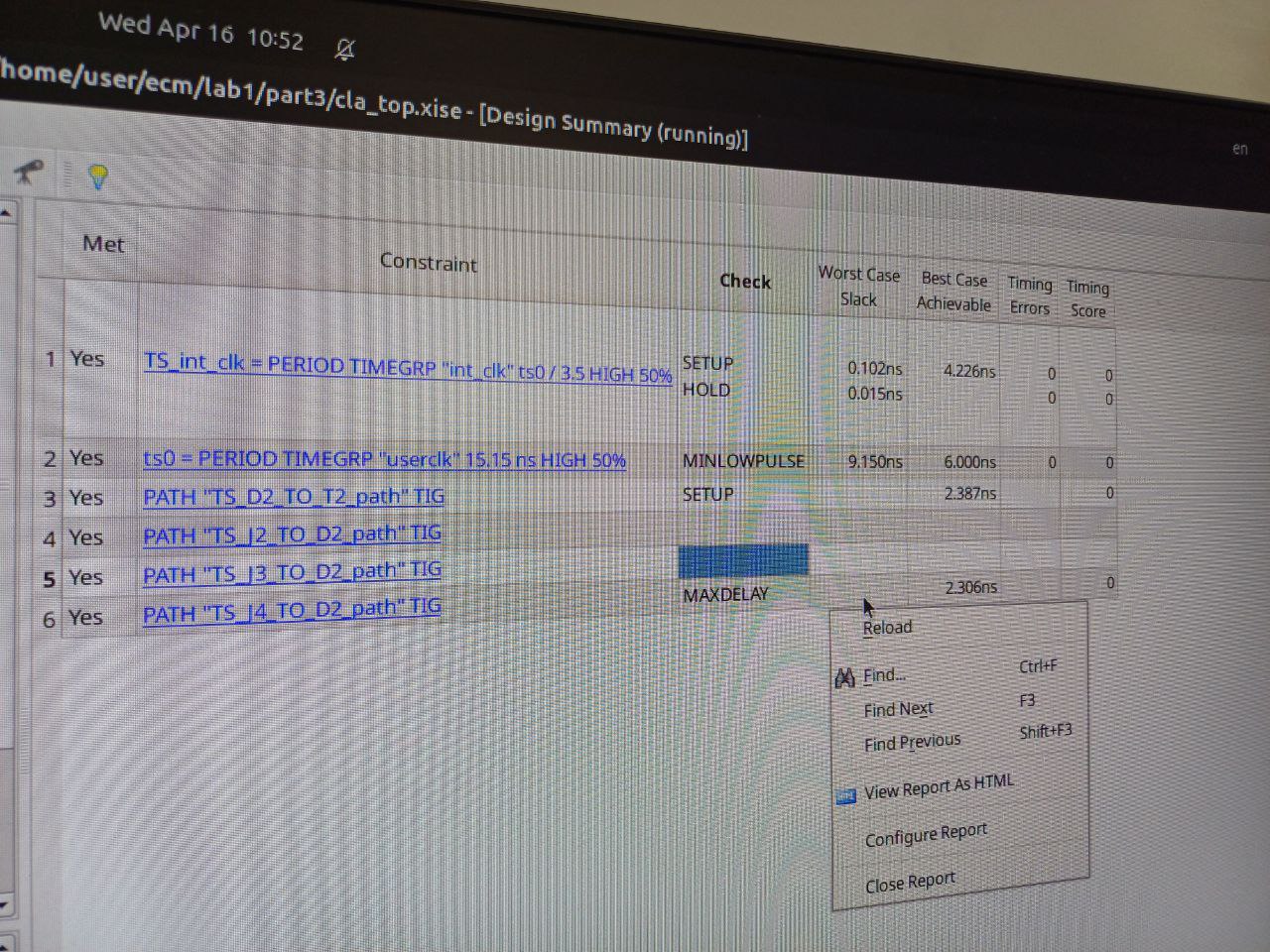


Рисунок 17 – Результат временного анализа для 4-х стадий

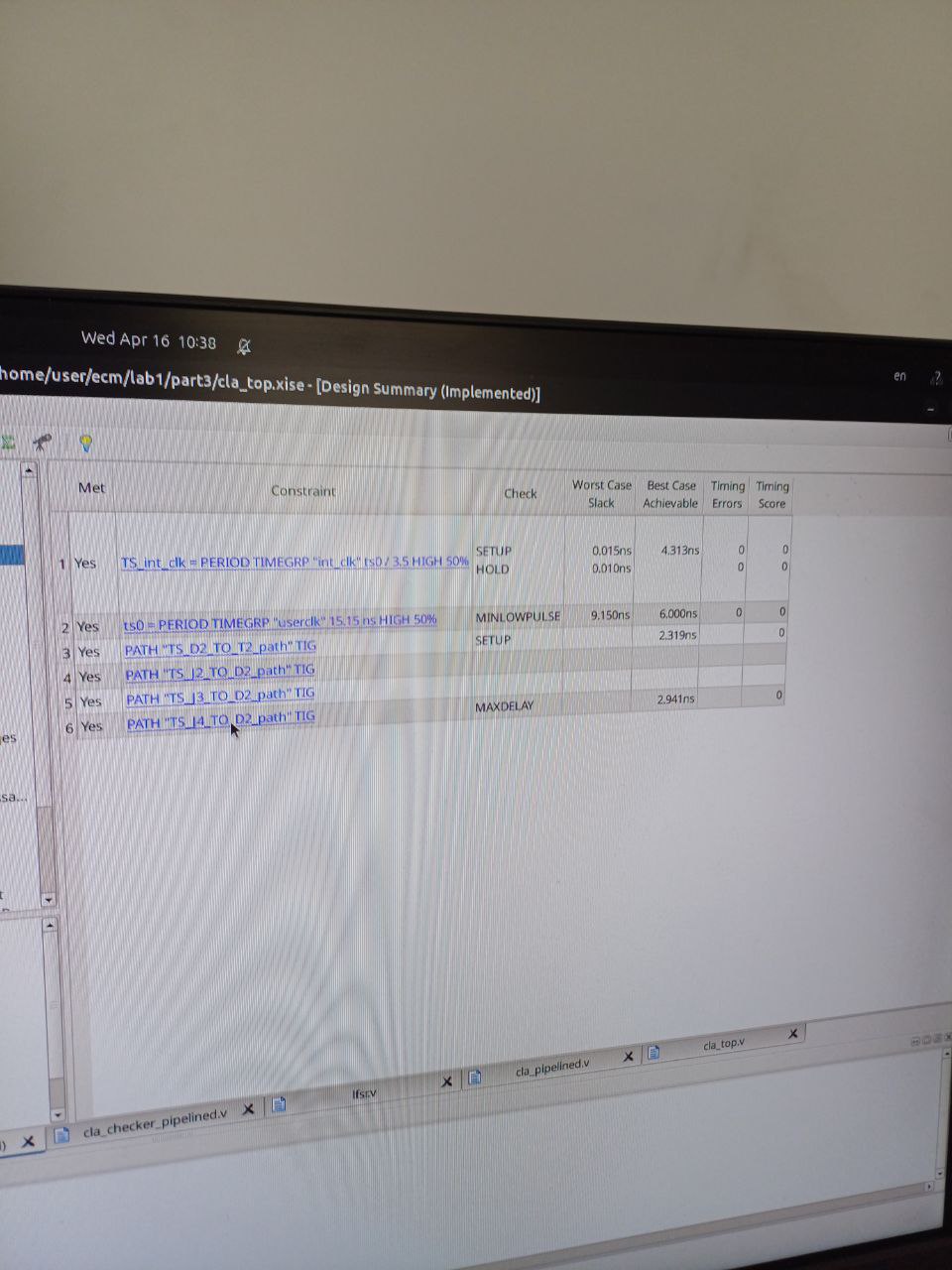


Рисунок 18 – Результат временного анализа для 5-х стадий

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

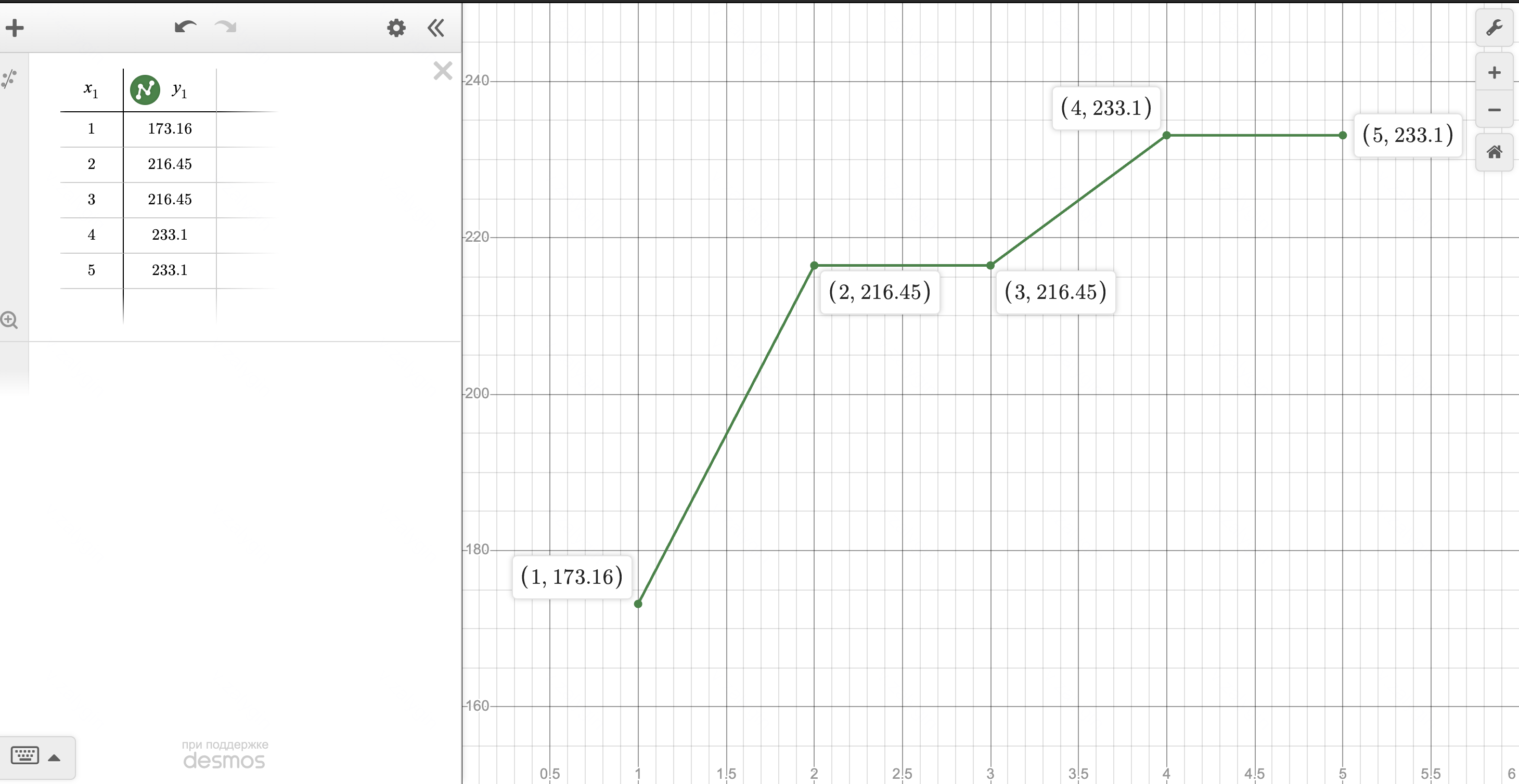


Рисунок 19 - График

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 внутрирсхемной 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