|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА - Российский технологический университет»**  **РТУ МИРЭА** |

Институт Информационных Технологий

Кафедра Вычислительной Техники (ВТ)

**ОТЧЁТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №** **2**

«Оптимизация RTL-модели устройства с целью улучшения временных показателей его работы»

по дисциплине

«Схемотехника устройств компьютерных систем»

|  |  |
| --- | --- |
| Выполнил студент группы  ИВБО-03-22 | Кулдина А.А. |
| Принял преподаватель кафедры ВТ | Дуксин Н. А. |
| Практическая работа выполнена | « » 2024 г. |
| «Зачтено» | « »\_\_\_\_\_\_\_\_\_\_ 2024 г. |

Москва 2024

АННОТАЦИЯ

Данная работа включает в себя 5 рисунок и 6 листингов. Количество страниц в работе — 20.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc179046035)

[1 Алгоритм CORDIC 5](#_Toc179046036)

[2 Оптимизация RTL-модели устройства ручным способом 8](#_Toc179046037)

[3 Оптимизация RTL-модели устройства при помощи размещения Pblock 9](#_Toc179046038)

[3.1 Размещение PBLOCK-ов в определенной последовательности 9](#_Toc179046039)

[3.2 Разделение PBLOCK-а на несколько PBLOCK-ов 10](#_Toc179046040)

[4 Оптимизация RTL-модели устройства при помощи конвейеризации внутри комбинационной логики 13](#_Toc179046041)

[5 Оптимизация RTL-модели устройства с использованием конвейеризации с помощью суматора 15](#_Toc179046042)

[ЗАКЛЮЧЕНИЕ 19](#_Toc179046043)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 20](#_Toc179046044)

# ВВЕДЕНИЕ

Цель работы: оптимизация RTL-модели устройства с целью улучшения временных показателей его работы.

Постановка задачи: произвести оптимизацию RTL-модели вычислительного устройства с целью улучшения временных показателей его работы [1].

# Алгоритм CORDIC

В качестве RTL-модели устройства будет использован в каждом способе оптимизации алгоритм CORDIC. Код модуля представлен в Листинге 1.1.

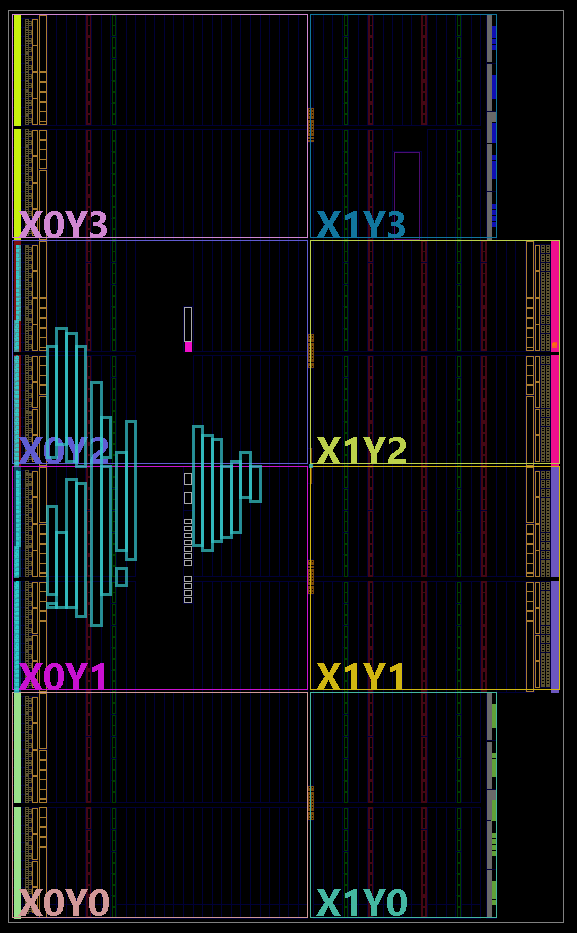
Листинг 1.1 – Модуль CORDIC

|  |
| --- |
| `timescale 1ns / 1ps  module CORDIC(  input clk,  input [31:0] angle,  input [15:0] X\_in, Y\_in,  output signed [16:0] sin\_out, cos\_out  );    wire signed [31:0] atan\_table [0:30];  `include "atan\_table.vh"  reg signed [31:0] X [0:31];  reg signed [31:0] Y [0:31];  reg signed [31:0] res\_acc [0:31];  reg signed [16:0] res\_acc\_add [0:31];  wire [1:0] quadrant = angle[31:30];  genvar i;  generate  for(i = 0; i < 31; i = i + 1)  begin: stage  if (i == 0)  always@(posedge clk)begin  case(quadrant)  2'b00, 2'b11: begin  res\_acc[0] <= angle;  X[0] <= X\_in;  Y[0] <= Y\_in;  end  2'b01: begin  res\_acc[0] <= {2'b00, angle[29:0]};  X[0] <= -Y\_in;  Y[0] <= X\_in;  end  2'b10: begin  res\_acc[0] <= {2'b11, angle[29:0]};  X[0] <= Y\_in;  Y[0] <= -X\_in;  end  endcase  end |

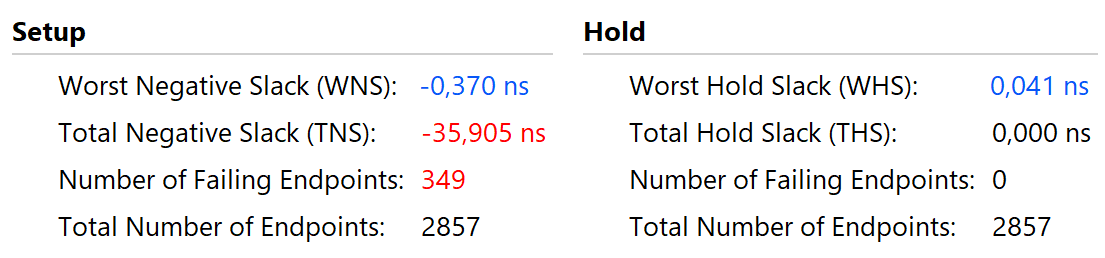
Продолжение Листинга 1.1

|  |
| --- |
| wire sign\_rotation;  wire signed [16:0] x\_shift, y\_shift;    assign x\_shift = X[i] >>> i;  assign y\_shift = Y[i] >>> i;  assign sign\_rotation = res\_acc[i][31];    wire [31:0] atan = sign\_rotation ? atan\_table[i] : -atan\_table[i];    always@(posedge clk)begin  X[i + 1] <= sign\_rotation ? X[i] + y\_shift : X[i] - y\_shift;  Y[i + 1] <= sign\_rotation ? Y[i] - x\_shift : Y[i] + x\_shift;    res\_acc[i + 1] <= sign\_rotation ? res\_acc[i] + atan\_table[i] : res\_acc[i] - atan\_table[i];    // res\_acc\_add[i] <= res\_acc[i][15:0] + atan[15:0];  // res\_acc[i + 1] <= {res\_acc[i][31:16] + atan[31:16] + res\_acc\_add[i][16], res\_acc\_add[i][15:0]};  end  /\*    wire [31:0] res\_acc\_result;  add\_tree RES\_ACC\_ADD (.clk(clk), .a(res\_acc[i]), .b(atan), .q(res\_acc\_result));  always@(posedge clk) res\_acc[i+1] <= res\_acc\_result;    wire [16:0] x\_result;  wire [16:0] x\_shift\_ = sign\_rotation ? -x\_shift : x\_shift;  wire [16:0] y\_result;  wire [16:0] y\_shift\_ = sign\_rotation ? y\_shift : -y\_shift;    add\_tree #(16) X\_ADD (.clk(clk), .a(X[i]), .b(y\_shift\_), .q(x\_result));  always@(posedge clk) X[i+1] <= x\_result;    add\_tree #(16) Y\_ADD (.clk(clk), .a(Y[i]), .b(x\_shift\_), .q(y\_result));  always@(posedge clk) Y[i+1] <= y\_result; \*/  end  endgenerate  assign sin\_out = Y[31];  assign cos\_out = X[31];  endmodule |

На Рисунках 1.1-1.2 представлено изначальное расположение на кристалле и значение Slack.



**Рисунок 1.1 — Изначальное размещение на кристалле**

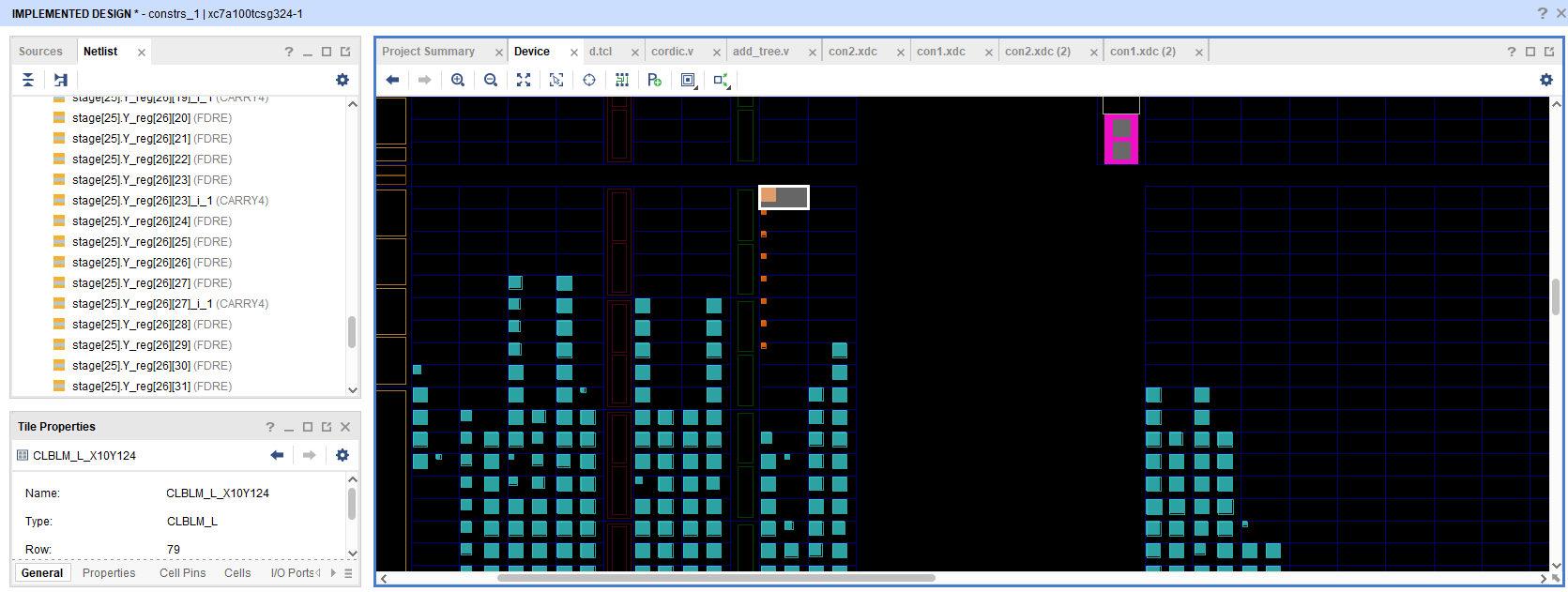


**Рисунок 1.2 — Значение Slack**

# Оптимизация RTL-модели устройства ручным способом

Одним из способов оптимизации RTL-модели устройства является ручное расположение ячеек на кристалле. Каждой конкретной ячейке можно определить соответствующее ей место, для этого нужно переместить ячейку с текущих координат на новые.

На Рисунке 2.1 представлен пример ручного перемещения ячейки на кристалле.

 **Рисунок 2.1 — Ручное размещение на кристалле**

# Оптимизация RTL-модели устройства при помощи размещения Pblock

Второй способ оптимизации RTL-модели устройства заключается в использовании PBLOCK-ов. PBLOCK – это свойство ячейки, доступное только для чтения, которое представляет собой область, которой назначена ячейка. Принадлежность ячейки к Pblock может быть изменена только с помощью команд XDC Tcl.

## Размещение PBLOCK-ов в определенной последовательности

Для того, чтобы разместить PBLOCK-и в определенной последовательности, к примеру «змейкой», был написан tcl-скрипт. Код файла tcl, который размещает данные по PBLOCK-ам по порядку в определенной последовательности, представлен ниже (Листинг 3.1).

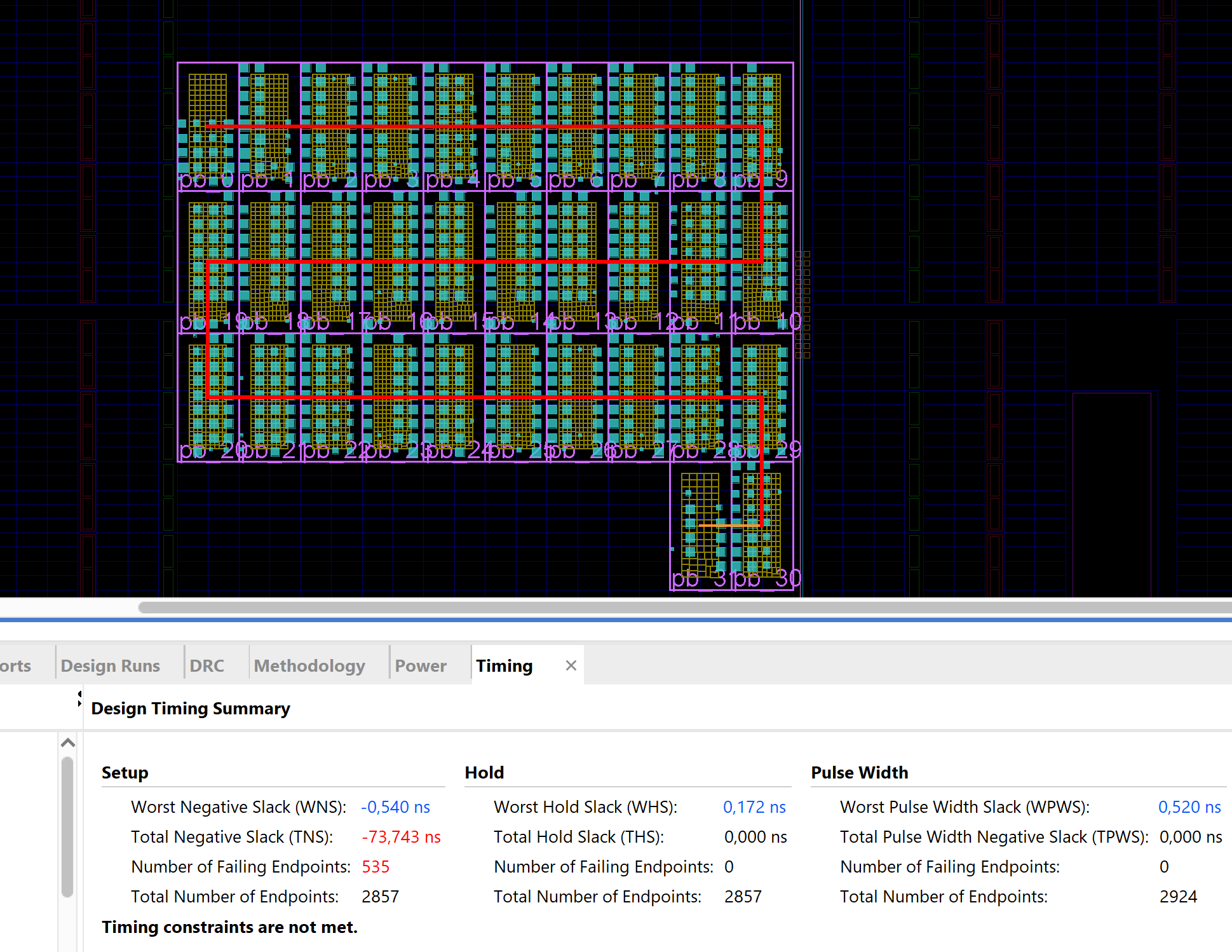
Листинг 3.1 – Tcl-скрипт

|  |
| --- |
| create\_clock -period 3.000 -name c -waveform {0.000 1.500} [get\_ports clk]  set\_property IOSTANDARD LVCMOS33 [get\_ports clk]  set\_property PACKAGE\_PIN E3 [get\_ports clk]  delete\_pblocks \*  set x0 12  set y0 183  set dir 1  set pb\_width 4  set pb\_height 9  for {set i 0} {$i < 32} {incr i} {    if { $i != 0 } {  if { [expr $i % 10 == 0 ] } {  set y0 [ expr $y0 - $pb\_height ]  set dir [ expr $dir \* -1 ]  } else {  set x0 [ expr $x0 + $pb\_width \* $dir ]  }  } |

Продолжение Листинга 3.1

|  |
| --- |
| set x1 [expr $x0 + $pb\_width - 1]  set y1 [expr $y0 + $pb\_height - 1]    create\_pblock "pb\_${i}"  resize\_pblock [ get\_pblocks "pb\_${i}" ] -add "SLICE\_X${x0}Y${y0}:SLICE\_X${x1}Y${y1}"  add\_cells\_to\_pblock [ get\_pblocks "pb\_${i}" ] [ get\_cells "stage\[\*\].\*\[${i}\]\[\*" ]  }  set\_property EXCLUDE\_PLACEMENT 1 [get\_pblocks "\*"]  set\_property IS\_SOFT FALSE |

На Рисунке 3.1 представлен пример размещения PBLOCK-ов на кристалле в виде «змейки».



**Рисунок 3.1 — Размещение PBLOCK-ов на кристалле**

## Разделение PBLOCK-а на несколько PBLOCK-ов

Другой вариацией размещения PBLOCK-ов является разделение PBLOCK-а на несколько других. В один из них размещаются все регистры, в другой комбинационная логика.

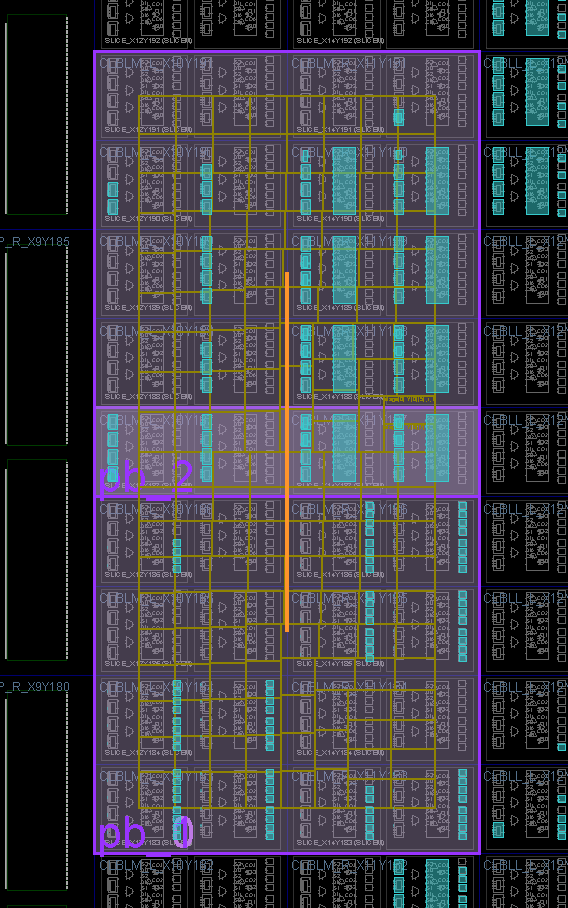
Второй вариант размещений Pblock-ов – разбиение одного pblock-а на два. В первом будет размещена комбинационная логика, во втором – регистры. Таким образом, накладывая pblock-и друг на друга, можно получить неплохое размещение на кристалле.

Код tcl-скрипта, который размещает данные в текущей последовательности по PBLOCK-ам, представлен ниже (Листинг 3.2).

Листинг 3.2 – Tcl-скрипт

|  |
| --- |
| create\_clock -period 3.000 -name c -waveform {0.000 1.500} [get\_ports clk]  set\_property IOSTANDARD LVCMOS33 [get\_ports clk]  set\_property PACKAGE\_PIN E3 [get\_ports clk]  delete\_pblocks \*  set x0 12  set y0 183  set y2 0  set dir 1  set pb\_width 4  set pb\_height 9  for {set i 0} {$i < 2} {incr i} {    if { $i != 0 } {  if { [expr $i % 10 == 0 ] } {  set y0 [ expr $y0 - $pb\_height ]  set dir [ expr $dir \* -1 ]  } else {  set x0 [ expr $x0 + $pb\_width \* $dir ]  }  }  set j $i  set x1 [expr $x0 + $pb\_width - 1]  set y1 [expr $y0 + $pb\_height - 1]  set y2 [ expr $y0 + 4]  create\_pblock "pb\_${i}"  resize\_pblock [ get\_pblocks "pb\_${i}" ] -add "SLICE\_X${x0}Y${y0}:SLICE\_X${x1}Y${y1}"  incr i  create\_pblock "pb\_${i}"  resize\_pblock [ get\_pblocks "pb\_${i}" ] -add "SLICE\_X${x0}Y${y0}:SLICE\_X${x1}Y${y2}"  add\_cells\_to\_pblock [ get\_pblocks "pb\_${i}" ] [ get\_cells -filter { REF\_NAME == FDRE } "stage\[\*\].\*\[${j}\]\[\*" ]  incr i  create\_pblock "pb\_${i}"  resize\_pblock [ get\_pblocks "pb\_${i}" ] -add "SLICE\_X${x0}Y${y2}:SLICE\_X${x1}Y${y1}"  add\_cells\_to\_pblock [ get\_pblocks "pb\_${i}" ] [ get\_cells -filter { REF\_NAME != FDRE } "stage\[\*\].\*\[${j}\]\[\*" ]  }  set\_property EXCLUDE\_PLACEMENT 1 [get\_pblocks "\*"]  set\_property IS\_SOFT FALSE [get\_pblocks "\*"] |

На Рисунке 3.2 представлен пример размещения PBLOCK-а на кристалле, внутри которого разграничены регистры и комбинационная логика.



**Рисунок 3.2 — Размещение PBLOCK-а на кристалле**

# Оптимизация RTL-модели устройства при помощи конвейеризации внутри комбинационной логики

Третьим способом оптимизации RTL-модели устройства является конвейеризация внутри комбинационной логики. Для этого способа потребуется изменить сам модуль, содержащий описание алгоритма CORDIC. Посмотрев на критический путь, можно заметить, что это операция сложения. Поэтому именно она и будет конвейеризирована. Максимальная тактовая частота определяется самым длинным физическим путем, следовательно, если разбивать весь путь регистрами, тактовая частота будет расти.

Измененный модуль, содержащий описание алгоритма CORDIC на Verilog HDL, представлен на Листинге 4.1.

Листинг 4.1 – Модуль CORDIC

|  |
| --- |
| `timescale 1ns / 1ps  module CORDIC(  input clk,  input [31:0] angle,  input [15:0] X\_in, Y\_in,  output signed [16:0] sin\_out, cos\_out  );    wire signed [31:0] atan\_table [0:30];  `include "atan\_table.vh"  reg signed [31:0] X [0:31];  reg signed [31:0] Y [0:31];  reg signed [31:0] res\_acc [0:31];  reg signed [16:0] res\_acc\_add [0:31];  wire [1:0] quadrant = angle[31:30];  genvar i;  generate  for(i = 0; i < 31; i = i + 1)  begin: stage  if (i == 0)  always@(posedge clk)begin  case(quadrant)  2'b00, 2'b11: begin |

Продолжение Листинга 4.1

|  |
| --- |
| res\_acc[0] <= angle;  X[0] <= X\_in;  Y[0] <= Y\_in;  end  2'b01: begin  res\_acc[0] <= {2'b00, angle[29:0]};  X[0] <= -Y\_in;  Y[0] <= X\_in;  end  2'b10: begin  res\_acc[0] <= {2'b11, angle[29:0]};  X[0] <= Y\_in;  Y[0] <= -X\_in;  end  endcase  end  wire sign\_rotation;  wire signed [16:0] x\_shift, y\_shift;    assign x\_shift = X[i] >>> i;  assign y\_shift = Y[i] >>> i;  assign sign\_rotation = res\_acc[i][31];    wire [31:0] atan = sign\_rotation ? atan\_table[i] : -atan\_table[i];    always@(posedge clk)begin  X[i + 1] <= sign\_rotation ? X[i] + y\_shift : X[i] - y\_shift;  Y[i + 1] <= sign\_rotation ? Y[i] - x\_shift : Y[i] + x\_shift;  res\_acc\_add[i] <= res\_acc[i][15:0] + atan[15:0];  res\_acc[i + 1] <= {res\_acc[i][31:16] + atan[31:16] + res\_acc\_add[i][16], res\_acc\_add[i][15:0]};  end  end  endgenerate  assign sin\_out = Y[31];  assign cos\_out = X[31];  endmodule |

# Оптимизация RTL-модели устройства с использованием конвейеризации с помощью суматора

Четвертым и последним можно выделить способ оптимизации RTL-модели устройства с использованием конвейеризации с помощью сумматора. Для этого был написан модуль, отвечающий за реализацию сумматора. Данный модуль представлен на Листинге 5.1.

Четвертый способ оптимизации – использование стадийного сумматора. Код модуля сумматора приведен в Листинге 5.1.

Листинг 5.1 – Модуль сумматора

|  |
| --- |
| module add\_tree #(SIZE = 32)(  input clk,  input [SIZE-1:0] a,  input [SIZE-1:0] b,  output [SIZE-1:0] q  );  genvar i, j;  generate  begin  for (i = 0; i < SIZE; i = i + 1)  begin: adder  reg [SIZE-i - 1 : 0] s;  reg [SIZE-i - 1 : 0] c;  end  for(i = 0; i < SIZE; i = i + 1) begin  for(j = 0; j < SIZE-i; j = j + 1)begin  if (i == 0)begin  always@(posedge clk)  {adder[i].c[j], adder[i].s[j]} <= a[j] + b[j];  end  else begin  always@(posedge clk)  {adder[i].c[j], adder[i].s[j]} <= adder[i-1].c[j] + adder[i-1].s[j+1];  end  end  end  for (i = 0; i < SIZE; i = i + 1)  assign q[i] = adder[i].s[0];  end  endgenerate  endmodule |

Также необходимо изменить модуль, отвечающий за реализацию алгоритма CORDIC. Код измененного модуля представлен в Листинге 5.2.

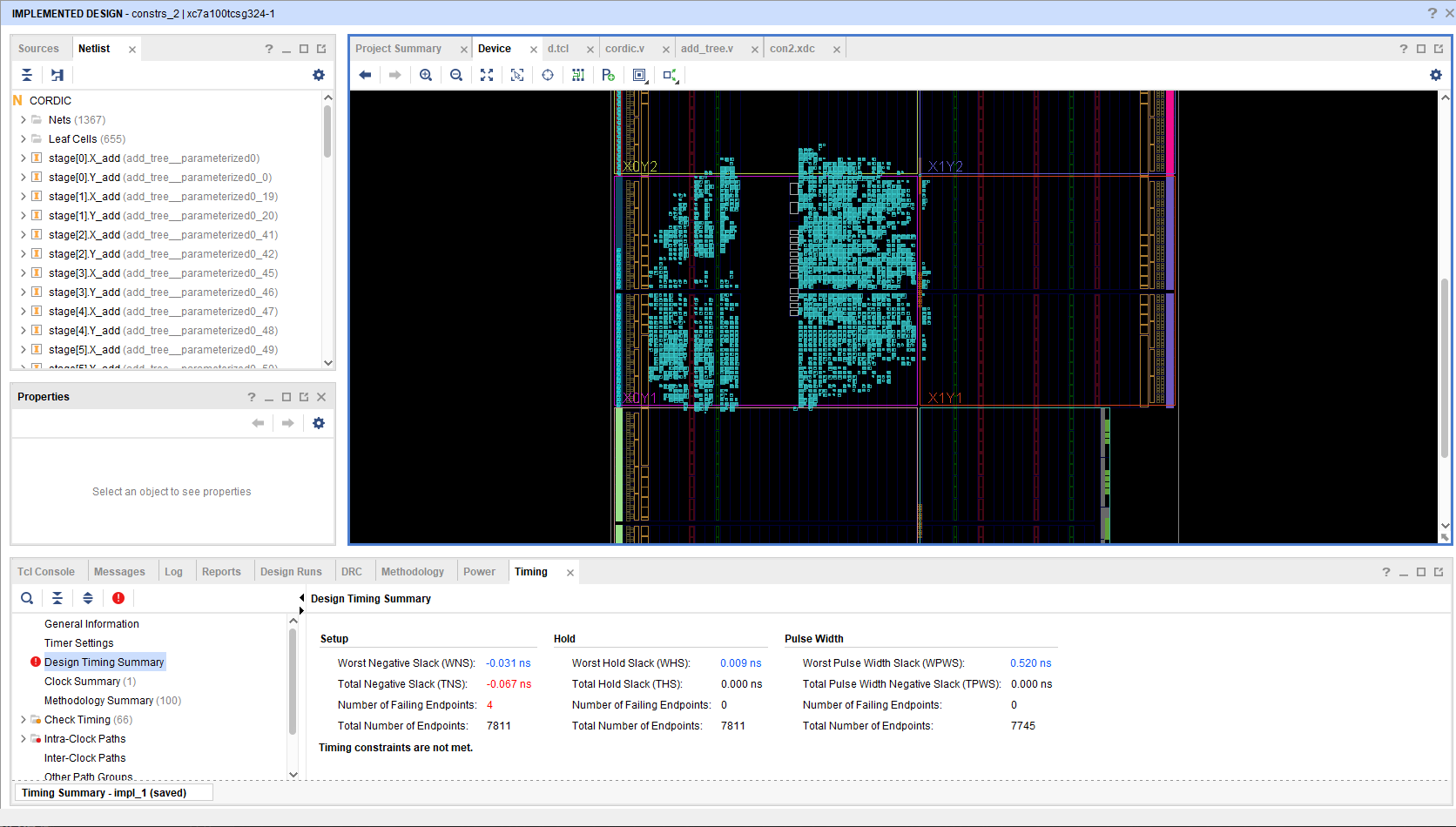
Листинг 5.2 – Модуль CORDIC

|  |
| --- |
| module CORDIC  (  input clk,  input [31:0] angle,  input [15:0] Xin, Yin,  output [16:0] COS\_OUT, SIN\_OUT  );  wire [31:0] atan\_table [0:30];  `include "atan\_table.vh"  reg signed [31:0] X [0:31];  reg signed [31:0] Y [0:31];  reg signed [31:0] RES\_ACC [0:31];  reg signed [16:0] RES\_ACC\_add [0:31];  wire [1:0] quadrant = angle[31:30];  genvar i;  generate  for(i = 0; i < 31; i = i + 1)  begin: stage    if (i == 0)  always@(posedge clk)  begin  case(quadrant)  2'b00, 2'b11:  begin  RES\_ACC[0] <= angle;  X[0] <= Xin;  Y[0] <= Yin;  end  2'b01:  begin  X[0] <= -Yin;  Y[0] <= Xin;  RES\_ACC[0] <= {2'b00, angle[29:0]};  end  2'b10:  begin  X[0] <= Yin;  Y[0] <= -Xin;  RES\_ACC[0] <= {2'b11, angle[29:0]};  end  endcase  end    wire rotation\_sign = RES\_ACC[i][31];  wire signed [16:0] X\_shift = X[i] >>> i;  wire signed [16:0] Y\_shift = Y[i] >>> i; |

Продолжение Листинга 5.2

|  |
| --- |
| //wire [31:0] atan = rotation\_sign ? atan\_table[i] : -atan\_table[i];    /\*always@(posedge clk)  begin  X[i+1] <= rotation\_sign ? X[i] + Y\_shift: X[i] - Y\_shift;  Y[i+1] <= rotation\_sign ? Y[i] - X\_shift: Y[i] + X\_shift;    //RES\_ACC[i+1] <= rotation\_sign ? RES\_ACC[i] + atan\_table[i] : RES\_ACC[i] - atan\_table[i];    //RES\_ACC\_add[i] <= RES\_ACC[i][15:0] + atan[15:0];  //RES\_ACC[i+1] <= {RES\_ACC[i][31:16] + atan[31:16] + RES\_ACC\_add[i][16], RES\_ACC\_add[i][15:0]};  end\*/      wire [31:0] atan = rotation\_sign ? atan\_table[i] : -atan\_table[i];  adder\_tree RES\_ACC\_add (.clk(clk), .a(RES\_ACC[i]), .b(atan), .q(RES\_ACC\_result));  always@(posedge clk) RES\_ACC[i+1] <= RES\_ACC\_result;    wire [16:0] X\_result;  wire [16:0] X\_shift\_ = rotation\_sign ? -X\_shift : X\_shift;  wire [16:0] Y\_result;  wire [16:0] Y\_shift\_ = rotation\_sign ? Y\_shift : -Y\_shift;    adder\_tree #(16) X\_add (.clk(clk), .a(X[i]), .b(Y\_shift\_), .q(X\_result));  always@(posedge clk) X[i+1] <= X\_result;    adder\_tree #(16) Y\_add (.clk(clk), .a(Y[i]), .b(X\_shift\_), .q(Y\_result));  always@(posedge clk) Y[i+1] <= Y\_result;  end  endgenerate  assign SIN\_OUT = Y[31];  assign COS\_OUT = X[31];  endmodule |

На Рисунке 5.1 представлено расположение на кристалле регистров и комбинационной логики после конвейеризации с помощью сумматора и изменившиеся значение Slack.



**Рисунок 5.1 — Размещение на кристалле**

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данной практической работы были изучены основные способы оптимизации RTL-модели вычислительного устройства с целью улучшения временных показателей его работы.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Методические указания по ПР № 2 — URL: https://online-edu.mirea.ru/pluginfile.php/1129098/mod\_resource/content/1/Практические%20занятия.pdf (Дата обращения: 14.09.2024).

2. Смирнов С.С. Информатика [Электронный ресурс]: Методические указания по выполнению практических и лабораторных работ / С.С. Смирнов — М., МИРЭА — Российский технологический университет, 2018. — 1 электрон. опт. диск (CD-ROM).

3. Тарасов И.Е. ПЛИС Xilinx. Языки описания аппаратуры VHDL и Verilog, САПР, приемы проектирования. — М.: Горячая линия — Телеком, 2021. — 538 с.: ил.