Санкт-Петербургский Политехнический Университет Петра Великого

Институт Компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

Лабораторная работа 7 Задание 1

Предмет: «Проектирование реконфигурируемых гибридных вычислительных систем»

**Тема: «Pipelining for Performance Demo Script»**

Студент:\_\_\_\_\_\_Волкова М.Д

Гр. № \_\_\_\_\_\_\_ [3540901/91501](https://vk.com/im?sel=c136)

Преподаватель: Антонов А.П.

Санкт-Петербург

2020

Оглавление

[1. Первое решение 4](#_Toc26754191)

[1.1. Открытие проекта 4](#_Toc26754192)

[1.2. Моделирование 7](#_Toc26754193)

[1.3. Производительность 8](#_Toc26754194)

[1.4. Использование ресурсов 8](#_Toc26754195)

[2. Второе решение 10](#_Toc26754196)

[2.1. Создание решения и применение директивы Pipeline 10](#_Toc26754197)

[2.2. Синтез 10](#_Toc26754198)

[2.3. Производительность 11](#_Toc26754199)

[2.4. Сравнение решений 11](#_Toc26754200)

[2.5. Использование ресурсов 11](#_Toc26754201)

[3. Третье решение 12](#_Toc26754202)

[3.1. Изменение директив 12](#_Toc26754203)

[3.2. Синтез 12](#_Toc26754204)

[3.3. Сравнение решений 13](#_Toc26754205)

[4. Выводы 13](#_Toc26754206)

1. Первое решение
   1. Открытие проекта

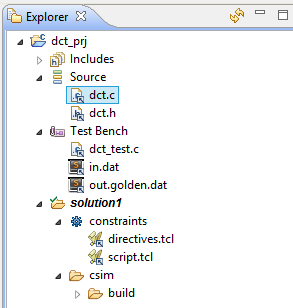


Рис. 1.1. Explorer opened project

Файл исходного кода dct.c:

**#include** "dct.h"

**void** **dct\_1d**(dct\_data\_t src[DCT\_SIZE], dct\_data\_t dst[DCT\_SIZE])

{

**unsigned** **int** k, n;

**int** tmp;

**const** dct\_data\_t dct\_coeff\_table[DCT\_SIZE][DCT\_SIZE] = {

**#include** "dct\_coeff\_table.txt"

};

DCT\_Outer\_Loop:

**for** (k = 0; k < DCT\_SIZE; k++) {

DCT\_Inner\_Loop:

**for**(n = 0, tmp = 0; n < DCT\_SIZE; n++) {

**int** coeff = (**int**)dct\_coeff\_table[k][n];

tmp += src[n] \* coeff;

}

dst[k] = DESCALE(tmp, CONST\_BITS);

}

}

**void** **dct\_2d**(dct\_data\_t in\_block[DCT\_SIZE][DCT\_SIZE],

dct\_data\_t out\_block[DCT\_SIZE][DCT\_SIZE])

{

dct\_data\_t row\_outbuf[DCT\_SIZE][DCT\_SIZE];

dct\_data\_t col\_outbuf[DCT\_SIZE][DCT\_SIZE], col\_inbuf[DCT\_SIZE][DCT\_SIZE];

**unsigned** i, j;

// DCT rows

Row\_DCT\_Loop:

**for**(i = 0; i < DCT\_SIZE; i++) {

dct\_1d(in\_block[i], row\_outbuf[i]);

}

// Transpose data in order to re-use 1D DCT code

Xpose\_Row\_Outer\_Loop:

**for** (j = 0; j < DCT\_SIZE; j++)

Xpose\_Row\_Inner\_Loop:

**for**(i = 0; i < DCT\_SIZE; i++)

col\_inbuf[j][i] = row\_outbuf[i][j];

// DCT columns

Col\_DCT\_Loop:

**for** (i = 0; i < DCT\_SIZE; i++) {

dct\_1d(col\_inbuf[i], col\_outbuf[i]);

}

// Transpose data back into natural order

Xpose\_Col\_Outer\_Loop:

**for** (j = 0; j < DCT\_SIZE; j++)

Xpose\_Col\_Inner\_Loop:

**for**(i = 0; i < DCT\_SIZE; i++)

out\_block[j][i] = col\_outbuf[i][j];

}

**void** **read\_data**(**short** input[N], **short** buf[DCT\_SIZE][DCT\_SIZE])

{

**int** r, c;

RD\_Loop\_Row:

**for** (r = 0; r < DCT\_SIZE; r++) {

RD\_Loop\_Col:

**for** (c = 0; c < DCT\_SIZE; c++)

buf[r][c] = input[r \* DCT\_SIZE + c];

}

}

**void** **write\_data**(**short** buf[DCT\_SIZE][DCT\_SIZE], **short** output[N])

{

**int** r, c;

WR\_Loop\_Row:

**for** (r = 0; r < DCT\_SIZE; r++) {

WR\_Loop\_Col:

**for** (c = 0; c < DCT\_SIZE; c++)

output[r \* DCT\_SIZE + c] = buf[r][c];

}

}

**void** **dct**(**short** input[N], **short** output[N])

{

**short** buf\_2d\_in[DCT\_SIZE][DCT\_SIZE];

**short** buf\_2d\_out[DCT\_SIZE][DCT\_SIZE];

// Read input data. Fill the internal buffer.

read\_data(input, buf\_2d\_in);

dct\_2d(buf\_2d\_in, buf\_2d\_out);

// Write out the results.

write\_data(buf\_2d\_out, output);

}

Файл-хедер dct.h:

**#ifndef** \_\_DCT\_H\_\_

**#define** \_\_DCT\_H\_\_

**#define** DW 16

**#define** N 1024/DW

**#define** NUM\_TRANS 16

**typedef** **short** dct\_data\_t;

**#define** DCT\_SIZE 8 /\* defines the input matrix as 8x8 \*/

**#define** CONST\_BITS 13

**#define** DESCALE(x,n) (((x) + (1 << ((n)-1))) >> n)

**void** **dct**(**short** input[N], **short** output[N]);

**#endif** // \_\_DCT\_H\_\_ not defined

Файл с тестами dct\_test.c:

**#include** <stdio.h>

**#include** "dct.h"

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**int** **main**() {

**short** a[N], b[N], b\_expected[N];

**int** retval = 0, i;

FILE \*fp;

fp=**fopen**("in.dat","r");

**for** (i=0; i<N; i++){

**int** tmp;

**fscanf**(fp, "%d", &tmp);

a[i] = tmp;

}

**fclose**(fp);

fp=**fopen**("out.golden.dat","r");

**for** (i=0; i<N; i++){

**int** tmp;

**fscanf**(fp, "%d", &tmp);

b\_expected[i] = tmp;

}

**fclose**(fp);

dct(a, b);

**for** (i = 0; i < N; ++i) {

**if**(b[i] != b\_expected[i]){

**printf**("Incorrect output on sample %d. Expected %d, Received %d \n", i, b\_expected[i], b[i]);

retval = 2;

}

}

**#if** 0 // Optionally write out computed values

fp=fopen("out.dat","w");

**for** (i=0; i<N; i++){

fprintf(fp, "%d\n", b[i]);

}

fclose(fp);

**#endif**

**if**(retval != (2)){

**printf**(" \*\*\* \*\*\* \*\*\* \*\*\* \n");

**printf**(" Results are good \n");

**printf**(" \*\*\* \*\*\* \*\*\* \*\*\* \n");

} **else** {

**printf**(" \*\*\* \*\*\* \*\*\* \*\*\* \n");

**printf**(" BAD!! %d \n", retval);

**printf**(" \*\*\* \*\*\* \*\*\* \*\*\* \n");

}

**return** retval;

}

Данный исходный файл использует дискретное косинусное преобразование (DCT). Функция реализует алгоритм 2D DCT, сначала обрабатывая каждую строку входного массива через 1D DCT, затем обрабатывая столбцы результирующего массива через тот же 1D DCT. Он вызывает функции read\_data, dct\_2d и write\_data. Функция read\_data состоит из двух циклов: RD\_Loop\_Row и RD\_Loop\_Col. Функция write\_data состоит из двух циклов для выполнения записи результата. Функция dct\_2d, вызывает функцию dct\_1d и выполняет транспонирование. Наконец, функция dct\_1d, использует dct\_coeff\_table и выполняет требуемую функцию, реализуя базовую итеративную форму алгоритма DCD 1D типа II.

* 1. Моделирование

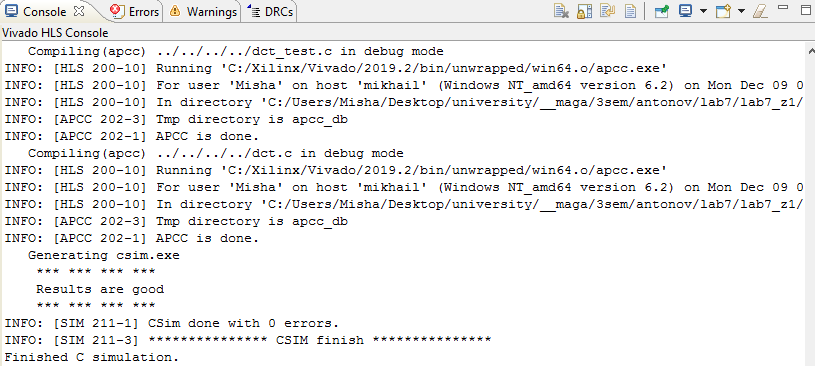


Рис. 1.2. Successful result of modeling

* 1. Синтез

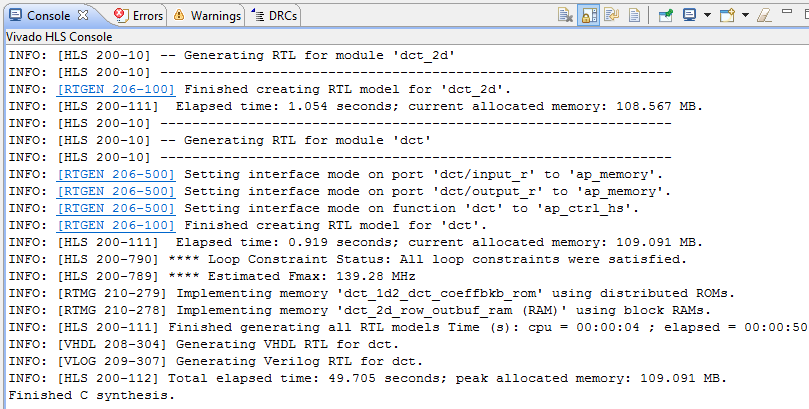


Рис. 1.3. Successful result of synthesis

* 1. Производительность

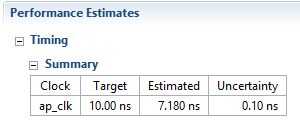


Рис. 1.4. Performance estimates

Полученная величина задержки укладывается в заданное значение.

* 1. Использование ресурсов

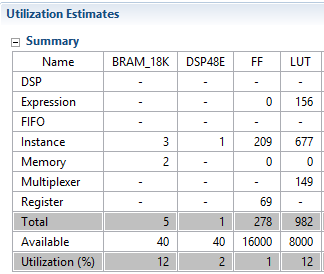


Рис. 1.5.1. Utilization estimates

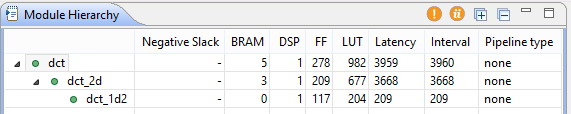


Рис. 1.5.2. Module hierarchy

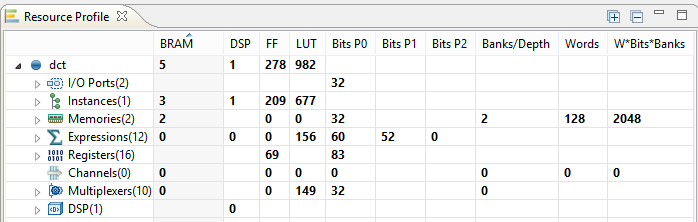


Рис. 1.5.3 Resource profile

Данное решение потребует на микросхеме 982 элемента LUT, 278 регистров, 1 DSP48E и 5 блоков RAM\_18K.

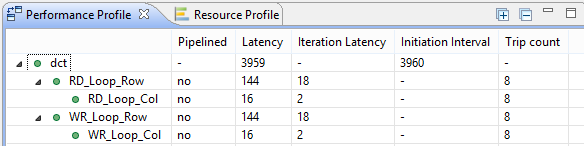


Рис. 1.5.4. Performance Profile

Задержка составляет 3959 тактов, интервал инициализации – 3960.

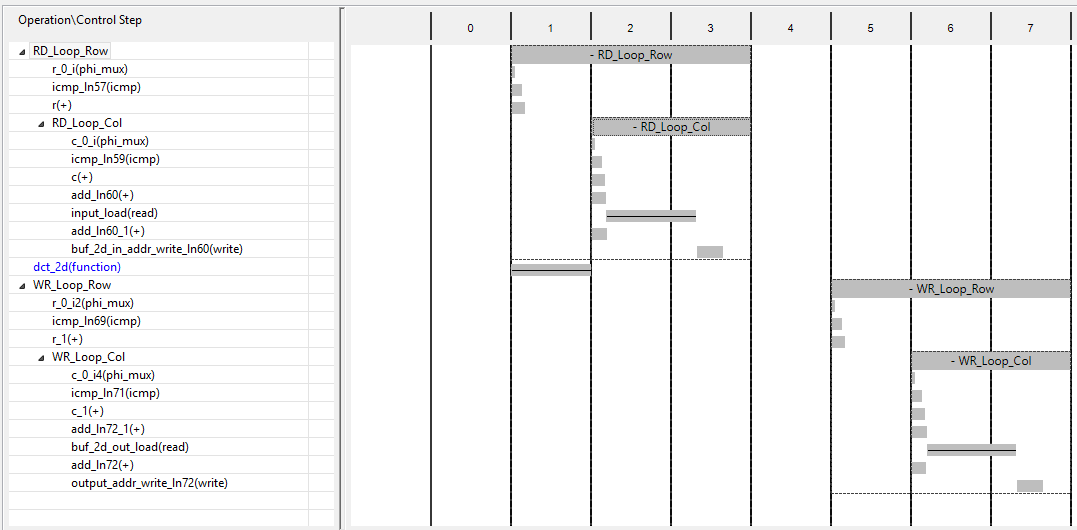


Рис. 1.5.5. Operation\Control Step

Максимальная величина задержки для dct\_2d равна 3668. Максимальная величина задержки для dct\_1d2 равна 209.

1. Второе решение
   1. Создание решения и применение директивы Pipeline

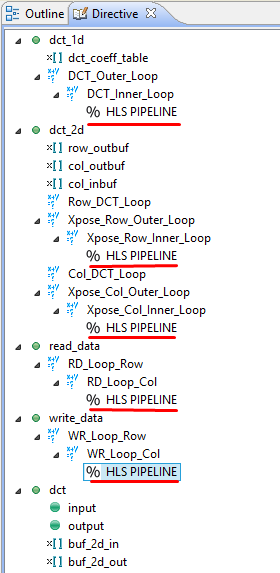


Рис.2.1 Directive Pipeline

* 1. Синтез

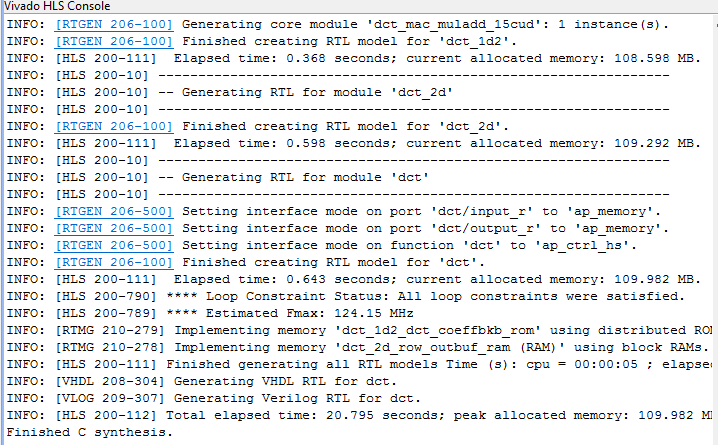


Рис. 2.2. Successful result of synthesis

* 1. Производительность

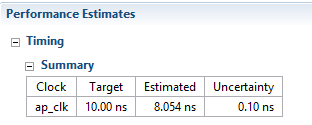


Рис. 2.3 Performance estimates

Величина задержки укладывается в заданное значение.

* 1. Сравнение решений

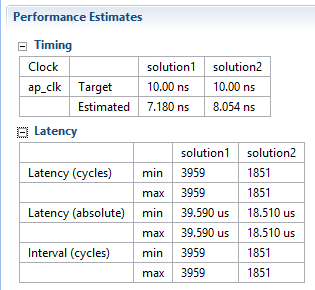


Рис. 2.4 Comparing solutions

На изображении выше видно значительное уменьшение количества тактов в задержке получения решения и интервале инициализации.

Максимальное значение задержки для второго решения составляет 1851 такт.

* 1. Использование ресурсов

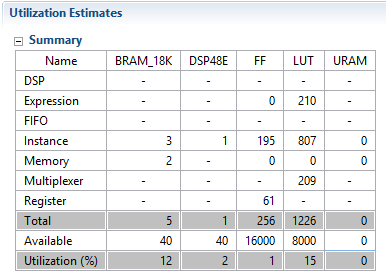


Рис. 2.4. Utilization estimates

Данное решение потребует на микросхеме 1226 элементов LUT, 256 регистров и как и в первом решении 1 DSP48E и 5 блоков RAM\_18K.

Максимальное значение задержки для dct\_2d равно 1718. Максимальное значение задержки для dct\_1d2 равно 97.

1. Третье решение
   1. Изменение директив

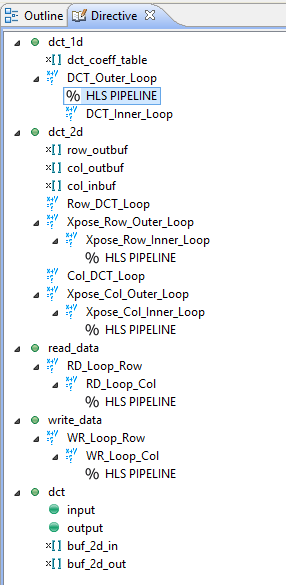


Рис. 3.1. Changing a directive

* 1. Синтез

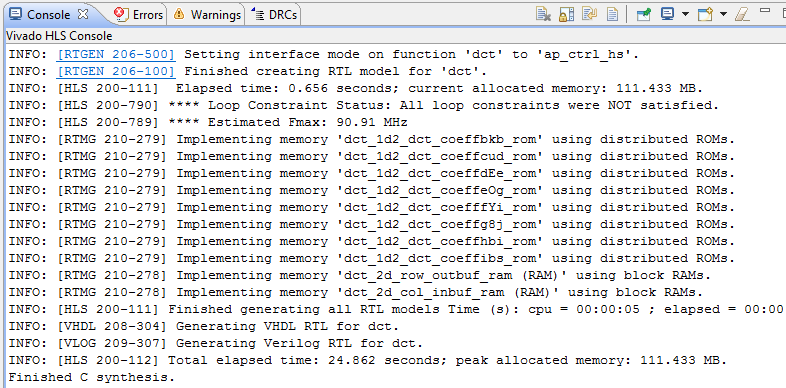


Рис. 3.2. Successful result of synthesis

* 1. Сравнение решений

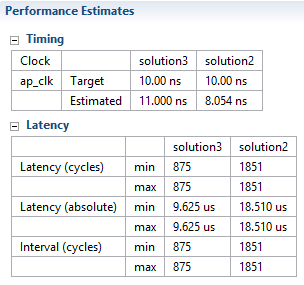


Рис. 3.2. Comparing solutions

На изображении выше видно, что время задержки третьего решения значительно выше, но при этом, количество тактов практически в 2,5 раза меньше.

1. Выводы

В ходе выполнения работы было получено три решения и проведено сравнение величин задержек в этих решениях. Директива PIPELINE встраивает конвейерные регистры в сгенерированный RTL, чтобы улучшить значение задержки. При правильном применении этой директивы есть возможность увеличить пропускную способность проекта.