- Создать проект lab4_z5
- Микросхема: xa7a12tcsg325-1q
- ***N = 16384, scale = { два случайных числа).***
- Создать две функции (см. Текст ниже) – исходную и модифицированную - и провести их анализ.

Conditional Execution of Tasks

The DATAFLOW optimization does not optimize tasks that are conditionally executed. The following example highlights this limitation. In this example, the conditional execution of Loop1 and Loop2 prevents Vivado HLS from optimization the data flow between these loops, because the data does not flow from one loop into the next.

```
void foo_b(int data_in1[N], int data_out[N], int scale[2], char sel) {
int temp1[N], temp2[N];

if (sel==0) {
Loop1: for(int i = 0; i < N; i++) {
temp1[i] = data_in[i] * scale[0];
temp2[i] = data_in[i] * scale[1];
}
}
else {
Loop2: for(int j = 0; j < N; j++) {
temp1[j] = data_in[j] * scale[1];
temp2[j] = data_in[j] * scale[0];
}
}

Loop3: for(int k = 0; k < N; k++) {
data_out[k] = temp1[k] / temp2[k];
}

}
```

To ensure each loop is executed in all cases, you must transform the code as shown in the following example. In this example, the conditional statement is moved into the first loop and second loop. Both loops are always executed, and data always flows from one loop to the next.

```
void foo_m(int data_in[N], int data_out[N], int scale[2], char sel) {
int temp1[N], temp2[N];

Loop1: for(int i = 0; i < N; i++) {
if (sel==0) {
temp1[i] = data_in[i] * scale[0];
}
else {
temp1[i] = data_in[i] * scale[1];
}
}

Loop2: for(int j = 0; j < N; j++) {
if (sel==0) {
temp2[j] = data_in[j]* scale[1];
}
else {
temp2[i] = data_in[i] * scale[0];
}
```

```
}

Loop3: for(int k = 0; k < N; k++) {
data_out[k] = temp1[k] * temp2[k];
}

}
```

- Создать тест lab4_z5_test.c для проверки функций выше (тест должен обеспечивать запуск функций при двух значениях sel: =0 и !=0).
- Для функции **foo_b**
    - задать: clock period , **выбранный в lab4_z3**; clock_uncertainty 0.1
    - осуществить моделирование (с выводом результатов в консоль)
    - осуществить синтез для:
        - привести в отчете:
            - performance estimates=>summary
            - utilization estimates=>summary
            - scheduler viewer (выполнить Zoom to Fit)
                - На скриншоте показать Latency
                - На скриншоте показать Initiation Interval
- Для функции **foo_m**
    - задать: clock period, **выбранный в lab4_z3**; clock_uncertainty 0.1
    - осуществить моделирование (с выводом результатов в консоль)
    - осуществить синтез для случая **FIFO for the memory buffers**:
        - привести в отчете:
            - performance estimates=>summary
            - utilization estimates=>summary
            - scheduler viewer (выполнить Zoom to Fit)
                - На скриншоте показать Latency
                - На скриншоте показать Initiation Interval
            - **Dataflow viewer**
    - осуществить синтез для случая **ping-pong buffers**:
        - привести в отчете:
            - performance estimates=>summary
            - utilization estimates=>summary
            - scheduler viewer (выполнить Zoom to Fit)
                - На скриншоте показать Latency
                - На скриншоте показать Initiation Interval
            - **Dataflow viewer**

    - Осуществить C|RTL моделирование для случая **FIFO for the memory buffers**
        - Привести результаты из консоли
        - Открыть временную диаграмму (все сигналы)
            - Отобразить два цикла обработки на одном экране
                - На скриншоте показать Latency
                - На скриншоте показать Initiation Interval
    - **Написать tcl файл автоматизирующий исследование**
- Выводы
    - Объяснить отличия в синтезе foo_b и двух вариантов foo_m между собой

Исследование времени выполнения на ПК

- **Используются исходные коды и результаты исследования проведенного ранее.**
- На базе использованного выше Си теста создать отдельный, модернизированный, тест для проверки времени выполнения синтезируемой функции на ПК:
    - добавить в тест операторы измерения **времени выполнения** синтезируемой функции (например, как-то так: https://solarianprogrammer.com/2019/04/17/c17-programming-measuring-execution-time-delaying-program/).
    - Увеличить количество запусков синтезируемой функции до 32. Для каждого запуска измерить время, найти среднее значение и вывести как результат.
    - Точность измерения времени (наносекунды).
    - Провести исследование времени выполнения синтезируемой функции на Вашем ПК
        - Осуществить компиляцию модернизированного теста и запустить его как отдельное приложение
        - В отчете привести:
            - Параметры Вашего ПК: тип процессора, частота работы процессора, объем ОЗУ
            - результаты измерения времени выполнения

- Оформить отчет, который должен включать
    - Задание
    - Раздел с описанием исходного кода функции
    - Раздел с описанием теста
    - Раздел с описание созданного командного файла
    - Раздел с анализом результатов (со снимками экрана с заполненной таблицей и полученным графиком)
        - Анализ и выбор оптимального (критерий максимальная производительность) решения
    - Результаты исследования времени выполнения на ПК и сравнение с аппаратными решениями.
    - Выводы

  Архив должен включать всю рабочую папку проекта (включая модернизированный тест и скомпилированное приложение), отчет