* Создать проект lab4\_z3
* Микросхема: xa7a12tcsg325-1q
* ***N = 16384, scale = { три случайных числа).***
* Создать две функции (см. Текст ниже) – исходную и модифицированную - и провести их анализ.

**Single-producer-consumer**

***For Vivado HLS to perform the DATAFLOW optimization, all elements passed between tasks***

***must follow a single-producer-consumer model. Each variable must be driven from a single task***

***and only be consumed by a single task. In the following code example, temp1fans out and is***

***consumed by both Loop2and Loop3. This violates the single-producer-consumer model.***

*void foo\_b (int data\_in[N], int scale[3], int data\_out1[N], int data\_out2[N]) {*

*int temp1[N];*

*Loop1: for(int i = 0; i < N; i++) {*

*temp1[i] = data\_in[i] \* scale[0];*

*}*

*Loop2: for(int j = 0; j < N; j++) {*

*data\_out1[j] = temp1[j] \* scale[1];*

*}*

*Loop3: for(int k = 0; k < N; k++) {*

*data\_out2[k] = temp1[k] \* scale[2];*

*}*

*}*

***A modified version of this code uses function Split to create a single-producer-consumer***

***design. In this case, data flows from Loop1 to Split and then to Loop2 and Loop3.***

***The data now flows between all four tasks, and Vivado HLS can perform the DATAFLOW***

***Optimization***

*void Split (in[N], out1[N], out2[N]) {*

*// Duplicated data*

*L1:for(int i=1;i<N;i++) {*

*out1[i] = in[i];*

*out2[i] = in[i];*

*}*

*}*

*void foo\_m(int data\_in[N], int scale[3], int data\_out1[N], int data\_out2[N]) {*

*int temp1[N], temp2[N]. temp3[N];*

*Loop1: for(int i = 0; i < N; i++) {*

*temp1[i] = data\_in[i] \* scale[0];*

*}*

*Split(temp1, temp2, temp3);*

*Loop2: for(int j = 0; j < N; j++) {*

*data\_out1[j] = temp2[j] \* scale[1];*

*}*

*Loop3: for(int k = 0; k < N; k++) {*

*data\_out2[k] = temp3[k] \* scale[2];*

*}*

*}*

* Создать тест lab4\_z3\_test.c для проверки функций выше.
* Для функции **foo\_b**
  + задать: clock period 6, 8, 10, 12; clock\_uncertainty 0.1
  + осуществить моделирование (с выводом результатов в консоль)
  + осуществить синтез и выбрать лучший вариант (максимум производительности при наименьших аппаратных затратах), привести результаты сравнения вариантов + таблица.
    - Для выбранного варианта привести в отчете:
      * performance estimates=>summary
      * utilization estimates=>summary
      * scheduler viewer (выполнить Zoom to Fit)
        + На скриншоте показать Latency
        + На скриншоте показать Initiation Interval
      * resource viewer (выполнить Zoom to Fit)
  + **Написать tcl файл автоматизирующий исследование**
* Для функции **foo\_m**
  + задать: clock period выбранный на предыдущем этапе; 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 исследования**
* 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