CS-470: Homework Set #3
High-level Synthesis

In this homework, you are required to optimize 5 High-Level Synthesis (HLS) designs. The main goal of the optimization is to shrink the number of cycles to finish the task defined in the hardware, without changing the given clock speed. You can apply any optimizations (unroll, pipeline, memory partition, false dependency hint, and modifying the code), as long as the function of the design is the same as before. You are also required to explain your optimizations, and discuss their positive points, and potentially their negative ones.

For each of the provided kernels, there is a set of tricks one can play to improve their performance. The goal is not to get the absolute best performance, but rather to identify this set of tricks and to apply them appropriately.

This homework should be done alone.

1 Designs

```
void kernel1 (int array[ARRAY_SIZE]) {
    int i;
    for(i = 0; i < ARRAY_SIZE; i++)
        array[i] = array[i] * 5;
}
```

```
void kernel2 (int array[ARRAY_SIZE]) {
    for(int i = 3; i < ARRAY_SIZE; i++)
        array[i] = array[i - 1] + array[i - 2] array[i - 3];
}
```

```
void kernel3 (float hist[ARRAY_SIZE], float weight[ARRAY_SIZE], int index[ARRAY_SIZE])
     {
    for (int i = 0; i < ARRAY_SIZE; ++i)
        hist[index[i]] = hist[index[i]] + weight[i];
}
```

```
void kernel4 (int array[ARRAY_SIZE], int index[ARRAY_SIZE], int offset) {
    for (int i = offset + 1; i < ARRAY_SIZE - 1; ++i)
        array[offset] = array[offset] - index[i] * array[i] + index[i] * array[i + 1];
}
```

```
float kernel5 (float bound, float a[ARRAY_SIZE], float b[ARRAY_SIZE]) {
    int i = 0;
    float sum = 0;
    while (sum < bound && i < ARRAY_SIZE) {
        sum = a[i] + b[i];
        i++;
    }
    return sum;
}
```

Besides the design file itself, we also provide a testbench and a tcl script for each design so that you can quickly establish the Vitis HLS project. By default, we turn off the automatic pipelining optimization so that you control everything.

2 Submission Requirement

For each design you optimize, you are required to submit the following:

- The modified code, with `HLS pragma` in the same file. You may also briefly explain the pragma you apply by writing comments in the same file.
- The scheduled dataflow graph of the single loop body. We will provide an example in this document.
- The interval reported by Vitis HLS, and a brief explanation why the interval is not one.
- The total number of cycles required to finish the task. Usually this number should be reported by Vitis HLS. However, if Vitis HLS fails to estimate the value, you have to estimate the number by yourself.
- A short explanation of the optimizations you applied, and why. If they come with downsides, also discuss them.

For designs that you think no optimization can be applied, you don't have to submit the modified code, but instead a reason why no further optimization is possible.

3 Example

We will use a simple design described in Figure 1. In the design, during each iteration, one element from array `a` and two elements from array `b` are read, processed, and stored to one specific position of array `o`.

```
void simpleDesign(int a[1024], int b[2048], int o[1024]){
    for(int i = 0; i < 1024; ++i){
        o[i] = a[i] + b[2*i] * b[2*i+1];
    }
}
```

Figure 1: The HLS design used by the example

Since, there is no inter-loop dependency, we may consider apply a `PIPELINE` pragma to the loop. We can also partition the array `b` so that we can access the two elements `b[2*i]` and `b[2*i+1]` in the same cycle. You may also apply `UNROLL` pragma to further shrink the time, but here as an example, we don't consider the further optimization. The optimized code is displayed in Figure 2.

Next, we draw the scheduled dataflow graph. In this graph, you must explicitly show every hardware unit, both the starting time as well as the latency of each operation, and the data dependency among

```
void simpleDesign(int a[1024], int b[2048], int o[1024]){
    #pragma HLS array_partition variable=b type=cyclic factor=2
    loop: for(int i = 0; i < 1024; ++i){
    #pragma HLS PIPELINE
        o[i] = a[i] + b[2*i] * b[2*i+1];
    }
}
```

Figure 2: The optimized version of the design

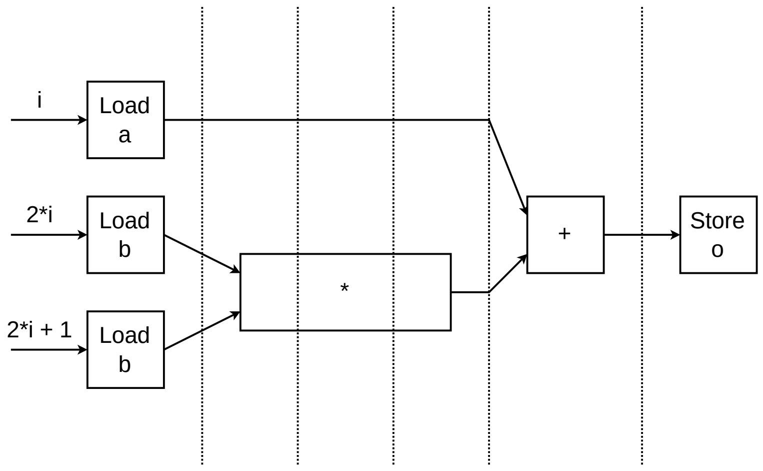

Figure 3: Scheduled dataflow graph of the loop body

hardware units. Vitis HLS should already suggest the schedule viewer, so the only task is to understand the schedule and draw the figure. Since most of the pipeline registers generated by Vitis HLS are anonymous, you can use dashed lines to represent them for your convenience. In the example, we assume 3 load operations happen in the first cycle and both load operation and store operation have one cycle latency. Add operation has 1 cycle latency and the multiply operation has 3 cycles latency, and they are scheduled as soon as its operand is ready. As a result, the dataflow graph is displayed in Figure 3.

You don't have to draw the same graph multiple times if you apply the `UNROLL` pragma in the top loop.

Since we also know that there is no inter-loop dependency in the design, the II reported by Vitis HLS is 1. And as a result, the total number of cycles to finish the task defined by the HLS design, is equal to number of iterations - 1 + single loop latency, which is 1029.