Homework 3

Vivado HLS

There are 8 kernels provided. Each is a simple synthetic example of a programmatic structure for which one would like to get a good hardware implementation using HLS.

For each kernel do the following:

1. Synthesize the naïve code in Vivado HLS and analyze the produced schedule.
2. Optimize the design for performance, when possible, by using the appropriate directives: pipeline, unroll, dependence, and partition; rewrite the code where necessary. Hand-in the resulting code.
3. Specify whether an initiation interval of 1 is achievable for the loop body, or report what is the best initiation interval you can achieve.
4. Explain the optimizations you performed and comment on the resulting performance. If an iteration interval of 1 is not achievable, explain clearly what in your opinion prevents Vivado HLS to improve the performance further.
5. Compare the resource requirements of the naïve design with those of your optimized design.

For each kernel, there is a tcl script to generate the project and a testbench to verify functionality.

Please note that Vivado HLS may fail to report function latency and initiation interval due to parameterized loop boundaries. In such case, estimate the latency and initiation interval from the HLS schedule.

Your optimized design must be constrained to the FPGA resources and must meet the clock period specified in the projects (10 ns) with extra 2 ns slack (worst case of 12 ns period).

Compile your answers in a report. Submit the report along with the cpp files for the optimized kernels and a zip file of the HLS solution(s) created.

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

```
int kernel6 (int x) {
    int i = 0;
    while (i * i < x)
        i++;
    return i;
}
```

```
float kernel7 (float a[ARRAY_SIZE], float b[ARRAY_SIZE]) {
    float sum = 0;
    for (int i = 0; I < ARRAY_SIZE; i++) {
        float diff = a[i] - b[i];
        if (diff > 0)
            sum = (sum + diff);
    }
    return sum;
}
```

```
void kernel8(int array[ARRAY_SIZE], int multiplier, int offset) {
    for (int i = 6; i < ARRAY_SIZE – 1 - offset; ++i)
        array[i] = array[i – 6 + offset] * multiplier;
}
```