**Q1. Coarse-Grained Reconfigurable Array Architecture Exploration for MIMO application**

The goal of the assignment is to specify and design coarse-grained reconfigurable array optimized for MIMO application. Use following five steps to answer this question.

**Step#1 - Analyze MIMO application**
1. Write a C code to implement the algorithm of MIMO decoder application (pseudo code is provided in Annex-1).
2. Profile entire MIMO decoder application by using an embedded processor like ARM, PowerPC, etc. (Cross compile(say armcc) and get the execution cycle count from the Simulator(say armsd).
3. Make a table showing profiling result and your analysis.

<table>
<thead>
<tr>
<th>functions</th>
<th>operation set</th>
<th>conditional branch</th>
<th>maximum data bit-width</th>
<th>Number of input/output</th>
<th>Execution Cycle Count</th>
<th>Ratio (%) in Entire Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>add, sub, and, or</td>
<td>High</td>
<td>32-bit</td>
<td>8/2</td>
<td>102345 cycles</td>
<td>20%</td>
</tr>
<tr>
<td>B</td>
<td>high</td>
<td></td>
<td></td>
<td></td>
<td>50268 cycles</td>
<td>10%</td>
</tr>
<tr>
<td>C</td>
<td>add, sub, shift</td>
<td>none</td>
<td>16-bit</td>
<td>8/8</td>
<td>152687 cycles</td>
<td>30%</td>
</tr>
<tr>
<td>D</td>
<td>add, sub, shift</td>
<td>low</td>
<td>16-bit</td>
<td>4/4</td>
<td>102369 cycles</td>
<td>20%</td>
</tr>
<tr>
<td>E</td>
<td>add, sub, shift</td>
<td>high</td>
<td>16-bit</td>
<td>2/4</td>
<td>4117 cycles</td>
<td>5%</td>
</tr>
<tr>
<td>F</td>
<td>add, sub, shift, and</td>
<td>high</td>
<td>32-bit</td>
<td>6/3</td>
<td>72934 cycles</td>
<td>15%</td>
</tr>
</tbody>
</table>

**Step#2 - Analyze base reconfigurable array architecture (Step#2 is necessary to go Step#4)**
1. Understand the entire structure of the base array architecture element given with the diagram provided (Annex-2).
2. The Processing element (PE) of the base reconfigurable array architecture at RT-level (Verilog) is provided (Annex-3).
3. Based on the base architecture analysis, you can consider what functions of MIMO can be mapped onto the reconfigurable array in Step#3. Otherwise, you can consider how to change the given base architecture to support the functions of MIMO well in Step#4. Step #3 and Step#4 are reconfigurable array architecture exploration process to generate optimized application source codes and architecture specification.

**Step#3 – MIMO source code optimization and parallelization**
1. Based on Step#1 and Step#2, you can select some candidate functions to be mapped onto reconfigurable array.
2. Remove all of the conditional branch statement in the selected functions if applicable. In addition, if the functions include 32-bit/64-bit data (variables), it should be modified to 20-bit data-type without hurting correctness of the function – If such a modification is not applicable, you can modify the
reconfigurable array architecture in step#4.

3. Figure 1 shows a parallelization example for matrix-vector multiplication assuming 4x1 reconfigurable array. 4x1 reconfigurable array means that the maximum 4 variables can be executed at the same time.

\[
\text{for (i = 0; i <= 3; i = i+1)} \\
\text{for (j = 0; j <= 3; j = j+1)} \\
\text{z[i] = x[i][j]*y[j] + z[i];}
\]

(a)

\[
\text{for (i = 0; i <= 3; i = i+1)} \\
\{ \\
\quad t1 = x[i][0]*y[0]; \\
\quad t2 = x[i][1]*y[1]; \\
\quad t3 = x[i][2]*y[2]; \\
\quad t4 = x[i][3]*y[3]; \\
\quad \text{Execute 1(EX1)} \\
\quad \text{Execute 2(EX2)} \\
\quad \text{Execute 3(EX3)} \\
\}
\]

(b)

Figure 1: Matrix-vector multiplication algorithm, C-code: (a) before parallelization and (b) after parallelization.

**Step#4 - Change and optimize base array structure at RT-level**

1. Based on Step#3, you can modify or optimize the base array structure.
   
   For example,
   
   a. If new interconnections are required to support the selected functions, you can add them.
   
   b. If new functional block other than adder or Multiplier is required you can add or remove them.
   
   c. If 32/64 bit data operations are necessary, you can extend the bit-width of the base architecture.

Then change the verilog code accordingly.

**Step#5 – RTL model simulation and verification**

1. Based on Step#4, you can verify your RTL implementation using logic simulator (for example vcs)
2. For functional verification, you can implement a simple test bench to check the correctness of the implemented model and compare with the output from C program.
3. Then you can synthesize the RTL implementation.
4. Make a table as below to show synthesis results.

<table>
<thead>
<tr>
<th>Area</th>
<th>Power</th>
<th>Critical path delay (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Items to turn-in for grading:

1. Analysis Table in Step #1

2. Optimized and parallelized source code of the functions in MIMO in Step#3 (Comment the changes you are making in the original C code).

3. Modified RTL source code of reconfigurable array architecture in Step#4 (Annotate the changes).

4. Synthesis result table in Step#5.

5. Functional verification by comparing output of verilog simulator with the C code. Provide the testbench program used in Step 5.

If you have questions, please contact at
Yoonjin Kim, awgsize@gmail.com
Suman Mandal, sumankalyan@gmail.com