CO503 : Practical 4 - JPEG MPSoC Design Project

In this practical, you will use FPGA design tools to implement an MPSoC for JPEG image encoding. You will need to use all the new skills and knowledge you have acquired in the previous practicals as well as some extra research on your own to successfully complete this practical.

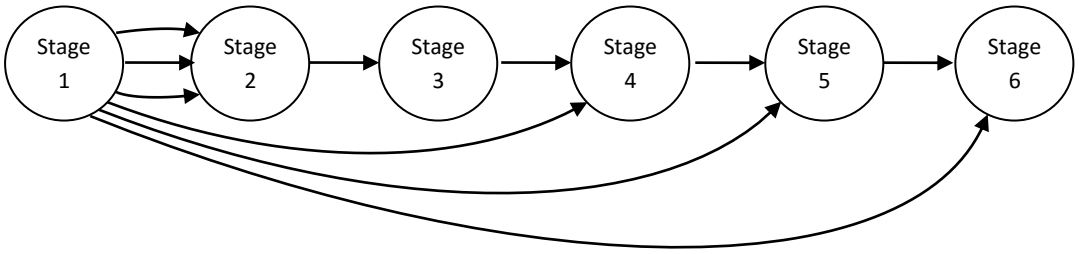
Part 1: Pipelined JPEG MPSoC

We have broken down our application program, JPEG Encoding, into six separates stages of operation:

- Stage 1: Read a macro-block from raw image, convert RGB to YCbCr format
- Stage 2: Level shifting
- Stage 3: Vertical and horizontal DCT
- Stage 4: Quantization
- Stage 5: Huffman encoding
- Stage 6: Write converted macro-block to JPEG image

We need these six stages to work as a pipeline, that is a macro-block from the raw image travels through the six stages to get converted into JPEG. In the steady state of operation each stage should be working on a macro-block at any given time.

The work of each stage should be carried out by a CPU. The communication between the stages should be implemented by FIFO queues, as shown in the diagram below:



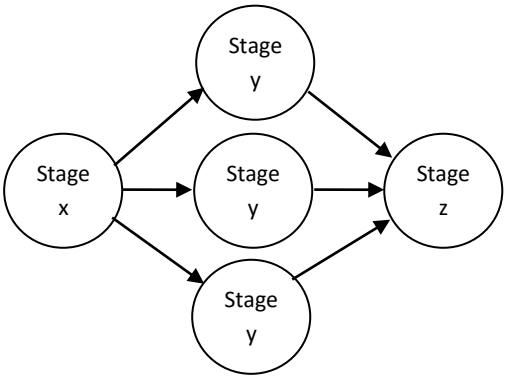
You are given an implementation of a JPEG encoder as described above, which uses Tensilica Xtensa embedded CPUs and hardware FIFO queues. Refer to the provided research paper to understand the implementation details.

Your task is to implement the same on FPGA hardware using Nios II CPUs (one for each stage), and hardware FIFO queues for communication between stages. Depending on the workload at each stage, some stages may operate faster than others. Your pipeline's performance will be limited by the slowest stage, so faster stages will need to wait (stall) until input data is ready or output FIFOs have vacant space. Once you have synchronized the system to work properly, measure the throughput of your pipeline (how often an encoded macro-block is generated in the steady state) and how much time each stage is consuming.

Part 2: Improving Performance

Your final task is to improve performance (throughput) of your system. That is to improve the ate you can generate encoded macro-blocks when in steady state. Here are some suggestions as to how you may gain better performance:

1. Processor customizations (custom instructions for frequently used slow operations, include instruction/data caches in your processor, configure instruction/data caches to suit the program, etc.)
2. Extend the pipeline (divide slow stages into multiple smaller stages)
3. Superscalar pipelines (if a particular stage is a lot slower compared to the others, have multiple CPUs for that stage which work in parallel on portions of data, i.e. four processors with each working on one quarter of the input macro-block. See example below)



4. The given research paper provides an analysis on how to improve performance of the system. Read it, and find other similar research papers for ideas.

Show your work once you have managed to obtain the best possible throughput. Then prepare a comprehensive report on how you managed to improve performance detailing the techniques you used.