Designing a JPEG DCT in C/C++

using Catapult Synthesis

June 2017

Mentor Graphics Corporation
8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777.
Telephone: 503.685.7000
Toll-Free Telephone: 800.592.2210
Website: www.mentor.com
SupportNet: supportnet.mentor.com/

Introduction

NOTE: This training material assumes that the reader has taken the Catapult Basic Training. Many of the Basic Training concepts are discussed and used in this document without giving detailed background information.

This class covers the design and optimization of a 2-D Discrete Cosine Transform (DCT), which is one of the functional units in a JPEG pixel-pipe image compression engine. The pixel-pipe engine takes a stream of RGB image data and compresses it into a bit stream.

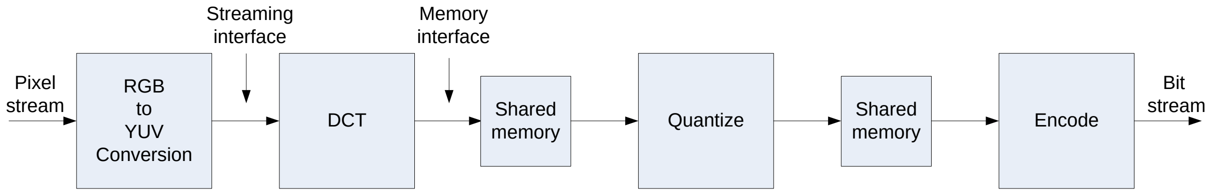

Figure 1 Pixel-pipe Hardware Architecture

The 2-D DCT algorithm transforms an 8x8 array of image pixels into an 8x8 array of spatial frequency values. This is done as a two-step process using two 1-D filters and some storage to store the intermediate calculations. The 1-D filters access of the intermediate array data is in column and row order respectively. Because of this the vertical processing filter cannot start until the horizontal processing filter has completed. This is an important aspect of the algorithm to understand in order to architect a high-performance hardware implementation

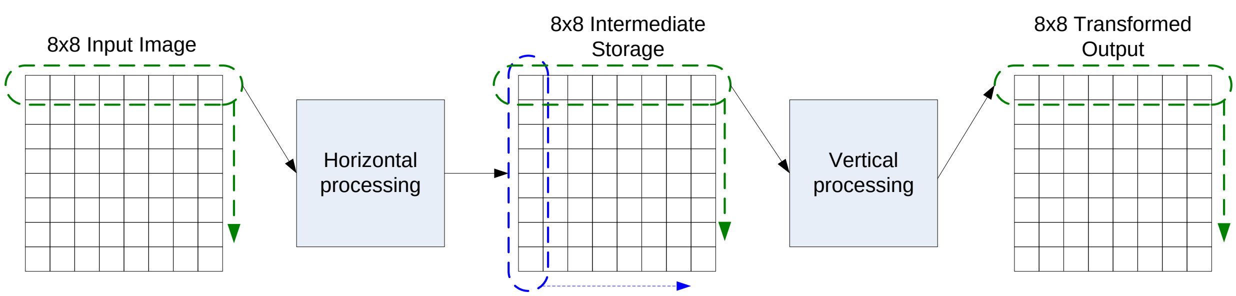

Figure 2 2-D DCT Algorithm

The design goals of this class are:

- Make the original design synthesizable with a streaming input interface
- Analyze the design in Catapult to find bit-widths that can be optimized
- Quantize the design using bit-accurate data types to reduce area

- See the effects of loop pipelining/loop flattening to achieve a design throughput of under 1200 cycles
- Use the Catapult automated verification flow to verify the design throughput and functionality
- Understand pipelined scheduling failures due to memory resource competition
- Make an architectural code change to pass scheduling and achieve a throughput of 128 cycles
- Make an architectural code change to reduce area
- Use Catapult "Hierarchy" to build a high-performance multi-block DCT with a throughput of 64 cycles
- Use Catapult CCOREs to reduce area

Algorithmic DCT

The starting point for a Catapult C design is often a piece of C++ code that was written by an algorithm or systems engineer. This code is often not suitable for C++ synthesis since it may contain non-synthesizable constructs such as new/malloc or certain types of pointer operations that are not suitable for synthesis. Furthermore, even when synthesizable, the C++ source may not have the architectural details that are sufficient to achieve the desired interfaces, performance, or area goals.

Compile the Design

- CD into DCT_Algorithm directory and launch Catapult by typing "catapult" at the command prompt (This step can be skipped if running the toolkit interactively).
- Go to File > Run Script and run the directives1.tcl file. If running interactively select the "Compile the original design" script and click on "Launch Project" in the toolkit window. This will attempt to compile and synthesize the design.
- Look in the Catapult transcript window and note that there is a compilation error indicating that pointers-to-pointers is not supported on the design interface.

```
#  Message
#  Found top design routine 'dct' specified by directive
#  Synthesizing routine 'dct'
   Unsupported synthesis feature 'pointers-to-pointers on the interface'
   Compilation aborted
   Completed transformation 'analyze' on solution 'solution.v1': elapsed ti
#  > end dofile /scratch1/sb8dbg/main/ixl/Mgc_home/pkgs/ccs_toolkits_inhous
   go compile: Failed compile
```

Error Analysis

- Double-click on the error message to cross-probe to the C++ source
- Look at the design interface and note that the interface uses arrays or pointers for the "input" and "output" arrays. There are two problems with this implementation, one is that arrays of pointers, or pointers to pointers, is not supported. The second problems is that this "algorithmic" implementation does not match the interface requirements of the pixel-pipe architecture, which needs a streaming interface for the DCT input.

```
// ----------------------------------------
// DCT 2D 8x8 Top
void dct(char *input[8], short *output[8]) {
    const int coeff[8][8] = {
```