Mentor®
A Siemens Business

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/

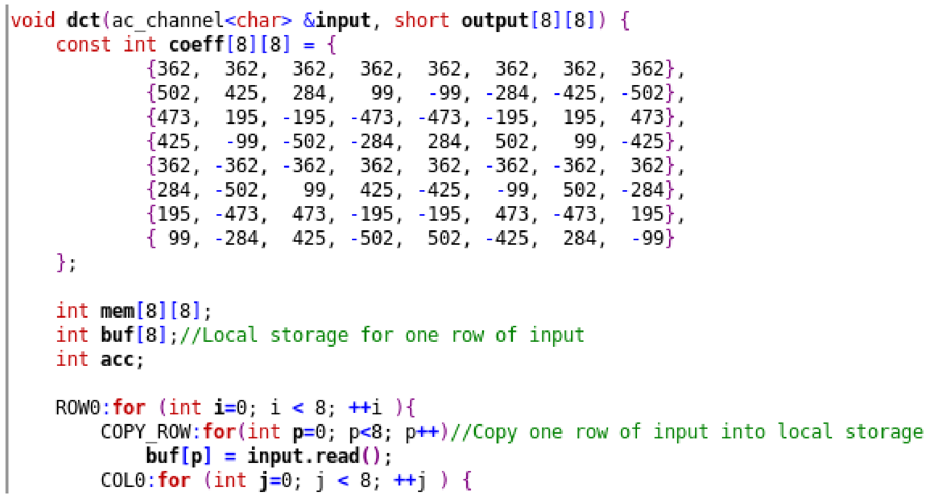
- Go to the next toolkit example

Adding ac_channel Streaming Input Interface

This Toolkit converts "input" into an ac_channel streaming interface and makes "output" an explicit array by defining both array dimensions.

Synthesize the Design

- CD into the DCT_channelized 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 "Synthesize the design" script and click on "Launch Project" in the toolkit window. This sets the technology to Sample 65nm, adds single-port RAM libraries, sets the clock to 300 MHz, and synthesizes the unconstrained design.
- Open the "dct.cpp" file a look at the design interface.
    - The interface has been re-written so that "input" is an ac_channel interface of type "char".
    - "output" has been explicitly defined as an 8x8 array.
    - A copy loop has been added to copy one row of the input image into a local storage buffer "buf[8]". The algorithm then operates on the local array "buf".



```
void dct(ac_channel<char> &input, short output[8][8]) {
    const int coeff[8][8] = {
            {362,  362,  362,  362,  362,  362,  362,  362},
            {502,  425,  284,   99,  -99, -284, -425, -502},
            {473,  195, -195, -473, -473, -195,  195,  473},
            {425,  -99, -502, -284,  284,  502,   99, -425},
            {362, -362, -362,  362,  362, -362, -362,  362},
            {284, -502,   99,  425, -425,  -99,  502, -284},
            {195, -473,  473, -195, -195,  473, -473,  195},
            { 99, -284,  425, -502,  502, -425,  284,  -99}
    };

    int mem[8][8];
    int buf[8];//Local storage for one row of input
    int acc;

    ROW0:for (int i=0; i < 8; ++i ){
        COPY_ROW:for(int p=0; p<8; p++)//Copy one row of input into local storage
            buf[p] = input.read();
        COL0:for (int j=0; j < 8; ++j ) {
```

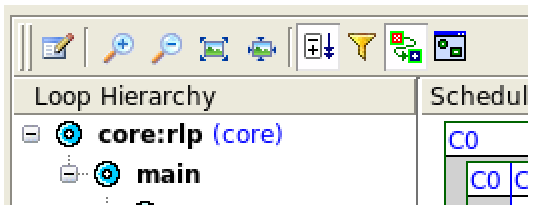
Architectural Constraints View

- Click on "Architecture" in the task bar to open the Architectural Constraints view.
- Expand the interface and arrays folder.
- Note that "input" has been auto-mapped to a "wait" handshaking interface and the "output" and "mem" arrays have been mapped to single-port RAM.

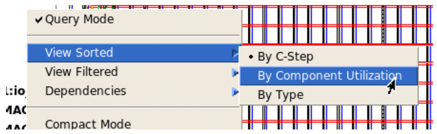
One of the things you may notice while looking at the dct.cpp file is that the internal variables are all defined as type "int". While it is ok to do this for constant arrays and loop variables in a bounded loop, it is not a good idea for internal arrays and some variables. Usually the algorithm should be "quantized" to use bit-accurate data types if Catapult cannot automatically reduce the internal bit-widths. The two places to analyze bit-widths are the Gantt chart and the RTL report file.

Scheduling and Analysis

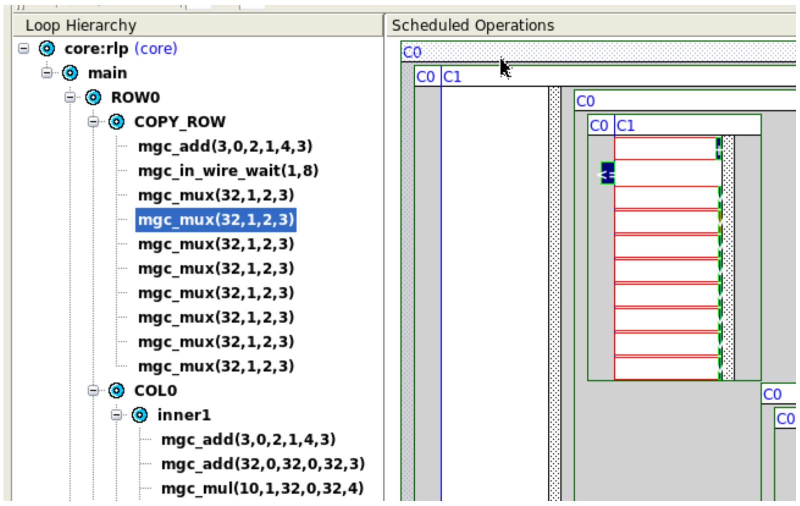
- Click on "Schedule" in the task bar and expand all the loops in the Gantt chart by clicking on the "down arrow" icon.



- Right-click in the Gantt chart and select "View Sorted > Component Utilization"



- The component utilization view shows the input bit-widths of the schedule operations. For a MUX this is the first number for the listed operation names. For adders and multipliers the first 4 numbers indicate the width and signedness of the "A" and "B" inputs of the operation. Look at the bit-widths of the multiplexors and the adders and note that they are 32-bits. If you see 32-bits or greater it's usually something that should be investigated.



```
Loop Hierarchy
⊟ core:rlp (core)
  ⊟ main
    ⊟ ROW0
      ⊟ COPY_ROW
          mgc_add(3,0,2,1,4,3)
          mgc_in_wire_wait(1,8)
          mgc_mux(32,1,2,3)
          mgc_mux(32,1,2,3)
          mgc_mux(32,1,2,3)
          mgc_mux(32,1,2,3)
          mgc_mux(32,1,2,3)
          mgc_mux(32,1,2,3)
          mgc_mux(32,1,2,3)
          mgc_mux(32,1,2,3)
      ⊟ COL0
        ⊟ inner1
            mgc_add(3,0,2,1,4,3)
            mgc_add(32,0,32,0,32,3)
            mgc_mul(10,1,32,0,32,4)
```

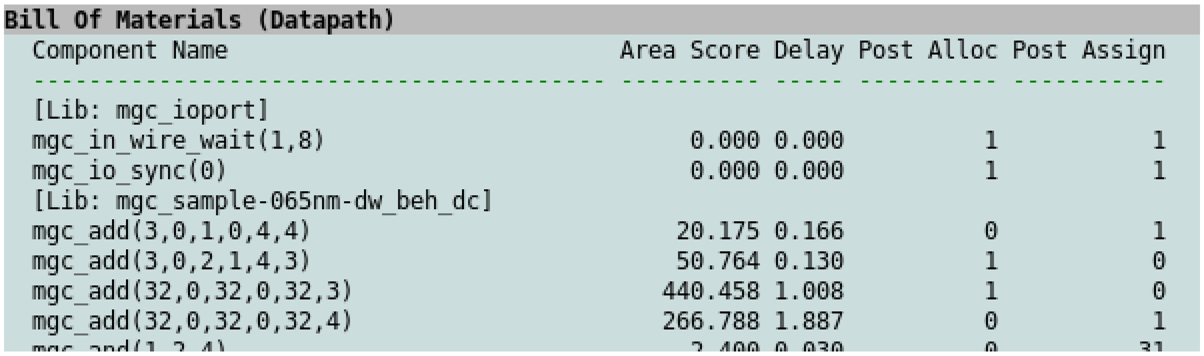
- Double-click on one if the 32-bit MUXes to cross-probe back to the C++.
- The 32-bit MUXes are created for "buf" which is defined as a 32-bit integer.

```
int mem[8][8];
int buf[8];//Local storage for one row of input
int acc;

ROW0:for (int i=0; i < 8; ++i ){
    COPY_ROW:for(int p=0; p<8; p++)//Copy one row of input into local storage
        buf[p] = input.read();
```

Reports and Analysis

- Expand the "Output Files Folder" then expand the "Reports" folder.
- Double-click on the RTL report.
- Go to the Bill of Material (BOM) and note a number of 32-bit components such as adders, muxes, registers, etc.



```
Bill Of Materials (Datapath)
  Component Name                            Area Score Delay Post Alloc Post Assign
  ----------------------------------------- ---------- ----- ---------- -----------
  [Lib: mgc_ioport]
  mgc_in_wire_wait(1,8)                          0.000 0.000          1           1
  mgc_io_sync(0)                                 0.000 0.000          1           1
  [Lib: mgc_sample-065nm-dw_beh_dc]
  mgc_add(3,0,1,0,4,4)                          20.175 0.166          0           1
  mgc_add(3,0,2,1,4,3)                          50.764 0.130          1           0
  mgc_add(32,0,32,0,32,3)                      440.458 1.008          1           0
  mgc_add(32,0,32,0,32,4)                      266.788 1.887          0           1
  mgc_and(1,2,4)                                 2.400 0.030          0          31
```

- Scroll down to the Register to Variable Mapping section of the RTL report. This shows what design variables are associated with what register.