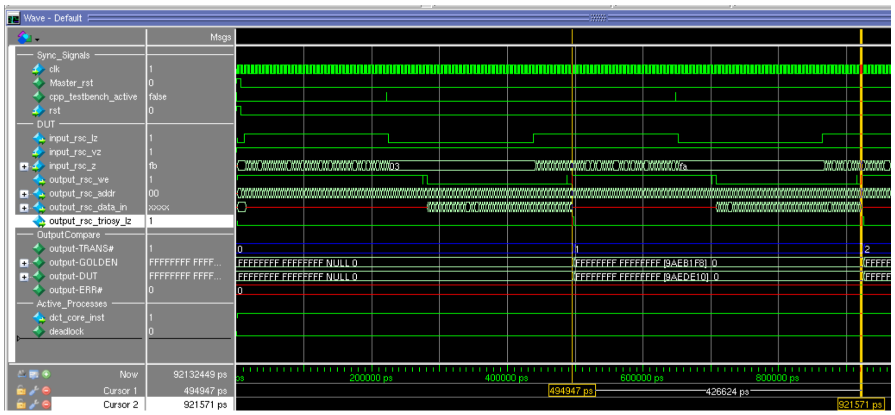
Designing a JPEG DCT in C/C++

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/



Looking for Improvements to Area

- Go to the Gant chart and view by Component Utilization. This view is useful to see the number and type of components used in a design. Scroll down to the bottom of the Gantt chart and observe that there are a large number of multipliers.
- Double-click on one of the multipliers and you will cross-probe to one of the "inner" loops that are multiplying some 8 data values against the values of "coeff".
- Scroll to the top of the dct.cpp file and look at the definition for "coeff". Observe that the coefficient values are symmetrical with a sign change every other line. We can exploit this symmetry to reduce the total number of multipliers.

```
void dct(ac_channel<ac_int<8> > &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}
    };
```

HLS, while very powerful, cannot automatically exploit coefficient symmetry. This is a design decision that must be reflected in the C++ code. Understanding that symmetrical coefficients allows us to "fold" filtering operations and reduce the number of multiplies in half is up to the designer.

- Go to the next Toolkit

Architectural Code Change for Better Area

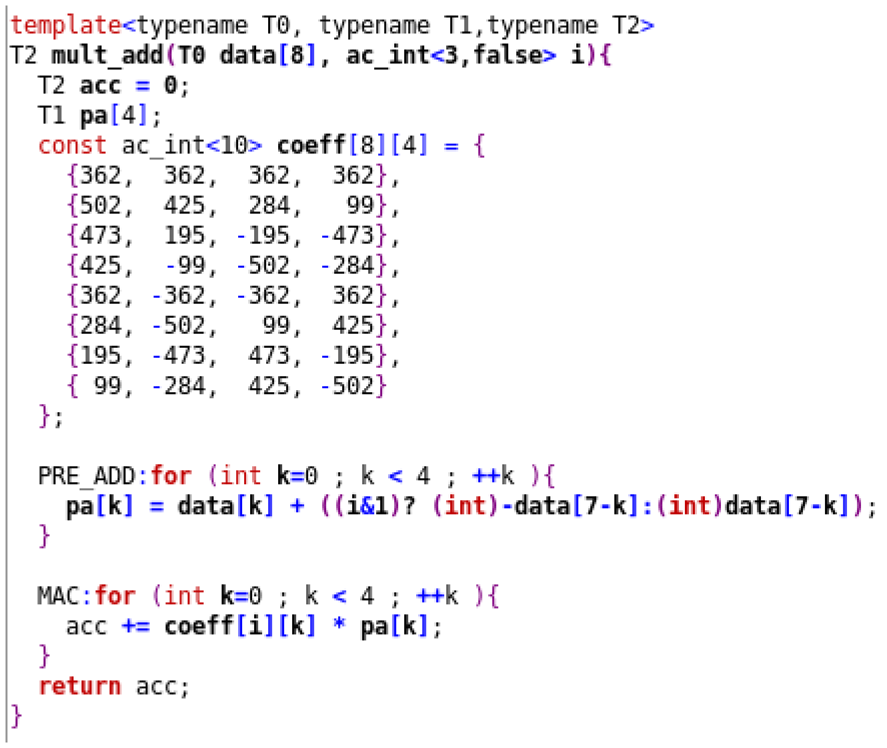
We saw in the previous Toolkit that the coefficients used in the DCT have symmetry that should allow the number of multipliers to be halved by making an architectural code change. This is the same kind of architectural decision that RTL designers make today when hand-coding RTL. However it is more likely to see C++ code written without this architectural feature unless it has been created by someone who is thinking about the kind of hardware they want. The key takeaway is that you should always be thinking about "what should the hardware look like" as you refine your C++ design.

Compile the Design

- CD into the DCT_Area 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 design" script and click on "Launch Project" in the toolkit window.

Review the Code Modifications

- Open the "dct.cpp" file and scroll to the top of the file. A templatized function "mult_add" has been created to implement the "inner1" and "inner2" loops. The C++ "template" keyword allows a C++ function to be parameterized for data type, number of array elements, bit-width, etc. This allows the function to be easily reused.



```
template<typename T0, typename T1,typename T2>
T2 mult_add(T0 data[8], ac_int<3,false> i){
  T2 acc = 0;
  T1 pa[4];
  const ac_int<10> coeff[8][4] = {
    {362,  362,  362,  362},
    {502,  425,  284,   99},
    {473,  195, -195, -473},
    {425,  -99, -502, -284},
    {362, -362, -362,  362},
    {284, -502,   99,  425},
    {195, -473,  473, -195},
    { 99, -284,  425, -502}
  };

  PRE_ADD:for (int k=0 ; k < 4 ; ++k ){
    pa[k] = data[k] + ((i&1)? (int)-data[7-k]:(int)data[7-k]);
  }

  MAC:for (int k=0 ; k < 4 ; ++k ){
    acc += coeff[i][k] * pa[k];
  }
  return acc;
}
```

- Look at the “coeff” array, it has been reduce from an 8x8 array of constants to an 8x4 array of constants.
- Look at the PRE_ADD and MAC loops. These implement the folded filter that is equivalent to the original design. Note that the MAC loop has only 4 iterations, hence only 4 multiplies. The PRE_ADD loop adds the array data and uses the “?” operator to switch the add to a subtract based on the row (index “i”) being processed.

Folded Filter Theory

The original design was performing a series of multiply and adds of the form:

result = x[0]*a + x[1]*b + x[2]*b + x[3]*a

This can be rewritten as:

result = x[0]*a + x[3]*a + x[1]*b + x[2]*b
result = (x[0] + x[3])*a + (x[1] + x[2])*b

- Scroll down in the “dct.cpp” file and look at the processing loops. The “mult_add” function has replaced the “inner1” and “inner2” loops. Note that the data types for each set of loops is passed to the “mult_add” function. The arguments to the “mult_add” function are the current row of data and the loop index.

```
ROW0:for (int i=0; i < 8; ++i ){
    COPY_ROW0:for(int p=0; p<8; p++)//Copy one row of input into local storage
        buf0[p] = input.read();
    COL0:for (int j=0; j < 8; ++j ) {
        acc0 = mult_add<ac_int<8>,ac_int<9>,ac_int<21> >(buf0,j);
        mem[j][i] = acc0 >> 5;
    }
}

COL1:for (int j=0; j < 8; ++j ) {
    COPY_ROW:for(int p=0; p<8; p++)//Copy one row of mem into local storage
        buf1[p] = mem[j][p];
    ROW1:for (int i=0 ; i < 8; ++i ){
        acc1 = mult_add<ac_int<16>,ac_int<17>,ac_int<31> >(buf1,i);
        output[i][j] = acc1 >> 15 ;
    }
}
```

Compare the Area

- Generate RTL
- Look at the Table View. The area has been reduce by almost 50%.
- Go to the next Toolkit