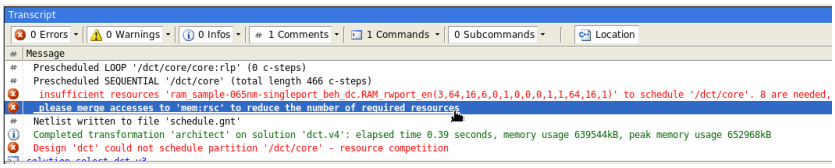
Mentor®
A Siemens Business

Designing a JPEG DCT in C/C++
using Catapult Synthesis

June 2017

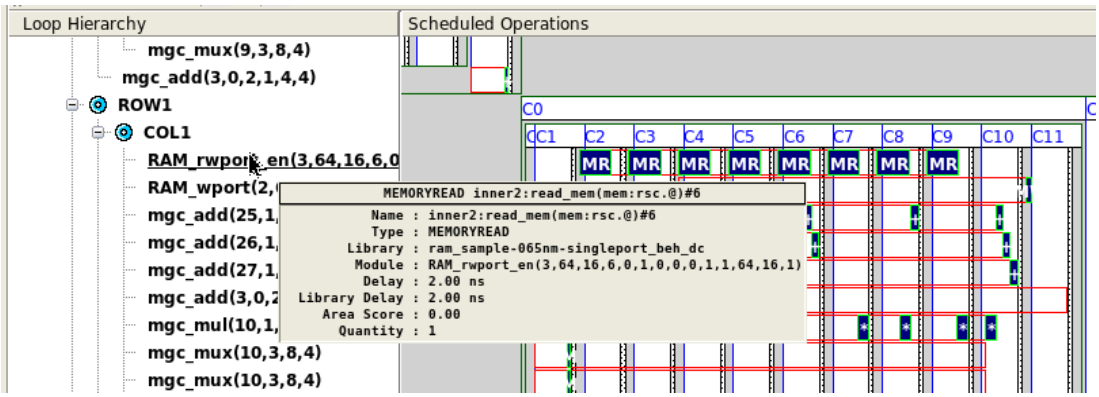
Mentor Graphics Corporation
8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777.
Telephone: 503.685.7000
Toll-Free Telephone: 800.592.2210
Website: www.mentor.com
SupportNet: supportnet.mentor.com/



- Double-click on the second line of the error message that "mem:rsc" accesses must be merged. This will cross-probe to the C++ source and show you that the mem array, which is mapped to a single-port RAM, is the source of the problem. The Gantt chart can provide more information on why "mem" is preventing scheduling.

```
ac_int<16> mem[8][8];
ac_int<8> buf[8];//Local storage for one row of input
ac_int<21> acc0;
ac_int<31> acc1;
```

- Go to Architectural Constraints and turn off pipelining for the entire design.
- Schedule the design and view the component utilization view of the Gantt chart. There are 8 memory reads of "mem:rsc" in the COL1 loop because the "inner2" loops has been unrolled 8 times. The "inner2" loop is reading "mem[j][k]". Thus it is not possible to pipeline the COL1 loop with II=1 with the current C++ architecture. The C++ needs to be re-architected to achieve the performance goal.



- Go to the next Toolkit.

Reducing Memory Accesses to Improve Performance

The previous Toolkit showed that the reading of the internal array "mem" was preventing Loop Pipelining because "mem" was mapped to a memory inside of an unrolled loop. This performance bottleneck can be removed by re-architecting the C++ to remove the memory access from the "inner2" loop. This kind of architectural change is something that we do all of the time when "massaging" C++ code that came from an algorithm or systems engineer.

Schedule the Design

- CD into the DCT_Performance 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 "Unroll inner1 and inner2" script and click on "Launch Project" in the toolkit window (These are the constraints that cause the scheduling failure in the previous toolkit). Note that the design passes scheduling.

Code Modifications to Reduce Memory Access

- Open the dct.cpp file and see how an internal array has been added to copy the rows from "mem". In order to reduce the copying of data from mem, the ROW1 and COL1 loops have been swapped and a COPY_ROW1 loop has been added to copy the row data from "mem".

```
ROW1:for (int i=0 ; i < 8; ++i )
    COL1:for (int j=0; j < 8; ++j ) {
        acc1 = 0;
        inner2:for (int k=0 ; k < 8 ; ++k )
            acc1 += coeff[i][k] * mem[j][k];
        output[i][j] = acc1 >> 15 ;
    }
```

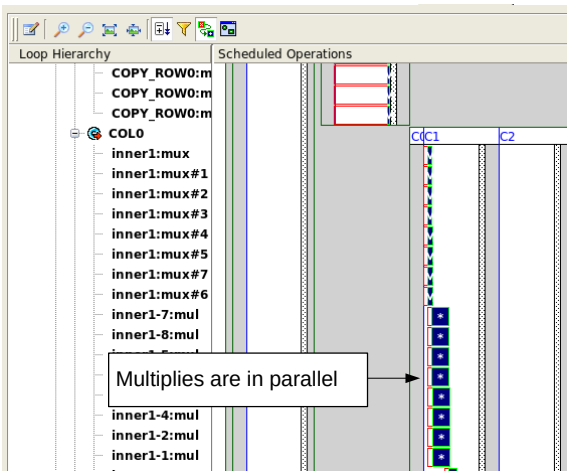
Figure 3 Original DCT Memory Accesses

```
COL1:for (int j=0; j < 8; ++j ) {
    COPY_ROW1: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 = 0;
        inner2:for (int k=0 ; k < 8 ; ++k )
            acc1 += coeff[i][k] * buf1[k];
        output[i][j] = acc1 >> 15 ;
    }
}
```

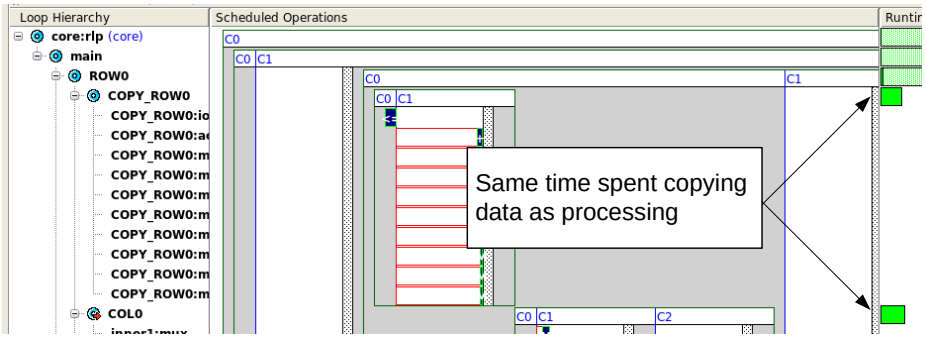
Figure 4 Re-architected Memory Accesses

- Look at the Table View. The throughput is now 290 cycles.
- Open the Gantt chart.

- The reason for the big increase in performance is due to the parallelism introduced by loop unrolling "inner1" and "inner2". You can see that all the multiplies and adds are now scheduled in parallel.

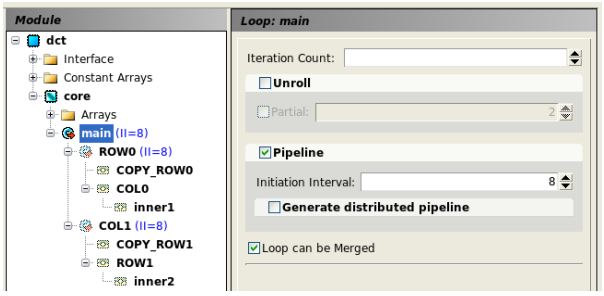


- The loop execution runtime is now split evenly between the COPY_ROW0, COL0, COL1, and COPY_ROW1 loops. In other words the entire design is idle while the COPY_ROW0 and COPY_ROW1 loops are running. We need a way to overlap the copying of the data with the processing. This can easily be achieved with a little more loop unrolling and pipelining at the top level.



Using Loop Unrolling and Pipelining Together

- Go to architectural constraints and fully unroll the COPY_ROW0, COL0, COPY_ROW1, and ROW1 loops. Then pipeline the "main" loop with II=8.



Unrolling the COPY_ROW0 and COPY_ROW1 loops will create 8 separate channel reads and 8 separate memory reads. However since the main loop is pipelined with II=8 there will be no overlap of the reads and the design will schedule. Furthermore by unrolling the COL0 and ROW1 loops we are making 8 copies of "inner1" and "inner2" which are flattened into ROW0 and COL1 respectively along with the row copy loops. Because "inner1" and "inner2" are both fully unrolled this creates 8 X 8 = 64 multipliers for both ROW0 and COL1. By pipelining the "main" loop with II=8 this allows the 64 multiplies to be performed of 8 clock cycles, which re-shares them down to 8 multipliers for each ROW0 and COL1 loop. The COPY_ROW0 and COPY_ROW1 read operations are overlapped with the multiplications in ROW0 and COL1 respectively.

- Schedule the design and look at the throughput in the Table View. It is now 128 cycles.
- Generate RTL.
- Write down the design area. _____

Verifying Throughput Using Automated Verification

- Run SCVerify and verify that the design throughput equals 128.
- Looking at the simulation waveforms you can see that the input reads (input_rsc_lz high) and output writes (output_rsc_we low) don't happen at the same time, except for a slight overlap due to pipelining. What this means is that the ROW0 set of loops is idle when COL1 set of loops is running and vice versa. This makes sense if you consider that each of these loops are accessing a single-port memory and the memory can only be read or written at any given time. A different type of architecture will be needed to achieve the next throughput requirement of 64 cycles. However before moving on to Catapult Hierarchy we should see if the current design can be optimized for area.