Fast Turnaround of a Structured Custom IC Design Using Advanced Design Tools and Methodology

Through the use of several new tools and methodologies, a small team of engineers was able to design and verify a 1.7-million-FET chip in eight months. The tools and methodologies used included a set of guidelines and timing constraints that were met by the customer, a data path compiler, a highly tuned custom multiplier cell that was used in 87 locations, and an automated top-level power connection scheme.

by Rory L. Fisher, Stephen R. Herbener, John R. Morgan, and John R. Pessetto

The HP IMACC chip was developed to provide image processing capabilities. The initial target application is medical imaging with geological applications as a potential area of expansion. The graphical capabilities of IMACC include spatial filtering, edge detection and enhancement, image pan and zoom, image rotation, and window and level control. IMACC consists of three major components:

- The convolver circuit has a 3×3 programmable kernel* and can perform low-pass or high-pass spatial filtering, edge enhancement, and other functions.
- The interpolator is an implementation of a 4×4 bicubic convolution kernel.* The interpolator can be configured to perform pan, zoom, and rotation.
- A RAM-based lookup table is used for windowing and leveling of image pixel intensities.

In support of the various user-selectable operating modes, any or all three of the functional blocks may be active at a time. The order of operations can be changed as desired, with the single limitation that the convolver must precede the interpolator if both modules are in the chain. When the image visualization accelerator board (IMACC is the heart of this board) is attached to the HP HCRX graphics subsystem, simultaneous convolution, zoom, rotation, and window and level control of 1024-by-1024 pixel, 16-bit medical images at 40 frames per second are possible. The accelerator can process more than 40 million pixels per second independent of the number or order of internal operations.

Customer Interaction

As a result of our experience in designing numerous ICs for various customers, our laboratory has developed some practical, informal guidelines for designing ICs. At the beginning of the IMACC project, we met with the customer (another HP laboratory) and discussed these guidelines along with project goals. The guidelines we provided to our customer are as follows:

- The prime directive: Signal groups such as multistate drivers on a single bus, multiple set signals into a flip-flop, or multiple set signals into a multiplexer, may cause drive fights and therefore need to be completely decoded from the current state of the control machine so that one and only one will fire. This requirement must hold even if the chip comes up in a random state. Exceptions to this have caused significant delays in schedule right before tape release.
- Signals require a consistent naming convention.
- Update flip-flops on the falling edge of the clock (single-edge timing).
- When glitchless values are required (Gray code counters, etc.), they must come directly out of flip-flops.
- Resets are typically heavily loaded and will probably cause timing problems. Have each control block latch its own version of the chip reset, then generate its own local reset. This helps timing at the expense of latency.
- Don't design multistate paths. Complete timing analysis of such a path is not possible in any design tool.
- Don't set and dump the same FIFO or RAM location at the same time.
- Keep Synopsys blocks small.
- Keep large register files in the data path.

* A kernel is a functional unit that can be repeated as needed. A 3×3 programmable kernel performs a programmable function on a 3×3 array of pixels. A 4×4 bicubic convolution kernel performs a bicubic convolution on a 4×4 array of pixels.

- Use no clock uncertainty (skew) in Synopsys. It will be there, but is better allowed for by reducing the period.
- Don't allow Synopsys to try to fix hold problems. There should be none by design.
- When setting your timing constraints, allow some slack for RC delays, the local clock generator, and incremental delays that will be introduced when actual routing capacitances are substituted into the timing model. For example, 15 ns might be a good period constraint at 60 MHz.
- As constraints (timing and loading) become more accurate, make sure to continue to update them in your design. Accurate is better than conservative in Synopsys.
- Simulate at the board level as soon as possible.
- Simulate timing between blocks as soon as possible (schematic simulations are fairly accurate).
- Simulate the chip coming up in random states as soon as possible. A proven way to do this is to make sure the chip can come up with unknown values in all memory elements, including flip-flops and registers.

The highest-priority design goal was to have a working IMACC system to demonstrate at an upcoming conference. We created a schedule consistent with this goal incorporating the necessary checkpoints. Two of the most important checkpoints:

- We were to deliver a top-level, schematic-based Verilog gate model to the customer so they could begin regression tests at the system level. This allowed them to identify design problems early.
- The customer was to freeze the function of major data path blocks by a scheduled date. This allowed us to construct the artwork in a single pass.

Setting a rigorous schedule as the first priority forced a streamlined design. Using a single clock domain made the design less complicated. A large data path block with noncritical functionality was eliminated because it would require too much design time. The die size was determined early and a previously characterized package was used. Timing budgets were kept conservative to ensure that IMACC would run at 45 MHz after parasitic loading was added.

Since all decisions were based on meeting our primary objective, "creeping featurism" was eliminated. The customer was informed of the schedule impact that the addition of a new function implied. In most cases they were not willing to suffer a postponement of the chip release date.

We also shortened the design time by making sure that when we sent them a new gate model of the chip, we had all the latest changes and we had no problems with the syntax of the model. To do this we used several tools (awsim, eval, etc.) to check for connectivity problems. We used an in-house history management system to maintain revision control of these gate models.

Custom Multiplier

One of the key pieces of circuitry on IMACC in terms of design leverage was an integer multiplier. The imaging algorithms that we implemented typically executed a large sum of product terms. As an example, the convolver block in IMACC sweeps a 3×3 matrix across the source image (a 2D array of pixels), multiplies the nine coefficients in the matrix by the corresponding pixels, and adds these nine products to compute a single new pixel value for the target image. Therefore, this block alone required nine multipliers and eight adders. With the customer's help we found that we could consolidate most of the multipliers into one design. The result is that in the IMACC design, a single 18×18 integer multiplier circuit is used 87 times.

It then became very important to make this multiplier as dense as possible. This problem was attacked on two levels. First, an area-efficient architecture was chosen, and secondly, the key multiplier cell was painstakingly designed and laid out to reduce its area as much as possible.

The architecture we chose was a radix-4 (two bits at a time) Booth-encoded array.[1] A standard multiplier array in our case consisted of 18 rows, each row representing the result of one of the multiplier bits times the whole multiplicand value. Since the Booth-encoded array looks at two bits of the multiplier times the whole multiplicand in each step, this cut the number of rows in half to nine. It turned out that the increase in the row height resulting from a more complex unit cell was well below the height of two of the standard rows. (However, there is additional circuitry that needs to be added to perform the Booth encoding of all of the sets of two multiplier bits.) As a side benefit, cutting the multiplier rows in half also increased the circuit's speed.

This speed-up benefit helped with our second task of reducing the area of the critical multiplier cell. Our design requirement was to execute the 18×18 multiply in a single clock state (22 ns). When the circuit was initially built, it executed the multiply in about two thirds of the required time (about 14 ns). Therefore, it was possible to shrink the critical cell by reducing the FET sizes until the cell delay increased the multiply time to the 22-ns limit. Along with some diligent artwork layout, this produced an extremely dense cell.

Our final 18×18 Booth-encoded multiplier is 864.0 μm wide by 307.8 μm high for a total area of 0.266 mm^2. Using the standard multiplier array, this multiplier would have been 648.0 μm wide by 446.7 μm high for a total area of 0.289 mm^2. Considering just the areas of the multipliers, the savings per multiplier is 0.023 mm^2 and the total savings for IMACC is 2.0 mm^2. However, the aspect ratio of the standard multiplier did not fit well with the IMACC data path blocks (18 bits wide versus 24 bits wide). Taking into consideration the empty spaces that the standard multiplier would have produced, the savings per multiplier increases to 0.123 mm^2 and the total savings for IMACC increases to 10.7 mm^2.

The custom multiplier design cut schedule time for two reasons. Using it 87 times in numerous blocks across the chip saved us the time we would have spent assembling (and possibly designing) multipliers for each specific need in the different blocks. Secondly, the significant area savings we realized made the job of top-level chip routing much easier and consequently faster, since the top-level blocks were smaller.

Data Path Compiler

We were able to save many hours of artwork layout time through the use of Dpc14,[2] a data path artwork generation tool that places and routes data path blocks. A data path is made up of hierarchical horizontal slices, often called macro cells, that are usually, but not always, bit-wise logic repeated as many times as needed. Examples of macro cells are a two-input, 32-bit register, an 18-bit ripple carry adder, another data path block, or a custom data path block. The program dpc_tiler was used to build many of these macro cells. Macro cells can be placed vertically or horizontally with respect to one another. Routing is done over and between macro cells to connect signals within the data path. The user has the ability to control placement and how signals are routed. The user can assign signals to specific data path tracks or defer routing of signals such that the *allow* layers for the signals (layers that can be used later in the design to route signals) are placed on unused tracks.

To use Dpc14, we first generated a Dpc14 input file, which was created from the schematic Block Description Language (BDL) file. The Dpc14 input file could also be generated from a Verilog netlist file. The tool bdl2dp was used to generate the input file from the schematic BDL for most of the IMACC data path blocks that used the schematic BDL. A simple input file is shown in Fig. 1.

```
-block dpc1
-trackassign a 5
-dpwidth 24
-newrow {
  -inst HOLE TOP 24 0 {
    -dpsig A[23:0] a[23:0] 0
    -dpsig B[23:0] b[23:0] 0
  }
}

-newrow {
  -inst REG224 rega 24 0 {
    -dpsig INA[23:0] a[23:0] 0
    -dpsig INB[23:0] b[11:0] 0
    -dpsig OUT[23:0] out[23:0] 0 -metal1
  }
}

-newrow {
  -chanroute { -signal cinput[23:0]
-right }
  -inst MUX224 muxa 24 0 {
    -dpsig INA[23:0] a[23:0] 0
    -dpsig INB[23:0] out[23:0] 0 -metal1
    -dpsig INC[23:0] cinput[23:0] 0
    -dpsig OUT[23:0] muxa[23:0] 0
    -cntlsig SELA sela 0
    -cntlsig SELB selb 0
    -cntlsig SELC selc 0
  }
}

-newrow {
  -inst HOLE BOTTOM 24 0 {
    -dpsig muxa[23:0] muxa[23:0] 0
  }
}
```

Fig. 1. *A simple input file for the Dpc14 data path artwork generation tool.*

Artwork encapsulation information needed to be extracted for each macro cell used in the data path. The Perl script dpEncapInfo was used to extract information about how to connect to ports within the macro cell. It was necessary to specify to dpEncapInfo any wart cells on the left or right side of the macro cell. For instance, a typical DPLIB14 register has two wart cells on the right side of the macro cell. In this case we used dpEncapInfo –r 2 to build the encap_info file correctly. This specified that there were two wart cells on the right side of the register macro cell.

After the Dpc14 input file was created and encapsulation information had been extracted, the Dpc14 program was used to generate an artwork archive that could be read into our artwork editor. The Dpc14 file could then be edited if the artwork needed to be modified, allowing us to make as many iterations as needed to produce the desired result.

Local Toolbox

A number of relatively simple scripts and programs of our own devising were combined into a local toolbox for the project. The more significant of these are described here.

The mkcntl script uses Synopsys, block place-and-route, and other tools to go from a Synopsys netlist through schematic to artwork with parasitics in one command. Of course, one must iterate on the place-and-route portion to ensure workable size, form factor, and port locations. An iteration can be accomplished with mkcntl –b.

We used a connectivity tracer that reads schematic (scip) BDL and reports the connectivity of the specified instance or net. The trace is limited to one level of hierarchy. For net names, regular expression pattern matching is available.

We automated a lot of the top-level power connection of IMACC using two scripts. Several steps were involved in this methodology. First, two symbolic layers were used, one for VDD and one for GND, to define where the metal-4 power buses would go. Next, the getpwrconn script used trantor to find the intersection of the metal-3 power buses with the symbolic layers and dumped them into an artwork editor archive file. Last, the gen_pg_conn script placed contact-4 contacts in the intersection areas and filled the symbolic layers with metal 4.

Our addallow script selects VDD, GND, and CLK metal-3 shapes by size and copies them to metal3.allow and contact4.allow layers, permitting connection to over-the-cell metal 4.

There are two shieldmaker scripts in our toolbox: addshield2 and addshield3. These scripts fill the unused areas of the intermediate (cross-channel) metal layer in routing channels. These areas are then tied to GND or VDD to provide crosstalk shielding for signals running the length of the channel.

Diode placing software was used to eliminate large numbers of charge collectors. This software finds traces longer than 1000 μm in a routing channel and locates areas where diodes can be added without introducing design rule check errors. This technique worked well for our project, which had a tight schedule, lots of charge collectors, and minimal timing problems.

All of the scripts that add shapes to a source block do so through intermediary block pieces to ease modification or rebuilding of the added function.

Results

The IMACC chip was demonstrated in systems at the Radiological Society of North America conference in Chicago on November 27, 1995.

The IMACC chip contains 1.7 million FETs in the HP BiCMOS14QC process operating at a 45-MHz clock frequency. It is predominantly a data path design with 98 integer multipliers performing 4 billion integer multiplies per second on 18-bit or larger operands. A breakdown by design style is as follows:

Style	Percent of Total FET Count	Number of Standard Cell Gate Equivalents	Thousands of FETs per mm^2
Data Path	60%	497,000	22.0
Standard Cell	10%	53,000	10.8
RAM, FIFO	21%	(>48K bytes)	42.7
Pads, Clock, etc.	9%		

Some additional statistics for the IMACC project include: 2342 FETs/day, 8673 FETs/mm^2, and more than 550,000 standard cell equivalent gates.

Acknowledgments

We would like to thank Rich Nash for his time spent developing Dpc14 and for his timely responses to our suggestions.

Reference

1. S. Waser and M.J. Flynn, *Introduction to Arithmetic for Digital Systems Designers*, Holt, Rinehart, and Winston, 1982, pp. 132-137.
2. R. Nash and R. Martin, “Datapath Requirements for Structured Custom Design,” *Proceedings of the 1995 HP Design Technology Conference*, pp. 411-418.

▶ Go to Journal Home Page