2

CONF-900146--1

SAND--89-2712

DE90 002902

CAE Tools' Limitations
Case Study: An SSI Design To An ASIC

Jeff Everts
Sandia National Laboratories
P.O.B. 5800, MS. 2312
Albuquerque, NM 87185
(505) 845-8300, 844-4024

Abstract

Every computer-aided engineering (CAE) tool has its limitations and shortcomings. Knowing where the pitfalls lie and how to get around them is extremely valuable. This paper takes a look at the problems and limitations encountered using the Daisy Systems suite of digital design tools (on a Logician 386 and a PKK386 MegaLogician, running DNIX 5.02A) to redesign a 169 small scale integrated (SSI) component design into an application specific integrated circuit (ASIC) gate array. Deficiencies were found in the libraries, ACE, MDLS, DTA, MCFS, and the Hotline support. Some solutions and workarounds to these deficiencies are presented.

Background

What hoops have to be hopped in order to design with ASICs? What pitfalls lie in the ASIC design/development cycle? Knowing what and where the pitfalls are and how to avoid them can save millions of dollars and considerable man-hours. Taking a design through the ASIC design/development cycle not only answers these questions but also provides the solutions and workarounds to the problems and limitations encountered. This lubricates some of the hoops and fills some of the pitfalls in the development cycle and significantly lowers the cost of future projects. Our approach to smoothing out the development cycle, for the purpose of this paper, focuses on identifying and overcoming our CAE tools' limitations.

Introduction

CAE tools have a significant impact on the design/development cycle. Therefore, any deficiencies or problems in the use of these tools need to resolved. A project was needed to help identify and solve the limitations and problems associated with our Daisy digital design tools. The project chosen was the redesign of an aging subsystem, containing 169, 5400-TTL components, into an ASIC gate array. The redesign involved the following steps: entering the production drawings (composed of 5400-TTL components) into the Daisy schematic capture package; simulating for functionality; creating macro-cells from the gate array vendor's primitive cells; converting the schematic from 5400-TTL components to the gate array primitive and macro- cells; repeating the functional simulations; verifying the timing; inserting a test methodology; repeating the functional and timing simulations; performing a fault analysis; generating test vectors; and delivering the design for implementation into a gate array. These steps would exercise Daisy's digital graphics and simulation libraries, ACE, CED, MDLS, DTA, MCFS.

Library Models

The first pitfall encountered was the incomplete 5400-TTL-components libraries. Many 5400-TTL components that we needed were not in the graphics or simulation libraries. These components had to created. This was done by copying a similar device into CED, changing its PART_NM, and saving it as a primitive. The SPARC source code libraries were modified to include these new components and their Texas Instruments' timing specifications and then recompiled. Another library deficiency encountered was the graphics libraries' lack of DeMorgan-equivalent symbols. Considerable time can be saved during the schematic capture and debug stages of a design if DeMorgan-equivalent gates are used in the schematic. The equivalents make following and verifying a signal easier, less time consuming, and therefore less likely to be the source of a logic or drawing mistake. We worked through this deficiency by building our own DeMorgan-equivalent symbols in CED.

Having hopped the library hoops, we were able to simulate a number of the blocks in the 5400-TTL version of the design.

ACE

The next two steps involved using ACE and CED to create TTL-equivalent macro-cells (macro-G_cells) from the gate array vendor's primitive cells (G_cells) and converting the 5400-TTL-based design to a G_cell-based design.

Occasionally, wires could not be drawn to component pins because the pins did not line up on a grid point. We drew the page one day and the next day, in the course of modifying it, deleted signals could not be redrawn. Solution? Turn snap off, start from a component pin, shoot from the hip, and hope you connect. Many times this approach did not work, so we ended up redrawing the affected portions of the schematic.

In CED, we created our own graphic models to represent the nested, functional G_cell subcircuits (e.g. 30-bit shifter, test block, etc.). They were saved as logical components in a common graphical library. When we tried wiring up these CED-created components in ACE, the wires would not end on the pins. The pins were about a quarter of a pixel off in both coordinate directions. So we redrew the components, being very conscious of the spacing and alignment of the pins, thinking we learned from our previous ordeal. This too was unfruitful. We overcame this problem by bringing a Daisy graphics component into CED and deleting everything except the pins. Then the component was created using those pins as a guide for the pin placement and spacing. Having hurdled pitfalls, we completed the schematic conversion from 5400-TTL components to our ASIC vendor's gate array G_cells without further problems.

MDLS

At this point we had two design versions, the 5400-TTL version and the G_cell version. We started simulating these in parallel to verify that the conversion was done correctly. The G_cell version was assigned to a local MegaLogician and the 5400-TTL version to a networked MegaLogician. Which workstation configures the MDLS environment? Which workstation administers the microcode? Finding the right microcode version for the 1-2-1 Mega configuration and establishing the linking pointers were accomplished by trial and error. The networked MegaLogician, once up and running, simulated extremely slowly over the network. The speed could not be increased by copying the design to the networked MegaLogician because of limited disk space and other usage requirements. The PKK386 MegaLogician was not bogged down by network communications so it served as our primary simulator.

The next problem encountered was SOM's inability to handle large numbers. Our functional vectors had lengths of 2,700,000,000 time units (and this was after we scaled for a X 10 speed improvement). MDLS did not allow continuing a simulation with different SOM files. This would have allowed us to break up the simulation into sequences. We maneuvered around these obstacles by grouping times into units that could be repeated. An example is (10000:F0,10000:F0)**100.

Using a preliminary simulation, we estimated that it would take 40,000 hours (4.5 years!) for the PKK386 MegaLogician to simulate one complete functional vector. There were no options in MDLS that would help decrease the time-to-simulate, like specifying a sampling interval or a data filter. After considering a number of workarounds, options, we modified the design (for simulation purposes only) to eliminate idle time and allow the relatively slow, internal clocks to run 10,000 times faster during most of the simulation. These modifications reduced the time-to-simulate to 4.5 hours. But problems still remained. The data being displayed for these long simulations were inconsistent. A wave window would show a 1-to-0 transition, then going to a list display there would be no transition, and upon returning to a wave display the signal would be a straight line. I called the Daisy Hotline for assistance. They told me that another company had run into a similar problem and that they would send me a modified microcode version that might solve the problem. The modified code corrected the display and data integrity problem and restored our confidence in the simulation results. Another thing we went to was shorter simulation vectors. (If a long simulation run aborted prematurely all the results would be lost.) This minimized the probability of having to rerun a particular simulation. The limited data that could be stored for any one save operation was another limitation that prevented us from a quickly reaching our functional and timing verification milestones.

DTA

The next step in the ASIC gate array design process was to implement a test methodology. Squeezing 169, 5400-TTL components into one integrated circuit (IC) and decreasing the number of external signals from 88 to 11 made observing and controlling internal nodes very difficult. Before we considered any test methodology we attempted to use DTA to see how bad off we were.

DTA ran extremely slowly and stopped prematurely. Giving it the ability to add virtually an unlimited number of input, output, and bi-directional pins to improve the testability, it added only two output pins and one input pin. This improved the testability figure for the design from 9% to 17%. With this kind of performance, DTA is not useful in the ASIC development cycle. So the DTA hoop was eliminated from our ASIC development cycle.

MCFS

The low testability figure indicated the need for a unique test methodology. The test methodology we developed combined level-sensitive scan design (LSSD), partial scan, pseudo-random number generation, and signature analysis into a form capable of interrogating highly sequential designs.[1] The implementation of the test methodology involved ripping up the schematic. With an initial, roomy layout the

---

1. J. Everts, D. Gelet, D. Deatherage, and M. Contreras, "ASIC Replacement For An SSI Component Design, A Case Study," Proceedings of The Second Annual IEEE ASIC Seminar and Exhibit, Sept. 25-28, 1989, Rochester, NY, IEEE Catalog # 89TH0280-8.

changes did not require redrawing the schematics. Signals were broken and intrapage (inter- or hierarchical as required) connectors were added. Test blocks were added along the margins of the page and connected up via the intrapage connectors. This went relatively smoothly. The functional and timing verifications were repeated before the various test modes were analyzed. To analyze the effectiveness of our test methodology on the fault coverage of our chip we required a fault analysis software package. The local Daisy office arranged for us to use our ASIC gate array design to evaluate MCFS. (I have a very high regard for the local office and its staff, especially T. Burrows, and J. Mervini who have often spent late hours bending over backwards to help me.) The two road blocks in getting MCFS up and running were getting a working memory board for the Mega accelerator (The first board had a hardware problem.) and scrounging up enough disk space on our small 65 megabyte disk to install and run the software. Aside from the small disk, speed was also another handicap. Our Sun386i/250s would have given us a considerable performance improvement over our PKK386 but Daisy does not have a version of CFS running on the Sun386i platform. Well, slow and steady wins the race, right?

MCFS has both a statistical and an exhaustive fault analysis mode. We focused solely on the exhaustive mode because we needed an accurate evaluation of our test methodology's effectiveness to detect faults and a precise stuck-at fault coverage number to include in the project's reliability report. Both of these reasons, to some degree, depend on the test vector sets. Daisy does not have an automatic test vector generation tool in their suite of digital design tools (that could take advantage of the test features we implemented). This meant that our evaluation was based on our manually-created test vectors and our ability to create an effective test vector set. The manual generation of test vectors is a sore spot in the ASIC development cycle.

MCFS's problems began almost immediately. While running in exhaustive mode, the system would consistently lockup and require rebooting once it reached 5000ns. After trying to solve the problem locally, we called the Daisy Hotline. Considerable time was spent trying to solve this problem. In the end they said, "Send us your database." To do this takes considerable time and stops the ASIC design cycle. Does Daisy not know that? (They have to come up with a better way to support their customers that run into serious problems such as this MCFS situation.) Getting out of MCFS's quicksand was very laborious. First we tried dividing the faults up by class: input faults and output faults. MCFS handled the output faults without a glitch, but it killed the system while working on the input fault list. Next we sorted the input faults between stuck-at-1 faults and stuck-at-0 faults and ran them separately. After blood, sweat, and tears, we finally had some fault analysis figures. The workable scenario, then, required three runs per vector set: 1) output faults case, 2) input faults stuck-at-1 case, and 3) input faults stuck-at-0 case.

One of the concluding steps along the ASIC development cycle is converting the simulation vectors and the test vectors into a form that can be used by ASIC testers to actually test and characterize the fabricated parts. The problem we ran into is how does one get 350 megabytes of ASCII 1s and 0s into an ASIX Systems (Irvine, CA) tester? ASIX claims to be able to translate Daisy simulation vectors into ASIX compatible test vectors. An internal test organization tried to do this on a much simpler circuit and were unsuccessful, so we were not going to tango with certain failure. The outside fabrication company had the same problem with our functional vectors. What we ended up giving the foundry was our set of short, fault analysis vectors for use in the chip's test mode. The only functional testing that could be done was in real time, in a real system, using a real part. That is not the way ASIC design/development should go.

Summary

The ASIC design/development cycle is littered with pitfalls. A subsystem redesigned into an ASIC gate array took us through the ASIC design/development cycle. This project helped us focus on the limitations and problems associated with our Daisy digital design tools. Deficiencies were found in the libraries, ACE, MDLS, DTA, MCFS, and the Hotline. The solutions and workarounds that we developed will be of considerable value to future ASIC gate array development work. Though Daisy's CAE tools have limitations, it was the tools' capabilities that enabled us to go from schematic capture to IC fabrication successfully.