



How to Make An ASIC Prototype

by Lars-Eric Lundgren, HARDI Electronics AB

At HARDI Electronics, we started working seriously with ASIC prototyping in FPGAs about five years ago, and we quickly realized what the challenges were. First, it was obvious that the prototyping system needed to have the required capacity corresponding to the gate size of the ASIC. In year 2000, that was quite a problem since maximum FPGA capacity was a couple of hundred thousand ASIC gates and the ASICs were 10-100 times larger (1-10 million gates). Since then, the FPGA gate capacity has grown significantly. Today the largest available FPGA, the Xilinx Virtex-4 LX200, has an equivalent ASIC gate capacity at about 1.5 million gates. By putting a few FPGAs together, we can build a prototype for quite a large ASIC. On the other hand, ASICs haven't stopped growing. A good rule is to say that FPGAs will continue to be about 1/10 the size of the average ASIC. Therefore, some designs will still need more than 10 FPGAs to make a good prototype.

However, the most critical parameter of an ASIC prototype in FPGAs is, in fact, not the gate capacity, but the number of interconnects between the different FPGAs. It doesn't matter how many wires there are to connect the FPGAs together, it will still be too few. The reason for this is that inside the ASIC you normally have many wide busses going from one block to another. You'll always need to use the biggest FPGA devices in the biggest packages. It doesn't matter if the price is higher, always use the biggest package.

Then you have the next problem with FPGA interconnect. How do you design the buses so they fit the buses in the ASIC? This is easy enough if you know how the ASIC looks when you start designing the FPGA prototype. You make the same buses on the FPGA board that you have inside the ASIC. At HARDI Electronics, making generic ASIC prototyping boards that are supposed to fit virtually any ASIC, this is a challenging issue. We had to make it possible to adapt to different kinds of ASIC designs, so we needed a configuration that

could change the interconnect in minutes. Designers of dedicated FPGA boards have also grappled with this problem. Even if they knew exactly how the ASIC looked, the board sometimes didn't fit the design when it came back from manufacturing. The reason is, of course, that the ASIC often changes. The ASIC designers put in more or wider buses, extra or bigger blocks, etc. This happens all the time during the evolution of an ASIC design. You simply can't decide how the prototype should look if you are too early.

So, we needed a way to make the system easily reconfigurable. This can be done in several ways. One possibility we explored was to connect the FPGAs with matrixes of interconnect devices - small devices that work like a telephone switchboard, but we determined that this plan would not be good enough. It wouldn't be flexible since it would be impossible to route any signal to any device. Additionally, it would be very difficult to connect signals that spanned many FPGAs.

Another possibility was switch matrixes, but several issues arose with this. First, they have to be programmed, and you might not be sure what state they are in if your design stops working. Another problem is that these switches add delay to the wires and influence the impedance matching of the signals. They also limit what signal levels you can use, and might stop you from using differential signaling when it's needed. On top of all this, they are quite expensive. So we decided switching matrixes are totally off for this purpose.

The remaining option, then, is to just go back to the old telephone switch. What if you could use connectors of a very high quality, impedance match all signals very carefully, and use some kind of connector cables and boards to make the connections? This is what we did with the HARDI ASIC Prototyping System (HAPS).

Our approach was this: put several FPGAs on a board but don't connect them tightly together. Instead, put a number of high-pin count, high-quality, connectors on each FPGA. Then work carefully with every trace on the board to achieve the best possible signal integrity. All wires should be impedance matched and length matched. This gives virtually no reflections and enables you to run signals very fast across the buses since the delay is the same for all wires in a connector. To run signals at very high speed, you'll have to deal with crosstalk. If you use high-speed connector boards and cables to connect

the signals from different FPGAs together, you can make virtually any connection you desire.

Interestingly enough, you get another benefit with the connector approach. If you don't want to use all the available FPGA pins for interconnects, it works fine to use them for I/O or attaching memories, etc. That means that, through the connectors, you get a totally open system with the greatest possible flexibility. And it still takes just minutes to build up and configure the whole system. To make sure that all connections work, you need to have a suite of tests that test all pins for opens and shorts. This is a simple design that downloads into the FPGAs. Shorts are very easy to detect, and with a little test board that you put on the connectors, it's also easy to test for opens.

By using the largest FPGAs in the biggest package and connecting them to many connectors we have gone far in making a good ASIC prototype system. But more challenges lie ahead.

Performance, meaning system speed, will always be a challenge in FPGA systems running ASIC designs. ASICs can be much faster than FPGAs, with a comparison of 100-200 MHz in the FPGAs to soon over 1GHz in the fastest ASICs. As a board manufacturer, we can of course not do anything about the speed inside the FPGAs, but we can do a lot for the speed outside of them. We have to make sure that a signal can travel across the board from one FPGA to another with the highest possible speed. To achieve this, you need experts in making high-speed PCB layout, and the best PCB software tools. But you should not only make sure that signals travel fast, they should also travel the same distance. For example, it is very important that the signals in a memory data bus arrive at the end point at the same time. This can even mean that you purposely slow down some signals in a bus, or length match them. You make all the traces from one connector to the FPGA the same length even if some could go a shorter way. This makes it much easier to connect high-speed memories like DDR2, etc. If you're using source synchronized clocking, this can give you very high-speed data rates (600 Mbps) even without using differential signaling (LVDS). When taking in external signals from outside the system, you can use LVDS to achieve data rates at 1Gbps at any FPGA pin.

Yet another issue is to distribute the system clock. Inside the ASIC, there are normally well-known techniques to distribute a fast clock over the entire chip with low skew. Inside the FPGAs, you also have good techniques for fast clocks, such as built-in global clocks. On the board, between the FPGAs, it's up to the board designer to make the clock distribution. You need dedicated high-speed clock drivers, and you arrange the clock lines to ensure that the clocks arrive at all FPGAs at the same time with a minimum skew. This is more challenging if you have so many FPGAs that you need to use several FPGA boards. If that's the case, you need to get the clock of one board and drive it to the next board. There should also be the possibility to have a number of high-speed clocks, since in the ASIC you normally have several clock regions with their own clock.

Since the open system you are building can have I/O devices, communication, and memories anywhere, it will be very important to have the option to run different parts of the system at different I/O voltages. You might have Ethernet or USB PHYs running at one voltage, and memories at another. The problem is, you never know what voltage will be where, so you need to make all connectors switchable to any I/O voltage of 3.3, 2.5, 1.8 and 1.5. Then you are safe, but make sure you have a lot of current to supply to the different regions.

But how do you make a system to handle 10 FPGAs equal to 15 million ASIC gates? It must be flexible and reusable, so you can't put all the FPGAs on one board. A board like this would be a nightmare to manufacture and also very expensive. Well, you can use a backplane and connect different smaller boards together. Hmm... we're talking high speed here. The backplane would be a bottleneck, not only slowing down the system, but also increasing the cost and really not adding any value. Why not just build a fairly small board with four FPGAs, and then put connectors on both sides of the board? Then you can stack the boards and have a fast, distributed bus going straight through all the boards. You get very short signal wires even if you stack several boards on top of each other. We call this bus HapsTrak, and use it in all HAPS generations.



Figure 1. Example of HAPS Motherboards Stacked Together

Now you have a very flexible, reusable, high-performance hardware system for ASIC prototyping. How will you use it? What software should you use for the prototype? What is the design flow going to be? Here you have to ask yourself a very important question: do you want to run on a compact, low-cost system, at very high speed? Or maybe you don't care about speed and cost as long as there is minimal design effort for you. If you want to run fast, you still have some work to do. You need to understand FPGA prototyping since this is not an emulator (emulators have other problems, but let's leave that for a future discussion). You need to have a good, well-documented, HDL design, but that's true for every ASIC. But you shouldn't worry about writing FPGA-specific HDL code. That is not needed these days. There are tools translating gated clocks to clock enables in the FPGA. You can also get translation of IP blocks into FPGAs automatically and so on.

Synplicity's Certify is a full-blown tool helping you with many tasks for ASIC prototyping. Apart from the obvious in assisting you with partitioning, it can also help you with pin assignment which is a tedious task if done by hand. Certify does quick estimates of area so you can figure out how the different ASIC blocks can go into the different FPGAs. In short, you can play around very freely and find a good partitioning quickly. If you know very well from the beginning how to partition your design, and if you're proficient at writing top-

level HDL, it might be easy just to use the FPGA vendor's low-cost synthesis. Or maybe Synplicity's new Synplify Premier or Pro, these are for only one FPGA at the time. Or you can use Mentor's Precision or Synopsys DC FPGA if you have dedicated functions for ASIC prototyping. After running the partitioning and synthesis, you always end up running the FPGA vendor's place & route tool. Then you get the optimized bit file for downloading to the prototype system.

Working this way will give you total control over the design, and you will also have the option to run at a very high system speed (150-200Mhz). If you then need debugging at real speed, you can choose one of the debuggers on the market, like Identify from Synplicity, DiaLite from Temento, or perhaps use the built in logic analyzer that you find in the tools from your FPGA vendor.

Following this design methods, you will hopefully find many bugs in the ASIC and the managers and all involved will be your best friends. It will take some work, but you'll feel good when you're done.

Lars-Eric Lundgren, HARDI Electronics AB

October 18, 2005