



Intelligent Integration

Considering the Costs of Convergence

We live in an age of integration and convergence. My typewriter, TV, telephone, record player, slide projector, file cabinet, and storage closet were long ago converged and integrated into my computer. Next, my computer, monitor, speakers, keyboard, mouse, and modem melded into my laptop. Now, my laptop, cell phone, MP3 player, digital camera, and pager are almost all rolled together into my smart phone. Moore's Law and modern life have compressed, converged, combined, and cost-reduced our components into integrated, versatile, do-it-all devices. There is no question that the integration trend has bought us continuously improved capabilities and economies. However, while each of these fusions is designed to improve communication, convenience, cost, and portability, not all integrations are successful, or even a good idea in the first place.

As system designers, we encounter the integration question at a much lower level. Processors, peripherals, memory, interconnect, and storage are all basic components of almost every modern electronic system design. A successful system designer has to balance the forces of form factor, power, price, performance, reliability, security, scalability, and product evolution in order to make the crucial decision of what and how to integrate, and what to leave discrete. While there are myriad options available, there is no reliable formula or roadmap to guide us through this complex engineering tradeoff.

As system designers, the first thing we should always identify is our secret sauce. Every product that isn't headed directly to commodity pricing has one. Secret sauce is the thing that differentiates our product from all others – making our version special and giving us our advantage in the market. The secret sauce could be buried in software, hardware, physical design, or some combination of those elements. Once we've identified it explicitly, we're much better equipped to make our integration decisions intelligently.

With the secret sauce issue in hand, we need to take a look at everything else in our system – the parts that aren't secret sauce. These are usually the bricks and mortar of our system design. We won't gain anything on our competitors by building them better, but they're absolutely required for our product to do its job. It pays to work these components into our system with the least possible allotment of engineering time, risk, cost, power, and/or space, depending on our overall design goals. In the make-versus-buy trade space, these items will tend toward the "buy" end of the spectrum.

Next, we should look at the integration options available to us. The range of viable choices will depend largely on our intended production volume and our development budget. At the zero end, available to everyone, are pre-assembled boards and modules. If our secret sauce is software, there's no quicker way to market than locating an off-the-shelf, pre-designed, tested, and manufactured board or module that will allow us to load our RTOS and software, snap on our physical connectors, and start doing useful work. Even if our embedded systems project will be completed in our garage and has a lifetime production volume of one, we can probably succeed with the pre-made board approach.

If some of our secret sauce is in hardware design, we can open our range of options a bit more. Many pre-fab boards are available with field-programmable gate arrays (FPGAs) that will allow us to implement and integrate a wide range of processors, peripherals, and custom hardware blocks into a single chip, with the remainder of the hardware on the board providing basic services such as memory and physical connections to the outside world. These FPGA boards are incredibly versatile, and a wide range of them is manufactured, aimed at various vertical applications such as industrial control, digital signal processing, and PC-compatible peripherals.

If we're aiming at a bit higher production volume, and we have the money and the moxy to develop our own circuit board, the integration options expand dramatically. We can pick and choose our chipset, optimizing our design for whatever parameters we consider most important. If we're wanting the fastest time-to-market and the least design cost, we should look for application-specific standard parts (ASSPs) to perform some of the heavy lifting on very specific tasks. As a rule of thumb, if an ASSP is available that does exactly the function you need, you should not design your own. You'll only add engineering cost, risk, and extra time to your design process, and ASSPs are almost never an element of secret sauce. It pays for us to leverage the work done by the ASSP vendor and spend our time and engineering energy on the elements that will truly set us apart.

The ASSP question becomes stickier if the ASSP almost meets our needs, but perhaps doesn't interface cleanly to some other part of the design. In these cases, a complex programmable logic device (CPLD) or perhaps a small FPGA are often used to "bridge" between ASSPs that don't know how to talk to each other, but sometimes the challenge of bridging between incompatible standards is more difficult than re-designing the hardware function from scratch.

If hardware is part of our secret sauce, or if we have too many required standard functions to make our design space- or cost-practical, we will end up designing some custom logic. Here we have a range of options, depending on our performance, density, cost, and volume requirements. The most accessible, again, are FPGAs that are relatively easy to design, can integrate a large amount of logic including our custom logic as well as standard functions such as processors, peripherals and memory, and have the added flexibility advantage of in-the-field reprogrammability. If we need to alter our design, even after our product ships, we can deliver a new bitstream to reconfigure our FPGA, giving us the flexibility of software, even on the hardware parts of our design.

One notch up the difficulty curve from FPGAs are structured and platform ASICs. These devices are typically designed in much the same fashion, and using most of the same tools as FPGAs, but they are custom fabricated specifically for our application. The upside of this is higher performance, lower power dissipation, significantly higher density for integrating more and larger functions, and lower unit cost. The downside is longer design and procurement cycles, the addition of a non-recurring engineering (NRE) charge (typically around \$100K per design spin), and the lack of hardware reprogrammability. Typically, it is either performance, density, or volume requirements that push engineering teams across the FPGA-structured ASIC boundary.

The technology of last resort is, of course, the cell-based ASIC. If your volume isn't at least seven figures and your development budget eight, you probably don't need to concern yourself with this option. Cell-based ASIC delivers absolutely the highest performance and density, and the lowest power consumption and unit cost. The price of admission to this club is development cycles typically approaching two years, NRE costs of \$1M per design iteration for the latest 90nm technology, and total development and tool costs running from \$5M up for a typical design.

For any of these custom silicon options, from FPGA through cell-based ASIC, you will ultimately face the decision of which components to integrate and which to leave as discrete components on your board. Priority one, of course, should be the most efficient and effective implementation of your secret sauce. Our custom device-of-choice must have at least enough logic, I/O capability, and performance to support a no-holds-barred, future-proof implementation of our special recipe. Depending on how static that content is, we might want to tend toward a more stable or more volatile technology – you don't want to have to re-spin a several million dollar ASIC every six months if your implementation improves often.

With the hardware component of secret sauce securely on board (or on-chip, actually), there will usually be space available for more stuff. For most peripherals or special functions for which we might consider an ASSP, there is probably an IP core available for our custom device technology that will give us the same function integrated into our custom chip. It's almost always more economical to bring those functions onto the chip than to buy and connect to them separately. If the I/O connecting these functions to our other logic is substantial, we can save a lot in board complexity and power as well.

Processors and MCUs are typically available in a variety of flavors as embedded cores for every custom silicon technology from FPGA through cell-based ASIC. On the FPGA side, the most popular cores are "soft" processors, which are implemented in regular logic fabric, and can be tailored and scaled to individual application needs. Typically, these are not as well known as the industry-standard names in processors, but they offer advanced architectures and a wealth of flexibility, including the ability to instantiate multiple cores on a single device. For structured ASIC and cell-based ASIC technology, there is also the option to use higher-performance "hard" or "firm" cores that are more highly optimized, with slightly less flexibility than FPGA-based soft-core processors. Increasingly, the line between these two approaches is being blurred – with Actel recently announcing their partnership with ARM to make soft-core ARM7 processors available on Actel FPGAs, and with Xilinx offering hard-wired PowerPC processors on their Virtex II-Pro and Virtex-4 devices, the distinction between soft-on-FPGA and hard-on-ASIC is melting away.

The final choice to make on integration is memory. Most custom devices include some amount of RAM on-chip to handle close-coupled needs such as caching, temporary storage, and FIFO implementation, but for larger blocks of memory, it is almost always necessary to go off-chip with a standard RAM architecture. When choosing a custom silicon technology, it is important to be sure that the one you choose supports integration with the class of memory

you need in your system. If you require DDR2 DRAM, for example, you don't want to find out late in the design cycle that there's no proven interface between your custom logic and that flavor of commodity memory.

Besides looking at power consumption, system cost, and form factor, we also need to consider product evolution in our integration choices. If, for example, we expect a new I/O standard to emerge that we want to be compatible with in the near future, we don't necessarily want to integrate the existing standard into our ASIC, requiring a re-spin to accommodate the new standard. We'd be smarter to leave that function external, and just swap out an ASSP, keeping our custom IC untouched when the standard change hits. Also, the cost of verification of a custom device increases as we add more and more cores. The difficulty of debugging an embedded version of virtually any function is much higher than for the corresponding discrete function, and the cost of correcting a mistake, far higher still.

Additionally, integrating some specialized functions such as analog may dramatically alter the requirements for our custom device -- changing our options on implementation technology, or requiring the addition of more power supply voltages. All of these costs and complexities should be taken into account in any integration decision as well. Sometimes a single core can boost the cost or complexity of the rest of our design to an impractical degree. We should always consider the system as a whole when deciding on any individual integration.

While the single-chip system always seems just the next generation away, for most projects that is a reality that is either unattainable or inadvisable in good engineering practice. Rather than being driven by a conceptual goal like "get the system down to two chips" it pays to be pragmatic in each integration decision and to weigh the costs of integration of each piece in terms of its contribution to our system as a whole. The result will be a product that takes maximum advantage of the integration and convergence opportunities available at the time, without restricting its evolution potential and flexibility in light of potentially changing market and competitive conditions.

Kevin Morris, Embedded Technology Journal

October 4, 2005