Embedded Systems Conference 2004
Class # 202: ***Choosing a 32-Bit Microprocessor***

Jim Turley
jim@jimturley.com
www.JimTurley.com

Everything You Know Is Wrong

Selecting a microprocessor is no easy matter. Looking just at 32-bit microprocessors for embedded applications, more than 115 different chips are currently available. How many can you name?

Compared to the 4–5 different PC processor chips available at any given time, selecting an embedded microprocessor is a daunting task. Moreover, the choice of processor affects everyone connected with the project – programmers, hardware engineers, technical support, and sometimes even marketing and sales. Microprocessors also dictate growth paths, upgrades, performance headroom, development tools, operating systems, programming languages, semiconductor vendors, ASIC availability, licensing fees, royalties, power consumption, RF interference, heat, PC board design, and still more factors. Clearly, this is not a decision to be made lightly.

Yet *lightly* exactly how many developers approach the task. "We'll just use what we used at my last job," some engineers might say. "Never underestimate the influence of in-flight magazines on your company's strategy," is how another engineer put it. Poorly informed executives or managers sometimes dictate the choice of this or that CPU based on some overheard tidbit of market gossip or a vendor's glossy article proclaiming its advanced technology. The truth is often hard to glean from the raw data and microprocessors are only partly made of silicon. The rest is software, support, and reputation.

There is seldom one right microprocessor for any given task. With so many choices, it's normal to find a handful of chips that are equally well suited. Then it comes down to intangible elements: does the chip have the right future roadmap? Is it supported by your favorite compiler or operating system? Does the chip maker have a good reputation in the market? Do any of your fellow engineers have previous experience with this particular device? If so, was it a happy experience?

The Nine Evaluation Criteria:

They are:

- Price
- Power consumption (including heat dissipation)
- Performance (measured along many axes)
- Software support (including hardware development tools)
- Code density
- Software compatibility
- Future roadmap and growth path
- Availability (including core IP and/or package type)
- Intangibles, including familiarity and reputation

Price

To no one's great surprise, price usually ranks first among all the criteria that designers use to evaluate a processor. Embedded designs are usually price-constrained and it's no use check out chips that cost ten times more than you can spend.

The good news is, there are a lot of 32-bit processors priced from as little as $5 to as much as $150 or more. More surprising, the expensive chips aren't always the fastest ones, depending on your definition of fast (see *Performance,* below.)

Obviously, you'll want to negotiate pricing with your processor vendor. Keep in mind that prices decline with volume, and that most customers overestimate their volume. We all want our next widget to be a million-seller, but very few actually are. Also keep in mind that although your product might be a big success in your market, it's not necessarily a big deal to the rest of the world. Your idea of "big volume" might be far different from Motorola's. Don't overestimate your own importance.

For example, a maker of industrial robots would be thrilled to ship five robots per month. That's big volume to them, but it's a rounding error to most CPU makers. Unless you work for Dell, Nokia, or General Motors don't think you can coerce CPU makers into special pricing or delivery arrangements just for you.

Prices are negotiable, and although silicon chips do cost something to make, the *cost* of a chip has little to do with the *price* of a chip. Prices are set by marketing departments, not by simply adding a fair profit to the manufacturing cost. Prices have more to do with market pressures and competition than with silicon, plastic, and labor.

Power Consumption

For some designers, power is a big deal. For others, it's a don't-care. Power equates to battery life, heat dissipation, and (to a minor extent) reliability but unless you're optimizing for one of these criteria, power can be a non-issue.

Claims of power consumption by the CPU makers are just as squishy and unreliable as are claims of performance. You would think that power consumption would be an objective, easily measured characteristic – but it's not. Be very suspicious of power claims and measure any important metrics on your own.

Power consumption varies as the *square* of the chip's voltage, so changing the supply voltage (which many chips allow) can make a very big difference. Power varies *linearly* with speed, so changing the chip's frequency (which many also allow) makes a noticeable difference as well. Workload also affects power, so the software the chip is running makes a difference – something that's either overlooked or turned to the manufacturer's advantage in most databook summaries.

Finally, don't overlook the chip's bus structure and the way it connects to memory chips (RAM, EPROM, etc.). These can make more difference than you might think. Today's low-power microprocessors are extremely efficient with their own power but they can't change the way DRAMs work. Every time the processor accesses external memory it burns energy by toggling 32 data-bus lines, a few dozen address-bus lines, several control lines, and who-knows-how-many DRAM chips. It's not unusual for the DRAMs to use more power than the processor, so optimizing the processor's power may be a complete waste of time.

Performance

You would think that performance would be the #1 criterion for choosing any processor, yet surveys repeatedly show that that's rarely the case. Performance usually ranks around third or fourth in priority among designers' important considerations. (Price, support, and availability rank higher.)

It's good that performance is not weighted more heavily, because it's a slippery characteristic to measure. The very concept of performance means different things to different users. There's no such thing as a single "drag race" that can identify the fastest chip; there's no single figure of merit. Almost any 32-bit processor can claim to be the best along some axis of performance, and justify it.

Unless you're going to personally benchmark different processors running your code in your expected environment, third-party benchmarks and datasheet specs will be of little use to you. Again, so-called standard benchmarks just aren't very useful. They're simplified to the point of uselessness, and those that aren't simplified generally wind up measuring aspects that aren't important to you. It's a dismal situation.

Processors that are good at some tasks, such as MPEG decompression, may be terrible at other tasks, such as handling complex decision trees. Chips have a surprising amount of variability even doing simple 32-bit multiplication: the differences can be as much as 20-to-1 among supposedly similar processors.

The much-misused MIPS (millions of instructions per second) rating is particularly dangerous. Apart from having three different and unrelated meanings, MIPS numbers are frequently padded by overeager marketing department in order to gain a perceived edge in performance. Be especially suspicious of any performance claims expressed in MIPS.

Software Support

Software support is often the #1 concern when choosing a new processor. It’s not unusual for design teams to pick their compiler, debugger, and operating system first – and *then* choose a processor that supports those tools. There’s nothing wrong with this software-centric point of view. In fact, more companies should do it.

This method places the concerns of the software developers above those of the hardware developers, but that’s a management-level decision. There’s little concrete guidance to be given here, since the decision is very personal and dependant on the organization. If tools and development systems are important, start there. If not, let your hardware team go wild picking the chip they prefer.

Code Density

“Code density” is simply the ratio between the size of your source code and the size of your object code. The smaller the object code, the better your code density. Good code density is a good thing because it means you need less memory (RAM, EPROM, or whatever) to execute your code.

The description above sounds as if the compiler controls code density – it doesn’t. Code density is determined by your processor, not your compiler. Sure, some compilers produce tighter object code than others, but all compilers are limited by the underlying assembly-language instruction set of the processor they’re targeting. Compilers can’t produce what the chip doesn’t support.

If code density (i.e., memory footprint) is important to you, then you need to pay special attention to your choice of microprocessor. Processors can make a 2:1 different in code density. That is, the exact same source code compiled for two different chips can produce executable binaries that are double (or half) the size of the other chip’s code. No amount of compiler tweaking will get around differences that big. It’s an inherent feature of each and every microprocessor.

As a rule of thumb, RISC processors have poor code density, while CISC processors have comparatively good code density. For example, MIPS, ARM, and PowerPC chips will have poorer code density than a 68030, ’386, or ColdFire processor.

Software Compatibility

This one's easy because it's binary. Your new chip is either compatible with your old/previous chip or it's not. At least, that's how it's supposed to work. You'd be surprised how gray this black-and-white decision can become.

Good examples of compatible chips are the x86 processors from Intel, AMD, and a few other companies. Nobody questions whether a 386 can run older 286 code – you know it can. Compatibility up and down the x86 product line is assumed. It's a given. That's why we keep buying x86-based PCs. We know our old software will run. These chips are *binary compatible.*

A less-compatible example is Motorola's ColdFire family. These chips are sort of, but not completely, compatible with the venerable 68K family (68000, 68020, etc.). For the most part, older 68K software will *not* run on a new ColdFire chip without recompiling. Programmers with 68K experience will feel instantly at home with ColdFire, and that familiarity may or may not be useful to you. If you've got to recompile your source code anyway, why not consider compiling it for a completely different processor? ColdFire and 68K chips are *source code compatible.*

Still another level of compatibility is that offered by some of TI's digital signal processing (DSP) chips. They are neither source- nor binary-compatible, but merely "architecturally compatible." This simply means that the chips have similar features and register sets but different instruction sets and resources. In other words, programmers familiar with some TI DSPs will quickly learn to program other TI DSPs, but no software (or at least, little software) will carry straight across. These chips are *incompatible.*

The quick-and-dirty rule of thumb about compatibility is that it's only important if your product runs third-party software. In other words, if you're writing all the code your product will ever need, compatibility's not important. On the other hand, if your product can run store-bought software then compatibility is far more important – in fact, it might be *the* most important characteristic of all (as with PCs). This is obviously a simplified view, and there are many variations in between these two extremes, but it's a start.

Future Roadmap and Growth Path

Some microprocessors are unique one-offs. Others are part of a long and distinguished family. Which category "Chip X" falls into might determine whether or not it's right for you.

At first, you might think that nobody would want to be stranded using a one-of-a-kind processor with no future growth path – a dead-end chip, if you will. Actually, that's not always a bad thing. (Usually, but not always.) Deeply embedded systems with no third-party software are "invisible" to the users, who couldn't care less whose chip is in their pager, antilock brake system, or network router. If you're designing these systems, the best chip for the job might not have an upgrade path. And that might be just fine, assuming you don't mind recompiling your code for the next-generation product. Even a dead-end chip might be just right for today's design.

By and large, programmers prefer a chip that's got some future ahead of it. Hardware upgrade paths and software compatibility *(above)* are related. If you've got a big investment in your software – and who hasn't? – then you'll want to protect it by coding for a chip that will have faster and better offspring next year. Generally, chip vendors are only too happy to talk about next year's model, or show off the PowerPoint graphic that show their CPU family shooting off up and to the right.

Availability

Obviously, you'll want your new chip to be available when you need it. And you'll want to be sure of an uninterrupted supply once you start production. That's often a matter for purchasing departments, but it affects the engineers' decisions as well.

Microprocessors are almost never second-sourced these days. That means no matter which chip you pick or which architecture or family it belongs to, it will only be available from exactly one company. Gone are the days of hedging your best with second-sourced devices. If that's not acceptable, then you need to restrict your shopping to 8-bit and 16-bit processors.

"Availability" can mean different things, however. Sometimes you don't want to buy chips at all, you want to license IP. Licensing an IP (intellectual property) core allows you to create your own chips that include someone else's processor design. This is a very popular – and immensely expensive – alternative to buying commercial off-the-shelf processors. Not all 32-bit processors are available for license, however. In fact, most are not. The decision to buy chips or license IP has more to do with your company's business model than with the choice of CPU, so we'll leave the rest of this discussion for another time.

Intangibles

This is what it all comes down to: the intangibles. It's the reason we buy the red sports car instead of the sensible brown family sedan; the designer shoes instead of the comfortable sneakers; the boat instead of, well, anything.

In the end, you've got to be happy with your processor. You're going to live with it for many years, and if your product will span many generations you'll be living with this chip's descendants as well. Gut feelings shouldn't be ignored when you make a decision this momentous.

Reputation counts for something. Does this CPU company have a good track record? Have they been in business – in the *embedded* business – a long time, or are they a newcomer? Have I used chips from this family before, or have my colleagues used them before? What do the newsgroups say?

Some chips have a "cool" aura surrounding them, while others are less exciting. It's hard to say why this is. In the 1990s RISC chips were far cooler than CISC chips, even though the CISC processors often had (and still have) real advantages. The same goes for anything made by Intel: some engineers avoid Intel processors while others prefer them. Go figure.

A lot of what leads designers, engineers, or programmers to a particular chip is misinformation and, frankly, propaganda. Without checking the facts, making their own measurements, or even simply asking for an objective opinion, normally intelligent designers can be swayed by hype and advertising. Engineers are people, too, and people sometimes make funny and irrational decisions. That's okay if you know you're doing it, if you go in with your eyes open. But most engineers and engineering managers would prefer to make objective, informed, and dispassionate decisions based on reality. It is for those people that this primer has been created.

30

Jim Turley is an independent analyst, columnist, and speaker specializing in microprocessors and semiconductor intellectual property. He is editor of Silicon-Insider*, a columnist for* Embedded Systems Programming *(ESP), was past editor of both* Microprocessor Report *and* Embedded Processor Watch, *and host of the annual* Microprocessor Forum *and* Embedded Processor Forum *conferences. For a good time call (831) 375-8086, write jim@jimturley.com or visit www.jimturley.com*.