What will next generation embedded design look like?

In future years, the ever-changing embedded systems design environment will continue to evolve. How must the development environment change to accommodate it? Ata Khan of Cypress has a few ideas.

By Ata Khan, Cypress Semiconductor
Embedded.com
(10/21/08, 01:25:00 AM EDT)

Embedded processors are everywhere. Almost everything powered by electricity has an embedded processor for the control or processing functions (or both) required by the application. The proliferation and the ever-increasing sophistication of these processors over more than three decades have been driven by decreasing costs and increasing processing and controlling power.

Moore's law, competitive pressure, and rapid innovation in consumer devices have led to everyday devices incorporating machines that have the same processing power that mainframe enterprise computers delivered not so long ago.

Much of this innovation has been driven by human consumer behavior that, after a while, tends to ignore linear improvements but responds strongly to exponential advances, thus driving the push to more powerful systems with more features and more complexity. As a consequence, it is now common for consumer devices to incorporate embedded processors with processing power in triple-digit MIPS,

Gigabytes of storage, and access to all kinds of commonly used communication protocols is not a problem with respect to the hardware available (the processors themselves) nor to the software development tools, operating systems, or compilers. In addition, the embedded designers themselves have the capacity to understand the evolving hardware and software and to use them to build marketable applications.

Rather, the problems that complexity brings are related to the fact that human design bandwidth has not grown at the same rate as processing power. As a result, ever-increasing complexity brings inevitable delays and simply adding people is not the answer as many well-known books on productivity (The Mythical Man-Month is perhaps the best known) have shown.

The current situation
Designing a modern embedded system is a complex multi-tiered activity. First comes the need to define the behavior required of the system by its potential customers and to understand the competitive landscape and how to leverage it.

Next comes determining if the needed competence exists in the organization, and reducing/relating the system specifications to the software, tools, processors, and other components required. All of these major tasks need to be completed before the actual task of designing hardware and writing code can begin.

Once all the background work to get a design underway is done, the hard, repetitive work of hardware and software design begins. Typically, embedded designs consist of selections from three major categories of blocks: Processing, digital (peripherals and logic), and analog (the physical interface for sensing and controlling).

A system is generally assembled of various components from the three major categories based on the hardware interfaces required by the system, and the software must be built to make the system perform the processing and control functions required. Once a particular design is "set" in software and hardware, it is difficult, costly, and time-consuming to change because of interactions between software and hardware within the classes of blocks.

Embedded designers need to get to market ahead of their competitors. To have an early market advantage and re-use their investments in knowledge and tools as much as possible, their selected architectures and design platforms have to be flexible and adaptable.

These tools and components must enable significant last minute changes (indeed, continuous changes), as well as be able to utilize industry standard eco-systems to take advantage of a large developer community and standard tools.

Compounding these challenges is the fact that the longer the design cycle, the higher the probability of the design requirements changing, which, of course, makes the design cycle even longer, and so on. Rapid product development therefore not only provides a financial advantage but reduces uncertainty and offers the possibility of earlier customer feedback and market leadership.

A new way of designing

Given that a typical embedded system is a combination of processing and digital and analog blocks, and that the embedded system consists of the whole of the combining these blocks with the appropriate software, developers need to ask the following key questions:

1. How do we make sure we're speeding up the total design process? Amdahl's law, paraphrased, states that it is the sequential portion of the total processes that determine the overall potential improvement. As applied here, some basic processes must change.

2. How do we accelerate the learning phases in order to converge more quickly on the marketable design? Experience teaches us that it's far better to go through a process of stepwise refinement and convergence than to push system testing out to the end. A corollary of this is that the earlier bugs are found, the cheaper they are to address.

3. How do we reduce overall design effort? Put another way, which parts of the system can be provided for in terms of reusable modules or generated intelligently to save time and effort.

4. How do we allow for rapid change in all phases of the design process? Design changes made earlier in the design process are easy to implement. Because of interdependencies across a design, it's not at all easy to make rapid changes once a design has firmed up.

5. How do we allow for updating both hardware and software in the field? It's relatively easy, given that some means of communications exist, to update embedded software. Additionally software updates have become quite common and are accepted by many consumers as part of the ownership experience of certain electronic devices. The difficulty is in updating hardware functionality as well. For example, what if the behavior of a certain state machine needs to be changed, or if the machine has to perform different tasks at different times?

6. How do we allow for standard tools to be used? As compilers and debuggers evolve over time, there is much learning and improvement (and at many users' expense) embodied in standard tools. New embedded design techniques must support the use of industry standard tools to maximize improvement gains while minimizing designer learning investment.

7. Are we providing a Platform? It is essential that a scalable platform, in terms of well-defined hardware and software architectural features, be provided so that designs may be scaled, depending upon the application, and previous efforts can be reused (whether hardware or software).

The ideal embedded platform would therefore have a basic functional structure that is highly flexible and dynamically modifiable. It would also automate all steps of the development cycle other than creating the code the embedded systems designer develops for the particular application.

The architecture would be highly scalable to support reusability of knowledge and tools and be based on the use of standard development tools supported by a large ecosystem. In addition, it would incorporate system-level integration and low power and size by being incorporated in silicon built using advanced processes so as to optimize value to the customer and reduce overall power consumption.

A new design flow

Such an architecture calls for an approach vastly different from those employed in building an embedded design using microcontrollers with fixed functionality and external components and developing software and firmware manually. This design approach would be based on an integrated embedded solution with well-defined hardware and software.

The hardware would consist of a scalable silicon platform consisting of a processing subsystem (including memory); an analog subsystem consisting of various functional blocks and commonly used fixed-function peripherals; and programmable digital blocks, all connectable via a true switched fabric.

The software would consist of a development environment which, starting with a blank canvas, allows the user to construct a complete system consisting of:

Commonly used peripherals

Serial Communication Blocks

Standard Analog blocks

Boolean primitives

Hierarchically created components (in other words, created by the user)

User-created RTL

The software tool would embody several complex functions:

A blank canvas for the user to place and connect components from the list above.

A comprehensive list of common prebuilt configurable components for users to select from and place on the canvas

The ability to place Boolean primitives, create components embodying other functions, and specify the behavior of components via RTL.

Fixed-function and/or programmable resources that would be selected as required.

Synthesis of user RTL, placement of components in the design and fitting functionality to programmable digital logic resources.

Placing and routing components on the silicon platform using a switched fabric including connections to the user's choice of I/O pins.

Generating APIs (including APIs for user-defined components) and device drivers for prebuilt components. APIs created for the user-defined components would allow passing of parameters to constituent blocks.

Allowing the functionality of blocks to be dynamically reconfigured based on software commands, if so required.

Generate datasheets for component blocks and the chip-level entity.

Once all the above are completed, the user would then be able to add application code, working with the compilers and debuggers of his choice, to the now-optimized device. This process may be, simplistically, represented in a flow-chart in **Figure 1**.

A PSoC design flow.

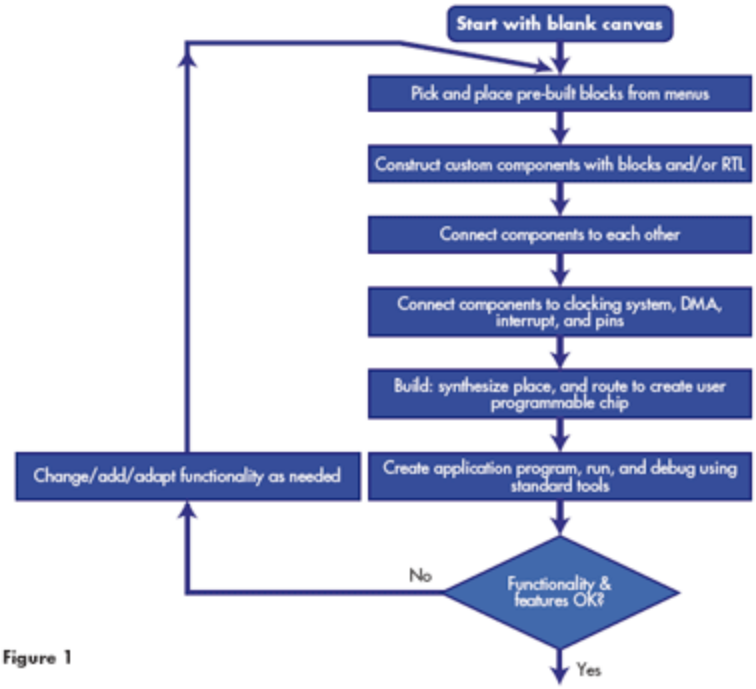

Figure 1

View the full-size image

Some examples of configuring various subsystems are shown below.

Clocking: The user is given the following menu to select clock sources from. Selection of a clock source is as easy as selecting and clicking; all connections on-chip, control register set-up, and initial clock configuration is done automatically.

Interrupts: An ISR (Interrupt Service Routine) editor allows any digital signal to be connected to an interrupt input. Such tools would allow assigning a particular Interrupt priority and vector number to a signal.

DMA: DMA is another example of a global resource (like clocking and interrupts) that many blocks may choose to use (in other words, it is up the user). The DMA editor would show all peripherals connected to the DMA request inputs and will allow priority settings so as to guarantee optimum throughput for all its clients.

What does it take to do this?
In our **Programmable SoC architecture**, Cypress has implemented a precursor of this new design flow. Experience gathered over several years of implementing and refining that flow with the help of thousands of customers has provided invaluable insight in the new directions the design flow can take. A new embedded design flow such as this cannot be defined in one area. It must encompass the following elements:

1. A Silicon platform that can be scalable and reusable in terms of both hardware and software.

2. A Software development environment that automates component selection and generation as much as possible with the goal of creating software interfaces (APIs) for all customer-specified blocks so that designers can focus on developing their value-added application code.

3. The use of standard tools and debuggers. In this way, designers can leverage reusability and take advantage of the learning curve that standard tools have benefited from, as well as tap into the large ecosystems that grow around industry standard tools.

***Ata R. Khan**, vice president of technical staff at **Cypress Semiconductor Corp.**, is responsible for strategic marketing and architecture for new Programmable System-on-Chip products. He joined Cypress in November 2006 from NXP (formerly Philips Semiconductors). Ata has an MSEE degree from UC Berkeley, an MBA from the University of Santa Clara, and a BSEE from the NED Engineering University.*

Please login or register here to post a comment