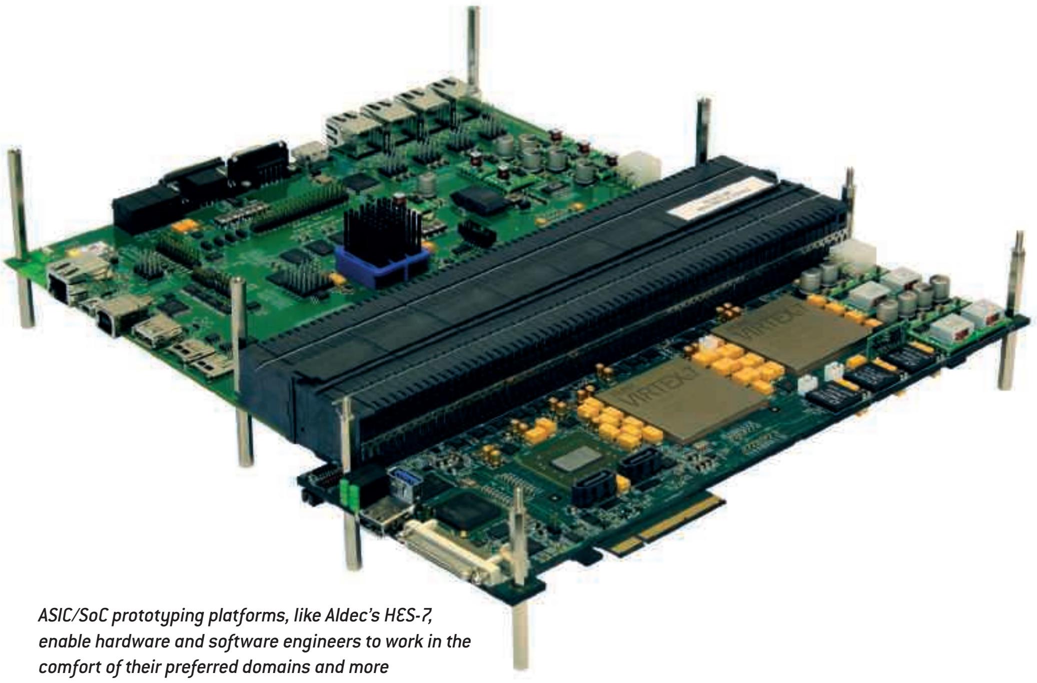
Divide and conquer

Prototyping platforms and an IP based development methodology enable SoC hardware and software engineers to work more productively. By **Bill Tomas**.


ASIC/SoC prototyping platforms, like Aldec's HES-7, enable hardware and software engineers to work in the comfort of their preferred domains and more productively.

Time to market pressures and growing design complexity are steering SoC designers towards an IP based development methodology, using a mix of off the shelf and home grown building blocks. For example, SoCs found in tablet computers or smartphones typically have between 20 and 30 different IP blocks.

Modern SoCs are designed using two disparate flows – one for hardware and one for software – and the lack of a good codesign approach makes specification revision difficult and prolongs the development phase. Meanwhile, the lack of a unified representation between the flows can lead to difficulties in verifying the system's functionality.

Therefore, a need exists for greater collaboration between hardware and software engineers, particularly in light of the intense 'embedded nature' of a modern SoC; which is typically a multicore platform with an operating system, a variety of processing pipelines, memory and so on.

Developing the SoC on some form of prototyping platform is essential. Such a platform would typically include one or more processors, memory, reconfigurable hardware (for example, fpgas) and I/O driver ports. Moreover, the platform needs to support concurrent engineering, where hardware and software engineers can work independently within familiar domains.

Who does what?

Many companies build a small number of dedicated pcbs for development purposes, although off the shelf asic/SoC prototyping platforms can save time and cost. Let's examine how some specific functionality might be developed on a prototyping platform.

Image and video processing engines are not only used in high volume consumer products (like tablets and smartphones), but also in lower volume specialist applications, such as medical devices and industrial controls. While they serve different purposes, the engines have common features, so there is no reason why they can't share resources.

Most image/video processing algorithms can be implemented in either software or hardware, so an early task is to decide which functions will be best implemented in which domain. For example, high speed, parallel processing can be implemented in hardware, while lower volume data tasks are best handled in software. Once the allocations have been made, the development teams can then follow their respective design flows (see fig 1).

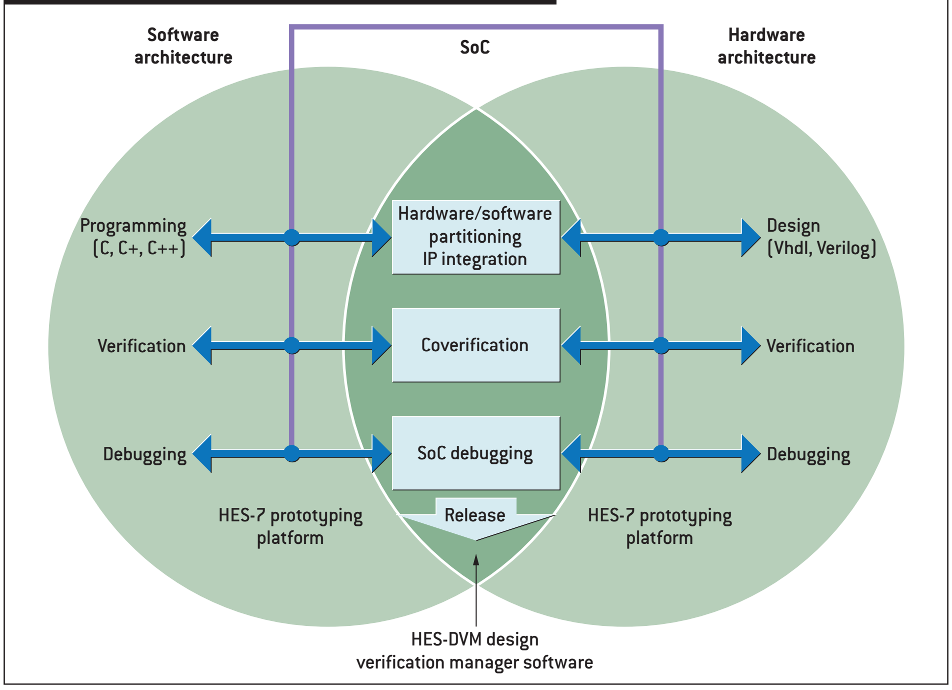
Partitioning hardware and software early on is of considerable importance when dealing with image/video processing, particularly if there is a need/wish to develop a processing engine in house, as much testing will be required and the more you can offload into hardware, the better.

For example, an HD frame might be 1080 x 1920 pixels; at a 30Hz refresh rate, that's more than 62million pixel refreshes a second. If that video is to be image corrected and have a 'special effect' added, what are the verification consequences of implementing 'home grown' algorithms in software or hardware?

Correcting and modifying each pixel in software may take 10 cpu cycles – that second's worth of video will require more than 620m emulation steps and that could take months to perform. Algorithms implemented in hardware can also be verified in hardware – and accelerators should be able to process the same amount of video in just a few minutes.

Much IP can be obtained as off the shelf 'components' – such as uarts, USB drivers, PCIe, Ethernet MAC, DisplayPort, RapidIO and Gigabit

Fig 1: The hardware assisted verification platform approach



transceivers – which come preverified for certain fpga targets. It is better to use these than make your own because of the time saved, plus the fact these are unlikely to be the things that differentiate your end product.

As for software IP, our theoretical image and video processing engine would probably use off the shelf algorithms for image reconstruction, automatic contrast/white balance and image coding (such as Jpeg, Mpeg and H.264 AVC), but remember that parallel algorithms, although available in software, are best offloaded onto hardware (fpga). Contenders here include colour correction, defective pixel correction and image noise reduction.

Video and image processing applications also require frame buffers for many tasks. Data stored in external memory is mapped to memory interfaces available on programmable logic and is typically accessed via high bandwidth communication channels. Many video processing engines make use of external buffers for temporary storage of video frames, but it can be challenging to synchronise video when data is clocked at different rates.

Off the shelf IP can help here: for example, Xilinx' Video Direct Memory Access (vdma) LogiCORE IP allows video cores, implemented within a number of its Spartan and Virtex fpgas, to access external memory via a video frame buffer controller. The vdma uses a synchronisation mechanism called Gen-Lock to simplify the movement of data from one processing (clocked) domain to another.

Consider also where the processing engine fits into the SoC. The input might be a still image or a video stream from a camera, or recalled from memory (within the end product) via USB or SD; the output might be to a display, an HDMI port or back to memory. Again, off the shelf IP cores are available.

From the software perspective, driver development can be similarly simplified. For example, the Android operating system has a Linux kernel with drivers for displays, a camera, USB, Bluetooth and WiFi, amongst others. Software developers create C, C++ or Java code for the Android software stack, incorporating the drivers, libraries and middleware to develop services and management during runtime operation.

The common ground

Hardware and software need to come together as soon as possible and that can be facilitated by a scalable and modular asic/SoC prototyping platform.

Aldec's HES-7 family of fpga based prototyping boards takes advantage of Xilinx' Virtex-7 2000T 3D fpga to give a hardware design capacity of up to 24m asic gates on a dual device HES-7 board. A non proprietary high speed backplane connector allows up to four boards to be connected; producing a considerable parallel processing capability for image and video processing pipelines and enabling HES-7 to scale and support 96m asic gates.

Most recently, Aldec added an SoC essentials daughter board to support the ARM dual core Cortex-A9 MPCore through a Zynq-7000 All Programmable SoC; allowing software designers to make use of the serial/sequential processing capabilities of the core for applications that require intensive computations. Coupled with open source Linux, Android and FreeRTOS solutions, this makes for a powerful development environment.

In addition to the scalable capacity and ARM support, HES-7 provides SoC peripherals, giving designers the ability to interface to real world stimuli. For instance, on board gigabit Ethernet transceivers can be used to develop networking applications, while IEEE802.11 b/g/n and Bluetooth v2.1 are available to develop wireless systems. There is also a high performance HDMI transmitter.

Summary

As increasingly complex SoCs are needed for smartphones, digital set top boxes, tablets and other consumer 'must have' products, development and verification activities are best assisted through the use of an embedded systems prototyping platform; a modular and scalable 'work environment' for software and hardware engineers to share.

Author profile:

Bill Tomas is HES-7 product engineer with Aldec.