White Paper

A USB Case Study

Accelerating Software Driver Development using Virtual Platforms

Frank Schirrmeister, Director of Product Marketing, Synopsys System-Level Solutions

August 2008

Introduction

Over the course of the last decade, the cost of software has become the undisputed dominant factor in electronics design. Software development has fundamentally changed the shape of the electronics industry, from large integrated device manufacturers (IDMs) to a complex disaggregated design chain of heavily interacting companies. Hardware intellectual property (IP) providers deliver blocks and subsystems to semiconductor companies, who in turn deliver chips and board subsystems to system houses who increasingly seek differentiation through software. All of the hardware-related players interact with software providers who create everything from low level drivers to complex end-user applications. The recent Nokia divesture of hardware efforts to STMicroelectronics combined with the full acquisition of Symbian is a good example of the increased differentiation through software in system houses. These steps allow Nokia to sharpen their focus in the key differentiating areas of software, services and specialized hardware (e.g., modem design.). Apple CEO Steve Jobs spelled it out explicitly: “Phone differentiation used to be about radios and antennas and things like that. We think, going forward, the phone of the future will be differentiated by software.” [1].

This white paper analyzes the root causes of escalating software development effort. It will highlight software driver development as a key factor in time-to-market results for combined hardware/software products, and how higher abstraction layers of the hardware allow some areas of software development (i.e., drivers, middleware and OS’s) to become largely independent of the target hardware architecture. We will introduce virtual platforms and FPGA prototypes as key ingredients to expedite the availability of suitable development targets for driver, OS, middleware and application software development Using the example of the Synopsys DesignWare® High-Speed USB On-The-Go (OTG) controller we will show how the specific challenges for software driver development productivity – early availability, visibility and control - can be addressed using virtual platforms and applied to other complex IP blocks.

Software Determines the Success of Chip Development Projects

Ever since Moore’s Law was formulated, hardware designers have been in a race to fill up complexities offered by more and more silicon real estate. In the last decade, software has gained more and more importance as part of chip development. This trend is largely driven by the flexibility requirements of a diverse set of application domains, particularly those serving consumer markets. Figure 1 shows that especially in process nodes smaller than 90nm, the effort of software development already exceeds the hardware effort (Source: [2]). With the rising non-recurring engineering (NRE) cost of chips at smaller geometries, the investment to develop them has to be recovered over a wider range of products and product generations. In addition, to properly recuperate investments, not only the “time to market”, but also the “time in market” (the time how long a chip can be used in a specific market) needs to be optimized by allowing customization and feature extensions to accommodate the increasing influence of end-users. Customization through software can be relatively low-cost compared to hardware customization and can support fast time-to-volume for new products.

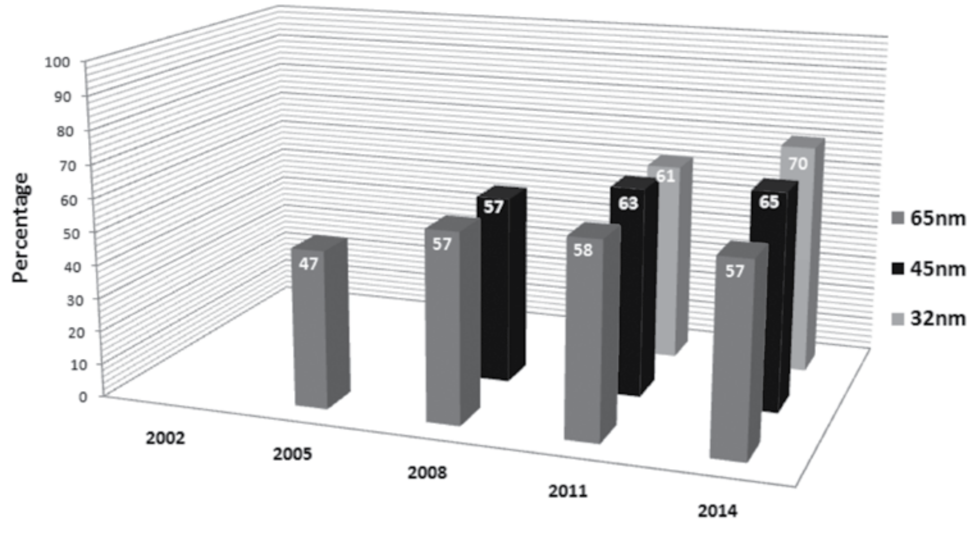


Figure 1: Software as percentage of total project effort

In addition, International Business Strategies, Inc. (IBS) predicts that at 65nm more than 90 percent of system IC designs will be implemented with embedded processor cores. Adding software functionality to these specific processor cores allows a wide set of system features to be supported, with relatively low costs for adding new features.

To understand the reason for the dominance and importance of software in modern electronic devices, it is important to understand the design flow as it is practiced most often today. Figure 2 illustrates a traditional design flow from chip to product design. For a new design, the semiconductor supplier will start with the definition of the chip architecture, often in coordination with lead customers such as system houses using the chip to be developed in their end products.

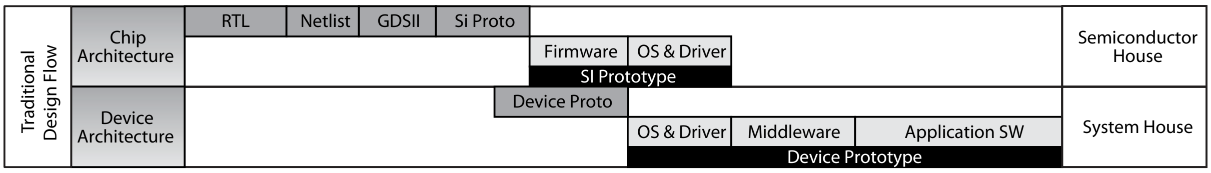


Figure 2: Traditional hardware-software design flow

While in theory the development of firmware and drivers can be done “blind” based on register specifications provided by the hardware teams, the reality is that integration with the hardware – which has to wait until silicon prototypes are available – is traditionally quite long and a source of unforeseen surprises. The best case is that any integration issues can be corrected in software without having to re-spin the silicon, causing relatively minor but still important delays. The worst case scenario of a silicon re-spin can be catastrophic for a program. Once the silicon prototype is available, the system house can integrate it into their product prototype and add the middleware and application software as indicated in Figure 2.

The border between semiconductor and system houses can be blurry in terms of who is responsible for which software. Often the semiconductor house has to provide the middleware as well. In any case, the length of the software development phase determines when the semiconductor house can actually start recuperating its chip development cost, which cannot happen in earnest until the end product containing the chip reaches the market.

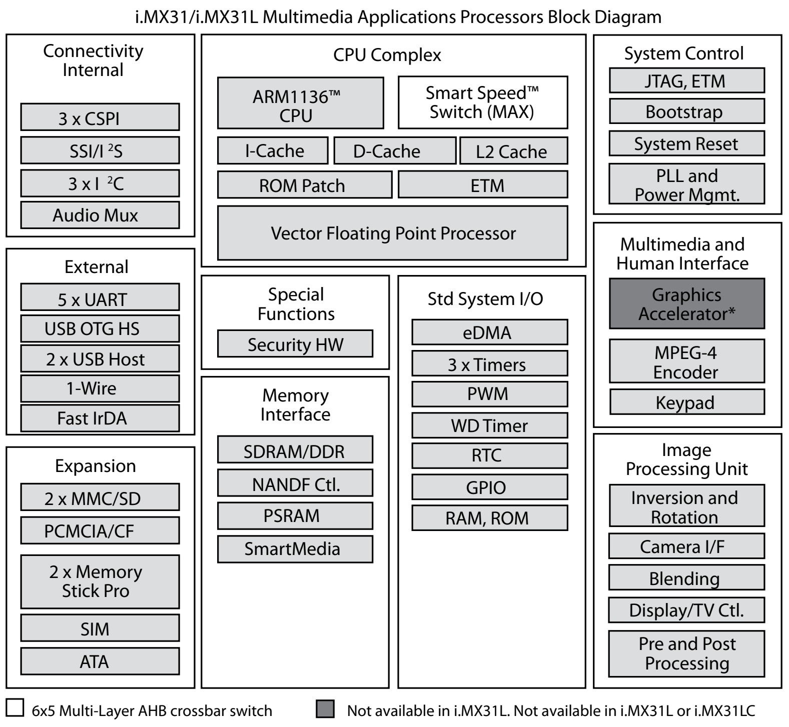


Figure 3: Block diagram of the Freescale i.MX31 application processor (from [3])

To get a better picture of software complexity, let's review a real hardware project and its software requirements. Figure 3 shows the block diagram of the Freescale I.MX31 Application Processor.

The I.MX31 is designed for the high-tier and mid-tier smartphone markets and portable media players, which provide low-power solutions for high-performance demanding multimedia and graphics applications. It is built around the ARM11 MCU core and implemented in 90nm technology. The I.MX31 includes various subsystems supporting features such as

- Multimedia and floating-point hardware acceleration for various video encoding and decoding standards
- 3-D graphics and other applications acceleration with the ARM® Vector Floating Point co-processor
- Advanced power management
- Multiple communication and expansion ports, including a fast parallel interface to an external graphic accelerator
- Security-related features, and
- Various connectivity channels like USB and SATA.

Figure 4 illustrates a software-architecture for the Freescale i.MX31 application processor. The software is typically designed in layers, starting at the level very close to the hardware. The firmware level is typically followed by real-time operating systems (RTOS) and drivers. Several lower level protocols HTTP, SMTP and RTP are shown together with multimedia codecs like MPEG-4 and H.264. At the application level, functionality for browsers, calendars, multimedia and other accessories can be implemented in a largely hardware independent fashion.

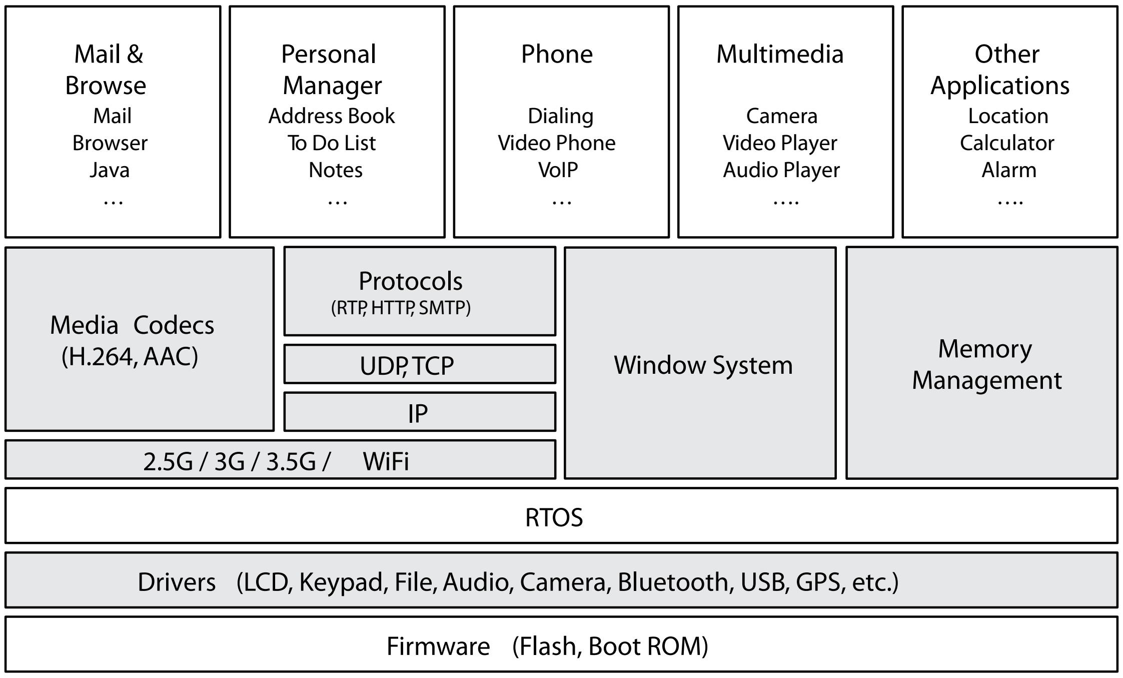
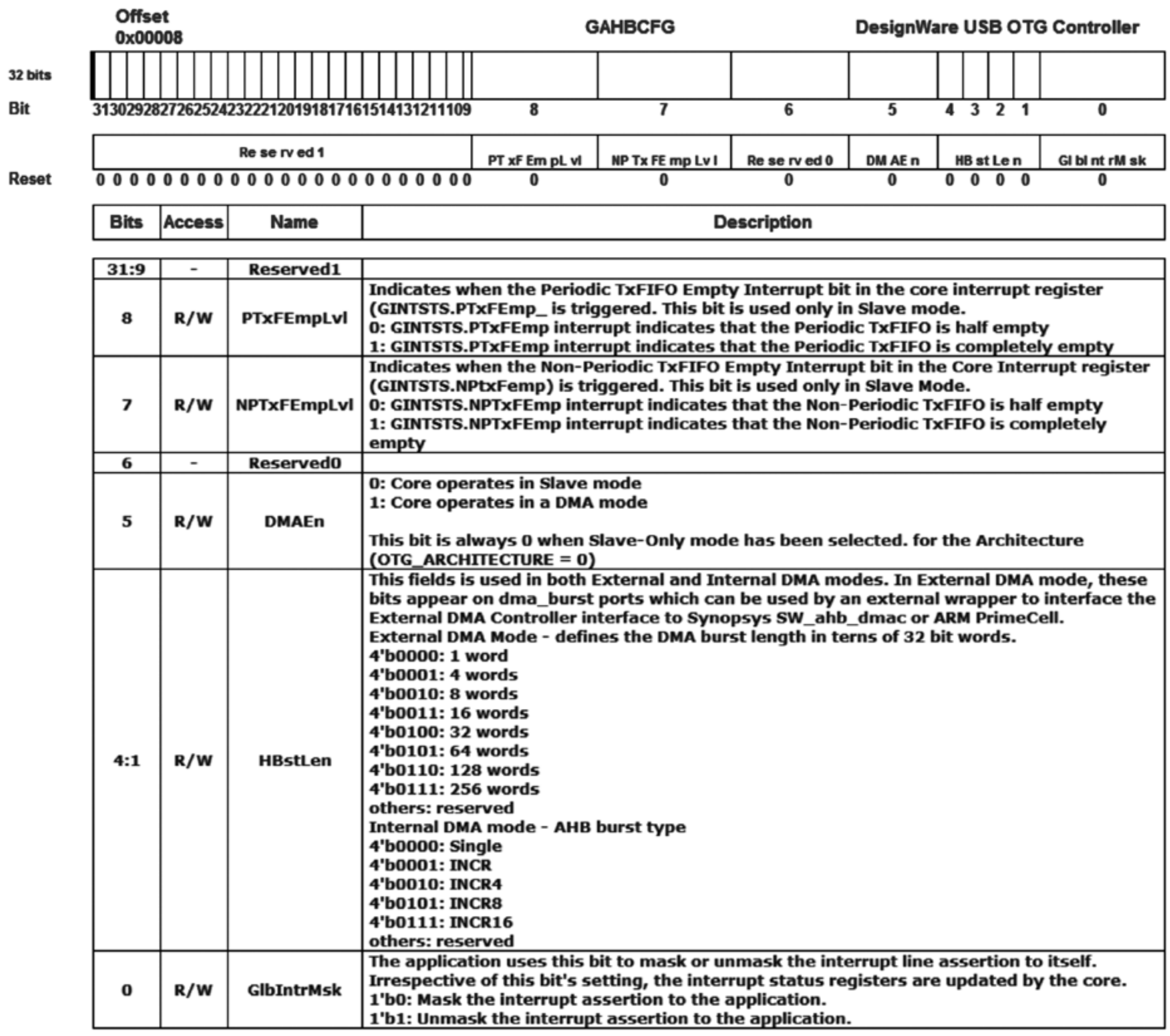


Figure 4: Example mobile phone software protocol stack

The "higher" software layers shown in Figure 4 has only selective dependency on the hardware, with the implication of relaxed accuracy requirements of the development target for software design. In some timing critical software, timing accurate response times cannot be neglected; for these cases, cycle accurate representations of the hardware may be required. However, in the majority of cases, only a precise representation of the register image of the peripherals together with correct functional representation of the peripheral's operation is required by the software team– even for the development of most of the lower level drivers.

Figure 5 shows an example of such a precise register description of the Core AHB Configuration Register as it is used in the Synopsys DesignWare USB OTG controller. This register is used to configure the core after power-on or a change in mode of operation. This register mainly contains AHB system-related configuration parameters. The application must program this register before starting any transactions on either the AHB bus or the USB. The USB controller has about 70 registers like this one in addition to various FIFOs that can be accessed as register accesses as well.



Offset 0x00008 GAHBCFG DesignWare USB OTG Controller

32 bits

Bit	313029282726252423222120191817161514131211109	8	7	6	5	4 3 2 1	0
	Re se rv ed 1	PT xF Em pL vl	NP Tx FE mp Lv l	Re se rv ed 0	DM AE n	HB st Le n	Gl bl nt rM sk
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0 0 0 0	0

Bits	Access	Name	Description
31:9	-	Reserved1	
8	R/W	PTxFEmpLvl	Indicates when the Periodic TxFIFO Empty Interrupt bit in the core interrupt register (GINTSTS.PTxFEmp_ is triggered. This bit is used only in Slave mode. 0: GINTSTS.PTxFEmp interrupt indicates that the Periodic TxFIFO is half empty 1: GINTSTS.PTxFEmp interrupt indicates that the Periodic TxFIFO is completely empty
7	R/W	NPTxFEmpLvl	Indicates when the Non-Periodic TxFIFO Empty Interrupt bit in the Core Interrupt register (GINTSTS.NPtxFemp) is triggered. This bit is used only in Slave Mode. 0: GINTSTS.NPTxFEmp interrupt indicates that the Non-Periodic TxFIFO is half empty 1: GINTSTS.NPTxFEmp interrupt indicates that the Non-Periodic TxFIFO is completely empty
6	-	Reserved0	
5	R/W	DMAEn	0: Core operates in Slave mode 1: Core operates in a DMA mode This bit is always 0 when Slave-Only mode has been selected. for the Architecture (OTG_ARCHITECTURE = 0)
4:1	R/W	HBstLen	This fields is used in both External and Internal DMA modes. In External DMA mode, these bits appear on dma_burst ports which can be used by an external wrapper to interface the External DMA Controller interface to Synopsys SW_ahb_dmac or ARM PrimeCell. External DMA Mode - defines the DMA burst length in terns of 32 bit words. 4'b0000: 1 word 4'b0001: 4 words 4'b0010: 8 words 4'b0011: 16 words 4'b0100: 32 words 4'b0101: 64 words 4'b0110: 128 words 4'b0111: 256 words others: reserved Internal DMA mode - AHB burst type 4'b0000: Single 4'b0001: INCR 4'b0010: INCR4 4'b0101: INCR8 4'b0111: INCR16 others: reserved
0	R/W	GlbIntrMsk	The application uses this bit to mask or unmask the interrupt line assertion to itself. Irrespective of this bit's setting, the interrupt status registers are updated by the core. 1'b0: Mask the interrupt assertion to the application. 1'b1: Unmask the interrupt assertion to the application.

Figure 5: Example register description of the Synopsys DesignWare® USB OTG Core

Given the disadvantages of the traditional serial design flow as shown earlier in Figure 2, developers have been looking for prototyping techniques to reduce the overall product turnaround time. Some of these techniques are indicated in Figure 6, contrasting the original serial design flow with an updated, parallelized design flow.

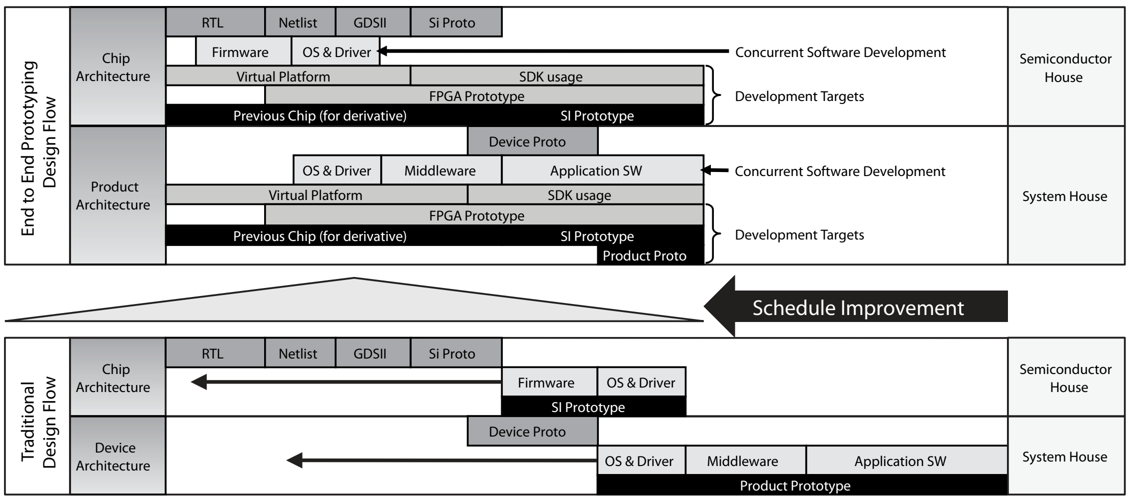


Figure 6: Parallelized hardware-software design flow using prototyping

In a derivative design, a portion of the software can be developed using the previous-generation chip. This approach often works best for the portions of the software higher up in a layered software-architecture stack as shown in Figure 4, specifically the hardware independent application software. However, given that the register fields are updated and enhanced from on chip generation to the next, this approach is difficult for lower-level portions of the software like drivers and middleware.

Later in the design flow, after the RTL is complete and has reached a stable state using functional verification techniques, FPGA prototypes can be used. They should be a pre-silicon, fully functional hardware representation of the SoC, board and I/O implementing unmodified ASIC RTL code. Optimally implemented, they can run at almost real-time with external interfaces and stimulus connected, and provide, in conjunction with hardware simulators, much better system visibility and control than the actual silicon prototype.

Virtual platforms offer a solution very early in the project, as soon as the architecture of the design has been worked out. Virtual platforms are a pre-RTL, register accurate and fully functional software model of SoC, board, I/O and user interfaces. They execute unmodified production code and run close to real-time with external interfaces like USB as "virtual I/O". Because they are fundamentally software, virtual platforms provide high system visibility and control including multi-core debug. And they can serve as an elegant vehicle of interaction between semiconductor and system house. Since the recent standardization of the OSCI TLM-2.0 transaction-level APIs, SystemC™ has become the suitable infrastructure to develop fast virtual platforms using interoperable transaction-level models and is supported by a variety of commercial products including Synopsys' Innovator product line.

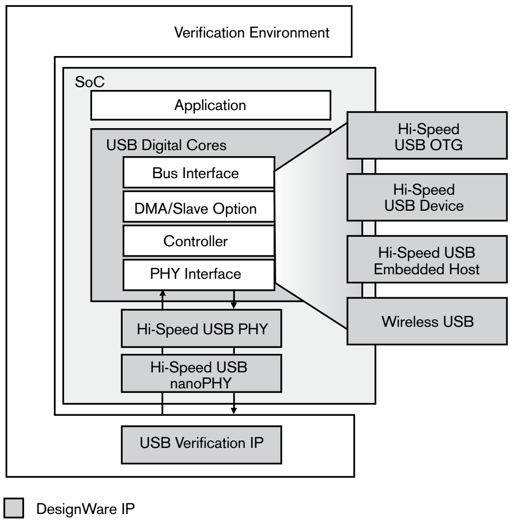


Figure 7: Synopsys DesignWare USB Solution

Even after FPGA prototypes and the actual silicon are available, virtual platforms continue to be used as software development kits (SDKs). Popular examples are the Nokia S60 SDK for the Symbian OS and the Apple iPhone SDK including an iPhone simulator, which were downloaded more than 100,000 times in the first four days of its availability (see [4]). Some SDKs like the Metrowerks Symbian Development Kits may also include hardware for timing accurate software development.

Going back to the Freescale i.MX31 application processor shown in Figure 3, a more detailed analysis reveals that this device needs about 25 drivers for general peripherals, seven drivers for image processing and disc access peripherals, and another 10 drivers for expansion and security. With the number of required drivers easily reaching 50 for modern devices, it becomes clear that firmware, driver and operating system development and porting are a crucial development steps. Early availability of development targets in any form to enable driver development – virtual, hardware or hybrid - can significantly accelerate project schedules and improve product quality by enabling earlier software development at several layers of the software stack.

In the following sections we will introduce the example of a driver development for the Synopsys DesignWare USB OTG controller. We will show how the use of virtual platforms accelerated the overall project by eight weeks for a derivative IP product.

Universal Serial Bus (USB) Implementation Models

USB enables cost-effective "outside-the-box" connectivity with hot-swap capability for industrial, consumer, embedded, PC, and PC peripheral products. USB requires a host to manage and control bus traffic to and from USB peripherals such as hard drives, cameras, flash card readers, printers, and scanners. USB peripherals must manage data translation between the peripheral application and the USB Host. OTG products typically must handle both host and peripheral functions and switch between the roles.

When integrating USB logic into a design, users face a make or buy decision. They can develop the block on their own or license predefined, configurable implementations from an IP provider. Figure 7 illustrates the Synopsys DesignWare USB solution. Synopsys provides a complete, integrated, silicon-proven USB IP solution consisting of a suite of configurable digital controllers, mixed-signal PHY, and verification IP.

The DesignWare USB IP is certified compliant to the USB 1.1, 2.0 and On-The-Go (OTG) standards. Integration risk is lowered with a single vendor solution by testing to ensure that all the IP functions (controller, physical interface and driver) work seamlessly together.

Together with the USB IP itself, Synopsys provides a reference driver as well, enabling developers to design USB host, peripheral, and OTG products. The source code is available on a Linux/ARM9™ platform and is the same code used for all certification and interoperability testing.

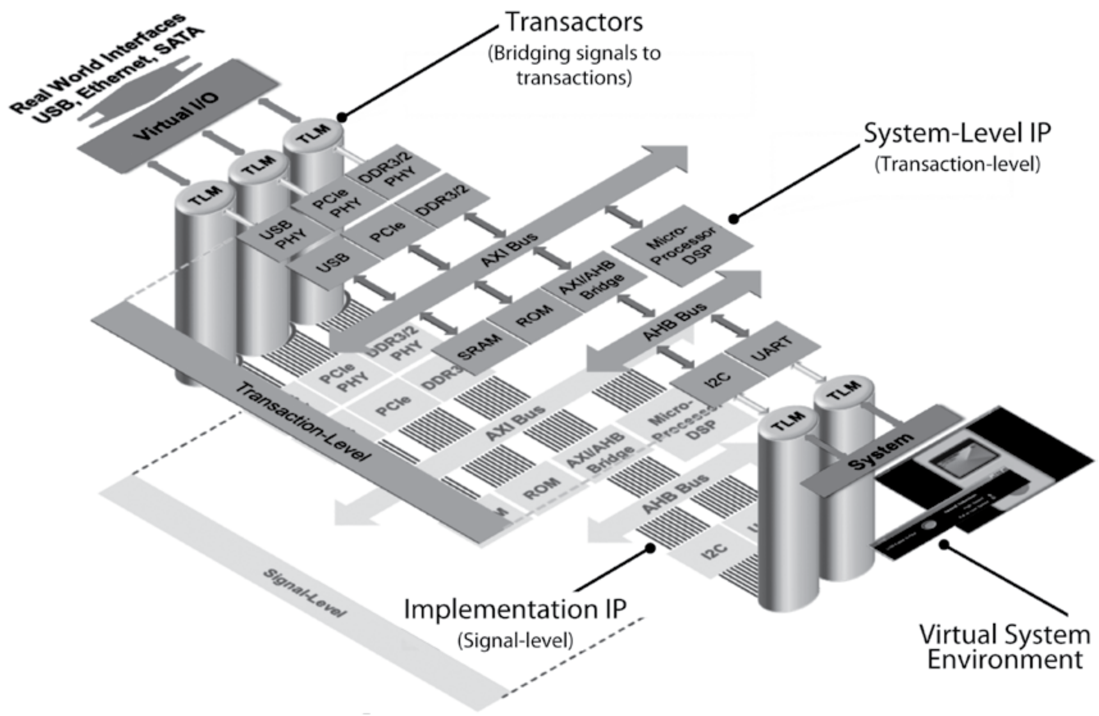


Figure 8: Transaction and implementation level models

Universal Serial Bus (USB) System Level Representation

Transaction-level models corresponding to the implementation IP are used to build virtual platforms. Figure 8 shows the representation of a chip design at both the signal and transaction-level. The design contains a microprocessor, an AXI bus, SRAM, ROM, a bridge from AXI to AHB and low speed peripherals for I2C and UARTs connected to the AHB bus. High speed peripherals for USB, PCIe and DDR 3/2 memory are connected to the high speed bus.

The design is shown at two different layers: the signal-level used for implementation and the transaction-level used for virtual platform execution for pre-RTL software development. The transaction-level design is in Figure 8 also connects to peripherals, the user interface and a general virtual representation of the end product shown as the “virtual skin” of a media player in Figure 8. High speed peripherals such as USB are also connected to real-world interfaces using virtual I/O, i.e., the USB port in the virtual platform can connect to the USB host on the machine on which the virtual platform executes. To interface between the transaction-level and signal-level, transactors translate between the two levels and are typically part of Verification IP (VIP).

For most of the models available from Synopsys as implementation IP, the DesignWare System-Level Library (DW SLL) provides register and functionally accurate, ready to use abstract representations. This specifically means that register definitions as indicated in Figure 5 will precisely represent at the system-level what is implemented in the actual core. In addition, the I/O interface and the underlying functionality of the core will be modeled accurately enough to allow execution as virtual platform. The USB 2.0 OTG and the USB Host Controller are directly available as SystemC TLM-2.0 compliant transaction-level models.

However, some users may need TLM models reflecting other USB implementations. For this case, the USB Model Authoring Library (MAL) included in the DW SLL enables a head-start for model developers, accelerating the development of the transaction-level model. Specifically, the Enhanced Host Controller Interface (EHCI) and Host Model Authoring Library provide generic USB functionality which can be extended by model developers to implement specific model characteristics like the actual register definitions. The libraries support C++ and SystemC model development and contain support for connecting to real-world devices and for test purposes, allowing virtual USB devices to be loaded into and exchange data with the host operating system.

Driver Development Challenges

Writing device drivers requires an in-depth understanding of how a given IP and platform functions, both at the hardware and the software level. Because many device drivers execute in kernel mode, software bugs often have much more damaging effects to the system. This is in contrast to most types of user-level software running under modern operating systems, which can be stopped without greatly affecting the rest of the system. Even drivers executing in user mode can crash a system if the device being controlled is erroneously programmed. These factors make it more difficult to diagnose problems, and the consequences of problems more impactful.

Figure 9 illustrates some of the instruments a virtual platform offers to ease the development, debug and verification of device drivers.

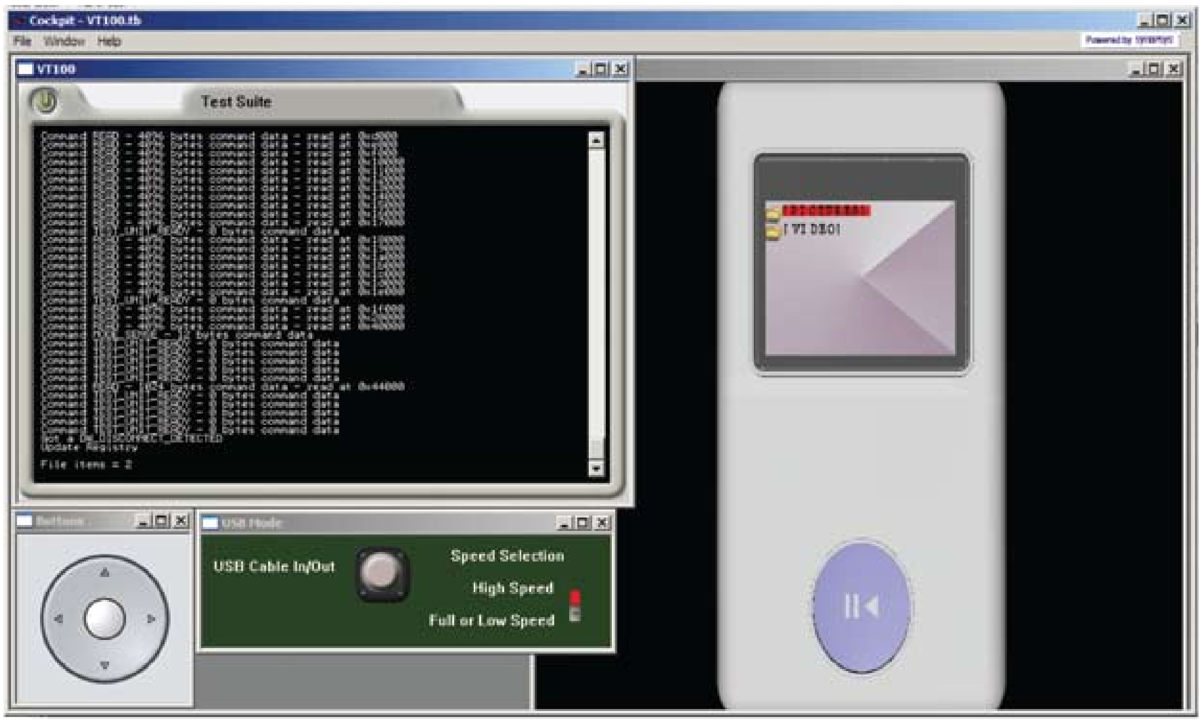
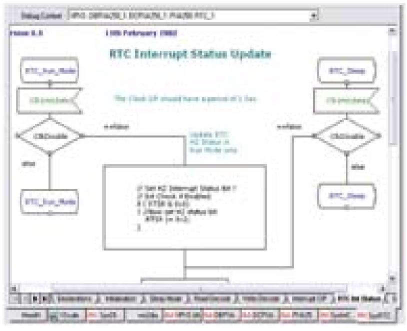
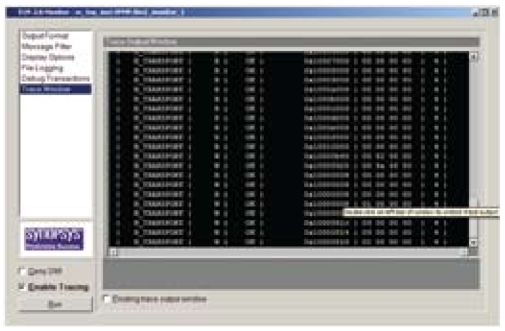


Figure 9a. Virtual product skin with test suite negotiating with the USB Host





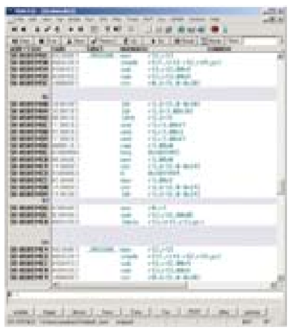


Figure 9b. Breakpoint in virtual platform

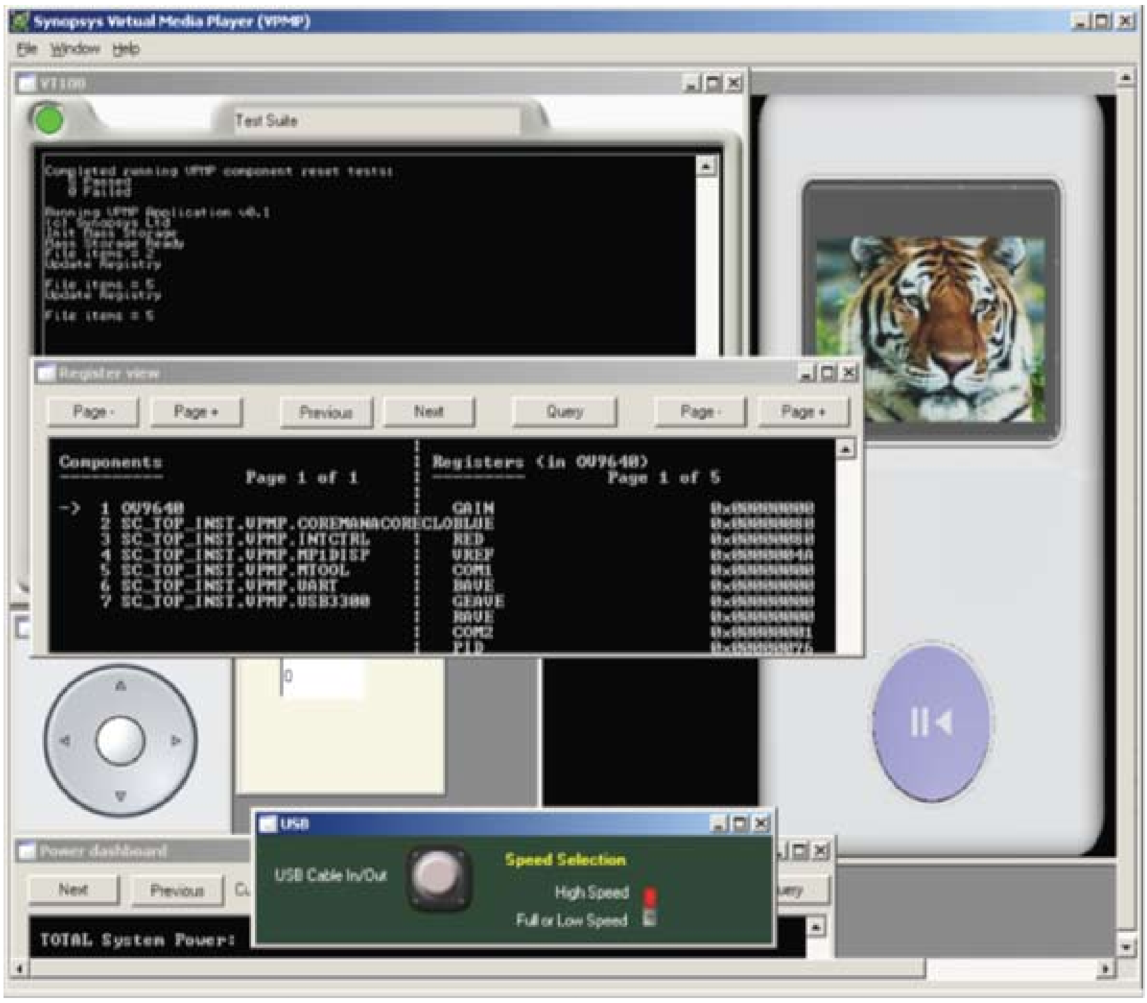
Figure 9c. SystemC TLM-2.0 transaction monitor

Figure 9d. Software debugger attached to virtual platform

Figure 9: Driver development using a virtual platform

Figure 9a shows the virtual device connected to the Host PC. The window labeled "Test Suite" shows the sequence of the device negotiating with the USB Host controller. Figure 9b shows a SystemC TLM-2.0 transaction monitor tracing the transactions actually passing into the USB OTG Controller in the virtual platform. Figure 9d shows a debugger attached to the ARM core in which the actual driver software can be traced. Stopping and starting the platform is possible from the debugger and the actual virtual platform GUI.

Breakpoints can be set in the software, the hardware models itself and the transaction interfaces of the USB OTG virtual device controller model as shows in Figure 9b. The amount of insight and control developers gain by using virtual platforms greatly increases software development productivity because traditional "trial and error" approaches are replaced with a predictable and repeatable software development process, including the ability to attach any software debugger of choice, keeping the environment familiar for software developers.



```
Completed running VPMP component reset tests:
    0 Passed
    0 Failed

Running VPMP Application v0.1
(c) Synopsys Ltd
Init Mass Storage
Mass Storage Ready
File items = 2
Update Registry

File items = 5
Update Registry

File items = 5
```

```
Components                         ! Registers (in OV9640)
-----------        Page 1 of 1     ! -----------        Page 1 of 5
                                   !
->  1 OV9640                       !   GAIN                    0x00000000
    2 SC_TOP_INST.VPMP.COREMANACORECLOBLUE                     0x00000080
    3 SC_TOP_INST.VPMP.INTCTRL     !   RED                     0x00000080
    4 SC_TOP_INST.VPMP.MP1DISP     !   VREF                    0x0000004A
    5 SC_TOP_INST.VPMP.MTOOL       !   COM1                    0x00000000
    6 SC_TOP_INST.VPMP.UART        !   BAVE                    0x00000000
    7 SC_TOP_INST.VPMP.USB3300     !   GEAVE                   0x00000000
                                   !   RAVE                    0x00000000
                                   !   COM2                    0x00000001
                                   !   PID                     0x00000076
```

```
TOTAL System Power:
```

Figure 10a. Media player application with Synopsys USB OTG Core

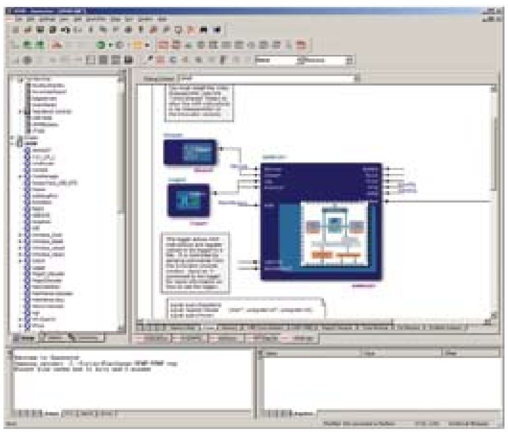


Figure 10b. Part of the SoC hardware architecture of the media player application

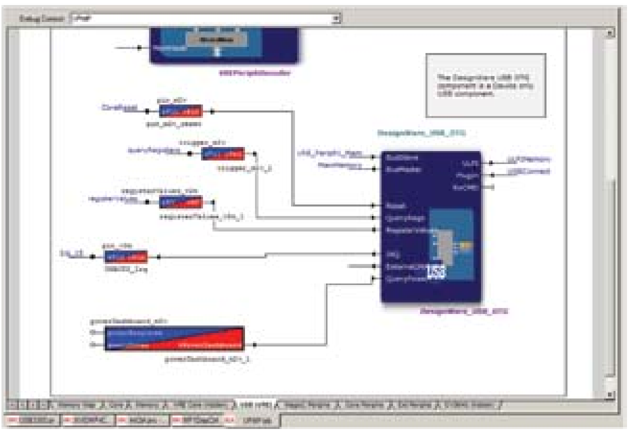


Figure 10c. USB OTG Core and its interface

Figure 10: Synopsys USB OTG model in the context of an ARM920T based SoC of a media player

Virtual Platform Use Models for USB Driver Development

Figure 10 shows the DesignWare® System-Level Library model of the USB OTG Device Controller in the context of an ARM920T-based SoC. This example SoC is part of a media player application capable of playing back still picture, video and audio. Figure 10c shows a representation of the USB OTG model with its interface, the middle right block diagram reveals some of the system context of the model. Figure 10a shows the actual executed media player as it is represented to the software developer with viewers for register content and power consumption. Software developers can attach their favorite software debugger to the virtual platform and access registers just like they would access them using the development board containing the actual chip.

There are two basic use models for virtual platforms supporting driver development in the context of a USB core.

First, a virtual platform can be used to model a device like a digital still camera, a USB video camera or a media player as shown in Figure 10a. Using Virtual I/O capabilities for software emulation of device I/O, the virtualized product can be connected to a Windows or Linux host on which the host controller resides and connects to the USB device. This process is illustrated in Figure 11.

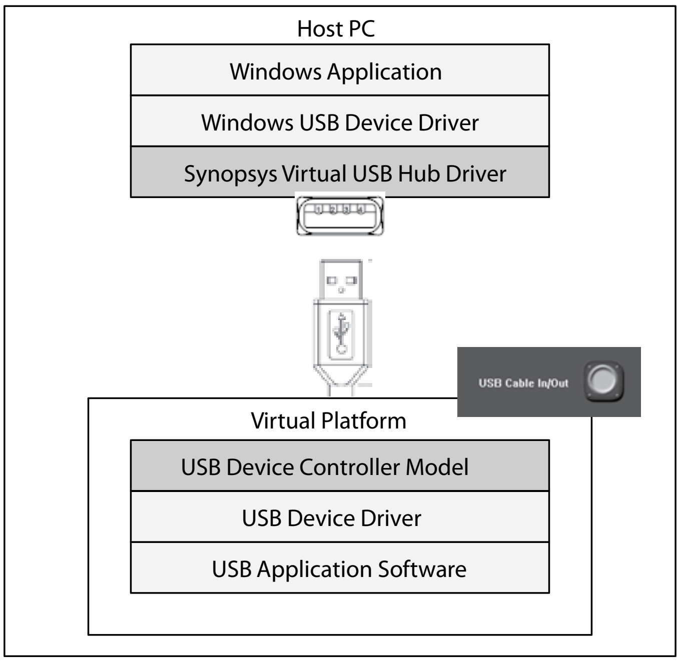


Figure 11: USB Device in virtual platform connected to real-world host

In this case the virtual platform simulates the USB device in question. The USB Device Controller model is part of the virtual platform and an ARM processor runs the device driver and application software within the virtual platform. A software debugger can be attached to the processor in the virtual platform (the ARM920T as shown in Figure 10b) and the function of the device driver can be developed and verified on the virtual platform long before RTL or silicon are available. In addition, the USB application software running on the device – a data transfer program for media files in the example of Figure 11 – can be developed, debugged and verified.

Pushing the button "USB Cable Input" connects the virtual platform to the USB Host Controller on the host on which the virtual platform executes. In effect, all the steps happening when the real device is connected will happen in this case as well, i.e., the Windows operating system will inform the user that a new hardware device has been detected – the hard disk of the media player. Together with Synopsys' virtual platform offerings, a virtual USB Hub Driver is provided. This Hub Driver interfaces to the Windows USB driver environment connecting the virtual hard disk in the media player to the Windows Application (in this case Windows Explorer) in which the virtual hard disk appears as a new disk to interact with.

Figure 12 illustrates another use model. This time the virtual platform simulates the USB Host environment and the Host PC is connected to a real-world USB device. The real-world device, a memory stick, is connected the Host PC using the Host Controller Driver and Hub Driver provided by the Windows operating system. Synopsys provides a USB Client Device Driver to connect this real-world device to the host executing in the virtual platform. This way it is easy to develop, debug and verify host controller drivers and applications for different host operating systems.

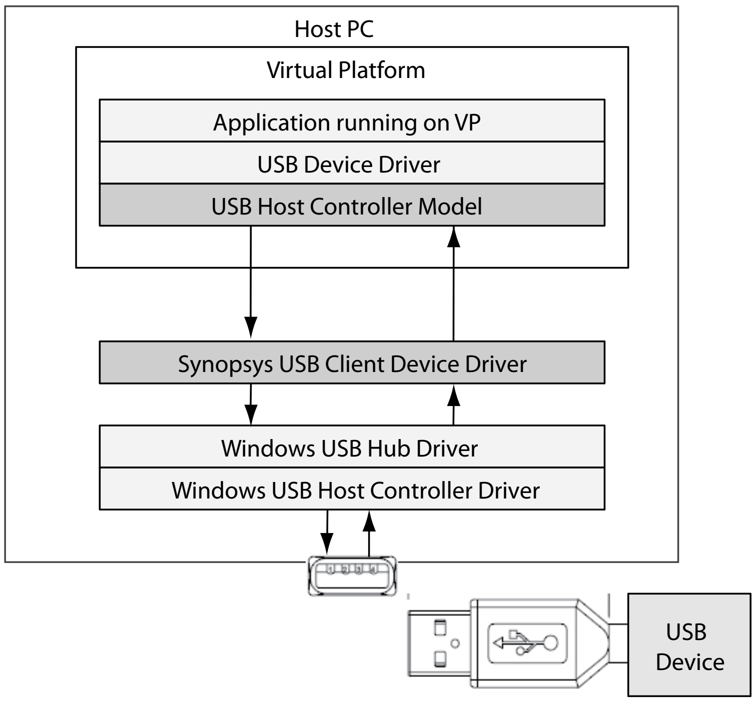

Host PC
Virtual Platform
Application running on VP
USB Device Driver
USB Host Controller Model
Synopsys USB Client Device Driver
Windows USB Hub Driver
Windows USB Host Controller Driver
USB Device

Figure 12: USB host in virtual platform connected to real-world device

USB Driver Development Case Study

After Synopsys' initial USB OTG implementation, hardware support for descriptor-based DMA transfers had to be added as a feature. The new feature was added for the device mode only, which required the Linux software device drivers to be extended to support it. A virtual platform was used for developing and testing the required driver updates.

The software engineering team extended the existing drivers based on published IP textual specifications. At the same time hardware support for the descriptor-based DMA enhancement in the DesignWare OTG IP core was added, the virtual platform modeling team enhanced the transaction-level model to support the new feature. The software engineering team completely debugged and tested the basic driver functionality early on the virtual platform. As a result the driver became available 4 weeks prior to FPGA prototype availability.

In addition to the early availability of the device driver, the actual software bring-up time was reduced by four weeks after the hardware prototype became available. The most critical bugs had already been removed during the development using the virtual platform, the hardware prototype bring-up focused on timing critical issues.

Note this case study outlined a relatively simple derivative update of an IP core; even more savings can be expected for developments starting from scratch or for more extensive IP updates.

Using Virtual Platforms for Functional Verification

Figure 13 shows the role virtual platforms play in the design flow. They are applied after the basic functional specification has been completed and are derived from the definition of the overall hardware/software system.

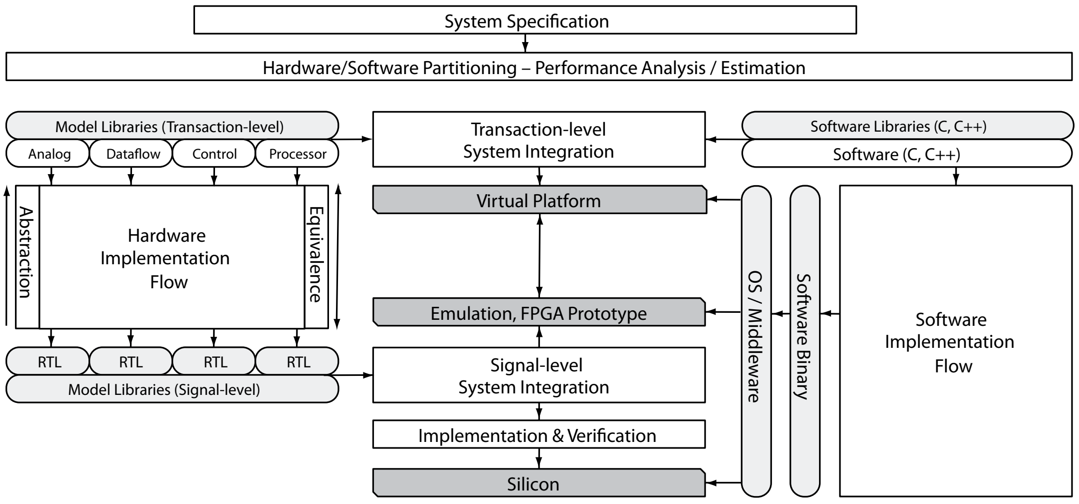

System Specification
Hardware/Software Partitioning – Performance Analysis / Estimation
Model Libraries (Transaction-level)
Analog
Dataflow
Control
Processor
Transaction-level System Integration
Software Libraries (C, C++)
Software (C, C++)
Abstraction
Hardware Implementation Flow
Equivalence
Virtual Platform
OS / Middleware
Software Binary
Software Implementation Flow
Emulation, FPGA Prototype
RTL
RTL
RTL
RTL
Model Libraries (Signal-level)
Signal-level System Integration
Implementation & Verification
Silicon

Figure 13: Virtual Platforms in the design flow

Virtual platforms are assembled from pre- transaction-level models, such as those provided in the DesignWare System-Level Library. They are then deployed to programmers for pre-RTL software development. In addition to the development of software, they can be used for functional, power and architecture analyses.

For functional verification, transaction-level virtual platforms can be connected to signal-level RTL simulation. Software executed on processor models can be used as the testbench for the actual hardware modules, enabling software driven verification. In addition, while RTL coding is still in progress, virtual models can be used to verify the test environment, into which the developed RTL is later inserted.

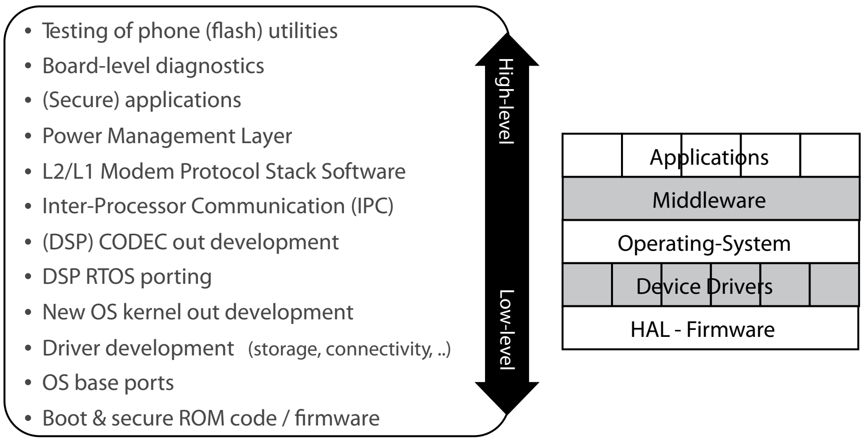

• Testing of phone (flash) utilities
• Board-level diagnostics
• (Secure) applications
• Power Management Layer
• L2/L1 Modem Protocol Stack Software
• Inter-Processor Communication (IPC)
• (DSP) CODEC out development
• DSP RTOS porting
• New OS kernel out development
• Driver development (storage, connectivity, ..)
• OS base ports
• Boot & secure ROM code / firmware
High-level
Low-level
Applications
Middleware
Operating-System
Device Drivers
HAL - Firmware

Figure 14: Virtual platform software development use cases

Summary

The case study for the Synopsys USB OTG driver development demonstrated an overall project savings was about 8 weeks, four weeks due to early start of software development and four weeks due to faster bring-up on the FPGA prototype. A savings of several staff-years of effort and months from the schedule can be realized if this is scaled to a more complex device like the i.MX31 (Figure 3) which has more than 40 drivers to be developed and tested.

In addition to software driver development, Figure 14 illustrates some of the other use models for which virtual platforms successfully have been used. For a layered software architecture as indicated on the right, starting from hardware abstraction layers through device drivers, OS and middleware all the way up to applications, virtual platforms have been used for all these software layers.

Customers like Marvell [5], Freescale [6] and Texas Instruments [7] publicly outlined their usage of Synopsys virtual platforms for all these layers, from ROM boot code and firmware through porting of operating systems and testing of flash utilities. Generally, more than 85% of the software was ported and ready when the first silicon prototype came back from fabrication, enabling single day integration.

Going forward, hybrid models of the different prototyping techniques – virtual and FPGA – are deemed to add additional value and even further increase software development productivity.

Sources

[1] Jobs: App Store could become a billion dollar marketplace, August 2008, http://arstechnica.com/news.ars/post/20080811-jobs-app-store-could-become-a-billion-dollar-marketplace.html

[2] Global System IC (ASSP/ASIC) Service Management Report, Software Trends, November 2006

[3] Graphic source: http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=i.MX31&nodeId=0162468rH311432973ZrDR

[4] "iPhone SDK Downloads Top 100,000", http://www.apple.com/pr/library/2008/03/12iphone.html

[5] "Synopsys Announces Virtual Platform for Marvell's PXA3xx Application Processors", http://synopsys.mediaroom.com/index.php?s=43&item=469

[6] "Virtio Delivers First Virtual Prototype for Freescale i.MX31 Platform", http://www.embedded-computing.com/news/db/?821

[7] 'Synopsys Delivers Software Developer Productivity Gains and Reduced Development Cycles", http://www.synopsys.com/products/sls/pdfs/vp_ti_ss.pdf

SYNOPSYS®
Predictable Success