

WHITE PAPER

Hardware Software Co-Design

Avinash Babu M
Mistral Solutions Pvt. Ltd



THIS DOCUMENT AND THE DATA DISCLOSED HEREIN IS PROPRIETARY AND IS NOT TO BE REPRODUCED, USED OR DISCLOSED IN WHOLE OR IN PART TO ANYONE WITHOUT WRITTEN AUTHORIZATION OF MISTRAL

Table of Contents

Abstract	3
Why Hardware-Software Co-Design?	4
Hardware Software Co-Design.....	6
Hardware versus Software Design	7
Hardware Software Co-Design Flow	9
Case Study 1 – Power Management.....	12
Case Study 2 – Partitioning	17
Case Study 3 – Design Correction.....	21
Advantages.....	24
Challenges	25
Conclusion	26
References	27

Abstract

Embedded product design is an experience of matching multiple perspectives. It is putting together many unilateral views of the system together to come up with a comprehensive view of the product. The design has to simultaneously address the hardware and software requirements at each juncture and how they constrain or contribute to each other to bring out the product features. Hardware and Software solutions with their advantages and disadvantages need to be used to piece in the solution architecture. Hardware and Software design is like two development paths that need to meet at certain places in order to assess risks, bring reliability, predictability and obtain product feature coverage. However hardware and software design life cycles are differently enabled in terms of cost of change in design, eco-system support, external dependencies etc.

In this paper we discuss aspects of Hardware and Software co-design with respect to embedded product design. We also discuss the need for co-design and the development flow that enables a designer to take advantage of hardware and software to optimize the solution offering. This paper builds some case studies that bring out various manifestations of Hardware and Software co-design in a product design. Finally we discuss the advantages, challenges and the real relevance of hardware and software co-design in the present context. We are not discussing the various co-design methodologies here; the focus is on how a product designer makes use of hardware software co-design to provide optimized design solutions.

This paper was presented by Avinash at the prestigious ARM TechCon™ 2012 held at Santa Clara, CA, USA.

About the Author

Avinash Babu is a Lead Architect with Mistral Solutions and has been instrumental in designing Consumer Electronics products like handheld devices, network video recorders, wireless cameras, rapid product prototyping modules and multimedia platforms based on ARM and PowerPC architecture. Avinash has an Engineering degree in Telecommunications from Bangalore University & an MBA from Indian Institute of Management, Bangalore.

Why Hardware-Software Co-Design?

As a designer, the paramount focus of a product design is to meet the system requirements in the most optimized fashion. By optimum we mean we have to find a solution point that needs to satisfy contrasting system, commercial and eco-system demands. A unilateral approach can match only one or a subset of these demands. The fact that hardware and software are differently abled allows us to put together a solution that takes advantage of both the worlds to meet the contrasting demands. Performance, reliability and configurability are the most basic system demands. Hardware design inherently can bring in reliable real-time performance while software designs bring in flexibility and configurability. In order to address all these requirements simultaneously, we need to put in hardware and software solutions for the appropriate sections of the system solution and stitch them together in the correct manner.

Time-to-market

Time-to-market is an important constraint that determines the choice of design approach. For example, most of the make or buy decisions are based on the time available for development, and budget outlay. Time constraint in a design cycle, includes certain 'fixed' artifacts and 'variable' artifacts. Fixed artifacts, which are generally commercially available off-the-shelf hardware modules can be drop-in solutions, hence reducing development and testing time and eventually time to market. 'Fixed' design artifacts come at a cost and what's more, they cost per instance of use and hence no means of amortization. This would not match up to the product price in the market. Hence we need to use them very prudently. Here is where the designer can make use of the 'variable' design artifacts which are generally self-developed/re-configured software modules. Of course, this comes with additional design, development and testing time. This also provides an opportunity to the designer to bring in value add. The value addition brought in by the engineer is in piecing together the hardware and software components prudently to deliver quality product features.

Product Cost

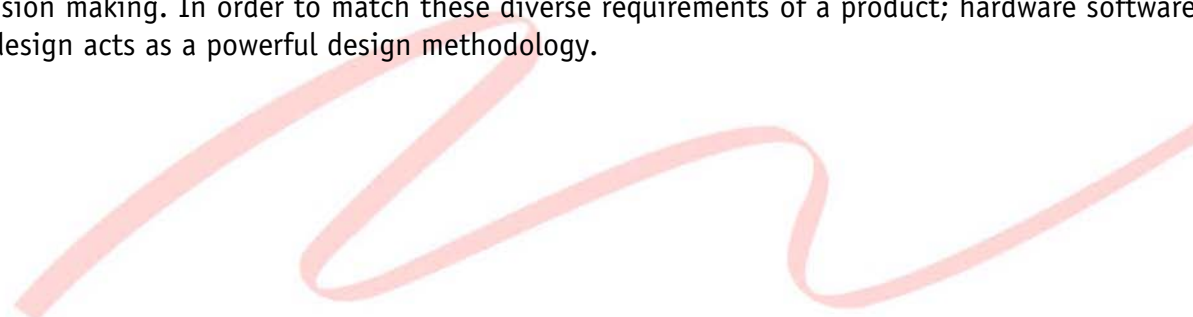
Every product has an optimum price for market acceptance which means the designer has a very important cost target. Everything in the product development should map to this target cost. The most important cost here is the recurring cost, which is mainly the product hardware cost; other costs get amortized over volume. For example, to reduce power consumption, we could move to a lower micron technology chip or use lower signaling level interface. These chips are by themselves costlier. This is where a designer should prudently use hardware software co-design to power on parts of the system relevant to a particular use case and save power.

Reuse of hardware or software modules dramatically reduces the time to market and cost of development. But every product in trying to create its own uniqueness creates a differentiation in offering. To match this product feature, designers need to augment existing reusable modules. This augmentation many a times requires hardware-software co-design. In putting together the solution, the current maturity of the implementation technologies is very critical.

Technologies that have matured over a period of time do not pose many challenges as support within the ecosystem is high. Newer technologies warrant putting together sub-optimal design modules and architecting a solution around them and thus, mandates substantial hardware software co-design.

Features and Applications

The diversity of present day applications also demands hardware software co-design. Nowadays, products require varying degree of programmability from application to the hardware. Most hardware components demonstrate ISA (Instruction Set Architecture), which is a programmer's view of the hardware. This provides the software a comprehensive control over the hardware. Also the applications warrant real-time data feedback and decision making. Hardware designs cater to real-time data feedback while software designs can handle complex decision making. In order to match these diverse requirements of a product; hardware software co-design acts as a powerful design methodology.



Hardware Software Co-Design

This design methodology exploits the synergism of hardware and software in the search for optimized solutions that use at best the current eco-system, the availability of hardware components and software infrastructure.

By nature hardware software co-design is concurrent; this allows system designers to optimize individual design approaches to accommodate and realign as required. Most importantly, this allows early detection of errors in system architecture and course correction. Often hardware and software modules are developed in a staggered fashion to enable or unblock each other. For example, to make sure that the software modules are ready by the time hardware is manufactured, software needs to be enabled with an early development platform. Today silicon vendors provide many reference designs that can be used for enabling early software development. Finally, hardware software co-design is an integrated design approach which means at every stage the design artifacts are vetted against the other approach to look for synergism, incompatibility and design outcome as against the optimized solution offering.

Any product development involves requirements, design implementation and validation. Requirements capture is done top down so that various facets of the requirements are captured as derived from the system requirements. Most importantly this exercise also helps determine the software and hardware component of a solution and brings clarity into hardware-software touch points. Choice of components is a very important phase in hardware software co-design. At each stage of the design the components need to be checked for compatibility with hardware and software. The choice of components is driven either by hardware or software design considerations. Cost, availability, design support and real estate are the criteria that drive the choice of hardware. Software criteria mainly include eco-system support, load on the processor etc.. When performance and reliability get higher weightage, hardware considerations drive the choice. Where flexibility, modularity and portability are important software considerations drive the choice. For example, in case of battery monitoring, we need to choose a fuel gauge with the right set of measuring capabilities and software needs to monitor this at right interval to implement a good battery management algorithm. The degree of decision making also decides the amount of hardware and software components. The design solution is partitioned in such a way that simple repetitive decision making is better handled in hardware, while complex decision making is better handles in software.

Validating hardware software co-design is slightly tricky as there is no clear distinction between hardware and software. This is where a system integration plan helps. Testing is broken down into unit tests, integration tests and system test cases. By progressively increasing the scope of design that is being tested, bug identification gets simplified.

The co-design approach varies depending on the application domain being catered to.

Demands of typical application domains are,

- Defense
 - Reliability – Hardware centric
 - System Interconnect – Blend of hardware and software
- CE
 - User experience – Software centric
 - Sensors integration – Blend of hardware and software

An important aspect of hardware software co-design is its ability to implement effective and responsive closed loop control systems. While hardware modules provide real time feedback, software allows system reconfiguration based on complex decisions. Programmable chipsets, sensor integration, re-configurable hardware blocks manifold the system's sensitivity and ability to adapt to changes.

Some of the enablers of hardware software co-design in the eco system are,

- Instruction Set Processors (ISPs) available as cores in many design kits (microprocessors, DSPs, micro-controllers, etc.)
- Increasing capacity of field programmable devices
- Architecture specific cross compilers for embedded processors
- Hardware synthesis capabilities
- Programmable chipsets
- Standard inter IC communication protocols like IIC, I2S etc.

Ultimately, hardware software co-design technique brings with it a sense of rationalization among various requirements. Even though it makes possible many design implementations with its ability to match performance with configurability, the multilateral approach to design will compromise on the performance of certain aspects but will cater to the most important needs of the product.

Hardware versus Software Design

Let's look at the natural tendencies of hardware and software design. This is important to understand as this will better enable the designer to service the system requirements. This will also help the designer understand modules where hardware and software are complimentary and modules where they are contradictory.

The cost implications of hardware and software designs differ; this is an important aspect that determines the use of hardware or software solutions.

- An NRE (Non-recurring engineering) charge is a one-time design or set up cost; this plus the unit cost to take the product to market determine the development costs. Software generally involves NRE charges and some licensing charges, if any while hardware involves both NRE and unit cost. The ratio of NRE to per unit software cost is

higher than the same hardware cost. Software costs are generally amortized over product volumes, but hardware unit costs are ever present. Each chipset has to be bought, and every hardware has to be manufactured there is little scope for amortization. Per unit cost of hardware is a function of the volume of production. It also involves committing on volumes to vendors which is a risk. Hence increase in software development costs are approved easier as compared to hardware design additions/changes.

- Tools used for software development are ubiquitous thanks to the open source movement, while the hardware development tools are proprietary. In fact, hardware development and simulation tools are an entry barrier to a hardware design and pushes designers towards more software based designs. The tools market in hardware is cartelized in the sense that there are a set of tools manufacturers for all the hardware design and simulations who command premium. This puts the designer at cost disadvantage.
- The cost of change is very prohibitive in hardware. This also means that hardware development is less iterative, being first time correct is very important in hardware. To ensure that the designs are first time correct it is important that a lot of simulation is done and simulation tools are costly.
- The nature of reuse varies between hardware and software. Software reuse costs are almost zero, in hardware, though the design costs can be zero; the per unit manufacturing costs can never be avoided; per unit cost of hardware is a function of the volume manufactured.

In general, hardware is always looked at per unit cost and software is looked at as costs that can be amortized over volume. This pushes the designer towards software based solutions. But the reliability and performance of hardware offsets this balance towards hardware. Software is generally designed to be flexible and modular which allows feature augmentation. The solution approach determines the amount of hardware and software components within the design. Products that are customized to a particular situation and mission critical generally have more hardware blocks than software. Hardware design once proven is reliable in performance. Product offerings that are more prone to product feature extension and market adaptations tend to have more software components.

Cost of design change is lower in software and hence allowable design iterations are high. This also leads to higher testing effort as the design is not optimized upfront. As software designs are more modular adaptation is possible till the very end of product launch and sometimes even after the product launch.

Product differentiation is a key factor for the product success. It is observed that this differentiation is manifested mostly by software modules. Hardware components are enablers within the system. Again if the differentiation is product performance it is the hardware module that is important, if the differentiation factor is adaptability, self-learning, programmability or ease of use software modules are important by virtue of modularity and portability. Hardware designs are better at performing repeated functions in a reliable manner,

while software designs are better in handling complex decision making. For example hardware is good at measuring the ambient light, while software is better at controlling the LCD brightness based on the inputs provided by hardware.

The Table 1 below summarizes natural tendencies of hardware and software design approaches.

Hardware Design Approach	Software Design Approach
Reliability	Programmability and adaptability
Performance	Decision making
Rigidity	Flexibility
Less iterative	More iterative
Highly cost sensitive	Cost implications not as pronounced
Customized	Standardized and Modular
Enabler	Manifestation

Table 1: Features that drive the hardware and software design approach

A designer has to make use of the natural tendencies of hardware and software designs to put together a solution, this at the crux of hardware software co-design.

Hardware Software Co-Design Flow

Let’s now discuss the normal execution plan of a hardware software co-design. Like any other product this involves requirements, design implementation and validation, but the kind of interaction between hardware design life cycle and software design life cycle makes sure that the implications of each on the other is addressed in a timely fashion.

Requirements Capture

During requirements capture, the most important product features are identified so that the solution can be optimized to those features. The industry segment also gives an indication of the implicit marquee product features. For example, products in the defense focus on reliability and performance, while products in consumer electronics focus on user interface, compatibility, battery life, size etc. This, in turn, decides the choice of hardware and software components. To start with, the system requirements are captured in totality, and then these are split into hardware, software, performance, user interface requirements to develop an overall perspective of the requirements. Most importantly, here the implication of system requirements onto hardware and software is captured by finding those parts of the model best implemented in hardware and those best implemented in software. In effect, the solution is partitioned into hardware and software. This is of critical importance because it has a first

order impact on the cost/performance characteristics of the final design. The formulation of the hardware/software partitioning problem differs according to the co-design problem being confronted with. In the case of embedded products, a hardware/software partition represents a physical partition of system functionality into application-specific hardware and software executing on one (or more) processor(s). Various formulations to this partitioning problem can be compared on the basis of the architectural assumptions, partitioning goals and solution strategy. While partitioning the requirements, it is very important to look at them with respect to pay load and control types. This in a way drives most of the partitioning.

Design Implementation

To start with, the implementation phase involves finalizing the solution approach. Here the hardware and software teams sit together to finalize the processor, operating system, programmable chipsets that will be used in the design. While hardware looks at availability, power, real estate and support; software designers look at programmability and eco-system maturity before finalizing the design components. This is an iterative process wherein the solution is vetted against some critical factors like cost-performance trade off, time-to-market etc. Software is dependent on hardware for development and testing and to meet the time-to-market constraints, the software should be ready by the time hardware is manufactured. This warrants an early software development platform, which is a quick and dirty solution that provides software developers access to hardware that will be used in the final design. This dramatically reduces testing time when the final custom hardware modules are ready to be tested. Detailed discussions on the memory map, interrupts, general purpose I/O etc. also bring about a lot of synergy between hardware and software. Important components like the size and speed of memory required for delivering the system performance is finalized upfront. Choice of software modules requires benchmarking performance; this is done using reference designs or early software development platforms. Matching hardware design and software design flow is imperative to make the design progress in tandem. Most important outcome of this is the course correction that is possible earlier in the design flow. Thus, the design implementation in hardware software co-design is complementary in nature and most importantly an early self-correcting mechanism.

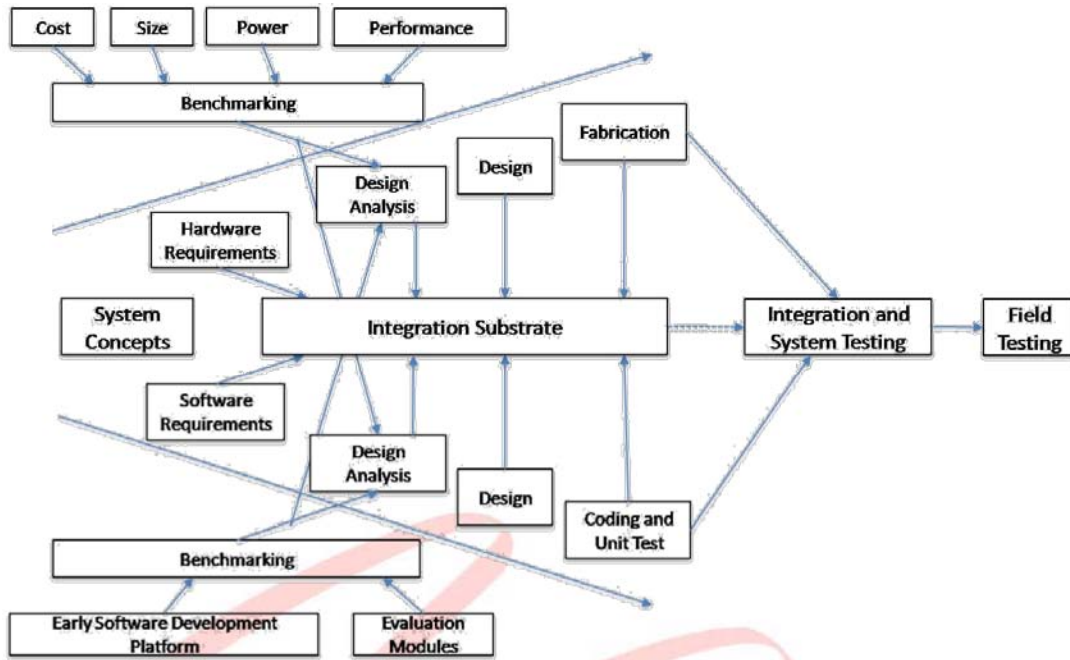


Figure 1: Features Hardware-Software Co-Design Flow

Figure 1 tries to capture the co-design flow. The most important part here is the integration substrate which manifests like memory maps, interrupt mapping, I/O pins mapping, boot strap settings etc. Independent bench marking paths on software and hardware feedback into the design corrections, allowing course correction.

System Validation

Finally the design paths integrate for system validation. Validation in any case aims at increasing the test coverage of the product. In case of co-design, the most important factor is the interdependency of the design which does not allow designers to independently test hardware and software modules. Here the designer needs to intelligently partition the testing in such a manner that deadlocks can be resolved. This is done by layering the test cases and testing features in progressively, increasing order of coverage. This helps in early bug finding and reduces testing time. Hardware software co-design, demands co-testing as well.

Case Study 1 – Power Management

Power management can be implemented at different levels, from energy efficient peripherals and adaptive digital systems to power aware software programs. Traditionally, power management has been about switching on/off devices as required for a particular use case. Recent developments in technologies have enabled chips with dynamic voltage and frequency scaling capabilities; this is generally seen in microprocessors. Unfortunately, main power hogs within the system need not be the microprocessor but the peripherals, which is where the choice of hardware and software design plays an important role. We need to choose the right hardware components that can be put to power down modes and enable software to put hardware modules into different states of activity based on use case requirements for effective power management.


 Level of Abstraction	Transistor Level	Dynamic Voltage Management, Dynamic Frequency Management
	Architecture Level	Device Drivers, Power aware memory and Bus Protocols
	System Level	Power Management at Operating System Level
	Application Level	Power aware algorithms implemented in applications such as multimedia

Figure 2: Levels Of Power Management Implementation

The Figure 2 above gives an idea of the various levels at which power management can be implemented. Architecture level onwards each approach involves hardware software co-design.

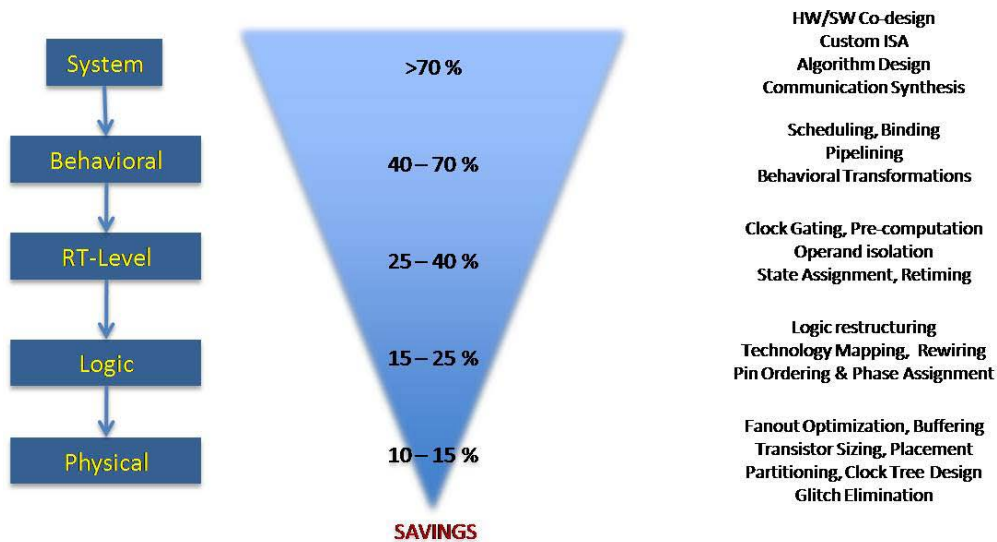


Figure 3: Power Savings At Different Levels Of Implementation

As shown in the figure above (Figure 3), statistically the amount of power savings at silicon level is very low compared to power savings that can be achieved at system level architecture using hardware and software co-design.

Choosing hardware with the desired capabilities is the starting point for power optimization using hardware software co-design. Developing power profiles is the next important step; this provides power consumption of the various sections of the design. Power analysis techniques are available for embedded software based on instruction-level characterization. Techniques to improve the efficiency of software power analysis through statistical profiling are also available. These methods allow the designer to forecast power consumption and take intelligent decisions to save power. It is very important to co-relate this information with the use case scenarios.

Table 2 below shows such a power chart. Power consumption is shown in milli watts and battery life is shown in hours.

USE CASE	Processor	DDR	uSD2	WLAN	Bluetooth	Amplifier	Camera	LCD	GSM	HOST USB	USB OTG	Vibrator	Watts	Battery Life (Hrs)
Processor + DDR + uSD2	300	296	106										722	10.04
CAMERA + Processor + DDR + LCD + uSD2	570	296	117				313	3066					4382	1.65
Processor + DDR + LCD + AMP + uSD2	425	296	117			10543		3066					14467	0.50
Processor + DDR + BT + LCD + Sensors + uSD2	500	296	117		1165			3066				100	5264	1.38
Processor + DDR + BT + LCD + AMP + Sensors + uSD2	500	296	117			10543		3066				100	14642	0.50
Processor + DDR + LCD + GSM + uSD2	500	296	117					3066	6105				10104	0.72
Processor + DDR + LCD + WLAN + uSD2	500	296	117	1165				3066					5164	1.40
DATA TRANSFER WITH USB HOST	500	296	117					3066		3223			7222	1.00
DATA TRANSFER WITH USB OTG	500	296	117					3066			2941		6940	1.04

Table 2: Power Consumption Chart for Various Use Cases

By looking at the chart we can observe that the processor is not the main power hog. It is the peripherals functional in a particular use case. Power analysis defines constraints on hardware and software blocks in the design. The system design is completed taking these constraints into consideration. To enable power management in a product the hardware and software contribute differently but complement each other.

- Hardware hooks
 - Chips – Chipsets provide different operational modes based on activity
 - Electrical design – Hardware design in itself can provide options for power on and powering off parts of the design based on the product use cases
- Software hooks
 - BSP and Drivers – Compliment the hardware design in enabling and disabling sections of design and controlling power states within devices
 - Application framework–Integrate with the drivers to manage use case power management.

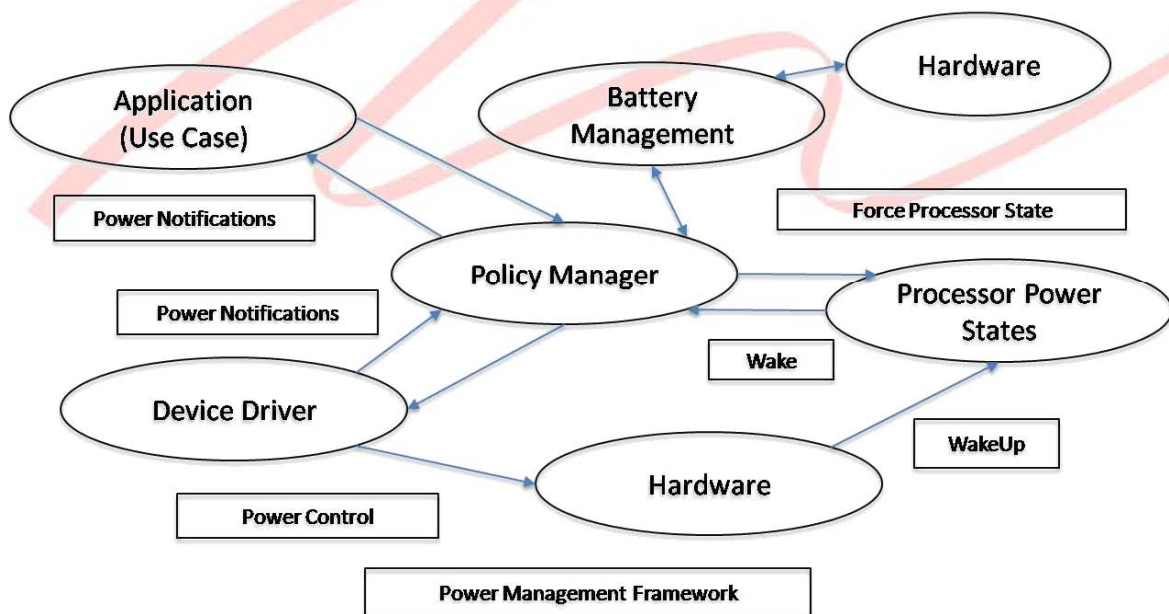


Figure 4: Power Savings At Different Levels Of Implementation

Figure 4 above gives the scheme for a typical power management implementation using hardware software co-design in a battery operated product. At the center of this is the policy manager which orchestrates power usage by monitoring the battery statistics, processor states and application demands. The effectiveness of this implementation is heavily dependent on how the hardware and software modules enable power management. This is where hardware software co-design increases the effectiveness manifold.

For example, battery management is done using battery charger chip and fuel gauge chip in hardware that performs power path management and provides battery statistics. Software needs to consistently monitor the battery statistics and help policy manager take decisions on

power budget allocation to requests from applications. This is how once the battery levels go below certain levels in smartphones; some of the services are disabled. Also, as soon as you try to play a video you get a low battery indicator signal. The application sends a power notification request to the policy manager; based on earlier profiling the policy manager knows the power budget; in correlation with the battery statistics the policy manager either services the request or rejects it. If the notification request is serviced the corresponding device drivers enable the low lying hardware units. And when the media player application is closed the power request notification is withdrawn and the corresponding hardware is turned off. Instruction amplifiers are used to profile a product's power consumption for varying use cases. This can be used to optimize and calibrate the policy manager algorithms.

On inactivity of the device, the policy manager automatically puts the device into power saving mode. Configuring the policy manager to enable features like dynamic voltage and frequency scaling, load balancing etc. for power management is a kernel driver setting. These are known as governors that configure the processor to different modes of operation like on-demand, performance, conservative etc. These are preconfigured power schemes of the CPU. Processors off late come with various power modes like wake, sleep and deep sleep. The policy manager puts the processor progressively in deeper power saving modes based on the length of inactivity. The various power saving modes switch off parts of the hardware chip depending on the degree of power saving. The wake up from these modes is triggered by hardware events like a key press, radio activity, touches screen input etc. On wake up progressively switching on the device and restoring device to the earlier state is done by the software infrastructure.

LCD Displays are one of the highest power consumers in a product. The TFT LCDs are lighted from backlight LEDs, which generally require a boosted voltage which in itself is a reason for inefficiency. To dynamically control the brightness of these LEDs hardware drivers controlled by pulse width modulators are available. The duty cycle of these PWMs are controlled by software. When a user uses a scroll bar on the screen to set the brightness level the corresponding driver controls the duty cycle of the PWM and in effect the brightness of the LEDs. If this is pushed further, ambient light sensor chipset provide real time feedback on the ambient light, based on this the software infrastructure controls the duty cycle on the PWM and hence brightness of the LCD. This demonstrates a tightly coupled control system with real time feedback used for power saving.

Similarly radios and video decoder chipsets are another source of power drain. These chipsets provide power controls through a power down option, reset option or control the supply voltage. Software needs to either enable power saving state machines internal to the chipsets or control power modes of the chipsets based on activity. Such hardware software design implementations are based on use cases of the product. Consistency is built across layers from application to the hardware using a policy manager and hardware hooks to enable these chipsets only when required.

Figure 5 below gives a representation of dynamic power switching and possible power savings.

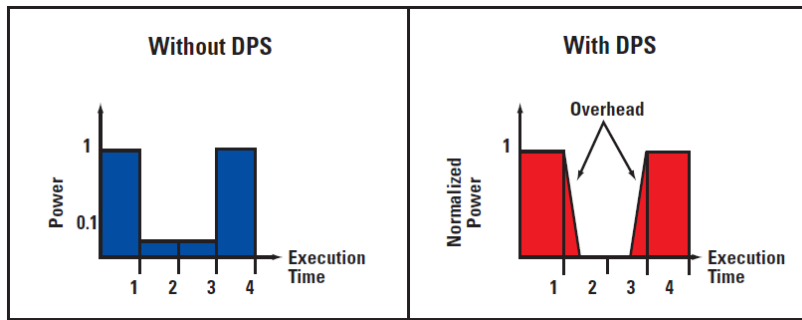


Figure 5: Dynamic Power Switching And Possible Power Savings

Figure 6 below shows how various power management techniques like AVS (Automatic Voltage Scaling), DVFS (Dynamic Voltage Frequency Scaling) and DPS (Dynamic Power Switching) can be used to optimize power utilization in a device like a multimedia player.

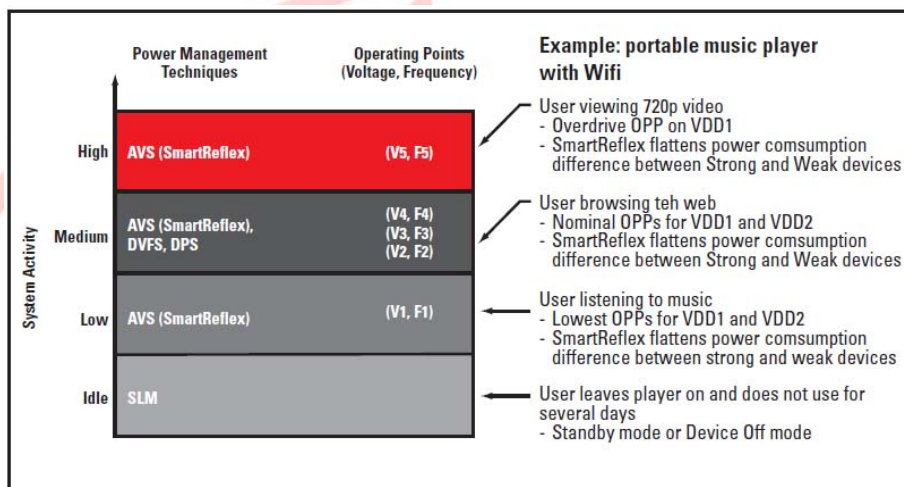


Figure 6: Power Management Techniques For Optimizing Power Usage

As shown in the figure; when system activity on a portable multimedia player is high such as when viewing high-resolution video an overdrive OPP (Optimal Performance Point) can be set on the voltage supply to the core. For web browsing that requires medium power consumption, nominal OPPs can be set for operational voltages. For listening to music, which has relatively low-power demands, the lowest OPPs can be set for operational voltages. In all three of these examples, you can activate AVS to flatten power consumption differences between “hot” and “cold” devices with varying power consumption characteristics due to manufacturing differences. When the user leaves the media player on but does not use it for several hours or days, static power management can automatically drop into device-off mode.

Hence, it is seen that using hardware software co-design the power management of a product can be effectively handled. This design approach brings in the flexibility to optimize power usage based on activity. When consistency is achieved across the product design; from application to hardware; the number of options and power management features of a product manifold. Effective use of this infrastructure is a very powerful tool for power management.

Case Study 2 – Partitioning

In this case study, we will try to illustrate the aspects of partitioning in hardware software co-design using a product design. Radar Scan Converter is a dedicated system used to transform the Radar inputs into a more decipherable and easily conceivable format. In order to achieve this, the radar video signal with the azimuth has to be acquired and processed at real-time. The rate of data acquisition at the front end is constant but the amount of data transfer between and processing that happens in these data processing engines varies. The Radar Scan Converter has to cater to these varying needs. This is done by sharing the tasks effectively between the software running on an embedded processor and dedicated hardware.

The requirements of the product can be broadly stated as:

- Front end analog video acquisition
- Data load balancing without losing information
- Data correlation between incoming data and strobes
- Effective data transfer between the various algorithms in real-time
- Distribution of the various algorithms between the data processing engines to handle data processing
- Adaptation to varying data processing rates
- Mechanism to map the incoming data to display type
- Control from a Host

The solution approach was arrived at by dividing the product implementation into four processes and observing the data flow and control flow as shown below. This also can be called modeling the product to explore options of hardware and software co-design. This will provide us information on the degree of data transfer, type of controls and helps in finalizing the driving engines for the product. Here we realized that the driving engines need to be of two types; i) which needs to be a data processing engine and ii) a data interpretation and representation engine. The data processing engine was realized using a Field Programmable Gate Array plus discrete electrical design and the data interpretation and representation was done using RTOS (Real Time Operating System) running application software on an embedded processor.

The diagrams below provide the functional representation, data flow and control flow of the product. This provides detailed information on the kind of data transfer and control transfer anticipated in the design.

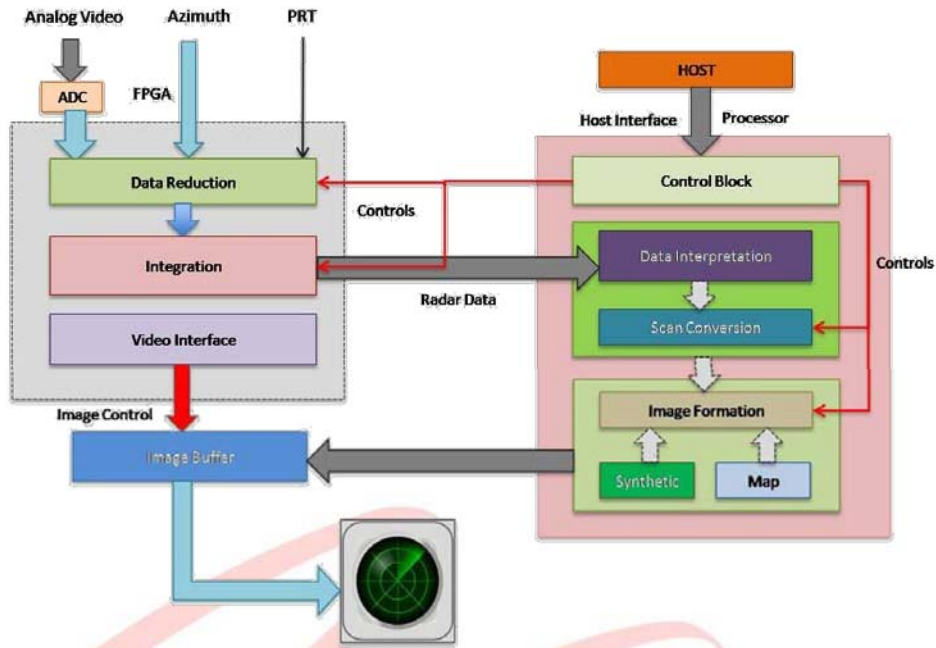


Figure 7: Functional Representation Of The Radar Scan Converter – Data And Control Flow

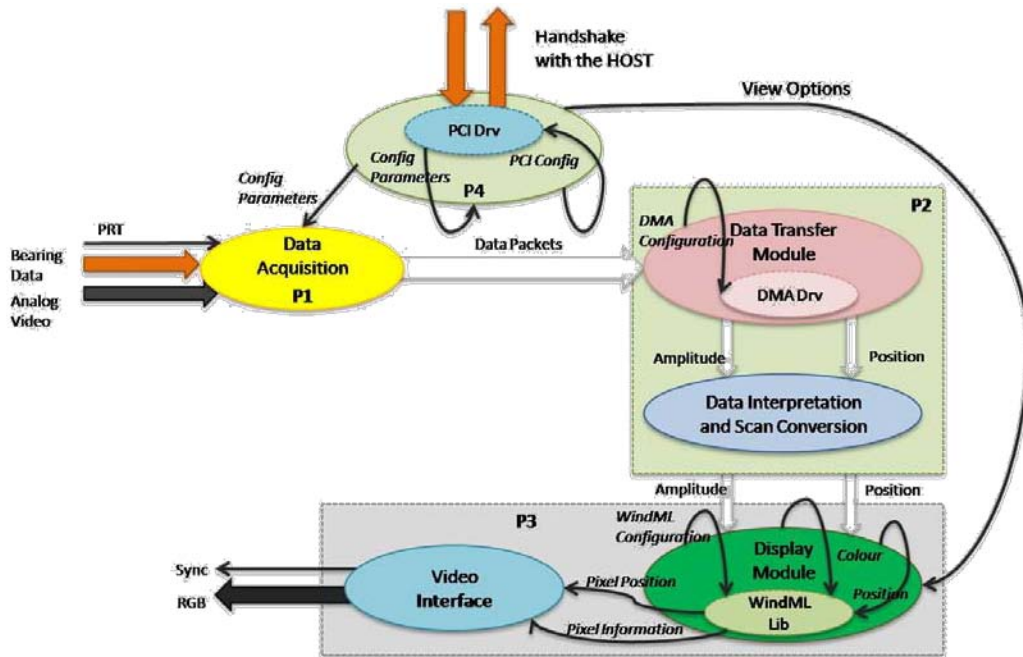


Figure 8: Data Flow Diagram

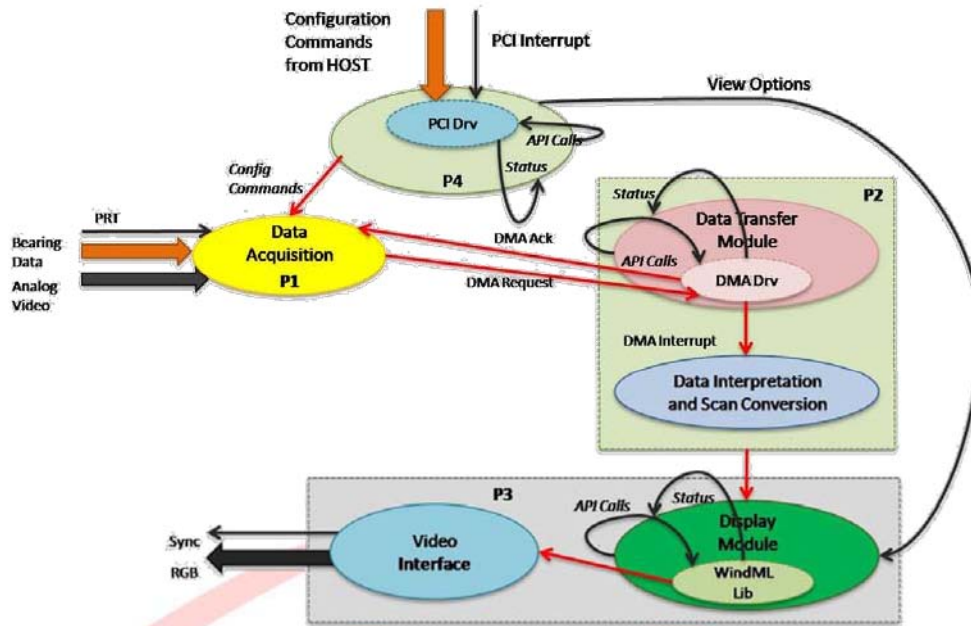


Figure 9: Data Flow Diagram

Looking at the data flow and control flow the system can be broken down onto four processes

- Data Acquisition Block
 - This block is responsible for acquiring the Radar Data and transferring data to software at a constant rate involves data acquisition and data reduction mechanisms.
 - Data acquisition is done real time using an ADC and FPGA logic to capture data at the required rate. Hardware algorithms within the FPGA dynamically handle data speed conversion.
 - Real time constraints on this block are really high and hence most of the design is realized in hardware.
 - Analog data is digitized by the ADC and buffered inside the FPGA. On request from the FPGA, software running on the processor reads the data from FPGA
- Data Processing Block
 - This block is responsible for processing the Radar data.
 - Data processing involves different types of algorithms. Algorithms like Constant False Alarm Rate elimination algorithm, Data reduction and Integration which involve repetitive functions are implemented in hardware logic blocks within the FPGA. Also data processing that require simultaneous multiple path approaches are implemented in hardware.

- Other algorithms like scan conversion and image blending requiring complex mathematical operations are implemented in software running on the embedded processor.
- Display Block
 - This block is responsible for displaying output on standard display.
 - Image formation and updating is done by software on the processor. Image formation involves preparing the image buffer, which in turn requires co-ordinate fixing and colour palette conversion. Graphics libraries within software provide a lot of flexibility in doing this. The second part of display is the rendering part which needs to be done in a timely fashion to match the refresh rate of the display. This is best implemented in hardware. FPGA is responsible for refreshing the VGA at the required rate.
- Control Block
 - This block is responsible for customizing the product online depending on the user inputs and the change in the external strobes.
 - The host control is achieved through a PCI interface, hardware and software help map configuration register into the PCI space.
 - The reconfiguration itself is done mostly by the hardware. Software provides an infrastructure to recognize the command and program the required hardware blocks.
 - Certain parts of the implementation require dynamic adaptation. Change in external strobes is recognized by hardware blocks, while software re-configures the right block based on the change.

This case study brings out the vagaries of hardware software co-design, with respect to how the type of operation determines the implementation approach and how the hardware and software implementations can complement each other to develop, complex and real time design protocols.

Case Study 3 – Design Correction

Here we discuss a case of video input solution wherein we explored multiple solution approaches and how hardware software co-design helped us find a solution. The product required a dual stream SDI output channel. The system was a video processing engine where in high definition and standard definition video from various input sources were to be encoded, streamed out or played locally. The processor inputs were always BT656 / BT1120 streams, FPGA and video decoder chips were used to convert the various sources to video input streams of BT656 / BT1120 streams. Then the processor would either stream the encoded video or playback locally. The local playback as SDI streams were implemented using the FPGA and the processor video output ports. The SDI core implementation was in the FPGA, the cable equalization was implemented as electrical blocks while the input data was to be streamed from a processor through video ports. The issue here was that the processor did not have enough video output ports to stream two videos required for a dual stream SDI output. The design allowed only one video channel stream from the processor to FPGA. This meant that we could not meet the requirement for dual stream video output. The block diagram below gives a snapshot of the system.

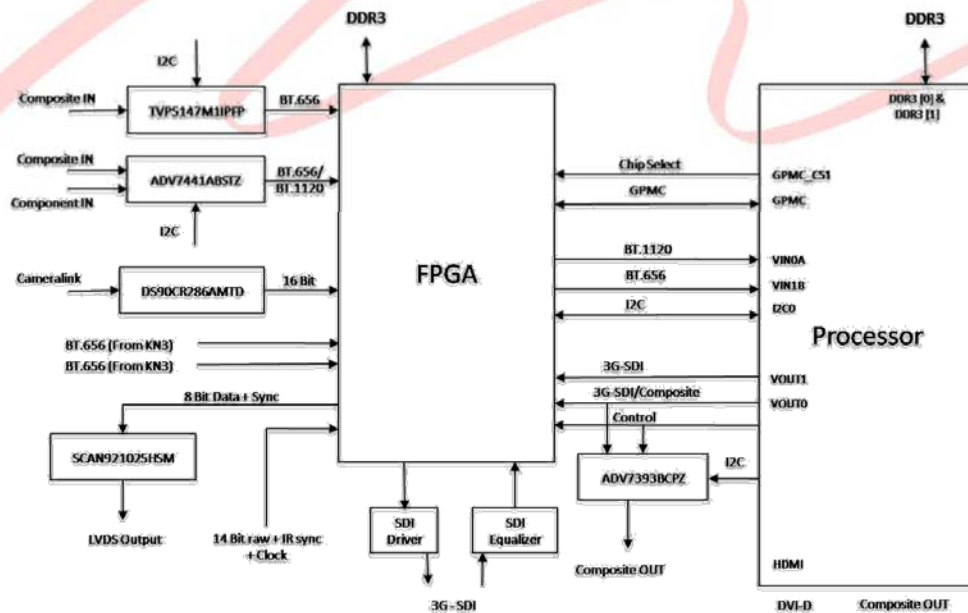


Figure 10: Block Diagram Of The Video Processing Engine

To address this problem, firstly we decided to route one of the video channels coming to the FPGA directly as the second input to the SDI core but the requirement was that the stream had to be processed in the processor before being played back; this meant that we had to make way for the second channel video streaming from the processor.

Then we started exploring options where we could use other video output ports of the processor, encode the output into BT656 / BT1120 formats using encoder chipsets and feed the output to FPGA as the second video input for the dual SDI stream IP. Multiple design

options were possible in this case because of the flexibility brought in by the software infrastructure in the processor, encoder chipset options and the FPGA. Figures 11, 12 and 13 illustrate the solution approaches,

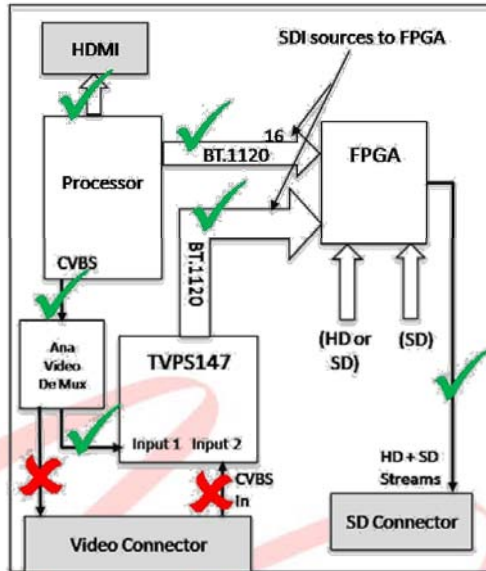


Figure 11: Configuration For Supplying BT.1120 + BT.656 Inputs To FPGA FOR DS-SDI Using TVP5147

Composite output from the processor passes through a video de-mux to be connected to TVP5147 video decoder second input. Output of the decoder is fed as the second input to the SDI core inside the FPGA. The block diagram also shows the product feature compromise in case this approach was taken.

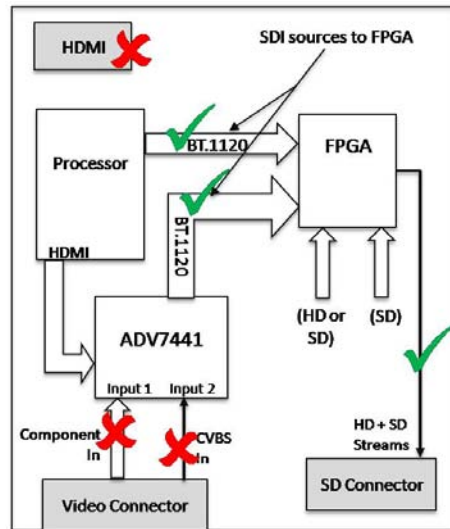


Figure 12: Configuration For Supplying BT.1120 + BT.1120 Inputs To FPGA For DS-SDI USING ADV7441

HDMI output from the processor routed to one of the ADV7441 decoder input. Depending on the output configuration, one of HDMI / Component / Composite input is selected in ADV7441 (In Dual stream SDI configuration – HDMI input will be selected). The block diagram also shows the product feature compromise in case this approach was taken.

As both the options above come with a reduction in product feature, these could not be pursued further. The final solution involved reconfiguring the pin mux allocations in the processor to enable another video port in the processor. An existing BT.1120 input port (Vin1A) to the processor was configured as BT.1120 output (Vout1) and Vin1B port was configured as BT.656 input. The video port was not with discrete sync signals, we could only manage a video port that could support embedded sync. The FPGA was used to embed the sync information and transfer video data to the processor. This scheme is depicted in Figure 12.

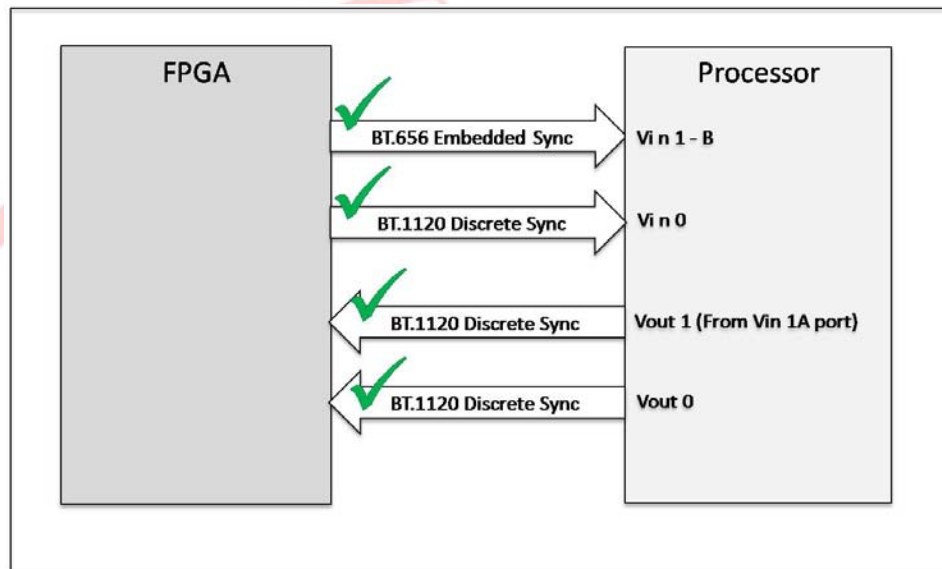


Figure 13: Reconfiguring The Pin Mux Allocations

By doing this exercise we realized that a hardware software co-design can enable a lot of options to resolve a design constraint. It was amazing to realize the many ways a design solution can be approached and the implications that each design option throws in terms of the effort, time, cost and performance optimization. The design was mature when this issue cropped up, this made the implications that much higher. We realized the importance of identifying design issues upfront. This is one case where hardware software co-design enabled us to find a solution even at a very advanced design stage.

Advantages

- Brings in a comprehensive and concurrent approach to product design with higher possibilities of meeting all perspective requirements. Choosing the apt design modules, taking advantage of the natural inclinations of a hardware or software design module helps reach close to the global performance optima.
- By taking the correct make, buy and reuse decisions in piecing together the solution we can effectively meet the time to market constraints.
- System feedback and adaptability that can be achieved using hardware software co-design makes the product design a self-correcting mechanism. This makes the product more pertinent to the use case. Seamless integration and course correction effectively reduces design iterations.
- Combining hardware and software features provides the possibility of solution enhancement of a product.
- Co-design enables ironing out software and hardware incompatibilities and brings seamless hardware and software interoperability, which is one of the most troubling aspect of product designs.
- Earlier interaction between software and hardware provides insight into possible conflicts and allows chances for earlier course correction. Also the design options brought in by combining hardware and software design capabilities allows corrections even at mature stages of design.
- Hardware and software complement each other to support growing complexity of embedded systems.
- This helps in putting together sub-optimal modules together to realize desired system functionality.
- Tends to create a more balanced distribution of system load across the hardware and software, increasing design reliability.

Challenges

- Hardware/Software partitioning is a complex task as this involves balancing many factors like,
 - Architectural assumptions regarding type of processor, interface style between hardware and software etc.
 - Partitioning objectives like maximize speedup, latency requirements; minimize size, cost, etc.

There is no standard partitioning strategy, these vary from high-level partitioning by hand, automated partitioning using various techniques, etc. The outcome of partitioning is also not standard. It is more a skill than engineering at this point in time. Early binding is widely used in industry because it supports complete planning of the design cycle. With this method, hardware/software partitioning decisions have to be made very early in the design. Late binding because of its flexibility, provides greater opportunity for performing hardware/software tradeoffs. Therefore, late binding generally results in a more optimal partition. But each one comes with its own set of problems; while one needs early commitment and lack of optimality, the other runs the risk of lack of planning and execution hassles.

- Scheduling the design execution and matching the hardware and software data flow and control flow.
 - Operation scheduling in hardware – This mainly involves matching the hardware development with the software development and enabling or unblocking software development.
 - Process scheduling in software – This mainly involves data path management and control path management.
- Modeling the hardware/software system during the design process is still a non-standard process. We need an integrated modeling substrate for incremental reviews throughout the design process. The industry lacks a common model or standard for unified exchangeable design representations which would greatly enhance usage of co-design. Very few comprehensive tools are available for hardware/software cross specification, development, simulation, integration, and test. However, efforts are underway to provide them.
- Complexity of handling multiple development paths those need to interact at the right points and finally integrate. The hardware and software models have their own development cycle characteristics, these need to be integrated during development. Also, we need to make sure that hardware does not block software development. In addition, feedback from hardware and software designs should flow into each other for course correction.

- The whole process involves lot of interaction, which also means interruptions in development. The development cycle needs to accommodate re-design and re-planning, this dynamism needs to be managed keeping time to market in place.
- Managing dependencies is more complex, given that the hardware and software development life cycles are tightly coupled adverse effect of delay / dependency is that much profound.

Conclusion

Hardware software co-design seems imperative given the present day design challenges of time to market, cost and ever increasing product features and design complexity. As a designer this is a very powerful tool to meet system demands in an optimized manner. The demarcation between hardware and software is dimming by the day. Cross-fertilization between hardware and software design methodologies has increased pertinence of co-design.

Co-design allows a methodology to view the product from varying perspectives and optimize the product performance. It involves matching different design cycles which involves high degree of interaction at apt points in the product development. It allows the designer good design flexibility and course correction. Sometimes it also allows design rectification at advanced stages. It is the skill of the designer to mix and match the different components of the design effectively. In fact this is the challenge and in a way the most satisfying part of product design.

Hardware is always looked as cost due to implications of per unit cost and cost of change but hardware brings reliability into design. Software brings in modularity, enhances product feature set and is portable. Ultimately the system requirements and solution approach determines the mix of hardware and software components within the design.

We have seen how co-design provides options for power management. This is one example of how software and hardware infrastructure enable each other to enhance product functionality. Partitioning is at the crux of hardware software co-design. This stage manifests itself finally in the product efficacy. The goal of partitioning is achieving performance within the overall system requirements. This is a multivariate problem. The techniques used are non-standard and depends on the designer to arrive at the best solution.

Hardware software co-design is a powerful tool to achieve and enhance complex product offerings. It comes with its own set of dynamism and complexities, which when handled with diligence and focus, yields great results.

References

- Hardware/Software Co-design Overview, RASSP Education & Facilitation Program, Module 14, Version 3.00
- Hardware/Software Co-Design, GIOVANNI DE MICHELI, FELLOW, IEEE, AND RAJESH K. GUPTA, MEMBER, IEEE
- A Comprehensive Approach to Power Management in Embedded Systems, Antˆonio Augusto Frˆohlich, Laboratory for Software and Hardware Integration (LISHA), Federal University of Santa Catarina (UFSC), 88040-900 Florianˆopolis, SC, Brazil
- Power Optimization and Management in Embedded Systems, Massoud Pedram, University of Southern California, Dept. of EE-Systems, Los Angeles CA 90089
- System Level Power Management for Embedded RTOS: An Object Oriented Approach, Ankur Agarwal ankur@cse.fau.edu, Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL 33431, USA; Eduardo Fernandez ed@cse.fau.edu, Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL 33431, USA
- Surveillance Radar Input Simulation; Avinash Babu M, Venkatakrisna Araveti & Samyeeer Metrani – International Radar Symposium INDIA.
- Hardware/Software Co-design Approach for an ADALINE Based Adaptive Control System, Shouling He, Department of Electrical and Computer Engineering, Penn State Erie, Erie, PA 16563, Email: sxh63@psu.edu; Xuping Xu, Department of Electrical and Computer Engineering, Penn State Erie, Erie, PA 16563, Email: Xuping-Xu@psu.edu
- Product design constraints and optimization (01 Jan 2012 – www.ednasia.com); Avinash Babu, Sr. Architect, Mistral Solutions Pvt. Ltd.
- Hardware/Software Co-design: Software Thread Manager - Michael Finley, EECS 891, Fall 2004, University of Kansas
- Power-Management Techniques for OMAP35x Applications Processors; Arthur Musah, OMAP3 Applications Engineer, Texas Instruments and Andy Dykstra, Marketing Manager, Power Business Development, Texas Instruments.

Mistral – Corporate Profile

Mistral is a technology design and systems engineering company providing end-to-end solutions for product design and application deployment. Mistral is focused in three business domains: Product Engineering Services, Defense Solutions and Homeland Security. Mistral provides total solutions for a given requirement, which may include hardware board design, embedded software development, FPGA design, systems integration and customized turnkey solutions.

Mistral's strategic partnerships with leading technology companies help provide customers with a comprehensive package of end-to-end solutions.

USA

Mistral Solutions Inc.,
4633 Old Ironsides Drive,
Suite 410, Santa Clara, CA 95054
Tel: +1-408-705-2240
Fax: +1-408-987-9665
E-mail: usa@mistralsolutions.com

India

Mistral Solutions Pvt. Ltd.
No.60, 'Adarsh Regent',
100 Ft. Ring Road,
Domlur Extension, Bangalore - 560 071.
Tel: +91-80-3091 2600
Fax: +91-80-2535 6444
Email: info@mistralsolutions.com

