

What Hardware Engineers Need to Know About Software and Vice Versa

-- Important systems-level trends in the embedded and chip industries

John E. Blyler

Editorial Director - Chip Design, Embedded Intel and Green Electronics magazines

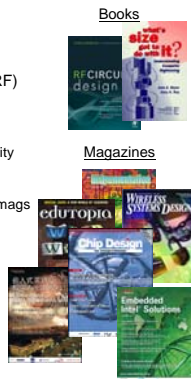
Affiliate Prof., Portland State Univ., Systems Engineering

Embedded Systems Conference

April 15, 2008

My Background

- BS Engineering Physics, MS EE (digital and RF)
- Engineering Background
 - 18 years - Complex hardware-software systems
- Teaching Background
 - Develop/taught courses for industry and university
 - Affiliate Professor - PSU, Systems Engineering
- Print/Online Media Background
 - Editor-in-Chief, Chip Design & Embedded Intel mags
 - Senior Tech Editor - Wireless Systems mag
 - Freelance, Edutopia magazine and others
 - Associate Editor, IEEE I&M magazine
 - Book author: IEEE-Wiley, Elsevier
- Contact: jblyler@extensionmedia.com
www.chipdesignmag.com, (503) 614-1082
www.cecs.pdx.edu/systems



Attendee Expectations

- _____
- _____
- _____
- _____
- _____
- _____
- _____

Agenda

- Why a discussion of hardware-software?
- Systems Engineering – Not again?
- HW-SW Implementation Case Study

Hammer-Nail Syndrome

- "To a hammer, everything looks like a nail."
 - Electronic engineer look for a electronic solutions
 - Software engineers (computer science majors) look for software solutions
- Case study
 - Students in electrical, civil, mechanical engineering and from an engineering of complex systems
 - Task: Develop a plan to create a system that could:
 - accept a list of names and addresses,
 - sort either the names or addresses in a specified order
 - sort as fast as possible a list of no more than 1,000 entries. .

Hammer-Nail Syndrome

-- Continued --

GROUP	STRATEGY	PROBLEMS
Electrical Engineers (Computer Science)	Develop software in C	Could not access computer No data tape reader Format problems
Mechanical Engineers	Use a mechanical card sorter	Format conversion problems
Civil Engineers	Use a spreadsheet	Data processing
Students that took course	Cut and paste manually or use spreadsheet	All problems anticipated

Electronics - Big Picture

- Electronics encompasses many domains and disciplines
- Each use “similar” but ideas and even terms:
 - Chip-package and PCB designers
 - EDA-semiconductor and CAD tools
 - Hardware-Software engineers
 - IP and Libs for reuse
 - Analog-Digital engineers

Systems Engineering as Big Picture

- With big “S” → pre hardware-software implementation
- Audience at university is System Engineers with BS degrees in EE (hardware), CS (software), CE, ME, Economist, etc
- Chief application of SE is in electronic industry
 - Need Rosetta Stone (or class) between SE-HW-SW!



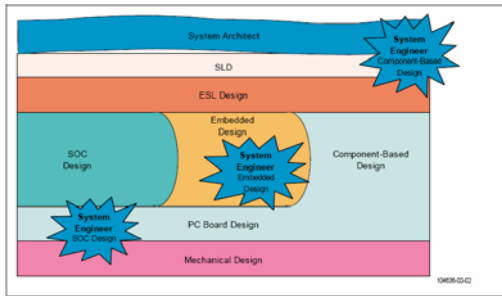
A Rose by any other Name

- Everyone thinks they're a systems engineer
 - And they are!
- Each discipline and domain reinvents the “SE” wheel
 - In part because SE is applied differently in different disciplines and domains, e.g., chip design vs commercial software
- “How does systems engineering differ for software systems and hardware systems, and why?”

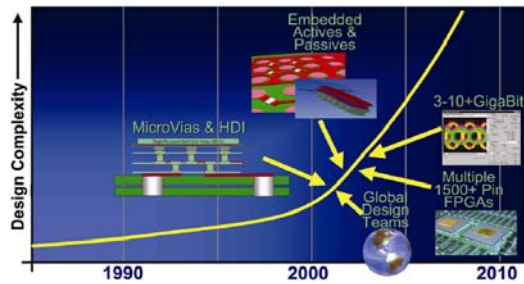
SE differences for HW and SW (Analysis of SPC Report)

- 10 Assertions in Paper
 - Requirements Criticality
 - Who tests the system?
 - Apportionment of complexity
 - Scope of "SE Work"
 - Interfaces
 - Engineering Degrees
 - Addressing "ilities"
 - Process focus
 - Object Orientation
 - Background and Emphasis
- 6 major differences
 - Physics
 - User Domain
 - Cohesiveness of HW and SW
 - Requirements Changes
 - Software Youth
 - "ilities"

SE view in Chip World (SoC View of Systems)

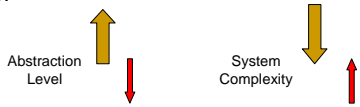


PCB vs. Chip

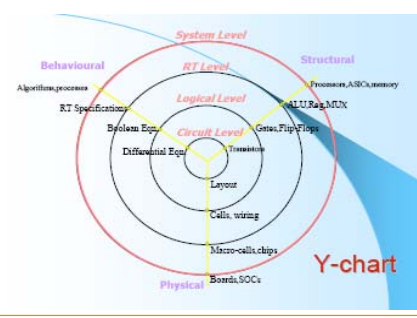


Framework for Complex HW-SW Embedded Systems

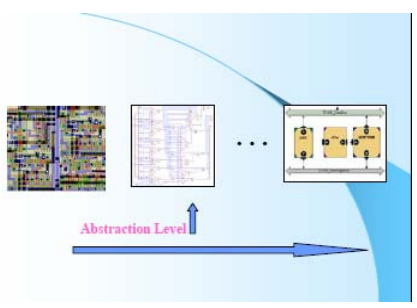
- Framework
 - Higher levels of abstractions for trade-off analysis
 - Design reuse/IP
- Abstraction - conceptual interpretation of a system



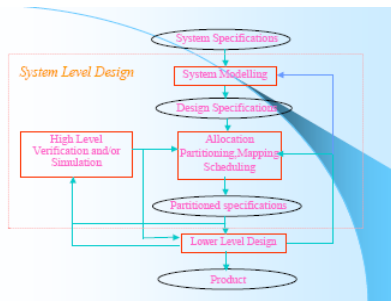
Y-Chart



Abstraction Level and Complexity



System-Level Design: What is it?



Main Hardware-Software System Engineering Concepts

Agenda – Key SE Concepts

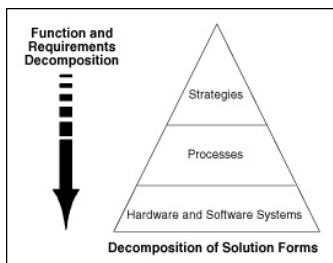
- SE Overview
- Life Cycle Cost analysis
- Iterative Process
- Technical Performance Measures (TPM)
- Make or Buy
- Risk Analysis
- Trade-off studies



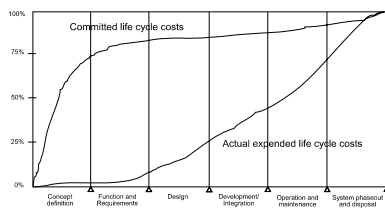
SE Topic Coverage

- SE integrates all technical areas of discipline, domain and processes
 - Concept generation
 - Requirements
 - Design
 - Solution
 - Trade-off analysis
 - "ilities"

Solutions Form Hierarchy

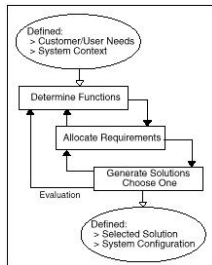


Life Cycle Costs



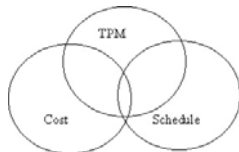
What's So Good to do With it? Understanding Computer Right-sizing, J. Byler and G. Ray, Figure 2-4, pg. 41

Iterative Process



TPM

- TPMs provide a technical leg to the otherwise lopsided program management stool of cost and schedule



TPMs

- Examples of key technical parameters include the following:
 - Performance -- speed, weight, temperature, range, receiver sensitivity, noise, and accuracy.
 - Software -- memory utilization/reserve, modules or libraries completed, unit test (modules passed), code size per module (lines-of-code), complexity measurements (e.g., McCabe), processor throughput
 - Supportability -- MTBF, MTTR, percent of standard components, level of modularity, availability of support equipment
 - Producibility -- availability of special materials and/or manufacturing equipment (e.g., flip-chip placement machines for ICs), special facilities
 - Engineering Process -- rework/redesign, yield, staffing, design progress (including document preparation), closed problem reports.

TPM Tracking -- VC "Trends Report"

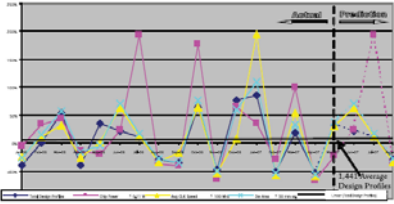
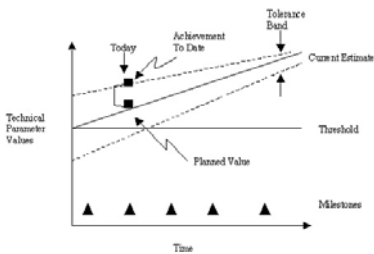


Figure 1: This trend chart illustrates the monthly changes in "interest" among the key technical parameters of chip power, performance, and die size.

TPMs



Make vs Buy: System Decisions

- **Make**
 - Proprietary product
 - Limited or nonexistent market
 - To reduce costs
- **Buy**
 - Workforce limited
 - Lack of skills
 - Outside of product line
 - Off-the-shelf availability

Trade-Off Study Process

Definition

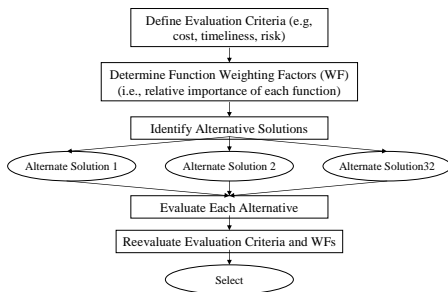
- Trade Study: A technical analysis comparing the technical cost, risk and performance attributes of two or more competing alternatives solutions against a predefined set of evaluation criteria to define the optimum solution.
- Is a trade study (or trade-off analysis) like an industry/market survey?

Trade Study Process

Decision Problems

- Common pitfalls to good decision making
 - Consider choices before defining the decision
 - Focus on favorite or popular alternative
 - Ignore consequences of a choice
 - Inadequate information
 - Study too long – indecision
- Reasons for above:
 - Schedule pressure
 - Participants viewpoints differ greatly
 - Don't understand which decision is really needed

Trade Study Process



Trade Study Process

-- Pre-Evaluation Criteria --

- Define decision
 - Define purpose of decision – be clear!
 - Identify correct level and scope
 - Maintain consistency with prior decision

Trade Study Process

-- Evaluation Criteria --

- Define evaluation criteria
 - Measurable/understandable (results, not features)
 - Meaningful differentiation
 - Mutually independent as possible
 - Consistent with policies, regulations, constraints
 - Relevance to decision statement
- Categorize functions into “musts” and “wants”
 - Must = Mandatory for success, i.e., go or no-go
 - Want = Desirable but not mandatory

Trade Study Process

-- Weighting Factors --

- Apply weighting factors to both musts and wants
 - Determine relative importance of each must and want (e.g., “1 to 10” with 10 being best)
 - Obtain several opinions on weighting factors
 - Assign weighting values
 - Devolve utility curves if appropriate
 - Assess factors and weights – do they look “right”?

Weighting factors answer the question:
What is the relative importance of this
element?

Trade-Study Process

-- Generate Alternatives --

- Review objectives
- Use brainstorming techniques
- Poll interdisciplinary teams
- Review prior similar programs
- Use simulation, prototypes, models, etc. to help define workable solutions
- Be ware of the level of uncertainty (risk)
- Keep weighting secret during the process

Trade Study Process

-- Quantify and Compute --

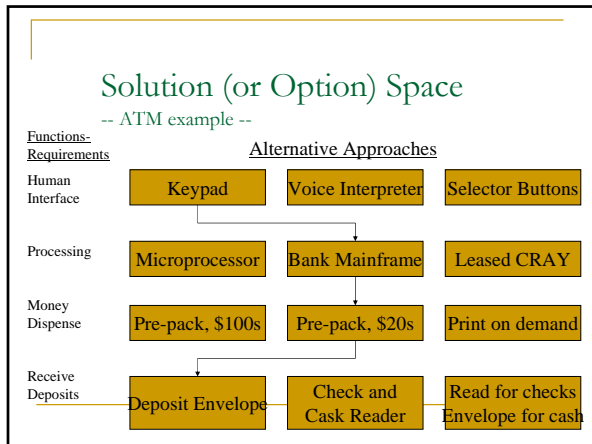
- Define raw score for each alternative solution
 - Primarily an engineering judgment
 - Obtain several options and combine
- Screen alternatives through "musts"
 - Musts must really be musts (ok to have wants)
 - Eliminate any alternative that don't meet all must criteria
 - Keep "close calls" for further study and fall-back
- Compute weighted score for each alternative
 - Multiply raw value by weighting factor
 - Record weighted value
 - Sum factors for each alternative

Raw scores: To what degree does this alternative meet this must or want?

Trade Study Process

-- Assess Risks and Decide --

- Assess risks of implementing highest-value solution
 - Select winner
 - Does it make sense? Any show stoppers?
 - Determine consequences of implementing winner
 - Assess potential problems/risks
 - Select next best alternative if risk too great
 - Iterate as required
- Understand sensitivity of the score to raw values and weighting factors
 - Remember: all are primarily judgments
 - A change in one or tow may change the result



Trade Study Process

-- Sample Weighted Summary Table --

		Alternative Solutions					
		Alternative 1		Alternative 2		Alternative 3	
i		Raw Value	Weighted Value	Raw Value	Weighted Value	Raw Value	Weighted Value
1	4	3	12	1	4	7	28
2	2	6	12	4	8	5	10
3	6	2	12	7	42	7	42
4	1	5	5	6	6	2	2
Total		--	41	--	60	--	82

Software Tool Selection

-- Example for Reverse Engineering --

Function	Sub-Function	Sub-Sub-Function	Priority (*) (Weighting Factor)	Evaluation Criteria (Specific Requirement(s))
1. Operating environment	1.1 Project Environment	1.1.1 Size of application supported	2	a) Up to 1 million lines of code. b) Up to 10 levels of nested code.
		1.2 Hardware/Software Environment	1	a) Pentium processor b) 32 MB RAM (or less) c) 30 MB Hard disk space (or less)
	1.3 Technology Environment	1.2.2 Software	1	a) MS-Windows 95/NT
2. Life cycle phase-orientation tool functions.	2.1 Modeling	2.1.1 Analysis	3	a) Three dimensional views of software files, function and class dependencies
		2.1.2 Diagramming	6	a) Generate flowcharts
		2.1.3 Hypertext aids	5	a) Source code components (files, functions and classes) linked by hypertext files.
	2.2 Implementation	2.2.1 Source code analysis	4	a) Calculate complexity metrics (number of branch points for a function). b) Lines of code.
		3.1 Documentation (Development)	3.1.1 Text	7
3. Common functions	3.1.2 Graphics	3.1.2.1	8	a) Flowcharts, trees
		3.1.2.2 Hypertext support	12	a) Output report in hypertext

HW-SW Differences

- And now, the differences

Differences Between Hardware and Software

- A symbiotic relationship, but differences do exist:
 - Development
 - Complexity
 - Defects
 - Manufacture
 - Maintenance
 - End Life

Embedded

- **1) Embedded**
- An embedded system is a special-purpose computer system within an electronic device. It is designed to perform a specific task, rather than be a general-purpose computer for multiple tasks (as in a PC). Embedded systems are a combination of computer hardware and software designed to performance a specific function (see Figure 4)
- An embedded system is any computing system that is not a desktop computer nor a server.
- The definition of an embedded system is applicable to either chips or board electronics. Either way, software plays perhaps the most critical role in any embedded system.

Difference between Embedded and Application SW

- Embedded software engineers must understand how software interfaces with hardware.
 - Must be more hardware-aware, for example to interface at a detailed level with hardware inputs/outputs – e.g., a UART chip
- Application software engineers – especially in the PC world - seldom need to understand anything about the hardware upon which their software runs.
 - Insert Application Programming Interfaces (API) – which calls a supported library routine - into the proper line of code.

Differences to Desktop Computing

- Interaction with physical environment
- Closed loop
- Malfunction may lead to damage
- No or very restricted human/computer interface
- No or very restricted maintenance possibilities
- Part of competitively priced products (high volumes)
- Tight resource constraints
- Often special hardware
- Part of engineering product
- High product generation frequency
- Often many variants
- **Implications for SW engineering?**

Hardware vs Software Concerns

- In embedded (chip and board space), different set of concerns:
 - Hardware engineers are much more sensitive to the cost of materials than software engineers.
 - After the software program has been developed, it costs virtually nothing to replicate.
 - Copy of hardware requires the consumption of physical materials – e.g., chips, other components, boards, power supplies, cables, cases, etc.
 - Cost of hardware grows as the volume of a hardware product increases (chip costs go down, but still cost).
 - Ship 50K hardware units per month, need to find most cost-efficient parts available.
 - Designing a hardware system that saves even 10 cents each for a particular type of chip will result in a savings of \$5K/month.
 - Designing system to use less expensive part takes requires engineering and purchasing manhours. → growth of SAP and initiatives

Hardware vs Software Concerns

- continued

- Cost - Number of hardware parts is critical
 - Every addition part cost money and takes up valuable space – on the chip or on the PCB.
 - Reducing their Bill-of-Materials (BOMs) is big deal in terms of cost and "going green" (process cost)
- Power - Addition active part uses a little more power.
 - For battery-operated, mobile devices, every bit of power is a concern.
 - Even for plug-in applications, like server-farms, power can be an expensive concern.
 - More power means larger power supply (more cost and board space) and/or increase need to power dissipation devices like fans and fins on chips (again, more cost and space).
 - As a rule of thumb, faster chips – those that run on a higher clock rate – require more power. [Biyler, John, "Understanding Low-Power Designs," Wireless Systems Design magazine, June 2001, pp 57-58.]

Hardware vs Software Concerns

- continued

- Electrical-Electronic Power:
 - "...Basic electronic theory indicates that to reduce power (P) usage, either the current (I) or voltage (V), or both, must be reduced:

$$P=I \times V$$

- Dynamic Power in clocked digital chips:

$$P \propto C \times V^2 \times f$$

Hardware vs Software Concerns

- Continued

- Power consumption reduced by running chips are slower clock speeds.
 - Multicore craze
- Power can also be reduce through the use of software to either;
 - Perform the same function as additional hardware chips
 - Through control of power usage throughout the system (power management applications).
 - Why power is one of the key TPMs in the trade-off analysis between hardware and software.
- All of these concerns are why hardware engineers try to implement as much product functionality as possible in software.
 - Rule of thumb, a product with more software and less hardware will be more desirable.
 - Functionality implemented in software often runs slower than in hardware.
 - Recall wireless encryption schemes – like WEP – used in mobile devices.
 - Software implementation to slow, so no one used it

Hardware vs Software Concerns

- continued

- Migrate software to hardware to improve performance (Part of power-performance-area trade-off):
 - Acceleration of software in hardware
 - Critical Blue → Using trade-off or "what-if" analysis tools, software examines existing object code written for a microprocessor system
 - Automatically determines what can usefully be migrated to hardware, then creates the hardware to run it.

Software's Affect on Hardware

- Software runs on hardware.
 - So hardware design must precede software, especially for embedded apps
 - But once the hardware is designed, it is very costly to change.
 - Wouldn't it be better to design the software first? But how can the design of software precede that of hardware, if the software requires hardware in order to work?

Software Affect on Hardware

-- continued

- Virtual Prototypes
 - Co-Ware
 - Virtutek
 - Vast
- Complete, bit-accurate model of a board or SoC that is sufficient for software developers to use as a target.
 - Main design objective is functional verification of the hardware-software interfaces (e.g., drivers).
 - Low-level software development can then be validated early.

Software Affect on Hardware

-- continued

- What functions should run in software and which in hardware?
 - Run simulation models.
- Important software/hardware trade-offs:
 - Selection of Microprocessor
 - Memory Requirements
 - Implementation of Peripheral Devices
 - Debugging Resources

Hardware Engineering

-- Chip Overview --

- “Chuck” of Silicon could be use to create:
 - General purpose processor
 - Pentium
 - DSP
 - FPGA
 - ASIC



Chip Overview

-- General Guidelines --

- General purpose processors
 - Pros: Low cost, mature product
 - Cons: Not as efficient as DSPs for some apps (audio) and consume more power than DSPs
- Fixed DSP
 - Pros: Low cost, mature product, great for numerical calculations
 - Cons: Lacks flexibility, less programmer friendly, not good for multimedia encoding algorithms

Chip Overview

-- General Guidelines (continued) --

■ FPGAs

- Often used to implement "system glue" logic

□ Ex: Diamond Rio MP3 player

- Uses Micronas MAS 3507D fo
- Controlled by an 8-bit microcontroller
- FPGA used interface microcontroller to memory, the parallel port and the decoder.



Design

■ Design:

- Define a system's functionality (what will it do)
- Realize that functionality with a physical implementation

■ Embedded design takes time – low productivity

- Tens of lines of code per day
- Hundreds of transistors per designer per day
- "Productivity gap"

Intellectual Property (IP)

- Hardware for Chips -

■ IP core

- Block of logic or data that is used in making a field programmable gate array (FPGA) or application-specific integrated circuit (ASIC)
- Essential elements of design reuse - repeated use of previously designed components.
- Should be entirely portable - that is, able to easily be inserted into any vendor technology or design methodology
- Examples:
 - UARTs
 - CPUs
 - Ethernet controllers
 - PCI interfaces

Intellectual Property

- Software Reuse -

- The software library is a good example of **abstraction**. Programmers may decide to create internal abstractions so that certain parts of their program can be re-used, or may create custom libraries for their own use. Some characteristics that make software more easily reusable are **modularity**, **loose coupling**, high **cohesion**, **information hiding** and **separation of concerns**.

HW IP called by SW

- FPGA Coprocessors -

- What?
 - Hardware IP, called by software
 - Offloads computationally intensive tasks
- Why?
 - Easy system design
 - Easy Creation of Coprocessor
- How?
 - Standard SW and HW interfaces
 - Design automation software

VSIA Approach

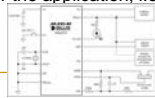
-- HW IP and Software Reuse --

- IP reuse was needed to address hardware issue
- Hardware-dependent software (HdS) needed for software
 - drivers, bootstrap code, algorithms, and the layers of RTOS that interface with hardware

HW-SW Implementation - Illustrative Example

HW-SW Implementation Example

- UART (Universal Asynchronous Receiver/Transmitter)
 - Serial port that is a common feature μ_c and μ_p .
 - Simple way for two μ_c to communicate without having to match their system clock rates.
- Why implement a UART in software?
 - Many μ_c don't have UART ports
 - Hard to find perfect fit for every design requirement – SW implementation adds to flexibility
 - Port may exist, but be inadequate for the application, i.e., not fit the requirements.
 - Ports may exist, but not enough



Simple Serial Routine (19200 baud)

```

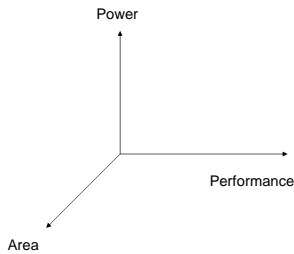
; TRANSMIT ROUTINE
;-----
TxRoutine  movl TxSend    ; temp store data to be sent
;          bsr PORT_TX    ; set start bit
;          call HBDelay5
;          call HBDelay5
;
;          movl 0x00      ; 8 data bits to send
;          movl 8Count
;
;          TxLoop: rrl TxSend
;                  test STATUS.C
;                  goto TxFB8
;
;          rnc
;          bsr PORT_TX
;          goto Don8bit
;          TxHB8: bsr PORT_TX
;                  goto 8x1
;
;          Don8bit: call HBDelay5 ; wait 1 bit length
;                  call HBDelay5
;                  decr 8Count
;                  goto TxLoop
;
;          call Du8bit
;          rnc
;          bsr PORT_TX ; stop bit
;          call HBDelay5
;          call HBDelay5
;          Du8bit: return

```

UART Example

- HW advantages
 - Usually **faster** than SW (if speed and/or bandwidth is an issue)
 - Free the microcontroller from the low-level details of the serial protocol
 - Tedious bit-sampling, time-slot counting, and input and output bit shifting
- HW Disadvantages
 - Less flexible
 - May take more **power** from system
- SW Advantages
 - Suitable for low performance apps where system ROM size and bandwidth don't drain crucial microcontroller time and resources.
 - Adds to flexibility of solution
 - Any aspect of the protocols can be updated to keep pace with changes to other devices or to evolving standards.
 - Takes no additional **area**
- SW Disadvantages
 - Not suitable for high performance apps
 - Typically slower than hardware

Technical Performance Measures - Power, Performance, Area



Power Trade-off

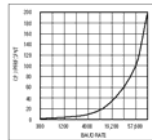
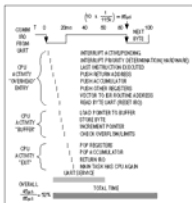
- Trade-off question: What is the amount of processing power required for this UART functioning – hardware vs software implementation?
- The worst-case interrupt code path for any of the states is 19 instruction cycles (applies when characters are transmitted or received continuously.)

Band Rate	Cycles per Timer Tick (at 1.5MHz)	Worst-Case Interrupt Cycle Count	Bandwidth Used by UART (%)
9600	372	19	5
19200	186	19	10
28800	124	19	15
57600	62	19	31

Performance Trade-off

- SW implementation
 - Except in the simplest cases, SW UART consumers HW resources, e.g. must include the percentage of computational time demanded from the CPU.
 - Typically ok for low performance apps (depending on system ROM size and bandwidth)
 - Example: Smallest and lowest power systems (hand-held industrial equipment, bar-code readers, test equipment, and consumer products)
- HW implementation (external UART)
 - Typically needed for high performance apps (depending on system ROM size and bandwidth)
 - Often costly, required a lot of power and PC-board real estate, and usually exceeded the needs of the application.

SW Performance Cost



The percentage of CPU time required for servicing a software UART rises sharply with the baud rate.

These details show how the CPU time is allotted in servicing a software UART

Courtesy of Maxim: New IC Caps Two Decades of UART Development

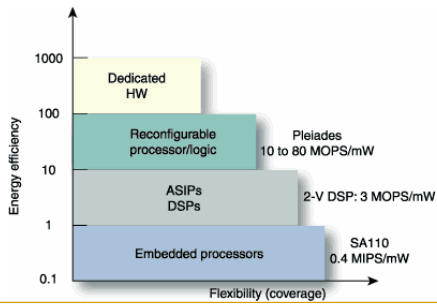
Area Trade-off

- SW implementation
 - No additional hardware is needed (assuming timer is available)
- HW implementation
 - UARTs typically big, though now have miniature versions
 - Larger BOM, cost per unit

Cost Trade-off

- Software implementation
 - Except for simplest cases, more expensive
- Hardware implementation
 - Typical hardware costs

Power Savings



Opportunities for Low-Power

Algorithms	Minimize Operation
Source Code	Optimized code
Compiler	Energy miser
Operating System	Scheduling
ISA	Energy Exposed
Microarchitecture	Clocked Gating
Circuit Design	Low voltage swing
Manufacturing	Low-k dielectric

Some Power Models

- Macro level
 - Arithmetic
 - Software
 - Memory
- Activity Based
 - Empirical
 - Information-theoretic
 - Signal modeling-based

Empirical

- Based on chip estimation system [Glaser ICCAD91]:
 - $P = \alpha G(E_g + C_L \cdot V_{dd}^2) f$
 - G = number of equivalent gates
 - E_g = energy consumed by an equivalent gate
 - C_L = average loading per gate including fanout
 - α = activity factor
- Demerit: lacks consideration on different logic styles

In General, Software Advantages

- Finally, with the communications peripherals implemented in software, any aspect of the protocols can be updated to keep pace with changes to other devices or to evolving standards.

Thank You!

In-Class Labs

In-Class Lab

- Consider this example of how differently we perceive problems and their potential solutions. After reading the paragraph below, indicate two potential solutions.
 - "A treasure hunter is going to explore a cave up on a hill near a beach. He suspected there might be many paths inside the cave so he was afraid he might get lost. Obviously, he did not have a map of the cave; all he had with him were some common items such as a flashlight and a bag. What could he do to make sure he did not get lost trying to get back out of the cave later?"

In-Class Lab

-- Board to SoC --

- Consider the ChipSat project described in Appendix B3 of the "NASA Complex Electronics Guidebook," which briefly explains a project that migrates an existing on-board computer – printed circuit board - to a single system-on-a-chip (SoC).
- Drawing from your experience on a similar project OR by searching the Internet for a similar project – i.e, migrating the functionality of an existing PCB to an SoC (either implement on an ASIC or FPGA chip) – briefly identify and describe the following:
 - Problems areas, that is, functions or devices that can not be done easily on an SOC.
 - Interface issues that are different between a printed circuit board and an SOC chip.

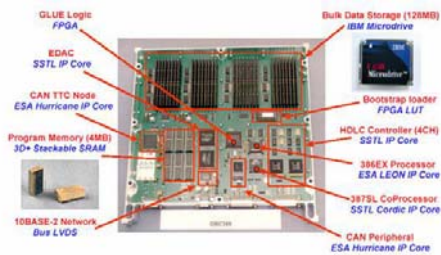
ChipSat

-- Continued --

- ChipSat is a long-term research program which aims to build a satellite-on-a-chip. As part of the program, an existing on-board computer (OBC) was scaled down to a system-on-a-chip (SoC). The OBC chosen was developed by the Surrey Satellite Technology Limited (SSTL), a company owned by the University of Surrey in Guildford, UK. The SoC is prototyped on a single high-density programmable logic array chip using soft intellectual property (IP) cores.
- The image below shows the parts of the OBC that were mapped into the system-on-chip. An entire board was shrunk down to a single chip. The experiment showed that it is possible to implement the functionality of a small satellite OBC on a single programmable logic chip.

ChipSat

-- Continued --



In-Class Lab

- Determine the key technical parameters and baseline margin levels for the wireless case study.

Trade-Study Analysis Lab -- Wireless Case Study --

Exercise

- 1) Read the Wireless Case study
- 2) Identify the systems trade study factors
- 3) Develop a trade study matrix of factors vs. options using the template provided.
- 4) Recommend the best solution. You may add additional options for consideration, with appropriate justification.

Wireless Case Study

Overview:

This case study centers on the product design of a next generation wireless PDA, a mobile, computational device. As a key wireless design engineer, you have been asked to incorporate multimedia capabilities into a significant upgrade of an existing PDA product line. The initial product specification from marketing is somewhat vague, but it does contain a few hard requirements. For example, the new design must be capable of decoding MPEG-4 streaming audio-video data. This multimedia file must be obtained wirelessly via Bluetooth or WLAN connection from the Internet. A wireless headset (WPAN) is an option. Of course, since the device will be portable (or perhaps even wearable), it must be as small and light as possible, consuming only a minimum of power. Also, the cost must be competitive and it has to reach the market place within 9 months. Marketing also requires that the additional of a camera, for sending of video pictures (i.e., MPEG-4 encoding), be considered.



Wireless Case Study

-- Continued --

There are also some important considerations in this design. For example, power is the independent variable for mobile devices. Everything else is function of power, including speed, performance and graphic resolution. Some "rules of thumb":

- > Power and signal/circuit speed are inversely proportional as trade-off items, i.e., the faster a circuit runs, the more power it requires. So an important design goal is to run a circuit or device at the lowest speed that will still provided an acceptable level of performance.
 - > Better graphic resolution typically means more power consumption.
- Performance is another concern: Studies have shown that an multimedia movie must run at 12 frames per sec to be of an acceptable quality. Below this frame rate, viewers notice blurring effects and the eye becomes distracted. A working number of 15 frames per second is identified as a frame rate for the project.

Wireless Case Study

-- Continued --

The specifications for the existing product are as follows:

- > Size = 5.25" x 3.25" x 0.75"
- > Weight == 7.8 oz (with battery)
- > Memory = 8 MB
- > Display = 16-bit color, 1.5 by 4.5 inch (HW) viewing area
- > Power = Rechargeable Li-ion battery
- > Exact power consumption = Uncertain, estimate 300 mW for busiest operation (Read/Write, CPU active and Backlighting)

Wireless Case Study

-- Continued --

Other questions must be address, such as whether the current foundry is able to support a new SOC design, if required. How will you be able to test/verify the decoder and RF/antenna designs, since these are technologies outside of your current area of expertise?

Your company has a low-risk policy of using proven technology – no new development – unless there is a convincing business case. Also, the company prefers to outsource for expertise it does not possess.

Wireless Case Study

-- Lab Approach --

The course participants will be mixed into groups of 4 to 5 people who will work together in two 45 minute lab sessions. The first 30 minutes will be devoted to working the lab problem, while the remaining 15 minutes will be used for presentations and Q&A. One person in each group will be selected to present the team's work. The first lab will try to establish a reasonable set of high-level requirements, based on the initial input from engineering, marketing, testing and manufacturing departments. The second lab will perform a brief trade-off analysis on one of the critical design issues (e.g., MPEG4 encoder implementation), based upon the requirements from the first lab.

Trade-Study Lab

-- Matrix of Factors vs. Options --

Options \ Factors	Option One	Option Two	Option Three
Total Cost Factor (\$K)			
Development Cost (\$K)			

TPM Lab

- Determine the key technical parameters and baseline margin levels for the wireless case study.

Checklists and Templates

- Requirements Evaluation
- Subsystems Specification
- Trade-off
- Architecture Design
- Test Plan

Checklist and Templates

-- Requirements Evaluation Checklist --

- Is the scope of the project adequately identified?
- Are the requirements organized appropriately to convey system partitioning?
- Are the requirements specific?
- If a requirement is not adequately specified (vague), does the document identify this and say that the design will reflect this uncertainty?
- Is each requirement uniquely identifiable? (trace anchor)
- Is there a clearly identifiable customer for the project?
- Is the expected response time, from the user's point of view, specified for all necessary operations?
- Do the requirements avoid specifying the design?
- Are the reqmts clear enough to be given to an independent group for implementation & still be understood?
- Is each requirement testable?
- Is each item relevant to the problem and its solution? Can each item be traced to its origin?
- Do any of the requirements conflict with each other?
- Is each requirement necessary?
- Does the requirements document meet documentation standards?

Checklist and Templates

-- Subsystem Specification Checklist --

- Are timing requirements that impact the design so stated?
- Are timing requirements not specified by the customer assumed? How?
- Do assumed time requirements agree with those already defined?
- Have the following timing requirements been considered (where applicable):
 - Throughput time
 - Response time to queries
 - Response time for major system functions
 - Priorities for input handling
 - Priorities for output
 - Sequence and interleaving of tasks
- Have all constraints been identified and specified?
- Have requirement constraints been realized as design constraints?
- Have the capacities of all communication interfaces been defined?

Checklist and Templates

-- Subsystem Specification Checklist (continued) --

- Concerning support software; have the following been considered:
 - Compilers and precompilers
 - Vendor development tools
 - Vendor software packages to support the application's software
 - Database management software
 - Telecommunication software
 - Non-vendor, existent software
 - Software to be developed outside the development organization, i.e., outsourcing.
 - Third-party software.
- Does the OS description define features such as inter-task communication and scheduling that are relevant to the design?
- Are all operator initiated programs and functions identified?
- Are the general operating procedures covered, such as: start-up, shutdown, restart, recovery, contingency, degraded operation?

Checklist and Templates

-- Trade-off Checklist --

- Exploring all candidate alternatives?
 - Is each alternative clearly defined?
 - Have the alternatives been pre-selected? How?
 - Can these pre-screened alternatives be defended?
 - Have affordability limits been established?
- Selection criteria identifies?
 - Are all significant criteria identified?
 - Do the criteria discriminate between alternatives?
 - Are the criteria measurable?
- Is the criteria weighting system acceptable?
 - Are rationales for the weighting factors explained?
 - Are criteria weights consistent with guidance?
 - Are criteria weights consistently distributed in the tree?

Checklist and Templates

-- Trade-off Checklist (continued) --

- **Utility (scoring criteria) determined?**
 - Is a defensible rationale established for each criterion?
 - Are criteria developed from operational cases, where possible?
 - Do all planes use the same numerical scale?
 - Is the location of the "zero point" explained?
- **Evaluation methods documented?**
 - Are test data reliability estimates (confidence levels) incorporated?
 - Are models validated? When? Who?
- **Sensitivity been estimated?**
 - Are error ranges carried through with worst case scenarios?
 - Have the effects of changes in the utility curve shapes been examined?
 - Have rationales for the limits been developed?

Checklist and Templates

-- Architecture Design Checklist --

- Does the design implement the requirements?
- Are the requirements complete?
- Are all parts of the design traceable back to requirements?
- Can all design decisions be traced back to trade studies?
- Have design trades been performed and documented?
- Have the assumptions been documented:
 - Is the trade space defined?
 - Are the goals defined?
 - Are the trade criteria defined?
- Will the selected design or algorithm meet all of its requirements?
- Are the primary performance parameters specified? (power, memory size, throughput, speed, etc.)
- Does the design reflect the actual operating environment?
- Is the design sufficiently detailed to proceed to the detailed design phase?

Checklist and Templates

-- Test Plan Checklist --

- **Test Strategy**
 - Do the integration test procedures exercise control and data interfaced as defined in the design document?
 - Have integration test procedures been defined? Do they match the order of integration?
 - For phased delivery, have test baselines been established in each phase for use in the next?
 - Are any areas which require testing not addressed by the Test Plan?
- **Traceability**
 - Are all test requirements traceable to higher level requirements documents?
 - Does the Test Plan list all the specifications, standards and documents necessary for its development?

Checklist and Templates

-- Test Plan Checklist (continued) --

- **Regression Test**
 - Are sufficient tests identified to reverify previously tested related functions (regression test procedures)?
 - Are all significant code and hardware assembly changes exercised – particularly interface modifications?
- **Simulation/Hardware**
 - Are there simulator and hardware dependencies that are not addressed?
- **Test Cases**
 - Is the testing approach reasonable and feasible?
 - Is there sufficient test coverage to show that the function being tested operates correctly within its intended environment?
 - Are the evaluation criteria determining success or failure of the test defined?
 - Have all nominal and extreme test procedures been identified?

Background slides

Systems Engineering

-- A Rose By Any Other Name --

- “Systems Engineering”
 - Network, Software, Systems-on-a-chip (SOC), architect, requirements ???
 - DOD stigma – ridge and unyielding
 - Sometimes viewed as academic (not at PSU)
 - “Integrated Product Development”
 - My preferred equivalent
 - Others: Platform-based development ...
-
