The Shrink Wrapped Myth:

Cross Platform Software

Asia and South Pacific Design Automation Conference
January 18-21 2010 – Taipei Taiwan

Mike Olivarez

freescale™ semiconductor

Embedded Software Definition

- Embedded software is thought of as that for non general purpose processing computers with few applications.

- The evolving definition of embedded systems.
  - Microcontroller – "The Elevator" sequence… Everywhere
  - Engine/Sensor Control – Sensor networks and response
  - Networking and wireless – Protocol stacks and translations, data server, security
  - Consumer Electronics – A world of possibilities…
    - Robotics
    - Media Players
    - Cellular
  - Crossovers – Smart Phone, MID: is it a computer or an embedded system?

- **If you can dream it… we can embed it!**

Issues With Shrink Wrap Software

Performance

- Getting the most of the system = efficiency
  - Performance (speed and power) is more than just the hardware
- Once the hardware architecture is set, efficiency is set by the software
  - **Option 1) Hand Coded Assembly**
    - Pros:
      - Highly efficient code based on system constraints for speed
      - Efficient with memory utilization
      - Easy to take full advantage of the architecture
    - Cons:
      - Time Consuming
      - Not portable, architecture dependant
  - **Option 2) Compiler Efficiency**
    - Pros:
      - Higher level of abstraction for code, more portable
      - Faster development, time to money
    - Cons:
      - May not be as efficient or use all the capabilities of the architecture
      - May require more memory

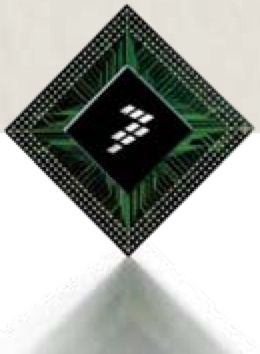
Efficiency Processor A

5 Stage Pipeline

- Cycles per Instruction (CPI):

RISC Processor A	Instruction %	Compiler Efficiency	Instructions by Type (millions)	Pipeline Penalty Cycles	CPI Penalty
ALU Instructions	40%	100%	471	0.000	0.000
Loads	25%	80%	294	0.465	0.116
Stores	15%	88%	177	0.120	0.018
Branches	20%	50%	235	0.825	0.165
Total				**Cycles**	**CPI**
	100%		**1177**	**1529**	**1.30**

- Pipeline Penalty Cycles Calculation:
  - Cycles = Σ(Instructions + (Instructions * Pipeline Penalty))
- CPI Penalty Calculation:
  - CPI = 1 + Σ(Instruction % * Pipeline Penalty Cycles)



Efficiency Processor B
8 Stage Pipeline

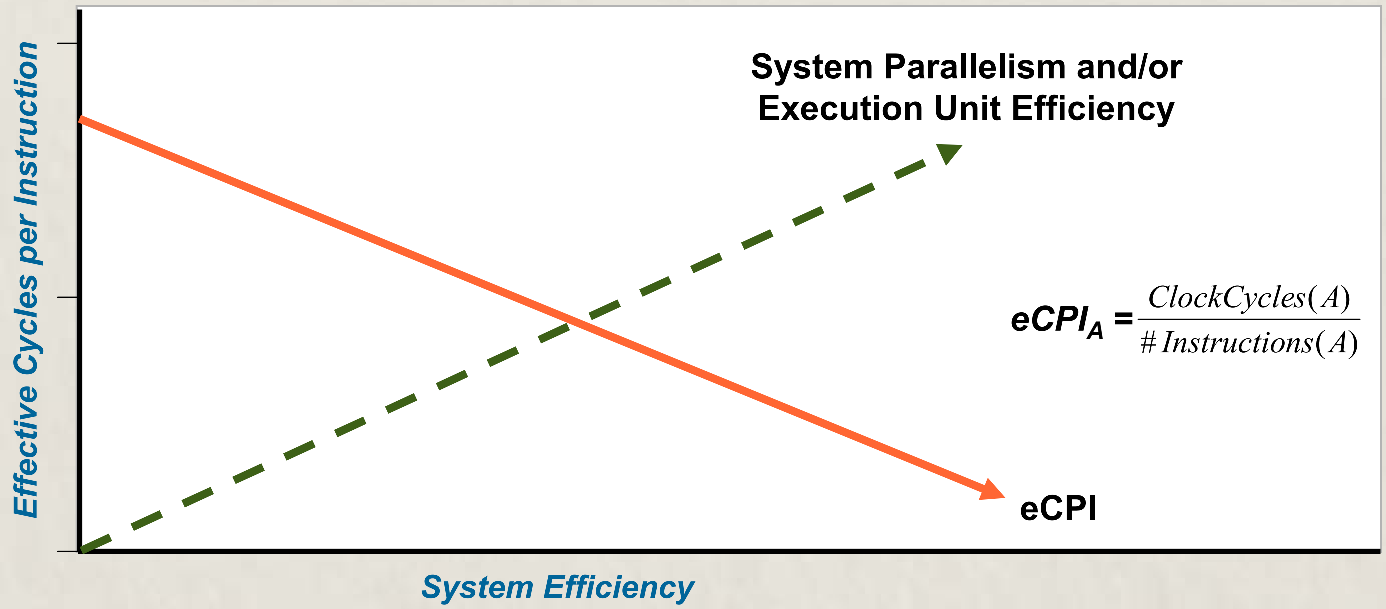
- Cycles per Instruction (CPI):

RISC Processor B	Instruction %	Compiler Efficiency	Instructions by Type (millions)	Pipeline Penalty Cycles	CPI Penalty
ALU Instructions	40%	130%	471	-0.300	-0.120
Loads	25%	80%	294	0.465	0.116
Stores	15%	100%	177	0.000	0.000
Branches	20%	80%	235	0.726	0.145
Total				**Cycles**	**CPI**
	100%		**1177**	**1343**	**1.14**

- Differences in Processor A and Processor B
  - SIMD adding efficiency for ALU Instructions
  - Backside Cache for Stores
  - Branch Prediction enhancements
  - Pipeline Stages – breaking down the work done in smaller chunks to increase clock speed.

Effective Cycles per Instruction

- Cycles per Instruction (CPI) is an effective measurement of how much computational work is being done per clock cycle, to compare architectures
- Effective CPI (eCPI) is a method which allows for dissimilar architectures to be compared, with a baseline for the number of instructions needed for a specified task



$$eCPI_A = \frac{ClockCycles(A)}{\#Instructions(A)}$$

Improvements though Parallelism

- *Improvement in the application due to parallelism is dependant on the sequential portion.* - Amdahl's Law
  - Essentially based on single or symmetric multi-core solution.

- Embedded System Solutions:
  - Adding Specialized Execution Units
    - Take advantage of the Architecture
    - Speeding up the limiting factors
  - Synchronous vs. Asynchronous Processing
  - Compiler and OS environments to take advantage of the architectural differences.

Compiler: Getting the Most of the Processor

- Compiler needed to take advantage of the architecture
  - Without the changes, the advantages of SIMD can't be used.
  - Branch Prediction Unit may not be as efficient
  - Instruction Sequence may not take advantage of the Pipeline

- Using Processor B without the change in the Compiler may not run the application faster.

- **Compiler is an integral part of the System Design.**

Operating System Examples

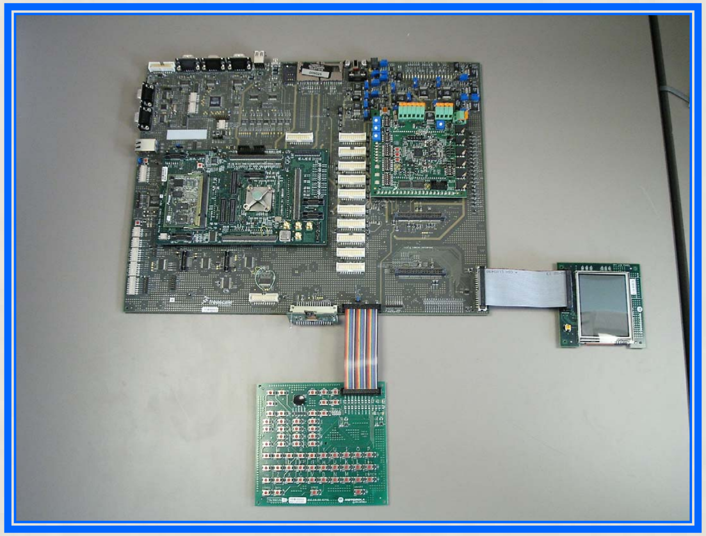
- “RTOS”
  - Vertex/Nucleus
  - QNX
  - RTXC
  - RTLinux
  - pSOS
  - LynxOS
  - eCos
  - µC/OS
  - many, many more…
- Easier to tune for few applications
- Few if any additional layers
  - Complexity added to applications

- Open OS
  - Linux
    - Android
    - Palm Web OS
    - LiMo
    - etc…
  - Windows CE/Mobile
  - Symbian
  - BlackBerry OS
  - iPhone OS
- Easy to port new apps
  - Layers to ease porting
- APIs for code reuse
  - Well documented APIs
  - Complexity abstracted from applications

freescale™ semiconductor

Debugging

- Application debugging
  - System Processor
    - From one to many processors
    - Symmetric and Asymmetric Processing
      - Multiple of instantiations of he same Processing Elements
      - Instantiations of different Processing Elements
    - Code Traces and Profiling



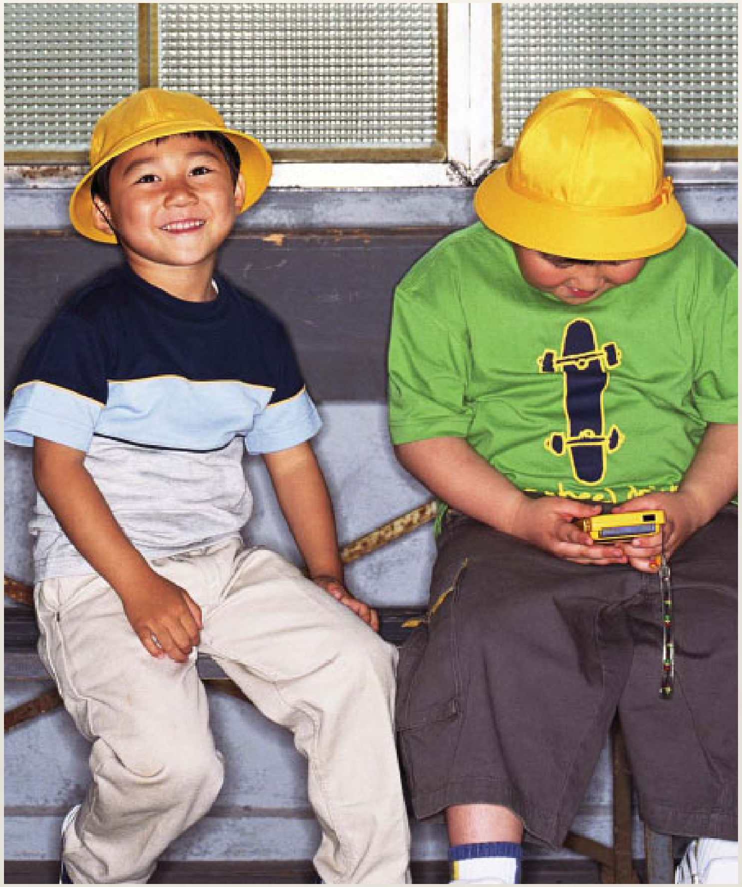
- System Debugging
  - System Peripherals
    - Interfaces, Buttons and Displays
    - Man/Machine Interface
    - Machine/Machine Interface

freescale™ semiconductor

Modeling and Simulation

- Model Creation
  - Instruction Set Simulator
  - Peripheral Simulator
  - Levels of Abstraction
    - Statistical Model
    - Cycle Approximate

- Model Uses
  - System Specification
  - Performance Estimation
  - Bandwidth Analysis
  - Early Code Generation

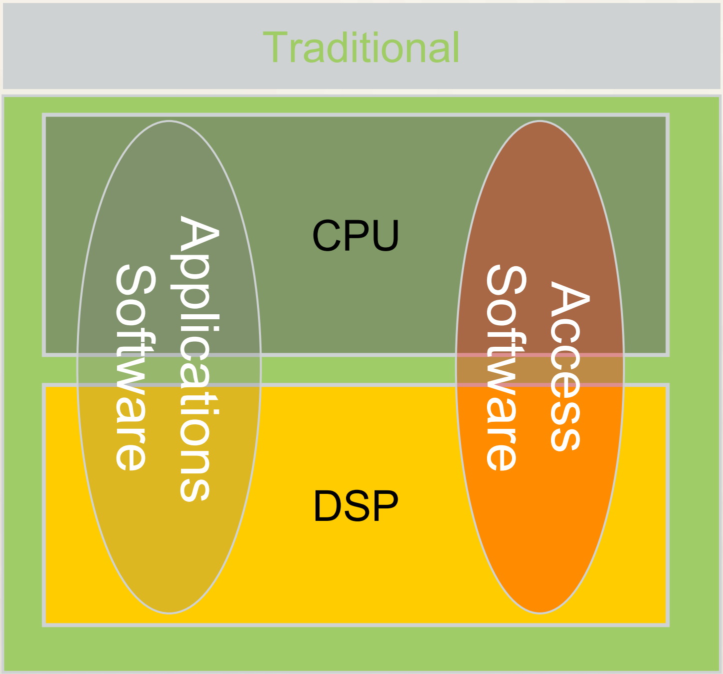


- See *“System Design: A Practical Guide with SpecC”* for more information.

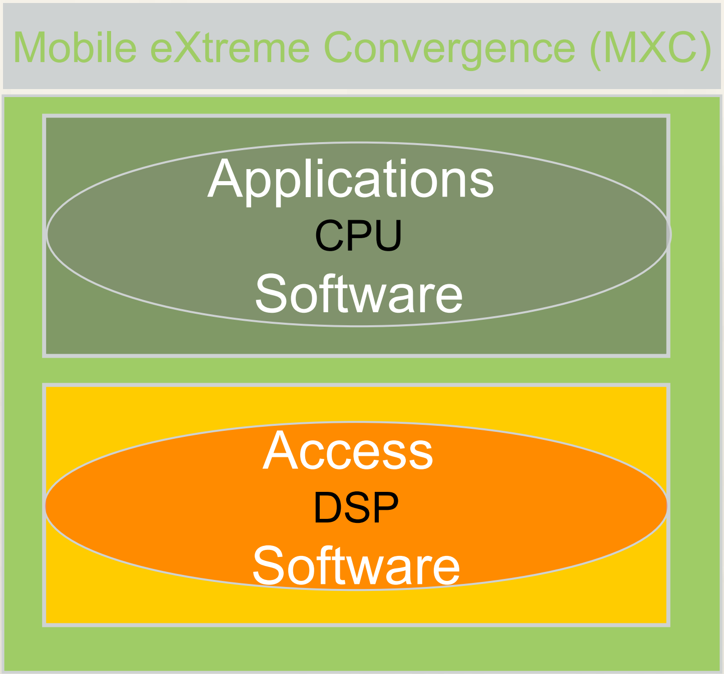
Freescale's MXC300-30.1 Platform Example

Because of an order from the United States International Trade Commission, BGA-packaged product lines and part numbers indicated here currently are not available from Freescale for import or sale in the United States prior to September 2010: MXC91321, MXC300-30.1

Value Proposition - Mobile eXtreme Convergence (MXC)

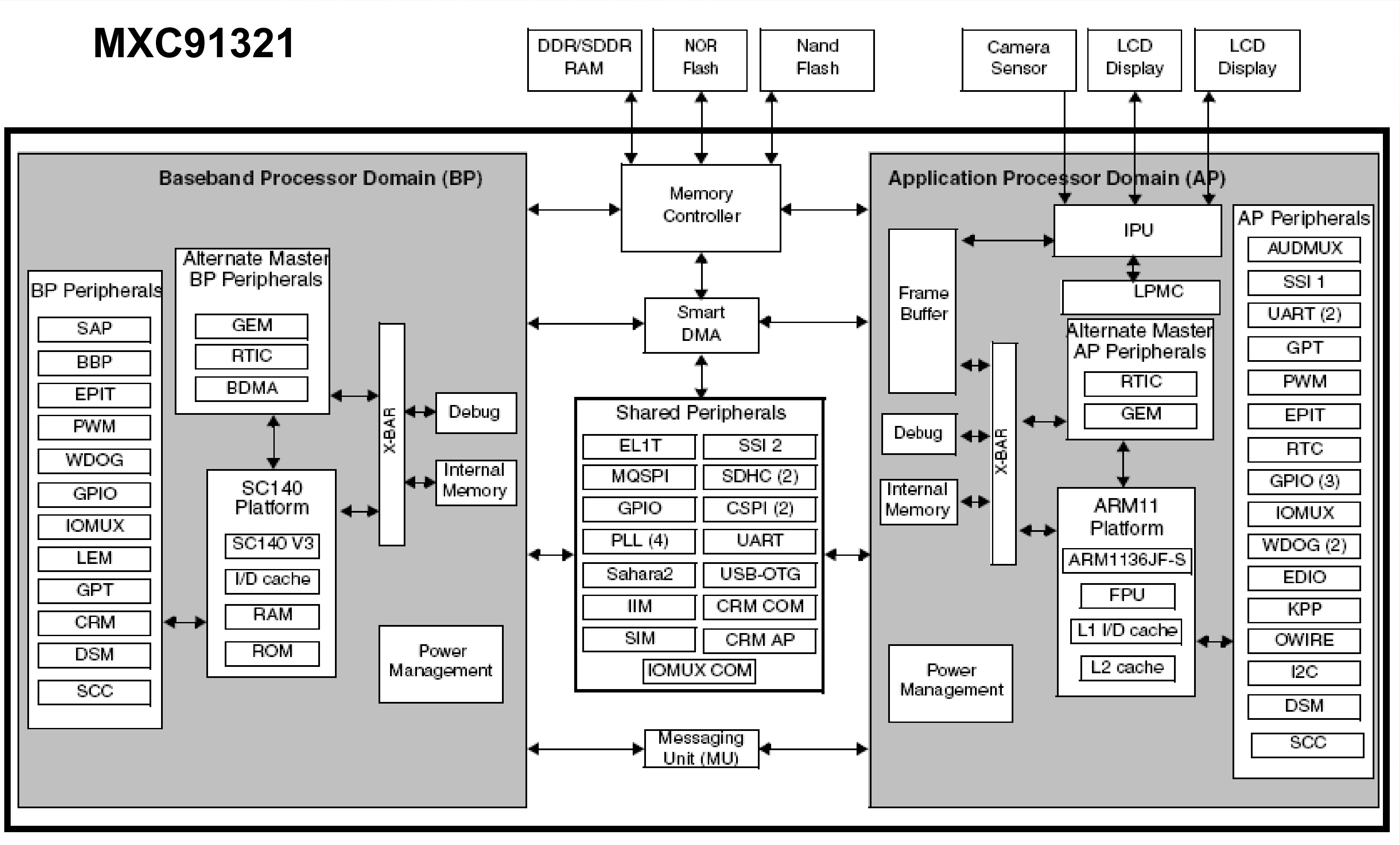


- Shared resource conflicts
- Difficult integration of 2 OS
- Difficult to scale
- Open OS can require extra processor and memory



- Clean separation of domains
- Simpler integration of 2 OS
- Easy to scale / tier
- More secure
- Enables separate modem / Apps

MXC Architecture Overview

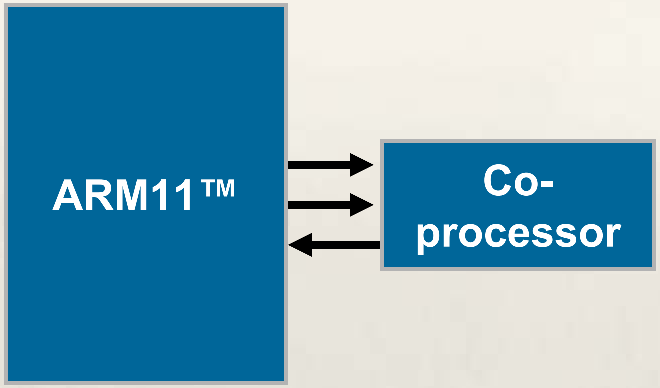


Because of an order from the United States International Trade Commission, BGA-packaged product lines and part numbers indicated here currently are not available from Freescale for import or sale in the United States prior to September 2010: MXC91321, MXC300-30.1

Specification Difference

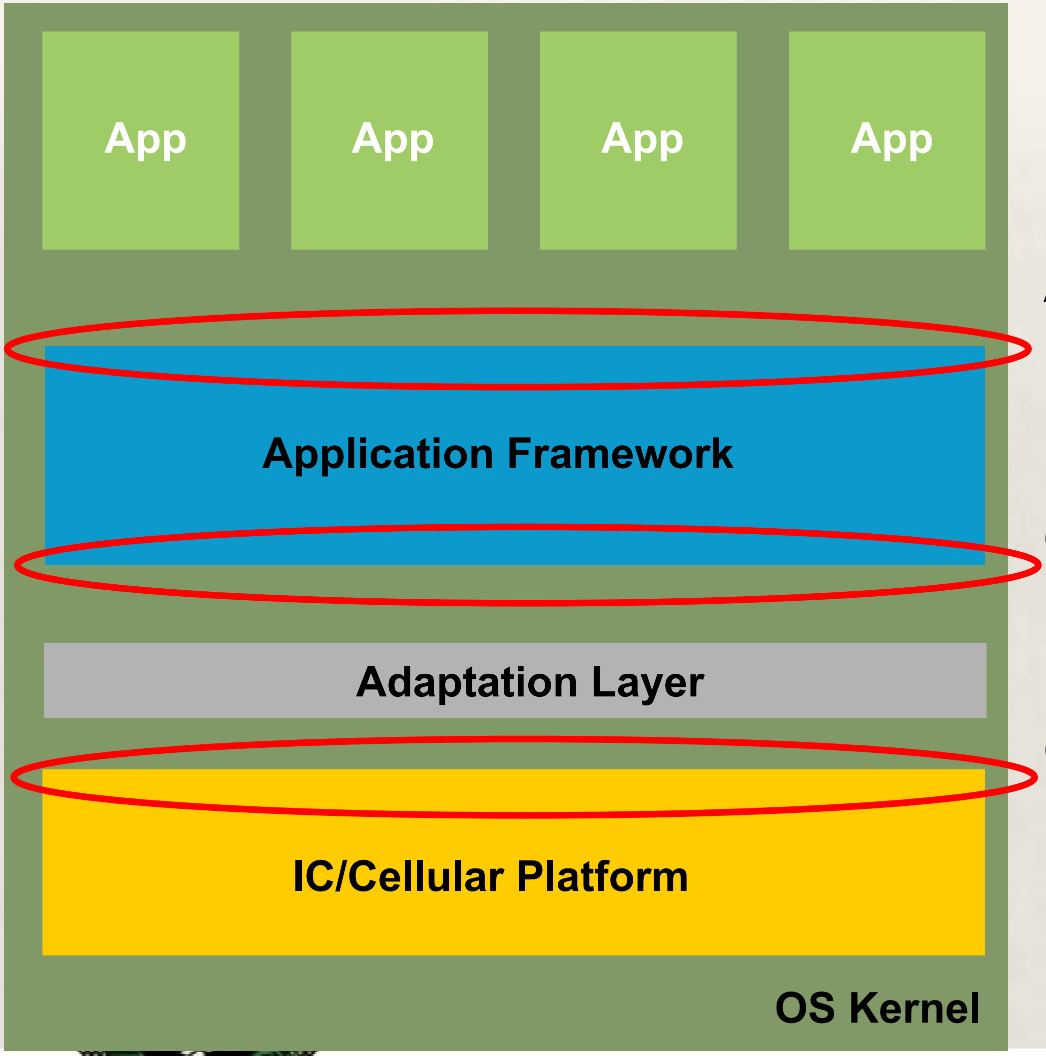
Creating the Specification Model helps answer:

- Will the behavior be needed in the system, based on the proposed use cases?
- What is the benefit of the behavior?
- What are should be the specification for the behavior?



Improvements over Floating Point Software	Floating Point Hardware Improvement Percentage	Fixed Point Software Improvement Percentage
Application Cycles	825 %	837 %
Floating Point calculation cycles	1525 %	793 %
Instructions issues for Floating Point Calculations	2223 %	923 %
Application energy savings	83 %	88 %
Floating Point calculation energy savings	91 %	88 %
Code memory footprint savings	16 %	16 %
Arithmetic code development time	15 min	~120 min

Open OS Cellular Software Layering

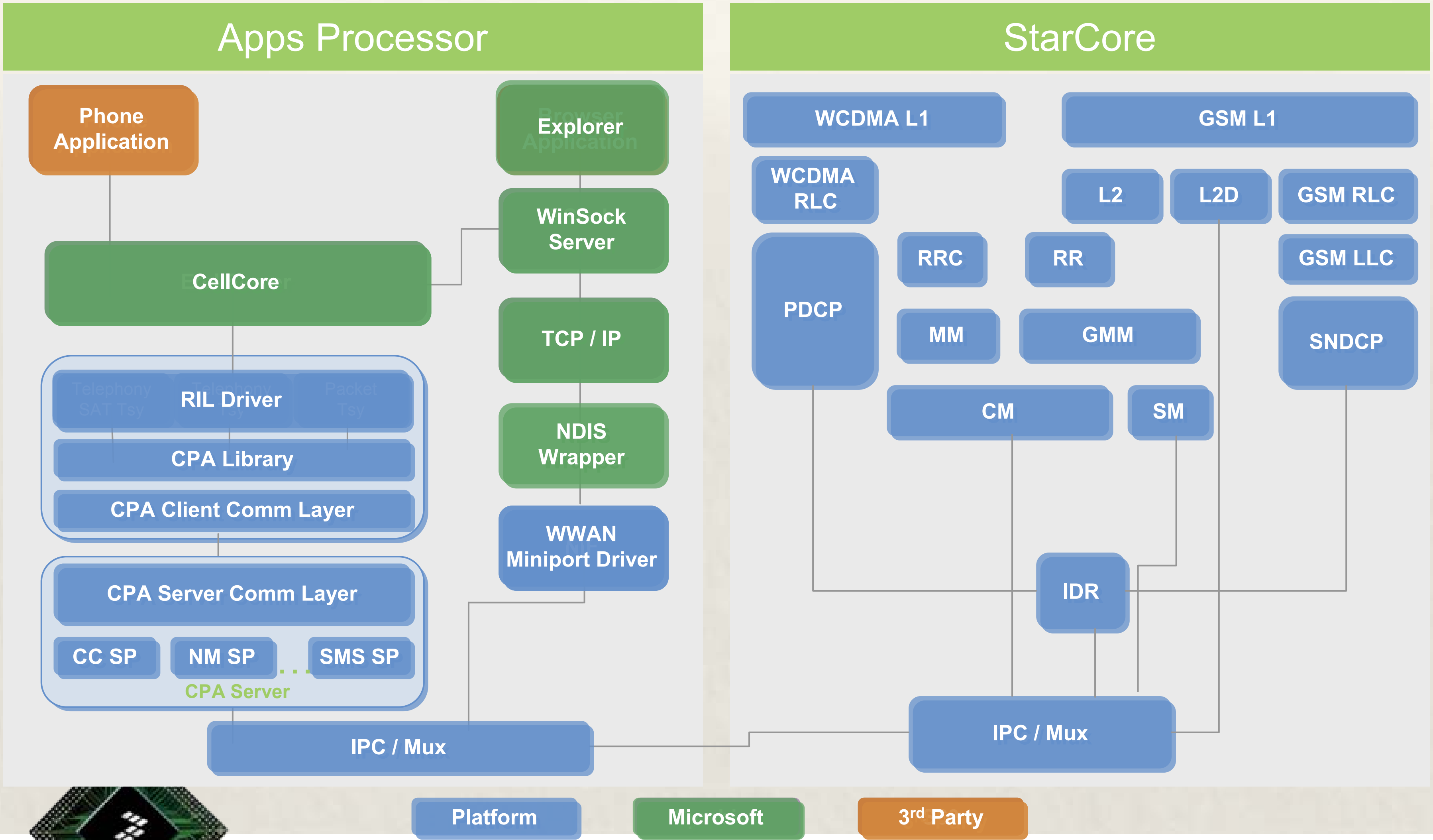


Application Programming Interface

OS/App FW defined, platform integration Interface

OS Independent Common Platform Interface, e.g., CPA

Cellular Platform Access (CPA) Abstraction Layer For OS Flexibility



The Shrink Wrapped Myth

- “Silicon without software is just sand” - Todd Conroy, Freescale Semiconductor
  - Software defines the usability.
  - Optimization of the code is critical to performance.
  - Operating System is an critical part of the System Design.
  - Software is an integral portion of the MXC platform to allow for small devices to provide a great user experience.

Thank You

freescale™
semiconductor