

Reusable Embedded IP

*Standards and Strategies for Development, Cataloguing,
and Re-factoring Existing Embedded Technology*

What do we mean by reusable “Intellectual Property” (IP)?

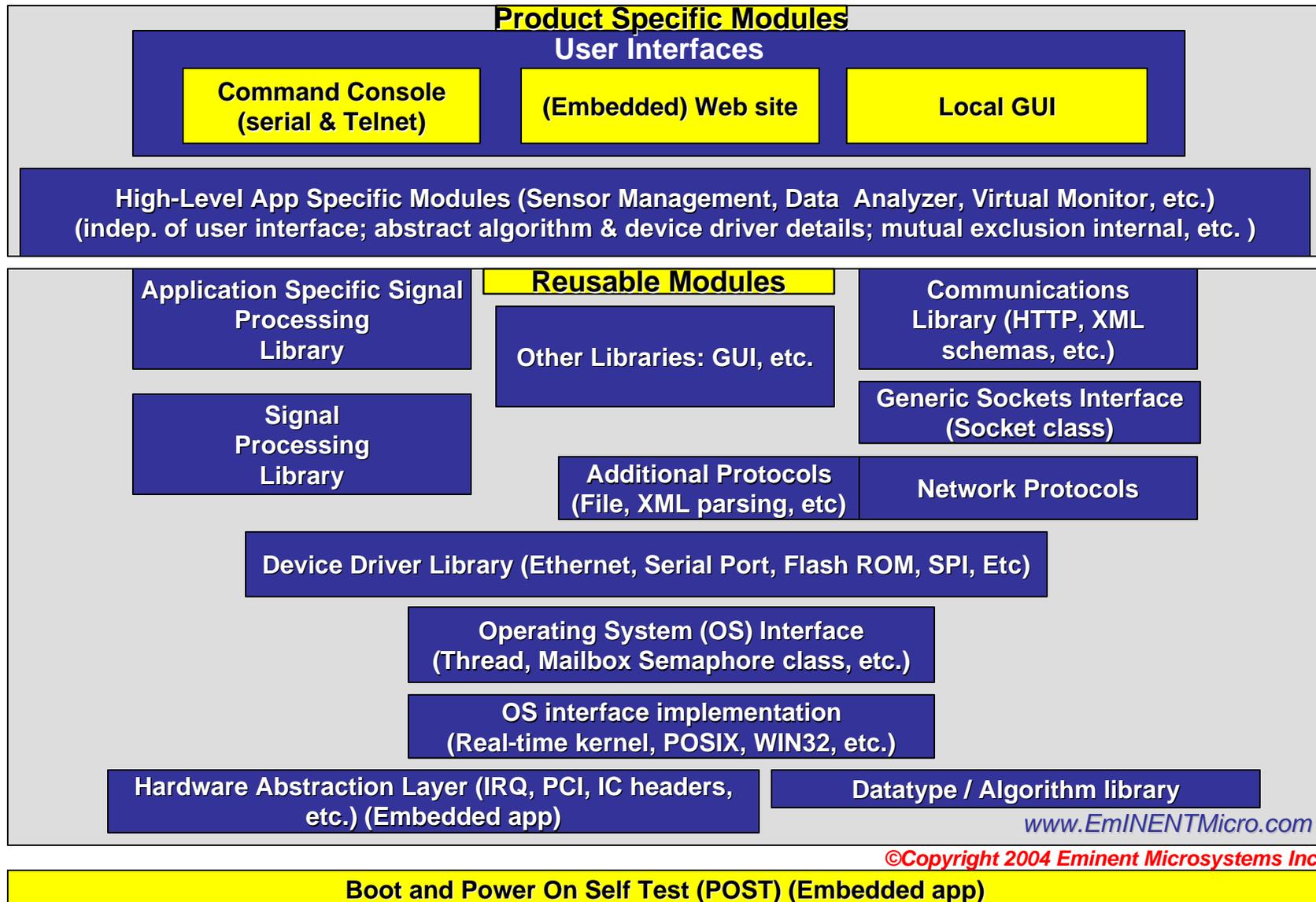
- A logical, hierarchical organization of software and hardware (fine and large grain) components constructed to support rapid engineering of standard and embedded computer platform based products in general, instrumentation, and consumer device markets.
- **Exact same** components are directly integrated with and used in design simulation tools, design test tools, and products. Software components are directly compiled and linked (NOT cut-and-pasted) into multiple products.
- Reusability and flexibility are NOT the same thing!
Envision “Lego blocks”, not a “Swiss army knife”.

- Addresses abstract issues such as reusability principles for architectural design, application construction, and component modularization and organization.
- Addresses concrete issues such as component implementation interfaces, existing libraries, build tools, integration with management tools and processes, product application examples.
- Complementary to design management strategies and processes
- Independent of specific design tools.
- Supportive of commonly accepted technology standards

- Organize the components in a hierarchy that matches a conceptual hierarchy already familiar to engineers
- A standard directory hierarchy that is organized according to “is a type of” and “is composed of” relationships.
- E.g.
 - Layers of abstraction in a computer system
 - Algorithms, data structures, protocols
- Directory branch naming provides an unambiguous choice to find any particular type of code module

- **Hardware Abstractions** – generic interfaces to low-level hardware. Device drivers use abstract interfaces to get to an interrupt controller, PCI bus, etc.
- **Operating System Abstraction** – no direct calls to any OS specific service. Thread creation, mutual exclusion, and communication services use a common abstract interface.
- **Environmental Headers** – environmental dependencies isolated to specific environ.h and library standard headers. Standard DebugPrintf() interface. No pollution of core code modules with hundreds of “ifdefs”.
- **Code Module Naming and Function** – module names are not too broad or too narrow to specify exactly the single (large-grain, high-level or small-grain, low-level) function.
- **Allocation** – allocation strategy is externally specified and not built-into libraries. I.e. no built in heap style allocation.
- **File I/O** – application specific file I/O is not built-into libraries.
- **Device drivers** – implementations divided into hardware dependent portion, OS dependent portion, and OS/HW independent portion.

- Follows general reusability principles.
- Vector/Matrix base classes for “Matlab/Octave style” data arithmetic.
- Fortran comparable numerical efficiency.
- Mathematical topological graph base classes and streaming mode support for easy creation and modification of topological properties – branching, insertion/deletion of processing components, etc.
- Event, Callback Abstractions to support application independence and application specific event handling.
- State Machine base classes.
- Core signal processing support – Filter, Signal, FIR, IIR, FFT, etc.
- Advanced signal processing support – Filter Banks, Adaptive Filtering, Pattern Recognition, Wavelets, Neural Nets, Time-frequency Analysis.



- **Strict one way layering** – higher layers can call lower layers, but NEVER vice/versa. Asynchronous information/events flow from lower layers to higher layers using registered “Callback objects” (reusable Callback code in library). Completely change (add a new) higher level with NO changes to a lower one.

- **Application domain abstractions** - high-level interfaces of minimum complexity that naturally capture application behavior and completely encapsulate and hide implementation detail – device driver interfaces, signal processing algorithms, communication methods, etc.

- **Primacy of application specific entities** – number of threads or processes is an internal implementation detail that should be hidden behind application domain interfaces.

- **Standard files for application layer configuration – E.g.**
 - Datatypes.h – datatypes and algorithm configuration (sizes of buffer pools, etc.)
 - Devices.h, devices.cpp –device driver layer configuration.
 - Protocols.h – protocol layer configuration.

- Develop the highest reputed, most capable, and most innovative products
- Faster new product time-to-market
- Reduced product development costs
- Reduced cost of field support and maintenance
- Higher product quality/reliability

Thank You