An efficient thread-based programming model for multi-core systems

MPSoC '04

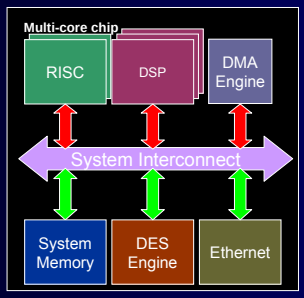
Mark Lippett, Ignios Ltd.

Agenda

- Introduction
- Comparing programming models
- An efficient thread based abstraction layer
- A pragmatic programming model
- Conclusions

Industry trend #1

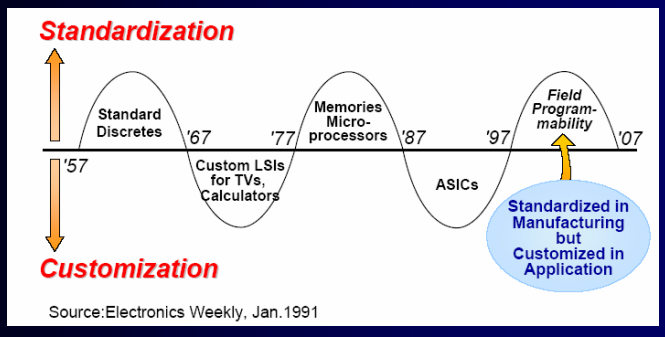
- Proliferation of multiple cores in embedded chips
  - Mandated by legacy (hardware & software)
  - Aggressive price/performance/power requirements (Si efficiency)



- Enabled and compelled by Moore's Law
  - ITRS: 2009, 90nm process, 100M gates = 2500 ARM7 cores

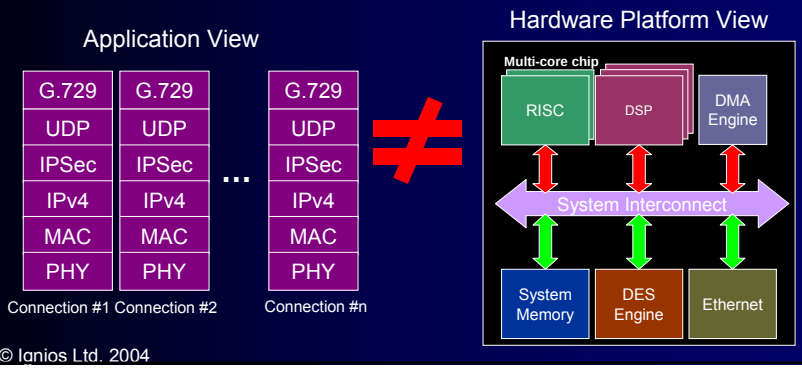
Industry trend #2

- "Software-programmable platforms" proliferating
- Many names, many "merchant-market" vendors
  - ASSP, DSP, FPGA, PSoC, microprocessor, NPU..

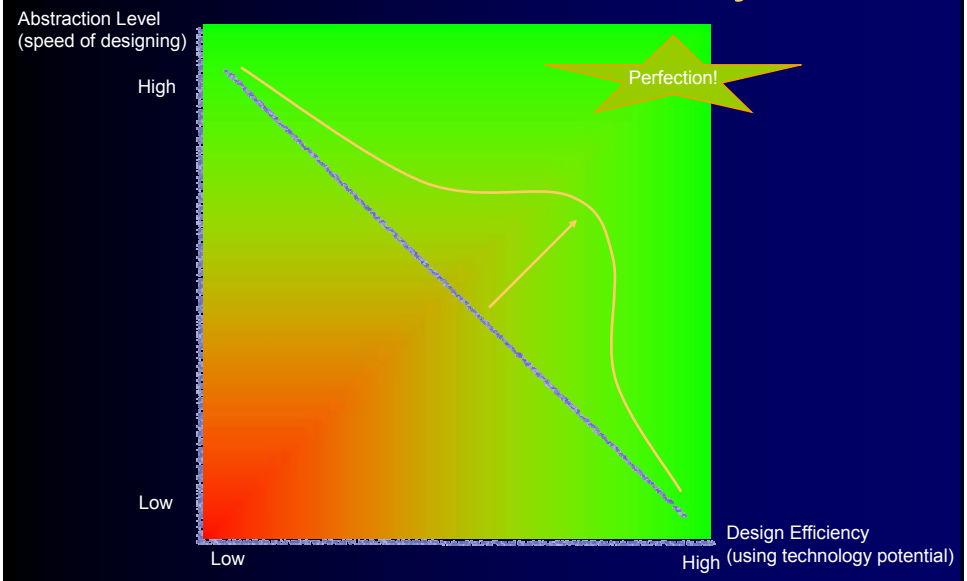


The key multi-core problem

- Multi-core devices are notoriously difficult to program
  - Mis-match of programming model and underlying hardware
  - Reduces efficiency of development (TTM) and deployment (BOM)
- Increases existing gap in Semi → OEM supply chain



"Achievable Efficiency"



Agenda

- Introduction
- Comparing programming models
- An efficient thread based abstraction layer
- A pragmatic programming model
- Conclusions

Comparison of Programming Models

Hardware-centric

- Efficiency unlocked
  - Intractably complex
- Multi-core limitations
  - Explicit partition
    - Unmanageable
    - Not scalable

Software-centric

- Complexity managed
  - Inefficiencies
- Multi-core limitations
  - Proxy agent or..
    - Bottleneck
  - ..limited comms
    - Restricts partitioning

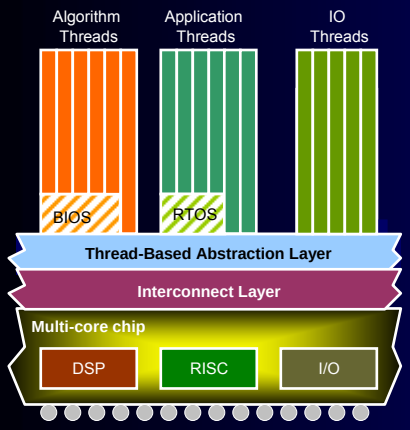
Enabling Achievable Efficiency

- It all depends on the programming model
  - Abstraction and efficiency are normally mutually exclusive
- No “one size fits all” programming model for MPSoC
  - Different types of user, application, supply chain interaction
  - *Requires an evolution, not a revolution*
- Requires the provision of an optimised, yet generic, foundation for high-level programming models
  - “Threads” are the common currency for concurrent systems

Agenda

- Introduction
- Comparing programming models
- An efficient thread based abstraction layer
- A pragmatic programming model
- Conclusions

Thread-based abstraction

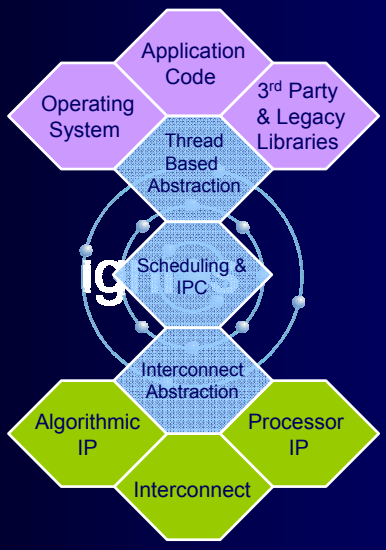


- Provides
  - A suite of thread control and synchronisation methods
  - access transparency
    - the same methods can be used when invoking a method from *any* core to *any* core
      - Abstraction from esoteric usage models
  - location transparency
    - it is not necessary for the invoker to know where the required processing resource resides
      - Abstraction from complex system topology
- A foundation for a unified programming model for heterogeneous MPSoC
  - Basis for 3rd Party OSes
- Achievable in software?

Delivering Achievable Efficiency

- SystemWeaver™ – a hardware solution to a software problem
  - An on-chip system management technology for complex multi-core architectures
  - Embodied in hardware yet maintaining flexibility
  - Improving efficiency
- Integration is minimally disruptive to hardware and software legacy
  - Hardware: IP core + integration framework - for bus and/or distributed architectures
  - Software: programming model leverages existing data-path code, control code through API compatible with 3rd-party RTOS

Combining Efficiency & Abstraction

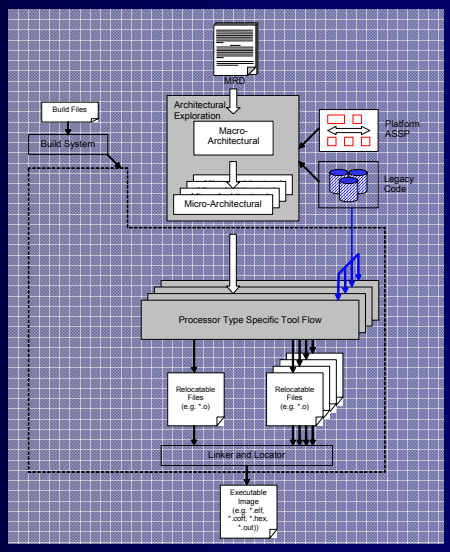


Agenda

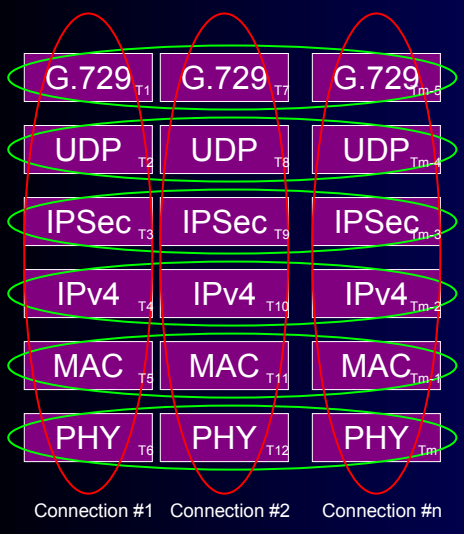
- Introduction
- Comparing programming models
- An efficient thread based abstraction layer
- A pragmatic programming model
- Conclusions

Top-down design flow

1. Review requirements
2. Explore architecture
   - Hardware defined?
   - Legacy software/libraries?
3. Define macro-architectural partition
4. Define micro-architectural partition
5. Create code
6. Compile
7. Link
8. Execute

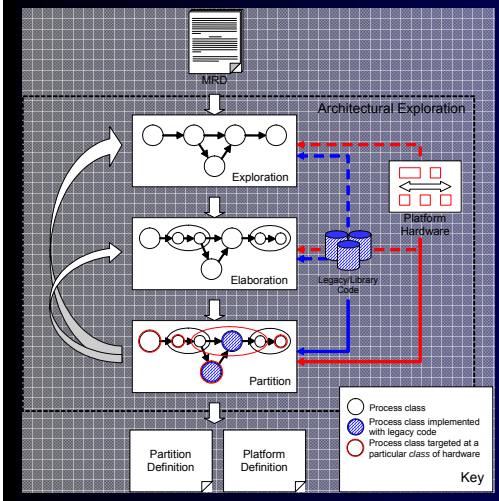


Macro-architectural partition



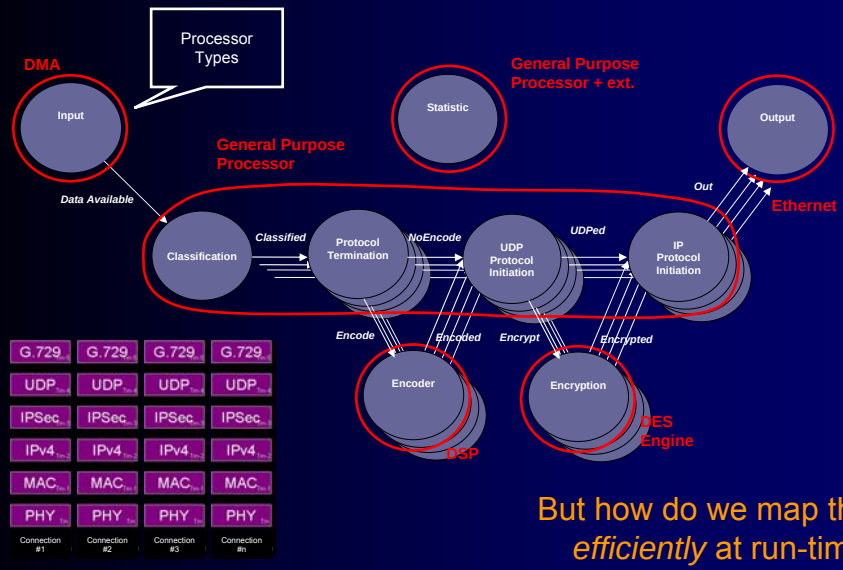
- Focussed on application-centric requirements
- Concurrency and functional partition expressed as communicating “tasks”
- No limitation on granularity of process decomposition

Micro-architectural partitioning



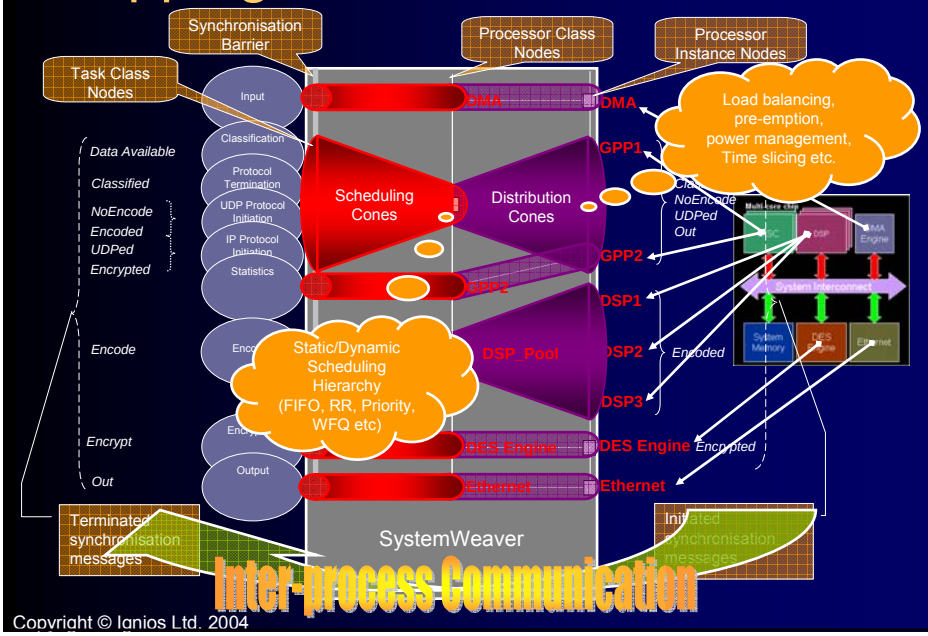
- From…
  - platform and legacy independent
- to…
  - platform and legacy specific
- Elaborate top-level partition
  1. considering predefined hardware architecture
  2. Considering existing code (legacy/3rd party)
- May require iteration to reach final process/ thread partition

Micro-architectural partition



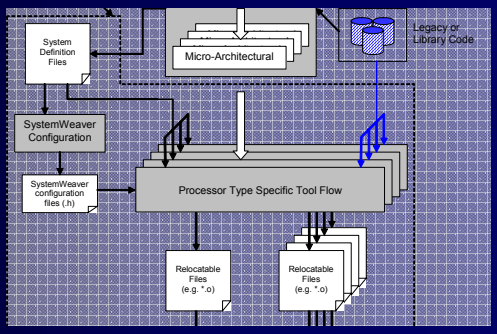
But how do we map this *efficiently* at run-time?

Mapping Software onto Hardware

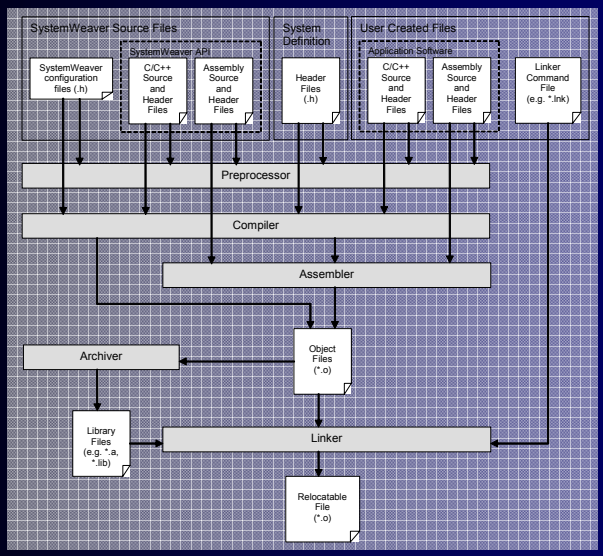


Source code

- Generation, creation, management
  - Processing resource specific tool flows
    - Compatible with existing tools
  - Greater ROI
    - Leveraging existing skills
    - Supporting legacy code and 3rd Party libraries
  - Intuitive system design
    - Source level system definition files

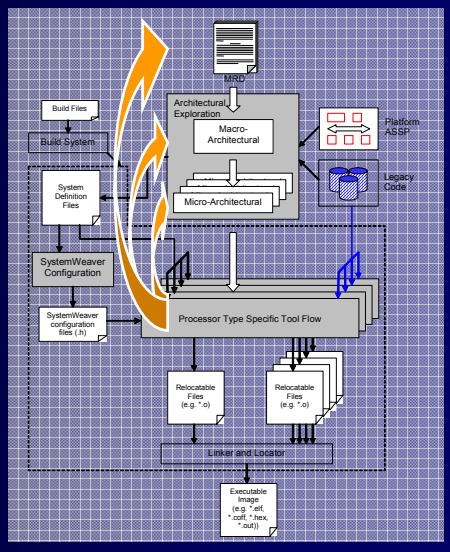


Build process



Top-down design flow

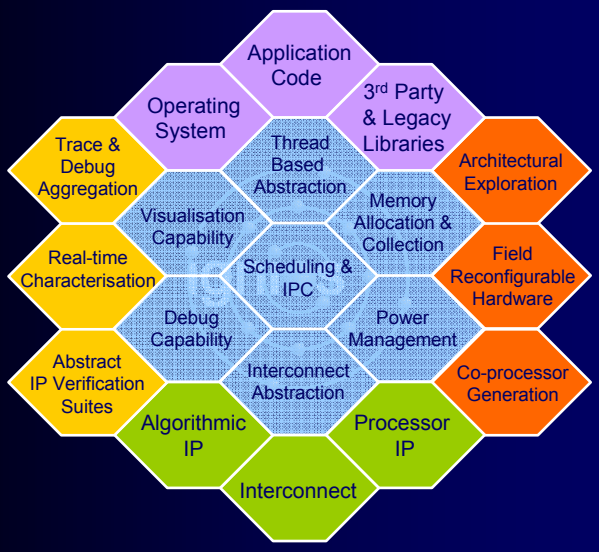
1. Review requirements
2. Explore architecture
   - Hardware defined?
   - Legacy software/libraries?
3. Define macro-architectural partition
4. Define micro-architectural partition
5. Create code
6. Compile
7. Link
8. Execute



Agenda

- Introduction
- Comparing programming models
- An efficient thread based abstraction layer
- A pragmatic programming model
- Conclusions

Ecosystem



Conclusions

- The programming model is the key barrier to *widespread* success for MPSoC
  - A balance between efficiency and abstraction
- A thread-based programming model could provide a means for application developers to realise the full potential of multi-core systems
  - Needs efficiency
  - Needs to be minimally disruptive (hardware, software, ecosystem)
- Efficiency *can* be delivered together with abstraction when using an optimised, flexible, hardware solution
  - Compatible with existing techniques and technologies
  - And also providing numerous other benefits

Thank-you