Extensible Processors for MPSoC

David Andrews
Computer Engineering Group
University of Paderborn

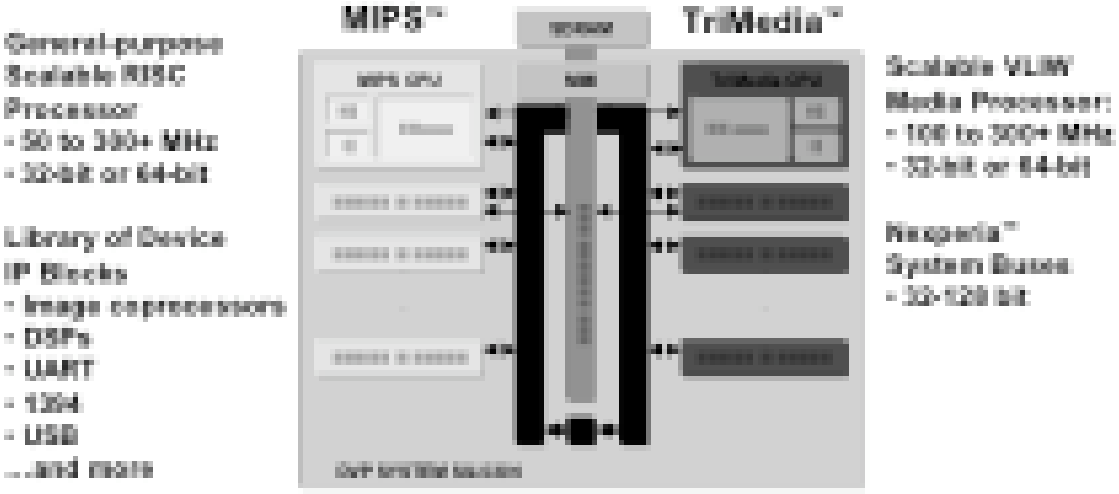
dandrews@ittc.ku.edu

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

MPSoC Example Architecture

- Let's Look at the Processing Resources…

A MPSoC Example: Nexperia™ DVP

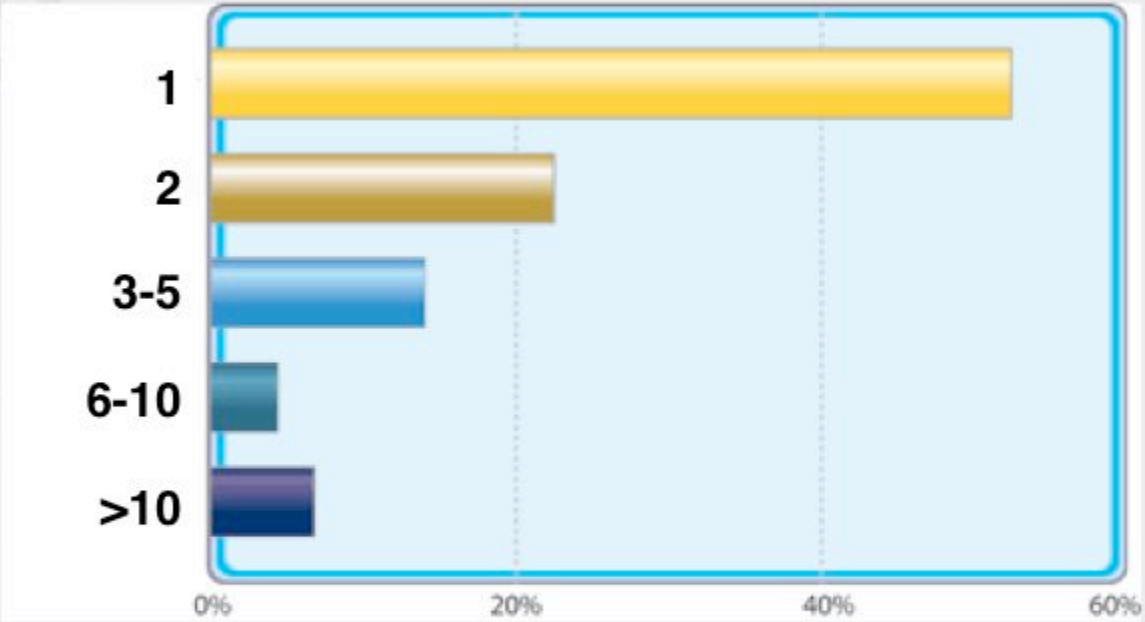


MPSoC Processors

- How Many and What Type of Processors?
- Requirements From
  - Performance
    - Throughputs, Turnarounds
  - Instruction Sets (Operations)
    - Operations
      - Arithmetic
      - Data Transfer
      - Special Custom
  - Flexibility
    - Programmability
    - Re-use
  - Cost
    - Parts Cost
    - Development Costs
  - Supporting Development Environments
    - Compilers, Debuggers, Run Time Systems
  - Power/Size Constraints
- Always The Classic Tradeoff: Customization versus Specialization
  - From : Cheap General Purpose Microprocessors
  - Through : Semi Custom (Extensible) Microprocessors
  - To: Fully Custom ASIC's

Multiprocessor Implementations

Emb. Systems Prog. Survey 2005: Number of Processors per chip

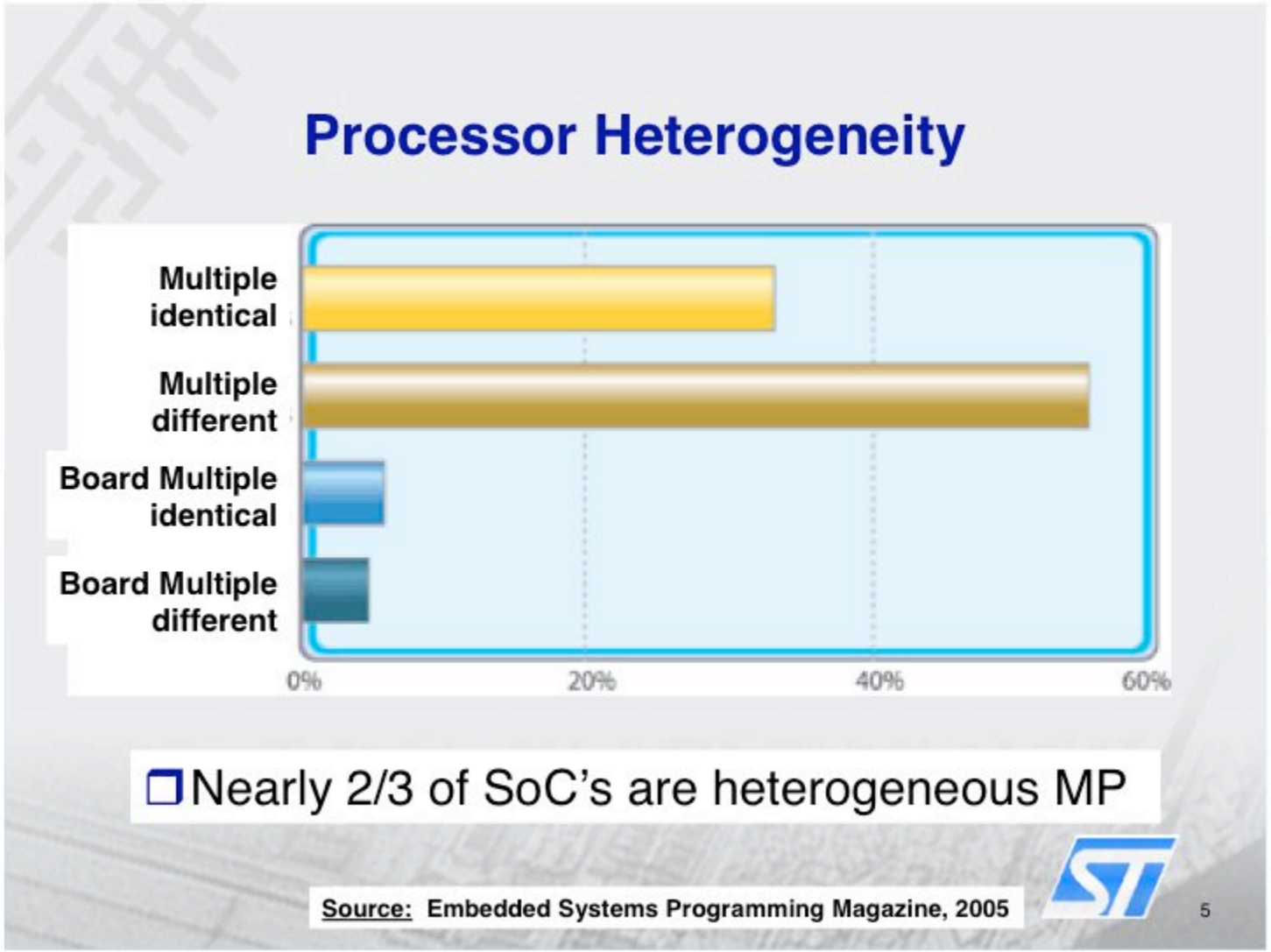


- Nearly 50% of chips use multiple processors
- Over 100 projects used >10 processors

Source: Embedded Systems Programming Magazine, 2005

4

Heterogeneity

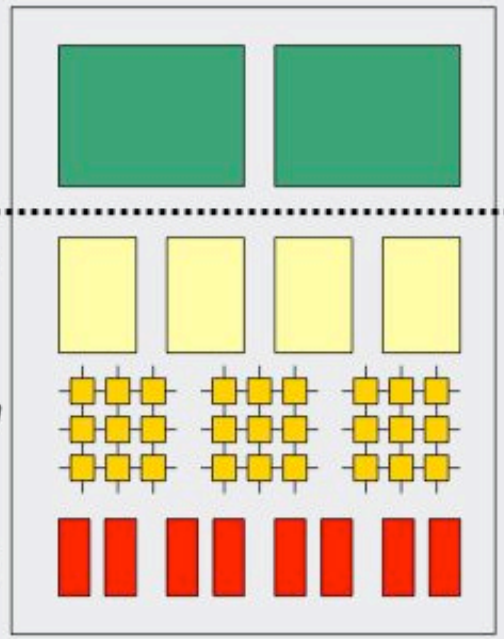


Why Heterogenous Solutions ?

- Applications Requirements Dictate:
- General Purpose
  - System Interface
  - RTOS Host
  - General System Processing
- Semi/Fully Custom
  - Network and I/O Controllers
  - Signal/Image Processing Data Paths
  - Support Large Data Transfers

CPU Generalization/Customization Tradeoffs

Subsystem Optimization

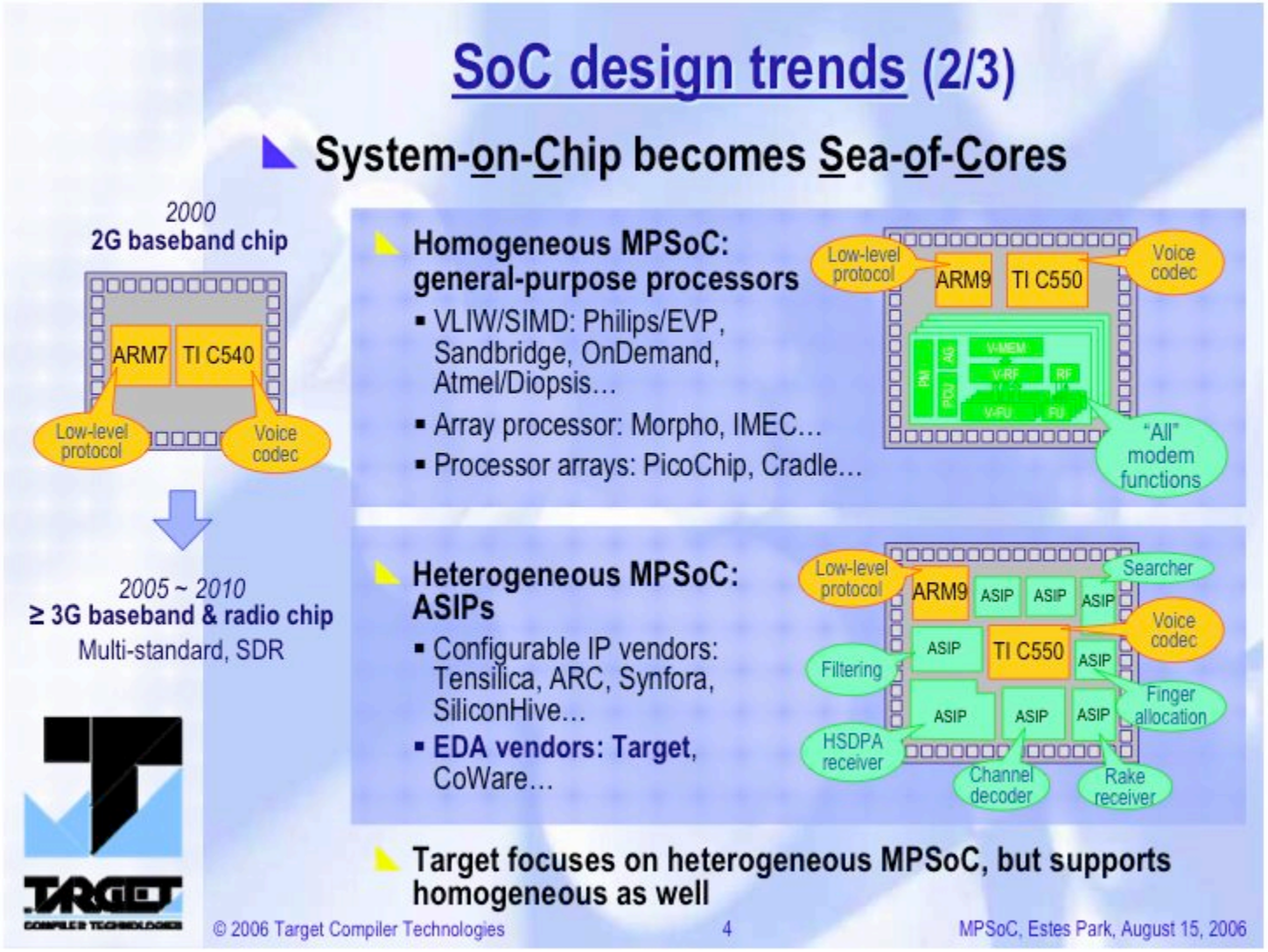


	Freq MHz	Gops/ mm^2	Efficiency %	Effective Gops
GP Core, Host	1000	1	20%	0.2
Application Oriented Cores	400	2	40%	0.8
Coarse grain Reconf.	150	5	80%	4
Specialized HW	150	15	100%	15

2 orders of magnitude

- Heterogeneity essential to obtain efficient platforms

MPSoC Heterogenous sysetm of IP's



A Little Caution

- What We Are Considering Is Largely Acceleration of a Portion of a Single Application.
  - Programming Language Analysis
  - Custom Processor Has 1 Program Counter
    - Although Data Paths Will Be Custom, Still 1 execution stream
    - Amdahls Law Applies

- Not Programming Model Acceleration (Yet)
  - Operating System to “Bind” All Assets Together
  - Programming Model to Delimit Independent Execution Streams

GP Processors

PROs:

- Good Scalability/Portability
  - Software Easier to Develop/Expand/Port
  - Easily Reprogrammable
- Economics
  - Legacy Software Development/Debug Environments
  - Cheap Components with Low NRE Costs

CONs:

- Low Performance
  - Data Paths, Control Paths Generalized, Not Tuned to Anything
  - Sequential, Limited ISA

Custom Circuits (ASIC's)

PROs:

- High Performance
  - Custom Data Paths, Control Paths Tailored to Your Application
  - Tuned Clock Frequencies, Delays etc.
- RTL Design
  - Low Level Descriptions in VHDL/Verilog
  - Synthesis Tools Immature
  - Verification Requires Long Cycles

CONs:

- Poor Scalability
  - Custom Data Paths, Components Designed for Specific Application Sizes.
    - 8 x 8 Image Filter Size Using Custom Tapped Delay Lines
- Economics
  - Costly NRE
  - Lack of Software Development/Debug Environments
  - Life Cycles/Reuse Limited

Extensible Processor Alternatives

What we would Like is to merge best of both worlds

- General Purpose Microprocessors
  - Reprogrammability: Reuse, Debug/Development
    - Flexibility
- ASIC's
    - Performance Level of Customized Solutions

- Todays (Lectures) Answer: Extensible (Configurable) Processors
  - Start With Familiar/Standard Computational Models.
    - PC, SP, Register File, ALU's, Decode Units
    - Extend with Mix/Match of Custom Components
      - Wider Data Paths
      - Wider/More ALU's
      - Specialized Operations
    - Reflect Extensions through Op_Codes

- Exploit Existing Compiler, Debug Environments

Key Questions for Extensible Processors

- What Target Characteristics of the Processor Can be Configured and Extended
  - Data Paths
  - Registers
  - Pipeline Stages
  - Data Movement
- How Does System Engineer Capture Target Characteristics
  - Design Tools
  - Profiling
  - Instruction Building
- What are Deliverables: Hardware and Software Components
  - New Compiler/Linker
  - Debuggers
  - RTL Generation

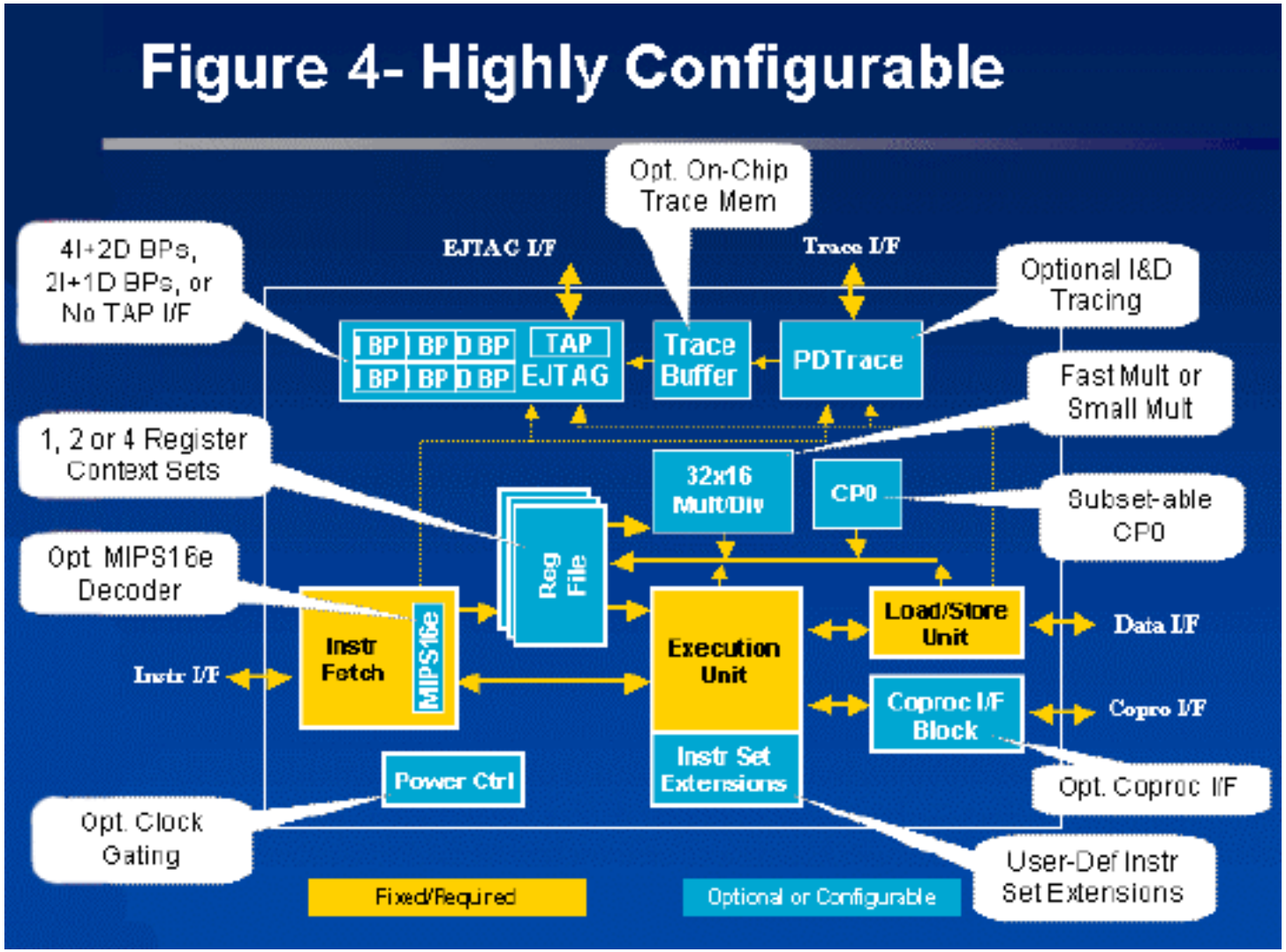
Processor Configuration Criteria

- Configuration Mechanism Must Accelerate and Simplify the Creation of Useful Characteristics
  - Can't Simply Be More Bureaucracy
    - Cost - Performance Ratios Also a Consideration
  - Usually Requires Significant Program Analysis
    - By Hand
    - Compiler Assisted
- Generated Processor Should Include:
  - Complete Hardware Descriptions
    - Synthesizable Verilog/VHDL Descriptions
  - Complete Software Development Tools
    - Compilers
    - Debuggers
    - Assemblers
    - RTOS's
  - Verification Software
    - Simulation Models
    - Diagnostics
    - Test Benchs/Support

Selected Range of Offerings

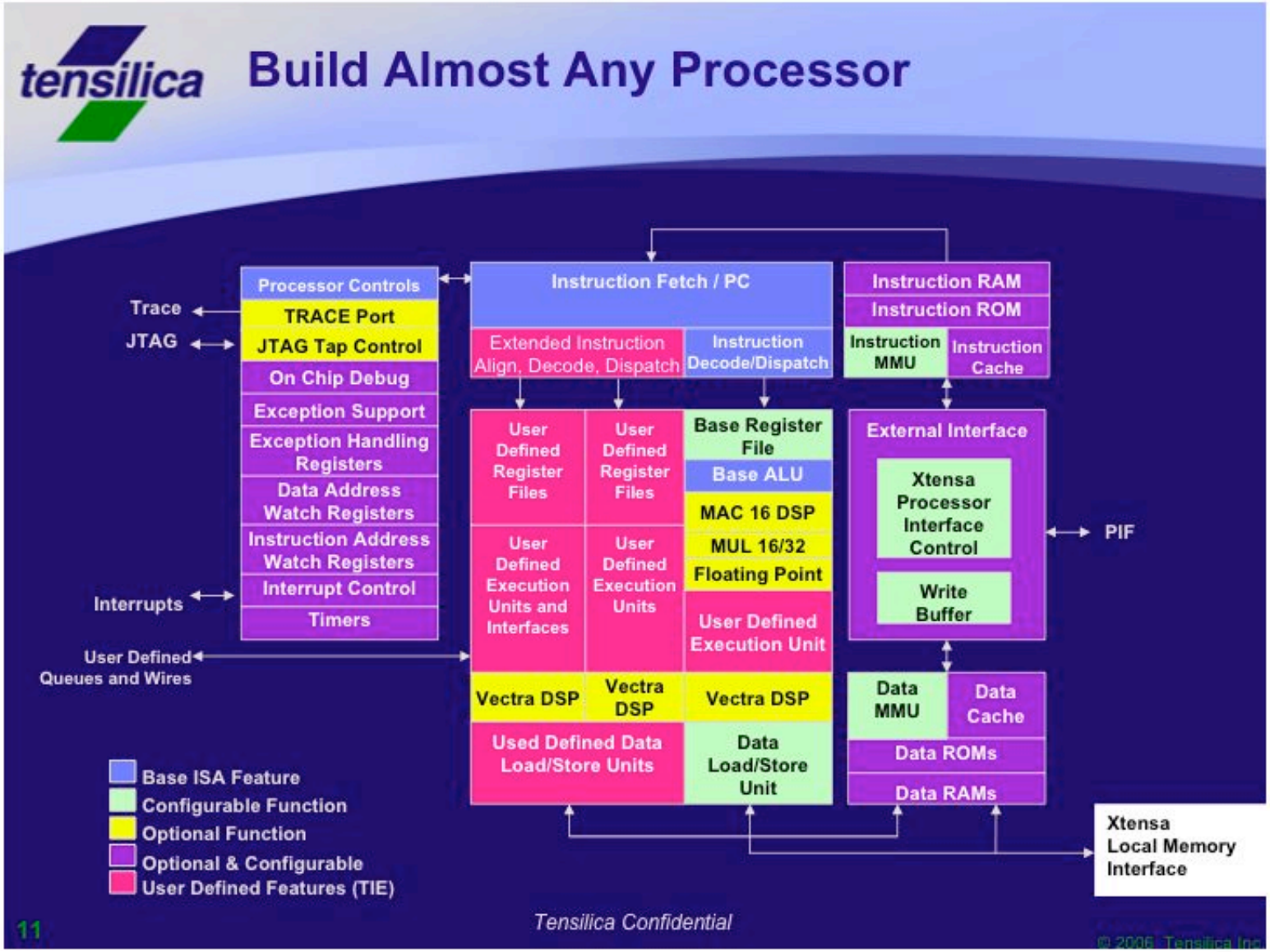
- Non-Architectural Processor Configuration
  - Not reflected within the ISA
    - Cache Sizes, DMA's
- Fixed Menu Processor Architecture Configurations
  - Preset Range of Features From Menu's
    - Hw/Sw tools configured in parallel (hopefully from 1 user interface)
- User-Modifiable Processor RTL
  - Processor Has Hardware Interface for Hand Addition/Modification of Instructions.
    - Generally Precludes Software Support from Compiler/Simulator/RTOS
    - MIPS M4K
- Instruction-Set Description Language
  - Automated Processor-Generation Tool Starts from ISA and Builds Silicon (RTL Descriptions) and Software Support (Compilers/Simulators)
    - Tensilica
- Fully Automated
  - Compilation/Synthesis Tools Analyze and Profile Applications and Generate Custom Everything

Example: M4K Core (MIPS Technologies)

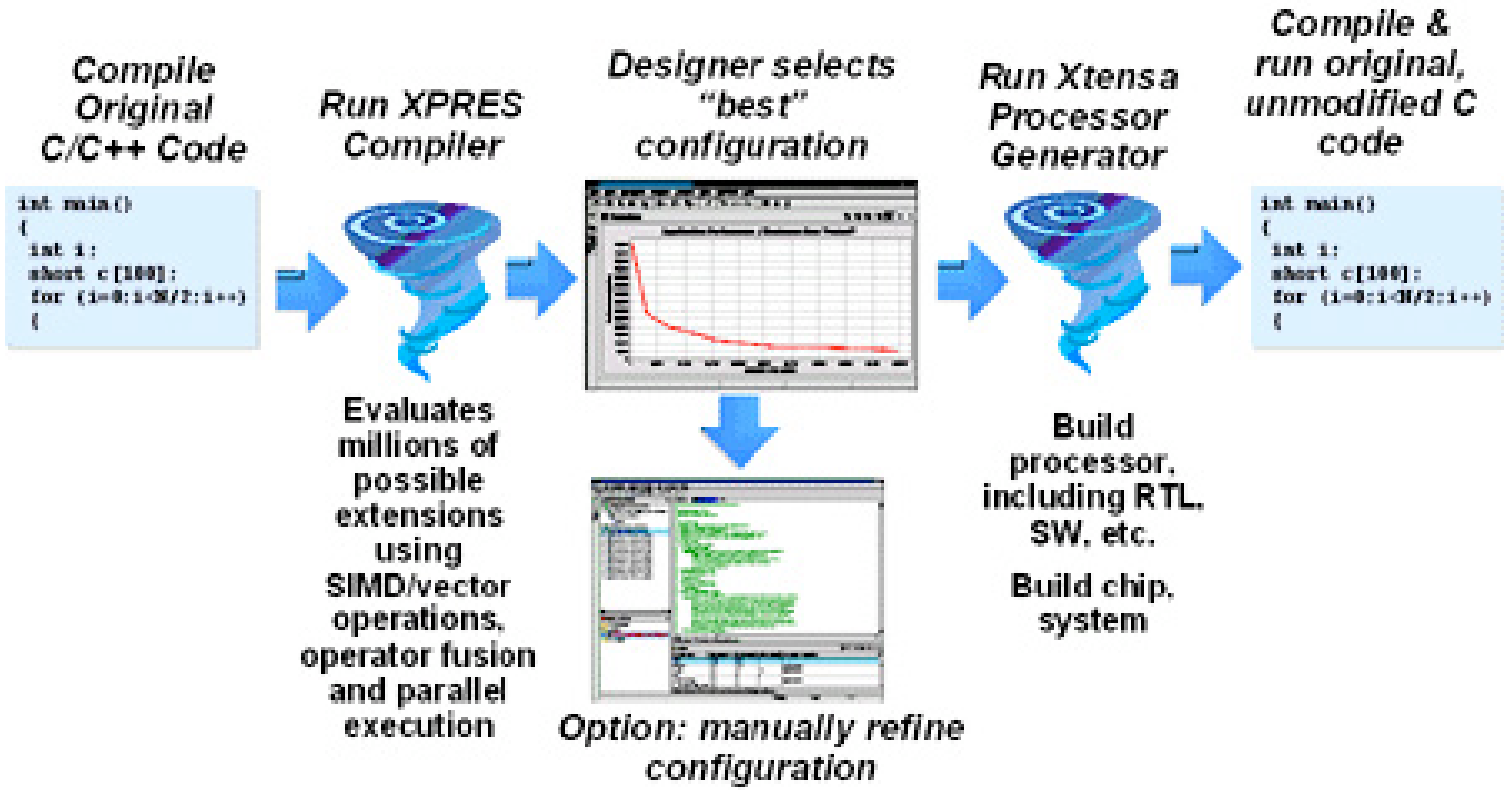


www.mips.com/content/PressRoom/TechLibrary/WhitePapers/multi_cup

Tensilica's Configurable Core



Tensilica Automatic Processor Generation



Development Tool Flow

- Several Interesting Options
  - Start With Unaltered C/C++ Code
    - Profile/Analyze
    - Automatically Generate Core
  - Create Custom Instructions
    - TIE a C/Verilog Language
  - Both Create “Updated” Tools
    - Compiler, Simulator, RTOS

Modifications

- Fusion
  - Identifies Instructions that can be combined

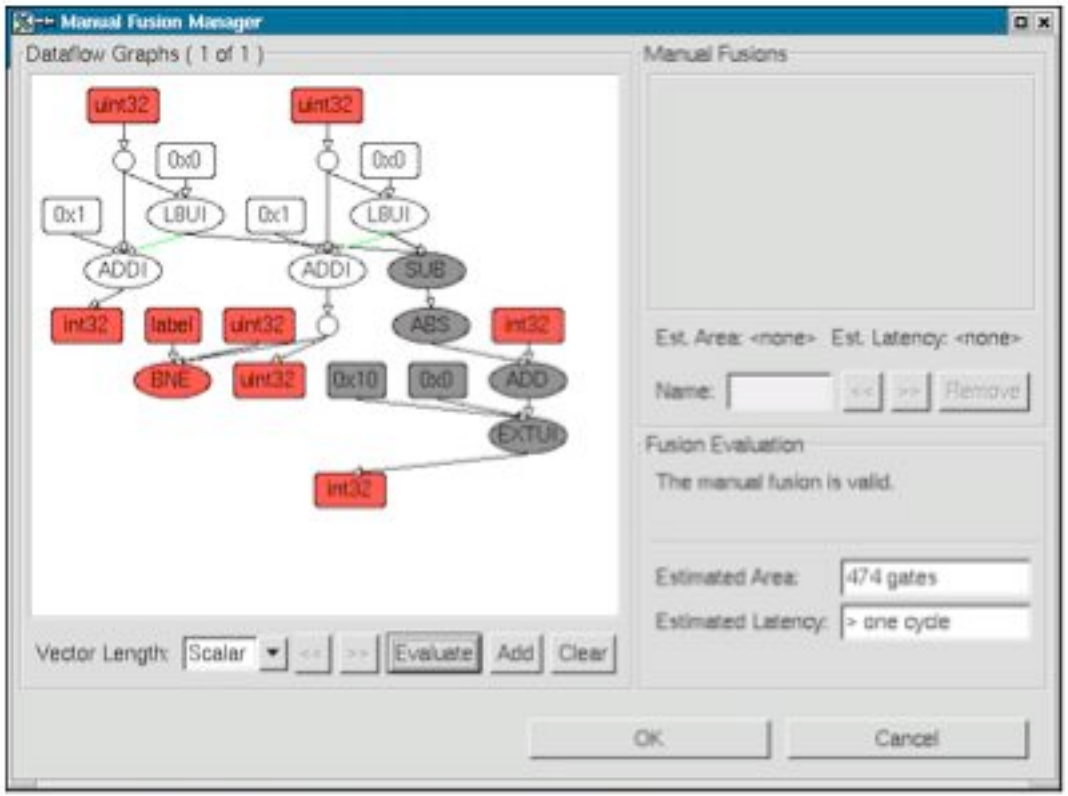
    Add R1,R2,R3

    Sll R1, R1, ##4

    Create: Add_sll R1, R2, R3, #4 /* 1 clock cycle instruction
- Vector/SIMD
  - Best Bet for Parallelization Using this Method
    - Attacks Loops: Unroll and Create New Wider Register File + ALU's of Depth 2, 4, 8
- VLIW: Called "Flix" (Flexible Length Instruction Xtensions)
  - 32 or 64 bit VLIW Instruction:
    - Can be multicycle

Fusion Example

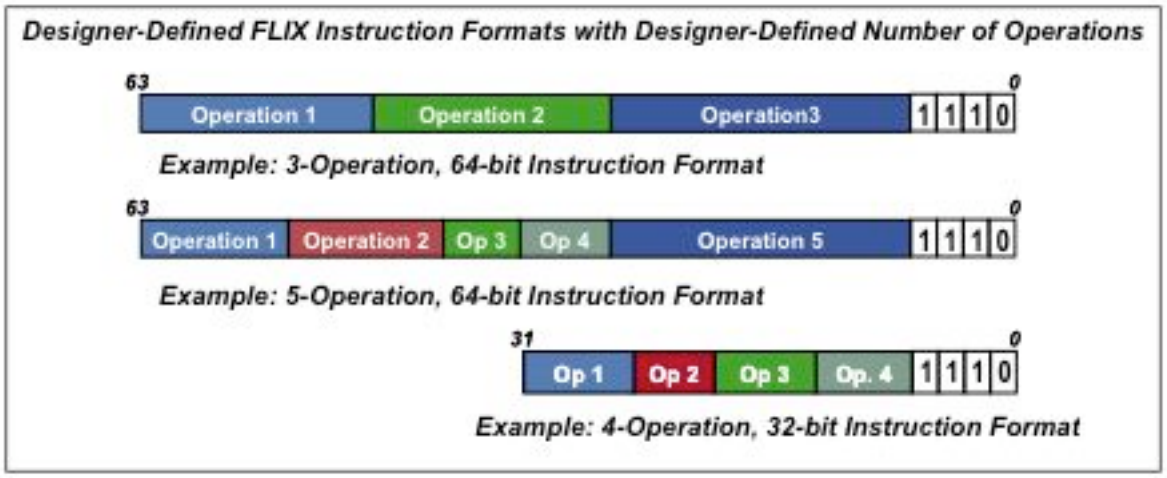
- Compiler Identifies Based on Dependencies and Frequency Counts (I.e. loops)
- sub,abs,add,extui can be combined into a single instruction
- 474 gates in 1 cy



This dataflow graph generated by the XPRES Compiler shows a series of operations marked as fusible.

Flix Example

- VLIW Packing of Instructions
  - Dependency Analysis
  - Long Instructions Issued In Sequence
  - Can Contain Fusion, SIMD Instructions



TIE Language

- Compiler Identifies Some Parallelism and Automatically Creates New Instructions/Architectures in “TIE”
- User Can Also Operate In TIE

- Tie: Tensilica Instruction Extension Language
  - Allows The Creation of New Custom Hardware Through ISA
  - State Declarations: Can Add State Registers and Register Files
  - Instruction Encodings and Formats - Operation Descriptions: Can Have Up to Six Source and Destination Operands:
    - GP Registers
    - Newly Defined Registers
    - New States
- TIE Feeds Back New Instructions/Types to Preprocessor Within C Compiler Chain

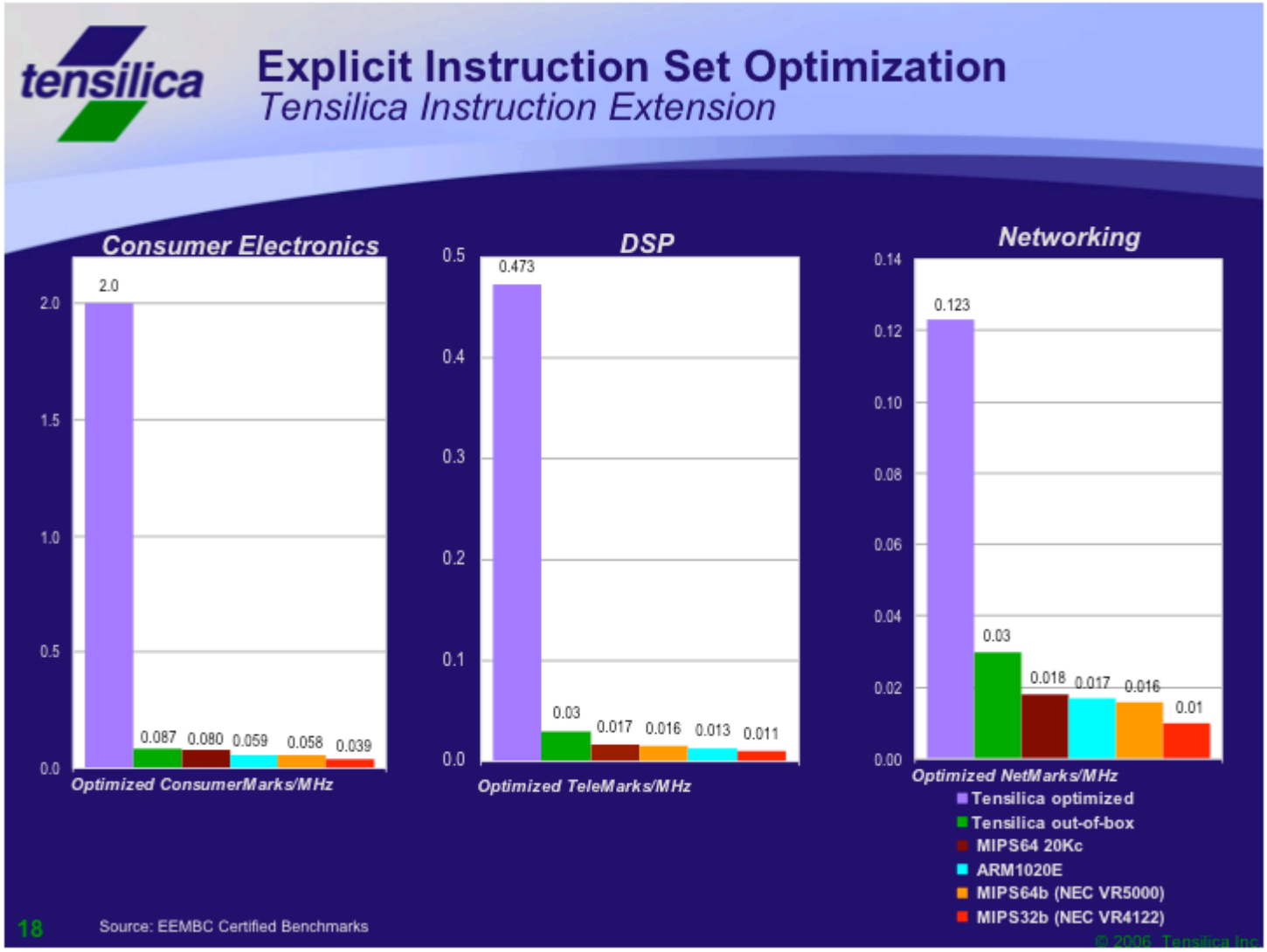
Example

Processor Generator Creates new Compiler with new data type LR
-will also generate new ld/st operations for this type

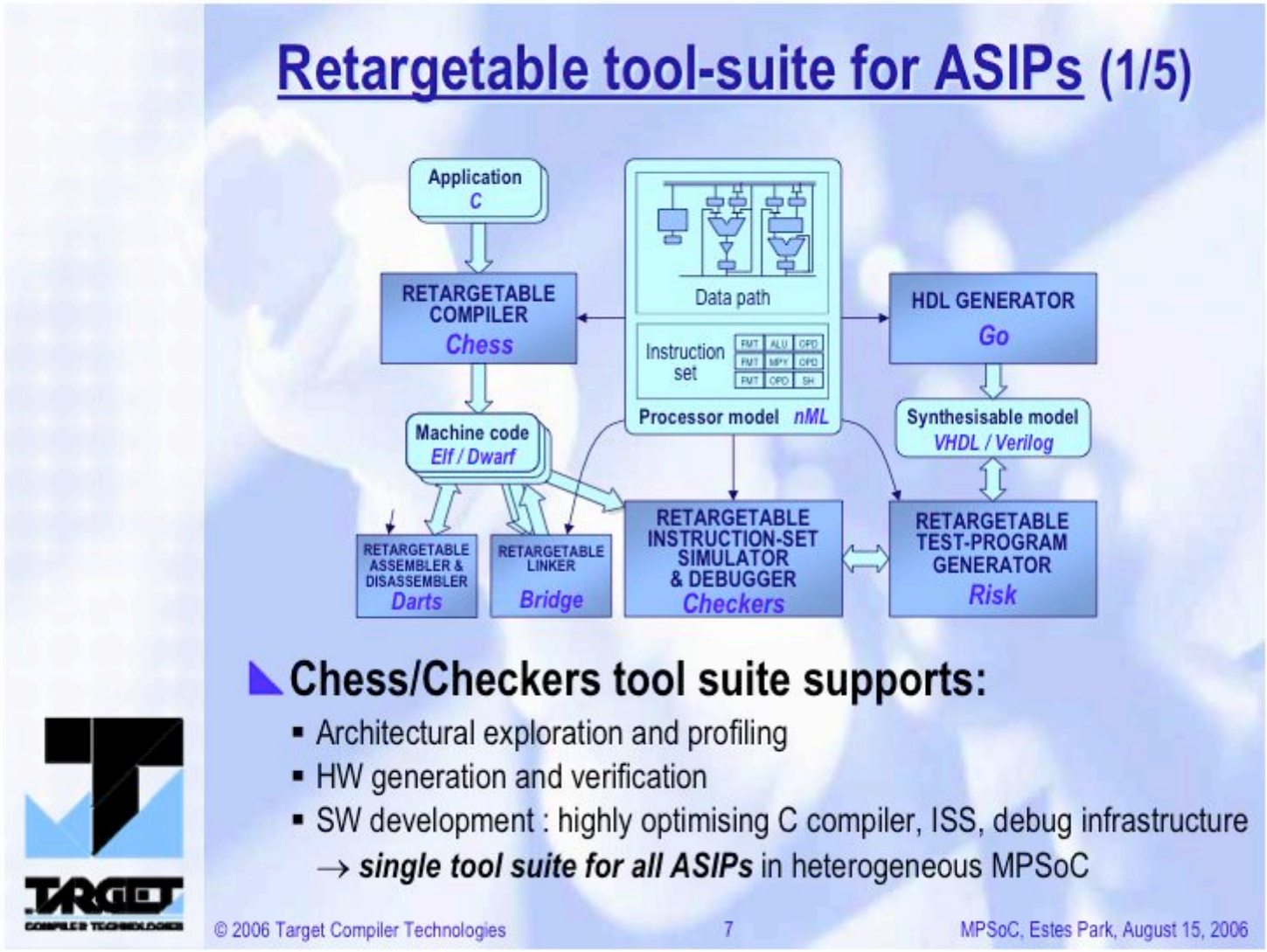

```
Regfile LR 16 128 1
Operation add128 {out LR sr, in LR ss, in LR st} {assign sr = st + ss;}
```

```
Main() {
int i;
LR src1[256], src2[256], dest[256];
    for ( i=0; i< 256; i++ ) dest[ i ] = add128(src1[ i ],src2[ i ]);
}
```

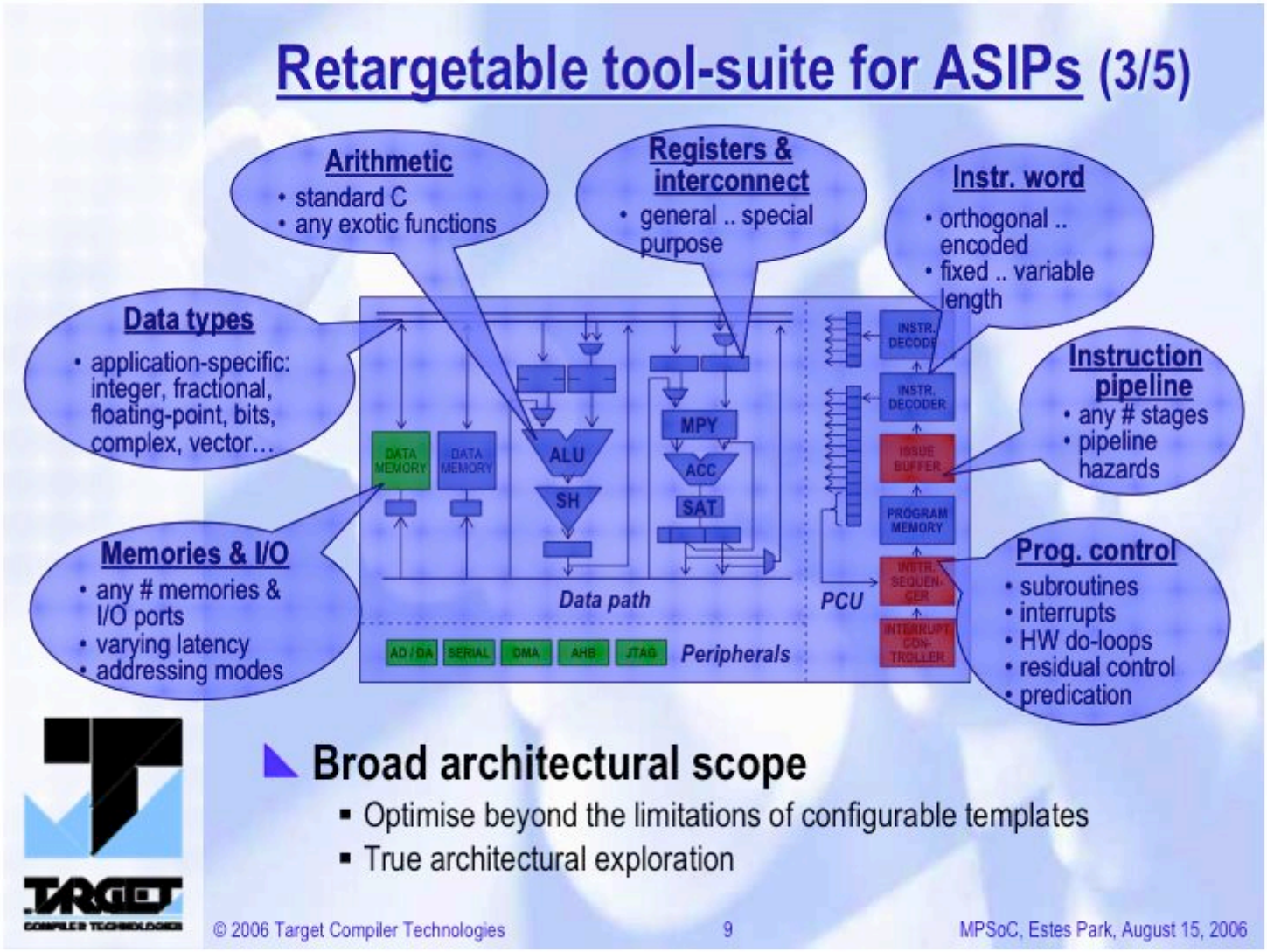
Performance



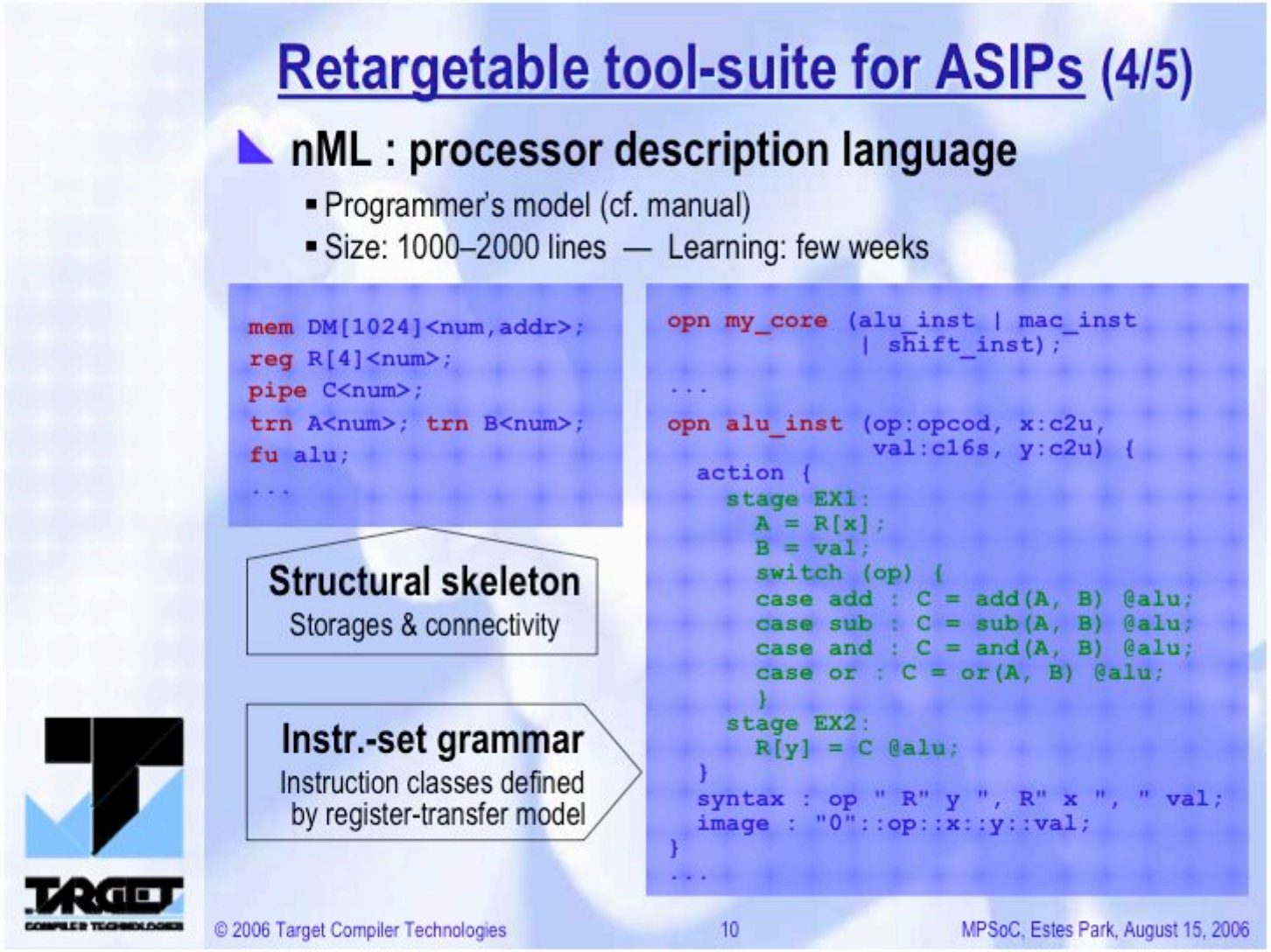
Another Example: Chess/Checkers



Configuration Capabilities



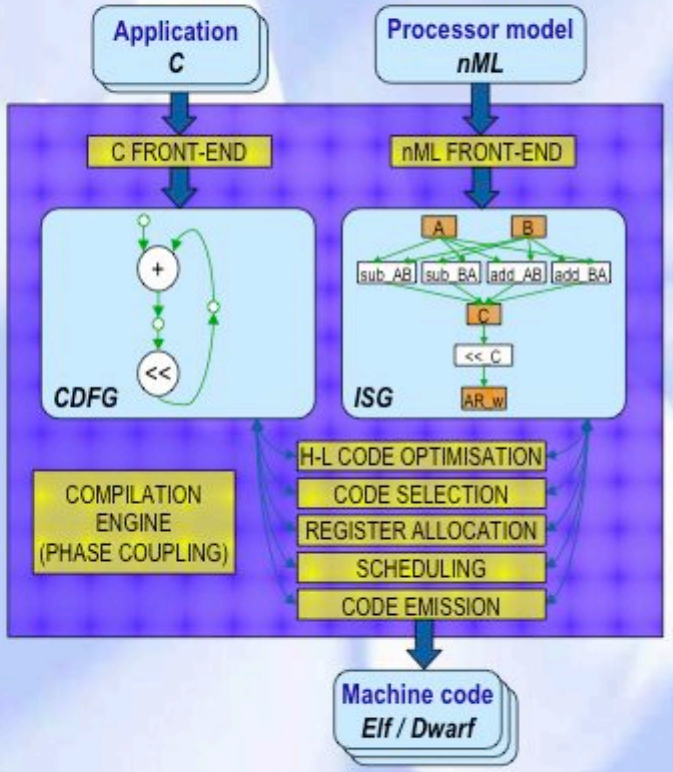
Processor Description Langauge nML



Chess

Retargetable tool-suite for ASIPs (5/5)

Chess : graph-based C compiler



- **Front end**
  - C → Control-Data Flow Graph
  - nML → Instruction-Set Graph
- **Compilation phases**
  - Map CDFG onto ISG
  - Graph algorithms
- **ISG contains structural info**
  - E.g. hardware resources, data-types, connectivity, instruction encoding, pipelining, parallelism, pipeline hazards
  - Much closer to hardware than conventional compilers (e.g. gcc)
  - Enables efficient compilation for "irregular" architectures
  - Patented