

Design and Architectures for Embedded Systems (ESII)

Prof. Dr. J. Henkel, Dr. M. Shafique

CES - Chair for Embedded Systems

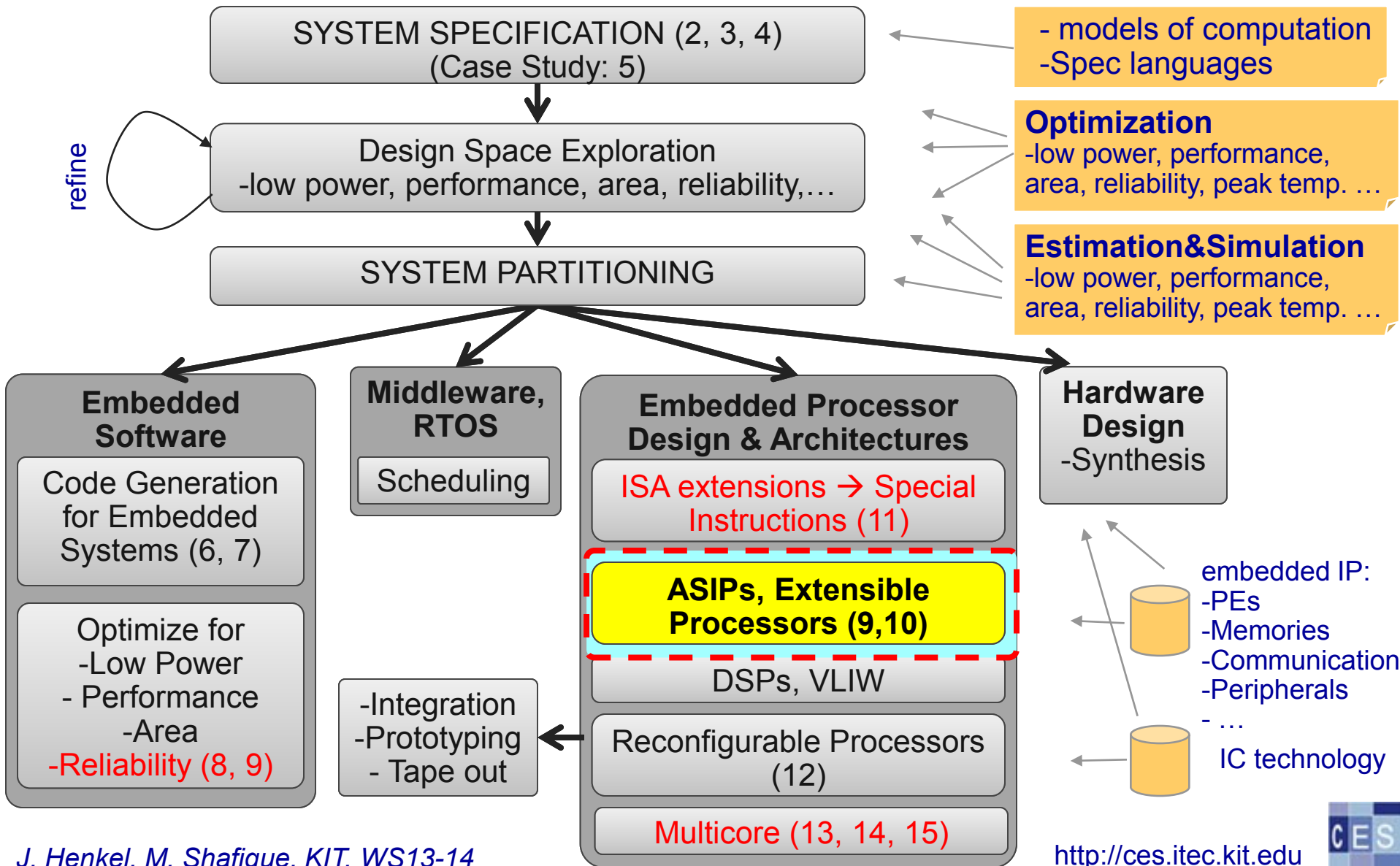
Karlsruhe Institute of Technology, Germany

Today: Embedded Processor Platforms

ASIPs and Extensible Processors

Where are We?

Introduction to Embedded Systems (1, 2)



Outline

❑ Introduction

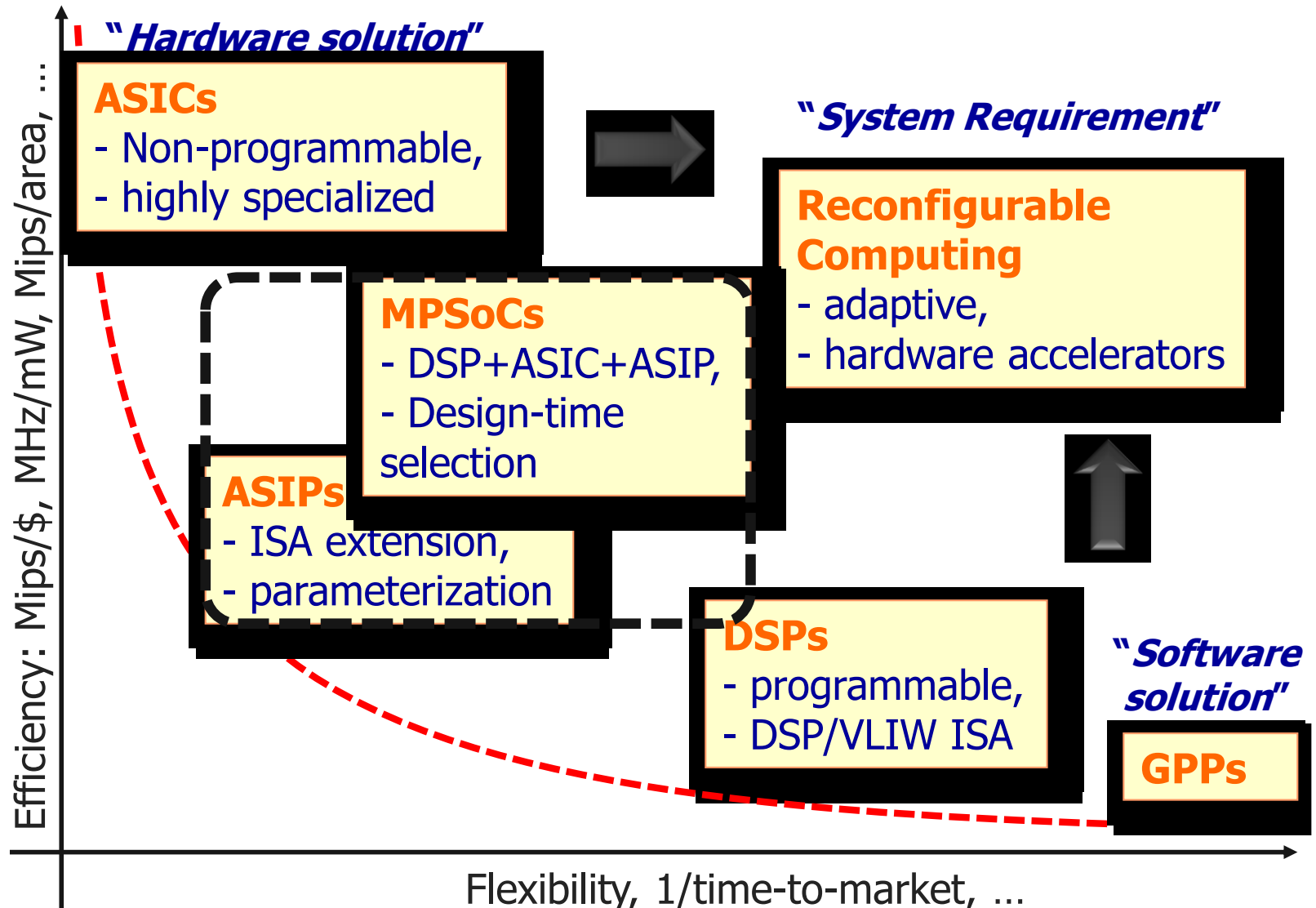
❑ Platforms

- ❑ Tensilica's Xtensa
- ❑ LisaTek (CoWare)

❑ Backup Slides

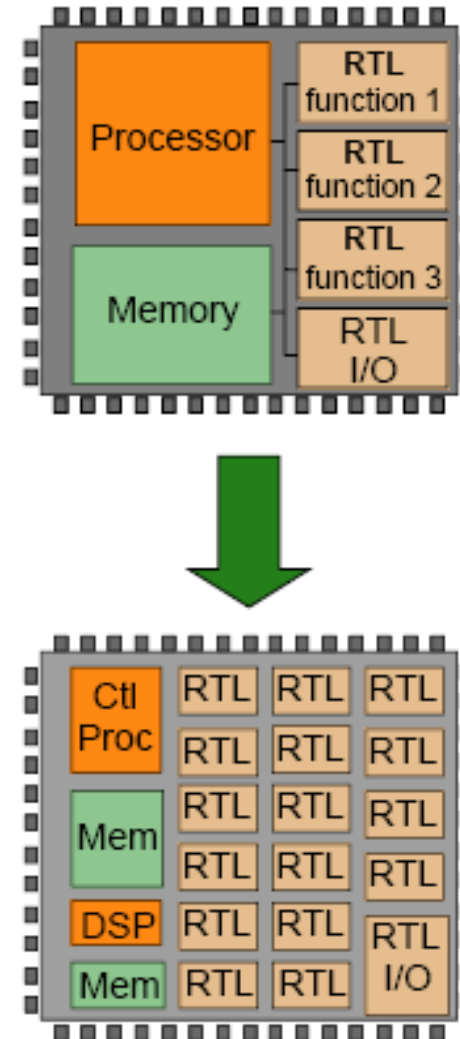
- ❑ Improv Platform
- ❑ HP's Pico Platform

Architectures: Options and Tradeoffs



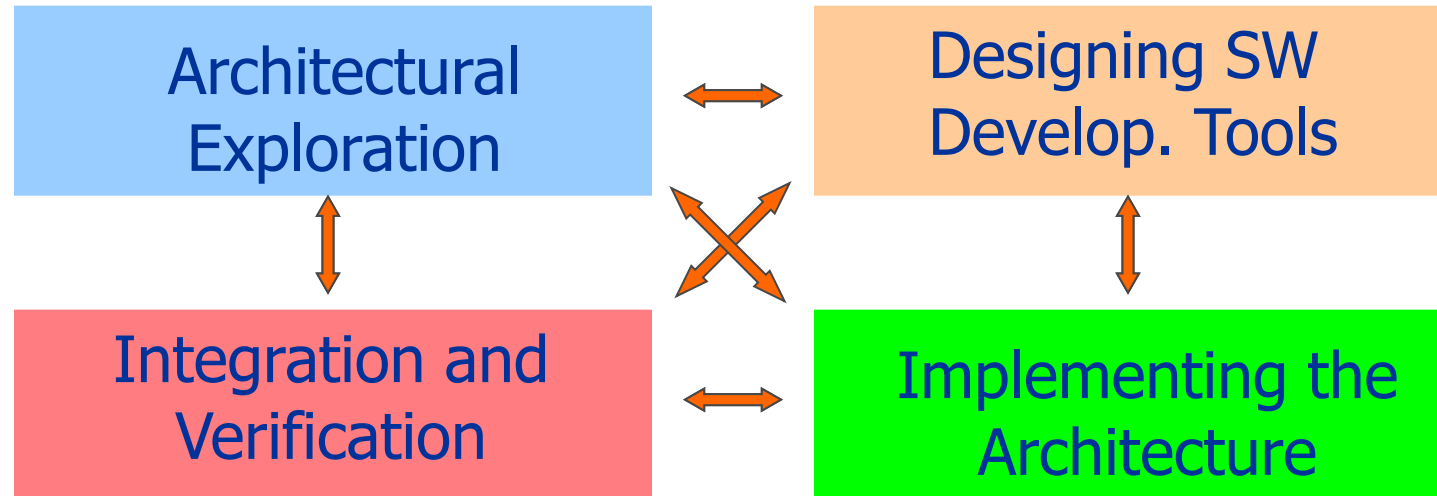
The Problem with RTL

- ❑ Rapidly increasing number of transistors require more RTL blocks on chip
- ❑ Hardcoded RTL blocks are not flexible
- ❑ Hand-optimized for application specific purposes



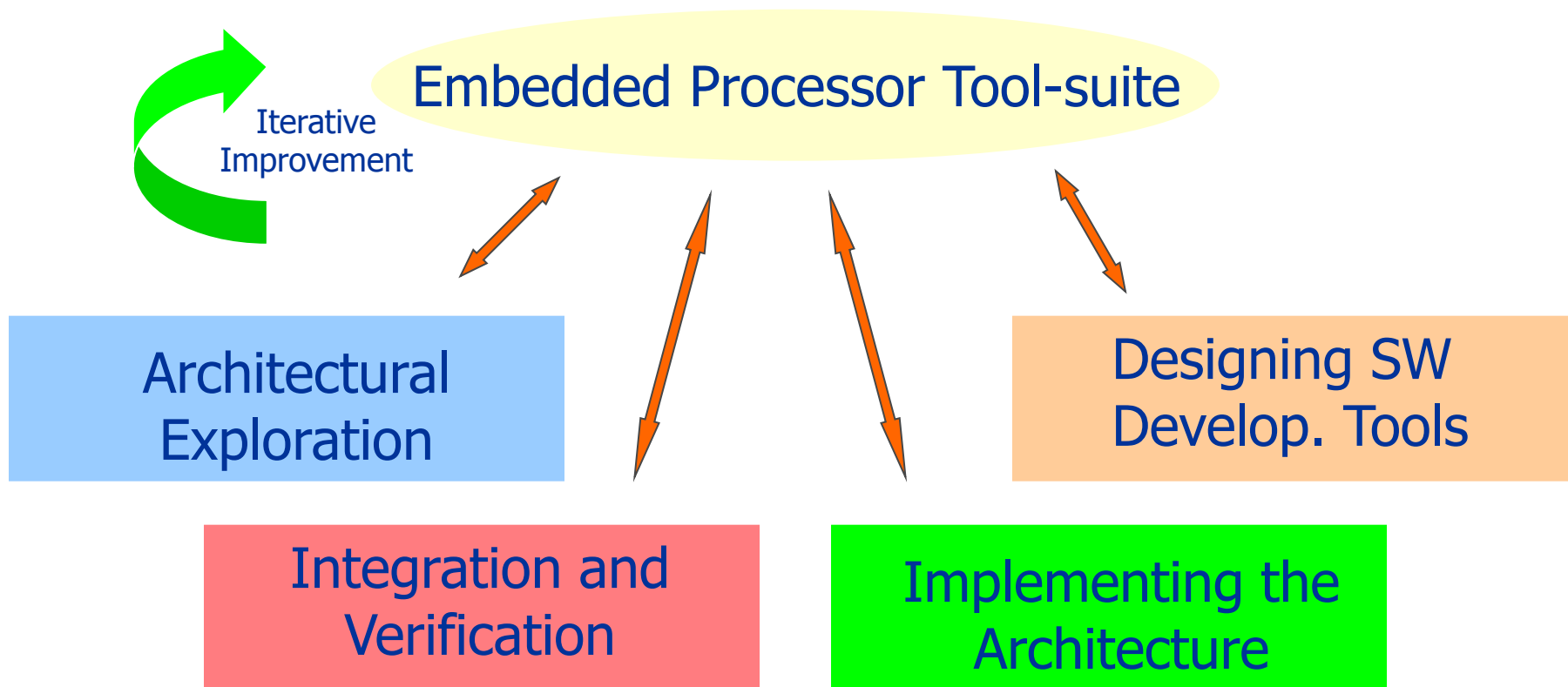
(source: Tuan Huynh, Kevin Peek & Paul Shumate
Advanced Processor Architecture)

Designing an embedded Processor: tasks



- ❑ Tasks are inter-dependent
- ❑ Improvement through iteration
- ❑ Each task is customized for one specific implementation of an embedded processor
- ❑ Many steps are manual since it is a one-time effort
- ❑ But product life times are short: can these tasks be combined and automated ?

Designing an embedded Processors: the alternative way



- ❑ There is only one generic tool-suite that generates all other parts: -> a) min. manual support b) higher flexibility c) re-use for next-generation embedded processor
- ❑ Iterative improvement is done without manually re-designing the tools

Designing a customized embedded processor: approaches

❑ Instruction set:

- ❑ Fully customized instructions (no predefined); but the instruction set might be domain-specific (e.g. DSP-type)
- ❑ Core instruction set is fixed; the instruction set can be enhanced:
 - ❑ The “bottlenecks” of an application are hard-wired as application-specific instructions (might be re-used, e.g. FFT, but might be specific to one application only); tool-suite provides a language to do define these instructions

❑ Processor components:

- ❑ The basic (general) core can be enhanced by pre-defined, fixed, specialized cores: e.g. a DSP core

❑ System components (to be added/omitted and parameterized):

- ❑ A) on-chip cache: size, policy, ...
- ❑ B) MMU
- ❑ C) ...

❑ On-Chip communication infrastructure:

- ❑ Busses, hierarchical buses (processor core, inter-core, peripheral) -> typically fixed

ASIP Design Technologies

❑ ADL Based (ASIPs from Scratch)

- ❑ Higher degree of flexibility, efficiency → Higher Design Effort
- ❑ LISATek (CoWare→Synopsis), Target, Expression

❑ Extensible/Configurable Processors

- ❑ Pre-Defined Base Core → Well tested
- ❑ Extended/Customized via Special Instructions (Instruction Set Extensions)
- ❑ Parameterizable → Function Blocks
- ❑ Tensilica, etc.

❑ Reconfigurable/Adaptive ASIPs/Extensible Processors

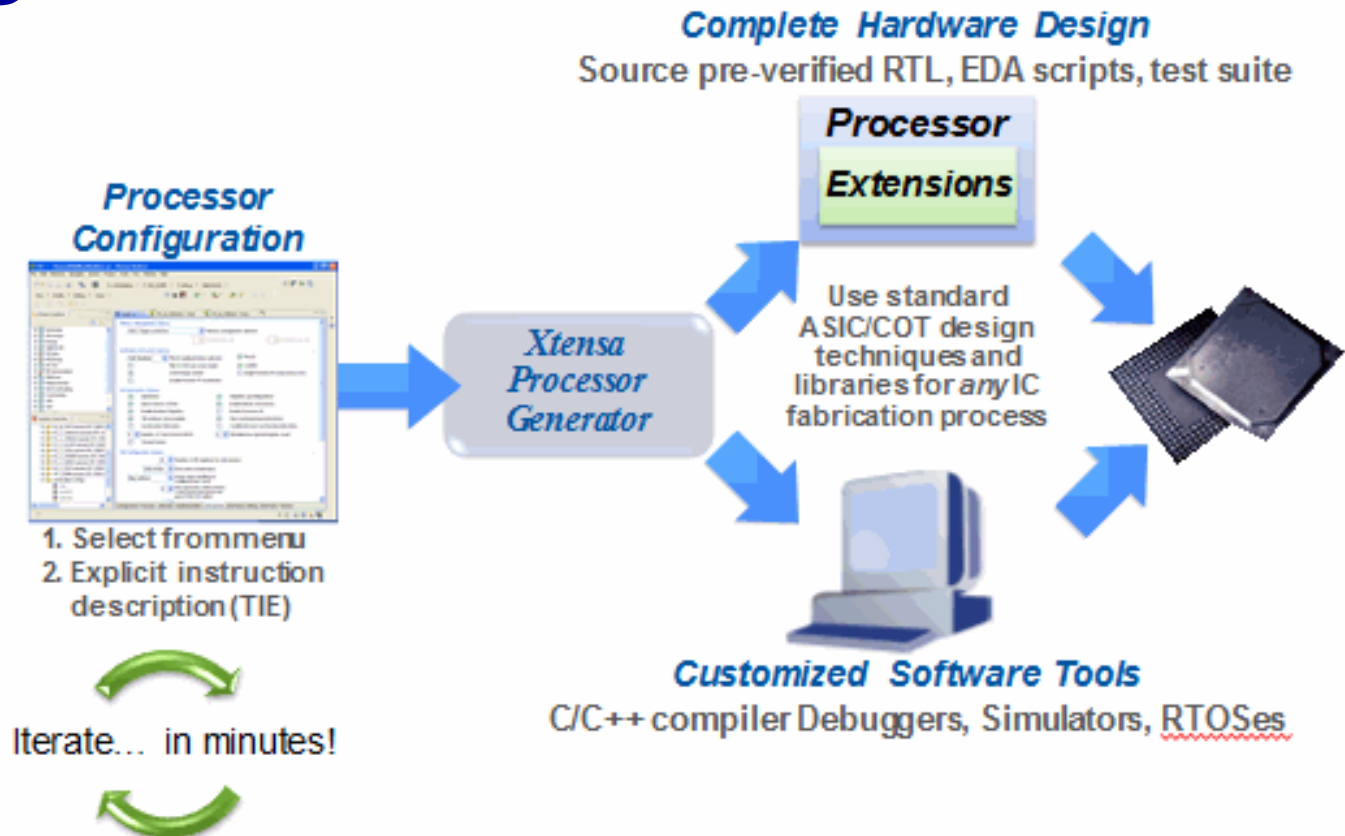
- ❑ Stretch → Using Tensilica Xtensa
- ❑ Research Projects
 - ❑ RISPP@CES, KIT: Bauer, Shafique, Henkel + Students
 - ❑ rASIP@Aachen: Leupers
 - ❑ Reconfigurable ASIP for communication: Wehn, TU Kaiserslautern


The Tensilica Platform

- ❑ Paradigm
- ❑ Xtensa Architecture
- ❑ Tensilica Design Flow
 - ❑ Hardware Development using TIE (Tensilica Instruction Extension)
 - ❑ Software Development
- ❑ Tools: Xtensa Xplorer
- ❑ Case Studies
 - ❑ Code Compression (Henkel, Lekatsas)
 - ❑ H.264 Video Encoder (Javed, Shafique, Parameswaren, Henkel)

Paradigm and Main Features

- * IP (cores)
parameterizable
- * TIE Instruction
Set Extensions
- * Customized
generated
Software tool flow



- ☐ Combines core-based design paradigm on the one side with ASIP features (application specific instruction set processor) on the other side
- ☐ User can adapt core parameters and define own instructions (if necessary)
- ☐  two levels of customization
- ☐ Status: commercial product

Example Phones with Tensilica DPU

- * Handset
- * Printers & Scanners; * Graphics (ATI Radeon, PowerColor Radeon)
- * Entertainment (Ninetendo 3DS, Sony, ...)
- * Networking; * Storage; * Wireless; * ...



(source: <http://www.tensilica.com/company/customer-profiles/>)

Xtensa

- ❑ 32-bit microprocessor core with a graphical configuration interface and integrated tool chain
 - ❑ Higher abstraction level for designing
- ❑ Configurable and Extensible
 - ❑ Add specialized instructions/functions to the core
 - ❑ Software development tool chain
- ❑ Basic Architecture
 - ❑ 5-stage pipeline with 78 instructions
 - ❑ 1 - load/store,
 - ❑ 32-entry orthogonal register file and 32 optional extra registers
- ❑ Processor Configuration
 - ❑ 170 MHz, 200mW, 0.25 μm , 1.5V
 - ❑ Cache: 16 KB I-cache, 16 KB D-cache, Direct mapped
 - ❑ 32 32-bit Registers, Extensible using TIE instructions
 - ❑ Others: No Floating Point Processor, Zero overhead loops

Xtensa LX –Architecture

❑ Basic Architecture: Processor Configuration

- ❑ 5-, 7-stage pipeline, Clock: 350, 400 MHz, Power: 76, 47 μ W/MHz
- ❑ Cache: up to 32 KB and 1,2,3,4 way set associative cache
- ❑ 64 32-bit general purpose and 6 special purpose registers
 - ❑ Optional Registers: 16 1-bit boolean, 16 32-bit floating-point, 4 32-bit MAC16 data registers, optional Vectra LX DSP registers
- ❑ 32-bit ALU, 80 core instructions (including 16- & 24 bit)
- ❑ 1, 2 Load/Store units
- ❑ Extensible using TIE and FLIX instructions
- ❑ Zero overhead loops

❑ General Purpose AR Register File

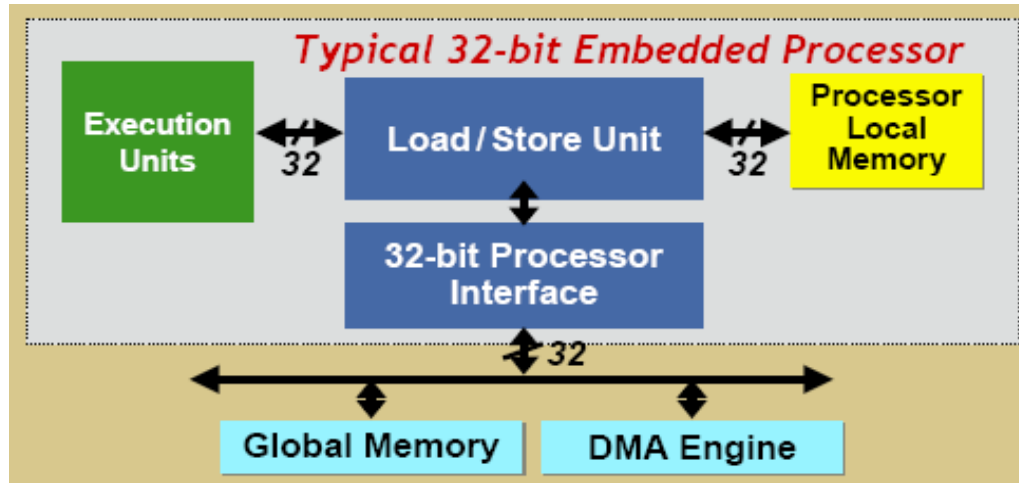
- ❑ 32 or 64 registers
- ❑ Instructions have access through “sliding window” of 16 registers.
- ❑ Window can rotate by 4, 8, or 12 registers
- ❑ Register window reduces code size by limiting number of bits for the address and eliminated the need to save and restore register files

Xtensa 8 vs. Xtensa LX3

	Xtensa 8	Xtensa LX3
Major ISA Configuration Options		
Max16	Yes	Yes
MUL16/MUL32	Yes	Yes
Floating Point	Yes	Yes
Vectra LX	Not Available	Yes
HiFi 2 Audio	Not Available	Yes
ConnX D2 DSP engine	Not Available	Yes
ConnX Baseband Engine	Not Available	Yes
Linux MMU	Yes	Yes
Pipeline/Architecture Options		
Pipeline Stages	5	5/7
FLIX Technology	Not Available	Yes
Processor Interface Options		
PIF and XLMI	Yes	Yes
Load/Store Units	One	One or Two
Designer-Defined Ports and Queues	Limited Config Options*	Yes

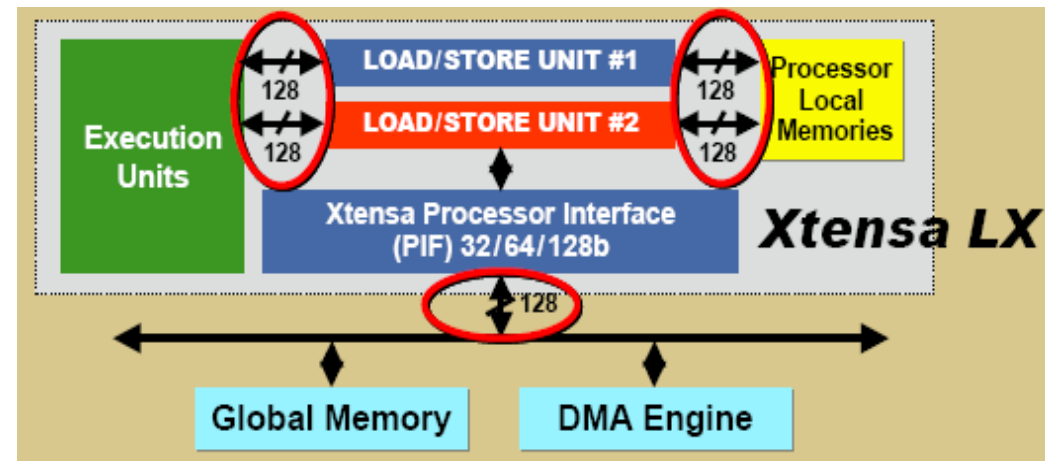
*Both processors have the GPIO32 option (two 32-wire ports) and the QIF32 option (two 32-bit queue interfaces).

Xtensa Benefits



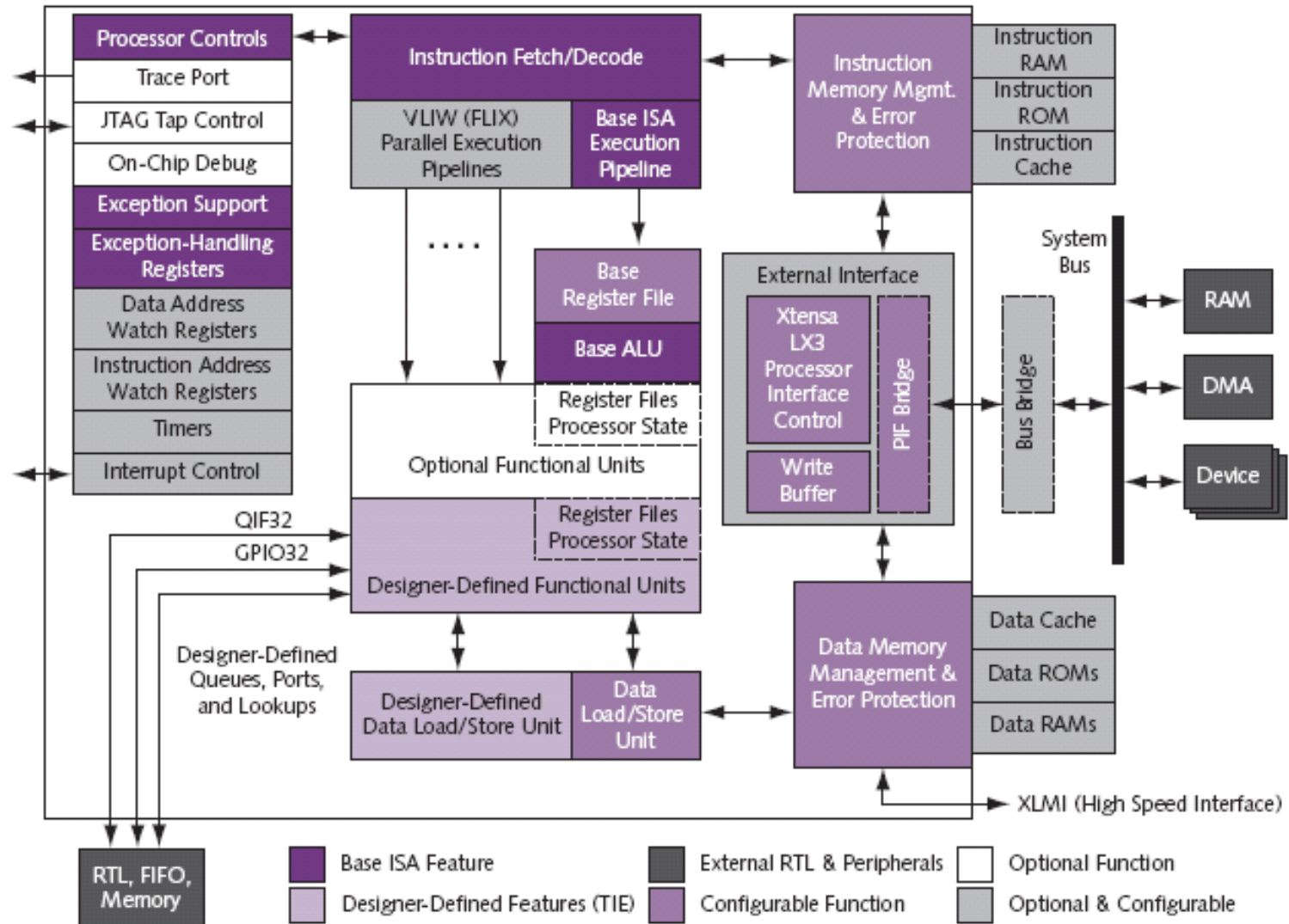
- ❑ 1 Operation / cycle
- ❑ Load/Store overhead

- ❑ Extra load/store unit, wide interfaces, compound instructions
- ❑ Up to 19 GB/sec of throughput



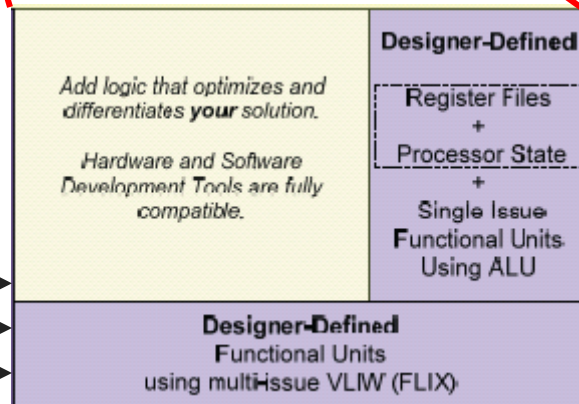
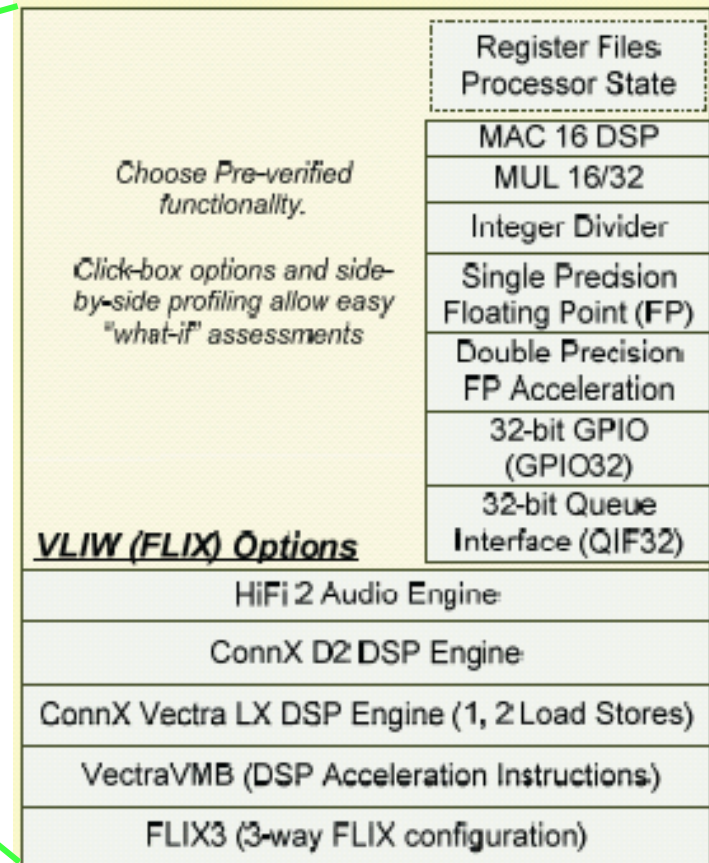
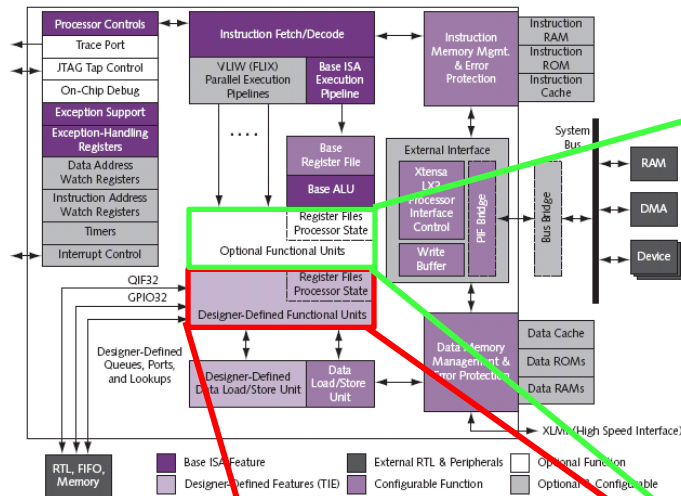
Xtensa LX3 Architecture

base ISA, configurable, optional, extensible



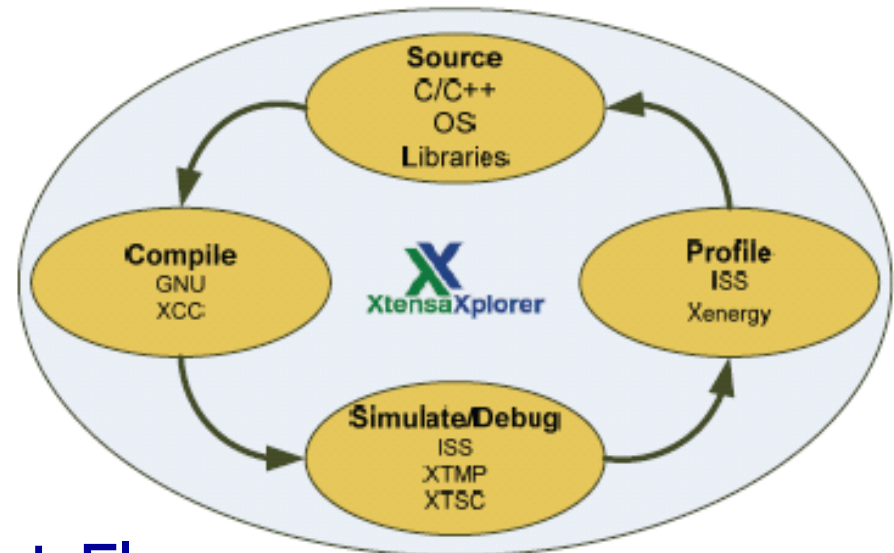
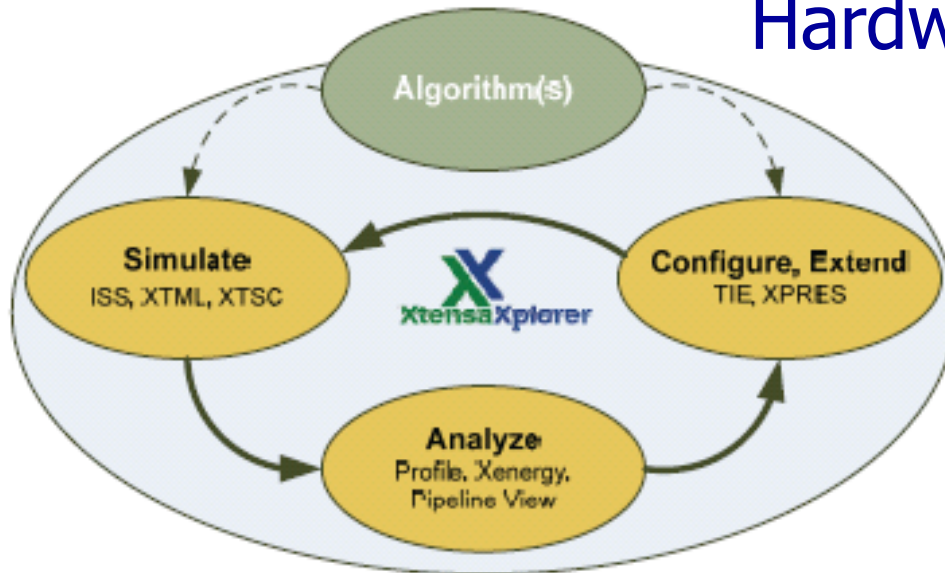
Xtensa LX3

Configuration Options & Designer Defined Functional Units



Tensilica's Xtensa Xplorer

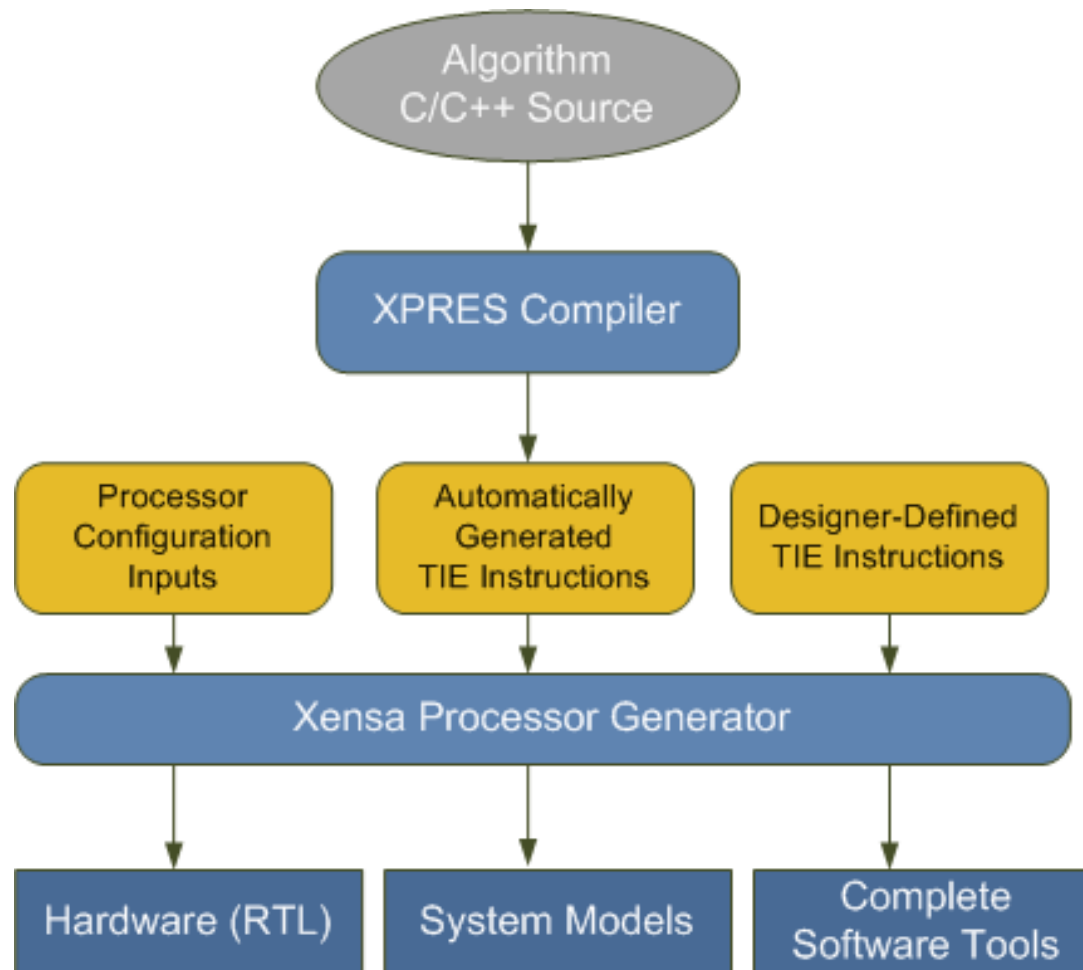
Hardware Development Flow



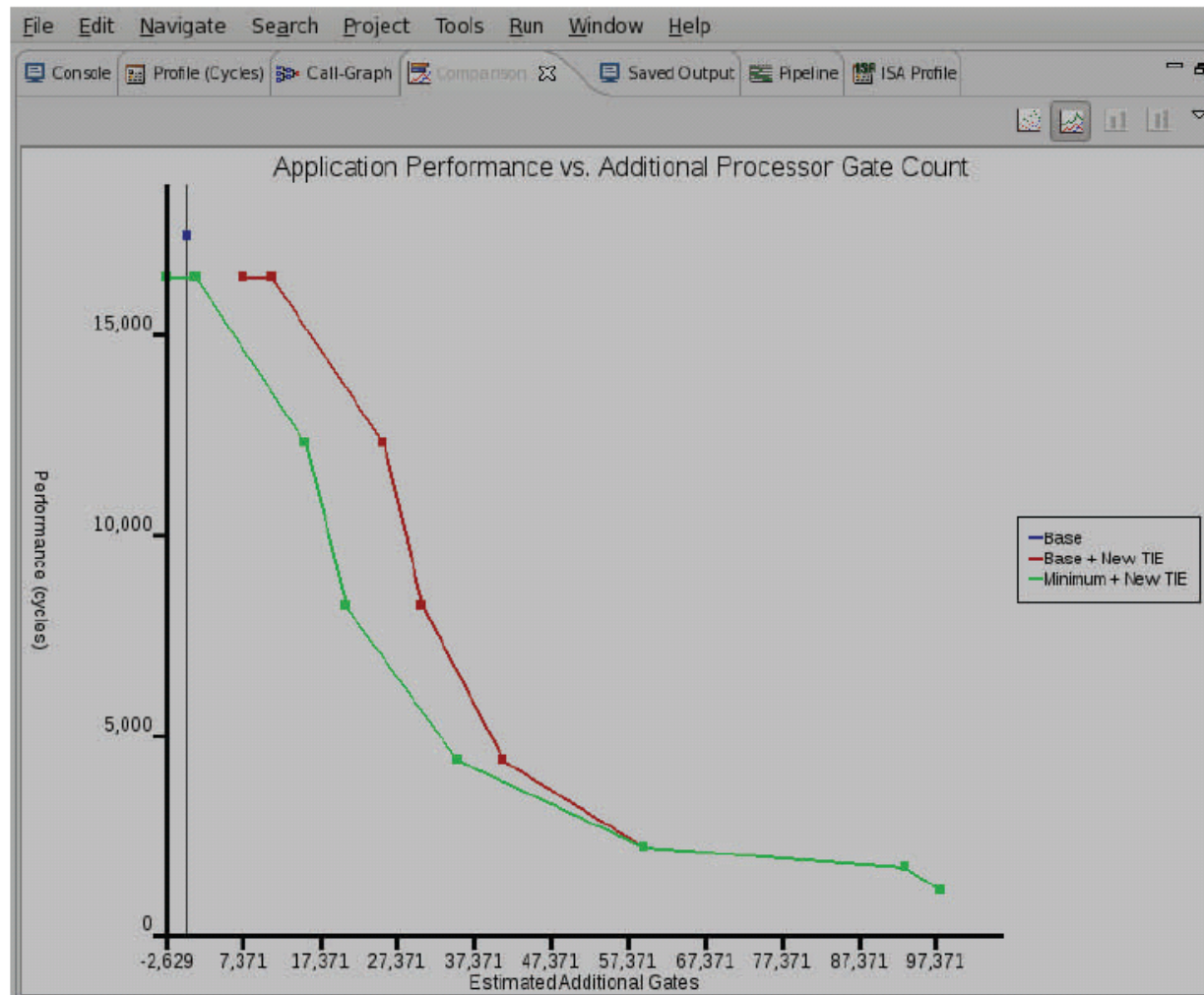
Software Development Flow

XPRES Flow

Xtensa Processor Extension Synthesis

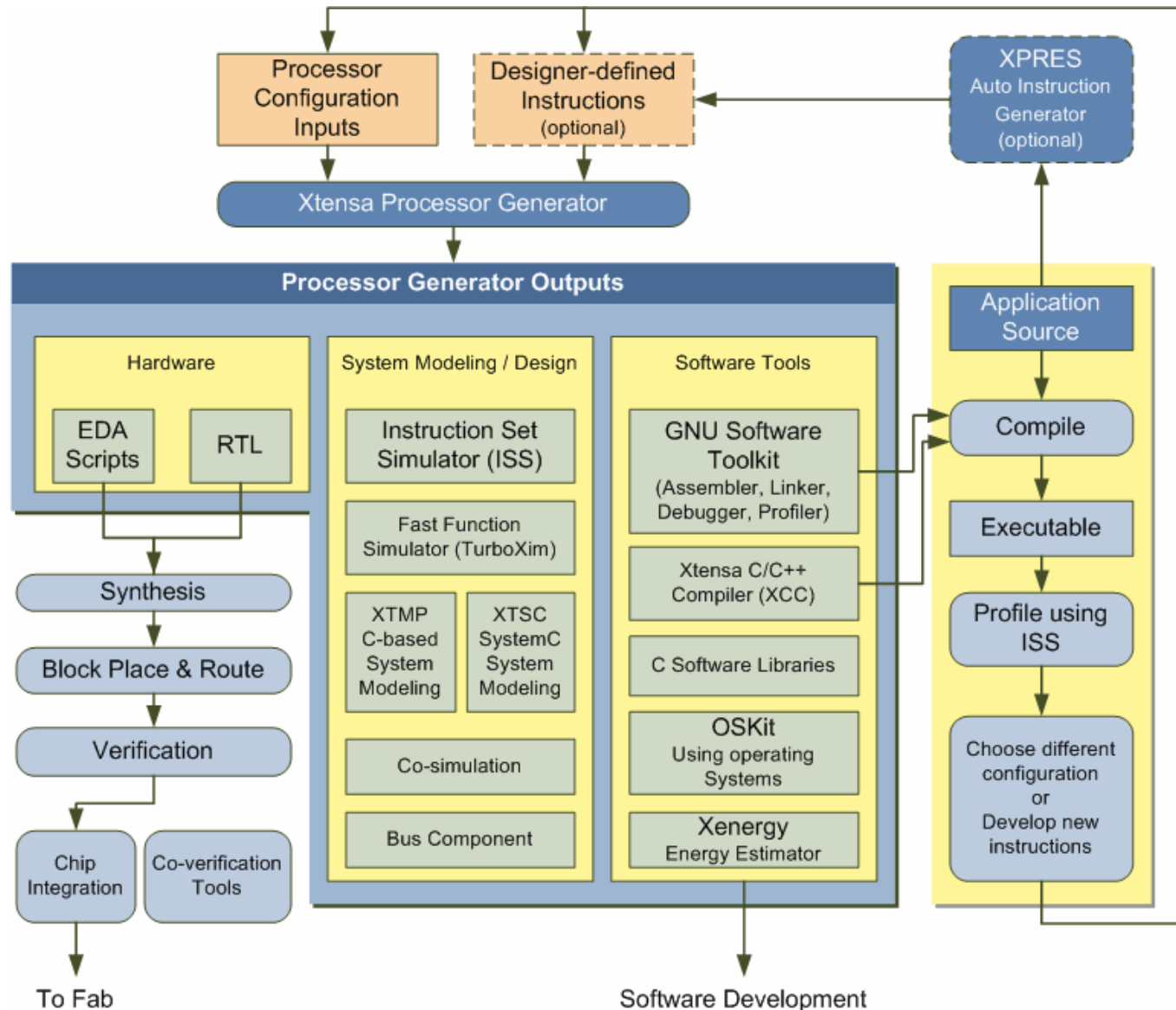


XPRES: Design Space Exploration

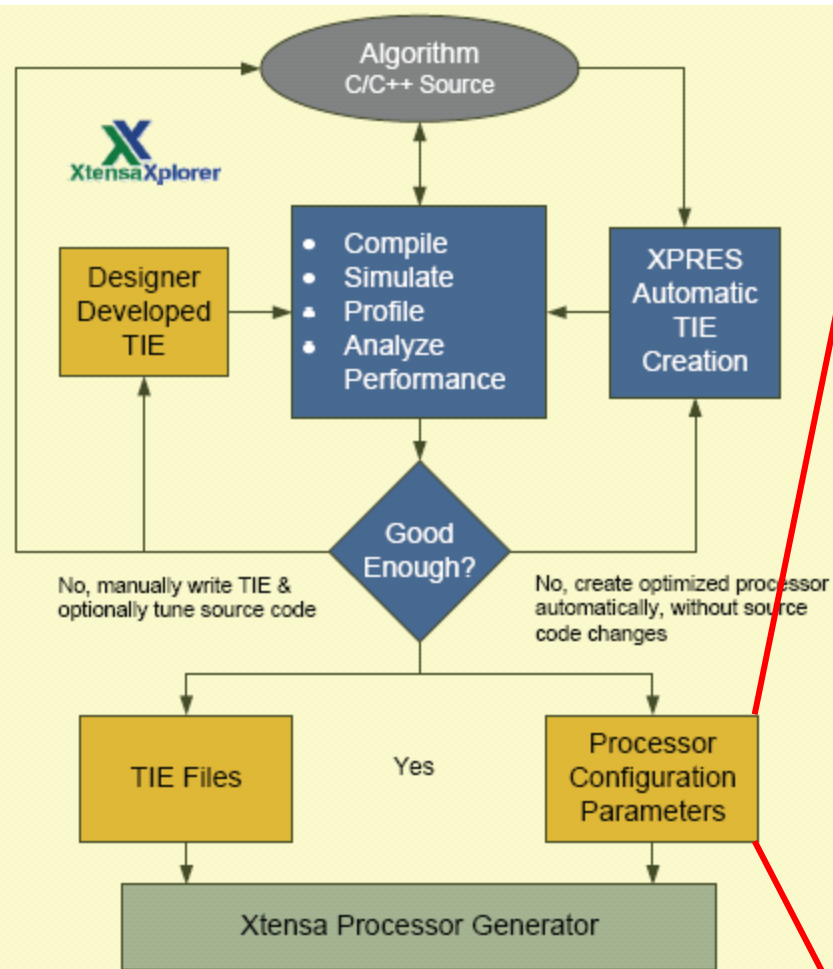


XPRES compiler rapidly explores millions of possible Processor Configurations

Xtensa Processor Automated Solution



Hardware Development



Memory Management Options

XEA2: Region protection Memory management selection
 1-entries per way 0-entries per way

Arithmetic Instruction Options

None MUL32 implementation selection ☐ MUL16
☐ MAC16 DSP instruction family ☐ CLAMPS
☐ 32 bit integer divider ☐ Single Precision FP (coprocessor id 0)
☐ Double Precision FP Accelerator

ISA Instruction Options

☒ NSA/NSAU ☒ MIN/MAX and MINU/MAXU
☒ Sign Extend to 32 bits ☒ Enable density instructions
☐ Enable Boolean Registers ☐ Enable Processor ID
☐ TIE arbitrary byte enables ☒ Zero overhead loop instructions
☐ Synchronize instruction ☐ Conditional store synchronizes instruction
 Number of Coprocessors (NCP) Miscellaneous Special Register count
☐ Thread Pointer

ISA Configuration Options

Number of AR registers for call windows
 Byte order (endianness)
 Action when handling an unaligned load / store
 Max instruction width in bytes. 4 and 8 byte instructions are part of the FLIX option
 Pipeline length

DSP Coprocessors

☐ Vectra LX DSP coprocessor instruction family
☐ Vectra/VME: Extra DSP Instructions
 Vectra adapts to match the number of configured Load/Store units
☐ ConnX D2 DSP

HIFI2 Audio Coprocessor

☐ HIFI2 Audio Engine DSP coprocessor instruction Family

Fixed Core Extensions

☐ FLIX: 3-way FLIX

TIE: Tensilica Instruction Extension

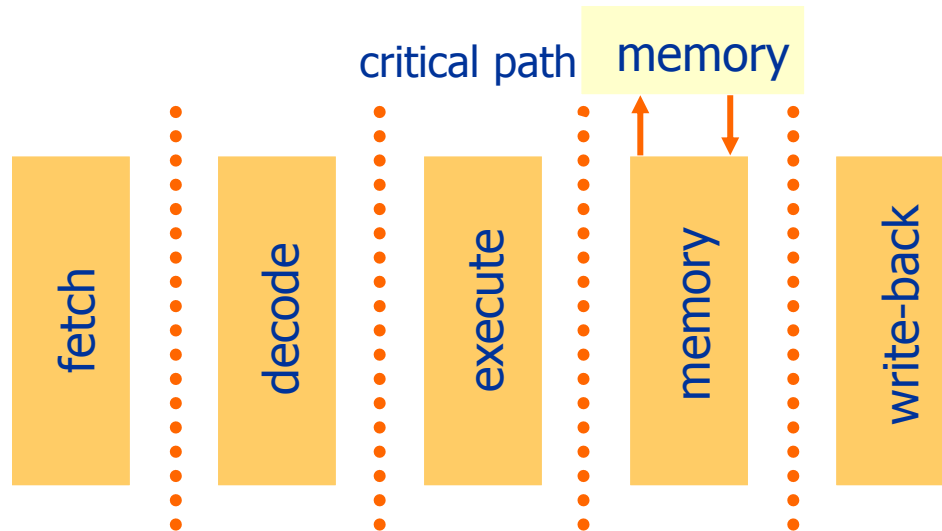
- ❑ Extend the processor's architecture and instruction set
- ❑ Resembles Verilog
 - ❑ More concise than RTL (it omits all sequential logic, pipeline registers, and initialization sequences).
- ❑ The custom instructions and registers described in TIE are part of the processor's programming model.
- ❑ Can be used for the TIE Compiler or for the Processor Generator
- ❑ TIE Combines multiple operations into one using:
 - ❑ Fusion, SIMD/Vector Transformation, FLIX

TIE Compiler

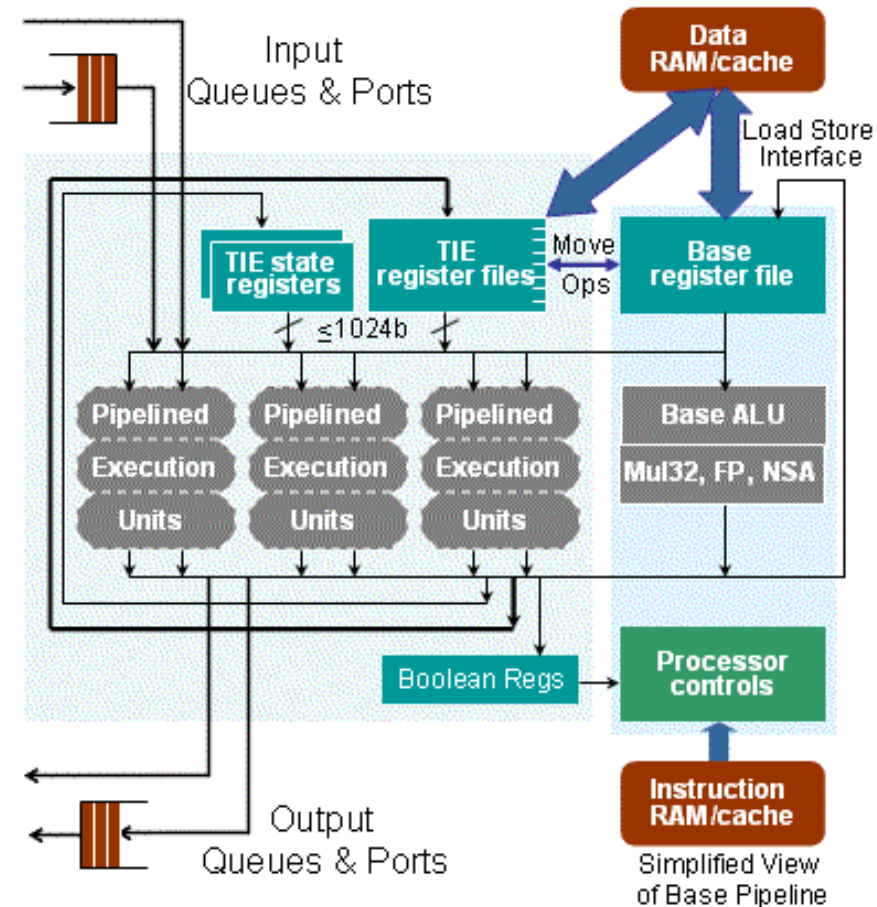
- ❑ After coding TIE, the compiler generates:
 - ❑ C-functions equivalent to TIE -> functional verification through usage in C with native software development environment
 - ❑ C-function declarations -> allow new instructions to be coded as functions in application code
 - ❑ Dynamic shared libs to be used by other Xtensa SW
 - ❑ HDL description (Verilog) -> hardware needed to support TIE instructions (gives also measure on HW costs and performance)
 - ❑ Synthesis scripts (for DC): allows to automatically synthesize the hardware from the HDL description

- ❑ Application code may be modified by the designer to exploit the new instruction and simulate for performance vs. hardware cost tradeoff

TIE Extensibility: Pipeline and TIE instructions



- ❑ Single-cycle access to memory
- ❑ 5 stage pipeline: “memory” often the critical path when it comes to high clock rates
- ❑ User can chose to avoid placing logic after memory result is read to avoid creating a critical path -> delay result assignment by one cycle using multi-cycle instructions



Fusion

❑ Combine dependent operations into a single instruction

❑ Example: Average of two arrays

```
unsigned short *a, *b, *c;
```

```
for( i = 0; i < n; i++)
    c[i] = (a[i] + b[i]) >> 1;
```

❑ Two Xtensa LX Core instructions required, in addition to load/store instructions

❑ Fuse the two operations into a single TIE instruction

```
operation AVERAGE{out AR res, in AR input0, in AR input1}{}{
    wire [16:0] tmp = input0[15:0] + input1[15:0];
    assign res = tmp[16:1];
}
```

❑ Essentially an add feeding a shift, described using standard Verilog-like syntax

❑ Implementing the instruction in C/C++

```
#include <xtensa/tie/average.h>
unsigned short *a, *b, *c;

for( i = 0; i < n; i++)
    c[i] = AVERAGE(a[i], b[i]);
```

(source: Tuan Huynh, Kevin Peek & Paul Shumate: Advanced Processor Architecture)

SIMD/Vector Transformation

- ❑ A single instruction by combining Fusion and SIMD
 - ❑ Fusing instructions into a “vector”
 - ❑ Replication of the same operation multiple times in one instruction
- ❑ Example: Four 16-bit averages in one instruction

```
regfile VEC 64 8 v
```

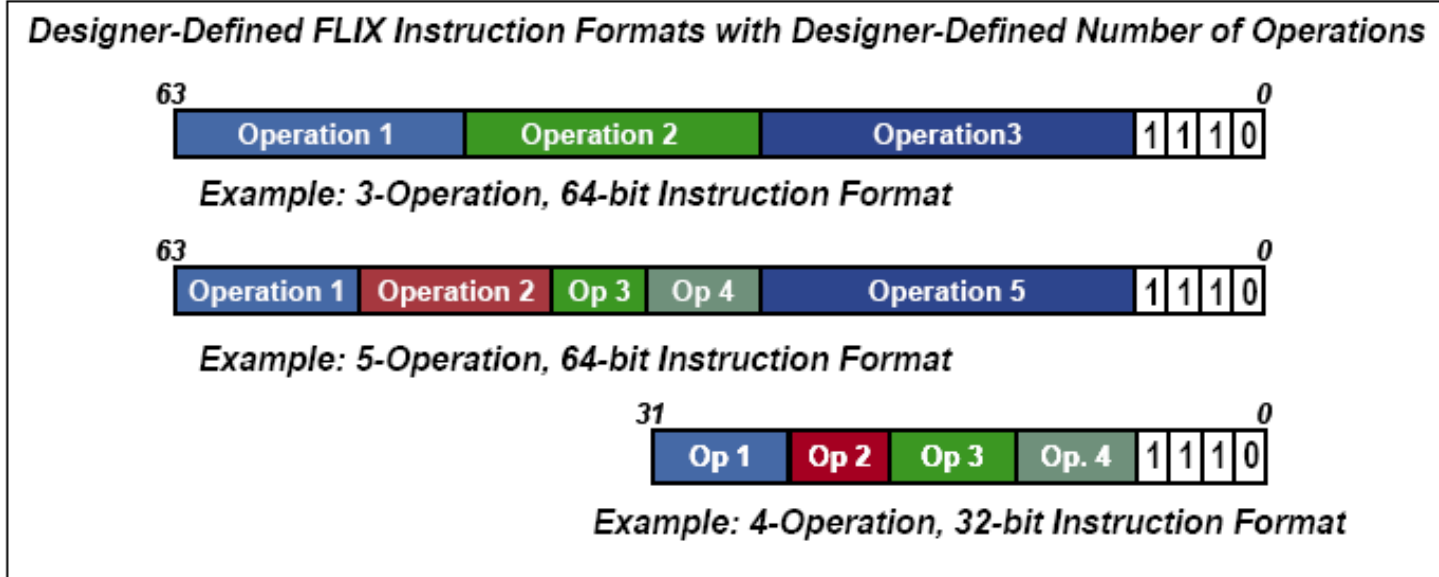
```
operation VAVERAGE{out VEC res, in VEC input0, in VEC input1} {} {
  wire [67:0] tmp = {
    input0[63:48] + input1[63:48],
    input0[47:32] + input1[47:32],
    input0[31:16] + input1[31:16],
    input0[15:0]  + input1[15:0]  };
  assign res = {tmp[67:52], tmp[50:35], tmp[33:18], tmp[16:1]};
}
```

- ❑ Create new register file, new instruction
 - ❑ VEC - eight 64-bit registers to hold data vectors
 - ❑ VAVERAGE - takes operands from VEC, computes average, saves results into VEC

```
VEC *a, *b, *c;
for (i = 0; i < n; i += 4){
  c[i] = VAVERAGE( a[i], b[i] );}
```

- ❑ TIE automatically creates new load, store instructions to move 64-bit vectors between VEC register file and memory

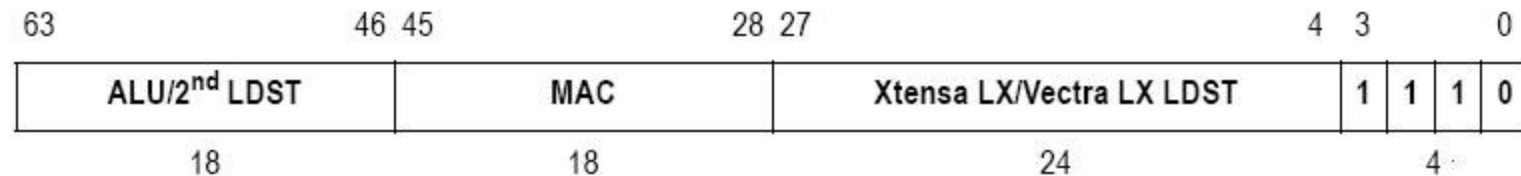
FLIX. Flexible Length Instruction Xtensions



- ❑ High-end Extensibility --> Used selectively when parallelism is needed
 - ❑ Similar to VLIW
 - ❑ But, customizable to fit application code's needs
 - ❑ Code size reduction
- ❑ Significant improvement over designs from the previous Xtensa series
 - ❑ Significant performance gains
 - ❑ DSP instructions formed using FLIX to be recognized as native to entire development system
- ❑ Created by XPRES Compiler

Vectra LX DSP Engine

- ❑ Optimized to handle DSP applications
- ❑ FLIX-based
- ❑ Vectra LX instructions encoded in 64 bits.



- ❑ Bits 0:3 of a Xtensa instruction determine its length and format, the bits have a value of 14 to specify it is a Vectra LX instruction
- ❑ Bits 4:27 – contain either Xtensa LX core instruction or Vectra LX Load or Store instruction
- ❑ Bits 28:45 – contains either a MAC instruction or a select instruction
- ❑ Bits 46:63 – contains either ALU and shift instructions or a load and store instruction for the second Vectra LX load/store unit

Vectra LX DSP Engine

Vectra DSP Engine Configurations					
	Vectra V1620-8	Vectra V1620-4	Vectra V1616-8	Vectra V0810-8	Vectra V3224-4
Elements per vector A	8	4	8	8	4
Memory width of each element B	16	16	16	8	32
Register width of each element C	20	20	16	10	24
Number of MAC units D	4	2	4	4	2
Multiplier and Multiplicand width E	16x16	16x16	16x16	8x8	24x24

(source: <http://www.tensilica.com>, Tuan Huynh, Kevin Peek & Paul Shumate: Advanced Processor Architecture)

Software Development

ISS with XTMP or XTSC for Modeling

Use XTMP/XTSC to instantiate and connect models/RTL

Xtensa ISS
Libraries

User Device
Models

Compile and link
on Host

Application Code

Run

XTSC

Device A
SystemC

RTL

Device B
SystemC

Pin Level XTSC

Producer core

FIFO

Consumer core

RAM

ROM

System
Memory

RAM

ROM

xtsc-run

```
-set_queue_param=depth=4
-create_core=Producer
-connect_core_queue=Producer,FIFO_OUT,fifo
-create_core=Consumer
-connect_queue_core=fifo,FIFO_IN,Consumer
```

XTMP

Device A
Model

RTL

Device B
Model

Producer core

FIFO

Consumer core

RAM

ROM

System
Memory

RAM

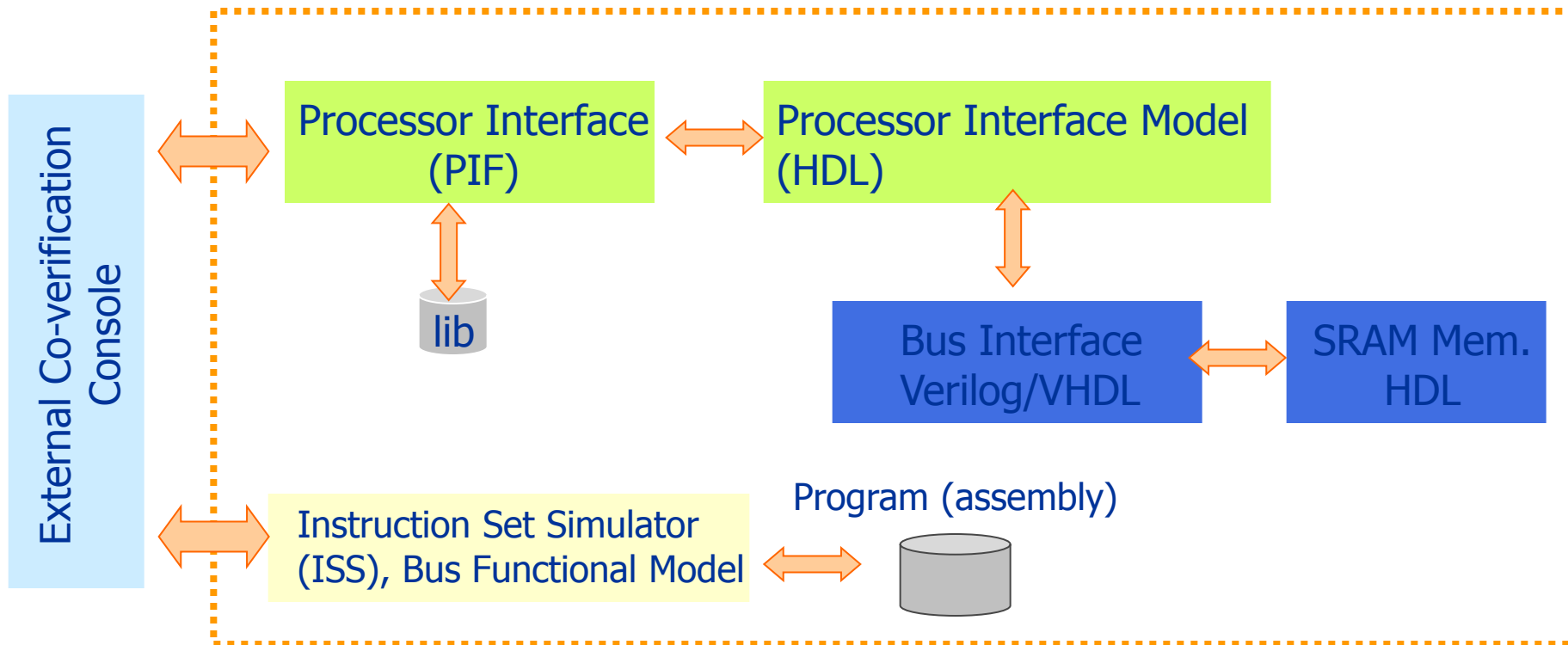
ROM

```
XTMP_core consumer = XTMP_coreNew("consumer", config);
XTMP_core producer = XTMP_coreNew("producer", config);

XTMP_queue fifo = XTMP_queueNew("fifo", width, depth);
XTMP_connectQueue(fifo, producer, "FIFO_OUT", consumer, "FIFO_IN");
```


Co-verification capabilities

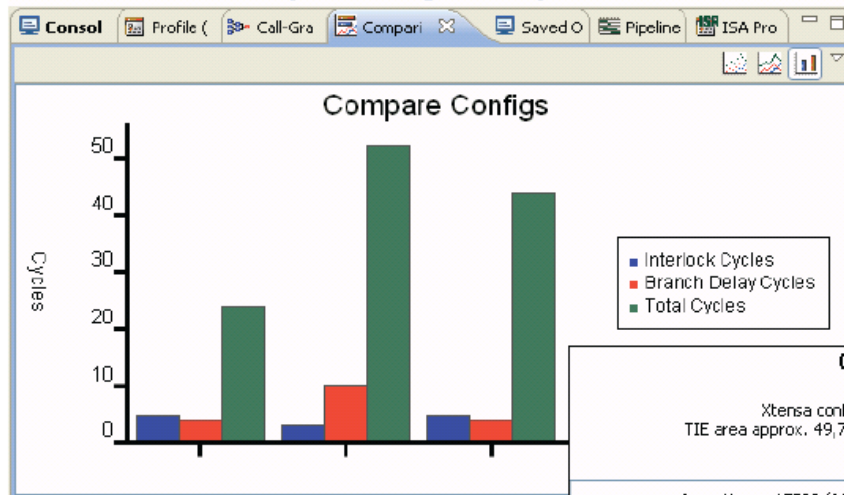
- An External co-verification tool links SW and HW simulation models:
 - ISS: cycle-accurate, models pipeline, handles interrupts/excptions
 - PIM: models processor interface signals; needs stand. HDL simulator
 - Third-part co-verification tool: e.g. Synopsys Eagle, ...



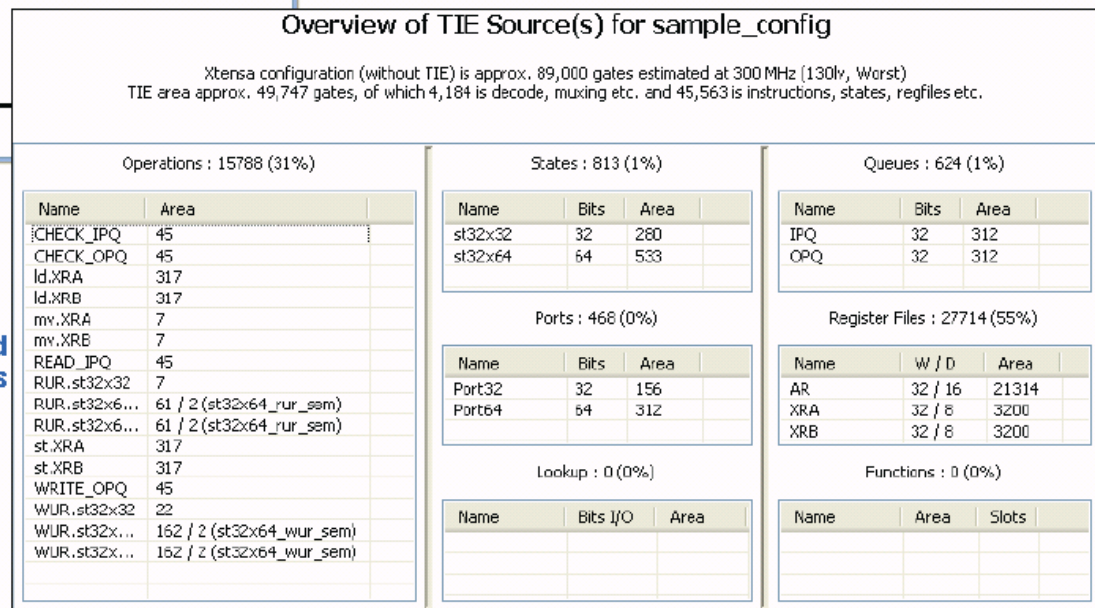
The Xtensa Xplorer

Automatic speed, size and power estimates as you build your processor

Benchmark and Compare configuration performance



Easily review Designer Defined instruction implementations



The Xtensa Xplorer

Debug, Trace and View

Variables

Name	Value
argv	0x40000000
a	05602537 <i>Vector, no formatting</i>
inc	1000
inc	1
a_vec	1001 1001 <i>Vector with formatting</i>
inc_vec	11

User Defined Display Formats

vector.c

```

1#include <stdio.h>
2#include <xtensa/tim/xt_connxd2.h>
3
4int main(int argc, char **argv) {
5    int a = 0;
6    short int inc = 1000;
7    short int inc = 1;
8    xd2_int16x2d a_vec = inc;
9    xd2_int16x2d inc_vec = inc;
10   a_vec = a_vec + inc_vec; // Overloaded
11   a = *(int *) &a_vec; // Scalar equivalent
12   return 0;
13}
14

```

Disassembled Instructions

Trace with interleaved C/Assembler and auto-link to source

Branch	Address	Instruction
main	Branch 15	
	600004a8	movh a7, 4
	600004aa	{xd2_ls_d16x xdd3, a1, a7; nop }
	600004b2	{xd2_s_d16x2 xdd3, a1, 8; nop }
		xd2_int16x2d inc_vec = inc;
	600004ba	movh a6, 6
	600004bc	{xd2_ls_d16x xdd2, a1, a6; nop }
	600004e4	{xd2_s_d16x2 xdd2, a1, 12; nop }
		a_vec = a_vec + inc_vec; // Overl...
	600004ec	xd2_ls_d16x2s xdd0, a1, 8
	600004f0	xd2_ls_d16x2s xdd1, a1, 12
	600004d2	{xd2_add_d16x2s xdd0, xdd0, xdd...
	600004da	{xd2_s_d16x2 xdd0, a1, 8; nop }
		a = *(int *) &a_vec; // Scalar equiv...
	600004e2	l32n a5, a1, 0
	600004e4	l32n a5, a1, 0
		return 0;
	600004e6	movh a2, 0
	600004e8	retw.n

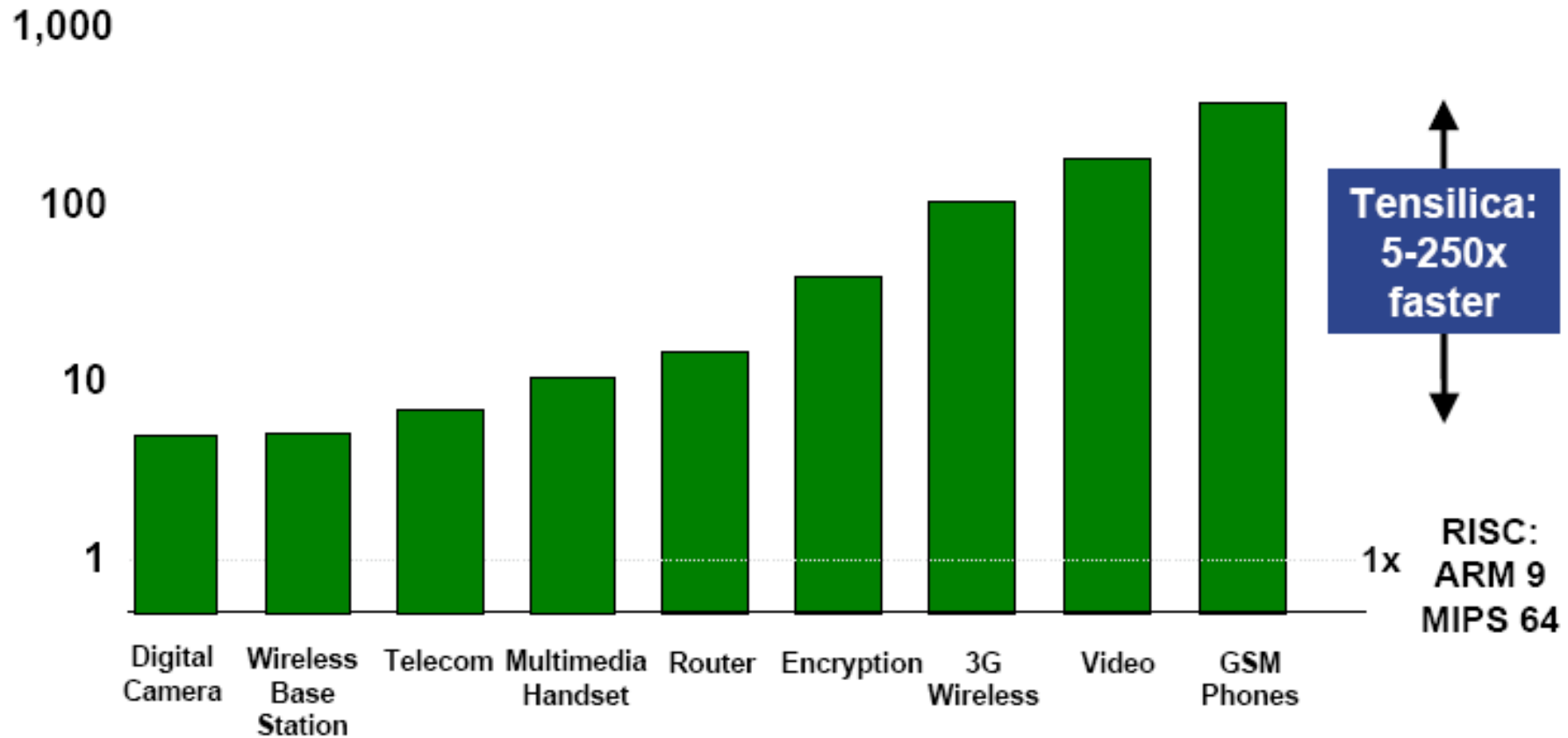
Pipeline Viewer

Multi-core Profiling

Cycles

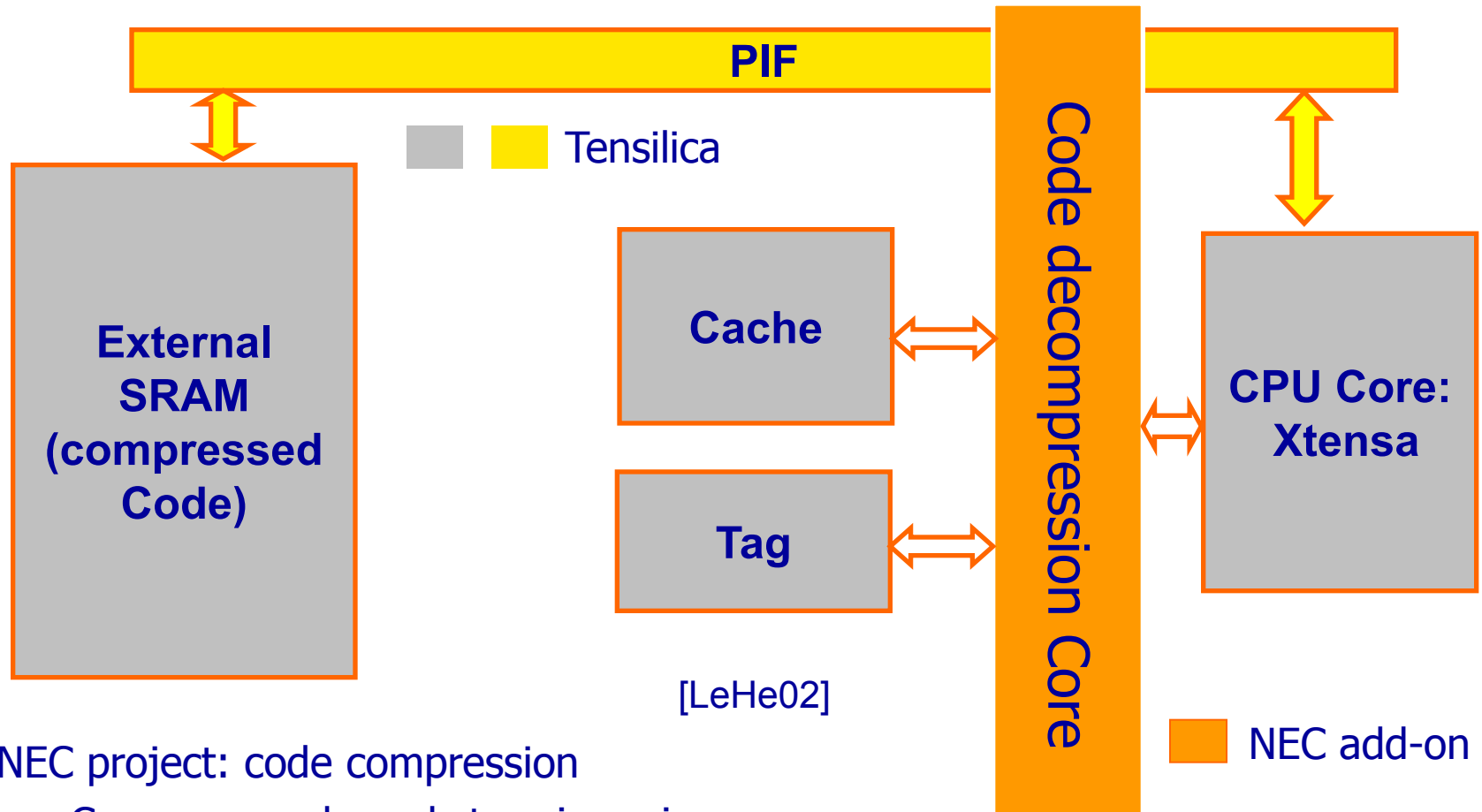
Category	core0	core1
ICache Miss Cycles	~2000	~2000
DCache Miss Cycles	~1000	~1000
Uncached Instruction Fetch Cycles	~5000	~5000
Uncached Load Cycles	~1000	~1000
Interlock Cycles	~1000	~1000
Branch Delay Cycles	~1000	~1000
Total Cycles	~15000	~15000

Xtensa: Benchmarks



Source: Tensilica
Performance of Tensilica vs. RISC (ARM 9 / MIPS 64) in core algorithms

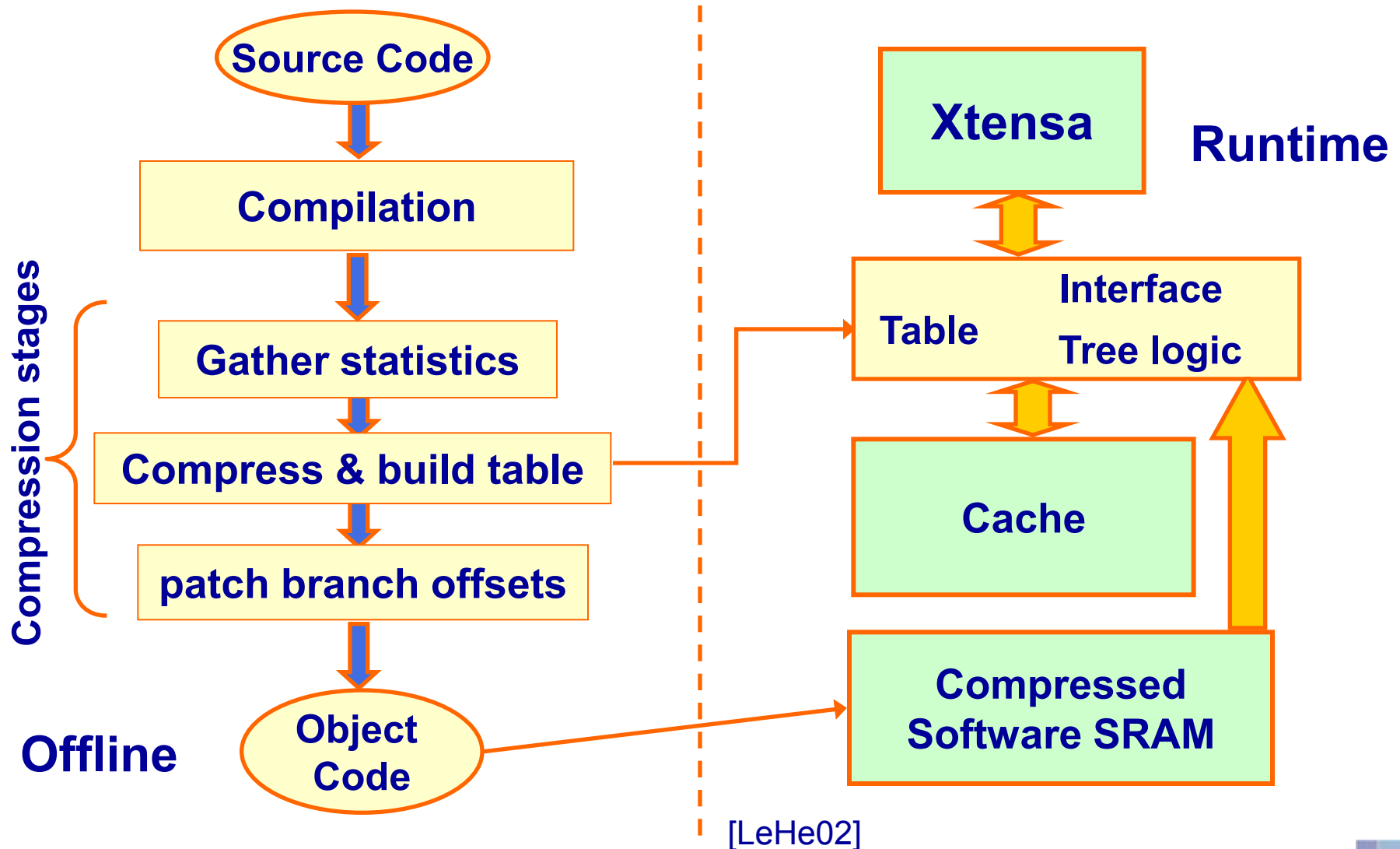
Code Compression Project using Tensilica's Xtensa (Henkel, Lekatsas)



[LeHe02]

- NEC project: code compression
 - Compress code and store in main memory
 - Decompress on-the-fly in just 1 cycle
- Use Tensilica's framework: IP cores, simulation and synthesis capabilities

Code Compression Project using Tensilica's Xtensa (Henkel, Lekatsas)

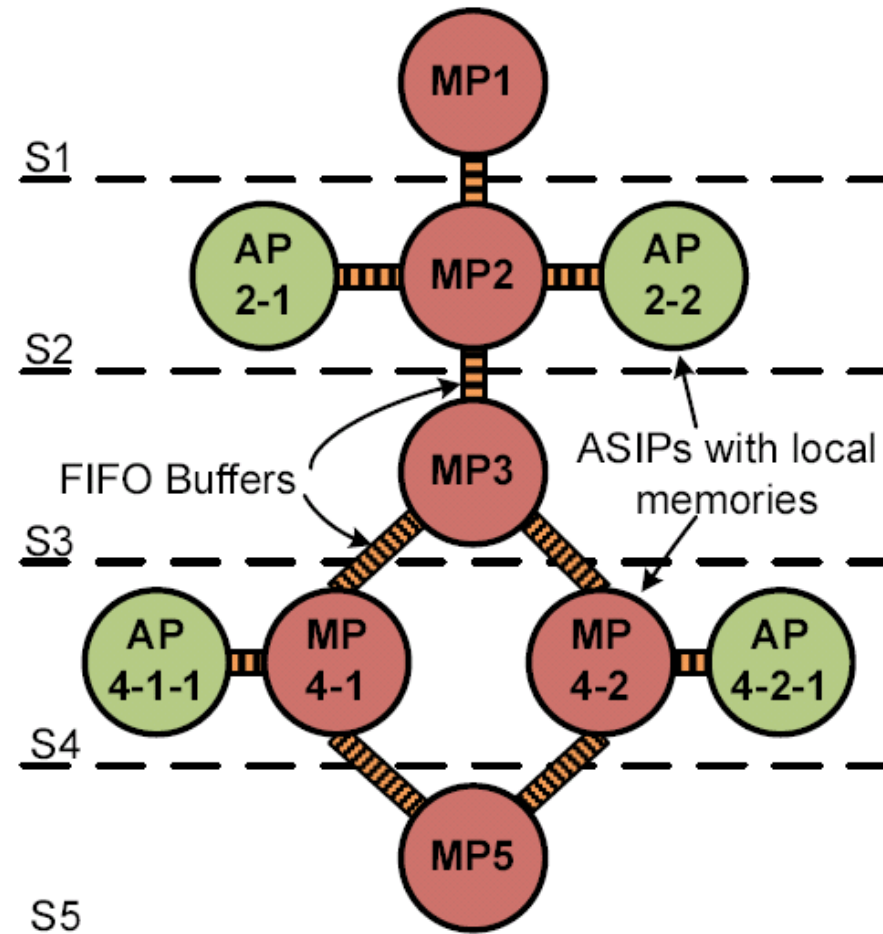


[LeHe02]

H.264 Video Encoder Project

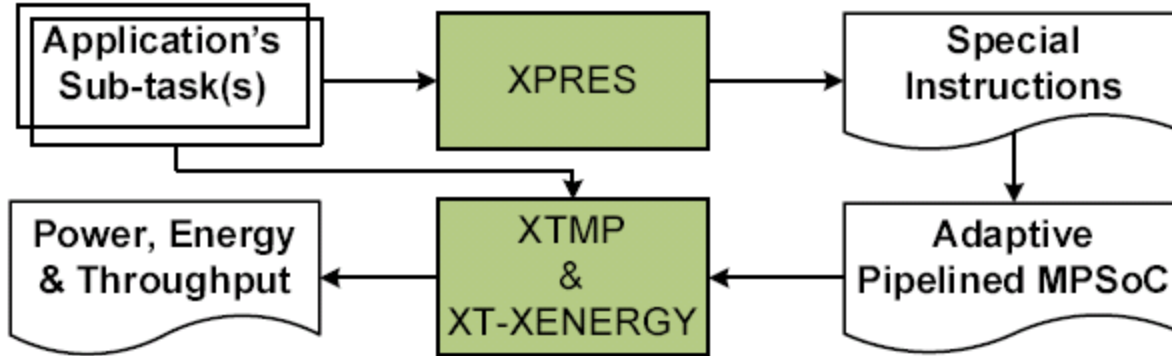
Adaptive Pipelined MPSoC

(Javed, Shafique, Parameswaren, Henkel)



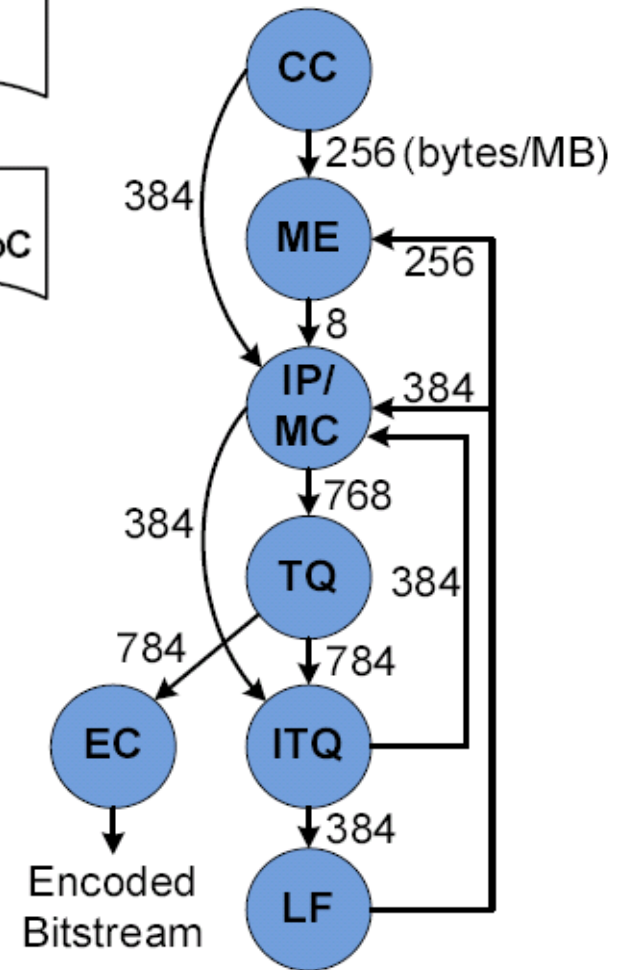
H.264 Video Encoder Project

(Javed, Shafique, Parameswaren, Henkel)



MP	Area (KGates)	Power (mW)	
		Dynamic	Leakage
CC	92.44	43.96	6.80
ME	103.23	40.59	8.92
IP/MC	103.65	40.96	7.69
TQ	91.56	49.34	6.50
ITQ	93.50	50.63	7.07
LF	87.76	46.23	6.05
EC	90.12	44.55	6.37





CC: Color Conversion
ME: Motion Estimation
IP: Intra Prediction
MC: Motion Compensation
TQ: Transform & Quantize
ITQ: Inverse TQ
LF: Loop Filter
EC: Entropy Coding



The LisaTek Platform\

(LisaTek: TH Aachen)

Overview

-  Paradigm
-  The LISA language
-  Design Flow and Tools
-  Simulation

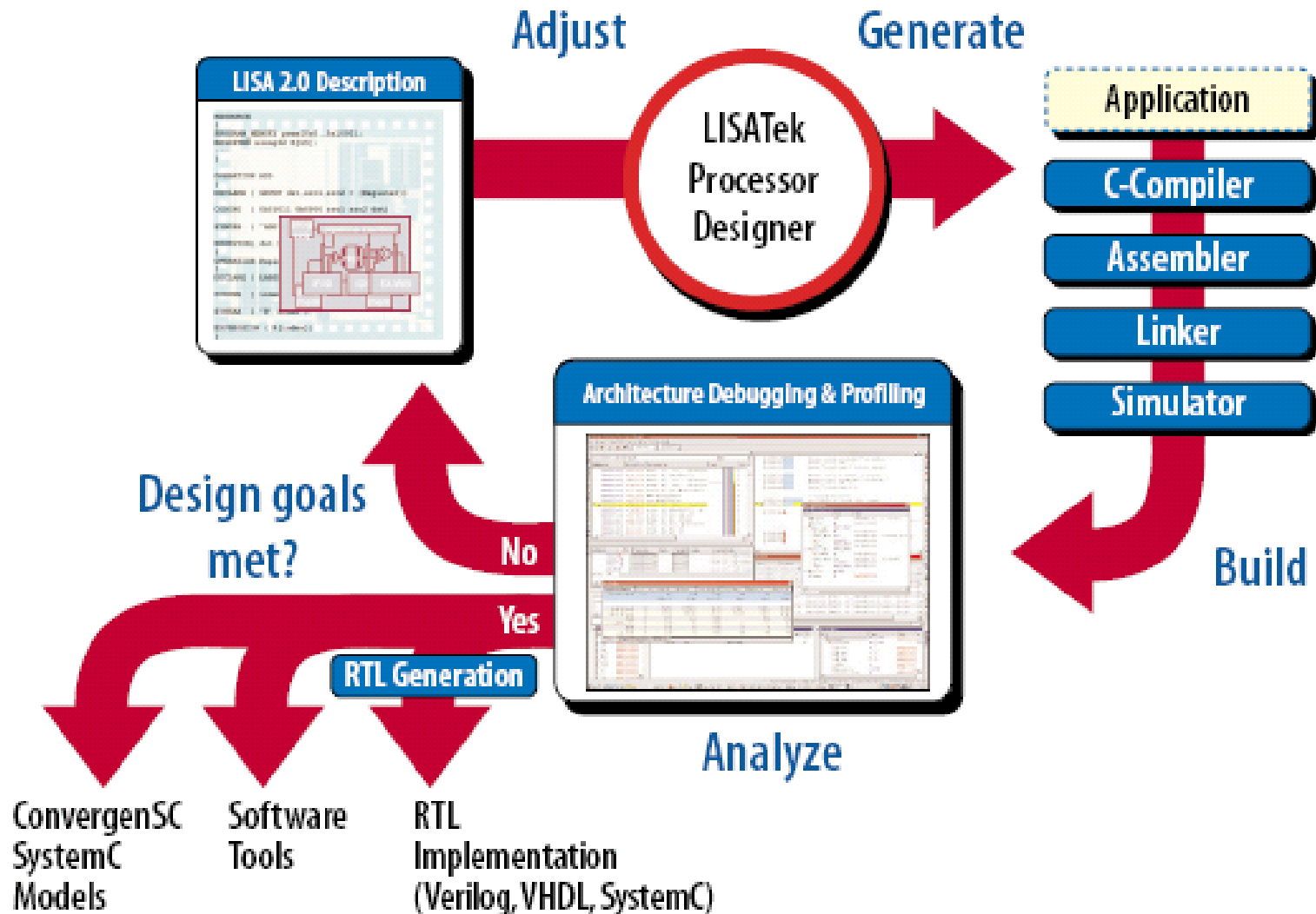
Paradigm and Features

- ❑ Combining architectural exploration and implementation in one tool suite
- ❑ Software development tools are derived (generated) from the description
- ❑ Not using a standard core; instead, the whole Instruction Set Architecture (ISA) is customized
- ❑ Status: commercial product

Features at a glance

- Textual description of target architecture
- Hardware Model
 - Behavior : C/C++ description
 - Resources : register, pipelines etc.
 - Timing information
 - Pipeline-model
- Software Model
 - Instruction-set description
- Hierarchical description style
 - LISA operations
- Different levels of abstraction
 - abstraction of time (instruction/ cycle accurate)
 - abstraction of architecture

LISATek: Design Flow and Tools



LISATek C Compiler Generation

**LISA
processor model**

```

SYNTAX {
  "ADD" dst, src1, src2
}

CODING {
  0b0010 dst src1 src2
}

BEHAVIOR {
  ALU_read (src1, src2);
  ALU_add ();
  Update_flags ();
  writeback (dst);
}

SEMANTICS {
  src1 + src2 → dst;
}

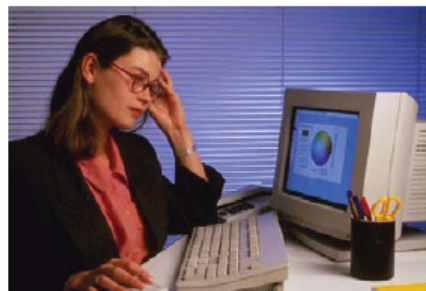
...

```

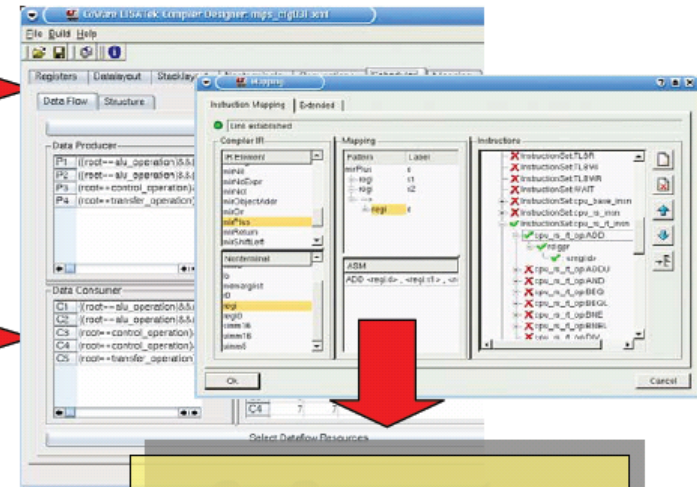


Autom. analyses

Manual refinement



GUI

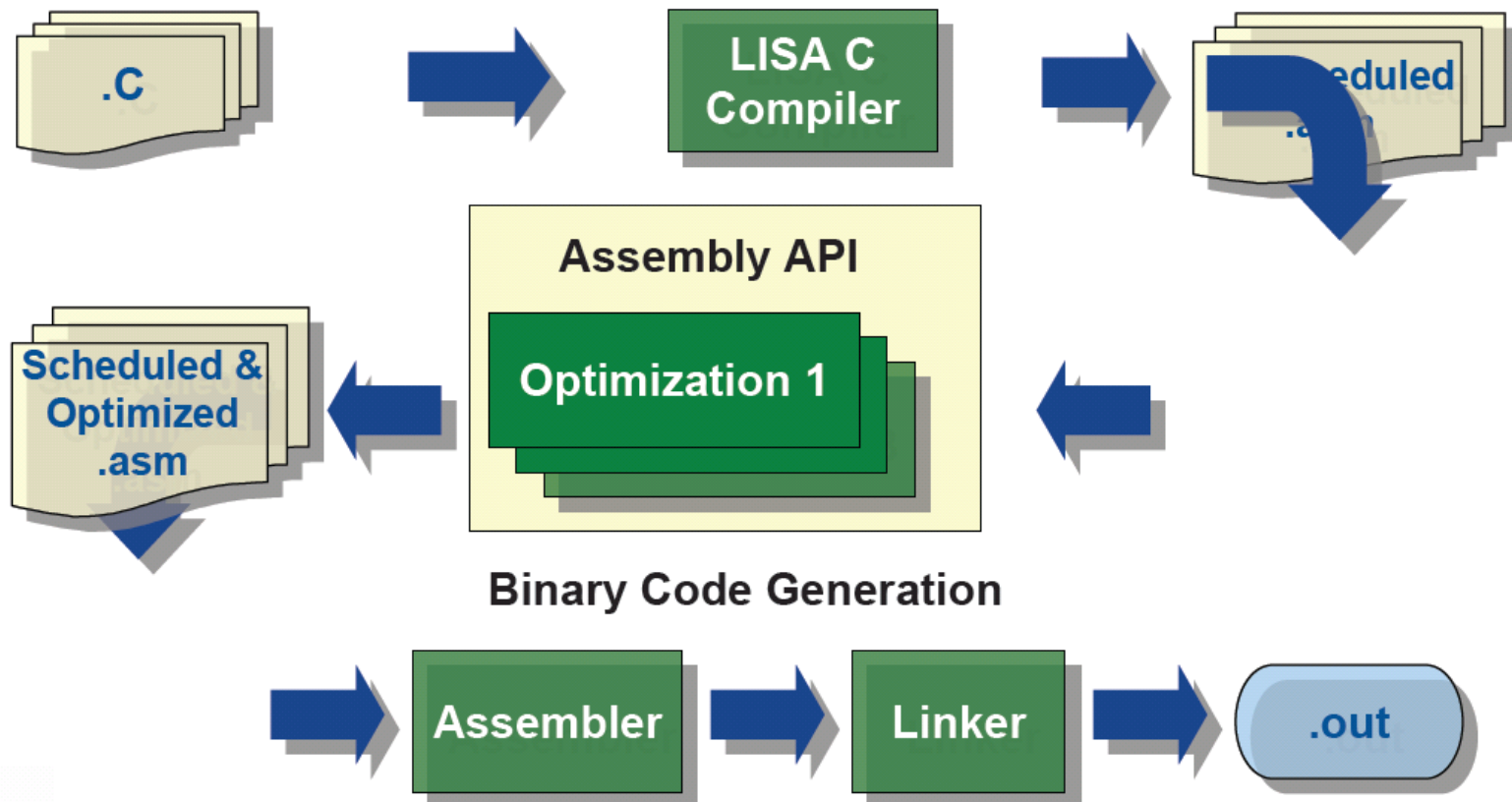
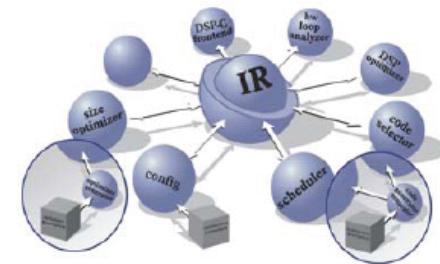


CoSy system

C Compiler

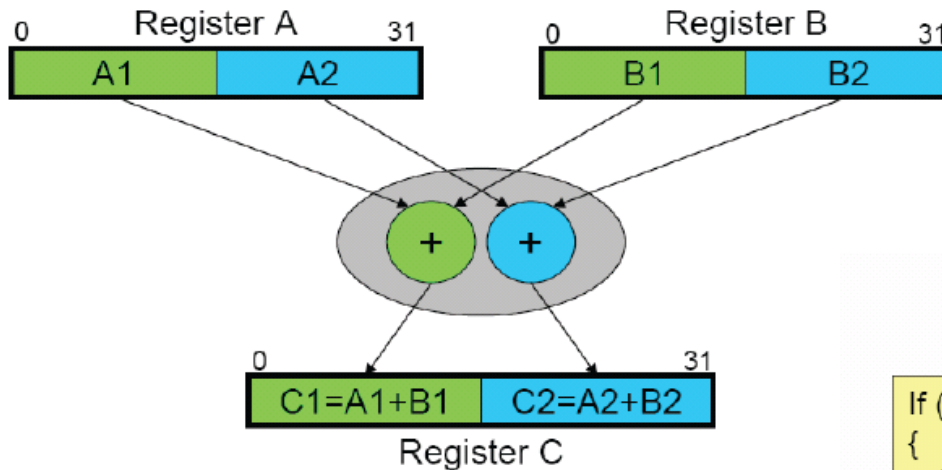
Adding Processor-Specific Code Optimizations

- High-level (compiler IR)
 - Enabled by CoSy's engine concept
- Low-level (ASM):



Code Optimization

➤ CoSy engines for SIMD and conditional instructions



```

If ( a > b )
{
    // Then Block
}
Else
{
    //Else Block
}

```



```

CMP  a,b
JMPGT Then
//Else Block
JMP  EndThen
Then:
    //ThenBlock
EndThen:

```

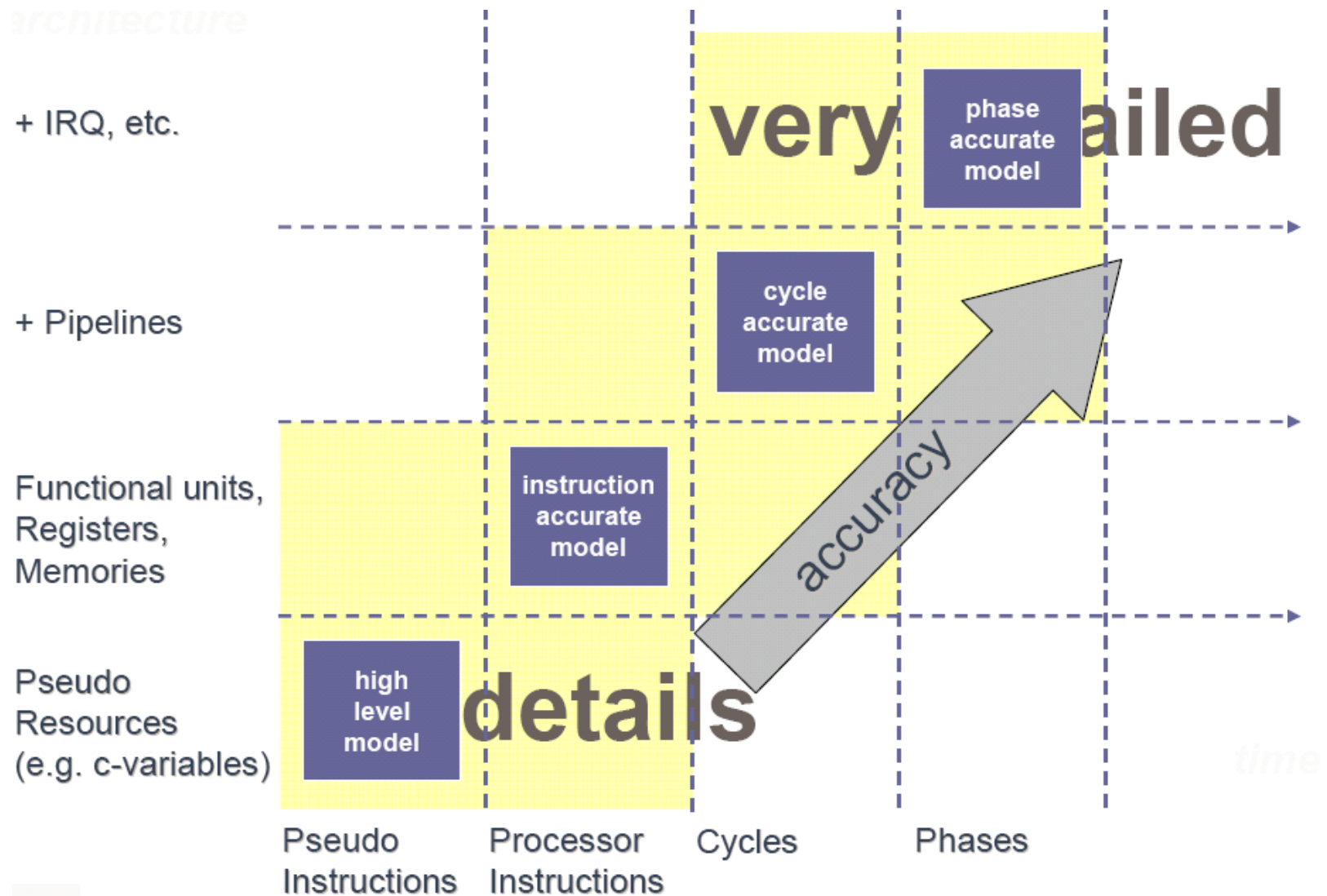
```

CMPGT a,b,flag
[ flag ] //ThenBlock
[! flag ] //ElseBlock

```



LISA 2.0 Abstraction Levels



LISA language: features

Tools and Models

- ❑ **Basic idea:** closing gap between structural oriented languages (HDL, Verilog) and instruction set languages
 - ❑ Support cycle-accurate processor models
 - ❑ Support for compiled simulation
 - ❑ Distinction between behavior and semantics
 - > freely determine abstraction level of the processor model
 - ❑ retargeting various tools: compiler, assembler, simulator
 - > Retargeting: having a generic tool that works various architectural scenarios
 - > **retargeting requires different types of architectural information**
- ❑ **Memory model:**
 - ❑ registers, memories with width ranges etc.
- ❑ **Resource model:**
 - ❑ specifies available hardware (like FUs, ...) and resource requirement of operations

LISA language: features

Tools and Models

❑ **Instruction set model:**

- ❑ instruction word coding, spec. of valid operands and addressing modes;
- ❑ written in assembly syntax
- ❑ collects all instructions as combinations of hw operations that are permitted by the CPU controller
- ❑ comprises instruction semantics

❑ **Behavioral model:**

- ❑ activities of hw structures are abstracted to operations
- ❑ notion of state for simulation; change state of the system
- ❑ abstraction level can vary widely between hw implementation level and high-level language

❑ **Timing model:**

- ❑ specifying the (activation) sequence of hardware operations and units

❑ **Micro-architectural model:**

- ❑ grouping of hardware operations to FUs;
- ❑ describes the details of micro-architectural implementation of RTL components

Processor Architecture

- Mixed behavioral/structural model:
 - based on C/C++
 - VLIW data-types
 - strong memory modeling capabilities (incl. caches)
 - include external IP (libraries)
- Enriched by timing information:
 - clocked register behavior
 - operation scheduling
 - extensive pipeline model with predefined functions
 - stall, flush

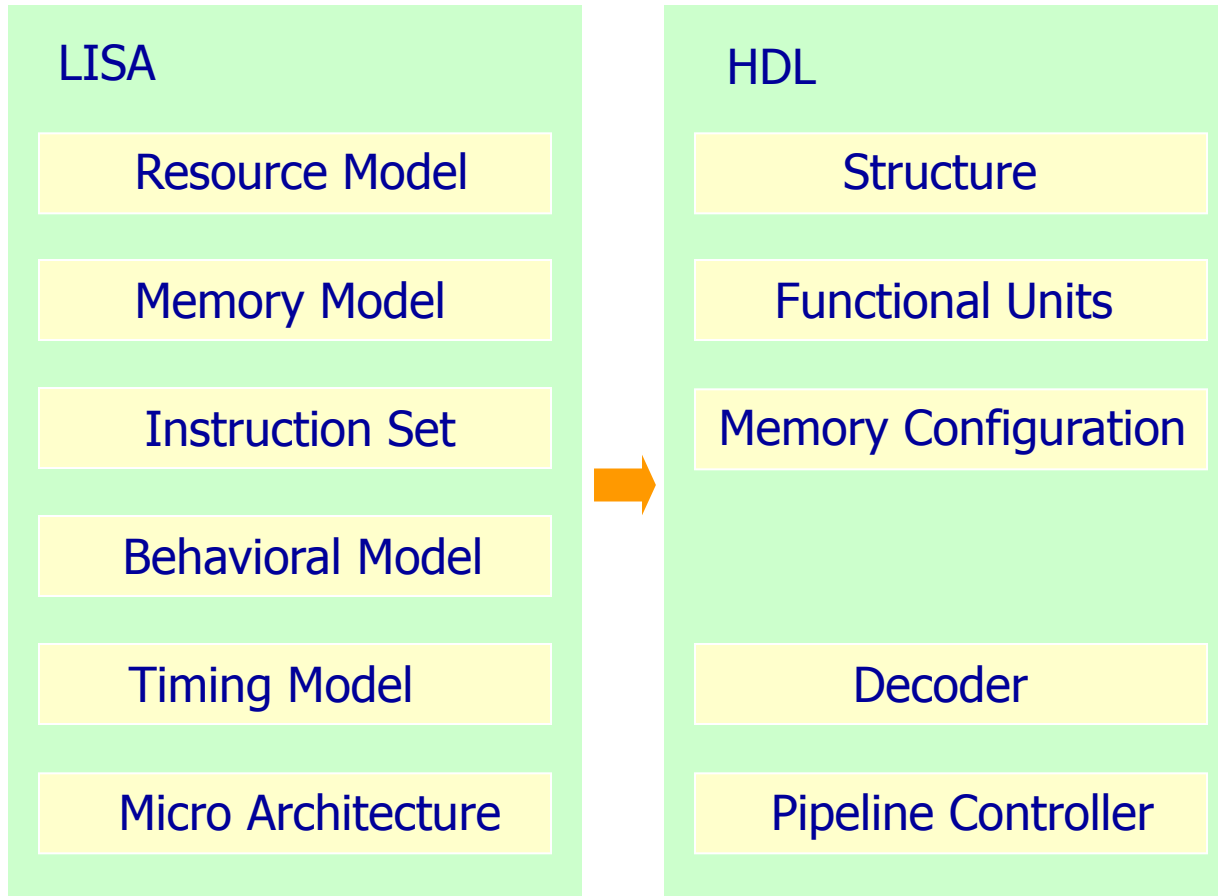
(source: LISATek)

Instruction Set Description

- **Instruction word coding**
 - variable widths
 - multiple words
 - distributed coding
- **Assembly syntax**
 - mnemonic based syntax
 - algebraic (C-like) syntax
- **Instruction semantics**
 - compiler semantics
- **Configurable instruction set information**
 - (power, etc.)

(source: LISATek)

Generating HDL from LISA



- Model does not consist of predefined components -> must be generated from description:
 - **Memory** derived
 - **Structure** stages: derived from resource, behavioral and micro-architectural models
 - **FUs** architectural model (fully functional or empty entities)
 - **Decoder** info in instruction set model

Architectural Features

■ Non-orthogonal coding elements

```

OPERATION Decode IN pipe.DC
{
    ENUM  InsnType=Type1, Type2, Type3;
    SWITCH (InsnType)
        CASE Type1:
            CODING {Decode==Decode_16}
        CASE Type2:
            CODING { (Decode==Decode_32)
                    && (Fetch==Operand) }
        CASE Type3:
            CODING { (Decode==Decode_48)
                    && (Fetch==Operand1)
                    && (Prefetch==Operand2) }
}

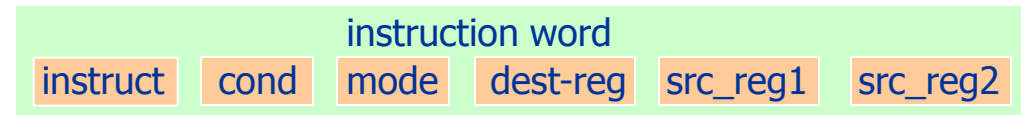
```

```

OPERATION add
{
    DCL ARE {REFE, RENCE mode; }
    if (mode==short) {
        BEHAVIOR {dest_lo=src1_lo+src2_lo; }
    }
    ELSE
        BEHAVIOR {
            dest_lo=src1_lo+src2_low;
            carry=dest_lo >> 16;
            dest_low &= 0xFFFF;
            dest_hi=src1_hi+src2_hi+carry;
        }
}

```

- Example for multiple instruction words and its implementation in LISA

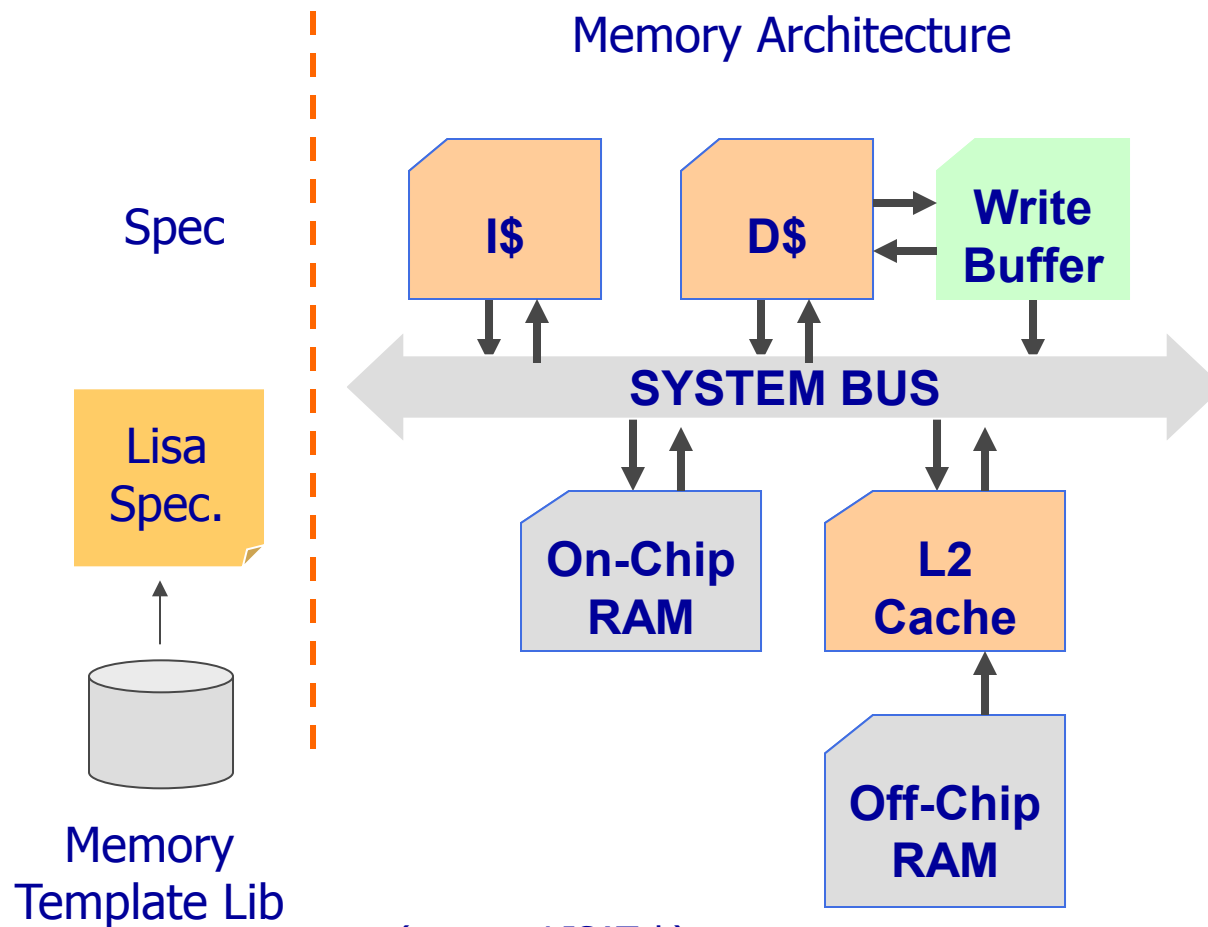


- Instruction:
 - add, sub, mul, ld, sto

- mode:
 - short
 - long

Modeling Memory

- ❑ To test the performance of the microarchitecture
- ❑ Non-Synthesizable



Features:

- dynamic address mapping
- user-defined memory modules
- different levels of abstraction
- C++ and SystemC simulation models
- bus redirect allows external memory access
- access statistics

(source: LISATek)

Case Study

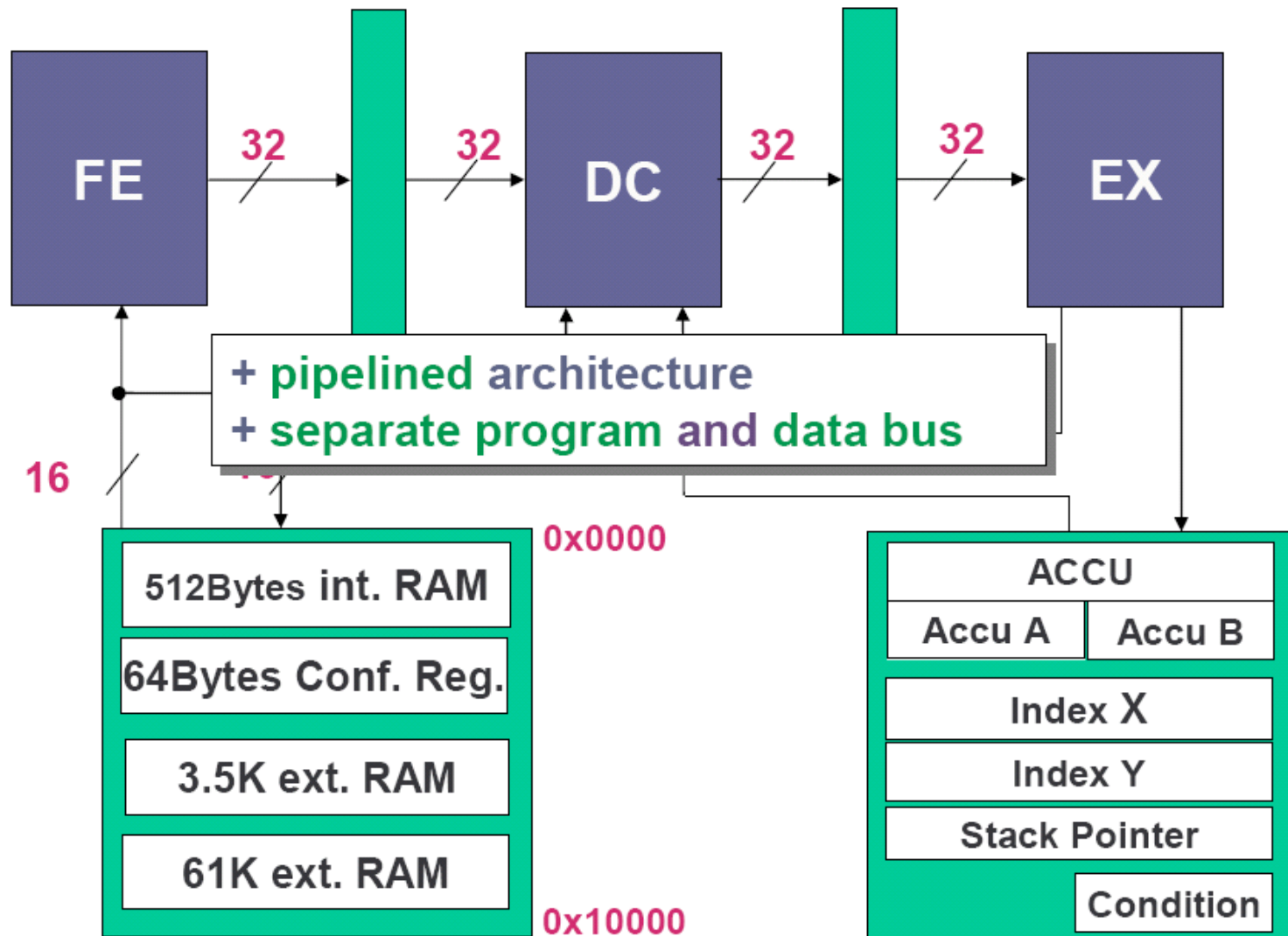
Motorola M65HC11 Architecture

M68HC11 CPU Architecture : Hot spots

- » 8-bit micro-controller.
 - » Harvard Architecture
- » 7 CPU Registers.
- » 6 different Addressing Modes.
- » Shared data and program bus. : stalled data access
- » Instruction width : 8, 16, 24, 32, 40 : multi-cycle fetch
- » 8-bit opcode : 181 instructions
- » Clock speed : ~200 MHz
- » Performance : : non-pipelined
- » Area : 15K to 30K (DesignWare® Library)

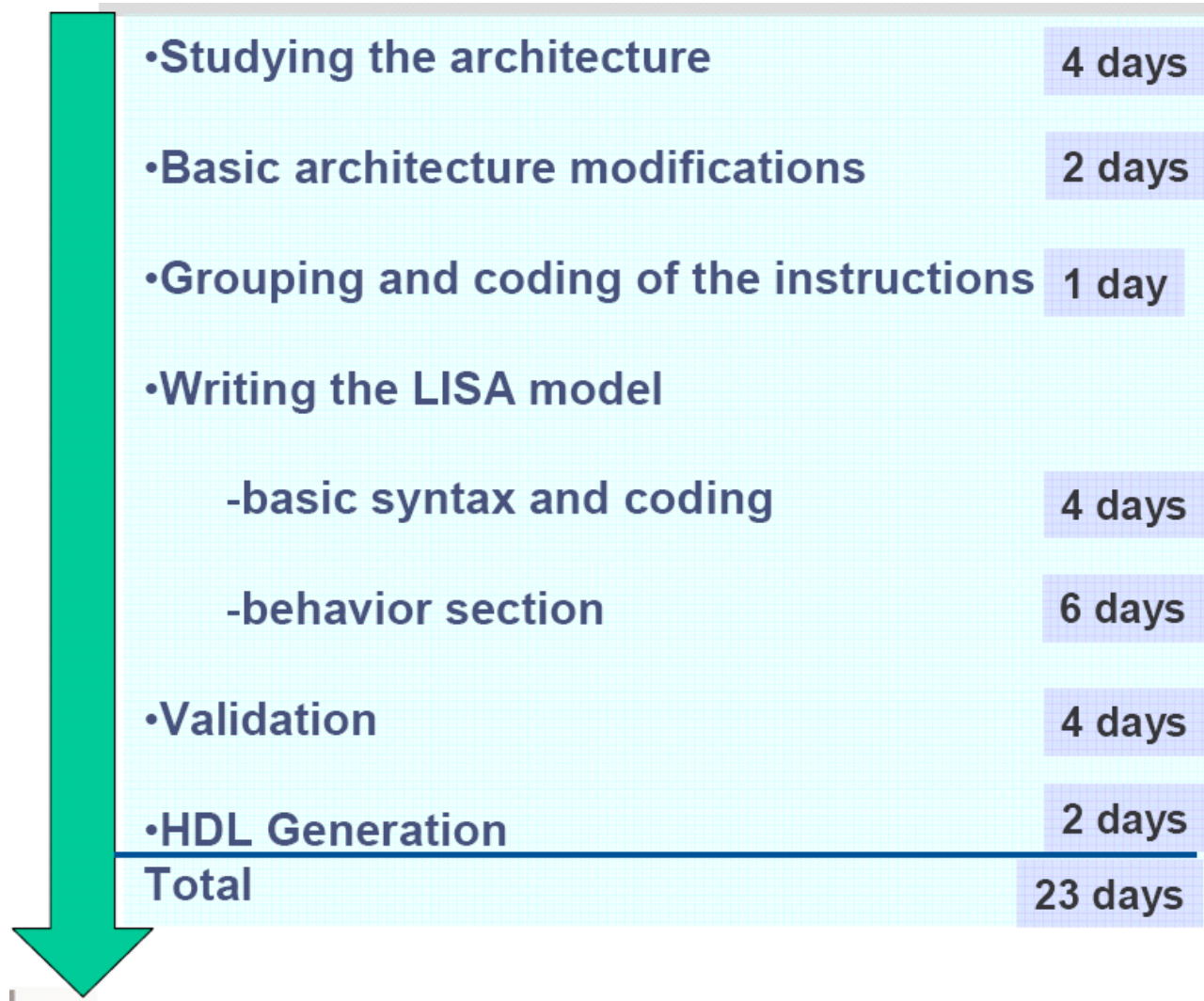
Motorola M65HC11 Architecture

Developing with LISA



Motorola M65HC11 Architecture

Developing with LISA

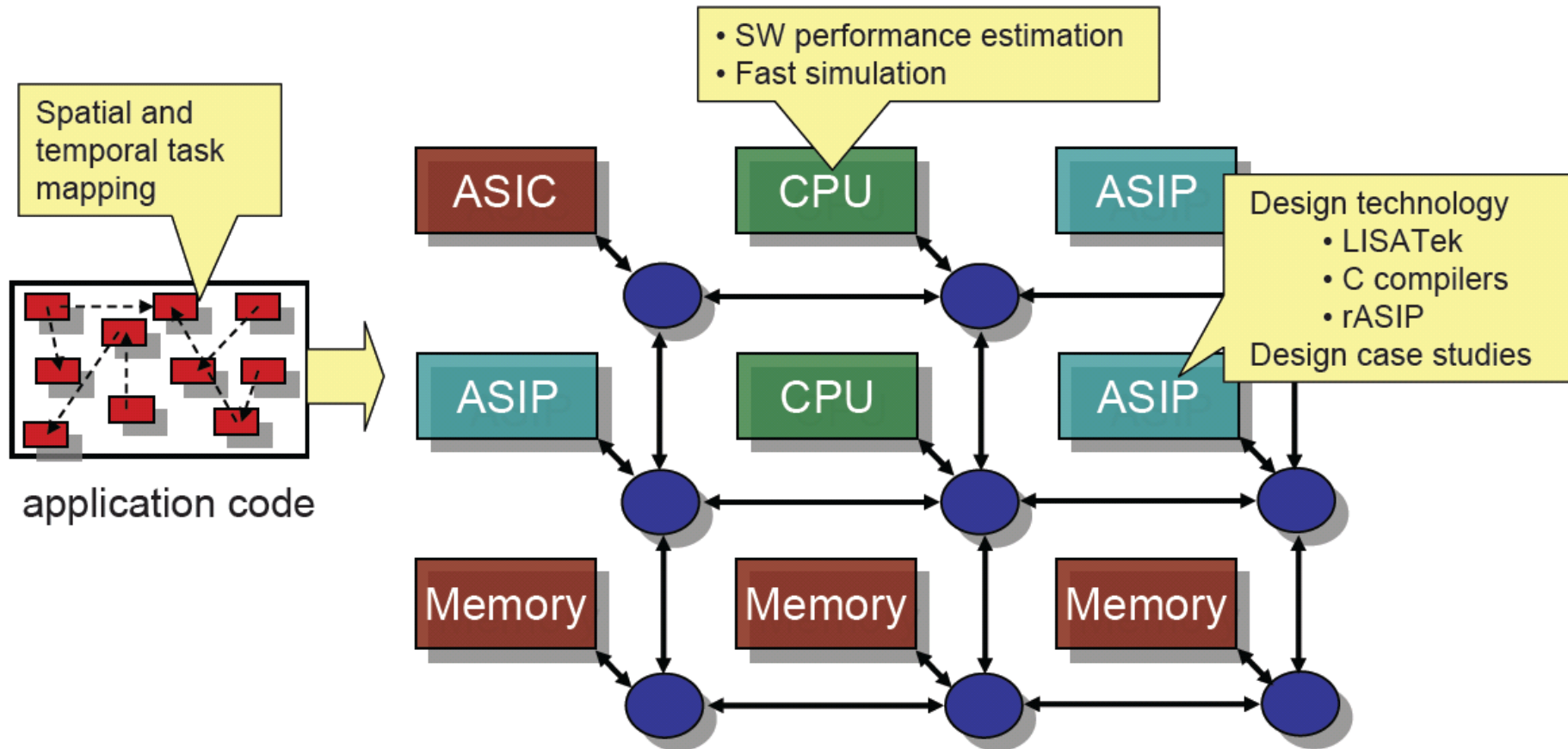


Case Studies (by LISATek)

- **Texas Instruments TMS320C6201**
 - cycle-accurate model
 - 9978 lines of LISA 2.0 and C code
 - design effort: 6 weeks
- **Analog Devices ADSP21xx**
 - cycle-accurate model
 - 11000 lines of LISA 2.0 and C code
 - inexperienced, undergraduate student
(neither knowledge on DSP nor on LISA)
 - design effort: 8 weeks
- **Advanced Risc Machines ARM 7 Core**
 - instruction-accurate model
 - 4000 lines of LISA 2.0 and C code
 - design effort: 2 weeks

(source: LISATek)

MPSoC: Exploration and Optimization



Research Activities

Extensible Processor

❑ Focal points

- ❑ Automating the process of selecting an appropriate instruction set given an embedded application
- ❑ Tasks: a) Autom. selecting appropriate code segments b) Autom. matching code segments to application-specific instructions c) Autom. adapting parameters of embedded processors to embedded applications, ...

❑ Some research approaches

- ❑ Sun/Raghunathan/Jha => automated design space exploration with custom-designed application specific instructions [Fei03]
- ❑ Cheung/Parameswaran/Henkel => Library-based approach to automatically selecting application-specific instructions given an embedded applications [INS03]
- ❑ Other research groups P. Iene, L. Pozzi, P. Brisk, T. Mitra, ...
- ❑ see following conferences: DATE, DAC, ICCAD, ESWeek, ...

Summary: Embedded Processor Platforms

- Silicon complexity allows for complex, whole SOC
- Customizable Processor HW platforms come in different flavors:
 - Configurable processor cores: parameters
 - Extensible instruction set
 - Fully customized instruction set
- Customizable Processor SW platforms:
 - Integration, optimization, estimation
 - Some platforms offer customized high-level tools that allow immediate evaluation of new parameters/instructions
- Customizable Processor platforms typically require new silicon masks as opposed to FPGA-based platforms but are not limited in silicon size
- Future: more complex platforms allowing heterogeneous multiprocessors on a single chip

References and Sources

- ❑ [Leu00] Leupers, R.; Code Optimization Techniques for Embedded Processors, Kluwer, 2000.
- ❑ [LeHe02] Lekatsas, H.; Henkel, J.; Jakkula, V. 1-cycle code decompression circuitry for performance increase of Xtensa-1040-based embedded systems, Custom Integrated Circuits Conference, 2002. Proceedings of the IEEE 2002 , Pages:9 – 12, 12-15 May 2002.
- ❑ A. Hoffman et al., “A Novel Methodology for the Design of Application-Specific Instruction Set Processors (ASIPs) Using a Machine description Language”, IEEE Tr on CAD, Vol. 20, No. 11, Nov 2001.
- ❑ O. Schliebusch, A. Hoffman, A. Nohl, G. Braun, H. Meyr, “Architecture Implementation Using the Machine Description Language LISA”, Proc of 15th Int. Conference on VLSI Design, 2002.
- ❑ [Fei03] Y. Fei, S. Ravi, A. Raghunathan, and N. Jha, “Energy estimation for extensible processors,” in DATE, 2003.
- ❑ [INS03] Cheung, N.; Parameswaran, S.; Henkel, J INSIDE: INstruction selection/identification & design exploration for extensible processors, Computer Aided Design, 2003. ICCAD-2003. International Conference on , 9-13 Nov. 2003, Pages:291 – 297.
- ❑ R. Schreiber, A. Aditya, S. Mahlke et al., “Pico-NPA: High-Level Synthesis of Nonprogrammable Hardware Accelerators”, HPL-2001-249, Oct., 2001.
- ❑ Tensilica, <http://www.tensilica.com>
- ❑ LisaTek, <http://www.lisatec.com>
- ❑ <http://www.siliconstrategies.com>

Extra Slides

Prof. Dr. J. Henkel, Dr. M. Shafique

CES - Chair for Embedded Systems

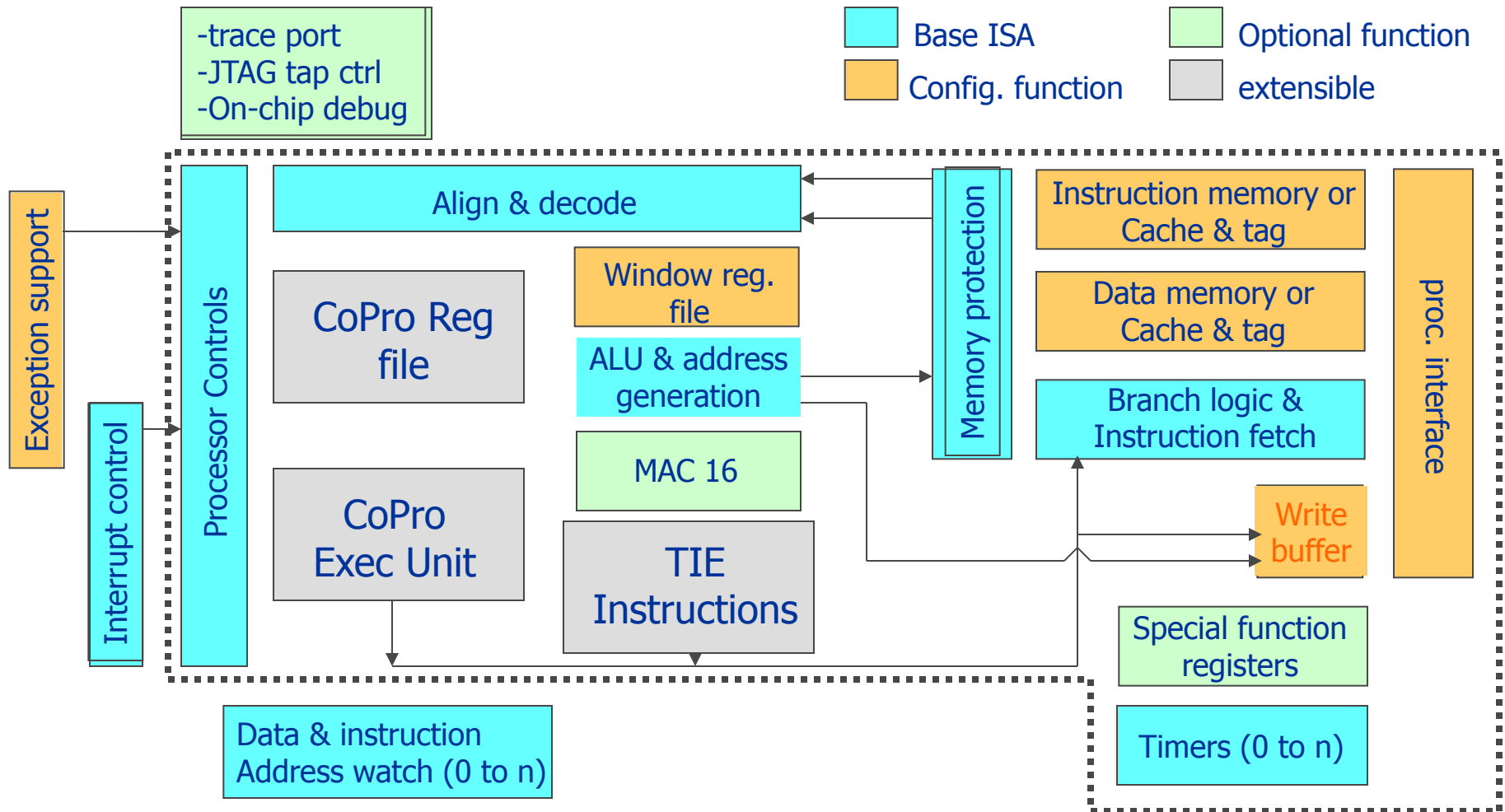
Karlsruhe Institute of Technology, Germany

Today: Embedded Processor Platforms

ASIPs and Instruction Set Extensions

General Xtensa Architecture

base ISA, configurable, optional, extensible

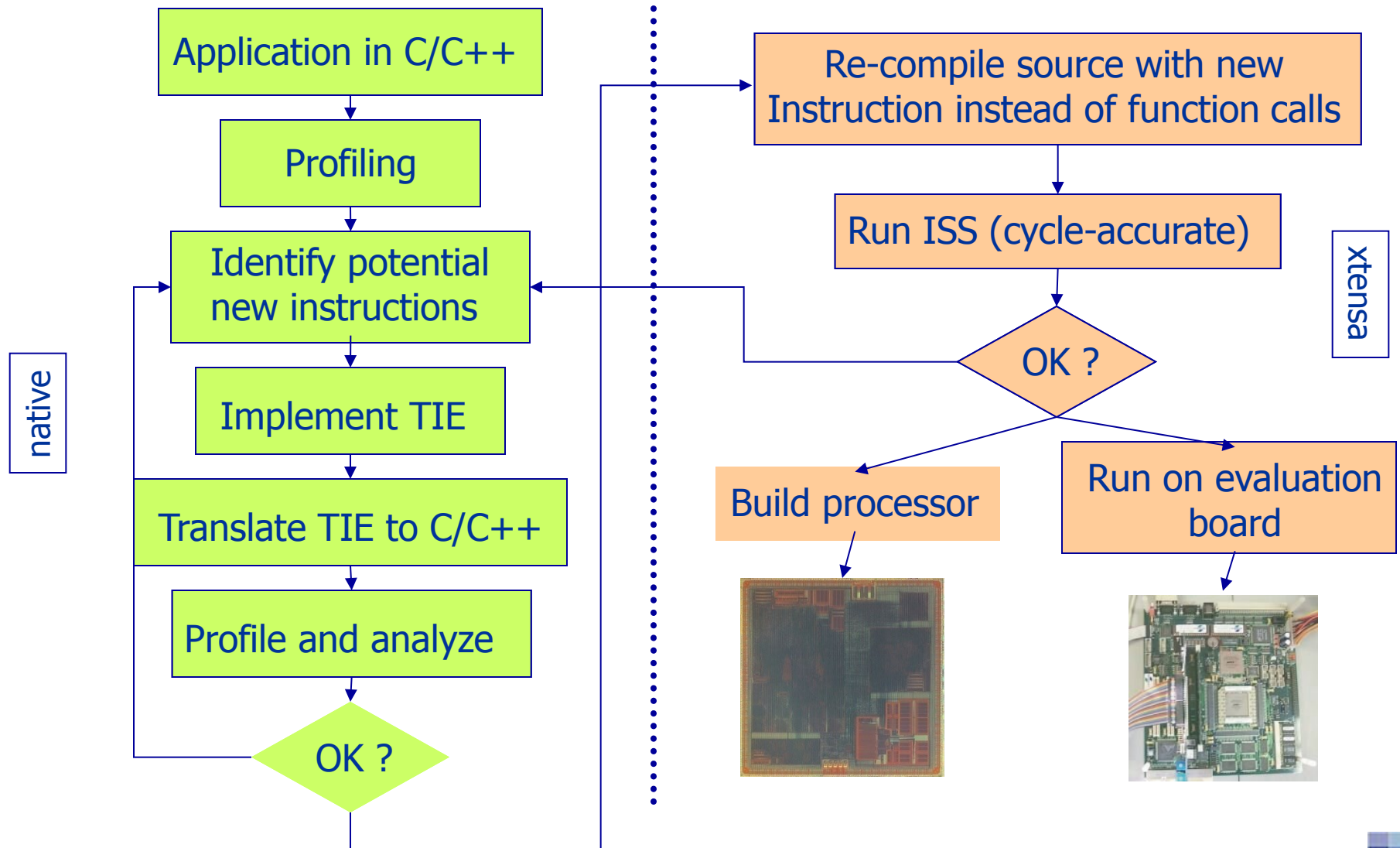


Example: configuring Xtensa

Target Options	
geometry/process	
frequency	[MHz}
power saving	y/n
register file impl.	
...	
Instruction options	
16-bit MAC	y/n
16-bit multiplier	y/n
...	
Types and # of interrupts	
# of timers	
Byte ordering	b/l endian
Registers for call windows	#
Processor interface (r/w)	width
Instruction Cache	
associativity	e.g. direct
cache organization	e.g. 4096x32
tag RAM addr x data width	e.g. 512x19
Debugging	
full scan	y/n
instruction ads break reg.	#
TIE Xtension	yes/no
TIE source	e.g. ./sample.tie
Board support	
...	

- ❑ Mixed level of configurability:
 - ❑ Fixed options that can be added or omitted (y/n)
 - ❑ Configuration of device parameters: sizes of caches, ...
 - ❑ Fully customized extensions to the instruction set: TIE

Design Flow for building TIE instructions



Example for TIE coding

```

state ANS2 32
user_register ans2 0 {ANS2}

opcode FREXP op2 = 4'b0001 CUST0
opcode LDEXP op2 = 4'b0010 CUST0

iclass frexp {FREXP} {out arr, in ars} {out ANS2}
iclass ldexp {LDEXP} {out arr, in art, in ars}

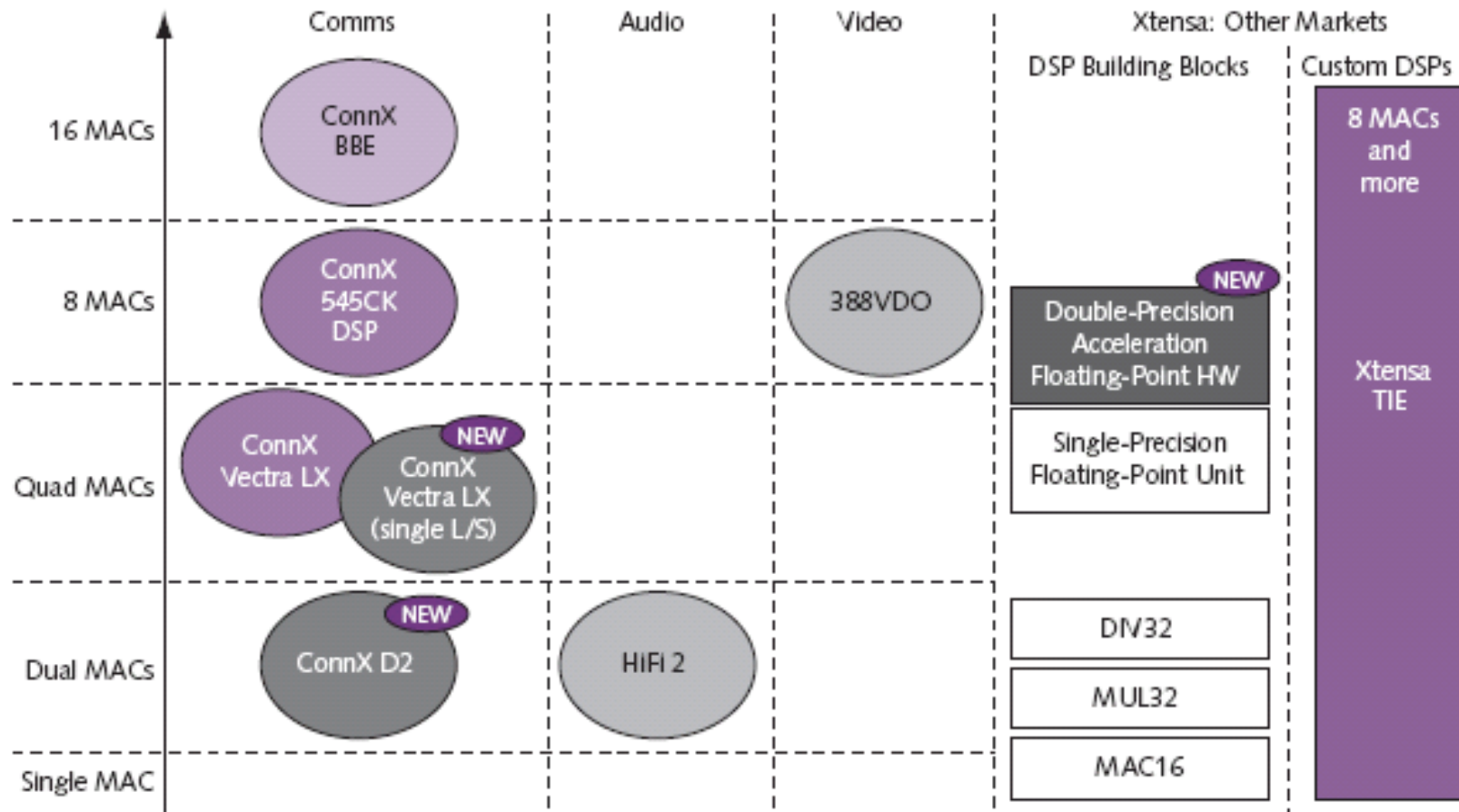
reference FREXP {
    wire [31:0] temparr;
    wire [31:0] tempans2;
    assign temparr = (ars[30:23] == 0 && ars[22:0] == 0) ? 32'b0
        : {ars[31], 8'b01111110, ars[22:0]};
    assign tempans2 = (ars[30:23] == 0 && ars[22:0] == 0) ? 32'b0
        : {24'b0, ars[30:23] - 127 + 1} ;
    assign arr = (tempans2[0] == 1)
        ? {temparr[31], temparr[30:23] + 1'b1, temparr[22:0]}
        : temparr ;
    assign ANS2 = (tempans2[0] == 1) ? (tempans2 - 1) >> 1 : tempans2 >> 1;
}

reference LDEXP {
    assign arr = {art[31], art[30:23] + ars, art[22:0]};
}

```

sqrt.tie

Tensilica's DSP Line



(source: Tensilica Tweaks Xtensa @ Microprocessor'09)

Performance Results for Xtensa LX3

	Tensilica Xtensa LX3	Tensilica Xtensa LX2	LX3 Difference
Optimized for Speed			
Core Frequency	384MHz	312MHz	+23%
Core Area	0.538mm ²	0.479mm ²	+12%
Core Power	150μW / MHz	186μW / MHz	-19%
Optimized for Area and Power			
Core Frequency	56MHz	56MHz	—
Core Area	0.281mm ²	0.317mm ²	-11%
Core Power	89μW / MHz	151μW / MHz	-41%

Tensilica Xtensa LX3	TSMC 40nm-LP	TSMC 40nm-LP	TSMC 45nm-GS	TSMC 45nm-GS
Minimum Configuration				
Synthesis Optimization	Low power	High speed	Low power	High speed
Core Frequency	60MHz	670MHz	62MHz	>1.0GHz
Core Area	0.024mm ²	0.044mm ²	0.024mm ²	0.044mm ²
Core Power	0.012mW / MHz	0.018mW / MHz	0.009mW / MHz	0.014mW / MHz
Core Power @ Freq	0.72mW	12.1mW	0.6mW	14.4mW
Typical Configuration				
Synthesis Optimization	Low power	High speed	Low power	High speed
Core Frequency	57MHz	493MHz	58MHz	780MHz
Core Area	0.163mm ²	0.295mm ²	0.158mm ²	0.283mm ²
Core Power	0.046mW / MHz	0.093mW / MHz	0.034mW / MHz	0.066mW / MHz
Core Power @ Freq	2.62mW	45.8mW	2.0mW	51.5mW

(source: Tensilica Tweaks Xtensa @ Microprocessor'09)

Feature Comparison

Feature	Tensilica Xtensa LX3	ARM Cortex-A5	MIPS MIPS32 74K	MIPS MIPS32 24KE	Virage ARC 750D
Architecture	Xtensa LX3	ARMv7-A	MIPS32-R2	MIPS32-R2	ARCompact
Integer Pipeline	5 or 7 stages In-order 1-way	8 stages In-order 1-way*	15 stages Out-of-order 2-way	8 stages In-order 1-way	7 stages In-order 1-way
Branch Predict	—	Dynamic	Dynamic	Dynamic	Dynamic
L1 Cache	0–32K each ECC	4K–64K each	0–64K each	0–64K each	8K–64K each
L2 Cache	—	Optional 16K–8MB	Optional	Optional	—
Scratchpad RAM	Optional Up to 495GB	—	Optional Up to 1MB	Optional Up to 1MB	8K–512K (code) 8K–256K (data)
16-Bit Instructions	Yes	Thumb, Thumb-2	MIPS16e	MIPS16e	ARCompact
DSP Extensions	Vectra LX, ConnX D2, ConnX BBE	Optional Neon	MIPS DSP ASE-2	MIPS DSP ASE-1	ARC XY Memory Subsystem
Java Extensions	—	Jazelle DBX, RCT	—	—	—
Custom Instructions	Yes	—	Yes	Yes	Yes
System Interface	AMBA-3 AXI 2 x 8–1,024 bits or AHB-Lite 2 x 8–1,024 bits	AMBA-3 AXI 1 x 64 bits Optional 2 x 64 bits (multiprocessor)	OCP 2.1 2 x 64 bits (Optional bridges)	OCP 2.1 2 x 64 bits (Optional bridges)	BVCI, AHB, AXI 1 x 32 bits or 1 x 64 bits
FPU	Optional SP or DP	Optional VFPv3 (DP)	Optional SP + DP	Optional SP + DP	Optional SP or DP
Memory Management	Optional MMU + TLB	MMU + TLB	Optional MMU + TLB	Optional MMU + TLB	MMU + TLB
Privilege Levels	4	2 + TrustZone	3	3	2
Core Frequency (Max)	>1.0GHz (45nm-GS)	480MHz–1.0GHz (40nm-LP, 40nm-G)	1.7GHz (40nm-G)	1.45GHz (40nm-G)	700MHz (90nm-GT)
Core Area @ Max Freq	0.044mm ² (45nm-GS)	0.27mm ² (40nm-LP)	0.78mm ² (40nm-G)	0.34mm ² (40nm-G)	0.53mm ² (90nm-LP)
Dhrystone 2.1	1.52Dmips / MHz	1.57Dmips / MHz	2.0Dmips / MHz	1.55Dmips / MHz	>1.5 Dmips / MHz
Power (typical)	0.014mW / MHz (45nm-GS)	0.12mW / MHz (40nm-LP)	< 0.25mW / MHz (40nm-G)	0.125mW / MHz (40nm-G)	0.12mW / MHz (90nm-LP)
Power Efficiency (Dmips / mW)	108.5Dmips / mW (45nm-GS)	13.0Dmips / mW (40nm-LP)	8.0Dmips / mW (40nm-G)	12.4Dmips / mW (40nm-G)	12.5Dmips / mW (90nm-LP)
Introduction	2009	2009	2007	2005	2004

Some TIE features

- ❑ Schedule sections:
 - ❑ Specifies implementation at the micro-architectural level (all others are ISA related)
 - ❑ Technique to define instruction that use more than one cycle (important for relaxing cycle time)
 - ❑ Example: one or more op code with same I/o spec can be grouped into one schedule

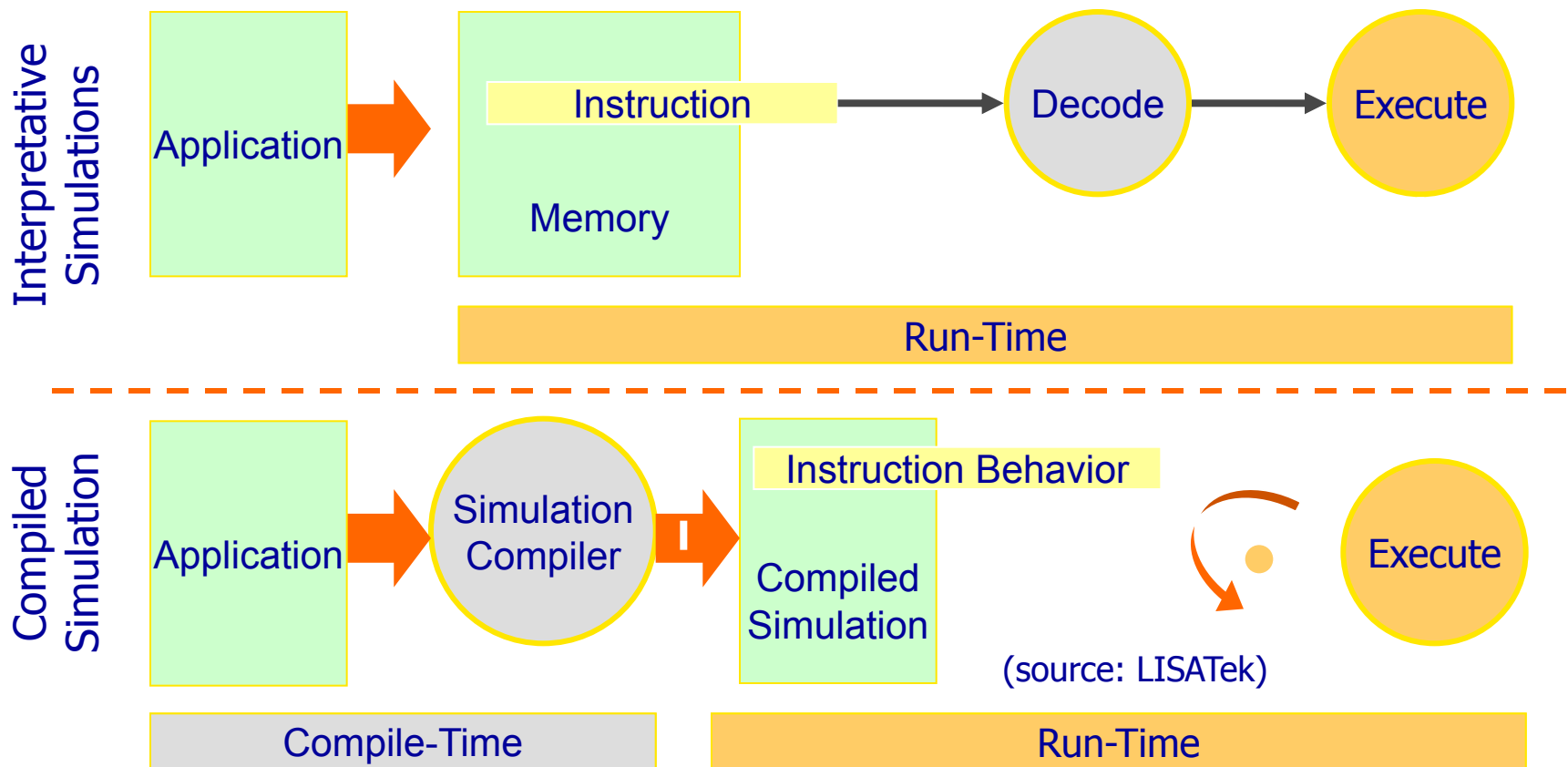
LISATek

- Multiple instruction words
 - Instructions that need multiple words like TMS320C54x DSP
 - E.g.: 1 or 2 or 3 words (2nd and 3rd carries mostly immediates)
 - Shown: coding root implemented as switch
- Non-orthogonal coding
 - Expressed by additional conditional statements
 - Purpose: express coding dependencies between different operations
 - Mode for add (C62x), sub, mul: selects between short and long operands and their specific arithmetic
 - ls, st: mode used for different purpose
 -

Simulation Approaches





Combining both paradigms in Lisatek:

- Just-In-Time Cache-Compiled Simulation (JIT-CCS)

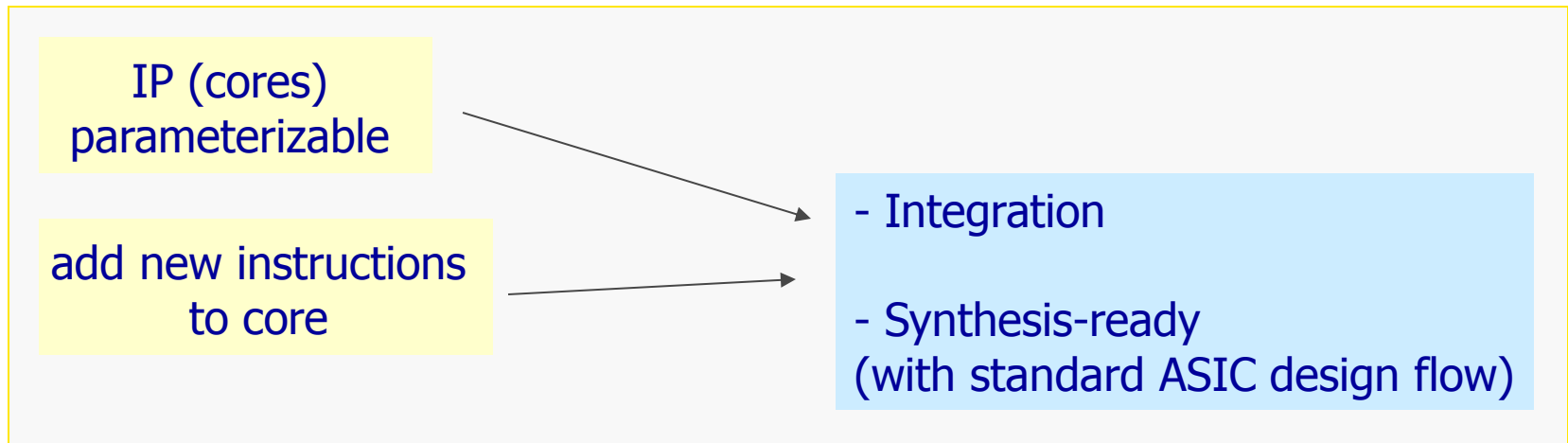


The Improv Platform

Contents

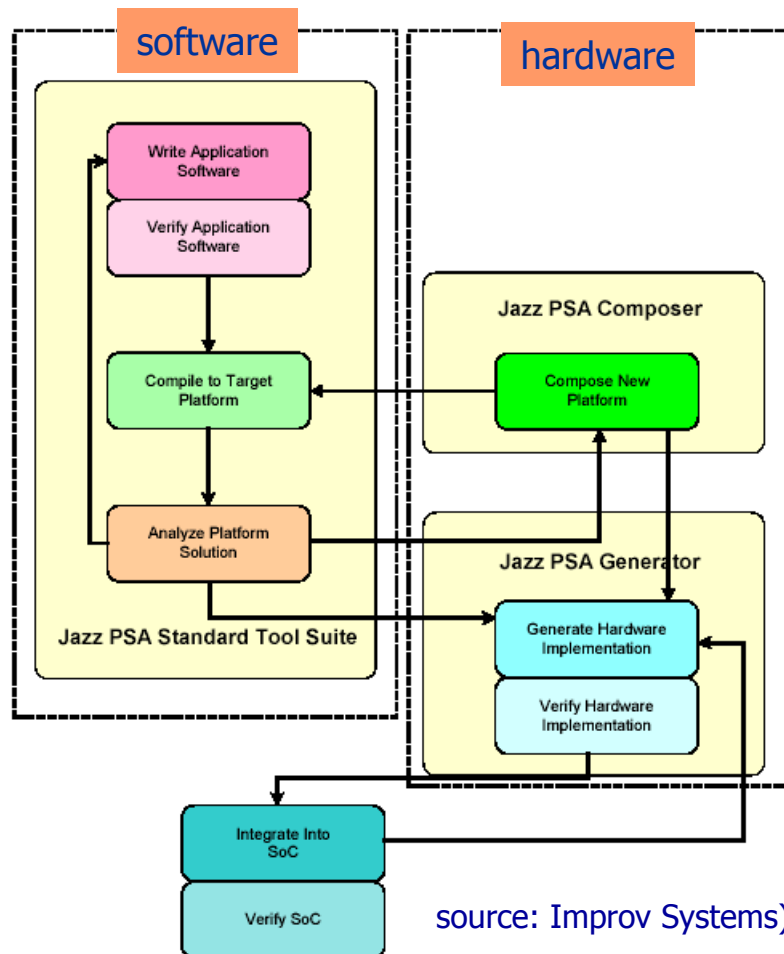
-  Summary of main Features
-  Platform
-  DSP core
-  Misc

Paradigm of the Improv Platform



- ❑ Modify/extend Instruction Set architecture
- ❑ Targets DSP oriented applications

Improv Platform



source: Improv Systems)

□ The components of the platforms

□ Composer:

- Facilitates instruction extension and adding new instructions; interrupts; memory ...

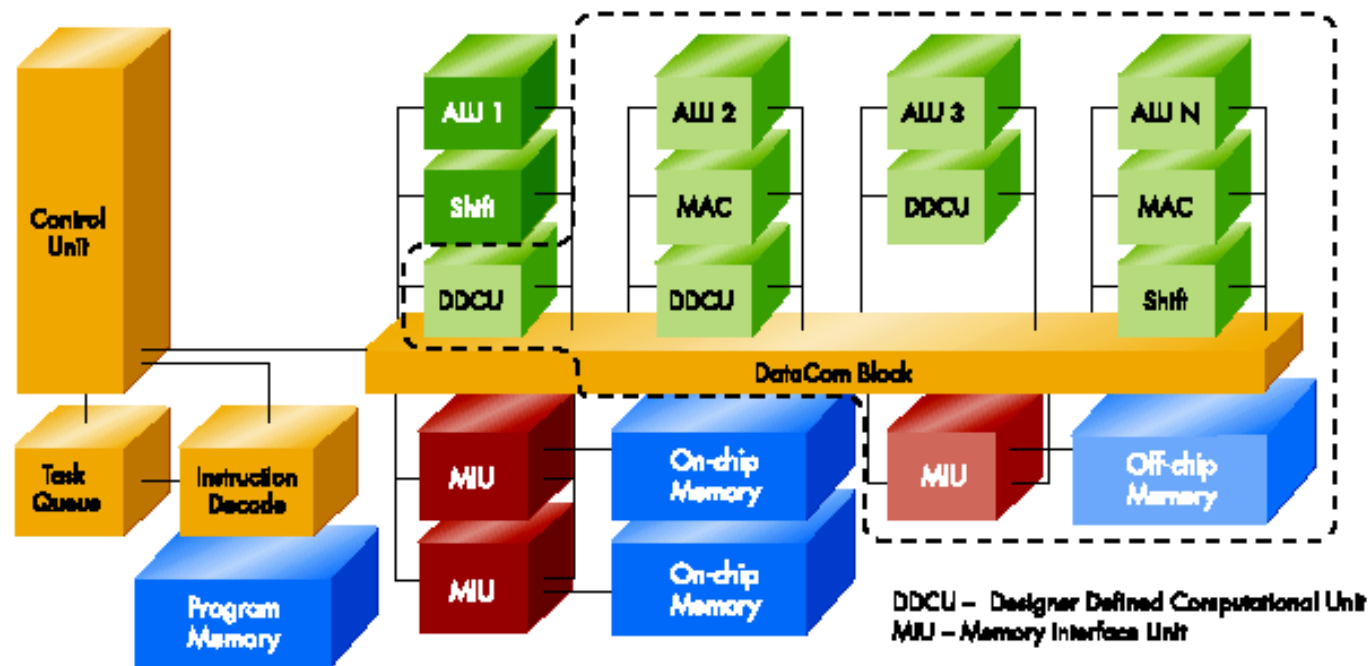
□ Generator:

- Interprets configuration from composer
- Generates RTL Verilog description as input for a standard ASIC design flow
- The RTL instances generated are verified and read from a data base and not automatically generated

DSP Processor (“Jazz DSP”)

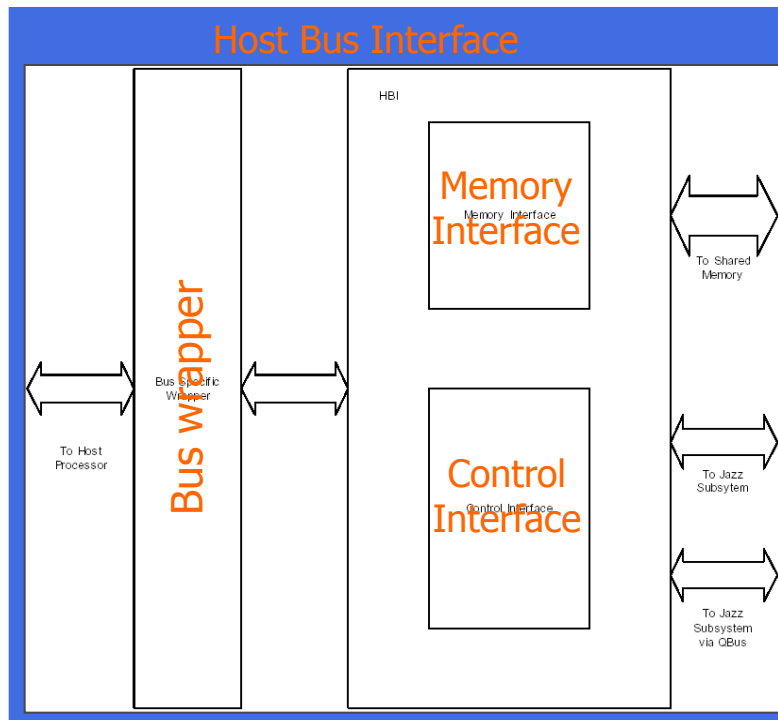
- ❑ Configurable VLIW architecture
 - ❑ user chooses data path (16-bit or 32-bit wide)
 - ❑ User can extend the ISA: either through as an option (like inclusion or non-inclusion of **multiplier, MAC** etc.) or by defining **custom instructions** and **custom functional units**; dual-operand load instructions; 40/64-bit accumulator; ...
 - ❑ Memory: instruction and data memories can be configured
 - ❑ Furthermore: interrupts (number and priority levels), system memory addresses etc.
- ❑ Features:
 - ❑ Power: < 0.1mW/MHz @ 0.13 micron
 - ❑ Chip size: <0.25mm² @ 0.13 micron
 - ❑ Performance: > 1000 MOPS @ 100 MHz
- ❑ Misc architectural features:
 - ❑ Distributed register files to avoid I/O bottleneck from and to FUs
 - ❑ 2-stage instruction pipeline
 - ❑ Single-cycle execution units

Jazz DSP architectural block diagram



(source: Improv Systems)

Further IP of the Improv platform



■ Time slot interchange block

- Managing voice data; interfaces to time-division-multiplexed PCM highways
- Configurable: #channels, type/speed/width of each highway

■ Host Bus interface





- memory mapped interface with address, data and control signal
- Interfaces between host processor and Jazz system
- Wrapper (bus specific) facilitates interfacing to standard bus systems like AHB, PCI, ...
- Control IF: manages task queuing via Qbus within the Jazz subsystem

■ Data Channel Interface

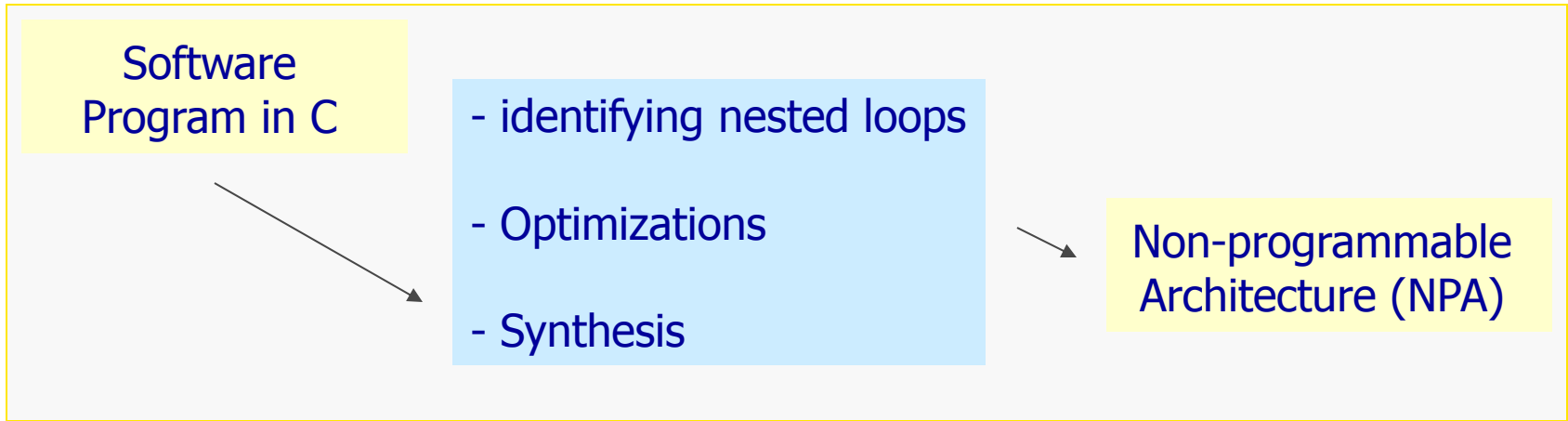
- Provides data flow management between host bus IF local IF
- Contains configurable 1 to N full-duplex channels
- User-defined data filters
- Interfaces to standard buses like AMBA

HP's Pico Platform

Overview

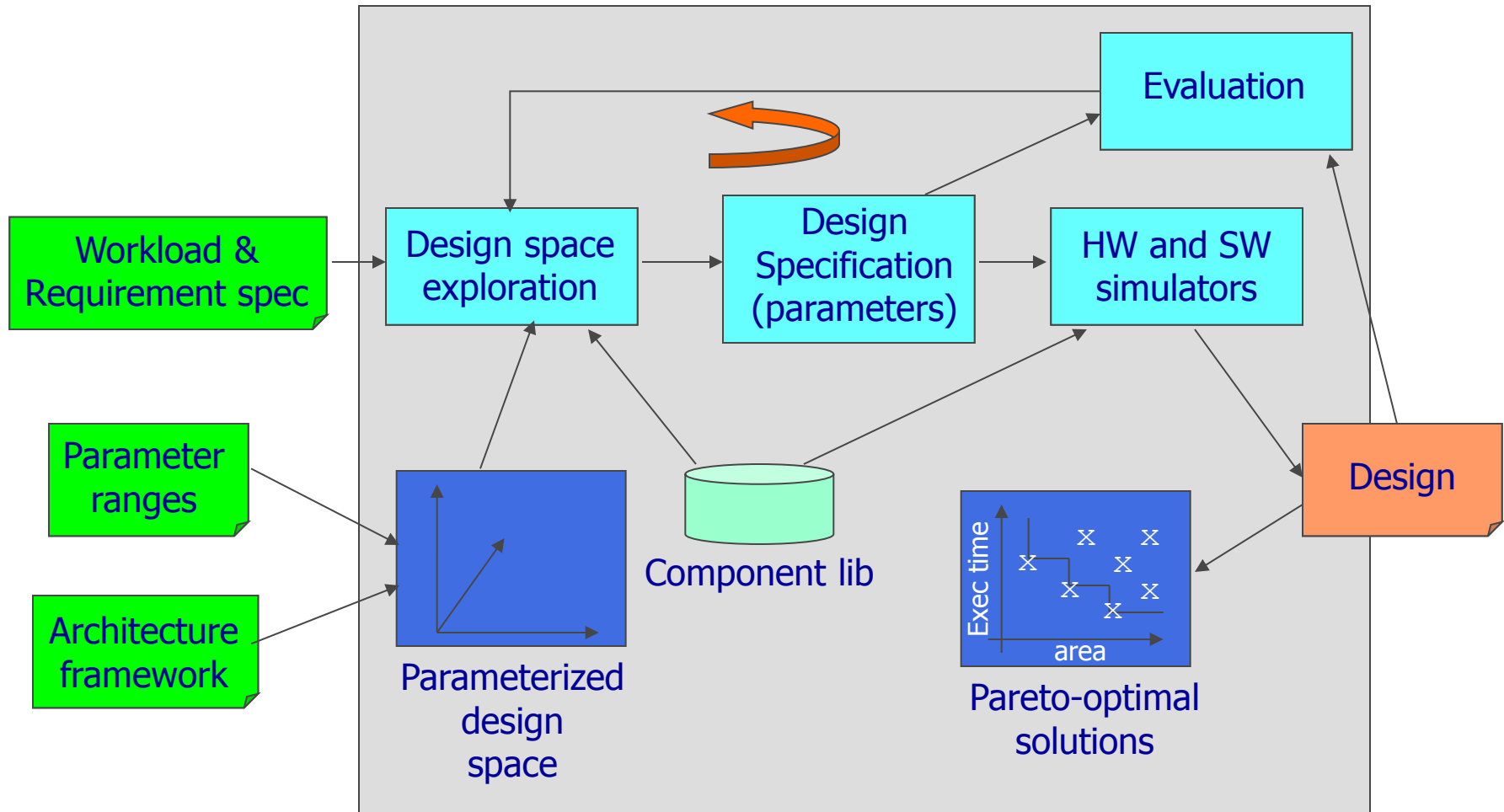
-  Design paradigm
-  Front-end optimization
-  Architecture
-  Synthesis and design flow

Paradigm of Pico (NPA)



- ❑ **PICO: Program In, Chip Out**
- ❑ Nested loops are identified and the hot spots are synthesized as hardwired, non-programmable hardware
- ❑ Output is a co-processor that can be used in conjunction with standard processor
- ❑ Aim is a low cost-design, low-cost production and high performance
- ❑ Status: research project

Design Paradigm



Front-end transformations & optim.

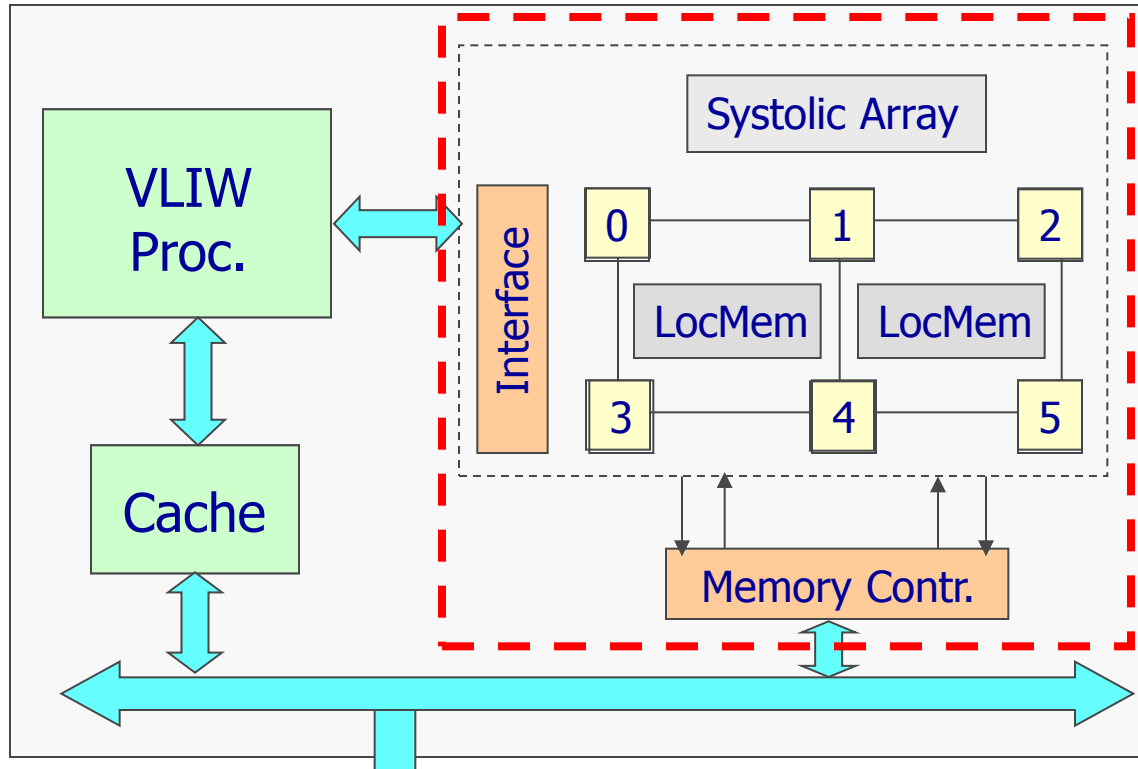
```
For (j1=0;j1<8192;j1++) {  
    j[j1]=0;  
    For (j2=0;j2<16;j2++)  
        y[j1]=y[j1]+w[j2]*x[j1+j2];  
}  
...
```



```
For (j1=0;j1<8192;j1++) {  
    For (j2=0;j2<16;j2++)  
        y[j1]=y[j1]+w[j2]*x[j1+j2];  
}  
...
```

- Create loops such that there is no code other than a single embedded “for” in the body of any but the innermost loop
- Abstract architecture specification
 - Pipelined implementation of loop nest; several iterations may be active
- Tiling and mapping
- Dependence analysis
- Iteration mapping
- Iteration schedules
- Load/store elimination for uniform dependence arrays
- ...

Generic Architecture of Pico



- Generic architecture:
 - VLIW,
 - Cache,
 - Bus system **are fixed**
 - Processor array
 - Array controller
 - Local memories
 - Interface to global memory
 - Control & data interface to host **are synthesized**
- System generates **structural, synthesizable RTL**

Synthesis Phases: from SW loop to HDL

❑ Analysis Phase

- ❑ Search for array access and dependencies



❑ Mapping and Scheduling

- ❑ Nested loops are mapped to processors and scheduled



❑ Loop Transformations

- ❑ Transform to an outer sequential loop and inner parallel loops



❑ Optimization at operation-level

- ❑ Word-width minimization and classical optimizations



❑ Processor Synthesis

- ❑ Allocation of FUs and scheduling of operations relative to loop start time



❑ System Synthesis

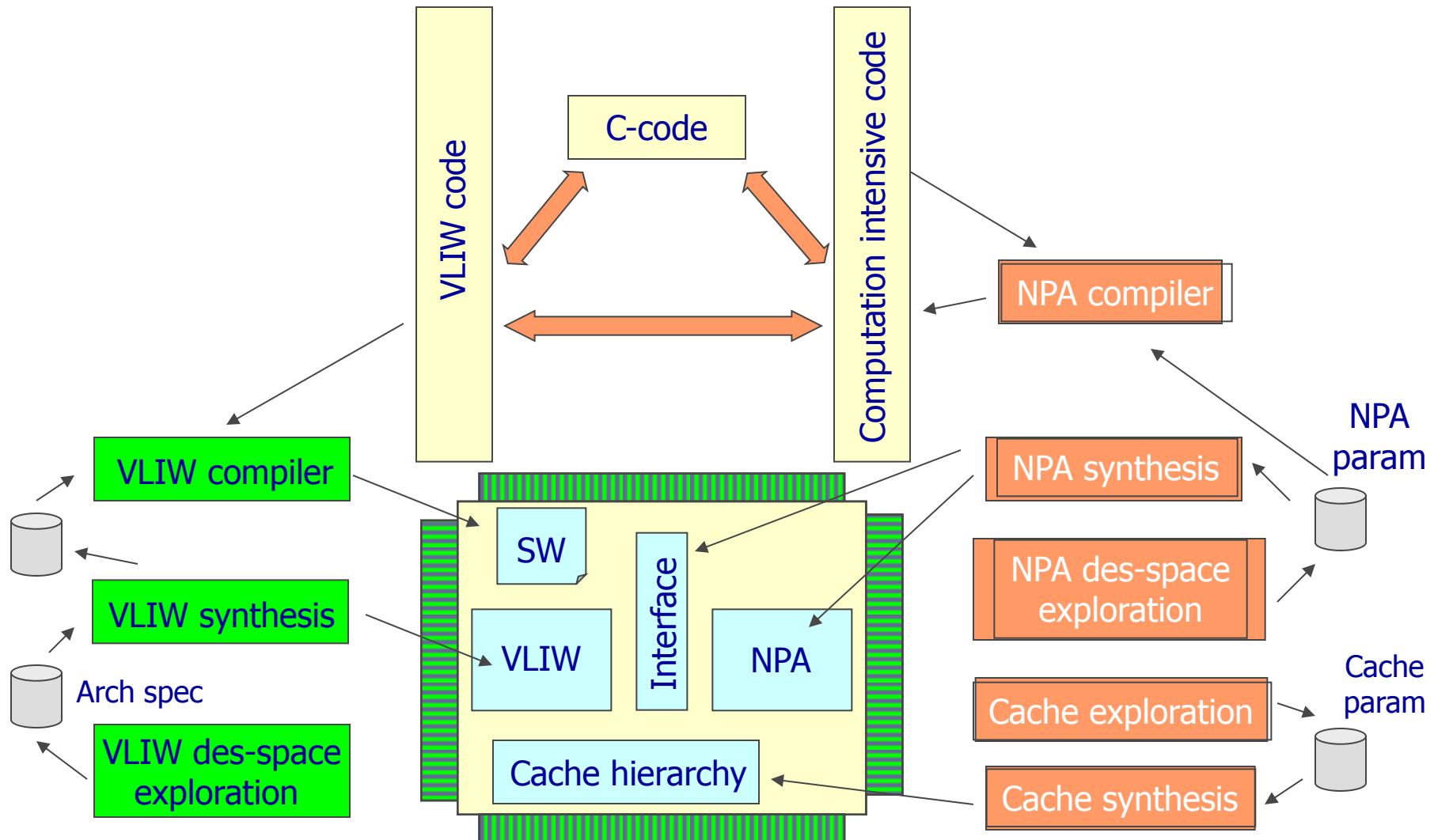
- ❑ Allocation of processors and their interconnect; controller and data interfaces



❑ Output:

- ❑ HDL description and cost estimation

The whole Pico design flow



Others: MIPS32 M4K

- ❑ Recently announced 32-bit synthesizable RISC core
- ❑ Aimed at future SOC's that integrate multiple cores on one chip to satisfy higher bandwidth in networking/broadband
- ❑ Features instruction set extension (first in terms of an extension for an already existing industry standard)
- ❑ Based on the enhanced MIPS32 architecture (faster and more flexible packet processing, ...)
- ❑ More features:
 - ❑ Power: 0.1mW/MHz
 - ❑ Area: 0.3mm²
 - ❑ ~300MHz
 - ❑ 0.13 micron process

References and Sources

- ❑ Improv Systems, <http://www.improvsys.com>
- ❑ HP Pico: <http://www.hpl.hp.com/research>
- ❑ <http://www.siliconstrategies.com>