Chapter 2-3: CPUs

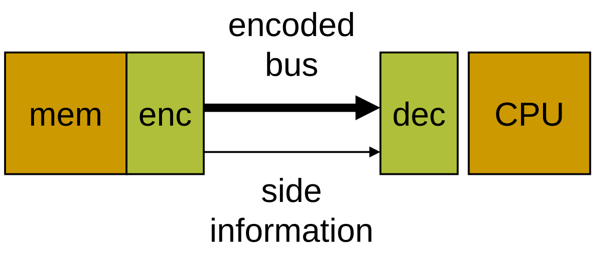
Soo-Ik Chae

Topics

- Bus encoding.
- Security-oriented architectures.
- CPU simulation.
- Configurable processors.

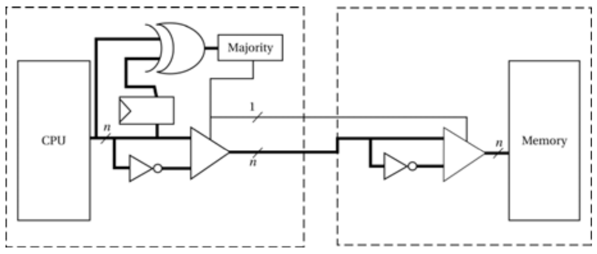
Bus encoding

- Encode information on bus to reduce toggles and dynamic energy consumption.
  - Count energy consumption by toggle counts.
- Bus encoding is invisible to rest of architecture.
- Some schemes transmit side information about encoding.



Bus-invert coding

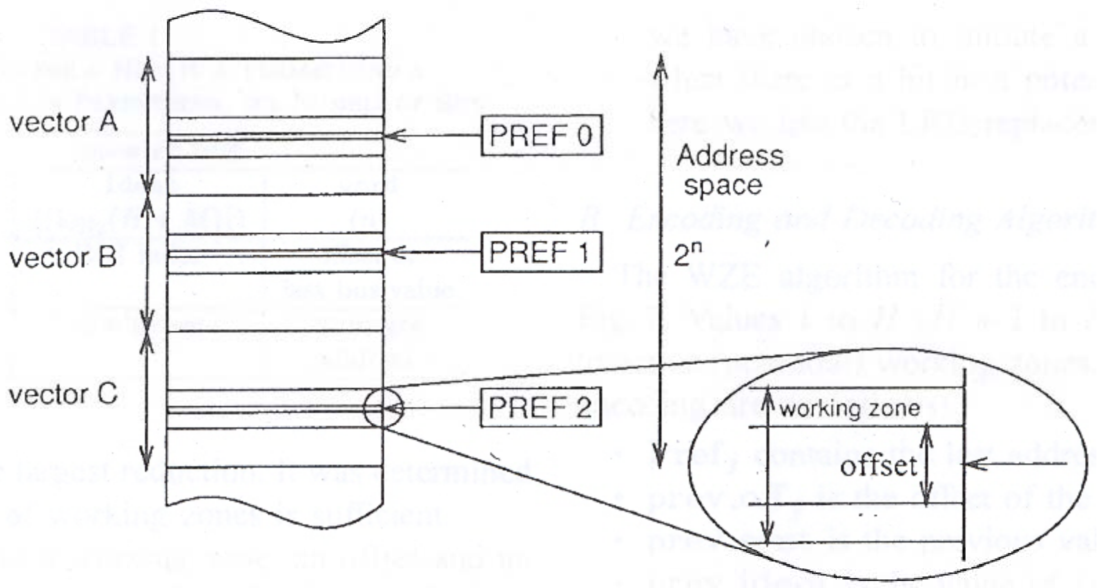
- Stan and Burleson: take advantage of correlation between successive bus values.
- Choose sending true or complement form of bus values to minimize toggles.
- Can break bus into fields and apply bus-invert coding to each field.



Working zone encoding

- Mussoll et al.: working-zone encoding divides address bus into working zones.
  - Address in a working zone is sent as an offset from the base in a one-hot code.
- To reduce the energy in microprocessor address bus

Working zone encoding



Working zone encoding

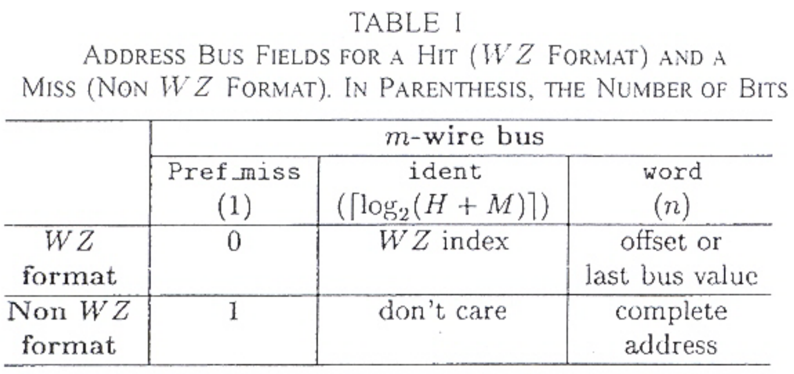


TABLE I
ADDRESS BUS FIELDS FOR A HIT (WZ FORMAT) AND A MISS (NON WZ FORMAT). IN PARENTHESIS, THE NUMBER OF BITS

	m-wire bus		
	Pref_miss (1)	ident ($\lceil \log_2(H+M) \rceil$)	word (n)
WZ format	0	WZ index	offset or last bus value
Non WZ format	1	don't care	complete address

Working zone encoding

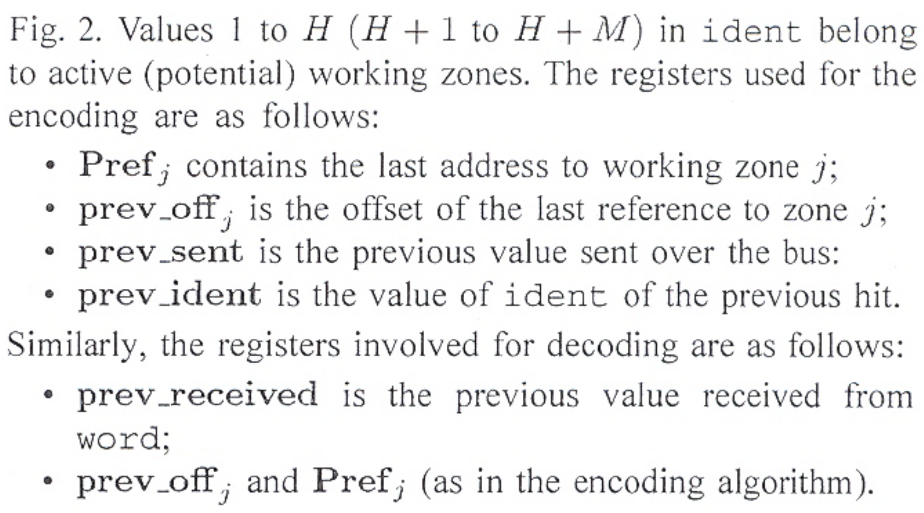


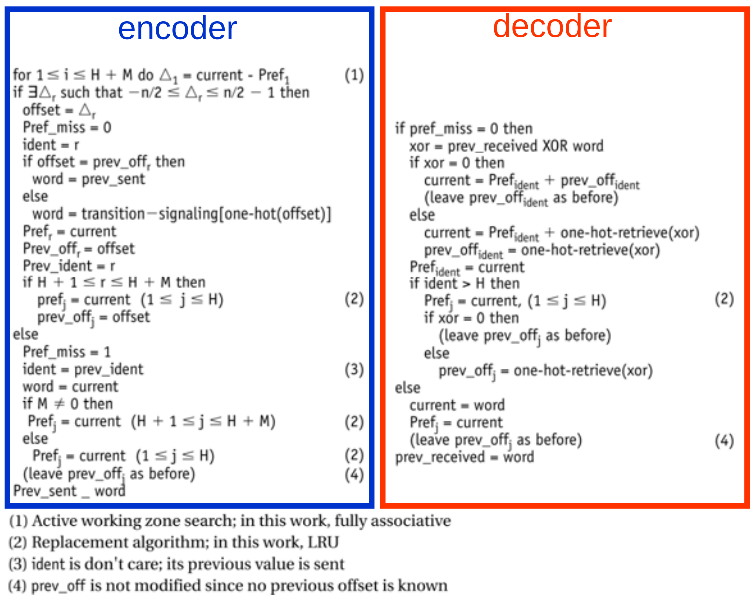
Fig. 2. Values 1 to H ($H+1$ to $H+M$) in ident belong to active (potential) working zones. The registers used for the encoding are as follows:

- **Pref**$_j$ contains the last address to working zone j;
- **prev_off**$_j$ is the offset of the last reference to zone j;
- **prev_sent** is the previous value sent over the bus:
- **prev_ident** is the value of ident of the previous hit.

Similarly, the registers involved for decoding are as follows:

- **prev_received** is the previous value received from word;
- **prev_off**$_j$ and **Pref**$_j$ (as in the encoding algorithm).

Working zone encoding



encoder

```
for $1 \leq i \leq H + M$ do $\triangle_1$ = current - $Pref_1$          (1)
if $\exists\triangle_r$ such that $-n/2 \leq \triangle_r \leq n/2 - 1$ then
  offset = $\triangle_r$
  Pref_miss = 0
  ident = r
  if offset = $prev\_off_r$ then
    word = prev_sent
  else
    word = transition−signaling[one-hot(offset)]
  $Pref_r$ = current
  $Prev\_off_r$ = offset
  Prev_ident = r
  if $H + 1 \leq r \leq H + M$ then
     $pref_j$ = current  $(1 \leq j \leq H)$                         (2)
     $prev\_off_j$ = offset
else
  Pref_miss = 1
  ident = prev_ident                                                 (3)
  word = current
  if $M \neq 0$ then
   $Pref_j$ = current  $(H + 1 \leq j \leq H + M)$                   (2)
  else
    $Pref_j$ = current  $(1 \leq j \leq H)$                          (2)
  (leave $prev\_off_j$ as before)                                    (4)
Prev_sent _ word
```

decoder

```
if pref_miss = 0 then
   xor = prev_received XOR word
   if xor = 0 then
      current = $Pref_{ident}$ + $prev\_off_{ident}$
      (leave $prev\_off_{ident}$ as before)
   else
      current = $Pref_{ident}$ + one-hot-retrieve(xor)
      $prev\_off_{ident}$ = one-hot-retrieve(xor)
   $Pref_{ident}$ = current
   if ident > H then
      $Pref_j$ = current, $(1 \leq j \leq H)$                       (2)
      if xor = 0 then
         (leave $prev\_off_j$ as before)
      else
         $prev\_off_j$ = one-hot-retrieve(xor)
else
   current = word
   $Pref_j$ = current
   (leave $prev\_off_j$ as before)                                   (4)
prev_received = word
```

(1) Active working zone search; in this work, fully associative
(2) Replacement algorithm; in this work, LRU
(3) ident is don't care; its previous value is sent
(4) prev_off is not modified since no previous offset is known

[Mus98] © 1998 IEEE

Security-oriented architectures

- A variety of attacks:
  - Typical desktop/server attacks, such as Trojan horses and viruses.
  - Physical access allows side channel attacks.
- Cryptographic instruction sets have been developed for several architectures.
- Embedded systems architecture must add protection for side effects, consider energy consumption.

Side channel attack

- Side channels
  - Power dissipation
  - EM radiation
  - Operating times
- Side channel attack
  - Requires a statistical analysis of waveforms (e.g. power traces)

Smart cards

- Used to identify holder, carry money, etc.
- Self-programmable one-chip microcomputer (SPOM) architecture:
  - Allows processor to change code or data.
  - Memory is divided into two sections.
  - Registers allow program in one section to modify the other section without interfering with the executing program.

Secure architectures

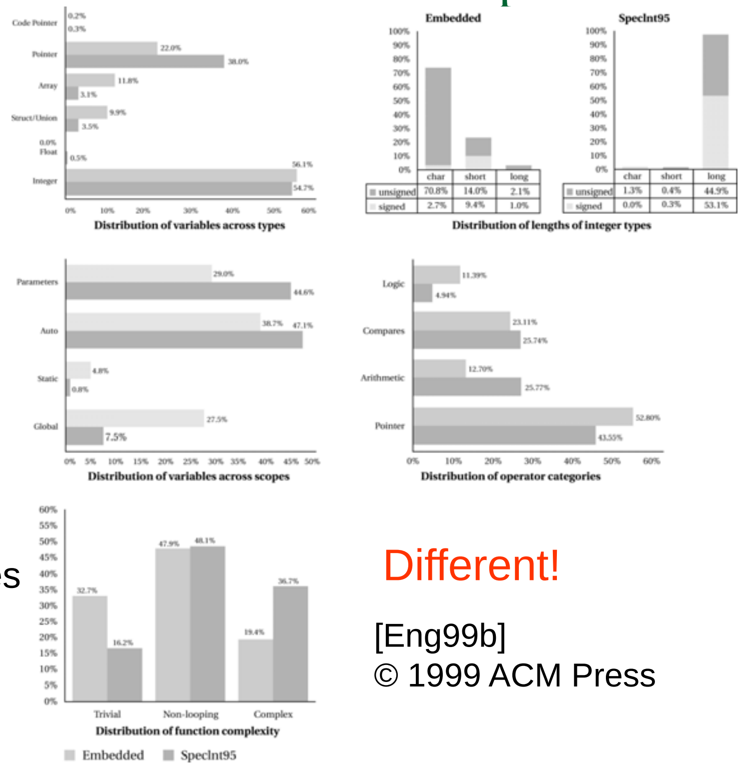
- SmartMIPS (MIPS), SecurCore (ARM) offer security extensions, including encryption instructions, memory management, etc.
- SAFE-OPS embeds a watermark (a verifiable identifier) into code using register assignment.
  - If each register is assigned a symbol, then the sequence of register used in a section of code represents the watermark for that code.
  - FPGA accelerator checks the validity of the watermark during execution.

CPU simulation

- A CPU's simulator means any method of analyzing the behavior of programs on the CPU.
- Performance vs. energy/power simulation.
  - Performance: less detailed but reasonably accurate
  - Power/energy: more accurate
- Temporal accuracy.
  - More detailed -> more accurate timing, slow
- Trace vs. execution.
- Simulation vs. direct execution.

Engblom embedded vs. SPEC comparison

- Embedded software has very different characteristics than the SPECInt benchmark set.
- SpecInt95
  - More dynamic data structure
  - More 32-bit variables
- Embedded programs
  - Mostly smaller data
  - More unsigned variables
  - More static and global variables



Different!

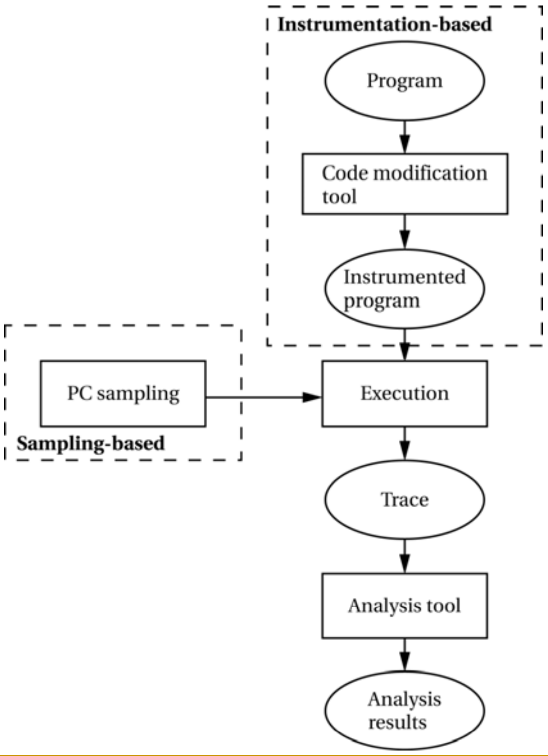
[Eng99b]
© 1999 ACM Press

Benchmarks for embedded applications

- MediaBench
  - JPEG, MPEGS, GSM speech encoding, G.721 voice compression, PGP cryptography package, Ghostscript
- EEMBC DenBench
  - Embedded microprocessor benchmark consortium
  - Denbench includes
    - MPEG EncodeMark
    - MPEG DecodeMark
    - CrytoMark
    - ImageMark
  - The final score is the geometric mean of four minisuites.

Trace-based analysis

- Instrumentation generates side information.
- PC-sampling checks PC value during execution.
- Can measure
  - control flow,
  - memory accesses.
- UNIX prof, GNU gprof
- Logic analyzer



How to generate a trace

- Hardware: logic analyzer
  - Trace buffer is limited and need to store at execution speed
  - Cannot see internal memory references due to on-chip cache
- A processor emulator
  - Able to observe internal behavior
  - Too slow to generate long traces
- Some CPU has hardware facilities for automatically generating trace information
  - A branch trace: source and/or target address of a branch. We can reconstruct the instructions executed within the basic blocks while greatly reducing the trace information.

How to generate a trace by software

- PC sampling
- Instrumentation instructions
  - The program can be instrumented with additional code that writes trace information to memory or a file.
- Direct execution: emulating architecture
  - Emulate the target machine on the host
  - Used primarily for functional and cache simulation, not for detailed timing.
  - Very fast because much of the simulation runs as native code on the host machine.

Microarchitecture-modeling simulators

- Varying levels of detail:
  - Instruction scheduler is not cycle-accurate.
  - Cycle timers are cycle-accurate.
- Can simulate for performance or energy/power.
- Typically written in general-purpose programming language (C), not hardware description language (VHDL, Verilog).

PC sampling

- Example: Unix prof.
- Interrupts are used to sample PC periodically.
  - Must run on the platform.
  - Doesn’t provide complete trace.
  - Subject to sampling problems: undersampling, periodicity problems.

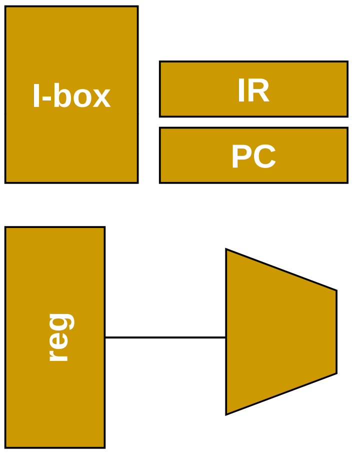
Call graph report

```
Main     100%
    f1
    f2
---
f1       37%
    g1
    g2
---
f2       23%
    g3
    g4
```

Cumulative execution time

Cycle-accurate simulator

- Models the microarchitecture.
  - Simulating one instruction requires executing routines for each pipeline stage.
- Models pipeline state.
  - Microarchitectural registers are exposed to the simulator.
- Somewhat slow



Trace-based vs. execution-based

- Trace-based:
  - Gather trace first, then generate timing information.
  - Basic timing information is simpler to generate.
  - Full timing information may require regenerating information from the original execution.
- Execution-based:
  - Simulator fully executes the instruction.
  - Requires a more complex simulator.
  - Requires explicit knowledge of the microarchitecture, not just instruction execution times.

Sources of timing information

- Data book tables:
  - Time of individual instructions.
  - Penalties for various hazards.
- Microarchitecture:
  - Depends from the structure of machine.
  - Derived from execution of the instruction in the microarchitecture.

Levels of detail in simulation

- Instruction schedulers:
  - Models availability of microarchitectural resources.
  - May not capture all interactions.
- Cycle timers:
  - Models full microarchitecture.
  - Most accurate, requires exact model of the microarchitecture.

SimpleScalar

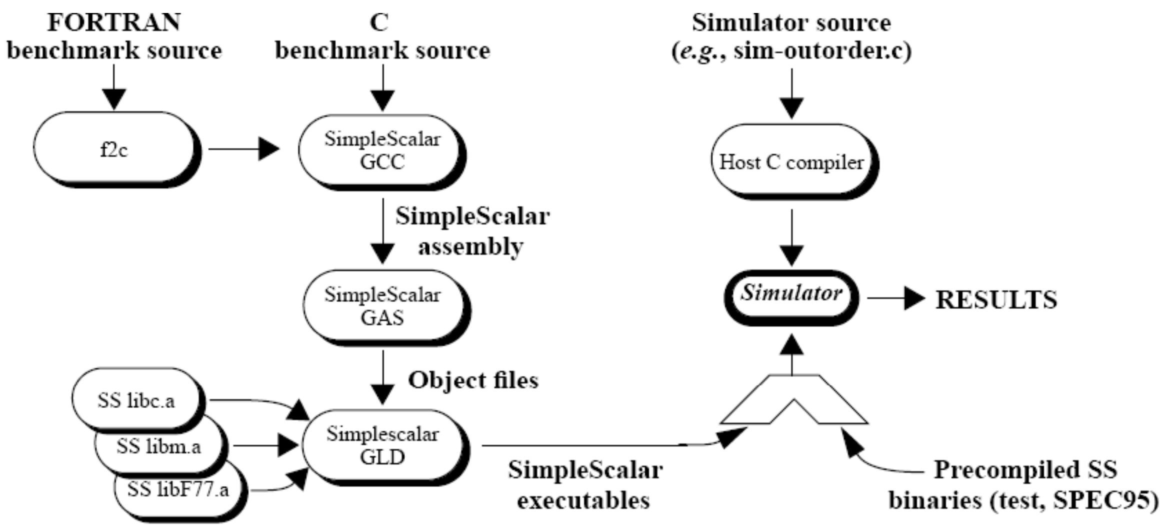


Figure 1. SimpleScalar tool set overview

Early approaches to power modeling

- Instruction macromodels:
  - ADD = 1 μw, JMP = 2 μw, etc.
- Data-dependent models:
  - Based on data value statistics.
- Transition-based models.

Power simulation

- Model capacitance in the processor.
- Keep track of activity in the processor.
  - Requires full simulation.
- Activity determines capacitive charge/discharge, which determines power consumption.

SimplePower simulator

- Cycle-accurate simulator.
  - SimpleScalar-style cycle-accurate simulator.
- Transition-based power analysis.
  - Estimates energy of data path, memory, and busses on every clock cycle.

RTL power estimation interface

- A power estimator is required for each functional unit modeled in the simulator.
  - Functional interface makes the simulator more modular.
- Power estimator takes same arguments as the performance simulation module.

Switch capacitance tables

- Model functional units such as ALU, register files, multiplexers, etc.
- Capture technology-dependent capacitance of the unit.
- Two types of model:
  - Bit-independent: each bit is independent, model is one bit wide.
  - Bit-dependent: bits interact (as in adder), model must be multiple bits.
- Analytical models used for memories.
- Adder model is built from sub-model for adder slice.

Wattch power simulator

- Built on top of SimpleScalar.
- Adds parameterized power models for the functional units.

Array model

- Analytical model:
  - Decoder.
  - Wordline drive.
  - Bitline discharge.
  - Sense amp output.
- Register file word line capacitance:
  - C_{diff} (word line driver) + C_{gate}(cell access)*n_{bit_lines} + C_{metal} * Word_line_length

Bus, function unit models

- Bus model based upon length of bus, capacitance of bus lines.
- Models for ALUs, etc. based upon transition models.

Clock network power model

- Clock is a major power sink in modern designs.
- Major elements of the clock power model:
  - Global clock lines.
  - Global drivers.
  - Loads on the clock network.
- Must handle gated clocks.

Instruction Set Simulator (ISS)

- Native code execution ISS
  - Target application code is compiled for the host and executed on the host.
  - Fastest
  - Inaccurate
- Interpretive ISS
  - Slow
  - Flexible and accurate

original assembly code

```
…
add r1, r2, r3
sub r3, r4, r1
…
```

$\xrightarrow{\text{Interpretive}}$

ISS code

```
int Reg[32];
…
while(1) {
        Fetch();
        Decode();
        Execute();
        Interrupt handler();
}
```

Instruction Set Simulator (ISS)

- Compiled ISS
  - (Binary translation) translated the target binary to the host binary
  - (C intermediate code) generate the C code from the target binary and compile it for the host
  - Fast
  - Accurate

original assembly code

```
…
add r1, r2, r3
sub r3, r4, r1
…
```

$\xrightarrow{\text{Compiled}}$

```
…
Add(r1, r2, r3);
Sub(r3, r4, r1);
…
```

Automated CPU design

- Customize aspects of CPU for application:
  - Instruction set.
  - Memory system.
  - Busses and I/O.
- Tools help design and implement custom CPUs.
- FPGAs make it easier to implement custom CPUs.
- Application-specific instruction processor (ASIP) has custom instruction set.
- Configurable processor is generated by a tool set.

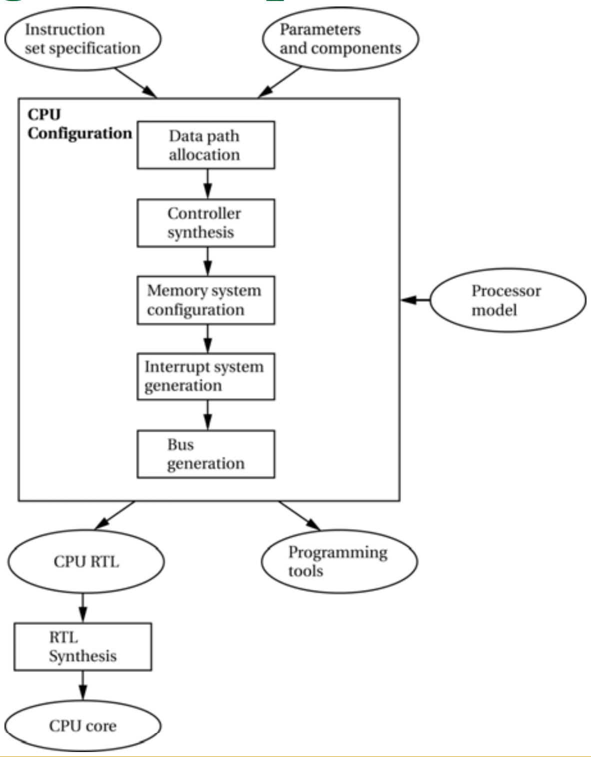
Types of customization

- New instructions: operations, operands, remove unused instructions.
- Specialized pipelines.
- Specialized memory hierarchy.
- Busses and peripherals.

Techniques

- Architecture optimization tools help choose the instruction set and microarchitecture.
- Configuration tools implement the microarchitecture (and perhaps compiler).
- Early example: MIMOLA analyzed programs, created microarchitecture and instructions, synthesized logic.

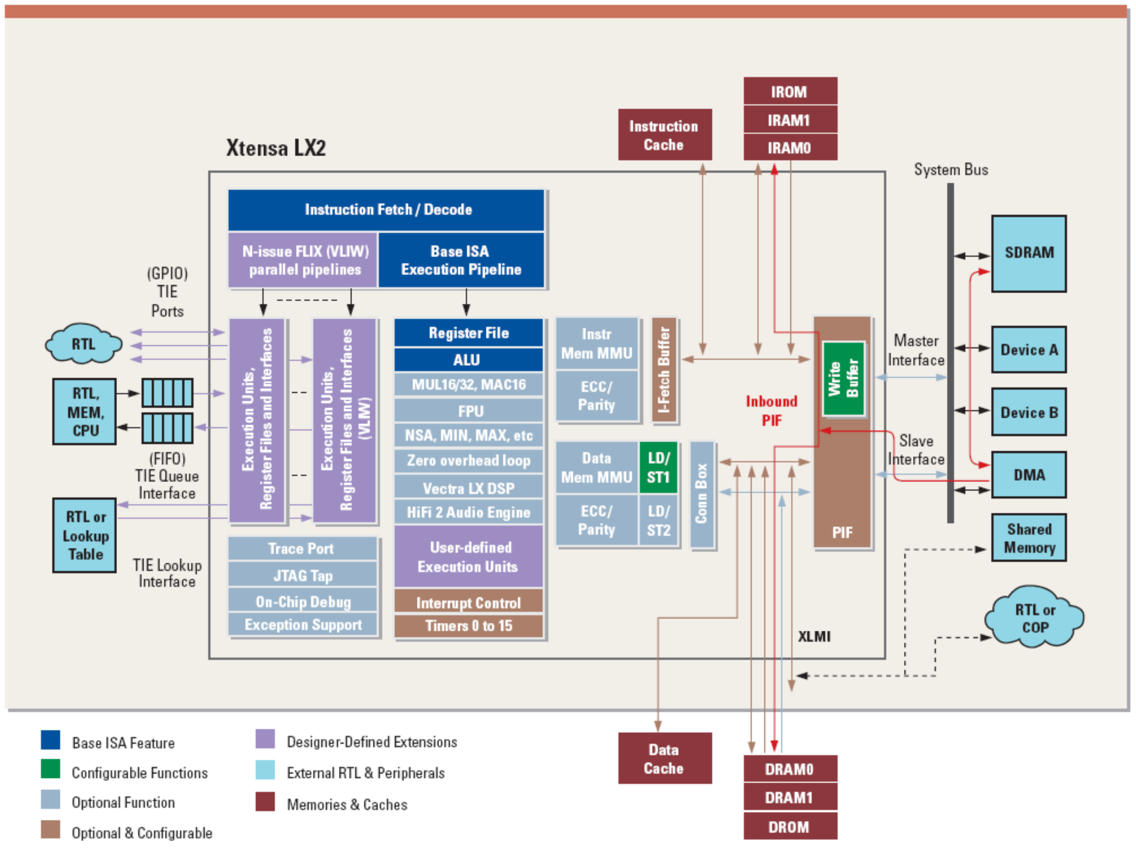
CPU configuration process

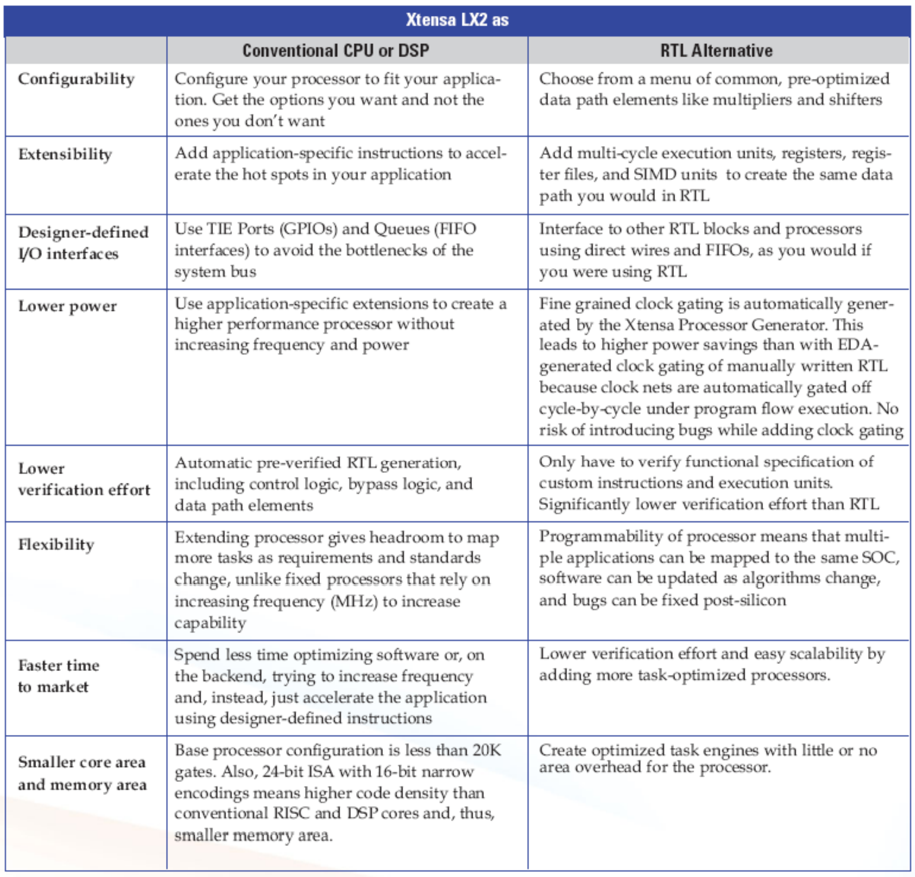


Configurable Processors

- ARC
- Tenslica Xtensa
- ASIP Meister
- Toshiba MeP core

Tensilica LX2 configuration options

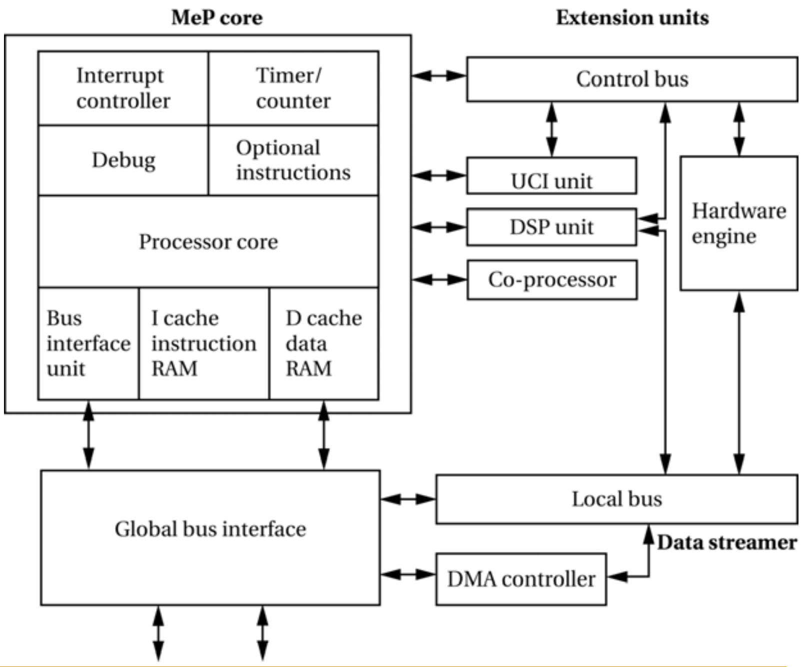




Xtensa LX2 as	Conventional CPU or DSP	RTL Alternative
Configurability	Configure your processor to fit your application. Get the options you want and not the ones you don't want	Choose from a menu of common, pre-optimized data path elements like multipliers and shifters
Extensibility	Add application-specific instructions to accelerate the hot spots in your application	Add multi-cycle execution units, registers, register files, and SIMD units to create the same data path you would in RTL
Designer-defined I/O interfaces	Use TIE Ports (GPIOs) and Queues (FIFO interfaces) to avoid the bottlenecks of the system bus	Interface to other RTL blocks and processors using direct wires and FIFOs, as you would if you were using RTL
Lower power	Use application-specific extensions to create a higher performance processor without increasing frequency and power	Fine grained clock gating is automatically generated by the Xtensa Processor Generator. This leads to higher power savings than with EDA-generated clock gating of manually written RTL because clock nets are automatically gated off cycle-by-cycle under program flow execution. No risk of introducing bugs while adding clock gating
Lower verification effort	Automatic pre-verified RTL generation, including control logic, bypass logic, and data path elements	Only have to verify functional specification of custom instructions and execution units. Significantly lower verification effort than RTL
Flexibility	Extending processor gives headroom to map more tasks as requirements and standards change, unlike fixed processors that rely on increasing frequency (MHz) to increase capability	Programmability of processor means that multiple applications can be mapped to the same SOC, software can be updated as algorithms change, and bugs can be fixed post-silicon
Faster time to market	Spend less time optimizing software or, on the backend, trying to increase frequency and, instead, just accelerate the application using designer-defined instructions	Lower verification effort and easy scalability by adding more task-optimized processors.
Smaller core area and memory area	Base processor configuration is less than 20K gates. Also, 24-bit ISA with 16-bit narrow encodings means higher code density than conventional RISC and DSP cores and, thus, smaller memory area.	Create optimized task engines with little or no area overhead for the processor.

Toshiba MePcore

- Optimized for media processing and streaming application
- MeP core + Extension units
- UCI unit (1 cycle)
- DSP unit (multi-cycle)
- Co-processor (VLIW)
- DMA controller for streaming



LISA EDGE

- A tool platform for embedded processor design from Coware
- LISA 2.0 architecture description language (ADL)
- Employs the CoSy system from ACE on compiler side.
- CGD (code generator description) in CoSy
  - A specification of target processor resources like registers and functional units
  - A description of mapping rules, specifying how C/C++ language constructs map to a block of assembly instructions
  - A scheduler table that captures instruction latencies as well as instruction resource occupation on a cycle-by-cycle basis.

LISATek EDGE

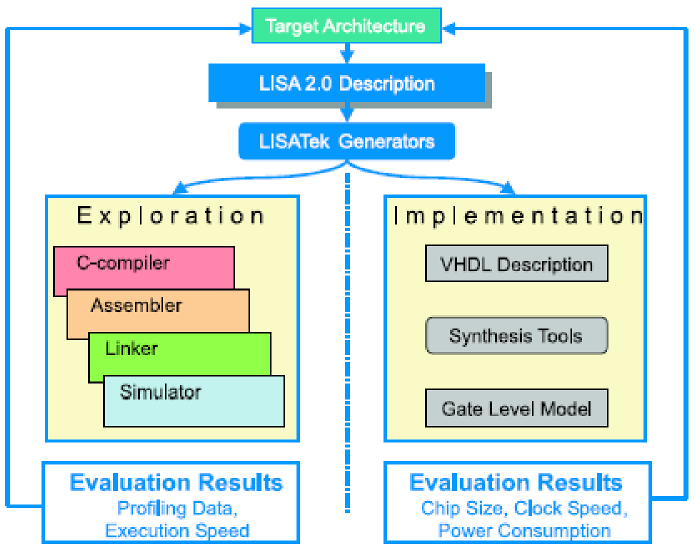


Figure 2. LISATek EDGE based ASIP design flow

CoSy

- Requires a CGD as well as some further information like function calling conventions, or the C data type sizes and memory alignment.
- LISA based C compiler generator can be coarsely viewed as a LISA-to-CGD translator.
- This translation is difficult because of the semantic gap between the compiler's high-level model of the target machine and the detailed ADL model that captures cycle and bit-true behavior of the machine operation.

LISATek EDGE

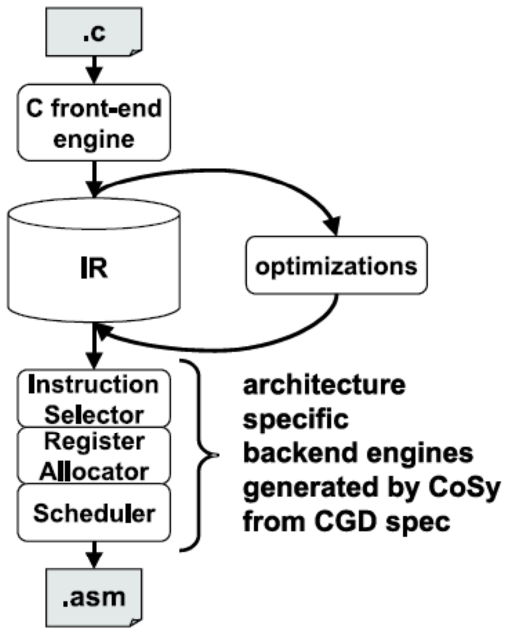
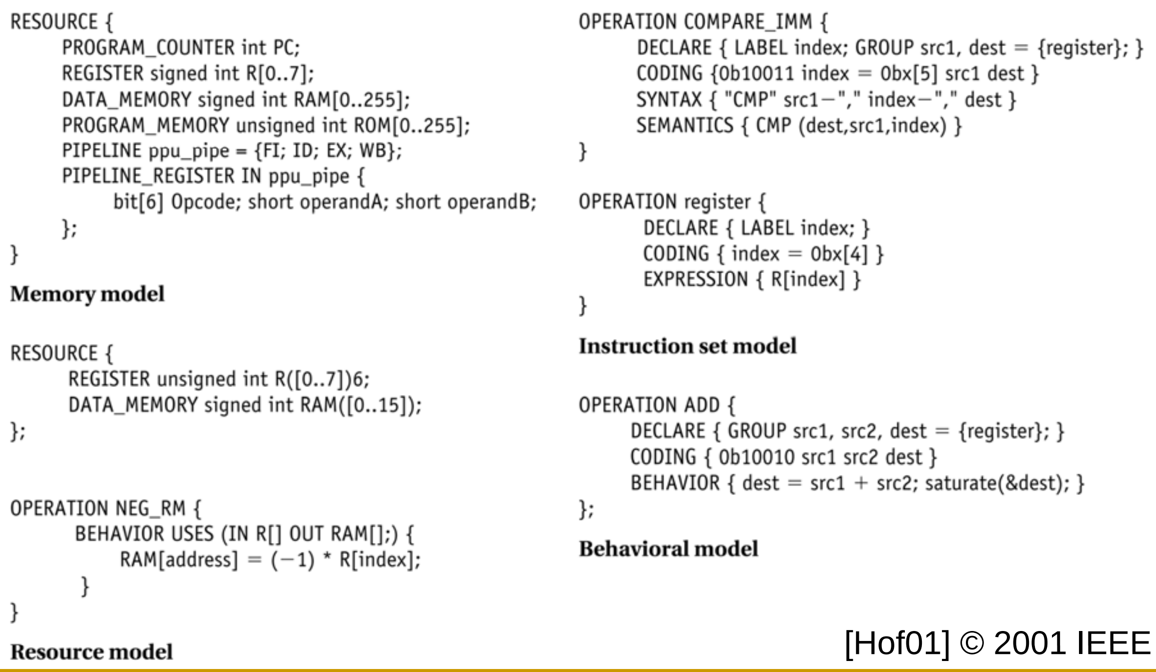


Figure 3. Generated CoSy compiler structure

LISA language



```
RESOURCE {
        PROGRAM_COUNTER int PC;
        REGISTER signed int R[0..7];
        DATA_MEMORY signed int RAM[0..255];
        PROGRAM_MEMORY unsigned int ROM[0..255];
        PIPELINE ppu_pipe = {FI; ID; EX; WB};
        PIPELINE_REGISTER IN ppu_pipe {
                bit[6] Opcode; short operandA; short operandB;
        };
}
```

Memory model

```
RESOURCE {
        REGISTER unsigned int R([0..7])6;
        DATA_MEMORY signed int RAM([0..15]);
};
```

```
OPERATION NEG_RM {
        BEHAVIOR USES (IN R[] OUT RAM[];) {
                RAM[address] = (−1) * R[index];
        }
}
```

Resource model

```
OPERATION COMPARE_IMM {
        DECLARE { LABEL index; GROUP src1, dest = {register}; }
        CODING {0b10011 index = 0bx[5] src1 dest }
        SYNTAX { "CMP" src1−"," index−"," dest }
        SEMANTICS { CMP (dest,src1,index) }
}

OPERATION register {
        DECLARE { LABEL index; }
        CODING { index = 0bx[4] }
        EXPRESSION { R[index] }
}
```

Instruction set model

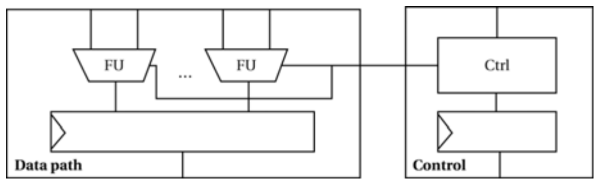
```
OPERATION ADD {
        DECLARE { GROUP src1, src2, dest = {register}; }
        CODING { 0b10010 src1 src2 dest }
        BEHAVIOR { dest = src1 + src2; saturate(&dest); }
};
```

Behavioral model

[Hof01] © 2001 IEEE

PEAS-III

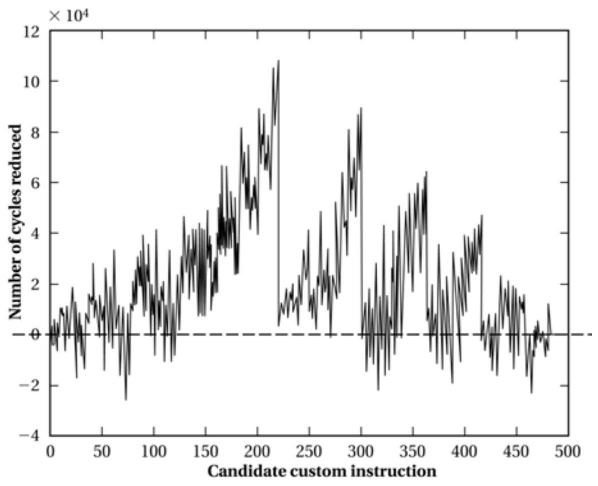
- Synthesis driven by:
  1. Architectural parameters such as number of pipeline stages.
  2. Declaration of function units.
  3. Instruction format definitions.
  4. Interrupt conditions and timing.
  5. Micro-operations for instructions and interrupts.
- Generates both simulation and synthesis models in VHDL.



A single pipeline stage

Instruction set synthesis

- Generate a set of candidate instructions from application program, other requirements.
- Sun et al. analyzed design space for simple BYTESWAP() program.



[Sun04] © 2004 IEEE

Holmer and Despain

- Viewing instruction set design as an optimization problem
- 1% rule---don't add instruction unless it improves performance by 1%.
- Objective function (C = # cycles, I = # instruction types, S = # instructions in program):
  - 100 ln C + I : optimizing execution time
  - 100 ln C + 20 ln S + I : optimizing execution time and code size
- They used microcode compaction algorithms to find instructions.