



The Configurable Processor Company

Automatic Generation of Processor Extensions

Embedded Processor Forum 2003
June 17, 2003

Dror Maydan
Director of Software Development
Tensilica, Inc.

■ Tensilica

- **Market leader for configurable, extensible processors**
- **Introduced processor generation to the market in 1999**
- **Delivered five generations of the Xtensa processor and associated tool chain**

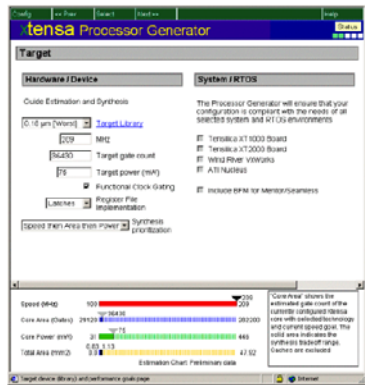
■ @ Microprocessor Forum, October 2002

- **Introduced FLIX architecture**
 - Flexible Length Instruction Xtensions
 - Next generation of Xtensa processor ISA enhancements
- **FLIX freely intermixes 16-, 24-, and 32 or 64-bit instructions**
 - Benefits of VLIW-style instructions *without code bloat*
 - Full backwards code compatibility with current Xtensa ISA

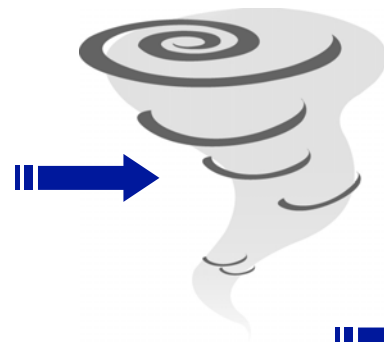
■ Disclosing Today @ Embedded Processor Forum 2003

- **New tool to automate the specification of processor extensions**

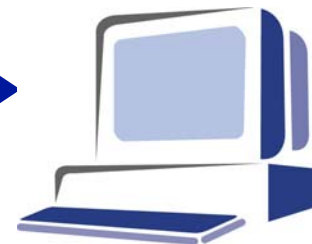
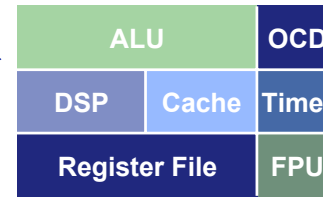
Complete Hardware Design Verilog/VHDL RTL, EDA scripts, test suite



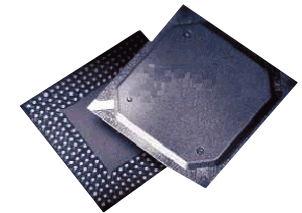
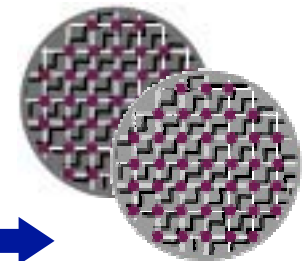
Electronic Specification
Configuration selection and custom-instruction description



Xtensa Processor Generator*



Customized Software Tools
C/C++ compiler
Debuggers
Simulators
RTOSes

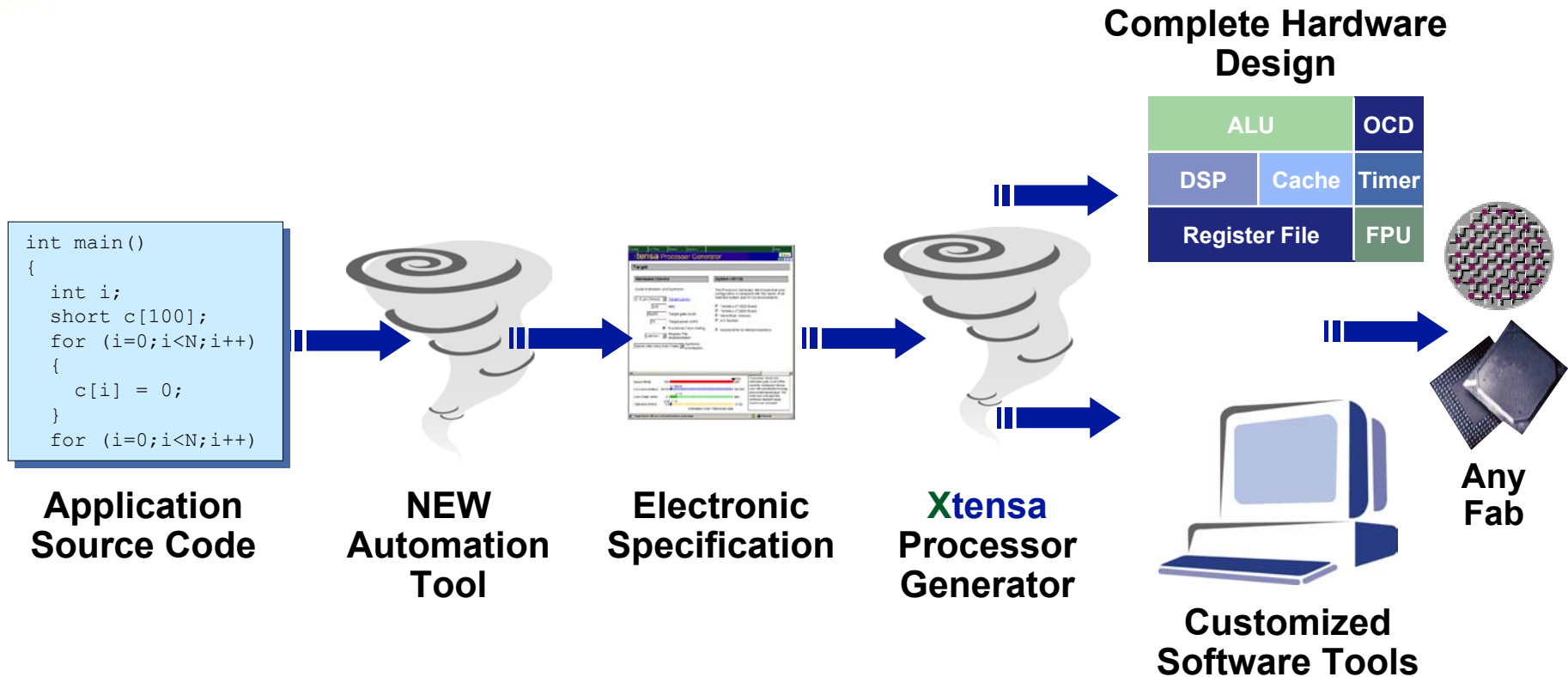


Use standard techniques and libraries for any IC fabrication process

Iterate in hours!

* US Patent: 6,477,697

The Next Generation of Automation



Techniques Available to Improve ISA For a Target Application

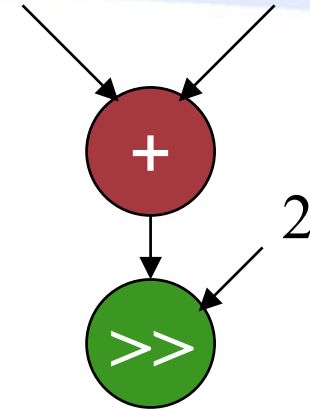
- **Three common and complementary techniques used to boost application-specific performance**
 - **Fusion**
 - Compound, dependent operations
 - E.g. a multiply-add or a multiply feeding a shift
 - **FLIX** - **F**lexible **L**ength **I**nstruction **X**tensions
 - Grouping together independent operations
 - Similar to VLIW without code size increase
 - **SIMD / Vector**

- **All three techniques are easily specified using TIE (Tensilica Instruction Extension) language**
 - **New instructions, new register files & state**

Techniques to Improve Performance: Fusion Example

Original C Code

```
int *a, *b, *c;
for (int i=0; i<n; i++)
    c[i] = (a[i] + b[i]) >> 2
```



Complete TIE Code

```
operation add_shift(out AR c, in AR a, in AR b) {
    wire t[31:0] = a+b;
    assign c = {2{t[29]},t[29:0]};
}
```

C Code Using TIE

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

- Hardware plus simulator, compiler, debugger, RTOS automatically generated from TIE description

Techniques to Improve Performance: FLIX example

Original C Code

```
for (int i=0; i<n; i++)
    c[i] = (a[i] + b[i]) >> 2
```

64 Bit Instruction with 3 Slots



Complete TIE Code

```
length 1 64 { InstBuf[3:0] == 14 }
format f 1
slot slot0 f[*]  ADDI, NOP
slot slot1 f[*]  ADD, SRAI, NOP
slot slot2 f[*]  L32I, S32I, NOP
```

Generated Assembly

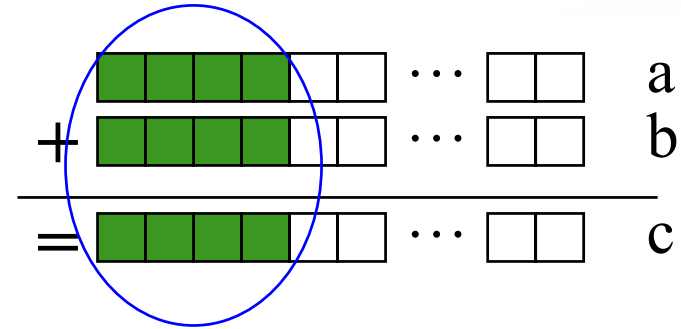
```
loop:
{ addi  a9,a9,4;    add  a12,a10,a8 ; l32i  a8,a9,0 }
{ addi  a11,a11,4 ; srai a12,a12,2 ; l32i  a10,a11,0 }
{ addi  a13,a13,4;   nop;           s32i  a12,a13,0 }
```

- Original C compiled to 3 cycles/iteration.
- Hardware plus simulator, compiler, debugger, RTOS are automatically generated from TIE description

Techniques to Improve Performance: SIMD/Vector

Original C Code

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



Complete TIE Code

```
regfile vec 64 16 v;
operation add16x4(out vec c, in vec a, in vec b) {
    assign c = {a[63:48]+b[63:48], a[47:32]+b[47:32],
                a[31:16]+b[31:16], a[15:0]+b[15:0]};
}
```

C Code Using TIE

```
vec *a, *b, *c;
for (int i=0; i<n; i+=4)
    c[i] = add16x4(b[i], a[i])
```

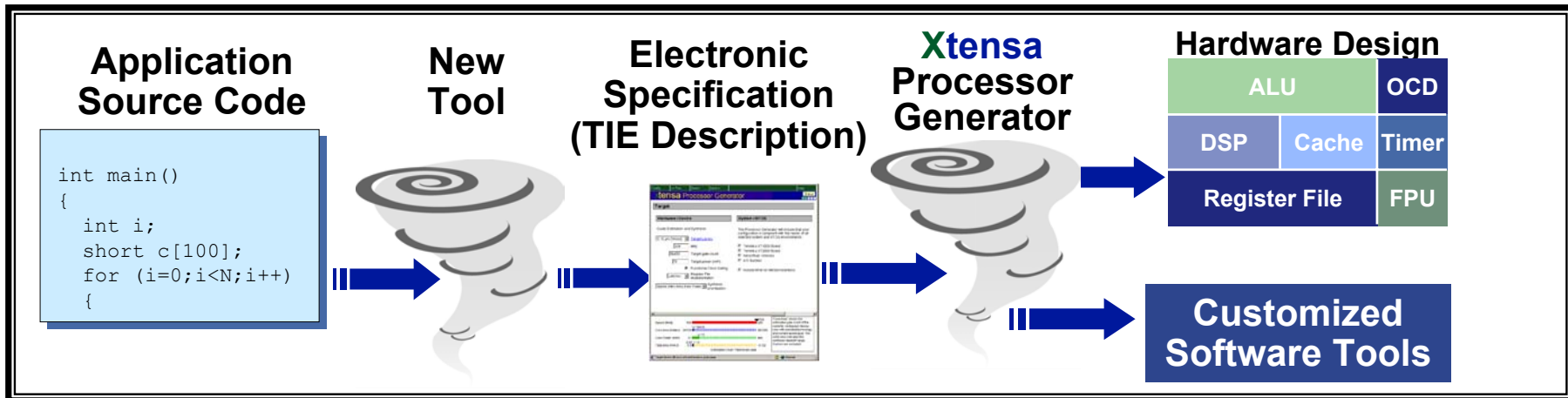
- Hardware plus simulator, compiler, debugger, RTOS automatically generated from TIE description

Deciding what instructions you need

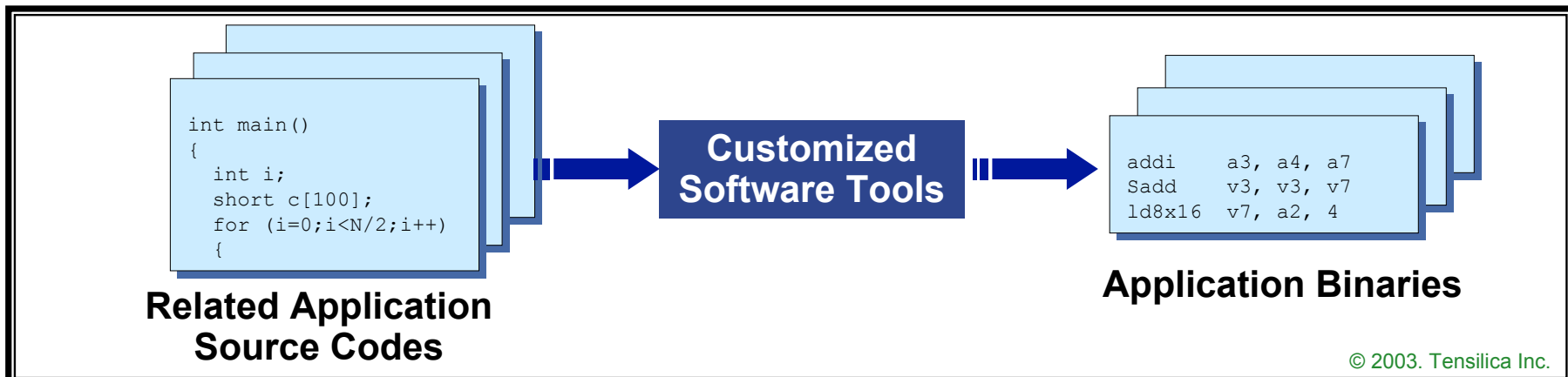
- **Which techniques apply to your code?**
- **How do you trade off techniques?
(SIMD vs FLIX vs Fusion)**
 - **SIMD gives larger performance gains but does not apply as often as FLIX**
 - **SIMD replicates functional units and widens memory interfaces but minimizes register usage**
 - **FLIX might not replicate functional units but might add ports to register file**
 - **32-bit FLIX is cheaper (less hardware, smaller code) than 64-bit FLIX but might not encode**
- **How much hardware should I add given my performance and hardware cost needs?**

Automating Processor Extension

Given an ANSI C/C++ application, performance goals, and a budget, generate a good set of processor extensions (TIE description) ...



... that is efficient over a range of similar applications



Goals for Automating Processor Extension

Flexibility

- Application code might be written / modified after SOC tape-out
- Generated TIE must be sufficiently general purpose so that small changes to application code do not degrade performance

Control

- **Full automation**
 - C/C++ in → TIE out
 - C/C++ and generated TIE in → binary code out
- **Full control**
 - Allow user to guide tool and/or to select instructions
 - Allow user to add to or change generated TIE
 - Allow user to tune application to better take advantage of TIE

Speed: minutes, not days

Automated Processor Extension: Step 1

- **Compile the C/C++ application code**
 - **Designer uses a new compiler optimization flag**
 - Standard makefiles and IDE supported
 - **Compiler generates comments to help user tune code**
 - Optimized code yields better results
 - **Compiler generates information from application**
 - Feedback optimization ranks code regions by frequency
 - Vectorizer determines which loops can be vectorized
 - Generates lists and counts of vectorized operations
 - Fuser generates dataflow graphs for important regions
 - One graph assumes a region is not vectorized
 - One graph assumes a region is vectorized
 - Operation counts for each type of opcode for every region

Automated Processor Extension: Step 2

- **Generated information used to select and generate TIE**
 - **For each code region, generate many potential sets of TIE extensions (configurations)**
 - Vectorize by 1, 2, 4, 8
 - Add **FLIX** functional units
 - Add fusions
 - Exponentially huge number of possible configurations
 - Generation guided by estimated performance
 - **Evaluate all generated configurations across all regions**
 - **Find best set of merged configurations given budget**
 - **Evaluation and searching done abstractly**
 - Allows the tool to consider millions of configurations

Automated Processor Extension: Step 3

- **Manually tune the TIE (optional)**
 - **Hardware designer can refine or optimize the implementations**
 - **Architect can add additional instructions that were perhaps not well represented in input application**

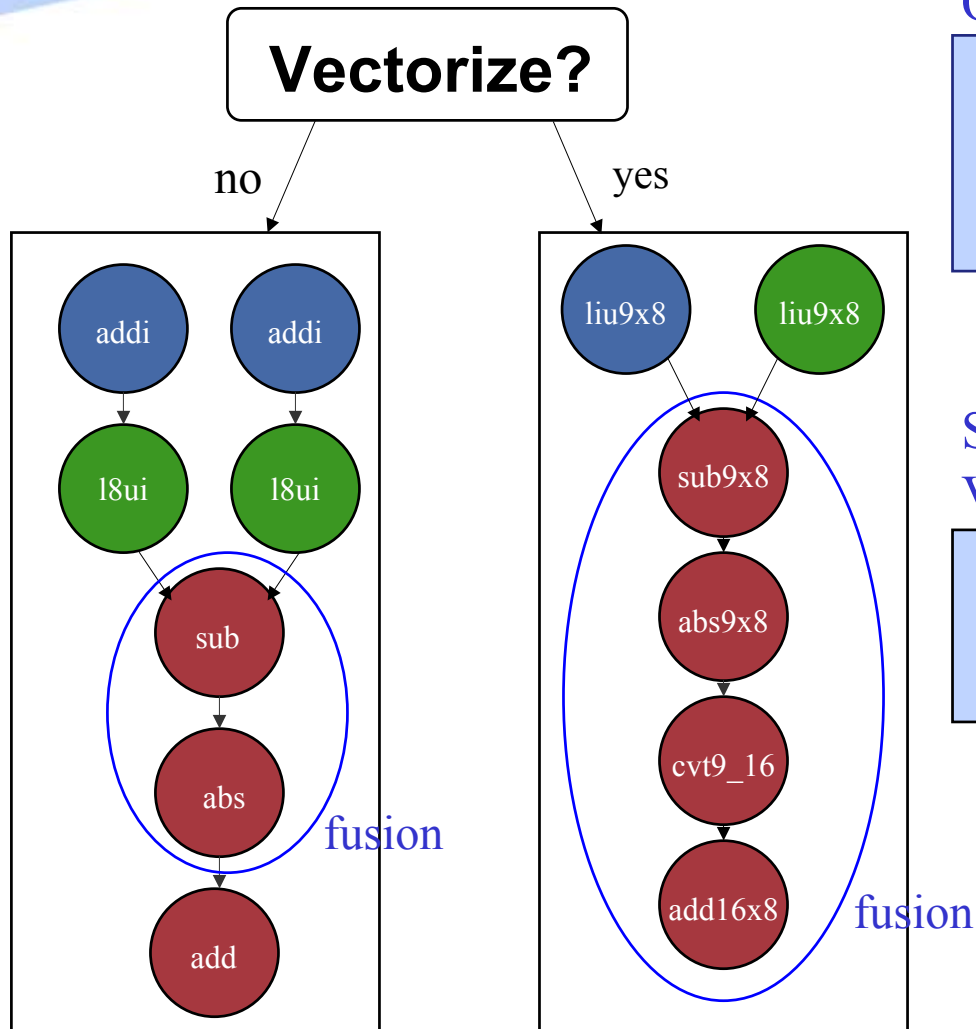


Automated Processor Extension: Step 4

Use the TIE with a C/C++ or assembly application

- **Compiler reads TIE (automatically or manually generated) and generates code**
 - FLIX slot/format TIE specification mapped to resource tables
 - used by compiler to schedule and bundle codes
 - formats less rigid
 - formats may used specialized ops
 - Generalized graph matcher generates dataflow graphs from TIE semantics and compiler IR, which are matched to find fusions
 - Vectorizer vectorizes a loop and checks if all required operations available in TIE description
- **User free to tune the code in ANSI C/C++ or assembly**
- **Simulator, assembler, debugger, RTOS support generated directly from TIE**

Example: SAD (sum of absolute differences)



Original C Code

```
short total=0;
char *p1, *p2;
for i=1,m
  for j=1,n
    total += abs(*p1++ - *p2++)
```

Sample Software Pipelined Schedule Vector+Fusion+FLIX Configuration

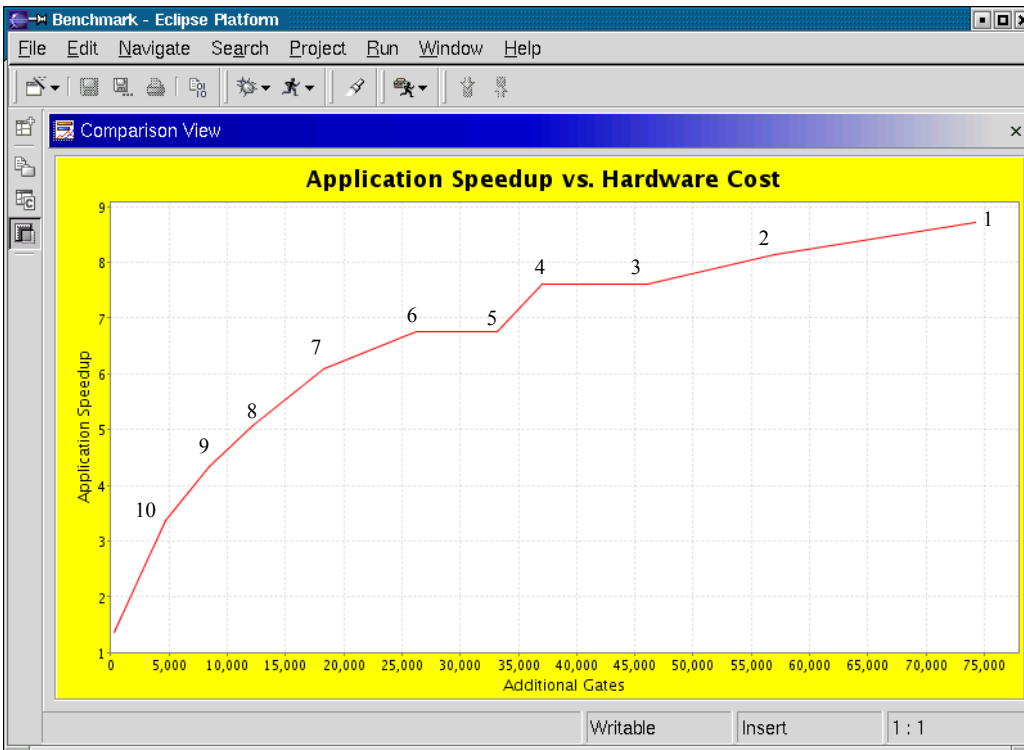
```
loop j=1,n/8 by 2:
  liu9x8[j]; liu9x8[j]; fusion[j-2]
  liu9x8[j+1]; liu9x8[j+1]; fusion[j-1]
```



Example: SAD Automatic Solution Search

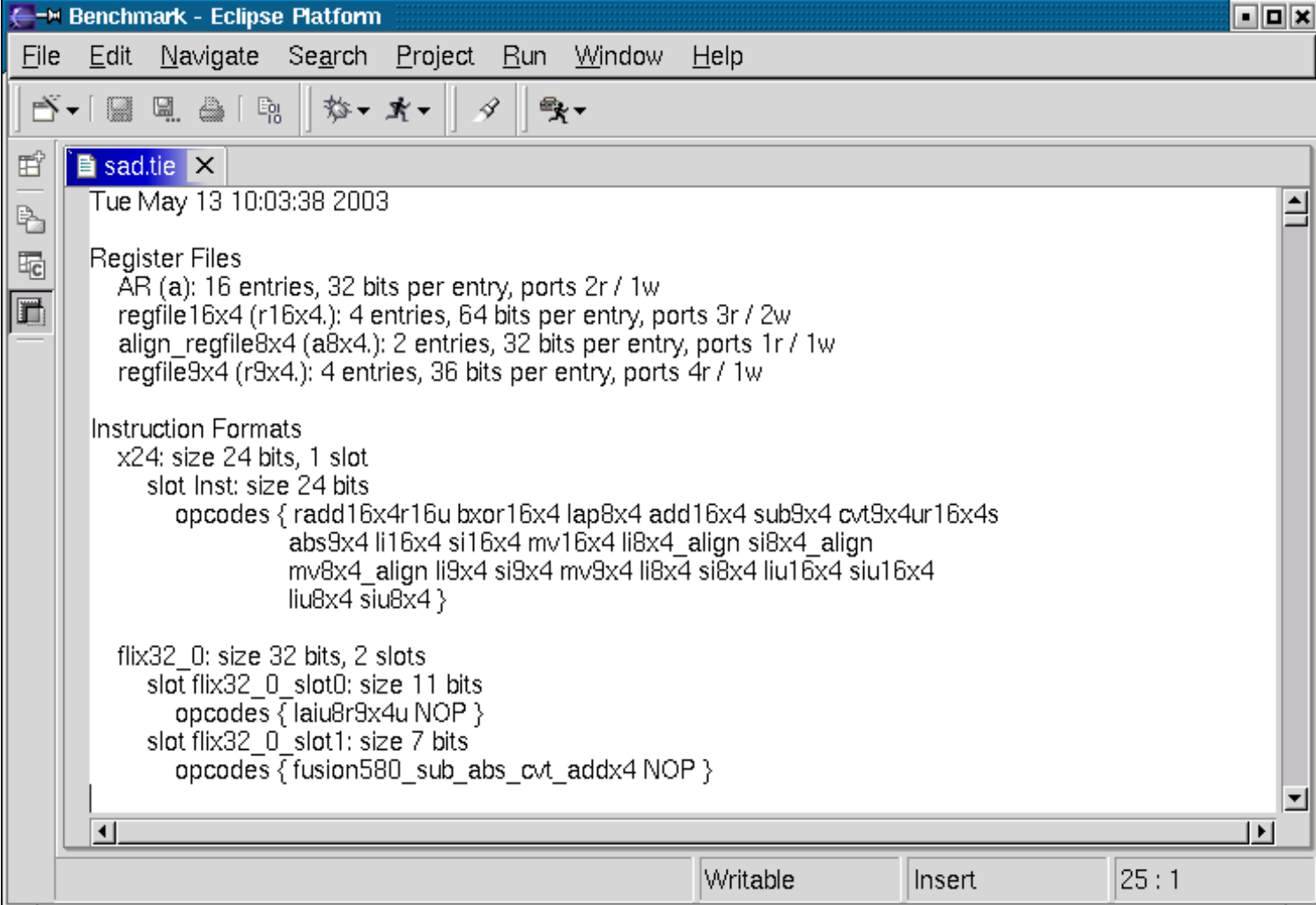
Generated Configuration Parameters

<i>i</i>	Speedup	Gates Added (K)	SIMD Factor	FLIX Width (Slots)	Load / Store Units	Fusion
1	8.7x	74	8	3	2	Yes
2	8.1x	57	8	2	2	Yes
3	7.6x	46	4	3	2	Yes
4	7.6x	37	8	2	1	Yes
5	6.8x	33	4	2	2	Yes
6	6.8x	26	8	1	1	Yes
7	6.1x	18	4	2	1	Yes
8	5.1x	12	4	1	1	Yes
9	4.3x	8	2	2	1	Yes
10	3.4x	5	2	1	1	Yes
11	1.4x	0.3	1	1	1	Yes



Wide Range of Choices of
Performance Increase versus Hardware Cost

Example: SAD Configuration #7 - Generated TIE



```

Benchmark - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Tue May 13 10:03:38 2003
Register Files
  AR (a): 16 entries, 32 bits per entry, ports 2r / 1w
  regfile16x4 (r16x4.): 4 entries, 64 bits per entry, ports 3r / 2w
  align_regfile8x4 (a8x4.): 2 entries, 32 bits per entry, ports 1r / 1w
  regfile9x4 (r9x4.): 4 entries, 36 bits per entry, ports 4r / 1w

Instruction Formats
  x24: size 24 bits, 1 slot
    slot Inst: size 24 bits
      opcodes { radd16x4r16u bxor16x4 lap8x4 add16x4 sub9x4 cvt9x4ur16x4s
                abs9x4 li16x4 si16x4 mv16x4 li8x4_align si8x4_align
                mv8x4_align li9x4 si9x4 mv9x4 li8x4 si8x4 liu16x4 siu16x4
                liu8x4 siu8x4 }

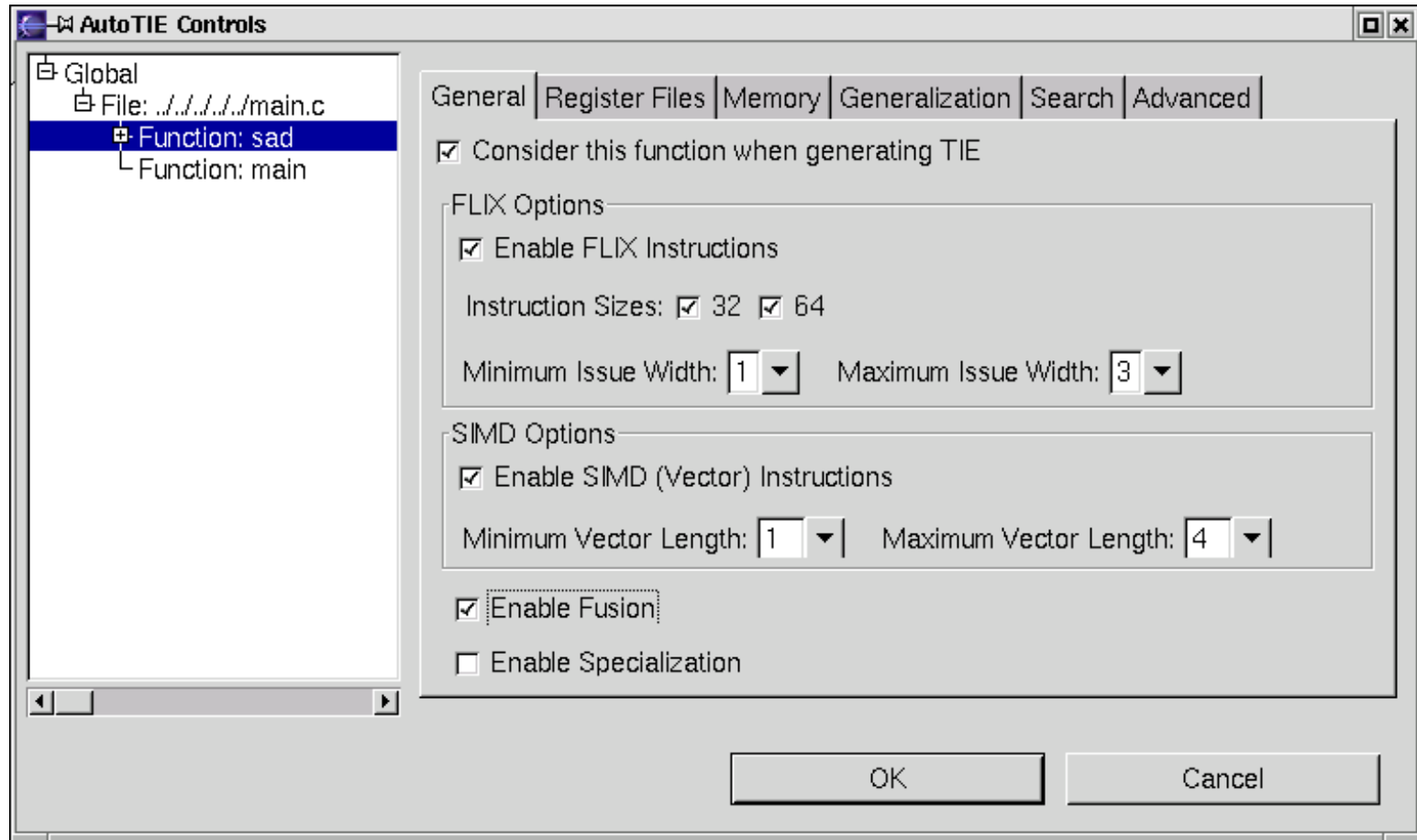
  flix32_0: size 32 bits, 2 slots
    slot flix32_0_slot0: size 11 bits
      opcodes { laiu8r9x4u NOP }
    slot flix32_0_slot1: size 7 bits
      opcodes { fusion580_sub_abs_cvt_addx4 NOP }
  
```

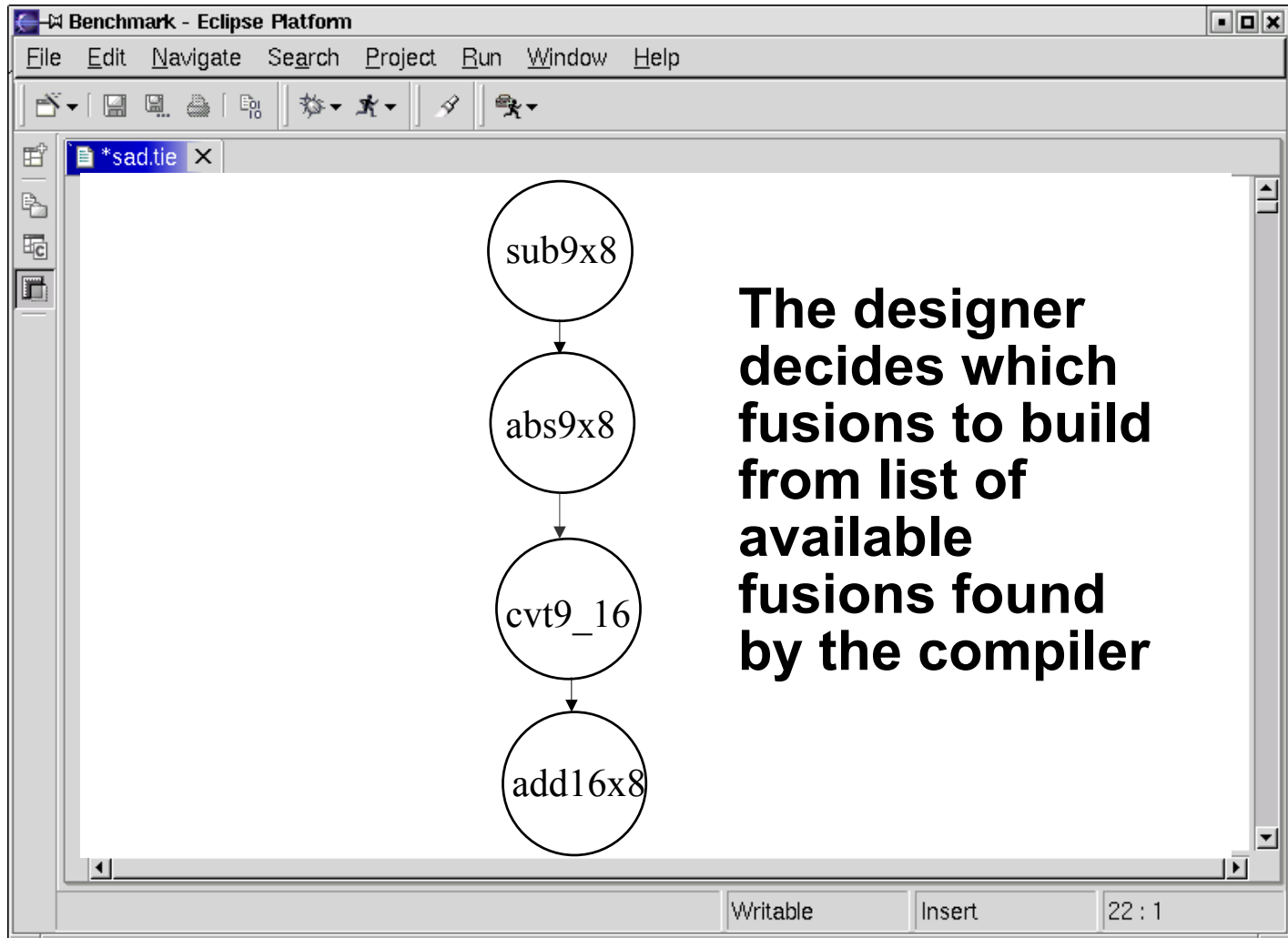
Writable Insert 25 : 1

Application Examples

Application	Speedup	Original Code Size (Before Acceleration)	Code Size After Acceleration	Code Size on MIPS32 (using gcc -O2)	Configurations Visited	Run Time to Generate Configurations
Radix-4 FFT	10.6x	1.5 KB	3.6 KB	4.4KB	175,796	3 minutes
GSM Encoder	3.9x	17 KB	20 KB	38 KB	576,722	15 minutes
GSM Encoder (using FFT TIE)	1.8x	17 KB	19 KB	38 KB	N/A	N/A
MPEG4 Encoder	3.0x	111 KB	136 KB	356 KB	1,830,796	30 minutes

The designer can control the type of TIE the tool will try to generate



A screenshot of the Eclipse Platform IDE window titled "Benchmark - Eclipse Platform". The window shows a menu bar (File, Edit, Navigate, Search, Project, Run, Window, Help) and a toolbar. A single editor tab labeled "*sad.tie" is open, displaying a vertical flowchart with four circular nodes connected by downward arrows: "sub9x8", "abs9x8", "cvt9_16", and "add16x8". To the right of the flowchart, the text "The designer decides which fusions to build from list of available fusions found by the compiler" is displayed in a large, bold, black font. The status bar at the bottom of the window shows "Writable", "Insert", and "22 : 1".

**The designer
decides which
fusions to build
from list of
available
fusions found
by the compiler**



Summary: Automated Processor Extension Generation

- **The next step in automating processor design**
 - Builds upon proven, performance leading Xtensa processor generation technology
- **High productivity**
 - Results in minutes; fully automatic or fine-grained control
- **Flexible, high-performance results**
 - Dramatic application acceleration
 - Retains software programmability in silicon
 - Acceleration applies even if software source code changes
- **Any engineer can optimize a processor**
 - System architects, software engineers, hardware engineers
- **Next significant step towards the Sea-of-Processors™ style of SOC design**