Design Automation Methodologies for Extensible Processors Platform

Newton Cheung

School of Computer Science & Engineering
The University of New South Wales
Sydney, Australia

ncheung@cse.unsw.edu.au

Outline

- System-On-Chips Design Challenges
- Extensible Processors Platform
- Background ñ Xtensa
- Problems in Extensible Processors Platform
- The Goal of the research
- Proposed Solution
- INSIDE
- MINCE
- Conclusions

SoC Design Challenges

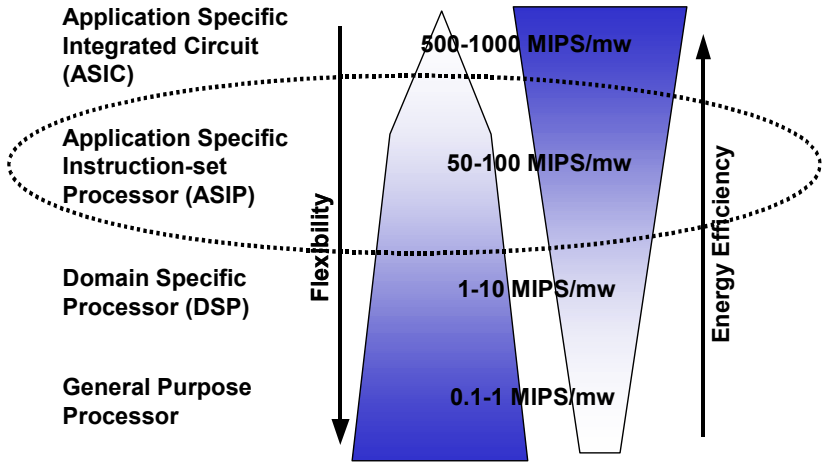
- Application functionality
- Ever changing nature of embedded product

Increasing software content → Flexibility

- Power consumption
- Chip area
- Cost

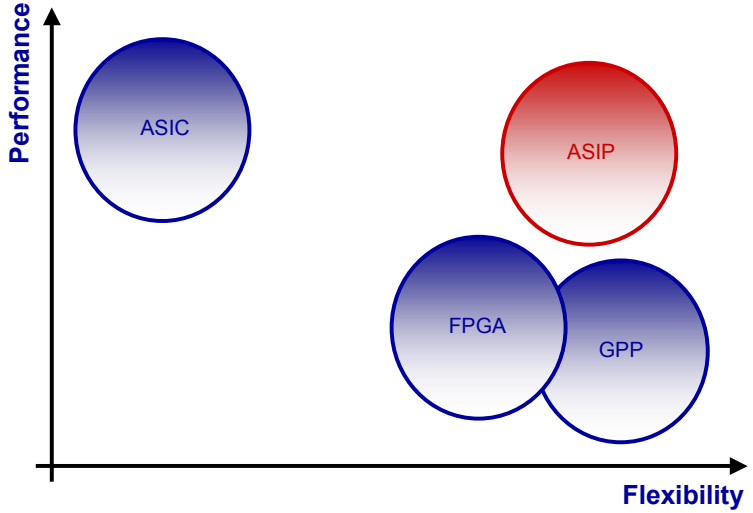
Customizing the architecture to the specific application (application domain) → Efficiency

Flexibility vs. Efficiency (Energy)

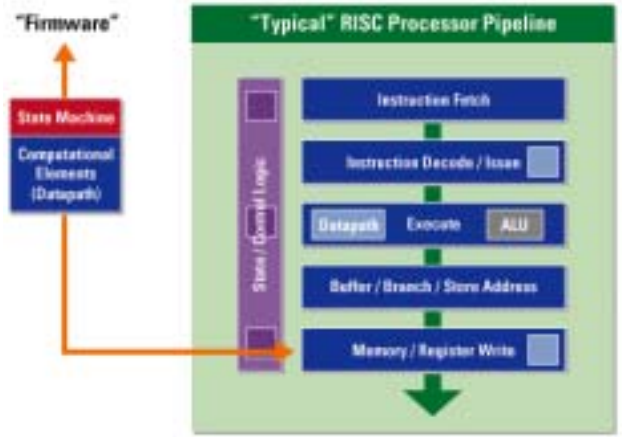


Source: Fei Sun @ ICCAD 2002

Flexibility vs. Efficiency (Performance)



Application Specific Processor



Optimized Processor Design (ASIP)

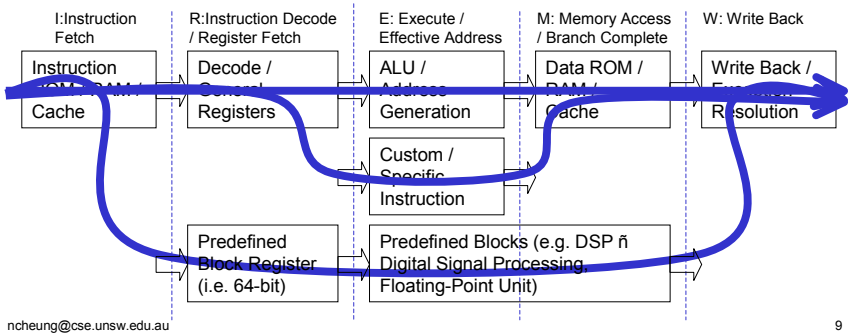
- Cost
  - Small size
- Performance
  - Application extensions
- Productivity
  - Rapid hardware and software

Extensible Processors Platform

- Represents the state-of-the-art in application specific instruction-set processor (ASIP)
- Consists of a base processor containing a base instruction set, plus the capability to customize their architecture to replace computationally intensive code segments
- The goal of designing extensible processors is typically to maximize the performance of an application, while meeting design constraints

Extensible Processors Platform

- Enables to address three architectural levels on the base processor:
  - Inclusion/Exclusion of predefined blocks
  - Instructions extension
  - Parameterizations



Outline

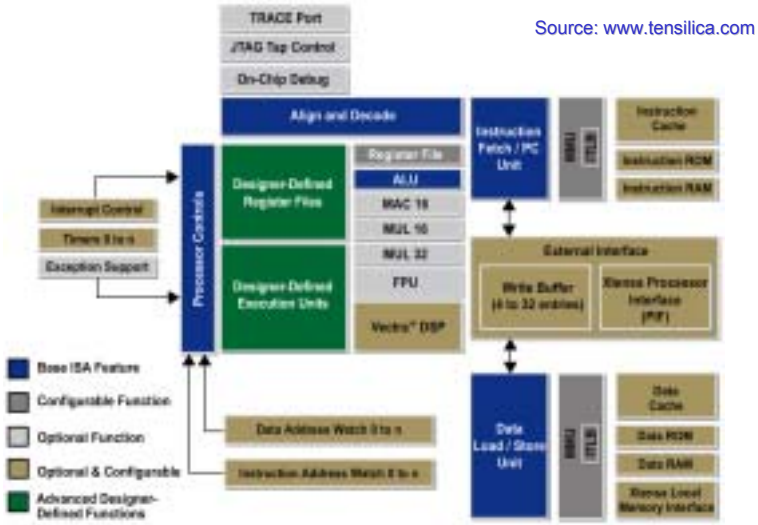
- System-On-Chips Design Challenges
- Extensible Processors Platform
- Background – Xtensa
- Problems in Extensible Processors Platform
- The Goal of the research
- Proposed Solution
- INSIDE
- MINCE
- Conclusions

Background (Xtensa)

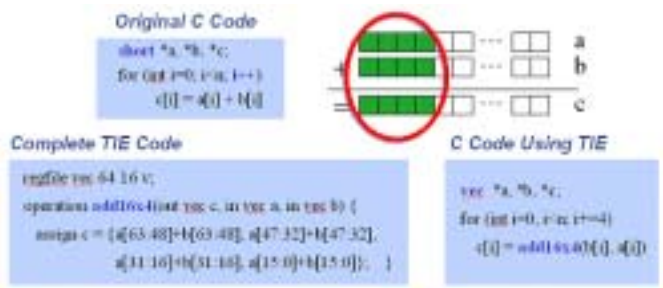
Source: www.tensilica.com

Xtensa Architecture

Source: www.tensilica.com

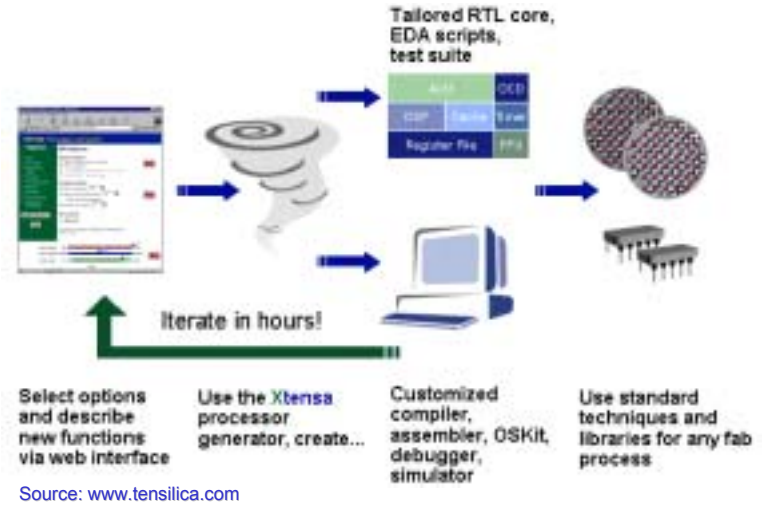


Xtensa Custom Instructions (TIE)



Source: www.tensilica.com

Xtensaís Design Flow



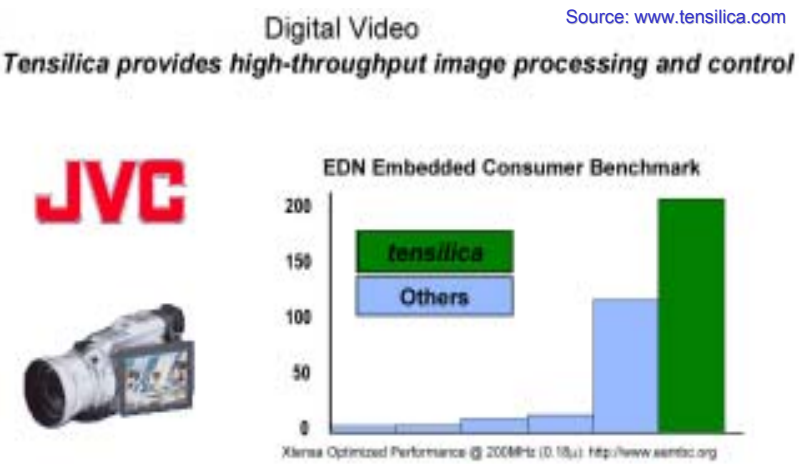
Source: www.tensilica.com

Extensible Processor (Why?)

	Standard Processor	Extensible Processor	ASIC (RTL Logic)
Application tuned data paths	NO	Yes: High-level TIE	Low-level RTL
Task Control	C/C++	C/C++	No
Simulation	Fast simulation or board	Fast simulation or board	RTL simulation: 100x slower
Multiple Engines	Limited	Simple directed MP interface	Possible, but hard to design and model

Example: Digital Video

Source: www.tensilica.com



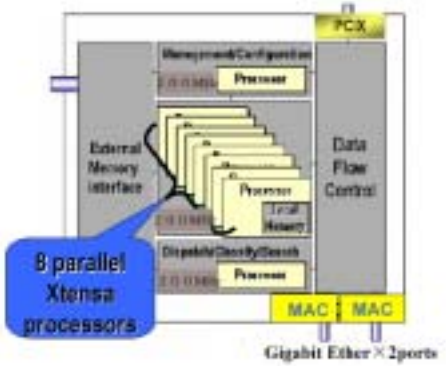
Example: Multiple Processors for TOE

TCP/IP Offload Engine (TOE)

NEC TOE can achieve full wire speed by eight parallel & two management and dispatch Tensilica cores (Total 10) for High Performance IP based Network Storage --- NAS & IP-SAN

NEC

iStorage



Source: www.tensilica.com

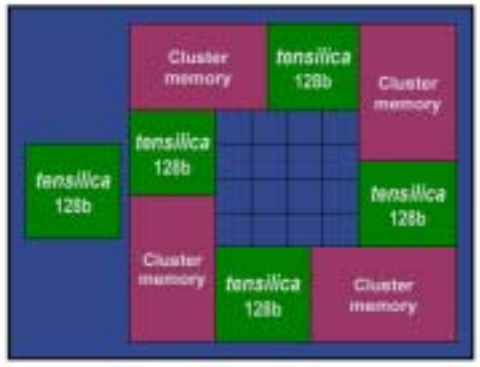
Example: Voice Gateway

Voice Gateway

Source: www.tensilica.com

Five Tensilica cores in common development system

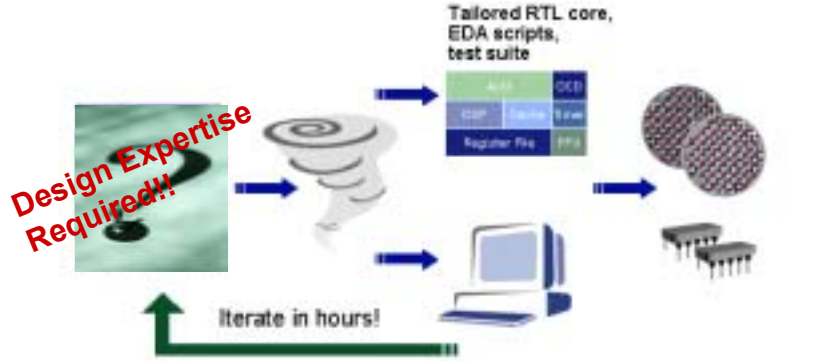
BROADCOM



Outline

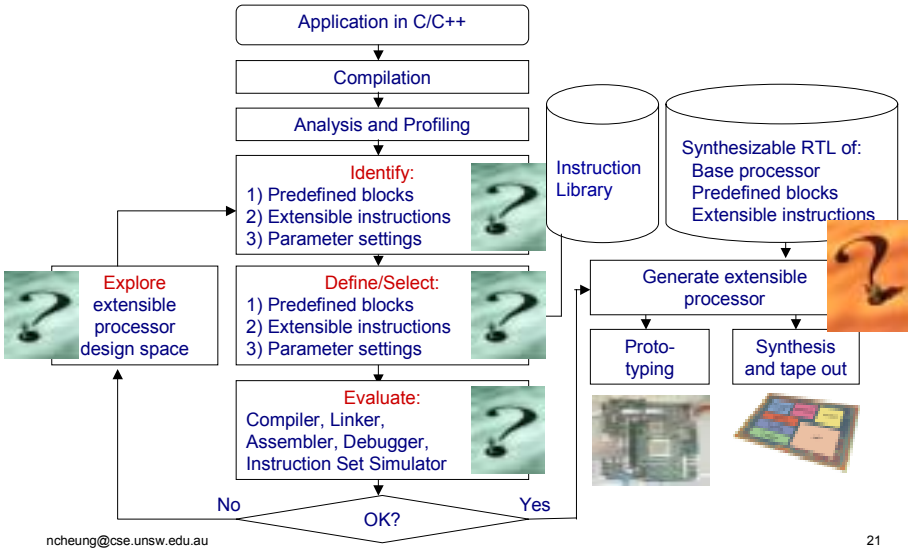
- System-On-Chips Design Challenges
- Extensible Processors Platform
- Background ñ Xtensa
- Problems in Extensible Processors Platform
- The Goal of the research
- Proposed Solution
- INSIDE
- MINCE
- Conclusions

Problems in Automation??



Source: www.tensilica.com

Generic Design Flow of Extensible Processor



Previous Works

- Profiling/Identification
  - [Binh 1998], [Yang 2002], ARC, Xtensa
- Design methodology for different aspects
  - [Gupta 2000], [Jain 2001]
- Instruction generation/selection
  - [Brisk 1998], [Kastner 2001], [Sun 2003], [Zhao 2002]
- Overall design flow for extensible processors
  - [Kathail 2002], [Lee 2002], [Sun 2002]
- Vendor and academic
  - ARC, Lisatek, Xtensa, ASIP-Meister

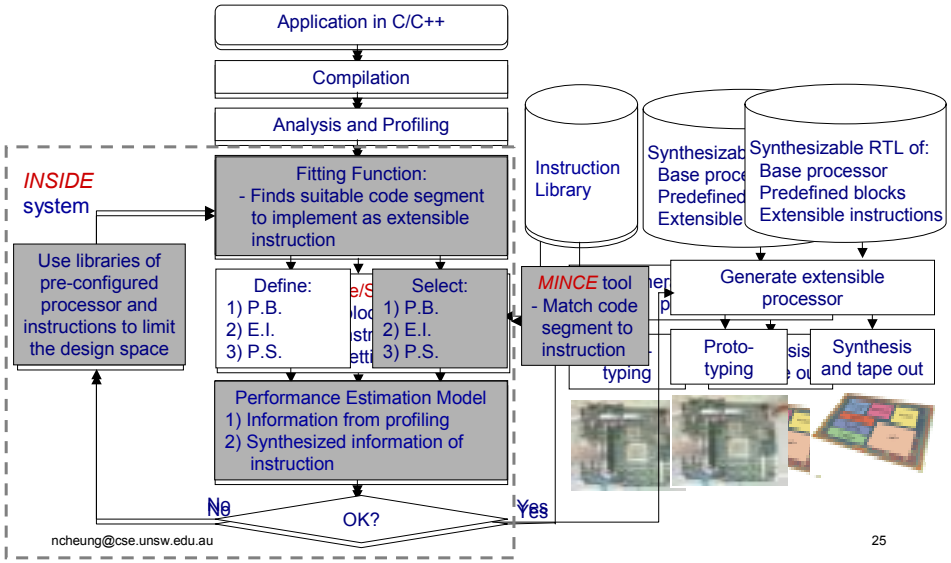
Outline

- System-On-Chips Design Challenges
- Extensible Processors Platform
- Background – Xtensa
- Problems in Extensible Processors Platform
- The Goal of the research
- Proposed Solution
- INSIDE
- MINCE
- Conclusions

Goal of the Research

- To automate the design flow of extensible processors platform.
- Given an application and design constraints, the system configures an extensible processor that maximizes the performance of an application while satisfying the design constraints.

Proposed Solution



INSIDE / MINCE

1. *INSIDE* system
   - ñ Identifies sections of code segments which are suitable for translation to instructions.
   - ñ A two-level hierarchy approach.
   - ñ A performance estimator.

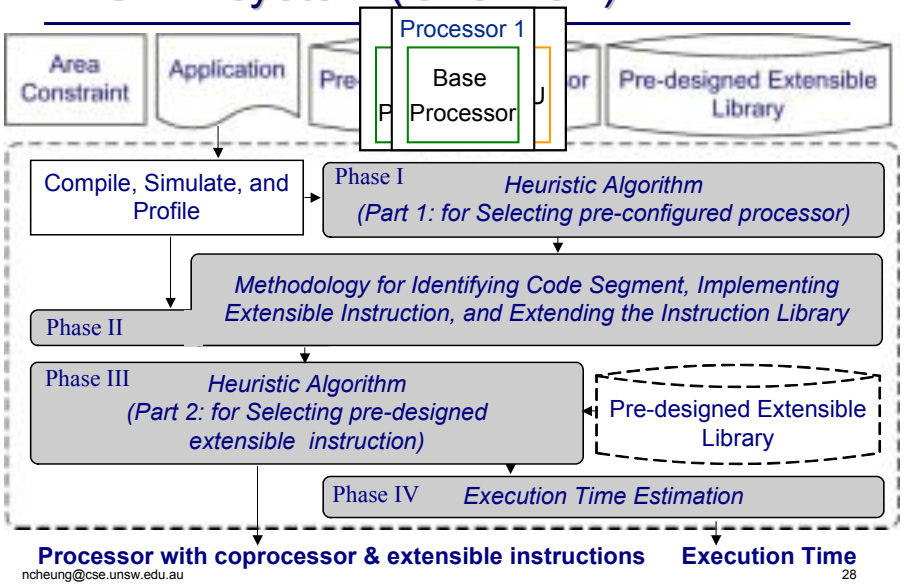
   Reduces the design turnaround time significantly.
2. *MINCE* tool
   - ñ Match code segment with pre-synthesized extensible instruction using combinational equivalence.

   Enhances reusability of the extensible instructions.

Outline

- ï System-On-Chips Design Challenges
- ï Extensible Processors Platform
- ï Background ñ Xtensa
- ï Problems in Extensible Processors Platform
- ï The Goal of the research
- ï Proposed Solution
- ï INSIDE
- ï MINCE
- ï Conclusions

INSIDE system (Overview)



Phase I: Heuristic Algorithm (Part I)

- For selecting pre-configured processor
- The area delay product, EP_i of processor *i* for a certain application

$$EP = \frac{1}{\#CC \times Period \times Area}$$

Processor	Clock Cycle	Clock Period	Area	EP
P1	12000	6ns	5000	2.78
P2	8000	8ns	8000	1.95

Phase II: Identify Code Segments

- Problem:
  - Application has millions of lines of C code, how do we know which code segment is good for converting to extensible instruction
- Fitting function
  - Identifies code segments which are suitable for translation to extensible instructions
  - Extracts characteristics of the code segment

Fitting Function

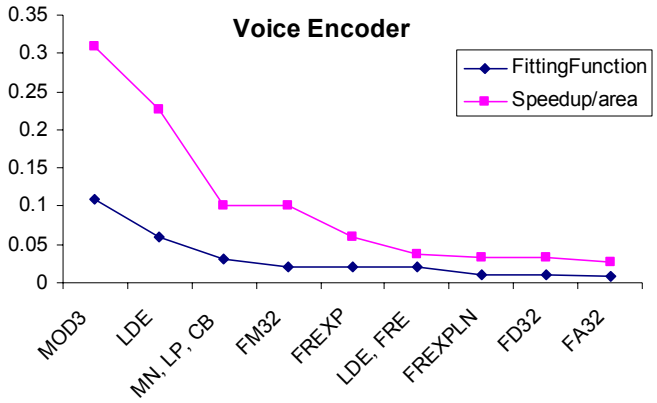
- The four characteristics:
  - The frequency of use of a code segment
  - The number of operands in a code segment
  - The percentage of integer (short) type operands in all the operands
  - The percentage of bit operations in all the operands
- The fitting function:

$$FittingFunction = F.U. \times \frac{1}{\left\lceil \frac{N.O.}{\alpha} \right\rceil} \times T.O. \times B.O.$$

α – the ideal number of operands in the code segment.

Relationship

- Relationship between the fitting function and the speedup/area ratio of the instruction



Phase III: Heuristic Algorithm (Part II)

- For selecting extensible instructions
- The potential speedup/area ratio, *PSAR*, of extensible instruction in processor:

$$PSAR = \frac{\%\_of\_\#CC \times Speedup}{Area \times Latency_{\max}}$$

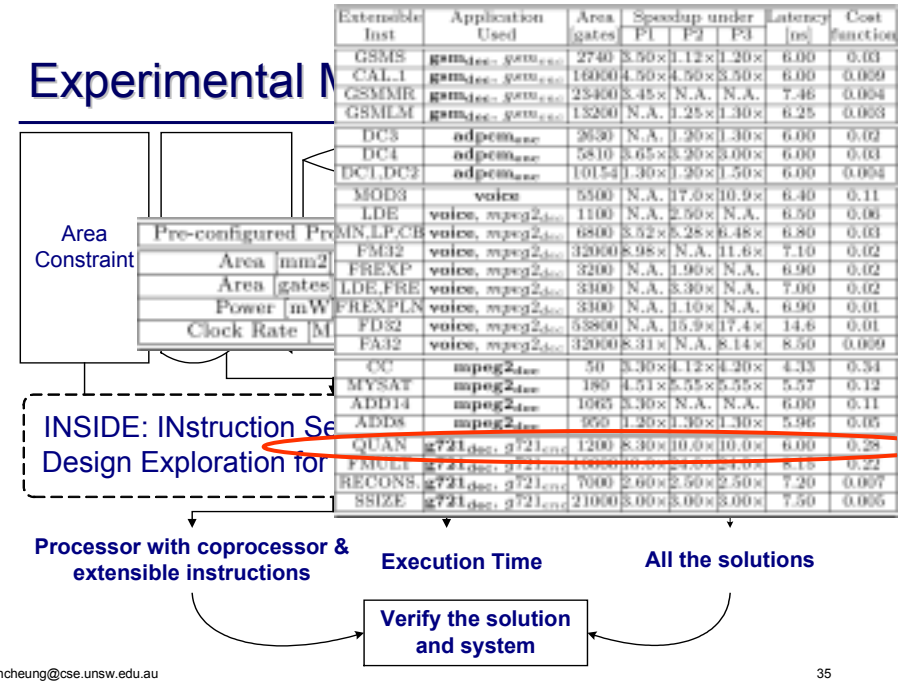
Instruction	% of total clock cycle	Speedup	Area	Latency	PSAR
Inst 1	13	3x	1500	6ns	43333
Inst 2	10	6x	2500	6ns	34286

Phase IV: Performance Estimation

- For rapidly estimating the execution time
- The execution time estimation, *ETE*, for an extensible processor with a set of selected extensible instruction:

$$ETE = \left\{ CC\_Unaffected + \frac{CC\_affected}{Speedup} \right\} \times Latency_{\max}$$

Experimental M

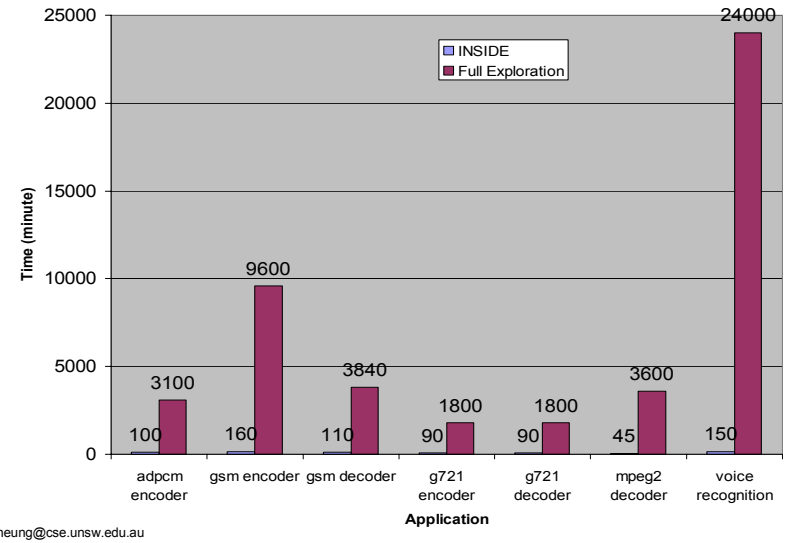


Extensible Inst	Application Used	Area [gates]	Speedup under P1	P2	P3	Latency [ns]	Cost function
GSMS	gsm_{dec}, gsm_{enc}	2740	3.50×	1.12×	1.20×	6.00	0.03
CAL.1	gsm_{dec}, gsm_{enc}	14000	4.50×	4.50×	3.50×	6.00	0.029
GSMMR	gsm_{dec}, gsm_{enc}	23400	3.45×	N.A.	N.A.	7.46	0.004
GSMLM	gsm_{dec}, gsm_{enc}	13200	N.A.	1.25×	1.30×	6.25	0.003
DC3	$adpcm_{dec}$	2630	N.A.	1.20×	1.30×	6.00	0.02
DC4	$adpcm_{dec}$	5810	3.65×	3.20×	3.00×	6.00	0.03
DC1,DC2	$adpcm_{dec}$	10154	1.30×	1.20×	1.50×	6.00	0.004
MOD3	voice	5500	N.A.	17.0×	10.9×	6.40	0.11
LDE	voice, $mpeg2_{dec}$	1100	N.A.	2.50×	N.A.	6.50	0.06
MN,LP,CB	voice, $mpeg2_{dec}$	6800	3.52×	5.28×	6.48×	6.80	0.03
FM32	voice, $mpeg2_{dec}$	32000	8.98×	N.A.	11.6×	7.10	0.02
FREXP	voice, $mpeg2_{dec}$	3200	N.A.	1.90×	N.A.	6.90	0.02
LDE,FRE	voice, $mpeg2_{dec}$	3300	N.A.	3.30×	N.A.	7.00	0.02
FREXPLN	voice, $mpeg2_{dec}$	3300	N.A.	1.10×	N.A.	6.90	0.01
FD32	voice, $mpeg2_{dec}$	53800	N.A.	15.9×	17.4×	14.6	0.01
FA32	voice, $mpeg2_{dec}$	32000	8.31×	N.A.	8.14×	8.50	0.009
CC	$mpeg2_{dec}$	50	5.30×	4.12×	4.20×	4.33	0.34
MYSAT	$mpeg2_{dec}$	180	4.51×	5.55×	5.55×	5.57	0.12
ADD14	$mpeg2_{dec}$	1065	5.30×	N.A.	N.A.	6.00	0.11
ADD8	$mpeg2_{dec}$	950	1.20×	1.30×	1.30×	5.96	0.05
QUAN	$g721_{dec}$, $g721_{enc}$	1200	8.30×	10.0×	10.0×	6.00	0.28
FMULT	$g721_{dec}$, $g721_{enc}$	1800	10.0×	4.5×	4.0×	8.15	0.22
RECONS	$g721_{dec}$, $g721_{enc}$	7000	2.60×	2.50×	2.50×	7.20	0.007
SSIZE	$g721_{dec}$, $g721_{enc}$	21000	3.00×	3.00×	3.00×	7.50	0.005

Results

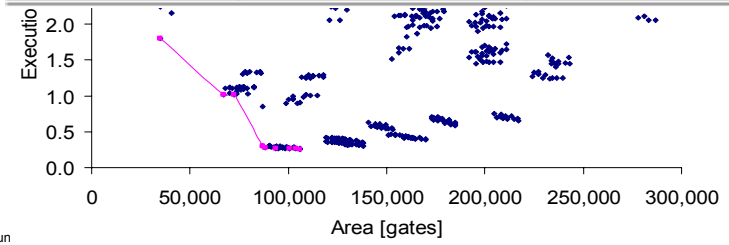
Application.	Our Best Solution wrt time		Original Solution	
	Area [gates]	Execution Time [sec.]	Area [gates]	Execution Time [sec.]
adpcm encoder	77,964	1.77	61,620	2.06
gsm encoder	79,540	13.36	61,620	13.68
gsm decoder	78,093	6.58	61,620	7.21
g721 encoder	73,200	1.96	61,620	2.69
g721 decoder	63,200	2.06	61,620	2.81
mpeg2 decoder	63,255	0.6321	61,620	0.8021
voice recognition	105,900	0.2638	61,620	1.8018

Results (Design Turnaround Time)



Results (Pareto Points)

Application.	Pareto Points obtained (Total Pareto Points)	Error rate of perf. estimation on Pareto Points [%]
adpcm encoder	3(6)	3%
gsm encoder	4(4)	7%
gsm decoder	5(5)	7%
g721 encoder	4(6)	4%
g721 decoder	2(3)	5%
mpeg2 decoder	5(5)	7%
voice recognition	9(9)	4%



Results of *INSIDE* system

- Mediabench applications
- Speedup of the applications:
  - On average 2.03x (up to 7x)
- Hardware overheads:
  - On average 25% (up to 72%)
- Pareto points:
  - Obtained on average 83% (up to 100%)
  - On average within 5% of the execution time

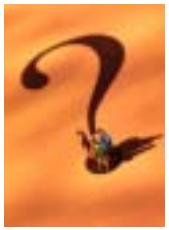
Outline

- System-On-Chips Design Challenges
- Extensible Processors Platform
- Background – Xtensa
- Problems in Extensible Processors Platform
- The Goal of the research
- Proposed Solution
- INSIDE
- MINCE
- Conclusions

The Aim of the *MINCE*

- Matching INstructions using Combinational Equivalence
- Automatically matching code segments to pre-synthesized specific instructions.

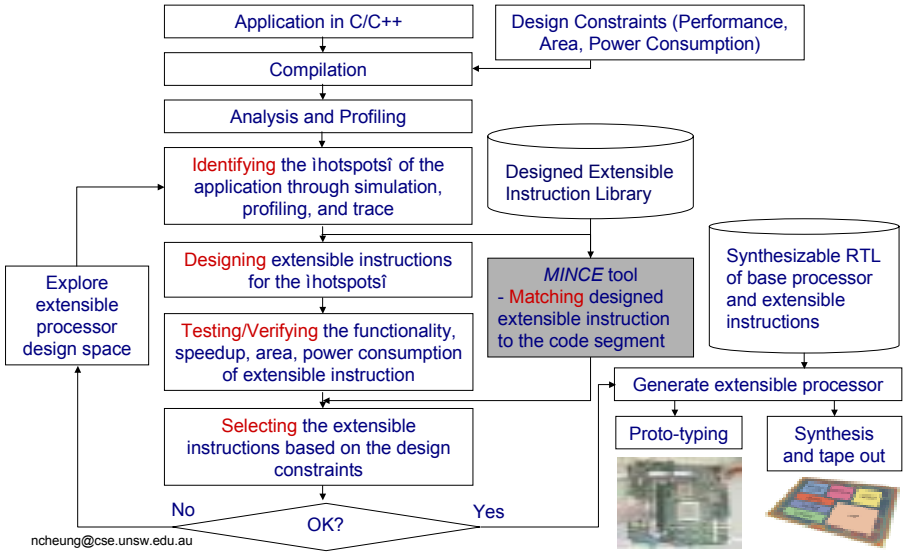
```
// Application in C
int main() {
    Ö Ö
    // Code segment
    for (x = 0; x < 100000; x++) {
        tmp = a[x] * b[x] << 4;
        total += tmp;
    }
    Ö Ö
}
```



Functional Equivalent

```
// Pre-synthesized specific instructions
state total 32
iclass ei EI {out arr, in art, in ars} {in state}
reference EI {
    wire [31;0] tmp
    assign tmp = TIEmul(art, ars, 1íb0) >> 4;
    assign arr = tmp + state;
}
```


Motivation

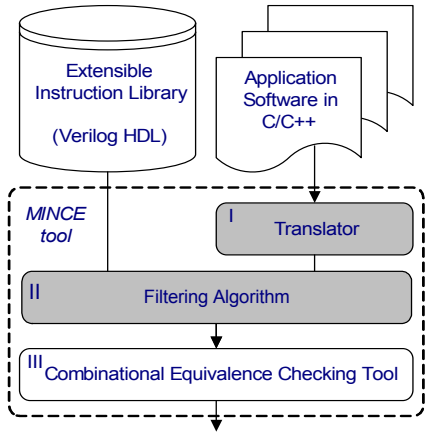


MINCE Tool

- An automated tool for matching pre-synthesized extensible instruction to the functional equivalence of code segments using combinational equivalence checking in the extensible processors platform.

MINCE Tool

- MINCE consists:
  - A translator
  - A filtering algorithm
  - A combinational equivalence checking tool



Functional equivalence implementation

Related Works

- Simulation techniques:
  [Stadler 1999]
- Graph matching techniques:
  [Corazao 1993] [Kang 1995] [Liem 1994] [Shu 1996]
- Equivalence verifications:
  [Clarke 2003], [Pnueli 1998], [Semeria 2002]

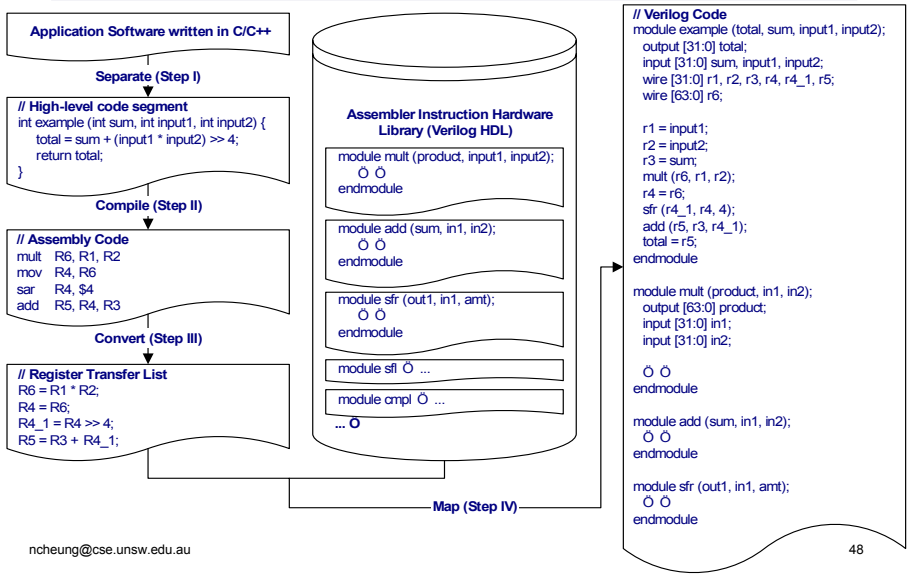
Our Contributions

- Enhances reusability of the extensible instructions.
- MINCE tool is superior to computation-intensive and error-prone simulation approaches.
- The usage of functional equivalence checking ensures that the results are largely independent of the programming style of the application.

Phase I: Translator

- The goal of the translator is to convert the application written in C/C++ to a set of code segment in Verilog HDL using a systematic approach.
- To reduce the granularity of the application written in C/C++.
- The translator consists of four steps:
  - Separate the application into code segments;
  - Compile code segments;
  - Convert to register transfer list;
  - Map to Verilog HDL file.

Translator



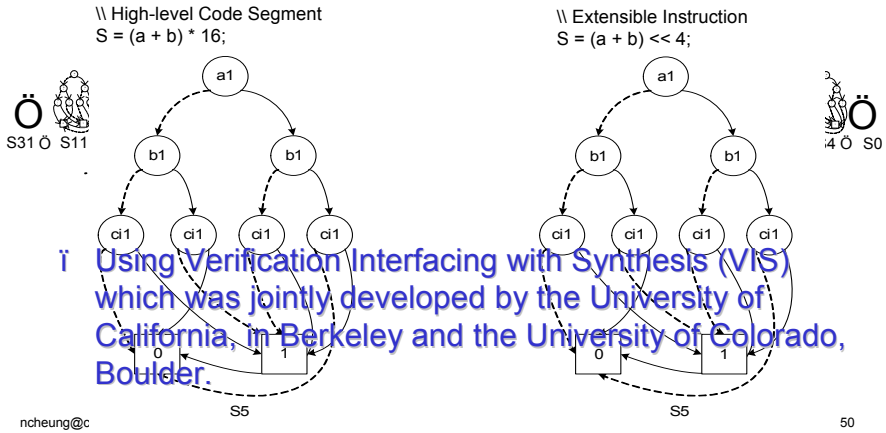
Phase II: Filtering Algorithm

- The goal of the filtering algorithm is to eliminate the unnecessary and complex Verilog HDL file into the combinational equivalence checking model.
- Verilog HDL files can be pruned as non-match:
  - Differing number of ports;
  - Differing port sizes;
  - Insufficient number of base hardware modules to present complex module.

Complex Module	Implementation - Hardware Module
Multiplier (32-bit)	Add, Shift
Multiplier (32-bit)	Multiplier (16-bit), Adder, Multiplexor
Division (32-bit)	Multiplier (32-bit), Reciprocal
Division (32-bit)	Subtract, Shift
Square Root (32-bit)	Multiplier (32-bit), Add, Subtract
Sine (32-bit)	Multiplier (32-bit), Add, Subtract
Cosine (32-bit)	Multiplier (32-bit), Add, Subtract

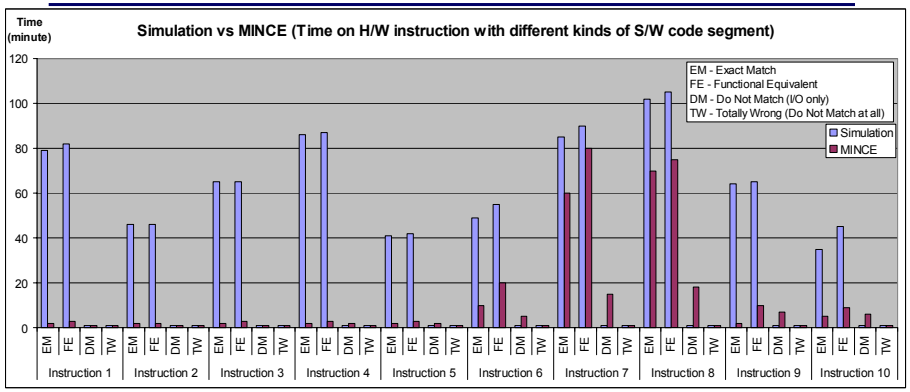
Combinational Equivalence Checking

- Binary Decision Diagram (BDD)
- For example,



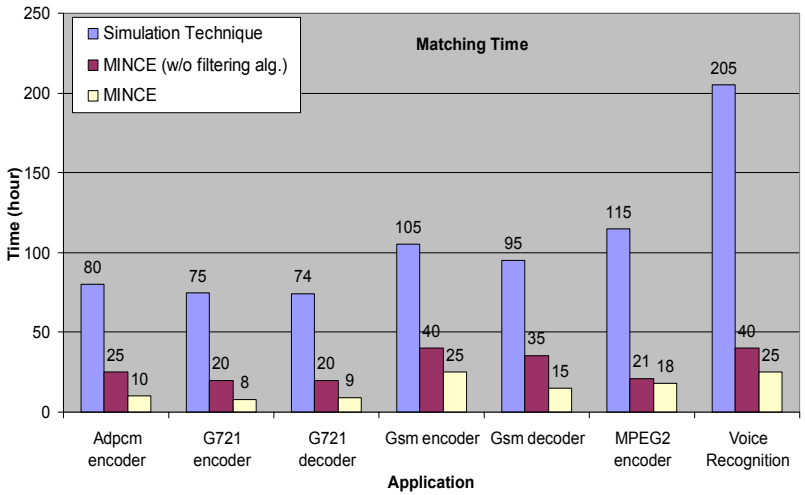
- Using Verification Interfacing with Synthesis (VIS) which was jointly developed by the University of California, in Berkeley and the University of Colorado, Boulder.

Results

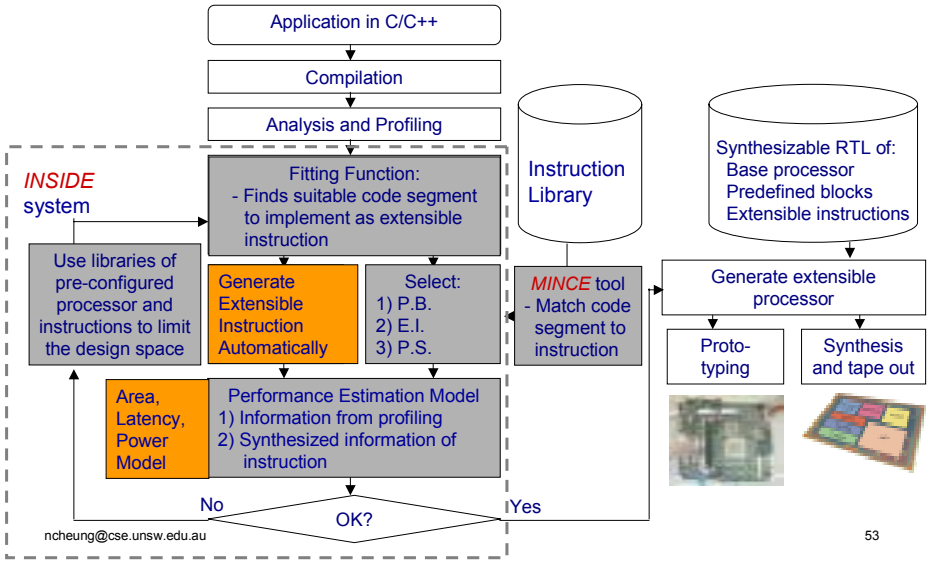


EM: Exact match
FE: Functional equivalence
DM: I/O match only
TW: Do not match

Results



Future Plan



Conclusion

- Extensible processors platform enables to address three architectural levels in order to tune for application specific:
  - Inclusion/Exclusion of predefined blocks
  - Instructions extension
  - Parameterizations
- The goal of the research is to automate the design flow of extensible processor platform.
- INSIDE system / MINCE tool