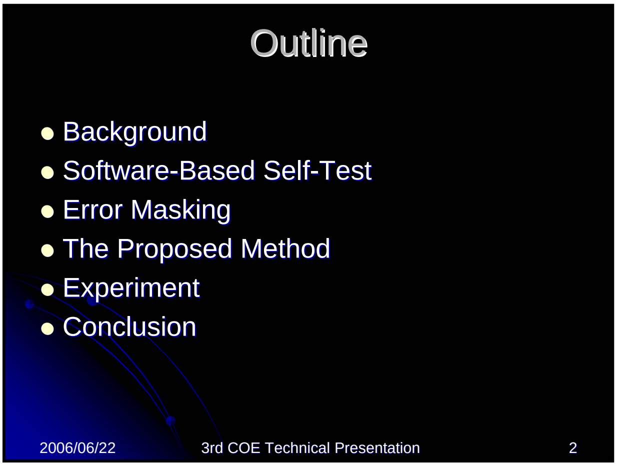
Design for Testability of Software-Based Self-Test for Processors

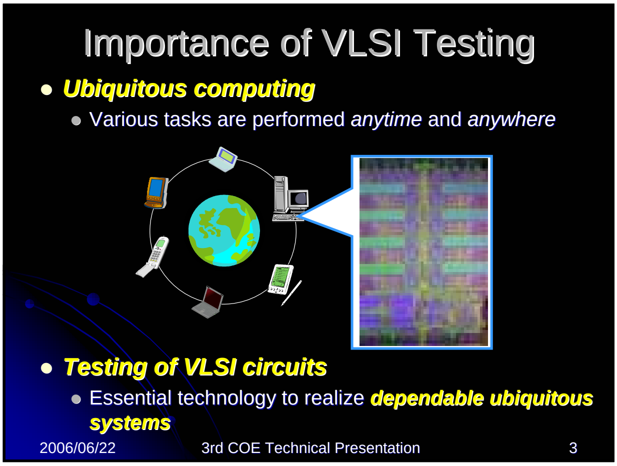
Masato NAKAZATO
Computer Design and Test Lab.

3rd COE Technical Presentation



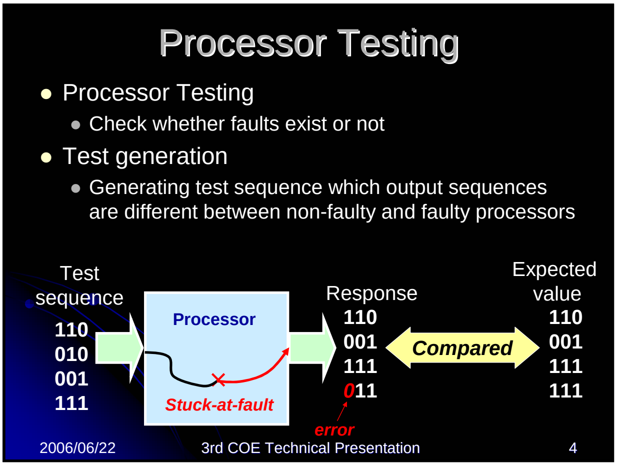
Outline

- Background
- Software-Based Self-Test
- Error Masking
- The Proposed Method
- Experiment
- Conclusion



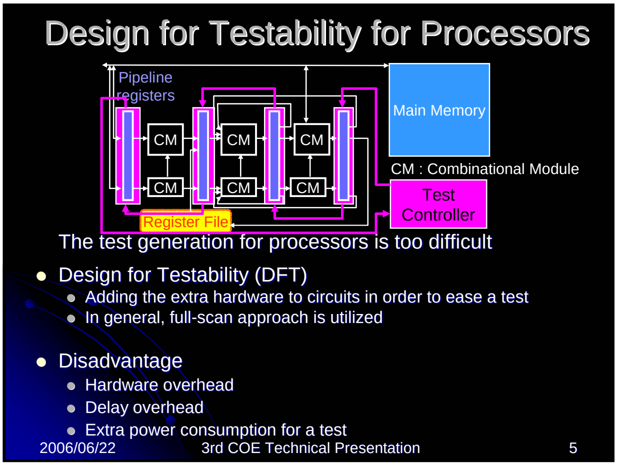
Importance of VLSI Testing

- ***Ubiquitous computing***
  - Various tasks are performed *anytime* and *anywhere*
- ***Testing of VLSI circuits***
  - Essential technology to realize ***dependable ubiquitous systems***



Processor Testing

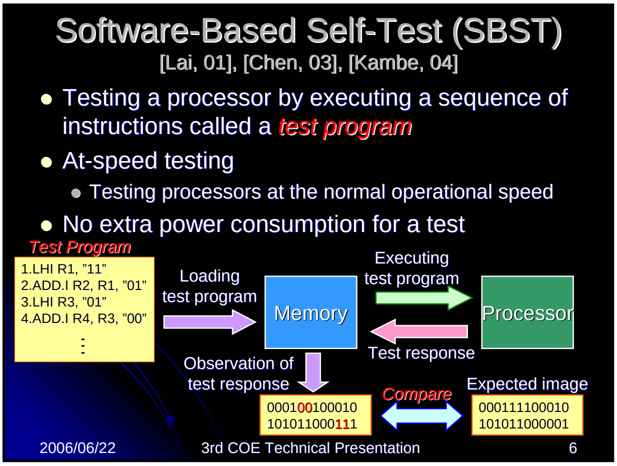
- Processor Testing
  - Check whether faults exist or not
- Test generation
  - Generating test sequence which output sequences are different between non-faulty and faulty processors



Design for Testability for Processors

The test generation for processors is too difficult

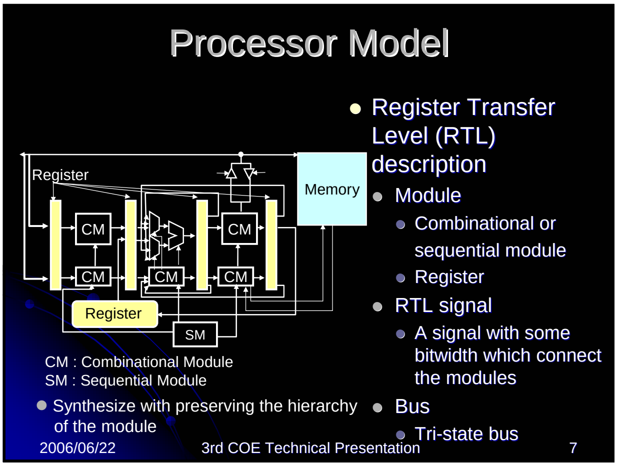
- Design for Testability (DFT)
  - Adding the extra hardware to circuits in order to ease a test
  - In general, full-scan approach is utilized
- Disadvantage
  - Hardware overhead
  - Delay overhead
  - Extra power consumption for a test



Software-Based Self-Test (SBST)

[Lai, 01], [Chen, 03], [Kambe, 04]

- Testing a processor by executing a sequence of instructions called a *test program*
- At-speed testing
  - Testing processors at the normal operational speed
- No extra power consumption for a test

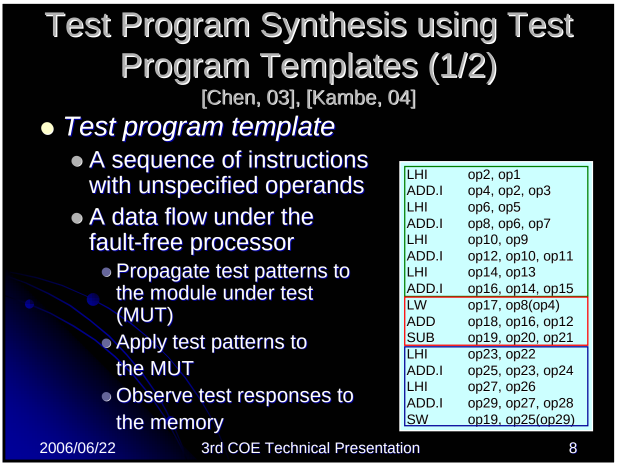


Processor Model

CM : Combinational Module
SM : Sequential Module

- Synthesize with preserving the hierarchy of the module

- Register Transfer Level (RTL) description
  - Module
    - Combinational or sequential module
    - Register
  - RTL signal
    - A signal with some bitwidth which connect the modules
  - Bus
    - Tri-state bus

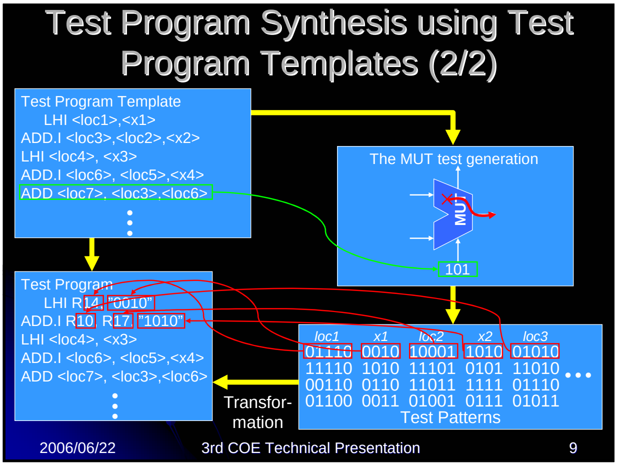


Test Program Synthesis using Test Program Templates (1/2)

[Chen, 03], [Kambe, 04]

- *Test program template*
  - A sequence of instructions with unspecified operands
  - A data flow under the fault-free processor
    - Propagate test patterns to the module under test (MUT)
    - Apply test patterns to the MUT
    - Observe test responses to the memory

LHI	op2, op1
ADD.I	op4, op2, op3
LHI	op6, op5
ADD.I	op8, op6, op7
LHI	op10, op9
ADD.I	op12, op10, op11
LHI	op14, op13
ADD.I	op16, op14, op15
LW	op17, op8(op4)
ADD	op18, op16, op12
SUB	op19, op20, op21
LHI	op23, op22
ADD.I	op25, op23, op24
LHI	op27, op26
ADD.I	op29, op27, op28
SW	op19, op25(op29)



Test Program Synthesis using Test Program Templates (2/2)

Test Program Template
LHI <loc1>,<x1>
ADD.I <loc3>,<loc2>,<x2>
LHI <loc4>, <x3>
ADD.I <loc6>, <loc5>,<x4>
ADD <loc7>, <loc3>,<loc6>
⋮

The MUT test generation

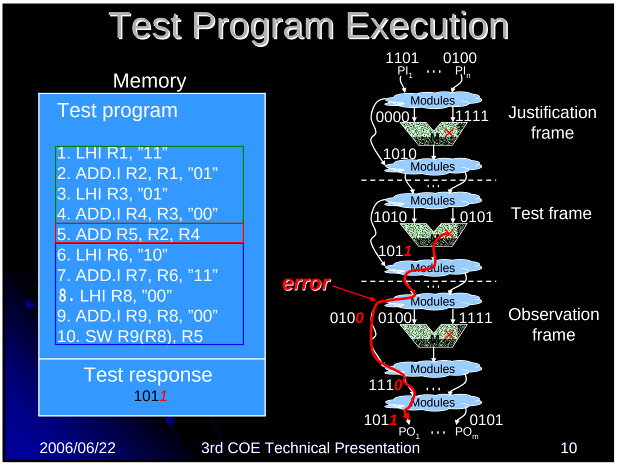
Test Program
LHI R14, "0010"
ADD.I R10, R17, "1010"
LHI <loc4>, <x3>
ADD.I <loc6>, <loc5>,<x4>
ADD <loc7>, <loc3>,<loc6>
⋮

Transfor-mation

loc1	x1	loc2	x2	loc3
01110	0010	10001	1010	01010
11110	1010	11101	0101	11010
00110	0110	11011	1111	01110
01100	0011	01001	0111	01011

• • •

Test Patterns



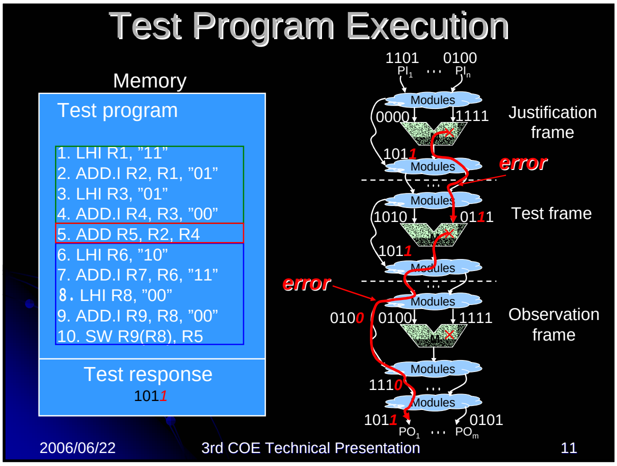
Test Program Execution

Memory

Test program

1. LHI R1, "11"
2. ADD.I R2, R1, "01"
3. LHI R3, "01"
4. ADD.I R4, R3, "00"
5. ADD R5, R2, R4
6. LHI R6, "10"
7. ADD.I R7, R6, "11"
8. LHI R8, "00"
9. ADD.I R9, R8, "00"
10. SW R9(R8), R5

Test response
1011



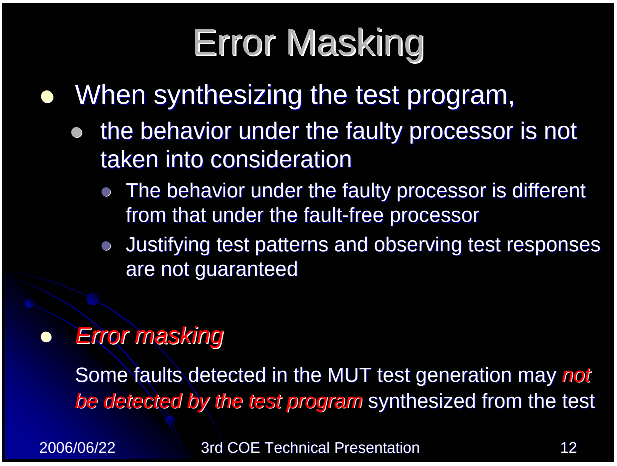
Test Program Execution

Memory

Test program

1. LHI R1, "11"
2. ADD.I R2, R1, "01"
3. LHI R3, "01"
4. ADD.I R4, R3, "00"
5. ADD R5, R2, R4
6. LHI R6, "10"
7. ADD.I R7, R6, "11"
8. LHI R8, "00"
9. ADD.I R9, R8, "00"
10. SW R9(R8), R5

Test response
1011

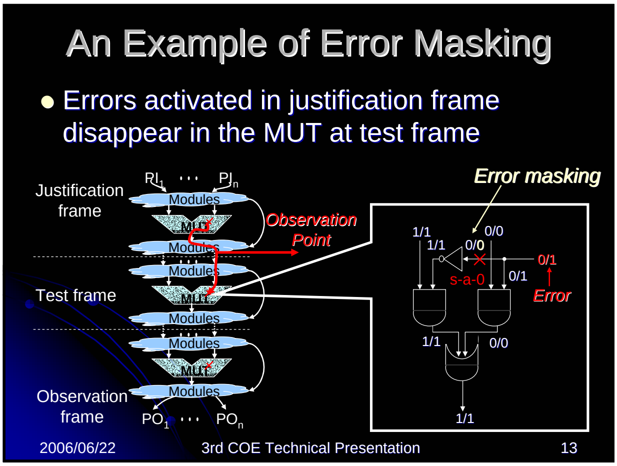


Error Masking

- When synthesizing the test program,
  - the behavior under the faulty processor is not taken into consideration
    - The behavior under the faulty processor is different from that under the fault-free processor
    - Justifying test patterns and observing test responses are not guaranteed

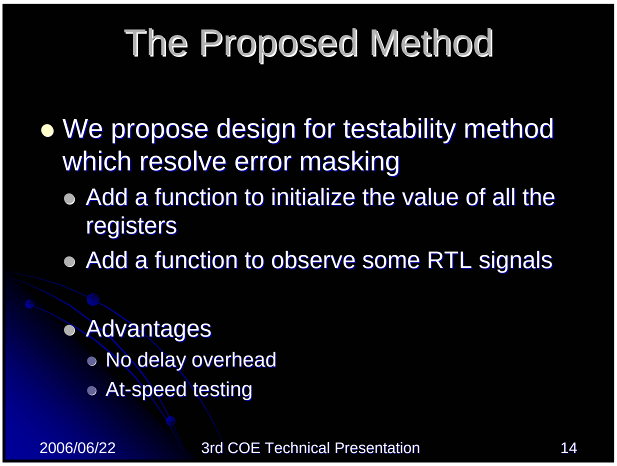
- *Error masking*

Some faults detected in the MUT test generation may *not be detected by the test program* synthesized from the test



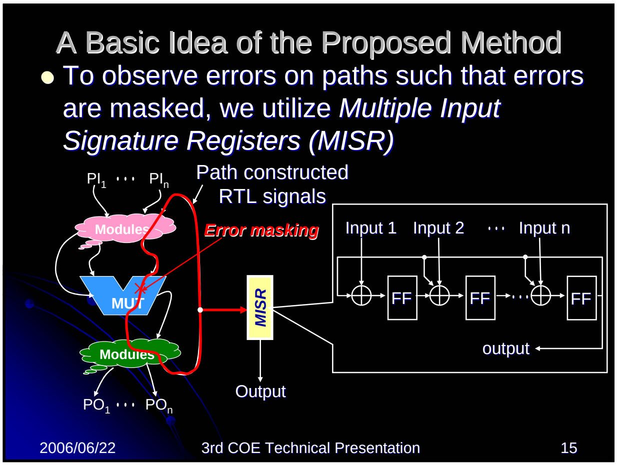
An Example of Error Masking

- Errors activated in justification frame disappear in the MUT at test frame



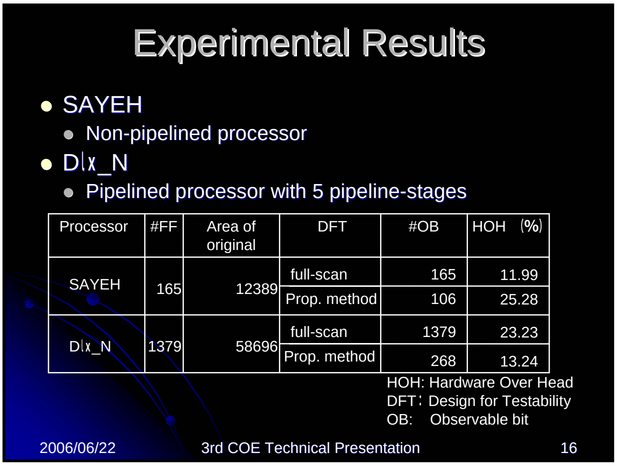
The Proposed Method

- We propose design for testability method which resolve error masking
  - Add a function to initialize the value of all the registers
  - Add a function to observe some RTL signals
  - Advantages
    - No delay overhead
    - At-speed testing



A Basic Idea of the Proposed Method

- To observe errors on paths such that errors are masked, we utilize *Multiple Input Signature Registers (MISR)*

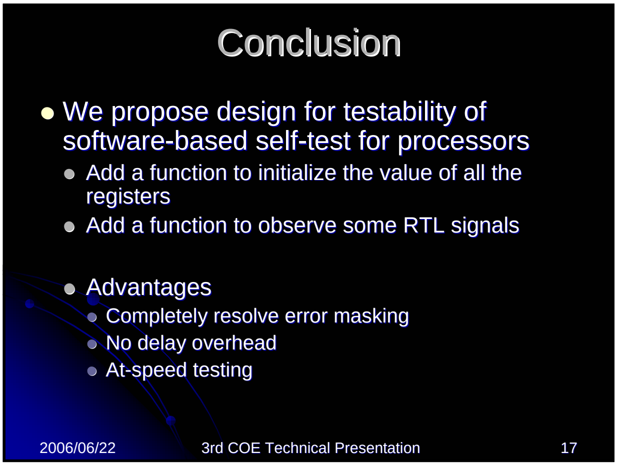


Experimental Results

- SAYEH
  - Non-pipelined processor
- Dlx_N
  - Pipelined processor with 5 pipeline-stages

Processor	#FF	Area of original	DFT	#OB	HOH (%)
SAYEH	165	12389	full-scan	165	11.99
			Prop. method	106	25.28
Dlx_N	1379	58696	full-scan	1379	23.23
			Prop. method	268	13.24

HOH: Hardware Over Head
DFT: Design for Testability
OB: Observable bit



Conclusion

- We propose design for testability of software-based self-test for processors
  - Add a function to initialize the value of all the registers
  - Add a function to observe some RTL signals
  - Advantages
    - Completely resolve error masking
    - No delay overhead
    - At-speed testing