RWTH AACHEN UNIVERSITY

Fast, Approximate Error Prediction for Unreliable Embedded Processors

M. Sc. Zheng Wang

Prof. Dr.-Ing. Anupam Chattopadhyay

Institute for Communication Technologies and Embedded Systems

Motivation

- **Reliability has become a serious design concern**

- **Fault tolerance techniques exist through design layers, however**
  - Effects of faults are difficult to evaluate under fault injection *[DAC‘13]*
  - Result in an over-protected system
  - Methodology for faults evaluation?

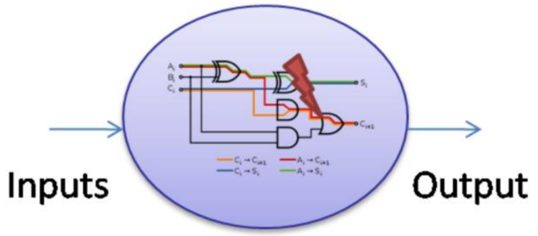
- **Solution: error prediction in architecture**
  - Analysis of logic masking effect
  - Accurate modeling of fault propagation

- **Approximate error prediction framework**
  - Integrated into commercial processor designer *[Synopsys PD]*
  - Generic for customized architecture (processor, ASIC, CGRA etc.)
  - Automated analysis toolflow

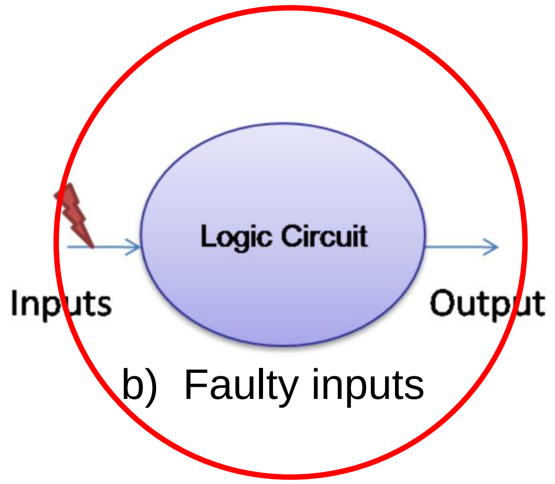
[DAC´13] H. Cho, S. Mirkhani, C. Cher, J. A. Abraham, and S. Mitra., “Quantitative Evaluation of Soft Error Injection Techniques for Robust System Design.”
[Synopsys PD] Synopsys Processor Designer: http://www.synopsys.com/IP/ProcessorIP/asip/processor-designer

Fault propagation analysis

- **Logic masking effect**



a) Faults inside circuit analysis by PTM *[KRISHNASWAMY'08]*



b) Faulty inputs

- **Probablistic error Masking Matrix (PeMM)**

Outputs *PeMM* Inputs

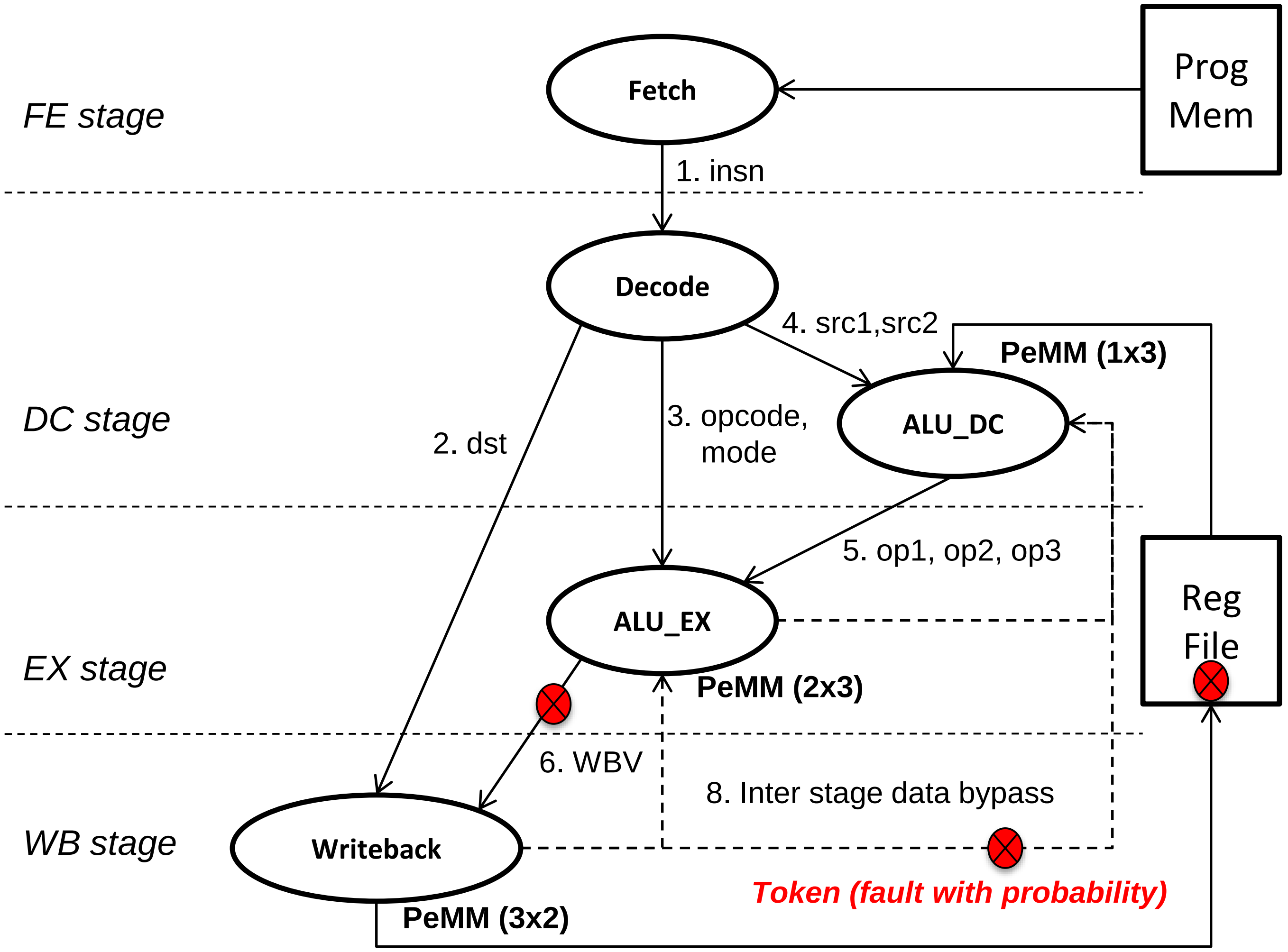
$$\begin{bmatrix} e_{out_0} \\ e_{out_1} \\ e_{out_2} \end{bmatrix} \Leftarrow \boxed{\begin{bmatrix} M_{in_0}^{out_0} & M_{in_1}^{out_0} & M_{in_2}^{out_0} & M_{in_3}^{out_0} \\ M_{in_0}^{out_1} & M_{in_1}^{out_1} & M_{in_2}^{out_1} & M_{in_3}^{out_1} \\ M_{in_0}^{out_2} & M_{in_1}^{out_2} & M_{in_2}^{out_2} & M_{in_3}^{out_2} \end{bmatrix}} \Leftarrow \begin{bmatrix} e_{in_0} \\ e_{in_1} \\ e_{in_2} \\ e_{in_3} \end{bmatrix}$$

- **Applicable to simulation model (LISA), Verilog, VHDL**

[KRISHNASWAMY'08] SMITA KRISHNASWAMY, GEORGE F. VIAMONTES, IGOR L. MARKOV, and JOHN P. HAYES., "Probabilistic transfer matrices in symbolic reliability analysis of logic circuits", TODAES, 2008

Fault propagation analysis

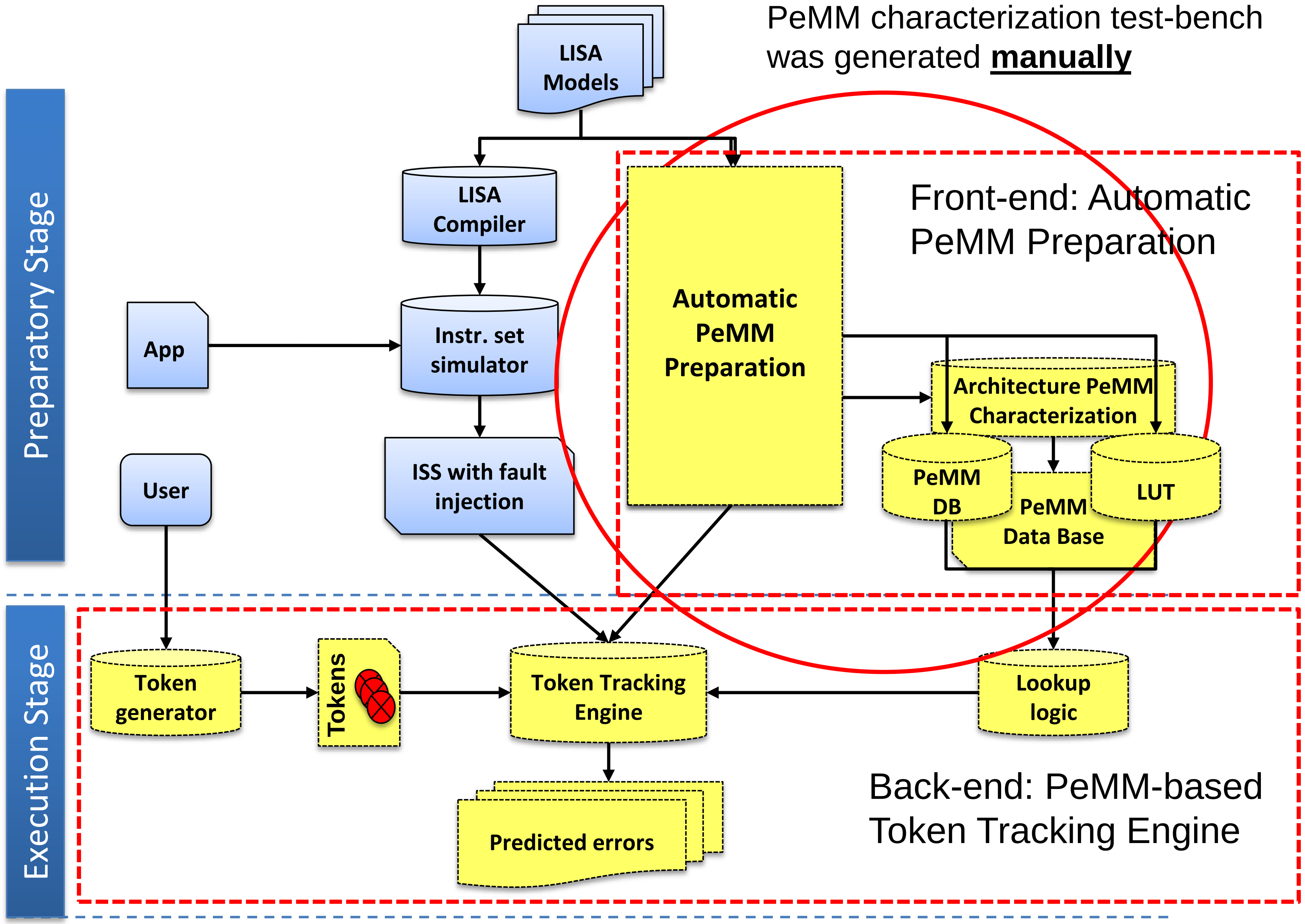
- **Fault (Token) tracking**



Example for ALU instruction

Error Prediction Framework

PeMM characterization test-bench was generated **manually**



Front-end: Automatic PeMM Preparation

- **PeMM Characterization Flow**
  - PeMM generator

ii. Record error-free outputs as OUT_g

iv. Record erroneous outputs as OUT_e

$$\begin{bmatrix} EX\_MEM\_WBV \\ EX\_MEM\_BPR \end{bmatrix} = \begin{bmatrix} M_{00} & M_{01} & M_{02} & M_{03} & M_{04} & M_{05} & M_{06} & M_{07} & M_{08} \\ M_{10} & M_{11} & M_{12} & M_{13} & M_{14} & M_{15} & M_{16} & M_{17} & M_{18} \end{bmatrix} \begin{bmatrix} EX\_MEM\_BPR \\ DC\_EX\_rs1 \\ EX\_MEM\_WBV \\ MEM\_WB\_BPR \\ MEM\_WB\_WBV \\ DC\_EX\_op1 \\ DC\_EX\_op2 \\ opcode \\ dst \end{bmatrix}$$

i. Initialize inputs with random values

iii. Inject a single bit-flip error to the input

Error-free outputs: EX_MEM_WBV_g, EX_MEM_BPR_g

Erroneous outputs: EX_MEM_WBV_e, EX_MEM_BPR_e

Compare and count:

```
if (EX_MEM_WBV_g != EX_MEM_WBV_e) EX_MEM_WBV_c++
if (EX_MEM_BPR_g != EX_MEM_BPR_e)   EX_MEM_BPR_c++
```

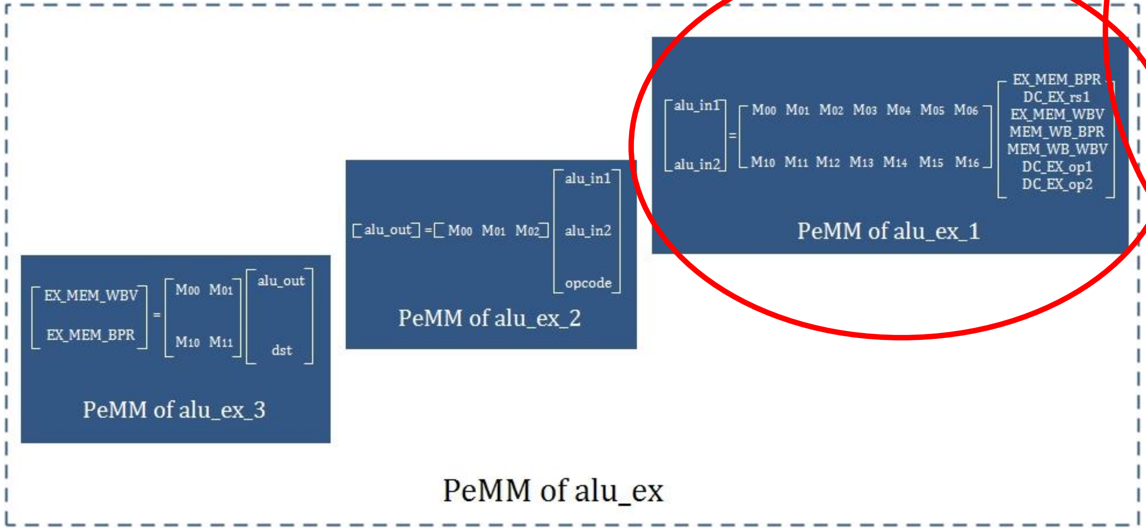
$$M_{00} = \frac{EX\_MEM\_WBV\_c}{N}$$

$$M_{10} = \frac{EX\_MEM\_BPR\_c}{N}$$

We do this process N times (accurate when N spans whole boolean space)

Front-end: Automatic PeMM Preparation

- **Accuracy Improvement**
  - Large operation
  - <u>Solution: decomposition</u>



1. Parsing *read_from*, *write_to* and *signal* lists for each child operation

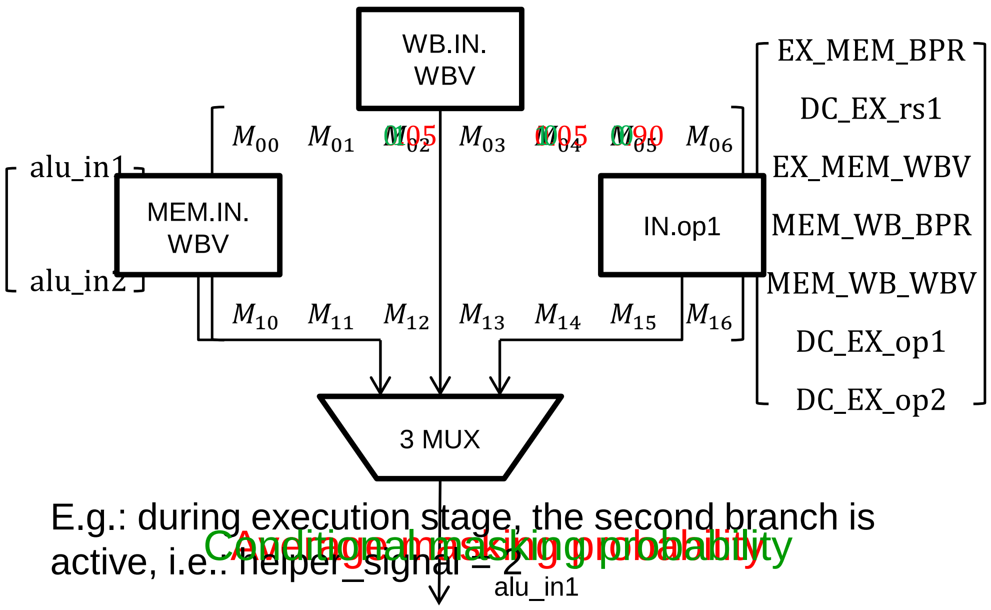
2. Characterizing PeMM for each child operation

```
OPERATION alu_ex IN pipe . EX
{
    // split
    BEHAVIOR
    {
        if (BYPASS_ACTIVE(MEM.IN.BPR, IN.rs1))
        {
            alu_in1 = MEM.IN.WBV;
        }
        else
        {
            if (BYPASS_ACTIVE(WB.IN.BPR, IN.rs1))
            {
                alu_in1 = WB.IN.WBV;
            }
            else
            {
                alu_in1 = IN.op1;
            }
        }
        alu_in2 = IN.op2;
    }                                          Data bypassing
    // split
    SWITCH (opcode)
    {
        CASE add: { BEHAVIOR { alu_out = alu_in1 +  alu_in2; }}
        CASE sub: { BEHAVIOR { alu_out = alu_in1  -  alu_in2; }}
        CASE and: { BEHAVIOR { alu_out = alu_in1 & alu_in2; }}
        CASE or   : { BEHAVIOR { alu_out = alu_in1  |  alu_in2; }}
        CASE xor : { BEHAVIOR { alu_out = alu_in1 ^  alu_in2; }}
    }                                          Arithmetic
    // split
    BEHAVIOR
    {
        if (dst != 0)
        {
            OUT.WBV = alu_out;
            OUT.BPR = dst;
        }
    }                                          Assignment
}
```

Front-end: Automatic PeMM Preparation

- **Accuracy Improvement**
  - Control flow handling
  - Solution: using helper signals visible from simulator

```
BEHAVIOR
{
    if ( BYPASS_ACTIVE(MEM.IN.BPR, IN.rs1))
    {
        alu_in1 = MEM.IN.WBV;   helper_signal = 1;
    }
    else
    {
        if ( BYPASS_ACTIVE(WB.IN.BPR, IN.rs1))
        {
            alu_in1 = WB.IN.WBV;   helper_signal = 2;
        }
        else
        {
            alu_in1 = IN.op1;   helper_signal = 3;
        }
    }
    alu_in2 = IN.op2;
}
```


WB.IN.
WBV
alu_in1
alu_in2
MEM.IN.
WBV
IN.op1
M_{00} M_{01} M_{02} M_{03} M_{04} M_{05} M_{06}
0.05 0.05 0.90
M_{10} M_{11} M_{12} M_{13} M_{14} M_{15} M_{16}
EX_MEM_BPR
DC_EX_rs1
EX_MEM_WBV
MEM_WB_BPR
MEM_WB_WBV
DC_EX_op1
DC_EX_op2
3 MUX
alu_in1
Conditional masking probability
Average masking probability

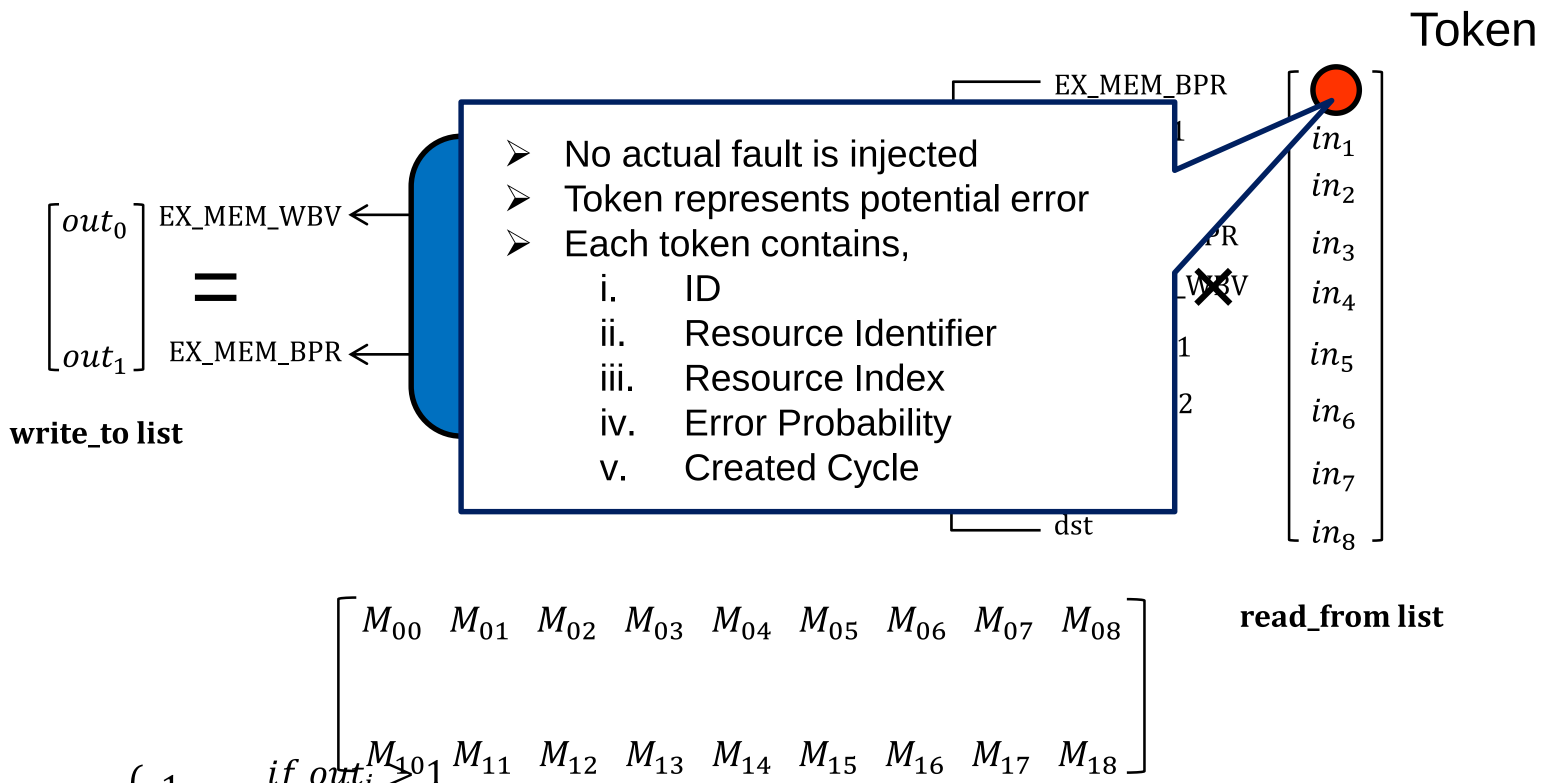
E.g.: during execution stage, the second branch is active, i.e.: helper_signal = 2

helper_signal	MEM.IN.WBV	WB.IN.WBV	IN.op1
1	1	0	0
2	0	1	0
3	0	0	1

LUT

Back-end: PeMM-based Error Tracking Engine

- **Error Propagation Modeling**
  - PeMM-based model
  - Abstract data structure: Token



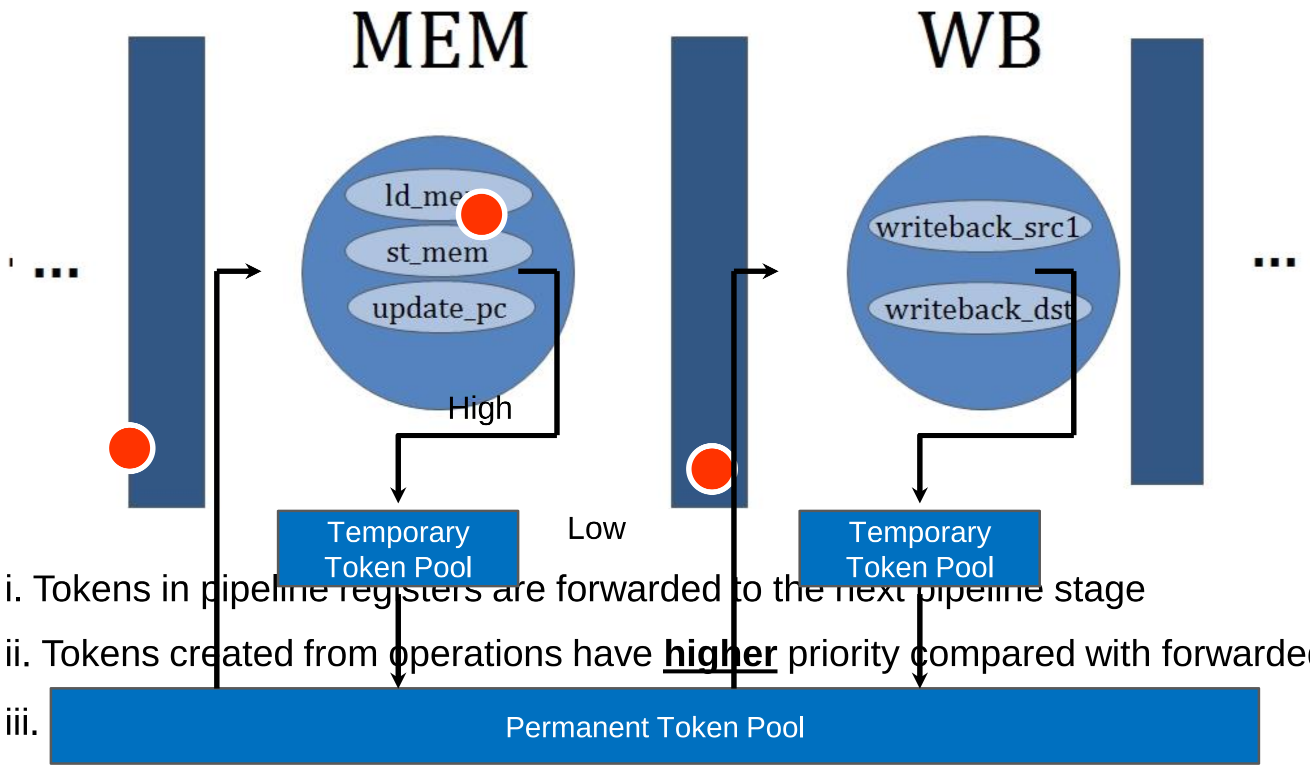
Token
EX_MEM_BPR
out_0
out_1
EX_MEM_WBV
EX_MEM_BPR
write_to list
in_1
in_2
in_3
in_4
in_5
in_6
in_7
in_8
dst
read_from list
M_{00} M_{01} M_{02} M_{03} M_{04} M_{05} M_{06} M_{07} M_{08}
M_{10} M_{11} M_{12} M_{13} M_{14} M_{15} M_{16} M_{17} M_{18}

- No actual fault is injected
- Token represents potential error
- Each token contains,
  - i. ID
  - ii. Resource Identifier
  - iii. Resource Index
  - iv. Error Probability
  - v. Created Cycle

$$out_i = \begin{cases} 1, & if\ out_i > 1 \\ 0, & if\ out_i < threshold \end{cases}$$

Back-end: PeMM-based Error Tracking Engine

- **Error Propagation Modeling**
  - Token propagation in pipeline
    - Permanent Token Pool and Temporary Token Pool



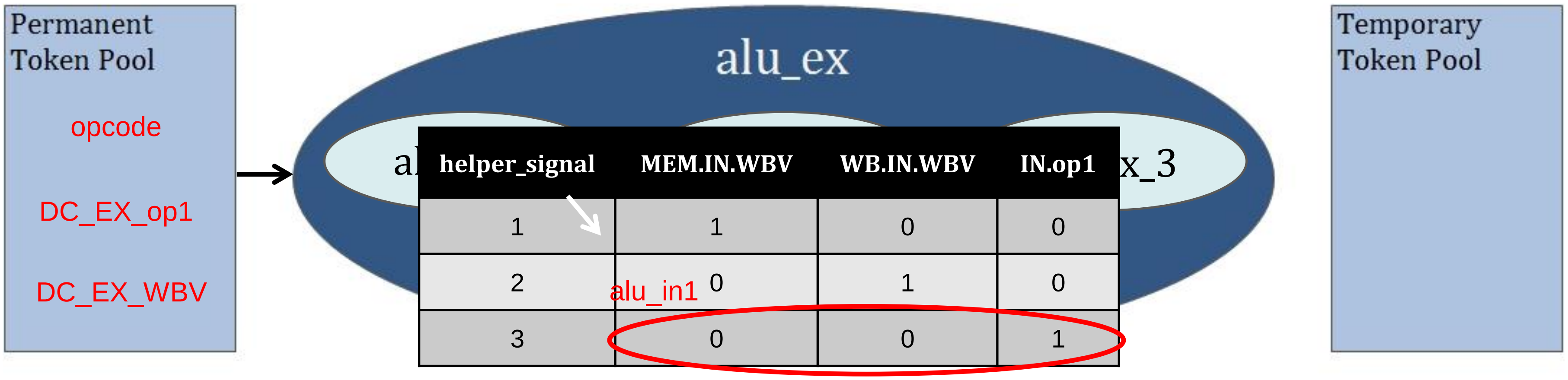
i. Tokens in pipeline registers are forwarded to the next pipeline stage

ii. Tokens created from operations have **higher** priority compared with forwarded tokens

iii.

Back-end: PeMM-based Error Tracking Engine

- Token propagation in logic operation



helper_signal	MEM.IN.WBV	WB.IN.WBV	IN.op1
1	1	0	0
2	0	1	0
3	0	0	1

LUT

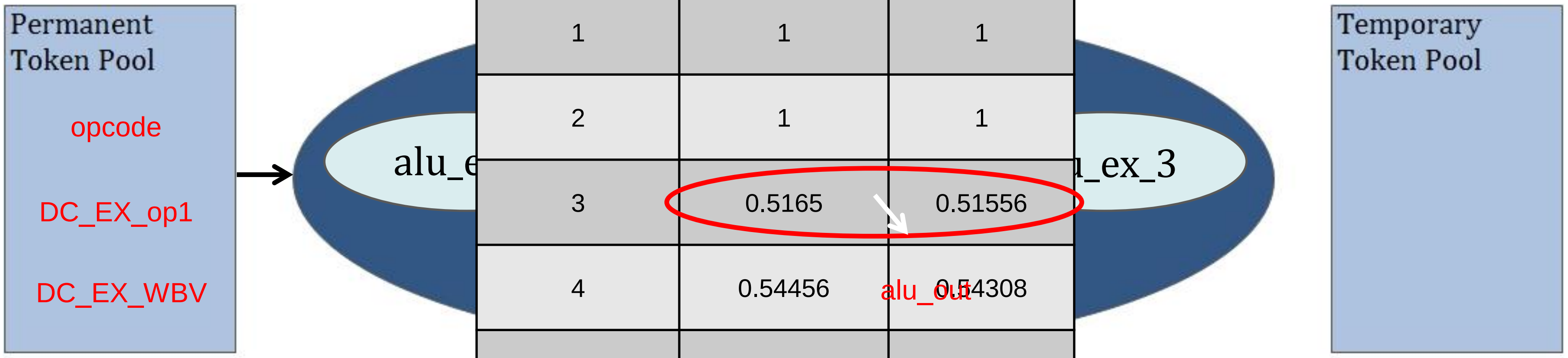
$$\begin{array}{l} \text{alu\_in1} \\ \\ \text{alu\_in2} \end{array} \begin{bmatrix} 1.00 \\ \\ 0.00 \end{bmatrix} = \begin{bmatrix} 0.09 & 0.17 & 0.05 & 0.08 & 0.05 & 0.90 & 0.0 \\ \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 1.00 \end{bmatrix} \begin{bmatrix} 0.00 \\ 0.00 \\ 0.00 \\ 0.00 \\ 0.00 \\ 1.00 \\ 0.00 \end{bmatrix} \begin{array}{l} \text{EX\_MEM\_BPR} \\ \text{DC\_EX\_rs1} \\ \text{EX\_MEM\_WBV} \\ \text{MEM\_WB\_BPR} \\ \text{MEM\_WB\_WBV} \\ \text{DC\_EX\_op1} \\ \text{DC\_EX\_op2} \end{array}$$

PeMM of Child Operation alu_ex_1

helper_signal = 3

Back-end: PeMM-based Error Tracking Engine

- Token p



opcode	alu_in1	alu_in2
1	1	1
2	1	1
3	0.5165	0.51556
4	0.54456	0.54308
5	1	1

LUT

$$\text{alu\_out} \quad \begin{bmatrix} 1.00 \end{bmatrix} = \begin{bmatrix} 1.00 & 0.52 & 0.52 \end{bmatrix} \begin{array}{c|l} \begin{bmatrix} 1.00 \\ 1.00 \\ 0.00 \end{bmatrix} & \begin{array}{l} \text{opcode} \\ \text{alu\_in1} \\ \text{alu\_in2} \end{array} \end{array}$$

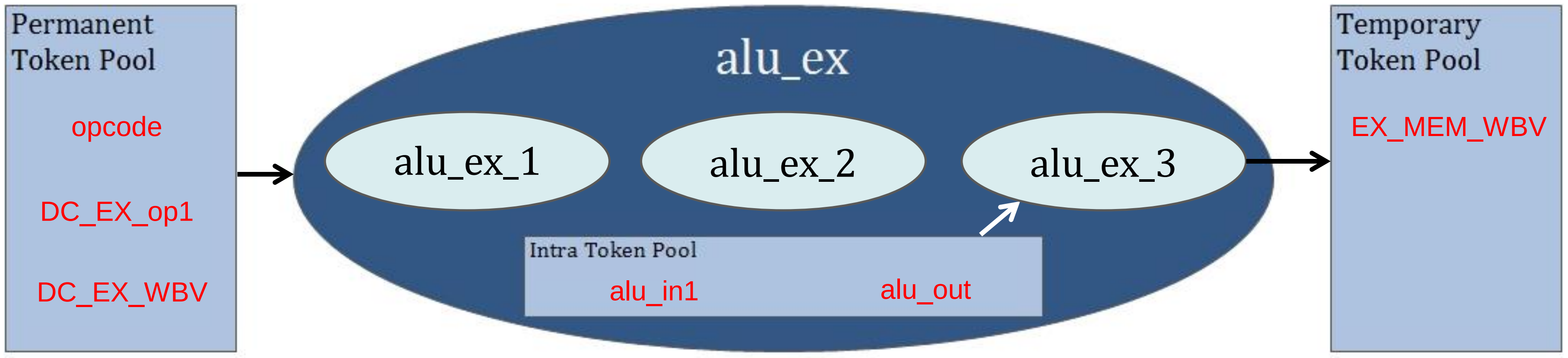
$$\begin{bmatrix} out_0 \end{bmatrix} = \begin{bmatrix} 1.00 & 0.81 & 0.81 \end{bmatrix}$$

PeMM of Child Operation alu_ex_2

opcode = 3 (AND &)

Back-end: PeMM-based Error Tracking Engine

- Token propagation in logic operation



$$\begin{array}{c}\text{EX\_MEM\_WBV}\\ \\ \text{EX\_MEM\_BPR}\end{array}\begin{bmatrix}1.00\\ \\ 0.00\end{bmatrix} = \begin{bmatrix}1.00 & 0.12\\ \\ 0.00 & 1.00\end{bmatrix}\begin{bmatrix}1.00\\ \\ 0.00\end{bmatrix}\begin{array}{c}\text{alu\_out}\\ \\ \text{dst}\end{array}$$

PeMM of Child Operation alu_ex_3

Experiment and Case Study

```
        lui    r1, 0x904f
        ldc    r2, 0x5624
        add    r3, r1, r2
        lui    r1, 0x8515
        ldc    r2, 0x4a5e
        add    r4, r1, r2

        nop
        nop
        nop
        nop

        addiu    r5, r3, 100
        addiu    r6, r3, 0
        b    L_2
L_1:
        addiu    r8, r6, 0
        addiu    r6, r5, 0
        addiu    r5, r8, 0
        b    L_2
L_2:
        cmpgt    r7= r5, r6
        bne    r7, L_1
        nop

        add    r9, r4, r4
        sub    r10, r5, r9
        and    r11, r9, r10
        or     r12, r4, r9
        xor    r13, r9, r11

        lui     r2, 0x0080
        cmpleu     r7= r3, r4
        cmpeq     r14= r14, 0x0000
        sw    r10, r2, 0x0011
        lw    r15, r2, 0x0011

        nop
        nop

_EXIT:
        nop
```

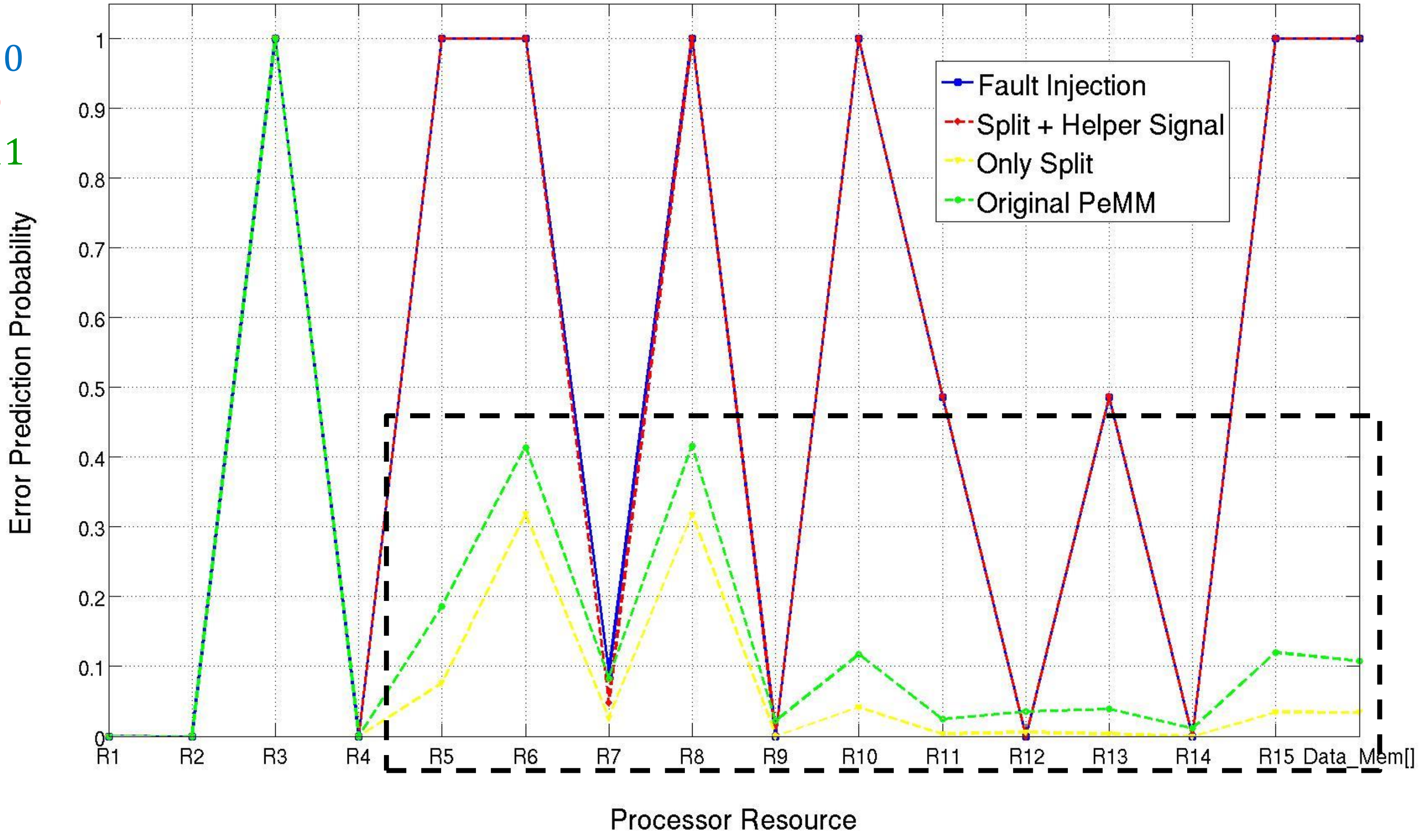
- **Experiment on LT_RISC processor (Synopsys IP)**

Activated Operation	Operation Decomposition	Using Helper Signal
cmp_rr	-	√
cmp_rr_ex	√	√
cmp_ri	-	√
cmp_ri_ex	√	√
alu_rrr	-	√
alu_rri	-	√
alu_rrr_ex	√	√
alu_rri_ex	√	√
lui_ri	-	√
ldc_ri	-	√
ld_rr	-	√
st_rr	-	√
address_generation	√	√
ld_mem	√	√
st_mem	√	√
fetch	-	√
update_pc	-	-
bra	√	√
brau	√	√
branch_execute	√	√
writeback_dst	-	√

Experiment and Case Study

- **Error Prediction Accuarcy against Fault Injection** *[ISQED'13]*
  - 1,000 testbenches with random inputs initialization
  - Test pattern for PeMM characterization : 10,000 times
  - Automatic characterization time: 2-3 seconds

```
add   r9, r4, r4
sub   r10, r5, r9
and   r11, r9, r10
or    r12, r4, r9
xor   r13, r9, r11
```



[ISQED´13] W.Zheng, C. Chen, and A. Chattopadhyay. "Fast Reliability Exploration for Embedded Processors via High-level Fault Injection."

Experiment and Case Study

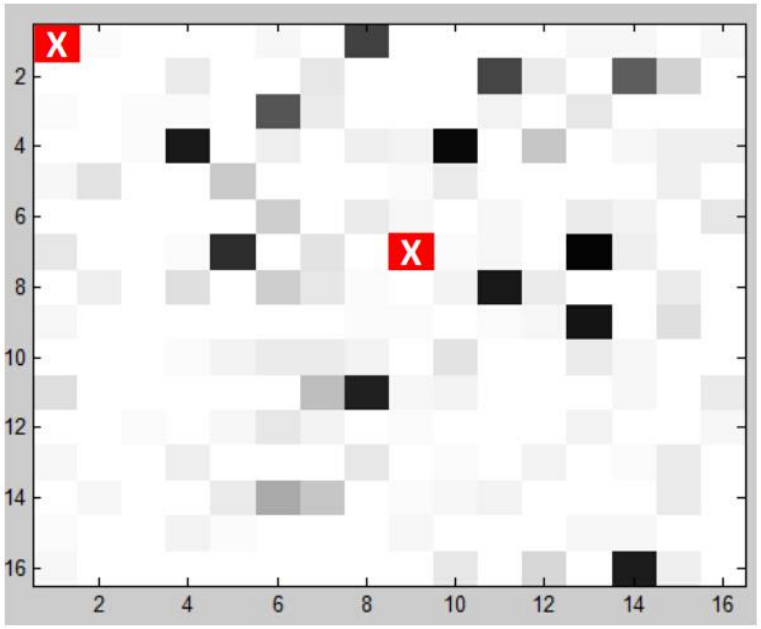
- **Execution Time for Token Tracking**
  - Comparison between O(n) *[Chafekar’13]* and O(1) implementations for token tracking engine
    - Q(n): QList for tracking
    - Q(1): QHash for tracking

Apps	0 token		1 token			20 tokens		
	O(n)	O(1)	O(n)	O(1)		O(n)	O(1)	
	(sec)	(sec)	(sec)	(sec)	Reduced Time	(sec)	(sec)	Reduced Time
Cordic	0.23	0.23	1.55	0.25	**- 83.87%**	1.98	0.33	**- 83.33%**
CRC	0.32	0.32	3.12	0.34	**- 89.11%**	5.23	0.62	**- 88.15%**
IDCT	0.26	0.28	5.83	0.29	**- 95.03%**	8.09	0.70	**- 91.35%**
Rijndael	0.35	0.35	4.75	0.39	**- 91.78%**	4.90	0.58	**- 88.17%**
Sobel	0.21	0.21	2.74	0.24	**- 91.25%**	2.87	0.64	**- 77.67%**
Viterbi	0.43	0.43	40.08	0.44	**- 98.90%**	49.82	0.81	**- 98.37%**

[Chafekar´13] Saumitra S. Chafekar, “Micro-architectural Error Propagation Analysis through Probabilistic Error Masking Matrices in Approximate Computing.” Master thesis at MPSoC Architectures, RWTH Aachen, Dec, 2013

Experiment and Case Study

- **Application-level Case Study**
  - Error prediction for median filter *[ASSP'79]*
  - Locations of errors after median filter are predicted
  - Result matches algorithm of median filter, i.e. output pixel is determined by itself and 8 surrounding pixels



Original 16x16 image with injected token

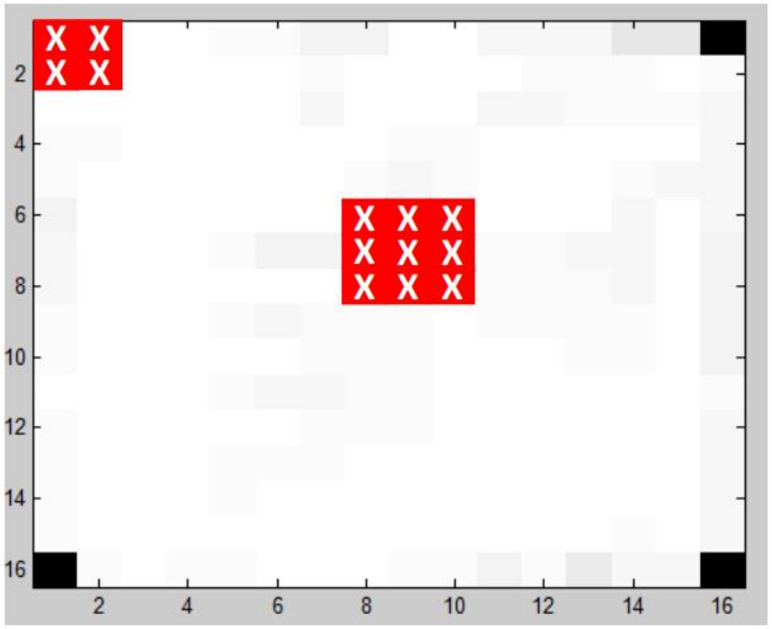


Image after median filter with predicted errors

[ASSP´79] Huang, T., G. Yang, and G. Tang. "A Fast Two-dimensional Median Filtering Algorithm."

Conclusion

- **Conclusion**
  - Fast and automatic approximate error prediction framework
  - Automatic PeMM characterization
  - Accuracy of PeMM improvement methods
  - PeMM-based error propagation investigation flow

- **Future Work**
  - Support to fine-grained PeMM characterization
  - Extend to other architecture models
  - Architecture vulnerability analysis using PeMM
  - Techniques for energy-quality tradeoff

Thank you!

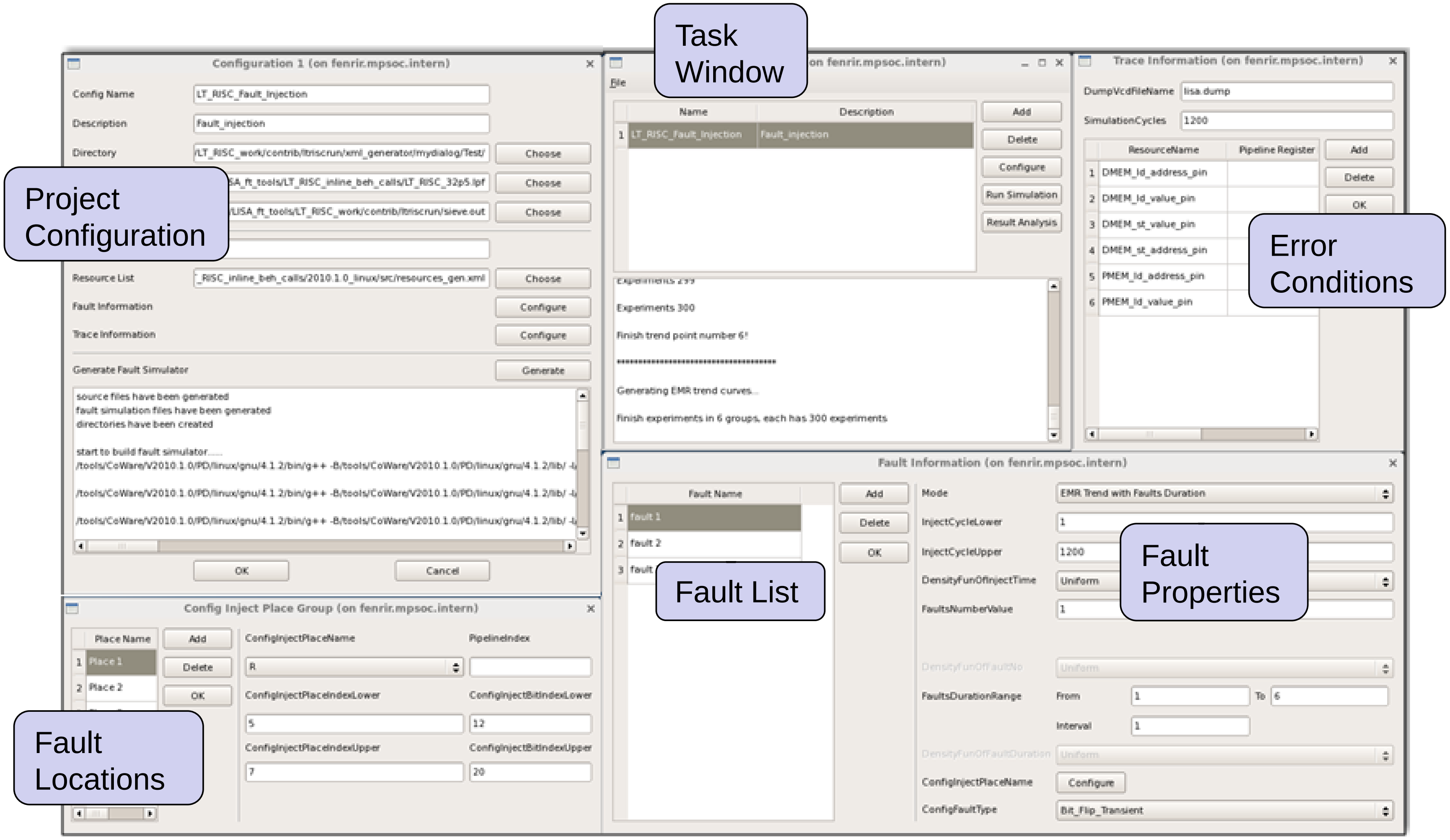
Questions?

Backups

Fault/Token Injection GUI

- **Graphical user interface**

Back



Configuration 1 (on fenrir.mpsoc.intern)

Config Name	LT_RISC_Fault_Injection
Description	fault_injection
Directory	/LT_RISC_work/contrib/ltriscrun/xml_generator/mydialog/Test/
	SA_ft_tools/LT_RISC_inline_beh_calls/LT_RISC_32p5.lpf
	/LISA_ft_tools/LT_RISC_work/contrib/ltriscrun/sieve.out
Resource List	_RISC_inline_beh_calls/2010.1.0_linux/src/resources_gen.xml
Fault Information	
Trace Information	
Generate Fault Simulator	

```
source files have been generated
fault simulation files have been generated
directories have been created

start to build fault simulator......
/tools/CoWare/V2010.1.0/PD/linux/gnu/4.1.2/bin/g++ -B/tools/CoWare/V2010.1.0/PD/linux/gnu/4.1.2/lib/ -L

/tools/CoWare/V2010.1.0/PD/linux/gnu/4.1.2/bin/g++ -B/tools/CoWare/V2010.1.0/PD/linux/gnu/4.1.2/lib/ -L

/tools/CoWare/V2010.1.0/PD/linux/gnu/4.1.2/bin/g++ -B/tools/CoWare/V2010.1.0/PD/linux/gnu/4.1.2/lib/ -L
```

on fenrir.mpsoc.intern)

	Name	Description
1	LT_RISC_Fault_Injection	Fault_injection

```
Experiments 299

Experiments 300

Finish trend point number 6!

*************************************

Generating EMR trend curves...

Finish experiments in 6 groups, each has 300 experiments
```

Trace Information (on fenrir.mpsoc.intern)

DumpVcdFileName	lisa.dump
SimulationCycles	1200

	ResourceName	Pipeline Register
1	DMEM_ld_address_pin	
2	DMEM_ld_value_pin	
3	DMEM_st_value_pin	
4	DMEM_st_address_pin	
5	PMEM_ld_address_pin	
6	PMEM_ld_value_pin	

Fault Information (on fenrir.mpsoc.intern)

	Fault Name
1	fault 1
2	fault 2
3	fault

Mode	EMR Trend with Faults Duration
InjectCycleLower	1
InjectCycleUpper	1200
DensityFunOfInjectTime	Uniform
FaultsNumberValue	1
DensityFunOfFaultNo	Uniform
FaultsDurationRange	From 1 To 6
	Interval 1
DensityFunOfFaultDuration	Uniform
ConfigInjectPlaceName	Configure
ConfigFaultType	Bit_Flip_Transient

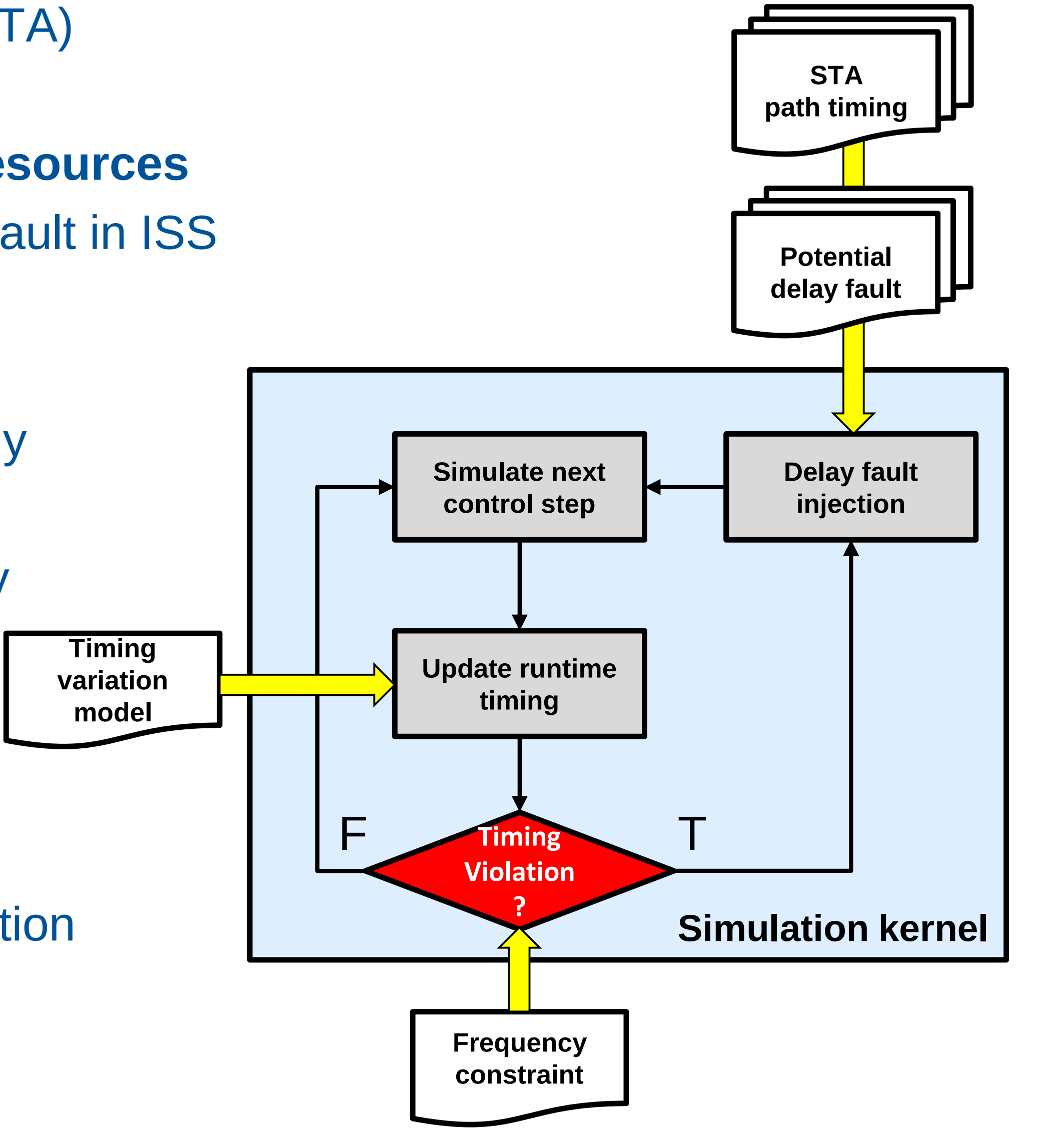
Config Inject Place Group (on fenrir.mpsoc.intern)

	Place Name
1	Place 1
2	Place 2

ConfigInjectPlaceName	R
PipelineIndex	
ConfigInjectPlaceIndexLower	5
ConfigInjectBitIndexLower	12
ConfigInjectPlaceIndexUpper	7
ConfigInjectBitIndexUpper	20

Modeling Delay Faults

- **Extract path timing from post placement & route netlist**
  - Apply static timing analysis (STA)
- **Annotate path timing for LISA resources**
  - Implement as potential delay fault in ISS
- **Runtime delay fault injection**
  - Update path timing dynamically
    - Timing variation model
  - Compare with given frequency
  - Inject during timing violation
- **Applications**
  - Frequency overscaling
  - Temperature-aware fault injection



Temperature-aware Delay Fault Injection

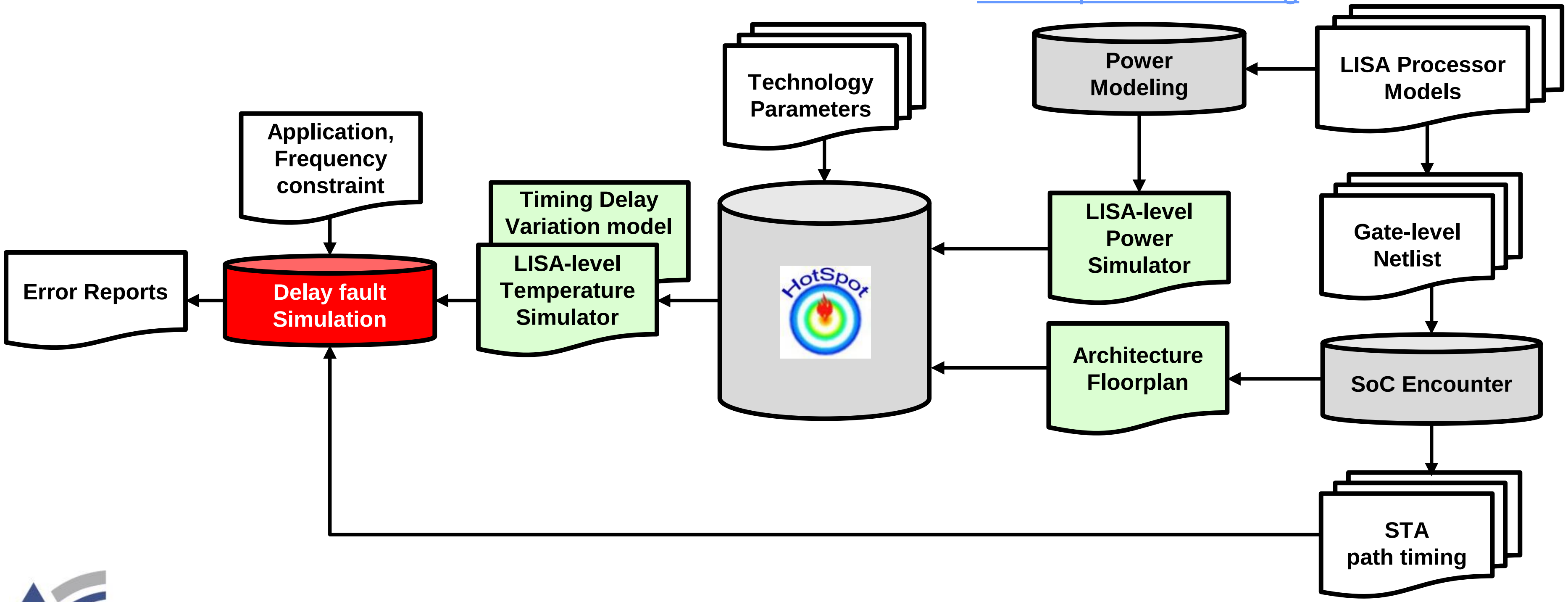
- **Temperature-aware timing variation**
  - Gate delay variation model [Lu@DAC'09]

$$\Delta d(T,\alpha,t) = \tilde{b} e^{-\frac{nE_\alpha}{kT}} \left(\frac{\alpha}{1-\alpha}\right)^n t^n$$

T: temperature, α: duty cycle, t: time

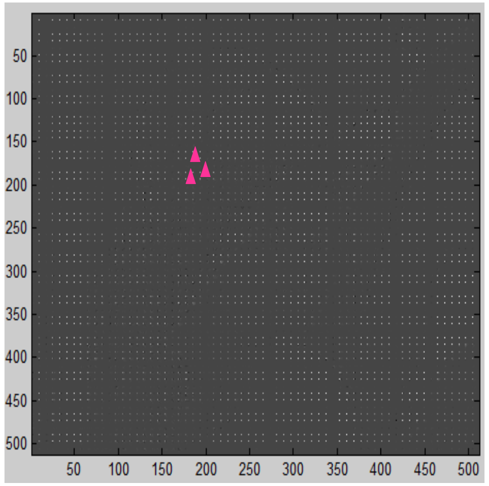
  - Approximate path delay variation using gate delay model

Link to power modeling

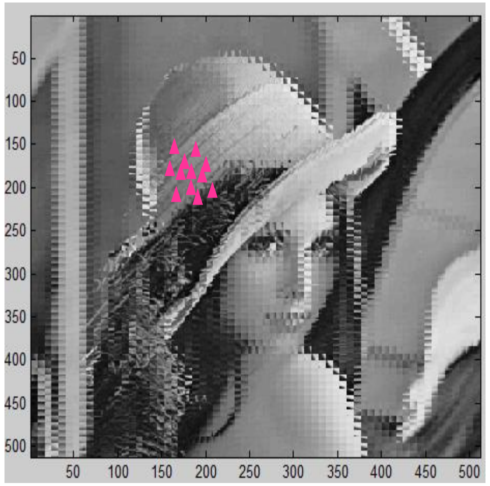


Error prediction example

Original



After DCT & Quant



After De-Quant & IDCT

- **Fine-grained error prediction**
  - Byte/nibble-wise error probabilities
  - Potential usage in approximate computing (e.g. DSP applications)
    - Predict Quality-of-Service
    - Avoid system over-protection

Token ID	Resource Name	Array Index	Created Cycle	Word Error (%)	Nibble-level Error Probability (%) (LS Nibble-MS Nibble)							
Token in bit 6 for *insn* register of *FE/DC* pipe stage @ cycle 3 ⇓												
11	R	3	8	100	0	100	23.5	1.48	0.11	0.005	0.001	0
14	R	4	9	98.42	0	84.4	32.5	2.04	0.13	0.01	0	0
18	R	6	11	96.49	0	84.4	32.5	2.04	0.13	0.01	0	0
21	R	7	12	95.54	0	84.4	32.5	2.04	0.13	0.01	0	0
28	R	11	14	50.87	0	50.2	0	0	0	0	0	0
30	R	5	15	50.87	0	50.2	0	0	0	0	0	0
32	R	12	16	56.54	0	68.7	0	0	0	0	0	0
33	dmem	0x7FFF	17	54.27	0	49.6	50.4	0	0	0	0	0
Token in bit 1 for *WBV* register of *MEM/WB* pipe stage @ cycle 17 ⇓												
3	dmem	0x7FFF	17	100	49.8	50.2	0	0	0	0	0	0
Token in bit 15 for *WBV* register of *EX/MEM* pipe stage @ cycle 9 ⇓												
5	R	6	11	100	0	0	0	100	23.1	1.5	0.10	0.01
6	R	7	12	98.42	0	0	0	84.3	32.4	2.1	0.14	0.01

Location **Timing** **Significance**