# Implementation and Evaluation of Soft-Error Resilient Method for *OASIS* Network-on-Chip

Ryunosuke Murakami
Adaptive Systems Laboratory
Supervised by Professor Yuichi Okuyama

# OUTLINE

- Background
- Motivation
- Research goal
- Approach
- Evaluation results
- Conclusion

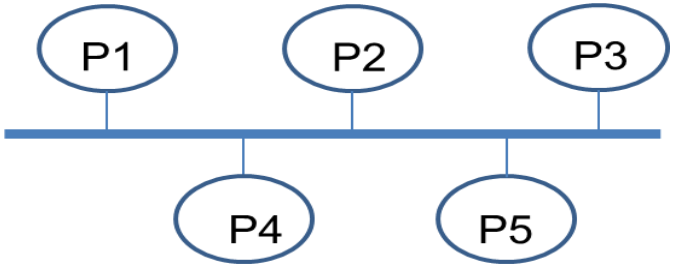
# Background

## ***System-on-Chip***

- Major communication backbone is *shared-bus*

Problems

  - low scalability
  - no parallelism
  - long wire problem




**Shared-bus**

- Demand for Many-core system is increasing

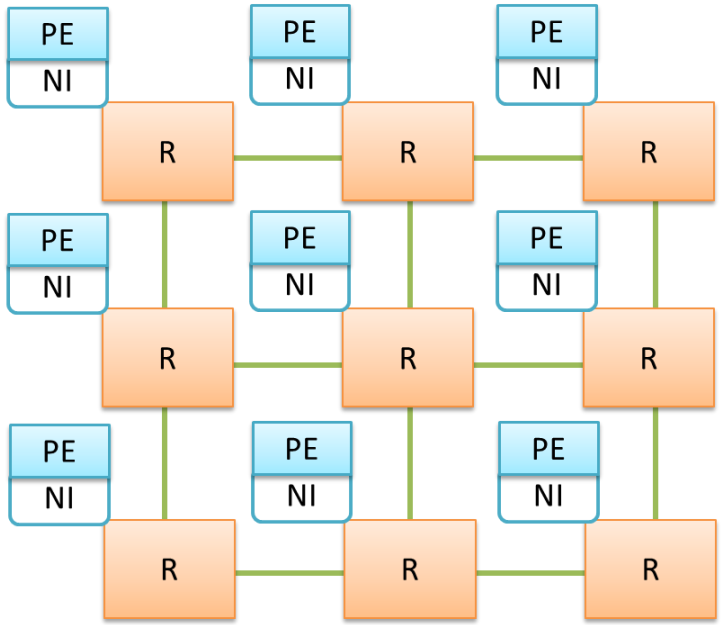
Efficient global interconnect is necessary

# Network-on-Chip

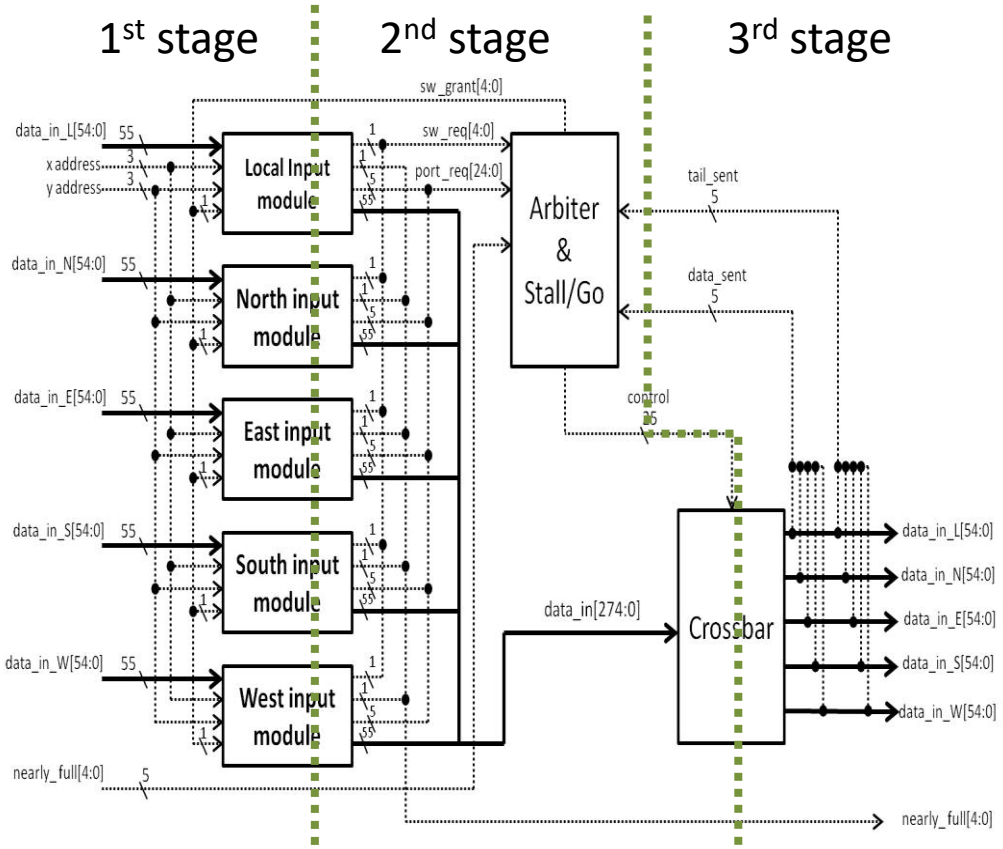
- Suitable for next generation *Many-core SoC*
  - Scalable and reusable
  - Data access is parallel
  - Low power consumption

PE: Processing element
R: Router
NI: Network interface

Network-on-Chip



# *OASIS* Network-on-Chip




1st stage:
BW(Buffer writing):
Receive data and store
2nd stage:
NPC(Next port computing):
Calculate next-out put port
SA(Switch allocation):
Arbitrating input ports to output ports
3rd stage:
CT(Crossbar traversal):
Route ports from input ports to output ports

Target is Multi / Many-core system

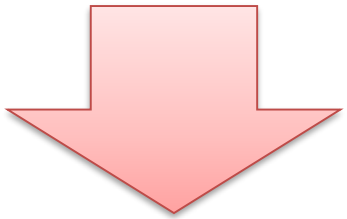
# Motivation

**As device scale shrinks,**
**on-chip interconnects are becoming a critical bottleneck**

- Scaling of supply voltages
- Increasing wire density
- Faster clock rates

Transient faults

- Data corruption
- Crosstalk noise
- Jitter problem



No error protection

- Performance degrade
- System crash

Soft-error resilience is critical problem

# Research goal

- Provide resilience transfer from soft-errors
- Data corruption

⇒ Error correcting code(ECC) is suitable

Single error correction, double error detection(*SECDED*)

*Achieve high level error protection with little performance degradation and small power overhead*

# *SECDED*

- Process quickly, however ability is not high
  - 1 bit error case: correct error
  - 2 bit error case: just detect error
- ***Intended for safety-critical applications***

```
Error =  | syndrome_bit;
One_error = Error & (^ syndrome_bit);
Double_error = Error & ! (^ syndrome_bit);
```

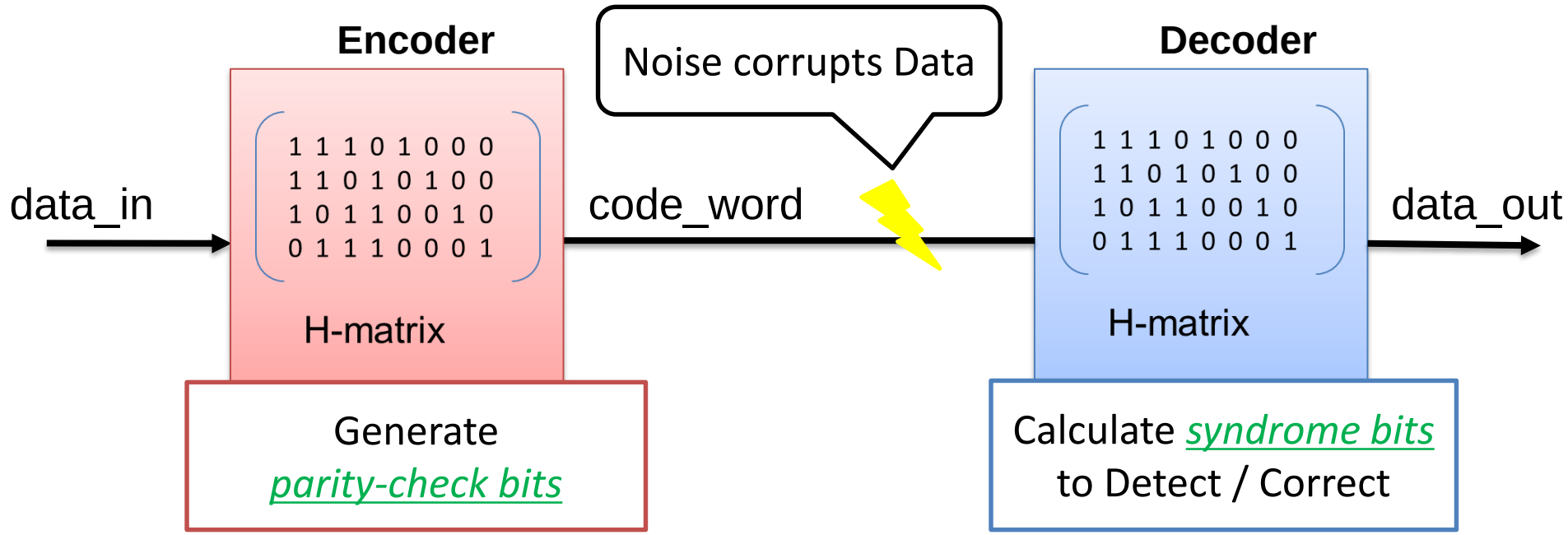
**Optimal minimum-odd-weight column *SECDED* written in verilogHDL**

# SECDED modules

*H-matrix*: contains relationship of parity-check and syndrome

*Parity-check bits: code for error check*

*Syndrome bits:* specify error (0, 1, 2-bit error)



# Encoding

Data: 1 0 1 1

$d_0$ $d_1$ $d_2$ $d_3$ $c_0$ $c_1$ $c_2$ $c_3$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |

(8, 4) H-matrix

*Parity check bits*

$c_0 = d_0 \wedge d_1 \wedge d_2 = 1 \wedge 0 \wedge 1 = 0$

$c_1 = d_0 \wedge d_1 \wedge d_3 = 1 \wedge 0 \wedge 1 = 0$

$c_2 = d_0 \wedge d_2 \wedge d_3 = 1 \wedge 1 \wedge 1 = 1$

$c_3 = d_1 \wedge d_2 \wedge d_3 = 0 \wedge 1 \wedge 1 = 0$

Code: 1 0 1 1 0 0 1 0

# Decoding - no error case

Code: 1 0 1 1 0 0 1 0

| | $d_0$ | $d_1$ | $d_2$ | $d_3$ | $c_0$ | $c_1$ | $c_2$ | $c_3$ |
|---|---|---|---|---|---|---|---|---|
| $s_0$ | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| $s_1$ | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| $s_2$ | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| $s_3$ | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |

(8, 4) H-matrix

*Syndrome bits*

$s_0 = c_0 \wedge d_0 \wedge d_1 \wedge d_2 = 0 \wedge 1 \wedge 0 \wedge 1 = 0$

$s_1 = c_1 \wedge d_0 \wedge d_1 \wedge d_3 = 0 \wedge 1 \wedge 0 \wedge 1 = 0$

$s_2 = c_2 \wedge d_0 \wedge d_2 \wedge d_3 = 1 \wedge 1 \wedge 1 \wedge 1 = 0$

$s_3 = c_3 \wedge d_1 \wedge d_2 \wedge d_3 = 0 \wedge 0 \wedge 1 \wedge 1 = 0$

**No error occurred**

# Decoding - single error case

Wrong Code: 1 1 1 1 0 0 1 0

| | $d_0$ | $d_1$ | $d_2$ | $d_3$ | $c_0$ | $c_1$ | $c_2$ | $c_3$ |
|---|---|---|---|---|---|---|---|---|
| $s_0$ | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| $s_1$ | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| $s_2$ | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| $s_3$ | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |

(8, 4) H-matrix

$s_0 = c_0 \wedge d_0 \wedge d_1 \wedge d_2 = 0 \wedge 1 \wedge 1 \wedge 1 = 1$

$s_1 = c_1 \wedge d_0 \wedge d_1 \wedge d_3 = 0 \wedge 1 \wedge 1 \wedge 1 = 1$

$s_2 = c_2 \wedge d_0 \wedge d_2 \wedge d_3 = 1 \wedge 1 \wedge 1 \wedge 1 = 0$

$s_3 = c_3 \wedge d_1 \wedge d_2 \wedge d_3 = 0 \wedge 1 \wedge 1 \wedge 1 = 1$

$d_1$ is wrong (Invert for correction)

1111 ⟹ 1011

# Decoding - double error case

Wrong Code: 1 1 1 0 0 0 1 0

| | $d_0$ | $d_1$ | $d_2$ | $d_3$ | $c_0$ | $c_1$ | $c_2$ | $c_3$ |
|---|---|---|---|---|---|---|---|---|
| $s_0$ | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| $s_1$ | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| $s_2$ | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| $s_3$ | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |

(8, 4) H-matrix

$s_0 = c_0 \wedge d_0 \wedge d_1 \wedge d_2 = 0 \wedge 1 \wedge 1 \wedge 1 = 1$

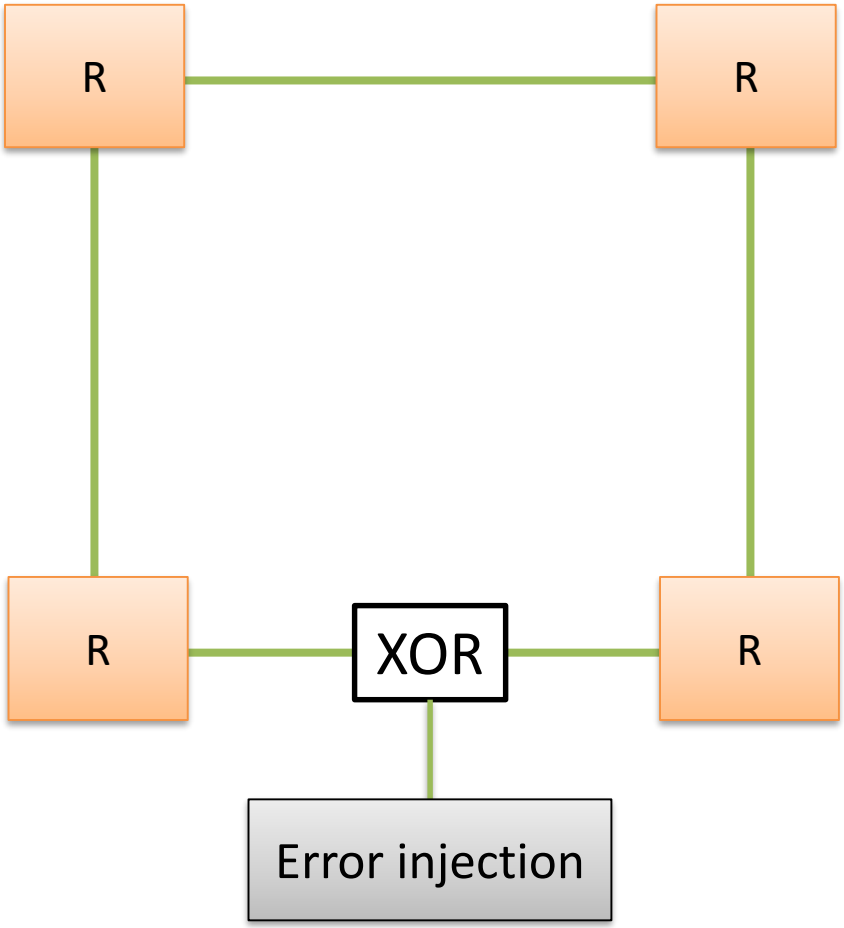
$s_1 = c_1 \wedge d_0 \wedge d_1 \wedge d_3 = 0 \wedge 1 \wedge 1 \wedge 0 = 0$

$s_2 = c_2 \wedge d_0 \wedge d_2 \wedge d_3 = 1 \wedge 1 \wedge 1 \wedge 0 = 1$

$s_3 = c_3 \wedge d_1 \wedge d_2 \wedge d_3 = 0 \wedge 1 \wedge 1 \wedge 0 = 0$

Mismatch: just detect

# Implementation



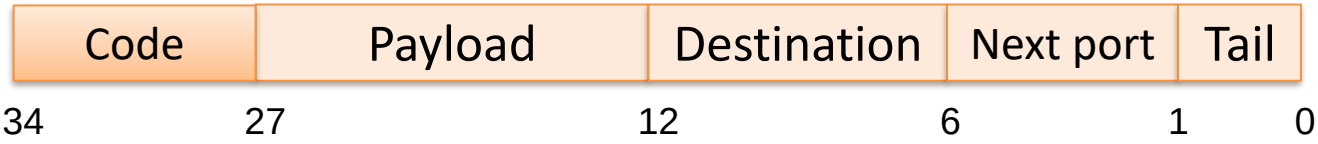

- Implemented 2x2 and 3x3 *OASIS router*
- Implemented ECC module and integrated into Router

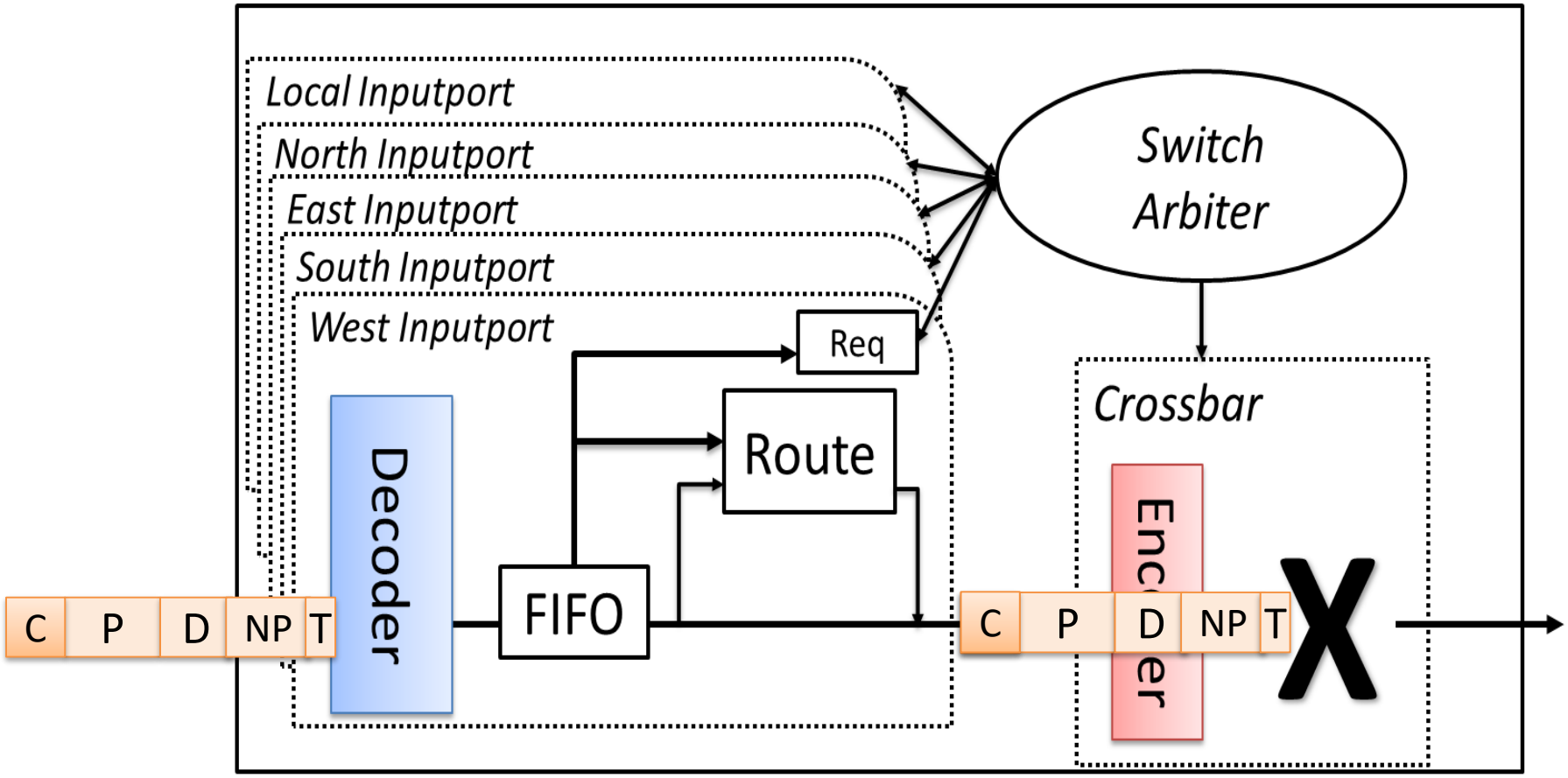
[Simulation]

- data transfer 100times with error-injection
- Verified function of ECC modules

# Implementation - Router

***Flit format***



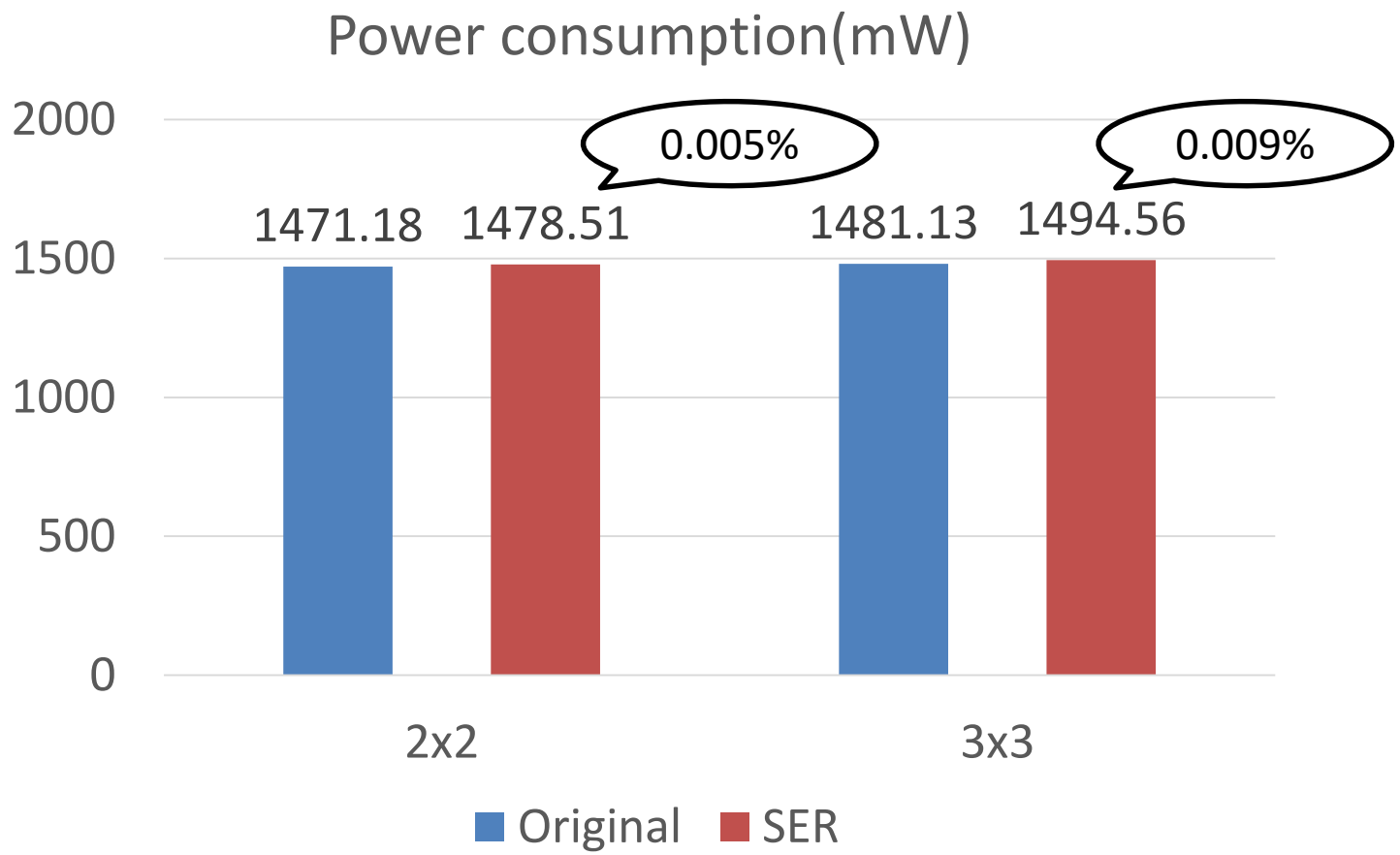
# Evaluation results (Hardware complexity)

- Design tools
  - Software Quartus II 13.0 sp1
  - Device: Stratix IV: EP4SGX530NF45C3

| | 2x2 | | 3x3 | |
|---|---|---|---|---|
| | Original | SER | Original | SER |
| Area - ALUTs (usage /424,960) | 1,748 (1%) | 2,881 (1%) | 5,790 (1%) | 8,913 (2%) |
| - Registers (usage /424,960) | 2,060 (1%) | 2,074 (1%) | 5,702 (1%) | 5,709 (1%) |

# Hardware complexity

Operation Frequency: 160MHz



# Conclusion

- Implemented *SER-OASIS NoC*
  - Organized 2x2 and 3x3 network size
  - Implemented ECC module and integrated
  - Evaluated hardware complexity
- Achieved error protection mechanism with small power and area overhead
  - Recover from 1-bit error and detect 2-bit error
  - Small area and power overhead

# Future work

- Implement retransmission mechanism for double error detection case
  - Implement Auto-repeat-request(ARQ) module and modify flow control
- Achieve *soft-error-resilient-algorithm(SERA)* that can resolve error in logical corruption
  - Additional pipeline for redundant calculation
  - Implement compare and rollback module


***Thank you for your attention!***