Neuromorphic Computing

6. Fault-Tolerant Neuromorphic System Design (Part – I and II)

Ben Abdallah Abderazek, Khanh N. Dang
E-mail: {benab, khanh}@u-aizu.ac.jp

This lecture is based on the book ''Neuromorphic Computing Principles and Organization,'' Publisher: Springer; 2022 edition, ISBN-10 : 3030925242, ISBN-13 : 978-3030925246, by Abderazek Ben Abdallah, Khanh N. Dang

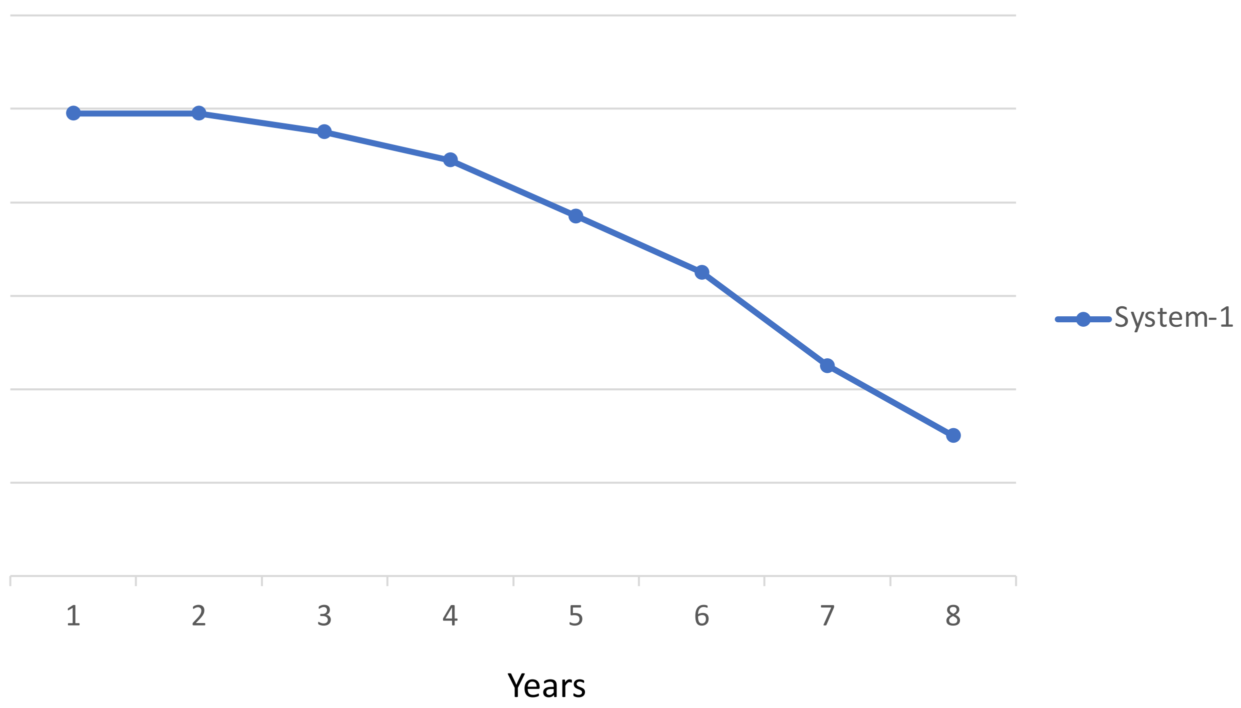
Lecture Contents

1. Introduction

- When manufacturing Integrated Circuits, “imperfection” may occur which lead to unwanted fabricated parts.
  - This inaccuracy in the design which lead to mistakes in the functionality which is “fault”.
- Another aspect is the wear-out or aging process:
  - Devices will be degraded over time.
  - Output of the gates can be erroneous
  - Disconnected wires may occur

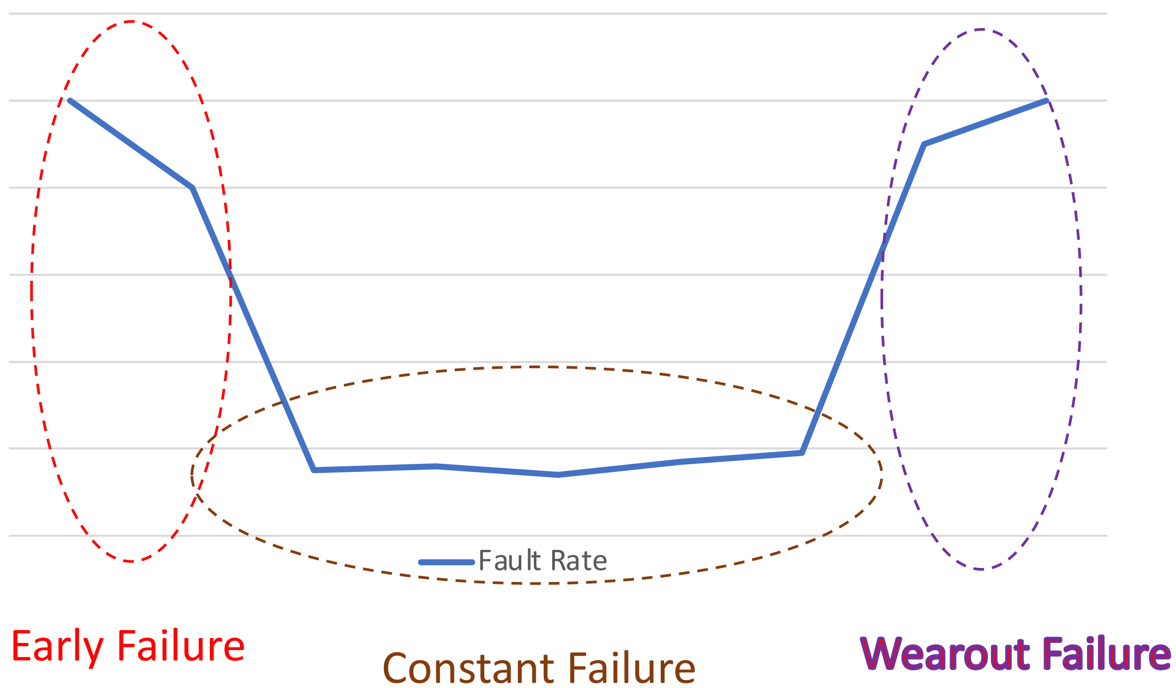
1. Introduction
Measure of Reliability

- Reliability can be expressed as $R(t)$: the probability of the system work normally.



1. Introduction
Measure of Reliability (cnt.)

- Reliability depends on the fault rate λ which may vary over time.
- Fault rate usually in bathub model



1. Introduction
Measure of Reliability (cnt.)

- Mean-time-to-Failure (MTTF):

$$\mathrm{MTTF} = \int_0^{\infty} \mathrm{R(t)dt}$$

R(t): the reliability of the system in the interval [0, t]

- MTTF represents the average time that the system work correctly ➔ Higher MTTF means more reliable system
- Availability (A(t)) is another measurement of the ratio of online time of the system (with faults and repair)
  - Example: In average, cars can run 1000 hours then repair for 2 hour (not running). MTTF is 1000 hours. Mean time to repair is 2 hours. The Avaibility is:

$$\frac{1000}{1000+2} = 0.998$$

1. Introduction
Faults in Semiconductor

- There are three types of faults:
  - **Permanent fault**: *fault occurs constantly and never return to be functional.*
    - *Example: disconnected wires, gate connected to ground wire*
  - **Intermittent fault**:*fault does not go away, but it usually oscillates.*
    - *Example: crack in wire that connect when a stress applied (thermal expansion can cause this stress)*
  - **Transient fault**: *fault causes a component malfunction some time and can go away after a short period.*
    - *Example: alpha particles flip the voltage of an output of a gate*

1. Introduction
Type of system

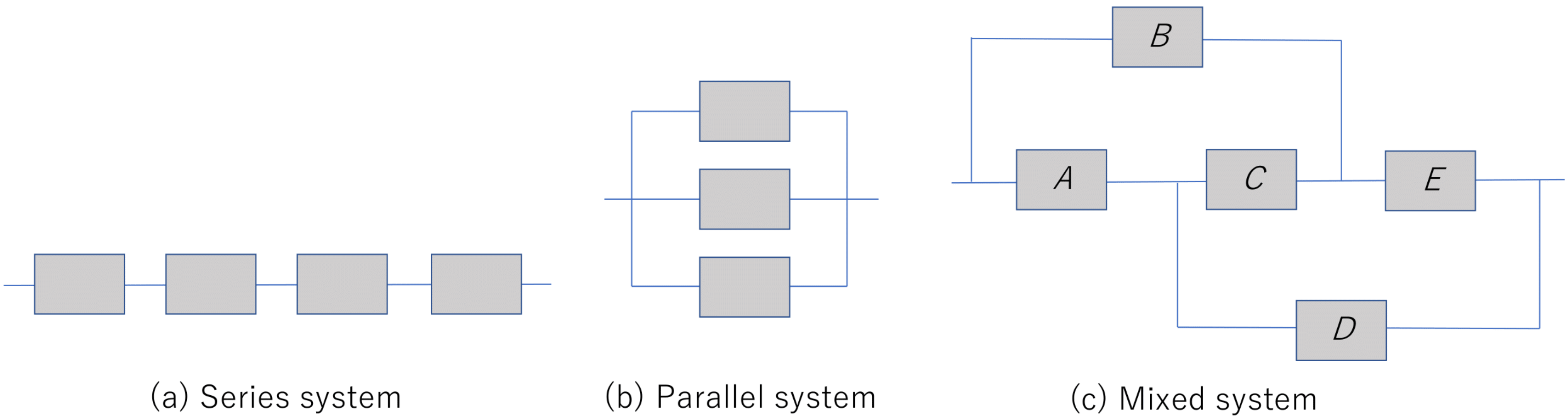


Fig. 6.2: System type.

- Any system can be classified into serial, parallel or mixed one
- In parallel system, parallel modules are exchangable which make the system failed once all module failed.
- In serial system, once a model failed, the whole system failed.

1. Introduction
Serial System

- If one of the modules fails, the whole system will malfunction.
- Assuming the modules fail independently, the fault rate of the system is the product of the fault rates of all modules:

$$R_{system}(t) = \prod_{all-modules} R_{module}(t)$$

Example of a serial system of four modules

Module-1	Module-2	Module-3	Module-4	System
0.9	0.95	0.85	0.99	0.7195

1. Introduction
Parallel System

- In the parallel system, the system only fails when all modules are failed
- Assuming the modules fail independently, the fault rate of the system is the product of the fault rates of all modules:

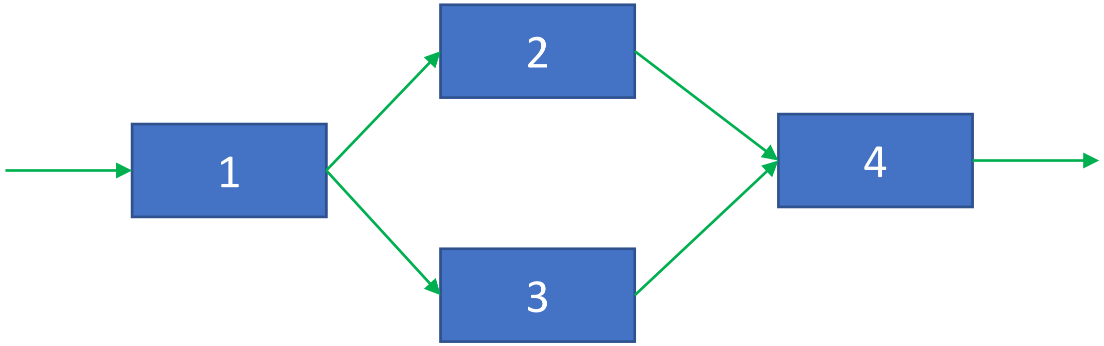
$$R_{system}(t) = 1 - \prod_{all-modules} (1 - R_{module}(t))$$

Example of a parallel system of four modules

Module-1	Module-2	Module-3	Module-4	System
0.9	0.95	0.85	0.99	0.9999925

1. Introduction
Mixed System

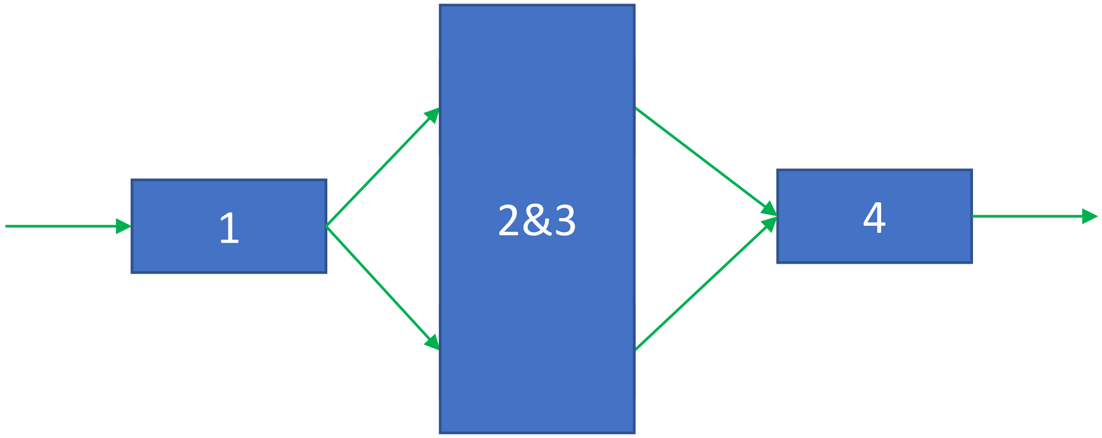
- For the mixed system, we can divide it into sub-systems to analyze.
- Example



Module-1	Module-2	Module-3	Module-4	System
0.9	0.95	0.85	0.99	?

1. Introduction
Mixed System (cnt)

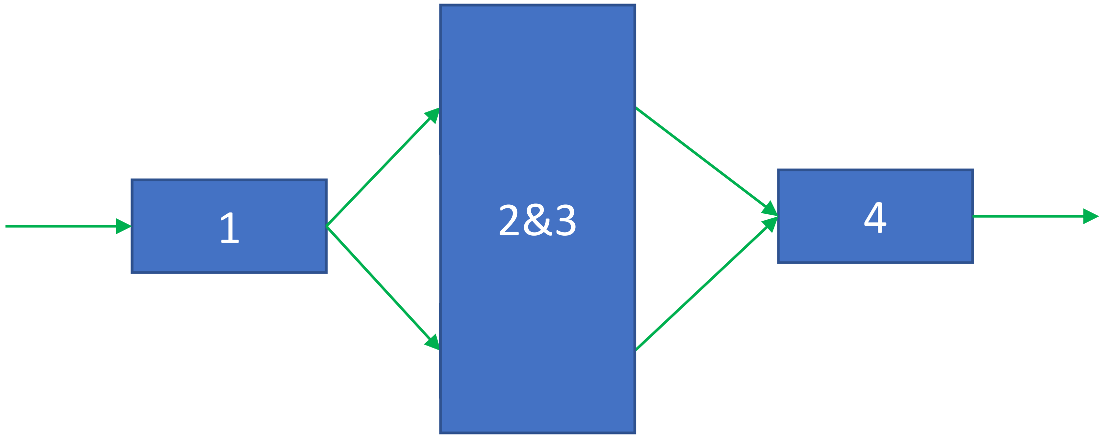
- First, let consider sub-system 2&3 which is a parallel system of module 2 and 3



Module-1	Module-2	Module-3	Sub-system 2&3	Module-4	System
0.9	0.95	0.85	0.9925	0.99	?

1. Introduction
Mixed System (cnt)

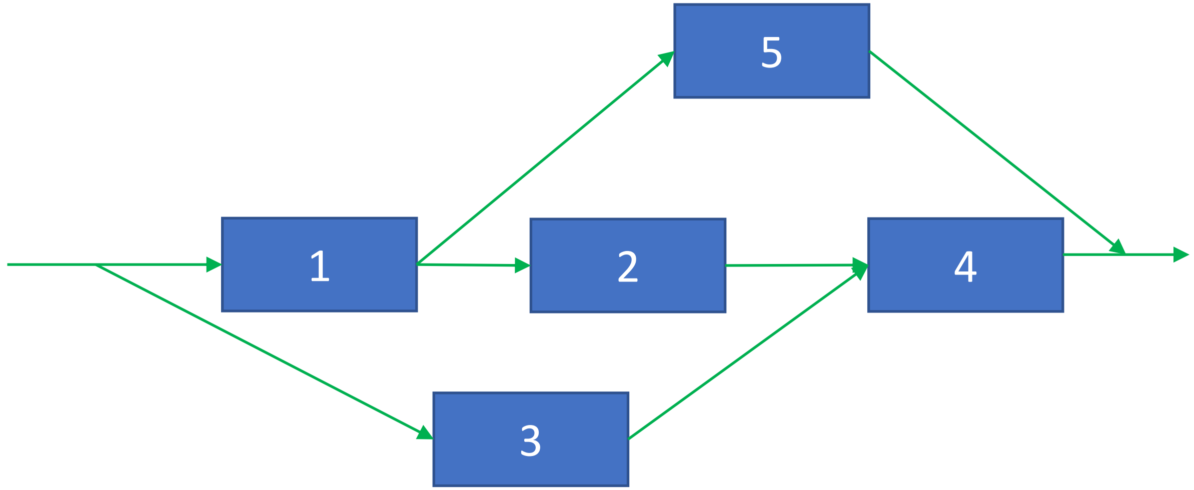
- Then, we can convert into a serial system of module-1, module 2&3, and module 4



Module-1	Module-2	Module-3	Sub-system 2&3	Module-4	System
0.9	0.95	0.85	0.9925	0.99	0.8842

1. Introduction
Mixed System (cnt.)

- However, sometimes the divide and conquer method is not feasible



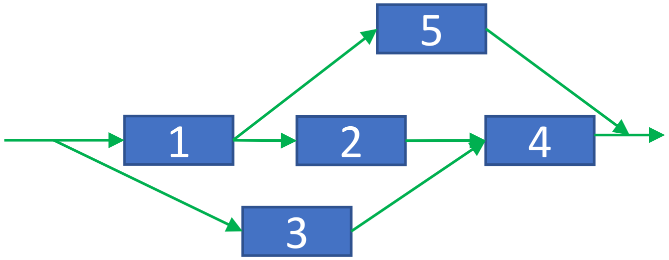
Module-1	Module-2	Module-3	Module-4	Module-5	System
0.9	0.95	0.85	0.99	0.8	?

1. Introduction
Markov State Model

- To analyze the mixed model, it is better to have the Markov state model. Each state represents one of scenario when one or more modules become failed.
- States are divided into:
  - Heathy state
  - Failure state
- Reliability = probability in the Heathy states; or
- Reliability = 1 - probability in the Failure states

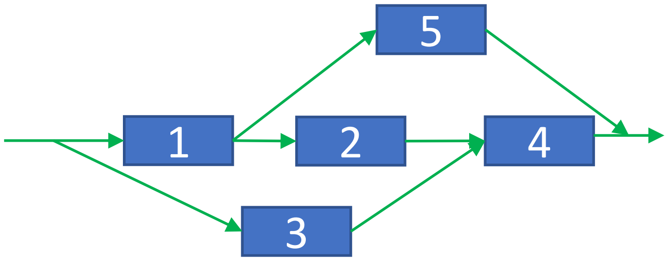
1. Introduction
Markov State Model

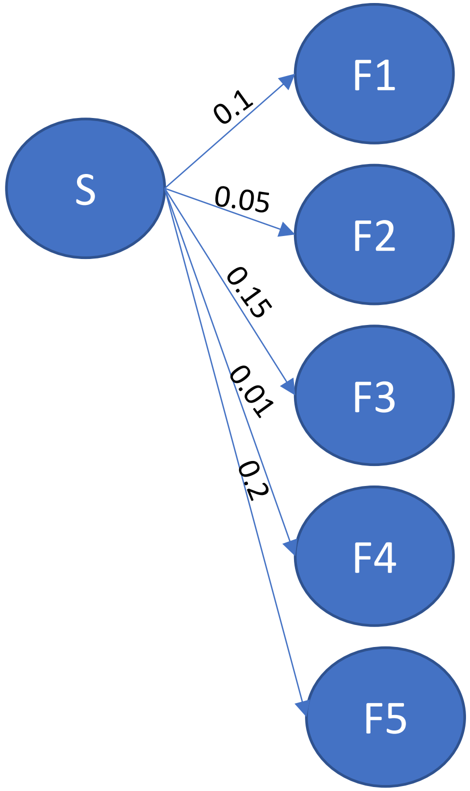
- Let's build a Markov satte model
- It is always start with initial state **S**
- **Probability for S start with 1.0 at the begining**



1. Introduction
Markov State Model

- Assuming a module is failed, we can have a new state



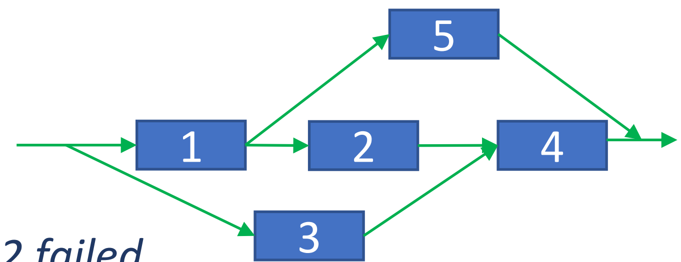


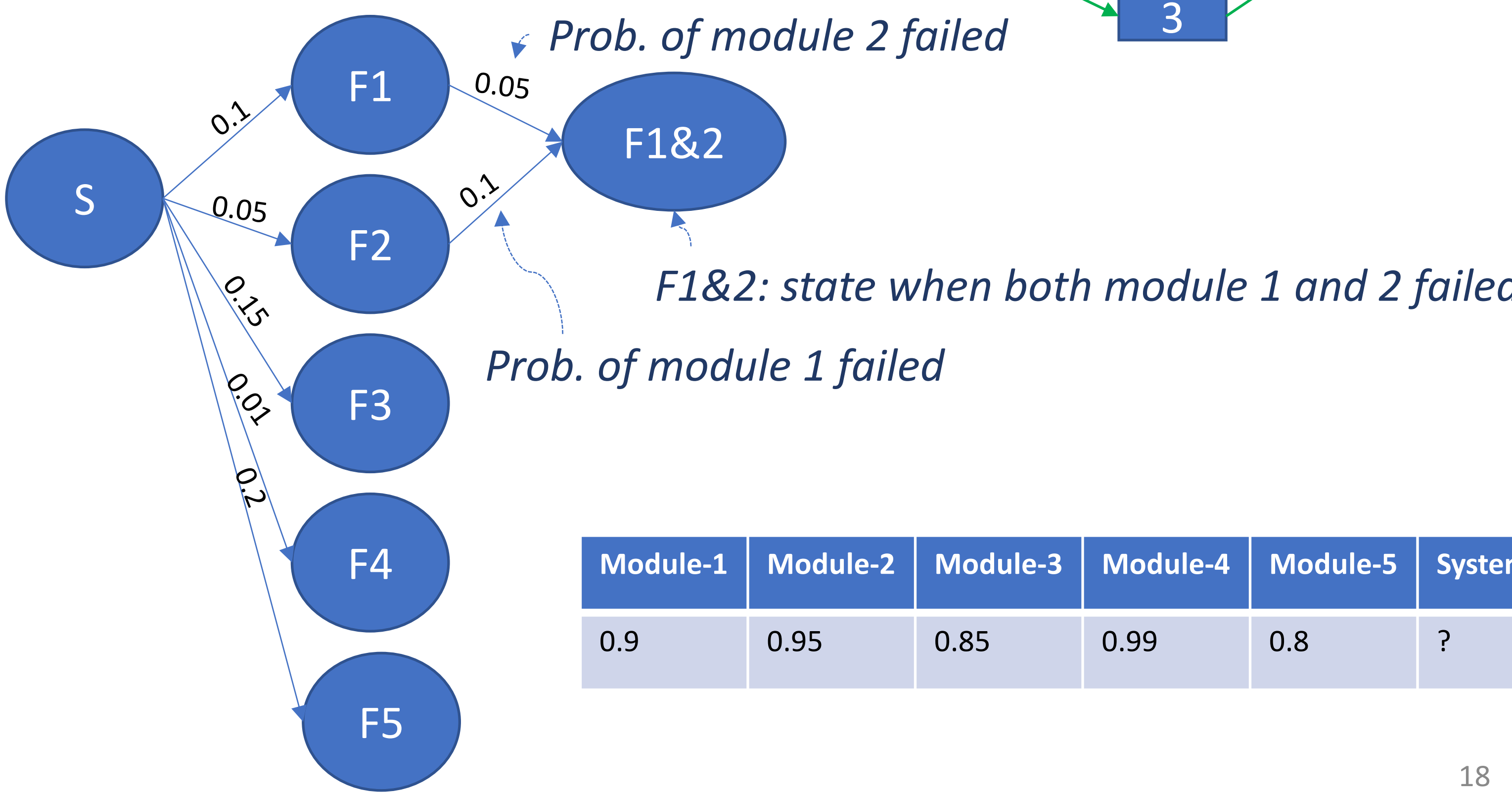
- Probablity to transfer for S to Fi (i = 1,2,3,4 or 5) is failure probability of module – i (or 1– reliability of module i).

Module-1	Module-2	Module-3	Module-4	Module-5	System
0.9	0.95	0.85	0.99	0.8	?

1. Introduction
Markov State Model

- Next, we add extra states for two or more modules failed

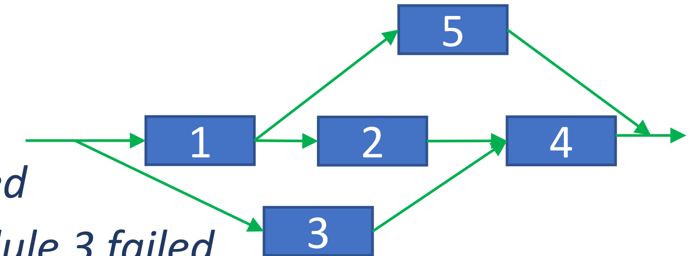


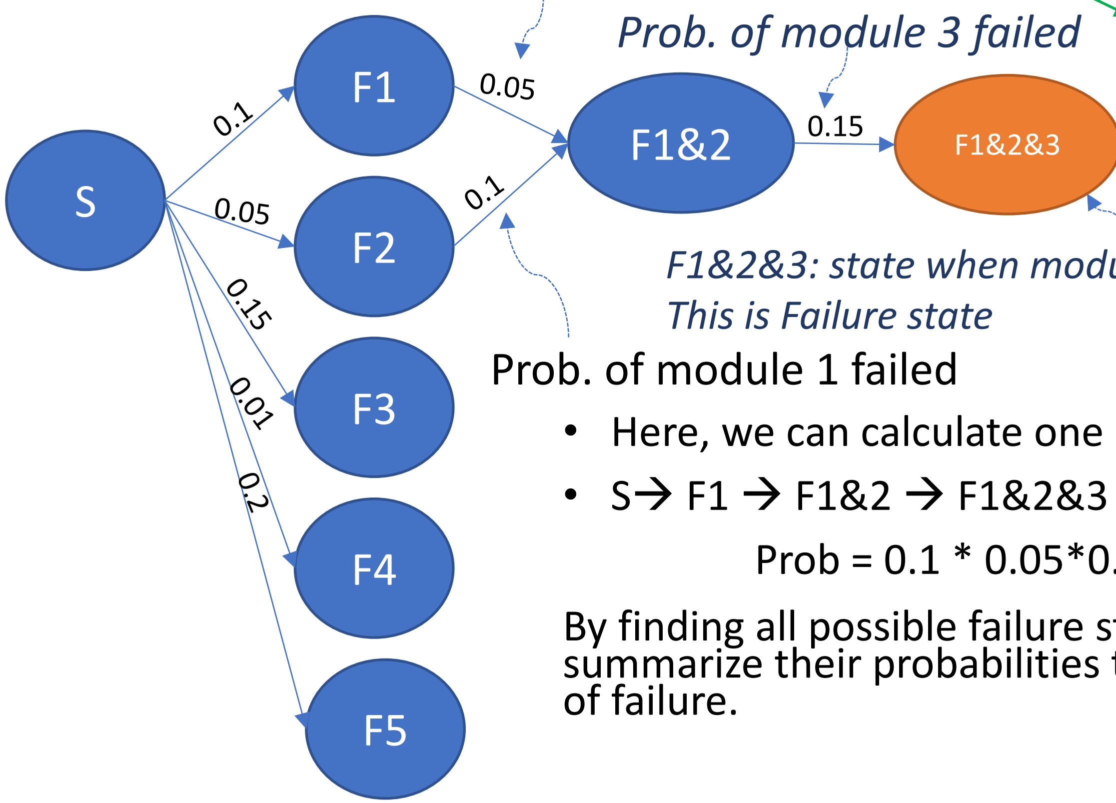


Module-1	Module-2	Module-3	Module-4	Module-5	System
0.9	0.95	0.85	0.99	0.8	?

1. Introduction
Markov State Model

- Next, we add extra states for two or more modules failed





- Here, we can calculate one branch of failures:
- S→ F1 → F1&2 → F1&2&3

  Prob = 0.1 * 0.05*0.15 = 0.00075

By finding all possible failure states, we can summarize their probabilities to have the probability of failure.

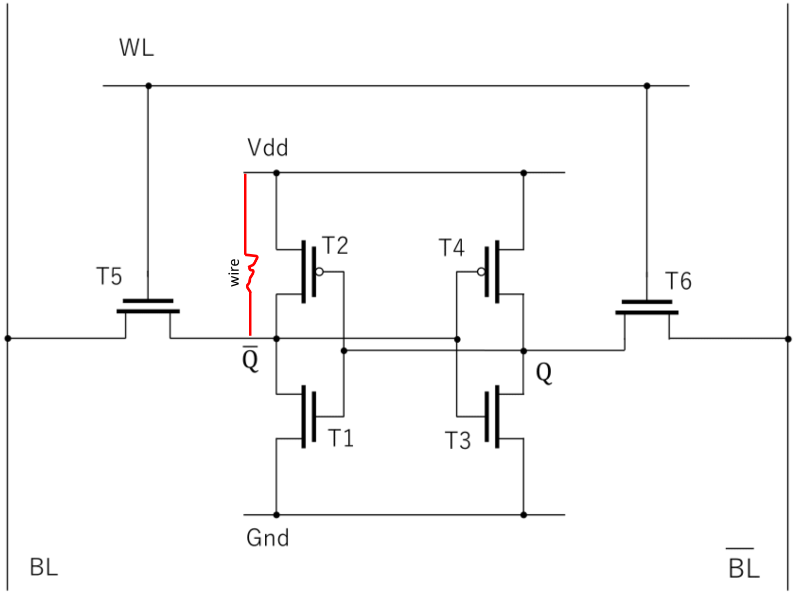
1. Introduction
Impact on SNN

Table 6.1 Taxonomy of faults: types, causes, behaviors, detection and recovery

Major part	Type	Causes	Behavior	Detection method	Recovery
Memory	Transient	Alpha particles/cosmic rays	Flip bit	Replicating and comparing/error detection code	Information redundancy
	Permanent/intermittent	Manufacture imperfection/aging/wear-out	Stuck-at, bridge	Testing algorithm	Spatial redundancy
Computing unit	Transient	Alpha particles/cosmic rays	Inaccurate output	Redundancy-based voting/multiple executions	Self-resilient/redundancy-based voting/multiple executions
	Permanent/intermittent	Manufacture imperfection/aging/wear-out	Stuck-at, bridge	Voting	Spatial redundancy
Communication infrastructure	Transient	Alpha particles/cosmic rays	Corrupted data, mis-routing	Error detection code/multiple executions	Error correction code/network re-routing
	Permanent/intermittent	Manufacture imperfection/aging/wear-out	Corrupted data, blocking connection	Error detection code	Spatial redundancy, fault-tolerant routing
Part of Neuromorphic systems	Fault Types	Causes of faults	Behaviors	How to detect	How to correct

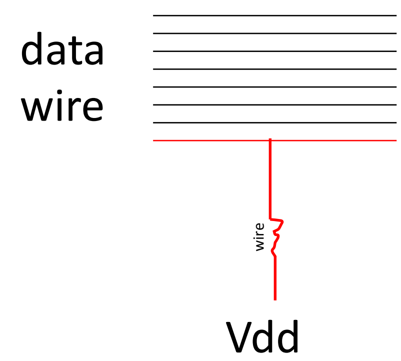
1. Introduction
Impact on SNN: Example

- Memory:
  - Permanent defect: stuck-at –zero
  - Value $\bar{Q}$ is shorten to Vdd which make it stay at `1`
  - Value Q is now stuck at 0
  - Value of memory cell is always 0
- Behavior:
  - If the 1st bit is stuck at 0
  - Writen weight: 11001010_2
  - Read weight: 01001010_2



1. Introduction
Impact on SNN: Example (cnt.)

- Computing Unit:
  - Permanent defect: stuck-at–one
  - An wire is shorten to Vdd
  - Value transimitting on the wire stay at 1
- Behavior:
  - If the 1st bit is stuck at 1
  - Transmitted value: 00001010_2
  - Read weight: 10001010_2



1. Introduction
Impact on SNN

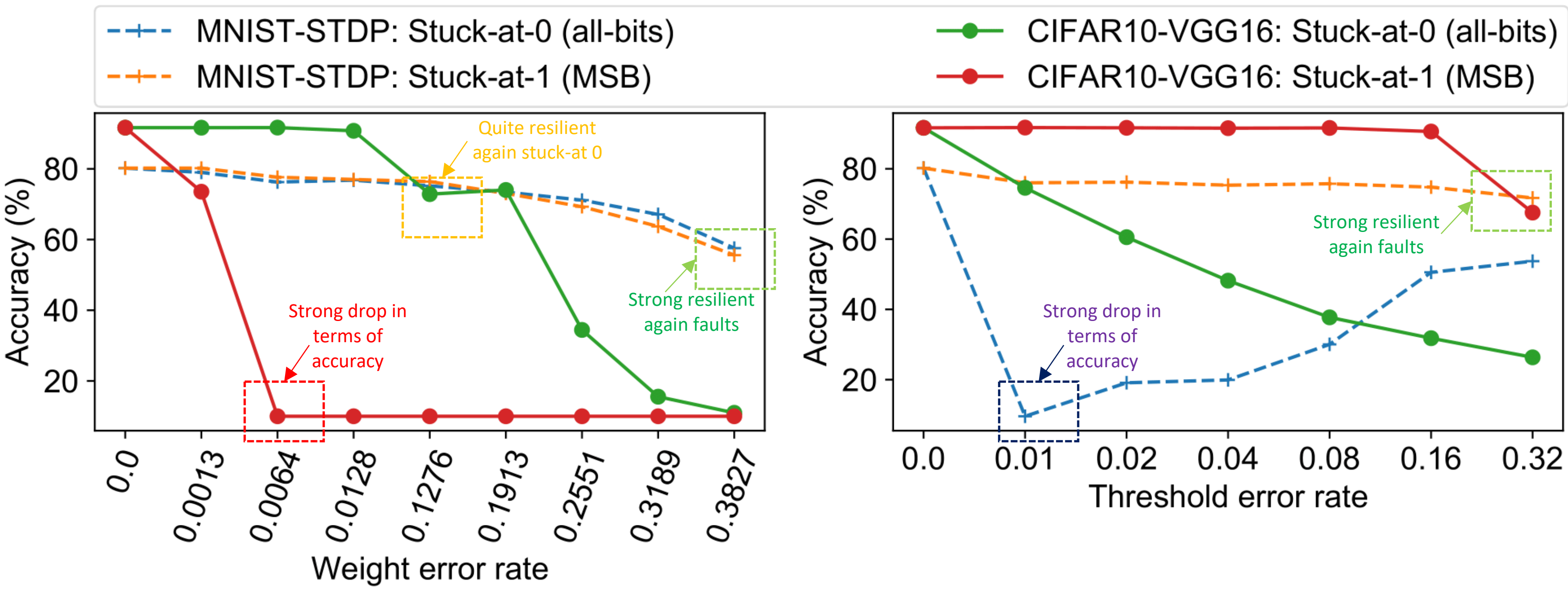


Fig. 6.1: Impact of faults on a neuromorphic system.

- Insert stuck-at-0/ stuck-at-1 into weight or threshold
- Two benchmarks: MNIST/CIFAR10

➔ SNN provides some resilient again the faults; however, under high defect rates, it starts to collapsed.

Lecture Contents

1. Introduction
2. Conventional Computing System Fault Tolerance
3. Fault Tolerance for Neuromorphic Computing
    1. Memory Protection
    2. Communication Protection
    3. Computation Protection
4. Mapping for Tolerating Faults in Neuromorphic Computing
5. Conclusions

2. Conventional Fault Tolerance
Overview

- In conventional computing system fault tolerance, we can classify them into three main approach
  - **Hardware approach**: adding extra hardware to correct the faulty modules
  - **Information redundancy**: adding extra information to correct the faulty information
  - **Software approach**: dealing with fault by using software

2. Conventional Fault Tolerance Hardware

- The most common approach in hardware fault tolerance is to add extra modules which work in parallel
- Multiple modules work in parallel can help detect the defect
- Once a module failed, we can use the added module as the replacement

2. Conventional Fault Tolerance
Hardware (cnt)

- Triple Modular Redundancy:
  - Replicate the module to have 2 extra copy.
  - Voter is added to decide the healthy status of the modules.
  - If one module has different result, that module is considered as failed.

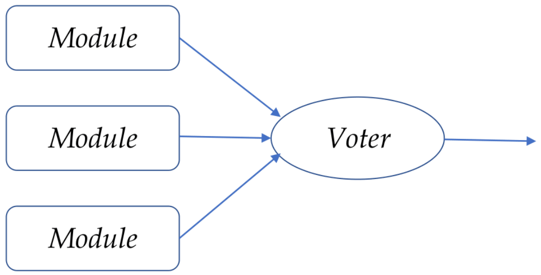


Fig. 6.3: Triple Modular Redundancy.

2. Conventional Fault Tolerance
Information Redundancy

- By adding extra information, we can prevent the data corruption.
- For instance (triple repetition code):
  - instead of storing value “a”, we stored three consecutive values of “a” as “aaa”.
  - If the second value is corrupted a → b, the three consectutive value become “aba”
  - By comparing three value “a”, “b”, and “a”, we can conclude that the correct value is “a”

2. Conventional Fault Tolerance
Parity Code

- Parity code is a systematic code with only one extra bit *P*
- *P* is *1* if the number of bit 1 in the data is odd
- *P* is 0 if the number of bit 1 in the data is even
- Example:
  - data: 1010 → P=0
  - data: 0111 → P=1

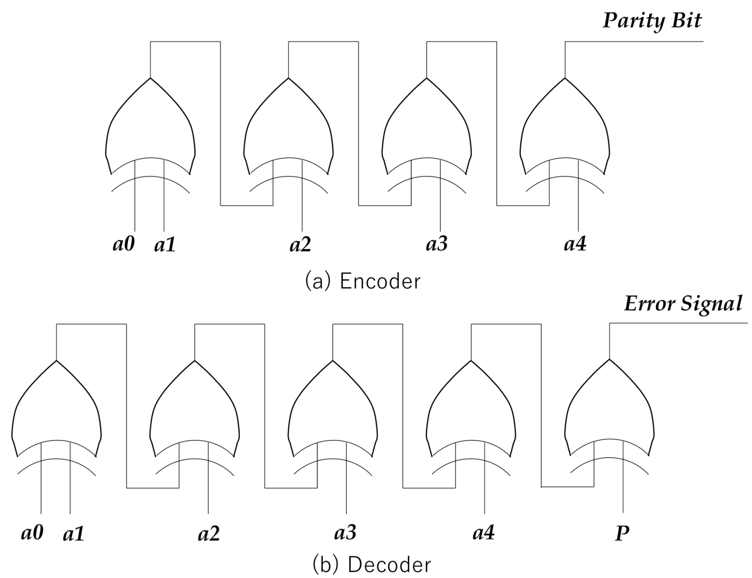


Fig. 6.5: Even parity code: (a) encoder; (b) decoder.

2. Conventional Fault Tolerance
Parity Code (cnt)

- Parity code can detect one bit error; but it cannot correct.
- Detection by counting the bit-1 in data:
  - If the number of bit 1 is even (in even parity codeword): the data is correct
  - If the number of bit 2 is odd (in even parity codeword): the data is incorrect
- For example:
  - Data : 0 1 0 1 0 1 1 0
  - Even parity codeword : 0 1 0 1 0 1 1 0 0
  - If one bit is error, we can detect:
    - 1st bit error: 1 1 0 1 0 1 1 0 0: check parity of this array
      - # bit – 1: 5 → the data is incorrect
    - 4th bit error: 0 1 0 0 0 1 1 0 0
      - # bit – 1: 3 → the data is incorrect
- In logic (and LSI), XOR function (⨁) is used for parity check.

Input		Output
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

2. Conventional Fault Tolerance
Hamming Code

- Hamming code provides the ability to correct one error bit
- An extension of Hamming code (SECDED) can correct one error bit and detect two error bits

Table 6.2 Parity bit combination for Hamming code

Data bit position		1	2	3	4	5	6	7	8	9	10	11
Parity bit	1	x	x		x	x		x		x		x
(Hamming)	2	x		x	x		x	x			x	x
	3		x	x	x				x	x	x	x
	4					x	x	x	x	x	x	x
Extended parity bit (SECDED)		x	x	x	x	x	x	x	x	x	x	x

2. Conventional Fault Tolerance
Hamming Code (cnt.)

- Assuming the data is "0110".
- The parity bit is:
  - $p1 = b1 \oplus b2 \oplus b4 = 1$
  - $p2 = b1 \oplus b3 \oplus b4 = 1$
  - $p3 = b2 \oplus b3 \oplus b4 = 0$
- Codeword: "0110110"

	data	p1	p2	p3
b1	0	x	x	
b2	1	x		x
b3	1		x	x
b4	0	x	x	x
		1	1	0

2. Conventional Fault Tolerance
Hamming Code (cnt.)

- Let's call codeword' is the value of the codeword after transmission/loading. codeword' can be different from codeword.
- To simplify the correctability, we can understand one bit can have three versions:
  - Version 1: the bit in the codeword' (received codeword)
  - Version 2& 3: the bit extract from parity bit and other bit
- For example, b1 bit can be obtained from:
  - b1' in the codeword'
  - $p1 = b1 \oplus b2 \oplus b4 \rightarrow b1'' = p1' \oplus b2' \oplus b4'$
  - $p2 = b1 \oplus b3 \oplus b4 \rightarrow b1''' = p2' \oplus b3' \oplus b4'$
- From three version: b1', b1'', b1''', we can decide the correct value of b1

2. Conventional Fault Tolerance

Hamming Code (cnt.)

codeword	0	1	1	0	1	1	0
codeword’	1	1	1	0	1	1	0
codeword’’	0						
codeword’’’	0						
Corrected codeword	0						

- Let’s flip bit b1 in the codeword’ 0→1
  - Ver1: 1
  - Ver2: 0
  - Ver3: 0

Corrected b1 = 0

	p1	p2	p3
b1	x	x	
b2	x		x
b3		x	x
b4	x	x	x

2. Conventional Fault Tolerance

Hamming Code (cnt.)

codeword	0	1	1	0	1	1	0
codeword’	1	1	1	0	1	1	0
codeword’’	0	0					
codeword’’’	0	1					
Corrected codeword	0	1					

- Let’s check another bit: b2
  - Ver1: 1
  - Ver2: 0
  - Ver3: 1

Corrected b2 = 1

	p1	p2	p3
b1	x	x	
b2	x		x
b3		x	x
b4	x	x	x

2. Conventional Fault Tolerance
Hamming Code (cnt.)

codeword	0	1	1	0	1	1	0
codeword’	0	1	1	0	0	1	0
codeword’’	0	0					
codeword’’’	0	1					
Corrected codeword	0	1					

- Let’s flip bit p1, we can see it only affect b1,b2 and b4. For instance we can see b2:
  - Ver1: 1
  - Ver2: 0
  - Ver3: 1

Corrected b2 = 1

	p1	p2	p3
b1	x	x	
b2	x		x
b3		x	x
b4	x	x	x

2. Conventional Fault Tolerance
Software Fault Tolerance

- Another approach is software fault tolerance:
  - ***Algorithm-based fault tolerance*** where the computation includes the correction method itself.
    - The computing algorithm has built-in fault-tolerance feature instead of realizing on hardware.
  - **Check-pointing** and ***rolling-back*** is another approach where checkpoint of the system is saved for rolling back when an error occurs.
    - This mostly works against soft errors the the error could disappear when the system rolls back.

Lecture Contents

1. Introduction
2. Conventional Computing System Fault Tolerance
3. Fault Tolerance for Neuromorphic Computing
    1. Memory Protection
    2. Communication Protection
    3. Computation Protection
4. Mapping for Tolerating Faults in Neuromorphic Computing
5. Conclusions

3. Fault Tolerance for Neuromorphic Computing
Overviews

- To protect the neuromorphic computing system, we can divide them into three major parts:
  - **Computing**: for all the modules that compute the neural network tasks
  - **Memory**: for storing and load the memory where the defect can cause incorrect reading, writing or storing
  - **Communication**: for transferring the message between modules which can lead to data corruption

3. Fault Tolerance for Neuromorphic Computing
Memory Protection

- To protect the information stored in the memory, we can use the ***information redundancy approach:***
  - By adding extra information to allow the system detect and correct potential error bits. For example: Hamming, SECDED (Hamming with one extra bit), …
  - Accepting the error bits with potential graceful acurracy loss by considering neuromorphic computing as an approximate computing method.
- ***Software approach*** can also be used for soft errors:
  - Once we detect error bit, we re-write the system's memory with the safe copy

3. Fault Tolerance for Neuromorphic Computing
Communication Protection

- To protect the communication, we first need to divide the errors into two types*:*
  - First one is the error in the transmitting data where the routing and handshaking is considered as corrected. Here, we can convert the problem into the memory protection problem.
  - The second one is the error in the routing or handshaking processes. With this type of error:
    - Finding the alternative routing path to avoid defective ones is necessary.
    - Retranmission with package dropping could be used since misrouted package could lead to deadlock/livelock.
    - Protection and recovery can be deal using hardware approach (adding redundancies).

3. Fault Tolerance for Neuromorphic Computing
Computation Protection

- To protect the computation errors*:*
  - The first approach is to ***accept the errors*** by considering them as noise. Furthermore, ***adjusting parameter*** (i.e., threshold voltage with input losses) can be used to alleviate the impact.
  - Another apporach is to use ***hardware redundancy*** *by* replicating the computing units and consider as replacements.
    - Redundancy can be at fine grained level.
    - Since computing units are identical, adding extra ones and perform system remapping is also possible.

Lecture Contents

1. Introduction
2. Conventional Computing System Fault Tolerance
3. Fault Tolerance for Neuromorphic Computing
    1. Memory Protection
    2. Communication Protection
    3. Computation Protection
4. Mapping for Tolerating Faults in Neuromorphic Computing
5. Conclusions

4. Mapping for Tolerating Faults for NC
Problem Formulation

- Assuming the neuromorphic system S has N nodes (neuron clusters) connected via Network-on-Chip.
- Each node i has E_i neurons (could be different between nodes).
- Total number of neurons:

$$X = \sum_{i=0}^{N-1} E_i$$

Reference: Khanh N. Dang, Nguyen Anh Vu Doan, Abderazek Ben Abdallah *"MigSpike: A Migration Based Algorithm and Architecture for Scalable Robust Neuromorphic Systems"*, **IEEE Transactions on Emerging Topics in Computing (TETC)**, IEEE, Volume 10, Issue 2, pp. 602-617, 2022.

4. Mapping for Tolerating Faults for NC
Problem Formulation (cnt.)

- Here, we assume the SNN application need W neurons to work ($W \leq X$).
- The number of spare neurons $\mathrm{R} = X - W$ which is the redundancy for correcting potential defects
- Once a defect in the neuron occurs, a spare neuron can be used for correction

4. Mapping for Tolerating Faults for NC

Correcting method

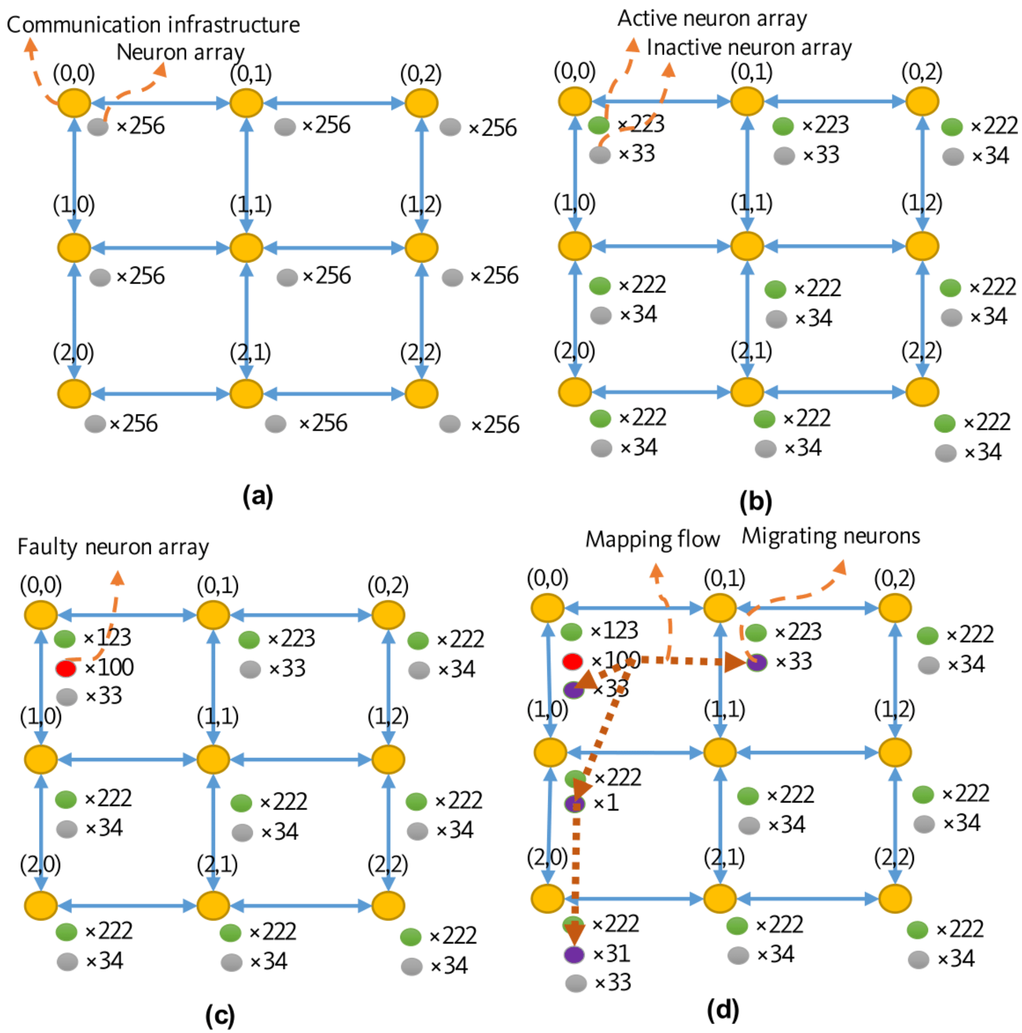


Fig. 6.6: System model for fault tolerance SNN: (a) Designed SNN system using nodes of neurons with an initial mapping; (b) node-level recovery; (c) The case node-level recovery fails to correct; (d) System level recovery: a mapping flow of 100 faulty neurons to its node's neighbors. Values next the circle indicate the number of neurons in the circle type (gray: healthy and utilized; gray: healthy and spared; red: faulty; purple: migrating).

4. Mapping for Tolerating Faults for NC
Problem Conversion

- Here, we convert the problem into the graph theory problem (max flow min cut).
- We create the graph of N+2 vertices:
  - N vertices for N nodes
  - 2 vertices for virtual source and virtual sink.
- Edges are added:
  - From the virtual source to each node with defects; capacity = number of defects
  - From each node with spares to the virtual sink ; capacity = number of spares
  - Between neighboring nodes; capacity = number of neurons can be migrated from one node to other node

4. Mapping for Tolerating Faults for NC

Generated Graph and Potential Solutions

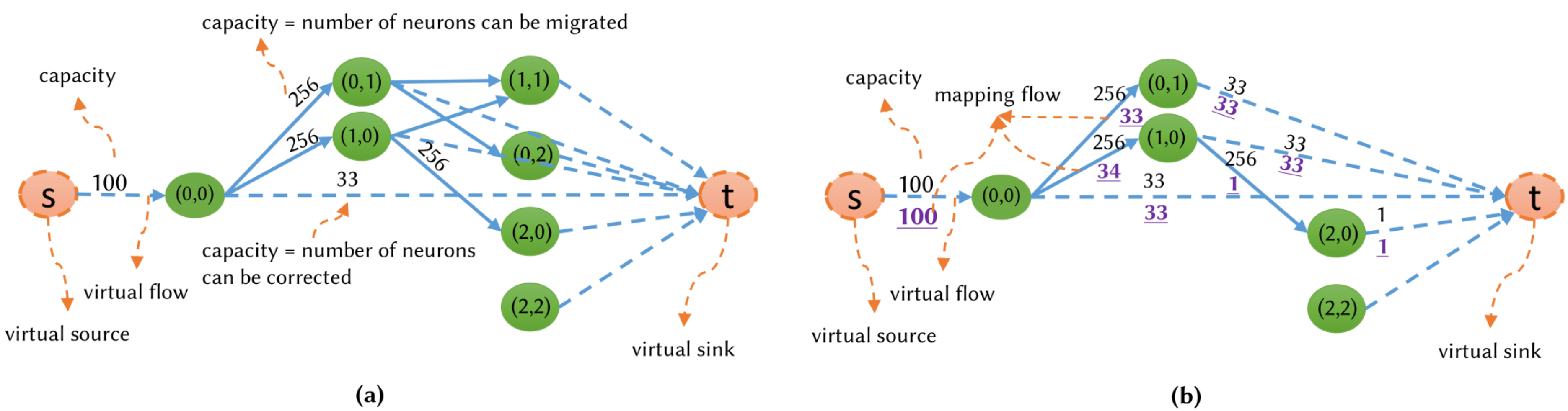
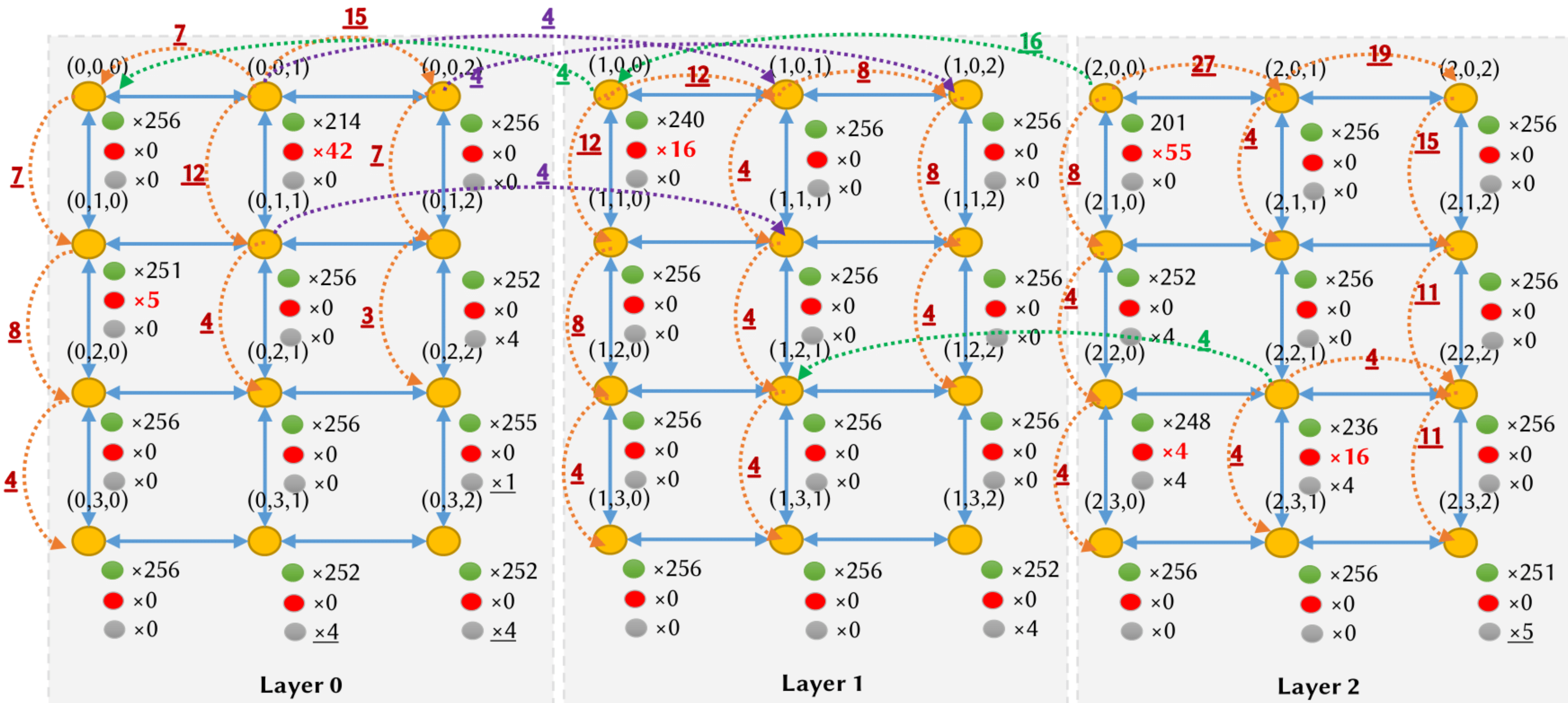


Fig. 6.7: Flow graph for max-flow min-cut problem: (a) Converted flow from the NoC-based SNN; (b) A solution of max-flow min-cut problem

4. Mapping for Tolerating Faults for NC

Input and Potential Solution



(b)

Fig. 6.8: An illustration of the proposed algorithm: (a) Faulty case; (b) Post-mapping using the proposed algorithm.

Lecture Contents

5. Conclusion

- In this lecture, we first review reliability measurement and defective types.
- Impact of faults into neuromorphic system is analyzed.
- Traditional approaches for fault-tolerance are reviewed with the potential usage in neuromorphic system
- Case study: remapping to tolerate defective neurons in large scale system