Fault Coverage: An Important Parameter for Dynamic PSA of Digitalized Safety Systems

Hyun Gook Kang, Man Cheol Kim, Seung-Cheol Jang
Integrated Risk Assessment Center
Korea Atomic Energy Research Institute
hgkang@kaeri.re.kr
1. Introduction

- Three important issues in a digital system PSA
  - Fault coverage
  - Digital System PSA
  - Common cause failure
  - Software reliability
1. Introduction

- Example of Fault Tolerance Mechanism: WDT

![A typical design of watchdog timer]

![Example of application]
1. Introduction

- Fault coverage
  - Mathematical definition: the conditional probability that a fault in a system is properly processed, provided that a fault exists in a system
    \[ C = \frac{\text{fault processed correctly}}{\text{fault existence}} \]
  - Qualitative definition: the ability of detecting, isolating, and recovering a fault in a system

< An example axiomatic model of fault coverage (Dugan and Trivedi, 1989) >
2. Experiments

- Hybrid method for the quantification of fault coverage of digital systems in NPPs

![Diagram showing the quantification of fault coverage](image)

**Simulation Process**

1. **Simulation**: Simulation of the system components
2. **Central Processing Unit (CPU)**
   - Control Unit (CU)
   - Arithmetic Logical Unit (ALU)
   - Registers
3. **RAM**
4. **Input/Output**
5. **ROM**
6. **Top Parameters**
   - Trip Signal
   - Heartbeat signal
   - Error Signal
7. **Top Signal**

**Quantification of fault coverage**

- CPU Fault
- RAM Fault
- ROM Fault
- I/O Fault

![Chart showing the quantification of fault coverage](image)
2. Experiments

- Summary of the experiment
  - Target system: Simplified CPU circuit
  - Experiment method: Simulated fault injection
  - Applied fault detection method:
    - watchdog timer, register write and read, checksum, parity bit

Fault detection and analysis

Fault injection

Hardware description
2. Experiments

- **Hardware description**
  - Simplified with 4 components
    - CPU (8051)
      - ALU & CU
    - RAM (64KByte)
    - ROM (64KByte)
    - I/O (4 channels)

- Simulation: C++ is used to describe the hardware
2. Experiments

- **Fault injection**
  - **Hardware fault injection**
    - Faults are injected into the hardware by physical or non-physical contacts.
  - **Simulated fault injection**
    - Faults are injected by modifying the software used to describe the hardware.
2. Experiments

- Register write and read
  - Algorithm
    - Reset all bits to 0
    - Write (Input) data 0x55(01010101) to an register
    - Read the register data and compare it to the input data.
    - Write (Input) data 0xAA(10101010) to the register
    - Read the register data and compare it to the input data.
  - If the data read from the register is not equal to the input data, faults can be detected.

<table>
<thead>
<tr>
<th>Input data</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>D7</td>
<td>D6</td>
<td>D5</td>
<td>D4</td>
<td>D3</td>
<td>D2</td>
<td>D1</td>
<td>D0</td>
</tr>
</tbody>
</table>

Input data | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
Register   | D7| D6| D5| D4| D3| D2| D1| D0|
2. Experiments

- **Parity bit**
  - An extra bit is used to check the validity and completeness of the data.
  - Computers use either an odd-parity check or an even-parity check.
  - A simple decoding circuit, using a set of XOR gates, will detect any single bit error in a parity-coded word.
  - **Even parity**: The total number of “1(on)” bits in a byte must be an even number.
  - **Odd parity**: The total number of “1(on)” bits in a byte must be an odd number.
2. Experiments

- Checksum
  - The sum of a group of data items, which sum is used for checking purposes.
    - Add up all the data in the memory and write the result at the end of the data.
    - Recalculate the checksum and compare it with the original one.
    - If any of the data, including the checksum is corrupted, a mismatch will be the result.
2. Experiments

- RAM data verification
  - It is necessary to confirm that a value placed on the data bus by the processor is correctly received by the memory device at the other end.
  - The most obvious way is to write all the possible data values and verify that the memory device stores each one. Inefficient!
  - Testing one bit at a time is an efficient method. A single data bit is set to 1 and marched along the entire data word.
    - Marching or Walking 1’s test.

![Consecutive data values for the test>

<table>
<thead>
<tr>
<th></th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
<th>hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0x01</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0x02</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0x04</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0x08</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0x10</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0x20</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0x40</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0x80</td>
</tr>
</tbody>
</table>
2. Experiments

- **Simple Watchdog Timer**
  - The heartbeats are periodically updated as long as the processor continues to execute its internal programming functions.
  - The heartbeat count number is sent to the proper digital output module point and it is monitored by watchdog timers.
  - An error can be detected when a heartbeat count number is not equal to the preset value in the watchdog timer.

![A typical watchdog setup]

Watchdog Timer  

Processor

Clock

Reset

Restart

<A typical watchdog setup>
2. Experiments

- **Experiment**
  - **Experimental parameters**

<table>
<thead>
<tr>
<th>Fault type</th>
<th>Permanent fault</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workload</td>
<td>2 out of 4 coincidence logic + Fault detection algorithms</td>
</tr>
<tr>
<td>Fault model</td>
<td>stuck-at (0, 1) fault</td>
</tr>
<tr>
<td>Fault Location</td>
<td></td>
</tr>
<tr>
<td>CPU</td>
<td>336</td>
</tr>
<tr>
<td>RAM</td>
<td>1,050,608</td>
</tr>
<tr>
<td>ROM</td>
<td>1,048,576</td>
</tr>
<tr>
<td>I/O</td>
<td>64</td>
</tr>
<tr>
<td>Fault detection methods</td>
<td>Heartbeat-Watchdog Timer, ROM Checksum, RAM data verification, Register write and read, Parity bit, Integrated error detection method (Heartbeat-Watchdog Timer + ROM Checksum + RAM data verification)</td>
</tr>
<tr>
<td>Result analysis</td>
<td>Percentage of activated errors, fault detection coverage</td>
</tr>
</tbody>
</table>
2. Experiments

- Fault Propagation: Fault, Error, Failure

  - **Fault**
    - Physical deficiency, incompleteness
    - Wrong instructions

  - **Error**
    - Inclusion of wrong data
    - Cannot perform intended functions

  - **Failure**
    - Provide wrong output
2. Experiments

- Percentage of activated errors of a component ($P_{A,\text{comp}}$)
  \[
  P_{A,\text{comp}} = \frac{N_{\text{activated,comp}}}{N_{\text{injected,comp}}}
  \]
  - $N_{\text{injected,comp}}$: the number of injected faults into a component.
  - $N_{\text{activated,comp}}$: the number of activated faults in a component.

- Percentage of activated errors of a system ($P_{A,\text{sys}}$)
  \[
  P_{A,\text{sys}} = \sum W_{\text{comp}} \cdot P_{A,\text{comp}} \quad \text{where,} \quad W_{\text{comp}} = \frac{\lambda_{\text{comp}}}{\lambda_{\text{total}}}
  \]
  - $\lambda_{\text{comp}}$: failure rate of a component (failures/10^6 Hours).
  - $\lambda_{\text{total}}$: failure rate of all the components in a system (failures/10^6 Hours).
  - Failure rates are calculated by using the MIL-HDBK-217F.

- Fault detection coverage of a component ($C_{d,\text{comp}}$)
  \[
  C_{d,\text{comp}} = \frac{N_{\text{detected}}}{N_{\text{activated,comp}}}
  \]
  - $N_{\text{detected}}$: the number of detected faults.

- Fault detection coverage of a system ($C_{d,\text{sys}}$)
  \[
  C_{d,\text{sys}} = \sum W_{\text{comp}} \cdot C_{d,\text{comp}}
  \]
2. Experiments

- Experimental results
  - Percentage of an error activation
2. Experiments

- Fault detection coverage

Components

System
3. Other Experiments

- KAERI is also performing the hardware-based fault injection experiment
  - Using ARM processor
  - Interim results: error activation and detection percentages are quite similar to the result of KAIST-KAERI software-simulation-based experiment
- KAERI is also planning to perform the experiment using the newly developed safety-critical digital PLC modules in which software of a plant protection system is installed.
3. Other Experiments

Investigation of SW-HW structure
3. Other Experiments

- Failure Mechanism Identification
- Fault Injection Profile Determination
3. Other Experiments

- Hardware-based Fault Injection Environment

Circuit Analysis & Fault Injection

Digital Board

JTAG

KNICS CPU Module

General Digital Board

Simulation using Debugging tools

Emulator

Simulation using Dedicated Emulator and Program
4. Conclusion

- Input data quality is one of the most important factors of the PSA model.
  - Dynamic PSA methodologies will increase the importance of realistic estimation of the fault coverage of digitalized system.

- Software-simulation-based experiment using fault injection method
  - Component checking algorithms effectively increases the fault coverage
  - By using these algorithms and the WDT, the 54% of activated faults could be detected.

- KAERI is performing the hardware-based experiments
  - More realistic results
  - Newly developed PLC processor module

- Fault coverage largely depends on the installed software
  - The effect of software in the presence of fault on the system behavior should be carefully investigated.
  - In addition to the hardware fault detection, the possibility of software fault detection should also be investigated.