


Karlsruhe Institute of Technology

# Reliable Computing I

## Lecture 9: Concurrent Error Detection

Instructor: Mehdi Tahoori


INSTITUTE OF COMPUTER ENGINEERING (ITEC) – CHAIR FOR DEPENDABLE NANO COMPUTING (CDNC)



KIT – University of the State of Baden-Wuerttemberg and  
National Research Center of the Helmholtz Association

[www.kit.edu](http://www.kit.edu)

## Today's Lecture



- Concurrent error detection
- Watchdog timers
- Watchdog processor
- Heartbeats
- Consistency and capability checking
- Data audits
- Runtime generated assertions

(c) 2012, Mehdi Tahoori

Reliable Computing I: Lecture 9

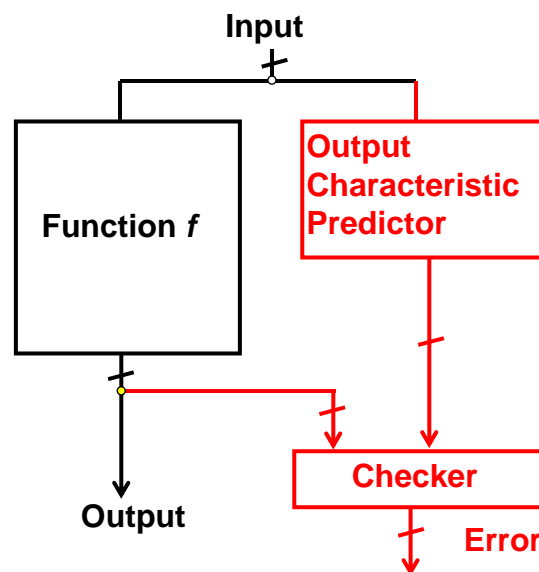
2

## Concurrent Error Detection (CED)



- Employed during normal operation
- Detect errors as they occur
  - Data integrity ensured
    - Correct outputs or
    - Error indicated for incorrect outputs
    - **Fault-secure property**

## General CED Structure



## Output “Characteristics”



- Output itself
  - Duplication
- Output parity
  - Parity prediction
- Residue
  - Residue codes
- 1s or 0s count in output word
- Many others

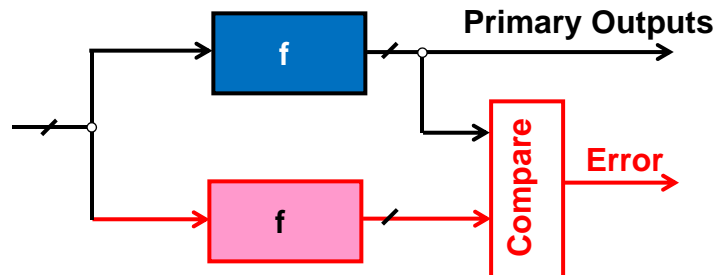
**All output characteristics are not equally effective**

## CED Classification & Examples



	Hardware	Software
Application independent	Identical Duplication Diverse Duplication Parity Prediction Residue codes Multi-threading Watchdog processor	Duplicated instruction Identical or Diverse Data Control-flow checking N-version programs
Application specific	Compression, Encryption, Signal processing, RESO, ...	Assertion checks Algorithm-based fault-tolerance

## Duplication for CED



- Two implementations: Identical or diverse
- Widely used (aka duplex system)
  - e.g., IBM G5, G6 processors, shuttle
- Issues
  - Common-mode failures, synchronization

## Fault Effects in Duplex Systems

- Single module failure
  - Guaranteed data integrity
- Multiple independent failures
  - Data integrity not guaranteed
  - Both modules generating identical errors
    - Very low probability
- Common-mode failures
  - Data integrity not guaranteed
  - More frequent

## Common-Mode Failures (CMFs)



- Multiple faults
  - Single cause
  - More probable than multiple independent failures
- Examples
  - Power-supply dip,
  - Single source radiation causing multiple upsets,
  - Design faults
- Antidote for CMF
  - Design Diversity
    - Diverse implementations
    - Error effects caused by CMFs are different

## Watchdog Timer



- An inexpensive method of error detection
- Process being watched must reset the timer before the timer expires,
  - otherwise the watched process is assumed as faulty
- Watchdog timers only detect errors which manifest themselves as a control-flow error such that the system does not continue to reset the timer
- Only processes with relatively deterministic runtimes can be checked, since the error detection is based entirely on the time between timer resets

## Watchdog Timer



- A watchdog timer provides only an indication of possible process failure
  - a partially failed process may still be able to reset the timer
- Coverage is limited, as neither the data nor the results are checked
- When used to reset the system, a watchdog timer can improve availability (the mean time to recovery is shortened) but not reliability (failures are just as likely to occur)
  - when the availability of a system is more important than the loss of data, the use of a watchdog timer to reset the system on the detection of an error is an appropriate choice.

## Example Applications of Watchdog Timers



- NASA's Mars Pathfinder mission
  - mars rover uses a real-time preemptive multithreaded operating system
  - tasks scheduled based on priorities that reflect their relative urgency
- **Major failure event:** priority inversion between tasks with different priorities
  - system deadlock
- Watchdog timer used to detect such scenario and restart the system
  - full restart causes loss of data
  - repetitive resets seriously limit the correct work of the system
  - the problem eventually diagnosed as a software bug
  - software patch reestablishes proper behavior
- **A traditional system reset is a drastic but robust measure used in engineering practice**
  - **availability of the system is more important than the lost data due to the system reset**

## Example Applications of Watchdog Timers



### ■ Telephone Switch System

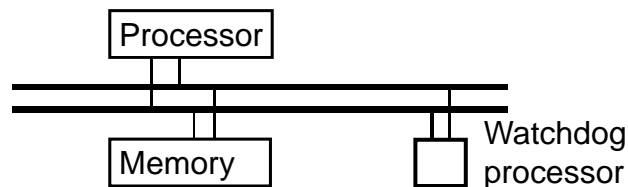
- External watchdog timers monitor correct program operation by triggering recovery when timers are not periodically reset
- Allows an early (before the error propagates) detection of problems caused by software errors and consequently easier recovery

## Watchdog Processor & Control Flow Checking



### ■ Watchdog processor: “Simple” processor

- Generalized version of watchdog timer
- Program control flow checked
- Assertion checks for computation errors
- Can be integrated into the processor itself

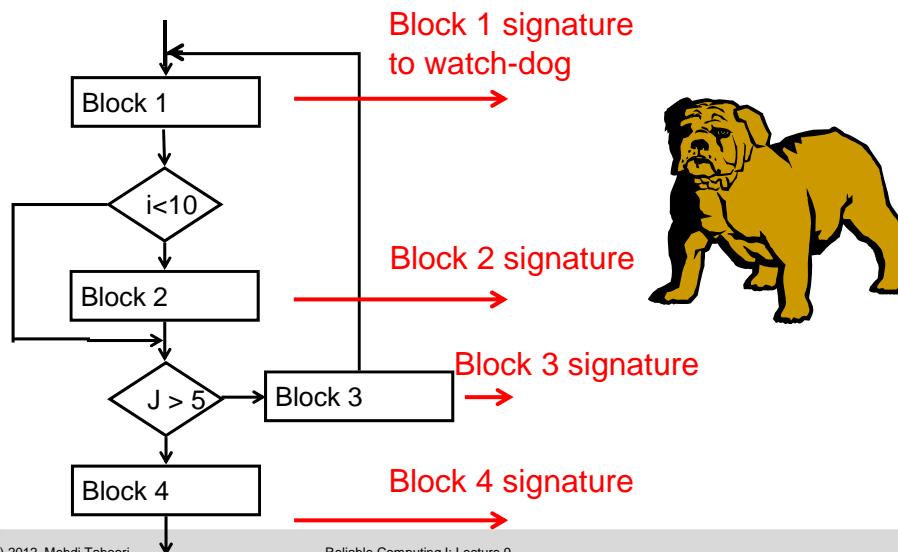


## Structural Integrity Checking (SIC)



- Program broken into basic blocks
  - Branch-free sequence of instructions
- Unique signature for each basic block
- Signatures explicitly transferred to watchdog
- Signature sequence checked by watchdog
- Automated at compiler level
  - Assignment of signatures
  - Instructions to send signatures to the watchdog.
  - Watchdog program can be automatically synthesized.

## Structural Integrity Check Example





## EDDI



- Error Detection by Duplicated Instructions
  - Intended for transient computation errors
- Duplicated Instructions
  - Master and shadow instructions
- Master and shadow results compared
  - Transient errors in computations detected
- Automated flow
- Performance overhead: 13%-111%
  - Super-scalar processors advantageous
  - No dependency between master & shadow

## EDDI Example



ADD R3, R1, R2	; R3 $\leftarrow$ R1 + R2
MUL R4, R3, R5	; R4 $\leftarrow$ R3 * R5
ST 0(SP), R4	; store R4 in location pointed by SP



ADD R3, R1, R2	; R3 $\leftarrow$ R1 + R2	master
ADD R23, R21, R22	; R23 $\leftarrow$ R21 + R22	shadow
MUL R4, R3, R5	; R4 $\leftarrow$ R3 * R5	master
MUL R24, R23, R25	; R24 $\leftarrow$ R23 * R25	shadow
BNE R4, R24, ErrorHandler	; compare master and shadow results	
ST 0(SP), R4	; store master result	
ST offset(SP), R24	; store shadow result	

## Heartbeats



- A common approach to detecting process and node failures in a distributed (networked) computing environment.
- Periodically, a monitoring entity sends a message (*a heartbeat*) to a monitored node or process and waits for a reply.
- If the monitored node does not respond within a predefined timeout interval, the node is declared as failed and appropriate recovery action is initiated.

## Heartbeats: Issues



- The timeout period is pre-negotiated by the two parties or sometimes even hard-coded by the programmer
- The predefined timeout value cannot adapt to changes in network traffic or to load variability on individual nodes
- The monitored node is assumed to be healthy if it is able to respond to a heartbeat message
- Process/thread responding to the heartbeat message may operate correctly, while other processes/threads may be in a deadlock situation or operating incorrectly

## Adaptive & Smart Heartbeat

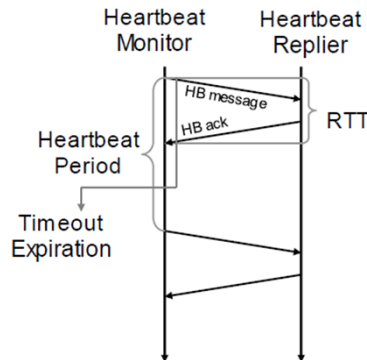


### Adaptive heartbeat

- the timeout value used by the monitor process is not fixed but is periodically negotiated between the two parties to adapt to changes in the network traffic or node load.

### Smart heartbeat

- the entity being monitored excites a set of predefined checks to verify the robustness of the entire process and only then responds to the monitoring process



(c) 2012, Mehdi Tahoori

Reliable Computing I: Lecture 9

21

## Consistency and Capability Checking



### Capability Checking

- can be implemented as a hardware mechanism or can be part of the operating system (usually the case)
- access to objects (memory segments, I/O devices) is limited to users (processors or processes) with the proper authorization

### Examples:

- virtual address management (MMU usually has a capability check)
- permission vs. activity; if these are not valid, there is an error trap
- password checking

### Consistency Checks

- range check - confirms that a computed value is in a valid range, e.g., a computed probability must be in the range 0 to 1
- address checking - verifies that the address to accessed exists
- opcode checking - checks whether the instruction to be executed has one of defined (documented) opcodes
- arithmetic overflow and underflow

(c) 2012, Mehdi Tahoori

22

## Data Audits



- Widely used in the telecommunications industry
- A broad range of custom and ad hoc application-level techniques for detecting and recovering from errors in a switching environment (in particular in a database).
- Data-specific techniques deeply embedded in the application can provide significant improvement in availability
- Static and Dynamic Data Check
  - A corruption in static data region detected by computing a golden checksum of all static data at startup and comparing it with a periodically computed checksum (e.g., Cyclic Redundancy Code)
  - For dynamic data, the range of allowable values for database fields are often stored in the database system catalog. This information is used to perform a range check on the dynamic fields in the database.

## Data Audits: Structural Checks



- The structure of the database is established by header fields that precede the data portion in every record of each table.
- Structural audit calculates the offset of each record header from the beginning of the database based on record sizes stored in system tables (all record sizes are fixed and known).
- The database structure (in particular, the alignment of each record and table within the database) is checked by comparing all header fields at computed offsets with expected values.

## Data Audits



### ■ Semantic Referential Integrity Check

- Traces logical relationships among records in different tables to verify the consistency of the logical loops formed by the record(s)
- Detects resource leaks
- Corruption of key attributes in a database leads to lost records, i.e., records participating in semantic relationships disappear without being properly updated

## Runtime Generated Assertions



### ■ Goals

- Generate runtime assertions by monitoring the values of selected variables in a program
- Use the monitored data to abstract out, via statistical pattern recognition techniques, the key relationships between the variables, separately and jointly, and to establish their probabilistic behavior

### ■ Approach

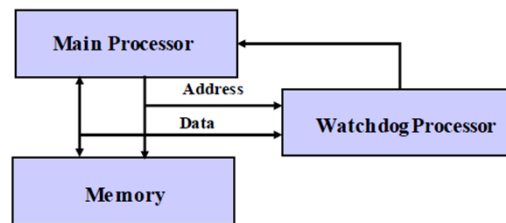
- Identify clusters of values traversed by different variables
- Use this information to automatically generate runtime assertions capable of capturing abnormal behavior of an application due to hardware or software errors
- Cross-check with other entities in the system their views on the state of selected variables
  - if a variable is globally accessible, then multiple entities (e.g., multiple execution threads) may have their own opinions about the correct value of the variable
  - can improve coverage and reduce false alarms

## Control-flow Monitoring Using Signatures



### ■ Hardware Approaches

- Employ a Watchdog (a simple co-processor) to monitor behavior of a Main Processor
- Suitable for a single embedded applications with little or no caching
- Limited applicability in off-the-shelf systems, as require additional specialized resources, e.g., watchdog, pre-compiler.



(c) 2012, Mehdi Tahoori

Reliable Computing I: Lecture 9

27

## Control-flow Monitoring Using Signatures



### ■ Hardware Approaches

#### ■ Embedded Signature Monitoring

- Pre-computed signature embedded in the application program
- Recompile of existing programs
- Performance degradation of application

#### ■ Autonomous Signature Monitoring

- Watchdog Processor stores pre-computed signature in the memory and mimics the control flow of application
- Watchdog Processor rather complex
- High memory overhead

(c) 2012, Mehdi Tahoori

Reliable Computing I: Lecture 9

28

## Control-flow Monitoring Using Signatures



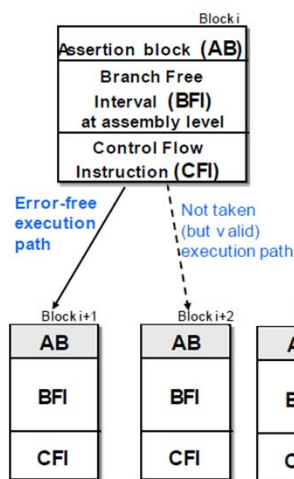
### ■ Software Approaches

- Software techniques partition the application into blocks, either in the assembly language or in the high-level language
- Appropriate instrumentation inserted at the beginning and/or end of the blocks
- The checking code is inserted in the instruction stream eliminating the need for a hardware watchdog processor
- Two classes of approaches
  - *non-preemptive* signature checking
  - *preemptive* signature checking

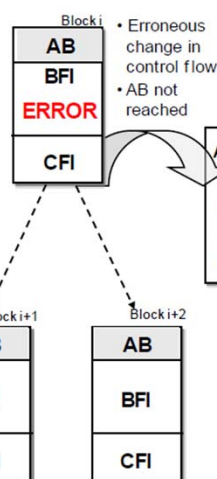
## Problems with Control Flow Signatures



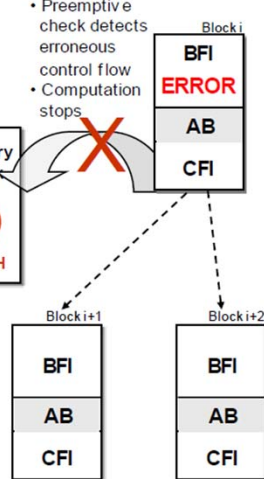
### Correct execution



### Incorrect execution without preemptive checking



### Incorrect execution with preemptive checking



## Preemptive Control Signatures (PECOS)

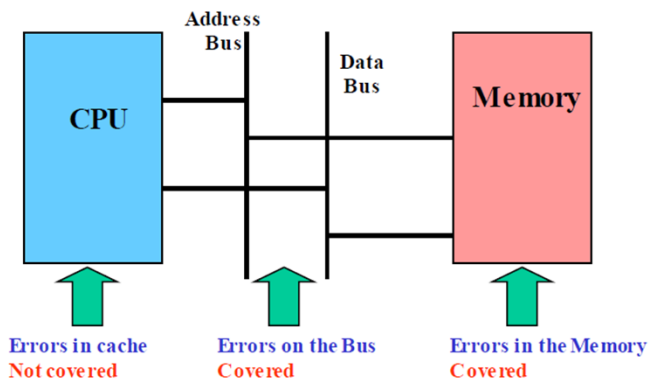


- PECOS determines the runtime target address and compares that against the valid addresses before the jump to the target address is made
  - executing instructions from an invalid target location is unlikely
- High-level control structure of Assertion Block
  1. Determine the runtime target address [= Xout].
  2. Extract the list of valid target addresses [= {X1,X2}].
  3. Calculate  $ID := Xout * 1/P$ ,
    - where,  $P = ![(Xout-X1) * (Xout-X2)]$
- Calculation of ID to raise a DIV-BY-ZERO exception in case of error
- Can handle single (jumps) or multiple (branches, calls, and returns) target addresses
- Assertion Block does not introduce any new control flow instruction

## PECOS



- What Can We Cover with Preemptive Software Control Signature?



- **Solution:** Insert programmable error detection core into the CPU