Development of Dependable Network-on-Chip Platform

Tomohiro Yoneda (National Institute of Informatics)
Masashi Imai (Hirosaki University)
Takahiro Hanyu (Tohoku University)
Hiroshi Saito (University of Aizu)
Kenji Kise (Tokyo Institute of Technology)
Recent cars are equipped with many ECUs

- Conventional ECU configuration

Sensors/Actuators

ECU1

ECU2

ECUi

ECUn

CAN, FlexRay, etc.
Recent cars are equipped with many ECUs

- Centralized ECU approach

![Diagram showing ECUs, sensors, and actuators connected via CAN, FlexRay, etc.]
Recent cars are equipped with many ECUs

- Centralized ECU approach

Any ECU can access any sensors/actuators

ECUs efficiently used by balancing loads
Tasks continuously executed even if some ECUs become faulty
(i.e., faulty ECU does not result in malfunction of its specific functions)
Our Project

- Platform for safety-critical automotive applications
  - Dependability in different levels
    - Circuit level
      - variation and delay faults
    - Routing level
      - link, router, chip faults
    - Task execution level
      - processor core faults
  - Evaluation kit
    - Tool for application development
    - Highly practical automotive application example
    - Pseudo plant model
Dependability in circuit level

- Fully asynchronous routers
  - robust against variation and delay faults
  - easily extendable

Code word detection by 0→1, 1→0

Odd phase
- 0011 → non-code word
- 01 → code word for 0
- 10 → code word for 1

Even phase
- 0011 → non-code word
- 01 → code word for 0
- 10 → code word for 1
Multi-Chip NoC
- Multiple NoCs are connected via off-chip links
  - On-chip networks seamlessly extended to multi-chip networks
  - Easily implemented thanks to fully asynchronous on-chip network
  - Efficiently implemented thanks to current-mode serial communication links
- Advantages
  - Cost-effective: small NoC chips are cheap, and various sizes of configuration are possible (without developing different sizes of NoCs)
  - Chip-level redundancy: tolerate a chip fault
Dependability in routing level
Dependability in routing level

- Online algorithm
  - Quick rerouting
  - At most one packet loss
- Simple and distributed algorithm
  - Implemented by hardware
  - Small performance overhead
- Tolerate single link/router/chip fault

Dependable routing algorithm

Faulty router
Dependability in task exec. level

- Duplicated execution, comparison, and pair-reconfiguration
Modified Pair & Swap

- Duplicated execution, comparison, and pair-reconfiguration

- Active tasks are also re-executed
  - Transient errors can be masked
Static / Redundant Task Allocation

Task graph

from IO

T₀
T₂
T₁

to IO

P₀  P₁  P₂  P₃  P₄  P₅

T₀  T₀  T₀

T₁  T₁  T₁  T₁

T₂  T₂  T₂  T₂
Static / Redundant Task Allocation

Task graph

from IO

\( T_0 \)

\( T_1 \)

\( T_2 \)

to IO

Alert should be indicated
Temporary TMR configuration

- Active tasks
  - should roll back their state variables

Diagram showing TMR command flow with input variables, previous states, state variables, and active/stand-by states.
Temporary TMR configuration

- To prepare for TMR configuration, stand-by task usually
  - Receives all input data given to active tasks
  - Receives the state variables updated by active tasks
Example of task execution

Sensor Inputs: i0, i1
State variables of tasks:

Control Cycle:
- P0: Active
  - T0: v0, v1, v0
  - T2: v0, v1, v2
- P1: Stand-by
  - T0: v0, v1
- P2: Just receive sensor inputs
  - T0: v0
- P3: Inactive
  - T0: v0
- P4: No action
  - T1: 0, 0, 0
  - T2: 0, 0, 0
- P5

Task graph:
- T0: from IO to IO
  - o2 is sent out
- T1
- T2
Example of task execution

Control Cycle

Mismatch

P0 fault detected
Fault pattern updated

TMR

Roll back state variables

T0

P0 fault detected
Fault pattern updated

T1

Roll back state variables

T0

P0 fault detected
Fault pattern updated

T2

P0 fault detected
Fault pattern updated

T1

Roll back state variables

T0

T0

P0 fault detected
Fault pattern updated

T2

P0 fault detected
Fault pattern updated

T1

Roll back state variables

T0

P0 fault detected
Fault pattern updated

T2

P0 fault detected
Fault pattern updated

T1

Old state variables are preserved, when state variables are updated
Evaluation Kit

- **Hardware platform**
  - 4 chip (4×4 2D mesh), V850E cores ×16
  - Virtex7 FPGA

- **Software development tool**

- **Automotive Application Example**

- **Pseudo Plant model**
Evaluation Kit
Evaluation Kit

Software Development tool

- Given by users
  - Simplex simulink program
  - Task declaration (by specifying atomic subsystems)
  - # of task copies allocated
  - # of processor cores available

- Front-end GUI tool supports
  - Allocation of multiple task copies to redundant processor cores with timing and memory constraints

- Back-end tool supports
  - C code generation for simulink codes
  - Wrapper code templates for receiving and sending data as well as handling TMR configuration
Evaluation Kit
Evaluation Kit

- Automotive Application Example
  - Integrated attitude control system for a four-wheel drive car
    - Torque, brake, and steering control of 4 wheels performed by ECUs
Evaluation Kit

- Pseudo Plant model
  - provide simulation environment like HILS
  - executed on a soft-processor in FPGA
Ongoing work

- **IO core duplication**
  - IO core plays simple but important roles
    - Implemented by hardware or a small processor
    - Simple crash fault assumed
      - Fixed duplex configuration
Ongoing work

- Maintaining real-time properties
  - Maximum latency should be obtained to check if real-time constraints are satisfied
    - Approach 1: using time slots to avoid congestion
      - E.g. Time-triggered NoCs [C. Paukovits, H. Kopetz: Concepts of Switching in the Time-Triggered Network-on-Chip, RTCSA ’08, pp.120-129]
    - Approach 2: using analytical model to estimate maximum latency
Dependable platform for safety-critical automotive applications has been developed

- Multi-Chip NoC based hardware
- Software development tool
- Practical automotive application example
- Pseudo Plant model