Scheduling in a Time-Triggered Protocol With Dynamic Arbitration

Jens Chr. Lisner
lisner@informatik.uni-essen.de

ICB / University of Duisburg-Essen
Introduction

Two methods of arbitration in TDMA-based protocols

- **Static arbitration**
  - Schedule pre-configured
  - Slots have fixed length
  - Can be implemented in a fault-tolerant way
  - Example: TTP/C, FlexRay (“static segment”)

- **Dynamic arbitration**
  - Schedule determined at runtime
  - Slots have dynamic length
  - Fault-tolerant implementation difficult
  - Example: Byteflight, FlexRay (“dynamic segment”)

The *Tea* protocol aims to solve the problem of fault-tolerant dynamic arbitration.
Introduction

*Tea* uses a mixed-mode approach:

![Communication cycle diagram]

- **Regular part**
  - Static slot length / static schedule

- **Extension part**
  - Dynamic slot length / dynamic schedule

Every controller can request one additional slot in the extension part.
Introduction

Schedule in extension provided by agreement algorithm

- slot request
- request vector

Mi = Message of controller i

Regular part

槽 1 槽 2 槽 (m/2)+2

A

B

M1 M3

M2 M4

M2 M4

M1 M3

Request phase: Contains request bit (request, no_request)

Confirmation phase: Contains vector of received requests (request, no_request, corrupted)

Schedule to channels is reversed in confirmation phase

Slots are shared by two controllers on two channels
Introduction

Architecture for fault-tolerant operation

- Two completely independent controllers reside on one node
- Double broadcast channel
- Controllers guard each other by controlling the other’s access to the bus
- Guaranteed fail-silent behavior
Fault-tolerant operation

Controller faults

- Controller sends “nonsense” data
  → Valid coding required

- Controller sends unexpectedly
  → Neighbor controller guards channels

- Controller can block neighbor
  → Does not have any impact on the agreement algorithm

Up to 2 controllers affected
Fault-tolerant operation

Channel faults

- Message corruption: Valid checksum (CRC, ...) required
- Message is delivered only to a subset of all controllers, while others receive corrupted messages or no signal:
  - Does not have any impact on the agreement algorithm
  - A request of a controller may be unknown, if it is sending on the faulty channel in the first half of the regular part.

Up to \( \frac{c}{2} \) controllers affected

Combinations of controller and channel faults possible
Scheduling Policies

Number of slots in the extension part is limited
→ scheduling policy required

Common criteria:
- Arrival time (cycle)
- Priority

Common strategies:
- First-in-first-out
- Static priorities
- Priority-first
- FIFO-first

HW requirements should be minimized.
The basic scheduling algorithm consists of three subroutines.

- **merge**: Merges the vector of current requests into the vector with outstanding requests
- **select**: Selects the next controller before the start of a new slot
- **adjust**: Adjusts the index registers
Basic Algorithm

The following algorithm implements a basic round-robin-strategy:

merge()

select()

adjust()
First-in-first-out

Implemented by modifying *merge*

Region of requests not yet processed remains untouched
Requests are processed in order of arrival
Static Priorities

Implemented by modifying *select*

- Every controller has a priority
- Controllers with higher priority are selected first

Danger: Lower prioritized controllers may never get a slot!

Solution: Use FIFO-first strategy
FIFO-first

Implemented by modifying \textit{merge} and \textit{select}

- \textit{merge}:
  Merge only if all outstanding requests have been processed.

- \textit{select}:
  (same as in static priorities)

Requests are always the same age and processed in the order of their priority.
Priority-first

Implemented by modifying *merge* and *select*

- Next
- \( S \)
- Priorities

- **merge:**
  - Follow the FIFO-strategy
  - Stop the *tmp* register if a request with same priority is found
  - Unique region for each priority level where requests cannot be merged
Priority-first

Implemented by modifying *merge* and *select*

- **select:**
  - Select next request with highest priority
  - Move next index only, if current request have been selected
  - Ignore priority when moving to the next request
Controller Faults

- Neighbor prevents bus access of faulty controller
- Empty slots are possible, but can be ignored
- Input to scheduling algorithm is the value unknown

Possible solutions:
- Count as no_request: Clear the respective bit if set
- Count as request: Set the respective bit if allowed
- Leave bit unaffected
  (best solution in connection with channel faults)
Problem:
Channel faults may lead to the value unknown for requests of fault-free controllers.

Channel Faults

regular part

request phase  confirmation phase
Channel Faults

Solution 1: Leave bit unaffected

- Slot is reserved if controller could successfully request slot at least once
- Good compromise if permanent faults are assumed to be unlikely
- Hardware changes not required
- No change of schedule in regular part required
- No extra cycle time required
- At least $c/2 - 2$ fault-free controllers may never successfully request a slot!
Solution 2: **Double cycle**

Reverse schedule of controllers in the regular part every two cycles

<table>
<thead>
<tr>
<th>Cycle n+1</th>
<th>Request Phase</th>
<th>Confirmation Phase</th>
<th>Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1 3 ...</td>
<td>2 4 ...</td>
<td>...</td>
</tr>
<tr>
<td>B</td>
<td>2 4 ...</td>
<td>1 3 ...</td>
<td>...</td>
</tr>
</tbody>
</table>

Fault-free controllers can successfully request a slot within a double-cycle

- Can cause further delays
Channel Faults

Solution 3: **Double request phase**

Reverse schedule of controllers in two consecutive request phases

Request of fault-free controllers are guaranteed within a cycle

Cycle length grows by $\frac{c}{2}$ static slots permanently
Channel Faults

Solution 4: **Additional link between both controllers in a node**
Channel Faults

Solution 4: **Additional link between both controllers in a node**

- Controller must also provide request for neighbor (extra bit necessary in request phase)
- Both controllers must be scheduled for different channels
- Request of fault-free controllers are guaranteed within a cycle
- No need to extend cycle
Conclusion

- A fault-tolerant solution for dynamic allocation in time-triggered protocols is provided by the Tea protocol.
- Controllers can be statically scheduled.
- Extra slots can be requested dynamically.
- Fault-tolerance can be assured.
- Dynamic allocation requires dynamic scheduling.
- Well known policies are available with low effort in hardware registers.
- Requests can be guaranteed in case of channel faults.
- Requests cannot be guaranteed for a fault-free neighbor of a faulty controller.