Scheduling in a Time-Triggered Protocol With Dynamic Arbitration

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

ICB / University of Duisburg-Essen
Germany
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:

```
| 1 | 2 | ... | r/2 | 1 | 2 | ... | r/2 | 1 | 2 | ... | e |
```

- **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**

- **Mi** = Message of controller i
- **request vector**

Regular part

| Slot | Mi
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>M1</td>
</tr>
<tr>
<td>2</td>
<td>M3</td>
</tr>
<tr>
<td>(m/2)+1</td>
<td>M2</td>
</tr>
<tr>
<td>(m/2)+2</td>
<td>M4</td>
</tr>
</tbody>
</table>

Scheduling to channels is reversed in confirmation phase

- **Request phase**: Contains request bit (request, no_request)
- **Confirmation phase**: Contains vector of received requests (request, no_request, corrupted)

- **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
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()
Implementation (Overview)

- FIFO ⇒ by changing *merge*
- Priority ⇒ by changing *select*
- FIFO-first ⇒ by changing *merge* and *select*
- Priority-first ⇒ by changing *merge* and *select*

HW requirements for registers in bit:

- $c + 3 \left\lceil \log_2 c \right\rceil$ for priority-first
- $c + 2 \left\lceil \log_2 c \right\rceil$ in all other cases
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)

A faulty controller can block a fault-free neighbour
Channel Faults

Problem:
Channel faults may lead to the value unknown for requests of fault-free controllers.
## Channel Faults

### Solution 1: Double cycle

Reverse schedule of controllers in the regular part every two cycles.

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

<table>
<thead>
<tr>
<th>Request Phase</th>
<th>Confirmation Phase</th>
<th>Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>A 2 4 ... 1 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B 1 3 ... 2 4</td>
<td></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 2: Double request phase

Reverse schedule of controllers in two consecutive request phases

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>request phase</th>
<th></th>
<th></th>
<th>request phase</th>
<th>confirm. phase</th>
<th>extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>3</td>
<td></td>
<td>2</td>
<td>4</td>
<td></td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>4</td>
<td></td>
<td>1</td>
<td>3</td>
<td></td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

- Request of fault-free controllers are guaranteed within a cycle
- Cycle length grows by $\frac{C}{2}$ static slots permanently
Channel Faults

Solution 3: **Additional link between both controllers in a node**

![Diagram of additional link between controllers](image)
Solution 3: **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.

- 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.