Formal Verification of Protocol Stack for MOST Network Service using SPIN

Dong-Ah Lee*, Sanghyun Yoon, Mu-Youl Lee, Hyun-Wook Jin, Junbeom Yoo
Contents

- Overview
- MOST Network Service
- u-OMNiPro 1.0
  - FBlock Framework
  - Message Core
- Modeling u-OMNiPro in PROMELA
  - Register & Unregister of FBlock
  - Control Message Transfer
- Verification and Result
  - Register & Unregister of FBlock
  - Control Message Transfer
- Conclusion and Future Work
Overview

- Network System for the Information of Automobile Status
  - CAN(Controller Area Network), LIN(Local Interconnect Network)

- Broadband Network System for In-vehicle Multimedia System
  - MOST(Media Oriented Systems Transport)
MOST Network Service

• Communication over MOST
  – Application
    • Virtual communication using Functional addressing
  – Network Service
    • APIs for implementing applications
    • Framework for a MOST device
    • Handling Messages
  – Network Interface Controller
    • Real communication using Physical addressing
Protocol Specification

- Stream Data
  - Audio, Video real time Data
- Packet Data
  - Navigation data, software download
- Control Data
  - Controlling of MOST devices

1 MOST Control Message = 16 Control Data Frame
Protocol Specification (Cont’d)

- Control data to manage user application
- 5 Bytes for Header
  - Information of Address (IDs) & Telegram (Tel ID, Tel Len)
- 12 Bytes for User Data
u-OMNiPro

- Structure of u-OMNiPro 1.0
- FBlock Framework
  - develop and manage FBlock
- Message Core
  - communication btw FBlock and Device Driver
- IPC
  - POSIX Message Queue
  - POSIX IOCTL

Overview of u-OMNiPro 1.0 Architecture
FBlock Framework

- APIs for implementing applications
  - Registration and Unregistration of FBlock
  - Registration and Unregistration of service function
  - Message transportation

<table>
<thead>
<tr>
<th>APIs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KUNS_FBlock_Register</td>
<td>FBlock Registration</td>
</tr>
<tr>
<td>KUNS_FBlock_Unregister</td>
<td>FBlock Unregistration</td>
</tr>
<tr>
<td>KUNS_FuncktionID_Register</td>
<td>Function Registration on FBlock</td>
</tr>
<tr>
<td>KUNS_FunctionID_Unregister</td>
<td>Function Unregistration on FBlock</td>
</tr>
<tr>
<td>KUNS_FBlock_RUN</td>
<td>FBlock Execution</td>
</tr>
<tr>
<td>KUNS_ctrl_recv</td>
<td>Receive Data from Message Core</td>
</tr>
<tr>
<td>KUNS_ctrl_send</td>
<td>Send Data from Message Core</td>
</tr>
</tbody>
</table>

FBlock Framework API
# Message Core

## Single Telegram Messages

<table>
<thead>
<tr>
<th>Tel ID</th>
<th>Tel Len</th>
<th>Data Byte 0 / Counter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 - C</td>
<td>Data</td>
<td>Single telegram</td>
</tr>
</tbody>
</table>

## Multiple Telegram Messages

<table>
<thead>
<tr>
<th>Tel ID</th>
<th>Tel Len</th>
<th>Data Byte 0 / Counter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C</td>
<td>0x00</td>
<td>First Telegram</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>0x01</td>
<td>Subsequent Telegram</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>...</td>
<td>Subsequent Telegram</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>0xFF</td>
<td>Subsequent Telegram</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>0x00</td>
<td>Subsequent Telegram</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>...</td>
<td>Subsequent Telegram</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>0x0n – 1</td>
<td>Subsequent Telegram</td>
</tr>
<tr>
<td>3</td>
<td>2 - C</td>
<td>0x0n</td>
<td>Last Telegram</td>
</tr>
</tbody>
</table>
Modeling (FBlock Register & Unregister)

- 3 Processes
  - Register
  - Unregister
  - regTableThread

- Register
  - Request Registration of FBlock

- Unregister
  - Request Unregistration of FBlock

- regTableThread
  - Registration & Unregistration of FBlock
  - Manage FBlock Table

typedef NSHandler{
  byte fBlock;
  byte instId;
  byte status;
};

Data Structure
Modeling (Echo)

- 5 Nodes
  - 1 Source Node (4 Processes)
  - 4 Echo Nodes (4 Processes)
- FBlock
  - Source – Generate Messages
  - Echo – Return Messages
- Send / Receive
  - Handling Messages
- Device Driver

```c
typedef perfect_Ctrl_Data { byte tgt_addr;
  short ctrl_data_len;
  byte ctrl_data[6] }

typedef ctrl_data_t { byte tgt_addr;
  byte ctrl_data[6] }
```

Data Structure
Verification Result \textit{(Register & Unregister of FBlock)}

- When the registration function is called with a service function, are properties of the service function registered to the table correctly?

\[
\begin{align*}
\text{If} & \quad \text{count} < \text{MAX\_Table\_Size} \rightarrow \text{assert}(\text{pNsHandler[count-1].fBlock != FBlock.buff[0]}); \\
\text{count} == 0 \rightarrow \text{assert}(\text{pNsHandler[MAX\_FUNCTION * MAX\_INSTID - 1].fBlock != FBlock.buff[0]}); \\
\text{fi};
\end{align*}
\]

- When the unregistration function is called with a service function, are properties of the service function unregistered from the table correctly?

\[\text{assert}(\text{pNsHandler[count].fBlock != FBlock.buff[0]})\]

- Process is terminated by force
- Unregistration function is not implemented
Verification Result (Echo)

• Is a message generated by the source node same with return message from a echo Node?

  \[\text{assert}(\text{recvdCtrlData}.\text{ctrl\_data\_len} \neq \text{sentCtrlData}.\text{ctrl\_data\_llen})\]
  
  – Process is terminated by force
  – A size of single telegram is not defined

• If a first telegram was sent, is it guaranteed that a final telegram is always sent eventually?

  \[\lnot (\text{sendTelID} == 1) \rightarrow \neg (\text{sendTelID} == 3) \]
Conclusion and Future Work

• Conclusion
  – Formal Modeling & Verification for u-OMNiPro 1.0
  – We found two possible problems
    • Unimplemented problem
    • Property outside of model

• Future Work
  – Formal Modeling & Verification for k-OMNiPro (kernel-level OMNiPro)