A model based design flow for CAN-based systems

Alexios Lekidis¹, Marius Bozga¹, Didier Mauuary², Saddek Bensalem¹

¹UJF-Grenoble 1 / CNRS-VERIMAG, ²Cyberio

14th international CAN Conference
Eurosites Republique, Paris (France)
November 12-13, 2013
1) Design challenges for CAN-based systems
2) Model-based design flow using BIP
   • BIP overview
   • CAN protocol model
   • Application software modeling
   • Construction of the system model
3) Application and experimental results
4) Conclusion and ongoing work
1) Design challenges for CAN-based systems

2) Model-based design flow using BIP
   • BIP overview
   • CAN protocol model
   • Application software modeling
   • Construction of the system model

3) Application and experimental results

4) Conclusion and ongoing work
Each unit in the network may incorporate many subsystems

- Increased communication complexity
- System design becomes difficult
Emergence of higher layer protocols

- Organize and abstract low-level communication complexity
- Extend its usage to a wide range of applications including:
  - Machine automation, medical devices, photovoltaic systems, maritime electronics e.t.c.

![Diagram showing a model-based design flow for CAN-based systems.](image-url)
Emergence of higher layer protocols

- Organize and abstract low-level communication complexity
- Extend its usage to a wide range of applications including:
  - Machine automation, medical devices, photovoltaic systems, maritime electronics etc.

![Diagram of CAN, CANopen, DeviceNet, J1939, CAN Controller, CAN Bus, System integration and validation is too difficult]

Lekidis, Bozga, Mauuary, Bensalem
Emergence of higher layer protocols

- Organize and abstract low-level communication complexity
- Extend its usage to a wide range of applications including:
  - Machine automation, medical devices, photovoltaic systems, maritime electronics e.t.c.

System integration and validation is too difficult

Successful design remains a challenge
Conformance testing

- Verifies the correct implementation and integration of the system
- Increases interoperability of devices
- Occurs late in the development cycle
- Requires the final system implementation

- Potential design errors can lead to a new implementation of the system
- Performance aspects are ignored during the design process
Solution: Model-based design

Formal approach expressing the behavior and functionality of embedded systems

- Validation and verification enabled at any stage
- Formal models for the software and hardware allowing:
  - Separation of concerns
  - Modularity
  - Reusability
Solution: Model-based design

Formal approach expressing the behavior and functionality of embedded systems
- Validation and verification enabled at any stage
- Formal models for the software and hardware allowing:
  - Separation of concerns
  - Modularity
  - Reusability

- Previous work is based on multi-language frameworks, in order to provide a design flow for automotive systems
Solution: Model-based design

Formal approach expressing the behavior and functionality of embedded systems
- Validation and verification enabled at any stage
- Formal models for the software and hardware allowing:
  - Separation of concerns
  - Modularity
  - Reusability

- Previous work is based on multi-language frameworks, in order to provide a design flow for automotive systems

Semantically unrelated formalisms lead to lack of continuity
Solution: Model-based design

Formal approach expressing the behavior and functionality of embedded systems
- Validation and verification enabled at any stage
- Formal models for the software and hardware allowing:
  - Separation of concerns
  - Modularity
  - Reusability

Rigorous design flow for CAN-based systems:
- Based on a single semantic framework
- Encapsulates the protocol’s communication mechanisms and primitives
- Incremental design using composite components

Semantically unrelated formalisms lead to lack of continuity
Design flow

- Application software
- CAN hardware/communication mechanisms
Design flow

A model-based design flow for CAN-based systems

Tools

Component reuse

Application software

CAN hardware/communication mechanisms

Translation

Modeling

Application model

CAN protocol model

Transformation

CAN System Model

BIP component framework

CAN component library
Design flow

A model-based design flow for CAN-based systems

Verification of safety properties

Tools

Component reuse

Platform dependent skeleton C/C++ code

Validation and model evaluation

Code generation

Deployable code
Outline

1) Design challenges for CAN-based systems

2) Model-based design flow using BIP
   - BIP overview
   - CAN protocol model
   - Application software modeling
   - Construction of the system model

3) Application and experimental results

4) Conclusion and ongoing work
BIP (Behavior-Interaction-Priority) is a formal language for the hierarchical construction of composite components.

BIP component framework

- Priorities (conflict resolution)
- Interactions (collaboration)

Composition glue
Atomic components

Provides a rich set of tools for analysis and performance evaluation
BIP: Atomic components

- Finite state automata (Petri nets) extended with data and functions described in C/C++
Communication between components involving data exchange

\[ s := 0 \]
\[ t := 0 \]
\[ r := 0 \]

\[ r := r + s \]
\[ t := t + 1 \]

\[ S:\text{SEND} \]
\[ s := s + 1 \]
\[ S:\text{TICK} \]
\[ t := t + 1 \]

\[ R:\text{RECV} \]
\[ r := 0 \]
\[ t := 0 \]

\[ R:\text{R\_COM} \]
\[ r := 0 \]
\[ t := 0 \]
\[ R:\text{TICK} \]
Communication between components involving data exchange

\[ r := r + s \]

\[ T I C K \]

\[ t := t + 1 \]

\[ R \_ C O M \]

\[ T I C K \]

\[ r := 0 \]

\[ t := 0 \]

\[ R \_ C O M \]

\[ T I C K \]

\[ r := 0 \]

\[ t := 0 \]
BIP: Priorities

- Used among competing interactions

\[
\begin{align*}
TICK < R\_COM \\
TICK \quad t := t + 1 \\
TICK \quad r := r + s \\
TICK \quad r := 0 \quad t := 0 \\
TICK \quad s := s + 1 \quad [s < 100] \\
TICK \quad s := 0 \\
R\_COM \quad t := t + 1 \\
R\_COM
\end{align*}
\]
The BIP toolset offers:

- Translators from various languages and models into BIP
- Source-to-source transformers
- Code generation by dedicated compilers

More information and related material at:

http://bip-components.com
Modeling the CAN protocol

Diagram showing the process of modeling CAN-based systems, including application software, translation, modeling, transformation, validation, and code generation.
Main aspects of the CAN protocol model

- Represents the classic CAN protocol functionality [CAN specification version 2.0]
- Supports the Basic CAN interface [ISO 11898-1]
- Is compliant with the High-Speed physical layer standard [ISO 11898-2]
- Does not consider transmission errors
Main aspects of the CAN protocol model (1)

- Represents the classic CAN protocol functionality [CAN specification version 2.0]
- Supports the Basic CAN interface [ISO 11898-1]
- Is compliant with the High-Speed physical layer standard [ISO 11898-2]
- Does not consider transmission errors
Main aspects of the CAN protocol model (1)

- Represents the classic CAN protocol functionality [CAN specification version 2.0]
- Supports the Basic CAN interface [ISO 11898-1]
- Is compliant with the High-Speed physical layer standard [ISO 11898-2]
- Does not consider transmission errors
Main aspects of the CAN protocol model

Each CAN frame:

- Is one of the following types:
  - Data transmission (data frame)
  - Data request (remote frame)
- Contains the following fields:
  - arb : frame identifier
  - rtr : Remote Transfer Request (RTR) bit
  - ide : Identifier Extension (IDE) bit
  - length : Length of data
  - payload : Frame data
Queuing policy is of type:
- First-In-First-Out (FIFO)
- High Priority First (HPF)

Acceptance filters:
I. Receive every frame from the Bus
II. Deliver it to the application or ignore it
CAN bus component
Discrete time step advance

Transmission of one bit ($\tau_{bit}$) corresponds to one tick
Modeling the application software

[Diagram of a model-based design flow for CAN-based systems]

- Application software
  - Translation
  - Application model
- CAN hardware/communication mechanisms
  - Modeling
  - CAN protocol model
  - CAN component library
- Transformation
  - CAN System Model
  - BIP component framework
  - Validation and model evaluation
  - Code generation
  - Deployable code
Modeling the application software

- Every application consists of a number of Devices
- Each Device generates a set of frames
  - Transmission is defined by the following attributes:
    - Periodic, event-triggered or purely stochastic
    - With or without offsets
    - Abortable or non-abortable request
- Currently provided by XML-based descriptions
  - Compliance with NETCARBENCH [Navet et al., 2007]
- Accordingly provided Device examples with focus on the transmission part
Device examples

**ECU with periodic transmission**

*N periodic frames, where periods are: \( P[i], i=1, \ldots, N \)

**ECU with stochastic transmission**

*N frames, with a transmission jitter chosen by a given distribution and periods: \( D[i], i=1, \ldots, N \)

Lekidis, Bozga, Mauuary, Bensalem  
A model-based design flow for CAN-based systems
Device examples

**ECU with periodic transmission**

*N periodic frames, where periods are: \( P[i], i=1, \ldots, N \)*

**ECU with stochastic transmission**

*N frames, with a transmission jitter chosen by a given distribution and periods: \( D[i], i=1, \ldots, N \)*
The CAN-based system model

A model-based design flow for CAN-based systems
Constructing the system model
Constructing the system model

Device 1
- REQUEST
- RECV

Device 2
- REQUEST
- RECV

Device 3
- REQUEST
- RECV

Device n
- REQUEST
- RECV

Application model

CAN station 1
- REQUEST
- RECV

CAN station 2
- REQUEST
- RECV

CAN station n
- REQUEST
- RECV

CAN bus
- COMM

CAN protocol model

Lekidis, Bozga, Mauuary, Bensalem
A model-based design flow for CAN-based systems 21/30
Constructing the system model

A model-based design flow for CAN-based systems

Lekidis, Bozga, Mauuary, Bensalem
1) Design challenges for CAN-based systems
2) Model-based design flow using BIP
   • BIP overview
   • CAN protocol model
   • Application software modeling
   • Construction of the system model
3) Application and experimental results
4) Conclusion and ongoing work
Case study using the BIP design flow, where:

- The application software consists of:
  - 5 Devices
  - 30 periodic frames with associated:
    - CAN identifier, period and payload
    - HPF queuing policy in every Device
    - Fixed 10% bit-stuffing for all the frames
    - No transmission offsets
  - The Bus has a bit-rate of 500 kbit/s
  - Load equally distributed among the Devices
The generated **system model** contains:

- 20 atomic components
- 60 connectors
- 255 transitions
- 1250 lines of BIP code
Results: Response times

- 1 hour of real system time was simulated in 5 minutes and 30 seconds.
Results: Response times

- The same results can be obtained using RTaW-Sim [RealTime-at-Work]
  - Much shorter simulation time (13.5 seconds)

- 1 hour of real system time was simulated in 5 minutes and 30 seconds
Stochastic powertrain network

- Extension to the previous case study:
  1. Probabilistic margin (jitter) for every period
  2. Stochastic bit-stuffing
Stochastic powertrain network

- Extension to the previous case study:
  I. Probabilistic margin (jitter) for every period
  II. Stochastic bit-stuffing
Stochastic powertrain network

- Extension to the previous case study:
  I. Probabilistic margin (jitter) for every period
  II. Stochastic bit-stuffing
Stochastic powertrain network

- Extension to the previous case study:
  I. Probabilistic margin (jitter) for every period
  II. Stochastic bit-stuffing

- This analysis cannot be obtained with RTaW-Sim
Outline

1) Design challenges for CAN-based systems

2) Model-based design flow using BIP
   - BIP overview
   - CAN protocol model
   - Application software modeling
   - Construction of the system model

3) Application and experimental results

4) Conclusion and ongoing work
Summary

Proposed method: Rigorous design flow resolving effectively the current challenges in CAN-based systems

- Encapsulates the primitives and communication mechanisms of the CAN protocol
- Separates software and hardware design issues
- Fully automated and tool-supported
- Leads to the construction of a mixed hardware and software system model used for:
  - Performance analysis
  - Verification of functional and extra-functional properties
  - Code generation
- Conducted experiments on existing benchmarks illustrate the capabilities and the method’s scalability
Ongoing work

- **CAN protocol model**
  - Selection of the most appropriate distribution for the bit-stuffing and the period margin
  - Design flow for CAN FD-based systems
- **Considered application software**
  - MATLAB/Simulink to BIP translation
- **Further extensions**
  - Analysis and verification of properties using the Statistical Model Checking BIP tool
  - Generation of optimal device configurations
  - Validation of CAN-higher layer protocols, such as CANopen
Thank you for your attention