

Description of Laboratory Experiments

Laboratory 1 Model Based Simulation and GUI Design

1 Background

Applications involving distributed embedded systems are generally complex. To deal with this complexity, a powerful paradigm is used: simulation. To make simulations more intuitive and easier to relate to humans, appropriate graphical user interfaces (GUI's) are used. In this lab, we will provide an introduction to a powerful simulation tool (CANoe) suitable for distributed embedded systems that are based on discrete events and on the CAN (controller area network) communication protocol. CANoe is a robust CAN tool that is capable of simulating an entire distributed application including the CAN sub-system, the bus, and its associated nodes. It also allows one to perform sophisticated analysis operations, and provides customizable graphical interfaces.

2 Experiment Description

This laboratory experiment is an introduction to simulation of distributed applications using CANoe and the main GUI feature used by CANoe: panels. First you will open a predefined CANoe simulation. In this simulation you will observe network activity on the bus by manipulating various controls on the panels and causing their related CAN message to be transmitted. In the second part of the lab, two panels will be built in order to interact with a CAN bus for future labs. In CANoe, each panel is defined by a *.cnp file. This file holds all the information about the panel, including what types of controls and indicators are on the panel, where on the panel they are located, and which environmental variables they are linked to in the database.

3 Experiment Learning Objectives

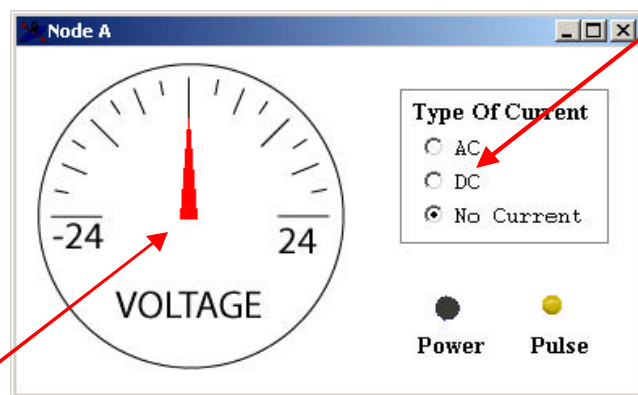
After completing this experiment you should be able to:

- 3.1 Identify and name the most important CANoe simulation components of a distributed embedded system.
- 3.2 Explain the functions of the various simulation blocks.
- 3.3 Run a CANoe simulation model and extract relevant simulation data.
- 3.4 Obtain detailed information of the application from simulation plots and simulation data.
- 3.5 Name and explain the various GUI objects available in CANoe
- 3.6 Configure the various GUI objects appropriately within a panel.
- 3.7 Associate environmental variables to the various GUI objects. Create appropriate panels for a specific distributed embedded application.

Panel A.cnp

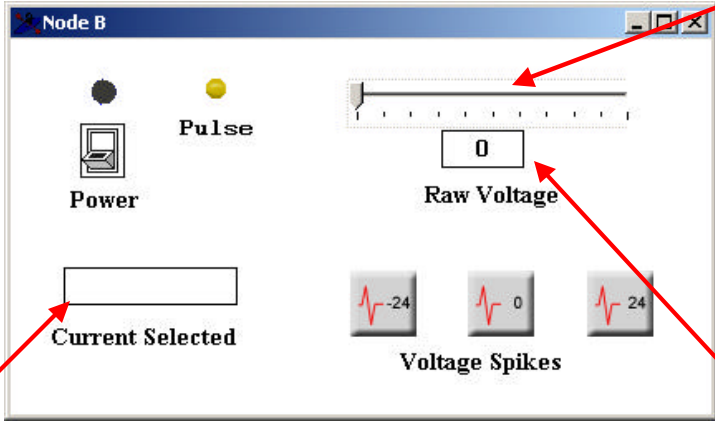
Elements: 3 Radio Buttons:

- 1) EnvVar: ev_CurrentSelector_A, Text: AC, Switch Value: 1
- 2) EnvVar: ev_CurrentSelector_A, Text: DC, Switch Value: 2
- 3) EnvVar: ev_CurrentSelector_A, Text: No Current, Switch Value: 0



Element: Meter
Type: Float
EnvVar: ev_Voltage_A
Arc Range: -90 to 90
Value Range: -24.0 to 24.0

Panel B.cnp



The screenshot shows a control panel for 'Node B' with the following elements:

- Power:** A black dot and a yellow dot, with a power icon below.
- Pulse:** A yellow dot.
- Raw Voltage:** A slider control with a value of 0 displayed below it.
- Current Selected:** A text input field.
- Voltage Spikes:** Three waveform icons labeled -24, 0, and 24.

Annotations with red arrows point to the following elements:

- Current Selected:** Element: Input/Output Control, Type: String, Read Only, ev_CurrentSelector_B
- Raw Voltage:** Element: Slider, Type: Integer, ev_Voltage_B, Minimum: 0, Maximum: 255
- Raw Voltage (Value):** Element: Input/Output Control, Type: Integer, ev_Voltage_B

Laboratory 2

ECU Simulation of a Distributed Embedded System

1. Background

To assist with rapid prototyping, CANoe's node simulation ability can be used. There are times when rapid development is required and quick access to a physical ECU and its actual software is impractical or impossible. The simulation ability cuts down on development time while still allowing the engineer to perform an accurate simulation of the system to determine how it acts under various conditions.

2. Experiment Description

In this experiment, a distributed system consisting of two nodes as depicted in Figure 1 will be simulated, Node A sends two messages corresponding to a selected current type and a heartbeat. Node B sends three messages corresponding to a voltage, power, and heartbeat. Most of the functionality of the simulated system will be given to you and you are to add the missing functionality to have a working system.

The experiment consists of two parts. In part one, a new simulation configuration will be created. This configuration will be used for the simulation performed in part two. Then in part two, node program files will be edited to add functionality. Two programs, one for each node, have already been written. They are roughly 80% complete. Several components of the software are to be added to complete the simulation and allow the two nodes to operate as desired.

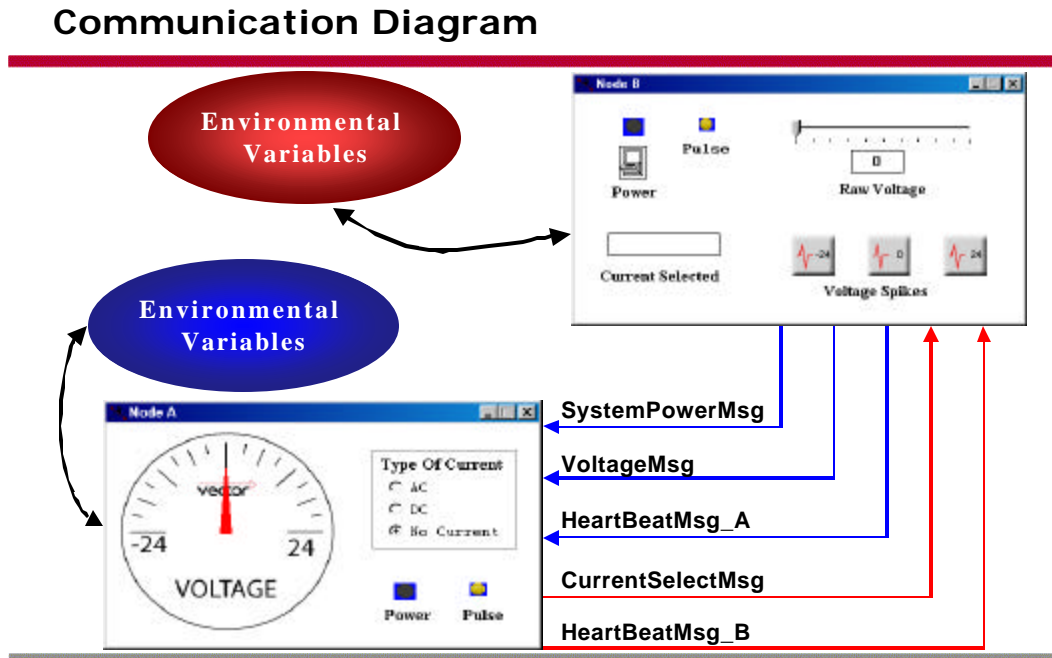


Figure 1 - Complete System Overview

4 Experiment Learning Objectives

After completing this experiment, the student should be able to:

- 4.1 List the advantages of node simulation in distributed embedded systems.
- 4.2 Set up a new simulation environment in CANoe.
- 4.3 List the main parameters of a CANoe simulation setup.
- 4.4 List the main components of a CANoe simulation environment.
- 4.5 Configure, edit, link, and run simulation scripts.
- 4.6 Add functionality to an application.
- 4.7 Code functionality into appropriate nodes using the CAPL language.
- 4.8 Run and debug CAPL code on appropriate nodes

Laboratory 3

Network Scheduler of a Distributed Embedded System

Background

By definition, a distributed embedded system involves a communication network. The distributed embedded application is distributed over a number of nodes that communicate by exchanging messages. A typical application involves the following messages: periodic, sporadic, and event based. The communication network must be able to deal with embedded applications that have a mixture of these types of messages. A scheduling algorithm or policy is used to schedule messages onto the actual communication protocol. For this lab, students will be introduced to a scheduling algorithm to schedule messages for the CAN protocol. In CAN networks, messages are scheduled by defining a priority for each message. This is done by defining identification (ID) numbers.

1. Experiment Description

In this experiment, you are to use the included Java software for an already developed scheduler to generate a message transmission schedule for a CAN network. You will evaluate performance measures of the scheduler, including an analysis of message latencies. In a later experiment you will verify these calculations by performing measurements on an actual network.

2. Experiment Learning Objectives

- Use Tindell's formulae to verify the message schedule and find the message latencies.
- Produce a table of messages and the associated parameters including type, size, period, and deadline.
- Schedule the messages for transmission on a CAN network.
- Evaluate scheduler's performance measures: distribution of message latencies.

Laboratory 4 Analysis of Message Communication Patterns

1 Background

Whereas experiment three, "*Network scheduler of a distributed embedded system*" deals with the issue of network schedulers from a theoretical and off-line basis, this experiment deals with the same issue but from an experimental basis. You are to use appropriate lab tools to load the schedule performed in the "*Network scheduler of a distributed embedded system*" experiment and verify that the theoretical schedule (i.e., the communication patterns) of lab three and the experimental schedule obtained in this lab are equivalent.

2 Experiment Description

In this experiment, you will use CAN tools to configure, load, and verify all messages corresponding to the schedule that was obtained in the "*Network scheduler of a distributed embedded system*" experiment. Using the CAN tools, you will verify the communication patterns associated with the schedule. This experiment will be performed using simulated CAN nodes rather than actual microcontrollers with CAN interfaces.

A CANoe procedure has already been created to measure the latencies of the messages included in the database. The CANoe procedure measures the time between a message-transmit request and the completion of a message transmission. A message-transmit request occurs when the application calls a function requesting a message to be transmitted. The message isn't actually transmitted at this time because the bus may be in use by another node. Rather, the message is placed in a transmit buffer and awaits transmission. The time at which the message transmission actually takes place is determined by the message's priority (ID). A lower ID corresponds to a higher priority. The highest priority message in the buffer but across all nodes is transmitted first. If a message with a very low priority is in the buffer and higher priority messages continually arrive to other nodes of the network, the low priority message transmission may be delayed significantly. The actual message transmission starts

when the particular node has acquired the bus and the pending message is in the transmit buffer. The time between a message-transmit request and an the completion of message transmission is known as the message latency.

Due to the irregular nature of individual latency readings, the procedure takes 10 readings and averages them together before they are displayed in the Write window (and also an output file). For example, if the message cycle attribute in the database is set for 1000ms, the message is transmitted every second. As the message is transmitting, the latency will be calculated. Nothing will be displayed until 10 seconds (10 periods) have passed. At this time the average latency is calculated and printed to the write window and the output file.

3 Experiment Learning Objectives

After performing this experiment a student should be able to:

- 3.1 Verify and change message parameters (data length, cycle period, and priority) using the CANoe database editor.
- 3.2 Use the *common attributes window* to change parameters.
- 3.3 Program a message schedule into a CAN network.
- 3.4 Start and stop CANoe experiments to measure message latencies (delays) for several load configurations.
- 3.5 Read and interpret measured message latencies.
- 3.6 Compared measured message latencies with theoretical values obtained in a previous experiment.

Laboratory 5 ECU Implementation of Distributed Embedded Functions

1 Background

In the previous labs, you have been experimenting with CANoe's ability to simulate a CAN bus and its corresponding nodes. When hardware is unavailable or impractical and a systems designer needs to know how the network might respond to various events, this simulation works fine. Eventually, there comes a time when testing with physical hardware becomes a necessity. One important feature of CANoe is its ability to have simulated and real nodes on a bus at the same time. This allows the engineer to slowly add components to the bus one at a time so the process of going from a simulated environment to a target environment is much easier.

2 Experiment Description

Up to this point, you have mainly been working with simulated nodes on a simulated CAN bus. The natural next step is to work with real ECUs on a physical bus. In this experiment you will be working on the exact same CANoe project that was used in Experiments 1 and 2. However, this time you will replace Node B with an actual ECU. The ECU will implement most, but not all of the functionality that existed in CANoe's simulated Node B in Experiments 1 and 2. In particular, you will not implement the +24v, -24v, or 0v spike buttons. Thus the following functionality from the original Node B will be programmed into the ECU/Microcontroller:

1. **Power On Switch**
2. **Raw Voltage Slider**
3. **Current Selected Indication**
4. **Pulse LED**
5. **Node B Heartbeat**

PLEASE READ APPENDICES A & B BEFORE PROCEEDING!

As a starting point, you have been provided with a Software Implementation Package (SIP) set up for the Motorola MC9S12DP256. This implementation is built for the two-node network depicted in Figure 1 which only shows the simulated node (NodeA). This node is identical to that used in labs 1 and 2. You will not need to edit this node whatsoever. There is no reason to have the real ECU (Node B) in CANoe's simulation setup (notice only node A appears in the configuration of Fig. 1). Node B operates completely independent of CANoe and CANoe does not 'know' it is there. Likewise, the Microcontroller can't tell that the other node on the network is actually simulated. When it communicates, it just assumes there is another real node on the bus.

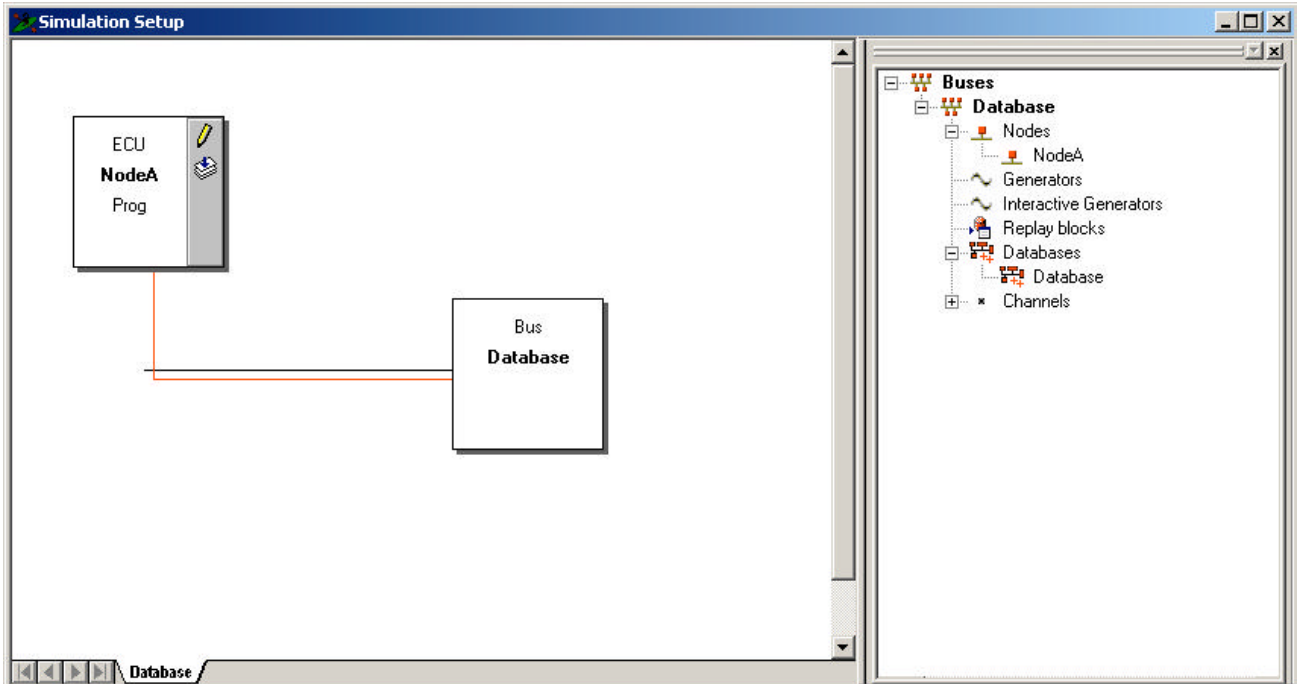


Figure 1 - Node Layout

3 Experiment Learning Objectives

After completing this experiment, the student should be able to:

- 3.1 Incorporate an ECU into an existing CAN network.
- 3.2 Implement new application functions on a microcontroller.
- 3.3 Program a set of application functions on a microcontroller using the C-language.
- 3.4 Use a CAN-communication SIP for a microcontroller.
- 3.5 Caused appropriate actions to be executed upon:
 - Start up and power down
 - State change
 - An interrupt
 - Timer expiration
 - Message reception
- 3.6 Use the following peripherals on a microcontroller:
 - CAN channel

- Analog to digital converter
- Timer
- Digital I/O

3.7 Configure, edit, compile, link, entire microcontroller code.

3.8 Integrate, test, and debug a complete network involving simulated and actual (i.e., target) nodes on a CAN bus.