



GMLAN Communication Strategy Specification

1 Introduction

Note: Nothing in this standard supercedes applicable laws and regulations.

Note: In the event of conflict between the English and domestic language, the English language shall take precedence.

1.1 Scope. This specification establishes the communication strategy for the GM In-Vehicle Local Area Network (GMLAN) Subsystem. Implementation of this strategy is required for all Electronic Control Units (ECU) connected to the network.

1.2 Mission/Theme. The GMLAN subsystem shall provide a reliable, cost effective, flexible and modular way to handle information sharing between different ECUs in the vehicle through the means of a family of serial communication buses. GMLAN is scaleable to support the range from simple control modules up to highly complex units. The communication system also supports the transfer of diagnostic services and the transformation from and to a specified set of globally accepted standards for diagnostic tester access and communication.

1.3 Classification. Not applicable.

1.4 Requirement Wording Conventions.

Within this document the following conventions are applied:

The word "Shall" is used in the following ways:

- a. To state a binding requirement on the GMLAN network, or the nodes which comprise the network, which is verifiable by external manipulation and/or observation of a node or the network.
- b. To state a binding requirement upon a node's requirements document that is verifiable through a review of the document.

The word "Must" is used in the following ways:

- a. To state a binding requirement upon nodes on the network that will have a corresponding node requirements document. These requirements will be verified as part of the component verification.
- b. To characterize an expected behavior that is the natural consequence of another requirement in this document.

The word "Should" denotes a preference or desired conformance.

2 References

Note: Only the latest approved standards are applicable unless otherwise specified.

2.1 External Standards/Specifications.

ISO 11898

ISO-WD 15765-2

2.2 GM Standards/Specifications.

GMW3089

GMW3122

GMW14241

GMW3110

GMW3173

3 Subsystem Definition

The GM in vehicle Local Area Network (GMLAN) is to provide the capability to transfer information between the different nodes on the vehicle, by a set of electrical wires, a serial communication bus.

3.1 Document Structure. There are two types of documents in GMLAN: General Specifications and Device Specifications. (See Figure A1).

3.1.1 General Specification. The general specifications establish the communications concept. For a given ECU, it is possible that not everything in the general specifications will apply.

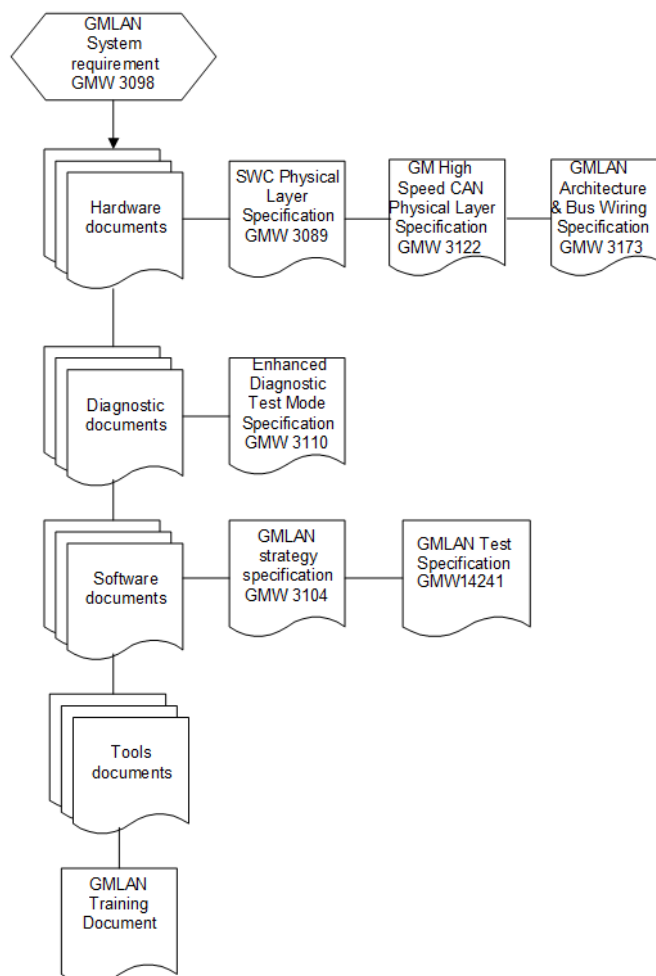


Figure 1: GMLAN Documents

3.1.2 Device Specification. The device specification is ECU specific. It contains:

- References to the optional parts of general specifications that applicable to the ECU.
- ECU specific information (e.g., signals and messages received by that ECU).

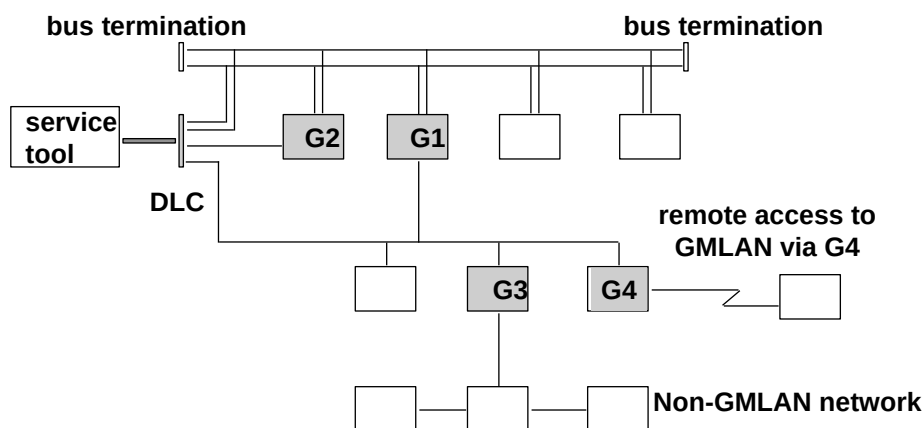
3.2 Overview.

3.2.1 GMLAN Buses. GMLAN provides a family of serial communication buses to enable Electronic Control Units (ECU) to communicate with each other. GMLAN comprises of three buses, a Low-speed bus, a Mid-speed bus, and a High-Speed bus, all of which are based on the Controller Area Network (CAN) communications protocol. The three buses are as follows:

- High-speed bus - typically used for sharing real time data such as driver-commanded torque, actual engine torque, steering angle, etc.
- Mid-speed bus - typically used for infotainment applications (display, navigation, etc.) where the system response time demands that a large amount of data be transmitted in a relatively short amount of time, such as updating a graphics display.

- Low-speed bus - typically used for operator controlled functions where the system response time requirements are of the order of 100 ms to 200 ms.

Use of a given bus in a vehicle depends upon how feature/functions in that vehicle are partitioned among different ECUs. Figure 2 is an illustration of in-vehicle networking using the high-speed link and the low-speed link. It should not be construed as actual vehicle wiring.



- G1 - gateway between GMLAN networks
e.g. between low speed network and high speed network
- G2 - gateway to a non-GMLAN protocol used for diagnostic functions
e.g. between GMLAN and KW2000 protocol
- G3 - gateway to a non-GMLAN protocol used for vehicle functions
e.g. between GMLAN and an industry developed entertainment bus
- G4 - gateway to provide remote access to GMLAN
e.g. between GMLAN and Onstar service center

Figure 2: GMLAN Buses

3.3 Communication. With respect to the International Organization for Standardization (ISO)/Open System Interconnection (OSI) reference model, communication is based on a layer model and consists of the physical layer, data link layer, transport layer, node management layer, network management layer and the interaction layer.

This illustration (Figure 3) is intended to describe the hierarchy of a communication kernel.

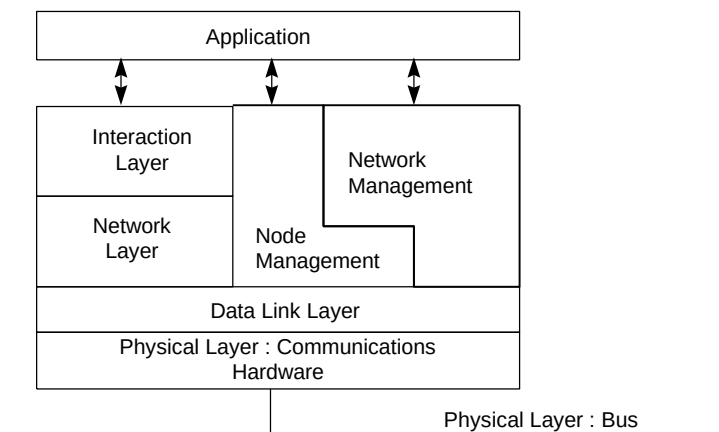


Figure 3: GMLAN Communication Kernel Concept Model

3.3.1 Interaction Layer. The interaction layer serves as the application program interface. It provides communication services to the application independent of the bus protocol and uses services of the transport layer for communicating on the network.

3.3.2 Network Layer. The network layer transfers data packets up to 4095 bytes using services of the data link layer. It provides an unacknowledged transport service (1:N communication, 1:1 communication). If long messages have to be transferred over the network, the transport layer provides services for segmented data transfer. On the transmit side, the message is broken into segments, each segment being short enough to fit in a CAN frame. On the receive side, individual segments are assembled into a message.

The network layer also provides connect setup and flow control.

3.3.3 Data Link Layer. The data link layer provides services for the transfer of individual frames between nodes. It handles the protocol of the bus system (bit timing, arbitration, error detection, etc.)

3.3.4 Node Management Layer. The node management layer is used by the node to control the start-up, shutdown and error handling for a node where these functions do not involve interaction with the other nodes of the network and, therefore, can be managed locally.

3.3.5 Network Management Layer. The network management layer is used to control the start-up, shutdown and error handling for a network of nodes, where these functions involve interaction among the nodes and, therefore, have to be managed globally.

3.3.6 Physical Layer. The physical layer provides a method for converting the digital data symbols (1's and 0's) generated by the data link layer into electrical signals transmitted on the communications medium. The Physical Layer includes the following.

- Transceiver: Electrical circuit that connects the protocol controller to the bus.
- Bus Loading: The electrical load, resistance and capacitance, on the bus line(s).
- Filter: Inductor to improve Electromagnetic Interference (EMI) immunity.
- Wiring: Physical media used to send the electrical signal.

3.4 Gateway Overview.

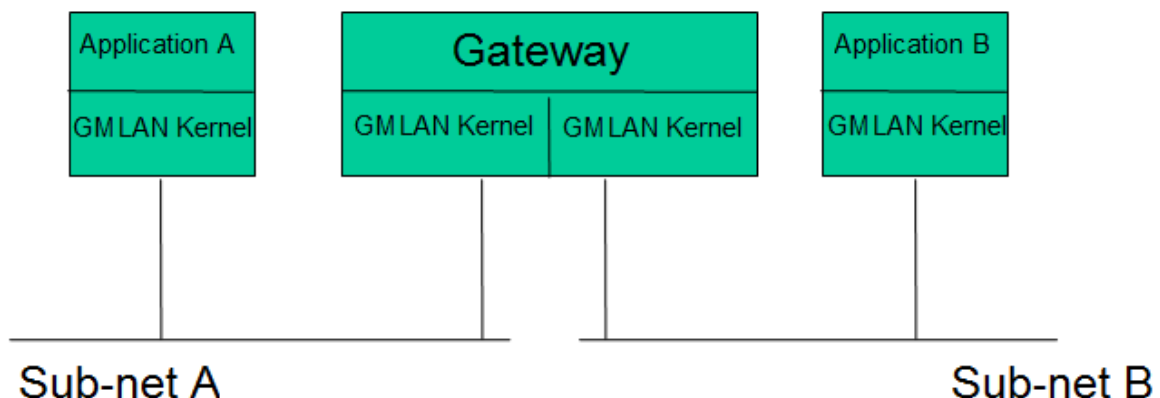


Figure 4: GMLAN Gateway Principal Overview

GMLAN consists of 1 to 3 subnets, the High-speed Bus, the Low-speed Bus, and the Mid-speed Bus. Despite the fact that networks are designed such that the bulk of the signals will stay in their local network, some signals will have to be transferred outside the local network. Gateway nodes perform this function.

A Gateway node is connected to at least 2 buses and will interact with each network according to its message strategy and transmission models (see Figure 4). These Gateway nodes normally carry out several application tasks where the Gateway is merely one of them.

The Gateway interacts with the bus kernels' Interaction Layers (IL) and performs some or all of the following services:

- Transfer wake up requests to other networks;
- Transfer Virtual Network (VN) information to other networks;
- Transfer signals between networks;
- Support signal parameter processing (event generation, data conversion, data filtering, etc.);
- Transfer block information with Unacknowledged Segmented Data Transfer (USDT) protocol.

4 Requirements

Note: Numerous requirements in this section rely on the concept of timers. For the purposes of these requirements, timers should be considered to be "count-down" timers that halt at a value of zero. The terms "set" and "reset" will be used to describe the operation of establishing a timer value, and the term "clear" will be used to describe the operation of setting the timer to a value of zero. The term "halt" describes the timer operation which causes the timer to freeze at its current value. The actual timer maintenance and update processes are not specified.

4.1 Interaction Functions. Communication between nodes on the GMLAN shall consist of standard frames governed by approved interface specifications, and of vehicle specific frames consisting of approved standard signals.

4.1.1 Message Transmission Models. Communication between nodes on the GMLAN shall consist exclusively of the message transmission models described in the following sections. Signals are associated with Virtual Networks and they shall only be transmitted while the Virtual Network is active.

4.1.1.1 Periodic Transmit Model.

4.1.1.1.1 Periodic messages shall be transmitted at a fixed repetition rate (see Figure 5) while an associated Virtual Network is active.

4.1.1.1.2 Periodic message interval accuracy shall be within $\pm 10\%$ of the repetition rate measured with the device in isolation.

Note: Nodes should spread their periodic messages to avoid burst situations, if required by the Component Technical Specification (CTS).

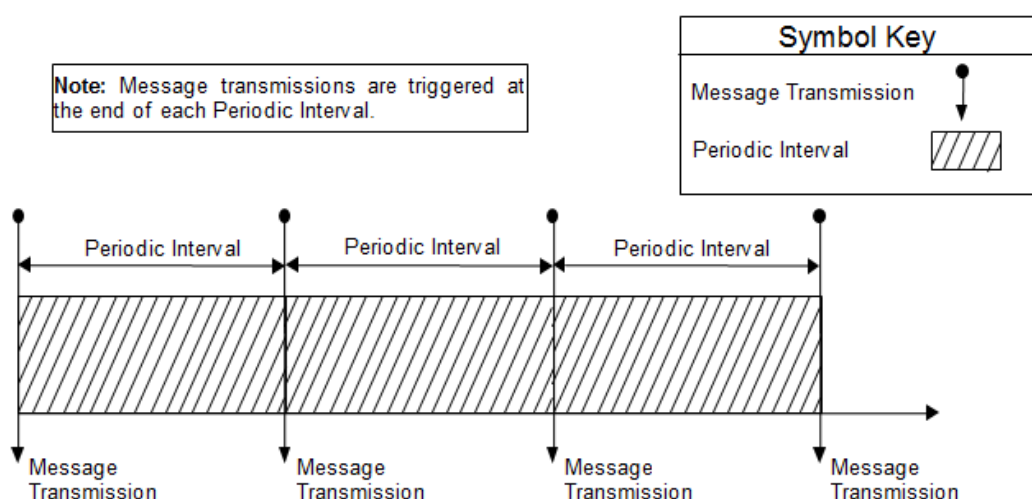


Figure 5: Periodic Transmit Model

4.1.1.2 Event Transmit Model.

4.1.1.2.1 Event messages shall be sent in response to an internal or external event. See Figure 6.

Note: In some cases, the application will queue the event messages at periodic intervals while an enabling condition persists. Though the resulting pattern is often referred to as “Periodic-when-Enabled”, it is not a fundamentally different transmit model from the Event Transmit Model.

4.1.1.2.2 Consecutive transmissions of the same message shall be separated by a specified minimum time known as the Min Update Time.

Note: It is important to note that the Min Update Time is applied on a message-by-message basis. Sometimes multiple signals in a single message have launch criteria associated with them, and immediately following a message transmission satisfying one signal’s launch condition a different signal meets a launch condition. The Min Update Time should be observed between these two successive transmissions. A similar circumstance can happen with a single signal, if it has multiple launch criteria. In particular, this is possible if a signal is contained in a single message that supports two different Virtual Networks.

4.1.1.2.3 All message transmission triggers which occur before the Min Update Time has elapsed shall be satisfied with a single transmission at the end of the update time.

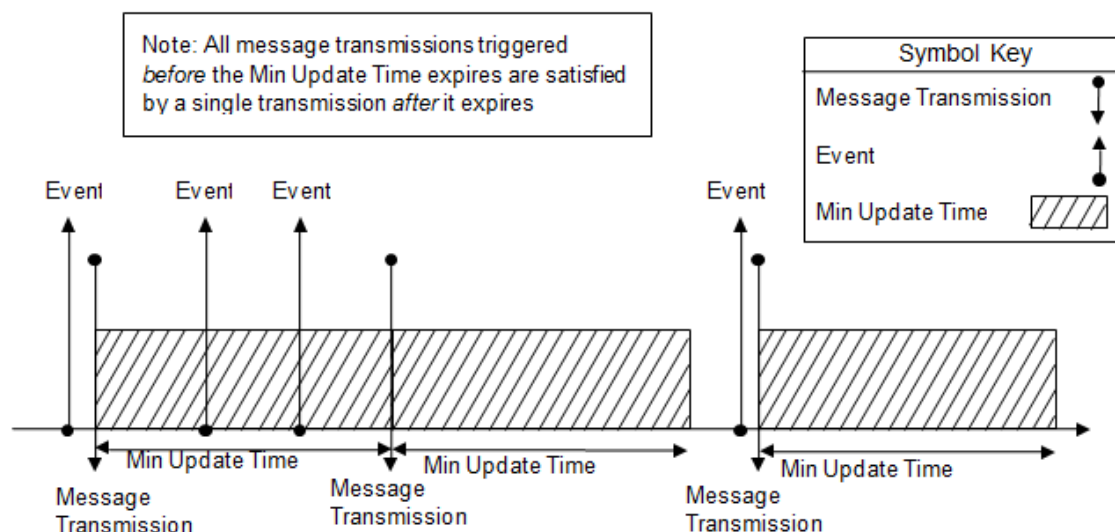


Figure 6: Event Transmit Model

4.1.1.3 Periodic with Event Transmit Model. The Periodic with Event Transmit Model consists of an Event Transmit Model superimposed on a Periodic Transmit Model except that the Min Update Time is enforced between two consecutive transmissions, regardless of which of the superimposed models triggers it. (See Figure 7).

4.1.1.3.1 Consecutive transmissions of the same message shall be separated by a specified minimum time known as the Min Update Time.

4.1.1.3.2 All message transmission triggers which occur before the Min Update Time has elapsed shall be satisfied with a single transmission at the end of the update time.

4.1.1.3.3 The Min Update Time shall be less than the period minus its negative tolerance.

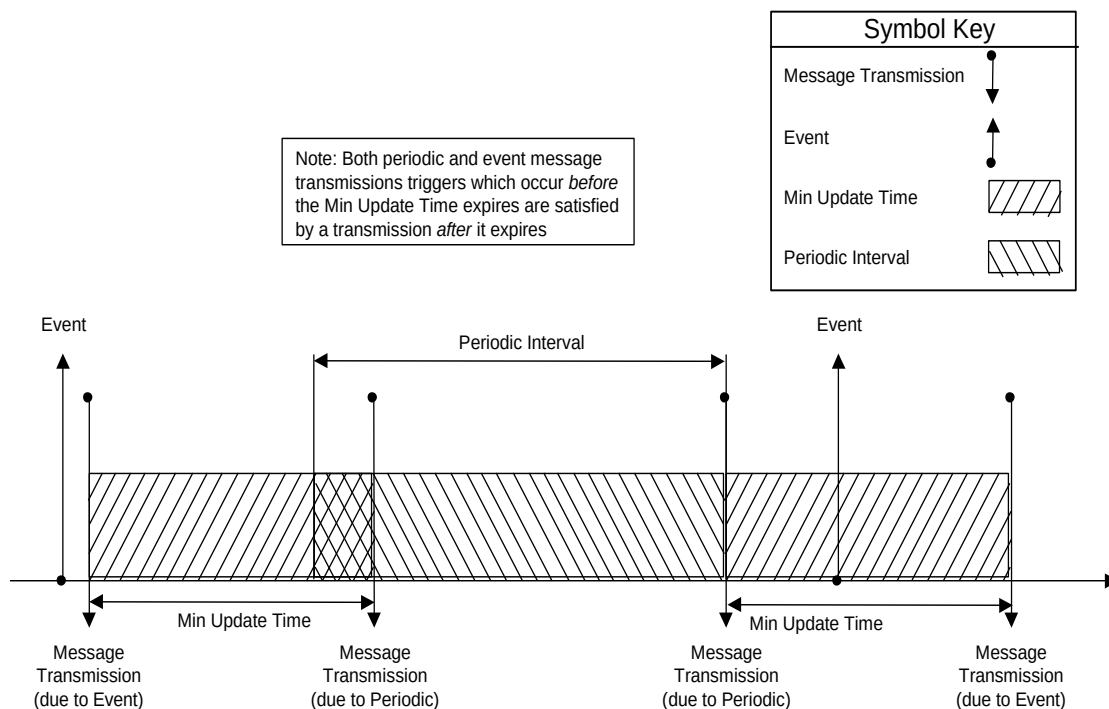


Figure 7: Periodic with Event Transmit Model

4.1.2 Signal Data Sampling.

- Message transmissions must include the most recently available signal data.
- Multiple byte signals should be retrieved without the possibility of an interrupt occurring between retrieval of each individual byte. The GMLAN handler shall allow application to retrieve/store the constituent signals of a packet signal individually (multiple signal access macros) or the entire packet (one signal access macro).
- Functions whose operation requires transmission of all state transitions must make sure that all state transitions are available sequentially.

Note: This requires a confirmation to be communicated to the application after each successful transmission.

4.1.2.1 Short Data Field Recovery.

- ECUs shall discard messages that contain fewer data bytes than expected, without storing the data, notifying the application of a reception, or resetting the signal supervision timer.

Note: This does not pertain to the CAN frame's Data Length Code (DLC). This pertains to the situation where a valid CAN frame containing a valid DLC is received containing fewer data bytes than the receiving ECU is configured to accept. From the receiving ECUs perspective, such messages are logically "corrupted", thus the requirement is to discard them as if they were never received.

4.1.2.2 Long Data Field Recovery.

- A node shall accept a valid frame with more data bytes than expected, but shall use only the amount of data that was expected.

Note: In this context a "valid" frame is one which is received without any bit timing, framing, or Cyclic Redundancy Check (CRC) errors.

Note: This is done so that in subsequent model years, additional signals may be added to a transmitted message without forcing devices which receive the message to make software changes if they do not actually consume the added signal data.

4.1.3 Receiver Capability.

Specified device behavior must continue during a burst of transmissions consisting of frames transmitted within their prescribed limits. Therefore, a device shall react as required by the governing CTS to a sequence of frames which:

- a. Includes all periodic frames the device receives, transmitted at their required transmission intervals,
- b. Includes all event frames the device receives, transmitted at their (minimum) update intervals,
- c. Includes sufficient additional frames not received by the device to ensure that all frames in the sequence are separated only by the CAN interframe separation time, and
- d. Is sufficiently long in time duration to include four repetitions of the least frequently transmitted message from a or b, above.

Note: This requirement introduces enough “additional” frame traffic to keep the bus at 100% utilization for the duration of the sequence.

Note: By requiring four repetitions it is guaranteed that a sequence of two transmissions will be preceded by, and followed by, a period of 100% bus utilization. This removes the possible test biases that might result if the least frequently transmitted message were only transmitted as the first and/or last transmissions of the sequence.

If an overrun situation occurs the overrun information from the CAN controller shall be reported to the application if so specified in the CTS of the ECU.

Note: The overrun information may be used for diagnostics.

4.1.4 Signal Supervision. Signal Supervision is the process monitoring signal transmissions to determine if expected signal receptions are no longer occurring. Its primary purpose is to allow reasonable default values to be substituted for the signals no longer received. A calibratable delay called Supervision Stability Timer shall be implemented in the GMLAN handler that must expire prior to the handler beginning supervision of signals.

4.1.4.1 Self-supervised Signals. A Self-supervised signal is sent with a Periodic or Periodic with Event transmit model. A timeout value greater than two times the signal's periodic interval shall be associated with a received signal through the receiving ECU's CTS. The self-supervised signal is considered failed by the receiving ECU when the signal is not received for a time greater than or equal to its timeout value.

4.1.4.2 Supervising Signals with Other Signals. A receiving device may supervise a group of signals by supervising a single signal designated to convey the supervision status of the group. The group is referred to as the supervised signals, and the single signal that represents (i.e., supervises) the group is referred to as the supervising signal. Periodic signals may be used to supervise other periodic signals. Signals may be supervised with other signals in two ways, by presence and by value.

Note: Event only signals cannot be self-supervised since they are not periodic. They must be supervised using another supervising signal that is periodic or by using supervision with Source ID learning.

4.1.4.2.1 Signal Supervision by Presence. The Signal Supervision by Presence process determines whether a received signal has failed by monitoring another periodic signal transmission whose “heartbeat” serves as an indication that the supervised signal has not failed. A supervision timeout value greater than two times the supervising signal's periodic interval is associated with the supervising signal through the receiving ECU CTS. The supervised signals are considered failed by the receiving ECU when the supervising signal is not received for a time greater than or equal to its timeout.

4.1.4.2.2 Signal Supervision by Value. Signal Supervision by Value is the process of determining whether a supervised signal has failed (or not) based on the received value a designated Boolean supervising signal. If a specific failure indication value (as defined in the ECU CTS) is received for the supervising signal, the supervised signals are determined to have failed.

Note: A signal used to supervise other signals by its value, can also fail. For this reason, the supervising Boolean signal should, itself, be supervised. Generally, when supervision of this signal fails, the signals that it supervises should also be considered to have failed.

4.1.4.3 Supervising Signals with Source ID Learning. If extended CAN identifiers are used, the GMLAN handler may supervise a signal by learning the Source ID of the transmitting ECU from the CAN identifier for the message containing the supervised signal. This learned Source ID is relearned each time the message is received. The supervised signal is considered failed by the receiving ECU when no messages from the learned Source ID have been received by the ECU for a time greater than or equal to its timeout value.

4.1.5 Unsupervised Signals. An unsupervised signal is one where no failsofting or error recovery is required. Typically these are signals which fall into one of the following categories:

- a. Operator in the loop (ones where an operator causes the event which triggers a signal);
- b. Signals that have no defined change trigger. For instance, a signal indicating transmission type, which is, sent once during an ignition cycle, but which cannot change;
- c. Signals for which the last received value or the power up default is the desired substitute value when the signal is not received.

4.1.6 Data Failsofting. Data failsofting is the procedure used by a node to adopt a safe value when a monitored signal fails. For the purposes of failsofting, a signal is classified as failed if one of the following events occurs:

- a. A data frame is received with an indication that the transmitter's ability to send valid data for a set of signals has been compromised (e.g., an indication that a sensor has failed);
- b. Supervision indicates that a signal has failed.

Note: The set may consist of a single signal.

Note: Gateways use this mechanism to indicate that an ECU on another subnet has failed and that all signals from that ECU should be failsofted.

4.1.7 Invalid Messages. An ECU shall discard, without processing, any received messages that it does not explicitly support. Specifically, the following frames shall be discarded:

- a. Remote frames not explicitly supported, and,
- b. Extended Identifier frames not explicitly supported.

4.2 Node Management Overview. Node Management is the process of maintaining the node in a set of predefined communication states. Node Management responsibilities include:

- a. Node start up and shut down.
- b. Communication Kernel start up and shut down.
- c. Recovery from node-detected communication error conditions.
- d. Management of special application-controlled communication modes.

The following sections briefly describe the desired behavior of GMLAN Node Management in these specific scenarios. Formal requirements are given afterwards.

4.2.1 Node Start Up and Shut Down. Following ECU power up or an ECU reset, the ECU initializes its communication capabilities and remains in a communication capable state for at least 8 seconds. This is done regardless of whether the ECU has detected a local condition which requires it to communicate. It is done so that the ECU can monitor the bus for Network Management messages that indicate whether other ECUs require it to communicate in support of control operations they need to perform.

Of course, the ECU should only shut down when no ongoing control operations rely on its participation.

Note: This is not meant to imply that Network Management frames are the only method for determining when communication is necessary.

4.2.2 Communication Kernel Start Up and Shut Down. GMLAN allows some ECUs to stay in a low-power state with their communication capabilities inactive while other ECUs actively communicate to perform vehicle control operations.

Activating communication on a distributed set of ECUs is the responsibility of Network Management and is based on collections of ECUs which form "Virtual Networks" to perform specific control operations. Typically, one ECU detects a condition that requires some control operations to begin and, therefore, warrants the activation of the appropriate Virtual Network (VN). For that ECU to notify other ECUs that their participation is required, it must first start its own Communication Kernel and then cause the other ECUs on the bus to start their Communication Kernels. All of the ECUs must then keep their Communication Kernels active for a pre-determined amount of time (8 seconds) so that Network Management messages can be transmitted. These messages allow all ECUs to determine if they are being requested to participate (i.e., if a VN they support is being activated). The ECUs forming the VN keep their Communication Kernels active and begin communicating control information. After the required 8 seconds, the other ECUs may shut down their Communication Kernels and revert to a low-power state if they have no other control obligations.

GMLAN supports a bus wake up capability to facilitate this process. When an ECU broadcasts a bus wake up, all ECUs on the subnet are required to start up their Communication Kernels (if they are not already active) and keep them active for 8 seconds.

Note: In this specification, “wake up broadcast” is intended to encompass all wake up mechanisms delineated in GMW3122 and GMW3089.

4.2.3 Node-detected Error Conditions. ISO 11898 describes the error conditions that the CAN hardware detects and can communicate to the ECU. GMLAN Node Management only requires specific corrective actions when the “bus off” condition occurs. When first notified of a bus off, GMLAN Node Management must immediately reset the CAN Data Link Layer (DLL). If the bus off condition continues to re-occur, DLL resets should be performed at ECU-dependent intervals (specified in the CTS).

In addition to this specific bus-off error management strategy, “error passive”, and bus off conditions detected by the DLL are also communicated to the application for application-specific error management.

4.2.4 Special Application-controlled Communication Modes. GMLAN supports two special Communication Kernel operational modes that are entered when requested by the application:

- Normal Communication Halted Mode
- High-speed Mode (Low-speed subnet only)

4.2.4.1 Normal Communication Halted Mode. Some service/diagnostic procedures require that normal (non-diagnostic) communication be halted before they can be executed. This may be done to reduce bus contention, or so that CAN IDs used during normal communication can be temporarily used for diagnostic support without fear of CAN ID conflicts. In GMLAN, the Communication Kernel supports a Normal Communication Halted service that the application can invoke for this purpose. Since halting normal communication disrupts most basic Communication Kernel functionality significantly, normal communication must be resumed by restarting the Communication Kernel.

4.2.4.2 High-speed Mode (Low-speed Subnet Only). GMLAN supports a special high-speed mode on the low-speed subnet. Using service/diagnostic messages, the low-speed subnet can be placed in a high-speed mode that permits high-speed communication for service/diagnostic purposes. High-speed mode is not permitted for normal communication between on-board ECUs. The protocol for requesting an ECU to enter high-speed mode is documented in GMW3110. However, after the protocol is properly executed, individual ECUs enter high-speed mode by invoking an Enter High-speed Mode service provided by the Communication Kernel. Similarly, the Exit High-speed Mode service is used to exit high-speed mode.

Note: High-speed mode is used primarily for ECU programming.

4.3 Node Management Requirements. The desired behavior of Node Management and its relationship to the rest of the ECU logic are graphically depicted in this document through the use of State Transition Diagrams (STD). Notational conventions for the state transition diagrams are given in Appendix A.

Sections are titled to correspond to the states in the STDs. Individual requirements are documented within the section (i.e., state) that is primarily responsible for implementing the requirement. If a requirement is implemented by a combination of states, cross-references are made to the others. Requirements that correspond to transitions between states are documented in the state from which the transition originates. Corresponding to each requirement is a “Figure Reference” which explains how and where the requirement is graphically represented. Within each section (state), other sections containing requirements that are partially implemented in this state’s STD are enumerated under “Related Requirements”.

Figure 8 contains a high-level ECU STD that encompasses all of the sub-states of the ECU, including Node Management.

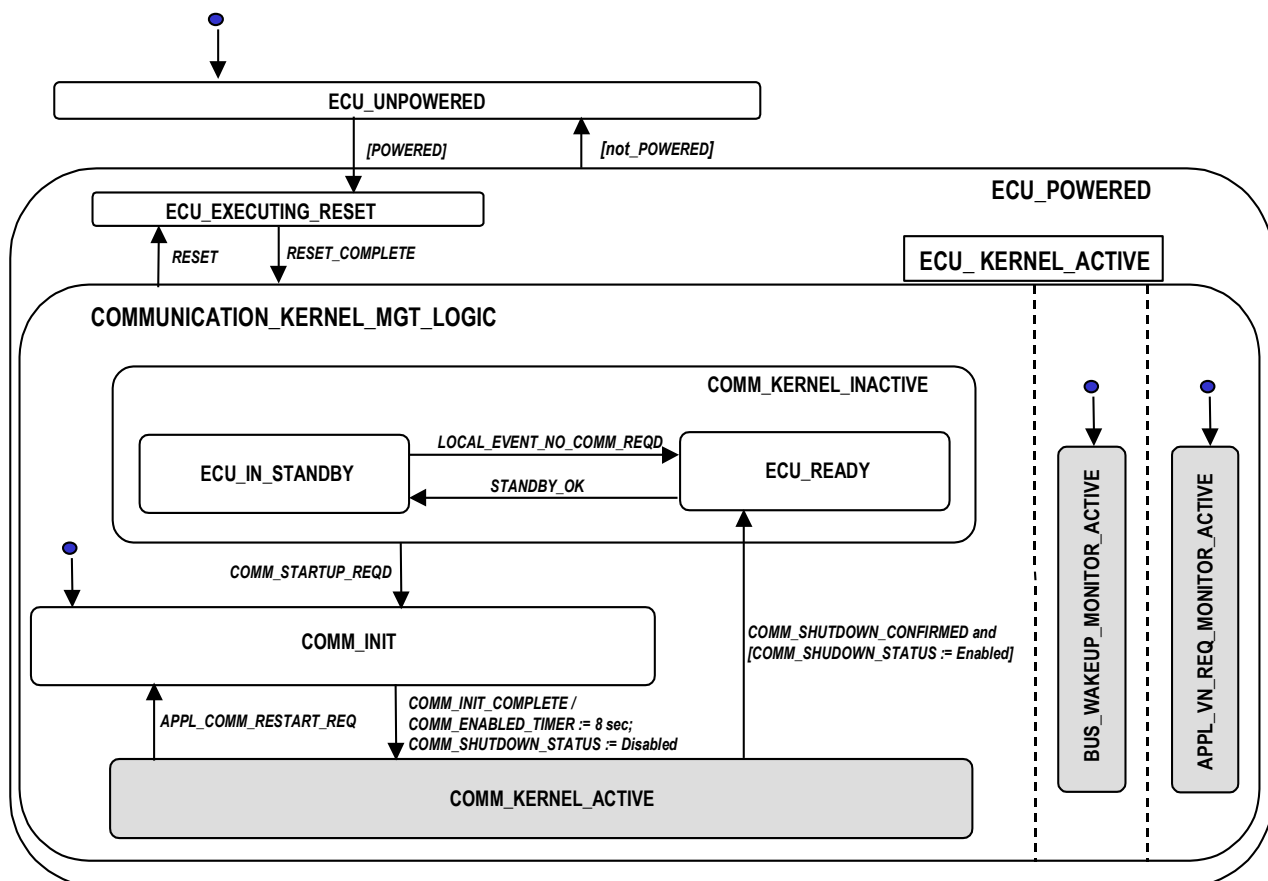


Figure 8: Primary Node Management States

4.3.1 ECU_UNPOWERED State. This is the initial state of the ECU. In this state the ECU has no power applied.

Requirement: Upon application of power the ECU shall execute its reset logic.

Figure Reference: The [POWERED] transition connects the ECU_UNPOWERED state and ECU_EXECUTING_RESET sub-state of the ECU_POWERED state. See Figure 8.

Related Requirements: Not applicable.

4.3.2 ECU_POWERED State. In this state the ECU is powered. This state decomposes hierarchically into two sub-states, ECU_EXECUTING_RESET and ECU_KERNEL_ACTIVE. Regardless of sub-state, when power is removed the ECU will return to the ECU_UNPOWERED state.

4.3.2.1 ECU_EXECUTING_RESET State. The ECU_EXECUTING_RESET state is a transient state. The ECU resides in this state only long enough to execute its reset logic.

Requirement: Following the execution of the ECUs reset logic the ECU shall initialize the ECU Kernel.

Figure Reference: The RESET_COMPLETE transition connects the ECU_EXECUTING_RESET state and the ECU_KERNEL_ACTIVE state. See Figure 8.

Related Requirements: Not applicable.

Other detailed requirements for behavior in the ECU_EXECUTING_RESET state are ECU-dependent and beyond the scope of this document.

4.3.2.2 ECU_KERNEL_ACTIVE State. Following execution of its reset logic, the ECU must transition to a state where it is able to detect state changes in its inputs. In this state, the ECU's basic low-level software (ECU Kernel) is executing. Detailed requirements for the behavior of the ECU Kernel are ECU-specific and beyond the scope of this document. However, GMLAN places some minimal requirements on the ECU Kernel.

At a minimum, the ECU Kernel must:

- Detect bus wake up broadcasts,
- Service application requests to start Virtual Networks,
- Start the Communication Kernel when bus wake up is received or a Virtual Network activation is requested, and,
- Shut down the Communication Kernel when the application confirms that communication capabilities are no longer needed.

The ECU Kernel is active if, and only if, the ECU is in the ECU_KERNEL_ACTIVE state. The ECU_KERNEL_ACTIVE state consists of three concurrent activities (see Figure 8) whose logic addresses the above needs:

- BUS_WAKEUP_MONITOR_ACTIVE is responsible for detecting received bus wake ups,
- APPL_VN_REQ_MONITOR_ACTIVE is responsible for processing application requests to start Virtual Networks, and,
- Communication Kernel Management Logic causes the ECU to start and stop its GMLAN Communication Kernel when required.

The combined logic of these three activities cooperate to determine when the Communication Kernel start up or shut down is required, and to perform the actual start up and shut down procedures. The three activities are detailed below.

Requirement: The following Node Management requirement applies to ECU_KERNEL_ACTIVE (i.e., to all sub-states):

- When an ECU reset occurs while the ECU Kernel is active, the ECU shall execute its normal power up reset logic.

Figure Reference: The RESET transition from ECU_KERNEL_ACTIVE state to ECU_EXECUTING_RESET is independent of sub-state. See Figure 8.

Related Requirements: Not applicable.

4.3.2.2.1 BUS_WAKEUP_MONITOR_ACTIVE State. Bus wakeups are broadcast by an ECU to cause all ECUs on the same subnet to initialize their Communication Kernels. Since synchronous detection of bus wake ups by all ECUs is important to an error-free start-up of distributed control functionality, it is essential that the reaction to wake up broadcasts be consistent across all ECUs in the vehicle. In particular, the ECU that broadcasts the wakeup reacts to its own broadcast just as other ECUs do. In addition, on the low and mid-speed buses, both transmitter and receiver of the wakeup message (ID \$100) react to the broadcast, regardless of whether the broadcast is performed using the high-voltage level or the normal level. The BUS_WAKEUP_MONITOR_ACTIVE logic (see Figure 9) is responsible for detecting these broadcasts and reacting to them properly.

Note: Since GMLAN supports several bus wakeup mechanisms, the specific mechanisms applicable to an ECU are specified in the ECU's CTS.

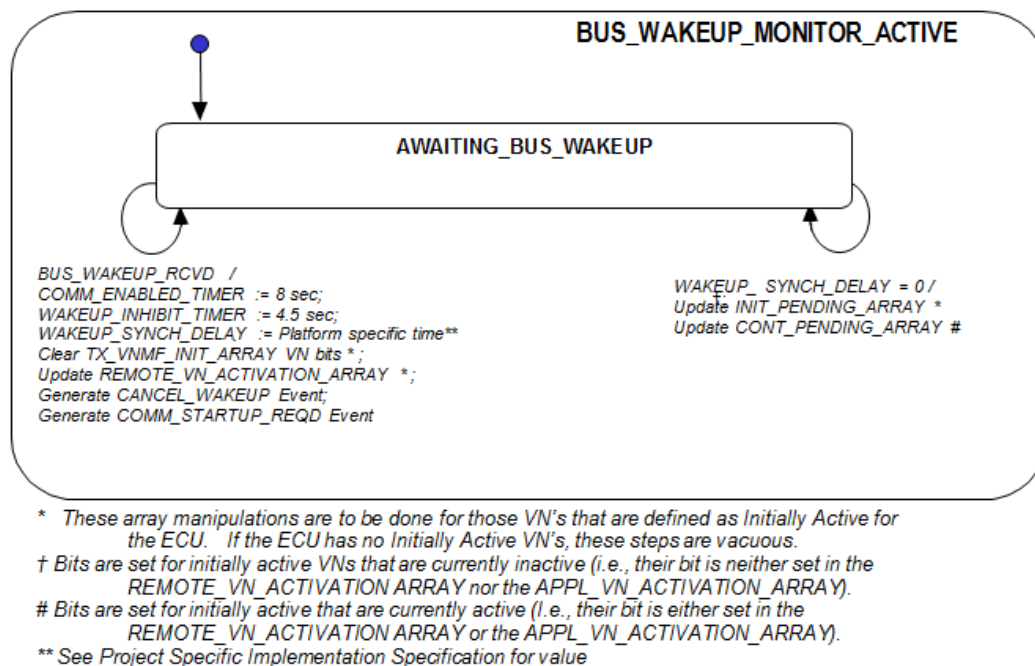


Figure 9: Bus Wake Up Monitoring Logic

Requirements: Most of the tasks performed by BUS_WAKEUP_MONITOR_ACTIVE are related to Network Management. However, the following Node Management requirements apply to BUS_WAKEUP_MONITOR_ACTIVE:

- The ECU shall detect all bus wake up broadcasts while the ECU Kernel is active.

Figure Reference: The AWAITING_BUS_WAKEUP state is designated as the default (initial) sub-state of the BUS_WAKEUP_MONITOR_ACTIVE state (see Figure 9) which is itself a default sub-state of the ECU_KERNEL_ACTIVE state. See Figure 8. This indicates that monitoring for bus wake ups (the BUS_WAKEUP_RCVD event) will begin when the ECU_KERNEL_ACTIVE parent state is entered.

- Upon receipt of a bus wake up, the ECU shall initialize its Communication Kernel if it is not already active.

Figure Reference: When a bus wake up is received (BUS_WAKEUP_RCVD event) the COMM_STARTUP_REQD event is generated. See Figure 9. The Communication Kernel management logic will react to COMM_STARTUP_REQD by starting the Communication Kernel (if it is not already active). See Figure 8.

- Upon receipt of a bus wake up, the ECU shall cause an active Communication Kernel to remain active for at least 8 seconds.

Figure Reference: When a wake up is received, the COMM_ENABLED_TIMER is set to 8 seconds. Expiration of this timer is prerequisite for requesting the application to confirm the exit from COMM_KERNEL_ACTIVE state. See Figure 11.

- Upon receipt of a bus wake up, the ECU shall inhibit transmissions of bus wake ups for 4.5 seconds.

Note: Though all ECUs are required to inhibit wake up broadcasts following the reception of a wake up, multiple wake ups may still be received. This stems from the fact that although ECUs are required to cancel pending wake up messages in a Data Link Layer queue, ECUs are not required to flush queued wake up messages from their CAN Controllers. Consequently, multiple wake ups queued on different nodes at approximately the same time could result in several wake ups reaching their respective CAN Controllers before the first wake up is broadcast.

Figure Reference: When a wake up is received, the WAKEUP_INHIBIT_TIMER is set to 4.5 seconds in Figure 9. This timer is used in Network Management to inhibit wake up transmissions. See Figure 19.

Related Requirements: See Sections: 4.5.2.1 COMM_KERNEL_ACTIVE State, 4.5.3.1 Requirements for Network Activated Virtual Networks, and 4.5.3.1.1 VNMF and Wake Up Broadcast Requirements for Network Activated Virtual Networks.

4.3.2.2.2 APPL_VN_REQ_MONITOR_ACTIVE State. When an ECU detects conditions that require it to begin a distributed control operation, it must establish communication capabilities between itself and the other ECUs participating in the operation. Such collections of cooperating ECUs form a Virtual Network (VN). Network Management is responsible for starting, stopping and monitoring VNs. However, Node Management supports Network Management by starting the Communication Kernel whenever the ECU's control application requests that a VN be started.

The APPL_VN_REQ_MONITOR_ACTIVE logic is responsible for servicing the application's VN activation and deactivation requests.

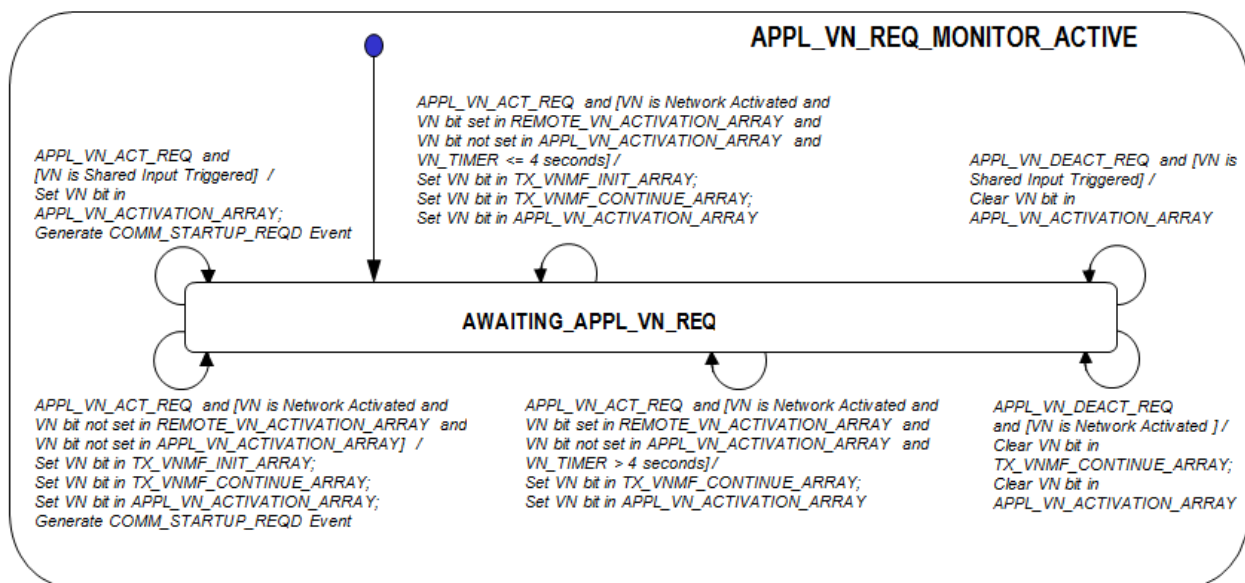


Figure 10: Virtual Network Activation/Deactivation Request Monitoring Logic

Requirements: Most of the tasks performed by APPL_VN_REQ_MONITOR_ACTIVE are related to Network Management. However, the following Node Management requirements apply to APPL_VN_REQ_MONITOR_ACTIVE:

- The ECU shall service all application VN activation and deactivation requests while the ECU Kernel is active.

Figure Reference: The Awaiting_APPL_VN_REQ state is designated as the default (initial) sub-state of the APPL_VN_REQ_MONITOR_ACTIVE state (see Figure 10) which is itself a default sub-state of the ECU_KERNEL_ACTIVE state. See Figure 8. This indicates that monitoring will begin when the ECU_KERNEL_ACTIVE parent state is entered.

- Upon receipt of a VN activation request, the ECU shall initialize its Communication Kernel if it is not already active.

Figure Reference: The Communication Kernel management logic will react to COMM_STARTUP_REQD by starting the Communication Kernel (if it is not already active). See Figure 8. Referring to Figure 10, there are four possible transitions that react to the APPL_VN_ACT_REQD event, which represents the activation request. Two (2) of the occurrences explicitly generate the COMM_STARTUP_REQD event (the two left-most transitions). The remaining two (the two center transitions) are predicated on "VN bit set in REMOTE_VN_ACTIVATION_ARRAY". Since this implies that the VN is already active, it also implies that the Communication Kernel is already active. Since these two transitions can only occur while the Communication Kernel is active, they need not start the Communication Kernel. Since the four possible activation requests either start the Communication Kernel or only occur while it is active, the requirement is met.

© Copyright 2013 General Motors Company All Rights Reserved

Related Requirements: See Sections: 4.5.2.1.2.1 COMM_ENABLED State, 4.5.2.1.2.2 COMM_ACTIVE State, 4.5.3.1.1 VNMF and Wake Up Broadcast Requirements for Network Activated Virtual Networks, 4.5.3.1.1.1 VNMF Transmission Timing on the Mid and Low-speed subnets, and 4.5.3.2 Requirements for Shared Input Activated Virtual Networks.

4.3.2.2.3 Communication Kernel Management Logic. The Communication Kernel management logic is responsible for starting and stopping the Communication Kernel as required. This logic contains three high-level states:

- COMM_KERNEL_INACTIVE, in which the Communication Kernel is not functional,
- COMM_INIT, in which the Communication Kernel is in the process of being initialized, and,
- COMM_KERNEL_ACTIVE, in which the Communication Kernel is functional.

This logic causes the ECU to initialize the Communication Kernel when required following a bus wake up or an application VN activation request. It also deactivates the Communication Kernel when the communication capabilities are no longer required and the application confirms that deactivation is to be permitted.

4.3.2.2.3.1 COMM_KERNEL_INACTIVE State. When an ECU detects that it no longer needs to communicate, the Communication Kernel can be shut down. The ECU resides in the COMM_KERNEL_INACTIVE state when the Communication Kernel is inactive.

The decision that communication is not needed involves the status of Virtual Networks and is made by Network Management. Once this decision is made, the application is requested to confirm the shutdown of the Communication Kernel. If this confirmation is given, the Communication Kernel must be shut down.

Note: If the application has knowledge of extenuating circumstances it may choose to keep the Communication Kernel active for an extended period of time. One (1) use of this capability is to extend the amount of time the Communication Kernel stays active following detected bus errors so that the ECU has an extended opportunity to detect if VNs are active that the ECU supports.

Requirement: (Applies to COMM_KERNEL_INACTIVE)

- The ECU shall begin initializing their Communication Kernel when a bus wake up is received or the application requests that a VN be activated.

Figure Reference: The COMM_STARTUP_REQD transition connects the COMM_KERNEL_INACTIVE state to the COMM_INIT state. See Figure 8. It is generated in the BUS_WAKEUP_MONITOR_ACTIVE and APPL_VN_REQ_MONITOR_ACTIVE states.

Related Requirements: Not applicable.

The COMM_KERNEL_INACTIVE state contains two sub-states.

- ECU_READY, in which the ECU is performing control operations which do not require communication, and
- ECU_IN_STANDBY, in which the ECU is not performing any control operations and can enter a low-power consumption state.

Transitions between these two sub-states are governed by ECU-specific requirements that are captured in the CTS. Generically speaking, the ECU transitions from ECU_IN_STANDBY to ECU_READY when it has control operations to perform that do not require communication and transitions the opposite direction when these operations are no longer necessary. LOCAL_EVENT_NO_COMM_REQD and STANDBY_OK, respectively, represent these state transitions in Figure 8.

4.3.2.2.3.2 COMM_INIT State. COMM_INIT is a transient state. In this state, the ECU is executing its Communication Kernel initialization logic. The ECU must transition from COMM_INIT to COMM_KERNEL_ACTIVE as soon as the initialization logic is executed. The Data Link Layer (DLL) is reset by COMM_INIT. Following COMM_INIT, the ECUs Communication Kernel is active and the ECU communication capabilities are fully functional.

Requirements: (Applies to COMM_INIT):

- Following power up or an ECU reset, ECUs shall initialize their Communication Kernels.

Figure Reference: The COMM_INIT state is designated as a default (initial) sub-state of the ECU_KERNEL_ACTIVE state. See Figure 8. This indicates that initialization will begin when the ECU_KERNEL_ACTIVE parent state is entered.

- Upon completion of the Communication Kernel initialization logic, the ECU shall be capable of communication for a minimum of 8 seconds.

Note: Unless power is removed from the ECU or an ECU reset occurs.

Figure Reference: The COMM_INIT_COMPLETE transition connects the COMM_INIT state to the COMM_KERNEL_ACTIVE state and COMM_ENABLED_TIMER is set to 8 seconds as part of this transition. See Figure 8. Expiration of this timer is prerequisite for requesting the application to confirm the exit from the COMM_KERNEL_ACTIVE state. See Figure 11.

Related Requirements: Not applicable.

4.3.2.2.3.3 COMM_KERNEL_ACTIVE State. In the COMM_KERNEL_ACTIVE state, the ECU is capable of communicating on the bus. It is entered when communication capabilities are required and exited when they are no longer needed. The Communication Kernel is active if, and only if the ECU is in the COMM_KERNEL_ACTIVE state.

Requirements: (Applies to COMM_KERNEL_ACTIVE)

- The ECU shall shut down the Communication Kernel when the application confirms the Communication Kernel's shutdown request, unless a Virtual Network is activated between the request and the confirmation.

Figure Reference: Referring to Figure 11, the event REQ_COMM_SHUTDOWN_CONFIRM represents the Communication Kernel's shutdown request. It is only generated on one transition which emanates from, and returns to, the COMM_ENABLED sub-state of the COMMUNICATION_START_UP_LOGIC. This transition also sets COMM_SHUTDOWN_STATUS to "Enabled". Referring to Figure 8, the application's confirmation of the shutdown is represented by the COMM_SHUTDOWN_CONFIRMED event, which when COMM_SHUTDOWN_STATUS is "Enabled" causes a transition from the COMM_KERNEL_ACTIVE to the COMM_KERNEL_INACTIVE state. This transition represents the shutdown of the Communication Kernel, which is the required behavior. If a Virtual Network is activated following the shutdown request and before the confirmation, the transition from COMM_ENABLED to COMM_ACTIVE in Figure 11 will be taken. Since this transition also sets COMM_SHUTDOWN_STATUS to "Disabled", shutdown can only occur if no Virtual Networks are activated between the request and the confirmation.

Note: Shutdown of the Communication Kernel is then delayed until conditions are met for another request from the Communication Kernel.

- If the Communication Kernel is active and normal communication is halted, the ECU shall re-initialize the Communication Kernel when requested to do so by the application.

Figure Reference: The APPL_COMM_RESTART_REQ transition connects the COMM_KERNEL_ACTIVE state to the COMM_INIT state. See Figure 8.

Related Requirements: See Sections: 4.3.2.2.1 BUS_WAKEUP_MONITOR_ACTIVE State and 4.5.2.1.1 VNMF_MONITOR_ACTIVE State.

Figure 11 depicts additional detail of the COMM_KERNEL_ACTIVE state.