

FlexRay Message Buffers

The Host View of a FlexRay System

▶ September 2006

Bill Rogers and Stefan Schmechtig
IPextreme, Inc.



In a FlexRay node, message buffers are the means by which the application can decouple its message transmission and reception from the actual FlexRay protocol, allowing asynchronous operation of the application with respect to the timing of FlexRay bus.

In this paper, we discuss the implementation, configuration, and use of transmit and receive message buffers in an example FlexRay node.

HIGHLIGHTS

- ▶ Hardware implementation
- ▶ Software configuration
- ▶ Transmit and receive examples
- ▶ Receive FIFOs

ABSTRACT

The message buffer concept essentially enables the application associated with a FlexRay node to:

- Place data to be transmitted into a region of shared memory and know that the corresponding FlexRay frame will be transmitted on the FlexRay bus in its assigned slot.
- Respond to a notification that a valid FlexRay frame has been received and retrieve the corresponding data from its assigned region of shared memory.

Setting up the message buffers in a FlexRay node involves implementing the required hardware and developing application software to configure, control, and monitor the message buffers according to the communication needs of the target system. Planning for these activities requires an understanding of how message buffers work.

TABLE OF CONTENTS

STRUCTURE OF A TYPICAL FLEXRAY NODE	3
REVIEW OF FLEXRAY TIMING HIERARCHY	4
Communication Cycle	5
Static Segment	5
Dynamic Segment	6
Symbol Window	7
Network Idle Time (NIT)	7
Time Units	7
Frame ID	8
MESSAGE BUFFER OVERVIEW	9
TRANSMIT MESSAGE BUFFER SETUP	10
MESSAGE TRANSMISSION	14
RECEIVE MESSAGE BUFFER SETUP	18
MESSAGE RECEPTION	22
RECEIVE FIFO	22
MESSAGE BUFFER LOCKING	23
INTERRUPTS	23
SUMMARY	24

STRUCTURE OF A TYPICAL FLEXRAY NODE

Figure 1 shows the structure of a typical FlexRay node connected in a 5-node FlexRay cluster using a dual-channel bus topology. (Star-based topologies and single-channel implementations are also possible, as are combinations of both in the same cluster. However, the focus of this discussion is on the FlexRay protocol timing and the purpose/implementation of message buffers.)

In a typical FlexRay node, there is:

- An electronic control unit (ECU) and other application-specific hardware. For example, the node may be a sensor or actuator in an automotive system.
- A FlexRay Communication Controller
- A FlexRay bus driver for each implemented FlexRay channel
- Memory that is shared between the application ECU and the FlexRay Communication Controller

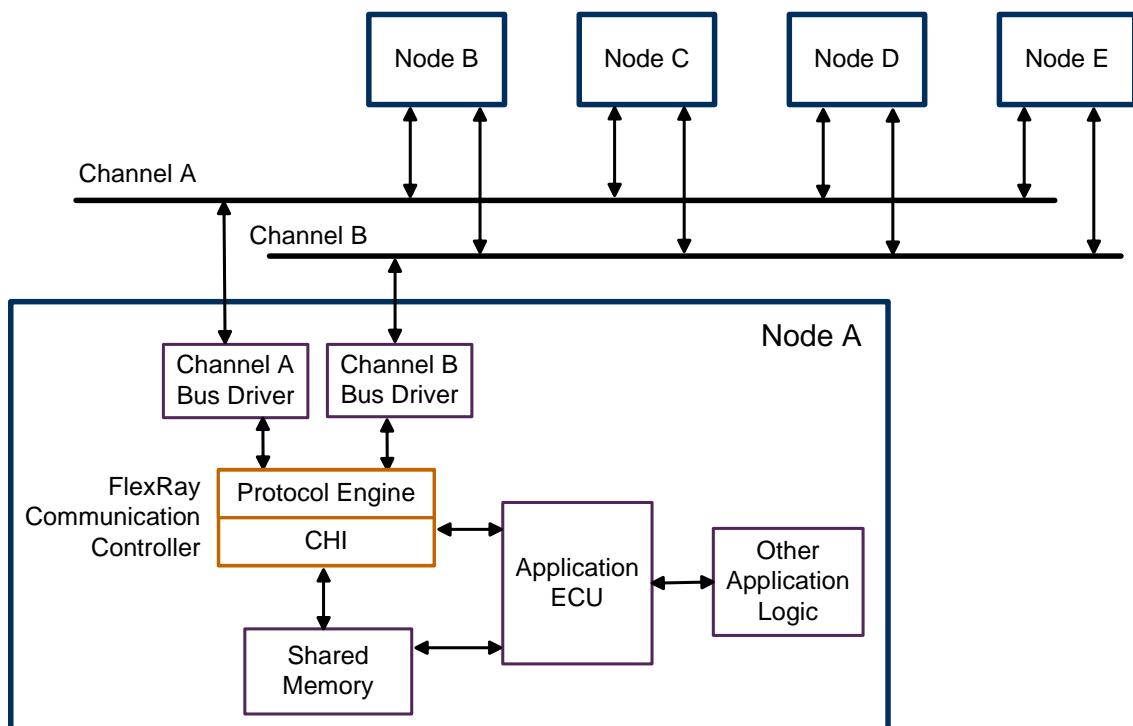


Figure 1. Example FlexRay Cluster

Within the FlexRay Communication Controller, there are two main blocks:

- The FlexRay Protocol Engine implements the majority of the FlexRay protocol including transmitting and receiving frames, maintaining clock synchronization with other nodes in the cluster, and generating and checking CRC.
- The Controller Host Interface (CHI) provides the application with the means to:
 - Configure, control, and monitor the Protocol Engine.
 - Exchange the message data and status between the application and the Protocol Engine. Message exchange is accomplished through a set of message buffers implemented in the node; each message buffer is assigned to a slot in the FlexRay bus timing heirarchy.

The FlexRay Communication Controller must provide the application with the means to configure, control, and monitor the message buffers used for message transmission and reception. A well-designed FlexRay Communication Controller will typically implement the configuration, control, and status registers for the message buffers in the CHI and use a region of the shared memory to store the actual transmit/receive frame header, payload data, and slot status data associated with the configured message buffers.

A FlexRay Communication Controller implemented as a reusable IP block should provide a hardware configuration option to allow you to choose the number of message buffers to implement. This allows you to optimize the size of the core and size of the required shared memory to the number of message buffers that are required for the application.

REVIEW OF FLEXRAY TIMING HIERARCHY

To understand the relationship between message buffers and frames transmitted and received through the FlexRay protocol, it is first important to review the FlexRay timing hierarchy. Figure 2 shows the fundamental timing of the FlexRay protocol.

Note: In the following discussions, the names of FlexRay variables (defined in *FlexRay Communications Protocol Specification, Version 2.1, Revision A*) are shown in *italics*.

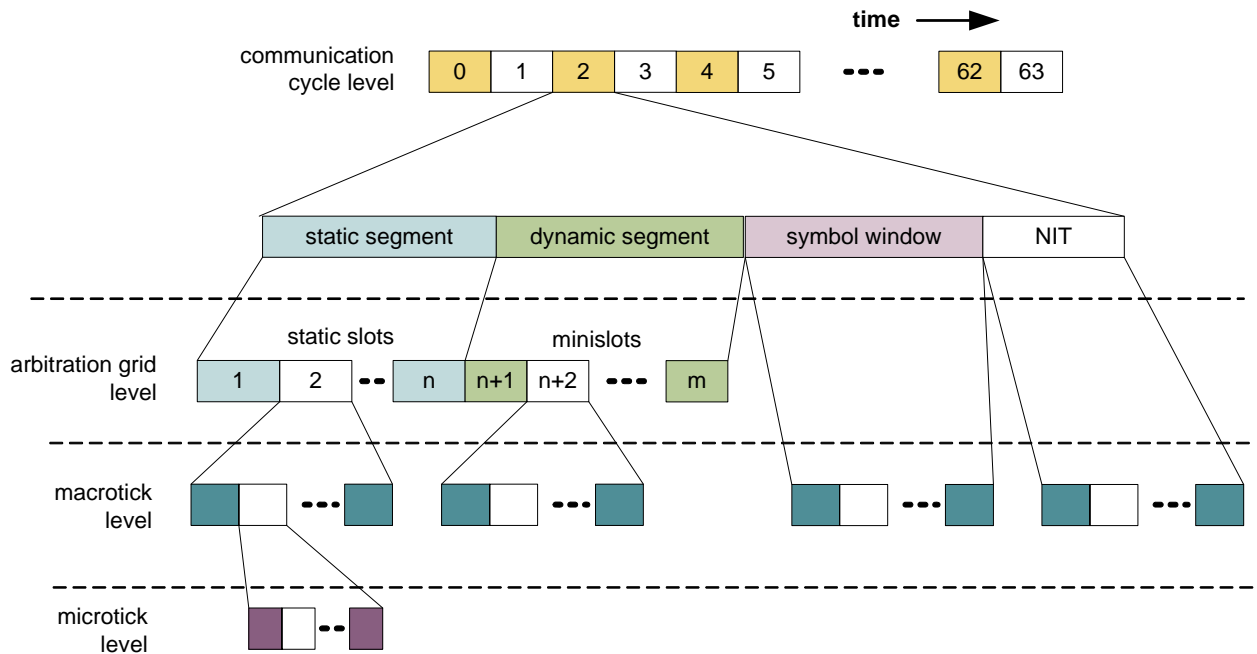


Figure 2. FlexRay Timing Hierarchy

Communication Cycle

FlexRay communication is based on a recurring sequence of 64 communication cycles numbered from 0 to 63.

Each node in the cluster keeps track of the current cycle count independently in a local variable called *vCycleCounter*, which counts from 0 to 63, then resets to 0 and repeats. Although each node tracks the cycle count independently, the current cycle count is the same for all nodes in the cluster.

The duration of a communication cycle is $gMacroPerCycle$ macroticks. $gMacroPerCycle$ is a global parameter for the cluster. Macroticks (defined later in this article) are the common unit of time used throughout the cluster.

Each communication cycle is divided into segments.

Static Segment

The static segment is present in every communication cycle and is used for time-triggered communication (TDMA-based communication scheme based on statically assigned communication slots).

The number of slots in the static segment (static slots) is determined by the cluster-wide configuration parameter *gNumberOfStaticSlots*, which has a minimum value of 2. There are always at least two static slots in the static segment for each channel.

All static slots are of the same time duration, as determined by the cluster-wide parameter *gdStaticSlot*.

Each node in the cluster keeps track of the current slot count in the FlexRay state variable *vSlotCounter* on a per-channel basis (*vSlotCounter* for channel A and *vSlotCounter* for channel B). In the static segment, the current slot count is always the same for channel A and channel B, since the number of slots and slot time duration are the same for each channel.

The payload size for each slot in the static segment (0–254 bytes) is determined by cluster-wide configuration parameter *gPayloadLengthStatic*. Every slot has the same payload length; if a transmitter has less than *gPayloadLengthStatic* two-byte words to transmit in a static slot, it can use padding to fill the remaining bytes in the transmitted frame.

Dynamic Segment

The dynamic segment is optionally present in the communication cycle and is used for ad-hoc, event-driven communication. In the dynamic segment, there are a configurable number of minislots as determined by cluster-wide configuration parameter *gNumberOfMinislots*. If *gNumberOfMinislots* is 0, there is no dynamic segment in the communication cycle. Numbering of minislots in the dynamic segment continues from the last slot number in the static segment. For example, if the last static slot is slot number 9, then the first minislot in the dynamic segment is minislot number 10.

All minislots are of the same number of macroticks as determined by cluster-wide parameter *gdMinislot*.

A dynamic communication slot occurs only if communication begins within a minislot in the dynamic segment. The width of a dynamic slot depends on the size of the payload in the associated frame. The maximum payload length in the dynamic segment is also 254 bytes. However, the actual transmitted payload length can vary from one dynamic slot to another.

Symbol Window

The symbol window is optionally present in the communication cycle and is used to transmit FlexRay-defined symbols. The duration of the symbol window is determined by cluster-wide parameter *gdSymbolWindow* (measured in macroticks). If *gdSymbolWindow* is 0, there is no symbol window in the communication cycle.

Network Idle Time (NIT)

The NIT is used by each node to calculate and apply clock correction. The duration of the NIT in macroticks is the communication cycle time minus the number of macroticks used by the static segment, dynamic segment, and symbol window.

The FlexRay timing parameters mentioned above are cluster (global) parameters with values that are statically set when the cluster is configured.

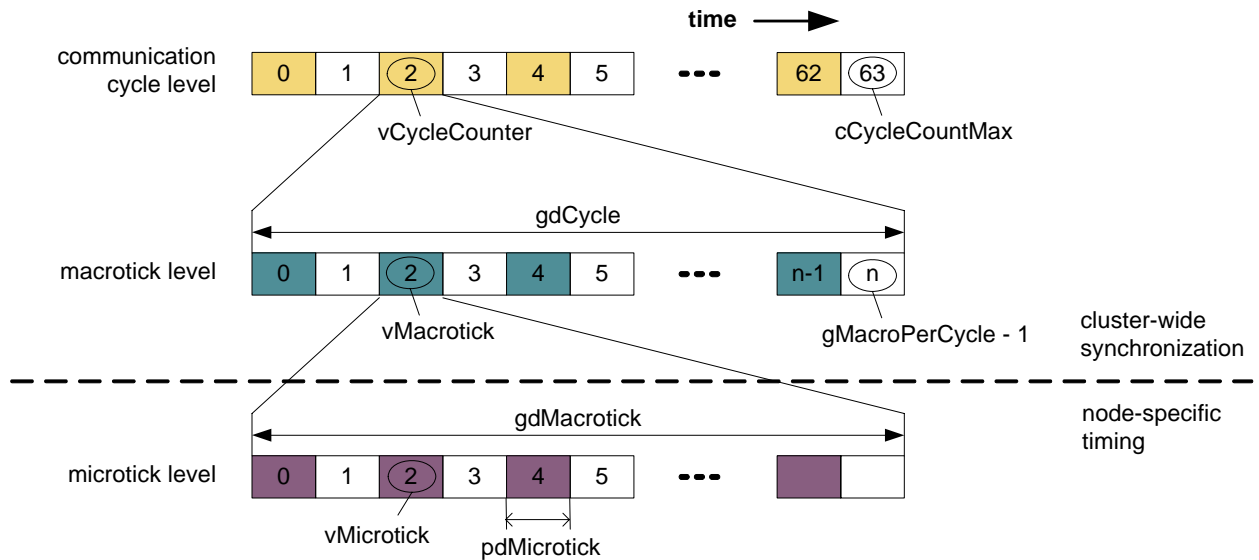


Figure 3. FlexRay Macrotick Timing

Time Units

As shown in Figure 3, two time units form the fundamental basis for measuring time in the FlexRay cluster:

- **Macrotick:** The basic unit of time by which the length of the higher-order time spans are measured:
 - Each static slot is *gdStaticSlot* macroticks in duration.
 - Each minislot in the dynamic segment is *gdMinislot* macroticks in duration.

- The symbol window is *gdSymbolWindow* macroticks in duration.
- Length of a communication cycle is *gMacroPerCycle* macroticks.
- The current macrotick count is maintained in each node by the variable *vMacrotick*.
- The absolute duration of a macrotick is determined by the parameter *gdMacrotick*, which is the same for all nodes in a cluster and has a range of 1-6 μ s.
- Microtick: The local unit of time as derived directly from the node's local oscillator with an optional prescaler. For example, a node that samples FlexRay bus data on an 80-MHz clock, and does not use a prescaler, for microtick timing would have a microtick duration (*pdMicrotick*) of 12.5 ns. The duration of a microtick varies from node to node.

The duration of each macrotick at the node level is in units of microticks. Therefore, the number of microticks per macrotick (*pMicroPerMacroNom*) can vary from node to node. Within a node, the number of microticks within a macrotick can vary from macrotick to macrotick as the node attempts to keep the average duration of each macrotick as close as possible to *gdMacrotick*. This variance of microticks per macrotick at the node level is the mechanism by which the global view of time (as seen in macroticks) remains consistent throughout the cluster even though the microtick duration varies from node to node.

The various dependencies between the FlexRay timing hierarchy related parameters are explained further in the FlexRay specification. In addition, there is FlexRay system development software available to help you define the communication schedule for a FlexRay system, calculate the associated FlexRay parameters, and model the effects of various parameter values and combinations. For the purpose of explaining how message buffers work, the important elements of the FlexRay timing hierarchy to understand are the static and dynamic slots because it is in the static and dynamic slots that the FlexRay frames stored in the message buffers are transmitted and received through the FlexRay bus.

Frame ID

Another important concept to highlight when discussing slot timing is Frame ID. Although Frame ID is not an element of the FlexRay timing hierarchy, it is related because it determines the slot in which a frame (contained in a message buffer) will be transmitted on the FlexRay bus.

Frame ID is therefore part of the fundamental connection between the host view of message transmission and reception (message buffers) and the slot timing of the FlexRay protocol.

MESSAGE BUFFER OVERVIEW

The CHI in a FlexRay Communication Controller serves two main purposes:

- Protocol data interface: Exchanges protocol-related control, configuration, and status data between the host and the FlexRay protocol.
- Message data interface: Exchanges messages and message-related control, configuration, and status data between the host and the FlexRay protocol. The message data interface implements the message buffer concept.

The specification identifies several services that the CHI must provide with regard to transmit and receive message buffers, but does not impose rigid requirements regarding the implementation method. As a designer, you have the choice to implement the FlexRay Communication Controller as a new design or integrate a pre-designed FlexRay Communication Controller, either as a chip or as an IP core.

The FlexRay Communication Controller solutions available today generally implement the message buffers as shown in Figure 4, where the host has access to the message buffer configuration, control, and status registers through the Controller's host interface, and both the host and the Controller access the message buffer header, payload, and status stored in a shared memory.

To set up a message buffer scheme for the system (or a node in the system) using a typical FlexRay Communication Controller implementation, you would need to:

- Plan the communication needs of the system, such as what types of messages need to be communicated between the various nodes.
- Determine the characteristics of the messages such as whether the messages should be transmitted in the static segment or dynamic segment and on which channel(s), and the required payload sizes.
- Configure the message buffers in the node to achieve the required functionality. This includes setting up the message buffer registers and allocating the required shared memory for the message data.

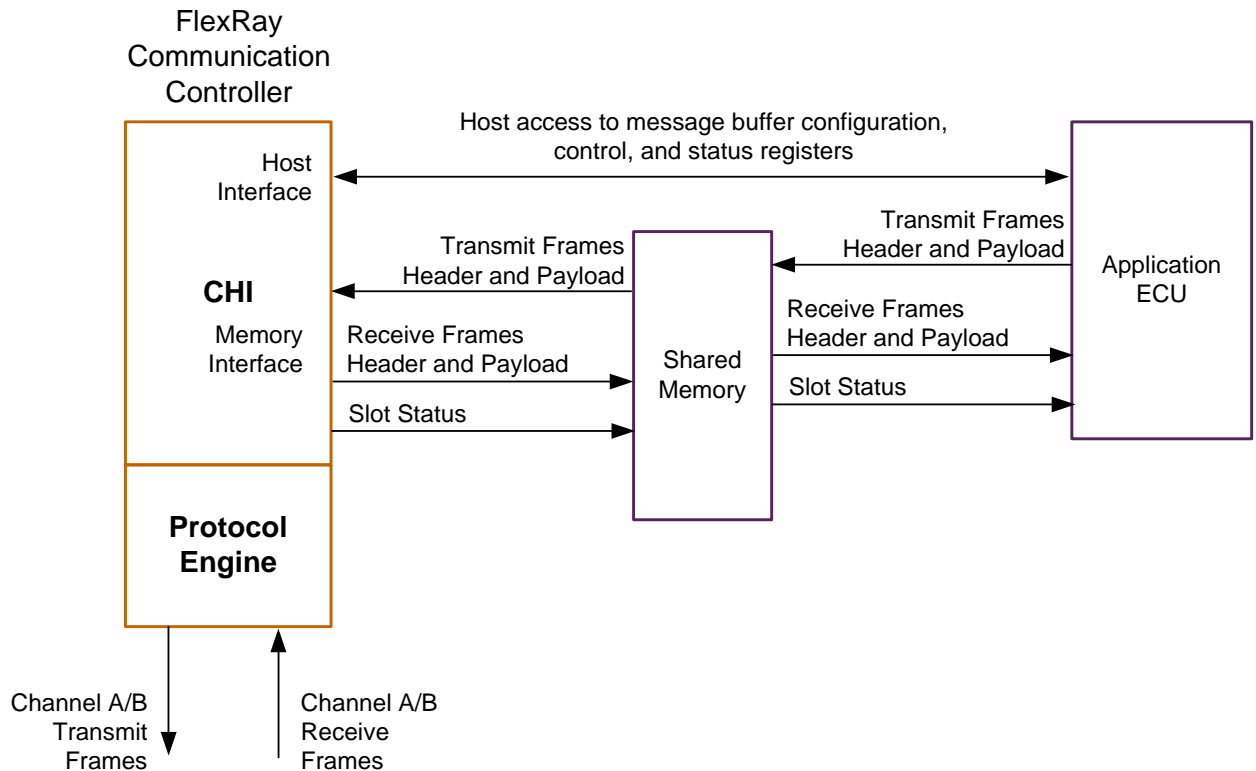


Figure 4. Example Message Buffer Data Exchange

For an IP core, some of the message buffer configuration is done at the hardware configuration stage. For example, you may be able to choose the maximum number of message buffers to be implemented and the maximum payload size to be supported. However, most of the message buffer configuration, such as assigning message buffers to slots and channels, selecting transmit or receive, and setting up cycle count filtering, is done when the cluster is initialized, during the configuration stage of the FlexRay protocol. Some re-configuration of the message buffers is also possible after the FlexRay protocol has transitioned from the configuration state to the run state.

TRANSMIT MESSAGE BUFFER SETUP

Message transmission (host to protocol) occurs on non-queued buffers, defined in the specification as meaning that the transmit message buffer has the following characteristics:

- Host has read/write access to transmit message buffer data.
- Protocol has read access to transmit message buffer data.
- New data replaces old data in the message buffer.

It is also possible to implement a queued transmit buffer scheme, such as is done in the FRCC2100 FlexRay Communication Controller, in which a 2-stage transmit queue is used to allow simultaneous access to the transmit message buffer from both the application and the FlexRay protocol.

The implementation of a transmit message buffer consists of two parts:

- Storage for the required message buffer configuration, control, and status data. This storage is typically implemented as host-accessible registers in the CHI.
- A region of shared memory allocated to the message buffer, the size of which depends on the maximum payload size of the message buffer. The shared memory stores the message header, payload data, and (typically) the slot status.

Assuming the required hardware is implemented already, setting up the transmit buffer is then a matter of programming the required configuration data into the message buffer configuration, control, and status registers in the CHI. The configuration data required for a transmit message buffer includes:

- Frame ID: Corresponds to the slot number in which the message will be transmitted. Whether the selected slot is a static or dynamic slot depends on how many static slots are configured.
- Channel: For static slots, a message buffer can be assigned to channel A or channel B or channels A and B. For dynamic slots, a message buffer can be assigned to channel A or B, but not both A and B.

For each transmit buffer, the CHI must also provide a means for the host to access the following control and status information:

- Data valid: An indication from the host that the transmit message buffer contains data that is valid for transmission.
- Slot status: A set of specification-defined slot status bits made available to the host from the CHI.

Other configuration information that may be implemented by a CHI includes:

- Cycle count filtering so that message transmission only occurs in the assigned slot in the assigned communication cycle (used only in the dynamic segment).
- Pointer to the location of the message header and payload data in the shared memory.

The FRCC2100 FlexRay Communication Controller is the IP core used in the FlexRay Communication Controller module from Freescale Semiconductor, Inc. and is available as a soft IP core from IPextreme, Inc.

- State or event mode transmission, which determines whether the node always transmits the valid data in each assigned slot or only when the data has been updated since the last transmission of that message buffer.

As a simple example, consider the cluster shown in Figure 1. Suppose that nodes B, C, D, and E must send a critical message to node A once per communication cycle; the message must be duplicated on channels A and B. Suppose also that nodes B, C, D, and E must also send diagnostic messages to node A on an ad-hoc basis (as requested by node A); separate diagnostic messages are required for channels A and B on each node. Nodes B, C, D, and E would then each require the following transmit buffer setup:

- One transmit buffer assigned to the static segment, channels A and B
- One transmit buffer assigned to the dynamic segment, channel A
- One transmit buffer assigned to the dynamic segment, channel B

To transmit the diagnostics request message to each node, node A could use:

- Four transmit buffers assigned to the dynamic segment, channel A (one for each node)
- Four transmit buffers assigned to the dynamic segment, channel B (one for each node)

Table 1 shows an example of the message buffer configuration data that could be used for such a setup, assuming that the FlexRay protocol timing is set up such that slot 10 is the first minislot in the dynamic segment (slots 1–9 are in the static segment). Note that:

- Nodes B, C, D, and E are each assigned a unique slot in the static segment. This is a FlexRay requirement. In the static segment, no 2 nodes can be assigned the same transmit slot in the same communication cycle.
- Slot 10, the first slot in the dynamic segment is used as a transmit slot by all the nodes, but each node uses a different communication cycle.
- In node A, each transmit message buffer is assigned to dynamic slot 10. However, each transmit message buffer has a unique channel and communication cycle assignment, so there will be no transmit conflicts on Node A.

- Nodes B, C, D, and E each use one transmit message buffer for its respective channel A/B diagnostic messages. Again, there is no transmit conflict because channels A and B are independent in the dynamic segment and each node is assigned a different communication cycle for its diagnostic messages.

In this example, node A could use an alternative approach by assigning a single message buffer for transmitting the diagnostic request message. A single message could be broadcast to all nodes to request their respective diagnostic messages. Or, to target individual nodes, node A could still use a single message buffer but embedded target node information in the Message ID part of the transmitted frame. Receiving nodes B, C, D, and E could then use Message ID filtering to accept only messages that they are intended to receive. Message IDs are used only in the dynamic segment.

Table 1. Transmit Message Buffer Setup

MB No.	Node A	Node B	Node C	Node D	Node E
1	Frame ID: 10 Channel: A Com. Cycle: 2 Purpose: Send channel A diagnostics request message to node B	Frame ID: 1 Channel: A and B Purpose: Send periodic critical data to node A	Frame ID: 2 Channel: A and B Purpose: Send periodic critical data to node A	Frame ID: 3 Channel: A and B Purpose: Send periodic critical data to node A	Frame ID: 4 Channel: A and B Purpose: Send periodic critical data to node A
2	Frame ID: 10 Channel: B Com. Cycle: 2 Purpose: Send channel B diagnostics request message to node B	Frame ID: 10 Channel: A Com. Cycle: 50 Purpose: Send channel A diagnostics data message to node A	Frame ID: 10 Channel: A Com. Cycle: 51 Purpose: Send channel A diagnostics data message to node A	Frame ID: 10 Channel: A Com. Cycle: 52 Purpose: Send channel A diagnostics data message to node A	Frame ID: 10 Channel: A Com. Cycle: 53 Purpose: Send channel A diagnostics data message to node A
3	Frame ID: 10 Channel: A Com. Cycle: 3 Purpose: Send channel A diagnostics request message to node C	Frame ID: 10 Channel: B Com. Cycle: 50 Purpose: Send channel B diagnostics data message to node A	Frame ID: 10 Channel: B Com. Cycle: 51 Purpose: Send channel B diagnostics data message to node A	Frame ID: 10 Channel: B Com. Cycle: 52 Purpose: Send channel B diagnostics data message to node A	Frame ID: 10 Channel: B Com. Cycle: 53 Purpose: Send channel B diagnostics data message to node A
4	Frame ID: 10 Channel: B Com. Cycle: 3 Purpose: Send channel B diagnostics request message to node C	Not Used	Not Used	Not Used	Not Used

Table 1. Transmit Message Buffer Setup

MB No.	Node A	Node B	Node C	Node D	Node E
5	Frame ID: 10 Channel: A Com. Cycle: 4 Purpose: Send channel A diagnostics request message to node D	Not Used	Not Used	Not Used	Not Used
6	Frame ID: 10 Channel: B Com. Cycle: 4 Purpose: Send channel B diagnostics request message to node D	Not Used	Not Used	Not Used	Not Used
7	Frame ID: 10 Channel: A Com. Cycle: 5 Purpose: Send channel A diagnostics request message to node E	Not Used	Not Used	Not Used	Not Used
8	Frame ID: 10 Channel: B Com. Cycle: 5 Purpose: Send channel B diagnostics request message to node E	Not Used	Not Used	Not Used	Not Used

MESSAGE TRANSMISSION

Once the message buffer configuration has been set up and the FlexRay protocol is in the running state, the host can transmit a message by:

1. Writing the message buffer header and data to the assigned location in the shared memory.
2. Notifying the FlexRay Communication Controller that valid transmit data is available for that message buffer.

Before the beginning of the next slot in the static segment, the FlexRay Communication Controller searches its message buffer configuration, control, status registers for a match between the Frame ID and the next slot number. If there is a match, the node transmits one of two frames on the assigned channel(s) in the next slot:

- If valid transmit data exists for the message buffer with the matching Frame ID and no implementation-specific transmit filtering prevents it, the node transmits the valid transmit message on the assigned channel(s).
- If no valid transmit data exists for the message buffer with the matching Frame ID or transmit message filtering prevents sending valid transmit data, the node transmits a null frame in that slot on the assigned channel(s).

In the dynamic segment, a similar Frame ID vs. slot number check occurs. However, null frames are not transmitted in this case:

- If valid transmit data exists for the message buffer with the matching Frame ID and no transmit filtering (such as cycle count filtering) prevents it, the node transmits the valid transmit message on the assigned channel.
- If no valid transmit data exists for the message buffer with the matching Frame ID or transmit message filtering (such as cycle count filtering) prevents sending valid transmit data, the node simply does not transmit the message in that slot. No null frame is sent.

At the completion of every communication slot, the FlexRay Communication Controller updates the corresponding slot status information to the host. The means by which this is done is implementation-specific—for example, there may be slot status registers in the Controller and/or the slot status can be written to the shared memory associated with the message buffer.

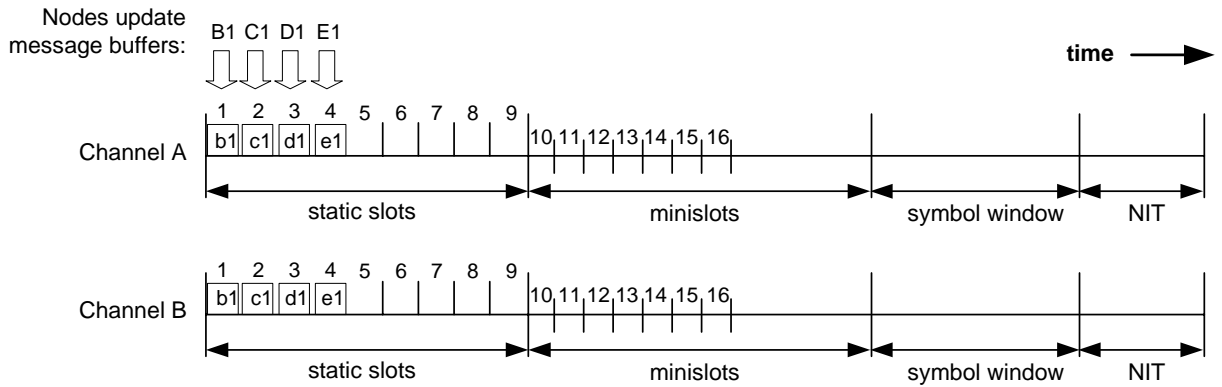
The above scenarios assume that the node uses the state transmission scheme. If the node uses the event transmission scheme, then transmission of the valid frame depends on whether the application has updated the message buffer data since it was last transmitted.

NXP (formerly Philips) Semiconductor developed sophisticated FlexRay e models. They are available in the FlexRay eVC kit from IPextreme, Inc and can help an integrator verify a custom CHI implementation, test a node configuration in a simulated network, and help ensure first time success in FlexRay conformance testing.

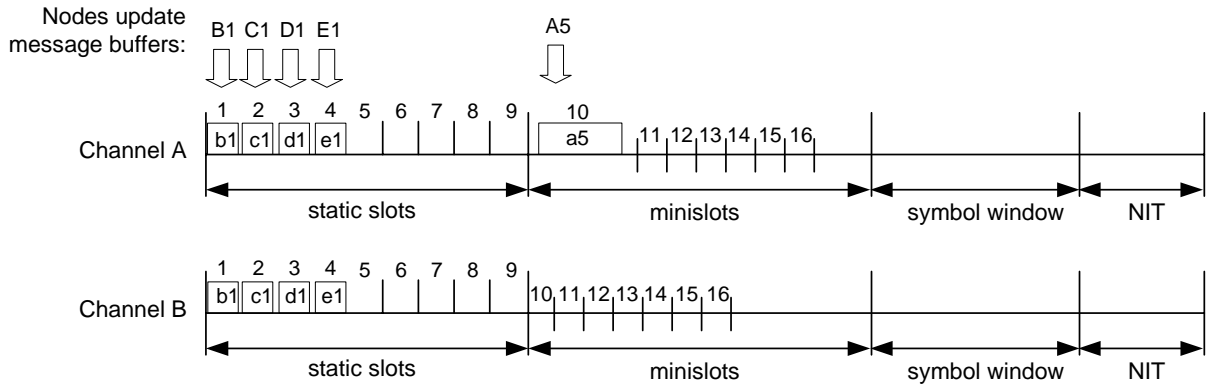
Figure 4 shows the FlexRay bus activity for this transmit message buffer setup. In this example:

- In nodes B, C, D, and E, the application updates the static slot message buffer data in each communication cycle. Each node transmits its critical data on both channels in every communication cycle.
- In communication cycle 4, node A has updated message buffer 5 to send a diagnostics request to node D on channel A. So, node A transmits the message in slot 10 in the dynamic segment.
- In communication cycle 52, node D has updated message buffer 2 to send a diagnostics message to node A on channel A. So, node D transmits the message in slot 10 in the dynamic segment. Assuming that node D always updates its message buffer in time, node A can expect a predictable latency (48 communication cycles) between sending its diagnostic request message to node and receiving the reply from node D.

Every communication cycle without dynamic segment transmission:



Communication cycle 4:



Communication cycle 52:

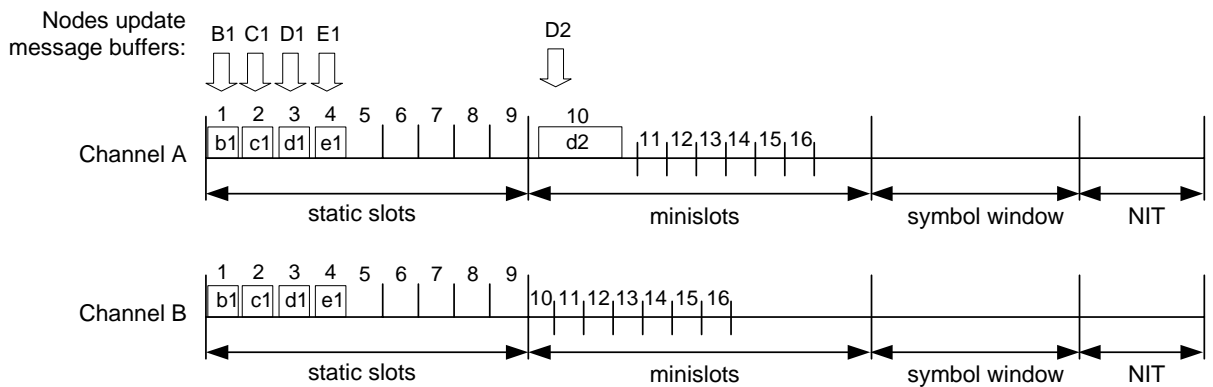


Figure 4. FlexRay Message Transmit Example

RECEIVE MESSAGE BUFFER SETUP

Message reception (protocol to host) operates on non-queued buffers and queued buffers. A non-queued receive message buffer is defined in the specification as having the following characteristics:

- Host has read access to receive message buffer data.
- Protocol has write access to receive message buffer data.
- New data replaces old data in the message buffer.

A queued receive message buffer has the following characteristics:

- Host has read access to receive message buffer data.
- Protocol has write access to receive message buffer data.
- New data is queued behind previous data in the message buffer.

The queued receive message buffer is otherwise known as a receive FIFO and is discussed later in this article.

Like the transmit buffer, the implementation of a receive message buffer consists of two parts:

- Storage for the required message buffer configuration, control, and status data, typically implemented as host-accessible registers in the CHI.
- A region of shared memory allocated to the message buffer, the size of which depends on the maximum payload size of the message buffer. The shared memory stores the message header, payload data, and (typically) the slot status.

The basic configuration data for a receive message buffer includes:

- **Frame ID:** Determines which slot the receive message buffer is subscribed to. An incoming valid frame passes the Frame ID filter check if it is received in a communication slot equal to the configured Frame ID for the specific receive message buffer. Whether the specified slot number is a static or dynamic slot depends on how many static slots are configured.
- **Channel:** Determines which channel(s) the receive message buffer is subscribed to. For static slots, a receive message buffer can be assigned to channel A or channel B or channels A and B. For dynamic slots, a receive message buffer can be assigned to channel A or B, but not both A and B.

- Cycle count: Determines which communication cycle the receive message buffer is subscribed to.

For each receive buffer, the CHI must also provide a means for the host to access the following control and status information:

- Data valid: An indication from the host that the receive message buffer contains valid data.
- Slot status: A set of specification-defined slot status bits made available to the host from the CHI.

Other configuration information that may be implemented by a CHI includes:

- Message ID: In the dynamic segment, Message ID filtering can also be applied to incoming valid frames.
- Pointer to the location of the message header and payload data in the shared memory.

Returning to the example cluster shown in Figure 1, node A needs:

- Four receive buffers assigned to the static segment to receive the critical data from nodes B, C, D, and E
- Eight receive buffers assigned to the dynamic segment to receive diagnostics data from nodes B, C, D, and E (channels A and B)

Nodes B, C, D, and E each need:

- One receive buffer assigned to the dynamic segment to receive diagnostics request messages from node A on channel A
- One receive buffer assigned to the dynamic segment to receive diagnostics request messages from node A on channel B

Table 2 shows an example of the receive message buffer configuration for the cluster to match the transmit message buffer configuration established in Table 1.

Table 2. Receive Message Buffer Setup

MB No.	Node A	Node B	Node C	Node D	Node E
1	Frame ID: 1 Channel: A and B Purpose: Receive periodic critical data from node B	Frame ID: 10 Channel: A Com. Cycle: 2 Purpose: Receive channel A diagnostics request message from node A	Frame ID: 10 Channel: A Com. Cycle: 3 Purpose: Receive channel A diagnostics request message from node A	Frame ID: 10 Channel: A Com. Cycle: 4 Purpose: Receive channel A diagnostics request message from node A	Frame ID: 10 Channel: A Com. Cycle: 5 Purpose: Receive channel A diagnostics request message from node A
2	Frame ID: 2 Channel: A and B Purpose: Receive periodic critical data from node C	Frame ID: 10 Channel: B Com. Cycle: 2 Purpose: Receive channel B diagnostics request message from node A	Frame ID: 10 Channel: B Com. Cycle: 3 Purpose: Receive channel B diagnostics request message from node A	Frame ID: 10 Channel: B Com. Cycle: 4 Purpose: Receive channel B diagnostics request message from node A	Frame ID: 10 Channel: B Com. Cycle: 5 Purpose: Receive channel B diagnostics request message from node A
3	Frame ID: 3 Channel: A and B Purpose: Receive periodic critical data from node D	Not Used	Not Used	Not Used	Not Used
4	Frame ID: 4 Channel: A and B Purpose: Receive periodic critical data from node E	Not Used	Not Used	Not Used	Not Used
5	Frame ID: 10 Channel: A Com. Cycle: 50 Purpose: Receive channel A diagnostics data message from node B	Not Used	Not Used	Not Used	Not Used
6	Frame ID: 10 Channel: B Com. Cycle: 50 Purpose: Receive channel B diagnostics data message from node B	Not Used	Not Used	Not Used	Not Used

Table 2. Receive Message Buffer Setup

MB No.	Node A	Node B	Node C	Node D	Node E
7	Frame ID: 10 Channel: A Com. Cycle: 51 Purpose: Receive channel A diagnostics data message from node C	Not Used	Not Used	Not Used	Not Used
8	Frame ID: 10 Channel: B Com. Cycle: 51 Purpose: Receive channel B diagnostics data message from node C	Not Used	Not Used	Not Used	Not Used
9	Frame ID: 10 Channel: A Com. Cycle: 52 Purpose: Receive channel A diagnostics data message from node D	Not Used	Not Used	Not Used	Not Used
10	Frame ID: 10 Channel: B Com. Cycle: 52 Purpose: Receive channel B diagnostics data message from node D	Not Used	Not Used	Not Used	Not Used
11	Frame ID: 10 Channel: A Com. Cycle: 53 Purpose: Receive channel A diagnostics data message from node E	Not Used	Not Used	Not Used	Not Used
12	Frame ID: 10 Channel: B Com. Cycle: 53 Purpose: Receive channel B diagnostics data message from node E	Not Used	Not Used	Not Used	Not Used

MESSAGE RECEPTION

The receiving node places the incoming valid frame into the corresponding message buffer if it passes all of the specified filtering criteria for that message buffer, such as:

- Frame ID filter
- Channel filter
- Cycle count filter
- Message ID filter (in the dynamic segment only)

To temporarily store a frame while it is checked and filtered, a receiving node may implement one or more so-called shadow buffers. The shadow buffer stores the frame until the frame has passed validity checking and filtering and is assigned to a specific message buffer, at which time the slot status is updated and the application is notified that the receive message buffer now contains a valid frame. The application can then read the message from the shared memory.

The shadow buffer is an implementation-specific concept, not explicitly defined in the FlexRay specification.

In the FRCC2100, the data transfer from the shadow buffer to the actual receive message buffer is accomplished by switching the pointers (memory indexes) of the shadow buffer and the receive message buffer, saving the memory read/write cycles that would otherwise be required to move the data.

RECEIVE FIFO

To accept frames that are valid but are not assigned to a specific non-queued message buffer, a node may implement a receive FIFO. A receive FIFO also typically consists of a set of configuration, control, and status registers in the CHI and a designated region of shared memory to store the incoming frames that pass the filter criteria configured for the receive FIFO.

Configuration and control data for a receive FIFO can include filter setup such as Frame ID filtering or Message ID filtering in the dynamic segment. FIFO status from the Controller to the application typically includes index pointers and empty/full status.

MESSAGE BUFFER LOCKING

With both the application and the FlexRay Communication Controller accessing the message buffer data in the shared memory, it is important to avoid read/write conflicts:

- The Controller should not read from a transmit message buffer while the application is writing to it.
- Similarly, the application should not read from a receive message buffer while the Controller is writing to it.

The Controller's access to the message buffer data is generally a function of the FlexRay timing. For example, if a transmit message buffer has Frame ID 3, the Controller will transmit the message buffer in slot 3 and therefore need to access the message buffer data in the shared memory at that time. By monitoring the current slot number and communication cycle number, the application can time its access to avoid shared memory access conflicts. In addition, the Controller should implement a scheme to allow the application and the Controller to lock message buffers on an individual basis during their respective access times.

To allow concurrent host/protocol access to a transmit message buffer, the FRCC2100 offers the option of setting up the transmit message buffer as a type of queued buffer, basically a FIFO with a depth of 2. This allows the application to write to one side of the transmit message buffer while the protocol reads from the other side.

INTERRUPTS

Interrupts provide a convenient mechanism to alert the host to changes in message buffer status such as:

- Frame transmitted
- Frame received
- Receive FIFO status

The application can then read the message buffer status registers in the controller to acquire additional information, and respond accordingly.

SUMMARY

Message buffers provide a convenient mechanism to de-couple the operation of the application in a FlexRay node from the real-time activity on the FlexRay bus. The host view of the FlexRay communication is the exchange of FlexRay frames between the application and the shared memory associated with the message buffers, and the setup and monitoring of the associated configuration/control/status data.

The FlexRay specification imposes a set of requirements on the message data interface services that the CHI block in a FlexRay Communication Controller must provide. However, the specific message buffer implementation and the host interaction with the message buffers can vary between FlexRay Communication Controllers from different providers. Understanding the fundamentals of the FlexRay bus timing and the correlation between message buffers and FlexRay communication slots is an important step in the planning of a FlexRay node or network and choosing the FlexRay Communication Controller that is best suited to your application needs.



www.ip-extreme.com

IPextreme, Inc.

307 Orchard City Drive
M.S. 202
Campbell, CA 95008
800-289-6412 (toll-free)
408-608-0421 (fax)

THIS WHITE PAPER IS FOR INFORMATIONAL PURPOSES ONLY. THE CONTENT IS PROVIDED AS IS, WITHOUT EXPRESS OR IMPLIED WARRANTIES OF ANY KIND. INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT NOTICE.

© Copyright 2006, IPextreme. All rights reserved. IPextreme and the IPextreme logo are trademarks of IPextreme, Inc. All other trademarks are the property of their respective owners.
