Zwischen-gespeichert

Effizienter Einsatz der FlexRay-Nachrichtenpuffer

Die optimale Nutzung der Nachrichtenpuffer in einem FlexRay-System trägt viel zur Übertragungssicherheit und Datenintegrität bei, denn diese Puffer dienen der Entkopplung der ausgeführten FlexRay-Frame-Übertragungen durch die Applikation. Hier ein Einblick in deren Arbeitsweise.

Von Stefan Schmechtig

Die Knoten eines FlexRay-Systems nutzen die Nachrichtenpuffer zum Versenden und empfangen von Nachrichten. Darüber hinaus lässt sich durch sie der Zugriff der Applikation auf die Daten des Nachrichtenpuffers vom eigentlichen FlexRay-Bus-Timing entkoppeln. Das im Folgenden vorgestellte Beispiel zeigt, wie eine Applikation FlexRay-Nachrichtenpuffer verwenden kann, um

- für die Übermittlung vorgesehene Daten in einem Bereich des gemeinsam genutzten Speichers (Shared Memory) abzulegen, wobei sicher gestellt ist, dass der zugehörige FlexRay-Frame im zugewiesenen Slot übertragen wird;
- auf die Meldung, dass ein gültiger FlexRay-Frame empfangen wurde, zu antworten und die zugehörigen Daten aus dem zugewiesenen Bereich des Shared Memory abzurufen.

Die Nachrichtenpuffer und ihre Aufgaben

Ein Nachrichtenpuffer eines FlexRay-Knotens („Node“) besteht aus der erforderlichen Hardware und der Applikations-Software, die für die Konfiguration, Steuerung und Überwachung des Nachrichtenpuffers entsprechend der Kommunikationsanforderungen des Zielsystems benötigt wird. **Bild 1** zeigt die Struktur eines typischen FlexRay-Knotens, der Teil eines fünf Knoten umfassenden, zweikanaligen FlexRay-Clusters ist. Der „FlexRay Communication Controller“ (hier gezeigt in Knoten A) setzt sich aus den folgenden zwei Hauptblöcken zusammen:

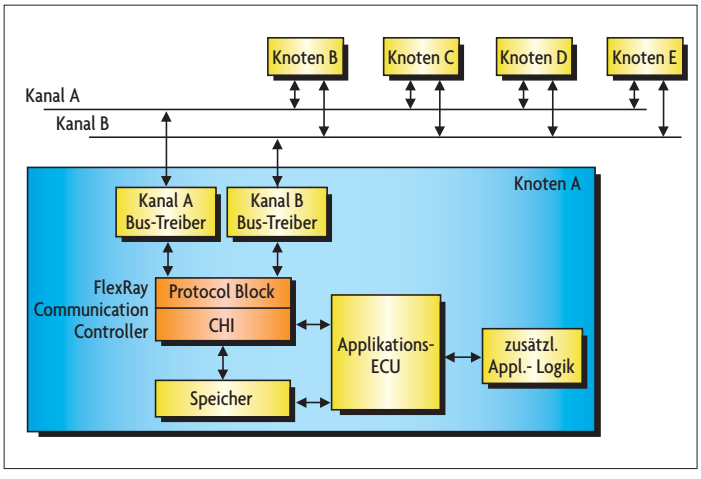


Bild 1. Beispiel eines FlexRay-Cluster mit den Knoten A bis E. (Bildquellen: IPextreme)

- Protocol Engine (PE): In der PE ist der Großteil des FlexRay-Protokolls implementiert. Sie übernimmt u.a. den Versand und den Empfang der Frames, die Aufrechterhaltung der Taktsynchronität und die Erzeugung sowie Kontrolle der Prüfsumme (CRC, Cyclic Redundancy Check).
- Controller Host Interface (CHI): Diese Schnittstelle ermöglicht der Applikation die Konfiguration, Steuerung und Überwachung der PE sowie den Austausch der Daten und des Status der Nachrichten mit der PE. Die Nachrichtendaten werden über den Nachrichtenpuffer ausgetauscht, wobei jeder Nachrichtenpuffer einem Kommunikations-Slot der FlexRay-Bus-Timing-Hierarchie zugewiesen ist.

Im CHI sind üblicherweise die Konfigurations-, Steuerungs- und Status-Register für die Nachrichtenpuffer implementiert, wobei gemeinsam genutzter (oder ein dedizierter) Speicher verwendet wird, um den Frame-Header, die Nutzdaten und den Slot-Status der Nachrichtenpuffer zu speichern.

Sendepufferaufbau zum Versenden von Nachrichten

Zu den Konfigurationsdaten eines Sendepuffers für Nachrichten gehören:

- Frame ID: Entspricht der Nummer des Slots, in dem die Nachricht gesendet werden soll. Dabei kann es sich um einen statischen oder dynamischen Slot handeln.
- Kanal: Im Falle statischer Slots kann ein Nachrichtenpuffer Kanal A oder Kanal B oder sowohl Kanal A und B zugewiesen werden. Im Falle dyna-

mischer Slots kann ein Nachrichtenpuffer entweder Kanal A oder Kanal B, aber nicht beiden gleichzeitig zugewiesen werden.

Für jeden Sendepuffer muss das CHI zudem für einen Host-Zugriff auf die folgenden Steuer- und Status-Informationen sorgen:

- Daten gültig: Der Host meldet, wenn der Sendepuffer gültige Daten für das Versenden bereithält.
- Slot-Status: Durch die Spezifikation definierte Slot-Status-Bits vom CHI zum Host.

Im CHI können auch folgende Funktionen implementiert sein:

- Das Filtern des Zykluszählers. Dies wird genutzt, um die Übertragung der Nachricht nur in den zugewiesenen Kommunikationszyklus (wird nur im dynamischen Segment verwendet) zuzulassen.
- Ein Zeiger auf den Speicherort des Nachrichten-Headers und der Nutzdaten im Shared Memory.
- Ein statischer oder ereignisgesteuerter Übertragungsmodus, der bestimmt, ob der Knoten die gültigen Daten immer im zugewiesenen Slot überträgt oder nur dann sendet, wenn der Inhalt des Nachrichtenpuffers seit der letzten Übertragung aktualisiert wurde.

Ein einfaches Beispiel: Angenommen, die Knoten B, C, D und E aus Bild 1 müssen jeweils auf den Kanälen A und B pro Kommunikationszyklus eine kritische Meldung an Knoten A senden. Weiterhin sei unterstellt, dass die Knoten B, C, D und E jeweils eine kanalspezifische Diagnosenachricht an Koten A senden, allerdings nur, wenn Knoten A diese anfordert. Die Knoten B, C, D und E würden dann jeweils die folgenden Komponenten benötigen:

- Einen dem statischen Segment zugeordneten Sendepuffer für die Übertragung auf Kanal A und B.
- Einen dem dynamischen Segment zugeordneten Sendepuffer für die Übertragung auf Kanal A.
- Einen dem dynamischen Segment zugeordneten Sendepuffer für die Übertragung auf Kanal B.

Für die Übermittlung der Diagnoseanforderung an jeden Knoten könnte Knoten A folgende Komponenten nutzen:

- Vier dem dynamischen Segment zugeordneten Sendepuffer für die Übertragung auf Kanal A.
- Vier dem dynamischen Segment zugeordneten Sendepuffer für die Übertragung auf Kanal B.

Bild 2 zeigt mögliche Nachrichtenpuffer-Konfigurationsdaten, die für den im Beispiel dargestellten Aufbau verwendet werden könnten. Dabei wird angenommen, dass Slot 10 der erste Minislot im dynamischen Segment ist (Slot 1 bis 9 liegen im statischen Segment). Dabei ist zu beachten:

- Den Knoten B, C, D und E ist jeweils ein eigener Slot im statischen Segment zugeordnet, wie dies für FlexRay festgelegt ist. Im statischen Segment kann pro Kommunikationszyklus nicht mehr als ein Knoten demselben Sende-Slot zugewiesen werden.
- Slot 10, der erste Slot des dynamischen Segments, wird von allen Knoten als Übertragungs-Slot genutzt. Allerdings verwendet jeder Knoten einen anderen Kommunikationszyklus und/oder eine andere Kanalzuweisung, so dass auch hier keine Kollisionen bei der Übertragung auftreten können.

Empfangspufferaufbau für den Empfang von Nachrichten

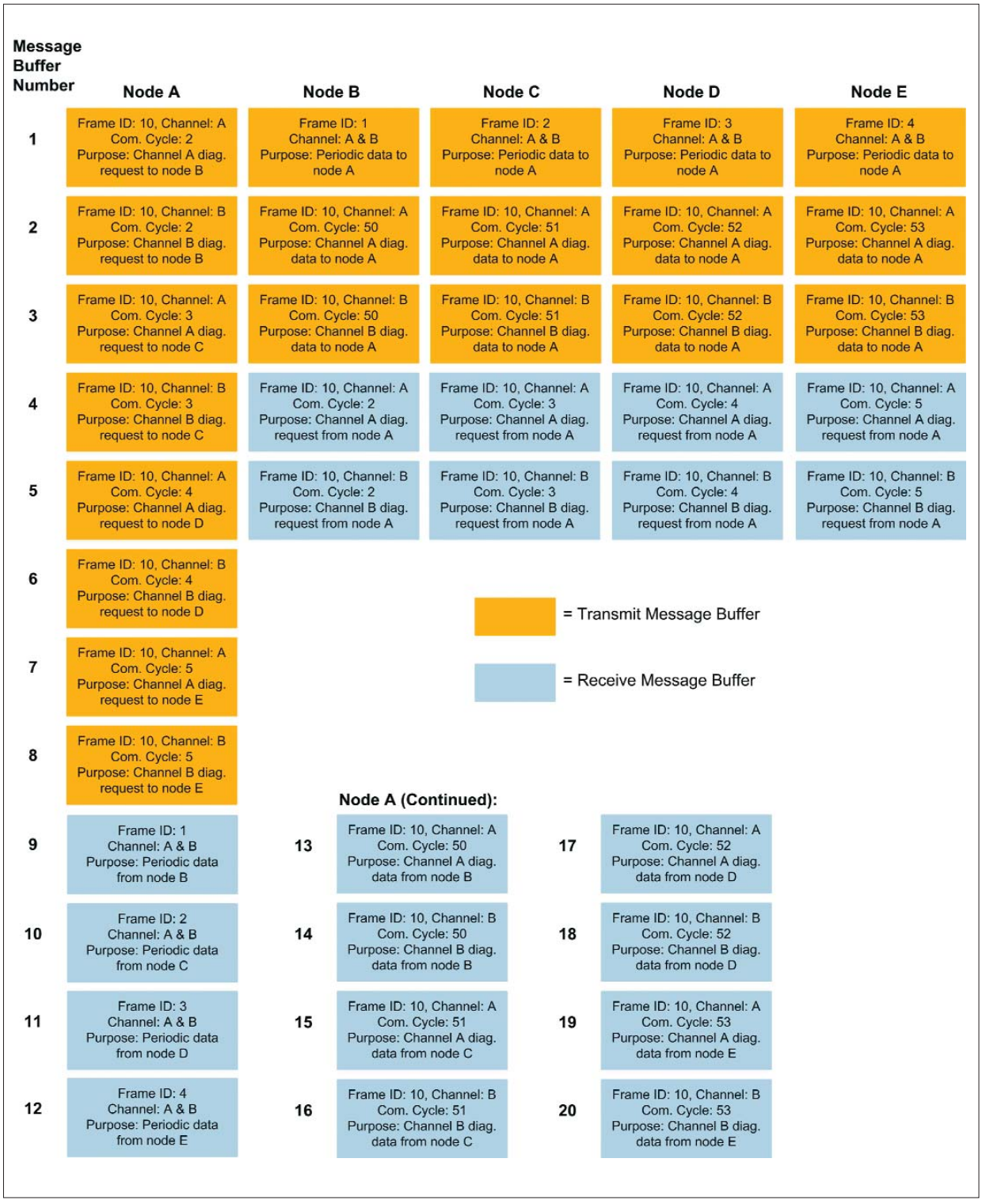
Für den Nachrichtenempfang können beide Empfangspuffer-Implementierungen verwendet werden: so genannte „non-queued“ Puffer (einzelne individuelle Puffer, in denen die alten Empfangsdaten durch die neuen Daten ersetzt werden) oder queued Puffer (Empfangs-FIFOs, in denen die Daten in einer FIFO-Struktur hintereinander abgelegt werden) – letztere kann man nutzen, um gültige Frames zu akzeptieren, die nicht einem bestimmten Non-queued-Nachrichtenpuffer zugewiesen sind.

Knoten A des Beispiel-Clusters aus Bild 1 benötigt folgende Komponenten:

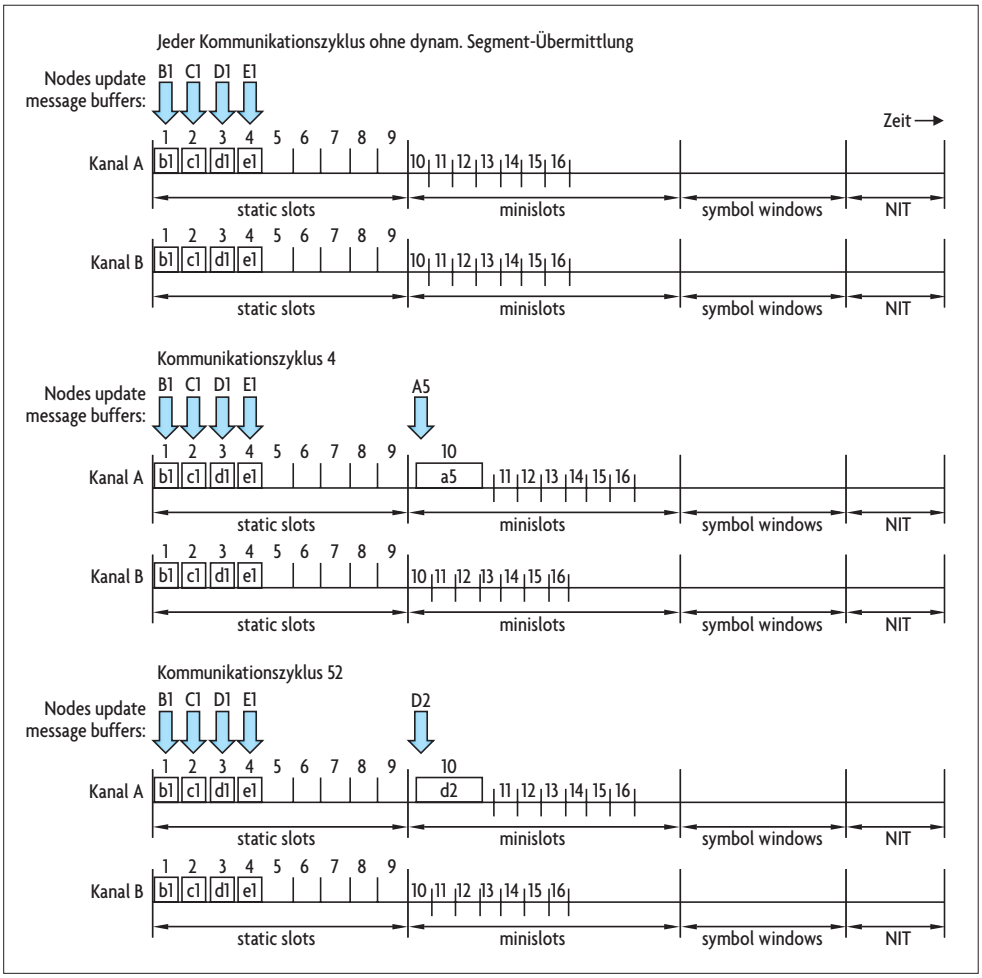
- Vier dem statischen Segment zugeordnete Empfangspuffer für den Empfang der von den Knoten B, C, D, und E gesendeten kritischen Daten.
- Acht dem dynamischen Segment zugeordnete Empfangspuffer für den Empfang der von den Knoten B, C, D und E (über Kanal A und B) gesendeten Diagnosedaten.

Die Knoten B, C, D und E benötigen jeweils zwei (einen für jeden Kanal) dem dynamischen Segment zugeordnete Empfangspuffer für den Empfang der Diagnoseanforderungsnachrichten.

Bild 2 zeigt die Nachrichtenpufferkonfiguration für den Beispielaufbau,



I Bild 2. Beispiel für den Aufbau eines Nachrichtenpuffers.



I Bild 3. Beispiel für eine FlexRay-Nachrichtenübertragung.

wobei die Empfangsnachrichtenpuffer (blaue Blöcke) so aufgebaut sind, dass sie mit den Empfangsnachrichtenpuffern (gelbe Blöcke) korrespondieren. **Bild 3** illustriert schließlich den sich in dieser Konfiguration ergebenden FlexRay-Busverkehr.

Nachrichtenpuffer dienen also letztlich zur Entkopplung der durch die Applikation ausgeführten FlexRay-Frame-Übertragungen (Senden und Empfangen) vom Echtzeit-Verkehr auf dem FlexRay-Bus. Das hier dargestellte einfache Beispiel beleuchtet den Zusammenhang von Nachrichtenpufferkonfiguration und Übertragung von Nachrichten in den zugehörigen FlexRay-Kommunikations-Slots.

Unter der Internetadresse www.ip-extreme.com sind ergänzend zum Thema „Nachrichtenpufferaufbau“ weitere detaillierte Informationen in Form von White Papers sowie eine Übersicht der FlexRay-Bus-Timing-Hierarchie verfügbar. *ha*

Dipl.-Ing. Stefan Schmechtig

hat an der Friedrich Alexander Universität Erlangen studiert. Seine berufliche Karriere begann er bei Synopsys Compiled Designs als Consultant. In dieser Position unterstützte er Design- und Verifikationsprojekte diverser Kunden aus unterschiedlichen Anwendungsbereichen. Heute arbeitet er als Projekt-Manager am Design Center von IPextreme in München. Vor seinem Wechsel zu IPextreme arbeitete er bei Synopsys als Senior Design Engineer und betreute Bluetooth- und Embedded-Mikroprozessor-Desingprojekte.
stefan.schmechtig@ip-extreme.com