



23

The FlexRay Protocol

Philip Koopman

Significant material drawn from
FlexRay Specification Version 2.0, June 2004

November 28, 2011

Preview

- ◆ **FlexRay – main competitor to TTP for X-by-Wire applications**
 - Created by industry consortium founded in 2000
 - Core members: BMW, DaimlerChrysler, General Motors, Motorola, Philips, Volkswagen, and Robert Bosch.

- ◆ **First public FlexRay protocol specification June 30, 2004 [FlexRay04]**
 - Combination Time-Triggered & Event-Triggered Approach
 - Intended for use in safety critical, fault-tolerant systems

- ◆ **Dec. 7, 2006:**

“FlexRay protocol has entered its production phase with devices from NXP(formerly Philips Semiconductors) and Freescale Semiconductor in BMW's newest X5 sport activity vehicle.”

- ◆ **High volume production reached in about 2010**
 - E.g., NXP had shipped 1 million Flexray chips by 2009; 2 million by mid-2010

FlexRay Prototype Hardware (Convergence 2000)



Topology – Active Star Plays Key Role

◆ Active star simplifies some aspects of distributed coordination

- Maximum delay through star is 250 ns, so it does not buffer full messages

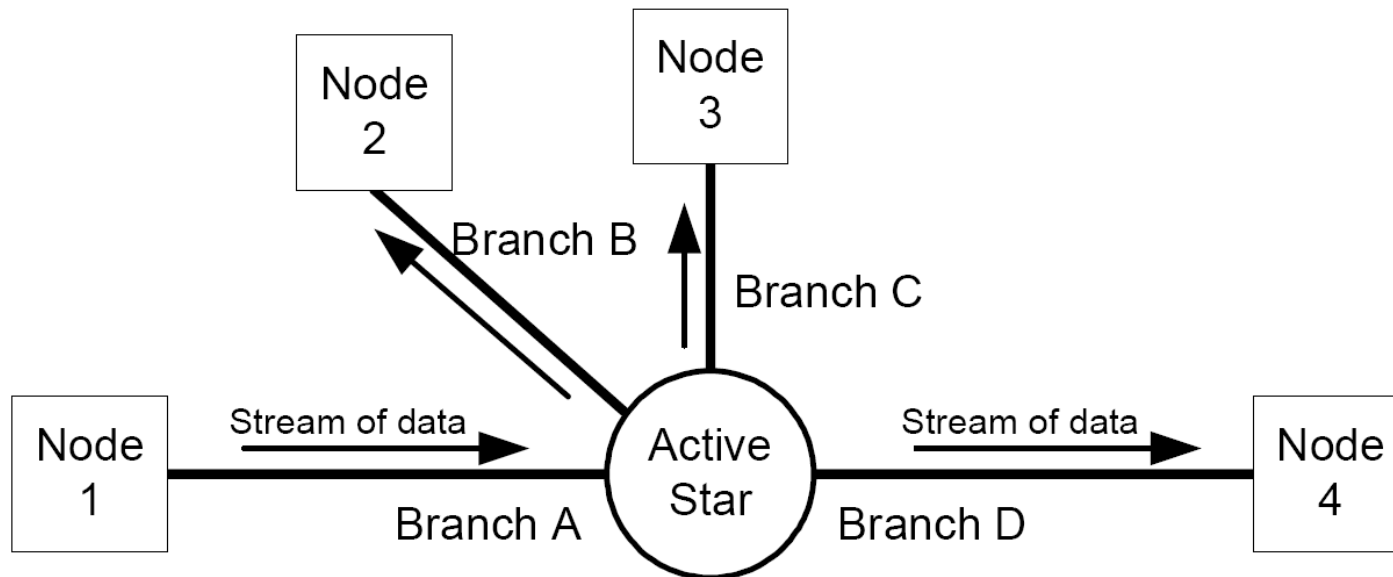
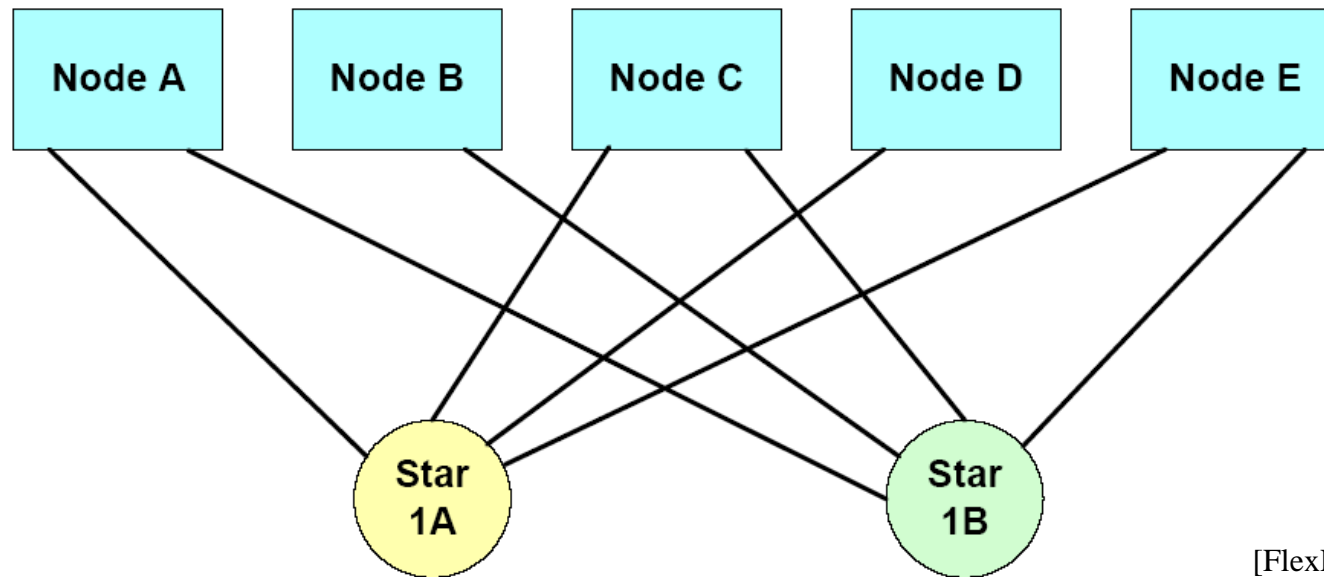


Figure 9-2: Active Star transfer functionality.

[FlexRay04]

Redundant Active Star

- ◆ **Intended to eliminate single-point failures for critical systems**
 - This seems the most likely configuration for FlexRay X-by-Wire



[FlexRay04]

Figure 1-2: Dual channel single star configuration.

- ◆ **TTP was found to have some distributed bus guardian issues**
 - Problems related to nodes listening to faulty network startup messages
 - Latest proposal is to move to dual channel star configuration for TTP as well

General FlexRay Node Block Diagram

- Host is application CPU
- Bus guardian controls enable line on bus driver

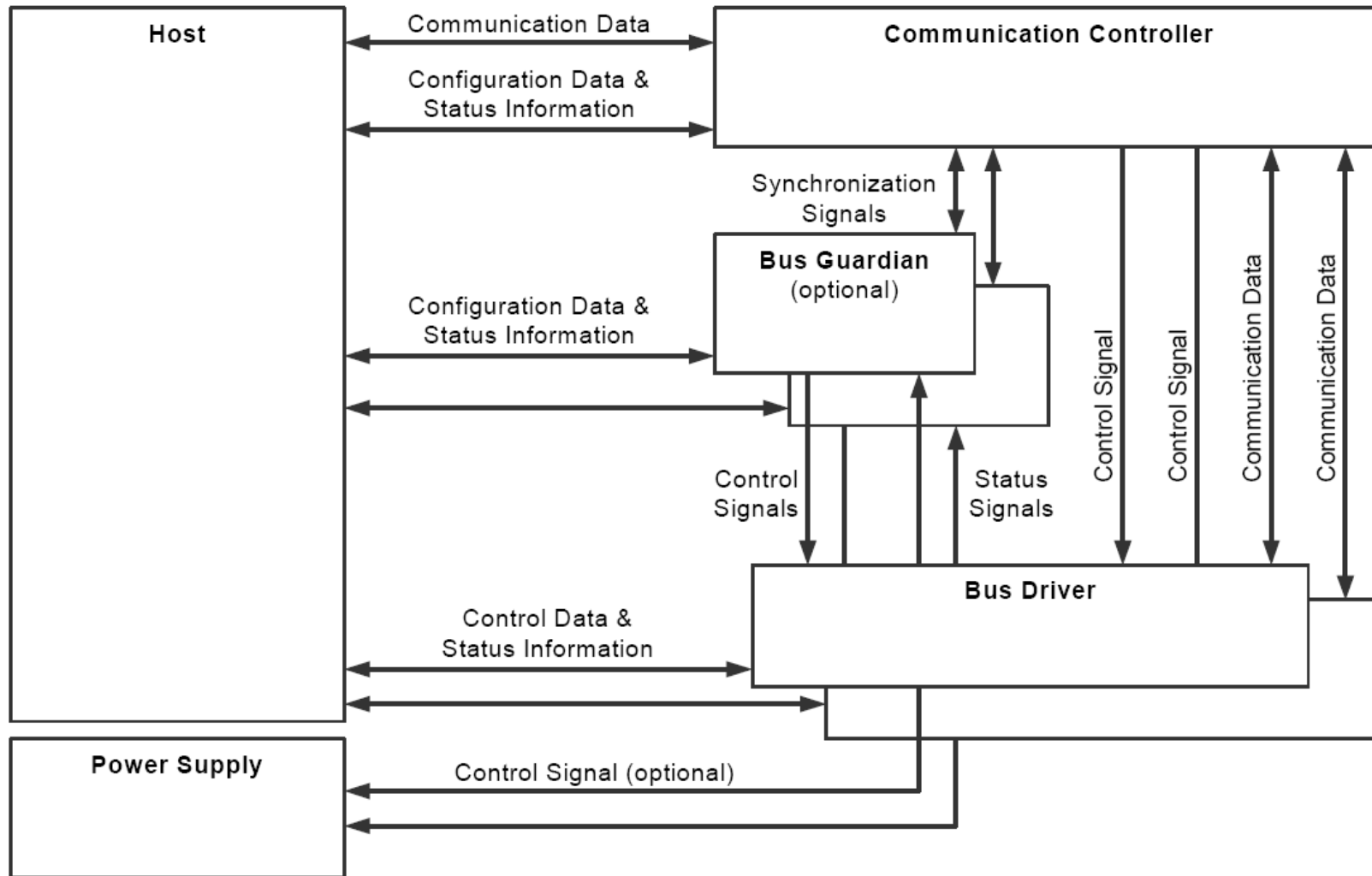


Figure 1-7: All logical interfaces.

[FlexRay04]

Physical Layer

◆ Differential NRZ encoding

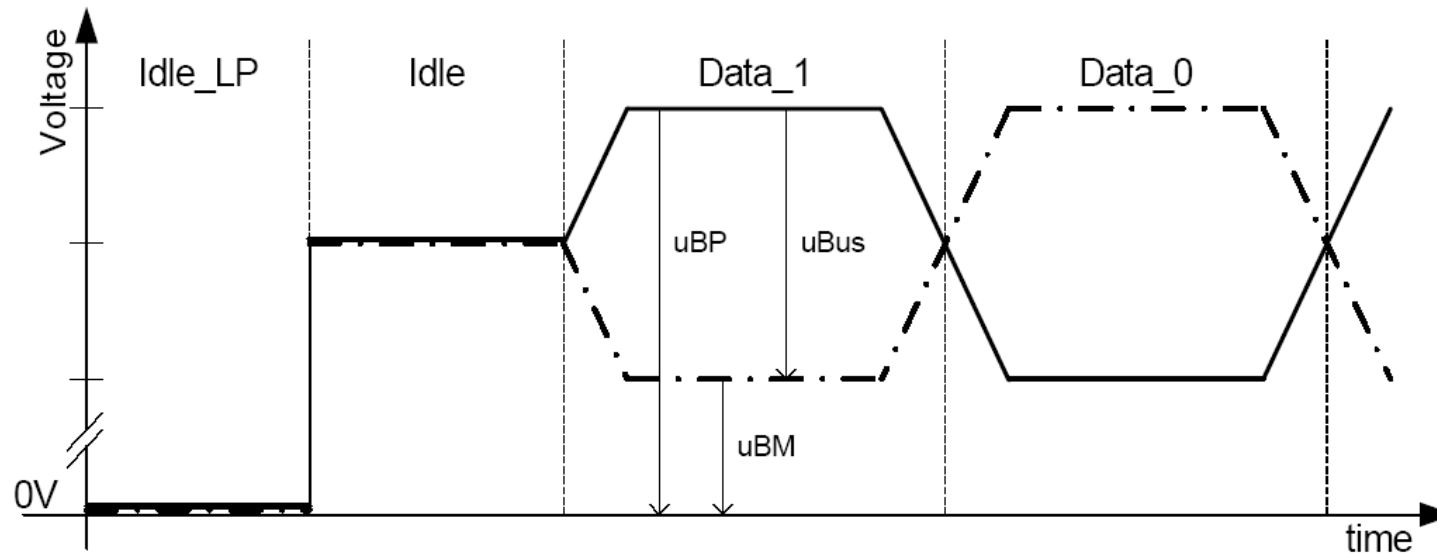


Figure 6-1: Electrical signaling.

[FlexRay04]

◆ 10 Mbps operating speed

- Independent of network length because, unlike CAN, doesn't use bit arbitration

FlexRay Encoding Approach

◆ Data sent as NRZ bytes

- TSS = Transmit Start Sequence (LOW for 5-15 bits)
- FSS = Frame Start Sequence (one HI bit)
- BSS = Byte Start Sequence (similar to start/stop bits in other NRZ)
- FES = Frame End Sequence (END symbol for frame – LO + HI)

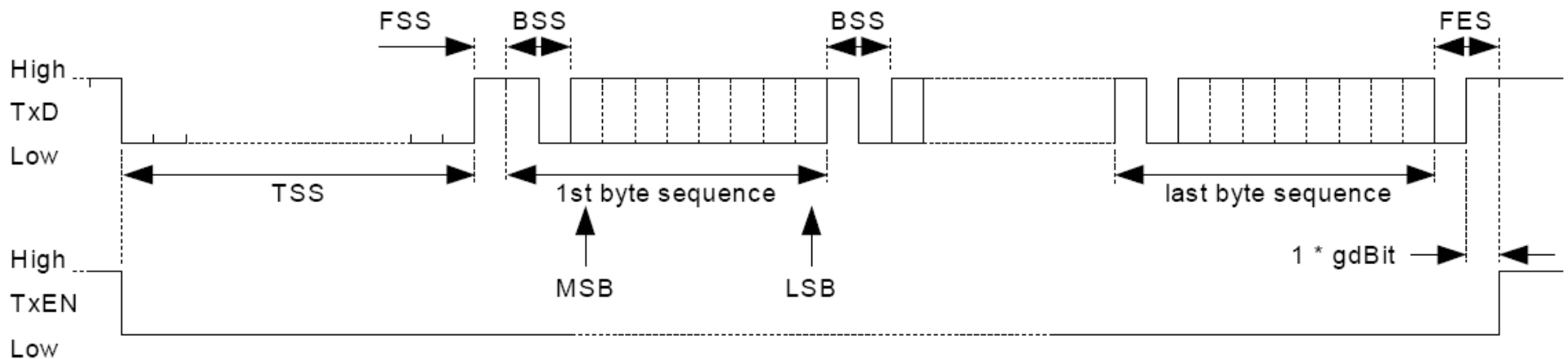


Figure 3-2: Frame encoding in the static segment.

[FlexRay04]

◆ Dynamic segment frames are similar

- Adds a DTS = dynamic trailing sequence field; helps line up minislots

FlexRay Frame Format

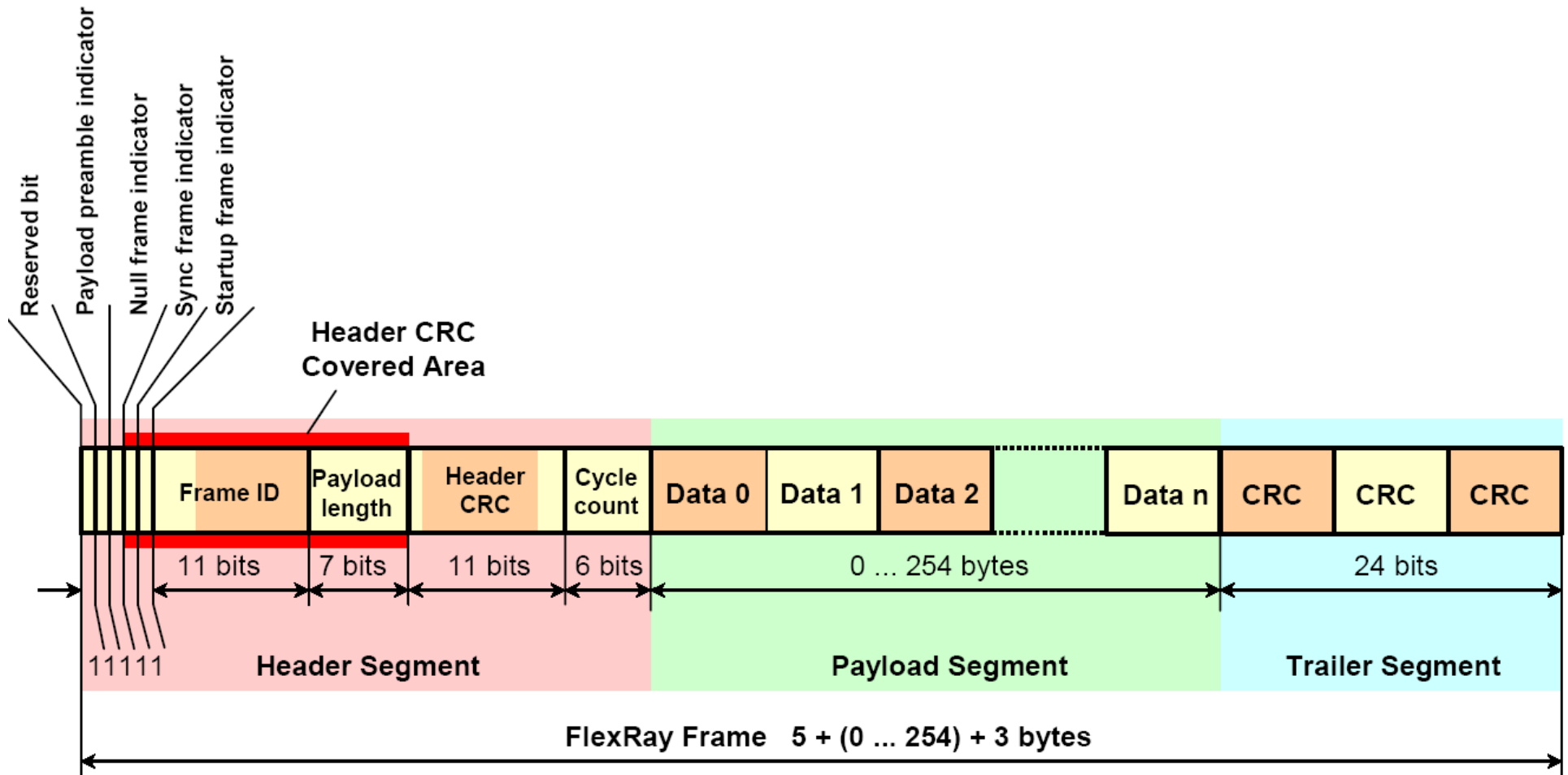


Figure 4-1: FlexRay frame format.

[FlexRay04]

- This data is encoded into NRZ bytes per the encoding format

FlexRay Frame Fields

◆ Frame ID

- Frame's slot number (1 .. 2047); unique within channel in communication cycle

◆ Payload Length

- # of 16-bit words in payload
- Must be same for all messages in static segment of communication cycle

◆ Header CRC

- HD=6 error detection for header data (optimal polynomial for 20 bits)

◆ Cycle Count

- Number of current cycle
- Even vs. odd cycle count values are used by protocol details
 - Example: clock sync corrects offset on odd cycles and rate on even cycles

◆ Data

- 0 .. 254 bytes (must be same for all static frames)

◆ CRC in trailer segment

- HD=6 up to 248 payload bytes; HD=4 above that until 508 payload bytes

FlexRay Message Cycle

◆ Two main phases: static & dynamic

- “Temporal firewall” – partition between phases protects timing of each phase

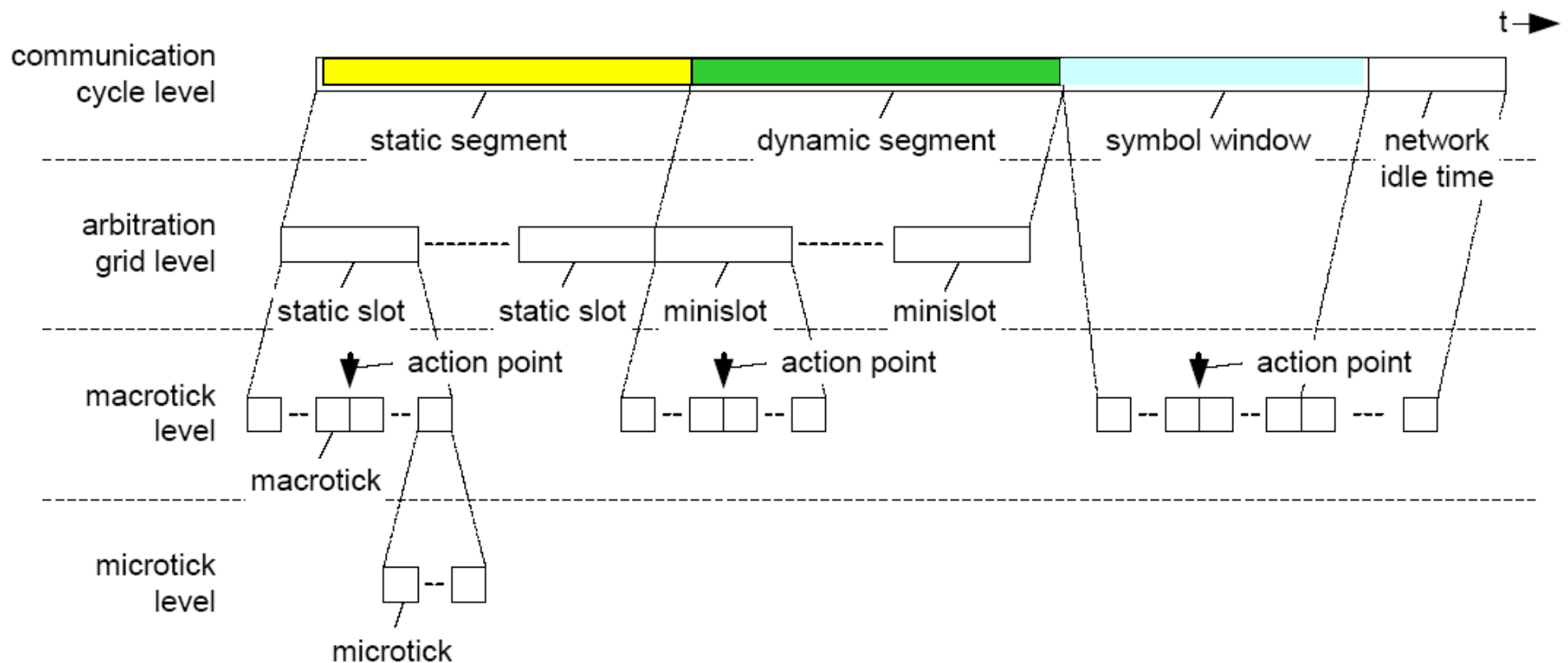


Figure 5-1: Timing hierarchy within the communication cycle.

[FlexRay04]

Microtick & Macrotick

◆ Microtick level

- Node's own internal time base
- Direct or scaled value from a local oscillator or counter/timer
- Not synchronized with rest of system – local free-running oscillator

◆ Macrotick level

- Time interval derived from cluster-wide clock sync algorithm
- Always an integral number of microticks
 - BUT, not necessarily the same number of microticks per node
 - Number of microticks varies at run time to implement clock sync

◆ Designated macrotick boundaries are “action points”

- Transmissions start here – static; dynamic; symbol window
- Transmissions end here – dynamic segment

Static Segment

- ◆ TDMA messages, most likely used for critical messages
 - All static slots are the same length in microticks
 - All static slots are repeated in order every communication cycle
 - All static slot times are expended in cycle whether used or not
 - Number of static slots is configurable for system ; up to 1023 slots

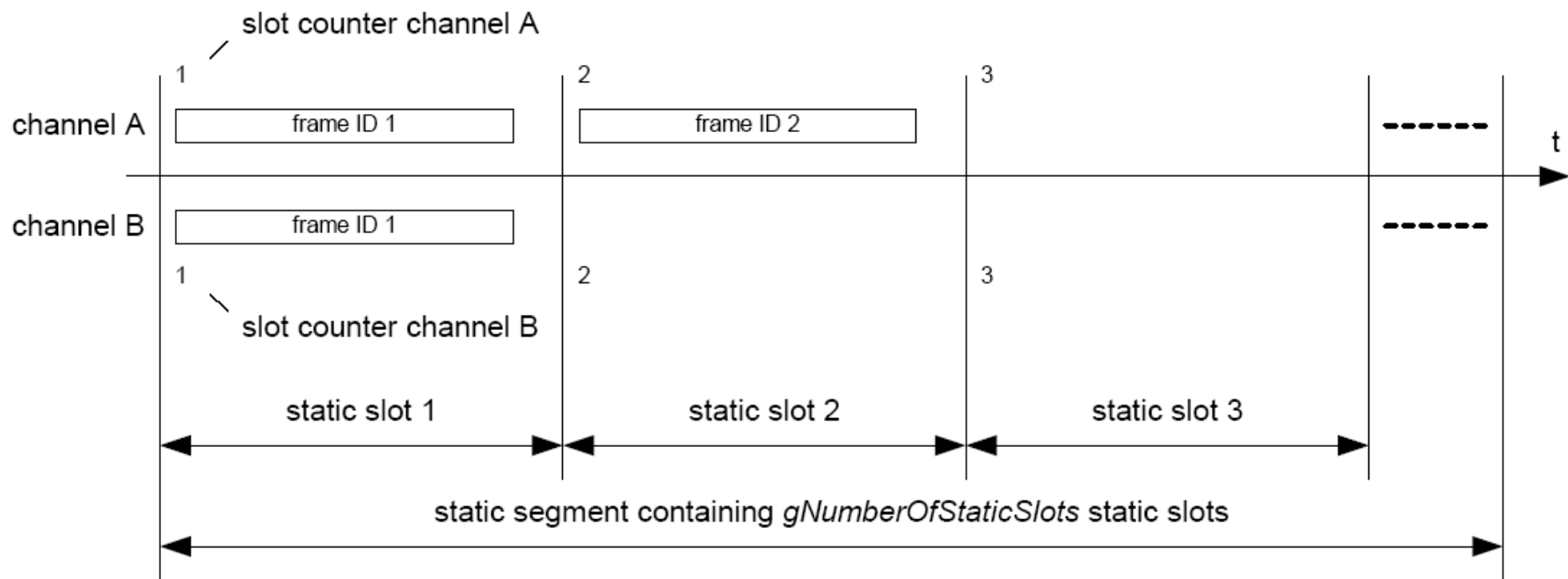


Figure 5-3: Structure of the static segment.

[FlexRay04]

Static Segment Details

◆ Two-channel operation

- Sync frames on both channels; other frames optionally 1 or 2 channels
 - Less critical/less expensive nodes might only connect to one channel
- Slots are lock-stepped in order on both channels

◆ TDMA order is by ascending frame ID number

- Frame number used to determine slot # by software
 - It is NOT a binary countdown arbitration mechanism – only one xmitter at a time
 - Optionally, there is a Message ID in the payload area that can be unrelated to slot number
 - Example use: each node uses its node # as frame # and multiplexes its messages onto a single time slot, distinguished by Message ID
- In contrast, TTP has a MEDL that can have sub-cycles
 - Need neither a Frame ID nor a Message ID
 - Extra information to be managed and coordinated

Dynamic Segment

- ◆ **High-level idea is event-based communication channel**
 - Want arbitration, but must be deterministic
 - Binary countdown not used (among other things, restricts possible media)
- ◆ **“Minislot” approach**
 - Can be thought of as a time-compressed TDMA approach (details on next slide)
 - Two channels can use independent message queues

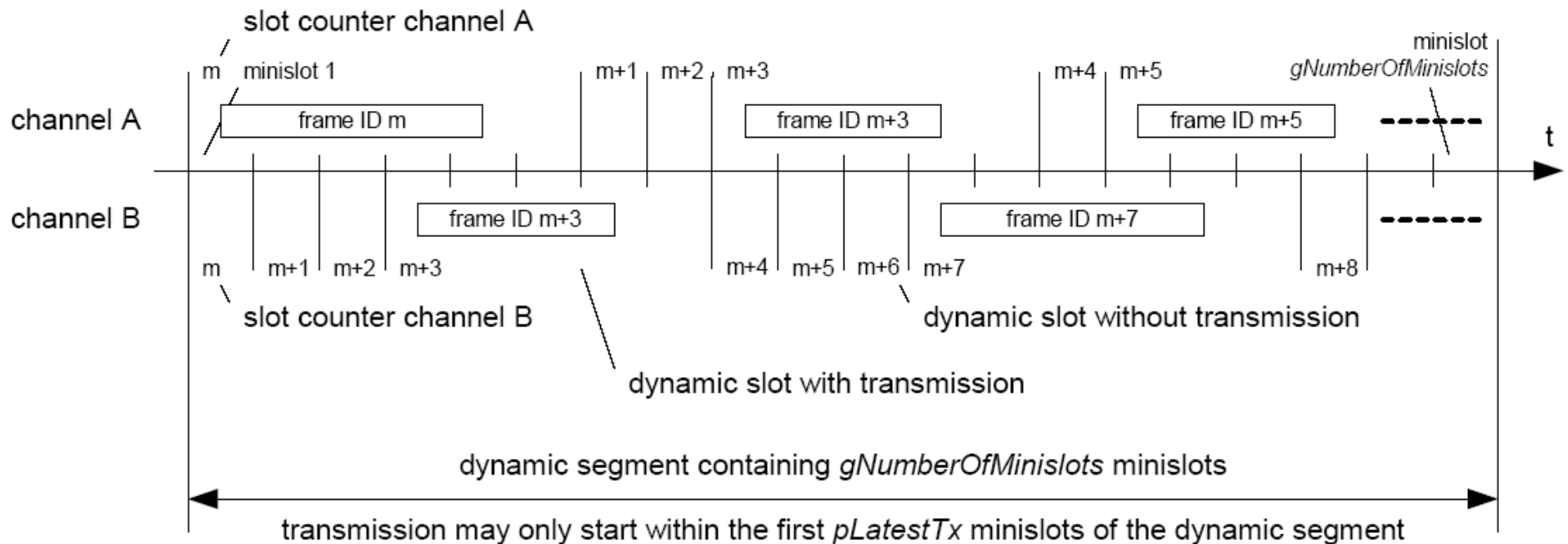


Figure 5-5: Structure of the dynamic segment.

[FlexRay04]

Dynamic Segment Details

- ◆ High-level idea is each minislot is an opportunity to send a message
 - If message is sent, minislot expands into a message transmission
 - If message isn't sent, minislot elapses unused as a short idle period
 - All transmitters watch whether a message is sent so they can count minislots

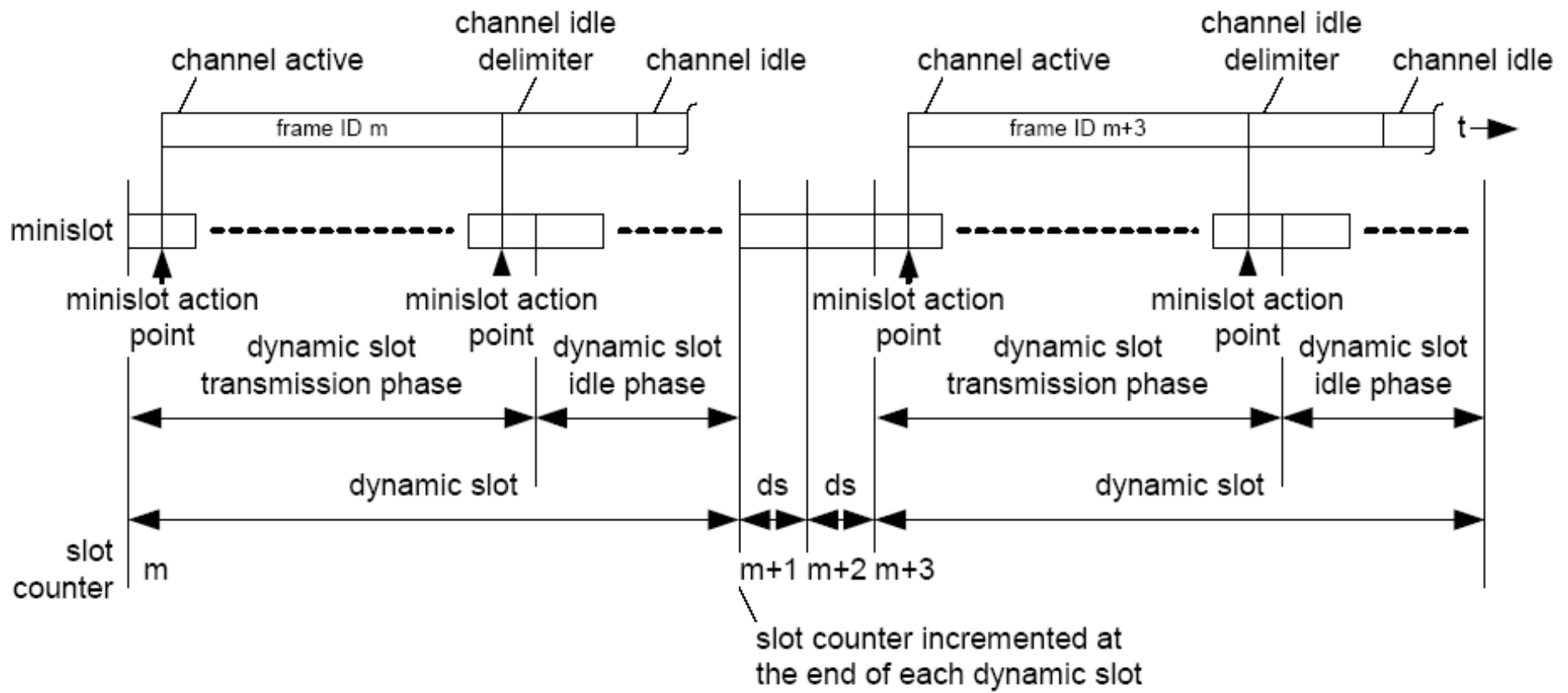


Figure 5-7: Timing within the dynamic segment.

[FlexRay04]

Minislot Performance

◆ **Frame ID # is used for slot numbering**

- First dynamic Frame ID = last static Frame ID + 1

◆ **Dynamic segment has a fixed amount of time**

- Fixed number of macroticks, divided up into minislots
- There might or might not be enough time for all dynamic messages to be sent
- When dynamic segment time is up, unsent messages wait for next cycle

◆ **Net effect: event-triggered messages**

- Messages with the lowest Frame ID are sent first
- Each Frame ID # can only send ONE message per cycle
- As many message as will fit in dynamic segment are sent
- This means that only highest priority messages queued are sent in each cycle
- Note that idle minislots consume dynamic segment bandwidth
 - But minislots are a lot smaller than messages

System Startup of FlexRay

◆ First, a wakeup procedure

- Transition controllers from “sleep” to “wake”
- (Not necessarily initiated by a coldstart node)

◆ Then, a coldstart, that actually starts the TDMA operation

- At least two coldstart nodes in system
- For systems with 3 or more nodes, ideal number is three coldstart nodes
 - More than 3 nodes might have problems with forming a single group (clique avoidance on startup works only for 3 nodes)
- Coldstart nodes are pre-designated in system

Wakeup Pattern

- ◆ **Wakeup pattern sent on one channel to alert other nodes to wake up**
 - Pattern sent multiple times in general – example shows sending it twice
 - Only sent on one channel at a time
 - Have to handle case where sending wakeup is a single-point fault
 - Wakeup isn't acknowledged by nodes
 - That is dealt with during startup

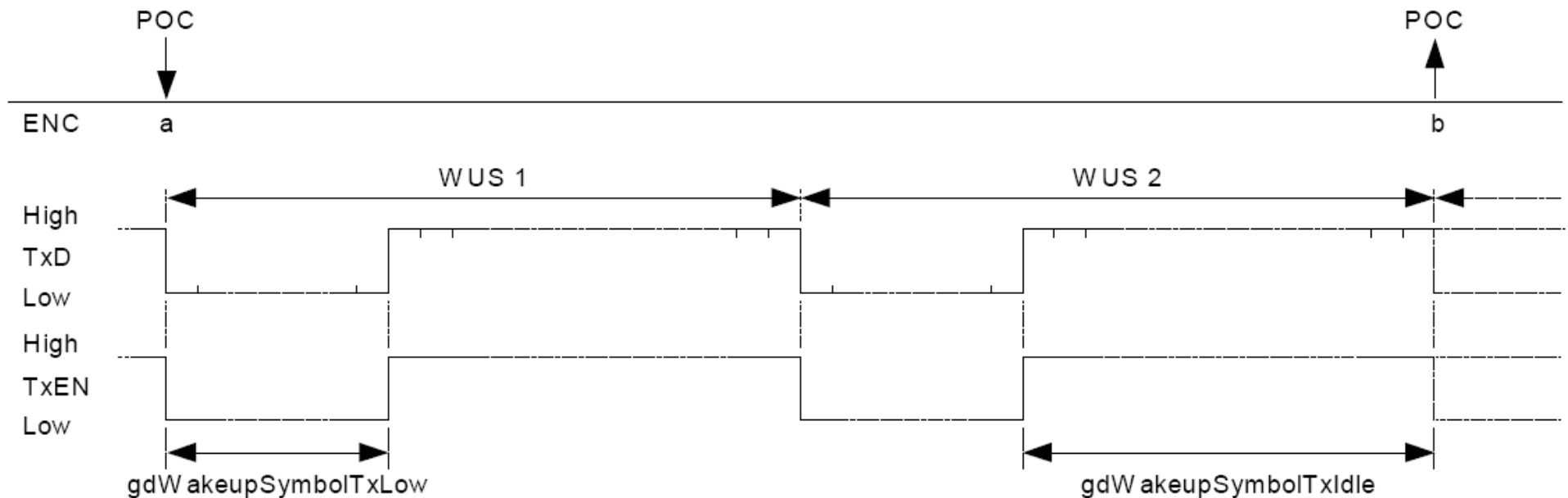


Figure 3-5: Wakeup pattern consisting of two wakeup symbols.

Wakeup Pattern Collisions

- ◆ One of the big problems with starting up a TDMA system is collisions
 - What if two nodes attempt to transmit a wakeup pattern at the same time?
 - Note that transmit and enable are both controlled together
 - “Low” part of wakeup signal drives bus
 - “High” part of wakeup signal doesn’t drive bus (TxEN is active low)
 - Thus, conflicting wakeup signals simply keep bus high for a long time and are OK
 - The other important part is that wakeup is just a signal, not a data-bearing message!
 - Quiet periods make it easier to detect collisions

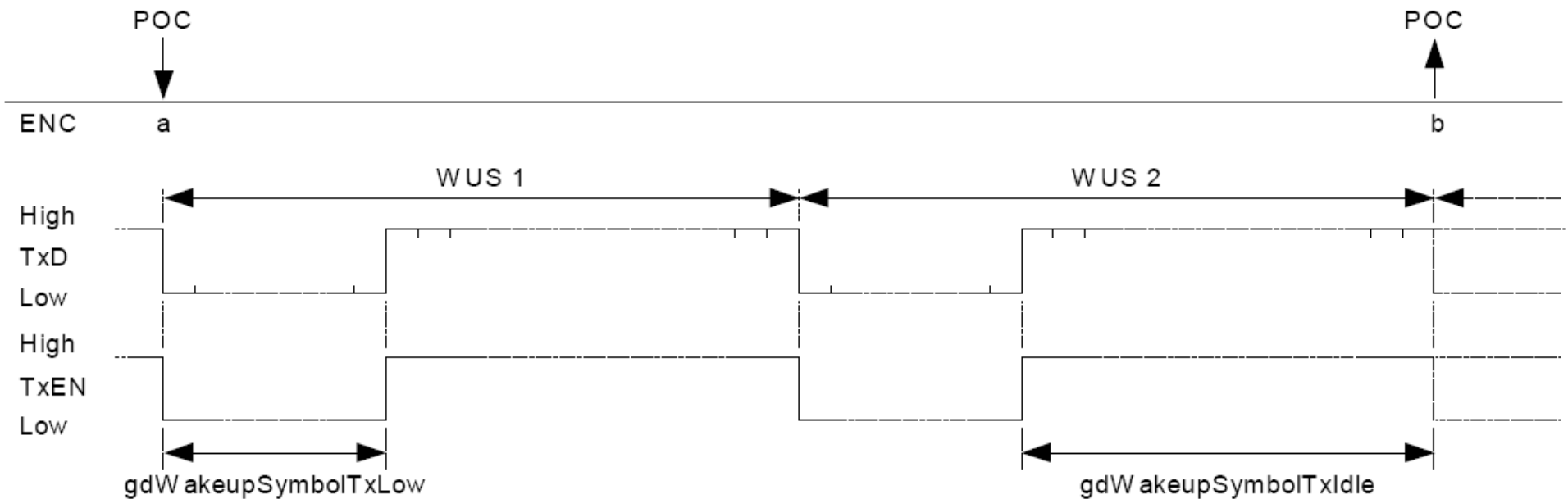


Figure 3-5: Wakeup pattern consisting of two wakeup symbols. [FlexRay04]

Fault-Free FlexRay Startup Overview

The point is – this is not simple!

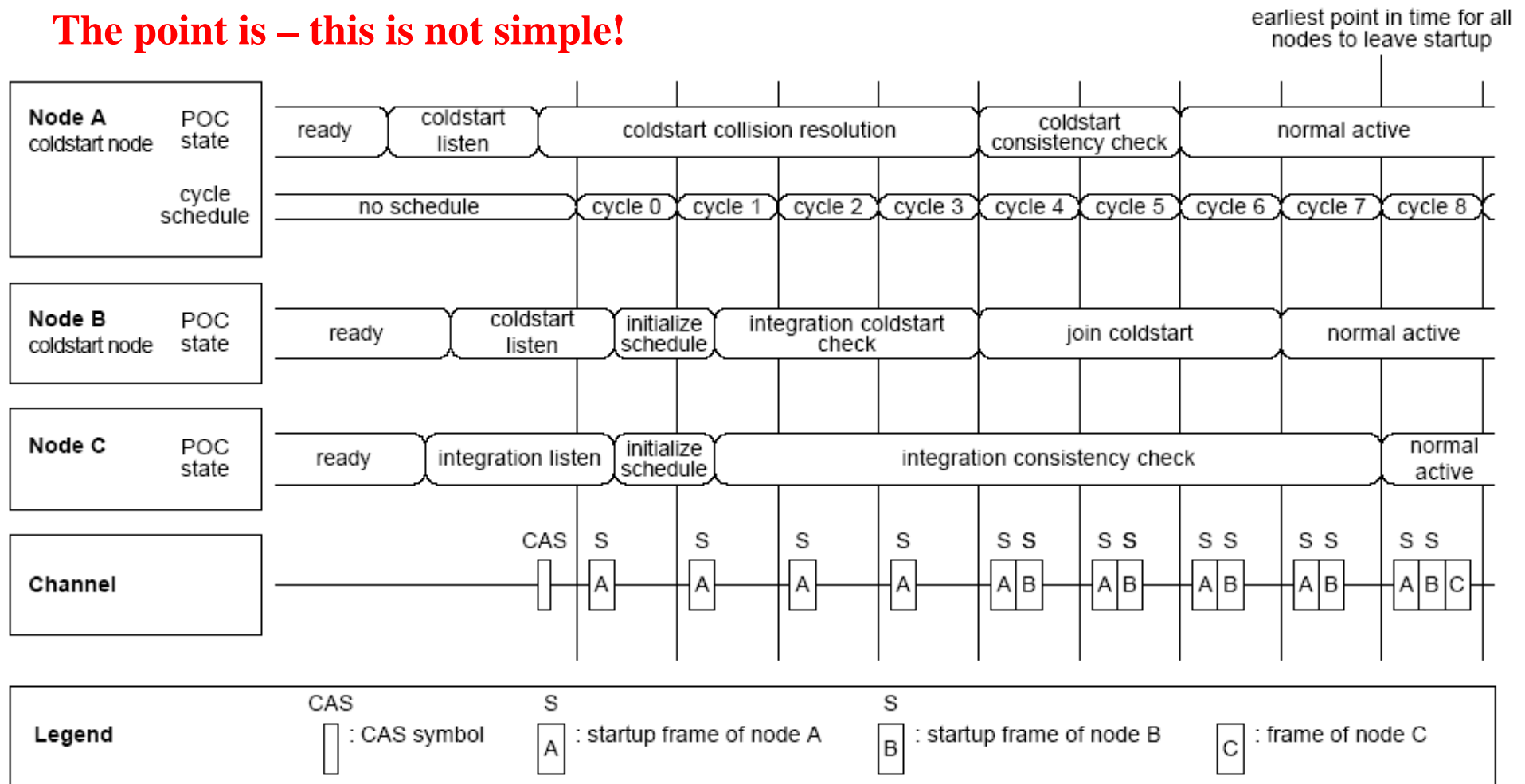


Figure 7-10: Example of state transitions for the fault-free startup⁸⁴.

[FlexRay04]

Note: “POC” = “Protocol Operation Control”, which means the control logic that runs the protocol 21

Simple Version Of Wakeup

◆ A coldstart host is powered up

- It finds out that no other wakeup is already in progress
- It sends a “wakeup” signal on one channel
- If necessary, it sends a “wakeup” signal on the other channel
 - But, forbidden to wakeup both channels simultaneously in case it is sending wakeup in error
 - Note that forbidding 2-channel simultaneous wakeup is a critical protocol property that has been imposed on the host software! So if host software is defective, protocol might not work

◆ After wakeup sent or received:

- Bus Guardian set into “guarding” mode
- Need two coldstart nodes to completely wake up, then can:

- Transition to coldstart operation

Cold Start (Happens After Wakeup)

◆ First listen

- If nothing heard and node is a coldstart node, then initiate a coldstart

◆ Send a CAS – Collision Avoidance Symbol

- (Same encoding as MTS – Media Test Symbol)

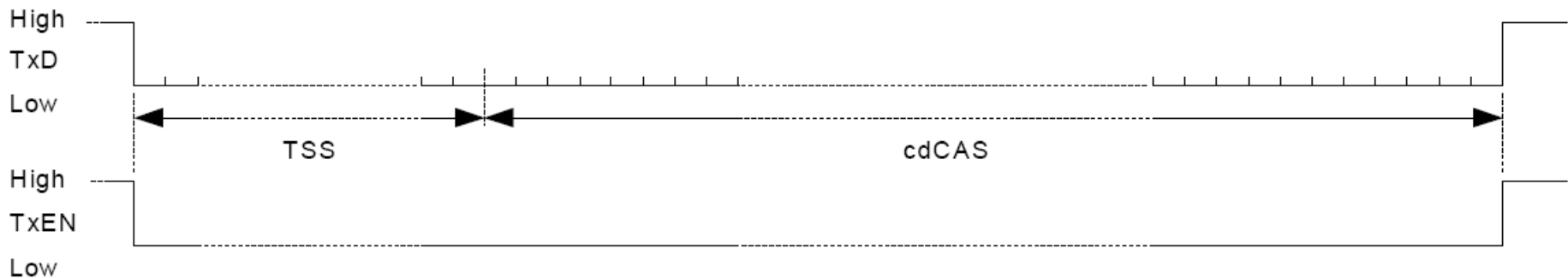


Figure 3-4: CAS and MTS symbol encoding.

[FlexRay04]

- CAS acts as a signal to notify other nodes that cold start has begun
 - Consists of TSS – transmission start sequence; and 30 more low bits CAS
 - Other nodes have to synch to that coldstart to start frame rotation

Time Keeping

◆ Macrotick is common unit of time across nodes

- Idea is that it is within one microtick of correct at each node
- Rate and offset correction performed every pair of cycles to keep in sync

◆ Two main timekeeping tasks:

- MTG – Macrotick Generation Process
 - Applies rate and offset correction values
- CSP – Clock Synchronization Process
 - Initialization
 - Calculation of rate and offset values
- Distributed time theory applies here (see lecture on that topic)
 - Uses fault tolerant midpoint calculation (how many errors does this tolerate?)

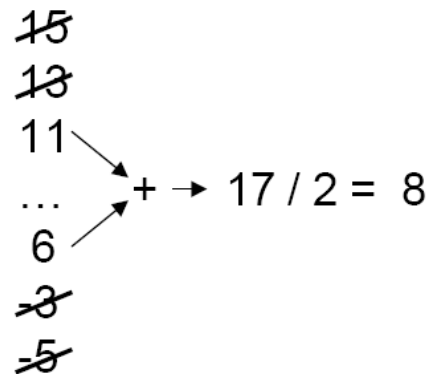


Figure 8-12: Algorithm for clock correction value calculation (k=2).

[FlexRay04]

Clock Sync Schedule

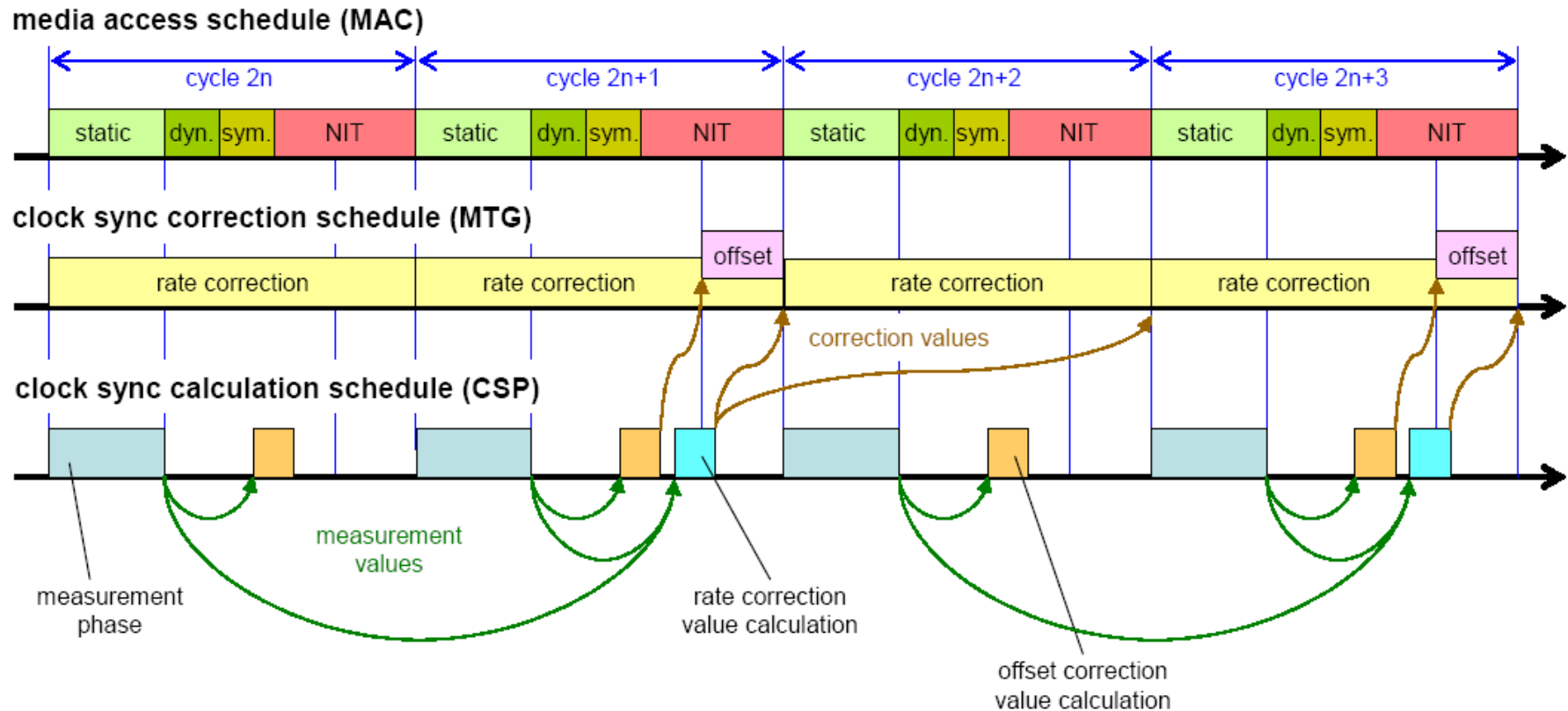


Figure 8-3: Timing relationship between clock synchronization, media access schedule, and the execution of clock synchronization functions.

[FlexRay04]

- ◆ **NIT = Network Idle Time at end of each cycle**

Clock Startup

- ◆ **Leading coldstart node sends its clock information**
- ◆ **Other nodes integrate:**
 - Receive an even-numbered startup frame
 - Record microtick it was received
 - Adopt its cycle number
 - Wait for odd-numbered startup frame
 - Since cycles are of known length, can compute offset and rate corrections

Bus Guardians

- ◆ Bus guardians prevent nodes from transmitting outside schedule
 - “optional” but as a practical matter required for critical systems

		BGSM monitoring modes			
BGSM operation modes		static segment	dynamic segment	symbol window	network idle time
STANDBY		<i>no operation</i>			
DISABLED		BG DISABLED			
WAKEUP		MINIMAL			
NORMAL			RELAXED		
	STRICT			BG DISABLED	BG DISABLED
			BG DISABLED		
REDUCED			RELAXED		
	RELAXED			BG DISABLED	BG DISABLED
			BG DISABLED		

Figure 10-1: BGSM operation and monitoring modes interrelations.

General Bus Guardian (BG) Operation

◆ Idea is to have an independent time source

- If communication controller attempts to transmit at wrong time...
bus guardian stops it because “enable” is removed outside correct time slice
- If BG is incorrect...
communication controller won't be attempting to transmit anyway
- Goal is “fail silent” operation
 - Both BG & communication controller have to enable & transmit for message to be sent

◆ Why is this required?

- What if a faulty node tries to send at the wrong time – takes down network!
 - Especially “babbling idiot” failure, where node broadcasts continuously
- It is very difficult to get this right at low cost
 - Ideally want separate chips for BGs to eliminate common mode failures
 - As a practical matter, want to integrate on chip to save cost

FlexRay Tradeoffs

◆ Advantages

- Probably has primitives necessary for critical x-by-wire applications
- Static segment provides timing guarantees and some fault tolerance
- Dynamic segment gives flexibility for event triggered messages
- Big industry consortium behind it
- It's "flexible"

◆ Disadvantages

- Relatively new protocol – these things take time to mature
 - For example, no fault hypothesis published upon which to build safety case
- Does not provide as complete a set of primitives as TTP
 - Group membership is an application problem, but will be needed for x-by-wire
 - Any safety critical operation on host might complicate safety case

◆ Other

- Does not encompass a complete system architecture
 - Provides flexibility for architectures... but not a blueprint for fault tolerance

Relationship To Selected Other Topics

◆ **Distributed systems:**

- Enables hybrid Time Triggered & Event Triggered designs
- Requires application to do their own atomic broadcast & group membership
- Built-in distributed timekeeping results (synchronous system approach)

◆ **Embedded networks:**

- Uses combination of TDMA and minislot (implicit token/compressed TDMA) approaches

◆ **Real time:**

- Requires both static scheduling (static portion) and dynamic scheduling (dynamic portion)

◆ **Fault tolerance:**

- Requires application support for Byzantine faults (e.g., group membership)
- Includes data integrity checks on header & payload
- Includes no security – that is an application responsibility
- Includes some support for system reset, but host must behave properly

◆ **Safety – FlexRay consortium is working on protocol analysis**

- Requires a safety case, including fault analysis