FlexRay communication protocol
(Wakeup and Startup)

Sergey Kosov.

Contents

1. Introduction

The FlexRay communication protocol is specified for a dependable automotive network. The protocol was developed for providing fault-free communication between various electronic components of communication system. Such a system, which consists of finite quantity of electronic components capable of exchanging data, is called *distributed system*. And the electronic control units we shall call *nodes*. A distributed system, which nodes are connected via at least one communication channel directly like in bus topology or by star couplers like in star topology is called *cluster*.

This article describes the wakeup and startup processes of a FlexRay cluster and the integration of newly configured nodes into a FlexRay cluster that is already in operation.

The FlexRay communication protocol uses time-division multiple access to the media for providing communication between nodes. It means that every node must has at least one unique predefined slot of time – *communication slot*; during this interval of time the node has access to the media for sending its data. To send the data to the media, there is a structure used by the communication system to exchange information within the system. We shall call this structure *communication frame*.

Since every node as a logical entity is capable of sending and/or receiving frames, every electronic control unit consists of several parts. They are the communication controller, the bus driver and the host, and are described in section 2.2.

We shall consider the problem of beginning communication in flexray cluster. The solution to this problem is two processes called wakeup and startup. The wakeup process is for wakeup the cluster and prepares all the nodes for starting communication. The wakeup process is described in chapter 3. Since flexray uses time-division multiple access to the media, all the nodes must be synchronous for participate in data exchange process. The startup process provides the cluster with conditions to become synchronous and to start communication. The wakeup process is described in chapter 4.

The startup process distinguishes two types of nodes – coldstart nodes and non-coldstart nodes. *Coldstart node* is a node capable of initiating the communication startup procedure on the cluster by sending special frames, which header segment contains an indicator that integrating nodes may use timer related information from this frame for initialization during the startup process. We call such frames *startup frames*.

2. Wakeup and Startup

2.1 Introduction

First, the cluster wakeup is described in detail. The current chapter provides us with some information about the interaction between communication controller and host. Afterwards, communication startup and reintegration, the process when a node reintegrates into the working cluster is described. Chapter 4 also describes the reintegration of nodes into a communication cluster.

2.2 Example node architecture

A node consists of one *communication controller* which is an electronic component in a node that is responsible for implementing the protocol aspects of the FlexRay communications system, one *host* that is the part of a node where the application software is executed and up to two *bus drivers* which is represented by electronic components consisting of a transmitter and a receiver that connect a communication controller to one communication channel. Each communication channel has one bus driver to connect the

node to the channel. These components as well as its logical interfaces are depicted in figure 2-1.

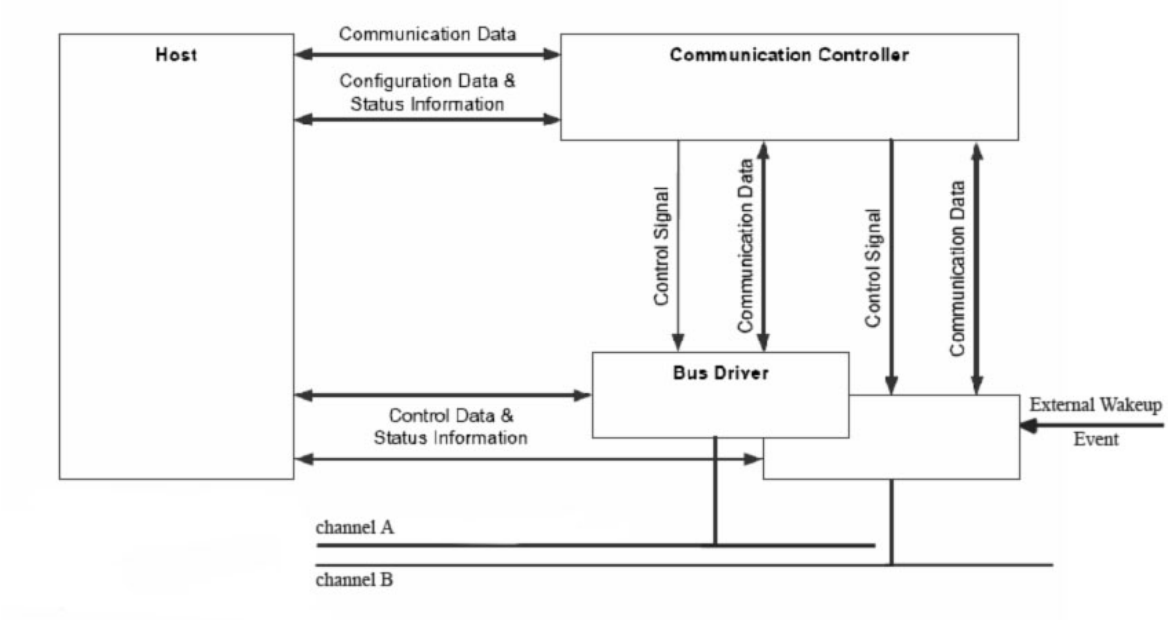

Figure 2-1: Node architecture.

The host and the communication controller exchange a huge amount of data through its interfaces: the host provides control and configuration information to the communication controller, and provides payload data that is transmitted during the communication cycle. The communication controller provides status information to the host and delivers payload data received from communication frames.

The interface between the bus driver and the communication controller consists of three digital electrical signals. Two are outputs from the communication controller that are called “transmit data” – TxD, “transmit data enable not” – TxEN and one is an output from the bus driver “receive data” – RxD.

The communication controller uses the TxD signal to transfer the actual signal sequence to the bus driver for transmission onto the communication channel. TxEN indicates that the communication controller wants the bus driver to send the data from the TxD line to its corresponding channel. The bus driver uses the RxD signal to transfer the actual received signal sequence to the communication controller.

The interface between the bus driver and the host allows the host to control the operating modes of the bus driver and to read error conditions and status information from the bus driver.

The minimum prerequisite for a cluster wakeup is that the receivers of all bus drivers are supplied with power. A bus driver has the ability to wake up the other components of its node when it receives either a wakeup pattern on its channel or external wakeup event. The bus driver must have ability to distinguish these wakeup sources by checking if it was wakeup pattern on its channel or locking electrical circuit.

2.3 Protocol operation control

At the figure 2-2, we depict the states of the protocol operational control process as well as the place of wakeup and startup states. Let us consider the purpose of these states and causes of switching between them in general.

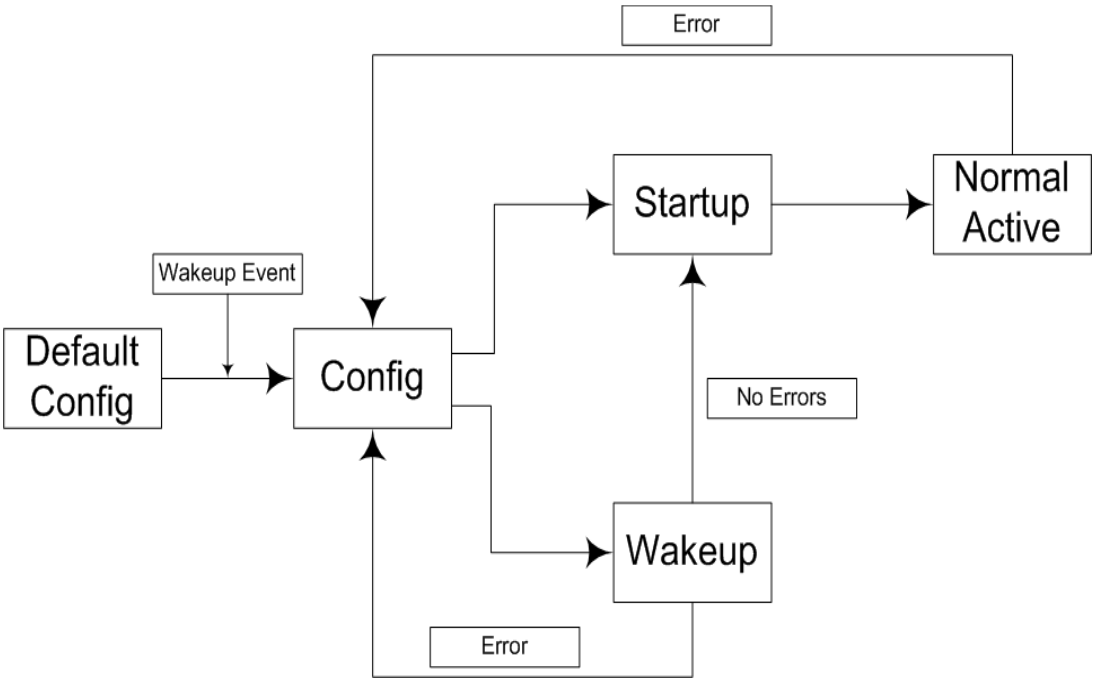

Figure 2-2: Overview of protocol operation control.

The node shall enter the default config state when the node is supplied with power. In this state the default, static configuration of the communication controller is performed. Host configures number of channels, media specification, etc. The state is left as soon as the bus driver receives a wakeup event which can be an external wakeup event or receiving the wakeup pattern on its channel.

In the config state, the host configures the communication controller with dynamic – configuration data. Then the host checks what kind of wakeup event is received – if it was external or internal – by receiving wakeup pattern from another node, and based on this information decides what state will be next. If the bus driver has received an external wakeup event, the host must command to wakeup other nodes on an attached channel. Otherwise, if the bus driver has received a wakeup pattern from the channel, the host may command to wakeup the second channel (if the node is attached to both channels), or to proceed to startup state.

In the wakeup state, the node tries to wakeup the defined channel. If errors occur during the wakeup state, the communication controller informs the host and the node switches to the config state. Otherwise the node switches to startup state. This state as well as meaning of “wakeup” will be described in chapter 3.

In the startup state, the node checks for ongoing communication on the media. If the bus driver detects communication signals on the attached channel, then the node integrates to the cluster communication, otherwise the node starts the startup procedure. This state will be described in detail in chapter 4. After startup, node proceeds to the normal active state. If errors occurred, the node comes to config state and tries the whole procedure again.

3. Wakeup

3.1 General notes

As it was written, the bus driver has the ability to wake up the other components of its node when it receives a wakeup pattern on its channel. So, at least one node in the cluster needs an external wakeup source.

The host completely controls the wakeup procedure. The communication controller provides the host the ability to transmit a special wakeup pattern on each of its available channels separately.

The wakeup pattern must not be transmitted on both channels at the same time. This is done to prevent a faulty node from disturbing communication on both channels simultaneously with the transmission. The host must configure, which channel the communication controller shall wake up. The communication controller ensures that ongoing communication on this channel is not disturbed.

The wakeup pattern then causes any fault-free receiving node to wake up if it is still asleep. Generally, the bus driver of the receiving node recognizes the wakeup pattern and triggers the node wakeup. The communication controller needs to recognize the wakeup pattern only during the wakeup (for collision resolution) and startup phases.

The communication controller cannot verify whether all nodes connected to the configured channel are awake after the transmission of the wakeup pattern since these nodes cannot give feedback until the startup phase. The host shall be aware of possible failures of the wakeup and act accordingly. The wakeup procedure supports the ability for single-channel devices in a dual-channel system to initiate cluster wakeup by transmitting the wakeup pattern on the single channel, to which they are connected. Another node, which has access to both channels, then assumes the responsibility to wake up the other channel and transmits a wakeup pattern on it. The wakeup procedure tolerates any number of nodes simultaneously trying to wake up a channel and resolves this situation such that eventually only one node transmits the wakeup pattern; for more details see section 3.2. Additionally, the wakeup pattern is collision resilient; so even in the presence of a fault causing two nodes to simultaneously transmit a wakeup pattern, the signal resulting from the collision can still wake up the other nodes.

3.2 Wakeup pattern

Since the FlexRay protocol is independent from the underlying physical layer, the description in this article assumes a binary medium with two distinct levels, called HIGH and LOW. Wakeup pattern consists of the defined number of the wakeup symbol. A wakeup symbol is composed of n bits that are transmitted at a LOW level followed by m bits of ‘idle’. An example of a wakeup pattern formed by a sequence of two wakeup symbols is shown in figure 3-1.

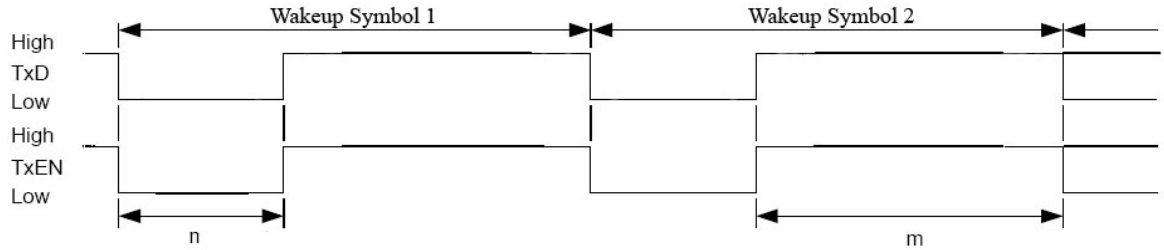

Figure 3-1: Wakeup pattern.

So the wakeup pattern has such a structure, that even if several nodes transmit it, wakeup pattern will not be destroyed, and other nodes will receive it like it was transmitted only by one node.

3.3 Wakeup state overview

At the figure 3-2, the sub states of the wakeup state are described. Let us consider the purpose of these sub states and causes of switching between them. Upon completing the wakeup procedure, the communication controller shall signal to the host the result of the wakeup attempt by initializing the *result value variable*. We shall also consider this result value variable. It contains "UNDEFINED" value, if wakeup procedure has not been performed yet.

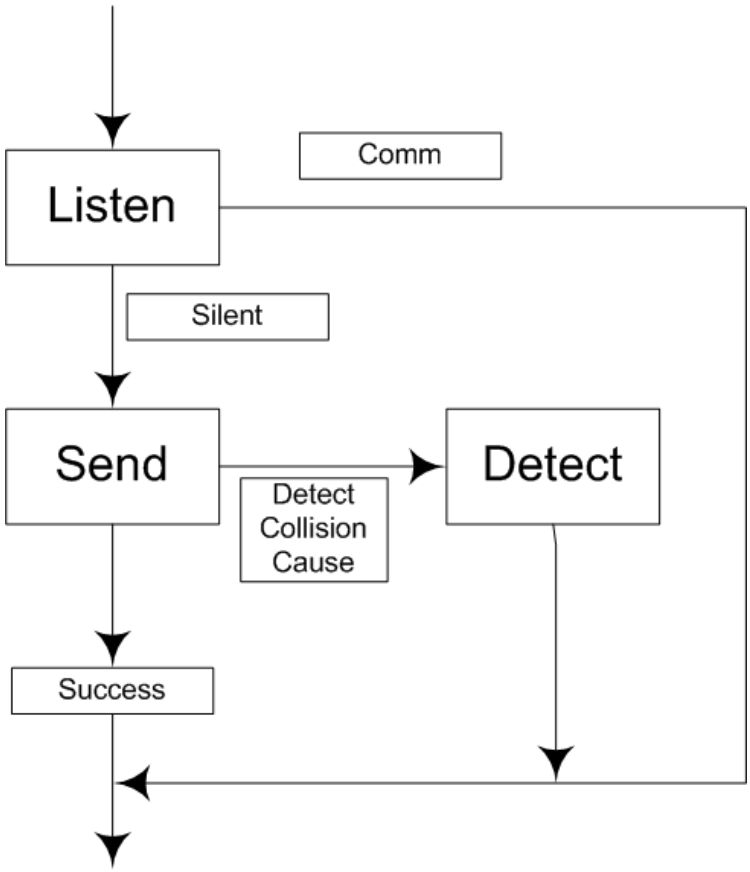

Figure 3-2: Wakeup state diagram.

In the listen state node listens predefined time to the media. This predefined time is at least more than one wakeup procedure delay long; to be sure that the channel is idle. The purpose of this state is to inhibit the transmission of the wakeup pattern if existing communication or a startup is already in progress. When ongoing communication is detected or a wakeup of the channel is already in progress, the wakeup attempt is aborted. If the communication controller has received a frame header without coding violation on either channel during the initial listen phase, the status variable will be set to the "RECEIVED_HEADER" value. If the communication controller has received a valid wakeup pattern on the channel during the initial listen phase, the status variable will be set to "RECEIVED_WUP" value.

In the send state the node transmits the wakeup pattern on the configured channel and checks for collisions. Since the communication controller transmits the wakeup pattern on the channel, it cannot really determine whether during its transmission another node sends a wakeup pattern or frame on the channel. Only during the idle portions of the wakeup pattern the node can listen to the channel. If activity is detected during one of these idle portions, the communication controller leaves the send phase and enters a succeeding monitoring phase (detect state) so that the cause of the collision might be identified and presented to the host. If the wakeup pattern is successfully transmitted and no collisions are detected, the status variable will be set to the "TRANSMITTED" value.

In the detect state, the node attempts to discover the reason for the wakeup collision that was encountered in the send state. The node listens to the media during predefined time. This predefined time like in the listen state is at least more than one wakeup procedure delay long. Either the detection of a wakeup pattern, indicating a wakeup attempt by another node or the reception of a frame header indicating existing communication, causes leaving of the state. If the communication controller has received a frame header without coding violation on either channel during the initial listen phase, the status variable will be set to the "COLLISION_HEADER" value. If the communication controller has received a valid wakeup pattern on the channel during the initial listen phase, the status variable will be set to "COLLISION_WUP" value.

There is also one more possible value for the status variable. The status set to "COLLISION_UNKNOWN" if the communication controller has detected a collision, but it could not recognize collision's cause (the two preceding cases).

After leaving the wakeup state, the host accepts only the "COLLISION_UNKNOWN" value in the wakeup status variable as a critical error. It means that we have some signal on the bus, but we can not recognize it. It might be strong noise, breakage of media, etc. In this case the host may try to command the node to reenter wakeup state, or it should have some defined error-recovery mechanism.

4. Startup

4.1 In general

Before communication startup can be performed, the cluster has to be awake. The startup is performed on all channels synchronously. The action of initiating a startup process is called a coldstart. Only a limited number of nodes may initiate a startup, they are called the coldstart nodes. A coldstart attempt begins with the transmission of a collision avoidance symbol (CAS). Only the coldstart node that has transmitted the CAS can transmit startup frames in the first four cycles after the CAS. It is then joined firstly by the other coldstart nodes and afterwards by all other nodes.

Each startup frame shall also be a sync frame; therefore each coldstart node will also be a sync node. At least two fault-free coldstart nodes are necessary for the cluster to start up because non-coldstart nodes require at least two startup frames from distinct nodes for integration.

A coldstart node that actively starts the cluster is also called a leading coldstart node. A coldstart node that integrates upon another coldstart node is also called a following coldstart node. During startup, a node may only transmit startup frames. Any coldstart node shall wake up the cluster or determine that it is already awake before entering.

The system startup consists of two logical steps. In the first step dedicated coldstart nodes start up. In the second step the other nodes join the coldstart nodes. Let us first consider the behavior of coldstart nodes in startup.

4.2 Startup of coldstart nodes

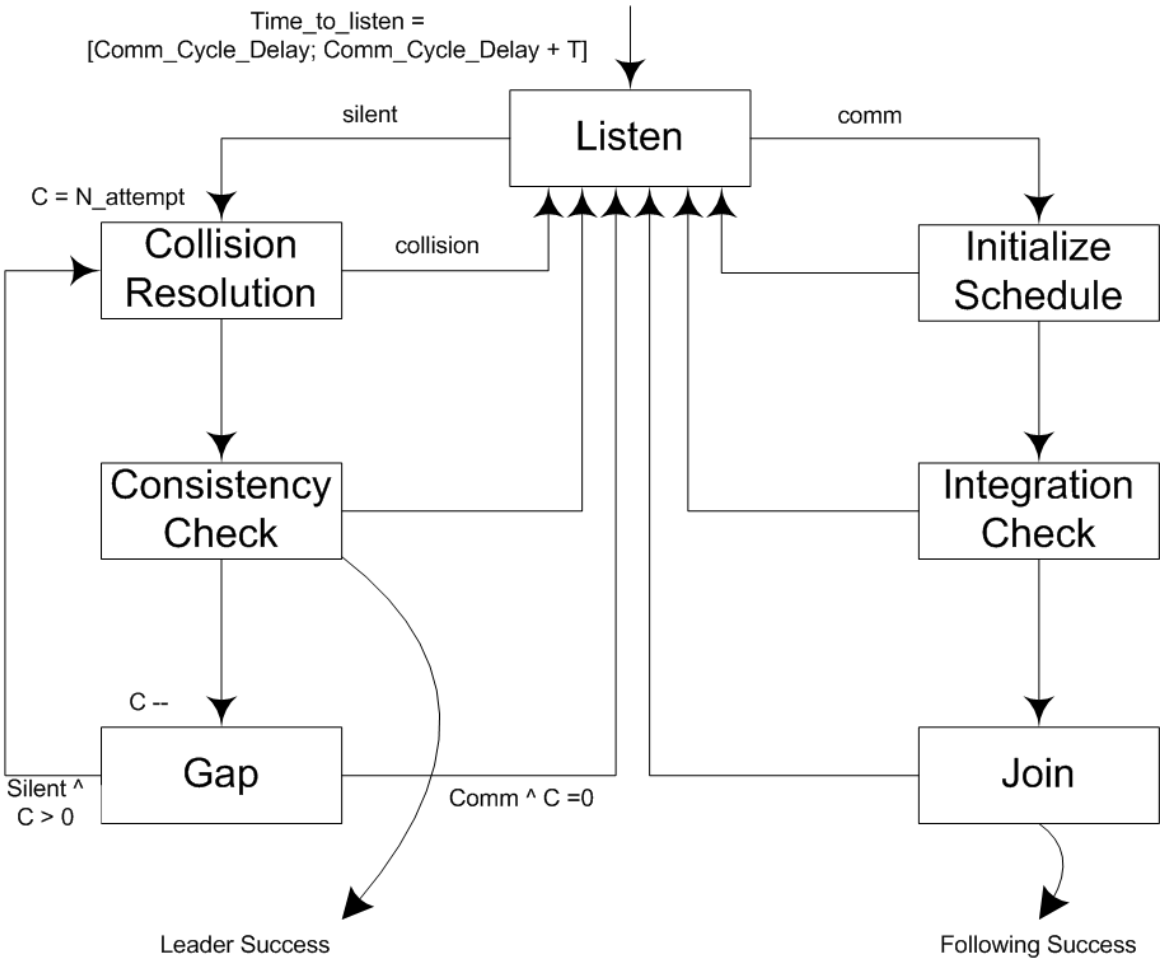

Figure 4-1: Startup state diagram (coldstart nodes).

Startup procedure is performed by the election of a leader coldstart node, begins a startup attempt and only then following colstart nodes join the leader. The purpose of listen and collision resolution states is to elect a leader among the coldstart nodes. Let us consider them first, see figure 4-1.

In the listen state, a node listens for a predefined time that must be greater than one communication cycle delay to be sure that the channel is idle, to the media in order to detect ongoing frame transmissions and coldstart attempts. This state is left for the initialize schedule state as soon as a valid startup frame has been received. And the node tries to integrate on the node that has transmitted this frame (following coldstart node). When no communication can be detected for defined time duration, which is Time_to_listen at the figure 4-1, the node initiates the coldstart and enters the collision resolution state.

In the collision resolution state, the node begins its startup attempt. First of all, it initializes its own schedule. Then the node transmits to the channel one collision avoidance symbol (CAS), followed by four startup frames, corresponding to its initialized schedule. All the time except time slots of sending information to the media, the node listens to the channel for collisions. Only the leading coldstart node can proceed to the consistency check state. In case of two nodes of a cluster come to collision resolution state at the same time, one of them will detect collision in idle portion of its transmitting.

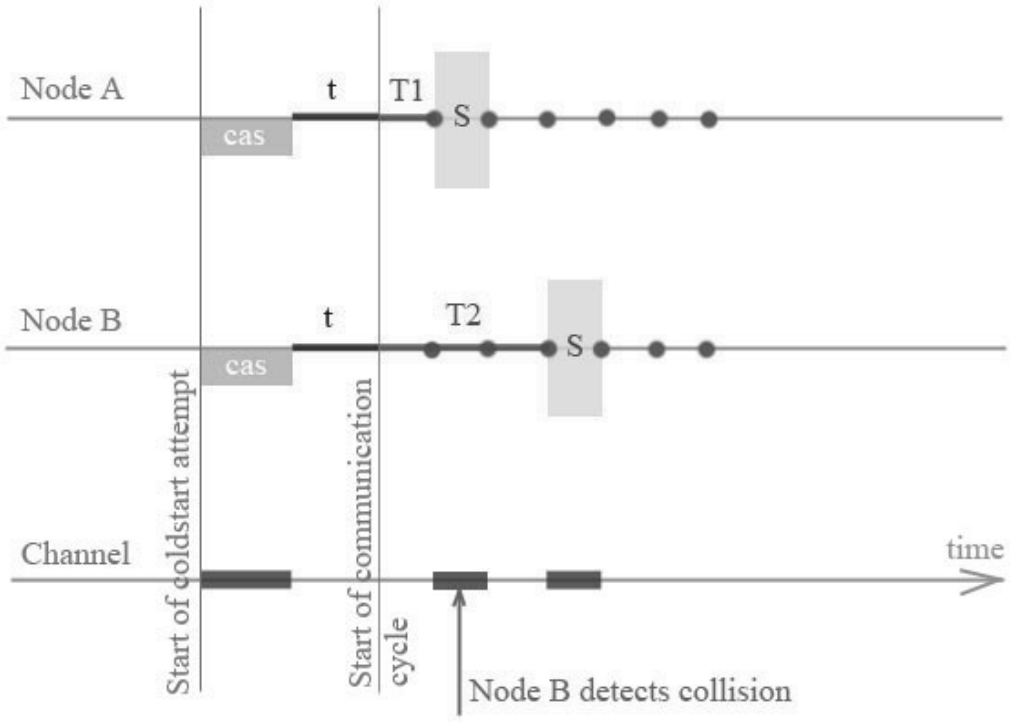

Figure 4-2: Collision avoidance symbol.

Let us consider the graph at the figure 4-2. The figure depicts two coldstart nodes which came to the collision resolution state at the same time. T1 is the time duration between the start of communication cycle and the first startup frame of node A. T2 is the time duration between the start of communication cycle and the first startup frame of node B. Time duration t is between CAS and the start of communication cycle. First of all, $T1 \neq T2$, then $(t+T1) \neq (t+T2)$. Two cases take place. In the first case, CAS of two coldstart nodes, that wants to become leaders, are transmitted simultaneously. Therefore startup frames will not be transmitted at the same time, because every node has a unique time slot for sending its frames. Hence, the node that is scheduled to transmit its startup frame later, will detect a collision as soon as it sees the startup frame of another node (node B at the graph). Even if the clock of node B occurs too fast and the first startup frames will be at the same time – following frames will have deviation in time and collision will be detected, because the second startup frame of node B will be send earlier than the second startup frame of node A.

In the second case, CAS symbols are transmitted at different times. In this case the collision will be detected even sooner.

In the initialize schedule state only following coldstart nodes come. Following coldstart nodes come to this state, after they had received one valid startup frame in the listen state. In the initialize schedule state, a node waits for the second startup frame to derive the schedule from two frames. A following node needs at least two startup frames received to find out about the leader's schedule and to slave to it. If a valid frame is received, the node comes to the integration check state, otherwise back to the listen state.

In the integration check state, the clock correction is activated. Node waits for two more startup frames to check its own clock correction mechanism. If the clock correction has success with these new frames, the node comes to the join state, otherwise back to the listen state. The purpose of this state is to eliminate faulty nodes before they start sending theirs own startup frames that are not corresponding to the leader's ones. Nodes, which have faulty clocks, will never get success with applying clock correction algorithm.

In the send state, a following coldstart node starts sending its own startup frames. If the node still receives valid startup frames from other nodes, the startup procedure is left for normal active.

Let us now return to the leading coldstart node. In the consistency check state, the leading coldstart node is still sending startup frames. The node waits for other nodes answering with theirs startup frames. If the node receives no answer during a defined time, it means that other nodes are not ready yet. In that case, the node leaves the consistency check state for the gap state to wait for other nodes get ready for startup. If the node receives an invalid answer – startup frame which does not match its own schedule, the state is left for the listen state. If the node receives valid answers – startup frames matching its own schedule, the startup procedure is left for normal active.

In the gap state, the leader node looses its leader status, because other nodes may be trying to become a leader at that time. Hence, the node stops transmitting its startup frames and waits a defined time silently, but listening to the channel. If no communication could be detected during that time, the state is left for the collision resolution state. If communication is detected or it was too many loops "gap -> consistency check -> gap" the state is left for the listen state.

4.3 Startup of non-coldstart nodes

The path of a non-coldstart node is not that different from a coldstart node's path. Let us consider figure 4-3. There are depicted only three states, which were already described, but with a small difference:

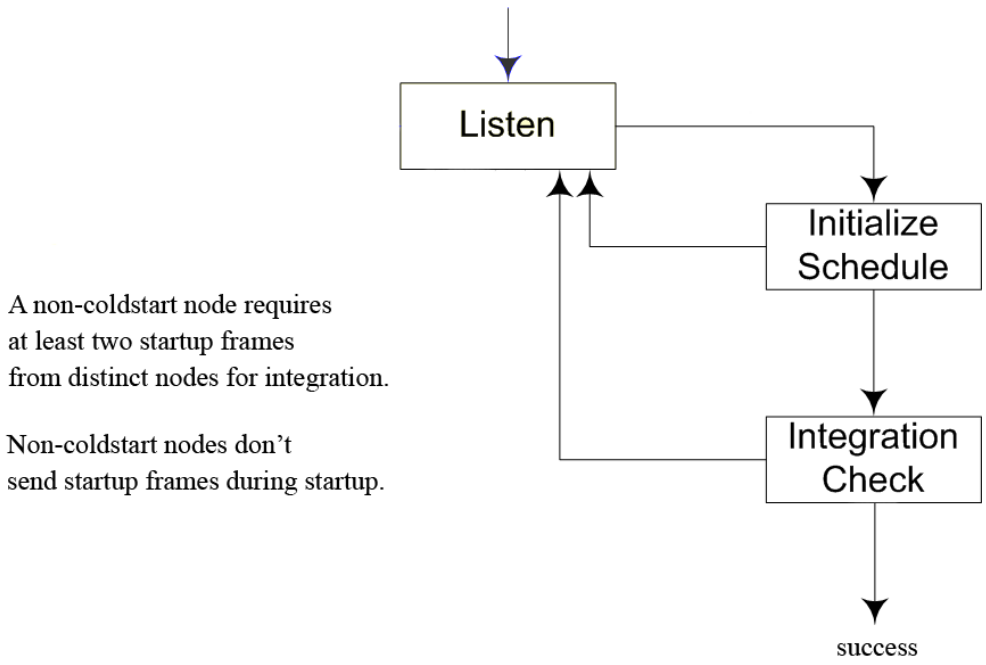

Figure 4-3: Startup state diagram (non-coldstart nodes).

4.4 Reintegration

As we have seen at figure 2-2 in case of critical errors it is possible that a node comes from the normal active state back to the config state. Such kind of a recovery mechanism leads the node to be reintegrated into the working cluster. Startup procedure for the reintegrating clusters is the same as it was described in the current chapter.

4.5 Summary

To sum up all sub states of startup state, let us consider the graph, depicted at the figure 4-4. There are three nodes shown: two coldsart nodes and one non-coldstart node. We assume that node A become the leader node. We can study all described states with time flow at this graph.

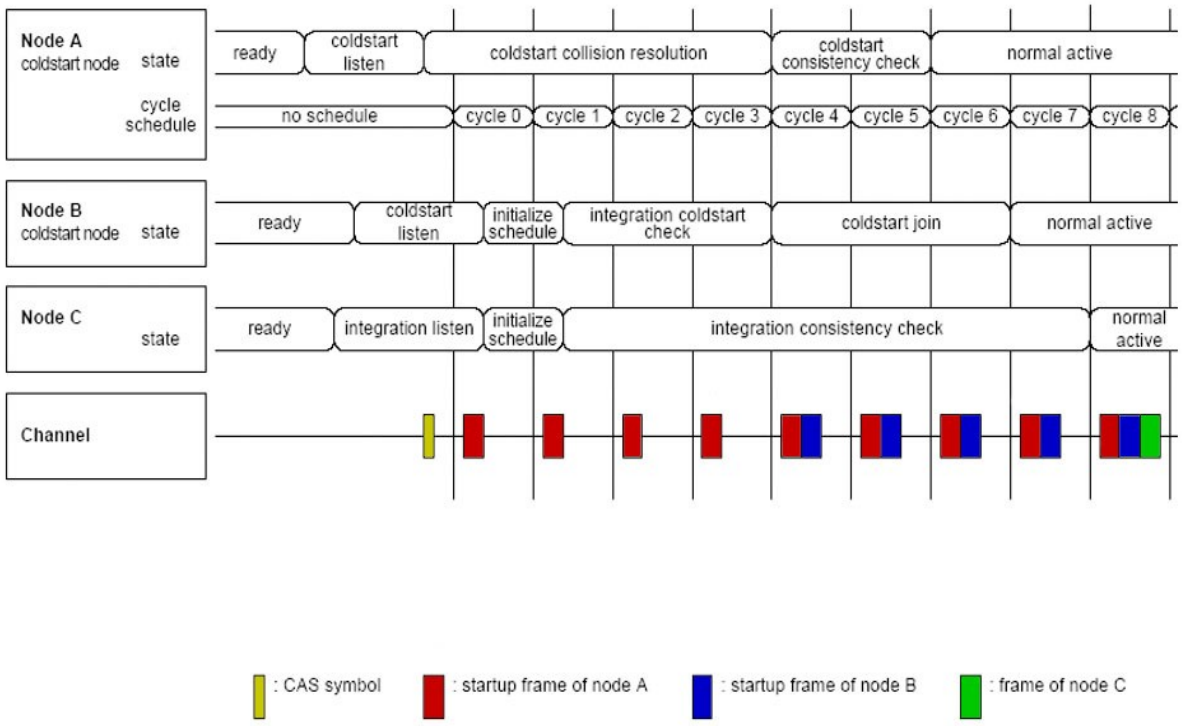

Figure 4-4: Example of state transitions for startup.