# Flex Ray: Coding and Decoding, Media Access Control, Frame and Symbol Processing and Serial Interface

Michael Gerke

November 24, 2005

## Contents

# 1 Introduction

The FlexRay Communication Protocol provides time triggered communictaion between different electronic systems in cars. The protocol should tolerate minor errors. The lower levels of the FlexRay Communication Protocol have to solve a serious problem: communication between *electronic control units* with different, potentially asynchronous clocks.
In the following, a more detailed description of these lower levels will be given.

## 1.1 Structure of the document

The FlexRay Communication Protocol is divided in different parts. These parts communicate and work together. Each part has a different functionality.
I will go into detail about three parts of the FlexRay Communication Protocol:

- the *media access control* (MAC)
- the *frame and symbol processing* (FSP)
- the *coding and decoding* (CODEC)

The latter one will be explained further by presenting it in form of a model, the *Serial Interface*, and proofs giving some arguments showing that the communication will work under certain conditions and restrictions.

## 1.2 Remarks

The MAC controls the CODEC and the FSP. Figure 1 gives an overview about the relations between the three parts, the bus and the other parts. The arrows indicate communictaion between the parts.
In order to communicate, an electronic control unit (ECU) sends a message to all other ECUs that are attached to the bus. The MAC tells the CODEC

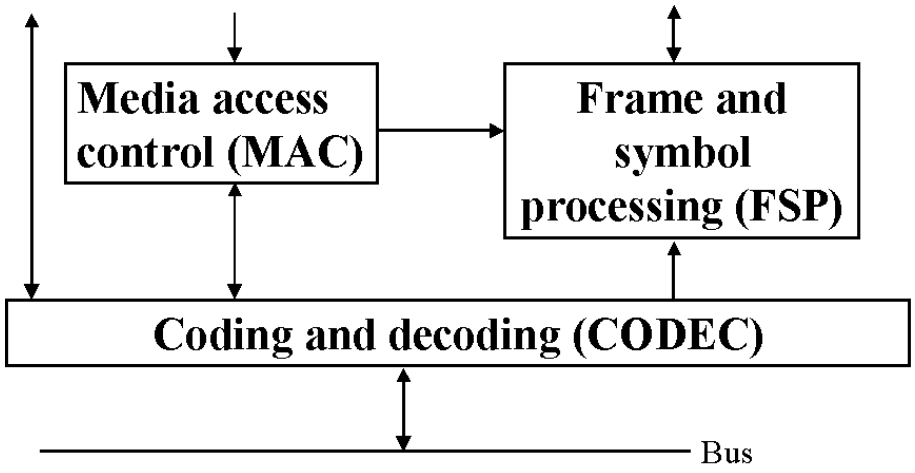


Figure 1: overview of relations between the three presented parts of FlexRay

to send a message after some preparations by the high-level parts of the FlexRay Communication Protocol, such as the clock synchronization. In the other ECUs, the CODEC receives the message and passes it on to the FSP for further computation.
In general, there are two kinds of communication elements: *symbols* (see 3.1.2) and *frames* (see 3.1.1).
Symbols serve as means of controlling the protocol's operation, whereas frames are mainly used to transport data between the ECUs.

## 2 Media access control

The MAC works as an interface from the *controller host interface* (Host) to the CODEC. If the Host wants to send a frame, the MAC imports the data from the Host and assembles the header from the information which is provided by the host.
The MAC is also tells the *encoding* (ENC) subprocess of the CODEC when to sent a symbol.

The MAC generally controls the timing of all accesses to the bus. It guarantees compliance with the schedule. The usage of a communication cycle is organized in four segments:

- The *static segment*, where the same number of frames are sent in every cycle according to a preconfigured schedule.
- The *dynamic segment*, which can be used for communication on demand. We will not consider this feature in our FlexRay variant.

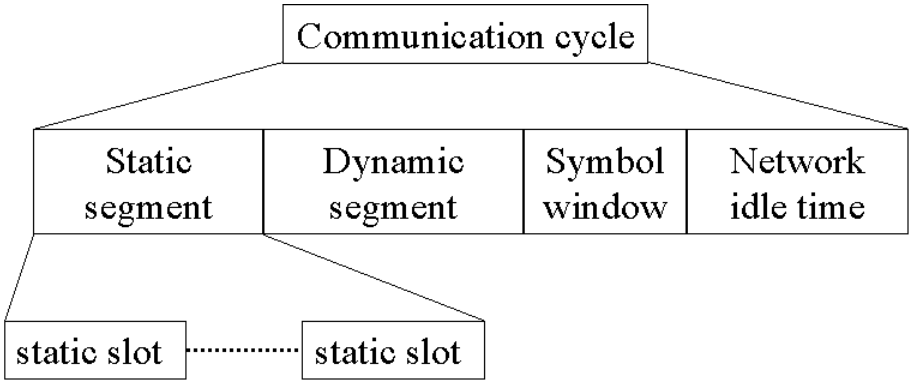


Figure 2: Time usage

- The *symbol window*, which is used to send symbols.
- The *network idle time* (NIT), where there is no communication.

The static segment is divided in a fixed number of *static slots*, each of which can be used to send one frame. Figure 2 gives a graphic description of the time organization in a communication cycle.
The time in static slots is not fully used for communication. There are safety margins on the boundaries of the slots. So the slot consists of a short idle time followed by the time used for the transfer of the frame. After the frame transfer, there is a idle time which is long enough to be recognized as an idle time by the other ECUs. In the idle time, the ECU does not sent anything. As the bus is high-idle, the receiving ECUs will receive bits with the logical value 1.

# 3 Coding and decoding

The CODEC is used to transfer the messages between two ECUs. The CODEC consists of five subprocesses:

- the *encoding* (ENC),
- the *bitstrobing* (BITSTRB),
- the *wakeup symbol decoding* (WUSDEC),
- the *channel idle detection* (IDET) and
- the *decoding* (DEC).

A formal approach to the ENC, BITSTRB and DEC will be given in the Serial Interface model (see 5). On the sending ECU, the ENC is active when

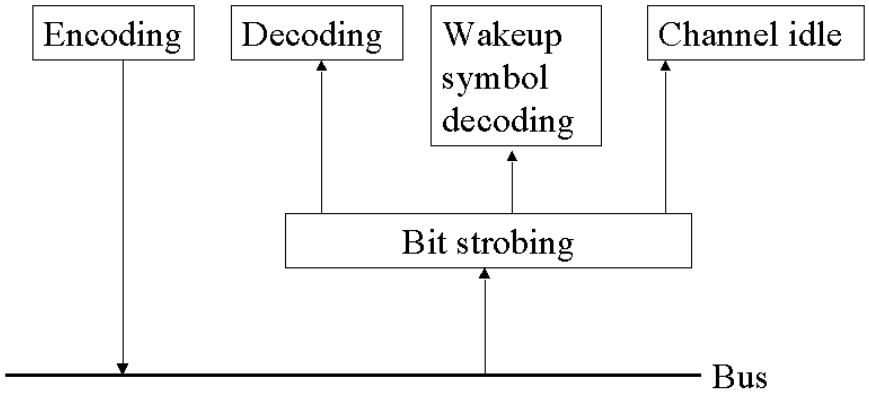


Figure 3: CODEC subprocesses

communication is going on, while the other processes work on the receiving ECU.

## 3.1 Encoding

The ENC transmits frames and symbols via the bus. In order to do so, it is connected to the bus via an open collector driver. When the MAC orders the ENC to send a message, the ENC puts a sequence of bits on the bus. Every message is coded according to a certain scheme. In order to make the bit strobing algorithm work, each bit is sent for eight consecutive clock-cycles before the next bit is sent.

### 3.1.1 Frames

To send a frame in a static slot, the ENC starts with sending a *Transmission Start Sequence* (TSS). The TSS is a low signal of predconfigurable length consisting of five to fifteen bits according to the FlexRay standard. According to the FlexRay variant we use, the TSS consits of just one bit. The TSS is followed by a *Frame Start Sequence* (FSS), consisting of one high bit. Then the actual frame is transmitted, every byte of the frame preceded by a *Byte Start Sequence* (BSS), which consists of one high bit followed by one low bit.
When all bytes have been send, the sending of a *Frame End Sequence* (FES), which consists of a low bit followed by a high bit, concludes the transmission. Figure 4 shows the format af a frame.

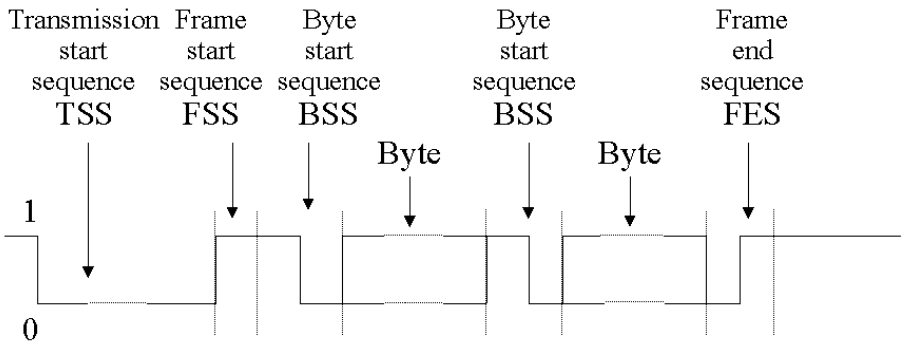


Figure 4: frame coding

### 3.1.2 Symbols

There are two different symbol patterns, the *Collision Avoidance Symbol / Media Access Test Symbol* (CAS/MTS) and the *Wakeup Symbol* (WUS).
A CAS or an MTS is coded as a TSS followed by a thirty-bit sequence of low bits.
A WUS is coded as a low sequence of a preconfigurable length of 15 to 60 bit, followed by an idle period of three times that length. WUSs are never send alone. They are sent in a *Wakeup Pattern* (WUP). This also has a preconfigurable length and consists of 2 to 63 WUSs.

## 3.2 Bit strobing

The BITSTRB process has to take care of the following tasks:

- Take a sample from the bus every cycle of the clock.
- Do a *majority voting* (see 3.2.1) to retrieve the so called *voted value*.
- Pass one voted value per sent bit on to the DEC, IDET and WUSDEC. This means one for every 8 samples.
- Perform low level synchronization.

The sampling is done via two consecutive registers. The Serial Interface model will go into more detail about this in 5.2. We assume that the samples always have a logical value of either high or low.

### 3.2.1 Majority voting

Due to interference from alpha particles, electromagnetic fields and the likes, it is possible that the value of some of the bits sent via the bus may be

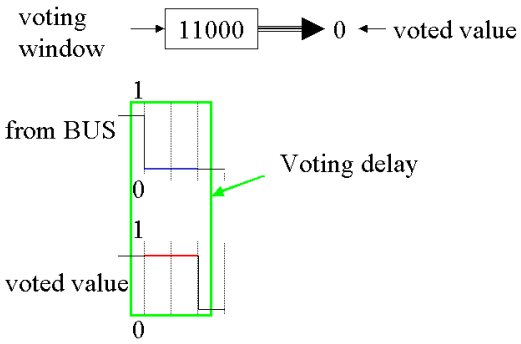


Figure 5: Voting delay

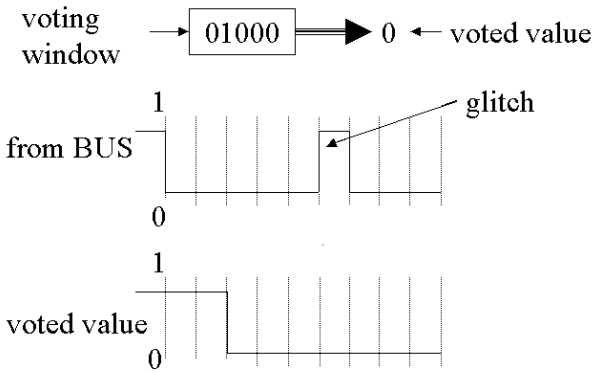


Figure 6: Voting compensates for glitches

flipped. In order to adjust for these errors, the voted values are obtained by taking the majority of the last five received values.
If the *voting window*, i.e. the last five values, is filled with high values, at least three low samples have to be received in order to produce a switch from high to low in the voted value. This means that changes in the value of the samples are only recognized by the higher levels of the CODEC after two more samples with the new value have been received. This causes a *voting delay* of 2 clock cycles, as illustrated in figure 5.

The advantage is that this mechanism can compensate for flipped bits, so called *glitches*. If one sample was flipped to high out of a sequence of five samples sent as low, this is not reflected in the voted value. This effect is illustrated in figure 6. Unless, of course, the values of the samples change while there is still a glitch in the voting window. In this case, the voting delay could be shortened or lengthened by one clock cycle.

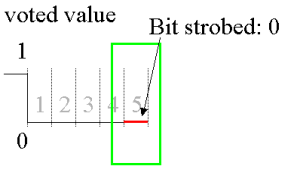


Figure 7: Bit strobing

### 3.2.2 strobing

In every cycle, there is a new voted value obtained. As bits are sent for 8 consecutive cycles, we have received 8 times as many bits as needed.
To reduce this number again, only every 5th bit out of 8 consecutive bits is *strobed*, i.e. forwarded to the higher level processes of the CODEC. This algorithm is used to compensate for clock drift not corrected by the clock synchronization. As the fifth value out of eight identical ones is strobed, the drift could only affect the strobed value if the missalignement between the two clocks was bigger than 3 clock cycles. The probability for wrong voted values is also lower if the strobed value is not close to the falling or rising edges of the value on the bus. This later effect is partly due to the voting process (see 3.2.1).
For the strobing there is a counter, the *strobecounter*, which counts from 1 to 8. Every time when this counter reaches 5, the current voted value is strobed as illustrated in figure 7.
When the strobecounter reaches 8, it is reset to 1 in the next clock cycle. As different ECUs probably have oscillators with slightly different frequencies, the strobecounter does not always have a value of 5 when the voted value corresponding to the 5th sample of a bit that has been sent is received. To keep the mechanism up and running, some means of synchronization is needed. This will be provided by the *low level synchronization*.

**Low level synchronization** At the beginning of the reception of a transmission, the strobecounter is reset to 1. This beginning is recognized by falling edge on the bus, represented by a change in the voted value. If the bus was idle and then a low value is voted, a transmission is starting. As symbols consist of very long periods of the same value, more advanced low level synchronization is only needed for the reception of frames.
During the reception of a frame, the strobecounter is periodically reset to 1 at *sync edges*. These sync edges are falling edges of the bus during a BSS. As a BSS consists of a high bit followed by a low bit, we can thus synchronize the strobecounter at the beginning of every byte of the frame.
Figure 8 shows the first half of a BSS (byte start sequence) beginning when the strobecounter, who's value is indicated by the gray numbers, has a value

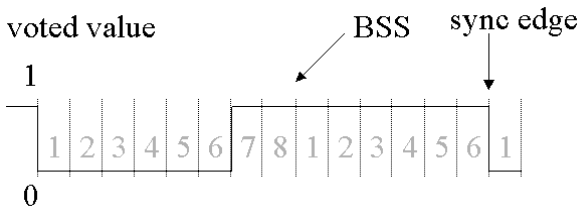


Figure 8: Synchronization during BSS

of 7, which is a missalignement. At the sync edge, the counter is reset to 1 before having reached the 8. This realigns the counter to the format of the incomming frame.

In the Serial Interface (see 5), there will be an argument showing that this suffices to ensure synchronized communictaion.

### 3.3 Wakeup symbol decoding

The strobed bits have to be interpreted as a message. The WUSDEC is waiting for WUPs (see 3.1.2). Whenever a pattern is received that resembles a WUP, it signals the reception of a WUP to protocol operation control (POC). These WUPs are needed during the startup phase of the FlexRay cluster.

If low is received for 15 to 60 cycles, followed by a high (idle) period of three times this length, and this is followed by a low sequence like the one received in the first place, a WUP is recognized. This pattern will be produced by one WUS followed by the low period of a second WUS, so a sequence of 2 consecutive WUSs is enough.

### 3.4 Idle detection

Not every high bit indicates that the bus is idle. To distinguish between high bits that are used in communication elements and an idle bus, the length of a sequence of consecutive high bits is decisive. If the bus is assumed to be active, whenever 11 consecutive high bits are received, IDET will signal that the bus has fallen idle.

This event is called a channel *idle recognition point* (CHIRP). It is safe to assume an idle bus at CHIRP because symbols consist only of low sequences and, in case of the WUS, of idle sequences. Frames, on the other hand, are divided into bytes (see 3.1.1). As a byte and a BSS sequence are together 10 bits long, and the second bit of a BSS is low, at most 9 consecutive high bits can be strobed while receiving a frame. As the first bit of a FES (frame end sequence) is low, the last byte and the following FES can have at most

8 consecutive high bits.

### 3.5 Decoding

In order to reconstruct the message that was transmitted as a frame, the sequences that were added to structure the data (as shown in 3.1.1) have to be removed again. The DEC (decoding) process handles the reception of frames, CASs and MTSs. It works as an interface from BITSTRB to the FSP and POC.
In addition, DEC performs some error checks trying to verify the integrity of the received communication elements.
In case of a frame, DEC checks the format of the frame. If the frame looks as expected, it removes the TSS, FSS, BSS and FES sequences. At the end, DEC checks the *cyclic redundancy code* (CRC) attached to the message, a kind of checksum. DEC produces a CRC using the received frame and compares it with the received CRC. The CRC check allows to notice if the data has changed since it left the sending node.
In case of a CAS or MTS, the DEC just checks if the length of the signal does not deviate from the length defined in 3.1.2.

## 4 Frame and symbol processing

The FSP works as an interface from DEC to the Host. It performs further checks on the integrity of communication elements. As certain types of communication elements can only appear in certain segments of the communication cycle, the FSP checks if the communication element was received in the right segment. The lengths of the communication elements are also checked.
A frame has a lot of information in its header, and this information is verified by the FSP. For instance, the length of the frame should be as stated in the header.
Any errors are reported to the Host and the POC.
The FSP passes on only the payload of frames, so the Host gets the binary message that was given to the sending CODEC by its Host.

## 5 Serial Interface - a formal model for coding and decoding

The Serial Interface gives a formal model for the encoding, the transfer and the reception, i.e. the voting and strobing, of frames. The model

demonstrates how communiction is going on between a single sender and a single receiver, which are connected via a bus. At the end we will prove a theorem stating that the received message equals the sent one.
There are some differences between the model and the protocol specified by the FlexRay standard. The most important ones are:

- The TSS has a length of one bit in the model.
- The model assumes the absence of glitches, i.e. flipped bits due to external interference.
- The state machines that control the reception of frames slightly differ, e.g. a FSS sequence has to be received in the model, whereas the protocol accepts a missing FSS.
- The mechanisms that support more ECUs and more features than one-directional communiction between two ECUs are not considered in the model.

In the following, I will give several definitions. Lower indices denote the ECU, so $X_s$ denotes the $X$ of the sender. $r$ denotes the receiver. Upper indices denote the clock cycle of the ECU. So $X^t$ denotes $X$ directly before the end of cycle $t$, when all hardware has stabilized.

### 5.1 Clocks

A FlexRay cluster consists of serveral ECUs. Here we just look at the sending ECU and one of the receiving ECUs, as these two suffice to demonstrate the correctness of the message transfer algorithm. The two ECUs are connected via a bus.
The ECU number $u$ is denoted by $ECU_u$. It has an own oscillator with a clock signal $ck_u$ which has a cycle time of $\tau_u$.
We assume that the drift between two clocks is at most 0.15%.
To be able to translate cycles into time, let $e_u(i)$ denote the time of the $i$-th rising edge of $ck_u$. Thus the $i$-th cycle of $ECU_u$ is denoted by $[e_u(i), e_u(i+1)[$.

### 5.2 Bus connection

The sender $s$ is connected to the bus via an open-collector driver. The receiver $r$ samples the signal from the bus through two consecutive registers.

Figure 9 shows the setting: $S_s$ is the signal that is send, $Doe_s(t)$ is the enable signal of the open collector driver, which is enabled during the transmission. Moreover, $B_s(t)$ is the value of the output of the open-collector driver at time $t$, and thus of the bus connected to it.

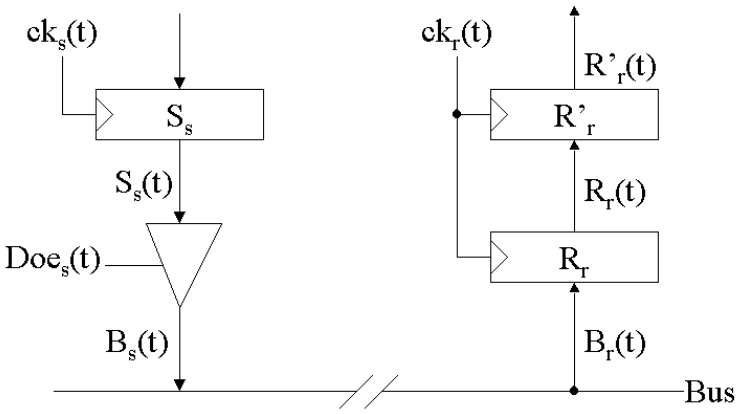


Figure 9: Bus connection

$B_r(t)$ is the value of the bus as seen by the receiver at time $t$. This value is sampled through the two consecutive registers $R_r$ and $R'_r$.
Note that the second register is responsible for an additional delay of 1 on behalf of the reception of values, this delay will often show up as a +1 at the end of formulas.

### 5.2.1 Register semantics

Let the old value of register $R$ be $y$. Assume that the register is clocked at the $i$-th rising edge of $ck$. The input has to be stable for the setup time $t_s$ before the edge, and the hold time $t_h$ after the edge. If it is not stable, the register could become *metastable* and we do not know which value it holds. So the input has to be stable during the sampling interval $[e(i)-t_s, e(i)+t_h]$. In the example given in figure 10, the register's input is connected to the bus which holds a stable value of $x$ during the sampling interval. For the $i$-th rising clock edge the value of the register is defined by

$$R(t) = \begin{cases} y & t \leq e(i) + t_{\mathrm{p-min}} \\ \Omega & e(i) + t_{\mathrm{p-min}} < t < e(i) + t_{\mathrm{p-max}} \\ x & t \geq e(i) + t_{\mathrm{p-max}} \end{cases}$$

with $\Omega$ denoting a physical value (either a 1, a 0, or no logical value).
The register still holds the old value until the minimum propagation delay $t_{\mathrm{p-min}}$ has elapsed, and it holds the new value after the maximum propagation delay $t_{\mathrm{p-max}}$ has elapsed. If the output of $R$ is connected to the input of $R'$, which is clocked in the next cycle, $R'$ holds the new value at time $e(i+1) + t_{\mathrm{p-max}}$.
If the sender puts a new value on the bus at edge $e_s(i)$, one sampling interval

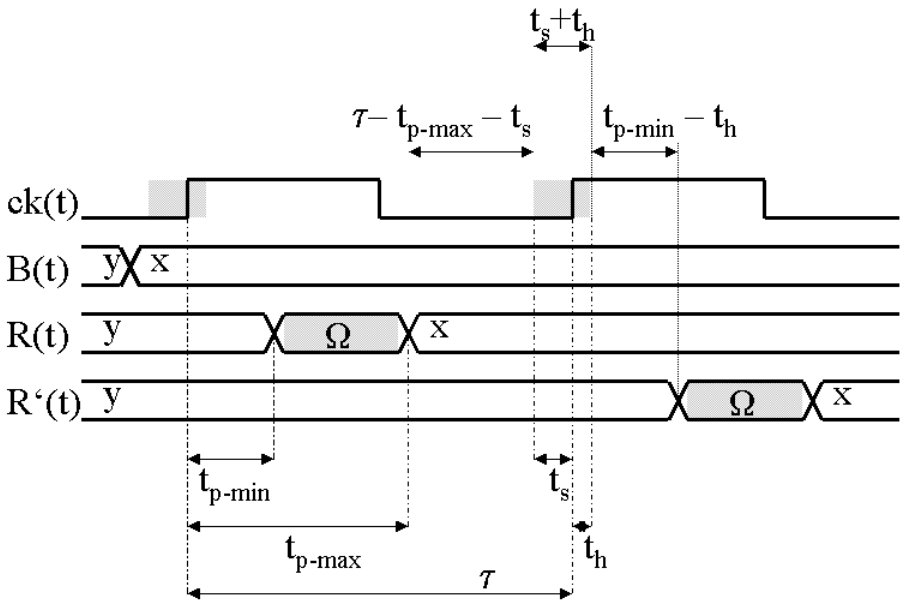


Figure 10: Two consecutive registers attached to the bus

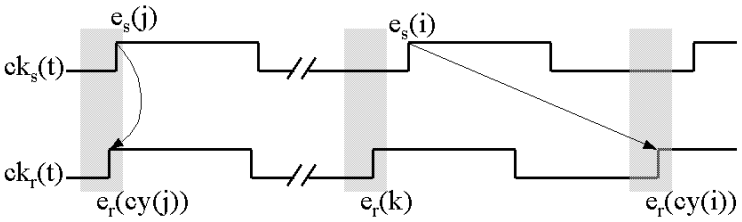


Figure 11: First affected receiver cycle

of the receiver is the first one to be affected by this new value. To translate between the clock cycles of the sender and the clock cycles of the receiver, the function $cy(i)$ is used. It returns the index of the receiver cycle who's sampling interval is as the first one affected by the value that is put on the bus by the sender in cycle $i$. Figure 11 demonstrates an example. Formally we define $cy(i) = max\{k|e_r(k) + t_h < e_s(i)\} + 1$ (It is the index of the edge after the last unaffected sampling interval). This assumes that we have no delay, or a delay smaller than half the time of a cycle, when transferring a signal via the bus.

### 5.2.2 Sampling from stable bus

From 5.2.1 we know that sampling is unproblematic if the bus is stable. The sender sends every bit as eight consecutive samples, so the bus is stable for a certain time. We formulate this as a lemma.

**Lemma 1** If the sender holds the bus stable for eight consecutive cycles, then the receiver samples during at least 7 consecutive cycles the correct value x. Formally:

> If $x = B_s^i = \cdots = B_s^{i+7}$
> then $R_r'^{cy(i)+k+1} = R_r^{cy(i)+k} = x$ with $k \in [\beta : \beta + 6]$ and $\beta \in \{0, 1\}$.

The value of $\beta$ depends on the difference between sender and receiver clock.

**Proof of Lemma 1** The clock drift is bounded by 0.15% and only one node is sending.
The sampling intervals of all receiver edges $cy(i) + k$ are in a region of time where the bus is stable.
If the sampling interval for $k = 0$ is not in this region, then the sampling interval for $k = 7$ is in this region and vice versa, so I can select $\beta \in \{0, 1\}$ such that lemma 1 holds.

### 5.2.3 Sampling changing values

Bits are always sent as 8 consecutive samples. If different bits are sent, each of them is sent for 8 consecutive cycles. But for a short period of time, the bus is unstable, as the value is changed. Lemma 2 formalizes the effect of different bits being consecutively sent on the reception of these bits.

**Lemma 2** If the sender transmits $x$ in cycles $i-8$ to $i-1$ and $\neg x$ in cycles $i$ to $i+7$, then the cycle $i'$ in which $\neg x$ occurs for the first time in $R_r'$ is bounded by an interval of two cycles: $i' \in cy(i) + [0 : 1] + 1$. Formally:

> If $x = B_s^{i-8} = \cdots = B_s^{i-1}$ and $\neg x = B_s^i = \cdots = B_s^{i+7}$
> then for $i' : \neg x = R_r'^{i'} \neq R_r'^{i'-1} : i' \in cy(i) + [0 : 1] + 1$.

**Proof of Lemma 2** As the clock drift is bounded by 0.15%, we know that for two succeeding intervals of 8 consecutively sent bits the value of $\beta$ is the same.

### 5.2.4 Clocks overtaking each other

Drifting clocks could overtake each other. This is formalized in lemma 3.

**Lemma 3** During 600 cycles, a clock can get at most one cycle difference to the idealized clock due to drift. Formally:

$$\forall i : \forall k < 600 : cy(i+k) \in cy(i) + k + [-1:1].$$

**Proof of Lemma 3** Usually $cy(i+1) = cy(i) + 1$, but clock drift can cause $cy(i+1) = cy(i)$ or $cy(i+1) = cy(i+2)$.
As drift is bounded by 0.15%, this can happen at most once in $1/0.0015 > 600$ cycles.

## 5.3 Frame assembly

To transfer messages between ECUs with different clocks requires a message format that allows to draw conclusions about the sender from the received data. The format presented in 3.1.1 is used. It allows for instance to conclude when a message starts or ends. It also offers the opportunity to syncronize the receiving mechanism to the incomming data stream at the beginning of every byte of the data, because it uses recognizable bit patterns. This synchronization was explained in 3.2.2. Thus, communication between non synchronized clocks becomes possible.
In the following, $m$ is the message to be transferred, $f(m)$ is the frame to be sent (and to be reassembled), and $F(m)$ is the bit vector to be transmitted. Moreover, $l$ is the length of $m$ and $l'$ is the length of $f(m)$.
We define $f(m) = TSS \circ FSS \circ BSS \circ m[0] \circ \cdots \circ BSS \circ m[l-1] \circ FES$.
In the following definition, the upper index of 8 means the bit repeated 8 times. As each bit is transmitted for 8 cycles, we define $F(m) = f(m)[0]^8 \circ \cdots \circ f(m)[l'-1]^8$.
Sender cycles are numbered such that $B_s^i = F(m)[i]$.

### 5.3.1 Sampling of message bits

As we know the time when a bit of a message is sent, we can determine at which point in time it will be received.

**Lemma 4** The bit $f(m)[i]$ is correctly sampled at receiver edge $cy(8 \cdot i) + k$. Formally:

$$\forall i \in [0:l'] : \exists \beta \in \{0,1\} : \forall k \in [\beta : \beta + 6] :$$
$$R_r'^{cy(8\cdot i)+k+1} = R_r^{cy(8\cdot i)+k} = f(m)[i]$$

**Proof of Lemma 4** The bus is stable for 8 consecutive cycles, and by lemma 1 we know that in at least 7 consecutive cycles, the correct value will be sampled.

## 5.4 Voting

On the receiving ECU, there is a mechanism that compensates for glitches, as seen in 3.2.1. We do a majority vote to determine the voted values. Let $v^j$ be the majority vote over the last five $R'$ values $R'^j, \ldots, R'^{j-4}$. So if three or more of the values received during the last 5 cycles are 1, the voted value is 1 and otherwise it is 0.
Note that we get a delay of 2 cycles caused by the voting process. This will show up as a +2 in some formulas.

### 5.4.1 Voting of message bits

From lemma 4 we know when a message bit is sampled. The effect of the voting on these samples is reflected by 5.

**Lemma 5** The bit $f(m)[i]$ is correctly voted at receiver edge $cy(8 \cdot i)+k+1$. Formally:

$$\forall f(m)[i] : \exists \beta \in \{0,1\} : \forall k \in [\beta + 2 : \beta + 8] : \\ v^{cy(8 \cdot i)+k+1} = f(m)[i].$$

**Proof of Lemma 5** Lemma 4 entails that in cycles $cy(8 \cdot i) + k + 1$ for $k \in [\beta + 2 : \beta + 8]$ we have received at least three copies of bit $f(m)[i]$.

## 5.5 Frame reception control

The receiving ECU needs a mechanism to handle the reception of the structured data seen in 3.1.1 and 5.3.

The automaton presented in figure 12 controls the reception of frames. Its transition function is denoted by $\Delta(s, i)$ and the state transitions of the automaton occur at the signal $strobe^t$. Here, $s$ is the current state of the automaton, and $i$ is the strobed value. As indicted by the blank arrows entering the $b[x]$ states, some transitions occur regardless of the value of the strobed bit.
This automaton enables us for instance to clearly define which part of the message is used for synchronization. It would also find errors in the format of the frame, but we assume a correct assembly of the frame in this model.

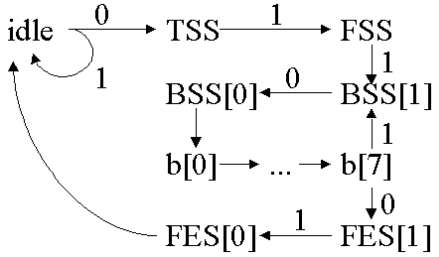


Figure 12: Frame reception automaton

### 5.6 Bit strobing

As seen in 3.2.2, the receiving ECU needs a mechanism to get the right sample out of the eight consecutive samples that are sent for each bit of the frame. This mechanism is called strobing.
The voted value is strobed at so called *strobe points*. These points occur whenever the strobecounter reaches 5. In our model the counter counts from 0 to 7, so we define $strobe^j = (cnt^j = 4)$.
We can access the state of the controlling automaton (see Fig. 12). This state is defined as:

$$state^{t+1} = \begin{cases} \Delta(state^t, v^t) & strobe^t \\ state^t & otherwise \end{cases}$$

The strobecounter is defined by:

$$cnt^{j+1} = \begin{cases} 1 & sync^j \\ (cnt^j + 1) \mod 8 & otherwise \end{cases}$$

Finally, the sync signal is defined by:

$$sync^j = ((state^j = idle) \wedge v^j) \vee ((state^j = BSS[1]) \wedge v^{j-1} \wedge \neg v^j)$$

To be able to talk about strobing, we need some functions that enable us to name certain strobe- and sync-signals. We also some functions meassuring the amount of data that is transfered in certain intervals:

**str**(**h**) denotes the index of the cycle of the $(h + 1)$-th activation of the strobe signal.

**sy**(**h**) denotes the index of the (last) cycle of the $(h + 1)$-th activation of the sync signal.

**nb**(**h**) is the number of bits of $f(m)$ sent in synchronization interval $[sy(h) : sy(h + 1)]$.

**NB(h)** $= \sum_{h'<h} nb(h')$, the number of bits sent in the synchronization intervals before $sy(h)$.

## 5.7 Theorem for low level message transfer

We want to show that the message is correctly reassembled by the receiver. In order to do so, we will show that the automaton and the syncronization work as described in 5.5 and 5.6 and thus the right bits are strobed. These criteria are formulated as an invariant.

### 5.7.1 Invariant

1. The automaton correctly monitors the received bits,
2. the message bits are correctly strobed,
3. the transitions of automaton occur fast enough, i.e. before the next bit can be sampled,
4. the sync signals are activated at expected times and
5. the strobe signals are activated at expected times.

Lemma 6 shows that a formal version of this invariant holds for all cycles during the transmission of a frame.
To prove this, we do an induction over $j$. For a better understanding of the lemma, keep in mind that the actual sync number is denoted by $h$, $NB(h')$ refers to the bits sent in previous sync intervals, $nb(h')$ refers to the number of bits to be sent in this interval and $k$ is the number of the actual bit in the interval.

**Lemma 6** For any receiver cycle $j$, for any $k = NB(h') + k'$ with $str(k) \leq j$ and $k' \in [0 : nb(h') - 1]$ and for any $h$ with $sy(h) \leq j$ it holds:

1. If strobe $k$ is the last strobe before cycle $j$, i.e. $j \in [str(k)+1 : str(k+1)]$, then $state^j$ is given as expected:
   In the first sync interval ($h' = 0$) $state^j$ is equal to $TSS$ for $k' = 0$; $FSS$ for $k' = 1$ or $BSS[1]$ for $k' = 2$.
   In the other sync intervals (of length $nb(h') \in [10 : 11]$), $state^j$ is equal to $BSS[0]$ for $k' = 0$ or $b[k'-1]$ for $k' \in [1 : 8]$.
   All but the last sync interval ($h' < l$) end with $state^j = BSS[1]$ for $k' = 9$.
   For $h' = l$ we have $state^j = FES[10 - k']$ for $k' \in [9 : 10]$.

2. The sampled signals satisfy $v^{str(k)} = f(m)[k]$.
3. $str(k) + 1 < cy(8 \cdot (k + 1)) + [2 : 3] + 1$
4. $sy(h) \in cy(8 \cdot NB(h)) + [2 : 3] + 1$
5. $str(k) = sy(h') + 8 \cdot (k - NB(h'))) + 4$

If we have shown condition 2 and condition 3 for $j+1$, condition 1 holds for $j + 1$.
We will show that condition 4 and condition 5 for $j$ implies condition 2 and condition 3 for $j + 1$ in lemma 7.
Then we will show that condition 1 and condition 3 for $j+1$ imply condition 4 and condition 5 for $j+1$. Thus we will have completed the induction step, as we will have shown that all five conditions hold for $j + 1$ if they hold for $j$.
Assuming that sender cycles NB(h) and corresponding receiver cycles are not to far apart we conclude:

**Lemma 7** If (1) the strobepoint occurs in the expected time bounds and if (2) the syncronization occurs in the expected time bounds
then (i) the message bits are correctly strobed and (ii) the transitions of the automaton occur fast enough, i.e. before the next bit can be sampled. Formally:

> If h' maximal such that (1) $str(k) = sy(h')+8\cdot(k-NB(h')))+4$ and if (2) $sy(h') \in cy(8 \cdot NB(h')) + [2 : 3] + 1$ then (i) $v^{str(k)} = f(m)[k]$ and(ii) $str(k)+1 < cy(8\cdot(k+1))+[2 : 3] + 1$

**Proof of Lemma 7** Part(i) using lemma 3 and lemma 5:

$$\begin{aligned} str(k) &= sy(h') + 8 \cdot (k - NB(h')) + 4 \\ &\in cy(8 \cdot NB(h')) + 8 \cdot (k - NB(h')) + [6 : 7] + 1 \\ &\in cy(8 \cdot (NB(h') + k - NB(h'))) + [5 : 8] + 1 \end{aligned}$$

so it holds: $v^{str(k)} = f(m)[k]$
Part(ii) using lemma 3:

$$\begin{aligned} str(k) + 1 &\in cy(8 \cdot NB(h')) + 8 \cdot (k - NB(h')) + [6 : 7] + 1 + 1 \\ &= cy(8 \cdot NB(h')) + 8 \cdot (k - NB(h') + 1) + [0 : 1] \\ &\in cy(8 \cdot (NB(h') + k - NB(h') + 1)) + [-1 : 2] \\ &< cy(8 \cdot (k + 1)) + [2 : 3] + 1 \end{aligned}$$

.

**Proof of Lemma 6** We want to show that the sync signals are activated at the expected times. In order to do so, we have to show that (i) the falling edge that triggers $sy(h)$ affects the receiver in the right cycle $j$ and that (ii) the automaton is in the state $BSS[1]$ during cycle $j$.
Lemma 2 combined with lemmas 4 and 5 shows that the falling edge which

triggers $sy(h)$ is seen in $v^j$ for $j \in cy(8 \cdot NB(h)) + [2:3] + 1$.
Condition 1 of the invariant implies $state^j = BSS[1]$ for cycles $j \in [str(k)+1 : str(k+1)]$ if $k$ is maximal and $(h' = 0 \wedge k' = 2) \vee (h' \in [1 : l-1] \wedge k' = 8)$. Outside these time intervals the sync signal cannot become active.
Condition 3 of the invariant implies $str(k)+1 < cy(8 \cdot (NB(h')+k'+1))+[2:3]+1$. Thus the automaton is in state $BSS[1]$ one cycle before the first zero of the $BSS[0]$ bit can be possibly sampled. So it is shown that condition 4 of the invariant holds.
We still have to show that strobe signals are activated at expected times.
For the case $k' = 0$:

$$\begin{aligned} sy(h') &\in sy(h'-1) + 8 \cdot nb(h'-1) + [-1:1] \\ &= sy(h'-1) + 8 \cdot (nb(h'-1)-1) + 8 + [-1:1] \end{aligned}$$

From the induction hypothesis we know that $str(k-1) = sy(h'-1)+8 \cdot (k-1-NB(h'-1))+4 = sy(h'-1)+8 \cdot (nb(h'-1)-1)+4$. Thus $str(k-1)$ is before $sy(h')$ and there is no additional strobe between them. For the case $k' > 0$ condition 5 follows from the induction hypotheses. So part condition 5 holds, too.

### 5.7.2 Frame reassembly

After reset we start with an empty reconstruction frame $f'^0$. Each strobed bit is attached to this frame.

$$f'^{t+1} = \begin{cases} f'^t \circ v^t & strobe^t \\ f'^t & otherwise \end{cases}$$

### 5.7.3 The theorem

Finally we are able to state that the transfer of the message will be succesfull.
Lemma 6 shows that the following theorem holds:

> Let clock drift $\sigma$be bounded by 0.15% and let $L = 8 \cdot l'$ be the length of $F(m)$.
> It holds:
> $f'^{\lceil (1+\sigma) \cdot L + 8 \rceil} = f(m)$

So the message will be received after enough time has elapsed.