

# Model Checking the FlexRay Physical Layer Protocol

Michael Gerke

Reactive Systems Group  
Saarland University  
Germany

25.09.2013

If you hit the brakes ...



**BMW 7er (F01)**

... and they don't work:



# Brakes should work!





*FlexRay*



# Drive-by-Wire



*FlexRay*

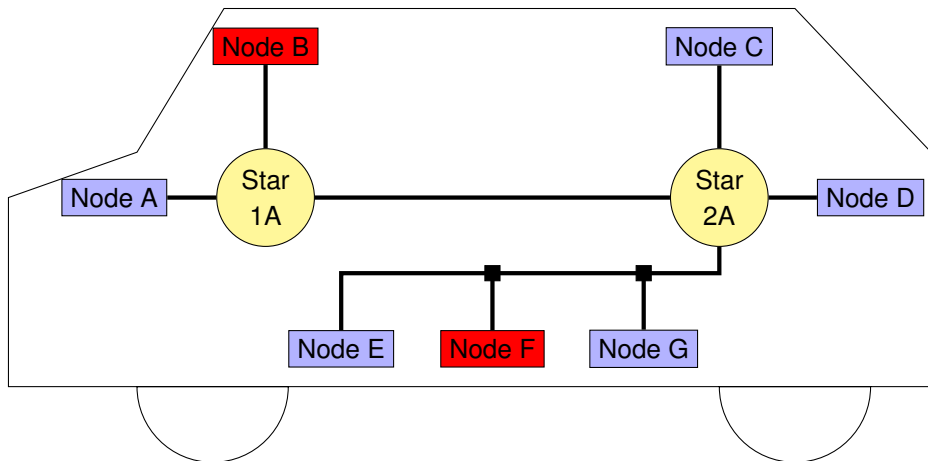




## FlexRay

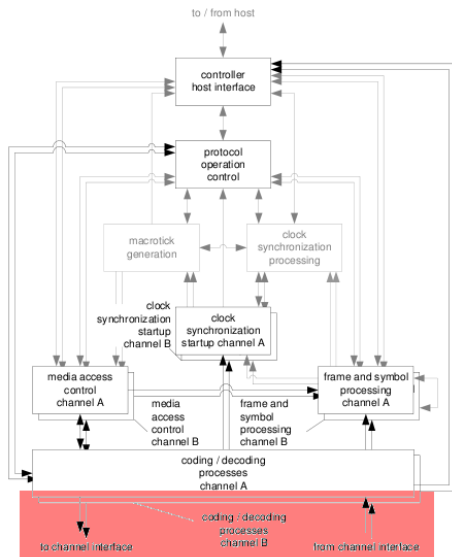
- communication protocol for distributed components in cars
- used in BMW X 5 and BMW's 7 series for X-by-wire
- developed by: BMW, Bosch, Daimler, Freescale, General Motors, NXP Semiconductors, Volkswagen, et al.

# FlexRay Network

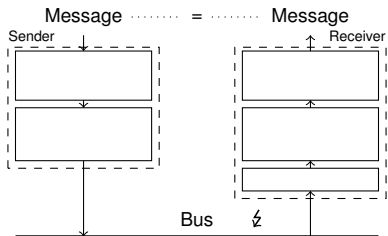




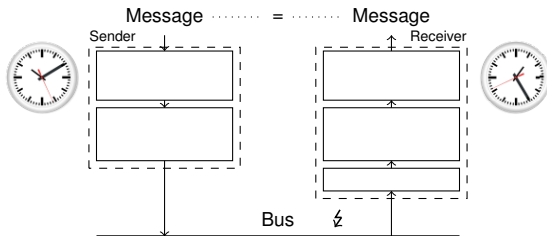
# Focus: Physical Layer



# Case Study: FlexRay Physical Layer Protocol



# Case Study: FlexRay Physical Layer Protocol



The only (but important) source of time in the model: the two oscillators

# Why is Non-synchronized Hardware a Challenge?

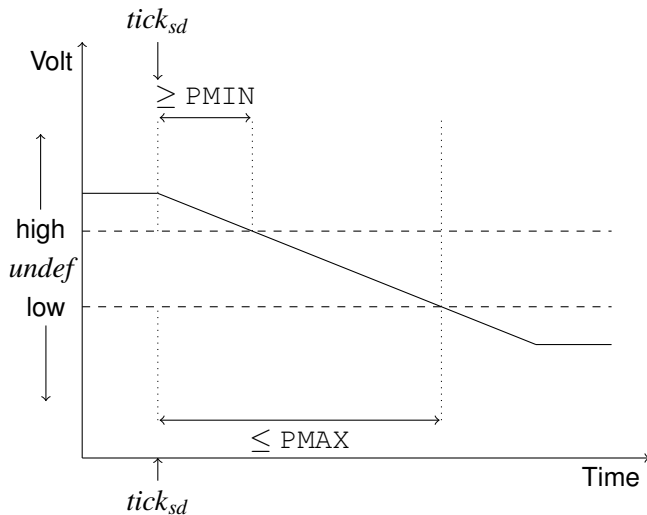


Figure: Transition between voltage levels takes time.

# Register Semantics

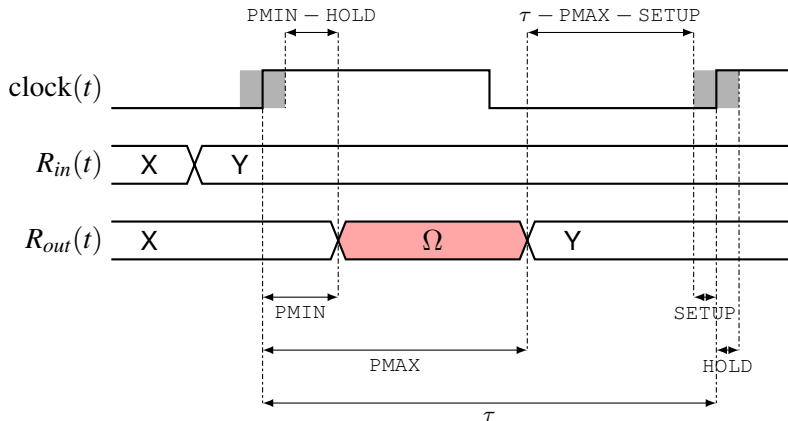
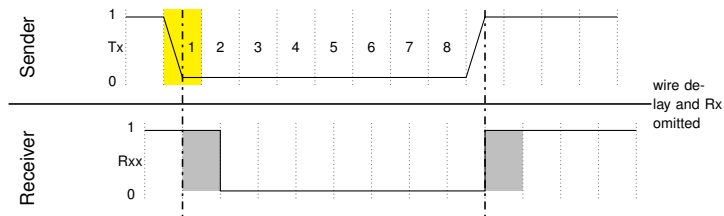


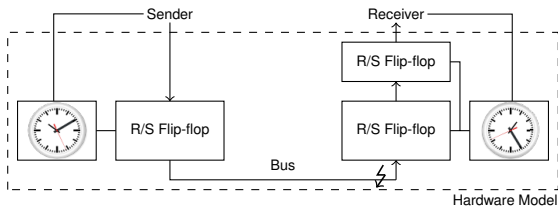
Figure: Value of enabled register  $R$  over time  $t$

# Jitter Effects



- clock drift → asynchronous communication
- delay variance → signal changes take varying amounts of time
- unstable value during setup / hold → error

# Error Sources in Communication Scenario



Sources of error:

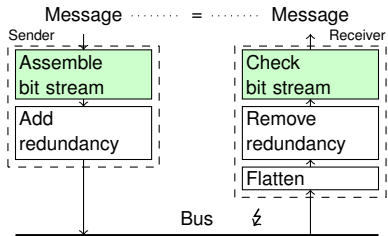
*jitter:*

- clock drift
- variance in delay
- unstable value in sampling interval (setup/hold)

*glitches:*

- bits flipped on the bus

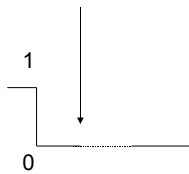
# Protocol Architecture





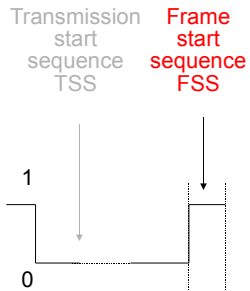
## Encoding (ENC) Frames

Transmission  
start  
sequence  
TSS



Frame coding in static segment

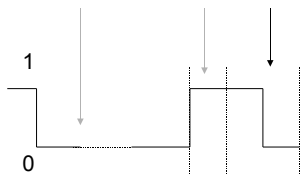
## Encoding (ENC) Frames



Frame coding in static segment

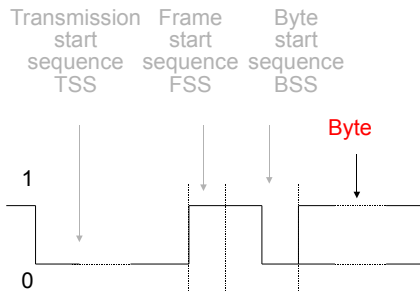
## Encoding (ENC) Frames

Transmission start sequence TSS  
Frame start sequence FSS  
Byte start sequence BSS



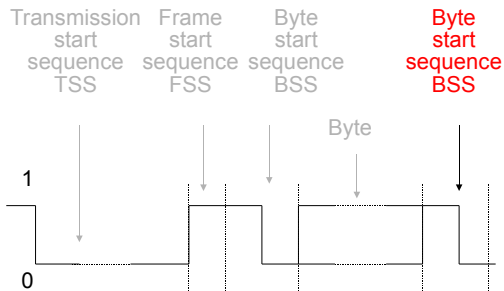
Frame coding in static segment

## Encoding (ENC) Frames



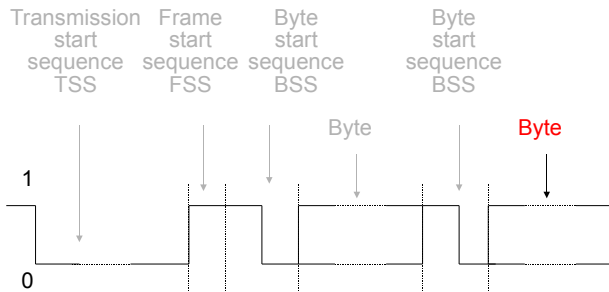
Frame coding in static segment

## Encoding (ENC) Frames



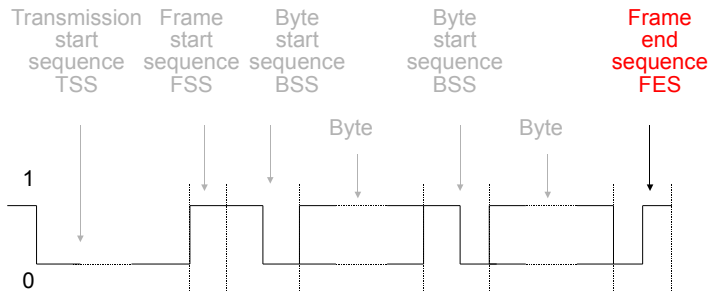
Frame coding in static segment

## Encoding (ENC) Frames



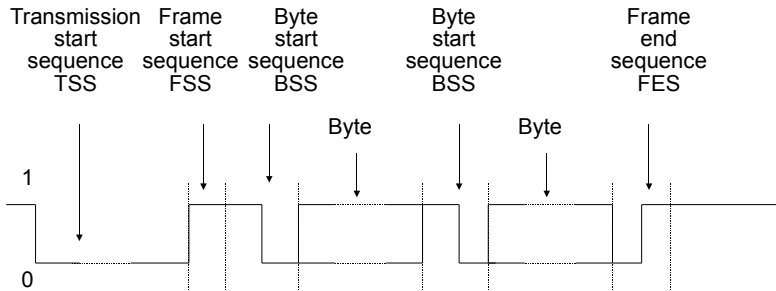
Frame coding in static segment

## Encoding (ENC) Frames



Frame coding in static segment

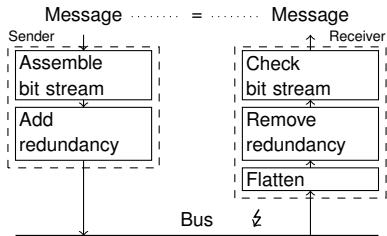
## Encoding (ENC) Frames



Frame coding in static segment

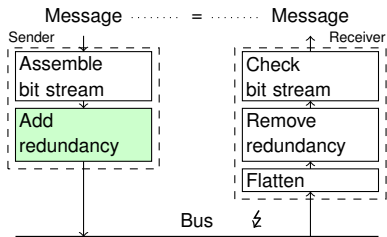


# Protocol Architecture



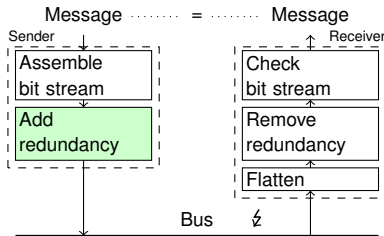
- Add redundancy: send each bit for 8 cycles (bit cell)
- Flatten: take majority of last 5 samples (voting window size 5)
- Remove redundancy: pick 5<sup>th</sup> voted value from each bit cell

# Protocol Architecture



- **Add redundancy:** send each bit for 8 cycles (bit cell)
- Flatten: take majority of last 5 samples (voting window size 5)
- Remove redundancy: pick 5<sup>th</sup> voted value from each bit cell

# Protocol Architecture



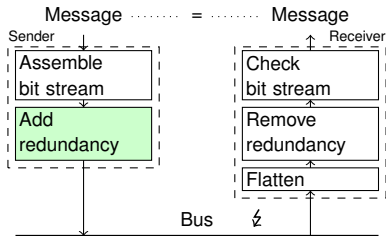
- **Add redundancy**: send each bit for 8 cycles (bit cell)

Stream: 1 = Bus: 

1	1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---	---

- Flatten: take majority of last 5 samples (voting window size 5)
- Remove redundancy: pick 5<sup>th</sup> voted value from each bit cell

# Protocol Architecture



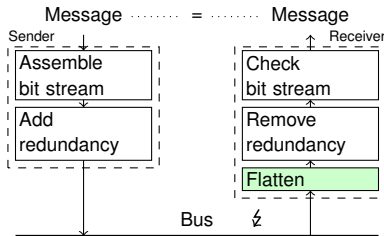
- **Add redundancy**: send each bit for 8 cycles (bit cell)

Stream: 10 = Bus: 

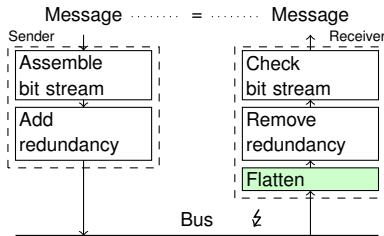
1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

- Flatten: take majority of last 5 samples (voting window size 5)
- Remove redundancy: pick 5<sup>th</sup> voted value from each bit cell

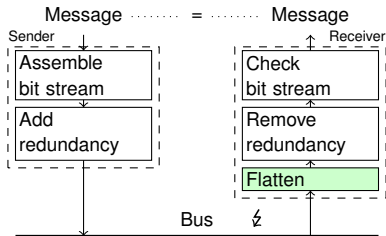
# Protocol Architecture



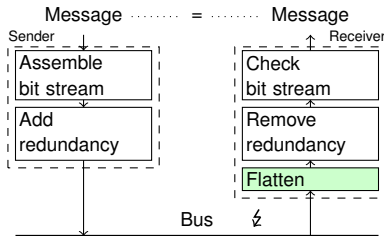
- Add redundancy: send each bit for 8 cycles (bit cell)
- Flatten: take majority of last 5 samples (voting window size 5)
- Remove redundancy: pick 5<sup>th</sup> voted value from each bit cell



- Add redundancy: send each bit for 8 cycles (bit cell)
- **Flatten**: take majority of last 5 samples (voting window size 5)  
E.g.: ... 1 1 1 1 1 = 1
- Remove redundancy: pick 5<sup>th</sup> voted value from each bit cell



- Add redundancy: send each bit for 8 cycles (bit cell)
- **Flatten**: take majority of last 5 samples (voting window size 5)  
E.g.: ... 1 1 1 1 1 0 = 1
- Remove redundancy: pick 5<sup>th</sup> voted value from each bit cell

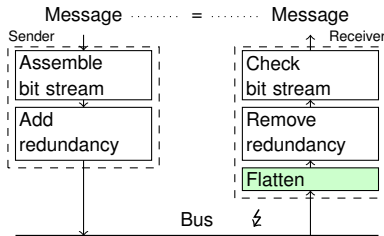


- Add redundancy: send each bit for 8 cycles (bit cell)
- **Flatten**: take majority of last 5 samples (voting window size 5)  
E.g.: ... 

1	1	1	1	1	0	0
---	---	---	---	---	---	---

 = 1
- Remove redundancy: pick 5<sup>th</sup> voted value from each bit cell



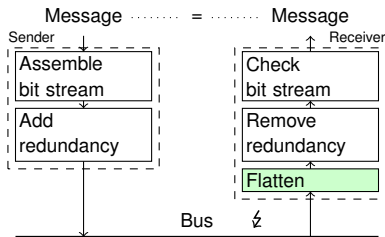


- Add redundancy: send each bit for 8 cycles (bit cell)
- **Flatten**: take majority of last 5 samples (voting window size 5)  
E.g.: ... 

1	1	1	1	1	0	0	0
---	---	---	---	---	---	---	---

 = 0
- Remove redundancy: pick 5<sup>th</sup> voted value from each bit cell

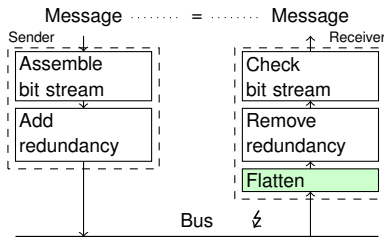
# Protocol Architecture



- Add redundancy: send each bit for 8 cycles (bit cell)
- **Flatten**: take majority of last 5 samples (voting window size 5)  
E.g.: ... 

1	1	1	1	1	0	0	0	0
---	---	---	---	---	---	---	---	---

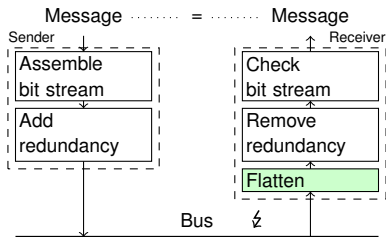
 = 0
- Remove redundancy: pick 5<sup>th</sup> voted value from each bit cell



- Add redundancy: send each bit for 8 cycles (bit cell)
- **Flatten**: take majority of last 5 samples (voting window size 5)  
E.g.: ... 

1	1	1	1	1	0	0	0	0	1
---	---	---	---	---	---	---	---	---	---

 = 0
- Remove redundancy: pick 5<sup>th</sup> voted value from each bit cell

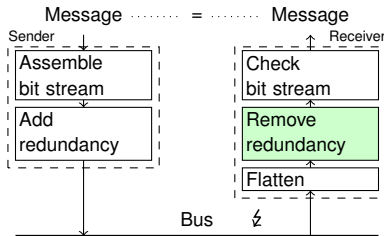


- Add redundancy: send each bit for 8 cycles (bit cell)
- **Flatten**: take majority of last 5 samples (voting window size 5)  
E.g.: ... 

1	1	1	1	1	0	0	0	0	1	0
---	---	---	---	---	---	---	---	---	---	---

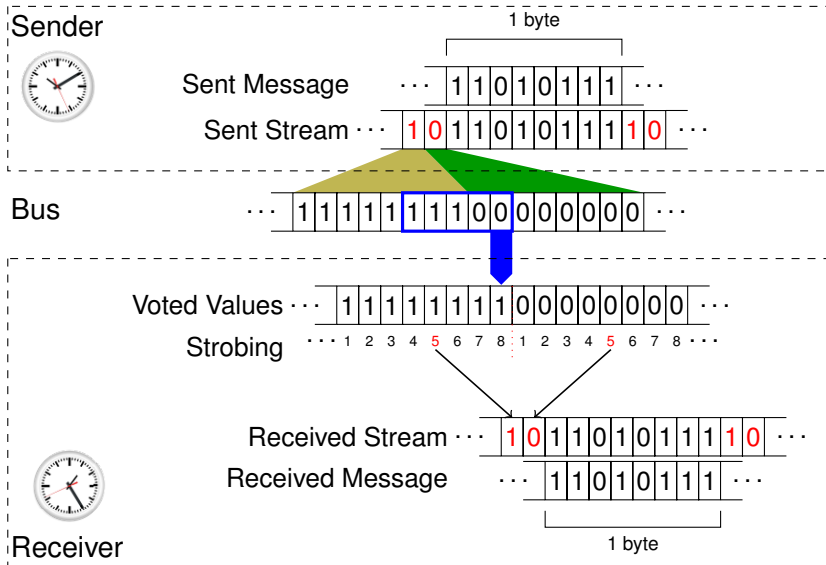
 = 0
- Remove redundancy: pick 5<sup>th</sup> voted value from each bit cell

# Protocol Architecture

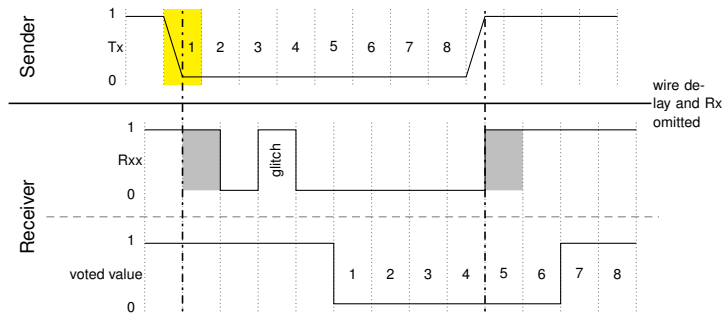


- Add redundancy: send each bit for 8 cycles (bit cell)
- Flatten: take majority of last 5 samples (voting window size 5)
- Remove redundancy: pick 5<sup>th</sup> voted value from each bit cell

# Overview: FlexRay Physical Layer Protocol



# Why? Jitter and Glitch Correction!



# FlexRay Error Resilience



⚡ FlexRay



FlexRay Protocol Specification

3.2.6.7 Phase 3B error assembly

3.2.6.8 Phase 3B error assembly

3.2.6.9 Phase 3B error assembly

3.2.7 Signal integrity

**FlexRay Communications System**  
**Protocol Specification**  
**Version 2.1**  
 Revision A

3.2.7 Signal integrity

In general, there are several conditions (e.g. clock oscillator differences, electrical characteristics of the transmission media or the transceivers, EMC etc.) that can cause variations of signal timing or introduce anomalies/glitches into the communication bit stream. The decoding function attempts to enable tolerance of the physical layer against presence of one glitch in a bit cell when the length of the glitch is less than or equal to one channel sample clock period. If:

1. These are specific cases where a single glitch cannot be tolerated and others where two glitches can be tolerated.

3.2.7.1 Signal integrity

3.2.7.2 Signal integrity

3.2.7.3 Signal integrity

3.2.7.4 Signal integrity

3.2.7.5 Signal integrity

3.2.7.6 Signal integrity

3.2.7.7 Signal integrity

3.2.7.8 Signal integrity

3.2.7.9 Signal integrity

3.2.7.10 Signal integrity

3.2.7.11 Signal integrity

3.2.7.12 Signal integrity

3.2.7.13 Signal integrity

3.2.7.14 Signal integrity

3.2.7.15 Signal integrity

3.2.7.16 Signal integrity

3.2.7.17 Signal integrity

3.2.7.18 Signal integrity

3.2.7.19 Signal integrity

3.2.7.20 Signal integrity

3.2.7.21 Signal integrity

3.2.7.22 Signal integrity

3.2.7.23 Signal integrity

3.2.7.24 Signal integrity

3.2.7.25 Signal integrity

3.2.7.26 Signal integrity

3.2.7.27 Signal integrity

3.2.7.28 Signal integrity

3.2.7.29 Signal integrity

3.2.7.30 Signal integrity

3.2.7.31 Signal integrity

3.2.7.32 Signal integrity

3.2.7.33 Signal integrity

3.2.7.34 Signal integrity

3.2.7.35 Signal integrity

3.2.7.36 Signal integrity

3.2.7.37 Signal integrity

3.2.7.38 Signal integrity

3.2.7.39 Signal integrity

3.2.7.40 Signal integrity

3.2.7.41 Signal integrity

3.2.7.42 Signal integrity

3.2.7.43 Signal integrity

3.2.7.44 Signal integrity

3.2.7.45 Signal integrity

3.2.7.46 Signal integrity

3.2.7.47 Signal integrity

3.2.7.48 Signal integrity

3.2.7.49 Signal integrity

3.2.7.50 Signal integrity

3.2.7.51 Signal integrity

3.2.7.52 Signal integrity

3.2.7.53 Signal integrity

3.2.7.54 Signal integrity

3.2.7.55 Signal integrity

3.2.7.56 Signal integrity

3.2.7.57 Signal integrity

3.2.7.58 Signal integrity

3.2.7.59 Signal integrity

3.2.7.60 Signal integrity

3.2.7.61 Signal integrity

3.2.7.62 Signal integrity

3.2.7.63 Signal integrity

3.2.7.64 Signal integrity

3.2.7.65 Signal integrity

3.2.7.66 Signal integrity

3.2.7.67 Signal integrity

3.2.7.68 Signal integrity

3.2.7.69 Signal integrity

3.2.7.70 Signal integrity

3.2.7.71 Signal integrity

3.2.7.72 Signal integrity

3.2.7.73 Signal integrity

3.2.7.74 Signal integrity

3.2.7.75 Signal integrity

3.2.7.76 Signal integrity

3.2.7.77 Signal integrity

3.2.7.78 Signal integrity

3.2.7.79 Signal integrity

3.2.7.80 Signal integrity

3.2.7.81 Signal integrity

3.2.7.82 Signal integrity

3.2.7.83 Signal integrity

3.2.7.84 Signal integrity

3.2.7.85 Signal integrity

3.2.7.86 Signal integrity

3.2.7.87 Signal integrity

3.2.7.88 Signal integrity

3.2.7.89 Signal integrity

3.2.7.90 Signal integrity

3.2.7.91 Signal integrity

3.2.7.92 Signal integrity

3.2.7.93 Signal integrity

3.2.7.94 Signal integrity

3.2.7.95 Signal integrity

3.2.7.96 Signal integrity

3.2.7.97 Signal integrity

3.2.7.98 Signal integrity

3.2.7.99 Signal integrity

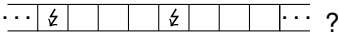
3.2.7.100 Signal integrity



# FlexRay Error Resilience



↘ FlexRay



FlexRay Protocol Specification

3.2.7 Signal integrity

In general, there are several conditions (e.g. clock oscillator differences, electrical characteristics of the transmission media or the transceivers, EMC, etc.) that can cause variations of signal timing or introduce anomalies/glitches into the communication bit stream. The decoding function attempts to enable tolerance of the physical layer against presence of one glitch in a bit cell when the length of the glitch is less than or equal to one channel sample clock period.<sup>18</sup>

<sup>18</sup> There are specific cases where a single glitch cannot be tolerated and others where two glitches can be tolerated.

**FlexRay Communications System Protocol Specification Version 2.1**

Revision A

Page 41 of 66

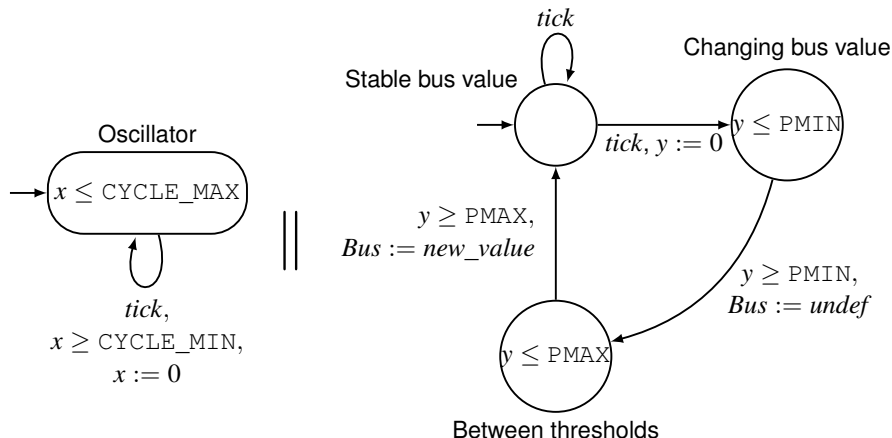
We compose our model from several components.

We need to model:

- the hardware (with jitter)
- the FlexRay physical layer protocol
- the bit flips (glitches)

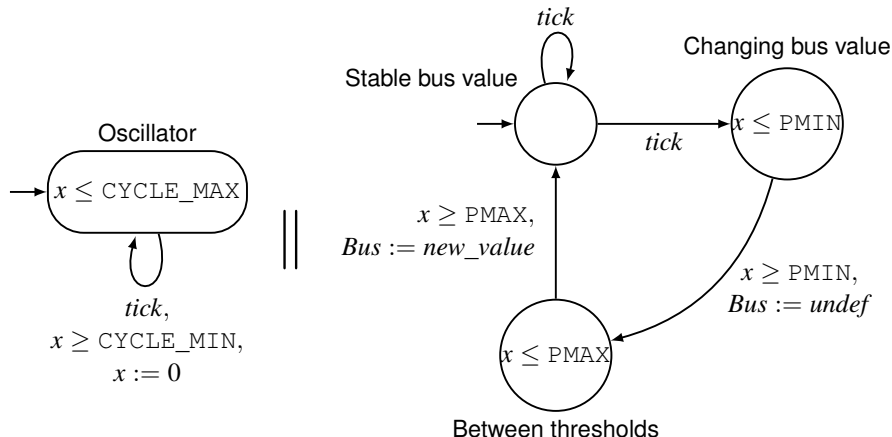
W.l.o.g., it suffices to model one sender connected to one receiver.

# Modeling Hardware: Parameterized



**Figure:** Model of the sender's oscillator and the bus, synchronizing on action *tick*. One clock for the oscillator, one clock for the time to reach the first and the second threshold. Parameterization allows for easy adjustment to different hardware.

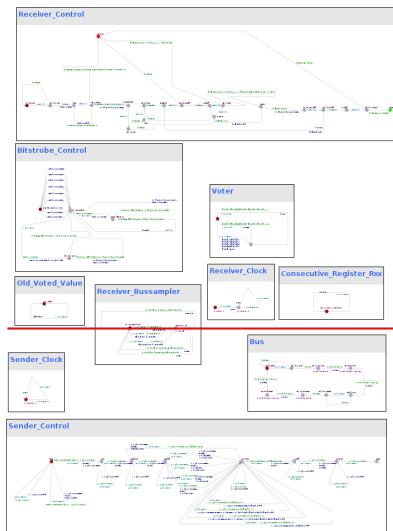
# Modeling Hardware: Parameterized, Clock Dependencies



**Figure:** Optimized model of the sender's oscillator and the bus, synchronizing on action *tick*. Here, we use just the clock needed to generate the *tick* action of the sender's oscillator, under the assumption that  $\text{PMIN} \leq \text{PMAX} \leq \text{CYCLE\_MIN}$  holds.

# Modeling: Sender / Receiver Partition

partition model into sender / receiver  
→ just 2 clocks needed



# Modeling Message Transfer: Message Format

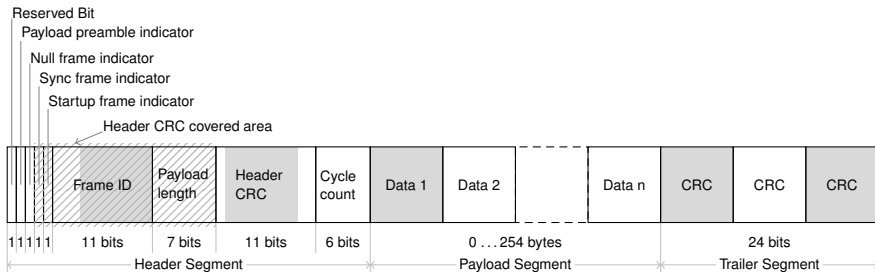
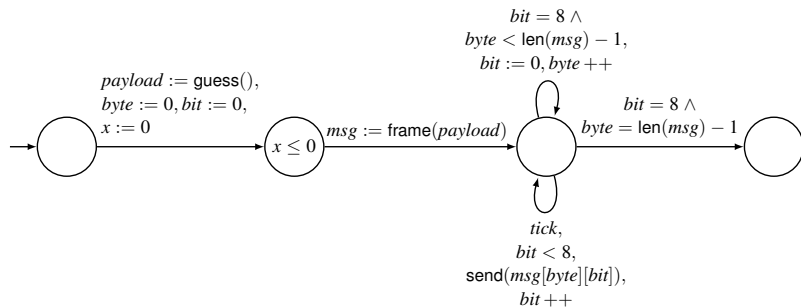


Figure: Format of a message frame.

# Modeling Message Transfer: The Hard Way



**Figure:** Generating a message payload, encasing it in the frame format, and sending it.

# Modeling Message Transfer: Use Nondeterminism

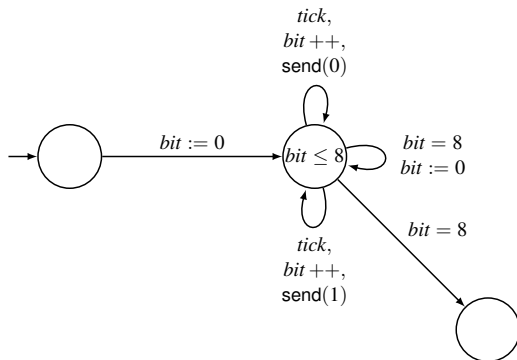
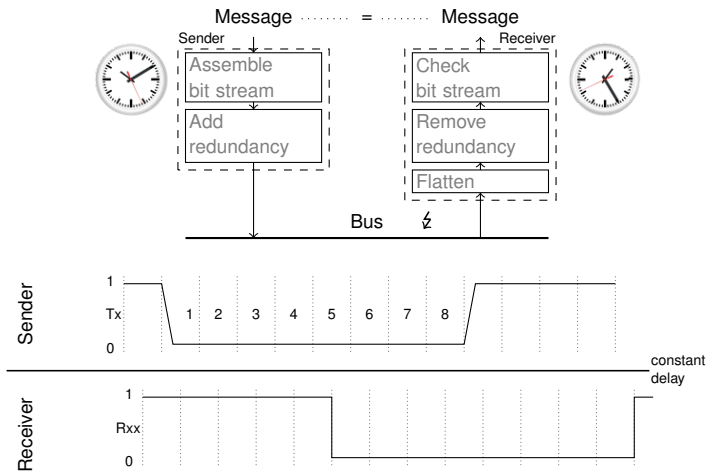


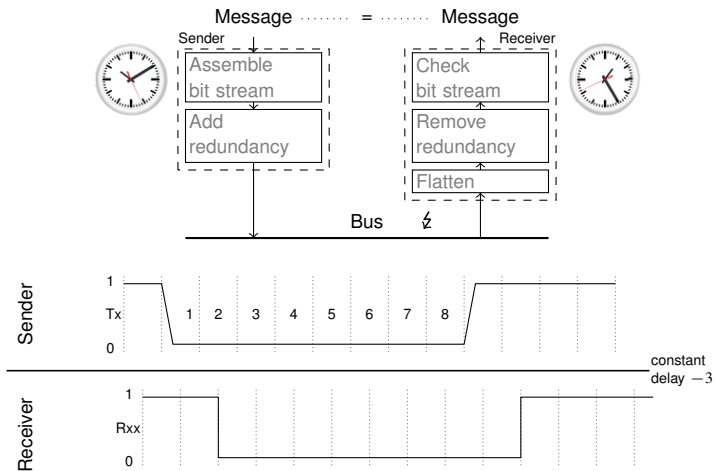
Figure: Abstracting from its actual length, contents and format.



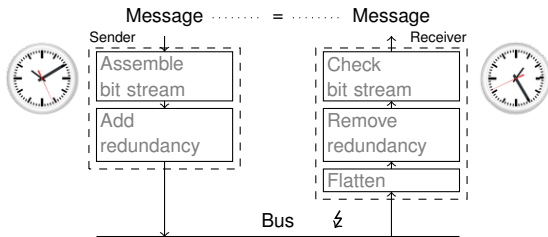
# One-way communication? Freely choose constant delays!



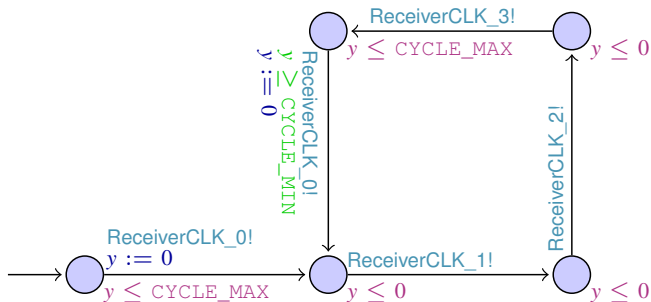
# One-way communication? Freely choose constant delays!



# Time only counts at the bus



# Abstract from time using order: Receiver Clock Model

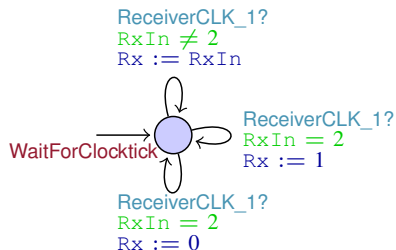


Component B should be executed after component B:

Synchronize B with "ReceiverCLK\_i?" if

$\exists j < i$  s.t. A synchronizes with "ReceiverCLK\_j?".

# Sampling from the bus



2 is nondeterministically resolved to 1 or 0.

RxIn: value seen by receiver

Rx: value received

# Modeling: Sampling at time $t$

The bus should be stable in  $[t - \text{SETUP}, t + \text{HOLD}]$ .

The bus can be unstable in  $[tick_{sd} + \text{PMIN}, tick_{sd} + \text{PMAX}]$ .

$$\begin{aligned} & \forall tick_{sd}. \forall t' \in [t - \text{SETUP}, t + \text{HOLD}]. t' \notin [tick_{sd} + \text{PMIN}, tick_{sd} + \text{PMAX}] \\ \Leftrightarrow & \forall tick_{sd}. (t + \text{HOLD} < tick_{sd} + \text{PMIN}) \vee (t - \text{SETUP} > tick_{sd} + \text{PMAX}) \\ \Leftrightarrow & \forall tick_{sd}. (t < tick_{sd} + \text{PMIN} - \text{HOLD}) \vee (t > tick_{sd} + \text{PMAX} + \text{SETUP}) \\ \Leftrightarrow & \forall tick_{sd}. t \notin [tick_{sd} + \text{PMIN} - \text{HOLD}, tick_{sd} + \text{PMAX} + \text{SETUP}] \end{aligned}$$

Add WIREDELAY = HOLD - PMIN:

$$\begin{aligned} & \forall tick_{sd}. t \notin \\ & [tick_{sd} + \text{PMIN} - \text{HOLD} + \text{WIREDELAY}, tick_{sd} + \text{PMAX} + \text{SETUP} + \text{WIREDELAY}] \\ \Leftrightarrow & \forall tick_{sd}. t \notin [tick_{sd}, tick_{sd} + \text{PMAX} + \text{SETUP} + \text{WIREDELAY}] \end{aligned}$$

## Modeling: Sampling at time $t$

The bus should be stable in  $[t - \text{SETUP}, t + \text{HOLD}]$ .

The bus can be unstable in  $[tick_{sd} + \text{PMIN}, tick_{sd} + \text{PMAX}]$ .

$$\begin{aligned} & \forall tick_{sd}. \forall t' \in [t - \text{SETUP}, t + \text{HOLD}]. t' \notin [tick_{sd} + \text{PMIN}, tick_{sd} + \text{PMAX}] \\ \Leftrightarrow & \forall tick_{sd}. (t + \text{HOLD} < tick_{sd} + \text{PMIN}) \vee (t - \text{SETUP} > tick_{sd} + \text{PMAX}) \\ \Leftrightarrow & \forall tick_{sd}. (t < tick_{sd} + \text{PMIN} - \text{HOLD}) \vee (t > tick_{sd} + \text{PMAX} + \text{SETUP}) \\ \Leftrightarrow & \forall tick_{sd}. t \notin [tick_{sd} + \text{PMIN} - \text{HOLD}, tick_{sd} + \text{PMAX} + \text{SETUP}] \end{aligned}$$

Add WIREDELAY = HOLD - PMIN:

$$\begin{aligned} & \forall tick_{sd}. t \notin \\ & [tick_{sd} + \text{PMIN} - \text{HOLD} + \text{WIREDELAY}, tick_{sd} + \text{PMAX} + \text{SETUP} + \text{WIREDELAY}] \\ \Leftrightarrow & \forall tick_{sd}. t \notin [tick_{sd}, tick_{sd} + \text{PMAX} + \text{SETUP} + \text{WIREDELAY}] \end{aligned}$$

## Modeling: Sampling at time $t$

The bus should be stable in  $[t - \text{SETUP}, t + \text{HOLD}]$ .

The bus can be unstable in  $[tick_{sd} + \text{PMIN}, tick_{sd} + \text{PMAX}]$ .

$$\begin{aligned} & \forall tick_{sd}. \forall t' \in [t - \text{SETUP}, t + \text{HOLD}]. t' \notin [tick_{sd} + \text{PMIN}, tick_{sd} + \text{PMAX}] \\ \Leftrightarrow & \forall tick_{sd}. (t + \text{HOLD} < tick_{sd} + \text{PMIN}) \vee (t - \text{SETUP} > tick_{sd} + \text{PMAX}) \\ \Leftrightarrow & \forall tick_{sd}. (t < tick_{sd} + \text{PMIN} - \text{HOLD}) \vee (t > tick_{sd} + \text{PMAX} + \text{SETUP}) \\ \Leftrightarrow & \forall tick_{sd}. t \notin [tick_{sd} + \text{PMIN} - \text{HOLD}, tick_{sd} + \text{PMAX} + \text{SETUP}] \end{aligned}$$

Add WIREDELAY = HOLD - PMIN:

$$\begin{aligned} & \forall tick_{sd}. t \notin \\ & [tick_{sd} + \text{PMIN} - \text{HOLD} + \text{WIREDELAY}, tick_{sd} + \text{PMAX} + \text{SETUP} + \text{WIREDELAY}] \\ \Leftrightarrow & \forall tick_{sd}. t \notin [tick_{sd}, tick_{sd} + \text{PMAX} + \text{SETUP} + \text{WIREDELAY}] \end{aligned}$$



## Modeling: Sampling at time $t$

The bus should be stable in  $[t - \text{SETUP}, t + \text{HOLD}]$ .

The bus can be unstable in  $[tick_{sd} + \text{PMIN}, tick_{sd} + \text{PMAX}]$ .

$$\begin{aligned} & \forall tick_{sd}. \forall t' \in [t - \text{SETUP}, t + \text{HOLD}]. t' \notin [tick_{sd} + \text{PMIN}, tick_{sd} + \text{PMAX}] \\ \Leftrightarrow & \forall tick_{sd}. (t + \text{HOLD} < tick_{sd} + \text{PMIN}) \vee (t - \text{SETUP} > tick_{sd} + \text{PMAX}) \\ \Leftrightarrow & \forall tick_{sd}. (t < tick_{sd} + \text{PMIN} - \text{HOLD}) \vee (t > tick_{sd} + \text{PMAX} + \text{SETUP}) \\ \Leftrightarrow & \forall tick_{sd}. t \notin [tick_{sd} + \text{PMIN} - \text{HOLD}, tick_{sd} + \text{PMAX} + \text{SETUP}] \end{aligned}$$

Add WIREDELAY = HOLD - PMIN:

$$\begin{aligned} & \forall tick_{sd}. t \notin \\ & [tick_{sd} + \text{PMIN} - \text{HOLD} + \text{WIREDELAY}, tick_{sd} + \text{PMAX} + \text{SETUP} + \text{WIREDELAY}] \\ \Leftrightarrow & \forall tick_{sd}. t \notin [tick_{sd}, tick_{sd} + \text{PMAX} + \text{SETUP} + \text{WIREDELAY}] \end{aligned}$$

## Modeling: Sampling at time $t$

The bus should be stable in  $[t - \text{SETUP}, t + \text{HOLD}]$ .

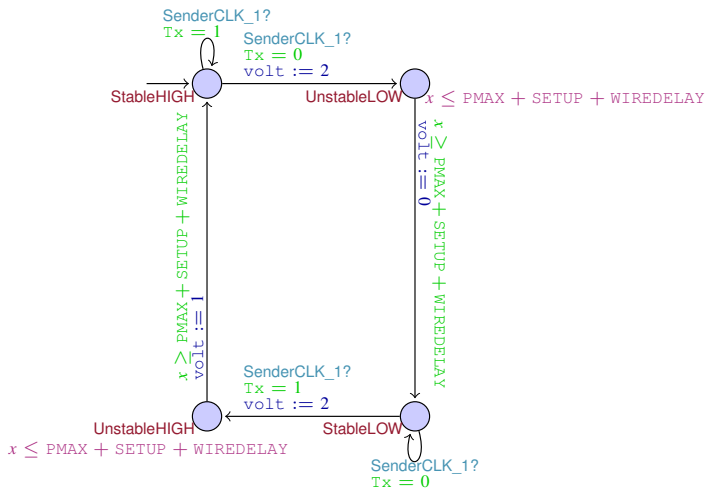
The bus can be unstable in  $[tick_{sd} + \text{PMIN}, tick_{sd} + \text{PMAX}]$ .

$$\begin{aligned} & \forall tick_{sd}. \forall t' \in [t - \text{SETUP}, t + \text{HOLD}]. t' \notin [tick_{sd} + \text{PMIN}, tick_{sd} + \text{PMAX}] \\ \Leftrightarrow & \forall tick_{sd}. (t + \text{HOLD} < tick_{sd} + \text{PMIN}) \vee (t - \text{SETUP} > tick_{sd} + \text{PMAX}) \\ \Leftrightarrow & \forall tick_{sd}. (t < tick_{sd} + \text{PMIN} - \text{HOLD}) \vee (t > tick_{sd} + \text{PMAX} + \text{SETUP}) \\ \Leftrightarrow & \forall tick_{sd}. t \notin [tick_{sd} + \text{PMIN} - \text{HOLD}, tick_{sd} + \text{PMAX} + \text{SETUP}] \end{aligned}$$

Add  $\text{WIREDELAY} = \text{HOLD} - \text{PMIN}$ :

$$\begin{aligned} & \forall tick_{sd}. t \notin \\ & [tick_{sd} + \text{PMIN} - \text{HOLD} + \text{WIREDELAY}, tick_{sd} + \text{PMAX} + \text{SETUP} + \text{WIREDELAY}] \\ \Leftrightarrow & \forall tick_{sd}. t \notin [tick_{sd}, tick_{sd} + \text{PMAX} + \text{SETUP} + \text{WIREDELAY}] \end{aligned}$$

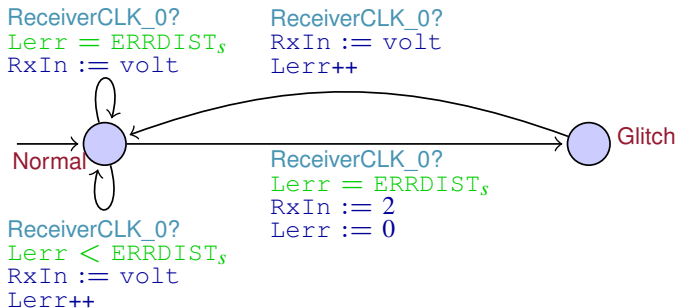
# Bus model



`volt`: value of the bus, with 1=HIGH, 0=LOW, and 2=undefined

`Tx`: value sent

# Bit flips: Sample Glitch Model: 1 in $(ed+1)$



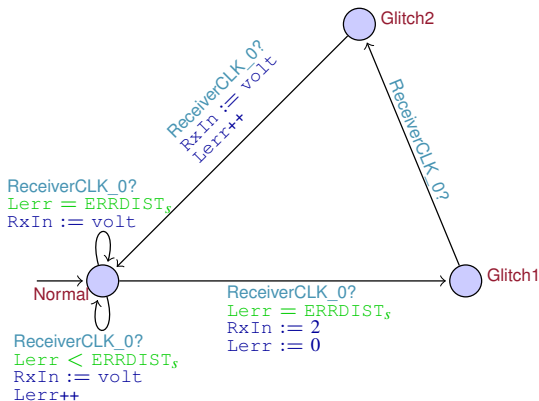
$ERRDIST_s$ : minimum distance between two glitches

$Lerr$ : distance from the last error (if smaller  $ERRDIST_s$ ) (initialized with  $ERRDIST_s$ )

`volt`: value from the bus, with 1=HIGH, 0=LOW, and 2=undefined

`RxIn`: value seen by receiver

# Bit flips: Sample Glitch Model: 2 adjacent in (ed+2)



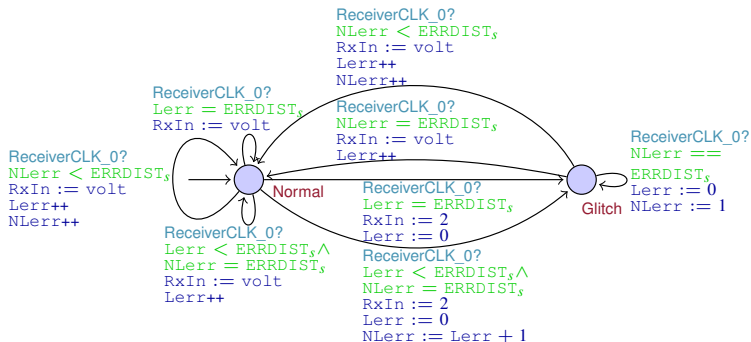
$ERRDIST_s$ : minimum distance between two glitches

$L_{err}$ : distance from the last error (if smaller  $ERRDIST_s$ ) (initialized with  $ERRDIST_s$ )

`volt`: value from the bus, with 1=HIGH, 0=LOW, and 2=undefined

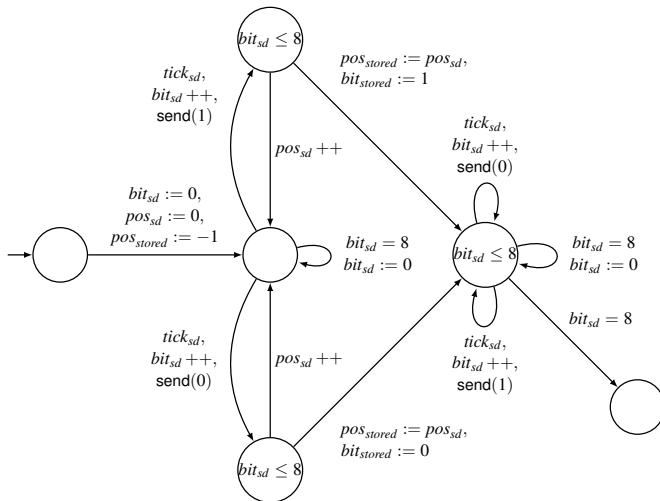
`RxIn`: value seen by receiver

# Bit flips: Sample Glitch Model: 2 arbitrary in (ed+1)



NLerr: distance from the second to last error (if smaller  $ERRDIST_s$ )  
 (initialized with  $ERRDIST_s$ )

# Verifying Message Transfer: Use Nondeterminism



**Figure:** Sender model: nondeterministically choose a bit to store and store its position as well.

# Verifying Message Transfer: Use Nondeterminism

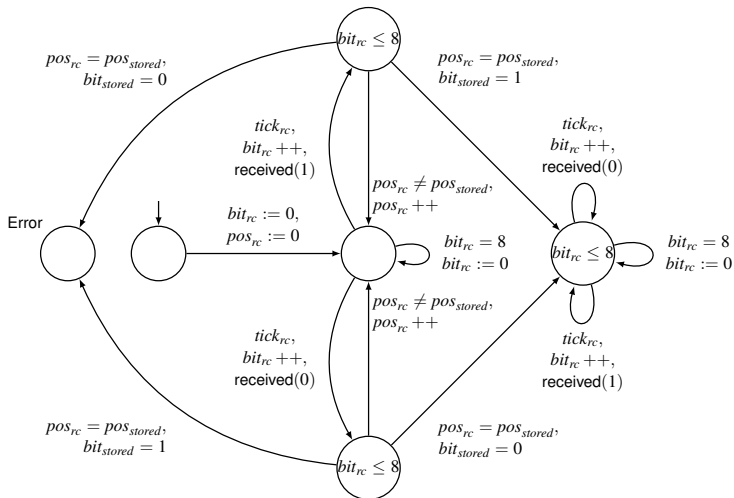


Figure: Receiver model: check the bit received at the position of the stored bit.



# Verifying Message Transfer: Use Nondeterminism

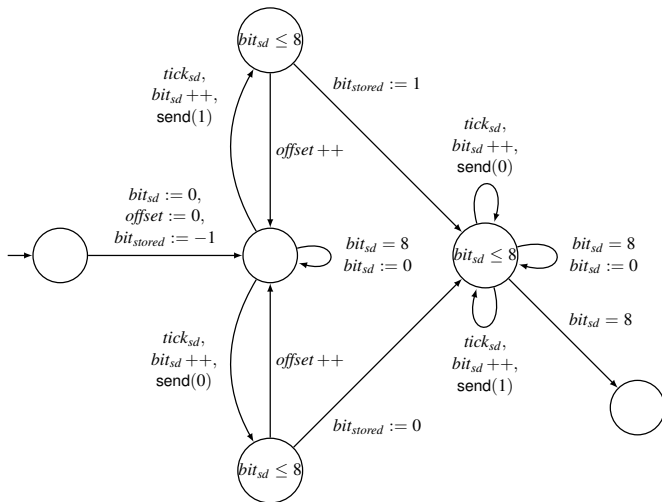


Figure: Sender model: store the number of bits in transit in *offset*.

# Verifying Message Transfer: Use Nondeterminism

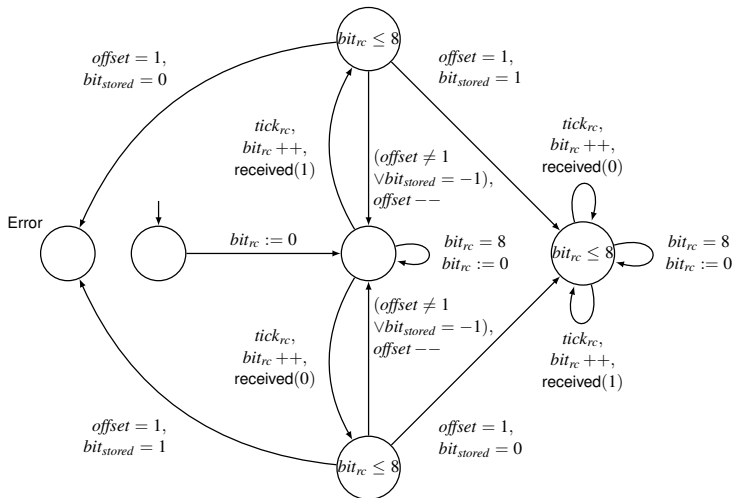
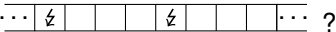
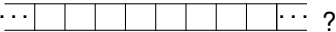


Figure: Receiver model: if a bit is stored, check the bit received with  $offset = 1$ ).

# Recall: FlexRay Error Resilience



⚡ FlexRay



FlexRay Protocol Specification

3.2.1.4 Frame bit error assembly

3.2.2 Bitstream error

3.2.7 Signal integrity

**FlexRay Communications System Protocol Specification Version 2.1**

Revision A

In general, there are several conditions (e.g. clock oscillator differences, electrical characteristics of the transmission media or the transceivers, ESD, etc.) that can cause variations of signal timing or introduce anomalies/glitches into the communication bit stream. The decoding function attempts to enable tolerance of the physical layer against presence of one glitch in a bit cell when the length of the glitch is less than or equal to one channel sample clock period. If

There are specific cases where a single glitch cannot be tolerated and others where two glitches can be tolerated.

3.2.1.4 Frame bit error assembly

3.2.2 Bitstream error

3.2.7 Signal integrity

The protocol guarantees tolerance for

- 1 glitch in every sequence of 4 consecutive samples (1 out of 4)
- 2 glitches in every sequence of 88 consecutive samples (2 out of 88)

Note: one message  $\approx$  21.000 samples

The protocol guarantees tolerance for

- 1 glitch in every sequence of 4 consecutive samples (1 out of 4)

E.g.: ... 

z				z		z
---	--	--	--	---	--	---

 ...

- 2 glitches in every sequence of 88 consecutive samples (2 out of 88)

Note: one message  $\approx$  21.000 samples

The protocol guarantees tolerance for

- 1 glitch in every sequence of 4 consecutive samples (1 out of 4)

E.g.: ... 

z				z		z
---	--	--	--	---	--	---

 ...

- 2 glitches in every sequence of 88 consecutive samples (2 out of 88)

Note: one message  $\approx$  21.000 samples

The protocol guarantees tolerance for

- 1 glitch in every sequence of 4 consecutive samples (1 out of 4)

E.g.: ... 

z			z		z
---	--	--	---	--	---

 ...

- 2 glitches in every sequence of 88 consecutive samples (2 out of 88)

Note: one message  $\approx$  21.000 samples

The protocol guarantees tolerance for

- 1 glitch in every sequence of 4 consecutive samples (1 out of 4)

E.g.: ... 

✓			✓		✓
---	--	--	---	--	---

 ...

- 2 glitches in every sequence of 88 consecutive samples (2 out of 88)

Note: one message  $\approx$  21.000 samples



The protocol guarantees tolerance for

- 1 glitch in every sequence of 4 consecutive samples (1 out of 4)

E.g.: ... 

z				z		z
---	--	--	--	---	--	---

 ...


- 2 glitches in every sequence of 88 consecutive samples (2 out of 88)

Note: one message  $\approx$  21.000 samples

# Glitch Model: Glitch Patterns

The protocol guarantees tolerance for

- 1 glitch in every sequence of 4 consecutive samples (1 out of 4)

E.g.: ...  ...

- 2 glitches in every sequence of 88 consecutive samples (2 out of 88)

Note: one message  $\approx$  21.000 samples

The protocol guarantees tolerance for

- 1 glitch in every sequence of 4 consecutive samples (1 out of 4)
- 2 glitches in every sequence of 88 consecutive samples (2 out of 88)

Note: one message  $\approx$  21.000 samples

The protocol guarantees tolerance for

- 1 glitch in every sequence of 4 consecutive samples (1 out of 4)
- 2 glitches in every sequence of 88 consecutive samples (2 out of 88)

E.g.: ... 

	z		z			
--	---	--	---	--	--	--

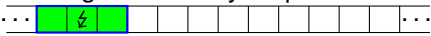
 ...

Note: one message  $\approx$  21.000 samples

# Glitch Tolerance vs. Voting Window Size

Voting window size of

- 3, tolerates 1 glitch in every sequence of 3 consecutive samples.

E.g.: 

- 5, tolerates 1 glitch in every sequence of 4 consecutive samples.

E.g.: 

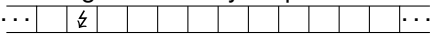
- 7, tolerates 1 glitch in every sequence of 5 consecutive samples.

- 9, tolerates 1 glitch in every sequence of 6 consecutive samples.

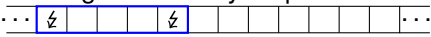
# Glitch Tolerance vs. Voting Window Size

Voting window size of

- 3, tolerates 1 glitch in every sequence of 3 consecutive samples.

E.g.: 

- 5, tolerates 1 glitch in every sequence of 4 consecutive samples.

E.g.: 

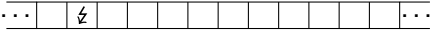
- 7, tolerates 1 glitch in every sequence of 5 consecutive samples.

- 9, tolerates 1 glitch in every sequence of 6 consecutive samples.

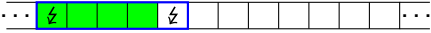
# Glitch Tolerance vs. Voting Window Size

Voting window size of

- 3, tolerates 1 glitch in every sequence of 3 consecutive samples.

E.g.: 

- 5, tolerates 1 glitch in every sequence of 4 consecutive samples.

E.g.: 

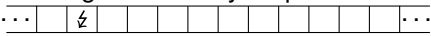
- 7, tolerates 1 glitch in every sequence of 5 consecutive samples.

- 9, tolerates 1 glitch in every sequence of 6 consecutive samples.

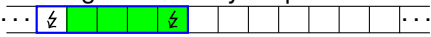
# Glitch Tolerance vs. Voting Window Size

Voting window size of

- 3, tolerates 1 glitch in every sequence of 3 consecutive samples.

E.g.: 

- 5, tolerates 1 glitch in every sequence of 4 consecutive samples.

E.g.: 

- 7, tolerates 1 glitch in every sequence of 5 consecutive samples.

- 9, tolerates 1 glitch in every sequence of 6 consecutive samples.



# Glitch Tolerance vs. Voting Window Size

Voting window size of

- 3, tolerates 1 glitch in every sequence of 3 consecutive samples.

E.g.: 

...		z										...
-----	--	---	--	--	--	--	--	--	--	--	--	-----

- 5, tolerates 1 glitch in every sequence of 4 consecutive samples.

E.g.: 

...	z			z						...
-----	---	--	--	---	--	--	--	--	--	-----

- 7, tolerates 1 glitch in every sequence of 5 consecutive samples.

- 9, tolerates 1 glitch in every sequence of 6 consecutive samples.

# Glitch Tolerance vs. Voting Window Size

Voting window size of

- 3, tolerates 1 glitch in every sequence of 3 consecutive samples.

E.g.: 

...		z											...
-----	--	---	--	--	--	--	--	--	--	--	--	--	-----

- 5, tolerates 1 glitch in every sequence of 4 consecutive samples.

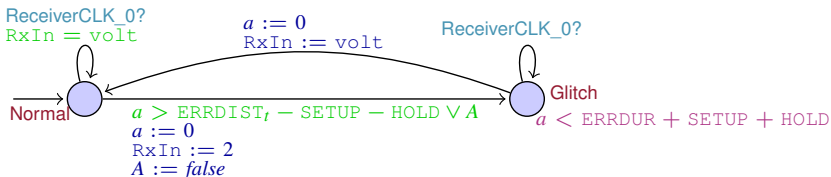
E.g.: 

...	z				z							...
-----	---	--	--	--	---	--	--	--	--	--	--	-----

- 7, tolerates 1 glitch in every sequence of 5 consecutive samples.

- 9, tolerates 1 glitch in every sequence of 6 consecutive samples.

# Bit flips: Real-Time Glitch Model: 1\*Y during XX



fresh clock  $a$  (clocks are always initialised with 0)

$A$ : flag saying there was no glitch so far (initialized with `true`)

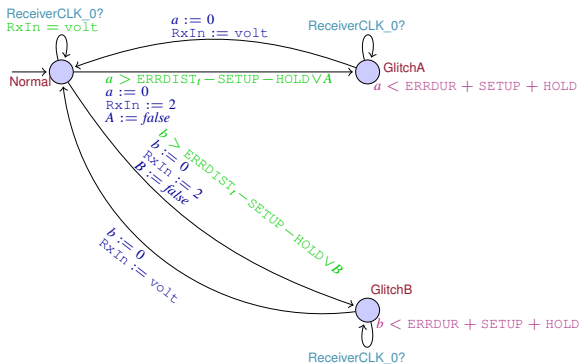
$ERRDIST_t$ : distance from the last error

$ERRDUR$ : maximal duration of an disturbance on the bus

`volt`: value from the bus, with 1=HIGH, 0=LOW, and 2=undefined

`RxIn`: value seen by receiver

# Bit flips: Real-Time Glitch Model: 2\*Y indep. during XX



fresh clocks  $a$  and  $b$  (clocks are always initialised with 0)

$A(B)$ : flag saying there was no glitch  $A$  (glitch  $B$ ) so far (initialized with `true`)

$ERRDIST_t$ : distance from the last error

`volt`: value from the bus, with 1=HIGH, 0=LOW, and 2=undefined

`RxIn`: value seen by receiver

# Parameter Exploration

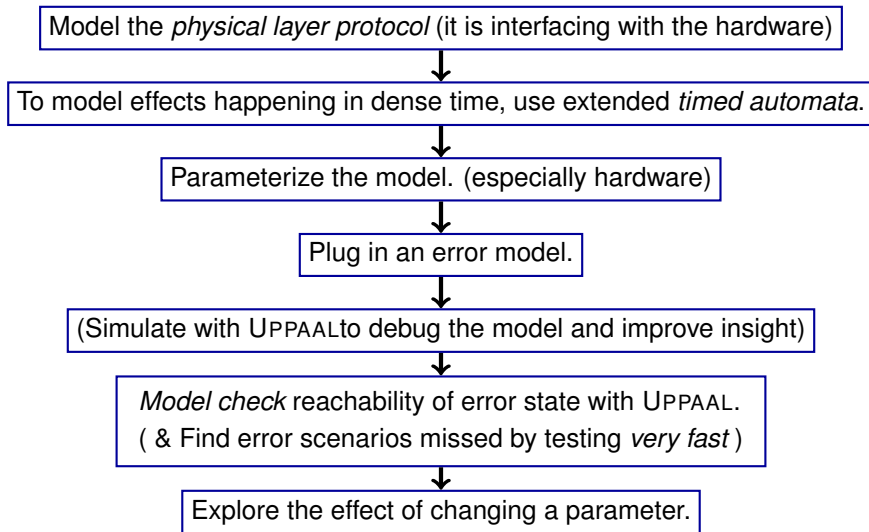
Parameter exploration using binary search:  
boundaries for variation of a single parameter (Voting window size 5)

glitch tolerance	delay variance
(1 of 4),(2 at most),(short Real-Time Glitch (RTG))	1.435ns → 7.6075ns
(2 (adj./ind.) of 88),(long RTG),(2 ind. short RTGs)	1.435ns → 7.5700ns
(1 at most)	1.435ns → 12.020ns

glitch tolerance	deviation of clock from standard rate
(1 of 4),(2 at most),(short RTG)	0.15% → 0.46%
(2 ind. of 88),(2 ind. short RTGs)	0.15% → 0.40%
(2 adj. of 88),(long RTG)	0.15% → 0.45%
(1 at most)	0.15% → 1.09%
(no glitches)	0.15% → 1.74%

(RTGs configured to be equivalent to the respective Sample Glitches)

# Summary:Our Approach



# Summary: Model Design Principles

- Separate the protocol into *components*.
- Abstract from time where order suffices.
- You need time? *Ignore constant delays* if communication is one-way only.
- Reduce the number of *clocks* by exploiting dependencies.
- *Nondeterminism* allows you to forget message information (content, length, format).
- Use a *parameterized hardware* model for easy adoption to different hardware.