ISSN 0280-5316
ISRN LUTFD2/TFRT--5805--SE

Distributed System Control with FlexRay Nodes in Commercial Vehicles

Carl Hoffstedt

Department of Automatic Control
Lund University
November 2007

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
November 2007

Document Number
ISRNLUTFD2/TFRT..5805--SE

Author(s)
Carl Hoffstedt

Supervisor
Anders Lindqvist, Haldex AB Landskrona
Anton Cervin Automatic Control in Lund
(Examiner)

Sponsoring organization

Title and subtitle
Distributed System Control with FlexRay Nodes in Commercial Vehicles (Distribuerad systemkontroll med FlexRay-noder I tunga fordon)

Abstract
Most vehicles today use some kind of hydraulic brake system in order to stop the vehicle. There are other techniques that will not only decrease the stop length, but also create a more stable vehicle; an example of such system is the Electromechanical braking system. The problem with such a system is that the requirements of data transmission and reliability are different from the system that is used today.
To be able to implement such a system, a relatively new network protocol called FlexRay can be used.
Different design alternatives will be introduced and one important subject is how the local nodes in the wheels can take over calculations, in a distributed way, from more central nodes. The best design was simulated using Matlab and a library called TrueTime to investigate if it is possible to implement such a system.
The results from the simulation shows that it is possible to implement the system and that there is a lot of free space on the network for other tasks. An animation shows how the different nodes exchange redundancy responsibilities. One conclusion was that it is better to choose a simple design of the system instead of an advanced one where the nodes are connected to several networks to add redundancy. Once again the expression “less is more” is valid!

Keywords
FlexRay, distributed system control, safety and redundancy, real time performance, brake system, wheel modules, simulation, TrueTime.

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
65

Recipient’s notes

Security classification

Acknowledgements

First of all, I want to thank Jacob Svendenius for the contact with Anders Lindqvist at Haldex Brake Products AB.

I would also want to thank the following people for their engagement in the process of this master thesis:

- *Peter Nilsson*, technical expert at *Haldex Brake Products AB*.
- *Anders Lindqvist*, supervisor at *Haldex Brake Products AB*.
- *Anton Cervin*, supervisor at *Lund University*, LU.

A special thought is given to my lovely Maria Olofsson, for her understanding while this master thesis was written. I love you!

Table of Contents

Terminology

Abbreviations

ECU	Electronic Control Unit.
CAN	Controller Area Network.
TTCAN	Time Trigged CAN
FlexRay	FlexRay is a new automotive network communications protocol under development by the FlexRay Consortium
EMB	Electromechanical braking system
DSP	Digital Signal Processor
TDMA	Time Division Multiple Access
DCS	Distributed Control System
TMR	Triple Modular Redundancy

1 Introduction

FlexRay is a relatively new protocol, developed to accomplish the new requirements of data transmission and reliability in local automobile networks. Haldex Brake Products AB develops an *electromechanical brake system* – EMB – for heavy vehicles. FlexRay offers new opportunities compared to the CAN (*Controlled Area Network*) standard which dominates the automobile market today. There is a need to develop functions to take care of the new possibilities FlexRay offers to a safety critical system.

Some of the interesting topics are; distribution of the nodes used for the calculations of the dynamics of the vehicle, real time performance, synchronization of nodes in the network, safety and redundancy, and configuration of the network.

These topics give rise to questions like; what is the best way to design the network to handle redundancy but still use the full capacity of the network? What happens if one or more nodes are disconnected, or even worse – if the link between several nodes fall off?

1.1 Purpose

The thesis is intended to investigate possible cost and performance improvements in a future electromechanical brake system for a heavy vehicle by the introduction of a FlexRay-network and a distributed system control. The task is to design a system where the central vehicle brake-control is managed by advanced local brake control units, which are communicating through a non-determined number of FlexRay networks (see figure 1). Vehicle and driver information are interfaced through FlexRay or other communication networks e.g. CAN.

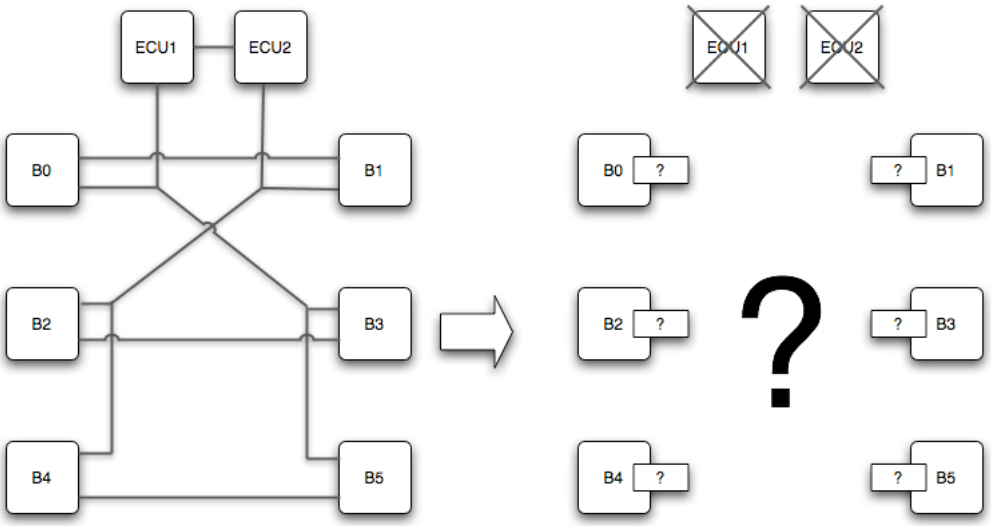


Figure 1 – *The left picture illustrates an example of one present system. The right picture symbolizes one of the main subjects of the thesis – to determine a new design without the Electronic Control Unit (ECU) controllers.*

1.2 Thesis objectives

The main objective of the thesis is to describe how to design a future distributed network at minimal cost but still at high performance. At this time, there is no such system at Haldex that responds to the new requirements. So hopefully, this thesis can add a new dimension to Haldex advanced engineering team.

1.3 Problems to solve

The main problem is to determine a new network design for the FlexRay nodes and the corresponding channels in a distributed system. To solve this several questions need to be answered:

1) How should the FlexRay communication channels from each node be connected in the network to create an optimal distributed control system?

2) How should the FlexRay channels be configured to optimize the correct number of messages to be sent from each node?

3) In the system used today the *Electronic Control Units* (ECU) have some sort of redundancy to handle errors in calculations e.g. regarding the dynamics of the vehicle. How can the same level of redundancy be achieved in the distributed network without the ECU:s?

4) If there is an error in the calculations or if a node falls off – how can this be detected and corrected?

5) In order to send and receive dynamic data over the network the synchronization of the messages needs to be solved.

1.4 Delimitations

Due to the extent of the subject some delimitations needs to be stated.

- The work is mainly theoretical. It means that no real hardware will be used.
- The hardware in the wheel modules will be seen as black boxes with two FlexRay communication channels. The inside of the boxes consist of one to several processors connected to a FlexRay-controller.
- The evaluation of the design assumes 4 – 10 wheel modules divided on 2 – 5 axles.

1.5 Methodology

Most of the work is to investigate different design alternatives and evaluate them. To be able to compare the alternatives a literature study is required. The literature studies will mostly concern distributed system control, redundancy and real-time performance, but in some cases the FlexRay-protocol needs to be evaluated – especially in order to understand the synchronization of messages and how the filtering of messages can be used.

The evaluation of the different design alternatives will consist of calculations and simulations using Matlab/Simulink.

2 Background material

Ever since the first car was set in motion there has been a need of stopping the vehicle sooner or later. Modern cars get more and more advanced due to the increasing number of ECU:s which soon control every movement of the vehicle. In this chapter some background information about the newest brake technology and an example of a FlexRay network architecture is introduced, showing the access bus, the cycle and a schedule table of the messages.

2.1 The FlexRay protocol

FlexRay is a time trigged communication protocol, built on *Time Division Multiple Access* (TDMA), first and foremost designed for automotive applications. The protocol offers two 10 Mbit/s channels for transferring data on the medium. The nodes can be connected to each other by various network topologies. The possible topology configurations are; bus, star or a combination of both. Each node is using the A, B or A+B channel for accessing the medium (see figure 2). [4]

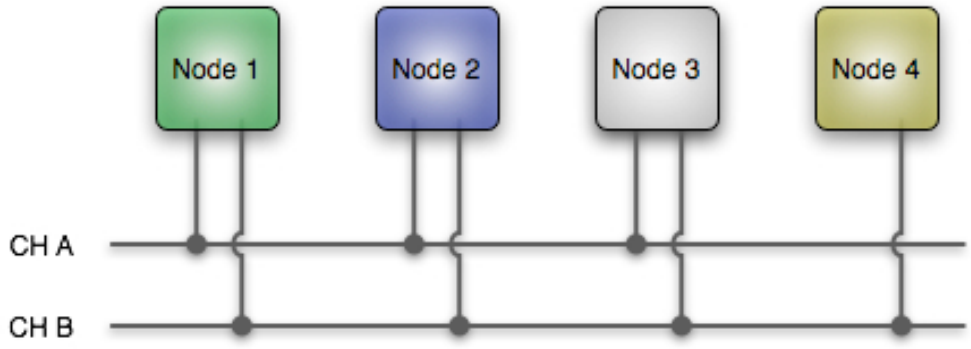


Figure 2 – *Four nodes are sharing the same medium using the bus topology.*

Every node has it own time slots in a cycle that repeats over and over again. The cycle time is 10 – 16000 µs. Each cycle consists of 2 – 1023 static slots depending of the length of the cycle and the slot length [5]. The cycle also includes an optional part where the nodes can send dynamic messages using the byteflight-protocol [6] (see figure 3).

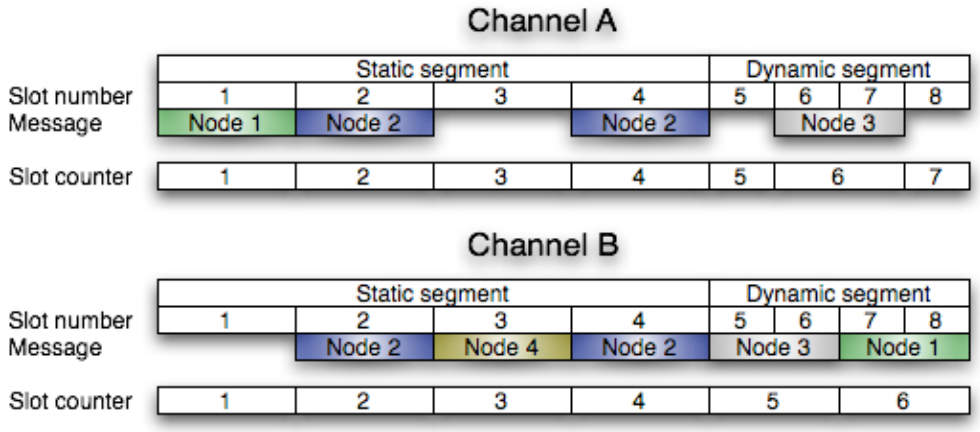


Figure 3 – *The table shows a simplified example of one cycle of each channel. The network has four nodes, which are sharing the same medium sending static and dynamic data.*

As the protocol is deterministic, the configuration of the static part of the FlexRay-cycle results in a schedule table (see figure 4).

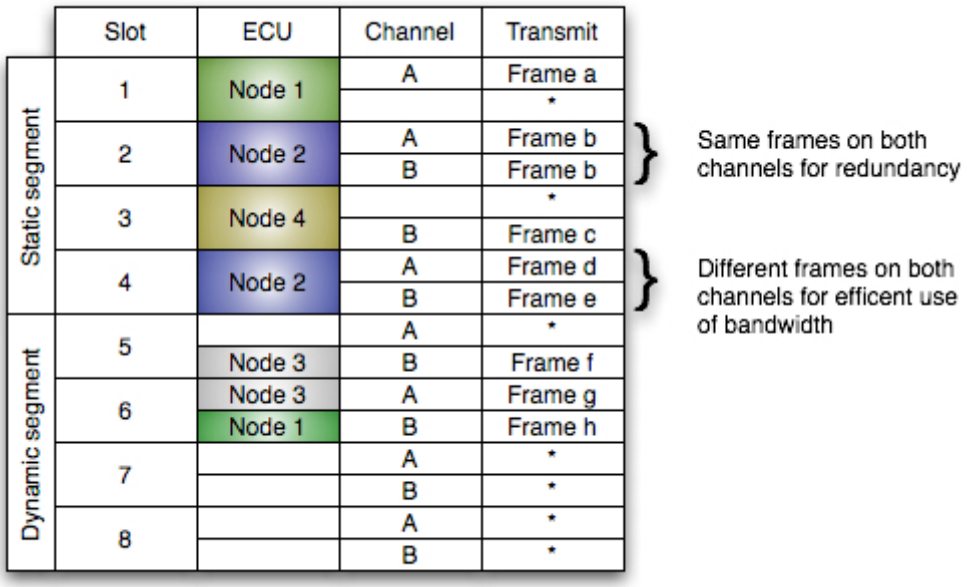


Figure 4 – *The timetable shows what frames are sent on the different channels in the different slots. The table is divided into static and a dynamic segment. Note the special about slot 2 and 4.*

There is a possibility of using different schedule tables in different cycles. Using a more technical term this is called cycle multiplexing. This might be used if there is a task, which needs to be transmitted once every 10 cycle-time for example.

2.2 Configuration of FlexRay

The settings for the FlexRay communication are extensive. There are at least 30 parameters and constants, which need to be within the right limits to get the network working properly. Every path in the network needs to be specified in the configuration in terms of which channels to use, which slots each node uses, and so on. As a result of this, a totally known deterministic timetable is created, as was shown in figure 4.

To determine how much data is transmitted in a cycle the following statements and calculation can be used:

Every message contains: `cPayLoadLengthMax`

A FlexRay cycle is: `gdCycle` [µs]

A slot time is: `gdStaticSlot` [µs]

```
nbrOfSlotPerChannel = gdCycle/gdStaticSlot                    (1)
```

```
nbrOfWordsPerCycle = nbrOfSlotPerChannel * cPayLoadLengthMax  (2)
```

This means that the cycle can use at most *`nbrOfWordsPerCycle`* for static or dynamic messages.

2.3 Electromechanical braking systems

Electromechanical braking systems, EMB, or brake-by-wire, replace regular hydraulic braking systems with a completely “dry” electrical component system. In many solutions there is no backup of hydraulic or mechanical systems, which means that the reliability is critical and that the system must be fault-tolerant. To implement such system there is an important need of redundancy in terms of power supply but also in fault-tolerant communication protocols. Present

standards such as CAN do not fulfill the requirements; better candidates are TTCAN or FlexRay. [3]

According to FreeScale [3] some of the key benefits of an EMB-system are:

- connects with emerging systems, such as adaptive cruise control
- reduces system weight to provide improved vehicle performance and economy
- reduces maintenance requirements and the pollutant sources by eliminating corrosive, toxic hydraulic fluids

2.4 The FlexRay-hardware

There are a lot of companies, including FreeScale, Fujitsi and Vector, specialized on the FlexRay hardware. Basically the hardware consists of a *Digital Signal Processor* (DSP) and a FlexRay controller board for interfacing the both channels (A and B) defined in the standard (see figure 5).

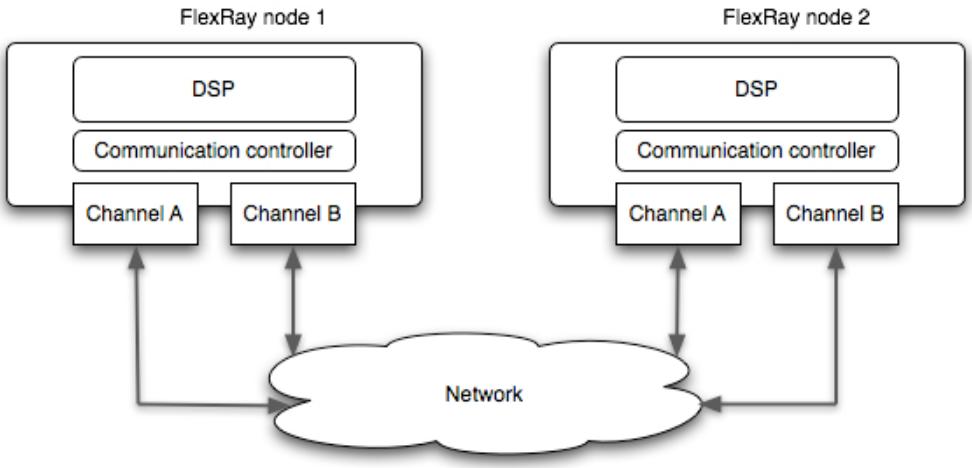


Figure 5 – *Two FlexRay nodes are connected to some kind of network.*

There is some delimitation to the settings of the different channels using the same communication controller. It is for example not possible to have different cycle times on the different channels.

To each channel there is a component called bus guardian (BG) which performs management of schedules and data, independent of the communication control-

ler. If there is a time gap, the bus guardian sends a signal to prevent the bus driver from sending data. At the same time it notifies the host of the error. This is only true for the static part of the frame [6]. All hardware does not have the bus guardian; this is the case with the hardware at Haldex at the moment.

An important point behind FlexRay is redundancy; because of this many developers choose to use a multi-processor solution. But as mentioned in section 1.4, the work in this thesis will consider the internal hardware as black boxes to make the subject more general.

2.5 The TrueTime library for Simulink

The TrueTime library is a Simulink-based tool for simulation of networked and embedded real-time control systems developed by Martin Ohlin, Dan Henriksson and Anton Cervin at the Department of Automatic Control, Lund University.

It is possible to connect nodes to a network and simulate different protocols. Unfortunately, there is no support for FlexRay but it is possible to use the TDMA protocol, which means that the static part of the segments can be simulated.

In some way, the Byteflight protocol used in the dynamic part, is based on a hybrid of synchronous/asynchronous TDMA. Basically, if the dynamic parts are going to be simulated, using the TDMA protocol with some modifications can do it.

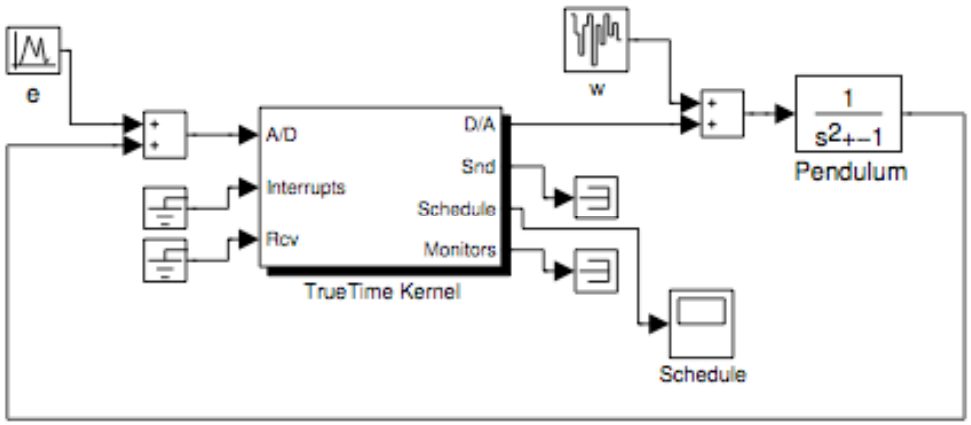


Figure 6 – *This example from the TrueTime reference manual shows a TrueTime kernel block connected to a continuous pendulum process. The TrueTime kernel can also be connected to a network to simulate TDMA traffic.*

3 Design alternatives

In section 1.1 – Figure 1 – the purpose of the thesis was illustrated by two images showing an example of a present system and a new one without communication channels. Figure 7 shows the same illustration but with electric wires added.

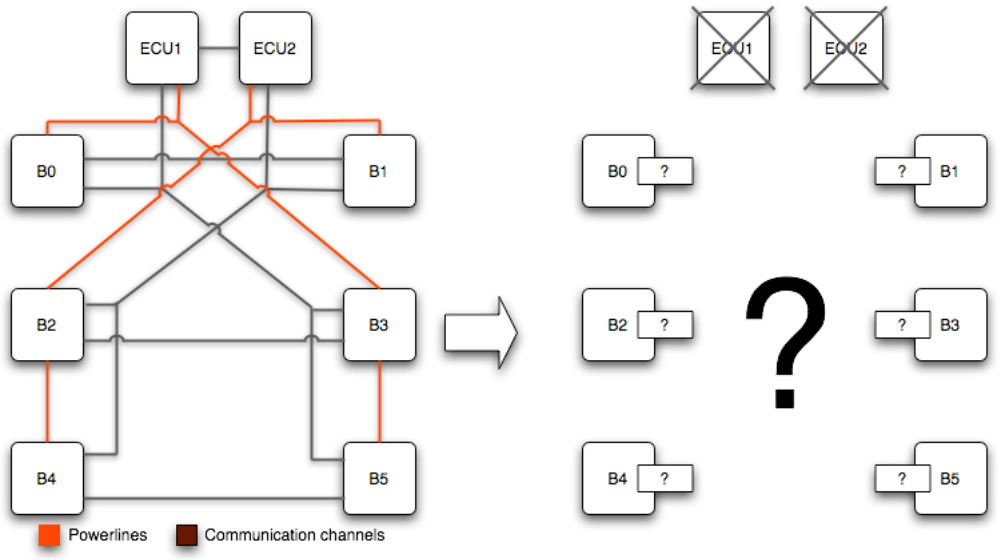


Figure 7 - *The left picture illustrates an example of a present system. The right picture symbolizes one of the main subjects of the thesis– to determine a new design without the Electronic Control Unit (ECU) controllers. Instead the ECU:s will be distributed in the network in some way.*

In this section different design alternatives will be discussed, using the right picture in figure 7 as a starting point.

3.1 Background

Vehicle and driver information is interfaced through FlexRay or other communication networks e.g. CAN. In the examples it will be assumed that every node is interfaced through CAN.

The different design alternatives will take into consideration that 4 to 10 wheels are in motion and that each wheel needs a FlexRay node.

In practice, every node is a brake module on each side of an axle on a vehicle. Every brake module has a need of sending X messages per unit of time. Haldex Brake System AB makes the following assumptions:

Unit of time	*Number of messages* per brake module and unit*	
2.5 ms	2	(Send sensor data, calculate slip-control)
10.0 ms	2	(Update the dynamics of the vehicle)

*) One message is equivalent to one FlexRay-slot with 16 words including the header. 1 word is 16 bits.

In fact, the 2.5 ms cycle is used for the most critical parts of the system and the 10 ms cycle is used for the dynamics of the vehicle. The reference value from the driver is also sent once every 10th ms.

3.2 Distributed control system

A *distributed control system*, also known as DCS refers to a control system, in which the controller elements are not central in location but are distributed throughout the system with each component sub-system controlled by one or more controllers [8].

In the example in figure 8, the existing system needs to get distributed by the removal of the two ECU:s. In this case it means that some or all nodes need to take over the responsibilities from the ECU:s and in some way determine the same results as the ECU:s did before.

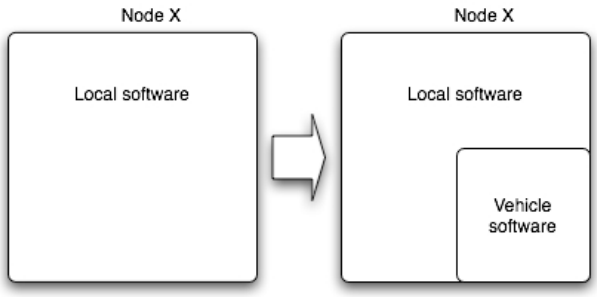


Figure 8 – *The new nodes needs to handle vehicle control software which was previously handled by the ECU:s.*

Some or all brake control units in a distributed control system execute both the local control software and the vehicle brake control software (see figure 8).

3.2.1 Alternatives for the distribution

The following alternatives are interesting:

- One or several nodes make all the calculations.
- All the nodes make all the calculations

3.3 Safety and redundancy

In this section different choice of network topologies and modularization of safety-critical parts of vehicle control software and local control software will be discussed regarding safety and redundancy for an optimal cost/performance ratio. Some kind of error handling will also by discussed.

An important question is which parts are safety-critical. As a definition, all common decisions concerning the vehicle control software and the slip-control of the wheels can be seen as highly safety-critical. The control of the electrical motor, which controls the brake force, or the reference signal from the driver are not that critical.

3.3.1 Finding errors

There are basically three kinds of main errors that can occur:

- A complete node or a network goes down. This could be the result of problems with power for example.

- A hardware error inside the node (in the DSP), meaning that some parts of the node are down. It could for example be an error in the memory that leads to a calculation error.

If a node fall off it is possible to see that the node is not sending anything to the network. If there is an error inside the node, but it still has power, one option to find the error is to use a specific kind of transmission type in the FlexRay-controller [9]. A buffer can be configured in two transmission modes; *event* and *state* mode. The different modes have the following specifications:

- *Event mode*: Only one transmission is started when the application has committed the message buffer.

- *State mode*: The frame is transmitted in every following cycle and if the message buffer is not updated a null frame will be transmitted.

Assume that an error will occur in a node for some unknown reason. When it is important to discover the non-working node, it can be done, by using the state mode in order to discover null-frames from a specific node. If there is a null-frame, a serious error has occurred. The same solution could be used to find internal soft- and hardware errors between the DSP and the FlexRay controller.

It is also possible to add filters to match the cycle-counter of a frame to see if the message is valid or not.

Now, assume that there is a calculation error (and that the reason to the error depends on some variables, which has got the wrong values because of a memory error). It is possible to find the erroneous node by comparing the results from other nodes. However, the question is how to handle the error. Some possibilities are:

- Isolate the node in a fail-safe state and prohibit the node from making calculations for other nodes.
- Using the dynamic messages to send over a memory area from a healthy node.
- A combination between both alternatives, which mean that the node will be isolated until all the variables are correct. Perhaps the node needs to be restarted as a last option.
- Restart the node and reset the internal states

It might be important to the system to know if something went wrong. Because of this, a three-step mode controller with for an example the colors green, orange and red can be used. The green color means that everything is working fine, the orange color means that something is wrong, but not critical. It could be that one of the nodes got another value as the results compared to the other nodes. If the red color mode is active it means that a common decision between the nodes could not decide what to do because of more then one error. In this case the system should probably inform the driver in some way.

If a node stays in the orange mode, the node can be seen as non-working and should be isolated. If a node gets isolated it is important to add redundancy. It is easy to forget that the isolated node actually has redundant calculations to make to another node. The question is how to solve this. One option is to let another node perform the calculations but if this is going to work, some kind of schedule over which node is backup-node is needed (see figure 9).

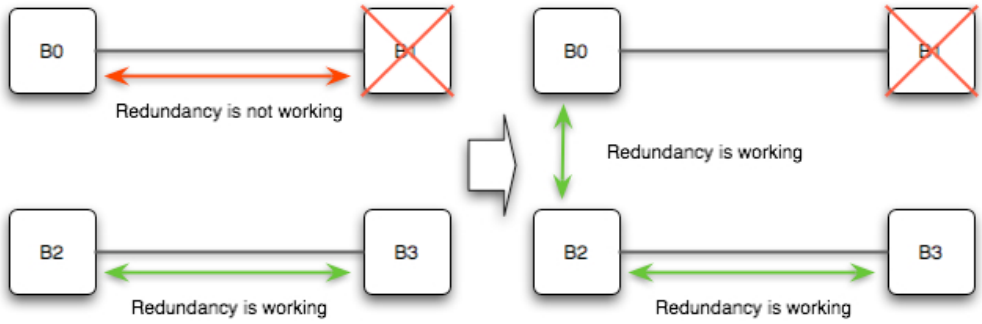


Figure 9 – *If node B1 is incorrect it can be isolated, node B2 acts as a backup-node.*

3.3.2 Alternatives for redundancy

It is very important to consider the safety-critical software modules in terms of redundancy. Some possibilities are:

- Each node compares its calculated values with another node.
- Each node compares its calculated values with two or more other nodes.
- A combination of statement 2 and 3, which means that each node compared its own calculated values with one other nodes, but if there is an error, other nodes are consulted.

If each node compare its results with other nodes, it is important to add a voting component that can make a majority decision.

In fact, there are mainly two kinds of tasks, which need redundancy. The first one is the slip-control, which is local to every node. For now, the slip-control can be forgotten because it does not involve the distributed system. The second one is the calculation of the dynamics of the vehicle. Before the dynamics of the vehicle were calculated in the two ECU:s. Because of their removal, the calculations need to be distributed between the nodes in the wheel-modules.

Because of this, the calculation of the dynamics of the vehicle will get an own network (the A channel). From this point, that network will be referred to as the *distributed network*. The B channel will be used for transporting sensor data from all the nodes to the nodes that calculate the dynamics of the vehicle. This network will be called *the axle-connections* or *the sensor data networks*.

3.4 Real time performance

The nodes in a FlexRay network are synchronized over the network. The design of a safe and redundant distributed brake control system requires software modules in each node to be synchronized with each other for best performance. The choice of software schedule and how to perform an optimal synchronization will be studied in this section.

It is important to remember that the static and the dynamic messages in FlexRay are actually based on different protocols (see section 2.1). The static part does not need any scheduling because of the schedule table. When it comes to the dynamic part it is important to let each node know what kind of "state" it should be in to receive the dynamic message – synchronization in some form is needed!

The same discussion can be applied on the alternative where one FlexRay network is divided into two smaller ones. As mentioned in section 2.1 it is possible to use cycle multiplexing, which means that for example every 10^{th} cycle differs from the others (see figure 10). This provides a possibility to add synchronization messages between the networks using the connection over the axles. Another way is to use the dynamic messages or perhaps a combination.

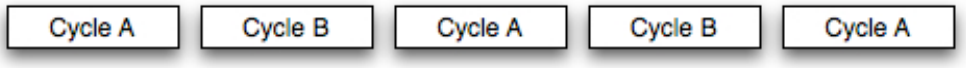


Figure 10 – *Shows an example of two different cycles (A and B), which consists of different settings for the communication.*

3.4.1 Alternatives for synchronization

The following alternatives are possible for synchronizing the nodes:

- Using a message-header with message/slot-id.
- Using the current cycle-counter in the FlexRay controller in combination with the message buffer settings.
- A combination between the first and second statements.
- Using cycle multiplexing.

3.5 Examples of topologies of the FlexRay-network

The possible topologies defined in the standard are:

- Bus
- Star
- Hybrid of both bus and star

The following examples shows a model of a three-axle vehicle but the examples can be reduced or expanded to two - five axles.

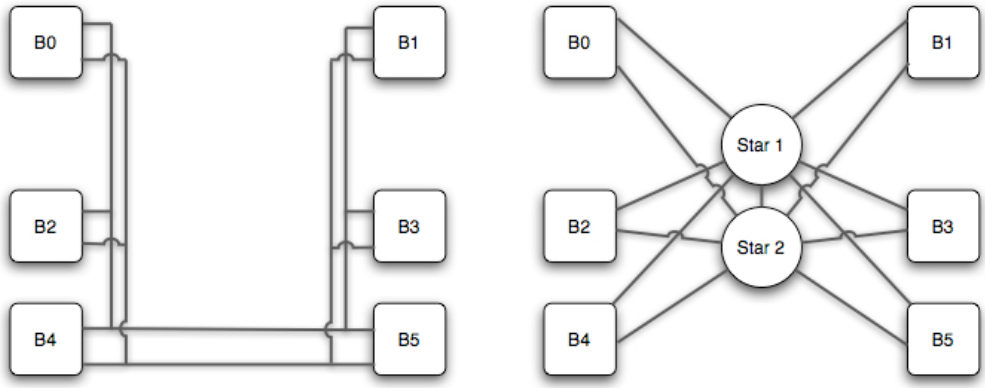


Figure 11 – *The left picture illustrate six wheel modules connected by a bus. Notice that each channel has its own bus to increase redundancy. The right picture shows the nodes connected to two star couplers.*

The examples in figure 11 are not optimized in terms of bandwidth or cost efficiency, but show two ways to connect the nodes. As a result of the TDMA protocol there is a relation between the number of nodes in a single network and the quantity of data that fits in one communication cycle.

```
nbrOfSlotsPerNodeInAChannel = nbrOfSlotPerChannel/nbrOfNodes        (3)
```

Because of this the left picture in figure 11 with all six nodes on the same network is a bad solution in terms of how many slots each node can use since `nbrOfSlotsPerNodeInAChannel` should be as large as possible to maximize the amount of data each node can send in a communication cycle. In this case, it would be better to split the network into two smaller networks with three nodes each (see figure 12). It is also important to remember that one problem with splitting the nets into smaller ones is that synchronization between the networks is needed - more about this in the next section.

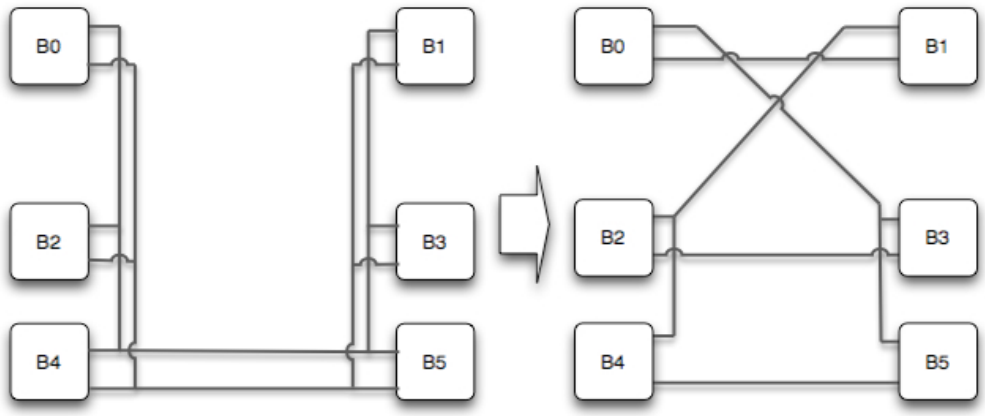


Figure 12 – *Shows a better solution in term of data transmission then the left picture in figure 11.*

The networks could be connected to each other by using the other channel to connect wheel modules over the axles (see figure 12). The reason to the cross link between the sides is to get brake force to each side of the vehicle in case of an error in one of the networks.

In the figure the wheels are connected to each other over the axles, but there is nothing that stops the axle-connections from being connected to each other. In section 2.2 the configuration of the FlexRay-network was introduced and earlier in this section the variable `nbrOfSlotsPerNodeInAChannel` was known as the number of slots each node has in the cycle. Now, assume that a point-to-point connection is made over the axles as the example in figure 12. Then each node can send a huge amount of data in a single cycle and perhaps this is not the most efficient way of using the medium because there is no need for this large amount of data. Because of this it might even be a very good solution to connect some of the axles to each other as they create a separate network with 6 nodes (see figure 13). It also makes the comparison between results from different nodes easier.

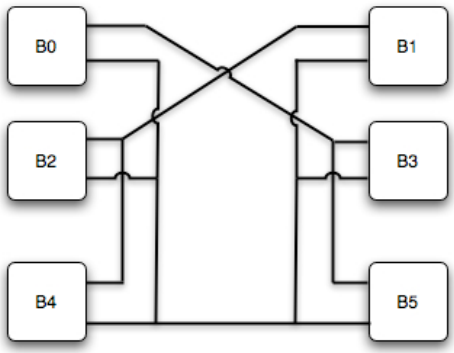


Figure 13 – *In this case all the axles are connected together but in most cases networks with 4 nodes will be more common. It might also be safer to use a star coupler instead of the bus.*

In figure 14 the inputs to each star coupler has been reduced from 7 to 3 and the biggest reason why to do such thing is to minimize the cost of the hardware. The more inputs to the star coupler – the higher cost. However, instead one gets a fault tolerant system with higher redundancy. One problem with the star couple is the increased number of nodes on the network since there is no division of the network. But if a single link between the star coupler and a node gets destroyed it can be isolated and the traffic can choose the other way to its destination. In some cases it might be a good idea to take the star couplers in consideration for example between the axle-connections, but it is all a question of cost corresponding to the performance.

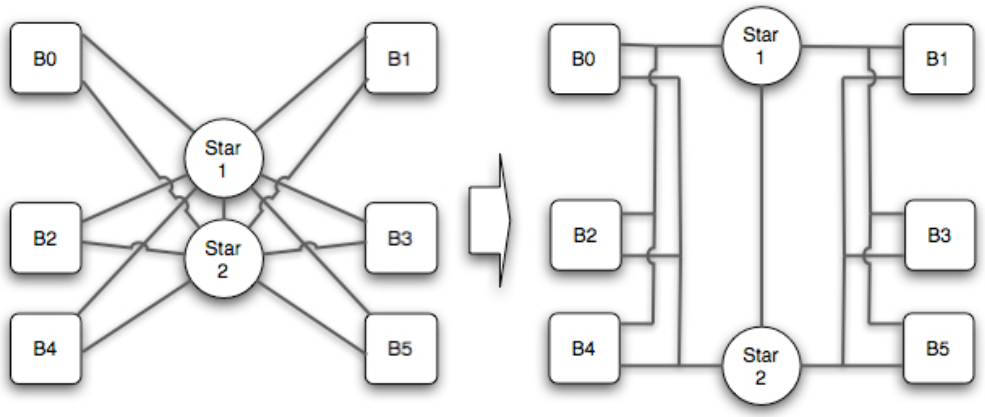


Figure 14 – *Shows a more cost efficient solution of the right picture in figure 7. However, it is still expensive.*

3.6 Summary

Now, lets go back to section 3.1. In the table the number of messages per brake module and unit of time was specified as follows:

Unit of time	*Number of messages* per brake module and unit*	
2.5 ms	2	(Send sensor data, calculate slip-control)
10.0 ms	2	(Update the dynamics of the vehicle)

*) One message is equivalent to one FlexRay-slot with 16 words including the header. 1 word is 16 bits.

In section 3.2 and 3.3 it was discussed which of the nodes should make the calculations and how to add redundancy to the calculations. The stated alternatives were that one to all nodes make the calculations. Then one to several nodes compare the results so that one master node can make the correct decision. It was also known that the FlexRay controller has two modes; *Event* and *State* mode. In Event mode the transmission is only started when the application has committed the message buffer. In the State mode the frame is transmitted in every following cycle and if the message buffer is not updated a null frame will be transmitted.

Later in section 3.4, synchronization of the nodes was discussed. It was known that it is possible to use cycle multiplexing to have different configurations of the cycles. Other methods were to add a message-header that tells the receiver what kind of message the sending node is sending or using the cycle counter in the FlexRay-controller in combination with the message buffer settings.

At last, in section 3.5 the different topologies were discussed. The relation between the number of nodes and the number of slots each node can use in a cycle was shown.

Starting from this, a few design alternatives based on the best alternatives of this chapter can be analyzed. This is what the next chapter is all about.

4 Analysis of the design alternatives

4.1 Preliminaries

Starting from the beginning with the distributed control system, two alternatives of which nodes should make the calculations, was introduced. The alternatives were that all nodes make the calculations compared to that only some of the nodes make the calculations. In this case, the alternative where only some of the nodes make the calculations will be used. The reason why this alternative is chosen was that the system would be the same on all vehicles irrespective of how many nodes the vehicle has got. The common denominator is in this case 4 nodes because there are no vehicles with fewer wheels.

Now, assume that some node has an error. The easiest way to discover the non-working node is to use the *state mode* in the FlexRay-controller because of the null-frame that the other nodes can discover. If it is memory error the results of the calculations can be wrong. In this case, redundancy is needed. Earlier it was discussed how many nodes should contribute in the redundancy check. The alternatives stretched from one to several nodes. In the distributed case, it is 4 working nodes and because of this there is no meaning to exclude some of the nodes. But it might be interesting to investigate if there are any advantages to compare the result with one other node, and if there is an error, other nodes are consulted instead of letting every node take part in the redundancy check.

To be able to determine a common result out of the different individual results from the nodes it is important to add a voting element as was stated in section 3.3.2 that can give an output based in the results from the other nodes.

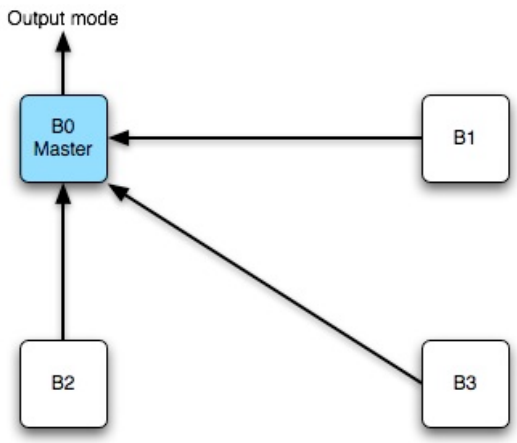


Figure 15 – *The master decides what to do based on the results from the other nodes. In the next cycle B1 will be the master, then B2, B3, B0....*

In fact, this voting component might be one of the nodes itself, but the role should maybe be changed over time – perhaps through a token so each node take care of the voting and then gives the token to the next one. This means that the node with the token acts as a master at a given time. (See figure 15)

The question is what is going to happen if one of the nodes goes down. In section 3.3.2 a system where another node takes over the redundancy was shown. In this case then it is about the distributed nodes, perhaps it does not matter if a node is down as long the other gives correct results. The system must be adaptive so the other nodes know which node is down, otherwise the system will halt before the node that is down becomes master. In this case the node will be isolated and the mode will go from green to orange. If there is another error (in the nodes) the status should probably be changed to red.

When it comes to how the channels should be designed it is almost impossible to tell without further analysis. The two factors to take into account is the number of slots each node gets in each cycle at a certain network and how robust the network is to errors of different kinds. Because of this different suggested solution alternatives will be introduced. One or several of these alternatives will also be analyzed using the TrueTime-library using simulink and Matlab.

Note that the maximum number of nodes is 10. Because of this, 10 nodes will be used in the analysis because if it works with 10 nodes it will also work with any design down to 4 nodes.

4.2 Suggested designs alternatives

Three different designs will be proposed where the difference between the designs is how the networks are configured; not only how the channels are used but also how the redundancy between the nodes is solved. The second and third design alternatives are almost the same. Because of this, some parts of the analysis will be equivalent.

4.2.1 Suggested design number 1

In this design the bus topology will be used. There will only be two networks, one on each channel and as stated in section 3.3.2 the A channel is used for the distributed nodes and the B channel is used for the sensor data (see figure 16). From the specification it is known that each node need to send at least 2 messages per unit of time. In order to add redundancy to the distributed nodes the example with the master node will be used.

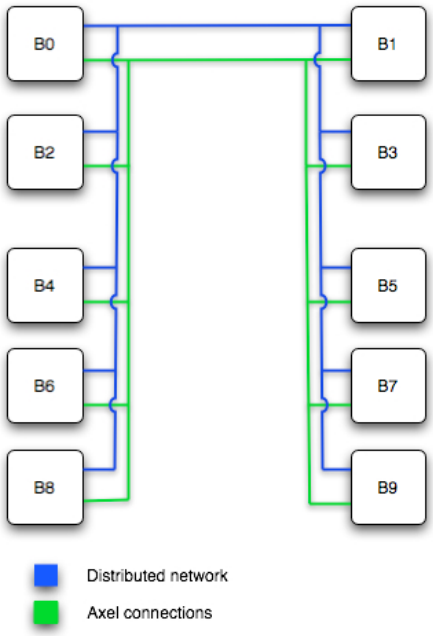


Figure 16 – *Shows the design of the network, which is divided into two networks, one for each channel.*

4.2.2 Suggested design number 2

Also in this design the bus topology will be used. As opposed to solution (number) 1 there will be 4 different networks instead of 2 (see figure 17). This means that there will be 2 networks for the distributed nodes and 2 for the sensor data. The requirements are the same here, which means that each node need to send at least 2 messages per unit of time. In order to add redundancy to the distributed nodes the example with the master node will be used.

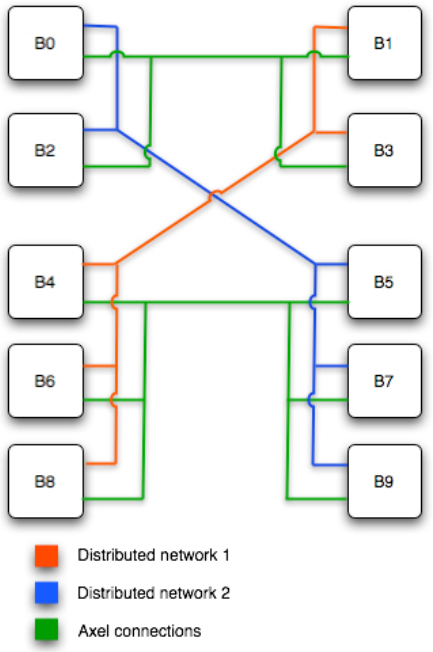


Figure 17 – *Shows the design of the network divided into two smaller ones with 5 nodes each (the red and blue lines). The green wires are axle-connections.*

4.2.3 Suggested design number 3

This design is almost the same as (number) 2. The only thing that differs is the way the voting by the master is performed. Instead of using the results from all the distributed nodes the master only gets the result from one other node. The compartment is then between the masters result and the result of the other node. In case of a mismatch, the other nodes are consulted.

4.3 Analysis of the slots in the FlexRay-cycles

The formulas used for calculating the number of slots per node in a channel was stated in an earlier section. The two different unit of times which will be used are 2.5 ms and 10 ms. Unfortunately, from section 2.4, it is known that it is not possible to use different cycle times on the different channels using the same FlexRay-controller.

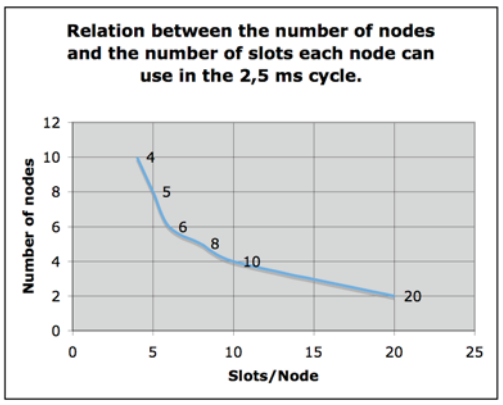
`gdCycle` = 2,5 ms

`gdStaticSlot` = 60[1] µs

The number of slots per channel in the 2,5 ms cycle is 40. It might be interesting to see how the number of slots each node gets depends on the number of nodes.

The 2,5 ms cycle

# of Nodes	Slot/Node
2	20
4	10
5	8
6	6
8	5
10	4



[1] The value is based on the fact that we want to use 10 Mbit data rates on each channel and send 16 words of data in each slot. In fact, other values can be used.

This is interesting to use as a reference to the other alternatives and see how much more data each node is possible to send in a cycle. The more data it is, the more extra functionality as fail-safe routines can be added.

The following table shows the number of slots each node can use in the different networks if the cycle time is 2,5 ms:

Design 1		**Design 2 & 3**	
Blue network:	4 slots/node	Blue network:	8 slots/node
		Red network:	8 slots/node
Green network:	4 slots/node	Green network:	10 and 6 slots/node

It is now known that the requirements of sending 2 messages at a unit of time are fulfilled in all designs! If the unit (10 ms) is used the number of slots per node can be multiplied by 4 (10 ms is 4 cycles of the 2,5 ms cycle). This is the fact in the blue and the red networks, which are used for the vehicle control software. As a result of this the table can be updated:

Design 1		**Design 2 & 3**	
Blue network:	16 slots/node	Blue network:	32 slots/node
		Red network:	32 slots/node
Green network:	4 slots/node	Green network:	10 and 6 slots/node

This means, that each node at least can send 4 messages per unit of time and the requirements from the specification are reached!

It might be suitable with a clarification. The distributed nodes that calculate the dynamics of the vehicle get all their data for calculations from the other nodes and from the driver through the reference value. Each node sends the values from the sensors to the network and the other nodes that need it can fetch it. This is not a problem in design 1 but it gets enormous in design 2 & 3 because of the fact that the nodes are not on the same network! In this case we will need the free space on the networks to send the data. This will be simulated in TrueTime later on.

The question is how to use the different slots in the cycle, and perhaps more important, how should the static and dynamic part of the cycle be used? This is the topic in the next section.

4.4 Configuration of the FlexRay-cycle

Now that it is known how many slots each node can use on each channel it is important to find a solution that optimizes the use of the cycle. In section 2.1 the cycle was described as a number of slots in static and a dynamic part.

In section 3.3 the definition of which data that is safety critical was stated as *all common decisions concerning the vehicle control software and the slip-control of the wheels can be seen as high safety-critical.* It means that this data always need to be carried on the medium in every cycle as the system is running. Therefore, this data will always be in the static part of the cycle. Other information that is not so frequently sent can be sent in the dynamic part of the cycle. In order to make the network less complex it might be a good solution to use the dynamic part as ”half static”. This means it has its own slot, but the data is dynamic. In this way the nodes can exchange different kind of data without changing the rules of the cycle since the static slots will be the same all the time. The kind of the message is determined by some message-header. It is also possible to use cycle multiplexing to alter the different cycles. If the node does not have anything to send, an empty struct is sent. (See figure 18)

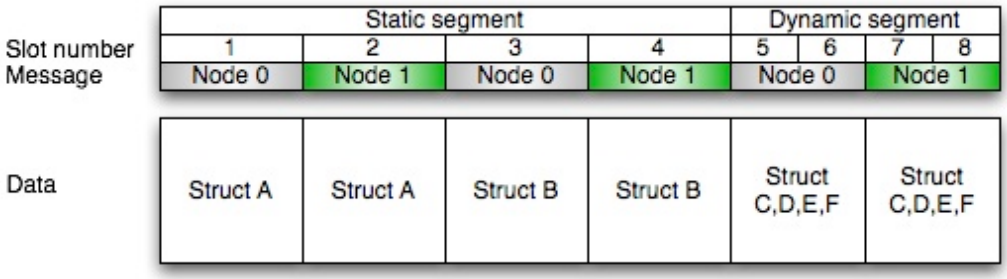


Figure 18 – *In the dynamic part different kind of messages can be sent.*

In some cases this might be inefficient because every node takes up a slot in the dynamic segment. In these cases, it could be better to let every node send whenever it wants to.

4.4.1 Configuration of design 1

In this alternative there are two networks, one on each channel. The A channel is used for the distributed software and the B channel is used for transporting sensor data to the nodes that handle the dynamics of the vehicle (see figure 19).

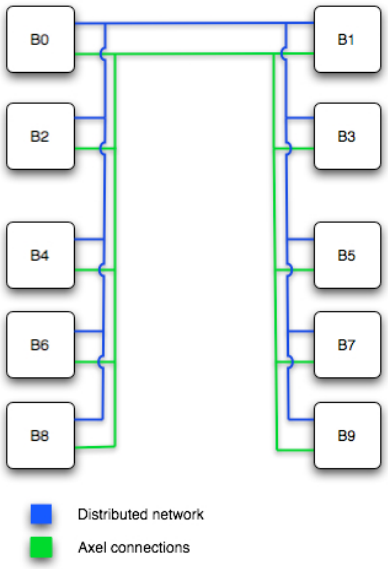


Figure 19 – *Shows the design of the network, which is divided into two networks, one for each channel.*

In the network on channel B all nodes are connected and because of the number of nodes each node has 4 slots in each cycle of 2.5 ms (see figure 20 and 21).

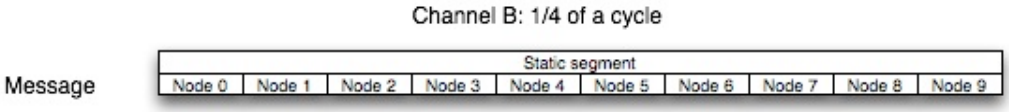


Figure 20 – *¼ of the cycle is shown. Since everything is periodic the other parts of the cycle are the same.*

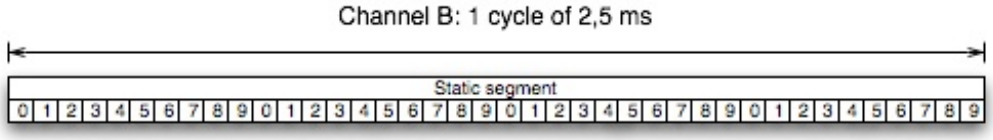


Figure 21 – *All the slots are shown.*

Each node sends the sensor data to everyone but itself. However, only the 4 nodes, which calculate the dynamics of the vehicle, actually take care of the messages. The nodes do not need to send the same sensor data in all the 4 slots, because only the last part of the cycle can be used (see figure 22).

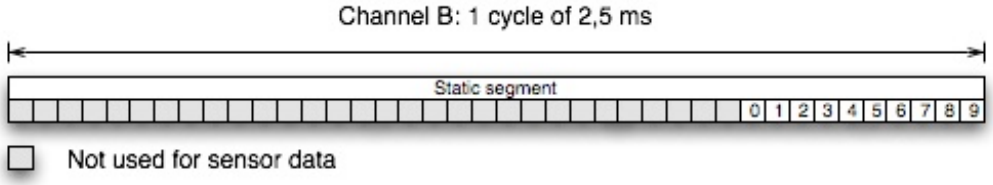


Figure 22 – *Only the last part of the cycle is used*

When it comes to the network on channel A, it has the same amount of nodes but the schedule is a little different. Only 4 nodes should calculate the dynamics of the vehicle (see figure 23) and the other nodes should be non-operating. Note that the non-operating nodes dose has their own slots but they are not sending anything. This can be seen in figure 24 and 25.

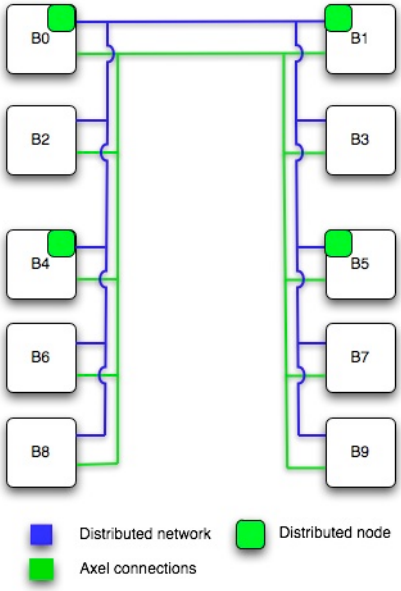


Figure 23 – *4 nodes take care about the calculations of the dynamics of the vehicle.*

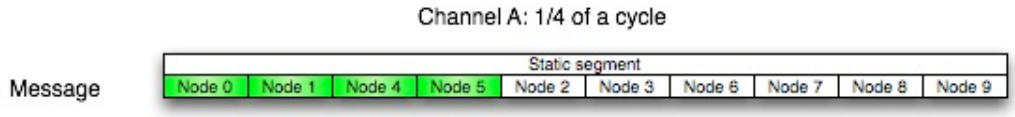


Figure 24 – *¼ of the cycle is shown. Only the green slots sends to add redundancy. Note that slot 2 & 3 has changed place with node 4 & 5 compared to the order in channel B. This is to get more time for the compartments in the master node.*

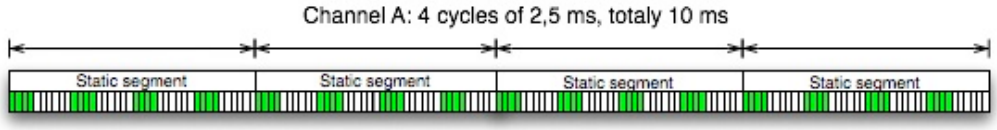


Figure 25 – *All the slots are shown.*

Now, the traffic on channel A is the comparison of the results of the dynamics of the vehicle. From the beginning of chapter 3 the idea about the master-node that acts as a voting component can be applied. One solution is to use the inputs of all 4 nodes (including the masters result) to the master and then let the master decide the next mode. The problem with that solution is that it is a bad idea to have an even number of inputs because there might be a situation where the voting result is even.

Instead it is better to exclude the master's result in each cycle and only compare the three results from the other nodes (see figure 26). This is called a Triple Modular Redundancy (TMR) [10].

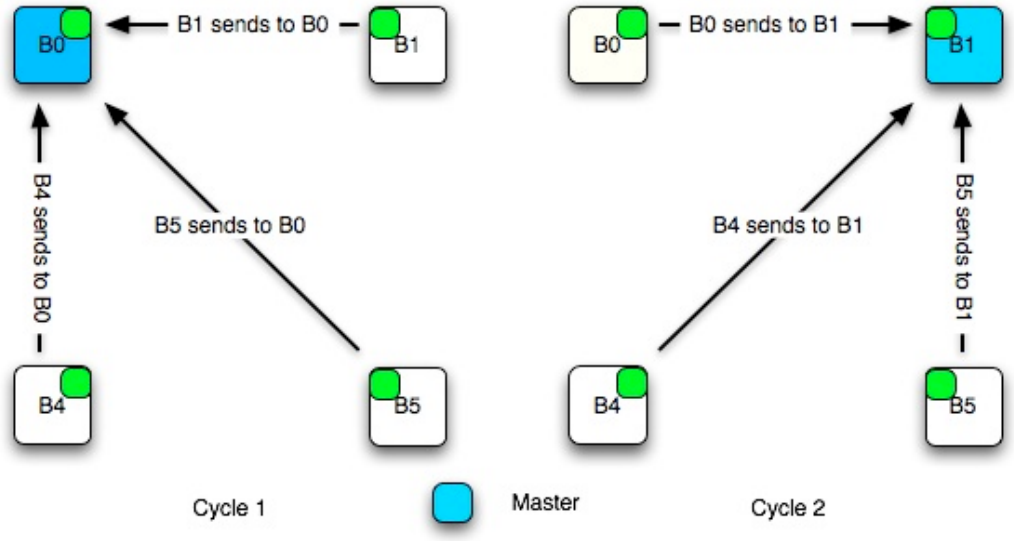


Figure 26 – *All the nodes send their results to the master in each cycle. The master-token is changed every time there is a new cycle. In the picture 2 different cycles are shown.*

But, if a node goes down, then there are only three working nodes and the voting will not work because the result can be even. In this case the master node's result must be included in the decision.

The dynamics of the vehicle is updated every 10th ms, which means that each node has 4 slots in 4 different cycles. There is no meaning of sending the same data in all 4 cycles; instead the cycle multiplexing can be used so only one cycle is used (see figure 27).

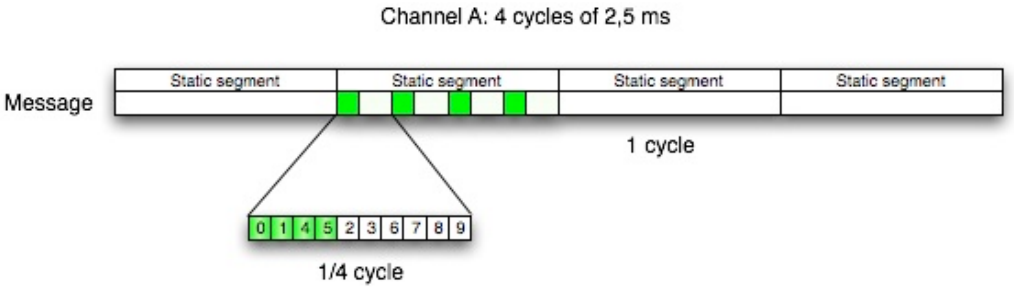


Figure 27 – *Which cycle should be used depends of the schedule of the different tasks in all nodes.*

The attentive reader realizes that it is pointless to send 4 equal results in every cycle of 2,5 ms to the master node. The question is how this should be solved. It is not possible to add multiplexing inside each cycle because the order of the

sending nodes is settled by the schedule. However, it is up to the developer to add a message-header (to the message) that tells the receiver what was sent. In this way it is possible to send 4 different messages in each cycle!

By this time it is well known that the dynamics of the vehicle are updated every 10th ms and that new sensor data is available every 2,5 ms. This means that the sensor data will be sent to the nodes that calculates the dynamic of the vehicle 4 times in every period of 10 ms. In fact, the data is only needed once. This can be handled by using a message-header or cycle multiplexing see figure 28.

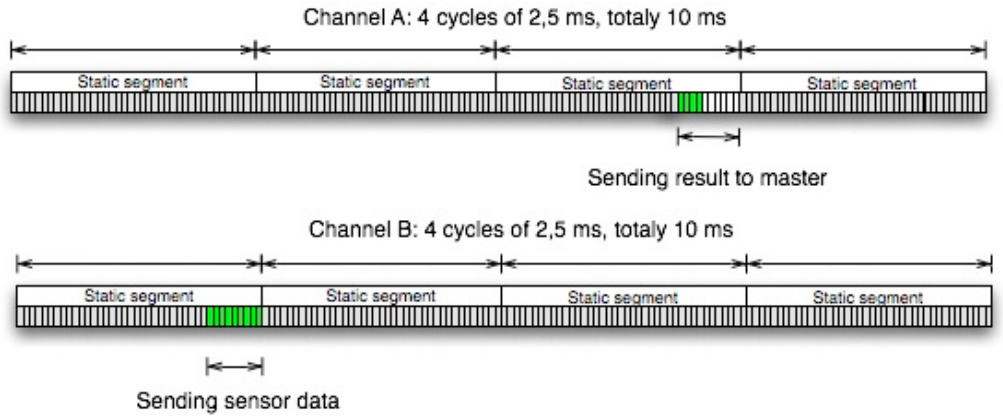


Figure 28 – *Using a message- header and cycle multiplexing the data can be sent just once.*

(All these optimizations concerning that only some of the slots and cycles are used will of course also be true in the other design alternatives later on. But it is not sure that it will be mentioned all the time.)

There is one important thing left. After the master node has made a decision of the next node, the other nodes need to be informed. This message must be sent every 10th ms in the same cycle where the mode was settled. The message can be sent in either the static or dynamic part. The advantage with sending the message in the dynamic part is that if the decision was made first in the last part of the fourth cycle of 2,5 ms, there is still a possibility for the master to send the message in time (see figure 29).

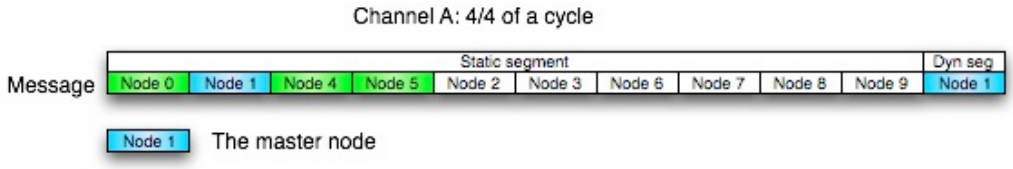


Figure 29 – *The last cycle in the 10 ms cycle is shown. Node 1 is the master and because of this the master needs to send out the decision to the other nodes.*

But this should not be a problem since the system should be completely deterministic. Because of this it might be easier to use the solution with the message-header instead. Another argument that support not to use the dynamic part is that the bus guardian only supports static segments. And the decision of the next mode is very critical which means that the static segment *must* be used!

Conclusions of design 1

In figure 28 it could be seen that there is a lot of free slots left in both channels.

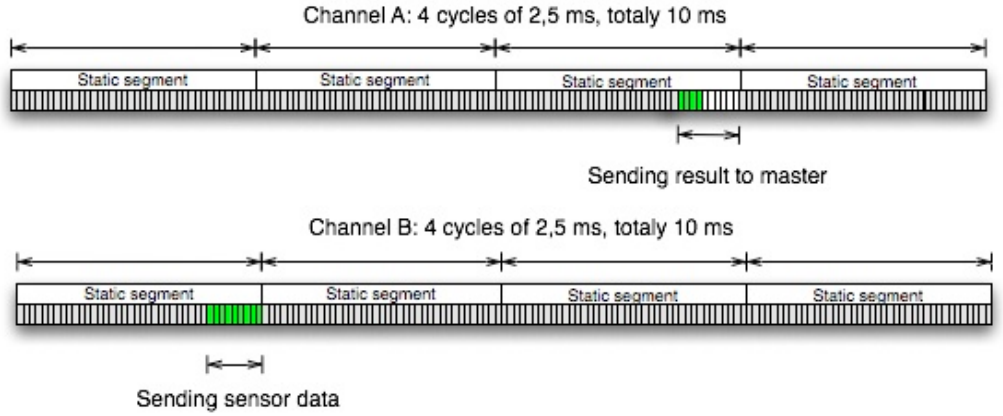


Figure 28 – *Using a message- header and cycle multiplexing the data can be sent just once.*

This means that it is a lot of space free to other tasks. An example of another task that needs access to the network is the slip-control of the wheels. It has not been included in the discussion above because it does not affect the discussion about distributed system control.

4.4.2 Configuration of design 2 & 3

Because of the similarity between solution 2 and 3 they will be analyzed under the same section. In this case we have 4 different networks, 2 for the axle-communication and 2 for the distributed software. The theory where each node sends out sensor data to the network that the distributed nodes fetch is used here too. The problem is that the nodes no longer are on the same networks and that the distributed nodes need the sensor data from all the other nodes (see figure 30).

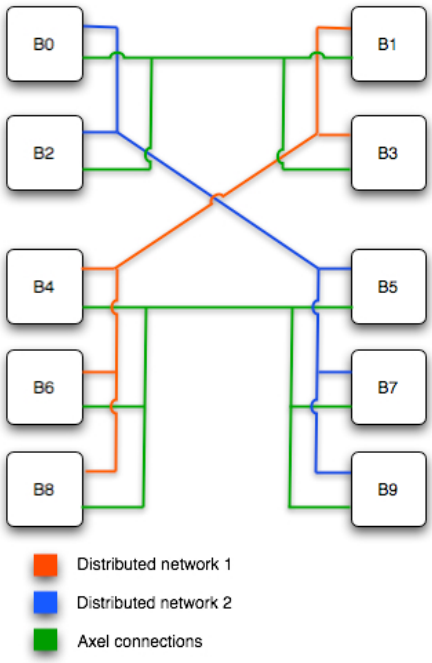


Figure 30 – *Shows the design of the network divided into two smaller ones with 5 nodes each (the red and blue lines). The green wires are axle-connections.*

As a result of this, the networks used for the distributed nodes need to carry the sensor data to the nodes in the other networks. Now, to make it a little easier to follow, we use the nodes with the distributed software to act as an interface to the other networks (see figure 31).

For example, when node B0 has all data from the nodes in the front axle-network it sends the data to node B5. The same is valid for node B1, which sends the data to node B4.

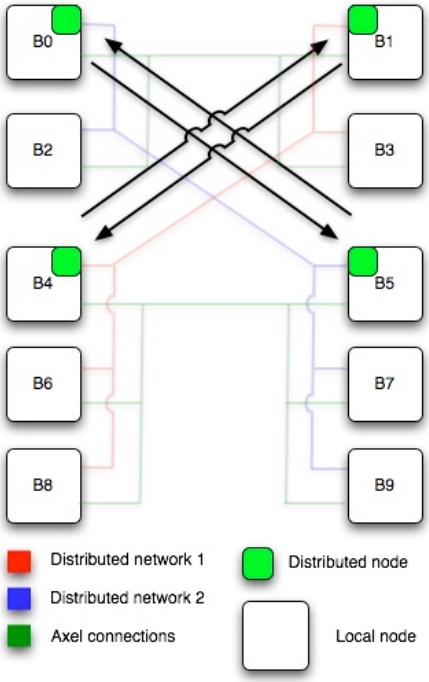


Figure 31 – *Shows how the sensor data is distributed through the network to the vehicle control nodes.*

It is obvious that node B0 and B1 also needs the data from the back axle-network. The same theory is applied here; B4 sends the data to B1 and B5 sends the data to B0.

One problem with sending all the sensor data from one node to another is that the data from the nodes does not fit in one slot. Every slot consists of 1 message, which is the same as 16 words. In this case we assume that each slot is filled with sensor data to be able to get some maximum value of slots needed in the worst-case scenario. This means that node B0 and B1 needs 4 slots to send the data from the front axle-nodes and B4 and B5 need 6 slots to send the data from the back axle-nodes.

It is a fact that the distributed networks first of all are used for the compartments of the data. This means that every distributed node at least needs to send one message containing the results. Because of the number of nodes in each network, each node can send 8 messages per cycle.

Now, as the comparisons of the results are highly safety-critical they must be sent in the static segment (see figure 32).

Figure 32 – *Shows the different cycles used for the distributed nodes. Note that the order of the slots is changed so the sending nodes are in the beginning of the cycle.*

There is no meaning of sending the results from the calculations in every cycle because there will be 4 cycles in every 10^{th} ms. Actually, it will be enough if the results are sent in the last of the 4 cycles. This means that we can use the other 3 cycles to transport sensor data to the distributed nodes (see figure 33).

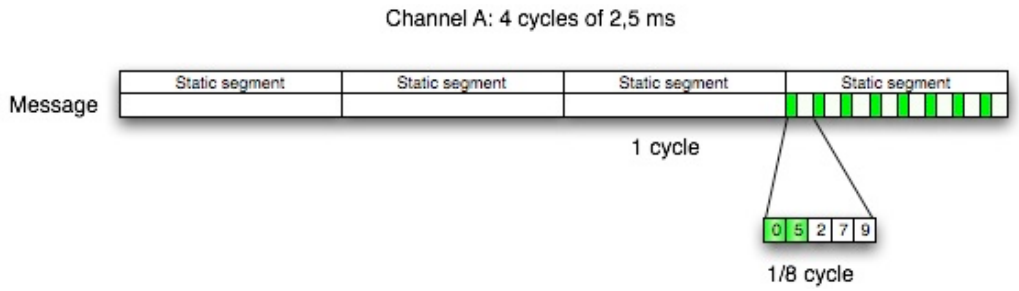


Figure 33 – *The results of the calculations of the dynamic of the vehicle is sent in the last cycle of 2,5 ms. The other three cycles can be used to transport sensor data or something else.*

Actually the other cycles look the same but the data that is sent is different. Because of this there is no meaning of using cycle multiplexing. It is better to use a message-header to inform the receiver what kind if message it is.

The front axle-network has 4 nodes, which means that each node has 10 slots in each cycle. In the back axle-network there are 6 nodes. The number of slots each node can use in a cycle is 6 (see figure 34). The network is only a carrier for sensor data and all cycles will be the same so we do not need to use the dynamic part here.

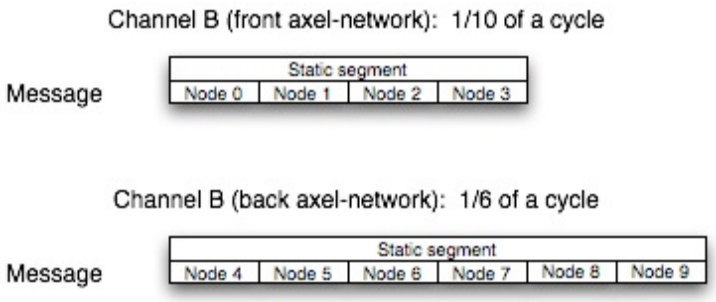


Figure 34 – *Shows the configuration of the cycles on the B channels.*

Starting from this, the nodes used for calculation of the dynamics of the vehicle have all the sensor data from all nodes. Here ends the common part of the section for solution 2 and 3.

Adjustments for design 2

In this design each node should compare the result with other nodes in the same way as in solution 1. Lets again take a look at the picture, which describes how the messages are sent to the master node (see figure 35).

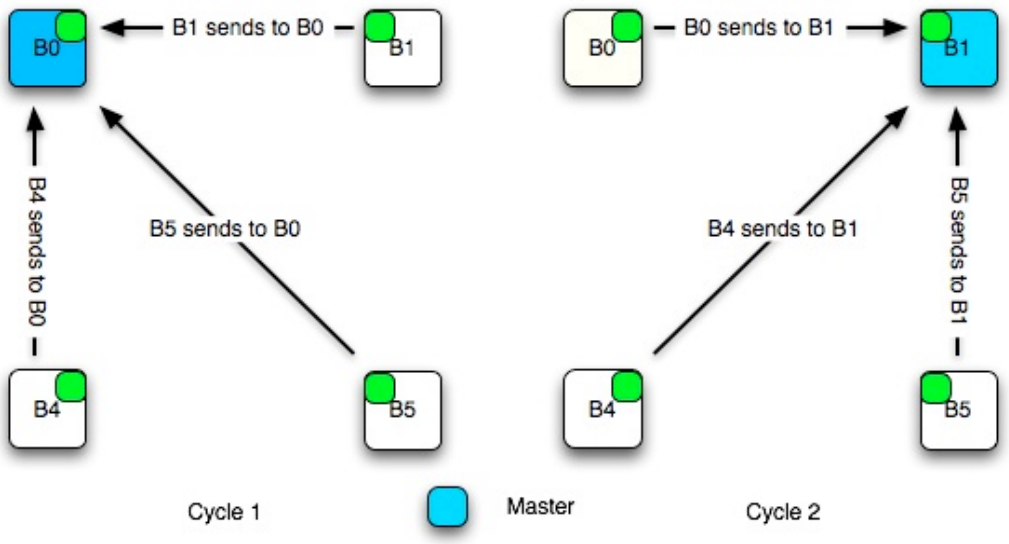


Figure 35 – *All the nodes send their results to the master in each cycle. The master-token is exchanged every time there is a new cycle. In the picture, two different cycles are shown.*

Here we can see a problem right away. The only connection in the distributed networks is the link between node B0 and B5 and B1 and B4. To be able to get all the nodes results to the master the axle-connections must be used. But this does not solve the problem completely because there is still one node that cannot send the result to the master in every cycle. In figure 36 it can be seen that node B4 cannot send its result directly to node B0. Instead node B4 must send the message to B5 first.

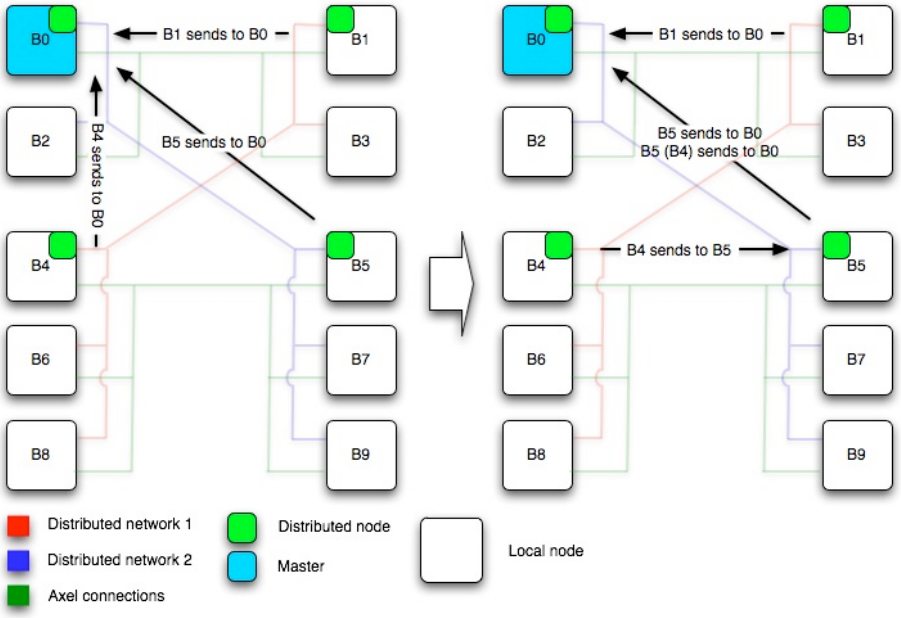


Figure 36 – *Node B4 must send its result to node B5 so that the massage can be transported to the master, which in this case is node B0.*

Now, the message from node B4 to node B5 is not needed in every cycle of 2,5 ms. Because of this, the dynamic part of the cycle might be used. But for the same reasons as stated in section 4.4.1 about the bus guardian and that the common decisions is safety-critical the static part should be used instead.

This can be solved in many different ways. One way is to use a message-header that tells the receiver what kind of message is sent. Another way is to add more slots to the node that need to send data of a different kind. Both these alternatives will be analyzed, starting with the alternative where more slots are added, or in fact, changed.

This means that node B0, B1, B4 and B5 need more slots in every cycle. Figure 37 shows how the B channel can be expanded. Because of the new slots, the other nodes cannot send as many messages anymore. This will not be an issue – there is still plenty of free space on the network.

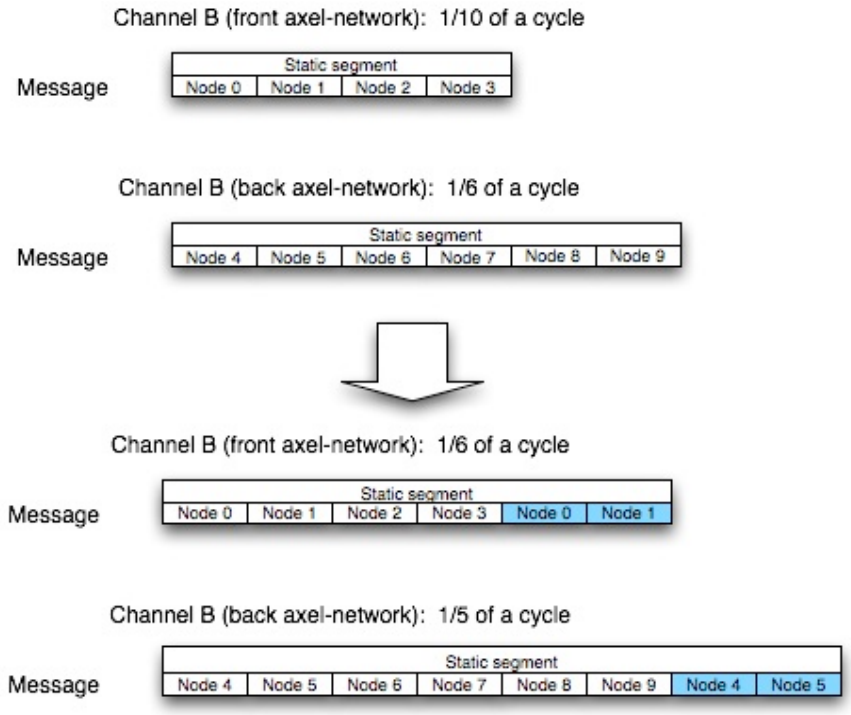


Figure 37 – *Node B0, B1, B4 and B5 has now 6 respective 5 new slots.*

It will work, but a message-header must be used anyway because there is no meaning in sending the same data 5 or 6 times depending on the networks. Because of this it seems better only to use the message header.

From this point, all the distributed nodes can get the results from the others! This means that a common decision can be made. The only thing that needs to be done is to send out a message with the new mode to all the other nodes!

Unfortunately, this is not as simple as is suggested design number 1 where all nodes were on the same network. The master can always send the message to all nodes in the same axle-network. It can also send a message to all nodes on the same distributed network. The only nodes which, will not receive the message, are some of the nodes on the other distributed network (see figure 38). The green nodes get the message but the red do not get it.

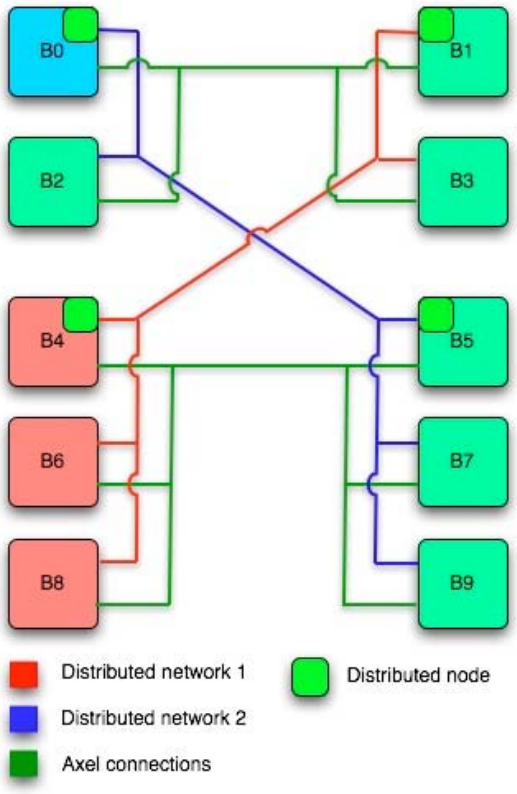


Figure 38 – *The green nodes got the message from the master and the red one did not.*

The easiest way to send the message to the red nodes is in this case to let node B1 repeat the message to all nodes in the distributed network. Another way is to let node B5 repeat the message to all nodes on the same axle-network. It will be done using the message-header that will tell the receivers that it is an important mode-message from the master.

Conclusions for design 2

The utilization of the networks is much higher compared to design alternative 1. This is because of the fact that the different networks need to shift a lot of data. The biggest advantage is that the networks are separated in a physical meaning, which improve the robustness of the system if a link should fall off.

There is still a lot of free space on the networks so even in this case it should not be any problems to add redundancy to the slip-control.

Adjustments for design 3

The only difference between this alternative and alternative 2 is that the voting procedure in the distributed networks is a little different. Instead of using the TMR-method the master uses its own result compared to the result from one other node (see figure 39).

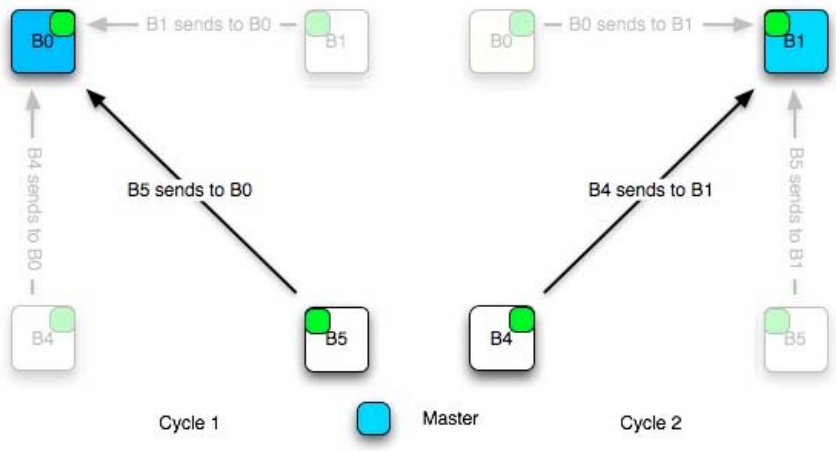


Figure 39 – *Only one node sends its result to the master in each cycle.*

In this way, the network does not need to exchange that much data through the different networks. But if the comparement does not correspond to each other, the results from the other nodes need to be sent to the master in the same way as in solution alternative 2 (see figure 40).

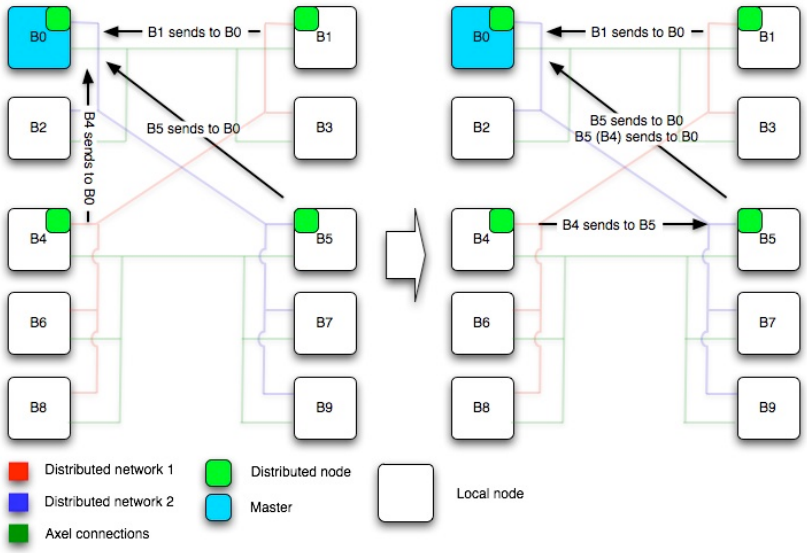


Figure 40 – *This is exactly the same solution as in solution alternative 2.*

Depending on how the tasks are scheduled, the results might arrive to the master in the next cycle. But when they arrive, the master needs to send out the result to all other nodes. The procedure is the same as in the solution alternative 2; the master sends the results to every node on both networks connected to the master. One of the other nodes in the other distributed network has to repeat the message to all nodes in that network.

Conclusions for design 3

The utilization of the networks is almost equal to design 2. One problem with the solution where the master checks it value with one other node is that the number of nodes included in the comparement is even. It means that there is a bigger risk that both nodes are wrong! One other thing to think about is that there is not that much extra space released on the network by using one node instead of all 3. The most loss in terms of space on the networks is when the sensor data needs to be sent to another network.

Another thing that has not been stated before is the fact that the nodes have some internal states that do not show up on the bus. This means that the states need to be checked every time there is a change to the original schedule, for example if there is an error. It is important that the same state can be retrieved again. Perhaps the easiest way is to reset the states?

4.5 Scheduling of the nodes

From the last section it is known how many slots are needed in each cycle to send all the data. The question is how the tasks in the nodes should be scheduled so that each sending node has all the data needed at certain points. To make this more general, the data that needs to be sent will be sent as late as possible in every cycle. The reason to do this is to maximize the time for calculations and other tasks in the nodes. The tasks will not be referred to anything but tasks, because there is no meaning in knowing whether a node is getting sensor data or calculating values at a certain time. Here it is up to the hardware to be fast enough.

4.5.1 Scheduling of design number 1

In figure 41 the schedule of the nodes is shown. The different blocks represent different tasks. (It can be a calculation, sending data or something else.) The activity on the different channels is represented by the white blocks, inside the channels, which are colored after the designs of the networks.

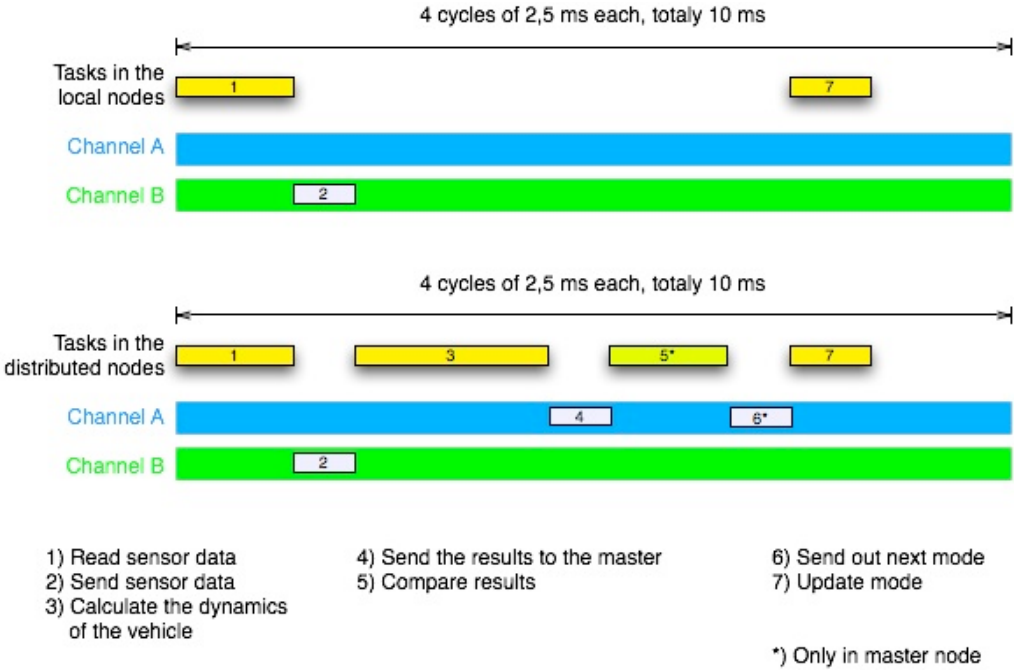


Figure 41 – *The different tasks for the local and the distributed nodes are shown. Note that the distributed and the local have common tasks to perform.*

4.5.2 Scheduling of design number 2 & 3

The scheduling of design number 2 and 3 is shown in figure 42. Due to the increased number of networks the scheduling is much more complex. The figure does not cover every step because it would make the graph much harder to follow.

The figure does not show what the scheduling will look like if there is a mismatch in the voting of design 3.

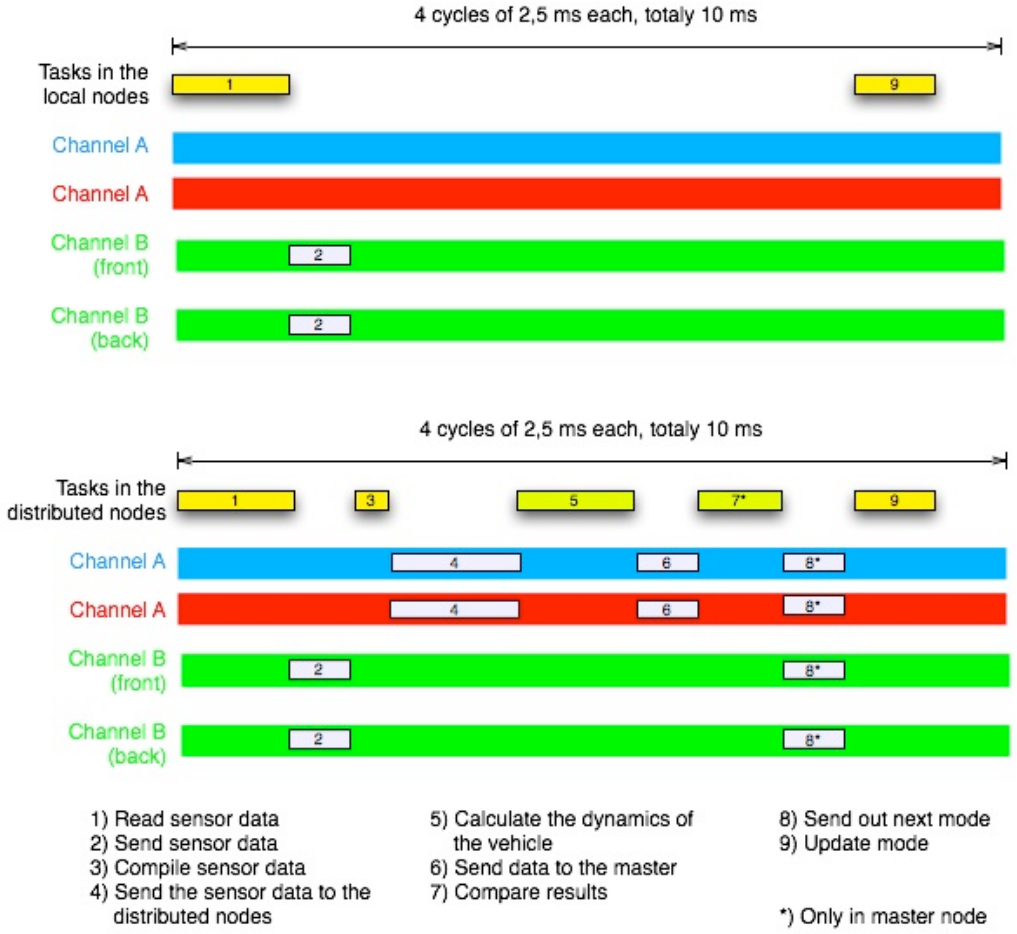


Figure 42 – *The different tasks for the local and the distributed nodes are shown. Also in this example the distributed and the local have common tasks to perform. Some part has been excluded, for example how the next mode is transmitted to all the nodes in all networks.*

4.6 Conclusions

In figure 41 and 42 it is clear that design 1 generates less traffic on the network compared to design 2 and 3. Another important thing is that in design 2 and 3 there are some cases, which will be true very rarely. This means that the system is not completely deterministic compared to design 1. One example of such case is if there is a voting error in design 3, the scheduling will look different. Because of this, it seems better to always use the results from 3 other nodes as an input to the master node.

When it comes to which network design should be used, it is important to remember that design 1,2 and 3, will all work! But, design 1 is much more simple. This is illustrated in figure 43!

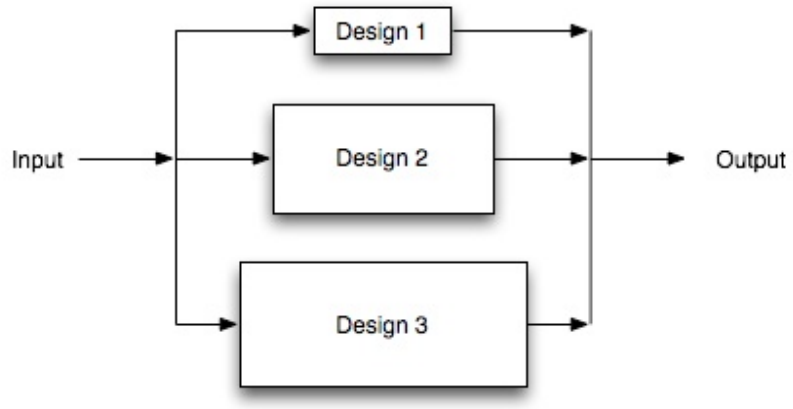


Figure 43 – *The systems generates the same output. The size of the boxes illustrates how complex the designs are.*

The advantage with using several networks, in this case 4 is to add redundancies to the physical links. If there is an error in one of the links in design 1, the whole channel will go down. From section 3.5 it is known that a star coupler could replace the bus to solve this.

As a result, design 2 and 3 are dismissed! There is no point in even trying to simulate the designs because the results will be the same as simulating design 1 which is much easier to implement.

5 Simulation results

From the last chapter it is known how the nodes should be scheduled and how the data is supposed to be sent in every cycle to make sure that the nodes can complete their tasks in time. The analyses of the suggested designs are very theoretical. Because of this, the purpose of this chapter is to show the results from the simulation of design 1.

The model in TrueTime reminds a lot of the real system. In figure 44 a model of a two-axle vehicle is shown. The blue boxes are the wheel-modules and the yellow boxes are the A and B channels. Each node has a power port so it is possible to simulate that a certain node goes down.

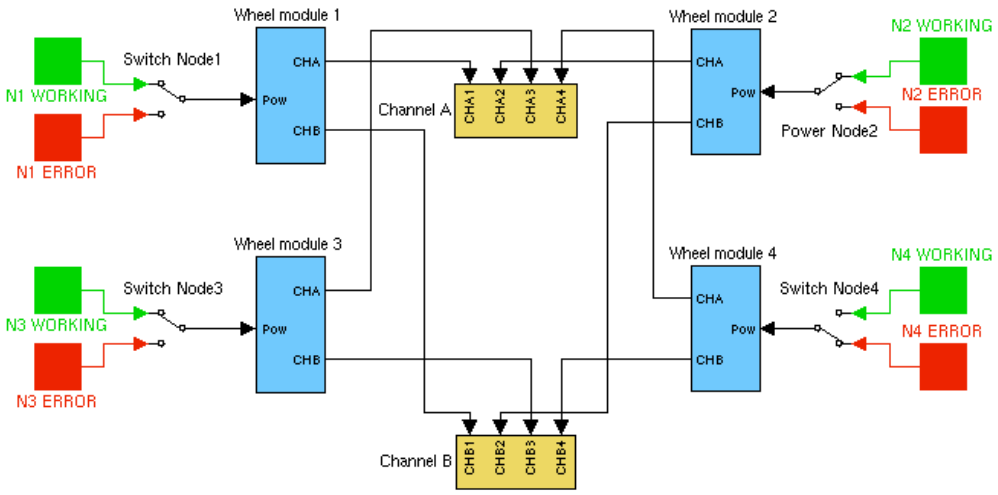


Figure 44 – *The model of a 2-axle vehicle where every node is connected to channel A and B. There is a power switch for every node, which makes it possible to simulate that a certain node goes down.*

To be able to evaluate the simulation and get some results the following will be simulated:

- A 10-wheeler vehicle that is in motion where 4 of the nodes calculate the dynamics of the vehicle using the solution with a master node.

- The nodes will be turned off, and the system should be able to handle one erroneous node at each time. If two nodes are turned off, the system will halt.

- If the distributed node acting as a master in some cycles gets disconnected, another node (of the distributed) will use its own result to vote a common result with the other nodes results.

The theory from section 4.4.1 will be used and as a summery, the most important parts are restated (this is also illustrated in figure 41 from section 4.5.1.):

- Each node sends its sensor data to the distributed nodes, using channel B.

- The distributed nodes calculate the dynamics of the vehicle and the result is sent to the master node, using channel A.

- The master node compares the results and sends out a common decision of the next mode to all the nodes, using channel A.

All details about the simulation and the model are found in the appendix A – The simulation in TrueTime.

5.1 Results

The main result of the simulation is that design 1 will work! All the nodes send the sensor data to the distributed nodes on channel B and the distributed nodes calculate the dynamics of the vehicle. The results are sent to the master that makes a common decision of the next mode.

By studying the plots of the transmission on both channels, the result can be shown. The following two plots show the system when there are no errors in the nodes (see figure 45 and 46). Each spike means that the node is sending data.

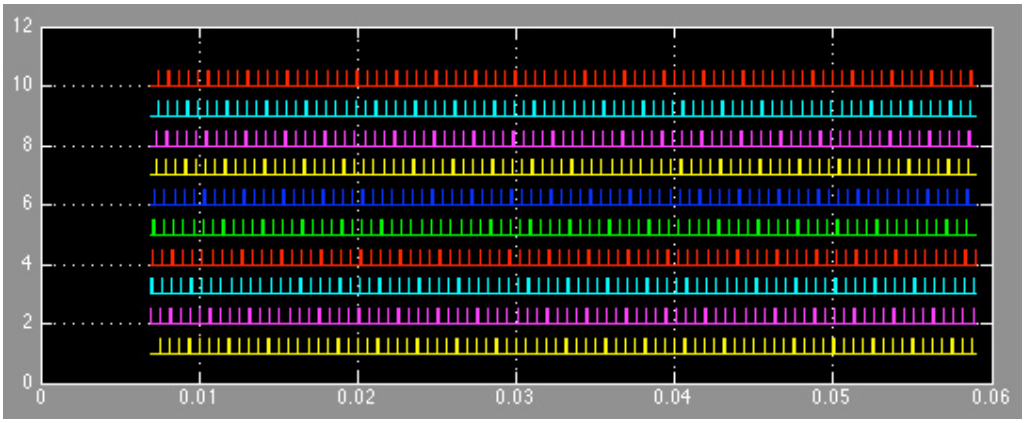


Figure 45 – *The transmission in channel B. All the spikes have the amplitude of 1which means that there are no collisions.*

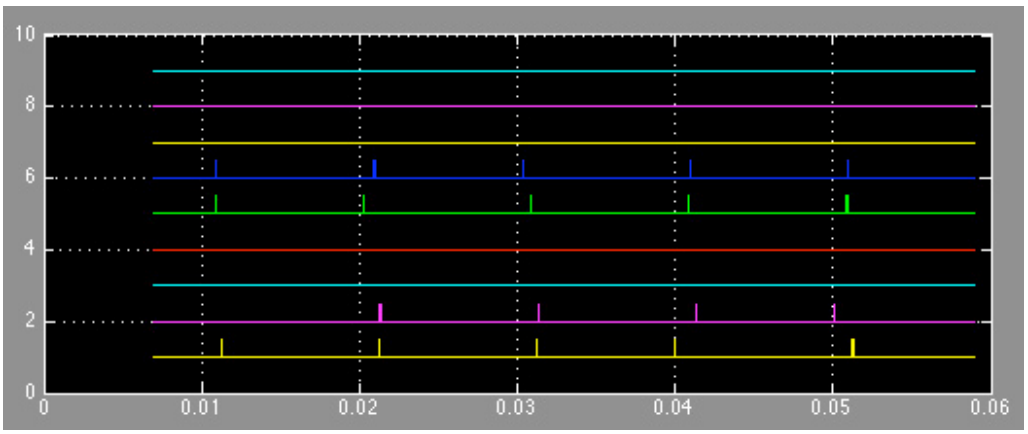


Figure 46 – *The transmission in channel A. All the spikes have the amplitude of 1which means that there are no collisions. Note that there are only 4 nodes that actually send data – this is the distributed ones.*

If there is an error, let say that a certain node goes down the traffic on channel A will be changed due to the null-frames that will be sent out every 2.5 ms (see figure 47).

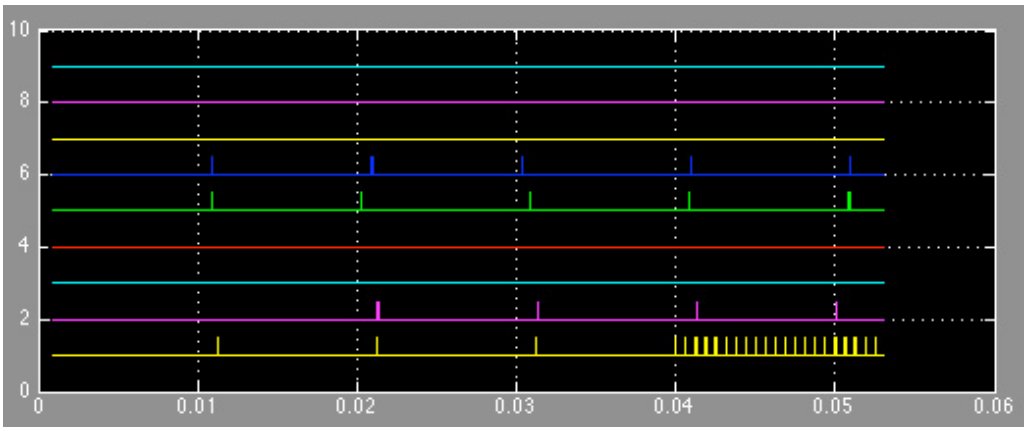


Figure 47 – *The transmission in channel A if there is an error, in this case node number 0 is off (the yellow spikes). As the plot shows, the yellow spikes are must more closed together because of the fact that the FlexRay-controller is sending out null-frames.*

But still, the system is 100% working!

The simulation does also have an animation of a vehicle in motion. This is to show how the master-role is changed and what will happen if a node is disconnected. The system can reconfigure the network after an error if the node gets connected again, which means that all four nodes will participate in the calculations (see figure 48).

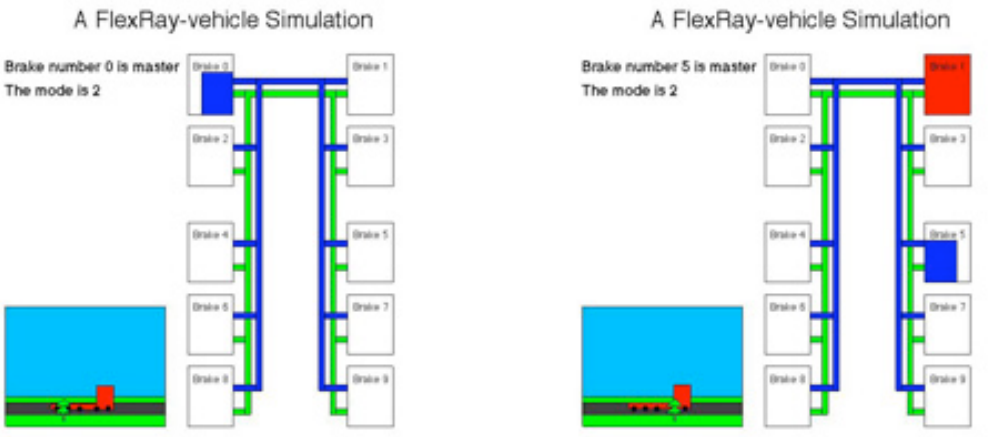


Figure 48 – *The left picture shows a snapshot of the animation when everything is working. In the right picture node 1 is disconnected. This means that node 0, 4 and 5 now calculate the dynamics of the vehicle and vote a common result. Node 1 is isolated.*

6 Discussion

To be able to make any conclusions it is important to return to the list with problems that needed to be solved in the first chapter under section 1.3. To get a better overview, the questions and some short answers are shown in the table below:

Questions:	**Answers:**
How should the channels from each node be connected in the network to create an optimal distributed control system?	From the last chapter it is known that many different smaller networks cause much harder schedule of the nodes, and the states are almost impossible to get deterministic. Because of this all nodes are connected to the same network, one for each channel.
How should the FlexRay-channels be configured to optimize the correct number of messages to be sent from each node?	This is not really a problem since there is a lot of free space on the network. But it is still important that each node can use an equal number of slots in each cycle.
In the system used today the Electronic Control Units (ECU) have some sort of redundancy to handle errors on calculations e.g. regarding the dynamics of the vehicle. How can the same level of redundancy be achieved in the distributed network without the ECU:s?	Instead of using the ECU:s, the distributed nodes take over the responsibility for the redundancy checks using the example with the master node and the TMR-voting.
If there is an error in the calculations or if a node fall off – how can this be detected and corrected?	The easiest way is for listen to null-frames or if a node is not sending anything. If there is a calculation error this can be detected by comparing the results from other nodes.
In order to send and receive dynamic data over the network the synchronization of the messages needs to be solved.	The dynamic slots are not used. But still, it is possible to alter the data to be sent in the static slots but this is not done because it is better to always know what data that will be sent in every slot to get a more deterministic system. If wanted, the message-header can be used to send different kinds of messages.

With the answers as a starting point, I will now give my own reflections on the system and how I think it should be designed.
From section 4.2.1 where the suggested design 1 first was introduced the nodes was connected to each other using one bus for each channel. However, this is not the best way to connect the nodes if there is important with redundancy because if there is an error to the whole link, all nodes will be disconnected! Instead I think it is better to use star couplers to connect the nodes (see figure 49).

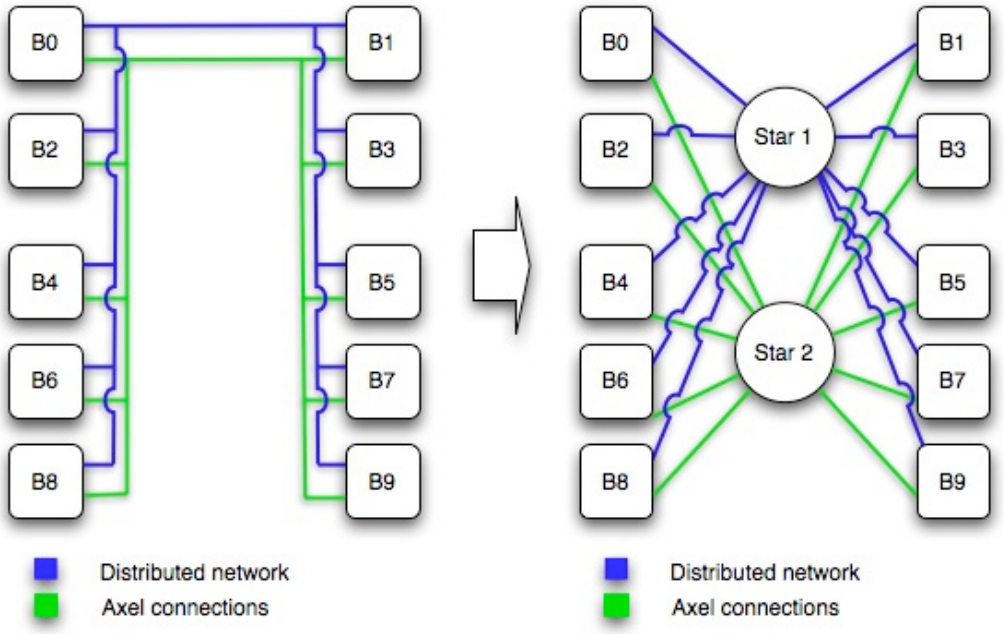


Figure 49– *The design uses star coupler instead of the bus to create a more robust network.*

It is important to remember that this choice does not affect the result of how the data is sent, only the robustness of the network. The cost is a little higher because of the star couplers, but I think this is the best solution anyway, because of the fact that the implementation is so much easier and the cost will be much lower than using a more complex system with many smaller networks to achieve the robustness.

When it comes to the configuration of the FlexRay-cycle in design 1, the dynamic part of the cycle was not used. I think the dynamic part is more suited for other applications and not for high safety and time-critical applications. That conclusion is based on the fact that the system will be more complex and that the bus guardian is not supported by support the dynamic part in all hardware.

Instead it is better to use cycle multiplexing and a message-header to alter the information that is being sent in the different cycles. This does also add more control of the different cycles because it is the developer that makes the decision of what will be sent instead of states internal to the nodes.

There are two important thing left. In suggested designs 2 and 3 that were dismissed, there was a question about how many nodes should take part in the voting mechanism. In design 2, all the nodes were present and in design 3 only one extra node did the calculations. Using only one extra node is a bad idea according to me. My decision is based on the fact that the scheduling is much more advanced and that more internal states is required. The second thing I had in mind was the choice only to use 4 nodes as distributed nodes. I think it is very important to find design solutions that do not depend on how many nodes the vehicle have. The main reason why this is my conclusion is the lower cost of having one solution instead of many. Another interesting thing is how the master-role is adaptive. Assume that a certain node needs to be replaced. Then it is important that the system does not need to be reconfigured, anyone should be able to install the new brake-module without any knowledge of the system.

Finally, there is a lot more work that needs to be done before vehicles use this kind of braking-system. In the specification there were two units of times, 2.5 ms and 10 ms. I think it would be very interesting to investigate where the limits are. Is it possible to decrease the 10 ms cycle to let say 5 ms? And what dose it has for consequences for the dynamics of the vehicle? It is all a question of how fast the hardware and the sensors are. Another important thing that has not been optimized is the slot-time. In all the examples the slot-time was 60 µs in order to send 16 words in each slot. The question is if 16 words is the perfect value? The solution to this question is to investigate what data that really needs to be sent and optimize the slot-size to this value.

It would also be interesting to compare different kind of master-slave designs to find advantages with a specific solution in order to find a safer system.

6.1 Conclusions

As a summary, I will now state some rules of how I think the system shall be designed.

- Using star-couplers to add robustness
- Only use two networks, one for each channel
- Only use the static part of the frame
- Use a message-header to send different kinds of data
- Make the system *simple*!

Now at last, I want to go back to the beginning of chapter 1, figure 1. The main subject of this master thesis was illustrated by the figure (see figure 50).

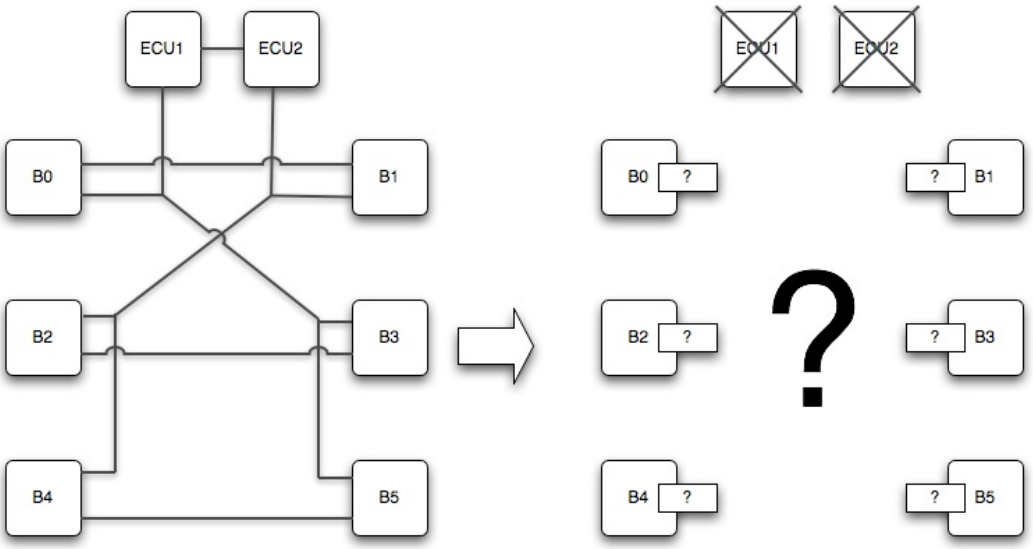


Figure 50 – *The first picture in this document that showed the main subject of the work, to find a new design without the ECU:s.*

Now, the picture can be replaced by the following (see figure 51).

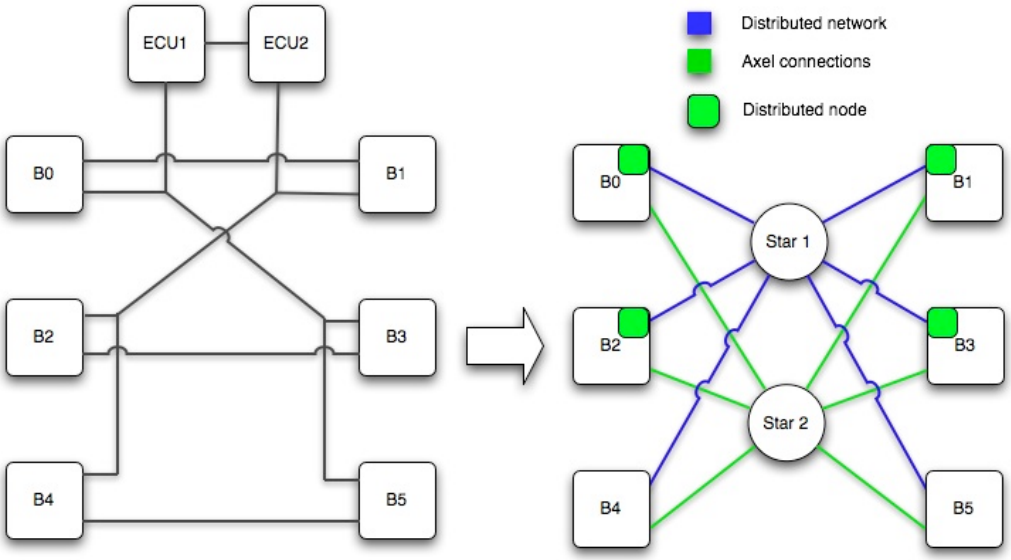


Figure 51 – *The final design of the network.*

The system does now consist of two networks, one for sending sensor data, and one for handle the traffic generated by the distributed nodes.

We have seen that there is a lot of capacity on the FlexRay-networks. My strongest advice is to keep the design as open and simple as possible. The different solution designs in this report can be seen as examples. In fact, they are only showing one way of implementing the system. The possibilities are endless!

References

[1] Controller Area Network – Wikipedia the free encyclopedia, http://en.wikipedia.org/wiki/Controller_Area_Network Fetched: 2007-07-10

[2] FlexRay – The communication system for adv. Automotive control applications, ”about FlexRay”, http://www.flexray.com/index.php?sid=57fd30239bbe4660d9f5e4b9b1a2e259&pid=80&lang=de Fetched: 2007-07-11

[3] Electromechanical Braking (Brake-by-wire), ”Overview”, http://www.freescale.com/webapp/sps/site/application.jsp?nodeId=02Wcbf07jSLR2Q Fetched: 2007-07-13

[4] FlexRay-Protocol Specification A2.1, http://www.softwareresearch.net/site/teaching/SS2007/ds/FlexRay%20-%20Protocol%20Specification_V2.1.rev%20A.pdf Fetched: 2007-10-17

[5] FlexRay Protocol Reference chart – Vector, https://www.vector-worldwide.com/ va_catalog_us,,4165.html, Fetched: 2007-10-19

[6] FlexRay, Technology: FUIJUTSI, http://www.fujitsu.com/global/services/microelectronics/technical/flexray/index_p10.html Fetched: 2007-10-15

[7] Byteflight, http://en.wikipedia.org/wiki/Byteflight Fetched: 2007-10-19

[8] Distributed Control System, http://en.wikipedia.org/wiki/Distributed_Control_System, Fetched: 2007-10-19

[9] FlexRay-Protocol Specification A2.1, http://www.softwareresearch.net/site/teaching/SS2007/ds/FlexRay%20-%20Protocol%20Specification_V2.1.rev%20A.pdf Fetched: 2007-10-17

[10] Neil Storey, Safety-Critical Computer Systems, 1996, Harlow, ISBN:0-201-42787-7

Appendix A: The simulation in TrueTime

The simulation consists of a model of a 10-wheeler vehicle. Every wheel is in fact a model of the hardware of the real node. The nodes are connected to each other using channel A and B, which is represented by two network blocks.

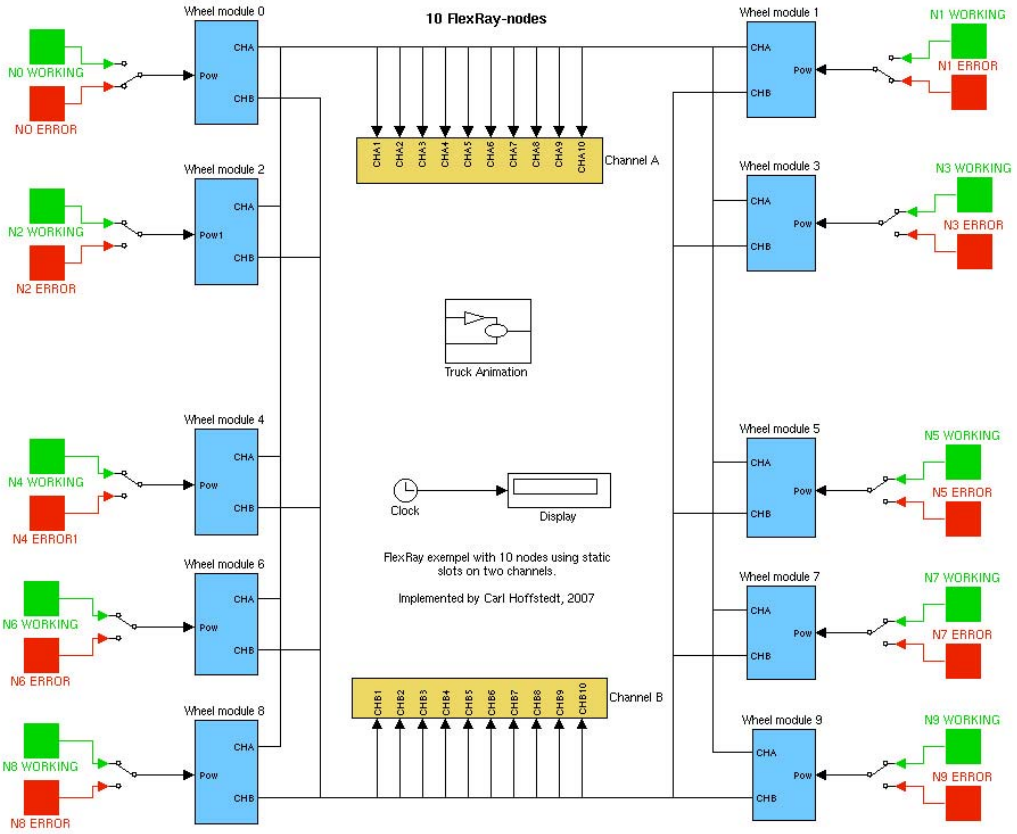


Figure A1 – *The main model of the simulation. Each node (the blue blocks) can be turned off by the switches. The yellow boxes are the networks.*

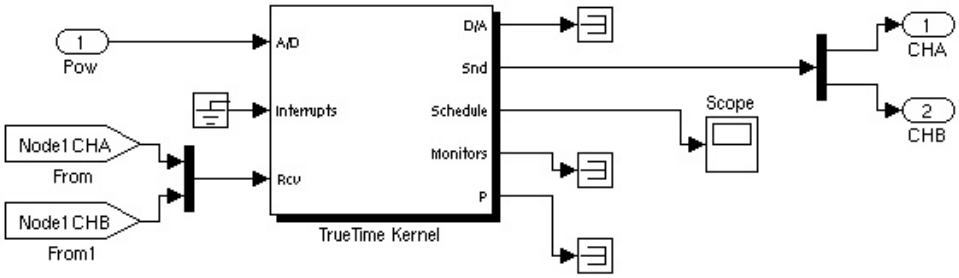


Figure A2 – *Each node consists of a TrueTime-kernel that has one input for receive and one input for send. On the receive port, the data from the different channels are connected using a multiplexer. The same is true for the send-port. On the A/D-port the signal from the switch is transported into the channel.*

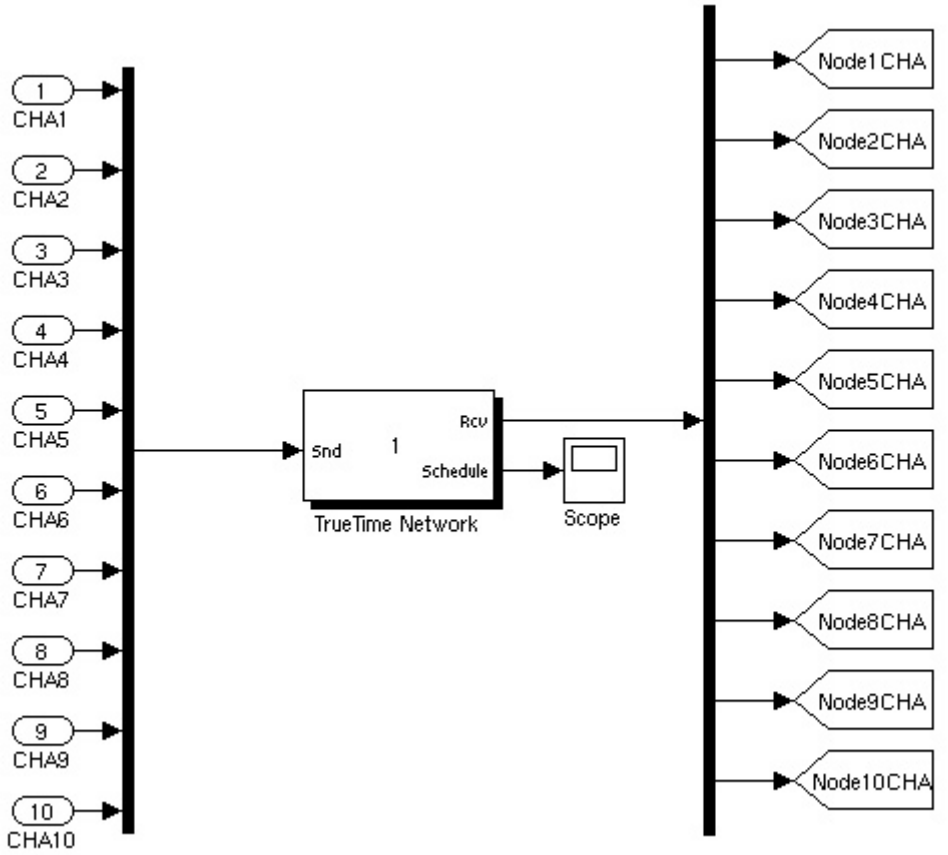


Figure A3 – *The blocks used for representing the networks consists of a TrueTime-network where all the nodes are connected. This picture shows the network used for channel A. The network used for channel B look the same but the labels are different.*

Index

R

S

T

U

V