

Institutionen för systemteknik

Department of Electrical Engineering

Examensarbete

Simulation and synchronization of distributed real-time systems

Examensarbete utfört i Fordonssystem
vid Tekniska högskolan i Linköping
av

Andreas Karlsson and Joakim Leuhusen

LITH-ISY-EX--09/4161--SE

Linköping 2009



Linköpings universitet
TEKNISKA HÖGSKOLAN

Simulation and synchronization of distributed real-time systems

Examensarbete utfört i Fordonssystem
vid Tekniska högskolan i Linköping
av


Andreas Karlsson and Joakim Leuhusen

LITH-ISY-EX--09/4161--SE

Handledare: **Erik Hellström**
ISY, Linköpings universitet
Roger Eriksson
BAE Systems Hägglunds AB
Martin Hallberg
BAE Systems Hägglunds AB

Examinator: **Lars Nielsen**
ISY, Linköpings universitet

Linköping, 13 February, 2009

	Avdelning, Institution Division, Department Division of Automatic Control Department of Electrical Engineering Linköpings universitet SE-581 83 Linköping, Sweden	Datum Date 2009-02-13
Språk Language <input type="checkbox"/> Svenska/Swedish <input checked="" type="checkbox"/> Engelska/English <input type="checkbox"/> _____	Rapporttyp Report category <input type="checkbox"/> Licentiatavhandling <input checked="" type="checkbox"/> Examensarbete <input type="checkbox"/> C-uppsats <input type="checkbox"/> D-uppsats <input type="checkbox"/> Övrig rapport <input type="checkbox"/> _____	ISBN _____ ISRN LITH-ISY-EX--09/4161--SE Serietitel och serienummer ISSN Title of series, numbering _____
URL för elektronisk version http://www.control.isy.liu.se http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-ZZZZ		
Titel Simulering av distribuerade realtids system i Stateflow och TrueTime Title Simulation and synchronization of distributed real-time systems Författare Andreas Karlsson and Joakim Leuhusen Author		
Sammanfattning Abstract <p>Today we are very much dependent on different kinds of real time systems. Usually, a real time system is a system which is interacting with a physical environment with sensors or activators. There are many advantages by replacing mechanical components with electrical ones. For instance, it is usually cheaper and possible to add new functions to the device without replacing the electronic part, which would have been necessary with a mechanical one.</p> <p>The possibility of simulating a distributed system is used throughout the vehicle industry. With the simulation of connected sub systems, using modeled buses and real time kernels, one could increase the correctness of the behavior of the system and consequently decrease the amount of time spent later in the developing process.</p> <p>In this master thesis we used modeled CAN-buses and real time models to simulate the connection and execution time of the systems. The simulation results are used to validate the functionality of the distributed system. Additionally, a worst-case response time analysis is made to set timing constraints on the system to fulfill given deadlines.</p> <p>During the work, different settings of the network are tested to analyze the system frequency needed to sustain deadlines and correctness on the network.</p>		
Nyckelord Keywords Realtime, distributed, network, synchronization, TrueTime, Stateflow		

Abstract

Today we are very much dependent on different kinds of real time systems. Usually, a real time system is a system which is interacting with a physical environment with sensors or activators. There are many advantages by replacing mechanical components with electrical ones. For instance, it is usually cheaper and possible to add new functions to the device without replacing the electronic part, which would have been necessary with a mechanical one.

The possibility of simulating a distributed system is used throughout the vehicle industry. With the simulation of connected sub systems, using modeled buses and real time kernels, one could increase the correctness of the behavior of the system and consequently decrease the amount of time spent later in the developing process.

In this master thesis we used modeled CAN-buses and real time models to simulate the connection and execution time of the systems. The simulation results are used to validate the functionality of the distributed system. Additionally, a worst-case response time analysis is made to set timing constraints on the system to fulfill given deadlines.

During the work, different settings of the network are tested to analyze the system frequency needed to sustain deadlines and correctness on the network.

Acknowledgments

First of all we would like to thank our supervisors, Erik Hellström at the department of Electrical Engineering at Linköping University, for his encouragement and positive attitude. Secondly we thank Roger Eriksson and Martin Hallberg at BAE Systems Hägglunds for all the support during the work.

We would also like to thank the following people for their engagement during the progress of our master thesis:

- Soheil Samii at the department of Computer and Information Science, Linköping University
- Anton Cervin at the department of Automatic Control, Lund University
- Palencia Gutierrez, Jose Carlos, departamento Electronic Y Computadores, Universidad De Cantabria

Contents

1	Introduction	1
1.1	Purpose	2
1.2	Problems To Solve	2
1.3	Limitations	3
1.4	Method	3
2	Design Of Simulator	5
2.1	Warning Lights Model	5
2.1.1	Network	5
2.2	Hybrid Brake Model	6
2.2.1	Mechanical Brake	12
2.2.2	Electrical Brake	12
2.2.3	Network	12
2.3	Design Alternatives	13
2.3.1	TrueTime	13
2.3.2	Stateflow In TrueTime	14
2.3.3	TrueTime/Stateflow	14
2.4	Choice Of Design	15
3	Real-Time System Analysis	17
3.1	Response-Time Analysis	17
3.1.1	Exact Response-Time Analysis	17
3.1.2	Upper-Bound Approximation for Worst-Case Analysis	22
3.1.3	Analysis for Tasks with Dynamic Offsets and Distributed systems	24
4	Experiments	27
4.1	Task Model	27
4.2	Results	30
5	Methodology Summary	37
5.1	True-Time Implementation	37
5.2	Task Model	40
5.3	Synchronization	40

6	Conclusions	41
7	Future Work	43
	Bibliography	45
A	Matlab Code	47
A.1	The TU-Code File	48
A.2	The Messagehandler File	49
B	Tables	50
B.1	Simulation Parameters	51

Chapter 1

Introduction

This master thesis is performed in collaboration with BAE Systems Hägglunds in Örnköldsvik, Sweden, and the Division of Vehicular Systems at the Department of Electrical Engineering at Linköping University.

A lot of research has been done in the subject of distributed systems and real-time-synchronization and it is a hot subject because of its very wide application. There are various definitions of what a distributed system is so in this thesis we define it as follows: *a distributed system refer to a system where different single-processors have been connected by a network making it a system that consists of several subsystems*. In distributed systems it is important that all nodes have the same time synchronized on their internal clocks. In many cases this synchronization is done automatically by the network, as they are in our system. Also, one needs to know how long time it takes from sending messages until they reach their final destinations. This time can ideally be calculated off-line using worst-case-response time analysis or in some cases be estimated using simulation.

There is a lot of research being done in response-time analysis. One of the earliest articles, which has spawned many follow ups, is: "Holistic Schedulability Analysis for Distributed Hard Real-time Systems" [7]. In the article it is shown how response time analysis is done in a CPU which have tasks with static priorities. This is further developed in [8], showing how scheduling is made in distributed systems with Rate Monotonic (RMA). Lately there has been a trend towards studying scheduling analysis for dynamic distributed systems and a less pessimistic analysis.

"The Gast Project" is a relatively new project in combination with industrial companies that are trying to come up with good solutions in x-by-wire problematics. Nowadays there are only military vehicles that use x-by-wire for safety critical applications like steering and breaking. In civilian vehicles the hardware is only operating the non safety critical applications. The goal is that in the future even civilian vehicles will be running with x-by-wire.

A very interesting and important problem in x-by-wire structures is how the system responds to a fault in terms of redundancy. If a node breaks down or malfunctions, it is important that the systems runs smoothly anyway or at least

informs the operator. In this thesis we will not try to implement redundancy but in reality it would be needed.

As an aid in the developing process we used a Simulink compatible program called TrueTime, made by Lund University in Sweden. This is a series of S-functions, mex and Matlab files, used to simulate a real-time processor kernel with scheduling and buses. TrueTime is free to download at [11] and is used in various programming courses throughout Sweden. There are six different blocks that can be used to simulate real-time and buses in the Simulink environment. The main one is called the "TrueTime Kernel" which is used to simulate the processor core. TrueTime can be used for a variety of system validation analysis. One can see how different schedules affect the nodes or how different systems work together. One problem we faced early on in the development was that TrueTime is not completely compatible with Stateflow models. We needed to come up with a solution to this problem to be able to continue with our work.

1.1 Purpose

The main purpose with this master thesis is to come up with a methodology that can be used when testing distributed Simulink/Stateflow models, communicating over a simulated CAN and real-time network. The analysis will contain sample rate, disturbance and response time calculations of the generated distributed system. Our goal is to be able to analyze the models and find the worst-case response time of the system.

The main objective of this thesis is to describe a methodology for testing distributed systems. At this time, there is no such help at Hägglunds which could ease and speed up the developing process. So hopefully, this thesis can add a new dimension to Hägglunds engineering team.

1.2 Problems To Solve

The main problem in this thesis is to find out how to build a model of a distributed system that can be used when analyzing the interactivity between different computer nodes. We basically have four main difficulties that need to be solved:

1. The first problem is to come up with a plan on how to implement the distributed system and choose which tools that should be used.
2. Without some sort of communication the nodes will not be able to synchronize which means that we have to come up with a way for the nodes to communicate with each other and agree on how and when to simultaneously cooperate.
3. The third problem is how to find the response times of different signals in the system.
4. When we have found out how to calculate the response times in a distributed system we also need to apply the analysis to our models.

1.3 Limitations

There are many difficulties when studying distributed systems. To name a few there are: clock drift, redundancy and utilization of computer capacity. However, we will assume that the clocks are perfect and there is no redundancy. We will not study different scheduling policies.

1.4 Method

We started the work by doing an extensive research on the subject concerning CAN, real-time information, TrueTime, Stateflow and more. In the developing stage, we used a simplified model of a blinking light system which is used in BAE Systems Hägglunds "SEP". The blinking system represents the vehicles warning lights and is used as a simple demonstration over how this template is supposed to function. There are 7 different lights on 5 different nodes making it difficult for the system to make all lights blink at the same time. Our second model simulates a hybrid braking system which consists of one mechanical brake and one electrical brake that are used together, over a set of nodes, making a single brake force out of the two systems. This is a far more complex system than the warning lights with more nodes, buses and systems involved in the process. The system contains a simple car model representing the vehicle, node-blocks representing the actual hardware used by the vehicle and CAN buses used for communication between different nodes. We implemented the models in different ways to see which approach that was best suited in different cases.

Chapter 2

Design Of Simulator

In this chapter we introduce the models and different alternatives for building a model of a distributed system in computer software. We compare different programs that can be used, benefits and drawbacks, and conclude about the best way to build the model. We describe both with images and text how the systems work. We have only been using a small portion of the SEP-vehicle's internal nodes but these are the nodes used in the braking and blinking system of the vehicle. Since the SEP is a military vehicle we can, in some cases, not give out exact information but instead write in more general terms.

2.1 Warning Lights Model

The warning light system, illustrated in Figure 2.1 and 2.2 below, contains 6 nodes and 3 buses. All the buses are CAN buses, with a maximum transmission speed of 1 Mbit/s, and all the nodes represent real-time kernels, CPUs. The model represents nodes used in the SEP and shows the problem with synchronizing all the warning lights. Since the nodes are connected with different buses they have to communicate data between one another which would make the lights blink asynchronous because of the time delay, if nothing was done to fix the problem. This made the system a good starting ground for testing and understanding the problem with unsynchronized nodes. In order to synchronize the system one have to come up with an idea on how to make the nodes concurrent.

2.1.1 Network

When the driver pushes the warning lights button, illustrated by the switch in Figure 2.1, the system should respond accordingly by sending a signal to the actuators which turn the lights on and off synchronous. Node A2 is the first node in the communication chain who receives a signal from the on/off button. When A2 senses a change in the lights button it sends a activation signal over the CAN1 bus. The signal then propagates over the CAN1 bus and reaches node A1 and A3. Node A1 is then able to start the warning lights as it desires but A3 have to

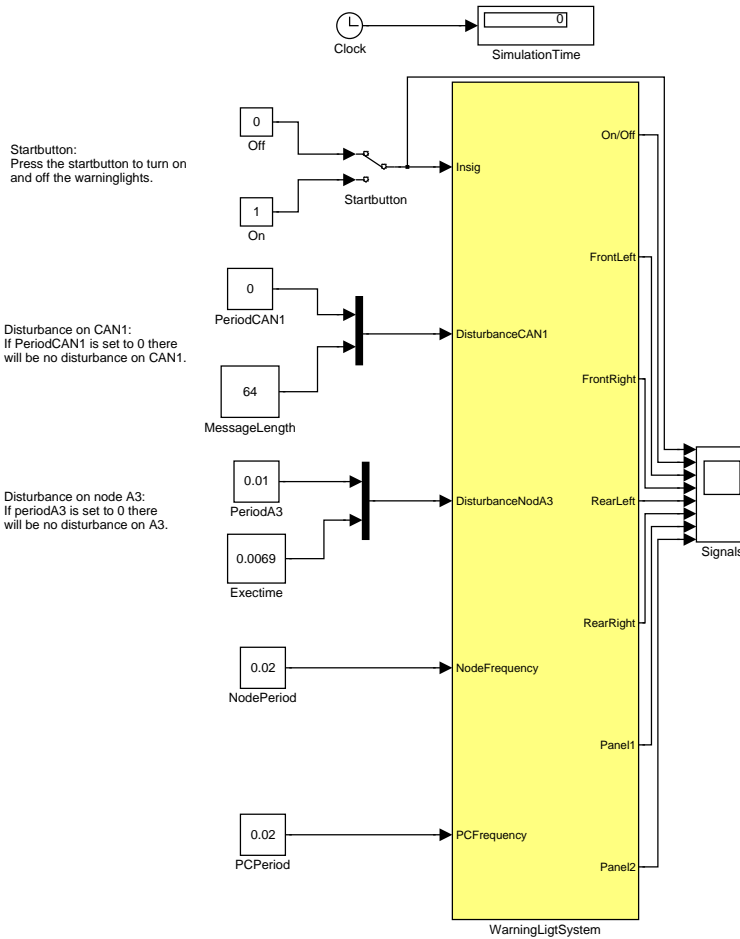


Figure 2.1. The Simulink implemented warning lights, user interface

send another signal over CAN2 and CAN3, telling node A5, PC1 and PC2 that the system is active. Now all the nodes have information about the state of the warning lights button and are able to compute the information. The CAN buses that we simulate are CAN 2.0 A, standard format [3]. The priority of each message is unique and set in the identifier bits of each message frame.

2.2 Hybrid Brake Model

The hybrid brake system contains 11 nodes and 6 buses. All the buses are CAN buses, with a maximum transmission speed of 1 Mbit/s, and all the nodes represent real-time kernels, CPUs.

We can see the model in Figure 2.3 what the shell of the brake system looks

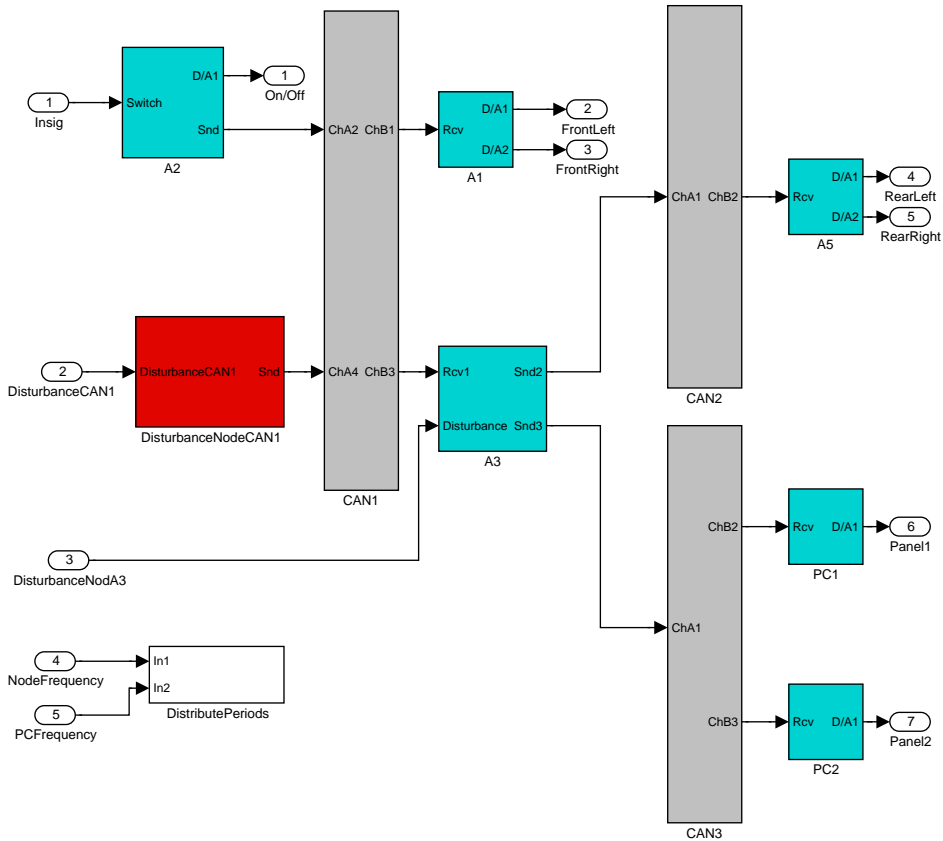


Figure 2.2. User interface of the warning lights model

like. We are able to excite the system by simply using one of the signal inputs and looking at the scope. Inside the shell lies the brake model which can be seen in Figure 2.6. The vehicle starts at a predefined speed and decreases at a rate that is proportional to the pedal position until it reaches a minimum velocity of zero.

In Figure 2.4 it can be seen how we created the BU node with Simulink, True-Time and Stateflow. Since we could not come up with another way of using Stateflow without losing the internal states, we let the Stateflow be in an enabled subsystem so that we can trigger it for as long as we would like. Now we only want the Stateflow model to be active for one “tick“ at the time, this means that we send a signal that enables and disables it after a preset period of time which coincides with a time set in the TrueTime kernel. In this way we can enable and then disable the model, forcing it to only do one “tick“.

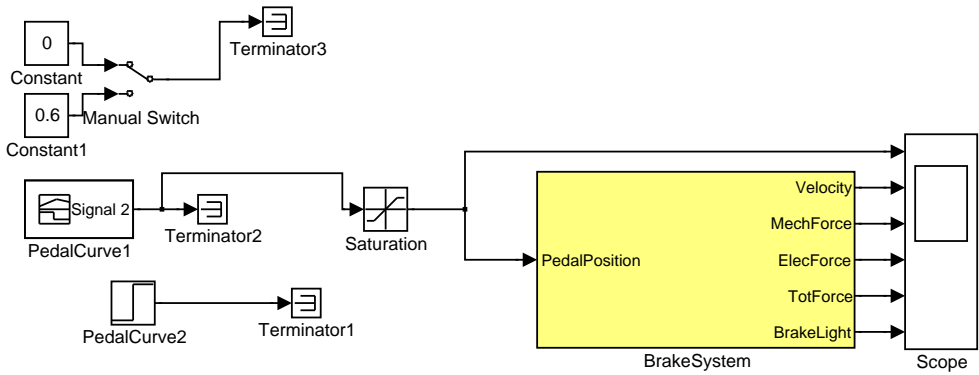


Figure 2.3. The Simulink implemented brake model

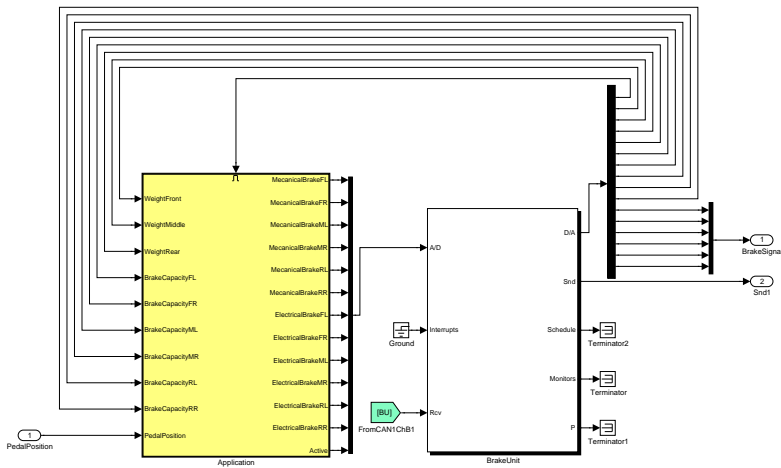


Figure 2.4. The Brake Unit implemented with True-Time and Simulink

The model in Figure 2.6 represents nodes used in the SEP and shows the problem with synchronizing the electrical brake and the mechanical brake. Since the nodes are connected with different buses they have to communicate data between one another which would make the brakes brake asynchronous because of the time delay. In order to synchronize the system one have to come up with an idea on how to make the nodes work concurrently.

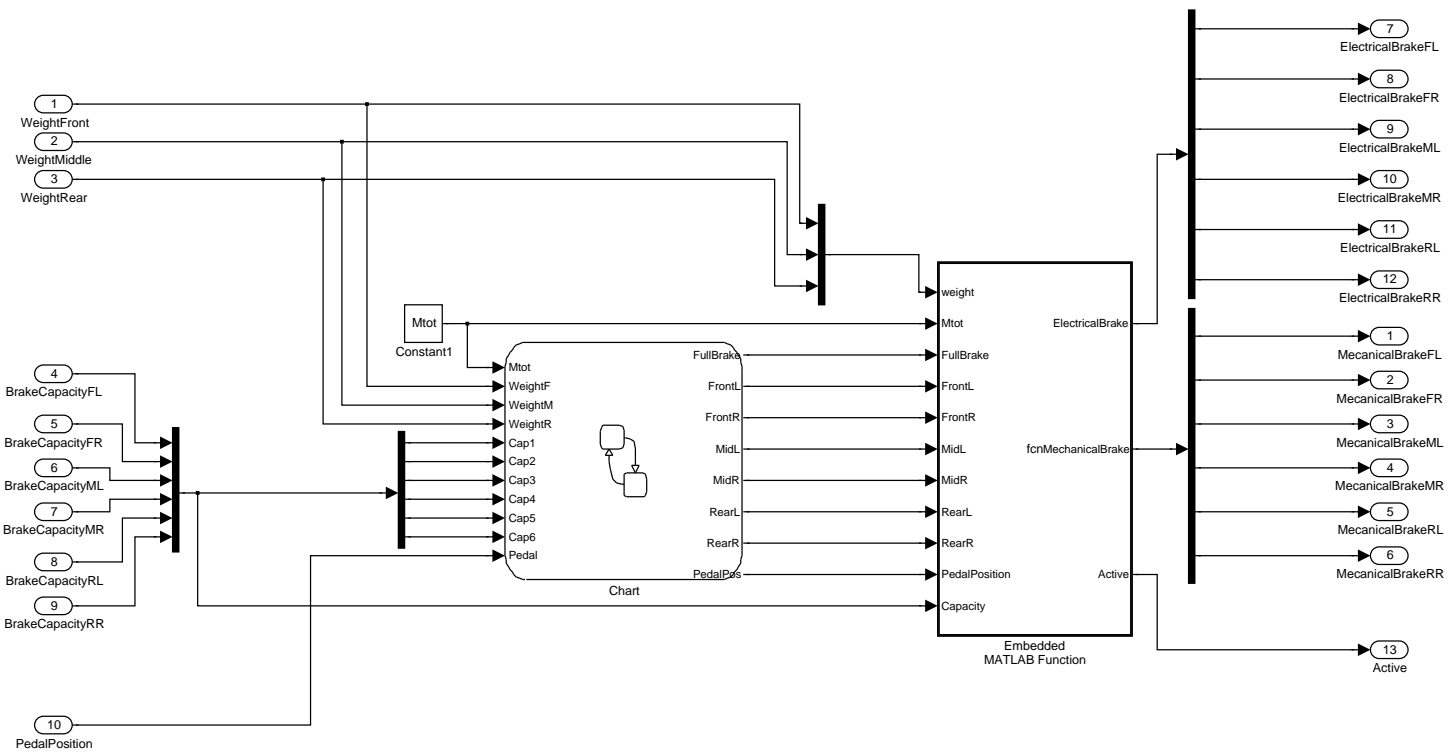


Figure 2.5. The Simulink implemented application on Brake Unit

In the real SEP vehicle the nodes do far more calculations than just the brake control. For our simulation we need to simplify these nodes but still keep all the time aspects intact. All the nodes are working periodically and the problem is to find at which period the nodes should be running, while still managing to cope with given deadlines and find how the communication works with that periodicity. The nodes can only handle a limited amount of messages every period even though the bus is faster. This means that every node has to be set either a fixed amount of messages it can handle every period, or include this in the modeling so that different messages take different time to process in the nodes. Given deadlines for the system have been supplied by BAE Systems Hägglunds. From the time the driver starts braking to the time the actuators give the signal to brake should not exceed 0.6 seconds according to law statements. Our goal is to reduce this time considerably down to satisfying 0.1 seconds. When the brake pedal is pushed, the brake lights should be lit in 0.1 seconds. To be able to study the functionality of the hybrid brake model we need to implement a simple vehicle model which responds to the brake signals. The electrical signals coming from the electrical system need to be transformed to physical brake force. We also introduced the relationship $F = ma$ as well as a mapping between the vehicle velocity and the electrical brake capacity. The mapping is done so that the electrical brake capacity is higher when the velocity of the vehicle is small and then decreases when the velocity increases. This was all done in the VehicleModel in Figure 2.6.

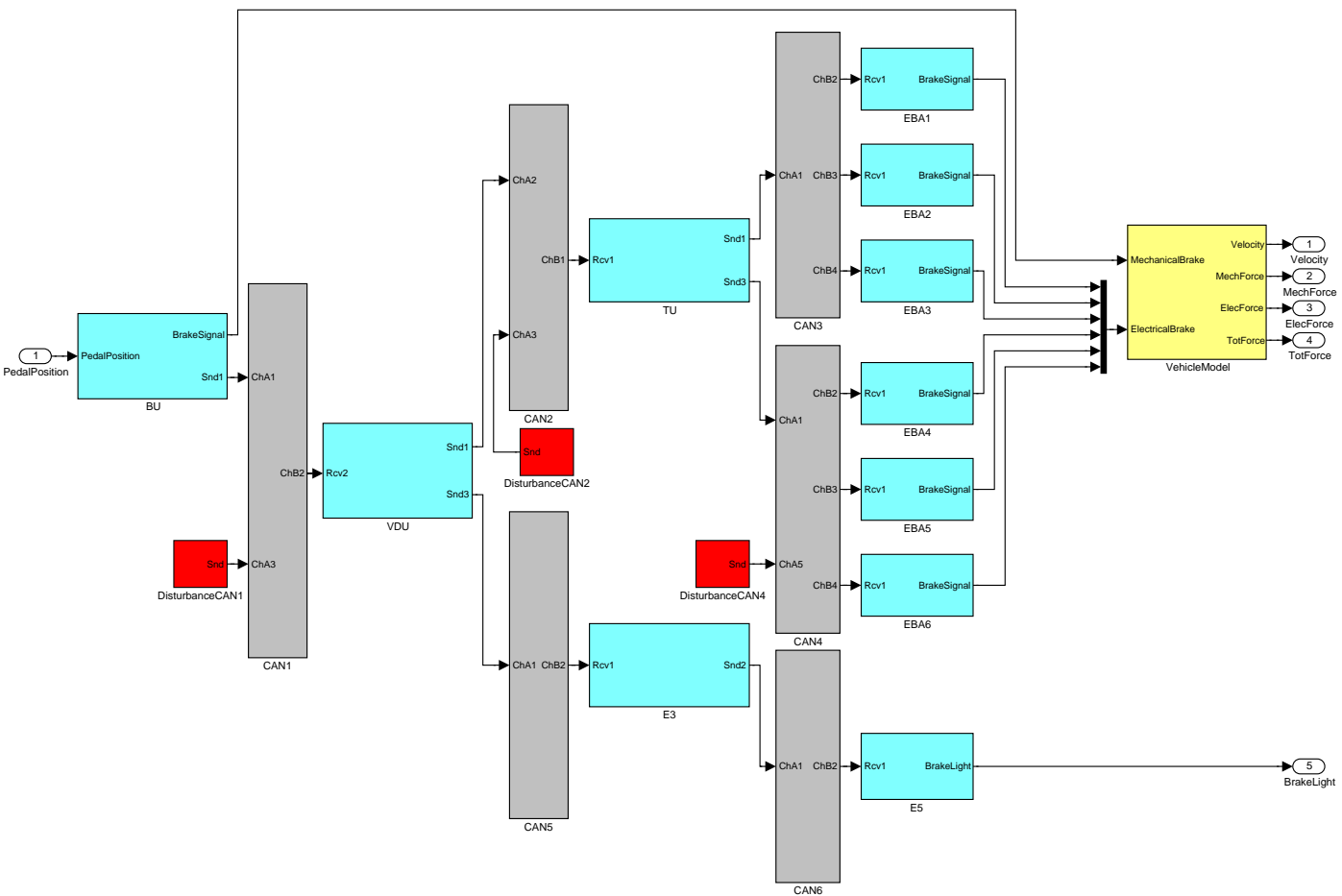


Figure 2.6. The Simulink implemented brake model

2.2.1 Mechanical Brake

The mechanical brake is a conventional brake system containing hydraulic. This means that the system is using hydraulic pressure to enforce the muscle power given by the driver to the brake pedal.

The basic idea is that one can control pistons, and thereby the pressure, by shifting the brake pedal a fluid called brake fluid can apply pressure to pistons in the wheel cylinders so they brake the forward motion. Because of the friction and heat created by the braking one will eventually wear out the pads.

In most four wheeled vehicles, the master cylinder is divided into two which each controls two wheels. This is mainly done to increase safety so that if one system brakes you will not lose the entire brake capacity of the vehicle [2].

In the SEP the driver's pedal sends a signal which propagates to a computer which validates and transforms the signal into a current which controls a valve. The valve controls the fluid to the pads or shoes which brakes harder the more fluid that is sent to the brake block by opening or closing this valve.

2.2.2 Electrical Brake

The functionality is basically the same as for the mechanical brake but instead of having pressure from fluid one have pressure from actuators on the brake ring. These electrical brake actuators are given signals from a control box which has been given signals from a computer. The computer is directly connected to the brake pedal so that depending on the pedal the computer calculates the electrical signals from the pedal into signals that can be understood by the control box [1].

An electrical brake works like a reversed electrical motor. Instead of feeding the electrical motors with current so that it helps pushing the vehicle forward, the system gives less than the necessary current to the electrical motors so that it starts braking the vehicle instead. This current could then be stored in a battery and be used in other applications instead. The drawback is that the electric brake is able to brake less for increased speed. This means that the system needs to use the mechanical brake only at high speed and then uses more and more electrical brake the more the vehicle has slowed down. Also, the brake capacity is measured by the EBA and then sent back to the BU which means that the measured value is evaluated and applied a bit later, depending on sample rate and bus capability.

2.2.3 Network

In the SEP model, every node is working with a fixed period rate with only one task in every node. The node called BU, Brake Unit, is the node that needs to do the most application in our model. It needs to send signals to node E5, who turns on the warning lights, as well as all the EBA nodes and turn on the mechanical brake. It needs to consider many factors before it sends out signals, such as:

- Pressure on axles, sent from TU over VDU.
- Brake capacity of the EBA, sent from EBA over TU and VDU.

- Pedal position, given by the driver.

When all of these factors have been considered BU will send signals over the network to the desired nodes with information about how hard to brake and when.

Vehicle Dynamics Unit, VDU, is the node that knows the pressure on each wheel relative the ground. Also, in the network it is the node that is positioned between BU and all the rest of the nodes which means that it will be a gateway in the system since all messages needs to go through it.

Traction Unit, TU, is the node which lies between VDU and all the EBA. In our model it is not doing any calculations but in reality it has more applications then simply sending received messages. It is however receiving many messages from many nodes so it is a likely bottleneck for the communication in the system.

E3 is simply sending received messages from VDU to E5.

E5 will turn on the brake light as soon as it gets a signal from E3. Here there is not any synchronization problem but instead it just needs to warn vehicles from behind that it is braking.

All of the Electrical Brake Actuator, EBA, nodes are working in the same way. The only difference is that three nodes are positioned on CAN3 and the last three are positioned on CAN4. These are the nodes that acts upon the electrical brake. Their application is waiting for a message from the BU that tells the EBA to start braking, and how hard, as well as sending information on how much the brake is able to brake at given speed to BU. The nodes know how much the brakes can brake at all time simply by measuring the wheel speed.

2.3 Design Alternatives

2.3.1 TrueTime

TrueTime is a Simulink library developed at the department of automatic control at Lund University in Sweden [11]. The TrueTime library offers some new Simulink blocks, for example the "TrueTime Kernel", the "TrueTime Network" and the "TrueTime Battery" blocks. Figure 2.7 below shows the TrueTime Kernel block to the right and the TrueTime Network block to the left, which were the only TrueTime blocks needed in our models.



Figure 2.7. The TrueTime Kernel and TrueTime Network blocks

The Snd and Rcv ports on the Kernel block are used for sending messages to and receiving messages from the network blocks. The A/D and D/A ports are used for analog signals to and from the node and can be connected to ordinary Simulink blocks. The Interrupts port is used for external interrupts and triggers and the Schedule port can be connected to a scope block which makes it possible to study the execution order and execution time of the tasks on the node and the Monitors port can be used for task synchronization. The last port, P, is used together with the TrueTime Battery block to simulate power consumption.

Our first design alternative was to build the systems with only TrueTime/Simulink blocks and to implement all functionality as tasks using Matlab code functions. We built the system in an iterative way by adding the nodes one by one to the model and connecting them to the CAN networks. After a new node was connected to the network we implemented all functionality on the node as periodic and aperiodic tasks. The aperiodic tasks can be triggered by external events from the interrupts port or by internal events like deadline overruns etc. When a trigger signal is detected a related interrupt handler will be activated to start the execution of a task. After we had implemented the node's functionality we tested the communication between the new node and the other nodes, connected to the same CAN-bus. If the system behaved as supposed, we would begin implementing the next node.

2.3.2 Stateflow In TrueTime

Stateflow is a graphical design tool that can be used with Simulink to model complex logical behavior of a system [10]. The operating mode of the system is represented as states in a chart, where transitions between states happen when a certain criterion is met. Stateflow is widely used in the automotive industry and therefore it would be nice to implement the system behavior using Stateflow charts.

Our second design alternative was to build the system in the same way as in the first design alternative, but instead of implementing the functionality directly in the task code we called a Simulink subsystem from the code. Our intention was to call a subsystem containing a Stateflow chart and thereby reduce the code and make the model more lucid.

2.3.3 TrueTime/Stateflow

Our third design alternative was to combine both TrueTime and Stateflow blocks on the same level in the model. In this case we used a single task in the kernel block to send messages, read analog signals and write analog signals. One of the analog signals from the node was used to enable a subsystem containing the nodes application in a Stateflow chart. When the subsystem becomes enabled, the states in the chart are updated and a new input to the node is generated. Figure 2.8 illustrates the connection between a Kernel block and a triggered subsystem and Figure 2.9 shows the Stateflow chart together with a user defined code function that is placed inside the triggered subsystem.

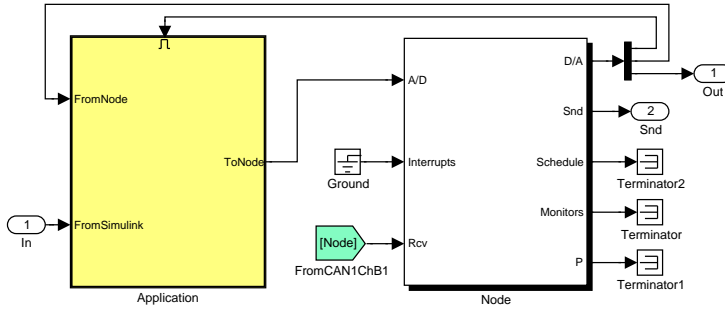


Figure 2.8. Design alternative 3, the kernel node triggers a subsystem containing the Stateflow chart

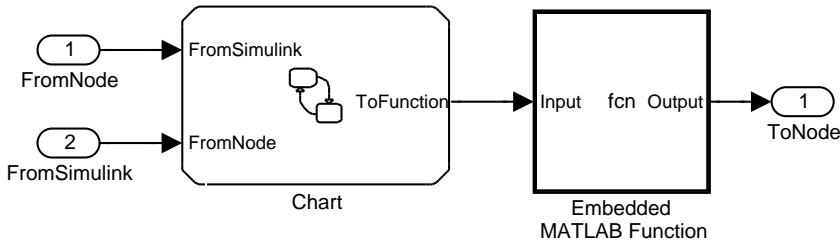


Figure 2.9. Design alternative 3, the Stateflow chart

2.4 Choice Of Design

One of the biggest benefits with using TrueTime is that buses and real-time kernels can easily be modeled, and connected to each other, to form a distributed system. This makes TrueTime a very suitable tool indeed when simulating real time aspects of a system, such as response times, synchronization and schedulability of tasks on the CPUs.

The first design alternative, using only TrueTime, is well suited when building a new model of a small system. The application on each node can be implemented as periodic and triggered tasks and one can get a working model quickly. In a more complex system it might be hard to implement all functionality in the tasks and it can be difficult to get a good overview of the model.

Other advantages that comes with TrueTime is that it is compatible with Stateflow and other important Simulink libraries which makes it possible to combine and use different simulation tools to create a good model. It is free to download, tasks can be implemented as M-files or C++ functions and it is also possible to call Simulink block diagrams from within the code functions. In addition the TrueTime source code is open and free to be modified by the user, and it contains C, Simulink, Matlab and S-functions.

The second alternative would have been a good solution for complex systems and an easy way to test how different models, already modeled, work together in a distributed system. The problem is that, when the Stateflow block system is being called from TrueTime it does not remember the internal states from the last call. On the other hand if the states in the block system do not need to be saved, this would be a good design choice.

The third design alternative is also intended for complex systems. With this design one need to place the Stateflow models beside the kernel nodes and connect them to each other, which requires a little more work than calling them from a block system. The advantage is that internal states can be saved.

Since BAE Systems Hägglunds request an easy way to test their models in a distributed system we choose to implement our brake system, using design alternative three.

Chapter 3

Real-Time System Analysis

Real-time analysis has been used as long as there have been computers. The idea is to divide every program into small subprograms, called *tasks*, and define different attributes such as worst-case execution time, C , period time, T , and many more. When these tasks have been assigned the attributes, one wants to know if they will be able to run without malfunction and if they will be able to meet their deadlines. The problem in computers, and distributed systems, is often that one has many subprograms and applications which all need to compete for the same time and even the same resources. This can lead to clashes when some programs want to start at the same time or when the tasks try to allocate a shared resource at the same time. One must therefore be able to measure these effects before the system is put into practice and this is where real-time analysis comes in.

3.1 Response-Time Analysis

There are many different ways to analyze real-time systems. The difference lies in which scheduling policy that is being used, in other words the way the priority is assigned to the tasks. There are dynamic priorities, where the priority is changed over time according to some criteria, and static priorities, the priority is determined before according to some criteria and stays that way the entire time. There are different advantages and disadvantages with both scheduling policies but in this thesis we have focused on fixed priority scheduling and with dynamic offsets. Dynamic offsets are needed when dealing with distributed hard real-time systems, and systems in which tasks suspend themselves.

3.1.1 Exact Response-Time Analysis

The main theory used in our thesis is developed in [5]. In our thesis, the real-time systems taken under consideration are composed of tasks, τ , with dynamic offsets, grouped into entities called *transactions* [7]. Every transaction, Γ , is a set of tasks with the same period and are ordered in some way, this is easy to understand if you think off Γ as a computer program with many small applications in it. More

definitions can be found in [5]. Every task with higher or equal priority in the system will contribute to the worst-case response time, $WCRT$, of the task under analysis, τ_{ab} . In order to understand how the WCRT will develop, we need to find out how every task contributes to the WCRT. Since we are dealing with tasks with offsets that can be larger than a whole period we need to consider the possibility that the critical instance (the time at which the task under analysis is activated, that gives the worst case response-time) may not be the simultaneous activation off all higher priority tasks, as is the case when all tasks are independent. Since we are working with dynamic offsets which can be larger then a whole period T_i we need to define a reduced task offset

$$\phi_{ij} = \Phi_{ij} \bmod T_i \quad (3.1)$$

The main reason for doing this is that offsets of a higher priority task, other than the one under analysis, may be changed by adding or subtracting whole periods of the latter task, without any effects on the response time of the lower priority task. This will help us in further analysis.

The first stage of the analysis is to understand how every task contributes to the WCRT. We only need to know at what time the critical instant occurs the critical instants will be derived later on. Taking any task under consideration, τ_{ij} , with phase relation between the critical instant and the task, ϕ , so that $0 \leq \phi < T_i$, we will find out the worst-case contribution on lower priority tasks depending on the relationship in size of ϕ and ϕ_{ij} . In figure 3.1 we can see four effects that will affect the outcome.

- Scenario1: τ_{ij} can be delayed by the jitter J_{ij} until task τ_{ab} busy period and with $\phi \geq \phi_{ij}$. This makes it necessary to include τ_{ij} in the analysis.
- Scenario2: τ_{ij} can be delayed by the jitter J_{ij} until task τ_{ab} busy period and with $\phi < \phi_{ij}$. This makes it necessary to include τ_{ij} in the analysis.
- Scenario3: τ_{ij} can not be delayed by the jitter J_{ij} until task τ_{ab} busy period making it unable to affect τ_{ij} WCRT.
- Scenario4: the activation of τ_{ij} starts after τ_{ab} busy period, making it unable to affect τ_{ij} WCRT.

Now that we know what is important for the WCRT we simply need to categorize τ_{ij} into the four scenarios above and use the following theorem which is given in [5].

Theorem 3.1 *Given a task τ_{ab} critical instant, t_c , and a phase relation ϕ between the arrival pattern of transaction Γ_i and the critical instant, the worst-case contribution of task τ_{ij} to the response time of τ_{ab} occurs when the activations of tasks that can coincide with τ_{ab} busy period, have an amount of jitter such that they all occur at the critical instant, and when activations of tasks that occur after the busy period of τ_{ab} have an amount of jitter equal to zero.*

The proof can be found in [5]. Here, we only sketch the proof.

ProofTasks that occur before the critical instant and that can not be delayed to the critical instant by jitter are not involved in the busy period.

For tasks that can coincide with the busy period it would be best to activate them exactly at the critical instant so that they don't miss the busy period of τ_{ab} .

For tasks that start after the critical instant it would be best to start them as soon as possible so that the probability for them to coincide with the busy period would be as big as possible. This would mean that the jitter is zero. \square

Now that we know what needs to be done to get an optimal result, we will calculate how many activations may be accumulated at the critical instant. There is a possibility that the same task will stop itself because it has experienced a jitter that coincides with the critical instant. To be able to calculate this number, n_{ij} , we need to define a variable that tells us at what time the last activation of tasks before or at the critical instant would have occurred. We define this variable as Δ which shows us the difference in time from the nearest activation of τ_{ij} and the critical instant. We now get two different results for this Δ , one for which $\phi \geq \phi_{ij}$ and one for which $\phi < \phi_{ij}$. This give us by inspection of figure 3.1

$$\Delta = \begin{cases} \phi - \phi_{ij} & \text{if } \phi \geq \phi_{ij} \\ T_i + \phi - \phi_{ij} & \text{if } \phi < \phi_{ij} \end{cases}$$

or simply:

$$\Delta = (\phi - \phi_{ij}) \pmod{T_i} \tag{3.2}$$

Now we can come up with two relations by simply examine the figure 3.1 and reminding that the first activation that may occur at or before the critical instant is t_0 .

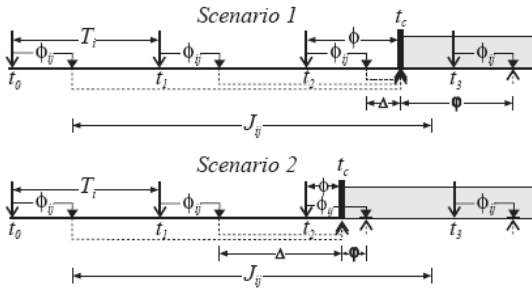


Figure 3.1. Computational model of a system composed of transactions with static offsets [5].

$$t_0 + \phi_{ij} + J_{ij} > t_c \tag{3.3}$$

$$t_0 - T_i + \phi_{ij} + J_{ij} < t_c \quad (3.4)$$

$$t_c = t_0 + (n_{ij} - 1) * T_i + \phi_{ij} + \Delta \quad (3.5)$$

Now we get two equations of n_{ij} by replacing (3.5) in (3.3) and (3.4).

$$n_{ij} - 1 \leq \frac{J_{ij} - \Delta}{T_i}$$

$$n_{ij} - 1 > \frac{J_{ij} - \Delta}{T_i} - 1$$

Since n_{ij} is an integer number we get the solution to the two above equations:

$$n_{ij} = \left\lfloor \frac{J_{ij} - \Delta}{T_i} + 1 \right\rfloor \quad (3.6)$$

By applying (3.1), we know that the worst-case contribution of the task τ_{ij} to a lower tasks busy period is n_{ij} activations of τ_{ij} at the critical instant, plus the sequence of periodic activations of the same task because of its preemption of itself starting at $T_i - \Delta$, we define this time as φ , time units after the critical instants. When we define this mathematically we get:

$$W(\tau_{ij}, \phi, t) = n_{ij} C_{ij} + \left\lceil \frac{t - \varphi(\phi)}{T_i} \right\rceil C_{ij} = \left(\left\lfloor \frac{J_{ij} + \varphi(\phi)}{T_i} \right\rfloor \left\lceil \frac{t - \varphi(\phi)}{T_i} \right\rceil \right) C_{ij} \quad (3.7)$$

This is the interference of one task on one τ_{ab} so if we want to know how all higher priority tasks contribute we simply need to add them all together:

$$W(\Gamma_i, \theta, t) = \sum_{\forall j \in hp_i(\tau_{ab})} W(\tau_{ij}, \theta, t) \quad (3.8)$$

$hp(\tau_{ab})$ is the usual all higher or equal priority tasks except τ_{ab} itself. One might wonder why we include all higher or equal priority tasks in the definition since one should never have two tasks with equal priority in static real-time systems. However, in many actual real-time systems this is the case, because of the involvement of many developers and different solutions, which makes us having to consider this as well. For distributed systems we extend this definition to:

$$hp(\tau_{ab}) = \left\{ j \in \Gamma_i \mid \text{priority}(\tau_{ij}) \geq \text{priority}(\tau_{ab}) \wedge \text{processor}(\tau_{ij}) = \text{processor}(\tau_{ab}) \right\} \quad (3.9)$$

The reason for this definition is simply the fact that tasks on other processors can not affect the task under analysis.

We want to find out the worst case contribution of an entire Γ_i to the task under analysis. By doing so we can find out the last unknown variable in our analysis ϕ , which is defined as the phase between the arrival pattern of Γ_i and the critical instant. This theorem and its proof is given in [5].

Theorem 3.2 *The worst-case contribution of transaction Γ_i to a task τ_{ab} critical instant is obtained when the first activation of some task τ_{ik} in $hp_i(\tau_{ab})$ that occurs within the busy period coincides with the critical instant after having experienced maximum amount of jitter, J_{ik} .*

Now we have a tool for determining the phase between the event arrivals and the critical instant. Note that the theorem does not say any but some, meaning that this τ_{ik} is in fact the task which started the critical instant. If we suppose that we know which task started the critical instant we can use the relation

$$\phi = (\phi_{ik} + J_{ik}) \pmod{T_i} \quad (3.10)$$

This is the same relation as was given in (3.2)

Now we can solve equation (3.8) since we know how to solve ϕ . This gives us the relation:

$$\varphi_{ijk} = \varphi(\theta)|_{\theta=(\theta_{ik}+J_{ik}) \pmod{T_i}} = T_i - ((\phi_{ik} + J_{ik}) \pmod{T_i} - \phi_{ij}) \pmod{T_i} \quad (3.11)$$

which can be simplified using the the properties of the modulus function,

$$\varphi_{ijk} = T_i - (\phi_{ik} + J_{ik} - \phi_{ij}) \pmod{T_i} \quad (3.12)$$

Since we don't know which task is the τ_{ik} we need to test all tasks in (3.8) and use the task that contributes to the highest response-time. Also we have only said that this is done for one transaction Γ_i but there can be many more transactions in the system and we need to know how all these transactions contributes to the worst response-time. This leads to an exponentially increasing number of combinations that are needed to be tested, in fact the number of variations are the number of all task belonging to $hp(\tau_{ab})$ in all transactions, plus one because of the possibility that τ_{ab} can be the one that starts the critical instant, multiplied by each other:

$$N_v(\tau_{ab}) = (N_a(\tau_{ab}) + 1) * \prod_{\forall i \neq a} N_i(\tau_{ab}) \quad (3.13)$$

notice that N_i is a function that returns the number of tasks belonging to $hp_i(\tau_{ab})$ if there are any, and a one otherwise. Each of the $N_v(\tau_{ab})$ variations has a set of tasks v , which contains one task from each transaction Γ_i . We call the task that initiates the critical instant, $v(i)$. This means that we need to do the analysis for one task in every transaction until we find which choice that gives us the biggest contribution.

Now we need to number the jobs of the task under analysis with p , having subsequent numbers ordered according to the activation time that they would have if they had not experienced jitter. Also we will use $p = 1$ to the activation of τ_{ab} that range in the interval $[0, \tau_{ab}]$. This is followed by $[T_a, 2T_a]$ where $p = 2$ as well as $p = 0$ when the activation would have occurred in the interval $[-T_a, 0]$ but was delayed because of jitter. This is convenient because we now have positive number for activations that occur after the critical instant and negative for previous jobs.

For each variation v we obtain the completion time for each of the jobs of τ_{ab} in the busy period. This time, $w_{ab}^v(p)$ is obtained by adding the execution time of τ_{ab} with the interference of all other tasks:

$$w_{ab}^v(p) = B_{ab} + (p - p_{0,ab}^v + 1) + \sum_{\forall i} W_{iv(i)}(\tau_{ab}, w_{ab}^v(p)) \quad (3.14)$$

where $p_{0,ab}^v$ is the lowest-numbered job:

$$p_{0,ab}^v = - \left\lfloor \frac{J_{ab} + \varphi_{abv(a)}}{T_a} \right\rfloor \quad (3.15)$$

We solve this in the normal iterative way where we start at zero and stop when we have two subsequent equal results. The analysis needs to be repeated for all jobs in the busy period. To be able to know for which p the analysis needs to be done, we need to calculate the length of the busy period, L_{ab}^v . We can solve this by another iterative solution where we add the contribution of interference from all other tasks in the system and the execution of τ_{ab} :

$$L_{ab}^v(p) = B_{ab} + \left\lceil \frac{L_{ab} - \varphi_{abv(a)}}{T_a} \right\rceil - p_{0,ab}^v + 1 + \sum_{\forall i} W_{iv(i)}(\tau_{ab}, w_{ab}^v(p)) \quad (3.16)$$

which is the first instant after the critical instant at which all jobs of τ_{ab} of higher priority tasks have been completed. The maximum value of p can now be calculated as the number of busy periods in a period:

$$p_{L,ab}^v = \left\lceil \frac{L_{ab} - \varphi_{abv(a)}}{T_a} \right\rceil \quad (3.17)$$

By calculating $L_{ab}^v(p)$ we have enough tools to calculate the entire completion time. However, in order to obtain the global response time we need to subtract the time the instant at which the external event that activated the transaction arrived, $\varphi_{abv(a)} + (p - 1)T_a - \Phi_{ab}$. This leads us to the following expression:

$$R_{ab}^v(p) = w_{ab}^v - (\varphi_{abv(a)} + (p - 1)T_a - \Phi_{ab}) = w_{ab}^v - \varphi_{abv(a)} - (p - 1)T_a + \Phi_{ab} \quad (3.18)$$

This has to be done for all p in the range of $[p_{0,ab}^v, p_{L,ab}^v]$ and for all choices of v . The amount of calculations that needs to be done is exponentially growing and in most cases, as in ours, too many to be dealt with. This is the main problem with the exact analysis and the main reason why we need to extend the analysis into a more usable one in which we decrease the number of calculations that needs to be done. There is an easy way to do this, and at the same time get results that are not too pessimistic, which we will describe in the next section.

3.1.2 Upper-Bound Approximation for Worst-Case Analysis

The easiest way to reduce the number of calculations is to do some kind of approximation in the analysis. Since we don't know which task τ_{ik} must be used to

create the worst-case busy period we needed to test *all* combinations and use the one that contributed the most. If we think about it, the maximum contribution $W_{ik}(\tau_{ab}, w)$ of higher priority tasks is equal to or less, than the maximum contribution in every transaction. This means that we could create an upper-bound approximation by defining a function, $W_i^*(\tau_{ab}, w)$ that finds the maximum of all possible inferences that could be caused by considering each of the tasks of Γ_i and use it as the one that originates the busy period:

$$W_i^*(\tau_{ab}, w) = \max_{\forall k \in hp_i(\tau_{ab})} W_{ik}(\tau_{ab}, w) \quad (3.19)$$

We could use this on every transaction but in order to introduce less pessimism, we will not use that function for the transaction that belongs to the task under analysis but instead try all belonging to $hp_a(\tau_{ab})$ plus the task itself. This means that, the number of possibilities will simply be the number of tasks in the same transaction Γ_a plus the task under analysis itself τ_{ab} . Now we can calculate the worst-case response time by testing all choices, c , and saving the worst one. For a critical instant created by τ_{ac} , the worst case response time is determined by:

$$w_{abc}(p) = B_{ab} + (p - p_{0,abc} + 1) + W_{ac}(\tau_{ab}, w_{abc}(p)) + \sum_{\forall \neq a} W_i^*(\tau_{ab}, w_{abc}(p)) \quad (3.20)$$

where the first activation that occurs at the critical instant corresponds to:

$$p_{0,abc} = \left\lceil \frac{J_{ab} - \varphi_{abc}}{T_a} + 1 \right\rceil \quad (3.21)$$

The length of the busy period :

$$L_{abc}(p) = B_{ab} + \left\lceil \frac{L_{abc} - \varphi_{abc}}{T_a} \right\rceil - p_{0,ab}^v + 1 + C_{ab} + W_{ac}(\tau_{ab}, L_{abc}) + \sum_{\forall \neq a} W_i^*(\tau_{ab}, w_{ab}^v(p)) \quad (3.22)$$

with the maximum activation that occurs at the critical instant:

$$p_{L,ab}^v = \left\lceil \frac{L_{ab} - \varphi_{abv(a)}}{T_a} \right\rceil \quad (3.23)$$

Now we get the global worst case response time by, as we did in the exact analysis, subtracting from the completion time the instant at which the associated event arrived:

$$R_{abc}(p) = w_{abc}(p) - \varphi_{abc} - (p - 1)T_a + \Phi_{ab} \quad (3.24)$$

Then we simply take the worst of all the response times obtained:

$$R_{ab} = \max_{\forall c \in hp_a(\tau_{ab}) \cup b} \left[\max_{p=p_{0,abc} \dots p_{L,abc}} (R_{abc}(p)) \right] \quad (3.25)$$

3.1.3 Analysis for Tasks with Dynamic Offsets and Distributed systems

The main advantage with introducing dynamic offsets is that we can model distributed systems and systems that suspend themselves. The reason for this is that both distributed systems and systems that suspend themselves will be suspended by either, as in the case with distributed systems, waiting for a message to arrive or when tasks waits for an earlier task to finish. The effects will be the same because when we model distributed systems we need to model the messages that are being sent between the processors as tasks, with a short additional blocking time because there is no preemption on the buses [4]. This would also work for point to point lines and other scheduling strategies for the messages in the systems. Now we have the possibility of modeling our system where we group chains of events in transactions with tasks by knowing that each task, τ_{ij} , is activated by the completion of the previous task in its transaction τ_{ij} .

In order to be able to use the analysis above, we need to make some changes. Theorem (3.2) shows that the activation phase represented the minimum interval of time that could exist between the arrival of the external event and the activation of the associated task. Also, the jitter term represents the maximum amount of delay that the task activation could suffer, counted from the arrival of the external event plus the task's offset. For that reason, we can model the case in which the offsets may vary as a special case of a system with static offsets, simply by defining a equivalent static offset Φ'_{ij} and associated equivalent jitter term J'_{ij} for each task in the following way:

$$\begin{aligned}\Phi'_{ij} &= \Phi_{ij,min} \\ J'_{ij} &= J_{ji} + \Phi_{ij,max} - \Phi_{ij,min}\end{aligned}\tag{3.26}$$

With these changes we can now use the same analysis that was used for the static analysis to calculate the worst-case response times. The problem here is that in most systems where task offsets can change dynamically, their minimum and/or maximum value are dependent on the response times of the previous tasks in the transaction, as it is in our distributed model. We know that, when we speak of distributed systems, every transaction is a chain of tasks that all have to wait for the previous task to finish. The first task in the transaction is activated by an external event, let us assume that this has no jitter. Now we can model the behavior of a distributed, triggered and suspending, system by giving a jitter and offset term of zero for the one that initiates the transaction and with the following values for every other task:

$$\begin{aligned}\Phi'_{ij} &= \Phi_{ij,min} = R_{ij-1}^b \\ J'_{ij} &= J_{ji} + \Phi_{ij,max} - \Phi_{ij,min} = R_{ij-1} - R_{ij-1}^b\end{aligned}\tag{3.27}$$

where R_{ij-1}^b is a lower bound to the best-case response time of task τ_{ij} , and R_{ij-1} is an upper bound for the worst-case response time. In our distributed system

we do not have the same behavior as was described in [5]. Instead of the tasks suspending themselves until the previous task in the transaction is ready, the tasks run periodically without suspending themselves. However, they will not have the desired information until the message has been sent throughout the transaction and reached the task under analysis so the behavior is very similar. The main difference is that we have forced offsets, $\Phi_{node,ij}$, so that the messages need to arrive before the task is ready to use the information. To model this we need to know the offsets and we need to consider the fact that if the message comes too late it can not be sent until the next period. The modified equations are:

$$\Phi_{ij} = R_{ij-1}^b = \max \left(\sum_{k=1..j-1} C_{ik}, \Phi_{node,ij} \right) \quad (3.28)$$

To calculate the offset for task τ_{ij} we have to find the earliest starting time of a new execution of τ_{ij} that begins after the best case response-time of task τ_{ij-1} and use this value as offset. There is more detailed information on this in Chapter 4.1.

Now we see the main difficulties with the analysis, the response times are dependent on the task jitters and the task jitters are dependent on the response times. The solution to this problem is to start with a Jitter term of zero and iterating over the analysis until a stable solution is found. This analysis can be found in [7]. We start with a Jitter term of zero and obtain a response time of every task. Then we use these response times and recalculate the jitter terms and do the analysis all over until we obtain two successive iterations with:

$$R_{ij}^{(n-1)} = R_{ij}^{(n)} \quad \forall i, \forall j \quad (3.29)$$

The analysis will converge to the worst-case response times of the system under study, if possible with the scheduling policy being used. Otherwise the response times will be equal to infinity.

Chapter 4

Experiments

To be able to compute the worst-case response times of our systems we needed to come up with a correct model of the tasks, nodes and the messages which are sent over the CAN networks. When we have done this we can calculate the WCRT and use this in our simulation model to verify that the calculations are correct as well as if our proposed solution to the synchronization problem is correct. This chapter describes our task model and the different experiments we have done.

4.1 Task Model

To be able to use the WCRT analysis we need to model our distributed system by modeling the task model. The first thing we had to do was to find which tasks that depend on each other and which messages that need to be delivered before the next task can begin executing. These tasks and messages form transactions where they are ordered in the sequence in which they will be executed. The tasks on the computer nodes and the messages sent over the networks are modeled in the same way except that tasks on the computer nodes can be preempted during execution which messages can not be.

The next thing we had to do was to find the offset for each task in each transaction and which tasks that originate from the same original task, to avoid that a task preempts itself. This was done by creating an original matrix with structures in Matlab containing the name of the original task so that one can ask of which task τ_{ij} originated from. For a system in which tasks are triggered, when the task before in the same transaction has finished its execution, the offset of task τ_{ij} can be calculated as the best case response time of task τ_{ij-1} according to (3.28)

$$\Phi_{ij} = R_{ij-1}^b = \max \left(\sum_{k=1..j-1} C_{ik}, \Phi_{node,ij} \right) \quad (4.1)$$

The offset for the first task in each transaction is set to zero.

In our model the nodes can not start executing as soon as the message arrive, because the nodes only handle messages at the start of their execution, but instead they have to get the message before the start of a new execution in order to send the message further in the chain. This is shown in figure 4.1, where Task1 sends a message to the other tasks. The dotted line shows when the message has been sent from Task1 and the arrows show when the other tasks can compute the message.

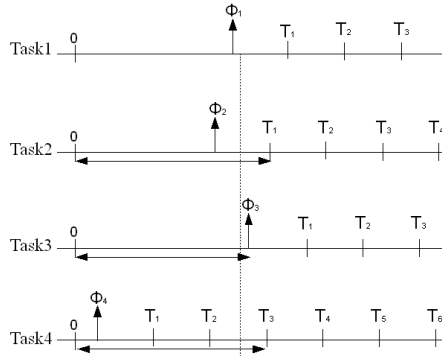


Figure 4.1. Task offsets.

In this case the offsets will be a little bit harder to find. To calculate the offset, $\Phi_{ij,min}$, for task τ_{ij} we have to find the earliest starting time of a new execution of τ_{ij} that begins after the best case response-time of task τ_{ij-1} and use this value as offset. This was done by comparing the starting time of each execution of τ_{ij} with the worst case time at which it could receive the message. If the message arrives before the first execution of τ_{ij} the offset will be equal to the offset of the node, in other case the offset will be calculated as the offset of the node plus the number of periods needed to cover the worst case response time of the message.

The last thing we had to do before calculating response-times was to determine the blocking times for messages sent over the network. The blocking time for a message is equal to the longest time a lower priority message can preempt the message and thereby equal to the longest transmission time of lower priority messages. This is done for all the network nodes. All other data that we needed for the response-time analysis, execution times and priorities, were already defined in the Simulink models.

Figure 4.2 shows our task model for the brake system.

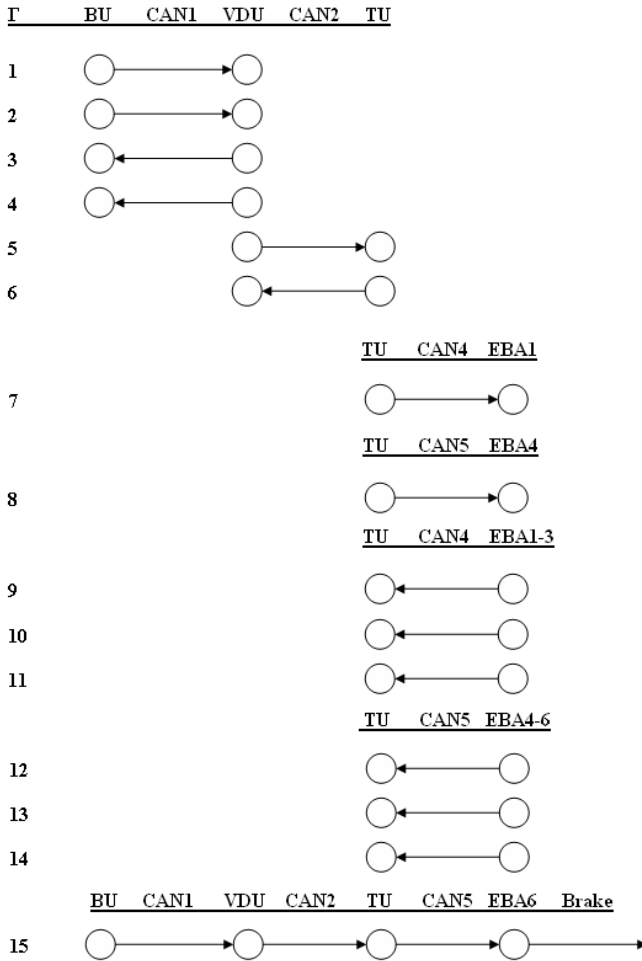


Figure 4.2. The task model shows the transactions and how the tasks communicate with each other.

The first 14 transactions model all messages that are being sent between nodes in the system every period. The 15:th transaction model the communication path from brake unit to the electrical brake actuators, for the messages that contain the new information about the brake signal. Since all the tasks and messages in transaction 15 can be described by the tasks and messages in the other transactions, we have assigned the same name in the original matrix to avoid preemption. The deadline of each task, D_{ij} , is set equal to its period, T_{ij} .

With this task model we could now perform the experiments needed to analyze the system performance under different working conditions.

4.2 Results

The purpose with our experiments was to find which frequencies that are needed on the nodes to meet the timing constraints on the response times of the system. When we have found the desired response times, we can later use them to synchronize different messages or signals in the system. For the first model that we developed, the warning lights model, we had to keep in mind that the lights should blink synchronously with a variance under 0.01 [s] in order to fulfill the requirement given by BAE Systems Hägglunds. We would of course want to have an even better synchronized model and if our assumptions are correct we will have just that. For the brake system our goal was to get a response time under 0.6 seconds, from a change in pedal position to the activation signals from the brake actuator nodes. We will begin to study the systems without any disturbances and when we have found proper frequencies on the nodes we will study how disturbances affect the system response.

Warning Lights Experiments

Here we have not tried to calculate the exact WCRT in order to get the best possible synchronization but instead tried to verify that one could synchronize a distributed system by telling every node to blink at a predefined time that is longer than the worst time it could possibly take for the message to be sent through the system. This was done by simulating many different input signals and use the worst one of the response times after adding a bit more time to be sure that we get a time that is bigger than the WCRT but not too big. At 50 [Hz] we get the behavior given in figure 4.3.

We see that the lights are not synchronized and the demand of a maximum variance of 0.01 [s] is not satisfied. When we introduce the synchronization time that was given by simulating the system we get the synchronized behaviour as shown in figure 4.4. Here we have managed to make the lights synchronized and with a less variation than we needed to clear the 0.01 [s] demand.

Brake System Experiments

In order to find the requested response times for the brake system it was not necessary to take all nodes into account. Since the nodes E3 and E5 controls the brake light and therefore only receives messages from VDU, we could ignore them when calculating the response time for the brake signal. The data rate on all buses was set to 500 kbits/s in every experiment. In the first experiment we used the settings stated in Table B.1, in the appendix, for the task model shown in Figure 4.2. To calculate a WCRT of the system we run the digits in the analysis program. The result of the longest calculated WCRT is 0.030887. This time is well under the law given requirement of 0.6[s] as well as our desired time of 0.1[s]. The WCRT of the system is a bit to optimistic since it is the time from which BU knows the change in pedal position and is able to run directly. However, there is a possibility that the change in pedal happened after BU's reading phase so that BU needs to

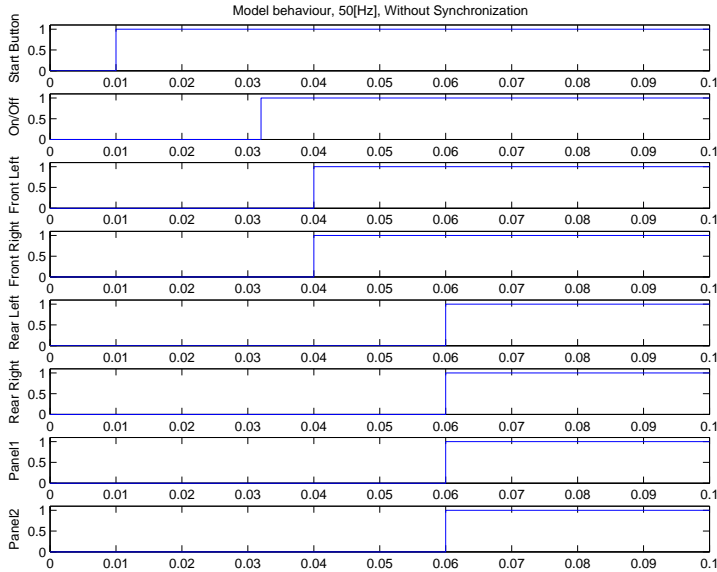


Figure 4.3. Unsynchronized Warning lights model

wait a whole period until it can use the information and hence send it further. Even with this time, 0.01[s], added we are clearly below both requirements.

Now we try to lower the frequency to 50[Hz] and run the simulation again. We get the value of 0.07887 which when adding the period of the system gives us the true WCET of 0.09887. This time is very close to our desired maximum WCRT so we can compare the results given by simulating the model with different signal inputs. In our model it is quite easy to simulate the WCRT because of the fixed execution time of the tasks on the nodes. We get the results in Figure 4.5 where we see that our calculated value exceed the simulated one. We continue to decrease the frequencies down to 10[Hz]. In Table 4.1 we can see the data describing our simulated response times together with the calculated ones. The table shows that a frequency of 50[Hz] is the lowest frequency where we can still manage our desired time of 0.1[s] and 10[Hz] is the lowest frequency with which we can still manage the law given requirement.

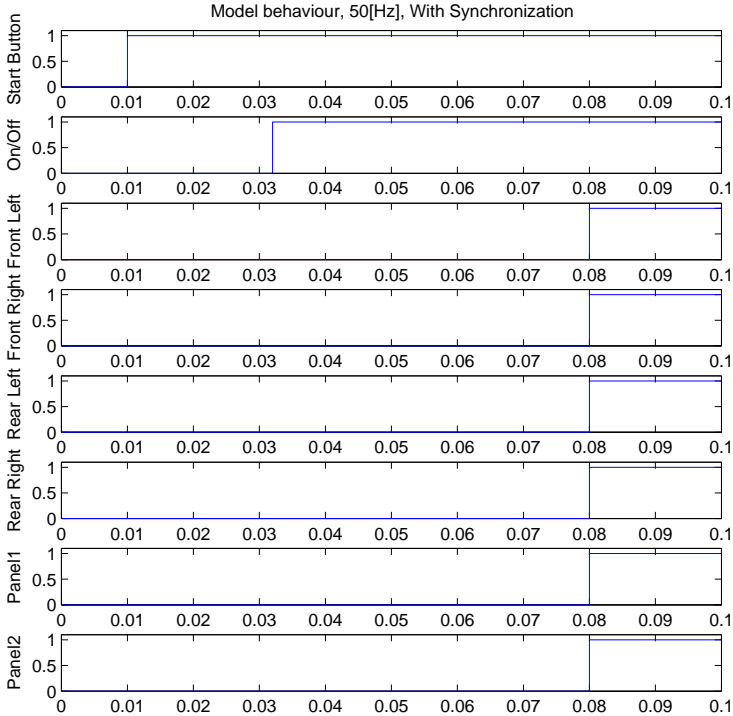


Figure 4.4. Synchronized Warning lights model

Frequency [Hz]	Simulated WCRT [s]	Calculated WCRT [s]
100	0.0358	0.04887
50	0.0725	0.09887
10	0.3658	0.49887
2	1.8325	2.49887

Table 4.1. Simulated and calculated response times

We can see in figure 4.6 what happens to the system when we do not include our synchronization algorithm in the system. The mechanical brake signal is activated before the electrical brake signal and thus we need to apply the synchronization algorithm to the system. We can also see that the electrical brake signal appears a bit fluctuating. This is because the electrical brake signal in figure 4.6 shows the total electrical brake force. The electrical brake actuators are thus not activated at the same time either and the wheels will not brake at the exact same time.

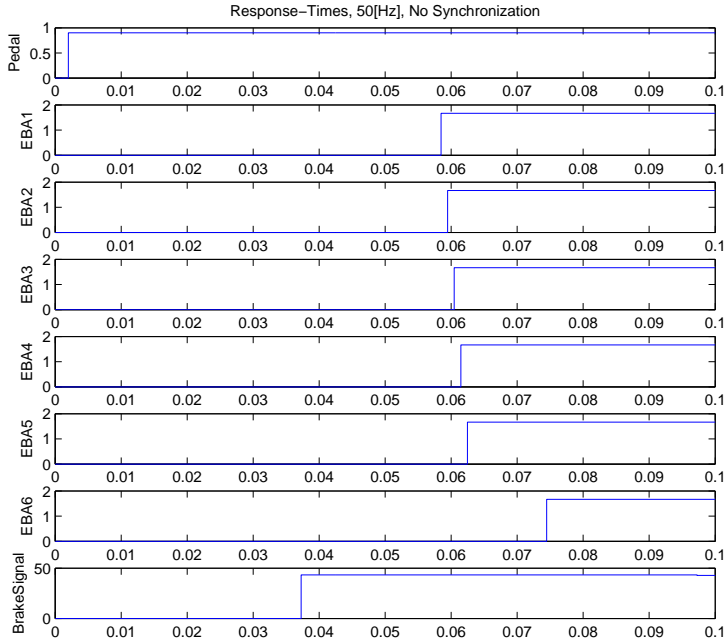


Figure 4.5. Plot of the signal responses to a change in the pedal position

Further we can see that the vehicle slows down monotonic anyway so the fact that the signal aren't perfectly synchronized does not affect the braking capability of the vehicle that much. One can only speculate how it would effect the real vehicle since we have done some simplifications but one get the sense that it would probably not be that much.

When we use the synchronization algorithm we get the results shown in figure 4.7. All the wheels brake at the same time and the mechanical and electrical brake signals are activated concurrently.

The Response To Disturbances

To simulate disturbances on the network we simply added two more nodes in our model, one node that sends disturbance messages to the CAN buses and one that recieves the messages. Since the messages sent on the buses can not be preempted and the time it takes to send a message is so short (ca 0.1ms) it will require very many disturbance messages (about 100 messages) to see any effect on the response times.

Disturbances on the nodes, for example tasks with another application on the same nodes, can be simulated with periodic tasks which only take time on the

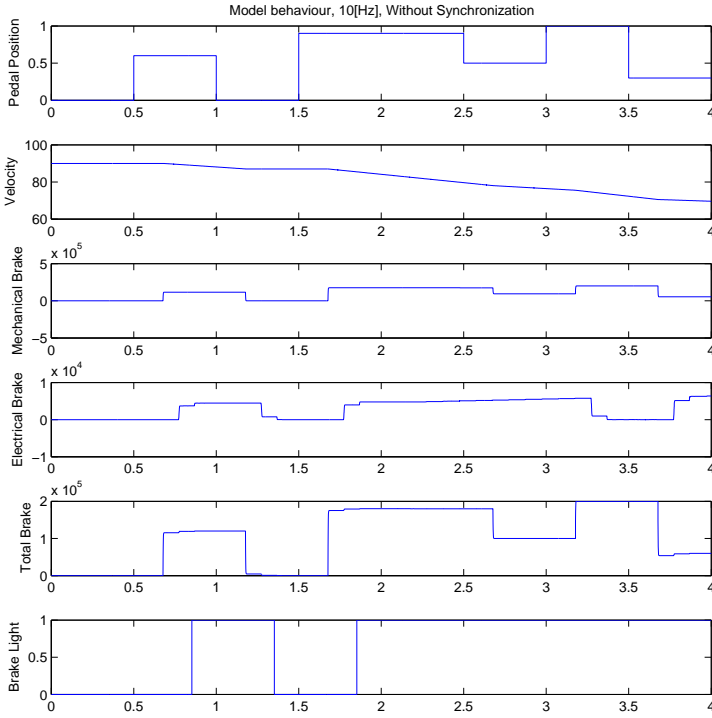


Figure 4.6. Brake system without synchronization

nodes. These disturbances can have a larger impact on the response times and need to be considered, if present in the real system.

With the disturbances added in the system one have to update the task model with tasks and transactions describing the disturbances.

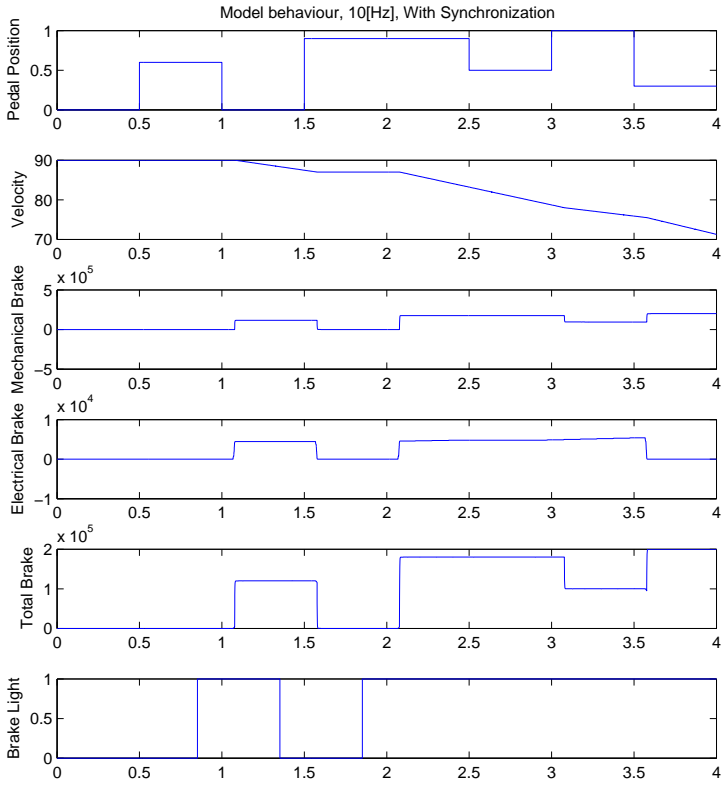


Figure 4.7. Brake system with synchronization

Chapter 5

Methodology Summary

5.1 True-Time Implementation

The Matlab/Simulink toolbox TrueTime is well suited for Simulating real-time systems and we chose to use this toolbox together with Stateflow to build our models. Since True-Time is able to use one input parameter, we decided to send in Matlab structures into all the nodes with the necessary information. The structs are stored in an initiation file in Matlab. In order to make the handling of the program as smooth as possible, we have divided the initiation file in two. One is where the different design variables that the user should be able to change lie, such as period on the nodes, node offsets, synchronization time and vehicle speed. The other one contains initiation parameters for the nodes to be able to communicate with each other such as network connections.

All the nodes have two unique True-Time files, one initiation file and one code file. In the initiation file we use the standard True-Time approach of defining a function with an input that we call "data". This input is used when we define the tasks on the node with either "ttCreatePeriodicTask" for the periodic tasks on the node or "ttCreateTask" for tasks that are triggered. Since our nodes only contain one task that runs periodically, we use one "ttCreatePeriodicTask" for each "init" file with inputs from the "data" input as well as the respective "code" name.

In order to handle the network we need to use "ttCreateInterruptHandler" and the "ttInitNetwork" functions. There have to be one combination of these for each outgoing channel to the network, you can read more in the True-Time manual [11]. To make the network act in a way so that all messages are saved and queued as soon as they arrive at the node but not taken cared of until the periodic task starts its periodic execution, we used the "ttCreateTask" to create one task for each network that take the incoming message and puts it in a queue without taking any time (exectime = 0). This message is later read by the periodic task and depending on the amount of messages in the queue it will take an increasingly amount of time to read them by the periodic task. To make the code general we have used "Reciever" files that invoke "ttCreateJob" with the name of the "ttCreateTask" that is used to put the network's messages in a queue. We need to have different names of the

```

function TU_Init(data)

%kernel:
%ttInitKernel(nbrInp, nbrOutp, prioFcn, cs_oh)
ttInitKernel(0, 0, 'prioFP');

%tasks:
%ttCreatePeriodicTask(name, offset, period, priority, codeFcn, data)
%ttCreateTask(name, deadline, priority, codeFcn, data)

%Periodic on the node
ttCreatePeriodicTask('TractionUnit', data.Offset, data.Period, data.Prio_TU, 'TUCode', data);

CANdata.Node = 'TractionUnit';
CANdata.Port = 2;
ttCreateTask('Net', 1, 2, 'CANCode', CANdata);
CANdata.Port = 3;
ttCreateTask('Net1', 1, 2, 'CANCode', CANdata);
CANdata.Port = 4;
ttCreateTask('Net2', 1, 2, 'CANCode', CANdata);

%network & interrupt handlers:
%ttCreateInterruptHandler(name, priority, codeFcn)
%ttInitNetwork(network, nodenumber, handlername)
ttCreateInterruptHandler('NetworkHandlerCAN2', 1, 'Reciever');
ttCreateInterruptHandler('NetworkHandlerCAN3', 1, 'Reciever1');
ttCreateInterruptHandler('NetworkHandlerCAN4', 1, 'Reciever2');
%network interface
ttInitNetwork(2, 1, 'NetworkHandlerCAN2');
ttInitNetwork(3, 1, 'NetworkHandlerCAN3');
ttInitNetwork(4, 1, 'NetworkHandlerCAN4');

```

Figure 5.1. The Matlab code for BU-Init file

"Reciever" files because every "ttCreateInterruptHandler" needs different names of every internal "ttcreateTask". This means that we need a maximum of three "Reciever" files in total because there will never be a node that is attached to more than three networks.

The other file that is needed for every node is the "code" file. This is a function that returns [exectime data] and is divided into different cases. In the SEP, every node is working in the same fashion with CAN, I/O read, application and last I/O-write. Every node starts with calling the same function "MessageHandler", see A.2 in the appendix, which, depending on the node calling it, reads messages from the message queue and returns exectime depending on the number of messages.

When the reading is done, the node needs to send messages to the nodes in its presence. Here it is important to divide messages that will be sent over the same line in to different cases. Hence, we divide every message so that every message takes the same amount of time so that the solver doesn't need to be using to small

```
function [exectime, data] = Reciever(seg, data)

ttCreateJob('Net');
exectime = -1; %End
```

Figure 5.2. The Matlab code for the reciever file

```
function [exectime, data] = CANCode(seg data)

NodeData = ttGetData(data.Node);
Msg = ttGetMsg(data.Port);
NodeData.MsgQueue = [NodeData.MsgQueue Msg];
ttSetData(data.Node, NodeData);

exectime = -1; %END
end
```

Figure 5.3. The Matlab code for the CANcode file

steps. More info on how messages are sent over network in True-Time can be read in the True-Time manual [11].

Now we come to the stage where we use our developed method with True-Time, Stateflow and Simulink in collaboration. We start by sending data to the enabled Stateflow model and at the same time we enable it. The exectime is put to $\text{period}/4$ or $\text{period}/3$ depending on how big we can put it without overriding the period with execution time. Now it is important that the enabled Stateflow model has the same setting so that the time of the enabling will coincide with the True-Time signal, making the model do one "tick". In the next case following we simply read the outputs of the model and disable it. This case needs $\text{exectime} = 0$!

The last case is where we do I/O-out which means that the simulated analog signals are sent, this can for instance be blinking or braking.

5.2 Task Model

To be able to use the analysis made in [5] one have to model the system in a way so that the analysis in Chapter 3 will give satisfying results. This is described in Chapter 3 and Section 4.1. Also, we need to create the task model for the system, including all tasks and transactions. This is done by looking at every message sent between nodes and creating transactions of the relation. Every node is marked as a ring and every CAN connection is marked by an arrow. When one has done every message in this way one must also do the same thing for the entire chain that the message travels from start to finish. Figure 4.2 shows how we created the taskmodel for the brake system.

5.3 Synchronization

The solution to this problem is quite intuitive. We want to know the longest time for a message to arrive from the BU. This was done in Chapter 3. When we know the longest time for a message to arrive to all nodes, we can use this time by including it in the message that is sent so that the receiver knows when the blinking system should blink or the brake system should brake in order to be synchronized. This means that when BU gets the message from the brake pedal it sends a message over the network so that both the mechanical brake node and the electrical node knows at what time they are going to brake and how hard. The same thing happens for every change in pedal position which means that all nodes will be synchronized at every change.

It will however be impossible to exactly synchronize the nodes because of the way they are built. The nodes are given different internal offsets which means that they are a bit different in time so that they will not be able to execute the signal at a specific time. This problem could be solved by including some kind of continuous check in the nodes instead of having fixed functionality in them that execute in a predefined order and time so that they can be completely synchronized.

Chapter 6

Conclusions

We have investigated how to model a distributed system and how to synchronize its nodes. Early in the development of a complex system it is important to be able to test the communication between nodes. By testing in an early stage in the development one can save money by finding faults so that one doesn't need to go back and fix them later on in the process, when the errors can be hard to trace and correct.

The main conclusion of this thesis is that one can use TrueTime and Stateflow at the same time to model distributed systems. This enables the programmer to reuse large Stateflow models and test them when considering real-time aspects as well as network aspects while communicating with other models. Another application of a distributed system model is to find where the bottle necks lie in the communication. This means that one can find at which frequencies the nodes are working and at which frequencies they are not. Also, this could be used to decide the hardware architecture.

One way to synchronize distributed systems is to calculate the WCRT and let the faster nodes wait until the slowest node is ready. By calculating the WCRT one knows the longest response time possible for all nodes in the entire system which makes it possible to synchronize the system without first building a large model of the system. This means that for simply synchronizing a system one does not need to build a model but instead calculating the WCRT and let faster nodes wait this time before doing the desired application.

Chapter 7

Future Work

- In this thesis we have limited ourselves not to include redundancy. In a distributed system one would like to know how the system responds when one node (or more) malfunctions and what to do about it. It would be possible to let nodes cooperate and take over other nodes calculations if a node brakes or sending information a different route. There has been a lot of research on the subject and one can find a lot of books and other thesis's dealing with the issue.
- Another interesting issue is to be able to handle other schedulers than fixed priority. In [6] one can read how J.Palencia develops an analysis for handling Earliest deadline first, EDF. This scheduling is optimal when one wants all the task to execute before its given deadline but it is more difficult to analyze and can behave strangely when tasks start to miss their deadlines. This approach is not used in the current SEP but it would be possible to implement in a future version or in a subsystem in the SEP.
- The fact that by having non periodic tasks one could optimize the utilization and get a more dynamic system. There have been articles written on the subject so it would be possible. One article concerning the subject is [9]. This would probably require more than just a dynamic scheduling but also a new approach in the overall modeling of the system.
- We have not been able to test the simulated results on actual hardware in the SEP. One would like to be able to verify the results in this thesis with the real hardware, thus verifying even further the simulated results. This should be possible to examine in a future thesis at BAE-Systems Hägglunds where the execution, network, and receiving times are measured on hardware and then tested on the actual SEP hardware and later evaluate with the simulated results.
- The simplifications done in this thesis such as the vehicle model and the braking system could be expanded. There are many ways of getting a more sophisticated vehicle model that interacts with the weights in a more dynamic

and natural way. One really need to take into consideration that when the vehicle turns the weight will shift from the center to the side as well as the shifts in weights when riding a bumpy road.

Bibliography

- [1] Electric Brakes. <http://www.messier-bugattiusa.com/img/pdf/elecbrake.pdf>.
- [2] Hydraulic Brakes. <http://library.thinkquest.org/c007574/data/hydraulic.htm>.
- [3] Jukka Mäki-Turja Hans Hansson Henrik Thane Jan Gustafsson Christer Norström, Kristian Sandström. *Robusta realtidssystem*. Addison-Wesley, 2001.
- [4] J.J. Gutiérrez García and M. González Harbour. Increasing schedulability in hard real-time systems. *Proceedings of the 7th Euromicro Workshop on Real-Time Systems*, pages 99–106, June 1995.
- [5] M.G. Harbour J. Palencia. Schedulability analysis for tasks with static and dynamic offsets. *in: Proceedings of the 19th IEEE Real-Time Systems Symposium, Madrid, Spain*, 1998.
- [6] M.Gonzalez Harbour J.C. Palencia. Response time analysis of edf distributed real-time systems. 2005.
- [7] J. Clark K. Tindell. Holistic schedulability analysis for distributed hard real-time systems. *Microproc. Microprog.* 50, 14(3):342–351, 1994.
- [8] J.J. Gutiérrez García Palencia Gutiérrez and M.González Harbour. On the schedulability analysis for distributed hard real-time systems.
- [9] Alfons Crespo Sergi Saez, Joan Vila. On accepting aperiodic transactions in distributed systems. 2000.
- [10] Stateflow. <http://www.mathworks.com/products/stateflow/>.
- [11] TrueTime. <http://www.control.lth.se/truetime/>.

Appendix A

Matlab Code

A.1 The TU-Code File

```

function [exectime, data] = TUCode(seg, data)
switch seg,
case 1,
    %Recieve messages:
    [exectime, data] = MessageHandler(data, data.Period/3);

case 2,
    %Send messages:
    if length(data.ResponseTime > 0)
        msg1.type = 'RequestedElectricBrake';
        msg1.value = data.RequestedBrake(end,1:3);
        msg1.time = data.ResponseTime(end);
        ttSendMsg([3 0], msg1, 8, 1);
        msg4.type = 'RequestedElectricBrake';
        msg4.value = data.RequestedBrake(end,4:end);
        msg4.time = data.ResponseTime(end);
        ttSendMsg([4 0], msg4, 8, 1);

        if length(data.ResponseTime) == 1
            data.RequestedBrake = [];
            data.ResponseTime = [];
        else
            data.RequestedBrake = data.RequestedBrake(1:end-1,1:end);
            data.ResponseTime = data.ResponseTime(1:end-1);
        end
    end

else
    msg1.type = 'NoChange';
    msg1.value = 0;
    msg1.time = 0;
    ttSendMsg([3 0], msg1, 8, 1);
    msg4.type = 'NoChange';
    msg4.value = 0;
    msg4.time = 0;
    ttSendMsg([4 0], msg4, 8, 1);
end
end

```

A.2 The Messagehandler File

```

function [exectime,data] = MessageHandler(data,time) %
RecievedMsg = data.MsgQueue;
exectime = 0;
if length(RecievedMsg) > 0
    tmp = 0;

    for i = 1:floor(min(length(RecievedMsg), time/(65/500000)))

        if cmp(RecievedMsg(i).type, 'RequestedBrake')
            data.RequestedBrake = [RecievedMsg(i).value; data.RequestedBrake];
            data.ResponseTime = [RecievedMsg(i).time data.ResponseTime];
        elseif cmp(RecievedMsg(i).type, 'RequestedElectricBrake')
            temporary = RecievedMsg(i).value;
            if cmp(data.Reciever,'EBA1') || cmp(data.Reciever,'EBA4')
                data.BrakeQueue = [temporary(1) data.BrakeQueue];
                data.BrakeTimes = [RecievedMsg(i).time data.BrakeTimes];
            elseif cmp(data.Reciever,'EBA2') || cmp(data.Reciever,'EBA5')
                data.BrakeQueue = [temporary(2) data.BrakeQueue];
                data.BrakeTimes = [RecievedMsg(i).time data.BrakeTimes];
            elseif cmp(data.Reciever,'EBA3') || cmp(data.Reciever,'EBA6')
                data.BrakeQueue = [temporary(3) data.BrakeQueue];
                data.BrakeTimes = [RecievedMsg(i).time data.BrakeTimes];
            end
        elseif cmp(RecievedMsg(i).type, 'BrakeLight')
            data.Active = RecievedMsg(i).value;
        elseif cmp(RecievedMsg(i).type, 'BrakeCapacity')
            data.BrakeCapacity = RecievedMsg(i).value;
        elseif cmp(RecievedMsg(i).type, 'Weights')
            data.Weights = RecievedMsg(i).value;
        elseif cmp(RecievedMsg(i).type, 'Capacity')
            data.Capacity = RecievedMsg(i).value;
        elseif cmp(RecievedMsg(i).type, 'CapacityEBA1')
            data.Capacity(1) = RecievedMsg(i).value;
        elseif cmp(RecievedMsg(i).type, 'CapacityEBA2')
            data.Capacity(2) = RecievedMsg(i).value;
        elseif cmp(RecievedMsg(i).type, 'CapacityEBA3')
            data.Capacity(3) = RecievedMsg(i).value;
        elseif cmp(RecievedMsg(i).type, 'CapacityEBA4')
            data.Capacity(4) = RecievedMsg(i).value;
        elseif cmp(RecievedMsg(i).type, 'CapacityEBA5')
            data.Capacity(5) = RecievedMsg(i).value;
        elseif cmp(RecievedMsg(i).type, 'CapacityEBA6')
            data.Capacity(6) = RecievedMsg(i).value;
        end
        tmp = tmp + 1;
        exectime = exectime + 65/500000;
    end

    if tmp == length(RecievedMsg)
        data.MsgQueue = [];
    else
        %Shorten the MsgQueue so that only the ones not handled remains
        data.MsgQueue = data.MsgQueue(tmp+1:end);
    end
end

```


Appendix B

Tables

B.1 Simulation Parameters

Transaction	Task	Original	C	B	T	D	Priority	Offset
Γ_1	τ_{11}	task1	0.005	0			3	0
	τ_{12}	task6	0.00013	0.00013	0.01	0.01	1	0.005
	τ_{13}	task2	0	0			1	0.00513
Γ_2	τ_{21}	task1	0.0075	0			3	0
	τ_{22}	task6	0.00013	0	0.01	0.01	4	0.0075
	τ_{23}	task2	0	0			1	0.00763
Γ_3	τ_{31}	task2	0.005	0			3	0
	τ_{32}	task15	0.00013	0.00013	0.01	0.01	2	0.005
	τ_{33}	task1	0	0			1	0.00513
Γ_4	τ_{41}	task2	0.0075	0			3	0
	τ_{42}	task15	0.00013	0.00013	0.01	0.01	3	0.0075
	τ_{43}	task1	0	0			1	0.00763
Γ_5	τ_{51}	task2	0.005	0			3	0
	τ_{52}	task7	0.00013	0.00013	0.01	0.01	1	0.005
	τ_{53}	task3	0	0			1	0.00513
Γ_6	τ_{61}	task3	0.006667	0			3	0
	τ_{62}	task8	0.00013	0	0.01	0.01	2	0.006667
	τ_{63}	task2	0	0			1	0.006796667
Γ_7	τ_{71}	task3	0.006667	0			3	0
	τ_{72}	task9	0.00013	0.00013	0.01	0.01	1	0.006667
	τ_{73}	task4	0	0			1	0.006796667
Γ_8	τ_{81}	task3	0.006667	0			3	0
	τ_{82}	task9	0.00013	0.00013	0.01	0.01	1	0.006667
	τ_{83}	task13	0	0			1	0.006796667
Γ_9	τ_{91}	task4	0.006667	0			3	0
	τ_{92}	task17	0.00013	0	0.01	0.01	2	0.006667
	τ_{93}	task3	0	0			1	0.006796667
Γ_{10}	τ_{101}	task10	0.006667	0			3	0
	τ_{102}	task18	0.00013	0	0.01	0.01	2	0.006667
	τ_{103}	task3	0	0			1	0.006796667
Γ_{11}	τ_{111}	task11	0.006667	0			3	0
	τ_{112}	task19	0.00013	0	0.01	0.01	2	0.006667
	τ_{113}	task3	0	0			1	0.006796667
Γ_{12}	τ_{121}	task5	0.006667	0			3	0
	τ_{122}	task20	0.00013	0	0.01	0.01	2	0.006667
	τ_{123}	task3	0	0			1	0.006796667
Γ_{13}	τ_{131}	task12	0.006667	0			3	0
	τ_{132}	task21	0.00013	0	0.01	0.01	2	0.006667
	τ_{133}	task3	0	0			1	0.006796667
Γ_{14}	τ_{141}	task13	0.006667	0			3	0
	τ_{142}	task22	0.00013	0	0.01	0.01	2	0.006667
	τ_{143}	task3	0	0			1	0.006796667
Γ_{15}	τ_{151}	task1	0.005	0			3	0
	τ_{152}	task6	0.00013	0.00013			1	0.005
	τ_{153}	task2	0.005	0			3	0.009
	τ_{154}	task7	0.00013	0.00013	0.01	0.01	1	0.014
	τ_{155}	task3	0.006667	0			3	0.021
	τ_{156}	task9	0.00013	0.00013			1	0.0276667
	τ_{157}	task13	0.00987	0			3	0.029
	τ_{158}	task14	0	0			1	0.03887