A SCHEDULING ALGORITHM FOR CAN BUS

WANG YONG

(B.Eng., Tsinghua University)

A THESIS SUBMITTED
FOR THE DEGREE OF MASTER OF ENGINEERING
DEPARTMENT OF MECHANICAL ENGINEERING
NATIONAL UNIVERSITY OF SINGAPORE

2003

To my parents

Acknowledgements

I owe my deepest gratitude and appreciation to my supervisor, Professor Teo Chee Leong, for his instructive advices and guidance during my research work.

I feel extremely fortunate to have worked in Control and Mechatronics Lab, ME, NUS. Many talented colleagues here give me help, encouragement and support. I greatly appreciate their generosity in devoting their time to help me with this project. Specially, I would like to express my thanks to Dr. Chu Wei, who spends lots of time to discuss with me about the construction of a simulation platform, to Chen Yinghe, for his help with programming issues, to Chen Xiaoming, for his advices in scheduling simulation, to Yu Weixiao, who clarifies some basic conceptions for me. I am also sincerely grateful for the friendship and companion from Boy Eng Seng, Duan Kaibo, Shu Peng, Wang Xiaopeng, Zhang Han, Zhang Lihua, Zheng Xiaoqing, Zuo Jing and friends from Switzerland, Roger Gassert and Frederic Mani.

I would like to express my gratitude to all stuff in Control and Mechatronics Lab for their help on facility support so that the project may be completed smoothly.

Last, but not least, I would like to thank my family for their continuous love and support through my student life.

Contents

Summary

This thesis discusses the issues of communication scheduling and priority mapping in CAN bus. In recent years, fieldbuses are widely adopted as the communication network in distributed control systems and other industrial applications. Meanwhile, scheduling algorithms for fieldbuses are studied to improve the performance of fieldbus. Priority-driven scheduling algorithms (fixed-priority or dynamic-priority) are popular in fieldbus systems. However, due to the limited number of priority levels in practical fieldbus, the performances of scheduling processes can not be as effective as theoretical analysis under the condition of unlimited priority levels. Although centralized scheduling schemes can avoid this problem, they are not flexible and can not utilize native advantages of CAN bus.

This project proposes a new priority assignment scheme, the Dynamic Local Priority (DLP) scheme, to solve the problem of limited number of priority levels for CAN bus. In order to calculate and assign priorities to messages, a local priority list (Network Node List, NNL) is maintained in every node on the CAN bus to store the information of the message set. And the nodes can update the local NNL at optimized time intervals and synchronize with other nodes on the bus. Theoretical analysis shows that DLP scheme has performance closer to ideal non-preemptive EDF scheduling than the schedulers implemented by other non-preemptive algorithms.

And a new effective schedulability condition for non-preemptive EDF algorithm is proposed, which can precisely test the schedulability of any given message set. In addition, general scheduling algorithms in fieldbus, especially in CAN bus, are

analyzed and summarized. Most popular scheduling algorithms, including Rate Monotonic (RM), Deadline Monotonic (DM) and Earliest Deadline First (EDF) are studied.

Finally, a platform-independent scheduling simulator software is developed so as to compare the performances of different scheduling algorithms for CAN bus. And the simulation result shows that both the new schedulability condition and DLP scheme are effective and have advantages over previous methods. When the bus load is heavy, the DLP scheme can get about 20% better performance than traditional fixed-priority scheduling algorithms.

Nomenclature

δ	variance
μ	mean
ν	the bitrate of the bus
τ_i	message i
τ_{bit}	the time to transmit a bit on the bus
B_i	the worst case blocking time of message τ_i
C_i	computation time or transmission time of message i
CAN	Controller Area Network
D_i	relative deadline of message i
d_i	absolute deadline of message i
DCS	Distributed Control System
DLP	Dynamic Local Priority scheme
DM	Deadline Monotonic algorithm
ECU	Electronic Control Unit
EDF	Earliest Deadline First algorithm
I_i	the worst case interference time of message τ_i
J_i	jitter time of messag i
L	the length in bit of a message
LAN	Local Area Network
LRT	Latest Release Time algorithm
LSF	Least Slack Time First algorithm
MIT	Minimum Interval Time of a sporadic message
NNL	Network Node List

O_i	phasing offset of message i
OSI	Open System Interconnection
R_i	response time of message i
r_i	release time of message i
RM	Rate Monotonic algorithm
T_i	the period of message i
w_i	the worst case queuing delay

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, the background and the purpose of the project are discussed. An overview of communication scheduling is also given, and finally the structure of the thesis is listed.

1.1 Background

Real-time systems have been developed for many years. In the earlier days, electric devices were mostly driven by analog components, which were always large and inaccurate. And the prices of components were high and they were not easy to maintain. With progress and achievements in microprocessor and semiconductor industry, analog devices were replaced by digital devices gradually, and heavy mechanical controllers were changed to electronic controllers. At the same time, systems are becoming more and more complex, involving large number of devices, sensors, actuators and controllers. The automobile industry is a good example. The first car, if we can call it a car, was simple and crude. It was composed of only several basic components. But now in modern cars, we can find very complex wire connections, various electronic devices, electronic control units to control the engine, the brake, suspension and many other hydraulic and mechatronics systems. Every system is relatively independent and could exchange information with oth-

ers. They combine to form a whole complex real-time system, a distributed control system (DCS).

When a car is running, the electronic control unit (ECU) for the engine will compute the volumes of air and fuel that are needed in every stroke and send commands to the engine. Another ECU monitors the status of the chassis, the pressure of tyres, the condition of the road, the force sent to the brake. In fact, there are hundreds of sensors and actuators in a modern vehicle, which are under the control of one or even several ECUs. When working, such intelligent devices exchange information constantly. This example reflects the infrastructure of most modern DCSs: several controllers work independently, meanwhile communicate with one another to maintain the running of the whole system.

1.1.1 Fieldbus

Since 1980s, microcontrollers become more powerful while their prices become lower, thus making it possible to develop computer network systems with high speed communication capacity. In order to implement information integration and comprehensive automation of an enterprise, an industrial low-level network is necessary. And it should be low cost with high reliability and able to support multi-point digital communication. Fieldbus [28] [32] is developed to meet those practical requirements.

Fieldbuses are special local area networks to connect distributed controllers, intelligent sensors and actuators. A twisted pair of wires is the most popular transmission media of fieldbuses while in some circumstances coaxial cable and optical fiber are used. Most of fieldbuses use standard or a simplified model derived from OSI (Open Systems Interconnection) Reference Model. Compared with traditional LANs (such as Ethernet), they have some characteristics more suitable for real time control and communications in DCS. For instance, the head of a message on fieldbuses is normally much smaller, which decrease the cost of bandwidth waste.

Meanwhile, because of the simplification of network model, fieldbuses has shorter jitter time and less overhead than LANs. And some fieldbuses use special protocols to improve the communication efficiency, such as CAN bus.

There are numerous kinds of fieldbus standards and techniques. And many fieldbus standards are designed for a specific purpose or applied in certain industrial fields. The foundation of SP50a fieldbus committee of the Instrument Society of America (ISA) in 1980s indicated the start of the development of fieldbus standard. At the beginning of 1990s, some companies created the Inter-operable Systems Project (ISP) and other major SP50 companies formed the WorldFIP (Factory Instrumentation Protocol) standard group [32]. Later the ISP and WorldFIP joined to form the Fieldbus Foundation and developed a new fieldbus technology, Foundation Fieldbus (FF), which defines low speed (H1, 31.25kbps) and high speed transmission (H2, 1Mbps). Nowadays, it has been a standard that is widely supported in the United States and Asia. At the same time, other fieldbus standards were developed in Europe. Profibus, which has three main types: FMS (Flexible Manufacturing Systems), DP (Distributed Peripherals) and PA (Process Automation), is a standard in Germany and popular in Europe.

Another important fieldbus standard is the CAN bus (Controller Area Network), which was developed by Robert Bosch GmBH [5]. The CAN bus was designed for automobile industry initially. It is a multi-master architecture with asynchronous transmission. The CAN bus uses a non-destructive bitwise arbitration, which can successfully avoid communication collisions in the normal network protocols, such as IEEE 802.3 Ethernet Protocol. It is widely accepted because of its acceptable speed, high noise immunity, low cost per node, and relative degree of determinism.

There are some other fieldbus standards. For example, HART (Highway Addressable Remote Transducer) is suitable for instruments. LON (Local Operating Network) is used to interconnect components of widespread functionality and factory automation. Because of the low cost, high reliability and powerful features,

fieldbuses are widely used in many industrial fields, such as integrated automation, process control and measurement [30] [31]. Selection of fieldbus standard depends on practical restrictions and requirements.

1.1.2 Scheduling

When an industrial network is set up, a very important issue is how to control the message transmission on the bus. This is the task of message scheduling [24]. Broadly speaking, scheduling can be classified as clock-driven, priority-driven and a hybrid of both (see Chapter 3 for details). In practical distributed real-time systems, priority-driven scheduling schemes are widely adopted. In a uniprocessor environment, task scheduling is the critical part of an operating system. And such systems always adopt clock-driven or weighted clock-driven scheduling algorithms [33]. However in distributed environments, due to the difficulty to synchronize clock among distributed nodes, clock-driven scheduling is rarely adopted. In stead, priority-driven scheduling algorithms are very popular in distributed systems [1].

Priority-driven scheduling algorithms can have either fixed (static) or dynamic priority. Rate Monotonic and Deadline Monotonic algorithms are excellent fixed-priority scheduling schemes while Earliest Deadline First algorithm is typical dynamic-priority scheduling. And due to the limited bits of fieldbus for priority, the limited number of priority levels could not guarantee the thorough utilization of scheduling algorithms. Hence, an effective priority assignment is the necessary for the high performance of the network.

1.2 The project and contribution

CAN bus has been widely used in various industrial fields for about 10 years. However, to meet the requirement of increasingly complex and flexible system, the CAN bus may have to be modified. The generic protocol of CAN bus is fix-

priority scheduling. In order to thoroughly utilize current CAN controllers in use, especially to improve the message transmission capacity of CAN bus, it is beneficial to introduce dynamic-priority scheduling into CAN bus. Meanwhile, CAN 2.0A can only provide limited bits for priority mapping. So an effective method to assign priority is necessary for such dynamic-priority scheduling algorithms in CAN bus.

The main work and contribution of this project is:

1. a new priority assignment scheme, the Dynamic Local Priority (DLP) scheme, is proposed to solve the problem of limited number of priority levels, so as to improve the transmission performance of CAN bus, without compromising the current CAN protocol.

   In DLP scheme, a local priority list (Network Node List, NNL) is maintained in every node on the CAN bus to store the information of the message set and carry out the real-time priority update. Further more, nodes can update the local NNL at optimized time intervals and synchronize with other nodes on the bus. By this way, the limited priority levels of the given message set can be utilized sufficiently, therefore improves transmission performance;

2. a new effective schedulability condition for non-preemptive Earliest Deadline First algorithm (EDF) is proposed, which can precisely test the schedulability of any given message set. In addition, general scheduling algorithms in fieldbus, especially in CAN bus, are analyzed and summarized. Most popular scheduling algorithms, including Rate Monotonic (RM), Deadline Monotonic (DM) and EDF are studied;

3. a software simulation platform for CAN bus is developed, which can simulate the communication of the network and carry out the schedulability test. The architecture of the simulation platform is open, so that it is easy to add more implementations of other scheduling algorithms to enable simulation to be carried out.

Theoretical analysis shows that DLP scheme has performance close to optimal non-preemptive scheduling. And the new schedulability condition can give a necessary and sufficient schedulability test. Simulation results show that both the new schedulability condition and DLP scheme are effective and have advantages over previous methods. In particular, when the CAN bus load is very heavy, the DLP scheme can get about 20% better performance than traditional fixed-priority schedulers.

1.3 Organization of the thesis

This thesis is organized as follows: Chapter 1 introduces the background and the purpose of this thesis. Chapter 2 gives an overview of the CAN bus, including its background, standard and application. Chapter 3 contains the discussion on scheduling algorithms, including RM/DM, EDF and their derivatives. Chapter 4 gives the analysis of current schedulability conditions for non-preepmtive EDF scheduling algorithm, and an new schedulability condition and corresponding algorithm are proposed. Chapter 5 contains discussion on priority inversion and the DLP scheme for CAN bus. Chapter 6 gives the implementation of simulation and the analysis of experiment results. Finally, Chapter 7 summarizes this thesis.

Chapter 2

Controller Area Network

This chapter introduces Controller Area Network (CAN) bus. Topics of communication model, arbitration mechanism and error handling of CAN are discussed.

2.1 Background

In the field of computer communication, protocols of RS-232 and CCITT V.24 were widely used. However, they are peer-to-peer data communication standard with low speed, and lack support to high level functions and operations between CPUs. In addition, complex and large-scale applications require a great number of sensors, actuators and controllers, which are distributed and far away from each other. Therefore, a communication system is needed to meet the requirements of low cost and high reliability in various working environments.

The Controller Area Network (CAN) [5] [10], a kind of fieldbus, was originally developed by Robert Bosch in 1980s. It is a serial data communication protocol to handle data exchanges between controllers and testing instruments in modern automotive vehicles. Normally, dedicated communication media are required to interconnect different control components because of the complexity of the control functions implemented by engine control units, anti-lock brakes (ABS), anti-skid controls and other controllers. However, with the increase in complexity and quan-

tity of control components, the traditional peer-to-peer connection by wires and connectors are not adequate. The invention of CAN provides an efficient way to reduce the wiring complexity and makes it possible to interconnect several devices using a common communication medium.

In automotive electronics, CAN could be found in engine control components, sensors, ABS, including some cheap electronic devices, such as electric windows and lamp clusters, to replace wiring harness otherwise required. As a matter of fact, CAN was adopted and migrated from vehicles into machinery and automation. Applications of CAN range from high-speed networks to low-cost multiplex wiring. It has been applied widely in medical instrumentation, elevator controls, public transportation systems and industrial automation control components.

After the invention of CAN bus, its development has never stopped. In 1991, the CAN specification version 2.0 was released. The most important modification in this version was the extension of the message format of CAN bus so as to meet the requirement of assignment of message identifiers in a large address range. CAN 2.0 Part A follows the definition in CAN specification 1.2 so that the message format in CAN specification 1.2 is still valid, while CAN 2.0 Part B describes both standard and extended message formats and the extended format can contain nearly 2^{29} different message identifiers. Nowadays, CAN bus is widely used in Europe and many deviations based on CAN bus are developed to extend it to more application fields.

2.2 Basic features

CAN bus is a kind of serial communication network. Because of some novel and unique designs, compared with normal communication buses, the data communication of CAN is outstanding in reliability, flexibility, and real-time characteristics. The basic features of CAN are:

- Multi-master and peer-to-peer communication: On the CAN bus, any node

can be the master or slave and no host address is needed. By means of frame filtering, it is simple to implement broadcast and peer-to-peer communications.

- Message with priority: On CAN bus, messages are encapsulated into frames with priorities to meet real-time requirements of different messages. The message with the highest priority could be transmitted in no more than $134\mu s$.

- Non-destructive bus arbitration: When several nodes send messages to the CAN bus simultaneously, messages with low priorities will abort automatically, while the message with highest priority will be transmitted successfully without any communication collision. Therefore the time cost in bus arbitration is greatly reduced. And the network could endure even very heavy network load without breakdown.

- Flexible transmission rate and distance: The transmission rate of CAN bus is configurable. The longest distance could reach $10km$ if the bitrate is under $5kbps$. And if the transmission rate reaches up to $1Mbps$, the longest distance will be $40m$.

- Large network capacity: The maximum number of nodes in a CAN bus system depends on the number of bits assigned to the node identifier. There are 2032 different frame identifiers (CAN 2.0A), while with CAN 2.0B extended standard, the number of frame identifiers is 532676608. However, in practice, the number of nodes is much smaller because of the limitations of the driving capacity of hardware. Normally the number of nodes on CAN bus is less than 110.

- Short data length: The short data lengths (no more than 8 bytes in data field) of CAN messages could reduce the transmission time and disturbance during transmitting.

- Error detection and correction: With CRC error detection and error handling

mechanisms,the probability of non-detected damaged frames is less than 4.7×10^{-11} in CAN bus.

- Flexible communication medium: The communication medium of CAN could be twisted wiring pair, multicore or coaxial cable or optical fibre. Network designers have flexible choices in different environments.

To achieve design transparency and implementation flexibility, CAN has been subdivided into different layers according to the ISO/OSI Reference Model:

- The Data Link Layer
  - The Logical Link Control (LLC) sublayer
  - The Medium Access Control (MAC) sublayer
- The Physical Layer

The above definition is adopted in CAN Specification 2.0 Part B. In Part A, the LLC sublayer and the MAC sublayer are described as the Object Layer and the Transfer Layer, respectively. Figure 2.1 [5] shows the layered structure of CAN and corresponding functions.

The main functions of the LLC sublayer is to provide service for data transmission and remote data request, frame acknowledging, overload notification and recovery management. The MAC sublayer represents the kernel of the CAN protocol. It passes received messages to the LLC sublayers and accepts messages to be transmitted from the LLC sublayer. The MAC sublayer is responsible for message framing, arbitration, ackowledgements, error detection and signalling. It is not allowed to redefine or modify the characteristics of the MAC sublayer.

The Physical Layer defines the electrical characteristics and how signals are actually transmitted. It deals with the descriptions of bit timing, bit encoding and synchronization. However, the Physical Layer is not defined in the CAN specification, and this makes it flexible to choose different physical medium according to the requirements of individual environments.

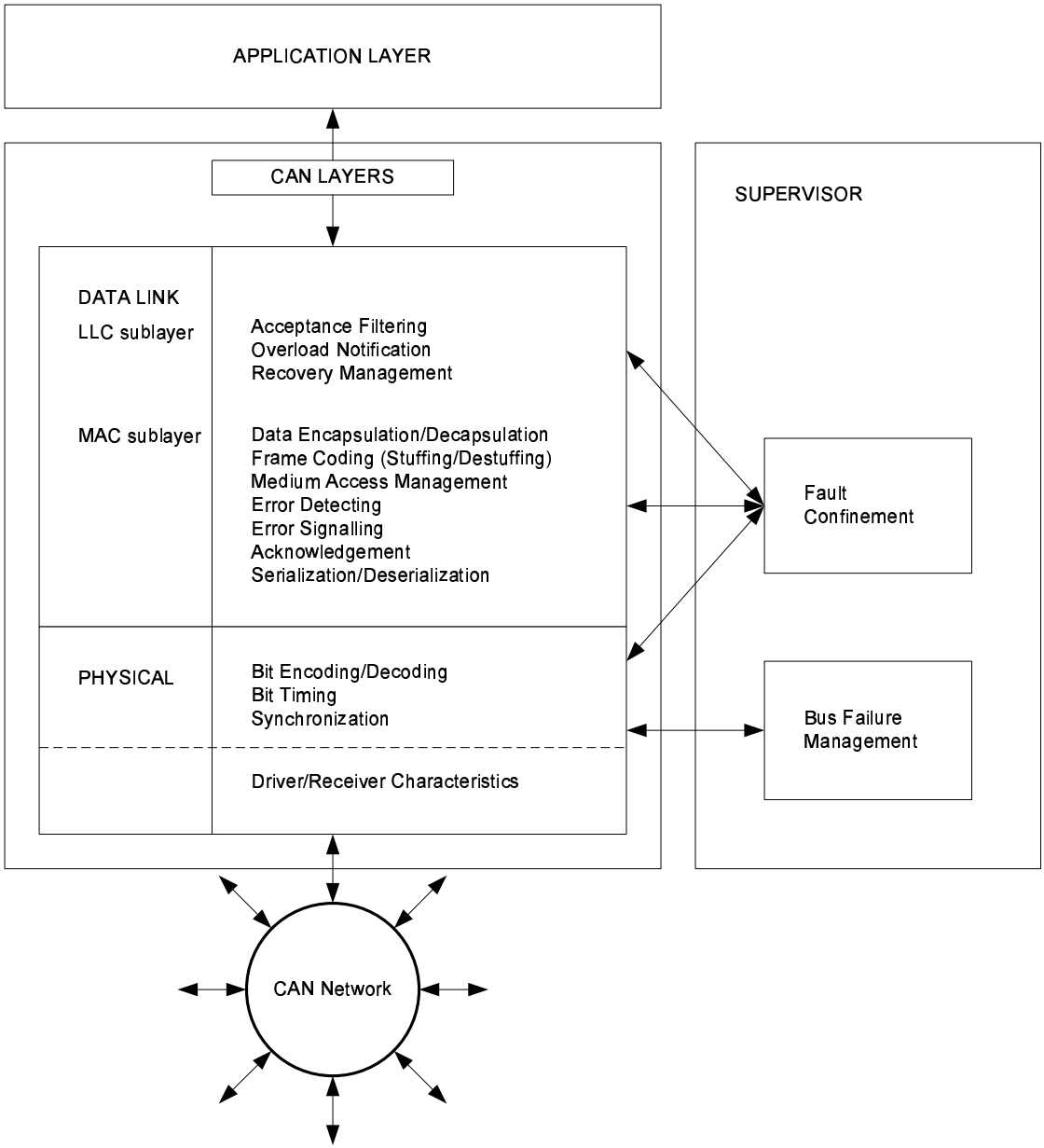

Figure 2.1: CAN layers

2.3 Fundamental specifications

2.3.1 Communication model

The basic concept of CAN protocol is similar to the producer/consumer model in the networking world, in which one node produces data on the bus and other nodes consume. But there is one significant difference between CAN and other fieldbus solutions: in the communication model of CAN, all nodes on the bus are equivalent.

No bus master or arbitrator is required.

When data is transmitted on the bus, no node has the host address. Instead, every message has a unique identifier in its data content to distinguish it from others. The identifier of one message has two functions: the first is to define the message content; and the second is to represent its priority, which determines the result of arbitration process when transmitting.

When a node wants to transmit data to the bus, it will send the data to be transmitted and an identifier to the CAN controller. The CAN controller then encapsulates data and the identifier into one or several frames and sends them to the buffer of the transmitter. Once the node gain the access to the bus, data in the buffer will be transmitted to the bus and all other nodes on the bus will become receivers. These nodes will test received frames to determine whether the data is what they want to accept or not, based on the identifiers of the frames. All frames they do not need will be discarded.

Because of the communication model described above, CAN can not only implement the peer-to-peer communication, which means one single node accepting data, but also perform broadcast and synchronized communication, which means multiple nodes can accept the same frame using one single transmission. And the advantage of this event-triggered transmission can keep the load utilization on the bus low.

2.3.2 Frame types and formats

Two types of complementary logical values could be transmitted on the CAN bus: *dominant* and *recessive*. The *dominant* is prior to the *recessive*, which means that when dominant and recessive bits are sent to the bus by different nodes simultaneously, the final value on the bus will be dominant. For example, in case of a wired-AND implementation of the bus, the dominant level would be represented by a logical '0' and the recessive level by a logical '1'. In the CAN specification, no

definitions of the electrical characteristics of bus medium devices (such as electrical voltage, light) are provided. Suggestions to device electrical characteristics of CAN bus based on twisted wiring pair are given in ISO 11898:1993.

There are two different frame formats in CAN communication.† The difference between them is the length of the identifer field. The standard format uses 11 bits to represent the identifier while the extended format uses 29 bits. We will show the detail below.

Four frame types exist in a message transfer of CAN protocol. The types and corresponding functions are listed below.

- **Data frame**: to carry data from a transmitter to the receivers;
- **Remote frame**: Sent by a bus node to request the transmission of the Data frame with the same identifier;
- **Error frame**: Sent by any node to detect a bus error;
- **Overload frame**: To provide an extra delay between the preceding and the succeeding Data or Remote frames.

Figure 2.2 shows the format of a CAN frame, which could be a data frame or a remote frame. And both those two types of frames could be in the standard format or the extended format.

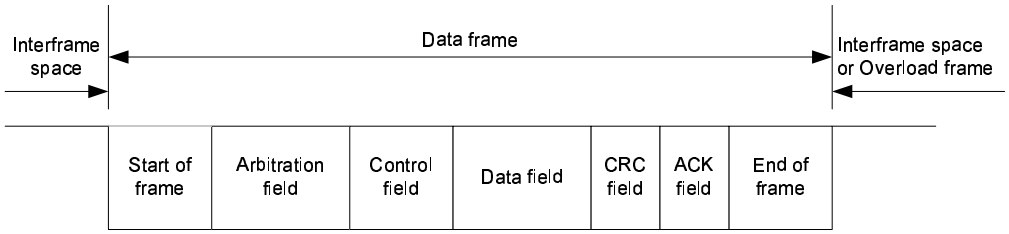

Figure 2.2: Data frame

†In the remaining part of this thesis, we suppose that all frame formats are consistent with CAN specification 2.0B, In addition, wired-AND is applied, which means logical '0' represents *dominant* level and logical '1' represents *recessive* level.

The Start of Frame (SOF), a single dominant bit, marks the beginning of Data frames and Remote frames. And its leading edge could be a signal to synchronize the bus nodes.

SOF is followed by the Arbitration field. The format of the Arbitration field is different for the standard format and the extended format. Figure 2.3 shows the details. Almost all new CAN controllers can support both the standard format and the extended format. Some of them support only the standard format, but are 2.0B passive devices, which means they can tolerate other devices transmitting frames in the extended format and still work correctly.

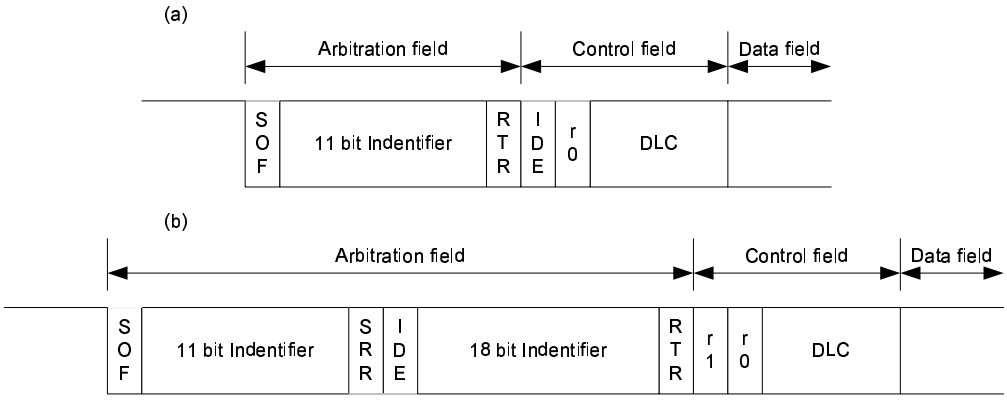

Figure 2.3: The difference between the standard format and the extended format. (a) The standard format; (b) The extended format

The RTR bit (Remote Transmission Request) is also a part of the Arbitration field. In Data frames the RTR bit is dominant. In Remote frames the RTR bit has to be recessive.

The SRR bit (Substitute Remote Request) in the extended frames is always a recessive bit, which replaces the position of RTR bit in standard frames. Therefore, this bit could help to solve the collision when a frame in the standard format and another frame in the extended format with the same base ID (the first 11 bits of the identifier) are sent simultaneously: the standard frame has higher priority level.

The IDE bit (Identifier Extension bit) belongs to the Arbitration field for the Extended format and is recessive. But it belongs to the Control field for the

Standard format and it is dominant.

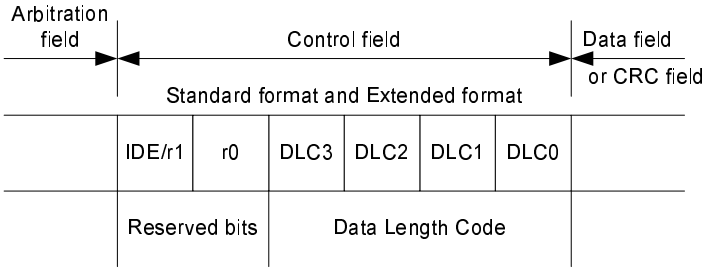

Figure 2.4: The format of the Control field

Figure 2.4 shows the format of the Control field in standard frames and extended frames. The Control field has 6 bits. For the Extended format, the first two bits (r_1 and r_0) are reserved, which means receivers could accept either dominant or recessive bits of them. For the Standard format, the first bit is IDE bit, dominant only. The other four bits are Data Length Code.

Following the Control Field is the Data field[1] that can be from zero to eight bytes in length.

The CRC (Cyclic Redundancy Code) field contains a 15-bit CRC sequence and a CRC delimiter. In order to carry out the CRC calculation, a polynomial is defined, the coefficients of which are given by the destuffed bit stream consisting of SOF, the Arbitration field, the Control field, the Data field (if present) and the 15 lowest coefficients are 0. This polynomial is divided (modulo-2) by the generator-polynomial:

$$X^{15} + X^{14} + X^{10} + X^8 + X^7 + X^4 + X^3 + 1$$

The remainder of this polynomial division is the CRC sequence. The algorithm could be implemented as follows:

```
; CRC_RG(14, 0): A 15 bit shift register
; NXTBIT: The next bit of the bit stream
;
CRC_RG = 0;
REPEAT
    CRCNXT = NXTBIT XOR CRC_RG(14);
```

[1]The Remote frame has no Data field.

```
        CRC_RG(14:1) = CRC_RG(13:0);
        CRC_RG(0) = 0;
        IF CRCNXT THEN
            CRC_RG(14:0) = CRC_RG(14:0) XOR 0x4599H;
        ENDIF
    UNTIL (CRC sequence starts or ERROR occurs)
```

After transmitting the last bit of the Data field, the content of `CRC_RG` will be transmitted as the CRC sequence.

The ACK (acknowledge) field consists of two bits: the ACK slot and the ACK delimiter. The ACK slot in a message from a transmitter is recessive. When a receiver has received the message correctly[2], it will superscribe the recessive bit of the transmitter by a dominant bit. The detection of this dominant bit by the transmitter means that the message is transmitted correctly and is accepted by at least one node. Another part of the ACK field, the ACK delimiter, is a recessive bit.

The end of frame field (EOF) indicates the termination of the frame. In a Data frame and a Remote frame, it is a flag sequence consisting of seven recessive bits.

2.3.3 Arbitration mechanism

CAN is a particular kind of carrier sense multiple access (CSMA) network. And its arbitration mechanism is similar to that in the IEEE Standard 802.3 [35], which is for 1-persistent CSMA/CD LAN. What CAN employs is a so-called carrier sense multiple access with collision avoidance (CSMA/CA), which enforces a clear medium access policy based on the priority of the messages to be transmitted.

The nodes that wish to transmit would keep monitoring the bus. Once they detect that the bus is idle, they may start to transmit their messages from the message identifier, bit by bit. At the same time, every transmitter compares the level of the bit transmitted with the level monitored on the bus. If these levels are equal the transmitter may keep transmitting. As mentioned previously, if a

[2]The CRC sequence is matched.

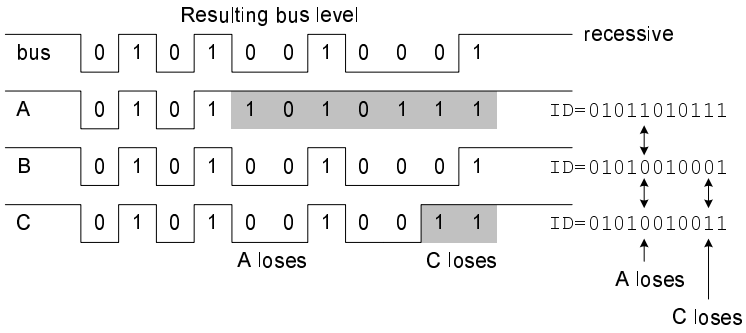

Figure 2.5: The arbitration mechanism in CAN

dominant bit and a recessive bit are sent to the bus simultaneously, the resulting bus value is a dominant bit. Therefore, if a transmitter that sends a recessive bit, but detects a dominant level on the bus (situation where A and C lose in Figure 2.5), it loses arbitration and must abort transmitting any further information immediately. What the lost node could do is to keep listening the bus until the bus becomes idle again. Then it may send its message for another round of arbitration. The arbitration process goes on and eventually only the message with the highest priority (lowest identifier value) wins the arbitration.

The arbitration mechanism of CAN guarantees that neither information nor time is lost. The node with the highest priority will continue to transmit without any interruption. On the contrary, in the CSMA/CD used by Ethernet, if a collision occurs on the bus, all colliding nodes have to terminate the transmission, wait for a random period of time, and repeat the whole process all over again. There is no random waiting in the arbitration process of CAN bus, which gives CAN a very predictable behavior. And because of the avoidance of collision, the CAN protocol can lead to very efficient utilization of the bus bandwidth.

2.3.4 Error handling and fault confinement

CAN bus has a network wide data check and error detection. When a message is being transmitted on the bus, all nodes check it for errors even if they may not need that message. If they receive the message correctly, they send a dominant bit in

the ACK slot to notify the transmitter that the transmission is correct. Otherwise an error signal will be sent to the transmitter.

There are five different error types:

- ***Bit Error:*** A node that is sending a bit on the bus also monitors the bus. When the detected bus value is not equal to the intended value, a *bit error* occurs. There are two exceptions: a dominant bit is detected when sending a recessive bit of the Arbitration field or sending the ACK slot.

- ***Stuffing Error:*** Bit stuffing is a type of encoding whereby a bit of opposite level is inserted into the bit stream when five consecutive bits of equal level are transmitted on the CAN bus. The stuffing bits are removed at the receiver end before the message is processed. Bit stuffing could help receivers on the CAN bus to re-synchronize. The detection of six consecutive equal bit levels in a message generates a *stuffing error*.

- ***CRC Error:*** Every receiver calculates the 15-bit CRC code when receiving a message and compares it with the received CRC field in the message. If they are not equal, a *CRC error* occurs.

- ***Form Error:*** If the receiver finds that there are one or more illegal bits in a predefined field of the received frame, a *form error* occurs.

- ***Acknowledgement Error:*** If a transmitter does not nonitor a dominant bit during the ACK slot, an *acknowledgement error* occurs. This error means that an error may be detected by receivers or the ACK slot has been corrupted or that no receivers exist on the network.

When a node detects an error condition, it signals this by transmitting an Error frame. The transmitter could react accordingly. CAN protocol has very efficient mechanism to handle errors and fault confinements. Every node on the bus may be in one of three states: *error active*, *error passive* and *bus off*. And every node has two counts: *transmit error count* and *receive error count* to keep

track of the number of errors detected during transmission and reception of frame, respectively. The value of the counts effects the state of a node. More details about error handling of CAN can be found in [5].

2.4 Summary

In this chapter, an overview of Controller Area Network is given. After a brief introduction of the background and development of CAN bus, we introduce the fundamental principles of CAN including network layer model, message format and error handling. In addition, some other hardware-related features of CAN can be found in Appendix B. We use MSCAN12 module of Motorola 68HC12BC32 microcontroller as the example to analyze the buffer storage, message filtering and bus timing of CAN bus.

In the next chapter, an in-depth discussion of the scheduling algorithms and their application in CAN bus will be given.

Chapter 3

Scheduling Algorithms

This chapter gives an overview of communication scheduling algorithms. Several priority-driven scheduling algorithms are studied, including their corresponding schedulability tests. Their implementations in CAN bus are introduced too.

3.1 Background and problem formulation

In this section, the definitions of messages in network are given. The problems to be solved are proposed.

Scheduling is derived from the design of operating systems [33]. In a modern uniprocessor operating system, multiprogramming is common. Several or even tens of tasks may run simultaneously. One essential task of the system is to arrange a feasible schedule to guarantee the running of all possible tasks, allocating memory, and applying for I/O and CPU time. From another point of view, scheduling is to allocate resources to tasks [38]. In one system, every hardware component is one type of resources of the system, such as memory, CPU, I/O, interrupts, etc. Generally speaking, resources in a system are always limited compared with the requirements of tasks, and therefore tasks need to compete for them [19][24].

3.1.1 Definitions and assumptions

For the purpose of describing and characterizing real-time systems and methods for scheduling, it is necessary for us to give general definitions used in this thesis.

Definitions

A *task* is a unit of work that is scheduled and executed by the system. It requires CPU time and other resources from the system. In a uniprocessor environment, for instance, when an application begins to run, the corresponding process (i.e., a task), will be loaded into memory. During its run time, it may suspend its execution to wait for external interrupts or I/O events, occupy or require more memory. When it finishes running, all resources that it obtains from the system will be released. Some tasks may be *periodic*, which means they can occur at a fixed time interval. Tasks can also be *aperiodic*. Those tasks have no fixed time interval. However, among aperiodic tasks some may be *sporadic*. They may repeat later, but no fixed time intervals exist and their maximum intervals may have no boundaries. But in most practical situations, we could use Minimum Interval Time (MIT) to characterize such tasks, and therefore the worst case situation could be studied. Within the study of network communication, the only resources are the network channels. The tasks are the messages sent by nodes in the network. Accordingly, messages may be periodic, aperiodic or sporadic. In the following part of this thesis, task has the same meaning with message without any particular declaration.

The *release time* of a message is the time instant that the message becomes available in a node and ready to be transmitted. In actual real-time system, when a message is released, the system needs extra time to handle it. For instance, generating information and moving the message into transmitting queue need extra time. Such extra time between the release and the time instant of beginning transmitting is called *jitter*. Since jitter time depends on system design and when a system is given it will be a random variable with fixed upper bound, it contributes

only a constant item in the schedulability analysis. Therefore, in order to simplify the analysis, without specific declaration, jitter will be ignored in this thesis. The *response time* of a message is defined as the length of time taken from its release time to the instant when it completes transmission and reaches the destination.

The *deadline* of a message is defined to be the time instant that the message must finish the transmission and arrive at its destination. This is the definition of the *absolute deadline*. Meanwhile, *relative deadline* is widely used in analysis, which is defined as the length of time taken from the release time of the message to the absolute deadline.

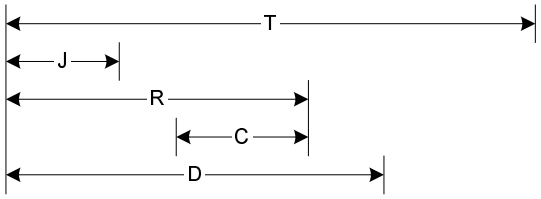

Figure 3.1: The relationship of timing parameters

Figure 3.1 shows the timing parameters relative to a message, where T refers to the period, J refers to the jitter time, R here refers to the response time, C refers to the transmission time, or the computation time in some articles and D refers to the relative deadline.

Assumptions

In an ideal industrial network, we need several assumptions to simplify our analysis. In the later sections, we will remove some of these assumptions.

The assumptions are:

- The messages on the network are independent. Releases of a certain message do not depend on the initiation or completion of other messages;
- All messages are periodic or sporadic. The period of a periodic message should be a constant during working, and MITs of sporadic messages work

as their periods in the following analysis;

- The length of a certain message is constant, although different messages may have different lengths;
- The bit rate of the network is a constant and does not vary with time. Therefore, the transmission time C_i for a certain message is also a constant for a given message. Suppose ν is the bit rate of the network, and L is the length in bit of the message, we will then get $C = L/\nu$;
- The relative deadline D_i of a message should be less than or equal to its period T_i. And as a preliminary condition, C_i should be less than or equal to D_i, i.e. $C_i \leq D_i \leq T_i$;
- Jitter time and the cost of content-switching are ignored.

From the above assumptions we could use $\tau_i(T_i, D_i, C_i), 1 \leq i \leq n$ to characterize n messages completely. Here τ_i denotes any periodic or sporadic message, T_i is its period, D_i is its relative deadline and C_i is its transmission time. Message τ_i is released at time $kT_i, k \geq 0$ and its deadline is $D_i + kT_i$. In some papers maybe $\tau_i(T_i, D_i, C_i, O_i), 1 \leq i \leq n$ is used to represent a message set, where O_i is the phasing offset relative to 0, $0 \leq O_i < T_i$. Therefore, the jobs corresponding to message τ_i are released at time $O_i + kT_i, k \geq 0$. The job ready at time $O_i + kT_i$ has $O_i + D_i + kT_i$ as its deadline. Although the existence of O_i may cause the possible variation of algorithms, when we discuss the worst case schedulability of the message set, O_i is always ignored. That is, we suppose that all messages are released simultaneously at time instant 0.

A message set $\tau_i(T_i, D_i, C_i), i = 1, \ldots, n$ is *schedulable* if for all $1 \leq i \leq n$, the maximum response time r_i of message τ_i is not greater than the deadline D_i. A test to validate whether a given message set is schedulable or not is called *schedulability test*.

Static and dynamic scheduling There are several commonly used scheduling approaches, such as clock-driven, priority-driven, and weighted round-robin scheduling [24]. For instance, many operating systems choose weighted round-robin scheduling, while in communication systems, priority-driven scheduling is adopted widely. And the priority-driven scheduling algorithm may be *static* or *dynamic*. If the priorities of messages could not be changed when the system starts, the scheduling is static, or has fixed priority. If the priorities of messages may vary, it is said to be dynamic. One advantage of static priority scheduling is its simplicity. It is very easy to implement and system overhead caused by it is relatively low. However, compared with dynamic priority scheduling, it may not achieve high bandwidth utilization. That is, if a message set is not feasible to be scheduled by a static priority scheduling algorithm, it may be feasible if a dynamic priority scheduling algorithm is applied.

Preemptive and non-preemptive If the system allows the task with higher priority to interrupt the running of current task with low priority, we can say that the system is *preemptive*. But in some cases, if a task is running, the system does not allow other tasks to interrupt it, regardless of their possible higher priority levels. Such system is *non-preemptive*. Normally in operating systems, whether real-time or not, task scheduling is preemptive. The system can store the status of the task with low priority when preemption occurs. When the task with high priority finishes, the system could continue to execute the interrupted task. However, if tasks are messages, for instance, in a distributed system, communications between nodes rely on the message transmissions, and the cost of preemption is relatively high. If preemption is allowed, the system has to stop the transmission of the message with low priority when a message with high priority comes. After that, the message with low priority can resume the transmission. This policy may be difficult to implement in practice, because it requires the capability of resumption of the aborted messages. For a network packet, the only way is to re-transmit

the whole content. Hence, much network bandwidth is wasted. Therefore, a non-preemptive scheduling is normally used.

3.2 Static priority algorithms

In this section, two typical static priority algorithms, Rate Monotonic and Deadline Monotonic, are introduced and the analysis of schedulability is given.

3.2.1 Rate Monotonic Algorithm

Rate Monotonic Algorithm(RMA) was first studied in [23]. The author provided the analysis of this famous algorithm in a hard real-time environment. Although there are several restrictions in RMA, it is a popular static scheduling algorithm that is widely adopted in applications because of its simplicity and easy implementation.

In addition to the assumptions listed in section 3.1.1, RMA requires two more assumptions:

- The relative deadline of a message is same with its period, $D_i = T_i$, which means the message must finish transmitting before the next release of it;
- The position of message in the message set is due to its priority. Since we use the period as the priority and short period represents high priority, we get $\forall\ 1 \leq i \leq j \leq n, T_i \leq T_j$.

When analyzing the schedulability of a set of messages, if the messages are released at random time instants, there are so many different cases that it is impossible to get all the potential results. In order to simplify the analysis and meanwhile to meet the most general requirements, worst case conditions are considered.

The *critical instant* of a message τ_i is defined to be the time instant at which a message released will have the longest response time. A *critical instant zone* of a

message is the time interval between the critical instant and the end of the response to the corresponding message. When a message is released simultaneously with all other messages with higher priorities, a critical instant for it occurs.

A message set is schedulable using RMA if every message could meet its deadline when it is released at a critical instant. Suppose for n tasks, the utilization of the system is:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i},$$

which could be the processor utilization factor for an operating system, or could be considered as the network bandwidth utilization if the task set has only messages.

C.J. Liu gave a sufficient condition of schedulability for Rate Monotonic Algorithm in [23], which is:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1). \tag{3.1}$$

From equation 3.1 we can calculate the worst case utilization bound of a message set. If the corresponding utilization is less than $n(2^{1/n} - 1)$, the message set must be feasible. When $n \to \infty$, $n(2^{1/n} - 1) \to \ln 2 \simeq 0.693$.

However, the above schedulability condition is sufficient but not necessary. In practice, some message sets that have a higher data utilization than the bound computed by equation 3.1 could be successfully scheduled, even if the data utilization may be around 0.90. Therefore, more accurate schedulability condition need to be studied. To help the analysis, several notations [20] are given:†

$$\begin{aligned} W_i(t) &= \sum_{j=1}^{i} C_j \cdot \left\lceil \frac{t}{T_j} \right\rceil, \\ L_i(t) &= W_i(t)/t, \\ L_i &= \min_{0<t\leq T_i} L_i(t), \\ L &= \max_{1\leq i\leq n} L_i. \end{aligned}$$

†Note: $\lceil x \rceil (x \geq 0)$ denotes the smallest integer greater than or equal to x, and $\lfloor x \rfloor (x \geq 0)$ denotes the greatest integer smaller than or equal to x. That is, $\lceil x \rceil = n, n-1 < x \leq n$ and $\lfloor x \rfloor = n, n \leq x < n+1$, n is integer.

Here $W_i(t)$ represents the cumulative time demands on the system made by the message $\tau_1, \ldots, \tau_i$ over $[0, t]$, when all messages are released simultaneously at time instant 0, hence 0 is the critical instant.

With the help of above notations, we can write the sufficient and necessary conditions for the schedulability test of Rate Monotonic Algorithm as in the following theorem [20]:

Theorem 1 *Given a message set of $\tau_1, \ldots, \tau_n$, if Rate Monotonic Algorithm is applied, τ_i could be feasible for all phasing offsets if and only if $L_i \leq 1$, and the entire message set could be feasible if and only if $L \leq 1$.*

The minimum range of t to test the schedulability should be $[0, T_i]$. Actually, only finite number of points in that range need to be tested. Since the function of $\lceil t/T_i \rceil$ is a monotonically increasing step function, we can find that the function $\lceil t/T_i \rceil / t$ is strictly decreasing except for a finite set of values of t. When t is a multiple of the periods $T_j, 1 \leq j \leq i$, the function has a local minimum and is left continuous and jumps to a higher value to the right. Such discontinuous points are the testing points. Therefore, instead of testing the continuous range $[0, T_i]$, only the local minimums of function $L_i(t)$ at such testing points need to be checked. Let

$$S_i = \{k \cdot T_j | j = 1, \ldots, i; k = 1, \ldots, \lfloor T_i/T_j \rfloor\}.$$

The elements of S_i are the testing points for message τ_i. Now we redefine L_i as following:

$$L_i = \min_{t \in S_i} L_i(t).$$

Hence the theorem 1 could be modified as following:

Theorem 2 *Given a message set of $\tau_1, \ldots, \tau_n$, if Rate Monotonic Algorithm is applied, τ_i could be feasible for all phasing offsets if and only if*

$$L_i = \min_{t \in S_i} W_i(t)/t \leq 1,$$

and the entire message set could be feasible if and only if

$$L = \max_{1 \le i \le n} L_i \le 1.$$

The proof of this can be found in [20].

3.2.2 Deadline Monotonic Algorithm

Although RMA could work well in most circumstances, it has some inconvenient restrictions. One severe constraint is that it requires the deadline of a message to be equal to its period. But in practical hard real-time environments, that is not always true. Messages may be required to arrive at the destination a little or much earlier to leave time to the system for handling or to compensate for extra delay during transmitting. A popular algorithm to handle messages with deadlines different from the periods is the Deadline Monotonic Algorithm (DMA).

Suppose the message set $\tau_i(C_i, D_i, T_i), 1 \le i \le n$, and $C_i \le D_i \le T_i$. The Deadline Monotonic algorithm uses a similar priority assignment scheme as RMA. In this case, the priorities assigned to messages are inversely proportional to the relative deadline [22]. Thus, the message with shorter deadline gets the higher priority. And when the deadline and the period are equal for every message in the set, the priority order of DMA is totally same with that of RMA.

Deadline Monotonic priority assignment is an optimal static priority scheme [22]. That is, if any static priority scheduling algorithm can schedule a message set, then a deadline monotonic algorithm will also schedule the message set.

Schedulability tests

Schedulability tests for Deadline Monotonic Algorithm introduced in [3][4] are founded upon the concept of critical instants [23]. Since messages are supposed to be released simultaneously, the worst case condition occurs. One schedulability

test is given by [3]:

$$\forall\, i : 1 \leq i \leq n : \frac{C_i}{D_i} + \frac{I_i}{D_i} \leq 1 \tag{3.2}$$

where I_i represents the interference of all messages with higher priorities than message τ_i and is given by:

$$I_i = \sum_{j=1}^{i-1} \left\lceil \frac{D_i}{T_j} \right\rceil C_j. \tag{3.3}$$

If equation 3.2 is written as $C_i + I_i \leq D_i$, then the condition that makes a message τ_i schedulable is that the sum of its transmission time (computation time) and the interference caused by higher priority messages must be no more than D_i. The condition above is sufficient, but not necessary. The reason is due to the calculation of the interference time I_i. Some messages with higher priorities may be released before D_i and completed after D_i. Therefore, I_i may be greater than the actual interference encountered by τ_i before D_i. Considering the above case, a more accurate representation of I_i is given by:

$$I_i = \sum_{j=1}^{i-1} \left[\left\lfloor \frac{D_i}{T_j} \right\rfloor C_j + \min\left(C_j, D_i - \left\lfloor \frac{D_i}{T_j} \right\rfloor T_j \right) \right]. \tag{3.4}$$

Equation 3.2 with I_i defined by 3.4 is still sufficient but not necessary in the general case. In [3], an even more complex representation of I_i is given.

Response time

Another way to analyze the schedulability is from the worst case response times of messages [2][6]. Using the time model shown in figure 3.1, the worst case response time r_i of message τ_i could be expressed as following:

$$r_i = C_i + B_i + I_i$$

where C_i is the transmission time or computation time, B_i is the worst case blocking time of message τ_i due to the priority ceiling protocol or other concurrency control protocols, and I_i is the worst case inference. In order to make message τ_i feasible

we must have $r_i \leq D_i$. The total interference I_i is given by:

$$I_i = \sum_{j=1}^{i-1} \left\lceil \frac{r_i}{T_j} \right\rceil C_j.$$

Combining above the two equations, we get:

$$r_i = C_i + B_i + \sum_{j=1}^{i-1} \left\lceil \frac{r_i}{T_j} \right\rceil C_j.$$

However, it is noticed that the unknown term r_i appears on both sides of the above equation. To solve the above equation, an simple iterative algorithm could be used. Let r_i^m be the m^{th} approximation of the true value of r_i, and we get the iterative expression of the above equation:

$$r_i^{m+1} = C_i + B_i + \sum_{j=1}^{i-1} \left\lceil \frac{r_i^m}{T_j} \right\rceil C_j. \tag{3.5}$$

When calculating r_i, the iteration begins with $r_i^0 = 0$, and terminates when $r_i^{m+1} = r_i^m$. It is very easy to get that $r_i^{m+1} \geq r_i^m$, and therefore if $r_i^{m+1} > D_i$ the iteration could be stopped. And it is shown in [16] that when the total utilization is $\leq 100\%$ the iteration must converge.[1] And the above solution could be used in a more general area. In fact, it could be used to solve any static priority driven scheduling problem, either Rate Monotonic Algorithm or Deadline Monotonic Algorithm because the above discussion does not mention the specific priority assignment algorithm. When the priority assignment scheme is given, equation 3.5 could be used to calculate response times of a specific message set to check whether it is feasible or not.

If preemption is not allowed, such as in CAN bus environment, when applying Deadline Monotonic Algorithm, response time can be used to do the schedulability test of a given message set. If response time of every message on the bus can be

[1] Please note here we suppose preemption is allowed. In a non-preemptive environment, r_i is not guaranteed to be converged even if the total utilization is $\leq 100\%$.

less than or equal to its deadline, this message set is feasible.

Suppose that jitter time exists and ignore any errors, the response time of message τ_i on CAN bus can be expressed as the following [36]:

$$r_i = J_i + C_i + w_i \tag{3.6}$$

where J_i is the queuing jitter of message τ_i, C_i is the transmission time of the message τ_i, and the term w_i represents the worst case queuing delay - the longest time between placing the message in the priority-ordered queue and the starting of transmission of the message. This is given by:

$$w_i^{m+1} = B_i + \sum_{j=1}^{i-1} \left\lceil \frac{w_i^m + J_j + \tau_{bit}}{T_j} \right\rceil C_j,$$

where the term B_i is the worst-case blocking time of message τ_i, and is equal to the longest time taken to transmit a lower priority message. τ_{bit} is the time to transmit a bit on the bus. Assuming that in the message set, $\forall 1 \leq i \leq j \leq n$, priority of message τ_i is higher than that of message τ_j. If τ_i is the lowest priority message then B_i is zero. Hence B_i is given by:

$$B_i = \max_{i<j\leq n}(C_j).$$

If we let ν denotes the bitrate of the CAN bus, we get $\tau_{bit} = 1/\nu$. As mentioned earlier, $C = L/\nu$, where L is the possible maximum length of a message on CAN bus, as shown in table 3.1. The extra length of a message except the data field is 47 bits long. Meanwhile, bit stuffing should not be ignored. Among all fields in a CAN message, the Arbitration field, Control field, Data field and CRC field are subjected to bit stuffing. Therefore, C_i is given by:

$$C_i = \left(\left\lfloor \frac{34 + 8\left\lceil \frac{l_i}{8} \right\rceil}{5} \right\rfloor + 47 + 8\left\lceil \frac{l_i}{8} \right\rceil \right).$$

Table 3.1: The possible maximum length of CAN message τ_i

Field	**Length**(bit)
SOF	1
Arbitration	12
Control	6
Data	$8 \cdot \lceil l_i/8 \rceil$
CRC	16
ACK	2
EOF	7
Intermission	3
Bit stuffing	$\left\lfloor \frac{34+8\cdot\lceil l_i/8 \rceil}{5} \right\rfloor$

l_i is the original data length in bit of message τ_i.

When every item is determined, equation 3.6 could be used to calculate the response times of messages on CAN bus. If $\forall i, 1 \leq i \leq n, r_i \leq D_i$ is met, the message set is feasible.

3.3 Dynamic priority algorithms

As opposed to static priority algorithms, where messages get their priority only once and never change during the running of the system, dynamic priority algorithms assign different priority to a message for every release.

There are several dynamic priority algorithms. The most famous one is the Earliest Deadline First (EDF) algorithm. This priority assignment scheme assigns priorities to messages according to their absolute deadline, the earlier the deadline, the higher the priority. If preemption is allowed and messages are independent, EDF is said to be *optimal*, that is, the EDF algorithm could generate a feasible schedule of a message set $\tau_i, 1 \leq i \leq n$, if and only if the message set has feasible schedules [24]. For non-preemptive scheduling, it is shown in [38] that EDF performs better than other simple heuristics.

EDF is not the only optimal dynamic priority algorithm. The latest release time (LRT) algorithm (or reverse EDF algorithm), which treats release times as

deadlines and deadlines as release times and schedules messages backwards, is also optimal for preemptive and independent tasks.

Another well-known dynamic priority algorithm is the Least Slack Time First (LST) algorithm.[2] At any time instant t, the *slack* (or *laxity*) of a message τ_i with absolute deadline d_i is equal to $d_i - t$ minus the time required to complete the transmission of the remaining part of message τ_i. The LST algorithm assigns message priorities based on their slacks: the smaller the slack, the higher the priority. And if preemption is allowed and messages are independent, it can also be shown that the LST algorithm is optimal [22]. Compared with the EDF algorithm, the LST algorithm requires that the computation times of tasks be known. In many cases this is a disadvantage because the actual computation time of a task are often not known until it completes, and also we could not get the exact slack times of the tasks. However, when we consider the message transmission in industrial network, the maximum length of messages on the bus is either fixed or can be predicted if the communication protocol is given. For example, in CAN bus, the maximum length of a data field could not exceed 8 bytes, so that the total length of a data frame will never exceed 111 bits long (CAN 2.0A standard in regardless of bit stuffing). And in most cases, the message length for a particular type is determined when the system is set up.

Another issue here is that if EDF scheduling is applied, one problem is that the value of the absolute deadlines of messages will increase to infinity with the running of the system time. But in practice, limited bits are not enough to store such values of the messages. Therefore the priority mapping problem occurs (refer to chapter 5 for details). But in the LST algorithm the slack time of a message is limited to the relative deadline of a message, and therefore it is easy to implement the priority mapping. This issue is the main topic of the next chapter.

[2]It is also called the Minimum Laxity First (MLF) algorithm.

3.4 Summary

In this section, the typical priority-driven scheduling algorithms are introduced, including Rate Monotonic Algorithm, Deadline Monotonic Algorithm and Earliest Deadline First Algorithm. And issues of schedulability test are studied. Some necessary and sufficient conditions are given and their confinements in non-preemptive environments are analyzed.

In the next chapter, schedulability condition for non-preemptive scheduling will be examined. And a new effective schedulability condition will be given including the corresponding algorithm.

Chapter 4

A New Schedulability Condition for Non-Preemptive Scheduling

In this chapter, a new sufficient and necessary schedulability condition for non-preemptive scheduling is given. And the corresponding algorithm is presented.

4.1 Previous work on schedulability condition

Suppose all time cost of context switching and other extra load (such as time cost in preemption or resumption) are ignored, then the basic idea of schedulability under deadline scheduling policy is straightforward; at any time instant t, every message in the message set should meet its deadline. Therefore $\forall t > 0$, the overall amount of time in $[0, t]$ to transmit all messages that arrive during $[0, t]$ with deadlines $\leq t$ should not be greater than t. So define a function $\lceil x \rceil^+ = n$, if $n - 1 \leq x < n, n = 1, 2, \ldots$, and $\lceil x \rceil^+ = 0$ for $x < 0$,then we get the following sufficient and necessary condition for schedulability test of EDF algorithm [39]:

Theorem 3 *When deadline scheduling policy is applied, if preemption is allowed and messages are independent, a set of n messages* $\tau_i = (T_i, C_i, D_i), i = 1, 2, \ldots, n,$

are schedulable if and only if

$$\forall t \geq 0, \sum_{i=1}^{n} \left\lceil \frac{t - D_i}{T_i} \right\rceil^{+} \cdot C_i \leq t. \tag{4.1}$$

Suppose the system is initialized at time $t = 0$, then for any message τ_i, during $[0, t]$, there are at most $\lceil (t - D_i)/T_i \rceil^{+}$ releases of τ_i with deadline $\leq t$, and the corresponding time to transmit those messages is $\lceil (t - D_i)/T_i \rceil^{+} C_i$. Therefore for the whole message set, the maximum time to transmit all possible releases during $[0, t]$ is $\sum_{i=1}^{n} \lceil (t - D_i)/T_i \rceil^{+} C_i$. This proves the necessary condition of the above theorem. And the sufficient condition could be proven by contradiction.

However, in the form of equation 4.1, Theorem 3 could not be used directly to check the schedulability of a message set, because it is required to verify the inequality over a continuous and infinite period $[0, \infty)$. Two points could resolve this difficulty. First, using the same method in the analysis of Theorem 1, we can notice that the left-hand side of the inequality 4.1 is a piecewise function. Therefore, the set of testing points is discrete and consists of the discontinuous points in $[0, \infty)$. Second, there exists a point t_{max} such that if the total data utilization of the network $\sum_{i=1}^{n} C_i/T_i \leq 1$, the inequality of Theorem 3 could always be met for $\forall t \geq t_{max}$. Hence, only a finite number of testing points need to be checked when applying Theorem 3 to test the schedulability of a message set, which means it is possible to execute such test by a computer. Therefore, Theorem 3 could be modified as following:

Theorem 4 *When deadline scheduling policy is applied, if preemption is allowed and messages are independent, a set of n messages $\tau_i = (T_i, C_i, D_i), i = 1, 2, \ldots, n$, are schedulable if and only if*

$$\begin{aligned} &1.\ \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1, \\ &2.\ \forall t \in S, \quad \sum_{i=1}^{n} \left\lceil \frac{t - D_i}{T_i} \right\rceil^{+} \cdot C_i \leq t \end{aligned} \tag{4.2}$$

where $S = \bigcup_{i=1}^{n} S_i$, $S_i = \{D_i + nT_i, n = 0, 1, \ldots \lfloor (t_{max} - D_i)/T_i \rfloor\}$, *and* $t_{max} = \max\{D_1, \ldots D_n, (\sum_{i=1}^{n}(1 - D_i/T_i)C_i)/(1 - \sum_{i=1}^{n} C_i/T_i)\}$.

The complete proof could be found in Appendix A.

For non-preemptive EDF scheduling, in [39] a schedulability condition is described as following: when preemption is not allowed and messages are independent, the message set is schedulable in worst case if and only if the following conditions are met:

$$\begin{aligned} &1.\ \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1, \\ &2.\ \forall t \in S, \quad \sum_{i=1}^{n} \left\lceil \frac{t - D_i}{T_i} \right\rceil^{+} \cdot C_i + C_p \leq t \end{aligned} \tag{4.3}$$

where C_p is the length of the longest possible non-realtime message, $S = \bigcup_{i=1}^{n} S_i$, $S_i = \{D_i + nT_i, n = 0, 1, \ldots \lfloor (t_{max} - D_i)/T_i \rfloor\}$, and $t_{max} = \max\{D_1, \ldots D_n, (C_p + \sum_{i=1}^{n}(1 - D_i/T_i)C_i)/(1 - \sum_{i=1}^{n} C_i/T_i)\}$.

4.2 Further analysis

The author in [39] asserts that inequality 4.3 is a necessary and sufficient condition. However, that conclusion is not always true in non-preemptive environment. Let us examine the following example: to do the schedulability test of a given message set, which contains two periodic messages and their properties as listed in Table 4.1.

Table 4.1: The example of non-preemptive EDF scheduling

Message	T	C	D
τ_1	3	1	1.5
τ_2	4	2	3

The given message set is, $\tau_1 = (T_1, C_1, D_1) = (3, 1, 1.5)$ and $\tau_2 = (T_2, C_2, D_2) =$

$(4, 2, 3)$. Using equation 4.3, we first check the total data utilization, and get

$$\sum_{i=1}^{2} C_i/T_i = \frac{1}{3} + \frac{2}{4} \simeq 0.833 < 1.$$

Since no non-real-time message exists, $C_p = 0$, and $t_{max} = 6$. Therefore, $S_1 = \{1.5, 4.5\}$, $S_2 = \{3\}$, and $S = S_1 \cup S_2 = \{1.5, 3, 4.5\}$. Applying the inequality 4.3, it is easy to verify that $\forall t \in S, \sum_{i=1}^{2} \lceil (t - D_i)/T_i \rceil^+ C_i \leq t$. Thus, according the above schedulability conditions we say that the given message set is schedulable. But the simulation of the real scheduling process in Figure 4.1 shows that the given message set is not always schedulable. It is found that message τ_1 misses its deadline at time instant $t = 10.5$.

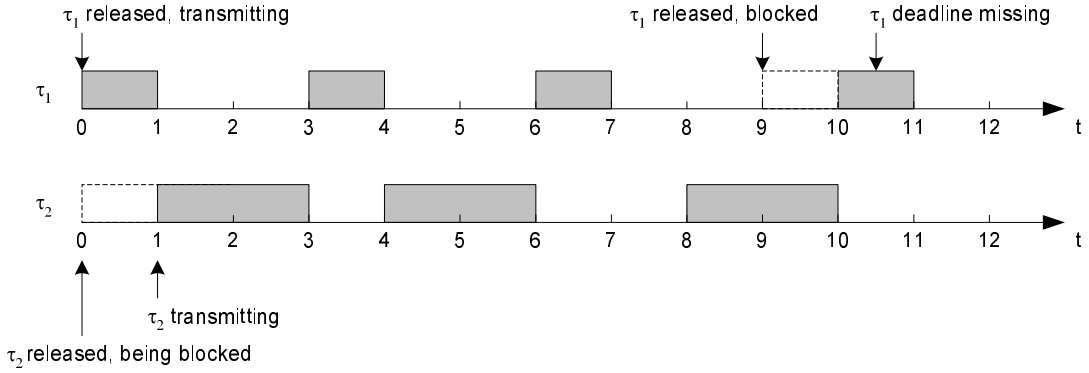

Figure 4.1: The example of non-preemptive EDF scheduling

Because preemption is not allowed, when message τ_1 is released at $t = 9$, although it has an earlier deadline ($d_1 = 9 + D_2 = 10.5$) than message τ_2 which was released at $t = 8$ ($d_2 = 8 + D_2 = 11$), message τ_1 is blocked until τ_2 finishes transmission at $t = 10$. Consequently, message τ_1 misses its deadline at $t = 10.5$.

Further more, even if we use $t = 10.5$ to examine the inequality 4.3, we find that the inequality still can be met, although the message set is shown to be unschedulable.

This example shows that the sufficient condition of 4.3 is not held. The proof of the sufficient condition in [39] omits the possible priority inversion in non-preemptive environments. But the necessary condition is always true. That is,

the above schedulability condition is *necessary but not sufficient*.

4.3 A new schedulability condition

In order to find a more effective method to check the schedulability conditions in non-preemptive environments, we can consider heuristic methodologies. W. Zhao studied heuristic algorithms for scheduling tasks with time and resource constraints [38]. However, periodic tasks are not considered in his work. In the case of network message transmission, tasks are periodic or sporadic messages, and the only resource constraint is the bus.

Define the *feasible time* for a message as the time instant at which the message could begin to transmit definitely without any blocking or interference. We propose the following lemma to test the schedulability of a message set.

Lemma 1 *The message is schedulable if and only if for every release of message* $\tau_{ij}, 1 \leq i \leq n, j = 0, 1, \ldots$, *the response time* $r_{ij} = F_i + C_i$ *is always less than or equal to its absolute deadline* $d_{ij} = R_{ij} + D_i$, *where* F_i *is the feasible time of message* τ_i, *and* R_{ij} *is the* j^{th} *release time of message* τ_i.

The proof of Lemma 1 is straightforward: the definition of *schedulable* requires $r_{ij} \leq d_{ij}$ for message τ_i. If every message in the set is schedulable, we can say that the message set is schedulable. Obviously, the worst case upper bound on the temporal axis to test the schedulability is $t_{max} = LCM(T_i, 1 \leq i \leq n)$, i.e. t_{max} is the least common multiple of the periods of all messages in the message set. So when t_{max} is determined, only limited points on the temporal axis are needed to be checked according to Lemma 1. Hence, a simple algorithm to check the schedulability is possible to be implemented.

Let us examine the following example shown in figure 4.2.

Suppose $d_1 \leq d_2 \leq d_3$, hence $Priority(\tau_1) \geq Priority(\tau_2) \geq Priority(\tau_3)$, and before the release of τ_1, the bus is idle. Therefore the feasible time of τ_1, namely F_1

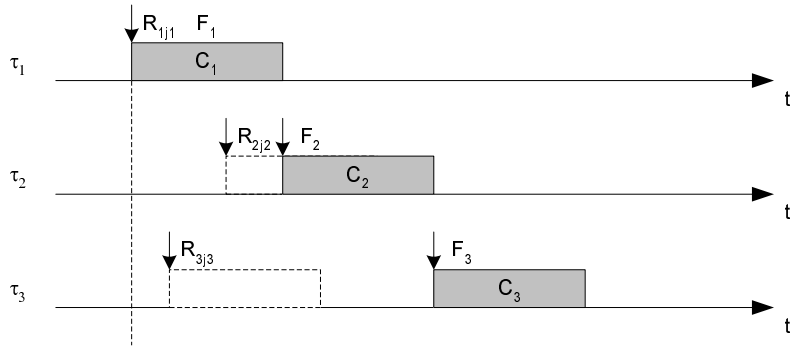

Figure 4.2: Feasible times of messages

is equal to its release time R_i. And the response time of τ_1 is $r_1 = F_1 + C_1$. So the next feasible time for all the other messages should be $F_2 = F_1 + C_1$. During the time period $[F_1, F_2)$, both message τ_2 and message τ_3 are released. Since preemption is not allowed, both τ_2 and τ_3 are blocked by τ_1. After the transmission of τ_1, although message τ_3 is released before message τ_2, since $Priority(\tau_2) \geq Priority(\tau_3)$, τ_2 will get the bus. The next feasible time of the bus is then adjusted according to the current feasible time, $F_3 = F_2 + C_2$. After τ_2 finishes transmitting, we check $[R_1, F_3)$ again and find that τ_3 is still blocked. Then τ_3 gets the bus and begin to transmit.

From the analysis of above procedures, we can generalize the heuristic algorithm shown in Figure 4.3.

Where routine *release_message()* returns the message set $\mathcal{M}$ that contains messages released during time period $[F, F')$. When given the message set $\mathcal{S}$, the algorithm could check whether $\mathcal{S}$ is schedulable or not.

However, some problems still exist in above algorithm. For example, in order to utilize the algorithm in real-time environments, the overhead of it should be as low as possible. In Figure 4.3 $t_{max} = LCM(T_i, 1 \leq i \leq n)$. If the periods of messages are chosen improperly, t_{max} may be too high to be acceptable because the upper bound t_{max} depends on the particular message set. For instance, if most relative deadlines in the message set are large prime numbers, the result of t_{max} may be very large so that the schedulability test will cost very long time. Q. Zheng

Input: Message set $\mathcal{S} = \{\tau_i, 1 \leq i \leq n\}$;
Output: *schedulable*; /* Boolean */

Method:

```
t_max = LCM(T_i, 1 ≤ i ≤ n);
F = F' = 0;
schedulable=TRUE;
while (F < t_max && schedulable) {
    if ((M = release_message(F, F')) ≠ φ) {
        i = highest_priority_in_set(M);
        F = F';
        F' = F + C_i;
        d_i = R_i + D_i;
        if (F' > d_i)
            schedulable=FALSE;
    }
    else {
        R = next_minimum_release(M, F);
        i = highest_priority_in_set(R);
        F = R_i;
        F' = R_i;
    }
}
return schedulable;
```

Figure 4.3: The heuristic algorithm for non-preemptive EDF

proposed an approximate upper bound in [39]:

$$t_{max} = \max\{D_1, \ldots D_n, (C_p + \sum_{i=1}^{n}(1 - D_i/T_i)C_i)/(1 - \sum_{i=1}^{n} C_i/T_i)\}.$$

But from the analysis in this chapter, it shows that t_{max} is not applicable for non-preemptive scheduling. More effective methods may be needed to optimize the calculation of t_{max} and then reduce the overhead. But consider in most industrial applications, most nodes in a network share similar characteristics. For example, it is very common that several nodes use the same type of sensors and actuators and work at the same frequency. Therefore, there may be very limited number of different frequencies, or periods. In that case, LCM of those periods is feasible to be as the upper bound of t_{max} in schedulability test.

4.4 Summary

This chapter gives a thorough analysis of schedulability condition for non-preemptive scheduling algorithm. And a new schedulability condition is proposed to do the precise schedulability test of a given message set.

In the next chapter, issues of priority mapping are studied. And DLP scheme is proposed to solve the priority inversion for limited priority level condition.

Chapter 5

Dynamic Local Priority Scheme

This chapter discusses scheduling schemes and priority mapping issues in bus scheduling. Priority inversion may arise due to limited priority levels. And a new scheme is proposed to solve this difficulty.

5.1 Scheduling schemes

Generally speaking, there are two typical schemes to handle the scheduling process in distributed environments: centralized scheduling or distributed scheduling. The advantages and disadvantages of each scheme are discussed in this section.

5.1.1 Centralized scheduling

Some centralized scheduling schemes could solve the scheduling problems of CAN bus, such as FTT-CAN (Flexible Time Triggered communication on CAN) protocol [29] [18] and the planning scheduler scheme [11]. Those schemes are so-called centralized scheduling scheme because a master node on the bus is required to control the message scheduling of the bus.

When FTT-CAN protocol is applied, at a fixed time interval, which is called elementary cycles (EC), the master node on the bus broadcasts a periodic message

to the bus to indicate the beginning of the EC. And that broadcast message contains the information about which nodes should send messages in the current EC. Therefore, the nodes that are informed by the master node will send messages to the bus during the rest of the EC. And the bus arbitration is solved by the native arbitration mechanism of CAN bus.

The planning scheduler scheme uses a similar idea to FTT-CAN protocol. There are plans in the scheduler. And each plan consists of a fixed number of ECs with fixed duration. The messages' periods and their transmission deadlines should be an integer multiple of the EC duration. The master node dispatches the schedule of message transmitting in one EC according to the current plan in it. When a plan is given, the scheduler works like a static scheduler. When the current plan is finished, a new plan will be built and loaded into the scheduler. So the scheduling information gets undated dynamically, and flexibility is guaranteed.

By adjusting the duration of EC, FTT-CAN and planning scheduler schemes could get a tradeoff between the overheads and the efficiency. However, the most serious problem in the above methods is: there must be a master node on the bus to work as the centralized scheduler. For an environment that only one controller exists and all others are devices, such schemes may work well. However, a practical industrial system is normally a distributed control system. If there are tens of controllers and each of them is identical on the bus, it will be difficult to choose the master node. Moreover, if the master node is down, the whole network would not work.

5.1.2 Distributed scheduling

Unlike centralized scheduling schemes, distributed scheduling does not require a master node in the network. Every node in the network competes for the right to transmit messages in the network individually. They are not controlled by a centralized scheduler. Normally such scheduling algorithms are implemented in

hardware and are the essential characteristics of the network, such as LAN (802.3 Ethernet) protocol or Token Ring (802.5) protocol.

The native CAN protocol, using an effective arbitration mechanism to control the distributed scheduling of message transmission, is a fixed-priority scheme. However, as we discussed in the previous chapter, fixed-priority scheme could not utilize all potential capacities of the network. Although for most real-time applications the fixed-priority scheme is competent, in a heavily loaded environment it is necessary to improve the schedulability of the network. In the past, the implementation of dynamic priority schemes has some problems, such as increase cost and overhead due to the complexity of algorithms that could result in a low speed of CAN controllers. But, in recent years as the speed of microcontrollers becomes faster and faster, this overhead costs very little CPU resources. And to avoid the necessity of modifying hardware, a new application layer, which controls the real-time priority assignment, could be implemented in the real-time system that work between applications and the MAC layer of the CAN protocol. Therefore dynamic priority schemes could be realized in CAN network.

Cena and Valenzano [7] proposed DPQ (Distributed Priority Queue) and PP (Priority Promotion) mechanisms to modify the original CAN protocol in order to get arbitration fairness and to guarantee the upper bound of transmission time. Later they proposed a new CAN-like network model fastCAN [8], which uses two unidirectional physical channels, to improve the network performance.[1] Livani, Kaiser and Jia proposed a hybrid bus scheduling algorithm [25] which tried to combine the advantages of TDMA (time-division multiple-access) and dynamic Least-Laxity-First scheduling. However, one disadvantage of the above methods is that they all use the extended identifier format of CAN frame. As we mentioned, transmitting frames of extended format wastes more bus bandwidth and reduces the ratio of effective data of a frame.

[1] Since fastCAN modifies the CAN model substantially, the discussion of fastCAN exceeds the boundary of this thesis. For more details of fastCAN, please refer to [8].

5.2 Limited priority levels

Normally, our schedulability analysis is based on unlimited priority levels, which means that we can allocate as many priorities as we can to messages. However, in reality, it is impossible to get unlimited priority levels.

In CAN bus, the Arbitration Field of CAN2.0A frame format is 11 bits long [5]. In CAN 2.0B extended frame format, the Arbitration Field has 29 bits, therefore more bits can be used to represent priority levels. However, in CAN2.0B extended frame format, the efficiency of the real network bandwidth utilization is lower than that in standard frame format. Suppose the length of a frame in the standard format is $44 + 8n$ bits, where n is the byte length of the Data field. On the other hand, the length of a frame in the extended format is $64 + 8n$ bits. Suppose $n = 8$, the maximum length. It is clear that the same data field of a frame in the standard format occupies 59.3% of the total frame, but only 50% in the extended format. For $n = 4$, the ratios of the length of the data field to the total length of the frame are 42.1% and 33.3%, respectively (Here the potential bit stuffing of the frame is ignored). If most data packages in the network are in short length, the effective bus utilization in two frame formats will be in big difference. Therefore, many solutions to CAN scheduling suppose that the data frames are based on the standard format so as to get higher effective utilization of the bus bandwidth.

Suppose CAN2.0A frame format is adopted, we can have at most 2^{11} priority levels which seem enough in most applications. But the actual priority levels that we can use are far less. There are two reasons for this: first, the highest 7 bits could not be recessive. Therefore 2^4 priority levels could not be used. Second, in the network, every nodes or devices connecting to the bus must have a unique identifier to avoid communication chaos. Suppose we can use 5 bits to represent those identifiers, which means the maximal number of the devices is $2^5 = 32$. Hence, only 6 bits ($2^6 = 64$) are left to represent priorities. Therefore, when assigning priorities to messages, messages with long deadlines may get the same priority

with those with short deadlines. Then when the system is running, a message with shorter deadline may be *blocked* when it is prevented from transmitting by a message with longer deadline. And we say that a *priority inversion* occurs whenever a message with lower priority (longer deadline) begins transmitting before some ready messages with higher priorities (shorter deadlines).

The relative schedulability bound of RMA

Consider the sufficient schedulability condition of Rate Monotonic Algorithm, i.e. if the data utilization of the message set $U = \sum_{i=1}^{n} C_i/T_i \leq n(2^{1/n} - 1)$, then the message set is schedulable. And when $n \to \infty$, $n(2^{1/n}-1) \to \ln 2 \simeq 0.693$. However, that is only for unlimited priority levels. When the number of priority levels is limited, the condition changes. In [17], issues on priority mapping into limited priority levels in fixed-priority scheduling algorithms were discussed. Suppose there are m priority levels, $\{L_1, L_2, \ldots, L_m\}$, where L_i is assigned a message period. If a message has a period T such that $L_{i-1} < T \leq L_i$, it will be assigned priority L_i. The approximate utilization bound is below:

$$U = \begin{cases} \ln(2G) + 1 - G & \text{when } G > 1/2 \\ G & \text{when } G \leq 1/2 \end{cases}$$

where G is the grid granularity defined as $G = \min\{(L_{i-1} + 1)/L_i, \forall i, 1 \leq i \leq m\}$ [21].

Introducing a constant ratio $r = 1/G$, and comparing the schedulability bound with $\ln 2$, which is the bound with unlimited priority levels, the relative schedulability R_S is defined as follows:

$$R_S = \begin{cases} [\ln(2/r) + 1 - 1/r]/\ln 2 & \text{when } r < 2 \\ 1/(r \cdot \ln 2) & \text{when } r \geq 2 \end{cases} \tag{5.1}$$

It is easy to find that when the number of priority levels reduces, the relative

schedulability deteriorates according to equation 5.1. For example, suppose the shortest and longest periods of the message set are $1ms$ and $100,000ms$, respectively. And the bus supports up to n priority levels. Therefore, $L_0 = 1ms$, and $L_n = 100,000ms$. To simplify the computation, we have $L_1/L_0 = L_2/L_1 = \cdots = L_n/L_{n-1} = r$. So we get $r = (L_n/L_0)^{1/n}$. Applying equation 5.1, Figure 5.1 shows the relation between relative schedulability and the number of priority levels, where n is from $2^1 = 2$ to $2^8 = 256$.

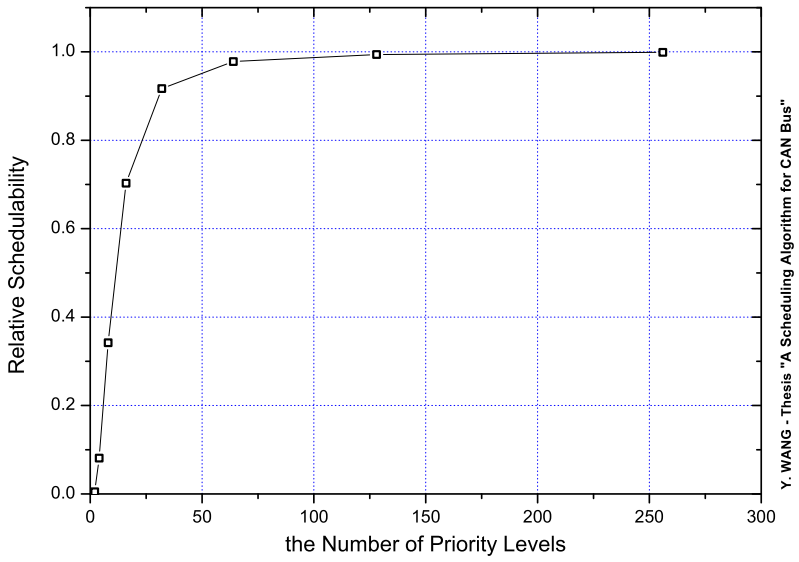

Figure 5.1: The effect of limited priority levels

From Figure 5.1, we see that when the number of priority levels reduces, the relative schedulability drops sharply. When the number of priority levels is less than 16, the relative schedulability will be less than 0.6, i.e. when the number of messages approaches infinity, the utilization should be less than $0.6 \ln 2 = 0.416$, which is quite low and unacceptable. For CAN bus, if 64 priority levels are applied, the result seems acceptable; the relative schedulability is equal to 0.978. And for fixed-priority scheduling algorithms, the priorities of messages are assigned before the network runs. If the priority mapping algorithm is chosen carefully, the priority levels can be used sufficiently. That is, if the priority levels are enough, messages with different periods or deadlines will be assigned different priorities.

5.3 DLP scheme

If dynamic priority scheduling algorithms are applied, the priority inversion and the problem of limited priority levels are completely different with those in RMA.

In dynamic scheduling algorithms, message priorities are assigned according to a function of their deadlines or latency time. Considering the Earliest Deadline First scheduling, a message with shorter deadline will get a higher priority. But the message deadlines may vary in a rather large range, from one or two milliseconds to tens of seconds. And normally their distribution along the temporal axis is not uniform. Therefore, when doing priorities mapping, it is possible that some priority levels are crowd while some are not used. For example, suppose 10 messages are in the system, and the system has 10 priority levels, 1 to 10. In a certain time instant, their deadlines are shown in Table 5.1, and the distribution of those deadlines in temporal axis is shown in the Figure 5.2.

Table 5.1: Sample message information

Deadline	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}
(ms)	10	25	30	42	44	46	48	50	95	100
Assigned priority	1	3	3	5	5	5	5	5	10	10
Expected priority	1	2	3	4	5	6	7	8	9	10

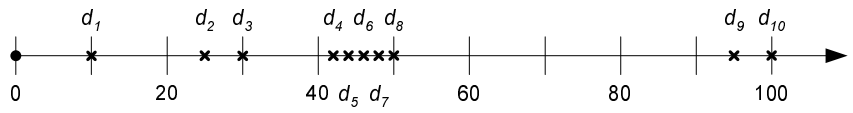

Figure 5.2: Point distribution in the temporal axis

When assigning priorities to messages, two methods can be used. The first one is to scale to temporal axis with equal size time slot, such as MTS (Mixed Traffic Scheduling) algorithm [41]. Here we consider the part to handle high speed messages only. The priority assignment result according to MTS is shown in Table 5.1. From Figure 5.2 we find that there are 10 priority levels, but only 4 of them are used, although each message has a different deadline from others. Five deadlines,

d_4 to d_8, are assigned into the same priority level, but 6 priority levels are wasted. Consequently, since messages with longer deadlines have the same priorities as those with shorter deadlines, when the system runs, priority inversion may occurs. Also, the capability of priority levels is not used sufficiently either.

The second method was proposed in [9]. To improve the efficiency and reduce priority inversion, a logarithmic quantization of message deadlines was applied. That means that the scale unit of the temporal axis is not a constant, but exponentially increasing. By defining a basic length δ_0, the length of every section I_i increases exponentially and is an integral multiple of δ_0. Therefore, nearer deadlines could get a finer resolution and further deadlines get coarse one. Figure 5.3 shows the temporal axis in logarithmic EDF algorithm.

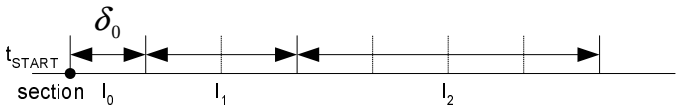

Figure 5.3: The temporal axis with logarithmic quantization

The logarithmic EDF algorithm could improve the performance, especially when the transmission rate is not very high. M.D. Natale [9] indicated that the approximate CPU load was below 3% at 125kbps. However, there are two major restrictions. First, the author supposed that the length of messages was fixed, 8 bytes long. But in real industrial network, message lengths are always varying. Therefore, if the priority-updating procedure still executes every $0.86ms$, the network bandwidth will be wasted. Second, the capability of priority levels is still not utilized sufficiently. Under some special conditions, the assigned priorities of messages may be crowded into a short period in the temporal axis.

In order to solve the above restrictions, a new priority mapping and scheduling scheme called DLP (Dynamic Local Priority) is proposed. In DLP, it is expected to assign ideal priority levels to the message set shown in Table 5.1.

5.3.1 The description of DLP

The fundamental concept of the algorithm is straightforward. Since it's difficult to assign the priority in a distributed environment, we can collect the information in the network so that the message priority could be assigned locally. Actually, this thought is not acceptable in a distributed system that has thousands of nodes, because the overhead to maintain a copy of local information is terribly high [34]. But for CAN 2.0A bus, as what we describe above, there are only 5 bits to represent the unique identifiers of messages, therefore the maximal number of nodes is limited to 32. So the overhead is also limited.

This DLP scheme is an extension of EDF scheduling. In order to calculate and assign priorities to messages, their periods and deadlines should be easy to obtain and update for every node on the bus. Hence, each node on the bus maintains a linked list locally, which is called NNL (Network Node List), to store such information and adjusts the sequence in the list dynamically according to current bus time and deadlines of messages.

Using DARTS method (Design Approach for Real-Time Systems) proposed in [13][14], we could design the structure of the algorithm. The state transmission diagram (STD) of the local scheduler is shown in figure 5.4.

The basic states of the local scheduler includes *Initialization*, *Idle*, *Priority updating*, *Send* and *Receive*. Signals and events trigger the transmissions between states. The following sections describe the approach in details.

5.3.2 Initialization

When connecting to the network for the first time, every node should initialize the connection and build its NNL, which records the priority levels of all nodes connected to the network. At the beginning of the initialization, the NNL in a node has the information of its own only. One node will broadcast its message information, i.e. its period and deadline. Such broadcast message causes other

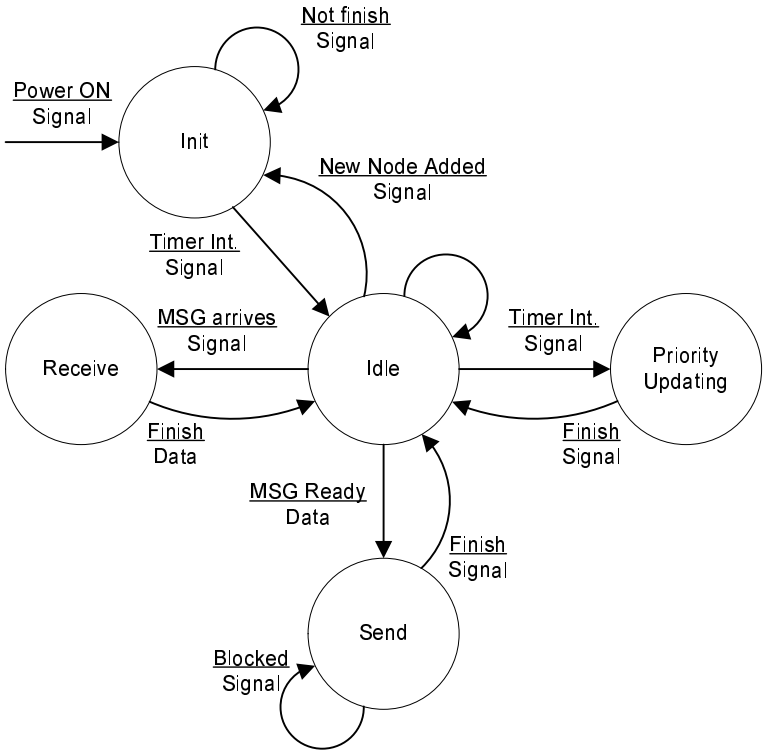

Figure 5.4: The STD of the local scheduler

nodes to reply and send their information. Then every node will receive messages from all the other nodes in the network and check whether such information is in its NNL or not. If it already exists, the node will ignore this message. Otherwise, it will insert such information into its NNL according to the ascending sequence of deadlines and send out the information of its own as the reply. When every node finishes building its NNL, the procedure of initialization ends and the system could run. Suppose there are n nodes on the bus, $n(n-1)/2+1$ broadcast messages are needed to generate the initial NNL of every node. The algorithm for initialization procedure is shown in Figure 5.5.

In order to assist nodes on the bus to build NNL, the message information should be encapsulated into one frame to broadcast on the bus. The format of this broadcast message is shown in Figure 5.6.

Since the last 5 bits in the arbitration field act as the unique identifier of a node, the first 6 bits of the broadcast message are blank, i.e. dominant, so that such messages could get the highest priority to access to the network. The data

Input: Message information: $info\_local$, $info\_new$;
Output: NNL; /* The Network Node List */

Method:

```
purge(NNL); /* Empty the old NNL table */
insert(info_local, NNL);
send(broadcast, info_local);

while (! init_finish) {
    /* init_finish is a boolean variable to indicate */
    /* the end of the initialization process */

    if (new_node_added()) {
        insert(info_new, NNL);
        send(reply, info_local);
    }
}
return NNL;
```

Figure 5.5: The algorithm of Initialization procedure

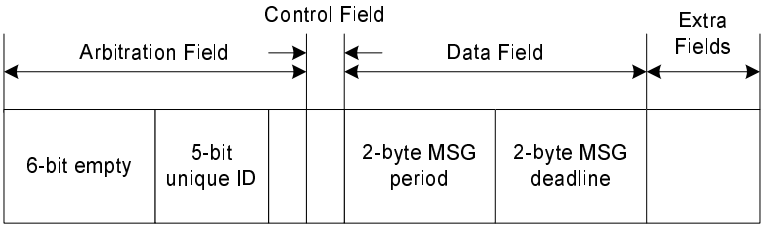

Figure 5.6: The format of broadcast message

field contains the message information: the first 2 bytes represent the period of a certain kind of message, and another 2 bytes represent its relative deadline. If one bit represents one millisecond, the minimal relative deadline will be $1ms$, and the maximum value will be $2^{16} = 64Kms$, which is enough for most practical applications on industrial networks. The total length of a broadcast message is 79 bits. If the task set in the bus needs more precise time resolution, we can use 4 bytes to represent the period and deadline, and change the time unit to one or several microseconds.

When a new node is added into the bus after the initialization, this node will send out a broadcast message first. When the other n nodes get the message, they

will send a reply message, just like the procedure of initialization. And they do this only after they are idle so as to avoid interrupting their current work. Since those n nodes have the information of each other, every one of them needs to send the reply message only once. The total number of messages is $2n + 1$.

Lag time

Since the network environment is distributed, it is not easy to determine when the procedure of initialization will finish and nodes should begin their practical data transmission. However, a simple estimation model can be built according to the specification of DLP scheme.

Suppose that the bus transmission rate is νMbps, the length of a broadcast message is l bits, and there is n nodes on the bus. In *initialization stage*, the maximum number broadcast messages is $n(n-1)/2 + 1$. Therefore the corresponding broadcasting time should be

$$T_i = \frac{l}{\nu} \cdot \left[\frac{n(n-1)}{2} + 1 \right] \ \mu s.$$

If node $n + 1$ is added into the bus, $2n + 1$ broadcast messages will be generated on the bus. So the corresponding transmission time should be

$$T_n = \frac{l}{\nu} \cdot (2n + 1) \ \mu s.$$

Suppose *lag time* is the time period of a node that the node should wait after booting up to build NNL list, we get

$$T_{lag} = T_b + T_i,$$

where T_b is the boot-up time of device controller, which is dependent on the design of the controller and may be different from one another. Generally speak-

ing, device controllers with operating system inside need more time to boot up. For example, if the microcontroller is in x86 architecture and uses MS-DOS, it needs normally several seconds to boot up, load OS kernel and begin to run application program. However, if no operating system is used and firmware is in internal/external EEPROM or flash memory, the boot-up time is of the order of microseconds. For example, boot-up time of Zilog microcontroller series is 18 system clocks [40]. When the external clock is 20MHz, the boot-up time is less than $4\mu s$. And nowadays in industrial market, CAN controller is normally integrated into simple microcontrollers in non-x86 architecture, so that T_b has very small contribution to T_{lag}.

As we mention above, in DLP case, 5 out of 11 bits in Arbitration Field (CAN2.0B standard format) are used to represent the unique identifier of a node, so the maximum number of nodes on the bus is limited to 32. Give $\nu = 1Mbps$, $l = 79bits$, we get $T_i = 39.184ms$, $T_n = 4.977ms$. Since T_b is in millisecond level, we can let $T_{lag} = 100ms$ for every node, which is safe enough for nodes to build their local NNLs.

5.3.3 Priority updating

For periodic messages, after the procedure of initialization, every node has the complete information in its NNL. The NNL of a node is shown in Table 5.2, which assigns priorities to items according to their deadlines. The item with shortest deadline will get the first position in it, and $\forall\ 1 \leq i < j \leq n$, we have $d_i \leq d_j$.

Table 5.2: NNL in a node

No.	Period	Deadline	Priority
1	T_1	d_1	1
2	T_2	d_2	2
$\cdots$	$\cdots$	$\cdots$	$\cdots$
n	T_n	d_n	n

Suppose the nodes update the priorities locally at every fixed *slice time* l. When

updating the NNL, $d_i^* = d_i - l$ will be computed as the new deadline of message τ_i. If $d_i^* < 0$, then let the new deadline be $d_i^* = d_i^* + T_i$. Then the item slides in the table to arrive at the new position, which is corresponding to its new priority. When the priority updating process finishes, the node will update the priorities of corresponding local messages in the queue or buffer. Since the maximal number of nodes in the bus is limited, simple insert algorithms can be used. And in order to improve the efficiency, the operation to NNL is always from the bottom to the top, i.e. from item n to item 1 of the table. The basic algorithm for priority updating is shown in Figure 5.7.

Input: NNL, l; /* The old NNL and slice time */
Output: NNL; /* The updated NNL */

Method:

```
i = number(NNL); /* Get how many items in NNL */
while (i > 0) {
    for (every item τ_i in NNL) {
        copy(τ_i^*, τ_i);
        d_i^* = d_i - l;
        if (d_i^* < 0)
            d_i^* = d_i^* + T_i;
        delete(τ_i, NNL);
        insert(τ_i^*, NNL);
    }
}
return NNL;
```

Figure 5.7: The algorithm of Priority Updating procedure

The DLP scheme copes with the priority assignment matter only, and does not affect the arbitration mechanism of CAN bus. It does not carry out the schedulability test of given message set. However, it can work together with scheduling algorithms to guarantee the schedulability. But higher CPU load is needed.

Since the priority updating mechanism is implemented in individual nodes on the bus, we must guarantee that all the distributed nodes are synchronized. Considering the hardware and other differences between nodes, an external clock source is a better choice to provide the universal clock synchronization signals. A sim-

ple algorithm proposed in [12] can synchronize clocks by using limited network bandwidth.

5.3.4 Implementation

Current CAN controller chipsets are designed for fixed-priority scheduling, and normally the arbitration field of messages would not be modified once the system begins to run. Therefore, in order to use the algorithm proposed in this chapter, it is necessary to add extra features to the chipset. One way is to modify the hardware design of the chipset itself. That is straightforward but the cost is quite high. Currently chipsets with CAN feature are widely used, so that the cost to replace them with new chipset may be too high to be affordable. In addtition, many other problems may be introduced, such as compatibility and interoperability. However, there still have two methods to solve the problem.

1. Every CAN controller in a node works with another low-cost microcontroller (such as Intel MSC51 series), which acts as the bridge between CAN controller and the system controller and is transparent to both sides. Firmware runs in it to control the communication scheme of the node. The advantage is no any other extra load with be introduced into the controller of the node, and firmware is easy to update. The disadvantage is that new hardware is added into the system and minor modification of current design is needed;

2. With the assistance of current multitask real-time systems, software solution could be implemented. In the system kernel, a single process keeps running and acts as the handler to cope with tasks related to message sending, receiving, priority updating and NNL initialization. By optimizing the parameters of the handler, it is expected to keep the overhead of local CPU under an acceptable bound. The advantage is no hardware modification is needed. But system load will be increased, and normally the upgrade to real-time operating system costs a lot.

5.4 Summary

In this chapter several issues related to limited priority levels in practical applications are discussed. First, the effect of the deterioration of schedulability is given. Second, problems of priority inversion due to limited priority levels is discussed. Lastly, an effective priority mapping scheme is proposed and analyzed to utilize the limited priority levels sufficiently.

The next chapter will give the simulation of CAN bus scheduling and the analysis of the result.

Chapter 6

Simulation and Result Analysis

This chapter presents the implementation and simulation results of the algorithms discussed in the previous chapters. The results focus on the performance of the algorithms in a network environment implemented by CAN bus.

Two ways can be used to test the schedulability of a message set: using the schedulability conditions and the simulation of the working of the network. The following parts of this section gives the experimental results under the above two tests separately.

6.1 The schedulability conditions

In the following experiment, unless otherwise stated, the experimental data set is generated according to the criteria in [9]. Suppose the bitrate of the CAN network is 250Kbps and the packet size is fixed at 8 bytes. The parameters of the first experimental environment are shown in Table 6.1.

There are 1000 message sets per testing point (12000 for total), and each message set consists of messages in three categories: periods in the range $(3ms, 12ms)$, $(30ms, 120ms)$ and $(250ms, 1s)$, uniformly distributed. The ratios of deadline to period are Gaussian distributed with $\mu = 0.8$ and $\delta = 0.2$. The following tests

Table 6.1: The data set information

Timestamp	1036612124
Bitrate	250 Kbps
Testing points	12
lowest	5%
delta	5%
Number of message sets	1000
Number of messages per set	30
Length of data field	64 bits (8 bytes)
Length of other fields	47 bits
Ratio of Deadline to Period	60%–100%

ignore sporadic messages, but it is easy to implement if we use minimum interval time (MIT, see chapter 3) as their period.

6.1.1 Deadline Monotonic

Figure 6.1 and Figure 6.2 show the differences of the three schedulability condition tests for DM algorithm discussed in Chapter 3. Please note that the network utilization shown on the x-axis is only the actual data utilization, while the real network utilization is higher because of the protocol overhead of CAN bus.

Figure 6.1 shows the case when preemption is allowed. Here message frames are still in CAN format, but the native arbitration mechanism in CAN protocol is not used. That means that a message with a higher priority than current bus holder's priority could get the bus immediately, and after its transmission, the suspended message could resume. No extra overhead is introduced. The curves marked with 'Basic', 'Advanced' and 'Iterated' are calculated by equation 3.3, equation 3.4 and equation 3.6, separately. We can see that the differences of the ratio of schedulable message sets of three algorithms are obvious. At the first several testing points, percentages of schedulable sets calculated by DM iterated algorithm is more than 20% higher than the percentages of schedulable sets in corresponding points calculated by DM basic algorithm and more than 15% higher than those calculated by DM advanced algorithm.

When preemption is not allowed, the simulation result are shown in figure 6.2.

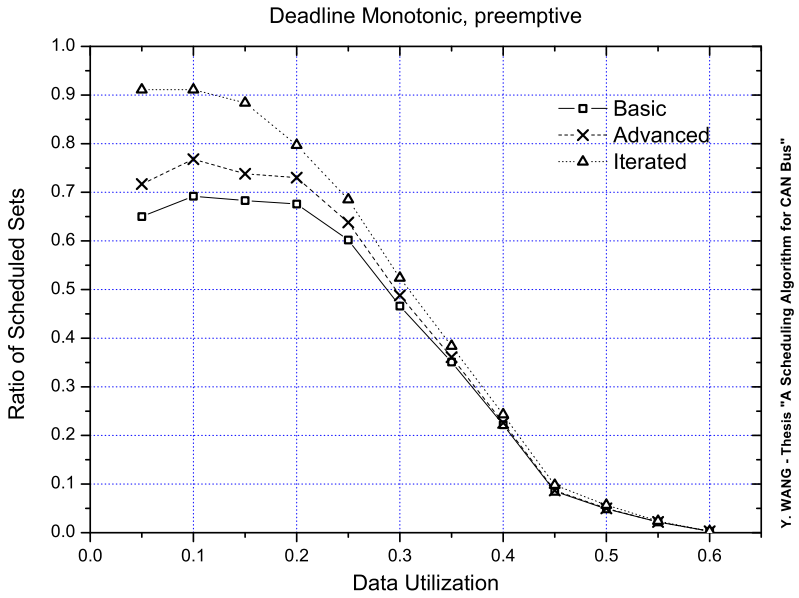

Figure 6.1: Percentage of schedulable message sets using DM, preemptive is allowed.

From the figure we can see that the performance of DM iterated algorithm becomes even better. The percentage of schedulable sets of DM iterated algorithm is even 30% higher than the other two.

One point needed to be explained: in both Figure 6.1 and Figure 6.2, we can find that from the curves, the percentage of schedulable sets at the testing points with low data utilization may be lower than that on points with high data utilization. This is due to the difference between data utilization and the real network utilization. For example, if the length of a data stream is 1 bit, when encapsulating the data stream into CAN frame, if variable length of data field is allowed, the data field will be 8 bits long (1 byte), otherwise the data field will be even 64 bits long (8 bytes). And the message set with low data utilization always has many data streams with very short data (less than 64 bits long). Therefore the real network utilization may be higher.

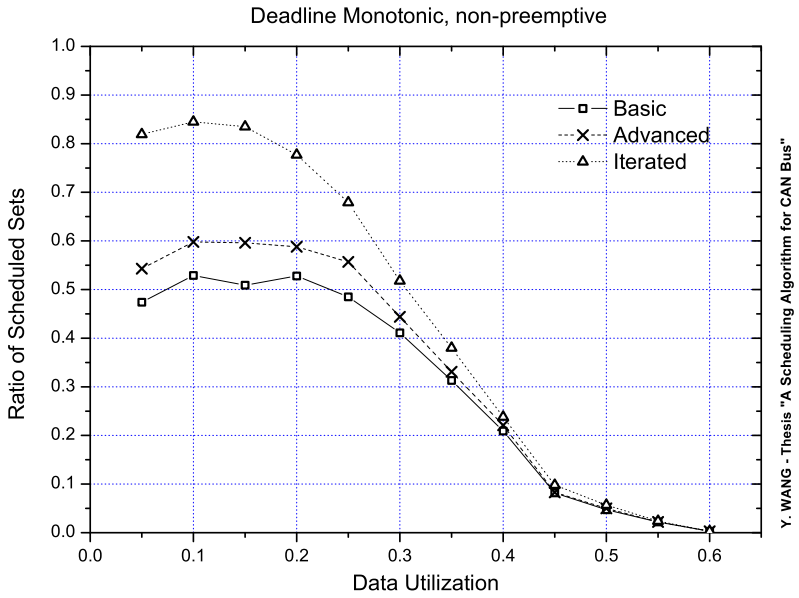

Figure 6.2: Percentage of schedulable message sets using DM, preemptive is not allowed.

6.1.2 Earliest Deadline First

We carry out two EDF schedulability condition tests, traditional method calculated by Equation 4.3 and the algorithm in Figure 4.3. Table 6.2 shows the difference of the results calculated by the above two methods. From the table we can see that percentage of schedulable sets calculated by the new schedulability condition is a little bit lower than that by traditional algorithm. Such a result is expected, since the traditional algorithm takes a necessary but not sufficient condition test, which may enlarge the range of schedulable sets. Meanwhile the new condition takes a necessary and sufficient schedulability test, which is more accurate.

Table 6.2: The difference between the results of two EDF algorithms.

Algorithm	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%
Traditional	0.956	0.943	0.92	0.855	0.775	0.653	0.538	0.386	0.211	0.141	0.075	0.024
New method	0.956	0.943	0.918	0.854	0.773	0.650	0.534	0.379	0.208	0.138	0.072	0.023

Figure 6.3 shows the difference of the results between non-preemptive DM and non-preemptive EDF algorithms. We can see that when the data utilization is

equal to or less than 0.50, the percentage of schedulable sets by EDF algorithms is about 10% to 15% higher than that by DM algorithm. Alternatively, if a given percentage of schedulable sets is considered, the data utilization is improved by 5% to 10% in the range 0.2–0.5.

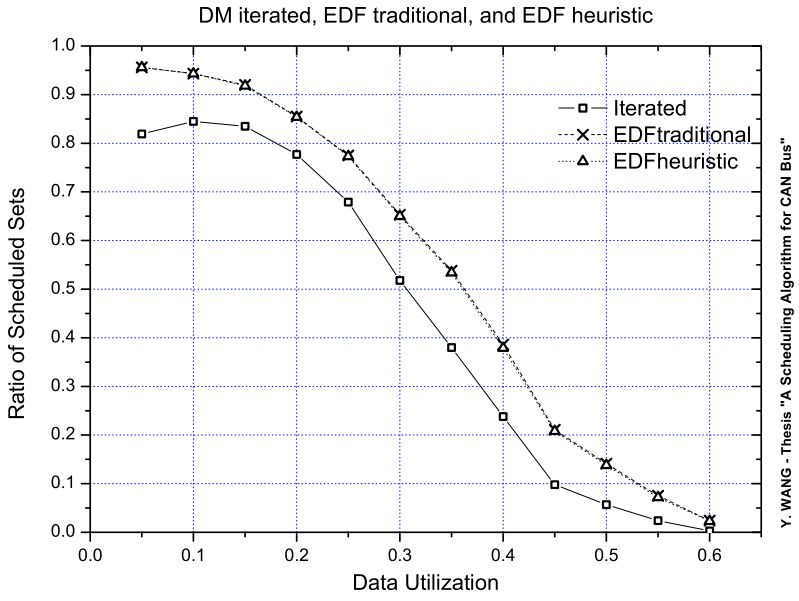

Figure 6.3: The difference between DM iterated, EDF traditional, and EDF heuristic, non-preemptive

6.2 The scheduling simulation model

The scheduling simulation model is shown in Figure 6.4. At least two methods could be used to implement the model. The best simulation method is to set up a real network via CAN bus. Several nodes are connected to the bus and simulation programs are running on each node to collect information of the scheduling process. However, due to the hardware cost and other uncertain factors, a pure software platform could be used to simulate the running of the network as an alternative solution. Although the results of software simulation may be not as accurate as those from the real network, the performance of the proposed algorithm could be

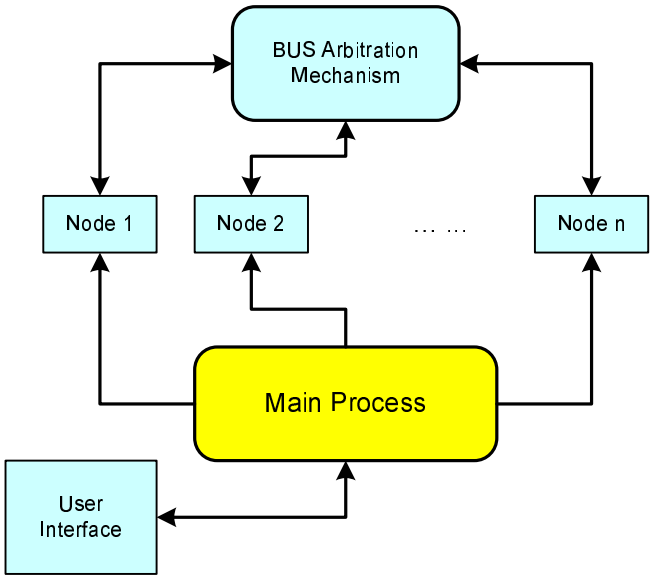

Figure 6.4: The simulation model

assessed by comparing the results of different algorithms in the same simulation environment.

Table 6.3: The simulation data set, workload of drilling machine

High-speed messages				
Type	*Class*	*Period/MIT*	*Deadline*	*# of message*
Fingers	Periodic	125.0μs(8kHz)	50.0μs	4
Joints	Periodic	166.7μs(6kHz)	66.6μs	6
Carriage	Periodic	250.0μs(4kHz)	100.0μs	2
Drill	Periodic	500.0μs(2kHz)	200.0μs	2
Sensors	Aperiodic	2s	30.0μs	2

The scheduling simulator, `stsimu`, is written in pure C/C++, console-based and therefore platform-independent. In this simulator, no system timer is used. Instead, a global counter works as the bus clock, hence the speed of simulation is increased greatly. An open framework for scheduling simulation is developed, so it is easy to add more modules to test some other scheduling algorithms. In order to test the correctness of the implementations of scheduling algorithms in `stsimu`, we use the data set in [41], which is the workload of a drilling machine with an attached robot arm to move workpieces as in table 6.3, to do the simulation before we use the data sets of our own. And we get similar results with those in [41].

Therefore, we can guarantee that the implementations of scheduling algorithms in the thesis are correct.

6.2.1 The structure of the scheduling simulator

The scheduling simulator is composed of several independent modules. Because of the console-based user interface, the `main()` function of the program works as a parser to analyze the input parameters, and then call the simulation procedure `st_simu()` (Figure 6.5).

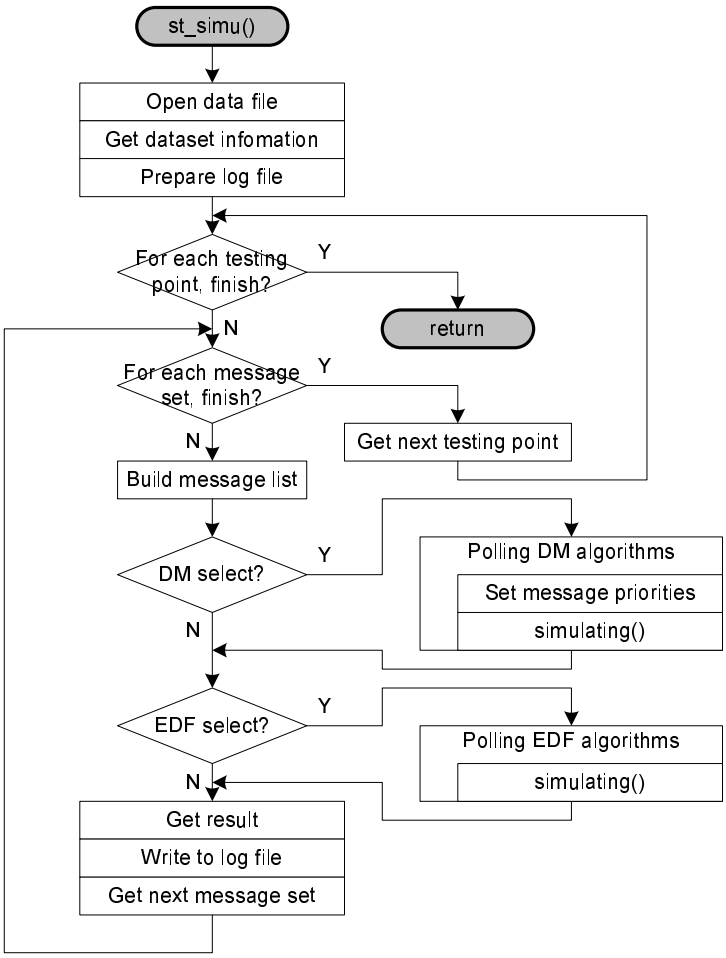

Figure 6.5: The data flow diagram of procedure `st_simu()`

The procedure `st_simu()` open the data file and gets the data set information. After polling the input parameters to find out which algorithm is required to run. the procedure tests the schedulability of the message set one by one for every testing

point by calling procedure `simulating()`. And the result will be stored in log file.

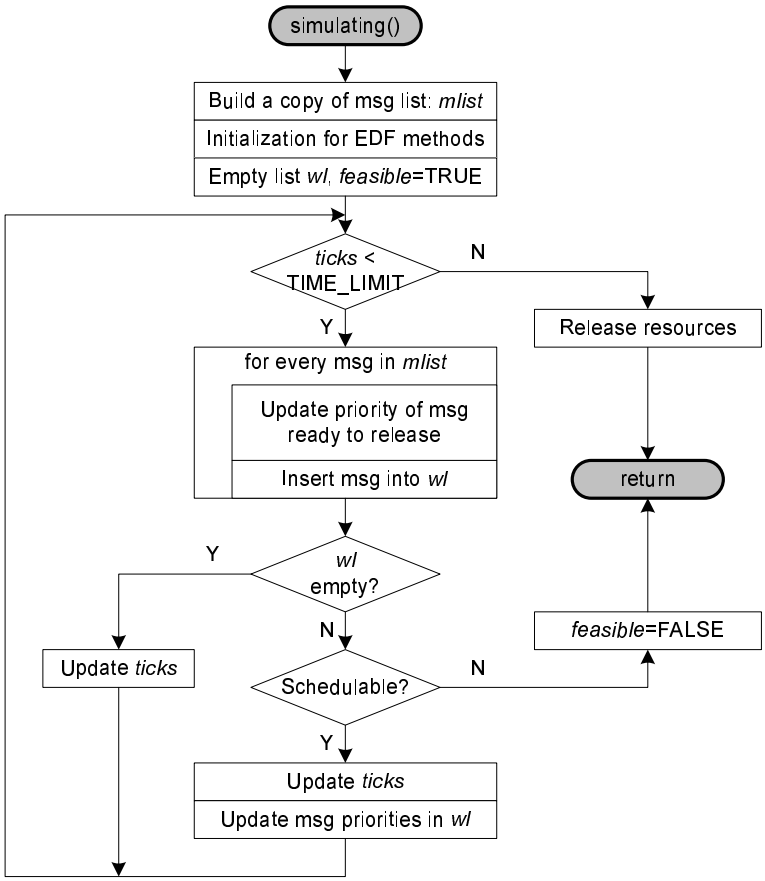

Figure 6.6: The data flow diagram of procedure `simulating()`

Figure 6.6 shows the diagram of procedure `simulating()`. This procedure tests the given message set in the range of $(0, \texttt{TIME\_LIMIT})$ and returns a boolean value to indicate whether the message set is schedulable or not. The procedure can update priorities dynamically if the simulation is for EDF algorithms.

6.2.2 Simulation results

The first example of simulation uses the data set shown in Table 6.1. The horizonal axis represents the data utilization. Figure 6.7 shows the simulation result of percentages of schedulable sets under different DM and EDF priority assignment schemes.

When Deadline Monotonic Algorithm is applied, we consider the case that the

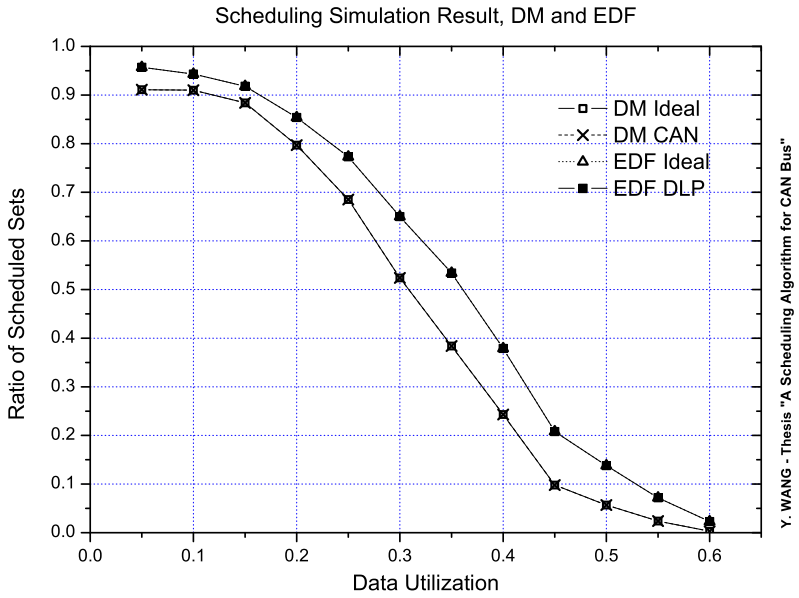

Figure 6.7: The comparison between DM and EDF

number of priority levels is unlimited (DM ideal) and the case that it is limited when simulating in CAN bus (DM CAN). According to [41], we use 9 bits of identifier as the priority. In Chapter 5 we discussed the effect of limited number of priority levels, and when more than 64 priority levels are applied the deterioration of performance is trivial. The simulation results in this given data set support the above analysis: there are no difference between DM ideal and DM CAN as shown in Figure 6.7. However, the performance is limited by the DM itself. The percentage of schedulable sets of DM is 10% to 15% less than that of EDF. The result of EDF under DLP mechanism is as good as that of EDF ideal (no limit of priority levels) from Figure 6.7. DLP mechanism maintains a priority table locally, and the number of priority levels is only limited by the allocated memory to store the table. Generally speaking, it is unlimited too.

In the second example, the data set uses network utilization, which can reflect the closer effects of real network environments, instead of data utilization used in the first example. In this data set, there are 21 testing points on the axis of network utilization, from 60% to 100% with step of 2%. And the ratio of deadline to period

is Gaussian distributed with $\mu = 0.5$ and $\delta = 0.1$. Other information is same with that in table 6.1. The simulation result is shown in Figure 6.8.

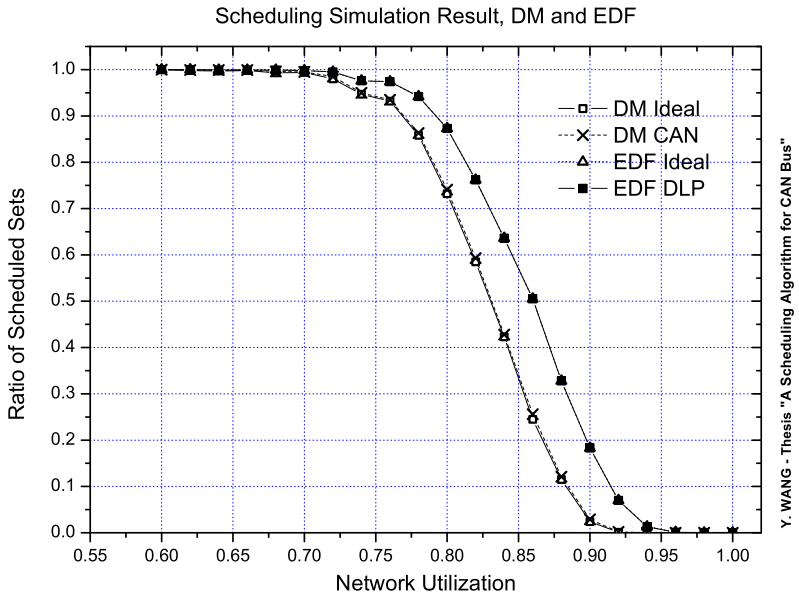

Figure 6.8: The comparison between DM and EDF

We can see that when network utilization is lower than 0.70, there is almost no difference between DM and EDF algorithms. With the increase of network utilization, the percentage of schedulable sets by DM algorithms drops much faster than that by EDF algorithms. Particularly, when network utilization is equal to 86%, the percentage of schedulable sets by EDF is as twice as that by DM. And it is shown in Figure 6.8 that when network utilization is applied, DLP algorithms still have the same result with EDF ideal.

6.2.3 Analysis of DLP scheme

The comparison of the performance of different priority assignment schemes under EDF is workload-dependent. The most persuasive test is practical workload in real world. We choose the data set which is generated according to [9]. The simulation results are shown in Figure 6.9, which uses data utilization, and Figure 6.10, which

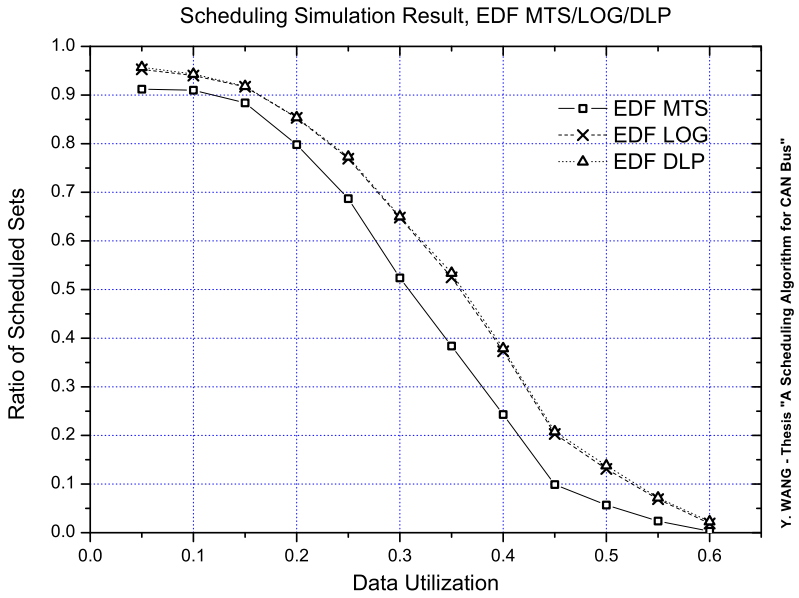

Figure 6.9: Comparison of three EDF algorithms, data utilization

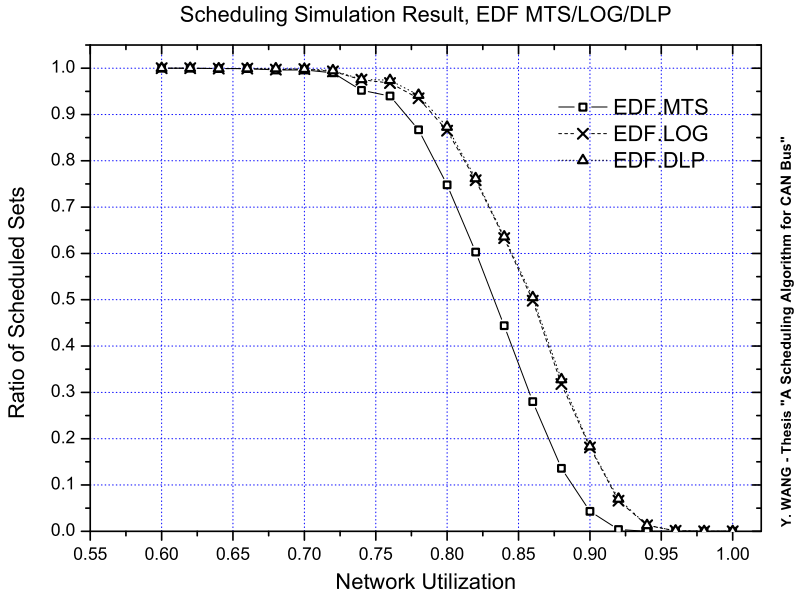

Figure 6.10: Comparisons of three EDF algorithms, network utilization

uses network utilization. In each figure there are three curves using priority assignment of EDF MTS, EDF by logarithmic quantization and EDF DLP, separately. We can see that the percentage of schedulable sets by EDF MTS is lower than that

by EDF LOG and EDF DLP in both figures. The EDF LOG is very close to EDF DLP but still a little lower.

The overhead of the EDF LOG algorithm is mainly due to the real-time calculation based on logarithmic temporal quantization to update the deadlines of messages. And such burden to CPU always exists. For the DLP algorithm, the overhead mainly exists in initialization process, because the real-time calculation to update deadlines in the DLP algorithm is much simpler than that in the EDF LOG algorithm. Therefore, the average load to CPU under the DLP algorithm should be less than that under the EDF LOG algorithm when other conditions are same.

However, the above simulation result is based on the ideal condition by ignoring all jitter, content-switching and other overhead. Under real network conditions and environments, the performance may be not as high as we have in the experiment. And the factors that restrain the application of DLP mechanism may be the scale of the network, network synchronization and the initialization process of the system. For example, there is still some work to do to optimize the initialization process to decrease the initialization time of NNL.

6.3 Summary

In this chapter, a simulation model is set up and the results of simulation are analyzed. Several scheduling algorithms, including DM, EDF and their derivations, are compared under preemptive and non-preemptive environments, especially in CAN bus. In the end, the analysis of DLP mechanism is given and shows that DLP is an effective mechanism for priority assignment.

In the next chapter, the conclusion and a discuss on further work is proposed.

Chapter 7

Conclusion

Issues of message scheduling algorithms in CAN bus are discussed in this thesis. Priority-driven scheduling algorithms are very popular in CAN bus and its native arbitration mechanism is a fixed priority scheme. Several important priority-driven scheduling algorithms are introduced, including fixed priority algorithms (RM/DM) and dynamic priority algorithms (EDF/MLF). And the schedulability condition tests are studied. As an improvement of non-preemptive EDF scheduling test, a new schedulability condition is proposed. Meanwhile, the priority mapping problem is studied thoroughly. In order to reduce the effect of limited priority levels and corresponding priority inversion, a new priority assignment scheme, dynamic local priority (DLP), is proposed.

In order to check the models proposed in this thesis, software platforms are constructed. From the results and analysis in chapter 6 we can find that both the new schedulability condition and the DLP scheme reach the level that we expect. The heuristic algorithm is shown to satisfy the sufficient and necessary schedulability condition. Unlike the schedulability condition tests of DM and EDF shown in [39], which contain some approximate upper-bound items, the heuristic algorithm can give a precise conclusion whether a set of messages is schedulable or not. DLP scheme introduces a local priority table, and hence it solves the problem of limited number of priority levels. Compared with MTS and EDF logarithmic

priority assignment schemes, the performance of DLP scheme is closer to that of ideal EDF. And the implementation of DLP scheme is simple. Only an extra software layer is needed between the application layer and the network layer (MAC layer).

However, there are some restrictions in the work. First, using LCM of the data set as t_{max} may not economical and the computation work may be very heavy. Second, the data set in experiments is generated randomly and may be not proper in practical distributed systems. Therefore, a more accurate model to generate data set for experiments is needed. Third, the DLP scheme is tested only on a software platform, not on a real network environment. All overloads are neglected. The real performance may be not as good as that in simulation results. If conditions permit, an experiment in real network environment is necessary to test the practical efficiency of DLP scheme.

The further work includes two aspects: first, do more experiments under different conditions to test the algorithms; second, try to implement the proposed ideas in this thesis to a more general network environments. Since the rapid pace of development in computer and network technology, more and more new techniques and interface standards occur in recent years, such as IEEE 1394 and USB 2.0. Those two communication interfaces and corresponding protocols could provide very high transmission speed (maximum speed 400Mbps and 480Mbps, respectively) without adding too much additional cost. It is necessary to keep an eye on such new techniques and their applications in industrial fields.

References

[1] Aras, C. M., J. F. Kurose, D. S. Reeves and H. Schulzrinne. Real-Time Communication in Packet-Switched Networks. *Proceedings of IEEE*, 82(1):122–139, January 1994.

[2] Audsley, A. N., A. Burns, M. Richardson and K. Tindell. Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling. *Software Engineering Journal*, 8(5):284–292, September 1993.

[3] Audsley, N. C. Deadline Monotonic Scheduling. Technical Report YCS 146, Dept. of Computer Science, Univ. of York, 1990.

[4] Audsley, N. C., A. Burns, M. F. Richardson and A. J. Wellings. Hard Real-Time Scheduling: The Deadline Monotonic Approach. In *Proceedings 8th IEEE Workshop on Real-Time Operating Systems and Software*, Atalanta, 1991.

[5] Bosch. *CAN Specification Version 2.0*. Robert Bosch GmbH, 1991.

[6] Burns, A., M. Nicholson, K. Tindell and N. Zhang. Allocating and Scheduling Hard Real-Time Tasks on a Point-to-Point Distributed System. In *Proceedings of The Workshop on Parallel and Distributed Real-Time System*, pages 11–20, April 1993.

[7] Cena, G. and A. Valenzano. An Improved CAN Fieldbus for Industrial Applications. *IEEE Transactions on Industrial Electronics*, 44(4):553–564, August 1997.

[8] Cena, G. and A. Valenzano. FastCAN: A High-Performance Enhanced CAN-Like Network. *IEEE Transactions on Industrial Electronics*, 47(4):951–963, August 2000.

[9] Di Natale, M. Scheduling the CAN bus with Earliest Deadline Techniques. In *Proceedings of 21st IEEE Real-Time Systems Symposium*, pages 259–268, November 2000.

[10] Farsi, M., K. Ratcliff and M. Barbosa. An Overview of Controller Area Network. *Journal of Computing & Control Engineering*, June 1999.

[11] Fonseca, J. A. and L. M. Almeida. Using a Planning Scheduler in the CAN Network. In *Proceedings of 7th IEEE International Conference on Emerging Technologies and Factory Automation*, pages 815–821, 1999.

[12] Gergeleit, M. and H. Streich. Implementing a Distributed High-Resolution Real-Time Clock using the CAN-Bus. In *Proceedings of 1st International CAN Conference*, September 1994.

[13] Gomaa, H. A Software Design Method for Real-Time Systems. *Communications of the ACM*, 27(9):938–949, September 1984.

[14] Gomaa, H. Software Development of Real-Time Systems. *Communications of the ACM*, 29(7):657–668, July 1986.

[15] Intel. *82527 Serial Communications Controller (Controller Area Network Protocol)*. Intel Corporation, 1996.

[16] Joseph, M. and P. Pandya. Finding Response Times in a Real-Time System. *BCS Computer Journal*, 29(5):390–395, October 1986.

[17] Katcher, D. I., S. S. Sathaye and J. K. Strosnider. Fixed Priority Scheduling with Limited Priority Levels. *IEEE Transactions on Computers*, 44(9):1140–1144, September 1995.

[18] Kopetz, H. and G. Grünsteidl. TTP - A Time-Triggered Protocol for Fault Tolerant Real-Time Systems. In *Proceedings of IEEE CS 23rd Fault Tolerance Computing Symposium*, pages 524–533, June 1993.

[19] Krishna, C. M. and K. G. Shin. *Real-Time Systems*. The McGraw-Hill Companies, Inc., 1997.

[20] Lehoczky, J., L. Sha and Y. Ding. The Rate Monotonic Scheduling Algorithm: Exact Characterization And Average Case Bahavior. In *Proceedings of 10th IEEE Real-Time Systems Symposium*, pages 166–171, Decemember 1989.

[21] Lehoczky, J. P. and L. Sha. Performance of Real-Time Bus Scheduling Algorithms. *ACM Performance Evaluation Review*, 14:44–53, 1986.

[22] Leung, J. Y. T. and J. Whitehead. On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks. *Performance Evaluation*, 2(4):237–250, 1982.

[23] Liu, C. J. and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM*, 20(1):46–61, January 1973.

[24] Liu, J. W. S. *Real-time Systems*. Prentice-Hall, 2000.

[25] Livani, M. A., J. Kaiser and W. Jia. Scheduling Hard and Soft Real-time Communication in a Controller Area Network. *Control Engineering Practice*, 7(12):1515–1523, December 1999.

[26] Motorola. *MC68HC912BC32 16-Bit Microcontroller Technical Summary*. Motorola Inc., 2nd edition, 1997.

[27] Motorola. *AN1798: CAN Bit Timing Requirements*. Motorola Inc., 1999.

[28] Patzke, R. Fieldbus Basics. *Computer Standards & Interfaces*, 19(5–6):275–293, October 1998.

[29] Pedreiras, P. and L. Almeida. A Practical Approach to EDF Scheduling on Controller Area Network. In *Proceedings of 2001 IEEE Real-Time Systems Symposium Workshop*, December 2001.

[30] Rehg, J. A., W. H. Swain, B. P. Yangula and S. Wheatman. Fieldbus in the Process Control Laboratory–Its Time Has Come. In *29th Frontiers in Education Conference*, volume 3, pages 13B4/12–13B4/17, 1999.

[31] Schumny, H. Fieldbuses in Measurement and Control. *Computer Standards & Interfaces*, 19(5–6):295–304, October 1998.

[32] Scott, A. and W. Buchanan. Truly Distributed Control Systems Using Fieldbus Technology. In *Proceedings of 7th IEEE International Conference and Workshop on Engineering of Computer Based Systems*, pages 165–173, Edinburgh, UK, April 2000.

[33] Stallings, W. *Operating Systems*. Prentice-Hall, 3rd edition, 1998.

[34] Tanenbaum, A. S. *Distributed Operating Systems*. Prentice-Hall, 1995.

[35] Tanenbaum, A. S. *Computer Networks*. Prentice-Hall, 3rd edition, 1996.

[36] Tindell, K. W., H. Hansson and A. J. Wellings. Analysing Real-Time Communications: Controller Area Network (CAN). In *Proceedings of 15th IEEE Real-Time Systems Symposium*, pages 259–265, December 1994.

[37] Yang, X., editor. *Fieldbus and Applications*. Tsinghua University Press, 1st edition, 1999.

[38] Zhao, W. and K. Ramamritham. Simple and Integrated Heuristic Algorithms for Scheduling Tasks with Time and Resource Constraints. *Journal of Systems and Software*, 7:195–205, 1987.

[39] Zheng, Q. and K. G. Shin. On the Ability of Establishing Real-Time Channels in Point-to-Point Packet-Swtiched Networks. *IEEE Transactions on Communications*, 42(2/3/4):1096–1105, 1994.

[40] Zilog. *Z8 Family Design Handbook*. Zilog Inc., 1989.

[41] Zuberi, K. M. and K. G. Shin. Non-Preemptive Scheduling of Messages on Controller Area Network for Real-Time Control Applications. In *Proceedings of Real-Time Technology and Applications Symposium*, pages 240–249, May 1995.

Appendix A

Proof of Theorem 4

Theorem 4 *When deadline scheduling policy is applied, if preemption is allowed and messages are independent, a set of n messages* $\tau_i = (T_i, C_i, D_i), i = 1, 2, \ldots, n$, *are schedulable if and only if*

$$\begin{aligned} &1.\ \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1, \\ &2.\ \forall t \in S, \quad \sum_{i=1}^{n} \left\lceil \frac{t - D_i}{T_i} \right\rceil^{+} \cdot C_i \leq t \end{aligned} \tag{A.1}$$

where $S = \bigcup_{i=1}^{n} S_i$, $S_i = \{D_i + nT_i, n = 0, 1, \ldots \lfloor (t_{max} - D_i)/T_i \rfloor\}$, *and* $t_{max} = \max\{D_1, \ldots D_n, (\sum_{i=1}^{n}(1 - D_i/T_i)C_i)/(1 - \sum_{i=1}^{n} C_i/T_i)\}$.

Proof: The condition 1 is the preliminary conditions of scheduling. $\sum_{i=1}^{n} C_i/T_i$ is the maximum total utilization of the message set. If it is greater than 1, the bus is always overloaded under whatever conditions.

To prove condition 2, we use Theorem 3. $\lceil (t - D_i)/T_i \rceil^{+} C_i$ is piecewise function, and changes only on the set $S_i' = \{D_i + nT_i, n = 0, 1, \ldots\}$, therefore, we only need to check the inequality of Theorem 3 on the set $S' = \bigcup_{i=1}^{n} S_i'$.

$\because \forall t \geq \max\{D_i, i = 1, \ldots, n\}$,

$$\lceil (t - D_i)/T_i \rceil^{+} \leq 1 + (t - D_i)/T_i.$$

Let

$$t' = \frac{\sum_{i=1}^{n} (1 - D_i/T_i)\, C_i}{1 - \sum_{i=1}^{n} C_i/T_i},$$

we get

$$\begin{aligned}
\sum_{i=1}^{n} \lceil t - D_i/T_i \rceil^+ C_i &\leq \sum_{i=1}^{n} (1 + (t - D_i)/T_i)) C_i \\
&= \sum_{i=1}^{n} (C_i/T_i) t + \sum_{i=1}^{n} (1 - D_i/T_i) C_i \\
&= \sum_{i=1}^{n} (C_i/T_i) t + (1 - \sum_{i=1}^{n} (C_i/T_i)) t' \\
&= t' + \sum_{i=1}^{n} (C_i/T_i)(t - t')
\end{aligned}$$

$\because \sum_{i=1}^{n} (C_i/T_i) \leq 1$, and if $t \geq t'$,

$$\sum_{i=1}^{n} \lceil t - D_i/T_i \rceil^+ C_i \leq t' + (t - t') = t.$$

$\therefore$ Let $t_{max} = \max\{D_1, \ldots, D_n, t'\}$, whenever $t \geq t_{max}$, inequality A.1 is always met. Therefore, the set of testing points becomes $S = \bigcup_{i=1}^{n} S_i$, $S_i = \{D_i + nT_i, n = 0, 1, \ldots \lfloor (t_{max} - D_i)/T_i \rfloor\}$.

□

Appendix B

Hardware-related Topics of CAN

In the CAN specification, no clear hardware-related definitions are given. Although different CAN controllers or integrated microcontrollers use different ways to implement, they all share common functionality from one controller to another. In this chapter, we use the CAN module of Motorola 68HC12BC32 microcontroller (MSCAN12) as an example to discuss these features [26].

B.1 Buffer storage

The message buffer could assist the communication process and reduce the cost of time. Different ways are implemented by different CAN controllers, integrated or stand-alone. For example, the Philips 82C200 CAN controller [37], which is a simple controller, has only two message buffers: a single 10 byte transmission buffer and a 10 byte double-buffered receive buffer. This controller works as a memory-mapped I/O device, raising interrupts to the microcontroller or accepting signals from it. In contrast, Intel 82527 CAN controller [15] uses dual-ported RAM to exchange data with the microcontroller, mapping permanently message identifiers to *slots*. And the analysis in [36] shows that Intel 82527 controller has a very much better worst-case timing performance than the Philips 82c200 controller.

MSCAN12 module utilize a sophisticated message buffer mechanism to imple-

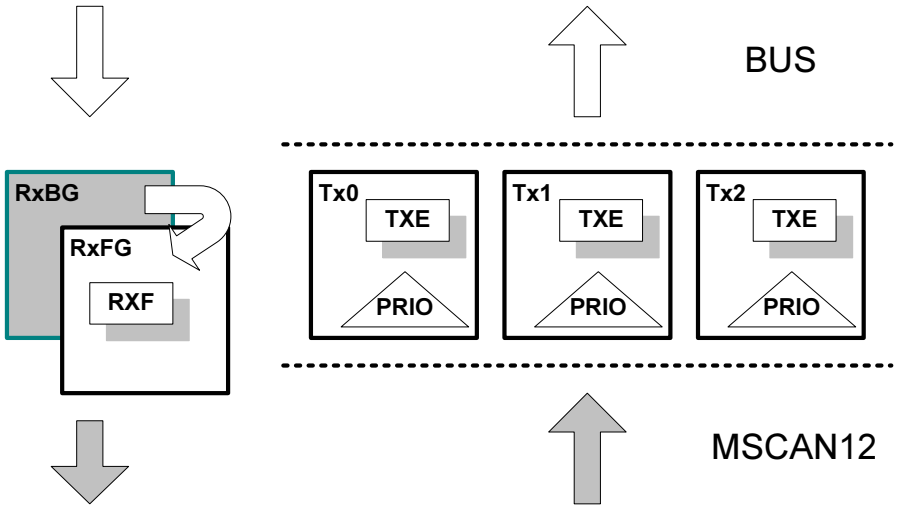

Figure B.1: The user model for message buffer organization of MSCAN12

ment the message storage. It has two receive buffers, which are mapped into a single memory area (see Figure B.1), and three transmit buffers. This scheme could guarantee that any node could send out a stream scheduled messages without releasing the bus between two messages, and multiple messages are allowed to be set up in advance in order to optimize the real-time performance.

B.2 Message filtering

A simple CAN controller has to receive all messages transmitted on the bus and leave the task of message selection to software. This can pose a heavy burden to the controller since every time a message is received, regardless of whether it is intended for the application or not, a software interrupt routine is invoked. This mechanism may cause a lot of code and processing overhead, resulting in deteriorating performance of the application or even the system.

One effective way to reduce such overheads is to provide an elementary message filtering mechanism to select messages automatically, accept the intended only and reject the rest. Message filtering is based on the whole identifier. There should be mask registers in a CAN controller that allow any identifier bit to take effect or to be set 'do not care' when filtering a message. Therefore, the controller could select

particular message groups or subsets by means of mask registers. Normally, CAN controllers should provide programmable mask registers. At least one identifier acceptance register and one identifier mask register are needed. In old CAN controllers, such as Philips 8x592 microcontroller and the Motorola 68HC05X family, the filter has only one 8-bit acceptance register and one corresponding 8-bit mask register. They could only filter messages of CAN2.0A standard format. In fact, since they have only eight bits in the mask register, the last 3 bits of the 11-bit identifier of standard format are ignored when doing the matching test. In MSCAN12 module of Motorola 68HC12BC32 microcontroller, on the contrary, there are eight 8-bit acceptance registers and eight 8-bit mask registers. Four identifier acceptance modes could be set to organize filter as two 32-bit, four 16-bit, eight 8-bit acceptance filters or disable filter, and therefore high flexibility could be achieved. Multiple-layer message filtering can be implemented in MSCAN12.

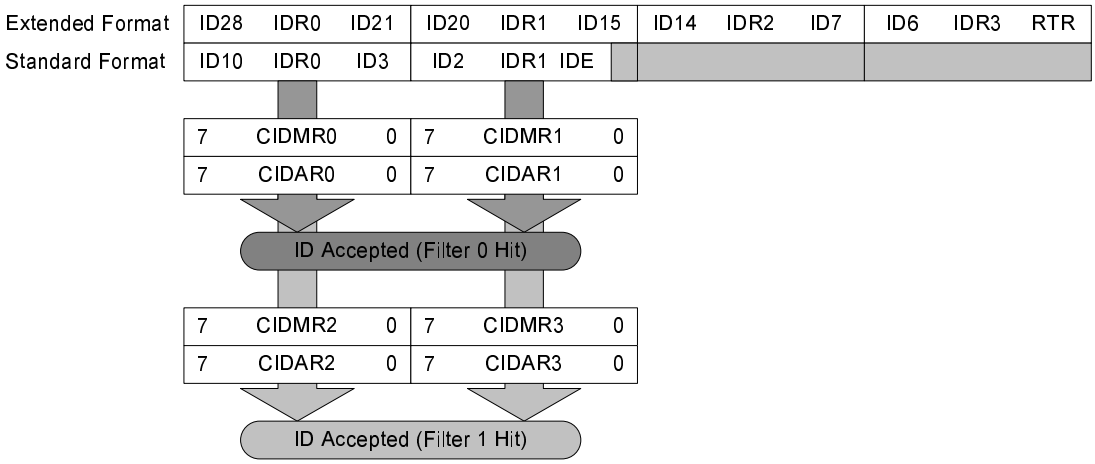

Figure B.2: 16-bit maskable acceptance filters in MSCAN12

Figure B.2 shows the case that four 16-bit acceptance filters are used, so as to verify the 11 bits of the identifier and the RTR bit of CAN 2.0A message, or the 14 most significant bits of the identifier of CAN 2.0B message. Meanwhile the figure shows how the first 32-bit filter bank (CIDAR0-3, CIDMR0-3) produces filter 0 and 1 hits. Similarly, the second filter bank (CIDAR4-7, CIDMR4-7) produces filter 2 and 3 hits. These hits could raise receive interrupt to inform the software that the intended message has arrived.

B.3 Bus timing

The Nominal Bit Rate of the network is given by:

$$f_{NBT} = \frac{1}{t_{NBT}}$$

where t_{NBT} is the Nominal Bit Time, the time to transmit a bit on the bus. t_{NBT} is divided into four separate non-overlapping time segments: SYNC_SEG, PROP_SEG, PHASE_SEG1 and PHASE_SEG2. Therefore, we get:

$$t_{NBT} = t_{SYNC\_SEG} + t_{PROP\_SEG} + t_{PHASE\_SEG1} + t_{PHASE\_SEG2}$$

Every segments above is an integer multiple of a unit of time called a *time quantum*, t_Q. A time quantum is the atomic unit of time handled by a CAN controller. Hence, t_Q is equal to the period of CAN system clock, which is derived from the microcontroller (MCU) system clock or oscillator by way of a programmable prescaler (Baud Rate Prescaler). Figure B.3 [27] shows the relationship between CAN system clock and the CAN bit period.

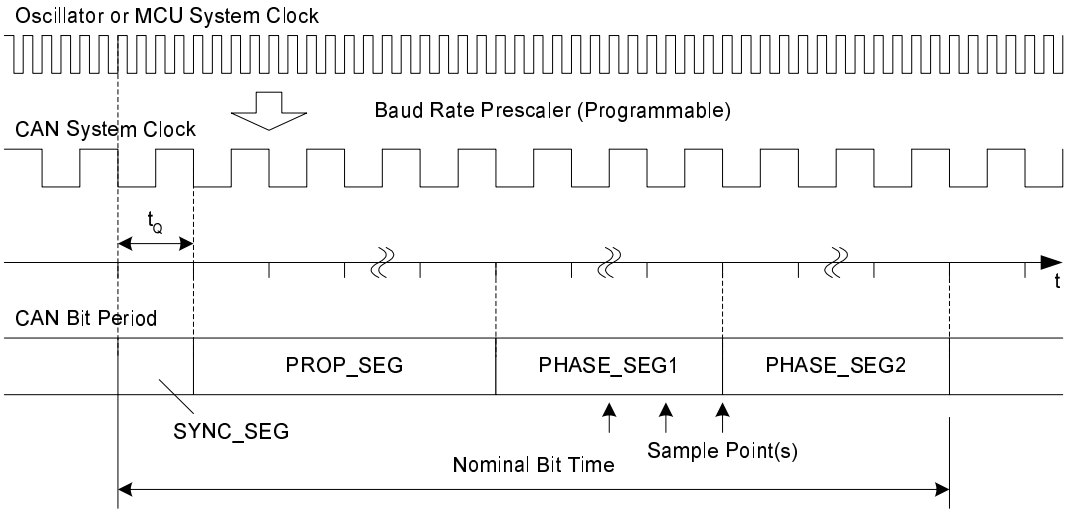

Figure B.3: the relationship between CAN system clock and CAN bit period

The duration of the segment SYNC_SEG is fixed to one t_Q. But the other

segments are programmable. Let

$$t_{SEG1} = t_{PROP\_SEG} + t_{PHASE\_SEG1}$$
$$t_{SEG2} = t_{PHASE\_SEG2}$$

The duration of the propagation segment PROP_SEG may be between 1 to 8 times of t_Q. If one sample per bit is selected, the duration of segment PHASE_SEG1 may be between 1 to 8 times of t_Q, and if three samples per bit are selected, it may be between 2 to 8 times of t_Q. The duration of the segment PHASE_SEG2 should be equal to the maximal value between PHASE_SEG1 and the Information Processing Time (IPT). Normally, the total number of time quantum in a bit time should be between 8 to 25.

The nominal start of a bit is the beginning of the SYNC_SEG segment. During this period, the incoming edge of a bit is expected. Due to the the non-destructive arbitration of CAN protocol and the requirement for in-frame acknowledgement, PROP_SEG segment is necessary to guarantee that nodes on the bus may not begin to sample the bus value until the transmitted bit values from all the transmitting nodes have reached all nodes. If it is needed to re-synchronize the bus timing, nodes could adjust the duration of PHASE_SEG2 segment. One parameter named Synchronization Jump Width (SJW) defines the number of time quanta that can be used to compensate for the phase shifts. With the minimum value of 1, the width can not exceed 4 t_Q and it should not exceed the duration of PHASE_SEG1 segment.

	BIT7							BIT0
CBTR0	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
CBTR1	SAMP	TSEG22	TSEG21	TSEG20	TSEG13	TSEG12	TSEG11	TSEG10

Figure B.4: Bus timing registers

In most CAN controllers, there are 8-bit registers to control the bus timing characteristics for CAN communication. The structures of the two bus timing

registers are shown in figure B.4. They could be used to set the prescaler, bit sample point, SJW, and the value of t_{SEG1} and t_{SEG2}. The bit SAML determines the sample point, at which point the nodes should sample the bus. If SAML=0 one sample is chosen. If SAML=1 then three sample points are chosen, and the most frequently sampled value is taken as the bit value.

The detailed references and examples of the calculation of bit timing parameters for MSCAN12 module can be found in [26] and [27].