The Society for engineering in agricultural, food, and biological systems

Paper Number: 021178
An ASAE Meeting Presentation

A Structural and Modular Approach to Implement Communication Interface For Tractor Electronics Communication Using CAN Kingdom

Jiantao Wei, Research Assistant
Naiqian Zhang, Professor
Ning Wang, Research Associate
Department of Biological and Agricultural Engineering

Donald Lenhert
Professor
Department of Electrical and Computing Engineering

Mitch Neilsen, Assistant Professor
Massaki Mizuno, Professor
Gurdip Singh, Associate Professor
Department of Computing and Information System
Kansas State University
Manhattan, Kansas, 66502

Written for presentation at the
2002 ASAE Annual International Meeting / CIGR XVth World Congress
Sponsored by ASAE and CIGR
Hyatt Regency Chicago
Chicago, Illinois, USA
July 28-July 31, 2002

Abstract. Modern tractors and implements will be equipped with an increasingly large number of electronic control units (ECU) together with a field bus for exchange of information. Two field bus standards, DIN9684 (Germany) and ISO11783 (ISO), both based on Bosch CAN (Controller Area network), have been developed to provide open systems to allow on-board-tractor ECUs built by different manufacturers to communicate with each other. These standards detail the necessary communication interface requirements, such as message types, identifier assignments, network management, etc. to enable a plug-and-play capability for these ECUs. However, the standards did not specify how to implement this communication interface, leaving it to individual ECU manufacturer. Given limited processing power and memory resources for embedded ECUs, development of distributed real-time control software conforming to ISO11783 or DIN9684 standard is a challenging task for engineering design teams, especially for small implement manufacturers. In this paper, a structural, modular software approach to implement the communication interface based on CAN Kingdom protocol was presented. This approach is illustrated by developing a DIN9684- conforming weed-sensing and spray-control system. It is also expected to work with the ISO11783 standard.

Keywords. CAN, DIN9684, ISO11783, CAN Kingdom, communication interface, modular, structural, weed sensor

The authors are solely responsible for the content of this technical presentation. The technical presentation does not necessarily reflect the official position of the American Society of Agricultural Engineers (ASAE), and its printing and distribution does not constitute an endorsement of views which may be expressed. Technical presentations are not subject to the formal peer review process by ASAE editorial committees; therefore, they are not to be presented as refereed publications. Citation of this work should state that it is from an ASAE meeting paper. EXAMPLE: Author's Last Name, Initials. 2002. Title of Presentation. ASAE Meeting Paper No. 021178. St. Joseph, Mich.: ASAE. For information about securing permission to reprint or reproduce a technical presentation, please contact ASAE at hq@asae.org or 616-429-0300 (2950 Niles Road, St. Joseph, MI 49085-9659 USA).

A Structural and Modular Approach to Implement Communication Interface For Tractor Electronics Communication Using CAN Kingdom

Jiantao Wei, Naiqian Zhang, Mitch Neilsen, Donald Lenhert
Ning Wang, Masaaki Mizuno,Gurdip Singh

1. Introduction

Controller Area Network (CAN) was developed by Robert Bosch GmbH with the support of Intel (Bosch 1991) in the 1980's. Although CAN was initially developed for the automotive industry, it also has been widely used in other industries such as process control, home automation, robotics, etc. It was estimated that more than 140 million CAN nodes have been installed worldwide. (Gabriel et al, 1999). The International Organization of Standard (ISO) documented CAN in a standard, ISO11893 for high-speed networking in 1993, and in ISO11519-2 for low-speed networking in 1994. (ISO, 1993; ISO, 1994)

CAN employs a contention-based, non-destructive medium access protocol to allow higher-priority messages to be transmitted before lower-priority messages. A CAN message consists of 11-bit (basic CAN) or 29-bit (extended CAN) identifier and up to 8 bytes of data load. The identifier part was used as access priority during bus contention as well as to identify a specific message. The CAN medium access and data link protocol was implemented in CAN controller hardware, which is now available from most of silicon manufacturers. A CAN controller is typically connected to the host processor via dual-port RAM, whereby the CPU and the controller can access the memory simultaneously. This area of memory typically was divided into a number of independent message slots (buffers). From programming perspective, a received message is copied into a corresponding slot by CAN controller, from where the CPU reads out the message. Similarly, a transmitted message is copied into a slot by CPU, and is read out by CAN controller and sent out to CAN bus.

CAN only defines the physical layer and data link layer according to the OSI 7-layer reference model. In practice, implementation of very simple CAN-based systems shows that, besides the two basic layers, further functionalities, such as identifier assignment, network startup, supervision of node etc., are desirable. For industrial automation applications, the need for an open, standardized higher layer, which support interoperability and exchangeability of devices from different manufacturers, was raised. In automotive industry, SAE J1939 was documented to be a higher level standard to govern car electronics communications (SAE, 1996). For sea borne target, NMEA 2000 is such a governing standard (NMEA, 2000).

In agricultural area, a requirement for such a standard was recognized as early as in 1986 in Germany. (Auernhamme, 1993) In 1998, a German standard, "LBS"- the Mobile Agricultural Bus document was published, and it was later standardized in DIN9684. Recently, the International Standards Organization working group ISO TC23/SC19/WG1 balloted on the ISO11783 standard (ISO, 2001). The DIN9684 standard will be abandoned and the ISO11783 standard will be the future-governing standard in agricultural electronics communications. ISO11783 was partially based on the DIN9684 standard, and partially based on the J1939 standard. Major agricultural machinery manufacturers, such as John Deere, AGCO, CASE, have developed proprietary CAN-based precision agriculture systems. Now they have taken steps to migrate their CAN systems to conform to the ISO11783 standard.

CAN Kingdom (Fredriksson, 1993) was developed by Kvaser, Sweden in 1990's, and it is a different kind of protocol from the above-mentioned standards. The functionalities of SAE J1939, NMEA 2000, DIN9684, and ISO11783 standards are very similar. All of them specify provisions such as device profiles, identifier distribution, and network management. CAN Kingdom (CK) is a set of protocol primitives that are commonly used in CAN communication. It gives system designers full control of the CAN network, and allows system designers and module designers work independently to a great extent.

One advantage of CK is that it can work with other open systems, such as J1939, DIN9684, and ISO11783. The main objective of this paper is to demonstrate how application of CK can be applied to help the development of a DIN9684 system.

As precision agriculture research and practice proceed, the ISO11783 standard will be more widely accepted. However, ISO11783 does not specify how to implement this communication interface, leaving it to individual manufacturer. Given limited processing power and memory resources of embedded ECUs, the development of distributed real-time control software conforming to ISO11783 or DIN9684 standard is a challenging task for engineering design teams, especially for small implement manufacturers. In this paper, a structural, modular software approach to implement the communication interface using CAN Kingdom primitives is presented. This approach is illustrated by developing a DIN9684- conforming weed-sensing and spray-control system. Our original intention was to build an ISO11783 system using CK. As ISO11783-conformant virtual terminal was not available at that time, we started to build our system based on DIN9684. We believe the approach we developed is expected to work with ISO11783 standard with minor modifications, since DIN9684 and ISO11783 are very similar standards.

2. DIN9684

In this section, we will briefly review the essential part of the DIN9684 standard including the message type definition and network management procedure.

(1). Message Types

Table 1: DIN9684 message types and their priority (identifier structure)

(Priorities) Identifier Bits 1-3	Identifier Bits 4-7	Identifier Bits 8-11	Message description
000	AGETYPOS		System functions A= 1: login; A= 0: system management GETY=Implement type, total 16 types POS=Implement mounting position, total 8 positions
001	0100	SEND	Basic data 1:driving velocity and traveled distance
	0101	SEND	Basic data 2: PTO speed, engine speed, hitch position
	0110	SEND	Basic data 3: content not specified
	0111	SEND	Basic data 4: content not specified
	10PD	SEND	Process data
	1111	SEND	Calendar: data and time
010	RCVR	SEND	Targeted message
011	RCVR	SERV	Service message: Service >>User
100	SERV	SEND	Service message: User >>Service
101	XXXX	USER	Partner system message
110	XXXX	XXXX	Free
111	XXXX	XXXX	Free

SEND: 4 bit address of the sender of a message
RCVR: 4 bit address of the receiver of a message
SERV: 4 bit address of the provided service
USER: 4 bit address of the job computer in the partner system
PD: 2 bit to indicate process data types: 00-set point; 01-actual value; 10-set point request; 11-actual value request

DIN9684 is based on CAN 2.0A, which implies 11-bit message identifier, and runs at 125 bits/second. The CAN messages were classified into 8 priority groups according to first 3 bit of the identifier (the smaller the number, the higher the priority). The remaining bits of the identifier are further divided into subgroups to represent different information. (Table 1) The group with priority '000' is system function message for network management purpose. The group with priority '001' defines the message types for basic tractor information, such as driving velocity, PTO revolution, engine speed,

etc. The message with priority ‘010’ indicates targeted message, which put receiver’s address in the identifier for hardware filtering messages. The messages with priority ‘011’ and ‘100’ are used for system service purpose. The message with priority ‘101’ is partner message, which can be seen as a secondary network that uses the infrastructure of the main network. Such a system can be a proprietary solution of a manufacturer for internal control of an implement. The message with priority ‘110’ and ‘111’ are not defined, and are free to use.

The address width of DIN9684 is only 4 bits; therefore it can only allow 16 ECUs to be connected to the bus at one time. The address of an ECU is acquired dynamically through an address claim messages after this ECU’s login process.

(2). DIN9684 Network Management
DIN9684 specifies a network management procedure to allow open system interconnection. This network management is implemented through system function message with priority ‘000’. There are two types of system function messages (Table 1): A=1 means a new user login message; (Figure 1) A=0 means system management messages. (Figure 2)

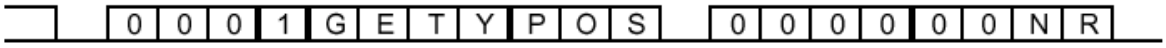


Figure 1: Log in message format

The GETY and POS bits should be determined based on implement type and mounting position by the standard. For example, GETYP/POS for Cereals harvesting front mounting will be 0111/001. The NR bits indicate the number of times login message have been sent. This message should be sent three times when an ECU logs in.

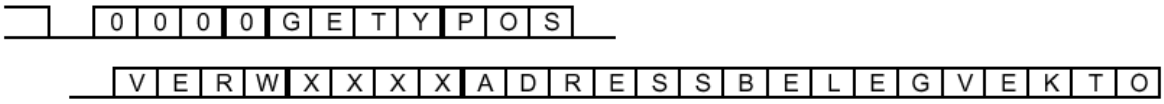


Figure 2: The format of system management messages.
The system management message has the general message format as shown in Figure 2. The bit group VERW designates the specific system management function according to Table 2. The bit group XXXX contains the dynamic user address. The remaining 16-bit group ADRESSBELEGVEKTO signifies the address claim table. A logic 1 at a 16-bit position signifies that the address have already been claimed, while a logic 0 means the address is free.

Table 2: DIN9684 system management functions

VERW	System management function description
0000	Address claim message
0001	Alive handling message
0010	Address release message
0011	“Implement is disturbed” message
0100	Alive handling message of service
0101	Not allocated
0110	Not allocated
0111	“Service is disturbed” message
1000	Request to send implement descriptor
1001	“Send implement descriptor” message
1010	System stop message
1011	Implement stop message
1100	Implement status
1101	Not allocated
1110	Not allocated
1111	On/off message

In the following section, we use an example to illustrate the network management procedure for DIN9684, and later on we will show how this network management is implemented using CAN Kingdom. In this example, a virtual terminal (VT) first is connected to CAN bus, then a weed sensor (WS) will log in. (Figure 3). The following steps describe the interaction between the weed sensor and the virtual terminal when the weed sensor logs in. The step numbers are used to help description, however, they do not necessarily represent timing sequence of the event. For example, step 3 may occur before step 5 or after step 5 depending on the response time of the VT and the weed sensor.

1.Weed sensor sends log in message three times with an interval of 1 second with NR bit of 1,2, and 3 respectively. (Figure 1)

2.Weed sensor listens to the bus until receiving another node's alive-handling message. From the second and third byte of the alive-handling message, which contains address claim field, the weed sensor selects its dynamic address from the free addresses in the field. And it will send out an address claim message, with the XXXX in the identifier filled by the selected address and the address claim field updated (Figure 2).

3.Weed sensor sends out an implement descriptor message.

4.Weed sensor sends out a request implement descriptor message.

5.When the VT receives the address claim message, it will send out a request implement descriptor message.

6.When the weed sensor receives request for implement descriptor from other nodes, it will sends its implement descriptor for every request. This will cause each node on the bus to send multiple implement descriptors for each log on. Typically, each node sends its implement descriptor equal to the number of nodes +1 times. From these messages, the newly logged ECU builds its system monitor table, which contains all the connected ECU's dynamic addresses and implement descriptors. The already logged ECUs (VT) will update their monitor tables with the newly logged ECU (WS) information.

7.The old ECUs (VT) also will respond the request for implement descriptor from the newly logged ECU (WS).

8.At this time, the weed sensor has successfully login to the LBS system. It needs to send an alive-handling message every second to keep other nodes informed.

When a node wants to log out, it sends out an address release message. All other ECUs will learn this information and updates their system monitor tables. During normal system operation, a node may send out other system management messages such as service disturbed, implement disturbed, implement stopped, system stopped, implement status, on/off, etc. if such a condition occurs.

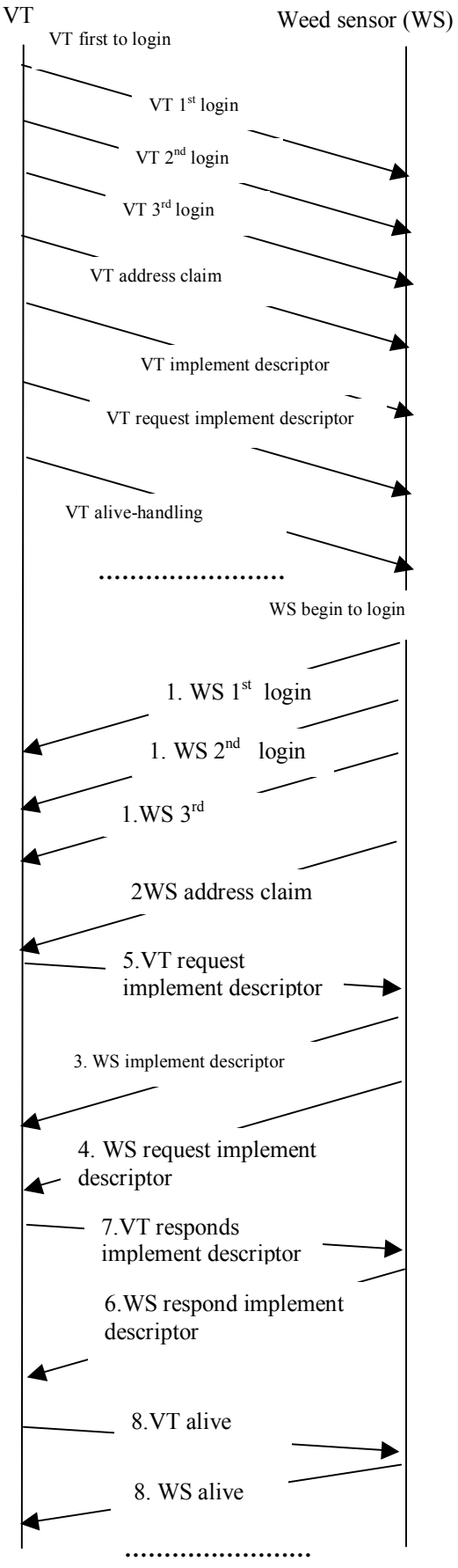


Figure 3: LBS network management procedure

3. CAN Kingdom (CK) Protocol

CAN Kingdom (CK) was initially developed by Kvaser, Sweden, and CAN Kingdom International (CKI) is the group responsible for CK. The corner stone behind the CK protocol is dividing tasks between the system designer, who designs the king node, and the module designer, who designs the city node. The city node running status, such as start, stop, silent etc., is controlled by the king node. Furthermore, all message identifiers are assigned by the king node. Therefore, king administers all the activities within the network, which is called "kingdom". The module designer only takes care about local issues within one module (city), while the system designer takes care about global issues within the kingdom.

For an open system, like the DIN9684 system built in this project, the identifiers have already been defined by the standard, and the node running status is not controlled by a central node. Therefore, the role of the system designer is very limited and a king node does not need to exist physically. During system startup procedure, city nodes can assume that there is a "virtual king", and configure the node status and message identifiers according to the open standard.

CK provides a set of primitives to address the common issues for CAN communication. These primitives are organized into groups and each group can be associated with one CAN hardware buffer. Before transmitting or after receiving a message from one buffer, the primitives can be called. Thus, these primitives not only can be called from inside a node, it can also be triggered from an outer node (such as the king) by receiving a message. In CK, a group of primitives is called a document, which is defined as "a set of forms describing information carried in one Envelope (identifier)". Each document (group) can include a number of pages (primitives) multiplexed by a data byte in CAN message. From the implementation point of view, a document is essentially a function to encode/decode messages with the same identifier. One folder is associated with a unique identifier and a unique document at any given moment. (Figure 3)

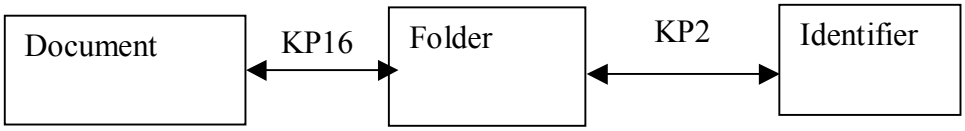


Figure 3: CK Post Office structure

There are four types of documents in CK specification: KingsDocument, MayorsDocument, TimeHeraldDocument and BlockTransferDocument. A KingsDocument includes a set of primitives (pages) to start/stop a city (Kings page 0, KP0), to assign an identifier to a folder (KP2), to place a document into a folder (KP16), to setup a group (KP3), to remove a group (KP4), to setup an action/reaction pair (KP5) etc. These primitives are usually triggered by receiving a message from the king node. Therefore, the document containing these primitives is called KingsDocument. A MayorsDocument provides city identification information upon request. A TimeHeraldDocument provides a global clock for CAN network, and a BlcokTransferDocument defines how to transmit data longer than 8 bytes.

After being familiar with notations of documents and pages, we can study what are the common issues for any CAN network, and how CK primitives address these issues.

(1). Define message format and assign identifier

All CAN communication systems need to define the format and identifiers of messages. CK designs a set of documentation format, including document list, form list, and form to specify message format (Figure 6). For identifier assignment, CK designs KingsDocument Page 2 (KP2) to assign an identifier to a folder dynamically (Figure 3).

(2). Decode a message

All CAN controllers need to parse CAN messages, and decide how to respond these messages. The CK proposed a concept of document (and its relation with a folder) to decode a message. Furthermore, KP16 is designed to place a document process into a folder (Figure 3).

(3). Transmit a message

Basically there are three ways of transmitting a message:

- Transmit a message each time a node asks for (poll) it.

  KP5 was designed to configure one transmit folder with a receiver folder. Whenever the receiver folder receives the polling message, the associated transmit folder will respond with a corresponding message.
- Transmit a message on a periodic basis.

  KP11 and KP12 can work together to provide periodic transmission of messages. KP11 sets up a circular time base, and KP12 sets up a repetition rate for a folder. This folder will be transmitted each time the system clock advances by a set amount of time defined in KP12.
- Transmit a message each time an event occurs.

  For event transmission, such as in the case of an alarm, a message needs to be sent out. In CK, each node has an event array. Events are assigned to folders. If a folder event occurs, the document process associated with this folder is called. This can be done by using KP5.

(4). Real time requirement

Most CAN communication systems are embedded real-time systems. Therefore most of them have some real-time requirements. CK defines a global clock, which will time-stamp each sent-out message or received message. Each message can be checked again the time requirements.

(5). Group nodes

Since CAN is a broadcasting system, every node receives CAN message at the same time. Sometimes, we only want part of CAN nodes to respond to a CAN message. In this case, the group address can be used to transmit such messages. KP3/ KP4 can add/remove a node to/from a group.

CK also has other primitives, such as start/stop node (KP0), assign base number (KP1), new city physical address assignment (KP9), bit timing register setting (KP8), which are primarily used for CK network management. Since the system we build is an open system, these primitives are not needed and can be removed to save memory space.

4. Application of CK to developing a DIN9684 system

In this section, we discuss a weed sensor communication interface development in details and describe how CK was used in the system. This development was based on CK primitives written by US Navy Surface Targets Engineering Branch [Purdy, 2001], which includes KingsDocument, MayorsDocument and TimeHeraldDocument.

The weed sensor system consists of a number of weed sensors and a touch screen terminal, which accepts inputs from operator and displays results on the touch screen. The touch screen terminal is a commercial product (AGCO DataTouch™ Terminal, AGCO, 2000). Data exchange with weed sensor is through CAN bus according to the DIN9684 standard. The weed sensor was developed in a previous project to identify weed from soil and crop (Wang, et al 2001). The weed sensor needs to be calibrated before it can identify weed in fields. The calibration procedure involves training the sensor with different subjects, including weed, soil and crop samples.

(1). Procedural approach to develop application modules

As in Figure 3, the relationships among documents, folders, and identifier are very important. The core data structure in this relationship is the folder, which contains fields for identifier, data, document name, event, an enable/disable flag, a direction flag, identifier extension flag, and a new message flag. The folder data structure is statically mapped to the CAN hardware buffer. A mail process function can be created to check the folder status. If there is a new message coming in (the

CAN receiver interrupt sets the new message flag), or new message to be sent out, (the application program sets the new message flag) the corresponding document process will be initiated. If the folder is configured as transmitting, this process function typically will fill up the CAN data field. If it's configured as receive, the associated document process will typically decode CAN data field, and modify appropriate variables shared with the application code. The mail process function can be called at any time. Typically, it is called when the application is waiting for a message to come, or/and called when a CAN message receive interrupt occurs.

A procedural approach to develop application codes can be designed as following:

a. Define the shared variable among the document process and the application code.
b. Define the document to process the incoming message and to modify the shared variable accordingly in the document process code.
c. Using KP2 to assign an identifier to a folder and KP16 to place a document into a folder.
d. Write an application program. The application program only checks the shared variable and decides program flow. Whenever a message needs to be sent, the application code sets the new message flag in the folder. Whenever a message needs to be received, the application code waits and checks the shared variable, which will be updated by the received message.

Through this procedure, we convert the problem of communication to simply checking shared variables between the document process and application code, which will significantly reduces programming complexity, and make CAN communication simple.

(2). LBS network management module

In this section, we describe how to use the above steps to develop application specific programs and how CK primitive were used in the module design.

In DIN9684 network management, for messages transmitting out from one node, the identifier consists of AGETYPOS as shown in Table 1. For a specific node, the GETY and POS are fixed. Therefore, there are two types of transmitting messages: login and system function, which require two transmit documents: LBSLogInDocument and LBSManagementTransmitDocument. For LBSManagementTransmitDocument, there are 16 different system management functions, each requiring a page for processing. This is implemented by using a switch-case statement in the document process. For receiving messages, there are many possible identifiers. We can use a masked folder to receive all the messages, and use the VERW value to differentiate processes. Although there may be many different identifier messages for this folder, the message functions are all contained in Table 2. Therefore, one receive document should be enough for all these messages.

We can apply the four-step procedure described above to implement LBS system management functions as following:

a. Define the necessary shared variables between document and application code: e.g., the number of login times for login document, etc.
b. Define 3 documents for LBS system management.
   - LBSLogInDocument fills in login message according to DIN9684.
   -LBSSystemManagementTransmitDocument transmits specific network management as defined in Table 2.
   -LBSSystemManagementReceiveDocument receives specific network management message and decode them based on Table 2.
c. Use KP2 and KP16 to connect document to folder and folder to identifier. The transmitting identifier is determined by implement type and position. The dynamic address is acquired through selecting a free address from the address claim field of other node's alive-handling message. The receiving folder identifier is masked to receive all messages from different ECUs.
d. Sequentially execute the network startup procedure described in Figure 3, and set the new message flag if a message needs to be sent.

Besides KP2 and KP16 were used in every document process, there are some other direct applications of CK primitives in LBS network management module design to simplify implementation.

(1). An alive handling message needs to be sent every second. Therefore it is a periodic transmission. We can easily implement this by using KP11 and KP12. KP11 sets the circular time base to start a clock. KP12 sets one CAN folder to LBSSystemManagementTransmitDocument, the page/form number to 0 (alive-handling message), and the repetition rate to one second. Thus, the alive-handling message will be transmitted every one second.

(2). For implementation descriptor request and response, we use Kings Page 5 to setup an action/reaction pair between LBSSystemManagementReceiveDocument page 8 (receive request implement descriptor) and LBSSystemManagementTransmitDocument page 9 (send implement descriptor) .

(3). Weed sensor application document
The weed sensor exchanges message with the touch screen using target message types. Therefore, two types of messages exist, and two documents need to be created: WeedSensorTransmitDocument and WeedSensorReceiveDocument. However, we still need to differentiate the types of weed sensor messages. This is done by WERT/INST field (3rd byte in CAN data field) in LBS target message. (Figure 4) The PD/MOD/ZAEHLNUM/D bits are defined in LBS specification.

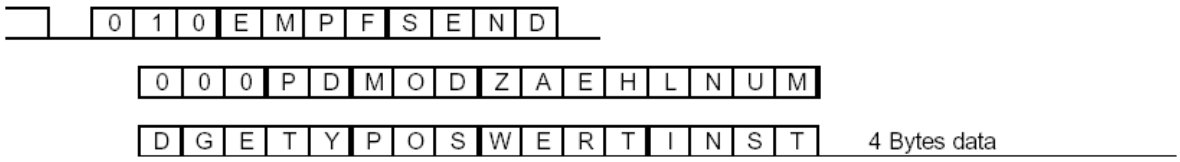


Figure 4: LBS target message format

All the weed sensor message types and their WERT/INST are shown in Table 3. In this table, WERT indicates the row number and the INST indicate the column number; together they define an entry for a message type. For example, WERT/INST =0000/0000 is defined as weed sensor setup message.

Table 3: WERT/INST table for weed sensor.

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	Setup	Start train	Pause	Resume	Stop	Sample Count	Detection result	Began Test					Basic sw version	Menu Sw version	Menu size	
0001																
0010																
0011	Speed															
0100																
0101																
0110																
0111																
1000	Worked Area															
1001	Operating Time											Effective time				
1010																
1011																
1100																
1101																
1110																Error alarm
1111	Impl type	Mfg. ID		Software Version												

Following the four steps in programming CAN interface, the WeedSensorTransmitDocument and WeedSensorReceiveDocument can be easily designed. Each entry in Table 3 should have one case (page) in the documents. For some entries, such as software version, and menu code size, are partner messages to communicate with AGCO virtual terminal. Therefore, we create WeedSensorPartnerTransmitDocument, and WeedSensorPartnerReceiveDocument to deal with them. The programs basically share the same structure and only differ in identifier assignment. The application program and the document process exchange information through shared variable.

(3). Other document

For our application, we need to upload weed sensor menu program from the weed sensor to the virtual terminal. The terminal uses ISO11783 transport protocol, (ISO11783, part 2) which is not compatible with the CK transport document. Therefore, we built a transport document to transmit menu code. Again this is easily implemented by using the above-mentioned procedure.

(4) Consistent documentation

Another advantage of CK is that it specified a consistent and systematic method for documentation: document, documents list, form, form list, etc. Following these conventions, the program is very easy to describe. The weed sensor system documentation based on CK is shown in Table 4, 5 and Figure 5.

Table 4: Transmit document list

Transmit_Document_List0			
Document	RTR	Description	Fixed Folder number
0	Yes	Mayors_Document	1
1	Yes	TimeHeraldSendDocument	No
2	No	LBSSystemLoginTransmitDocument	No
3	No	LBSSystemManagementTransmitDocument	No
4	No	WeedSensorTransmitDocument	No
5	No	WeedSensorPartnerTransmitDocument	No
6	No	TransportProtocolTransmitDocument	No

Table 5: Receive document list

Receive_Document_List0			
Document	RTR	Description	Fixed Folder number
0	No	Kings_Document	0
1	No	TimeHeraldReceiveDocument	No
2	No	WeedSensorReceiveDocument	No
3	No	WeedSensorPartnerReceiveDocument	No
4	No	TransportProtocolReceiveDocument	No

For each page, CK defines a format to design the message content. One example is shown in Figure 5. (LBSSystemManagementTransmitDocument page 9, send implement descriptor).

Figure 5: CK form format (to decode a page)

Document name:	LBS System Management Transmit
Document List:	0
Document Number:	3
Document Type:	Transmit

Page Description.

Page Number:	9
Number of Lines:	8
Page Description:	Send implement descriptor LBS System management page

Line description.

Line 0:	1001xxxx	Reserved by LBS xxxx: LBS dynamic address
Line 1:	00000000	Reserved by LBS
Line 2:	00000000	unused.
Line 3:	00000000	unused
Line 4:	xxxxxxxx	Implement descriptor, MSB
Line 5:	xxxxxxxx	Implement descriptor
Line 6:	xxxxxxxx	Implement descriptor
Line 7:	xxxxxxxx	Implement descriptor, LSB

5. Performance Evaluation

For a real-time embedded system, the primary criteria to evaluate the system are timing requirement and memory requirement (including RAM, ROM, etc.)

(1). Timing analysis

All the messages in the system use 11-bit identifier (CAN 2.0A) and most data load are 8-byte long. A standard CAN message with 8 bytes of data have a max length of 134 bits including intermission space and bit stuffing. At a baud rate of 125K/second, this allows the maximum transmission of about 1000 CAN messages in one second. In our system, implementation of folder scheduler (proc_mail) took roughly 20 microseconds, which allows receiving up to 10,000 packets of information in one second. (Purdy, 2001) Therefore, if the CPU is not overloaded by the application program, the system is fast enough to handle all the CAN messages.

The system we designed consists of a Virtual Terminal, a spray controller, a radar speed sensor, a GPS receiver, and two weed sensors. The periodic messages and their characteristics are listed in Table 6. During system startup, there are also some aperiodic messages, such as login, address claim, implement menu information exchange, and calibration command, etc. Because frequencies of these messages are very low, they do not contribute too much to busload. Thus, we only need to analyze the busload during normal operations. For the same reason, we do not consider the bus disturbing and error-reporting message.

Table 6 shows there are 30 messages occurring in the CAN bus during one second period, which accounts for about 30/1000 = 3% of CAN bus load. Although a complete response time analysis is required to analyze all the message against their deadlines (which is beyond the scope of this paper), obviously the CAN bus is far from full load, and many more implement controllers can be connected to the CAN bus without causing loading problem.

Table 6. Overview of the periodic message characteristics

Name of the sending participant and names of the generated messages	Name of the receiving participant	Frequency (Hz)
Virtual terminal		
Alive handling	All	1
Data and time	All	1
Service alive	All	1
Weed Sensor 1		
Alive handling	All	1
Detection result	Spray controller and VT	10
Weed Sensor 2		
Alive handling	All	1
Detection result	Spray controller and VT	10
Spray Controller		
Alive handling	All	1
GPS receiver		
Alive handling	All	1
Position	All	1
Speed	All	1
Data and Time	All	1
Total		30

(2). Memory requirement

From compiled codes, it's easy to obtain the ROM requirement for the weed sensor implementation. Because code size depends on many factors including compiler setting, user programming style, etc. Comparison of code size should be made using the similar criteria. Table 7 and Figure 6 shows the code size for the weed sensor program by using CAN Kingdom. In previous research (Wei, et al, 2001), we implemented a weed sensor program by using a proprietary protocol. Table 8 shows the code size for the weed sensor implementation without using CAN Kingdom. From Table 7 and Figure 6, the kernel of CAN Kingdom (including KingsDocument, Mayors Document, TimeDocument, CAN driver, etc.) only amounts to about 13.1K bytes (compiler without optimization) and 8.6K bytes (compiler with optimization for size) memory requirement. The application part of codes uses about 11.3K(compiler without optimization) and 9.2K(compiler optimization for size). From Table 8, the code size for the weed sensor implementation without using CAN Kingdom is 12.2K (compiler no optimization) and 9.8K(compiler optimization for size). The use of CAN Kingdom did increase the memory requirement slightly. This is the cost we had to pay to receive benefits in implementation and documentation.

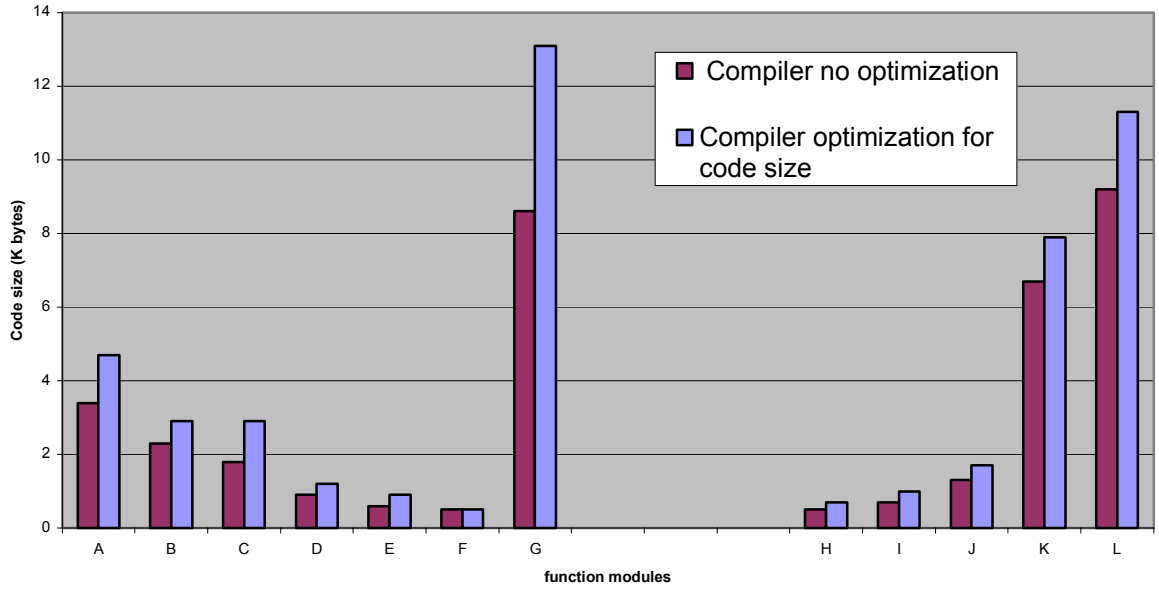
Because CAN Kingdom is only a set of primitive, it does not require more RAM, thus integrating CAN Kingdom to application would not significantly increase the RAM requirement.

Table 7: Code size for the weed sensor by using CAN Kingdom

Function modules	File name	Label	Code size (byte)	
			No optimization	Optimization for size
KingsDocument and Mayors Document	Kingspage.c	A	4.7K	3.4K
Timing related document	Timing.c	B	2.9K	2.3K
CAN driver	CANlib.c	C	2.9K	1.8K
CK basic services, including document, form, parameter generation, search, etc.	Citydoc.c	D	1.2K	0.9K
Mail Process	Postoff.c	E	0.9K	0.6K
CK initialization	Mayor.c	F	0.5K	0.5K
Sub total		G	13.1K	8.6K
LBS management document	Lbs.c	H	0.7K	0.5K
Transport document	Transport.c	I	1.0K	0.7K
Weed sensor document	Weed.c	J	1.7K	1.3K
Weed sensor application (algorithm, menu processing, etc.)	Da.c	K	7.9K	6.7K
Sub total		L	11.3K	9.2K

Table 8: Code size for the weed sensor without using CAN Kingdom

Implementation part	Code size (byte)	
	No Optimization	Optimization for size
Weed sensor application	11.1K	8.78K
CAN library (Infineon CAN driver library)	1.1K	1.1K



Note: Function modules are listed in Table 7.

Figure 6: Code size for the weed sensor implementation by using CAN Kingdom

5.Discussions

A CK project is organized like a tree, with CK project being the root, documents being its branches. Each branch can have many leafs (pages). From this tree, not only can we see the complete picture of the system, we also can easily trace the details of a specific function (a certain page). In addition, this tree structure makes CK system documentation very easy to understand. Figure 7 is a general structure of the CK system.

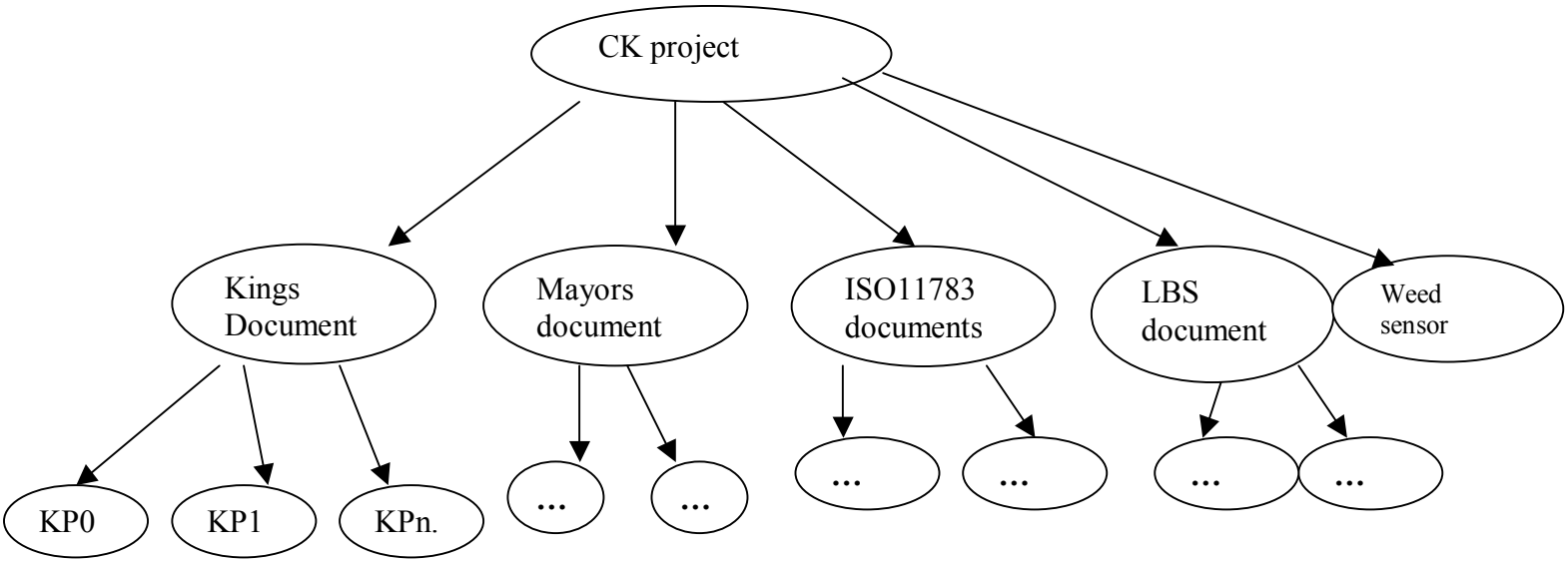


Figure 7: CK program structure

While the KingsDocument is used in every CK city, ISO11783 documents or LBS documents can only be used in agricultural machinery electronics for communication. Thus, CK provides a platform for organizing programs and code sharing.

In CK specifications, the default identifier for the KingsDocument is 0. In LBS system, however, the 0 identifier was used for other purposes. Therefore, KingsDocument identifier has been shifted to other numbers to avoid unnecessary confusion. If a king node is not physically present in the system, there is no need to connect these primitive documents to the folder. In this case, KingsDocument and MayorsDocument will no longer be a document, but a purely "function call". However, we leave these documents associated with folder 0, and 1, respectively. This arrangement would allow the city node to be dynamically configured for use with other open or proprietary systems, although the node is default to be used in LBS system.

In CK specifications, there is also a start-up procedure, which is not needed for an open system. However, we can follow this procedure for compatibility and then start the LBS network management procedure. Again, this allows the node to be used in both LBS and other systems.

In CK specifications, KP12 configures a folder for periodic transmission of a message. However, it doesn't specify the page number in this document to be transmitted. In LBSSystemManagementTransmitDocument, however, the alive-handling message is only a case/page of this document. Therefore, we need to somehow denote the page number in KP12 message. This can be done by modifying KP12 second byte in the message. The original design for KP12 in line 2 is the folder number. Since most CAN controllers only have less than or equal to 16 buffers, we can use the lower 4 bits of the second byte as folder number and the upper 4 bits as page number. This approach works well for LBSSystemManagementTransmitDocument, since it has only 16 maximum entries in Table 2.

Although CK has provided many advantages, integrating CAN Kingdom kernel to application codes did increase the code size by about 10K bytes. Furthermore, studying and applying CK to system design is a time-consuming task. In the long run, however, this overhead will become negligible.

6.Conclusions

CAN kingdom provides a systematic method to design CAN-based systems. In this paper, we present a weed sensor system, which demonstrated many benefits in designing a DIN9684 network using CK primitives and conventions, although CK is not specially designed for open systems.

(1). CK provides a set of primitive for CAN communication, which can significantly reduce the complexity in implementation.

(2). CK provides a platform, upon which other domain-specific primitives, such as the DIN9684 library and the ISO11783 library, can be built. For example, the LBSSystemManagementDocument developed in this project can be used in other ECU nodes that connect to a DIN9684 bus. These libraries will avoid repetitive development and improve software module reuse. Furthermore, CK primitives are scalable and modular, which can adapt to different application environments. This is an important feature for embedded system where both processing power and memory are very limited.

(3). CK provides a structural, four-step approach for module design:

a. Define the shared variable between communication document and application program
b. Define the document to process the incoming message and modify the shared variable accordingly.
c. Use KP2 and KP16 to connect the defined document with an identifier and a folder number
d. Write the application program.

If these steps are followed, programming CAN interface would become a less difficult task.

Acknowledgment

The authors highly appreciate the assistance from Dave Murray of AGCO, David Purdy of US Navy Seaborne Target Project, Joel Morton of Dickey-John, and Lars-Berno Fredriksson of Kvaser.

7.References

Auernhammer H., S. Dielktronische. 1993. In: Landwirtschaftliches BUS-System –LBS, Proceedings of the Congress, 30 November 1993, Frankfurt/Main (Auerhnhammer H;Frisch J eds) pp18-30, KTBL Arbeitspapier 196.

Bosch. 1991. CAN specification 2.0B.

Fredriksson, L. 1996. A CAN Kingdom, Rev 3.01, Kvaser AB.

Leen G, D. Heffernan , A. Dunne. 1999. "Digital networks in the automotive vehicle"; IEE Computing & Control Engineering Journal, December 1999, pp257 – 266

ISO (International Organization of Standard). 1993. Road vehicles -- Interchange of digital information -- Controller area network (CAN) for high-speed communication

ISO (International Organization of Standard). 1994. Road vehicles -- Low-speed serial data communication -- Part 2: Low-speed controller area network (CAN)

ISO (Internal Organization of Standard). 2001. ISO11783: Tractors and machinery for agriculture and forestry –serial control and communications data network

BUS-Schnittstellie. 1998. LBS, DIN9684, The mobile Agricultural BUS.

Purdy, D. 2000. CK programming lab.

Purdy, D. 2001. Personal Communication.

SAE. 1996. J1939: recommended practice for serial control and communications vehicle network.

NMEA 2000, Standard for serial-data networking of marine electronic devices.

Wei, J., N. Zhang, N. Wang, D. Oard, Q. Stoll, D. Lenhert, M. Neilsen,M. Mizuno,G. Singh. 2001. Design of an Embedded Weed-Control System Using Controller Area Network (CAN). ASAE Paper No. 013033