The XM Satellite Radio Software Module of an Embedded Car Audio System

Di Wu
Dalian University of Technology, Dalian, China
Email: wudi23893@sina.com

Chenxi Hou
Dalian University of Technology, Dalian, China
Email: xnhcx@163.com

Limin Sun
Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
Email: sunlimin@iie.ac.cn

Yan Ling
Dalian University of Technology, Dalian, China
Email: lingyan321.love@163.com

Jiangchuan Liu
Simon Fraser University, Vancouver, Canada
Email: jcliu@sfu.ca

***Abstract*—XM radio service is an emerging satellite digital audio radio service suitable for automotive environment, it offers users high-quality radio programs with better signal coverage by adopting communication satellite broadcasting technology. In order to incorporate XM radio function into a digital car audio system based on the embedded system industry standard ITRON, we analyze both software and hardware requirements of the XM function module in this work. Architectural design that separates the XM radio function module into five layers is given. The operating system selected for the microcontroller used in the car audio system is introduced. Implementation details of the XM function module, such as control commands and device communication protocol of the XM tuner, the state transition matrix layer and the application layer are also presented in this work.**

***Index Terms*—XM radio, digital car audio system, embedded system, ITRON, communication protocol, state transition matrix.**

I. INTRODUCTION

With the rapid development of the automobile manufacturing industry and rapid growth of the travel demand of consumers, the global automobile sales is magnifying year by year. The strong demand in the automobile consumer market has prompted the maturation and development of a series of other related industries, the digital car audio manufacturing industry is a typical example of these industries. As the extension of automobile manufacturing industry into electronic product design and manufacturing industry, digital car audio system [1] products are gradually getting into the vision of more and more automobile users. The digital car audio system products meet the visual and auditory entertainment demand of consumers for during a car ride, they offer brand new entertainment experience options for car passengers and drivers on the trip, and have transformed itself into an indispensable composing part of on-board digital systems. In recent years, car manufacturers are constantly launching car models by advancing and adopting more effective design and manufacturing techniques in order to meet the increasingly picky taste of hypercritical automobile consumers. The quality of digital car audio system configuration has a considerable decisive impact on the purchasing selection of automobile consumers. The improvement of function and quality of digital car audio systems is becoming ever more important for both automobile manufacturers and automobile consumers. This paper focuses on the design and implementation process of a function module named the XM radio module in a real development project of a digital car audio system. The digital car audio is a sub-system of an on board electronic control system, which operates within a software environment provided by the embedded real-time operating system compliant with industry standard ITRON [2,3]. General development techniques and the application status quo of car audio systems are displayed in this work.

II. XM BROADCAST RADIO SERVICE

doi:10.4304/jsw.7.9.1981-1992

XM is a SDARS (Satellite Digital Audio Radio Service) [4,5] broadcast radio service operated by the US company SiriusXM in North American regions including the USA and Canada of. SDARS is a branch of digital audio broadcasting [6] services which incorporates multiple communication satellites for radio program broadcasting.

XM radio program signal is delivered by five of the nine in-orbit satellites of the SiriusXM Company. The radio program signal is transmitted to and then broadcast from these satellites to the ground listening devices. The working band of XM radio service rests on the 2.3GHz S band, with frequency ranging from 2320MHz to 2345MHz.

XM service was launched on September 25, 2001 in the USA, providing high quality broadcast radio service that surpasses existing radio services like FM and AM both in sound quality and geographic coverage. In order to maintain signal strength, in areas where signals may weaken greatly, like places in the vicinity of high rise buildings in downtowns and under-ground car parks where satellite signal might be blocked, ground receivers are installed to rebroadcast signal to guarantee constant and full coverage for subscriber usage.

Since the XM radio service adopts a business model base on paid subscription, various kinds of programs with rich content and premium sound quality are provided to the users,. At present, there are over 180 stations whose types span of music, news & talk, sports, and traffic & whether available in the XM radio service, some automobile owner can listen to the XM radio service with the eight-digit identity number called Radio ID assigned to each on board XM tuner device for three month trial period usually provided by the automobile manufacturer. By the end of the year 2010, the number of subscribers to the XM radio service operated the SiriusXM company has exceeded 20 million to 20.19 million, showing strong market demand for satellite broadcast radio service, the sales of digital car audio systems with preinstalled XM radio listening devices and corresponding software solution will benefit from this process of market demand growth dramatically.

III. Requirement Analysis of the Implementation of XM Broadcast Radio Function

In general, the objective of the development project is to establish a digital car audio system that consists of microcontrollers, display screen, operation panel of keys, multiple external devices and speakers. These components provide users with a series of complete car audio system functionalities that include signal processing, system state display, user input acquisition, playback of media content in external connected devices and sound output to the users. XM broadcast radio is one of the functionalities that needs to implement in the car audio development project. The integration of XM radio may be discussed from two perspectives. On one hand, the XM radio module relies on the common infrastructure shared among other digital car audio system function modules like FM/AM radio, Bluetooth audio and so on,

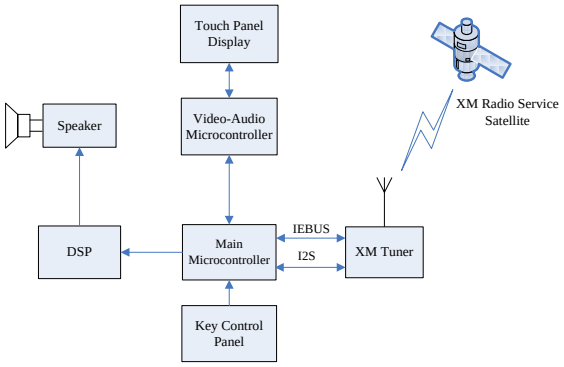

Figure 1. Block diagram of XM car audio system hardware related to broadcast radio functionality.

on the other hand, it has several special requirements for both software and hardware configuration of the digital car audio system. The detailed hardware configuration can be seen in Figure 1.

A. Hardware Requirements

The hardware requirements of XM broadcast radio functionality can be separated into two classes. The first class includes hardware shared among many different system functions, like microcontrollers on which the operating systems run, chips that can process and adjust audio signals, speakers that play processed audio signals input from the main microcontroller, display screen that displays system states, key control panels, and touch panel display. The second class includes special hardware specifically required by the XM broadcast radio functionality, which includes external devices that can receive broadcast radio signal from the satellites and the data communication bus that establishes data exchange on the connection between the microcontroller and the external device. In the development project of the digital car audio system, an XM satellite broadcast radio signal receiver called XM tuner is adopted to receive radio program signal from the satellites, and a data communication bus named IEBUS is used to connect the XM tuner to the microcontroller in the head unit of the digital car audio system.

During the design phase of the car audio system product, taking into account the conservation of manufacturing cost and differences of device control and data processing demands for the whole car audio system in the actual environment, two microcontrollers with different processing capabilities are adopted.

For the special demands for user interface drawing on the LCD screen and decoding processing of video media content, a video audio microcontroller customized for graphics performance optimization is adopted.

The other microcontroller, namely the main microcontroller is responsible for power management, audio signal output to DSP chips for sound output, detection of key press on the control panel, and data exchange with several external devices via different kinds of data communication buses.

The two microcontrollers are connected by 32-bit data communication bus named CSI (Configurable System

Interconnect). Via the CSI bus, the two microcontrollers can exchange data with each other according tocertain inter-microcontroller communication mechanism.

The implementation of XM function is divided onto two microcontrollers. The video audio microcontroller controls human-machine interaction functions like the drawing of XM user interface on the touch panel display and the detection and resolution of user touch input on the touch panel when the XM radio user interfaces are active. The main controller sends control commands to the XM tuner device whenever a user key press input takes place, and takes feedback execution result of control command, and passes the display data from the XM tuner device to the video audio microcontroller side to refresh the user interface accordingly.

This work is mainly about the implementation of the XM tuner device driving functions. The process of controlling the XM tuner device is done entirely by the main microcontroller of the car audio system.

In the digital car audio system development project, a microcontroller of V850ES/SJ3 series produced by NEC Company is adopted as the main controller, its highest working frequency can reach up to 32MHz. The total number of general registers is 32, the data length of each register is 32bits. The data size of ROM is between 384KB and 1024KB, the data size of RAM is between 32KB and 60KB. V850ES/SJ3 microcontrollers has a total of 128 I/O ports, its serial interfaces enable data communication via serial buses like UART, CSI and I2C, besides, IEBUS and CAN bus controller are also present for data communication with external devices. The A/D, D/A converters of V850ES/SJ3 enable conversion between analogue and digital signals. The timing function is provided by five timers, including three 16-bit timers, a watch timer and a watchdog timer. To conserve power consumption, power-saving modes named as HALT/IDLE/STOP/subclock/sub-IDLE are provided to the users of V850ES/SJ3 microcontrollers.

B. Software Requirements

Software requirements of XM function can also be separated into two classes. The first class includes software that controls fundamental hardware, like software that controls DSP chips for audio signal processing, which can adjust sound output effects like volume, treble, bass and so on. The first class of software includes not only graphics-related software that draws user interface and manages user interface transition, but also key press processing software that detects long press/short press, soft key press on the touch panel, dragging key operation and resolves pressed key type from coordinates. Software in the second class includes a device driver which manages and controls the working state of the XM tuner, sends control commands to and acquires responses and device and radio program information from the XM tuner, an IEBUS I/O driving software which enables data exchange between the main microcontroller and the XM tuner device by bus reading and writing, and a communication protocol that encapsulates operations of IEBUS I/O driving software.

IV. ARCHITECTURE OF DEVELOPMENT FOR XM BROADCAST RADIO FUNCTION

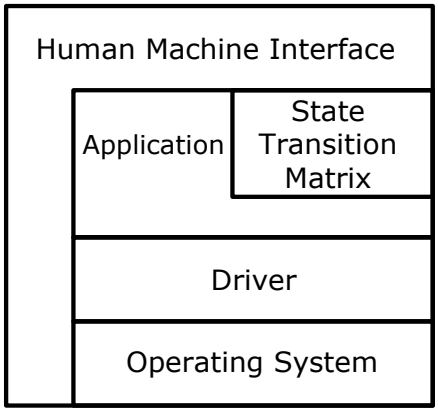

Figure 2. Layered development architecture of car audio system based on ITRON standard.

The digital car audio system development project adopts a layered architecture based on the ITRON embedded operating system standard. This architecture includes five layers, and they are human machine interface, state transition matrix, application, driver and operating system layers from the top to the bottom, as shown in Figure 2.

A. Human Machine Interface Layer

The human machine interface layer is at the top of the whole architecture, it is responsible for the interaction between the digital car audio system and the users. Its responsibility includes acquiring in input from the user, notifying the user of the current working state of the car audio system and notifying the user of the processing result of operation request of the user. The human machine interface layer takes input from the user, resolves the exact operation type, generates and passes operation request event message to the state transition matrix layer. The working state notification is mainly completed by the main display screen of the car audio system, screen text information like song name and list of available channels is displayed to the user. The notification of the processing result of any operation request is mainly conducted by refreshing the user interface, in some cases by sounding the beeper, for instance, when continuous multiple key press operations are conducted by the user in a short period of time, the key press processing capacity of the system may fail to deal with all these requests immediately, a beeping alarm can be sounded to inform the user to prevent conducting more invalid key press operations.

B. State Transition Matrix Layer

Inside the car audio system, there are many time consuming operations, each task may also receive messages containing service requests while an on-going processing is present. Due to the real-time requirement of the car audio system, setting up request message queue to postpone the processing of an incoming request message during busy time is not allowed. To guarantee the real-

time character of the system, for each service request message that has arrived at the task, the system must decide whether an immediate processing is possible. For those requests that are unable to be processed immediately, a request failure notification should be

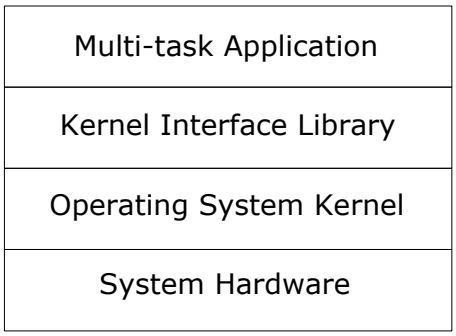

Figure 3. Layers of ITRON standard compliant embedded real-time operating system.

replied to the sender task, otherwise, the processing is performed and working state is transitioned accordingly. The differences of the event carried by the arriving message and working state of the system can both incur differences of system processing responses to the requests, thus, tasks can use the state transition matrix to decide the proper response to the arriving messages during different working states. When a request message arrives from the human machine interface layer, the state transition matrix needs to recognize the current working state of the task, and look up in the matrix cell for the processing action that should be taken and the state to transition to after the processing action in a matrix with row being the event of the message and column being the current working state, afterwards, the corresponding processing action is conducted and state is transitioned. The state transition matrix maintains the working state of tasks to conduct proper processing action on the input message and maintains the proper transition target of working state for the processing of future incoming messages.

C. Application Layer

The design purpose of application layer is to establish an isolation layer between the practical functions and bottom hardware functions, hiding the impact of changes of hardware beneath the human machine interface layer and state transition matrix layer and providing a collection of relatively stable application interfaces for upper layer design. Only single functions are implemented in the application layer interface, the actual implementation of a single application interface may contain call requests to multiple functions of the driver layer underneath according to certain timing and logic requirements. The design of using application layer hides implementation details of lower layer to the upper layers and improves the stability and portability of the system.

D. Driver Layer

The driver layer lies in the place nearest to the bottom hardware in the architecture, it is mainly used to conceal the details of the operation of digital car audio devices. The process of driving device includes the use of certain hardware control protocol and I/O communication protocol software and needs to comply with certain timing requirements and data format conventions. The driver layer encapsulates device operation processes into message interfaces for upper layers to implement the control function of devices.

E. Operating System Layer

The operating system layer is the foundation of the whole car audio system architecture, it is responsible for the management of fundamental hardware and software resources, accepting resource requests from tasks, allocating and recycling shared resources and ensuring correct and orderly use of resources. The detailed functions of the operating system layer includes the management of task life cycle, the scheduling of task execution, synchronization and communication of tasks, dynamic management of memory, interrupt management, timer management and so on. All layers above the operating system layer can issue system calls to request service of the operating system and implement designed functions under the support of the operating system.

V. ITRON STANDARD COMPLIANT EMBEDDED REAL-TIME OPERATING SYSTEM

The two microcontrollers of the digital car audio system, namely the video audio microcontroller and the main microcontroller are responsible for certain different types of system functions like computing, communication, key press detection and device control respectively, they must perform these functions in orderly, effective and real-time way cooperatively. The embedded real-time operating system is a type of operating system widely used in embedded real-time environment. As a type of embedded system, it has common characters of embedded software like reducibility, low occupation of resources and low power consumption; while as a type of real-time system, in addition to satisfying function requirements of application, it must further satisfy the real-time requirements of application, meaning that several important operation parameters measuring the real-time level of a system, like system response time, task switching time, interrupt delay time must be controlled within the range permitted by the users.

ITRON (Industrial the Real-Time Operation System Nucleus) is a real-time multi-task operating system standard. ITRON has a standard real-time kernel and is applicable for any small scale embedded system. ITRON has become an important industrial standard in embedded device industry and has been widely adopted by digital car audio system development projects. The main microcontroller in the car audio system development project adopts Rx850 operating system, which complies with ITRON v4.0 standard.

The architecture of ITRON can be divided into four layers, as shown in Figure 3. The lowest layer is the system hardware layer, including ROM, RAM, microprocessor, timer, I/O controllers and so on. Above the hardware layer is the operating system kernel, which is written completely in assembly code. The kernel is designed to provide fundamental system functions like task management, synchronization and communication, interrupt management, memory management, timer management, system scheduling and so on. The kernel interface library provides system service in the form of external functions, system calls issued in the form of external functions can thus invoke service programs within the operating system kernel. The multi-task application layer represents tasks responsible for the implementation of application level functions, which is dependent on project development. The most important part that needs to be implemented in the project development phase is the design of the main functions of tasks. The main function of a task acquires notifications or service requests from other tasks and sends notifications or service requests to other tasks to implement the system function which it is responsible for.

- Task Priority and Task Scheduling of ITRON.

Tasks of ITRON acquire opportunity of execution under the scheduling of the operating system. In the startup configuration file, different priorities of execution are set for different system functions according to their real-time level requirements. When multiple tasks are in ready state in the task scheduling queue, the kernel would execute the first task with the highest priority, for tasks with equal priorities, execution would start from the task that arrives first in the scheduling queue.

- Task Structure and Message Communication Mechanism of ITRON.

The execution of a task begins from the main function of the task, the structure of a main function is an infinite loop. At the beginning of the infinite loop, mail message is received from the associate mailbox of the task. Afterwards, the task can judge from the event code and source task data fields of the mail about what notification or service request is sent and which task the sending task is, and then acquires data from the option field of the mail message required by the execution of the service request or the processing of the notification. Thus, task may enter into a specific execution or processing period after the reception of the mail message. After the reception and processing of a mail message, the task execution would go back to the beginning of the infinite loop and wait for the next incoming mail message.

Within the ITRON standard, inter-task message communication mechanism is established primarily by the mail sending function *snd_msg*, the mail receiving function *rcv_msg*, the mailbox system object and mail data structure. The mailbox system object is set up in the system startup configuration file and will be initiated along with the initialization of the system. Every task that needs to communicate with other tasks in the system is associated with a mailbox system object. Tasks may receive mail message from associated mailbox by the mail receiving function *rcv_msg*. When tasks need to send mail message to other tasks, they can fill parameters into corresponding fields of mail variables, including the target mailbox, the identifier of the current task, the event code which identifies the type of the mail message and the mail option field. After parameters are properly set, a function call to the mail sending function *snd_msg* can complete the mail sending process.

Intensive use of inter task message communication mechanism can be found in the car audio system development project, tasks cooperate with each other frequently so as to implement system functions.

VI. Layered Implementation of the Driving Function of XM Tuner Device

A. Control Commands of XM Tuner Device

The XM tuner device is the special device for receiving XM radio signals, its antenna can receive program data from the XM service satellites, and the in-built memory can save user setting data. The IEBUS interface can be used for receiving control commands from the microcontroller inside the head unit of the digital car audio system, and command response and display data can be sent back to the microcontroller.

Upon receiving control commands from the microcontroller, the XM tuner device can change parameters of XM signal reception and the working state of the XM tuner device.

The control commands used in the development project can be classified into the following classes: preset station commands, tune mode commands, station scan commands, station adjustment commands, information acquisition commands and electronic diagnosis commands.

1. Preset Station Commands

Preset station commands are the control commands associated with preset station list display functions. The preset station list is a list of stations saved inside the non-volatile memory of the XM tuner device, the items in the station list are added by users during the listening period. The capacity of the preset is limited in the development project, the XM tuner device can accommodate a certain number of stations within the internal memory. The user may either choose an item added before in the preset station list to start playing or save the present active playing station into an item in the preset station list specified by the user. When the display screen switches into the user interface where the preset station list is present due to some user operation, the digital car audio system must also send request to the XM tuner for the complete content of the preset station list. The four preset station commands can be seen below.

- Preset Station Call
- Preset Station Write
- Preset Station List
- Preset Station Immediate Write

2. Tune Mode Commands

The tune mode is a state flag that decides whether the command execution to be the range of all channels or the

range of only channels within the current category for some user commands.

The XM tuner device doesn't maintain such a state flag by itself, however, all commands related to the tune mode are subject to the tune mode. The microcontroller has to keep a record of the tune mode in order to map operation requests to the commands that should really be sent to the XM tuner device. The tune mode has two values, corresponding to the two different modes: channel mode and category mode. The execution range will be set up as either for all currently available channels or for channels only within the current category at the time of the operation request. There are two tune mode commands: the Tune Mode Switching command can switch tune mode between the two available modes; the Tune Mode Information command can query the microcontroller for the current tune mode.

- Tune Mode Switching
- Tune Mode Information

3. Station Scan Commands

The station scan commands are the control commands to perform station scan actions on the available channels. The station scan action is a process in which each available channel within the execution range specified by the current tune mode is played for 10 seconds sequentially for one round in the direction in which station number increases.

The tune mode makes difference to the execution of the station scan request. When the tune mode is channel mode, the stations to be scanned are all available channels at present, otherwise, when the tune mode is category mode, the execution of the station scan action would be confined to all available stations within the present category. The two station scan commands are as follows.

- Channel Mode Scan
- Category Mode Scan

4. Channel Adjustment Commands

The channel adjustment commands are the control commands that are related to the adjustment of the current listening station. Channel Mode Channel Up and Channel Mode Channel Down commands are sent to the XM tuner device when the tune mode is channel mode, upon receiving these commands, the XM tuner device would switch the current listening station to the previous or the next available station to receive radio program signals. Category Mode Channel Up and Category Mode Channel Down commands are sent to the XM tuner device when the tune mode is category mode, upon receiving these commands, the XM tuner device would switch the current listening station to the previous or the next station within the current category to receive radio program signals. Channel Rough Up and Channel Rough Down commands can be sent to the XM tuner device only when the tune mode is channel mode. Every time these commands are received, the XM tuner device would switch the current listening station to 10th available station higher or lower than the present one to receive radio program signals, these commands offer the user an easier way to switch to the target station in big steps. Category Up and Category Down commands would cause the XM tuner device to switch the current listening station to the last station in the previous or the next category to receive radio program signals. Direct Tune command is triggered by the user operation of directly clicking an item within the channel list for listening, this command carries a station number which the XM tuner device can switch to and start receiving radio program signals. Here is a list of the channel adjustment commands.

- Channel Mode Channel Up/Down
- Category Mode Channel Up/Down
- Channel Rough Up/Down
- Category Up/Down
- Direct Tune

5. Information Acquisition Commands

The information acquisition commands are the commands that are used to acquire desired information from the XM tuner device. Normally, when the transitions in the user interface take place, the microcontrollers need to send these commands to the XM tuner device to acquire the information required by the user interface.

Active Channel Information command is used to acquire the text or number information related to the current listening station, information types include the station number, the channel name, the category number, the category name, the program title, the artist name. Channel List Information and Preset Channel List Information commands are used to acquire the list of all available channels and the list of all items of the preset list within the internal memory from the XM tuner device, respectively. Radio ID Information command is used to acquire the radio identity number information of an XM tuner device, this information is displayed when the current listening station is set to the 0th station. The Radio ID is a unique identity number assigned to each XM tuner device used for subscription to the XM broadcast radio service. The information acquisition commands are listed below.

- Active Channel Info
- Channel List Information
- Preset Channel List Information
- Radio Id Information

6. Electronic Diagnosis Commands

The electronic diagnosis commands are specifically used for the quality assurance process before the digital car audio system products leave the factory. A computer system with quality diagnosis functions would be connected to the target system, usually to an input port of a microcontroller. Internal diagnostic commands would then be sent to modules of the system under diagnosis. The execution results would be sent back for the diagnostic system to judge whether functions of the target product system is working correctly. The diagnosis information has plenty of diagnosis items, indicating whether the product quality meets the requirements and whether the product performances reach the factory standard. All electronic diagnosis commands are listed below.

- Diagnosis Mode Set
- Diagnosis QoS
- Diagnosis Change

B. IEBUS Command Format and Bus Communication Protocol Program for XM Tuner Device

Control commands that are used to control the operation of the XM tuner device are sent to the IEBUS communication bus by the main microcontroller in the digital car audio system head unit. The IEBUS communication bus is a communication method only for the data exchange between the microcontroller and the XM tuner device, several other external devices in the digital car audio system development project are connected onto the IEBUS communication bus to communicate with the microcontroller or other external devices on the IEBUS communication bus. For this reason, the IEBUS communication bus needs to define the format of IEBUS data frames and establish a communication protocol to distinguish different devices on the IEBUS communication bus, mark the total length of a data frame and provide data validation mechanism to detect transmission errors of data frames.

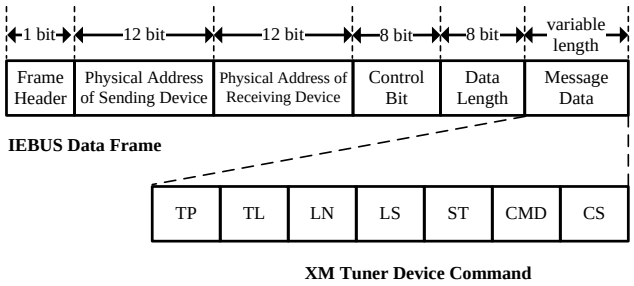

Figure 4. Composition of IEBUS data frame and the control command of the XM tuner device.

As shown in Figure 4, the IEBUS data frame contains the frame header and five data fields, which are the physical address of sending device, the physical address of receiving device, the control bit, the data length and the message data. The physical address fields can identify the sending device and the receiving device connected onto the IEBUS communication bus, the control bit field contains control information related to the control of the communication process, the data length field indicates the length of the message data, the message data field contains all the data used in the communication between upper applications, the control commands that are used to control the operation of the XM tuner device are contained in the message data field as well.

The control commands of the XM tuner device are stored in the message data field of the IEBUS data frame, as shown in Figure 4. The control command data consists of a total of seven data fields, where TP is the type of a command, CMD is the unique number of a command within a command type, TL is the function address of the sending device and LN is the function address of the receiving device, LS and ST are the state information of the sending device of the microcontroller side and the device side, respectively, and CS is the checksum.

For each control command in any of the several classes of XM tuner device control commands, namely the preset station commands, the tune mode commands, the station scan commands, the channel adjustment commands, the information acquisition commands and the electronic diagnosis commands, there is a definite value pair of TP and CMD. The control process of the XM tuner device can be completed by filling the corresponding values in the TP and CMD field in the control command of the IEBUS data frame, setting TL and LN fields with the function addresses of the microcontroller and the XM tuner device, and writing the whole IEBUS data frame to the IEBUS communication bus to send the commands to the XM tuner device. Upon receiving this IEBUS data frame, the XM tuner device would resolve the control command from the microcontroller and perform actions corresponding to the command, thus, the operation request of the user is received and executed.

In addition to the XM tuner device control commands, display data frames that are sent back to the microcontrollers are also sent via the IEBUS communication bus. In the digital car audio system development project, the IEBUS data frame wring and reading operations on the microcontroller side are encapsulated into the IEBUS device communication protocol. The IEBUS device communication protocol is implemented by IDC (IEBUS Device Controller) and IDS (IEBUS Device Sender) tasks. The IDC task is responsible for the communication with upper layer modules and the control of the IDS task to send data frames; the IDS task receives directives and parameters from the IDC task, forms IEBUS data frame and writes it to the IEBUS communication bus using the IEBUS driving program. When the IDS task tries to write the IEBUS data frame, a possible occupation of the IEBUS communication bus by other external device may block

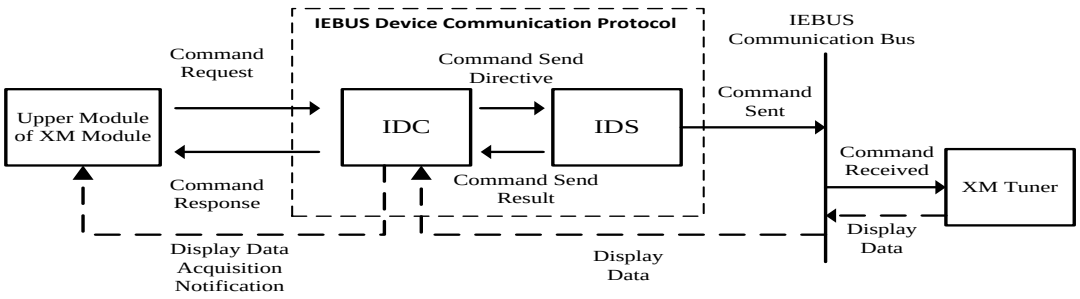

Figure 5. Data exchange of IEBUS device communication protocol.

the immediate completion of the data frame writing process. Hence, a maximum number of times of writing retries is set up in the IDS task. The writing retries continue before the retry count reaches the maximum number of retries. When the retry number limit is exceeded, the IDS task would give up on data frame writing and send busy state report message to the IDC task.

The IEBUS device communication protocol is at the driver layer of the XM function module, as shown in Figure 5, its encapsulation of IEBUS writing and reading operations offers the XM module the control command interfaces based on message communication. The XM module only need to set the event code and option fields of the message according to requirements of the IDC task and send the message to the IDC task, and then the IEBUS device communication protocol can recognize the message and write control commands to the IEBUS communication bus for the XM tuner device to take corresponding actions.

C. State Transition Matrix Layer of XM Module

There are multiple states for the XM broadcast radio function module during the real operation of the digital car audio system. The system state can be divided into *standby* and *normal* states, the XM module can also stay in *standby* and *normal* states. Because the car audio system have multiple sound sources, like XM, FM, AM ,CD and others, the XM module have *source on* and *source off* states. The user key press operations may cause the XM tuner device to enter into long lasting action states, like station scan. During this period of ongoing actions, the user might want to try changing the state of the XM tuner device by key presses, and the XM module need to choose between to accept or to reject such operation requests according to the current action state of the XM tuner device. At the end of a long lasting action, the XM tuner device would send state change notification to the XM module, and the XM module need to adjust its state accordingly to keep its action state consistency with the XM tuner device.

The existence of different states causes different processing methods of external inputs for the XM module. Hence, the XM module needs to establish a state mechanism to guarantee appropriate processing for each input request, response or notification message. The state transition matrix offers the XM module the method to establish the state mechanism, a XMSTM task is formed in the XM module to maintain the state transition matrix and process input messages.

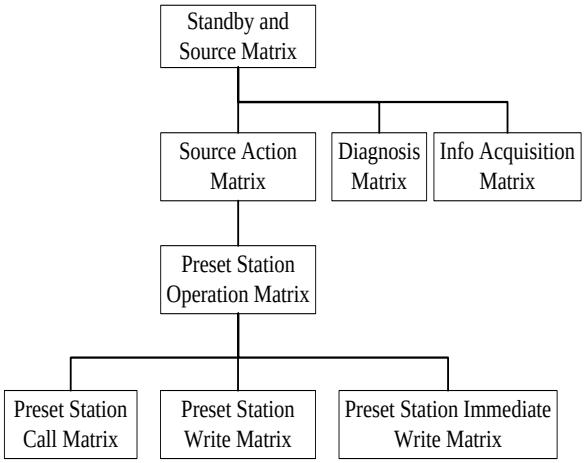

Figure 6. Tree hierarchy of state transition matrices of XMSTM task.

The state transition matrix is two-dimension table, with the two dimensions being the state and the event, the cells of the table contains the processing action and the target transition state. The state dimension corresponds to different states of the XM module, and the event dimension corresponds to the input messages of the XM module. When a message arrives at the XMSTM task, the XMSTM task would query the current state and look up for the corresponding cell along with the event contained in the message in the state transition matrix. With the content of each cell, the XMSTM task executes the processing action and alters the state of the XMSTM task to the target transition state. The completion of the state transition is also the completion of the processing of an input message, and then the XMSTM task will then wait for the next input message of the XM module.

There are a variety of events of input messages to the XMSTM task, whose source task may be one of the power manager task, the RPCAPP task, the sound task and the XMAPP task. The power manager task sends power management messages to notify the XM module to enter or leave *standby* state, the RPCAPP task sends to the XM tuner device control requests and XM sound source on/off requests, the sound task sends response messages of turning on or off the sound output in the XM tuner device control process, and the XMAPP task of the XM module sends response messages to report the result of XM tuner device control commands.

The working states of the XM module are divided into several layers, forming an obvious layered hierarchy. A top-down layered tree hierarchy stems from the containing relationship of states. Each group of states that can be changed by the same collection of events are organized into a state transition matrix and the event collection is mapped into one single event in the upper state transition matrix, in the upper matrix cell, a call to the lower matrix would start the processing process for each event in the collection of events. Therefore, the XMSTM task can use multiple state transition matrices in the tree hierarchy for state maintenance and message processing, as shown in Figure 6.

Standby and *normal* states would affect processing of other message events, the XM *source on* and *source off* events affect message events only next to *Standby* and *normal* states, these four states are put on the top layer of the XMSTM task to maintain. When the XM sound source is turned on, there are normal receiving, channel scan and preset transition states for each of the two tune modes, channel mode and category mode, there is also a channel rough up/down state for the channel mode, the seven states should be maintained considering the instant setting of the tune mode, they can form a state transition matrix at the second layer. Because all preset station operation-related XM tuner device control commands involves the enabling and disabling of the sound output, the third and fourth layer are formed for state maintenance of the preset operations. States associated with diagnosis and information acquisition operations are only affected by the state transition matrix at the top layer, so both of them are also put on the second layer.

D. Application Layer of XM Module

The application layer of the XM module lies between the state transition matrix layer and the driver layer, bridging the communication between the two layers. The state transition matrix layer takes processing actions whenever an input event arrives, for control command events, the processing actions include passing the type of the control command and corresponding parameters to the IEBUS device communication protocol and then drive the XM tuner device to perform corresponding operations. The application layer is primarily responsible for the conversion of message from the original form to the format ready for the IEBUS device communication protocol to send.

At this layer, API functions are defined for calls from the state transition matrix layer. API functions encapsulate the process of sending control command messages to the XMAPP task of the XM module. Upon receiving messages, the XMAPP task resolves the event type, converts the message into the required format and sends it to the IDC task of the IEBUS device communication protocol for control command generation and writing to the IEBUS communication bus.

The trunk of the XMAPP task is designed into an infinite loop, which begins by receiving an event from the associated mailbox into a variable whose type is a mail structure , subsequently the event code of the event is read from the variable whose type is the mail structure. The values of all event codes are divided into three consecutive segments, corresponding to three types of events consisting of request, response and information notice events. According to the requirements of XM radio functionality, there are several tens of kinds of request and response events, each pair of request event and its corresponding response event share nearly the same structure when it comes to event processing. Manual coding would generate abundant redundant code for the processing of the two kinds of events. This results in two consequences. On one hand, the memory occupation would dramatically increase when the whole project is compiled, on the other hand, the code comprehension and code maintenance in the future would be greatly complicated.

Therefore, a framework for event processing named as Process Engine is introduced to perform concentrated and standard processing on the request and response events.

1. Implementation Principle of Process Engine

A Process Engine can be seen as a collection of a group of associated process tables, each process table can be called as a top-level process table, the Process Engine uses an address table to record the starting addresses of all top-level process tables. In the Process Engine, the execution of a top-level process table corresponds to the processing of each pair of request and response events.

A process table is the collection of a group of processes, which is embodied as an array of process structure in the implementation program of the Process Engine. The processes of a process table lie in the array in a certain order, and each process is called a stage of a process table, which is referenced by array index corresponding to the process in the process table.

The smallest execution unit of a Process Engine is a process, the execution of a Process Engine is required to start from a specified process. Before the execution of a Process Engine, both the starting process table and the starting stage or the position of the starting process of the starting process table need to be specified when setting the Process Engine.

A process represents the execution of a series of programs, it could either be the execution of a process table consisting of other processes or merely be the execution of one single subroutine. The return value of the execution of processes in a process table can be one of two different return values, the control flow may jump into one of two branches according to the return value following the execution completion, and each branch is bound with one return value corresponding to the jump. The target of the jump may either be another process of the same process table, namely the stage of the process table or the tail of the process table containing the currently executing process, that is to say, following this jump, the execution of the whole process table is over. To non-top-level process tables, the control flow goes back to the execution environment of the process table which contains the current process, and the stage of that process table needs to jump correspondingly, similarly, two jump branches are also present, and the jump is performed according to the return value that would be set following the jump after the current process.

At the end of process execution, the jump target setting of two branches should also include information concerning continued or discontinued execution after the process jump of the Process Engine. For continued execution, the Process Engine performs the execution of the next process immediately after the process jump, while for discontinued execution, the Process Engine stops execution and exits after the process jump, and if no reset is performed on the starting process of the Process Engine before the next execution of the Process Engine, the execution would continue be performed right after the previous exiting process. This feature allows the Process

Engine to do the following, when the processing of a request event is finished, a process jump with discontinued execution may be done so that the execution of the Process Engine may stop before the processing process of the response event corresponding to the request event, until the response event arrives, the execution of the Process Engine is triggered and the processing of the response event may be completed.

The Process Engine uses a group of data structures and interface functions to implement the setting and execution initiation of the Process Engine, which is stated in detail in the following.

2. Data Structure of Process Engine

The mechanism of Process Engine relies on three data structures to implement: process control table block structure, process engine buffer element structure and process structure. The most important data structure is the process control table block structure, a variable defined in this type can be seen as a Process Engine. The process engine buffer element structure and the process structure are used to define the members of a process control table block structure variable respectively.

The process control table block structure contains a total of five comprising members, which are execution request setting flag, active buffer number, process engine buffer pointer, process engine process table address and subroutine address table, as shown in Figure 7.

The execution request setting flag is a one byte unsigned integral data whose effective value is either true of false, represented by 1 and 0, the field is used to mark whether the Process Engine has been requested to execute. Only by setting this field true before calling interface function to start the execution of Process Engine can the function call really start the execution of the Process Engine. Whenever the Process Engine finishes its execution, this field is set back to false.

The process engine buffer pointer is the pointer to process engine buffer element structure type, it points to the memory space required by the execution of Process Engine. Two members constitutes the process engine buffer element structure, they are process table number and execution stage number. The process table number

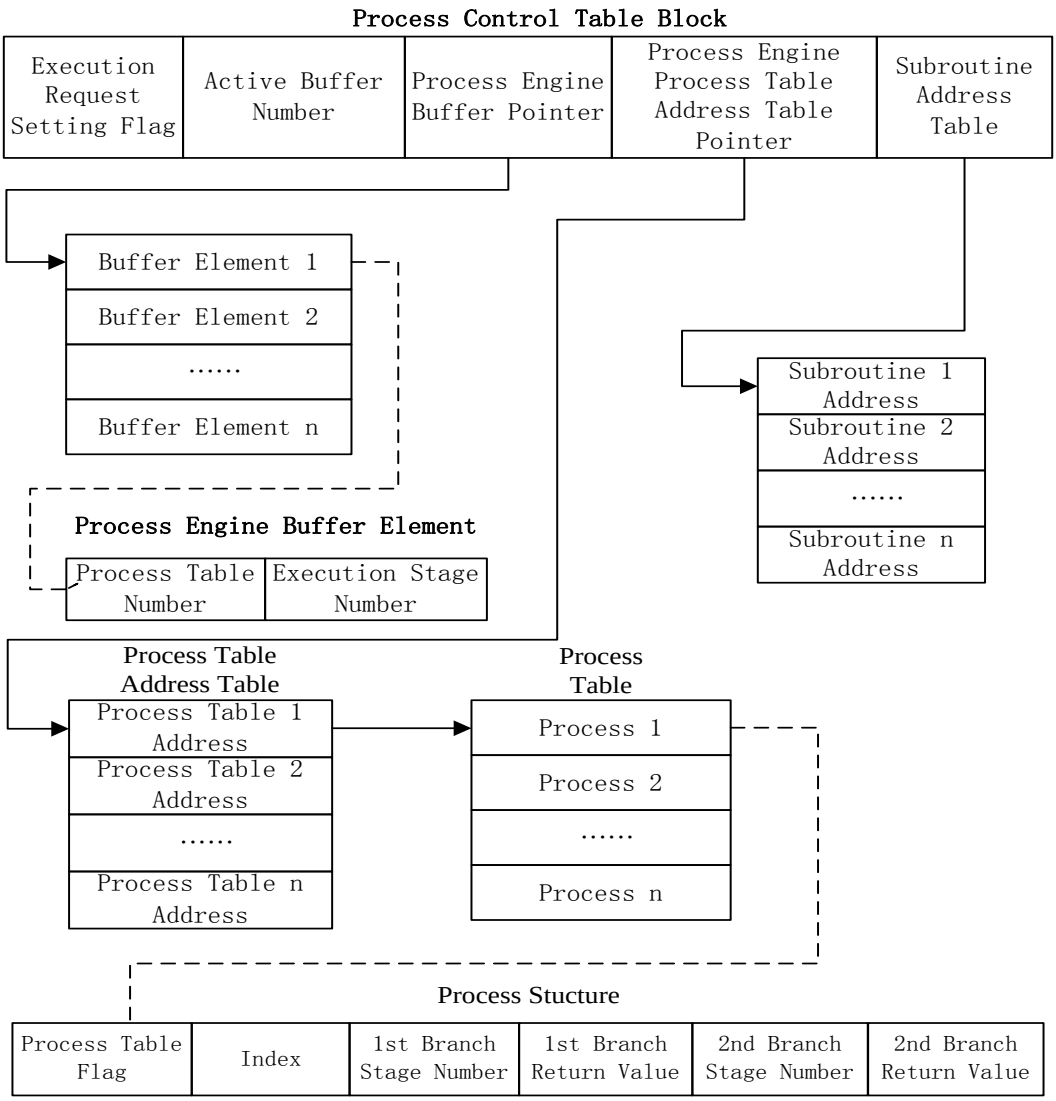

Figure 7. Data structure of process engine.

references the position in the process engine process table address table which holds the process table containing the current process, the execution stage number references the position of the current process in the current process table. Each element of the process engine buffer represents a process table and its stage of execution. At the beginning of the execution of the Process Engine, the first element of process engine buffer is provided for the first process to be executed by the Process Engine, the two members of the process engine buffer element structure together specify the process table and the stage to start from. Subsequently, with the advance of the execution of the Process Engine, the actual size process engine buffer would expand or shrink, each initialization of the execution of a process that represents a process table would cause the process engine buffer to expand by one element, and the process table number of the newly expanded element is the position of the process table in the process table address table, with the stage number of the newly expanded element being 0, the two members together specify the first stage of the process table to be the process to be executed next. If the execution comes to the end of the current process table, then the process engine buffer would shrink by one element, and the buffer element that represents the process table recently executed would be discarded, and the Process Engine would go back to the next stage of the process table one level higher over the discarded process table to continue the execution.

The active buffer number is a one byte unsigned integral value which is used to reference the element corresponding to the current executing process in the process engine buffer, this element records both the process table and the corresponding stage for the current executing process. The process engine buffer is in fact the stack memory required by the execution of the Process Engine, and each element in the stack represents the process table and its executing stage, the active buffer number is the current stack top pointer, with the advance of the execution of the Process Engine, the value of active buffer number increase or decrease to reflect the expansion and shrinking of the process engine buffer, and the process table to be executed is pushed into the stack and the process table to be discarded on execution completion is popped from the stack. When one process is over and the next stage of the same process table is going to be executed, the stage number of the process engine buffer referenced by the active buffer number needs also to be modified by the Process Engine to the stage number of the process to be executed, which is in fact the modification of the content of the top element in the stack memory for the advance of the execution.

The process engine process table address pointer is the pointer to the process address table, the type of each process table address table is the array of process structure. The process structure represents process, namely the smallest execution unit of the Process Engine. The process structure consists of a total of six members, process table flag, index, first branch stage number, first branch return value, second stage number and second stage return value. Both the process table flag and the index are one byte unsigned integral data, the process table flag is used to mark whether a process represents a subroutine or a process table, a subroutine is specified when the process table flag is set 0, the index is the position of the subroutine in the subroutine address table; a process table is specified when the process table is set 1, the index is the position of the process table in the process engine process table address table. The first branch stage number and second stage number is the target stage number following the current executed process, they could either be the actual target stage number to be executed next or one of the two special values that indicate the finish of the current process table with 0 or 1 as return value respectively. The first branch return value and second branch return value are the return values to be set for the whole process engine on the finish of every process, usually, this return value is used to return certain values concerning the timeout of message reception of tasks, which limits the longest waiting time for continued execution of the process engine.

The subroutine address table is defined as the array of function pointers of all subroutines used by the process engine. Because an array is formed to facilitate the process to reference the subroutine by index, all subroutines should have the same prototype, with no arguments present and one byte integral value be returned as return value. Subroutines that require argument input should use global variables to pass necessary input before any function call. The return values of subroutines are formulated into only two different values, corresponding to the two return branches on process execution completion, the two values represent whether the control flow continues along the first branch or the second branch respectively.

3. Interface Function of Process Engine

The interface function of process engine is used for initialization setting, state query and ultimate execution start of process engine. Defining a process control table block structure variable begins the use of a process engine, this variable record all the setting information and execution environment for the whole process engine. The process control table block structure variable will be used as an argument throughout all interface function to construct and operate the process engine.

Four steps are needed to create, set and execute a process engine, corresponding to four interface function calls, process engine set interface, execution request set interface, starting process set interface and process engine execution interface.

The process engine set interface does initialization settings on the process engine, besides arguments like process control table block structure, several members that need to be initialized externally are also included in the arguments, the process engine buffer pointer, process engine process table address table pointer and subroutine address table. The interface associate the externally created resources to the process engine and clear state values like execution request setting flag and active buffer number back to the original state.

The execution request setting interface has process control table block structure as its only argument. Only by setting the execution request flag to valid value can the process engine execution interface function call effectively start the execution of the process engine.

The starting process setting interface has arguments of process control table block structure and the process table number of the starting process. This interface sets the active buffer number to zero, set the process table number of the active buffer element according to the argument value, and initialize to zero the execution stage number of the active buffer element. This interface makes the first stage of the specified process table the process to start from of the whole process engine.

The process engine execution interface has only process control table block structure as its argument, when the execution request setting flag of this structure is set to valid state, the process engine is able to execute according to existing settings.

VII. CONCLUSION

This paper shows the design ideology and implementation method of the XM satellite broadcast radio function module in a software application development scenario with ITRON standard compliant real-time operating system and hierarchical architecture of a digital car audio system project. The XM function module is installed in the Rx850 embedded operating system within the digital car audio system project after compilation. After software defect elimination in the testing phases of unit test, integration test and system test, the real operation results of the digital car audio system suggests that the XM function module is compatible with other function modules in the system, it responds to user operations correctly and timely, and it is able to drive the XM tuner device to receive and play XM broadcast radio program.

ACKNOWLEDGEMENT

This work is supported by the Open Research Fund from the Key Laboratory for Computer Network and Information Integration (Southeast University, Ministry of Education, China), the Fundamental Research Funds for the Central Universities, the Program of National Natural Science of China (Grant No. 60933011 and 61073014), and the State Key Development Program for Basic Research of China (Grant No. 2011CB302902)

REFERENCES

[1] J. Kontro, A. Koski, J. Sjoberg, "Digital car audio system," *IEEE Transactions on consumer electronics*, vol. 39, pp. 514-521, August 1993.

[2] A. Hassan,, "RTK-spec TRON: A simulation model of an ITRON based RTOS kernel in SystemC," *Proceedings - Design, Automation and Test in Europe*, vol. 1, pp. 554-559, 2005.

[3] H. Du, S. Wang, X. Sun, "Research of hybrid OS architecture based on Linux on ITRON," *Jisuanji Gongcheng/Computer Engineering*, vol. 32, pp. 84-86, 2006.

[4] Mariottini, Francesco, "Design of a compact GPS and SDARS integrated antenna for automotive applications," *IEEE Antennas and Wireless Propagation Letters*, vol. 9, pp. 405-408, 2010.

[5] S. DiPierro, R. Akturan, R. Michalski, "Sirius XM satellite radio system overview and services," *2010 5th Advanced Satellite Multimedia Systems Conference and the 11th Signal Processing for Space Communications Workshop*, pp. 506-11, April 2010.

[6] M. Nariman, H. Hamed, F. Mehdi, "Digital audio broadcasting system modeling and hardware implementation," *Advanced Microsystems for Automotive Applications 2004*, pp. 313-324, 2004.