International Journal of Research in Engineering and Science (IJRES)
ISSN (Online): 2320-9364, ISSN (Print): 2320-9356
www.ijres.org Volume 5 Issue 6 ǁ June. 2017 ǁ PP. 13-23

The Design of an MVB Communication Controller Based on an FPGA

SHI Yong-gang, YU Chao-gang
(The School of Urban Railway Transportation, Shanghai University of Engineering Science, Songjiang Shanghai 201620, China)

Abstract: *According to the TCN standard (Train Communication Network Standard), in order to design the MVB controller simply and quickly, this paper presents an new design idea, which avoids the cumbersome process in the traditional design process and makes the MVB controller design become efficient and concise. The design realizes the real-time protocol associated with the multifunction vehicle bus (MVB) device and the design process for all layers associated with the MVB device link layer. The design is a concurrent, easy-to-parameter and reconfigurable top-down design process that is implemented through programmable gate arrays (FPGAs) and related circuits. The design provides an efficient and rigorous design idea, and was verified by some experiments successfully.*

Keywords: *MVB; Link Layer; Process Data; Message Data*

I. INTRODUCTION

The TCN is a network for connecting data communication between the on-board equipment of a rail vehicle. It includes two serial buses: the MVB (Multipurpose Vehicle Bus), and the WTB (Train Bus), both of which are capable of independent configuration. The MVB also provides three different physical media, the electrical short distance (ESD), Electrical distance (EMD) and optical glass fiber (OGF) [1], as shown in Fig1.

TCN transmits two types of data: the data with high timeliness for periodic transmission is called Process Data(PD); messages that are not urgent and may be sporadically transmitted(MD). TCN also transmits specific monitoring and control management data (ie, supervisory data). Both the MVB and the WTB are master-slave configured buses that include one or more master devices capable of bus management capabilities (BA Devices) that have the bus management function of the primary device to be periodically managed and Distribute the transmission of data between different devices[2].

This paper presents a link layer design idea for MVB equipment based on train communication network (TCN) and multi-function vehicle bus (MVB) standard. According to the standard (IEC61375), this design takes into account the whole process of the link layer and Its real-time protocol, the Link Layer Process Data Interface (LPI), the Link Layer Message Data Interface (LMI), and the Link Layer Monitoring Interface (LSI), implement a full suite of standards related to the MVB device link layer standards All agreements, all the processes and some other free options[3].

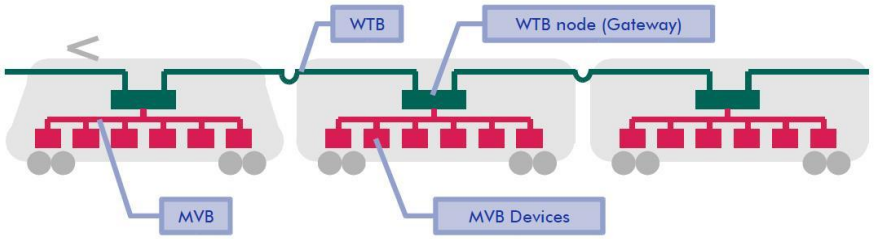

Figure 1: The Structure Of WTB and MVB

1. Data Storage Structure Of the Device

This design is the function of three types of equipment, but also compatible with Class 1 and 2 types of equipment functions, and support some of the four categories and five types of equipment functions. In this design, the MVB device MVB_Status object reads the device class and reconfigures its performance accordingly. The link layer data memory structure of each MVB device is shown in Fig 2:

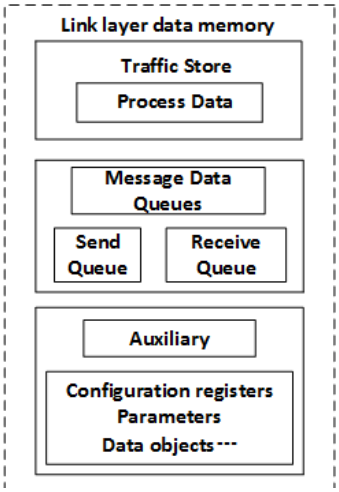

Figure 2: Link Layer Data Memory Structure

The first part, the bus and the user can access the shared memory Traffic Store;
The second part, Message Data Queues, accesses the ordered set of frames in a first-in, first-out (FIFO) queue, including the reception (or input) and the transmission (or output) of the message data;
The third part, Auxiliary, is dedicated to storing supplemental information to ensure proper link layer management. The auxiliary memory contains configuration registers, link-layer device parameters, and standard-specified data objects.

2. Link Layer Architecture for MVB Devices

The link layer of the MVB device is present between the physical layers and is linked to the intermediate associated channel driver and the upper layer via the real-time protocol. The TCN standard specifies the different services provided by the link layer of the MVB device. As mentioned earlier, in the MVB Class 3 device (same as Class 1 and Class 2 devices), the design consists of several parts as described below, According to the link layer function we divided into 11 dedicated units, as shown in Figure 3:

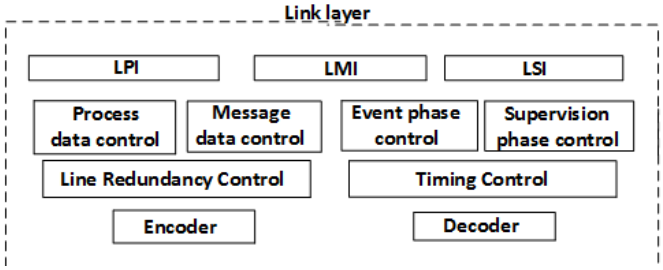

Figure 3: MVB Class 3 Equipment Link Layer Structure

The link layer unit of the MVB device is composed of an encoder, a decoder, a line redundancy control, a timing control, a process data control, a message data control, an event phase control, a monitoring phase control, an LPI, an LMI and an LSI part. A 16-bit wide bus connects these units together. The encoder and decoder units are dedicated to the encoding and decoding of data frames, CS calculations, checksums and data frame length checks, and the line redundancy control unit is responsible for bus communication control and trusted line selection. The timing control unit is designed to check the real-time nature of bus communication. The process data and message data control unit is intended to perform all mandatory tasks. The process data control section needs to manage the process data (PD) in the link layer Traffic Store, and the message data control section needs to manage the link layer input and output message queues FIFO), which also requires checking the value of the data Freshness and correctly activating the real-time interface (LPI and LMI) program or the program within the link layer [4].

Once the bus master requests a pending event for its slave device, the event phase control unit begins the control of the link layer, such as sending some pending messages to a device that has not yet been asked by the master but still on the bus Other devices, these messages can be process data or message data. Once the bus controller needs to acquire the device status of the MVB device, the monitoring phase control unit begins to be responsible for the control of the link layer. The requested information is obtained from the MVB_Status structure of the LSI. The host needs this information to check the slave device State is normal.

The LPI, LMI, and LSI units are designed to control the correct execution of different link layer procedures and the execution of the LPI application layer program and the LMI network layer program specified by the protocol standard. Figure 4 shows the structure of the encoder and line redundancy control based on the twisted pair system. The two lines share an encoder, but each line has a decoder, and the line redundancy control unit selects the correct decoder based on the information of the current trusted line.

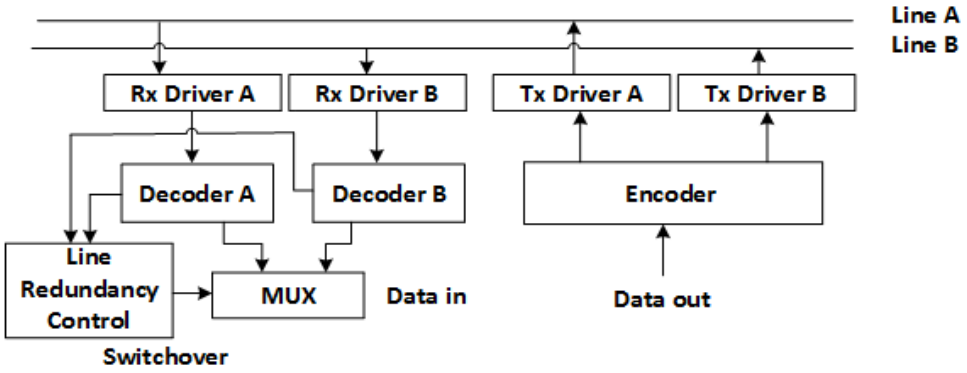

Figure 4: Encoder, Decoder and MVB Two-Wire Redundant System Structure

3. Encoder design

The encoder unit conforms to the TCN channel coding specification, i.e., data and non-data symbols, and the start and end characters for frame synchronization. The start separator depends on the frame type: the main frame or the slave frame. The end separator depends on the media type: ESD, EMD, or OGF. The data frame structure contains three fields, the start bit plus the start delimiter, the frame data, and the end delimiter. The frame data content includes a frame type (F_code), a main frame address, and a parity sequence (CS).

The "Device Status Response" of the decoder is used as a flag for the device status reporting condition (the state of each device is respected by the encoder to respond to an operation from the frame transmission to the line redundancy control unit). Figure 5 shows the internal structure of the encoder unit. The figures and the arrows shown in the accompanying drawings show information that is acquired / returned or input / output at the functional design layer. These tags are the mnemonics for their related content (see Section 16 for some of our synthesizable designs and their ports).

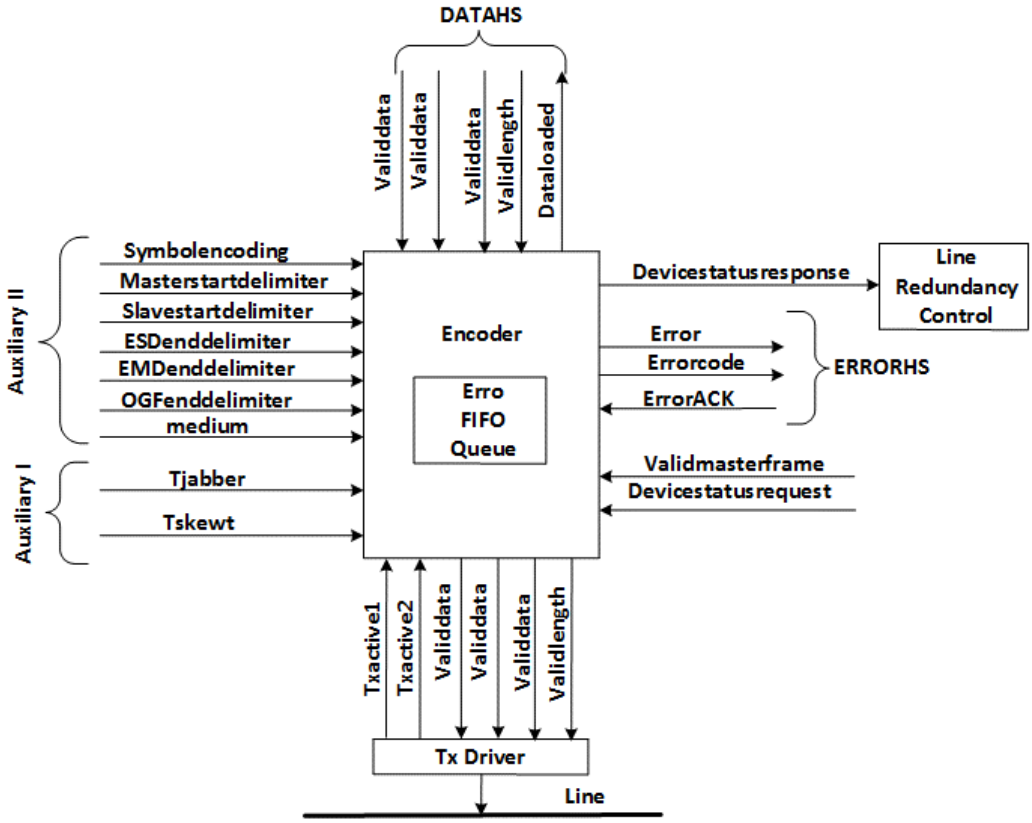

Figure 5: Encoder Internal Structure

In Figure 5, Auxiliary II gives information on channel coding and media-related signaling. Each of the Auxiliary II has its own registers in the link layer data memory structure (Figure 2) and is initialized when the system is initialized load. "DATAHS" shows the handshake protocol for user data transfer to other units. "ERRORHS" shows a handshake protocol for error code management, where the error code is stored in the error FIFO queue. "Validmasterframe" is activated by the decoder each time an active and correct main frame is received. If the previous Jabber stop condition has stopped the Tx driver, the decoder will force the Tx Driver to be activated. Some drive control flows are used to manage the execution of the driver (Figure 5, shown at the bottom).

4. The Structural Design Of the Decoder

The decoder unit is to check for valid and correct receive frames and also have the correct length since the unit transfers different frame data to other device units if and only if all of these conditions are met. The design of the decoder uses oversampling for bit synchronization, including jitter-compliant channel and data FIFO queues, and bilinear feedback shift registers (mainly 128-bit and 256-bit data lengths) for fast CS verification, Receive data and transfer it to other units, after the end of the received frame and in a very short clock cycle.

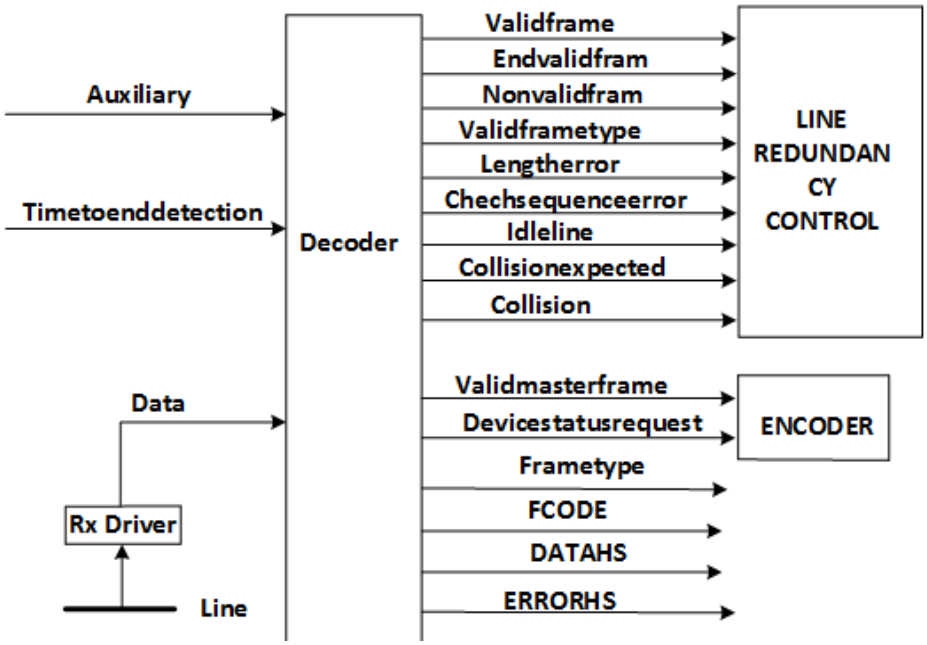

Figure 6: Decoder Internal Design Structure

Fig. 6 shows the structure of the decoder unit. Auxiliary II (left) is the same as in Figure 5. Once the end delimiter of a data frame is received, "Timeenddetection" checks whether the frame ends the transfer. "DATAHS" and "ERRORHS" (left) have the same meaning as in Fig. "Frametype" indicates the frame type (master or slave) of the received frame. "Fcode" reports the received Fcode code each time the main frame is received. The device status request "Devicestatusrequest" is activated each time the device status request master frame is received.

The information required for the line redundancy control unit to check the trust line (see section 7) is also shown in Figure 6. Idleline "means that the bus is idle; Collisionexpected will be activated whenever multiple of the MVB devices will be answered when the received primary frame has" 1001 "(General_Event_Request) or" 1101 "(Group_Event_request) for its Fcode; "Collision" is activated whenever there is a bus activity, but no valid frame is detected, and the last received main Fcode is "1001" or "1101"; the contents of the remaining part of the arrow can be easily To infer from their labels.

5. Control Design of Line Redundancy

The line redundancy control unit is used to receive information for the line selection and handover from the decoder. Each time the device is initialized, Line_A defaults to Trusted_Line ("LAT" = '1'), making Line_B an Observed_ Line, and "RLD" = '0'. Once initialized, the line configuration depends on the "sla" and "slb" bit values (see Figure 7). The "LAT" and "RLD" bits belong to the MVB_Status object, and the "sla" and "slb" bits belong to the MVB_Control structure (see Section 15). When "sla" = slb "= '1' is a two-line system.

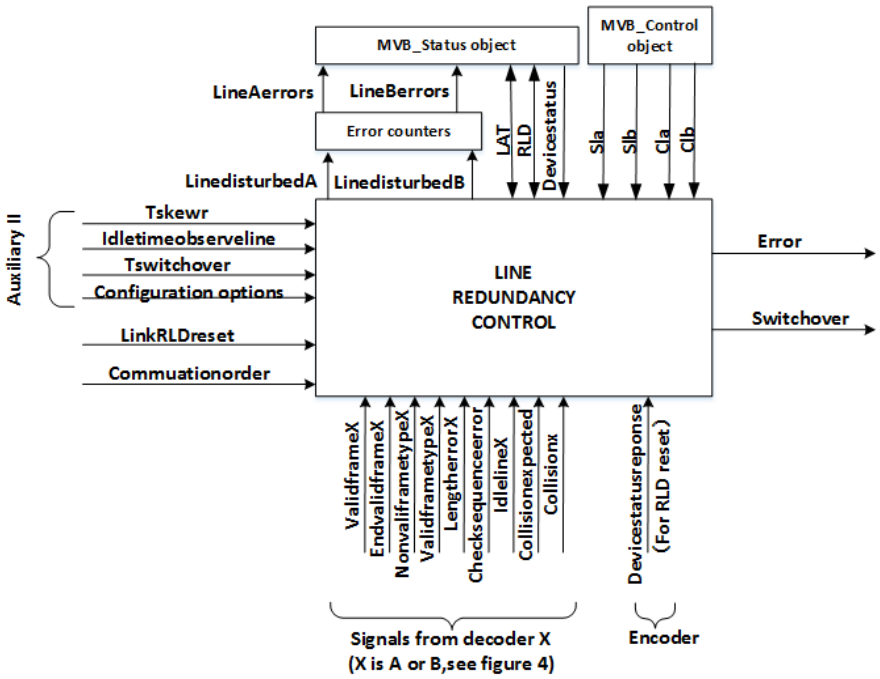

Figure 7: Double-line Redundant Control Unit Structure

The values of the "LAT" and "RLD" bits depend on different conditions, and their values are updated by the link layer. At the same time, "RLD" is set whenever Observed_Line is interrupted, and it is reset whenever the device sends a device status frame to the master device, for example, "Devivestatusrequest" is activated by the encoder.

Auxiliary III (see Figure 7) is used to toggle, and if activated, "LinkRLDreset" directly resets "RLD" and "Commutationorder" forces the toggle action. Note that some of the switching conditions depend on some user's options. "ERRORHS" (Figure 7) has been previously described. "Switchover" (Fig. 7) is dedicated to receiving data transmitted from one of the two decoders in (Fig. 4).

6. Timing Control design

The timing control unit checks the correct timing of the bus communication according to the specified T_ignore and T_source parameters. The T_source refers to the end of the received master frame (see Figure 8, 'MendX') that is received on the master device. The maximum run time of the active frame and the correct frame from the frame (see Figure 8, 'SstartX') of this main frame, and T_ignore is the result of the measurement from the end of the received and valid master frame that is measured on the MVB device The maximum run time of the main frame from the frame, and both parameters are measured on the Bus side.

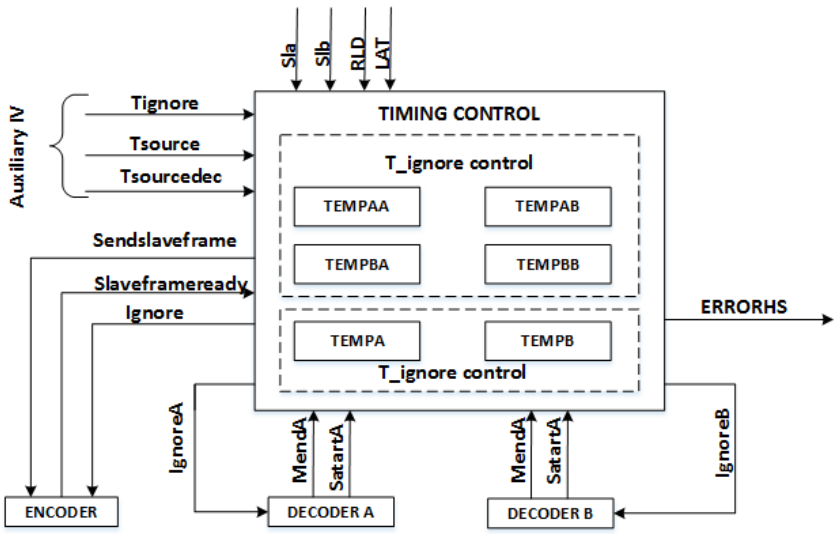

Figure 8: Timing Control Unit Structure

Therefore, the design checks the effective and correct host frame response (not just any host frame reception), and uses "MendX" to activate the time to start T_ignore. For the same driver and decoder for the main frame and from the frame processing delay, you can check the decoder side T_ignore: from the "MendX" activation to "SstartX" activation time required must be less than T_ignore. (X depends on whether the decoder A or B first activates it, see Figure 4).

The T_source check is more complicated due to the delay accumulation of the decoder and the driver. For example, in this design, the time elapsed from "MendX" to "Slaveframeready" is less than T_source minus T_sourcedec The inspection is more complicated. And it is necessary to take into account that Trusted_Line may change from counting to counting stop, our timing control unit has four counters dedicated to the T_ignore check, but it has only two counters dedicated to the T_source check (there is a stop condition for it: "Slaveframeready" activated). For the correct check, the values of "LAT", "RLD", "sla" and "slb" are read each time the stop condition occurs. "IgnoreA" or "IgnoreB" is activated each time the check is incorrect, and the associated frame is discarded.

7. The Design Of PD Control

The address of the process data needs to be identified on the bus. If the two devices are not at the same time as the source port, each device port can be configured as a source port or a sink port according to the standard. The periodic list contains the process data address and the address of each device connected to the bus for the main frame information request (see Section 11). Each process data is stored as a dataset in the Traffic Store (Figure 9). For the control of Freshness, each dataset has a relevant time (Freshness), and dataset and Freshness will be updated [5]. Once each process data is sent or received, the link layer initiates the "ap_event" program belonging to the LPI layer (see Section 13), but it can also perform some internal processes that are fully implemented in hardware by the hardware, And is simply identified by the corresponding program code (as shown in the left part of Figure 9).

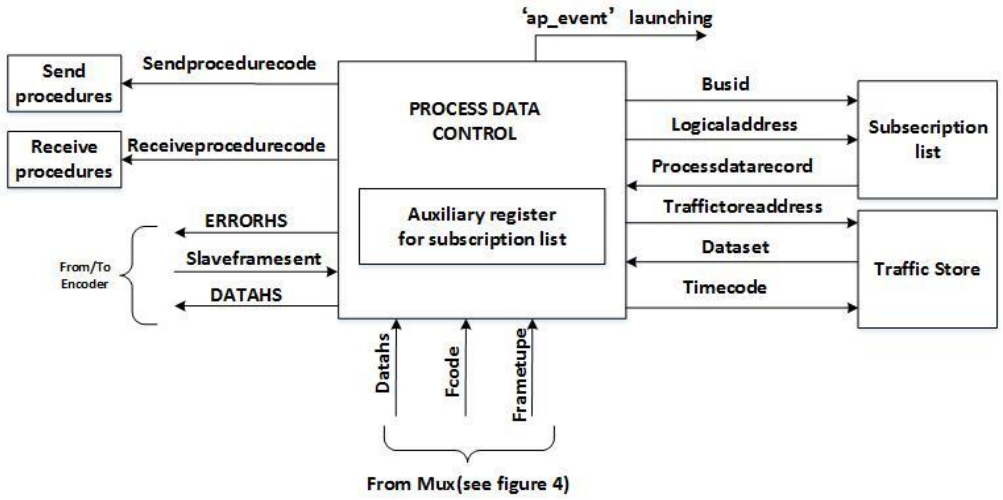

Figure 9: PD Control Unit Structure Diagram

The subscribe list that is loaded when the device is initialized also exists (as shown in the right part of Figure 9). It contains the information needed for process data management, such as sending and receiving process code, instance code, process data length, source or sink port, etc. However, TCN is not explicitly defined so that implementation is not restricted. "Fcode" and "Frametype" (Figure 9, bottom) are activated by the process data master frame request or process data received from the frame. "DATAHS" (Figure 9, bottom and left) supports data handshake protocols. "Slaveframesent" (Figure 9, left) indicates that the process data has been sent from the frame.

8. Message Data Control

The Message_Data function supports device data transfer, allowing device reconfiguration and reprogramming. The host will request the device to suspend the message data identified by its device address through the device address. In response, the MessageData from the frame structure contains the original and destination device addresses, and the broadcast message data has a special destination address. At the same time, the TCN standard defines some of the network layer programs that are called by the LMI (see Section 13). The network layer process code is loaded into the link layer memory at initialization time. The message data has a different structure at the network layer and is defined as a packet. The network layer packet storage area is

called a packet pool. The design includes sending (or outputting) a packet pool and receiving (or entering) a packet pool. The message data has its own Freshness to check the parameters of TCN PACK_LIFE_TIME.

Figure 10 shows the structure of the message data control unit. Auxiliary V includes "Messagepriority" and "Messagepriority" contains a three-part function for the output queue priority of the event phase control (see Section 11), and also includes "Networklayerprocedurecodes" and "Messagelinkstatus". The TCN defines an 8-bit message link status register, but only specifies the three bits of the 8 bits: once the message data is received from the frame, the bit1 bit is set to "1"; once the message data has been , Bit 2 is set to "1"; bit4 is set to "1" once message is received from the frame but the input queue is full.

As shown in Figure 10, the link layer accesses the packet state when the data pool is sent, that is, once each message data is sent, the link layer updates the packet status field in the sending pool. The information shown in the top part of Figure 10 shows the network layer process to be started. The process begins with the execution of the link layer, the network layer exception, and the "nm_status_indicate" process. After each message data is sent, the "nm_send_confirm" The "nm_receive_indicate" procedure is executed after each message data is received. The remaining information is used for input / output queue management or communication between the encoder and the decoder. It should be noted that although some of the "nm_status_indicate" start conditions exist in the design are not standard, they are still compliant with the standard, which are supported by two new parameters, MD_TIMEOUT, for reporting the packet lifetime timeout condition; MD_RQ_OVERFLOW, The status of the report input queue is full.

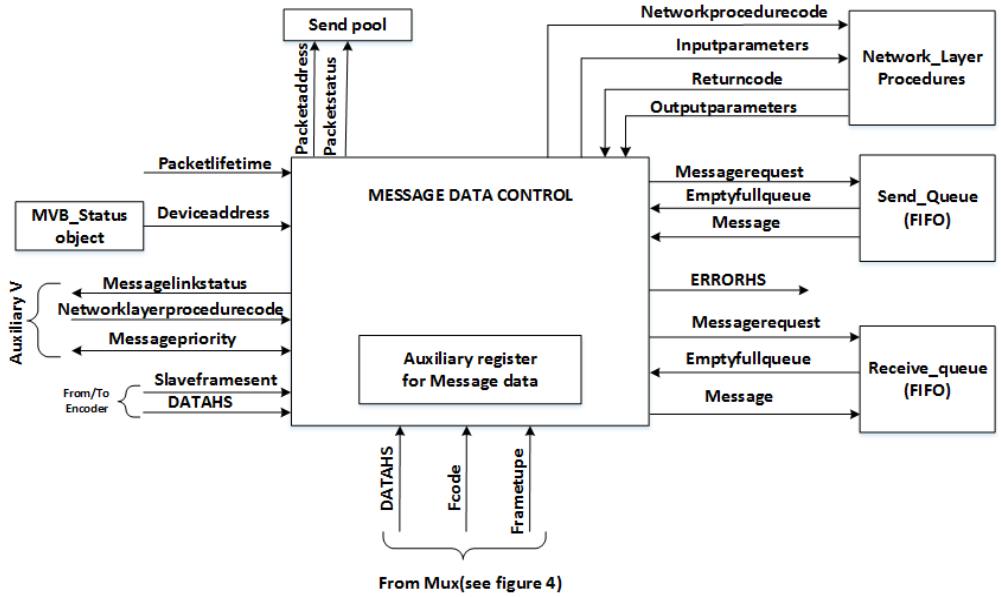

Figure 10: Message Data Control Unit Structure

9. Event Phase Control

In TCN, each device with BA capability is likely to become a master device in each polling cycle. Each rotation is divided into multiple basic periods. Each basic period is divided into periodic and incidental phases. Each incident phase is divided into monitoring phase, event phase and protection phase. The periodic phase is mainly used for process data requests, but message data requests are also possible. The monitoring phase contains the device status request and the master transfer[6]. The event phase is mainly used for message data transmission, but process data transmission is also possible. The protection phase is mainly for the master device to send a protection phase as a buffer after the sending of the spindles to provide the correct start of the next cycle phase. If the maximum response time to send is greater than the remaining time to the next basic period, the master will not send the primary frame.

During the event phase, the master asks the slave, "What data is needed to send to another device?", Each MVB device can send messages to the master that they are waiting to send, but not all MVB devices are allowed to respond. The bus management function will be presented in a future paper. The bus manager can manage the distribution of data by scanning the periodic table, sometimes depending on the MVB device's request (eg, the master device does not request and is not provided by the MVB device Including the process data in the periodic table). This option is consistent with the TCN specification, which also contributes to the correct performance of the system in the case of incorrect periodic table initialization. At the same time, the execution of each event phase process is accompanied by a request for a pending event by the device, and then the MVB device for each pending message has to send its device address. Each MVB device must send its

device address (for pending message data) or its process data address (for pending process data); then, the master device requests the message data / process data indicated; finally, the MVB device activates Its message / process data control unit and sends the requested information. And has two rounds of priority (low and high) and four polling modes. Each message data output queue has one of these priorities (note that the standard specifies this is necessary) so that only suspensions that have a priority declared at the time of round robin initialization can send data along such a round. Considering that the remaining modes are not clearly stated by the TCN standard, the design considers only the patterns "0000" and "0001".

Since the TCN standard does not give any description of the traffic store in the Event Phase, the four FIFO queues, namely PDFIFO0 to PDFIFO3, are implemented in the design (see Figure 11) to manage the process that is not requested in the basic cycle phase Data, which is set according to their different priorities, that is, have a high priority PDFIFO0 or lower priority PDFIFO1 advanced priority rounds than a high priority message data cycle has a higher priority, yet Or have a low priority of PDFIFO or lower priority The low priority of PDFIFO3 is lower than the priority of a low priority message data.

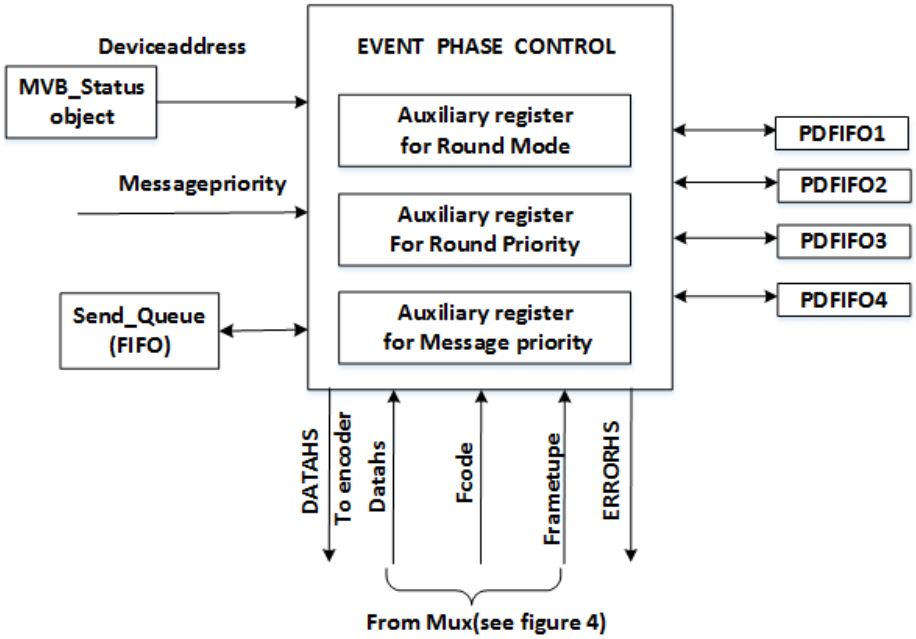

Figure 11: The Structure of Event Phase Control Unit

10. Monitoring phase control

The monitoring phase control unit is as simple as its task, and once it receives a request for a valid and correct master device for device status, it retrieves its device address and its device status from its MVB_Status object register and gives a command to send them[5].

11. Link layer interface (LPI) for process data

Because the memory contents of the different device layers are independent, some data transfer interfaces between them are mandatory. The LPI includes standardized object structures and procedures for memory (e.g., Traffic Store) and application layer memory data transfer for the link layer. The application process is initiated by the link layer and executed by the application layer, and the link process is initiated by the application layer and executed by the link layer, and there is also an initialization process for the initialization of the link layer memory, the parameter and the register value loading, As well as the monitoring phase process for interface control.

The design takes advantage of an embedded processor (see Figure 12) that is embodied in the application layer, so the application's startup will use interrupts while the linker starts using the port signal. Input and output process parameters and return values are port transmission. We should note that the application layer memory should be accessed by the application layer and the link layer, but the Traffic Store is accessed only by the link layer. Thus, each data transmission process is performed by the link layer.

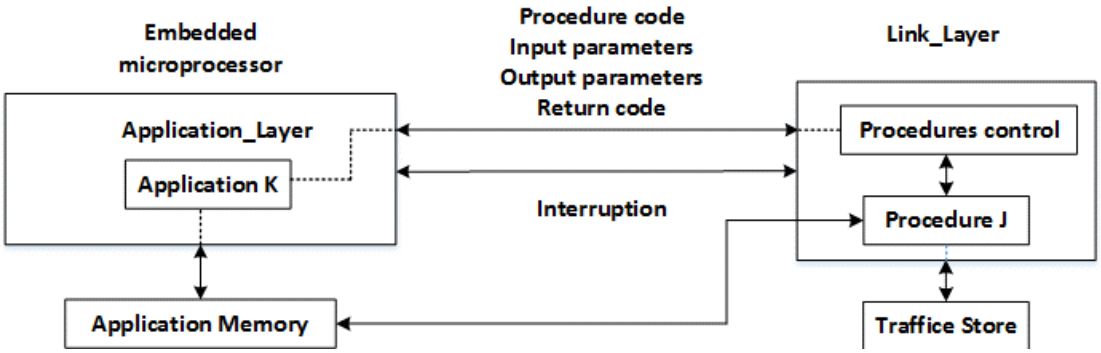

Figure 12: LPI Unit

All LPI link layer processes (described briefly in the next paragraph) are implemented in hardware by the hardware in order to achieve faster real-time performance. "Lp_init" is the initialization process. It loads the Subscribe list (Figure 9), specifying the addresses of the objects and registers that exist in the link-tier memory (Figure 2), and in this design, the options for initializing all Traffic stores are given. "Lp_put_dataset" is responsible for copying the process data from the application layer store to the link layer Traffic Store, and "lp_get_dataset" is responsible for copying the process data from the link layer Traffic Store to the application layer memory. After running the "ap_event" procedure, the link layer reports the process data for each transmitted or received to the application layer.

TCN specifies two additional supervisory processes, namely "ds_subscribe" and "ds_desubscribe". These processes are initiated by the application layer and executed by the link layer, and they update the Subscribe list. We have also added some other possibilities to the design (not specified in TCN, but meet the criteria), allowing real-time maintenance of the Subscribe list.

12. Message Data Link Layer Interface (LMI)

There are three types of LMI processes according to different related tasks:
1) message data transmission;
2) message data reception;
3) interface initialization and monitoring.
There are five steps.
1) the network layer has been prepared to distribute data;
2) the network layer starts the "lm_send_request" process;
3) During the operation of the link layer "lm_send_request", the packet is transferred to the output queue and becomes message data, and the packet status is updated to MD_PENDING in the output packet pool.
4) Upon receipt of a request from the master for message data, the MVB device sends the message data to be processed with the longest stay in its output queue.
5) Once the message data is sent, the link layer updates the packet status to MD_SENT in the output packet pool and initiates the "nm_send_confirm" procedure that will be run by the network layer to release the allocated memory output packet memory pool.

13. Link Monitoring Interface (LSI)

According to the standard, the LSI of the class 3 device has only three objects:
(1) MVB_Status; (2) MVB_Control; (3) MVB_Report.
When the system is initialized, they are loaded into the link-layer memory and updated by a previously designed control unit or a monitoring process.
The MVB_Status object has seven fields: (1) device_address; (2) mvb_hardware_name; (3) mvb_software_name; (4) device_ status (5) t_ignore; (6) lineA_errors; (7) lineB_errors.
The device_status field has three subfields: capabilities, class_specific, and common_flags (for example, "LAT" and "RLD").
The MVB_Control object contains three fields: (1) device_deit; (2) t_ignore; (3) command, which includes "sla", "slb", "cla" and "clb" bits.
Note the different definitions of the t_ignore fields in the MVB_Status (eight unsigned bits) and MVB_Control (16 unsigned bits) objects given by the standard.
We used 8 unsigned bits for MVB_Status and MVB_Control. In addition, we added a reserved field to the MVB_Control object to keep the memory aligned.

14. Implementation Of The Core Controller

We implemented the simulation through the digilent D2E board and our own driver interface board. Once fully simulated, our design at the link layer will be verified and tested. Each D2E board includes a Xilinx XC2S200E-PQ208 field programmable gate array (FPGA) with 200000 system gates, 4704 logic cells and 56K bit dual port block random access memory, and parallel access in each port and time division multiplexing have been widely used in link layer unit interface and real-time protocol implementation, but also to maintain data consistency.

The full implementation of our link layer requires more than 7,000 logical units, so hardware validation has been partitioned according to the XC2S400E-PQ208 resource. In particular, we examined the robustness of the design and its most critical features, namely, decoder unit bit and frame synchronization and jitter tolerance. At the same time oversampling and decision circuit technology and channel FIFO and data FIFO queues have been used for this function.

Based on the XC2S400E FPGA, PicoBlaze or MicroBlaze soft cores must be added to the model if more complex application layer requirements are required. It also needs to evaluate the feasibility of porting the design to ProASIC3, ProASICPLUS and Fusion Actel. For embedded analog-to-digital converter applications, the design uses the AFS1500, the goal is to enable the FPGA to get all the MVB device layer[4].

15. Simulation and Test Results

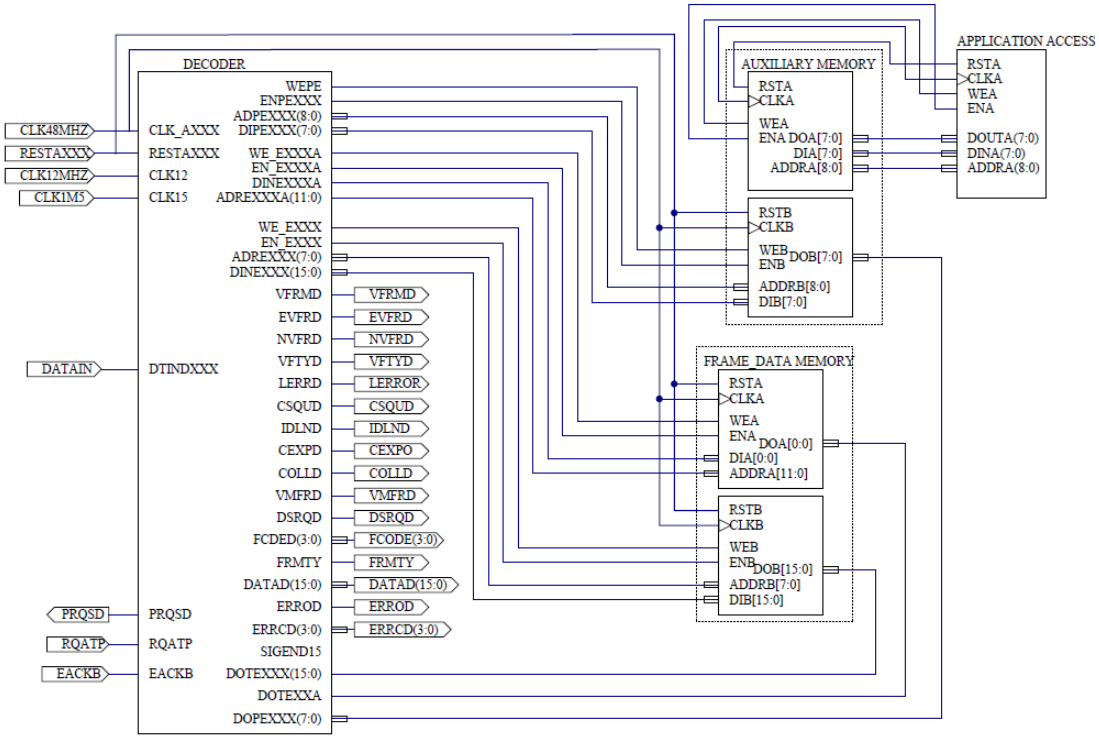

Figure 13: Decoder Structure Diagram

The decoder uses three different clock signals: (1) 48 MHz (memory access); (2) 12 MHz (channel oversampling) (3) 1.5 MHz (data bits). At the same time, the dynamic link library, the clock buffer and the reasonable allocation of the clock is also a reasonable application. The driver output receiver channel is also connected to the DATAIN port, and the port from VFRMD to DSRQD corresponds to the right half as shown in Figure 6. PRQSD, RQATP, FCDED and DATAD ports to achieve data transfer handshake protocol ("DATAHS" in Figure 6). ERROD, ERRCD, and EACKB ports to implement the error code handshake protocol ("ERRORHS" in Figure 6). Finally, the FCDED and FRMTY ports are used for "Fcode" and "Frametype" in Figure 6, respectively.

Once the decoder is initialized, the decoder will wait for a new data frame, and some other units will systematically evaluate the correct decoder's port. When the decoder receives and decodes the valid and correct frame, it will The frame is stored in its FRAME_DATA MEMORY and the PRQSD port is activated. The associated link layer unit then activates the RQATP port and requests the received data set via the DATAD bus. The following code consisting of 1, 0 shows that a 64-bit correct and valid slave frame process consisting of Manchester encoding is shown in Table 1:

Table 1: Transmission Process

Slave_Start_Delimiter	101010100011100011
Frame_Data	0101101001101001100101100101101 0 1001100110100110101001101001011 0 0101101001101001100101100101101 0 1001100110100110101001101001011 0 0110010101010110(CS)
End_Delimiter	0000(ESD)，0011(EMD)

The FRAME_DATA of the decoder is derived from the decoder's Frame_Data value on the decoder address 0 to 3 (addrexxx), as shown in Table 2:

Table 2: Memory Address

address 0	0011011010010011
address 1	1010110111011001
address 2	0011011010010011
address 3	1010110111011001

CLK_axxx is the clock signal that manages the data transfer from FRAME_DATA MEMORY (Figure 13) to the functional unit. The functional unit has the capability to process the Dataset dataset. Finally, summarize the performance of the design under unsatisfactory conditions: when sending / receiving billions of known data frames on ESD and EMD physical media, The worst test bit rate (BER) is $1.4 \cdot 10^{-10}$.

If the use of low-frequency and high-frequency high-power interference, but the transmission frame length is still 16 times longer than the standard, the transmission rate than the TCN standard requirements to be ten times faster, and this physical response is strictly related to the BER parameter is not expressed in Link layer or system performance similar to the failure rate, and the experimental results consistent with the expected[1].

16. Summary

Because of the closedness of MVB protocol, the research and development of MVB controller in China has been the difficult problem of train communication research in China. Through the understanding of TCN international standard, this paper provides a new design idea for MVB controller analysis and design , And verified by experiment, which shows that the design method can make the MVB control equipment to meet the standard requirements, the controller can be applied to the actual train network to go, which will play a great role in China's train network business.

REFERENCE

[1] Electric Railway Equipment Train bus Part 1: Train Communication Network, International Electrotechnical Commission (IEC), IEC 61375-1 Ed. 01, 1999.
[2] American National Standards Institute (ANSI), Communication Protocol Aboard Trains, 1999. 1473-1999.
[3] Duagon corporation.MVB User Guide [J]. 2005:5-21
[4] Duagon Corporation.D013LF User Guide [J] .2011:5-13
[5] Duagon Corporation.The structure of an MVB master device without Message Data datasheet[J]2009:40-32
[6] Jimenez J , Martin J L . A top-down design for the train communication network [J]. Proceedings of International Conference on Instrial Technology, 2003.4(2):1000-1005.