Realization of High-Speed Data Transmission between Modules within Single Computer

Shangyong Yu, Gang Fu and Guangzhi Wu

China Satellite Maritime Tracking and Control Department, Jiangyin, 214431, China

Abstract. According ACE_TASK technical solutions to achieve a single internal software bus speed data transfer capabilities. First ACE_TASK framework of the research, design, message block format used to transmit data, and to implement a message queue mechanism. Then, by introducing flexible bus interface based ACE_TASK framework, elaborated software bus for message forwarding, task scheduling, data transfer and other processing principles, and finally explain the process control system program.

Keywords: Software Bus; Data transmission; Reconfigurable

1. Introduction

Software bus is the foundation platform software system, provides for the development of the upper functional modules, and integrated software framework for interoperability. Soft bus needs to solve two key technologies, multi-machine data exchange between functional modules and high-speed data transmission between the single function within the module. This paper achieves the functional software bus from the single aspect.

2. ACE_TASK framework

Based ACE_TASK soft bus modules are designed for the active object in the manner of operation, ACE initiative object inherits from ACE_Task. Here ACE_Task and active objects.

ACE_TASK ACE is the base class for active objects; all objects must be derived from active ACE_TASK class. ACE_TASK dealing with objects, so more conducive to construct OO (Oriented Object) program, resulting in better OO software, it can be said ACE_TASK like a more advanced, more object-oriented thread class. Therefore, ACE_TASK used as: higher thread (task may be called); Active Object pattern of active objects.

Active objects is relative to the traditional "passive objects" is concerned. Passive object is the traditional object, they are passive snippet, the object code segment in the calling object thread is executed, and the calling thread is temporarily "loaned" to execute the code of the passive object. The active object has its own separate thread; it can be used to perform the method of any of their calls. The image of that, the active object is the traditional passive object inside more than one or more threads. Put another way, if the main thread calls the object's method is passive, call blocking (synchronous); and if the call is a method active object calls without blocking (asynchronous).

Each active object comprises one or more threads, there is an underlying message queue, to communicate between the active object through the message queue.

3. Task communication mechanisms between modules

Data transmission task between modules is mainly done through the message queue. To improve the efficiency of data transfer between modules, packaged into a message block of data for transmission. Sent to the next task module according to message blocks task configuration, software module bus output message queue for further processing. Data for the entire system in the form of a message stream for transmission.

3.1 Message Block format

The data system is packed into a message block transfer format. Message block contains a message head and a data block.

a) Head on behalf of the message type, there are two, one represents the custom data type, and the other a control message. If the data is of a type of message block scheduling system based on the type of data to pass data to the response processing module. If the message is a control message with a block, the system responds to the control message, in which case the contents of the data block can be empty. You can customize the message header, which helps to extend system functions. Issues involved in the first part of the message and its meaning in the following table 1:

Table 1. Table header Meaning

MB_HANGUP	Data processing module
MSG_TYPE_RAW_DATA	A/D sampling data
MSG_TYPE_FD_DATA	Doppler frequency
MSG_TYPE_PLL_DATA	Ranging / telemetry subcarrier
……	……

b) The actual data is not stored in a data block, but a pointer to the actual data memory area, so that the data blocks can be shared by multiple message blocks, as shown in Figure 1.

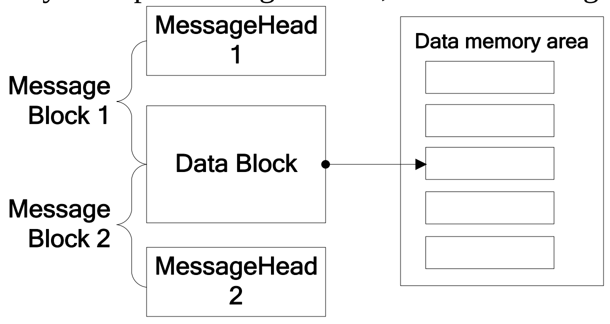

Figure 1. Message Block Format

Figure 1, the message block message block 1 and 2 points to the same data memory area, when the message block 1 is dealt with, simply re-set the message header becomes a message block 2, block 2 to process messages tasks module. Thus, the data communication between the software modules does not need to copy data, just pass the message block containing the data buffer pointer on it. This approach to data transfer by passing the pointer, which greatly improves the speed of data transmission, measurement and control system can meet the performance requirements of the A/D sampling module and other high-speed data streaming.

Message Block class provides several ways to message block operation; the main methods are as follows in Table 2:

Table 2. Operation of message function block

method	Function
msg_type（ ）	Get/set the message type
base（ ）	A pointer to the first block of data
length（ ）	Returns the total size of the data blocks of data
wr_ptr（ ）/rd_ptr（ ）	The next point to write/read data location pointer
release（ ）	Release Message Block

The above method, the read and write pointers are very important, and very prone to error. wr_ptr point to the beginning of the valid data, which is next write position data; rd_ptr point to the end of the valid data, which is next read location data, as shown in Figure 2. Two pointers require the programmer to manage and maintain, are not automatically updated. So if you want to add data to the message block, the need to ensure wr_ptr in the data header, after writing the write pointer to move to the head of valid data for next written correctly.

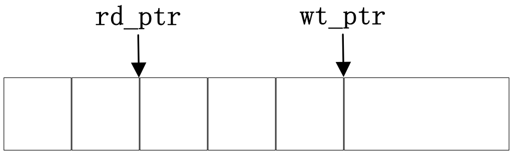

Figure 2. The read and write pointers

3.2 Message Queue implementation

Message queues are the main channels for data transmission between modules task, and also a key part of the soft bus. Each module in the system is assigned a task message queue, you can read the data maintained by the message queue, the data can be inserted into other tasks modules message queue. When the data provider module generates data, packed into the message block is inserted into the message queue consumer module, Consumers module removed from the message queue for processing. If the message queue is empty, the consumer will be blocked from going to sleep (ACE_TASK auto-complete); once the message block is inserted into the team, it will wake up the consumer, for processing.

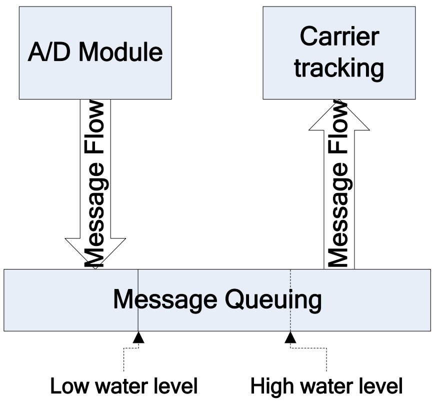

Figure 3. Message streaming schematic

Figure 3 is a schematic diagram between A/D module and carrier tracking module for message streaming through the message queue. FIG message queue is part of the carrier tracking module, A/D module will generate the message block is inserted into the message queue, module carrier tracking poll the message queue, the message blocks found will be removed after processing.

Message Queuing has the following functions and features:

a) Message Queuing decoupled producers and consumers of the message flow.

Because the data transmission between the modules through the message queue, the software modules can be handled independently, without regard to the working status of other modules, which facilitate the development of decoupling and parallel module.

b) Having a "FIFO" Features

Message queues have similar hardware characteristics "FIFO" structure follows the FIFO principle message blocks, message queue data read and write operations simultaneously in a cache. In order to achieve real-time data transmission and integrity, message queue structure, the data read speed of greater than or equal to the input speed data.

c) You can be configured flow control

Figure 3 Message Queue set two thresholds: the dotted line represents the high-water; the solid line indicates a low water level. In order to control the message queue message block traffic in the message queue designed water level to control incoming messages. High water level is used to determine the maximum capacity of the message queue, when the amount of data exceeds the high water level, the input module obstruction, stop the input data. When the capacity of the message queue data set below the low water level, the data input module can continue to enter data into the queue.

Here is the main method of message queue class:

```
    class MsgQueue:public ACE_Message_Queue
{
     MsgQueue( );
     ~MsgQueue( );
     …
     enqueue_msg(ACE_Message_Block* mb);// Insert message block
     dequeue_msg(ACE_Message_Block* mb); // Remove Message Block
     SetHighWater(size_t);// Setting message queue high water level
     SetLowWater(size_t);// Setting message queue low water level
     GetHighWater( );// Get the message queue high water level
     SetLowWater( );// Get the message queue low water level
     …
}
```

4. High speed data transmission

Under the control of the program, modules based on the soft bus interface frameworks transfer data through software bus, these modules together to complete the task of the system. System function modules independently developed according to the interface framework, individuals are independent from each other, need to control the main functional modules connected in a certain order to complete the function of the system. Function system control program is very simple, the main function modules to configure the system according to the task to be achieved, then start the function modules, the system can run. The system control process shown in Figure 4:

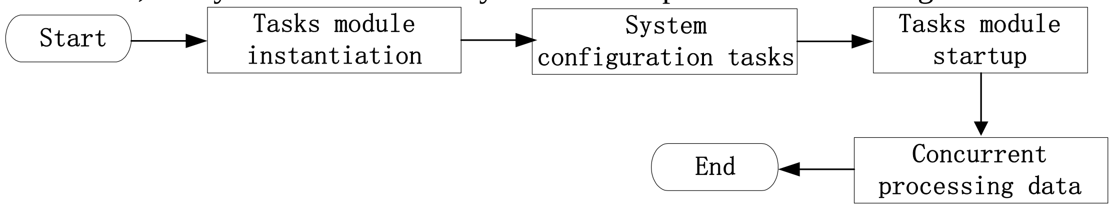
Start
Tasks module instantiation
System configuration tasks
Tasks module startup
Concurrent processing data
End

Figure 4. Process Control System

5. Summary

In this paper, high-speed data stream for the measurement and control system front-end based ACE_TASK research program, complete stand-alone high-speed data transfer software bus. First ACE_TASK framework of the research, design, message block format used to transmit data, and to implement a message queue mechanism, and finally illustrates the flow of the system control program.

References

[1] Karlv, Davis. “JTRS--An Open, Distributed-object Computing Software Radio Architecture”. IEEE 18th Digital Avionics Systems Conference, OTC, 2009.

[2] Software Communication Architecture Specification (Version 2.2) [R]. Washington: JTRS Joint Program Office, 2011.

[3] Mitchell L. Loeb, Andrew J. Rindos, William G. Holland. Gigabit Ethernet PCI Adapter Performance [J]. IEEE NetWork, March 2011: 42-47.

[4] Object Computing, Inc. TAO Developer’s Guide[R]. OCI Document, 2013.