An automated test framework for hard real-time communication systems

Nian Tang
Abstract

An automated test framework for hard real-time communication systems

Nian Tang

As there are a plenty of new real-time communication network technologies popping up today, manual testing for the performance of these technologies becomes increasing time consuming and costly. In this paper, we designed and implemented an automated test framework with software and hardware modules which can be applied to different real-time communication technologies. As the test process is fully automated, users of the test framework can focus on high level logic for testing. We also carried out a pilot case on EtherCAT system and verified the result from our test framework with manual testing. Since another important target is that it should be portable to different system, we tried to modulize out design and minimize the amount work to test other real-time networks to save human effort.
Acknowledgement

I would like to say thank you to my supervisor Morgan E Johansson for guiding my thesis study. I want to thank Mattias Rehaman for giving me support on EtherCAT stack. Also I want to say thanks to my thesis examiner Justin Pearson for his support. Thanks to my project partner Mithun Nagaiah as we had a very nice for working together. Finally, I would to like to thank everyone who offered me help during my thesis project.
# Table of contents

1 Introduction .........................................................................................................................3
   1.1 Motivation .......................................................................................................................3

2 Background ...........................................................................................................................5

3 Requirement ..........................................................................................................................6

4 Design of the test framework ..............................................................................................7
   4.1 Definition for the structure of the test framework ..........................................................7
   4.2 Architecture design ..........................................................................................................7
      4.2.1 Global synchronized clock with PTP .......................................................................8
      4.2.2 Round trip latency measurement ..........................................................................11
      4.2.3 Absolute latency measurement ............................................................................11
   4.3 Commands .....................................................................................................................14
   4.4 Types of slaves .............................................................................................................15
   4.5 Data Transfer ..............................................................................................................15
      4.5.1 Communication pattern ......................................................................................16
      4.5.2 The types of messages in the system ....................................................................17
   4.6 The I/O Controller ........................................................................................................20
   4.7 The cooperation between Test Manager and I/O controller ........................................23

5 Software implementation .....................................................................................................23
   5.1 Programming language .................................................................................................23
   5.2 Automated execution .....................................................................................................23
   5.3 Test procedure ...............................................................................................................24

6 Test case design ..................................................................................................................26
   6.1 Test case: Latency of the system ..................................................................................26
   6.2 Test case: Jitter of latency .............................................................................................27
1. Introduction

Real-time system is a system which can provide response guarantee within strict time constraints [1]. A hard real-time communication system is responsible for real-time distributed control including distributing instruction and gathering information within the system. In reality, many fieldbus technologies satisfy the requirements of the real-time communication systems. Hard real-time communication systems have very wide range of application in industry because many applications have very critical requirement on timing issue. A simple example of using real-time communication system would a system for motor control. The controller of the motor collects rolling information about the motor from a sensor via a real-time communication and then adjusts the speed of the motor by sending controlling data to actuator. So the user of the motor can control the motor in a desired way.

There is a list of hard real-time communication protocols available on the market such as CANOpen, EtherCAT, Ethernet Powerlink, PROFINET, ControlNet and etc. Different protocol provides different features. Nowadays, Ethernet based real-time communication protocol are becoming increasingly popular as they have advantages inheriting from Ethernet itself. The most obvious benefit of Ethernet-base technologies is the high-speed with relatively low-cost for the network because of the development of Ethernet.

1.1 Motivation

Even though these hard real-time communication protocols are different, a set of common features can be tested and make comparison. Also, manually testing the performance of each protocol consumes a lot of time because it will take considerable effect for developer to understand the protocol and design test cases for the protocol. As there are many test cases which could be applied to several different real-time communication protocols, finding a way to reuse these test cases can reduce the effect for repeatedly implementing them.

The aim of this project is to make an automated test framework for hard real-time communication protocols. The test framework should have high flexibility so it can be applied to several different communication protocols. Also, the test process should be automatic so the manual effect for testing can be reduced. Making a requirement for the framework is the first part of the project. The second
part will be definition of the framework and architecture design including both hardware and software modules. The final part is the implementation of the framework and pilot case study with EtherCAT.
2. Background

A real-time communication system is typically based on Master-Slave paradigm. The master is responsible for driving the network communication and normally acts as controller of the network. There are many different types of slaves such as sensors, motors, and etc and can be divided into two categories: collecting information and doing the actual work.

The real-time communication system is usually running on a degree of frequency which means the system has a cycle-time. Typically, the network master will deliver and collect various data once per cycle. The cycle-time of network is a key issue which has great influence on the performance of the system as having a smaller cycle-time means getting better accuracy of the control. Some of the advanced fieldbus technologies such as EtherCAT, PROFINET IRT can offer sub millisecond cycle-time.

Latency of a real-time communication system is a very sensitive factor. It symbolizes how fast the controlling data can be delivered from the controller to the slave devices or information could be collected from slaves.

Time synchronization is now supported by a few real-time communication systems e.g. EtherCAT, SERCOS III. With time synchronization functionality, devices inside the communication system can be synchronized to have a shared clock. The straightforward advantage of a synchronized system is that devices can perform action according to the local clock. Sub-microsecond even nanosecond is now achievable by using some real-time communication technology such as EtherCAT.
3. Requirements

After the investigation of previous documents [1, 2] of requirement of hard real time systems within ABB and several different real-time communication protocols, a number of critical requirements has been chosen to be including into the test framework. Below is the short list of the requirements which are needed to be tested in the framework:

1. Cycle-time: The cycle-time of a real-time communication system needs to be measured in order to verify that the system is actually running on the frequency which is specified by the user of the system.
2. Jitter of cycle-time: It is the difference between the maximum and minimum cycle-time measured at the system.
3. Latency of the system: It is the time taken for communication from one node to another. e.g.: Master to Slave, Slave to Slave, or Slave to Master. Here we do not measure the real delay on the bus but calculate the latency on the system level.
4. Jitter of the latency: It is the difference measured between the maximum and minimum latency.
5. Synchronization: It is the accuracy of time synchronization within the system. The different of the clocks on different devices will be checked.

To execute the measurement and ensure the quality of the testing, a number of requirements should be made on the test framework. Some of requirements are on hardware measurement device and some on the functionalities of the software.

1. Digital input and output modules: these modules can be used to listen to signals from the target system and generate signals to the target system.
2. Accuracy of latency, cycle-time, and synchronization: the accuracy of these testing items should be within 1 microsecond.
3. Automated executions: as the test framework is automated, the test script should be interpret and execute automatically.
4. Test report generation: after testing the framework, the test framework must create a test report which contains the testing results.

The full version of the requirement on the automated test framework can be found in the document [3]
4. Design of the test framework

As the goal of the test framework was to reduce the human effort for testing, the test process in the framework should be automated. The framework follows the test description and finishes the testing according to test description files. The test case script will be automatically generated by other modeling tool which is beyond the scope of this thesis project. For this test framework, the test script is the input of system and test result will be generated.

![Diagram of test framework]

Figure 1

In this chapter, a few designs of the test framework will be discussed first so one suitable approach can be selected. Then the designs for different components will be illustrated later in this chapter.

4.1. Definition for the structure of the test framework

The automated test framework has two components: a TestManager and a MasterController. Normally, the communication system under testing is independent system. Another system which stands outside with suitable measurement instrument can supervise the testing procedure and make objective test result. The MasterController implements the functions to be tested while the TestManager is responsible for creating related scenarios. In order to create testing environment, the TestManager should know which function the MasterController is executing, so the TestManager should control the behavior of the MasterController.

The TestManager is a computer with following functionalities:

1. It contains a hardware component for measuring signals (I/O controller)
2. It read the test script and creates scenarios according to the test script.

3. It has the ability to communicate with master node application for instant sending instruction and collecting data.

4. It should log the events happened within the framework.

5. It has the ability to integrate and process the data from different sources (Master node application, I/O device, Test script)

6. Test report should be generated.

The MasterController is an application installed on master of the target communication system. It is responsible for receiving instructions from the TestManager, executing them and then sending feedback to them.

4.2. Architecture design

In this section, three different kinds of hierarchy of test framework will be discussed. The first one is a structure with global synchronized clock. The second strategy measures the round trip latency while the last one measures the absolute latency.

4.2.1. Global synchronized clock with PTP

In order test latency of the communication system, a global synchronize clock within the framework can be a straight forward solution. For this approach, latency from the Master Controller to a slave can be measure easily. The structure of this approach is shown in the figure 2. For example, the Master Controller can order one digital output slave to generate a signal change so the latency needs to be measured. With a globe clock, the time when Master Controller sent out the instruction and the time I/O controller received a signal can be record separately on MasterController and TestManager. Then the latency value is difference between the timing value from MasterController and TestManager.

IEEE-1588 is time synchronization protocol which can offer sub-microsecond accuracy in computer systems. UDP protocol is used for data transmission in this protocol. This protocol can be used in the test framework as the requirement for the accuracy of test result is microsecond level.

However, several problems will be introduced to the test framework while using globe synchronized clock.
1. Additional hardware for synchronization becomes a perquisite. In order to achieve sub-microsecond accuracy, additional hardware is demanded for each component within the synchronized system. There are not so many products support IEEE 1588 on the market.

2. Hardware interfaces is also required. Another problem is that there should be a hardware interface on the MasterController such as PCI, PCI-E for the additional hardware.

3. Software issue needs to be handled carefully. OS software layers may introduce jitter and latency which will make the result less accurate because on the TestManager needs to read the I/O controller data.
4.2.2. Round trip latency measurement
The second approach for test latency is to measure the round trip time. The basic idea is to just measure the latency from the time giving an input to the target system and time moment when receiving an output on correspondent channel. What the TestManager needs to do is to start the counter when a signal is generated by the I/O controller and stop counter when it receives a signal change to correspondent slave. The structure of this solution is shown in the figure 3.

The advantage of this method is that as timings are measured on a single device by using hardware timer, the accuracy of result is very high. Finding a device with digital counter is not a big problem as digital counters are very common on the market. Also, no additional hardware is needed on the MasterController side and the functionality of each component becomes clearer by using this strategy.

The disadvantage of the approach is that the latency measured is the total time from input to output, but we cannot measure the absolute time of input to controller or controller to output.

4.2.3. Absolute latency measurement

The last design for the structure of the framework is that the signal is also measured on the master of the communication system. The strategy is based on an assumption that the Master device has the output functionality. The latency from the MasterController to an output slave can be measured in this way: Before the MasterController instructs the output slave to generate a signal change, it generates a signal on its own output port. Then the digital counter can be used to measure the duration of these two signals (from output port of MasterController and the output slave respectively).

The merit of this strategy is that the absolute latency could be tested as the signals are probed at each device itself. But at the same time, the Master of the target system needs to have an output functionality which is not quite common. The paradigm for the third approach is in Figure 4.

Finally, the second structure of the test framework is chosen to be implemented. As the target of the framework is applicable to several different communication protocols, too many requirements on the MasterController should be avoid. The first approach requires additionally hardware for synchronization and employs unpredictable software delay, and the third design needs the MasterController to have I/O functionality, so we decide to implement the second one in our automated test framework.
Figure 3
4.3 Commands

A list of common actions of the MasterController is defined and each action is called a Command. There are a number of reasons for defining these Commands. The first reason is that for testing the
performance of the target system, the MasterController needs to be able to finish a set of actions, for example, generating a signal change at a digital output slave, listening to a signal change at a digital input slave, detecting a signal change at an input node and generating a signal at an output slave.

Secondly when the TestManager is controlling the MasterController, the TestManager needs to know which action the MasterController is capable of. The last but not the least reason is that using these Commands makes the system design simpler. For each Command the correspondent function can be made in MasterController.

As the test framework aims to be applied to several different communication systems, definition these Commands could make it easier to transplant the MasterController to different protocols just by implementing these Commands on it. The TestManager will remain the same even though the system under test is different. Here is a short list of Commands and the full list Commands along with description, MasterController action and parameters can be found in the appendix B:

1. The command for initialization: Normally the target communication system has several different states. For example, EtherCAT stack has five different states. But only when the device is in Operational mode, it is ready to perform the function it has (an output slave able to generate signal). So the system under test must be initialized before doing any other Command. Cycle-time of the system should be provided to this Command normally as the system runs cyclically.

2. The command for getting the structure of the target communication system: In order to improve the usability, the Test Framework should aware of the structure of system under test. So before each Command sends to the MasterController, the Test Manager will check the device type to ensure the Command could be finished.

3. The command for generating a digital signal every cycle at an output device: For this Command, a signal change (from logic 0 to 1 or from logic 1 to 0) will be generated every cycle at the output port of a slave. The Command could be used for testing the cycle-time.

4. The command for listening to an input slave and generating a signal change the correspondent output slave: The target system listens to an input channel on a specified node. After receiving a signal change on that channel, generate a signal change to a corresponding output channel on specified output node. The Command could be used for testing the latency.

5. The command for send output signals at all output slaves with time stamping functionality: In the operation for this Command, all the output devices with timestamp functionality, the MasterController will order them to generate signal change according to a same timestamp.
One common behavior for these output Commands is that all the output ports will be set to logic 0 after finishing these Commands. In this way, the MasterController does not have to remember the status of slaves as it knows all the output ports are in the same default status.

4.4 Types of slaves

For different communication systems, there are hundreds of different types of slave devices on the market. For example, Beckhoff, one of the biggest EtherCAT device vendors, offers more than 50 different devices for EtherCAT systems. In order to represent different devices in the system, several types of device are defined in the Test Framework according to different functionalities and features. Because the Test Framework just uses basic functions of the communication system, all the devices are fall into a few categories:

1. Digital input
2. Digital output
3. Analog input
4. Analog output
5. Other types

In the future extension, many other different device types can be introduced. Other types of information about slave devices are also collected such as time stamping functionality, number of channels.

4.5 Data transfer

The TCP/IP protocol is used for data transmission between the TestManager and the MasterController. As the TCP/IP protocol is supported in most Operating Systems including Windows, Windows CE, Linux, Real-time Linux etc, it is easy to transfer data even when the TestManager and MasterController use different Operating Systems. In that sense, the test framework could be portable to different Controllers if TCP protocol is supported by the controller. Another benefit for using TCP is that delivery is guaranteed in the protocol so the program doesn’t need to take care of the data delivery and error checking issues.

The physical connection between the TestManager and the MasterController will use Ethernet. Ethernet interface is very common for normal computers, so it is reasonable to consider Ethernet as a good candidate for building the connection. Of course, other types of physical media can also be used in the Test Framework as long as TCP/IP protocol can run on it.
4.5.1 Communication pattern

As the Commands are defined in last session, a protocol needs to be introduced for data encoding and decoding and executing Commands. A common communication pattern is designed for these Commands so that all Commands share the same communication behavior. The communication starts after the TCP connection between the TestManager and the MasterController is built. The interworking between TestManager and MasterController for each Command is demonstrated in Figure 5.

For each Command, it always starts with the TestManager since the Commands are ordered by the TestManager and executed by the MasterController. Firstly, the TestManager sends a Command to the MasterController with data related to specific Command. When the MasterController receives a Command message, it should return an acknowledgement (ACK) message to the TestManager to indicate the Command has been received. Then the MasterController is executing the Command for certain amount of time. After the execution of Command, the MasterController will send a Finish (FIN) along with gathered data (depends on the Commands, could be empty) to the TestManager.
4.5.2 The types of messages in the system

The types of the datagram need to be determined so that the program will be able to write and read messages. The first part of the datagram indicates which type of message it belongs to as there are several types of message for communication. The messages are divided into five categories:

1. Connecting, disconnecting,
2. Acknowledgement (ACK),
3. Finish,
4. Error
5. Command.

The message body is empty if the message is connection message. For the error message, the message should specify the type of error. Normal communication messages such as Command, ACK and finish have message ID field to indicate communication sequence. If the message is a Command or Finish message, the next part after message ID specifies the type of Command. In the Finish message, there is field indicate whether the Command has been complete successfully or not. The rest part of the datagram relates to particular Command. The structures for different messages can be found in figure 6.

To make the program on the MasterController more scalable, the MasterController is divided into two parts: the communication part of the program which is responsible for sending and receiving messages and parsing data and the part actually implement operations for the Commands. In order to test different protocols, the functions for commands need to be realized. The communication part will be identical for all the testing protocols.
### Figure 6

#### Message Types

<table>
<thead>
<tr>
<th>Message type</th>
<th>Message ID</th>
<th>Command type</th>
<th>Data related to Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command Message</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finish Message</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACK Message</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connect or Disconnect Message</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Error Message</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 4.6 The I/O Controller
The I/O Controller is the device which measures important data in the Test Framework. It is responsible for measure the latency, cycle-time of the communication system. The I/O device should have high resolution hardware digital counter in able to measure these latency values. In order to achieve sub-microsecond accuracy, the digital counter should have more than 1MHz frequency. Another functionality the I/O Controller needs to have is capable to generate and acquire signal with voltage range from 0V to 48V.

National Instrument PCI-6624 Digital Counter will be used as the I/O Controller of the Test Framework. This device has 8 hardware counter channels and each channel has 20MHz frequency internal counter, so the requirement on accuracy of the test result can be satisfied.

NI Data Acquisition (DAQ) provides driver for NI PCI-6624. It also contains a library with a number of basic functions such as pulse generation, edge detection, frequency and two edges separation. These basic functions can be invoked from the TestManager to execute the testing. For instance, pulse generation and two edges separation can be used for measuring the round trip latency; cycle-time will be measured by pulse width function. The detailed testing procedures will be fined in later chapters.

A few different operations are defined for the I/O controller to finish. The first operation is to generate a signal at one channel and start counting on another channel. The counting channel will be stopped until it gets a signal channel. In this case one channel is used to generate signal change to the outside system and one channel which is doing two edges separation acquires the duration between the time when the signal sends out the device and the time when the counter receives another signal.

As the device should be able to listen to and generate signals, there is one operation for generate a pulse at specific channel and listen to signal on particular channel.

The last operation for I/O controller to accomplishment is listening to all channels. For this operation, actually the last channel is not listening but generating a pulse to all other channels which are all executing two edges separation. The pulse generated by the last channel will be considered as the first signal of the edge separation. Then the counter on each listening channel will start counting after receiving the signal and wait for the second signal. In this way, the timing difference of several input signals can be tested. This operation can be applied to test internal network latency. But generating a signal on all channels is not needed since it is impossible to generate signals from all channels simultaneously. The only way to do it is to just use one channel to create the signal and connect it to several devices.
4.7 The cooperation between the TestManager and the I/O Controller

In the TestManager, there are two functions which need to be running at the same time, one for controlling the MasterController, another for operating the I/O Controller. These two functions should work together to execute the test cases as one selects the Commands on the Master Controller and the other one creating measurement environment. A mechanism therefore needs to be introduced to synchronize these two tasks. With the help of synchronization, it is possible to make combination of different Commands and I/O controller operations.

There are some operations on the I/O controller which are needed to be setup before the Commands is executed by the MasterController. For instance, if the TestManager orders the MasterController to generate a digital output signal, the correspondent channel on the I/O controller should be set to listening status before the MasterController executes the output Command. The opposite situation could also happen in some scenarios such as when the TestManager instructs the MasterController to listen to a signal change on one input slave, the listener has to be established on the MasterController before the I/O controller performs any action.

The task for controlling I/O controller is running separately. Most of the time, this task is waiting for the event from the TestManager. To execute a specific operation on the I/O controller, the TestManager firstly needs to select the operation on the I/O controller. Then the TestManager signals the event which I/O controller is waiting for. After whether the operation is started immediately or still needs to wait for the second event is highly related to the operation itself. In most of the cases, if the operation is listening, the I/O controller should start to work immediately. When generating a signal on the I/O controller is the task, it should wait for the second event to do the operation because at that time the related setup on the MasterController has been done already. However, sometimes one of the channels on the I/O controller is used to trigger the counting of other channels. In this case output could be done before receiving the second event from the TestManager.

After the TestManager receives the finish message from the MasterController, the TestManager could read data from the I/O controller since finish messages symbolize the operation has been completed by the MasterController. Otherwise there must be some error occurred in the Test Framework. Then the I/O controller is waiting for another event to start next operation.

Of course, if the TestManager doesn’t use any operation of the I/O controller, the command can be still be executed by the Test Framework for other purpose of work. In figure 7, we can see the interworking between TestManager and I/OController.
Figure 7

TestManager

Setup operation for

Signal the event

Send Command

Signal the event
after receiving Ack
message from I/O

Waiting for

event

event

I/O controller

Waiting for

event

Setup operation

Initialize the device for
operation. Maybe start
operation immediately
or start operation on
receiving the second

Waiting for
5. Software implementation

5.1 Programming language

Choosing a suitable programming language to implement the basic programming part of the Test Framework could be a problem. As we decided to use TCP/IP protocol as the communication protocol between the TestManager and the MasterController and TCP has an implementation named “Winsock” on Windows platform in C language, the language must be able to call these APIs directly. The APIs for the I/O controller from National Instrument are in the C language as well. C language seems to be good candidate. However, in order to make the Test Framework in a more Object-Oriented way, C++ is the final choice for the program since it also can use C APIs directly and the EtherCAT master stack developed by ABB is implemented by C++.

5.2 Automated execution

As the testing process is automated, the Test Framework should be able to read test scripts and execute the testing procedure according to the test script.

One of the ways for achieving automated execution is to design a scripting language which is later used to describe the testing process and develop an interpreter to decode the test script and execute it on the real program.

Another approach to implement automated execution is using exist scripting language and exposing the real C++ program to the scripting language. Then the C++ functions can be invoked directly from the scripting language.

The first approach could cost a lot of effect for developing a new language and implementing parser for it. The goal of the project is to design a Test Framework, rather than invent a new scripting language for testing. As the result, the second method for using an existing scripting language is chosen as it will reduce the difficulties of developing the Test Framework.

There are a number of different scripting languages available which can be bound with C++ program such as Python, Perl, Lua, TCL, and Ruby. Most of them support Object-Oriented programming features directly or indirectly. Python, Perl and Ruby are native Object-Oriented scripting languages while TCL and Lua realize Object-Oriented in other way.

Python is chosen as the scripting language for the framework due to the following two reasons:
• Development of python is much quicker compared to other scripting languages mentioned above.
• Python has a clear syntax compared to other scripting languages.

SWIG is tool which can automatically generate scripting language interfaces from an interface specification. It takes ANSI C/C++ declarations and turns into scripting language interfaces. SWIG supports many different scripting languages such as Python, Perl, Lua, TCL and etc. However, SWIG cannot interface complex data type in C++. It needs to write Python C-API code for type conversions when creating interface. What is more is that it has more overheads when generating interface file.

Boost.Python is dedicated for binding the C++ libraries with python. It need to write simple function to describe functions and/or class you want to export to Python program and Python can read the library which is generated by boost tool as a python module. The Advantage is that it can deal with complicated typed conversion and good for binding existed C++ libraries.

Eventually due to the reasons above, the Boost.Python will used for binding out C++ program with Python.

5.3 Test procedure

In the Test Framework, hardware connections such as Ethernet connection between TestManager and MasterController, electrical circus should be setup appropriately. After that, there is some procedure below which needs to be followed:

1. TestManager starts and initializes the NI counter device and allocate related resource.
2. The MasterController begins to run and waits for the TestManager to connect to it.
3. With the IP address of the MasterController, the TestManager builds up the communication connection between the TestManager and the MasterController.
4. The TestManager should send Command Initialization to the MasterController so the target system can enter the operational state.
5. The TestManager should detect the structure of the system under test so it can check the type of device before sending out the Command to the MasterController.
6. The Test Framework is ready to execute testing such as latency, cycle-time.
7. The TestManager could stop the target communication system. Then the TestManager can send Initialization Command to the MasterController with different cycle-time values and the test flow should go to step 4. Otherwise the whole test process is over.
Start TestManager

Start MasterController

TM connect to MC

TM sends Command Init

TM gets the structure of MC

Close the Stack

Execute testing

Figure 8
6. Test case design
6.1 Test case: latency of the system

Prerequisite for the system under test: one digital input slave node and one digital output slave node and two PCI-6624 counter channels.

Test description: The round trip strategy is used for testing latency of the target system. When the input slave receives a signal change from PCI-6624 counter, the corresponding output device will generate a signal change. The command Listen2InputSend2OutputD is used to build up logic on the MasterController which is a connection between the input slave and the output slave on Master Controller. The command Listen2InputSend2OutputD will setup a listener on Master Controller which listens signal change on the specified input device and then trigger a signal change on a particular output slave.

The input channel on the input slave, the input slave address, the output channel on the output slave and the output slave address should be provided to command. Two counters channels are needed to finish the network latency test. The PCI-6624 counter on Test Manager will use one counter channel to generate a pulse to the target system and trigger the second counter to start counting. When the second counter receives a signal from the output slave, it stops counting and the counter value is the round trip latency of the target system.

Every time the latency is calculated, it will be compared with requirement value of external network latency if it is provided. The figure 9 is the interwork of this test case.
6.2 Test case: Jitter for latency.

Prerequisite for the system under test: one digital input slave node and one digital output slave node and two PCI-6624 counter channels.

Test description: The above method for calculating latency of the system will be used to test jitter of external network latency. The latency values are computed ten times and the difference in the maximum and the minimum value of these latency values is the jitter of external network latency between the given input and output slave node.

6.3 Test case: Startup time

Prerequisite for the system under test: cycle-time of Master Controller.

Test description: Test Manager needs to send the initialization command to Master Controller. When the Test Manager receives the ACK message from the Master Controller it starts the software timer and when the Test Manager receives the finish message, it will stop the timer.
Figure 10 Test case for start-up time

6.4 Test Case: cycle-time

Prerequisite for the system under test: One digital output slave node and a PCI-6624 counter.

Test Description:

The test framework should have the ability to test the difference in the cycle-time of target system and the cycle-time from the specifications.

The MasterController should instruct an output slave generating a signal change every cycle by using the command CMDSendOutputCycle. The output channel of the output slave, the output slave address and the duration for running the command should be provided to command. The PCI-6624 counter on Test Manager will calculate the pulse width for the signal. The pulse width is the cycle-time of the target system as the signal will keep logic 1 for one cycle and change to logic 0 at the next cycle. The cycle-time is taken and compared with the specification of the communication system under test to get the duration of the cycle-time.
Figure 11 Test case for cycle-time

6.5 Jitter of cycle-time:

Prerequisite for the system under test: One digital output slave node and a PCI-6624 Counter.

Test Description:

The test manager should generate signal change every cycle by using the command CMDSendOutputCycle. The output channel of the output slave, the output slave address and the duration for running the command should be provided to command. The process for testing the cycle-time will repeat 10 times and the cycle-time values will be gathered. The difference between the maximum and minimum of the cycle-times is taken system under test to get the jitter of the cycle-time.
Figure 12 Test case for jitter of cycle-time

6.6 Synchronization

Prerequisite for the system under test: two digital output slave nodes with time stamping ability, PCI-6624 counter.

Test description:

The test framework should have the ability to test the time synchronization accuracy target systems. The TestManager should first initialize the I/O Controller by setting all the listening channels to task Two Edges Separating. Then one output channel will generate a pulse to trigger all listening channels started to counting. In this way, all the counters are guaranteed to be start at the same time.

After that the TestManager sends the Command SendAllITSOutputD to the MasterController. For the Command SendAllITSOutputD, digital output slave nodes with time stamp will generate signals according to a same specified time stamp. When the counter channel receives a signal from the system under test, the counter will stop counting. The difference of the counter values is the difference of the local clocks on different output slave. The accuracy of time synchronization is tested by applying above procedure.
Figure 13
Figure 14 Test case for synchronization
7. Pilot test case on EtherCAT

7.1 Briefly introduction of EtherCAT

EtherCAT is hard real-time communication technology based on master and slave architecture. The communication part is using Ethernet but the datagram in EtherCAT system is modified in order to apply EtherCAT algorithm. The master device is industrial PC with Windows CE real time operating system. The slave modules are from Beckhoff including EL1252 (2 channel digital input terminal with Time-Stamp), EL2202 (2 channel digital output terminal). EtherCAT master software application is developed and used by ABB. The master software provides access to read data from the slaves and write data to the slaves [3]. The slave nodes time synchronization is also implemented on the master software.

EtherCAT slaves have five different states: Init, Pre-Operational (Pre-Op), Safe-Operational (Safe-Op), Operational (Op) and Boot. [5] Initially, after the device is power-on, all the slaves will be in the state of Init. Then the state of the whole system should be set to Pre-Op. After that the state of the system should be transferred from Pre-Op to Safe-Op. Finally, the system must enter Op state so the whole system is functional. In this process, the cycle-time of the system is set and other initialization operations are finished. Basically, these operations above have to be completed for the Command Initialization.
Process Data Objects (PDOs) originally defined in CANopen (Controller Area Network) are used in EtherCAT systems. PDOs which contain control and status data are highly related to the functions on the slaves [6]. For example, a PDO referring to a port on a digital output slave can be used to control the status of the port, whether it is in low or high state.

The EtherCAT master software provides two methods to get access to PDOs on the EtherCAT system: read and write.

For reading PDOs, Process Data Listener which runs every cycle can be used to monitor PDO. At first, the Data Listener must be created. Then the PDOs needed to be listened to should be subscribed to the Data Listener. Then the PDOs could be checked every cycle as the Data Listener executes cyclically.

To write the PDOs for controlling data, the master software offers a post method for PDOs. The PDOs which are posted by the master stack will be sent to slaves.
7.2 Implementation of Commands

This part of the report illustrates how the Commands are implemented by using the EtherCAT master software. The implementations for Command SetupInputListenerD and SendOutput2NodeD are quite straightforward by applying the methods giving by master software.

To realize the Command Listen2InputSend2OutputD, a Data Listener has to be established. The Data Listener is separate task other than the main task for handling the Commands. The PDO corresponding to the specified port on the input slave should be subscribed to the Data Listener. After subscribing the PDO to the Data Listener, the main task will wait for an event from the Data Listener which indicates the target signal has been received. As the Data Listener runs cyclically, the changes on the input channels will be detected at the next cycle. As soon as the Data Listener recognizes the change of PDO, the Data Listen will signal the event which the main task is waiting for. The main task will post the PDO for the related output port as soon as possible.
For the Command SendOutputCycle, the same approach of Command Listen2InputSend2OutputD is applied to realize this Command. After the Data Listener is constructed, no PDO will be assigned to this listener. So instead of listening to any PDO, the Data Listener is just used to signal the event for the main MasterController task every cycle. The main task will continuously wait for synchronization event. Every time when the main task receives the event, the main task will change the PDO for the digital output slave which is different from previous one.
7.3 Test Result and comparison with Oscilloscope

So far, the Test Framework is ready to experiment on a real time communication system—EtherCAT as the MasterController application based on is implemented. In this chapter, the Test Framework will conduct several test cases on EtherCAT system including latency and cycle-time. In order to verify
the quality of test result of the Test Framework, Oscilloscope will also be used to measure signals in
different test cases and comparison of the test results from the oscilloscope and the Test Framework
should be made to check the correctness and accuracy of the Test Framework.

For the EtherCAT system under testing, the cycle-time of the bus is set to 900 microseconds.

7.3.1. Test of cycle-time

The test framework is setup as described in test case for testing the cycle-time. One of the counter
channels is connected with a digital output slave. As the Command SendOutputCycle is sent to the
Test Manager, the digital output port will change its state every cycle. The NI counter will measure
the pulse width of the output port. In the output port, an oscilloscope probe will be also used to trace
the signal changes.

![Figure 18](image)

As it is showed in the picture, the pulse width of the signal is roughly 896 us with means the EtheCAT
system is actually on the frequency as it is supposed to be. The cycle-time reads out from the Test
Manager is 906000ns. Even though the results from the oscilloscope and the Test Manager are not exactly the same, this reason might be that it is very difficult for human to get the exact value from the oscilloscope. We prefer the value from the NI counter more accurate since the measurement of this device is running on 20MHz frequency clock.

### 7.3.2. Test of latency

The test case for latency is applied for measuring the latency of the EtherCAT system. In this test case, one digital output slave EL2202 and one digital input slave EL1252 will be used. The Master Controller will setup a Data Listener for the input slave and once the signal change is detect by the Data Listener, the EtherCAT master software will sent the PDO for output signal to EL2202. Two counter channels on the NI counter board will be used, as one generates the signal to EL1252 and starts the second counter and the other measures the latency value by counting the duration between the first signal to EL1252 and second signal from EL2202. Two channels on the oscilloscope corresponds two signals in this test scenario: input and output.

![Figure 19](image-url)
The latency from oscilloscope is 2.55 ms meanwhile the value from the TestManager is 2.565 ms. The latency value is larger than expected. The delay from input slave to output slave should be less than 2 cycle-time (1.8 ms in this case) as in the first cycle the signal change is captured by the Master Controller and the output signal will be generated at the second cycle. The reason is because of the natural of EtherCAT. In EtherCAT system, every cycle the master of the network sends out an EtherCAT packet and receives it for sending controlling instruction and collect data for slaves in the system.

1. The master receives the signal change on the input slave when the packet for the first cycle comes back to the master.
2. The master needs to wait for the next cycle where the packet for changing the output signal could be sending out.

However, the latency is significantly larger that the value we predicted. Even with different cycle-time values of the system, the latencies are always longer than 2 cycles. Why this situation is happened is unknown to us.
As we have seen so far, the test result from the test framework is accurate when the oscilloscope is used as a comparison for it. As the time limitation for this project, the accuracy of time synchronization hasn’t been carried out.
8. Conclusion

In this report, we have described the full design of the automated test framework for hard real-time systems including software and hardware modules to realize it. The test framework has the possibility to be extended to other Master-Slave based real-time communication systems since the requirements on the test framework are gathered based on several different real-time communication protocols. However, in order to test a specific system, extra implementation for Commands on the MasterController application is needed.

Abstractions for the operations in each parts of the framework are made to modularize the design our framework. A list of Commands has been made to define the common behaviors of the communication systems. These Commands have the same communication scheme between the TestManager and the MasterController. If later more Commands need to be added to the test framework, the communication pattern should be followed. The operations for the I/O controller are determined as well.

Pilot test case on EtherCAT master software has been carried out to verify the correctness of the test framework. As we have seen in chapter 6, critical characters such as cycle-time, latency of the EtherCAT system were successfully measured.

Additionally, the test framework also fulfilled the requirement on automatic execution for the test scripts by interfacing the functions for testing to Python. The test cases will be automatically generated by ComformIQ in Python programming language and then executed through Python parse.
9. Future work

The test framework is able to test basic requirements of a real-time communication system, but there are still more related work needs to be done. Since in the current implementation only I/O digital device are used for measuring, other types of devices such as analog I/O devices could be included in the framework.

One of the drawbacks of this framework is that the absolute latency is not calculated as the result of using the round trip approach. However, if the controller of the target system has the output functionality, then our framework could easy adapted to measure the absolute delay from the master to a slave or from a slave to the master device. Slightly change the behavior of the Command SetupInputListenD and SendOutput2NodeD can make the framework support measure the absolute latency value. For the Command SetupInputListenD, when the input listener detected the PDO change, the master device should generate an output signal on its output interface. Considering Command SendOutput2NodeD, the Master Controller should generate an output signal on its output interface before instructing the correspondent slave to generate a signal. Then the I/O controller is able to measure the latency with two edges separation function.

Payload is another sensitive issue in the real-time systems. But measuring the payload in the real-time systems is extremely difficult as it is highly related to the specific protocol underlying. In this test framework, we have not manager to conduct test regarding to payload and bandwidth issue because relating the payload with the performance of the real-time system could be very complicate. In the future, new methods can be developed to test this issue.
10. Appendix

**Design for command in the framework.**

(TM: Test Manager)

1. Initialization.

<table>
<thead>
<tr>
<th>Description</th>
<th>Command the Master Node to finish initialization process (e.g. put all devices in Operational mode).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>cycle-time</td>
</tr>
<tr>
<td>Feedback</td>
<td>Ack, Finish</td>
</tr>
<tr>
<td>TM action</td>
<td>When this command is sent out, the TM starts a timer. When received a Finish, stops the timer.</td>
</tr>
</tbody>
</table>

2. GetStructure.

<table>
<thead>
<tr>
<th>Description</th>
<th>Get the structure of our target communication system.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td></td>
</tr>
<tr>
<td>Feedback</td>
<td>Ack, Finish and Number of slaves in the target system, the functionality for each slave (digital input or output device, with time stamp or not, number of I/O channels).</td>
</tr>
<tr>
<td>TM action</td>
<td>Compare the structure specified in the test script</td>
</tr>
</tbody>
</table>

3. SetupInputListenerD.

<table>
<thead>
<tr>
<th>Description</th>
<th>Setup a digital signal listener on the channel of a specified input node and delete the listener after fixed duration.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>The input node ID and the channel number it wants to listen to. Duration of the listener.</td>
</tr>
<tr>
<td>Feedback</td>
<td>Ack. Finish and timestamp if the input device has the time stamping functionality.</td>
</tr>
</tbody>
</table>
4. **SendOutput2NodeD.**

<table>
<thead>
<tr>
<th>Description</th>
<th>Generate a digital signal on a channel at the specified output node and reset the signal state to original state after a fixed duration.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>The output node ID and the channel number on which it wants to generate a signal change and the duration to change it back.</td>
</tr>
<tr>
<td>Feedback</td>
<td>Ack. Finish and timestamp for Master node controller.</td>
</tr>
</tbody>
</table>

5. **SetupAllInputListenerD.**

<table>
<thead>
<tr>
<th>Description</th>
<th>Setup listener on all general input channels for signal changes and delete these listeners after fixed duration.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>Duration of the listeners</td>
</tr>
<tr>
<td>Feedback</td>
<td>Ack. Finish and for the inputs whether they has received signal changes or not.</td>
</tr>
</tbody>
</table>

6. **SendAllOutputD.**

<table>
<thead>
<tr>
<th>Description</th>
<th>Generate signal change on all output general channels on all output nodes and reset the signal states to the original states after a fixed duration. If the target system contains digital outputs with timestamp function, send out signals at the same time according to the time stamp.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>Duration of Outputs.</td>
</tr>
<tr>
<td>Feedback</td>
<td>Ack. Finish.</td>
</tr>
</tbody>
</table>

| TM action   | Read a signal change from the slave node using I/O controller.                                                 |
7. **SendAllTSOutputD**

<table>
<thead>
<tr>
<th>Description</th>
<th>As the target system contains digital outputs with timestamp function, instruct the nodes to send out signals at the same time according to the time stamp.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>Duration of Outputs.</td>
</tr>
<tr>
<td>Feedback</td>
<td>Ack. Finish.</td>
</tr>
<tr>
<td>TM action</td>
<td>Read signal changes from the slave node on I/O controller.</td>
</tr>
</tbody>
</table>

8. **Listen2InputSend2OutputD**

<table>
<thead>
<tr>
<th>Description</th>
<th>The target system listens to an input channel on a specified node. After receiving a signal change on that channel, generate a signal change to a corresponding output channel on specified output node.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>The port number of input device, input device ID, The port number of output device and the output device ID.</td>
</tr>
<tr>
<td>Feedback</td>
<td>Ack. Finish.</td>
</tr>
<tr>
<td>TM action</td>
<td>Generate the signal and start the timer on I/O controller. Stop the timer when the I/O controller gets a signal from the target system.</td>
</tr>
</tbody>
</table>

9. **GroupOutputD**

<table>
<thead>
<tr>
<th>Description</th>
<th>Group a number of channels on different digital output node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>Number of node you want to group and a list of node IDs and a channel on each node.</td>
</tr>
<tr>
<td>Feedback</td>
<td>Ack. Finish. Group ID.</td>
</tr>
<tr>
<td>TM action</td>
<td></td>
</tr>
</tbody>
</table>

*As the test framework mostly focuses on timing, the command will only take one channel of each device to the group.*
10. **GroupInputD.**

<table>
<thead>
<tr>
<th>Description</th>
<th>Group a number of channels on different digital input node.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>Number of node you want to group and a list of node IDs and channel(s) on each node.</td>
</tr>
<tr>
<td>Feedback</td>
<td>Ack. Finish. Group ID</td>
</tr>
<tr>
<td>TM action</td>
<td></td>
</tr>
</tbody>
</table>

*As the test framework mostly focuses on timing, the command will only take one channel of each device to the group.*

11. **SendOutputGroupD.**

<table>
<thead>
<tr>
<th>Description</th>
<th>Generate digital signals on channels at the specified output nodes and reset the signal states to original states after a fixed duration on all the output nodes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>The output node IDs and the channel number it wants to generate signal change and the duration to change these signal states back.</td>
</tr>
<tr>
<td>Feedback</td>
<td>Ack. Finish.</td>
</tr>
<tr>
<td>TM action</td>
<td>Read signal changes from the slave nodes on I/O controller.</td>
</tr>
</tbody>
</table>

12. **Listen2InputGroupD.**

<table>
<thead>
<tr>
<th>Description</th>
<th>Bind digital signal listeners on the channels of specified input nodes and unbind these listeners after fixed duration.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>The input node IDs and the channel numbers it wants to listen to. Duration of binding of these listeners.</td>
</tr>
<tr>
<td>Feedback</td>
<td>Ack. Finish and timestamps if the input devices have the time stamping functionality</td>
</tr>
<tr>
<td>TM action</td>
<td>Generate a signal change from I/O controller to the slave node</td>
</tr>
</tbody>
</table>


13. **DismissGroup.**

<table>
<thead>
<tr>
<th>Description</th>
<th>Dismiss the group which is created by GroupOutputD or GroupInputD and free the correspondent resource.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>The group ID.</td>
</tr>
<tr>
<td>Feedback</td>
<td>Ack. Finish</td>
</tr>
<tr>
<td>TM action</td>
<td></td>
</tr>
</tbody>
</table>

14. **ConnectInputD2GOutputD**

<table>
<thead>
<tr>
<th>Description</th>
<th>The target system listens to an input channel. When receiving a signal from the channel, generate a signal change to a corresponding output group which is created by GroupInputD.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>The channel number of input device, input device ID, the port number of output group ID.</td>
</tr>
<tr>
<td>Feedback</td>
<td>Ack. Finish. If the input and output devices have time stamp function, the result should also contain these timestamps.</td>
</tr>
<tr>
<td>TM action</td>
<td>Generate the signal and start the timers on I/O controller. Stop the timer when the I/O controller gets a signal from one channel of the target system.</td>
</tr>
</tbody>
</table>

15. **SendOutputCycle**

<table>
<thead>
<tr>
<th>Description</th>
<th>Send out a signal change every cycle.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>The port number of output device and the output device ID. Duration of sending out the signal changes.</td>
</tr>
<tr>
<td>Feedback</td>
<td>Ack. Finish.</td>
</tr>
</tbody>
</table>
### TM action
- Start a timer when get the signal change. Stop the timer when the I/O controller gets a signal from the target system read the timer value. Then start the timer and repeat the process again.

### CloseUp

<table>
<thead>
<tr>
<th>Description</th>
<th>Command the Master Node to enter init mode (e.g. put all devices in Init mode).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td></td>
</tr>
<tr>
<td>Feedback</td>
<td>Ack. Finish</td>
</tr>
<tr>
<td>TM action</td>
<td></td>
</tr>
</tbody>
</table>
11. Reference

[1] 9ADB000865-019 CGA Communication Technology Evaluation

[2] 9ADB000865-022 Communication selection criteria vs. candidate communication Protocols


