A Study on Verification System to Support Efficient Interface Test in Embedded System

Tai-Gil Kwon, Jin-Woong Cho
VR/AR Research Center
Korea Electronics Technology Institute
Seoul, Korea
tgkwon@keti.re.kr, chojw@keti.re.kr

Abstract**—** In recent years, embedded systems such as various electronic devices equipped with a microprocessor, which acts as a brain of electronic devices, have been spreading due to the improvement of computational capability of system semiconductors and the generalization of large-capacity memory. However, as the requirements of the application using the embedded system increase, the number of the peripheral devices and the interfaces built in the embedded system increases. As a result, it takes much effort and time to develop the embedded system and verify the function. Therefore, in this paper, we propose and study a verification system that can efficiently test the interface to the embedded system.

Keywords—Verification; Interface; Embedded System

I. Introduction

An embedded system[1] is a system that enables effective control by designing a microprocessor that acts as a brain of an electronic device. The software on which the device operates is not stored on a disk like a computer, but is built into the device in a chip. These embedded systems have been explosively increasing in demand in various fields such as industrial devices, communication devices, and home appliances over time, and system requirements are becoming increasingly complex. So many of today's commercial SoC (System on Chip) chipsets support various kinds of standard interfaces to connect CPU core and many peripherals, and have a separate operating system for high quality software development like PC. SoC technology in the embedded field has reduced the complexity of hardware design much more than in the past. However, as the complexity increases exponentially in software, the number of peripheral devices and interfaces that need to be functionally tested or debugged is increasing. much effort and time are required to develop an embedded system.

Therefore, in this paper, we propose and study a verification system that provides an efficient interface test method when developing an embedded system or performing functional verification.

II. Verification System Structure

A verification system for testing an embedded system is composed of devices to be verified, a verification module for performing actual verification, a server for providing an interconnection between a user and a verification module, and a user for verifying the device at a remote location. Verification modules can be physically connected to multiple devices with standard interfaces (usb, ethernet, i2c, spi, can, etc.) and servers can be connected to users and verification modules via the Internet. The verification module performs a verification on an interface interconnected with the device, and the server simultaneously performs a web server for providing a web environment to the user and a web application function for controlling the verification module. Therefore, if the user wants to control the verification module by accessing the web server, the web application converts the control command that can be used in the verification module and transmits the control command to the verification module through TCP transmission. The verification module can perform the interface verification connected to the device according to the control command. After the verification is completed, the verification result is transmitted to the web application running on the server through the TCP transmission. Then, the web server recognizes the result and converts it into a result that can be seen in the web environment, and notifies the user of the final result.

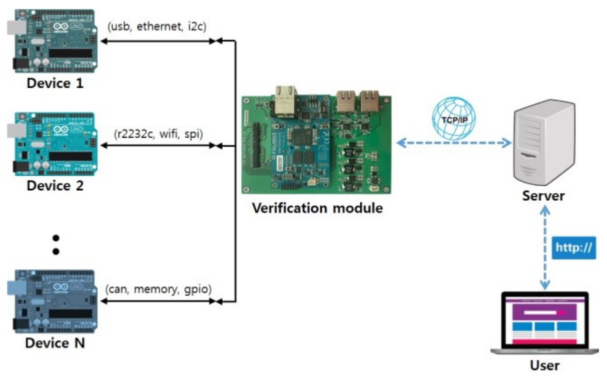

Fig. 1. Verification system diagram

III. Verification Module Software Architecture

The software of the verification module is largely composed of application programming interface (API) for controlling the verification module in the server and verification software running in the verification module hardware. In order to efficiently provide large capacity Web services, the server is designed based on a micro service architecture[2] in which one large application is divided into service units and these services are communicated with each other. Therefore, there may exist various services in the server, and one of them is included in the micro service architecture that is responsible for interface verification.

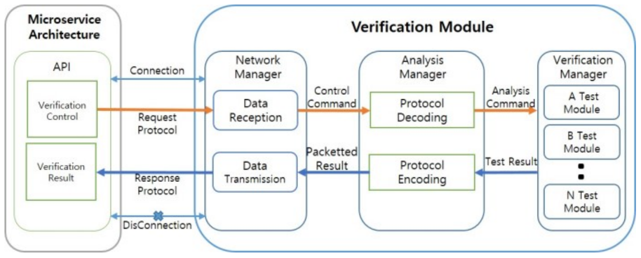

Fig. 2. Verification module software architecture

A. Transmission of verification command

In the micro-service architecture, the TCP network protocol is used for reliable data transmission / reception for network interoperability with the verification module. The microarchitecture can remotely control the verification of the interface to the device connected to the verification module and the physical interface by using the verification API.

TABLE I. VERIFICATION API

Name	Description
GpioReq(Direction, Port, Value)	GPIO Verification
EthernetReq(Target IP, Time, Data, Option)	Ethernet Verification
WifiReq(Target IP, Time, Data, Option)	WiFi Verification
Rs232DataReq(Port, Direction, Size, Data)	Rs232 Verification

When the verification API is executed, the verification command is transmitted as a request packet of the form of a request protocol to the verification module. The Request Packet consists of a header consisting of Length and ReqID and a payload with JSON[3] data format

TABLE II. REQUEST PACKET

2 Octect	1 Octect	Variable
Length	ReqID	Payload (JSON Format)

The JSON format used for payload is structured by collecting and organizing data, so all data is represented by key and value. Key and value are represented using a string enclosed in double quotation marks ("") Use a comma (,) to separate key and value. The table Ⅲ is an example of a JSON-format payload for the request packet

TABLE III. JSON FORMAT

```
{“Direction” : 0,
“Port” : 21,
“Value” : null}
```

B. Reception of verification Result

When a packet is received in the micro-service architecture, the packet is stacked in the buffer in the incoming order for buffering processing. The length of the packet header is compared with the length of the buffered data. If the length of the buffered data is less than the length of the packet header, the reception of the next packet is awaited. Otherwise, Separates the response packet from the buffer and performs parsing on the response packet to obtain the payload including the interface ID, the length of the payload, and the result data. When the parsing is completed, the verification result is converted into an event type result API, transferred to the micro-service architecture, and waits for reception of the next packet.

TABLE IV. RESULT API

Name	Description
GpioResp(Direction, Port, Value)	GPIO Result
EthernetResp(Target IP, Time, Data, Option)	Ethernet Result
WifiResp(Target IP, Time, Data, Option)	WiFi Result
Rs232DataResp(Port, Direction, Size, Data)	Rs232 Result
I2cResp(Port, Value)	I2C Result
SpiResp(Port, Value)	SPI Result

C. Analysis and Verification

The request packet received from the micro-service architecture is buffered in the network manager and then sent back to the analysis manager. The analysis manager obtains the payload including the interface ID, the payload length, and the verification command data through parsing of the received request packet. And then, parsing the JSON data for the payload is performed again to obtain additional data for the verification command. The parsing of JSON data distinguishes between a string and a number by checking whether the first character of the payload is {, followed by a double quotation mark or a number. The parameter data of the verification command is obtained from the payload by storing the values resulting from the iterative operation on the string or number in the token array. It then passes the analysis data to the test module within verification manager that matches the ReqID value and performs the interface verification. When the test module completes the interface verification, response packets are generated according to response protocol format to send the verification results to the micro-service architecture.

TABLE V. RESPONSE PACKET

2 Octect	1 Octect	Variable
Length	RespID	Payload (JSON Format)

In the response packet, Length indicates the length of the payload, not the total protocol length. RespID is an identifier for identifying the response packet and has the same value as ReqID of the request packet. The payload represents the verification result data, and the data conversion process must be performed in the JSON data format again. The process of converting the verification result data into the JSON format is similar to the reverse generation of the j JSON data parsing process. Generate a buffer to store the JSON data, and store the verification result data in the buffer according to the JSON syntax to generate JSON data. the generated response packet is delivered to the network manager, and then finally transmitted to the microarchitecture.

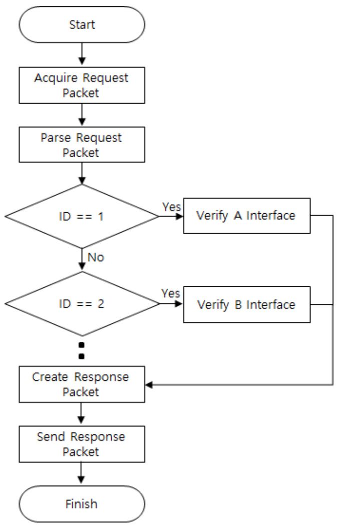

Fig. 3. Verification procedure according to control command

IV. EXPERIMENT AND RESULT

A verification process is required that the verification module software architecture operates normally. First, we test the interface physically connected to the verification module by controlling the verification module in a separate application operating in the PC environment. As a result, it was confirmed that the verification of the interface works normally according to the verification procedure as shown in the figure 4. And then the verification result of the JSON data format could be obtained.

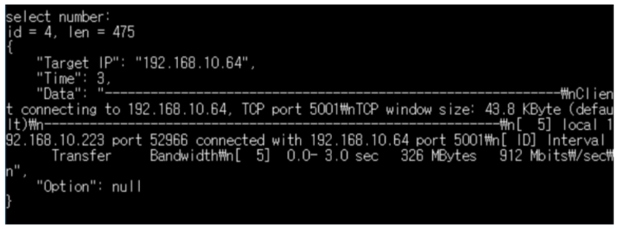
```
select number:
id = 4, len = 475
{
    "Target IP": "192.168.10.64",
    "Time": 3,
    "Data": "-----------------------------------------------------------₩nClien
t connecting to 192.168.10.64, TCP port 5001₩nTCP window size: 43.8 KByte (defau
lt)₩n-----------------------------------------------------------₩n[  5] local 1
92.168.10.223 port 52966 connected with 192.168.10.64 port 5001₩n[ ID] Interval
      Transfer     Bandwidth₩n[  5]  0.0- 3.0 sec   326 MBytes   912 Mbits₩/sec₩
n",
    "Option": null
}
```

Fig. 4. Interface verification results in the application.

After the normal operation test is successfully completed in the application environment, the user web browser accesses the server based on the micro service architecture and then tests the interface connected to the verification module. As a result, it was confirmed that the verification procedure was normally performed in the web environment as shown in the figure 5. And then, the verification result of the web data format was obtained.

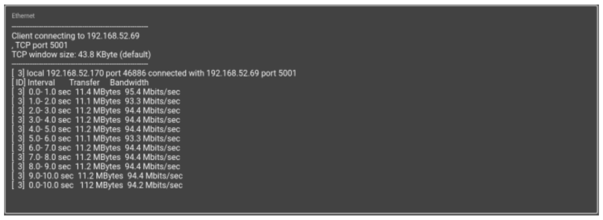
```
Ethernet
------------------------------------------------------------
Client connecting to 192.168.52.69
, TCP port 5001
TCP window size: 43.8 KByte (default)
------------------------------------------------------------
[  3] local 192.168.52.170 port 46886 connected with 192.168.52.69 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec  11.4 MBytes  95.4 Mbits/sec
[  3]  1.0- 2.0 sec  11.1 MBytes  93.3 Mbits/sec
[  3]  2.0- 3.0 sec  11.2 MBytes  94.4 Mbits/sec
[  3]  3.0- 4.0 sec  11.2 MBytes  94.4 Mbits/sec
[  3]  4.0- 5.0 sec  11.2 MBytes  94.4 Mbits/sec
[  3]  5.0- 6.0 sec  11.1 MBytes  93.3 Mbits/sec
[  3]  6.0- 7.0 sec  11.2 MBytes  94.4 Mbits/sec
[  3]  7.0- 8.0 sec  11.2 MBytes  94.4 Mbits/sec
[  3]  8.0- 9.0 sec  11.2 MBytes  94.4 Mbits/sec
[  3]  9.0-10.0 sec  11.2 MBytes  94.4 Mbits/sec
[  3]  0.0-10.0 sec   112 MBytes  94.2 Mbits/sec
```

Fig. 5. Interface verification results in web environment.

V. CONCLUSION

In the field of embedded systems, the number of peripherals and interfaces that developers need to test is increasing and there is a problem of longer development timer. In this paper, we propose and implement a verification system that can test the interface more effectively than the existing methods by identifying problems in the development site. In the future, field tests should be carried out in various embedded system environments, and it is necessary to identify the problems and requirements that arise in the field and to improve the function of verification system based on them.

ACKNOWLEDGMENT

This work was supported by the Korea Evaluation Institute of Industrial Technology (KEIT) grant funded by the Korea government Ministry of Trade and Industry [No. 10085632].

REFERENCES

[1] https://en.wikipedia.org/wiki/Embedded_system

[2] J.W. Kim and Y.H. Kim, "A Study on the Microservices Architectrue in the Cloud Environment", In the proceeding of the Korean Institute of Communications and Information Sciences. 2016, pp. 268-269

[3] K.M.Nguyen and H.T. Nguyen, "A Novel Approach for Accessing Semantic Data by Translating RESTful/JSON Commands into SPARQL Messages", IEIE Transactions on Smart Processing & Computing Vol.5 No.3, 2016, pp. 222-229