Turn antiquated peek and poke interfaces in embedded module to modern web APIs.

Brijendra Singh
CommScope

***Abstract*—In Embedded system, peek and poke are the fundamental command sets available in command line interface to read or write memory address or hardware registers. During the early stage of development and debugging embedded board designs, developers may need to read (peek) values from hardware and write (poke) values into its registers or memory address before developing the driver codes [1]. Most of the times these command sets are redesigned due to hardware change.**
This paper describes the design to turn traditional peek and poke command sets interface to generic web APIs which are built upon existing well proven technologies and network protocols like HTTP and RESTful. This paper not only describe the overall architecture and flow, but also defines the methods that extends the peek and poke capability to work with other on-board peripherals. Moving to modern web APIs design would open the prospect to use cloud computing power to peek and poke directly at hardware level of complex embedded devices deployed in the field.

***Index Terms*- embedded system, command line interface, Hyper Text Transfer Protocol (HTTP), REpresentational State Transfer (REST), Application Programming Interface.**

I. INTRODUCTION

Designing and development of an embedded systems are bringing more challenges to the software and hardware team due to increased complexity. Hardware and software are getting more sophisticated, wide-ranging and enhanced network connectivity. Therefore, developing software features are not only the key success of the product, also needs a solid debugging and user interfacing capability. Software can have Graphical User Interface (GUI) or simple to full featured Command Line Interface (CLI) tool to allow interaction with the system [2]. Some CLI implementations allow hardware level access. This will help to "peek" run time state of the hardware and on-board peripherals using "register read" or configure using "register write" set of operations. This hardware register level debugging support is not only critical during the development, it could also be utilized as a backdoor method of debugging when system is released in the field.

Conversely, for years accessing these methods are tied with the specific implementation of embedded software. These peek and poke methods can provide great insight of the system's run time state by dumping hardware registers details or can tweak state by changing register values. During development phase, physically attached debugger tools like In-circuit emulator (ICE) or JTAG provides various debugging support. The support includes hardware registers view window that shows and allows modifying the content of registers, lists microcontroller operation modes, system and internal states [3]. In released product, software provides simple to full featured Command Line Interface (CLI) with hardware level peek and poke options. Mostly peek and poke are implemented to access direct memory addresses or mapped hardware registers space. Though, this may not work directly with on-board peripherals connected with commonly used buses like SPI, and I2C.

Mostly these peek and poke commands are accessed via local serial console or remote virtual terminal such as telnet. Traditionally these commands interfaces and access methods that are designed specific to the software and hardware designs. As embedded designs vary widely, CLI associated commands and method to access may require redesigning due to the change of environment or to expand the debugging support for newly added on-board peripherals [4]. Therefore, designing a generic and scalable framework to perform peek and poke operations is necessary. This designing can address following areas like:

1. Adaptability.
2. Scalability.

3. Common interface to access all on-board peripherals.
4. Platform independent.

With the advancement of the technology and hardware designs, embedded systems are becoming complex, powerful and full-blown computing device. This device can run full featured operating system like embedded Linux, which includes networking software, web server, various run time programming environments and provide various hardware interfacing features. These networking software building blocks, allows to transform standard peek and poke interface to web enabled interface as shown in the Figure 1:

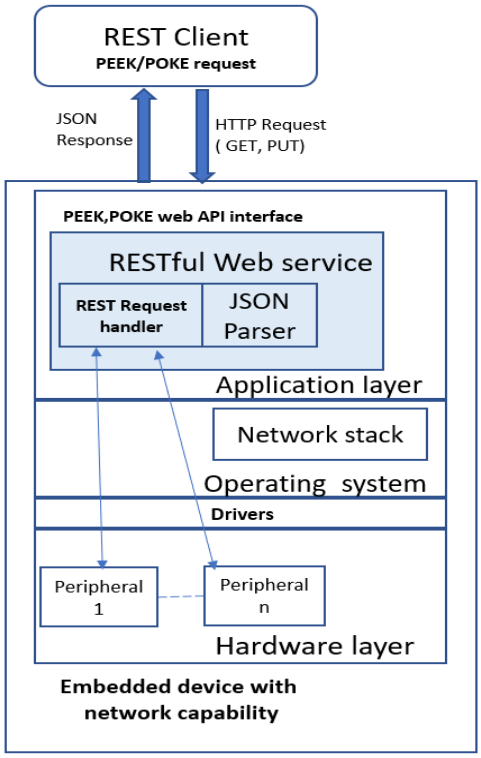

Figure 1. Peek and poke interface as web APIs

This proposed peek and poke web API interface design is implemented as a RESTful web services. REST architecture is the best fit to implement peek and poke interface due to its stateless and uniform interface guiding principles [5]. HTTP methods decides the peek or poke operation, and finally REST handler carry out read or write operation with the help of available device drivers in the system.

II. DESIGN

A. Define resources:
There are two resources required for read or write operations:
• Device (e.g. SRAM, Parallel flash, FPGA, SPI, I2C)
• Address (memory or register)

B. Define endpoints:
This will refer to locate specific device and register resource using Uniform Resource Identifiers [6]. Resource in REST are always manipulated through the URI:

/{device}
/{device}/{id}
/{device}/{id}/{address}
/{device}/{id}/{address}/{value}

C. Define uniform interface (HTTP Methods):
To carry out peek and poke operations using standard HTTP methods GET, POST and PUT will be used on the resources, identified by URI.

- To peek (read) memory address or device register, we are going to use GET or POST method as below:

Example of "Peek" web method:
Read single memory location or register -
GET http://<device-url>/{device}/{id}/{address}

Read multiple resources -
POST http://<device-url>/{device}/{id}/{address}/{value}
[here {value} will represent how many consecutive address or registers required to read]

- To poke (write) memory address or device register, we are going to use PUT method. It is idempotent method.

Example of "Poke" web method:
PUT http://<device-url>/{device}/{id}/{address}/{value}

D. Define data representation
For request and response data format, we are going to use Java Script Object Notation (JSON) data format to keep implementation lightweight.

Figure 2. Sample response JSON data for memory Peek

E. Define status/error codes

We are going to re-use standard HTTP response status codes to validate successful execution of peek and poke operation over web API.

III. IMPLEMENTATION AND TEST MEHOD OF PEEK AND POKE WEB APIS

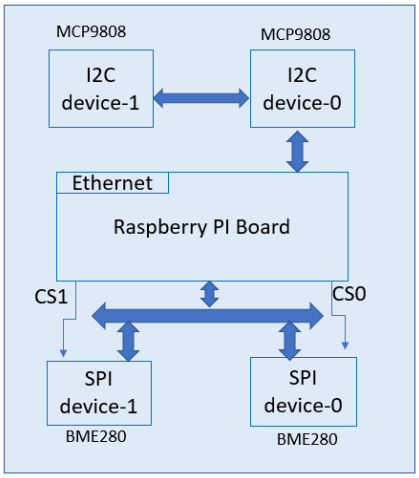

Figure 3. Test Board with multiple on-board devices

For testing purpose of the proposed design, a single board computing module Raspberry PI board has been used. This board is programmed with latest available Linux kernel and connected with multiple external peripherals. To build REST web server Flask micro web framework written in Python has been used. GET, POST and PUT handler are implemented to use underlaying various available drivers like I2C and SPI to interface with on-board peripherals. Implemented peek and poke web APIs interface has been tested using "curl" tool which is installed locally on the test board.

- TEST 1 - Peek device ID register (address=0xD0) of SPI-1 device(cs1)

curl --request GET http://localhost:8080/spi/1/208

RESULT - Response JSON data:
{"deviceName":"spi","deviceID":"1","registers":[{"address":"0xD0","value":"0x60"}]}

- TEST 2 - Peek device ID register(address=0x7) of I2C-0 device(address=25)

curl --request GET http://localhost:8080/i2c/25/7

RESULT - Response JSON data:
{"deviceName":"i2c","deviceID":"1","registers":[{"address":"0x19","value":"0x04"}]}

- TEST 3 - Poke (write value=1) in config register(address=0xF5) on SPI-0 device(cs0)

curl --request PUT http://localhost:8080/spi/0/245/1

RESULT – HTTP 200 OK

IV. CONCLUSION

Peek and poke tools are like Swiss Army knife for software and hardware engineers while debugging embedded systems. Either in the lab environment during development or remotely released device, these tools are very powerful to provide insight of the complex system. Accessing and using these tools can be improved in many ways including one mentioned in this paper. Using web APIs approach, we can have advantage of uniformly accessible, adaptable and scalable debugging feature in variety of complex embedded systems out in the field.

REFERENCES

[1] "Read and write register values of your device before a driver is available." 30 July 2001. Agilent Technologies. http://literature.cdn.keysight.com/litweb/pdf/5988-3310EN.pdf. 17 June 2020.

[2] Scott, Dale. "Writing Command Line Interfaces for Embedded Devices." paguilar.org n.d.: 2. https://paguilar.org/?page_id=17. 17 June 2020.

[3] Staff, Embedded. "The ten secrets of embedded debugging." embedded 15 September 2004: 2. https://www.embedded.com/the-ten-secrets-of-embedded-debugging/.

[4] X. Guo-ping and T. Dan, "The Design of a Versatile CLI that Supports Command Set Dynamically Loading," 2011 Fourth International Symposium on Computational Intelligence and Design, Hangzhou, 2011, pp. 91-94, doi: 10.1109/ISCID.2011.124.
[5] Fielding, Roy T. REST APIs must be hypertext-driven. 20 Oct 2008. https://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven. 17 June 2020.
[6] restfulapi.net. n.d. https://restfulapi.net/rest-api-design-tutorial-with-example/#object-model.