OTA Firmware Update for Texas Instruments C2000 Controllers

Hemalatha Eedi, Hema Avireni

Abstract*: *Firmware updates are necessary to enhance the embedded systems performance and to remove the bugs. But it's not an efficient manner to be installed it by the technician in the field. For c2000 controllers of Texas instruments dynamic memory allocation is not possible. So, using the proposed method we can maintain both updated firmware and current running firmware for the continuity of the system operation without dynamic memory allocation requirement. In this over the air (OTA) firmware update method, both application and the secondary boot loader (SBL) handles the update process without intervention of the technician. When there is any update the current running application copies the incoming updated firmware using its linker file. After the update process when there is a system reset the SBL calculates the checksum of the updated application, if it matches with the received checksum it executes the updated firmware else it executes the current running firmware without interrupting the application operation.

Keywords*: *About OTA; firmware update; secondary boot loader;

I. INTRODUCTION

Many embedded systems are deployed in places that are very difficult and impractical for a human operator to access. This is true for Internet of Things (IoT) applications, which are typically deployed in larger quantities which has limited battery life [1]. Some controllers don't support dynamic memory allocation. To improve the existing application performance and to fix the bugs, updates are necessary for any system which uses c2000 family of controllers. With the help of the OTA firmware update method, we can achieve this without any physical intervention [2],[3].

In some methods the updated firmware overwrites the previous firmware. If the received firmware is faulty there may be discontinuity of the running of the system [4]. Using the method proposed in this paper we can overcome it by maintaining both the updated firmware and current running firmware and can improve the system performance.

Using any GSM (Global System for Mobile communication) module we can establish communication between the server and the system and can receive the required updated application [5],[6].

The paper is structured as follows: Section II illustrates review of related work. Information about system design and its basic communication provides in Section III. Section IV describes the implementation of OTA firmware update using SBL and application and section V concludes the paper.

Revised Manuscript Received on November 06, 2019.

Hemalatha Eedi, Department of Computer Science and Engineering, JNTUH College of Engineering Hyderabad, Hyderabad, India. Email: hemamorarjee@jntuh.ac.in

Hema Avireni, Department of Electronics and Communications Engineering, JNTUH College of Engineering Hyderabad, Hyderabad, India. Email: hema.avireni@gmail.com

II. RELATED WORK

In this section related work on secure firmware upgrade and its integrity verification is briefly explained

In the paper prosed [2] by Andrew Cottrell, San Jose, CA (US); Jithendra Bethur, Newark, CA (US); Timothy J. Markey, San Jose, CA (US); M. Srikant, San Jose, CA (US); Lakshmanan Srinivasan, San Jose, CA (US) aims for the method of secure firmware update. In "Secure Firmware Update" the controller receives the updated or corrected firmware code. The received firmware and source are authenticated. If they are successfully authenticated the current firmware is replaced by updated firmware. If either of the sourcec or updated firmware is not authorized, the memory of the current firmware remains locked.

Samip Dhakal, Fehmi Jaafar and Pavol Zavarsky [1] used the concepts of block- chain technology and delta update. "Private Block chain Network for IoT Device Firmware Integrity Verification and Update" studies the viability of merging the two technologies for firmware updates on resource constrained IoT devices, for example, Wi-Fi smart plugs and sensors. They developed a private block chain network-based IoT device for firmware integrity verification and update mechanism. The proposed solution focuses to enhance the performance of firmware update. Many researchers proposed Block-chain based secure methods for firmware updation in an embedded and IoT devices [7], [8], [9], [10].

III. SYSTEM DESIGN AND COMMUNICATION

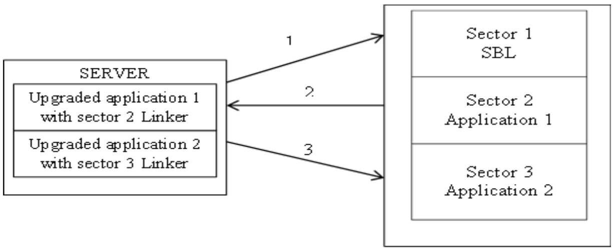

Fig. 1. System Design

For main system design, we use server and the controller memory, as will be explained as follows For this method to achieve the SBL and current running application always shares two parameters update flag which notifies whether there is an update, entry address of current running application.

A. Server

When there is any update the server generates two files i.e., same application with two linker files, one with sector 2 memory address and other with sector 3 memory address and sends it in particular frame format [2]. The data can be sent using any GSM module

B. Controller memory

The c2000 controller available flash memory is divided into three sectors secondary boot loader memory, application_1 memory and application_2 memory [3]. The application_1 memory and application_2 memory holds the current running firmware and updated firmware which can be interchanged after update process.

The system design is depicted in the Fig. 1 and the steps for communication are explained as follows.

1. The server sends a command to know which application is running. The command can be created as of developer wish.
2. The application responds by sending the command A1 (application 1) or A2 (application 2) using current running application entry address. If it receives A1 the server sends the upgraded application 2 or if it receives A2 the server sends upgraded application 1.
3. The current running application receives the updated firmware and copies it in the controller memory as per the linker file and makes the update flag high.

After every reset the secondary boot-loader checks for any update, and if there is any it calculates the checksum of the received firmware if it doesn't match with the received checksum, it jumps to the entry address of the current running firmware and starts execution. If it matches with the checksum it changes the entry address of current running application with updated firmware entry address and makes the update flag low [11][12][13].

IV. IMPLEMENTATION OF OTA FIRMWARE UPDATE

A. Implementation of OTA firmware upgrade in application as shown in Fig.2.

1. After any software reset the execution begins at secondary boot-loader.
2. Primarily it checks for current running image. The secondary boot-loader and the code share a common register (current running image register) which holds the current running image, image 1 or image 2 and upgrade flag to notify that there has been a firmware upgrade.
3. If there is no image running (the controller will be in idle state and indefinitely waits else it checks whether the upgrade flag is high.
4. If the upgrade flash is not high the current running image will be executed else it verifies the checksum of the received code.
5. If the calculated checksum matches the received checksum, the received code is valid else the code has errors.
6. If the checksum is not matched, the pointer jumps to current running image entry point & starts execution.

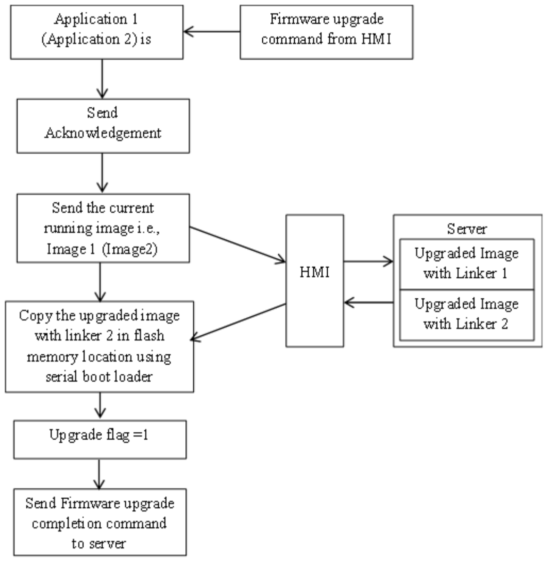

Fig. 2. Flow chart for implementation of OTA firmware upgrade in application

7. If the checksum is matched it updates the current running image and the pointer jumps to updated current running image entry point and starts execution.
8. In this way the secondary boot-loader will be implemented for Over the Air firmware upgrade.

Data Format of c2000 controller application:

For serial flashing the data format of the application is as follows

Byte Contents
1 LSB: AA (KeyValue for memory width = 8-bits)
2 MSB: 08h (KeyValue for memory width = 8-bits)
3 LSB: bytes 3-18 reserved for future use
4 MSB: bytes 3-18 reserved for future use
... ...
17 LSB: bytes 3-18 reserved for future use
18 MSB: bytes 3-18 reserved for future use
19 LSB: Upper half (MSW) of Entry point PC [23:16]
20 MSB: Upper half (MSW) of Entry point PC [31:24] (Note: Always 0x00)
21 LSB: Lower half (LSW) of Entry point PC [7:0]
22 MSB: Lower half (LSW) of Entry point PC [15:8]
23 LSB: Block size [7:0] (number of 16-bit words) of the first block of data to load
24 MSB: Block size [15:8]
25 LSB: Block load starting address [23:16]
26 LSB: Block load starting address [31:24]
27 LSB: Block load starting address [7:0]
28 LSB: Block load starting address [15:8]
29 LSB: First data word in the block
30 MSB: First data word in the block
... ...
n MSB: Last data word in block
n+1 LSB: Block size [7:0] of the next block of data
[Same structure as the first block]
x LSB: Block size [7:0] of 0x0000 indicates the end of the load

x+1 MSB: Block size [15:8] of 0x0000 indicates the end of the load

note: 08AA is the key value for serial flash programming for c2000 controllers

Pseudo code:

```
If firmware upgrade command is received
    Send acknowledgement;
    Send current running image number;
Receive upgraded code();
Upgrade flag = 1;
Send Firmware upgrade completion command to server;
Receive upgraded code()
{
If (key!= 0x08AA)
Return;
Read reserved bytes;
Read the entry address of upgraded application;
If application 1 is the current running application
Erase flash sectors of Block 3;
Else
Erase flash sectors of Block 2;
Read the block size;
While ( block size!= 0x0000)
{
Read the block load starting address;
Program the received block data using flash API functions;
Read the next block size;
}
}
Jump to starting address of current running application and starts executing;
```

B. Implementation of secondary bootloader for OTA firmware upgrade as shown in Fig.3.

1. After any software reset the execution begins at secondary boot-loader.
2. Primarily it checks for current running image. The secondary boot-loader and the code share a common register (current running image register) which holds the current running image, image 1 or image 2 and upgrade flag to notify that there has been a firmware upgrade.
3. If there is no image running (the controller will be in idle state and indefinitely waits else it checks whether the upgrade flag is high.
4. If the upgrade flash is not high the current running image will be executed else it verifies the checksum of the received code.

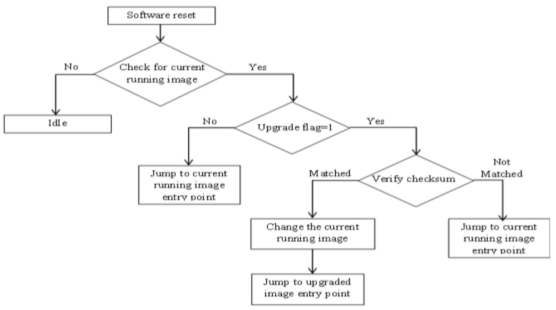

Fig. 3. Flow chart for implementation of secondary bootloader for OTA firmware upgrade

5. If the calculated checksum matches the received checksum, the received code is valid else the code has errors.
6. If the checksum is not matched, the pointer jumps to current running image entry point and starts execution.
7. If the checksum is matched it updates the current running image and the pointer jumps to updated current running image entry point and starts execution.
8. In this way the secondary boot-loader will be implemented for Over the Air firmware upgrade.

Pseudo code:

```
When power on/ any software reset occurs
Primary bootloader is executed as default;
Jump to SecBootEntAdr; // SecBootEntAdr- secondary bootloader entry address
SecBootEntAdr:
If (current running application register == 0x0000)
Idle;
Else
{
If (upgrade flag=1)
{
If (Received upgraded application checksum == calculated checksum)
{
Update the current running application register;
Jump to upgraded application entry address and start executing;
}
Else
Jump to current running application entry address and start executing;
}
Else
Jump to current running application entry address and start executing;
}
```

you are using *Word*, use either the Microsoft Equation Editor or the *MathType* add-on (http://www.mathtype.com) for equations in your paper (Insert | Object | Create New | Microsoft Equation *or* MathType Equation). “Float over text” should *not* be selected.

V. CONCLUSION

As firmware updates are necessary to enhance the embedded systems performance and to remove the bugs, the method proposed successfully flashed the upgraded firmware without over-writing the current running firmware. In this update process is done by using both update related code in application and secondary bootloader. This method also overcomes the traditional problem which if the received firmware is faulty, there may be discontinuity in the system operation by maintain the both currently operating and upgraded firmware.

REFERENCES

1. Samip Dhakal, Fehmi Jaafar and Pavol Zavarsky- “Private Block chain Network for IoT Device Firmware Integrity Verification and Update”: 2019 IEEE 19th International Symposium on High Assurance Systems Engineering (HASE).
2. Andrew Cottrell, Jithendra Bethur, Timothy J. Markey, M. Srikant, Lakshmanan Srinivasan, “Secure Firmware Update”: United State Patent Application Publication, 2006.
3. Peter L. Skan, Harpenden (GB) , “Method For Over-The-Air Firmware Update Of NAND Flash Memory Based Mobile Devices”: United States Patent, 2010.
4. Shao-Chun Chen, Aliso Viejo, CA (US); James P. Gustafson, Irvine, CA (US)- “Firmware Update Network And Process Employing Preprocessing Techniques”: United States Patent, 2012.
5. Hans Chandra; Erwin Anggadjaja ; Pranata Setya Wijaya ; Edy Gunawan- “Internet of Things: Over-the-Air (OTA) firmware update in Lightweight mesh network protocol for smart urban development”: The 22nd Asia-Pacific Conference on Communications (APCC2016).
6. Goran Jurkovic ; Vlado Sruk- “Remote firmware update for constrained embedded systems”: 37th International Convention of Information and Communication Technology, Electronics and Microelectronics (MIPRO), 1019- 1023, 2014.
7. M. Banerje et al., "A Blockchain future to Internet of Things security", Digital Communications and Networks,2017.
8. N.K. Nainar, C.M. Pignataro, R. Asati, "Block Chain Based IoT Device Identity Verification and Anomaly Detection" U.S. Patent Application No. 15/098,518, 2016.
9. Dorri et al., "Blockchain: A distributed solution to automotive security and privacy", *IEEE Communications Magazine*, vol.55, Dec. 2017.
10. Dorri, S.S. Kanhere, R. Jurdak, "Blockchain in internet of things: challenges and solutions", 2016. Available: https://arxiv.org/pdf/1608.05187.
11. Choi, Byung-Chul, Seoung-Hyeon Lee, Jung-Chan Na, and Jong-Hyouk Lee. "Secure firmware validation and update for consumer devices in home networking." *IEEE Transactions on Consumer Electronics* 62, no. 1 (2016): 39-44.
12. Tools.ietf.org. A Firmware Update Architecture for Internet of Things Devices,2018.Available:https://tools.ietf.org/id/draft-moran-suitarchitecture-02.html
13. Dhakal, Samip, Fehmi Jaafar, and Pavol Zavarsky. "Private Blockchain Network for IoT Device Firmware Integrity Verification and Update." In *2019 IEEE 19th International Symposium on High Assurance Systems Engineering (HASE)*, pp. 164-170. IEEE, 2019.

AUTHORS PROFILE

Ms. Hemalatha Eedi, Working As Assistant Professor In The Department Of Computer Science And Engineering, Jawaharlal Nehru Technological University Hyderabad College Of Engineering Hyderabad Since 2006. Her Areas Of Interest Include Parallel And Distributed Computing, Cloud Computing, Iot, And Machine Intelligence.

Ms. Hema Avireni Is Pursuing M.Tech. (Embedded Systems) In The Department Of ECE And CSE. She Has Participated Few Workshops, Technical Seminars In The Area Of Iot, Machine Learning And Embedded Systems.