Koc. J. Sci. Eng., 2(1): (2019) 01-06 https://doi.org/10.34088/kojose.494655

Kocaeli University

Kocaeli Journal of Science and Engineering

http://dergipark.org.tr/kojose

# CAN Bus Based Firmware Update System for Distributed Embedded Systems Consisting of ARM Cortex-M0 Series Microcontrollers

Ali Batuhan KINDAN [1,*], Selçuk KİZİR [2]

[1] *Department of Mechatronics Engineering, Kocaeli University, Kocaeli, 41310, Turkey,* ***Orcid Id:*** *0000-0001-6242-7067*
[2] *Department of Mechatronics Engineering, Kocaeli University, Kocaeli, 41310, Turkey,* ***Orcid Id:*** *0000-0002-0582-5904*

**Article Info**

**Research paper**

*Received : December 10, 2018*
*Accepted : January 28, 2019*

**Keywords**

*ARM Cortex M0*
*Bootloader*
*CAN Bus*
*Distributed Embedded Systems*
*Firmware Update*

## Abstract

In this study, design of a firmware update system for distributed embedded systems consisting of nodes with ARM Cortex M0 microcontrollers is presented. The nodes in the system are connected to each other over high speed CAN Bus 2.0A protocol and the system is controlled by a PC via a graphical user interface. The system allows user to update the Firmware of system nodes through the CAN Bus network. Complete firmware update system for distributed embedded systems with CAN Bus field network is provided by the system designed in this study. With the custom bootloader developed during this study, in application firmware update over CAN Bus feature is integrated to 32 bit STM32F072 microcontroller.

## 1. Introduction

While designing and developing embedded systems, the firmware is modified several times. As a result of the changes in the development process, the system is used in the field after obtaining a stable version providing the designed features. During the tests in system development, the system is usually in a test environment and low number of microcontrollers used in the tests can be programmed individually with special programmers. Modifying the software is relatively easy because of low numbers during the tests. Even if stable software is obtained that meets the requirements of the system after the test phase, there are a lot of feedbacks when the system starts to be used in the field, and in this direction, it is necessary to update the firmware for the various bug fixes that are not noticed during the development of the software. In addition, various updates to improve the system efficiency and to give new features to the existing system are among the reasons that require changes to the software. Unlike the development and testing phases, it is not possible to modify the software with the methods used in the testing phase after the system is used by the end users in the field. The reason for this is that after the stabilized version is obtained, it is not commercially possible to program a plurality of systems distributed to end users as commercial products, or large number of microcontrollers forming a single system. In current commercial systems, the software update needs of almost all kinds of applications are met by bootloader software.

Bootloader software is an embedded software that starts from the fixed reset address of the microcontroller and it is designed to be as simple as possible in order not to make any changes on it in the future [1].

When the supply voltage is provided to the system or the system is reset, the bootloader application runs first and, if there is no update request depending on the design of the application, it jumps to the Firmware application which is located at a different memory address (Figure 1) and designed to meet the user's need. When the firmware update is requested, the firmware resets itself via an external command and switches to the bootloader [2]. The bootloader deletes the flash memory sectors of the existing

* Corresponding Author: batuhan.kindan@gmail.com

Firmware and writes the program data of the new Firmware to the firmware address area from a predefined source, depending on the system design.

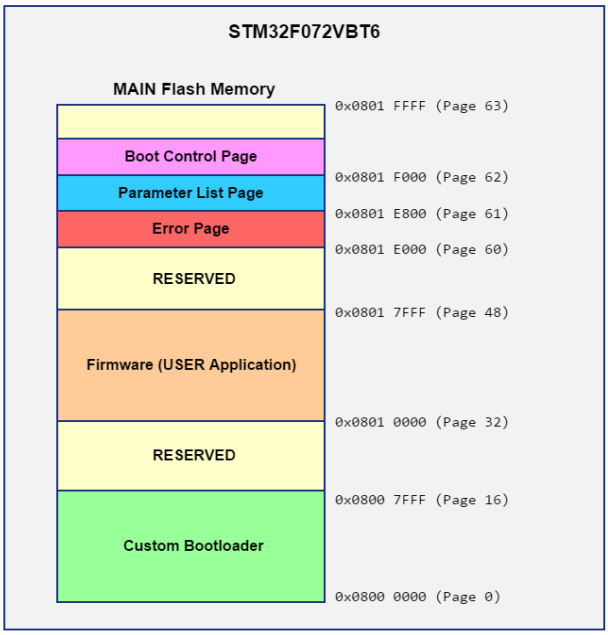


**Figure 1.** Memory map of one of the network elements in the system that shows the memory sectors of the bootloader and firmware.

After the bootloader writes all of the new firmware data to the corresponding address zone, the software will be used again by jumping to the firmware.

In this study, high-speed CAN Bus 2.0A protocol is used as the bootloader communication protocol. This communication protocol allows communication with 110 network elements at a maximum speed of 1Mbit / s over a total of 6.5 km [3]. The main reasons for using this protocol are as follows [4]:

1) The CAN bus protocol has a differential communication hardware so that it can operate smoothly in industrial environments with electrical noise.

2) Even if any network element connected to the CAN Bus network stops working, the network can continue to communicate without being affected.

3) The CAN bus protocol can operate as a Multi-Master. Each network element can have priority over the network when necessary.

4) Hardware prevention of the mixing of the similar priority messages that are sent by the network elements at the same time (Bitwise Arbitration).

5) The CAN Bus protocol has internal CRC (Cyclic Redundancy Check) in the message frame.

In addition to these, the reasons for choosing microcontrollers with ARM Cortex M0 architecture for the implementation of bootloader in the system are; reasonable price-performance values, 32-bit bus width, NVIC (Nested Vectored Interrupt Controller) hardware that allows flexible prioritization of the system interrupts [5], user friendly software development environments (KEIL, IAR, ATTOLIC vs.), easy findable software libraries for peripherals and some of the models in M0 series that have internal CAN Bus transceiver.

Similar to the designed system in this study, there are some bootloader designs for automobile ECU's (Engine Control Unit) developed on the 16 bit bus width NXP's Freescale MC9S12DP256 microcontrollers and that are using CAN Bus protocol [6]. ECU systems are one of the functional elements on the automobile which the software update is performed as a result of performance and efficiency improvements after the cars are produced and started to be used by end user. ECU software updates can be done through the communication line without removing the ECU component from the vehicle thanks to the developed systems. Another system developed for software updates of ECU components in the automotive industry is the software update system which incrementally updates the software by taking into consideration the differences between the old and the new program, rather than simply deleting or updating all program data for software updates [7]. In addition, similar to the designed system, there is also a system that updates a large number of ECU components on the CAN Bus communication line with a computer [8]. Except that, in addition to similar systems used in the automotive industry, there are also different bootloader and full-scale software update systems designed based on CAN communication protocol. An example of these systems is the software update system that enables the update of the embedded software of the DSP chips used in robotic systems to perform various control operations via the CAN communication line [9].

In this study, custom bootloader is designed to achieve software update feature for 32 bit bus width STMicroelectronics STM32F072 microcontroller. Also, it is aimed that the designed software update system can be used as a diagnostics mechanism by providing access to the desired parameters of the nodes of the distributed embedded system via the CAN Bus interface. The system is controlled by a computer via an interface device between the computer and the distributed embedded system communication network.

In Chapter 2, hardware information about the network elements used in the system, the user interface that allows the user to manage the system, the communication routines used in the system and the operating principle of the system are described. In Chapter 3, the observations of the system are explained and the contribution of the system is presented.

## 2. System Overview

The system consists of 3 types of components:

1) A PC which runs a graphical user interface (GUI) that allows the user to select which software to update the HEX file of the software that belongs to desired node to be updated.

2) A CAN Bus interface device which allows PC to communicate network nodes and the network elements.

3) Network nodes consisting of STM32F072 microcontrollers.

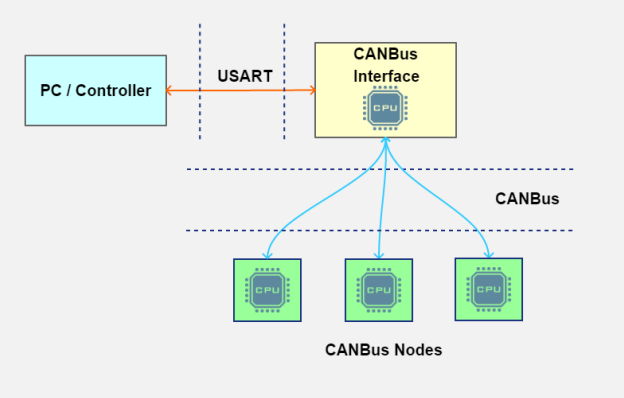


**Figure 2.** System communication scheme.

The PC and CAN Bus interface components exchange data with the USART protocol, and the CAN Bus interface component communicates with all network elements with the CAN protocol through the CAN bus network (Figure 2).

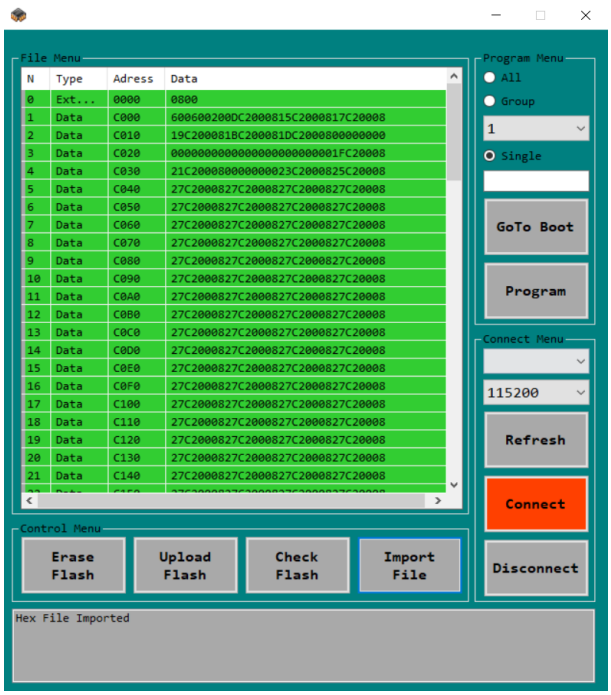


**Figure 3.** Graphical user interface on PC.

In order to update the embedded software of any network element on the CAN Bus network, the user first imports the HEX file of the new version of the firmware into the user interface application (Figure 3).

The graphical user interface program allows the user to update the firmware of the network elements on the distributed embedded system and inform the user about the success of the performed operations visually. The user can check the record type, the record address and all the data values of the HEX file lines with graphical user interface.

The HEX file uploaded by the user to the interface program is transferred to the flash memory of the Can Bus interface component which bridges the PC to the Can Bus network before programming the desired elements on the network. The main reasons of this transfer are, improving the software update time of the nodes and simplifying system message routines. Instead of using the CAN Bus interface component just for the message conversion between CAN and USART for every data message used in the programming process, the flash memory of interface component is used as a buffer to store program data. After saving the HEX data to interface component's flash memory, PC sends a program command to interface board. After CAN Bus interface board receives the program command message, it starts to send program data to desired node through CAN Bus network. Before each programming process, the user can check if the flash memory of the CAN Bus interface board is empty or not by using the graphical user interface on the PC. If the flash memory is full, user can also give command to the CAN Bus interface board to delete flash via the GUI.

The user can perform the software update process of the network elements in several different ways with the user interface on the PC. These ways are; entering each network element's individual network ID, entering the group ID associated with grouping the network elements and programming all the cards in sequence with a single command.

Each line of the HEX file loaded into the user interface contains the checksum data which is a control value specific for each line. The user interface application calculates the Checksum values for each line of the loaded HEX file and checks the validity of the file by comparing it with the control values in the lines of the HEX file and informs the user accordingly. In case of an error in any line of the uploaded HEX file, the corresponding HEX file line is indicated by red background color. After the valid HEX file is uploaded to the user interface, the HEX file of the Firmware is transferred to the Flash memory of the CAN Bus interface component by connecting to the interface component via USART with the valid Port number and Baud Rate.

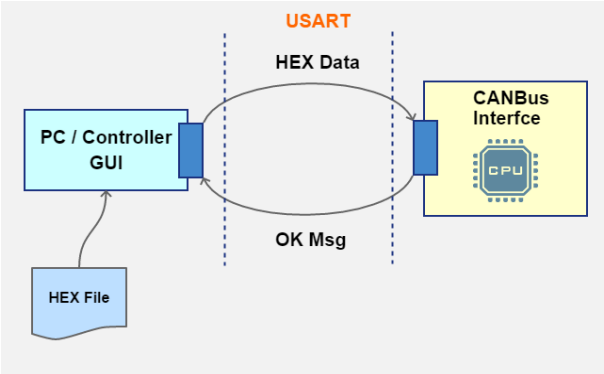


**Figure 4.** USART messaging schematic between PC and CAN Bus interface component.

Each line of the HEX file sent to the interface component as an USART message contains checksum information. The main reason of sending this checksum information in addition to HEX data is to make sure that the data is securely received by interface component. When the interface component receives the USART message for each HEX line, it calculates the control value using the relevant data and confirms the validity of the data by comparing it to the control data of the sent message. After this control, the received HEX data is written to the Flash memory of the interface component. When the received data are written to flash memory, flash record address value is also checked to ensure the writing process is successfully completed. If all of the control phases are passed, CAN Bus interface board sends the "OK" message to the PC through USART. All of these control phases and the message flows for single HEX value transfer defined as a single cycle of data transfer between PC and interface board (Figure 4). This routine is repeated for each HEX data until the last HEX data is sent. If an error occurs during the any line of HEX file transfer cycle, CAN Bus interface board sends the "ERROR" message to the PC and deletes its flash memory that keeps the HEX file data. When the GUI gets the "ERROR" message from the interface board, it stops the HEX file transfer and informs the user about the cancelled transfer routine. In addition, there is a threshold time defined for every HEX line transfer routine and if "OK" or "ERROR" messages are not sent by CAN Bus interface board before the threshold time, GUI considers the current HEX file transfer routine as unsuccessful and it informs the user about the cancelled transfer routine.

All of the elements on the CAN bus network in the system, have their own predefined 1 byte size network ID. These network IDs can be written to a specific flash address of the microcontroller while the bootloader is being loaded to each network element during production or can be defined with an 8-DIP switch hardware which can be located during the electronic card design.

After transferring all of the data in the HEX file to CAN Bus interface board's flash memory, the user enters the network ID of the desired node to be updated via the GUI and presses the "GoTo Boot" button. After the user pressed the "GoTo Boot" button, GUI sends the "GoToBoot" message with the corresponding network node ID to the CAN Bus interface board. When CAN bus interface board receives that USART message from PC, it sends the "GoToBoot" message to the network element with the corresponding node ID given by the PC. Then the network element with the corresponding ID that is running on the firmware application writes a control value to the Boot Control Page address 0x0801F000 of the flash memory that is specified in Figure 1 and resets itself. After the reset, the bootloader application which is located on the start address of flash memory runs and verifies the value at the Boot Control Page address then it starts to erase the flash sectors of firmware application. When the interface board completes the deleting process, it sends the "BootOk" message to the CAN Bus interface board and starts to wait for CAN Bus data messages of the new firmware. This procedure is referred to as placing the network element in Boot mode. When the network element jumpes into the Boot mode and sends the "BootOk" message to the CAN Bus interface board, the user is informed about the network node that is ready for the update. If the network node does not send the "BootOk" message before the threshold time, the system considers this procedure as unsuccessful. After the desired network element is put into Boot mode, the user orders CAN Bus interface board in order to program the corresponding network element by pressing the Program button via the GUI on the PC.

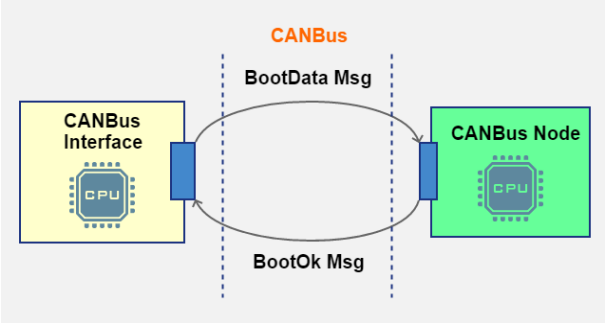


**Figure 5.** CAN messaging schematic between CAN Bus interface board and a network node.

Then the CAN Bus interface board starts to transfer the HEX data to network node with the "BootData" messages which contains the ID of the network node through CAN Bus. CAN Bus 2.0A message frame has an internal 16 bit wide CRC area. When a network message is received incorrectly even by just one node in the

network, sender detects this fault with 2 bit wide ACK zone in the frame and resends the current message again. Difference from the USART data messages in the system, no control values are sent in the "BootData" messages because there is already an error checking mechanism in the message frame. After each "BootData" message arrives at the corresponding network element, the message content is written to the firmware area in the flash memory of the network element. After network node writes the received data to its flash memory, it checks the value of corresponding flash address then sends "BootOk" message to the CAN bus interface board (Figure 5). When the interface board reads "BootOk" message, it sends another "BootData" message that contains the next HEX data. This routine is repeated for every HEX data. After all of the HEX data has been successfully sent to the network element, a special "BootData" message that contains transmission complete value is sent as a last message. If the "BootOk" message is not received by the CAN Bus interface board before the threshold time, CAN Bus interface board considers the transfer operation as unsuccessful and it stops the HEX data transfer. Then it sends the "ERROR" message to the PC in order to inform the user. After the "BootData" message that contains transfer complete value is received by the network element, the network element deletes the Boot Control Page of the Flash memory and resets itself. After rebooting, the bootloader controls the Boot Control Page address and starts the updated user application by jumping to the Flash address of the firmware because the control data was deleted during the previous run. This whole procedure that starting at entering of the desired node ID and sending the Program command from the GUI is referred as programming of a single network device. But except from the Firmware update of single node, the user can update the Firmware of the all nodes or the Firmware of the groups that are consisting of multiple nodes with a single command. In order to program multiple nodes with a single command, the user can select the Firmware update mode by using the "All" and "Group" radio buttons on the GUI. When the "Group" mode is selected, the GUI sends the Program command to CAN Bus interface board for every node ID in the selected group in order and performs the same procedure as the single node Firmware update one by one. If an error occurs during the process, update process stops and the user is informed by GUI about the node ID that has not been programmed. When the "All" mode selected, same procedure is performed by the GUI same as the "Group" mode for all node IDs.

The flowchart of HEX data transfer from the CAN Bus interface board to a single network element is represented in Figure 6. When the interface board receives the Program command with the desired node ID from the GUI, it starts the transmit routine of the HEX data which is stored in the flash memory.

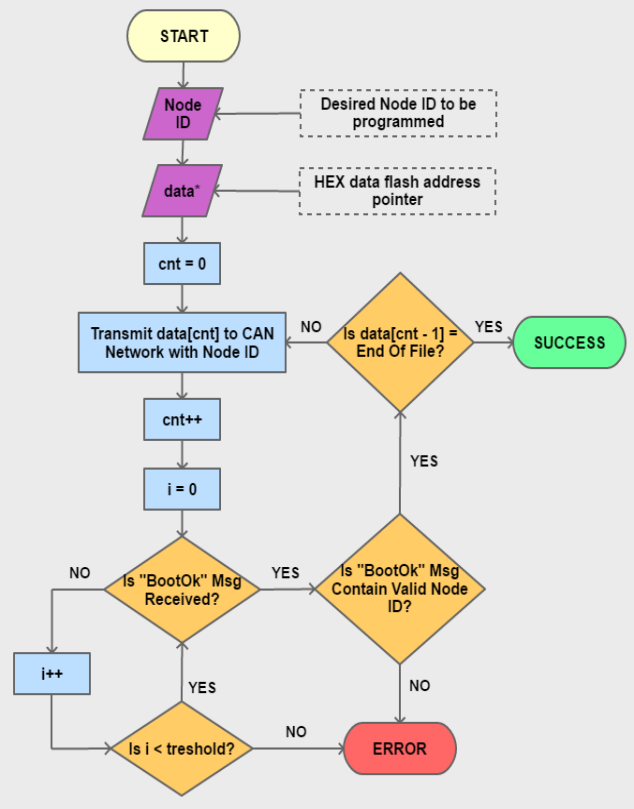


**Figure 6.** HEX data transfer from the CAN Bus interface board to a single network element.

The HEX data is represented with a pointer that addresses the starting address of the flash sector that contains the HEX file. The pointer value is incremented at every cycle of transmit routine and if the "BootOk" message is received until the threshold time, transmit routine continues until the last HEX data as mentioned before.

The message IDs used in the system are fixed for the Boot messages sent by the CAN Bus interface hardware but for all messages that is sent from the network elements, ID value is set to 0x20 + Network ID. That means the network IDs of the nodes can be understood from CAN Bus message IDs. The types of the messages that are sent from nodes are being sent inside the data frame of CAN Bus message as 1 byte. The reason for the use of variable message IDs according to the identity of the network element is to ensure that if multiple network elements send the same message at the same time, priority is given between the messages and, if desired, by a hardware filter to a particular ID group to ignore messages from these network elements by other network elements according to the design of the system.

## 3. Conclusions

The software update system has been implemented on an existing discrete embedded system that has high speed CAN Bus network and consists of 46 network elements. The embedded software of the network elements of the related system has been updated successfully without any errors. During the tests, the network elements were put into the Boot mode in order to be updated individually, in groups and all elements at the same time by the command that are sent from the computer.

In the programming process during the tests, the transmission of the 20 KB size HEX file which contains the updated firmware was completed by an average of 1.2 seconds for each network element by entering the network ID of each network element individually on user interface. Also during the programming tests with multiple network elements, transfer time of the same HEX file to all 46 network elements took about 50 seconds. Compared to the single network element file transfer mode, the average transfer time of one network element is faster in multiple transfer modes. The main reason is that in the multiple programming mode, the user interface informs the user only when an error occurs. Instead of informing user for every network element after the transfer is completed, this method uses the UART communication and GUI functions less than the single transfer mode.

In addition to software update processes, the computer was used to read and write the parameters of network nodes used on the current system. As a result of the test, it has been observed that the system can also be used as a fault diagnosis system in order to read predefined runtime errors of the network elements.

With the system designed in this study, complete firmware update system for distributed embedded systems with CAN Bus field network is provided for general purpose. Also with the custom bootloader that have been developed during the study, in application firmware update over CAN Bus feature is integrated to 32 bit STM32F072 microcontroller. In addition, depending on the distributed embedded system, if more computing power is needed in the future, the Cortex M0 network elements that are used in the system can easily be replaced with higher level Cortex M series microcontrollers.

## References

[1] Fan X., 2015. Real-Time Embedded Systems Design Principles and Engineering Practices, 1st ed., Newnes, Oxford, UK.

[2] Siegesmund M., 2014. Embedded C Programming, 1st ed., Newnes, Oxford, UK.

[3] Eisenreich D., DeMuth B., 2002. Designing Embedded Internet Devices, 1st ed., Newnes, MA, USA.

[4] Puri S. B., Nayse S. P., 2013. Green House Parameters Monitoring Using Can Bus and System on Chip. International Journal of Advances in Engineering Research & Technology, **2**(5), 1975-1978.

[5] Yiu J., 2011. The Definitive Guide to ARM Cortex-M0, 1st ed., Newnes, Oxford, UK.

[6] Chu L., Feng L., 2010. Implementation of CAN Bootloader Based on Freescale MCU. Journal of Suzhou University (Engineering Science Edition), **2**(15), 57-61.

[7] Zhang J., Zhu X., Peng Y., 2015. Implementation and Research of Bootloader for Automobile ECU Remote Incremental Update, AASRI International Conference on Industrial Electronics and Applications, London, UK, 27-28 June, doi: 10.2991/iea-15.2015.39.

[8] Xu Y., Wang R. G., Cheng A. Y., Li R., 2013. Design of online upgrade system for the software of vehicle ECU based on CAN-bus. International Journal of Advancements in Computing Technology, **5**(1), 79-87.

[9] Regenstein K., Kerscher T., Birkenhofer C., Asfour T., Zollner M., Dillmann M., 2007. Universal Controller Module (UCoM)-component of a modular concept in robotic systems. IEEE International Symposium, Vigo, Spain, 4-7 June, 2089-2094.