International Journal of Future Generation Communication and Networking
Vol. 13, No. 2s, (2020), pp. 228–233

An Overview of Wireless Programmer by using FOTA Technology

Shubham Kulkarni[1], Prof. S. K. Patil[2], Prof. M. D. Katole[3], Hrishikesh Zanjad[4], Omkar Kulkarni[5]

[1,2,3,4,5]*Dept. of E & TC Engg. Smt. Kashibai Navale College of Engineering, Savitribai Phule Pune University, Pune*

[1]shubhamskulkarni54@gmail.com

[2]skpatil_skncoe@sinhgad.edu

[3]katole.mukesh@gmail.com

[4]hrishikeshzanjad98@gmail.com

[5]omkarkulkarni984@hotmail.com

Abstract

Over the years, a rapidly growing number of IoT devices are found in the market in all the fields including industry and automation these devices, which are deployed in vast numbers are frequently used over many years without any modification or replacement. They pose a risk of malfunction or they might become prone to errors. We plan implementation of durable and stable system for building secure firmware updates for embedded devices based on microcontroller using Wi-Fi module or any wireless transmission medium. This will include mechanism to build the updates from source and automatically transfer and install it on the target device. Once the program is transferred wirelessly to the device the bootloader will burn it into the flash memory. The program that is recently installed into the microcontroller will be executed.

Keywords- *FOTA (Firmware over the air), SD card, bootloader, ESP8266, FTP.*

I. INTRODUCTION

In embedded systems, the software, also known as firmware, is an essential part of system. Embedded systems are often thought as systems that never change their requirements or functionality. However, practical use shows that the environment in which these systems run in fact, does change. In most of the cases the requirements can be met by changing firmware and without any modification in hardware. In embedded systems, software, also known as firmware, is an essential part of system [2]. Embedded systems are often thought as systems that never change their requirements or functionality. However, practical use shows that the environment in which these systems run in fact, does change. In most of the cases the requirements can be met by changing firmware and without any modification in the hardware.

II. LITERATURE SURVEY

A. An over the air update mechanism for ESP8266

Presented by, Dustin Frisch, Sven Reißm ann, Christian Pape. University of applied sciences, Fulda, Germany. Presented by, Dustin Frisch, Sven Reißm ann, Christian Pape. University of applied sciences, Fulda, Germany.

ESP8266 is a microcontroller which is easy to use as it This is an application paper published by Microchip regarding AN851 bootloader and its operation [4].

In this paper data memory partitioning of pic18f452 in order to use bootloader is studied. Bootloader performs operations such as read, write, erase etc. for removing the previous program and update the memory with newly updated program. This ensures that the microcontroller continues to perform the desired operation [1]. This paper will help to design a relevant bootloader

ISSN: 2233-7857 IJFGCN

for the required application by referring code and commands in the code of AN851.
has inbuilt Wi-Fi module. In this paper the author has built an OTA system using ESP8266. They have used git ASM server as a source server or as a build server.[1]

This system is essential part of the home-automation development and deployment in the *Magrathea Laboratories e.V.*
Hackerspace

B. Bootloader Design for Microcontroller in Embedded System Prepare Your Paper Before Styling

Presented by, Jacob Beningo, CSDP.Jacob Beningo is a Certified Software Development Professional (CSDP), chair of the IEEE Consultants Affinity Group, an independent consultant and lecturer who specializes in the design of embedded software for resource constrained and low energy devices. He has successfully completed projects across a number of industries including automotive, defense, medical and space.
In this paper author has discussed how to write a universal bootloader for any microcontroller.
During the initial stages of product development, it was quite obvious for the boot-loader to ignored by development team. Primary reason for this is that the boot-loader is not primary end product that is expected by the customer but after certain analysis boot-loader is potentially most important part of that product [5]. Boot-loader allows an organization to launch their product with software that only fulfils portion of their final requirement and then add features to their product once it has been launched into the market. It also allows them to fix bugs that are discovered after system has been released and installed by customer and tested into wild.
For an embedded software engineer, boot-loader requires a full understanding of how a processor works, how to utilize its memory and how to work on processor at lowest levels. Boot-loader development can be an extremely challenging endeavour to undertake but absolutely the rewarding one. Once a developer has gone through the process, each additional boot- loader becomes easier and easier to implement by following a common and consistent approach to the boot-loader des

C. PIC interfacing with SD card

Link: https://openlabpro.com/guide/sd-card-using-petit-file-system/

This website shows how to interface PIC microcontroller with the SD card in order to perform communication operation. Just like interfacing of any other I/O device with a microcontroller SD card interfacing also requires various commands in order to perform certain operations [5].

When we come across smaller microcontrollers, where the memory is limited and the RAM size is pretty small when compared to the size of a sector, creating a file system to operate an SD card becomes inconvenient. In order to overcome this issue in smaller microcontrollers, the FatFs (FAT FS) is broken down into a smaller, less complex and more convenient file system known as the Petit FatFs (petit fat file system).

Various operations provided by this file system are:

- pf_mount – Mount a Volume
- pf_open – Open a File
- pf_read – Read File
- pf_write – Write File
- pf_lseek – Move read/write Pointer
- pf_opendir – Open a Directory
- pf_readdir – Read a Directory Item

D. Using HTTP and an HTML interface to Upload and Download files to an ESP8266 Filing System

Link: https://github.com/G6EJD/ESP32-8266-FIle-

This website gives procedure to implement an interface to upload and download files an ESP8266 filing system. There are few steps which we need to follow and implement the server. Avoid combining SI and CGS units, such as current in amperes and magnetic field in oersted's. This often leads to confusion because equations do not balance dimensionally. If you must use mixed units, clearly state the units for each quantity that you use in an equation.

Sr.no.	Paper Name	Author Name	Technology used	Result
1	An over the air update mechanism for ESP8266 microcontroller[1]	Dustin Frisch, Sven Reißmanny, Christian Pape	Microcontroller consists of inbuild Wi-Fi module.	Easy to implement as interfacing is required is much less.
2	Bootloader design for embedded systems[2]	Jacob Beningo		Explains working of standard bootloader. Different bootloader commands and functions carried out by bootloader.
3	AN851 A FLASH Bootloader for PIC16 and PIC18 Devices[3]	Michannelrochip		Data memory partitioning in order to use Bootloader
4	https://openlabpro.com/guide/ sd-card-using-petit-file- system/ [4]			Interfacing of microcontroller with SD card in order to perform communication operation
5	Using HTTP and an HTML interface to Upload and Download files to an ESP8266 Filing System[5]	G6EJD YouTube Channel		Implement an interface to upload and download files an ESP8266 filing system

Table 1. Details of Literature Survey

III. BOOTLOADER

Integration of bootloader support in a product consists of three parts

- Host Application – Utility used to process End Application memory space and firmware version
- Device Bootloader – Dedicated firmware responsible for managing End Application on device
- Device End Application – Products operational application firmware.

The host application is responsible for loading the new hex file, and send it to the bootloader through supported command syntax. The end application is required to be aware of the bootloader, and must understand how to return control to the bootloader upon request, or configured events. The bootloader by default is generated to run upon start-up, and confirms if a valid application is loaded. If a valid application is present, control is relinquished, otherwise operation will remain within the bootloader. The host application used to manage the bootloader process can be Microchip's Unified Bootloader Application, a custom-made stand-alone application, or a separate external Microcontroller device. Either way, the end purpose remains the same; updating the end application firmware version through use of the Bootloader and supported commands. This user's guide describes one form of implementation, using the Bootloader Generator MCC software library to produce code which supports proper command syntax for interaction with Unified Bootloader host Application. It will describe how to configure an end-application to be aware of boot-loading possibility control back to the Bootloader under a trigging condition [2].

Each boot-loader will have its own unique set of requirements based on the type of application; however, there are still a few general requirements that are common to all boot-loaders

1) Ability to switch or select the operating mode (Application or boot-loader)
2) Communication interface requirements (USB, CAN, I2C, USART, etc)
3) Record parsing requirement (S-Record, hex, intel, toeff, etc)
4) Flash system requirements (erase, write, read, location)
5) EEPROM requirements (partition, erase, read, write)
6) Application checksum (verifying the app is not corrupt)
7) Code Security (Protecting the boot-loader and the application)

Use of an IO pin indicator is an option available to use a dedicated pin to show when if the device is in bootloader or end-application operation. Form of indication may vary depending upon the product design specification. The IO pin indicator can be simple, such as a LED used to indicate state, or as a signal connected to another device in the product circuit.

The bootloader should protect itself from accidental over-write. Therefore, attempts to write into the memory where the bootloader resides should be prevented. PIC® microcontrollers have two methods to ensure this: hardware and software. Write- protect Configuration bits can selectively write-protect various regions of the program memory. The advantage to hardware protection is a smaller code footprint. The downside is that the block size is fixed and may leave memory wasted [4].

The address protection can also be accomplished in software. Code can check the destination address of each Write/Erase command. If the address range conflicts with the bootloader region, the command request is rejected. The software requires a little more code space, but it has the advantage of additional flexibility [2]. For example, special bootloader code could be written to allow the original bootloader to be replaced with an updated version in case a bug emerges after production has already occurred/begun. This is not possible if protected via hardware through the write-protect Configuration bit is set.

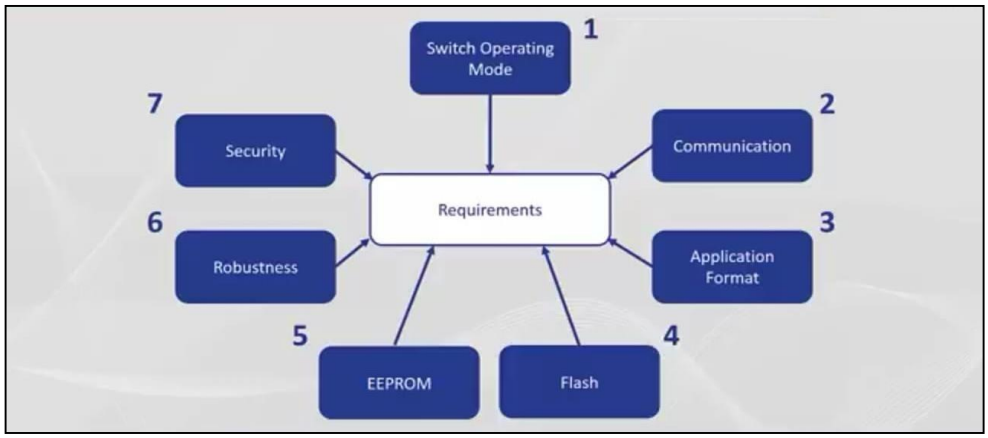

Fig. 1: Bootloader Requirements

IV. SD CARD

The secure digital card (SD) is a low cost, non-volatile memory card format developed by the SD Card Association. Since its inception back at the start of the century, the demand for this medium-sized, energy and space-efficient, the memory storage device has been growing at a fast rate. Therefore, to meet the market requirements, the SDA was set up as a non-profit organization to promote and create SD Card standards. There are various topics related to the SD card such as the

different device families, speed classes, smart cards, card security and so on and it is used in various markets like digital cameras, personal computers, and embedded systems. Some of the standard variations include SD, SDHC, SDXC, SD-ultra high speed etc. The microSD is the miniaturized SD memory card format with a small form factor and is widely used in various electronic devices[3].

What we are going to learn is the use of SD cards in an embedded system. To be specific, we will be dealing with the use of SD cards in small embedded systems.

SD card has a native host interface apart from the SPI mode for communicating with master devices. The native interface uses four lines for data transfer where the microcontroller has SD card controller module and it needs separate license to use it. Since the SPI is a widely used protocol and it is available in most low-cost microcontrollers, the SPI mode is the widely used interface in low cost embedded systems [1]. The working voltage range of SD family is 2.7V to 3.6V and this is indicated in the operation condition register (OCR).

Most micro-controllers use the SPI communication protocol to interface with the SD cards. The SD cards have a microcontroller that shows their availability to the master controller (microcontroller). The micro-controller sees the SD card as an addressable sector on which read/write functions are possible. Once the microcontroller is in the SPI mode, communication between the master and the slave is done via 4 pins viz. clock, chip select, data in and data out. It should be kept in mind that throughout the communication between the two devices, the micro-controller will be sending out the clock. Most development boards have a dedicated SD card slot. But to understand the connections, let us analyze this fairly Next comes the tricky part, initializing the SD card and performing the raw data communication [4]. A systematic approach to programming the software would make the task pretty easy.

But first, it is important to learn how the micro-controller activates the SD card. There are a fixed set of commands and responses, which must be followed to create a command to response structure in our program. The data is transmitted in a byte- oriented format with a definite length.

Fig 2: SD card

V. METHODOLOGY

As per the requirement of transmission system for the firmware transmission through local channel network via personal SSID and PASSWORD for encryption a suitable sensor network i.e. ESP8266 Wi-Fi module will be used.

Transmission part in categories as front- end and back-end. In the front-end system by using CSS and HTML languages a webpage is created for browsing .HEX file which is to be execute in target controller at the receiver end. Parallelly back end of the transmitter consists of an algorithm which transmits .HEX file using TCPIP protocol to the server. SD card is used as server storage in transmission [3]. File which is browse in webpage is stored in server of local network.

Receiver part of the system is totally an embedded part. In the beginning SD card contain a HEX file in a system format. This hex file is loaded into program memory of target controller by using Bootloader [2].

Several interrupts are use in bootloader to switch the main code to bootloader code. Flashing is used to erase the previous code in target controller. The aspects of bootloader are security switch, flashing,

testing, linker scripts and SD card commands in execution of the code.

Initially main code is executed in target controller after receiving interrupt commander Bootloader is executed and load new hex file into program memory of controller. Server storage and target controller is connected via SPI protocol for better synchronization this will increase the productivity of transmission also it will make system more immune to the losses.

VI. CONCLUSION

- The project will implement the system for providing OTA updates to the embedded devices compensating the shortcomings of other already available systems. Our work has high efficiency, it is more accurate and reliable [1].
- Main purpose of this study was to remove the requirement of physical connection to update or modify the system with reduction in man power and provide higher efficiency.
- This system will increase preparedness for emergency situations if encountered by the system.
- All the components used in this system are easily available and low cost .

REFERENCES

[1] An Over the Air Update Mechanism for ESP8266 Microcontrollers Dustin Frisch, Sven Reißmann , Christian Pape.J. Clerk Maxwell.
[2] Bootloader Design for Microcontrollers in Embedded Systems Prepare Your Paper Before Styling Presented by, Jacob Beningo, CSDP.
[3] AN851 A FLASH and Bootloader for PIC16 PIC18 Devices PIC interfacing with SD card.
[4] Using HTTP and an HTML interface to Upload and Download files To an ESP8266 web server.
[5] Link: https://github.com/G6EJD/ESP32-8266-FIle-
[6] PIC interfacing with SD card
[7] Link: https://openlabpro.com/guide/sd-card-using-petit-file-system/