Designing a MSP430 Bootloader

Septimiu Mischie
Faculty of Electronics, Telecommunications and Information Technologies
Politehnica University of Timisoara, Romania
septimiu.mischie@upt.ro

Robert Pazsitka
Faculty of Electronics, Telecommunications and Information Technologies
Politehnica University of Timisoara, Romania
robert.pazsitka@upt.ro

***Abstract* – This paper presents implementation of a serial bootloader for a MSP430G2553 microcontroller. It is implemented in assembly language and it is written in the information area of the microcontroller memory by using any usual tool for flashing. Then, the source of the application can be developed. The application code that is obtained by using the compiler and linker must be sent to the microcontroller. For this purpose, a serial communication between PC and the UART port of the microcontroller via an USB to serial converter is implemented. In this case a MATLAB application is run on the PC while the bootloader is run on the microcontroller. Finally, the bootloader receives and writes the application code in the main area of the Flash memory. Thus, an efficient method to update the application code, having advantages in comparison with those in literature is proposed.**

Keywords-microcontroller; bootloader; application; serial communication; flash memory;

I. INTRODUCTION

A bootloader represents a software statement which is resident in a part of the flash memory of a microcontroller [1], [2]. It writes the application in the rest of the flash memory without using any other device as a programmer-debugger. The bootloader requires a serial communication between a Personal Computer (PC) and the microcontroller. Thus, the bootloader receives the application code from the PC and writes it in the specified memory. It should have a low memory size and generally it not be changed. A bootloader is used especially in the manufacturing phase of electronic device. In this way, the need to use a programmer-debugger which is an expensive device is eliminated. In the literature there are presented bootloaders for different kind of microcontrollers, such as PIC [3], AVR [4], ARM [5], Renesas [6],[7] and NXP [8]. Texas Instruments MSP bootloaders are also presented in [9], [10] and [11]. The approach of [10] is implemented for MSP430G2001 device. It used software UART of 9600 bps to receive the application code. Furthermore, this process is executed in several steps in a terminal (HTerm): sending syncro, sending the file of application code, computing and sending the checksum and receiving the microcontroller answer. The bootloader presented in [11] is implemented for MSP40G2553 microcontroller but uses two development systems, a target and a host among the PC. This approach however complicates the use of the bootloader.

In this paper, we implemented a bootloader mainly following the idea of [10] with a few improvements. Thus, the bootloader is designed for a more advanced microcontroller, MSP430G2553. We use hardware UART having a faster Baud Rate, 56000 bps, to receive the application code. All the steps for sending the code from the PC are integrated into a MATLAB application and thus the proposed bootloader is very easy to use. Furthermore, we propose a more advanced application that uses all of the microcontroller's peripherals and has three interrupt sources. The paper is organized as follows. The second section describes the proposed bootloader. The third section presents some details regarding the implementation. The fourth section presents the experimental result while the last section concludes the paper.

II. DESCRIPTION OF THE BOOTLOADER

The paper presents design and implementation of a bootloader for a MSP430G2553 microcontroller (MCU). There is a single piece of software for MCU which is generally called bootloader whose architecture is presented in Fig. 1. It has two working modes: bootloader mode and application mode. Usually, at start up, it enters in *application mode*, because P1.3 pin is on a logic 1. This represents the normal mode. The first version of the MCU software contains a very simple application. In order to change the application with the real one, the bootloader must enter in *bootloader mode*. For this, the MCU is kept in reset while the P1.3 pin is set on logical 0. Then, the program waits for the *application code* to be written in the Flash. This code will be sent to MCU by the PC via a serial interface (UART) so it will be seen later. When all the bytes of the application code have been written, a Reset is executed, and thus the program restarts its execution and enters in application mode.

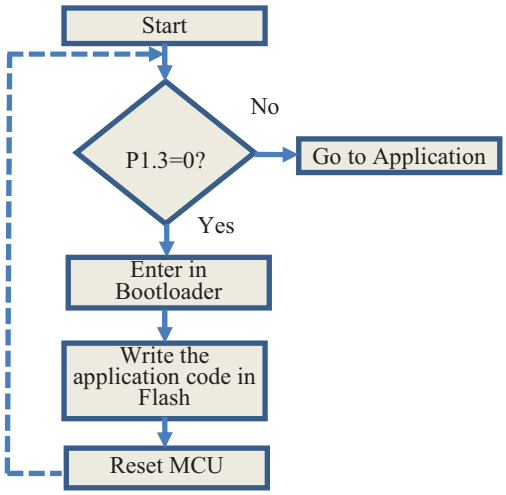

Figure 1. The architecture of the MCU software

ISBN 978-80-261-0812-2, © University of West Bohemia, 2019

A. *The MemoryMap*

Even if we have a single piece of software for MCU, it has two parts. Both of them are implemented in assembly language. The first part, which is called bootloader code, is written in the *information* part of the microcontroller flash memory, in the range address 0x1000 to 0x1FFF (Fig. 2). The second part that represents the application code is written in the *main* area of the microcontroller flash memory, in the range address 0xC000 to 0xFFFD. The last two memory locations of Flash, 0xFFFE and 0xFFFF contain the RESET vector, 0x1000, that represents the address where the bootloader starts. The application code can be rewritten as many times as it is needed, and can be in either assembly or C language. Thus, regarding Fig. 1, the Start is always in the information memory. Then, depending on the status of P1.3, the execution can continue in the information memory or can do a branch in the main memory at the address 0xC000, where the application code has already been written.

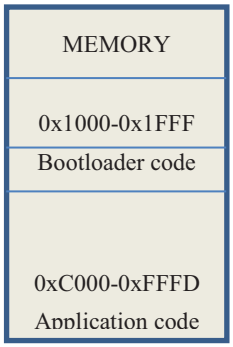

Figure 2. The memory map of the MCU software

B. *The hardware structure*

In order to write the bootloader code and the first version of the application code in the flash memory of the MCU, the MSP-EXP430G2 development system is used. This operation is executed only once. This system contains a programmer-debugger that uses 2 wire Spy-by-wire protocol and also has a 20 pin socket where the MCU (target) is introduced. In order to (re)write the application code, the MCU must be removed from the MSP-EXP430G2 and a serial connection to the PC is needed. The USB-TTL (UART) converter CH340 is used for this purpose, Fig. 3. In this way, a serial (UART) connection between the two modules, using the two wires, Tx-Rx and Rx-Tx, is established. MCU is also powered by USB-TTL converter. Other elements of the hardware structure are the reset circuit, including the R and C components and two push buttons, S1 that can connect the RST pin to GND and S2 that can connect the P1.3 pin to GND. These two buttons (S1 and S2) are needed to force the entrance in bootloader mode as in the first part of this section was presented.

C. *The software tools*

In order to create and use the proposed bootloader, two software tools are needed:

1. IAR Embedded Workbench is used on the PC to edit, compile and link both the bootloader code and the application code. While the bootloader code and the first version of the application code are written in the

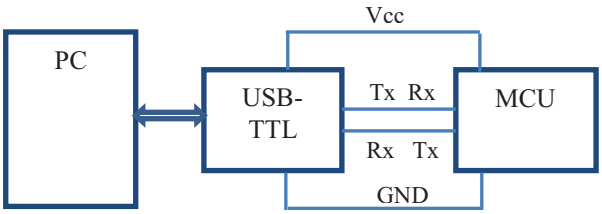

Figure 3. The hardware structure

microcontroller memory by using IAR and the MSP-EXP430G2, the regular application code is only saved in a file on the PC.

2. The MATLAB environment is used to implement the communication between the PC and the UART port of the microcontroller. Thus, by using this software, the application code is sent from the PC to the microcontroller. The bootloader code that is resident into information memory receives each byte of the application code and writes it in the main area of the flash memory. In addition, the checksum is computed and sent to the microcontroller. Furthermore, some synch characters are exchanged between PC and microcontroller using the MATLAB.

III. DETAILS ABOUT THE IMPLEMENTATION

The most important part of this paper is the bootloader code which runs in the information part of the MCU Flash memory. This communicates with the PC via the MATLAB that accesses the USB-TTL converter as a COMx port. Fig. 4 presents the two algorithms that must synchronize with each other.

Thus, the MCU algorithm in Fig. 4 represents actually the content of the “Write the application code in Flash” block of the Fig. 1. At the beginning, the algorithm waits for the Sync character from the PC. If the received character does not have the expected value, that is 65, a Reset is generated. Otherwise, the main area of Flash is erased by segments of 512 bytes. At the end of this operation the RESET vector is restored with the address from the beginning of the information memory (0x1000), where the bootloader code is started.

After erasing the main area of Flash, the MCU sends the byte Ch=110 to the PC to notify it that the erasing has been finished. Thus, the PC starts to send the application code to the MCU, one byte at a time. As a consequence, MCU receives each byte and writes it in the Flash memory. In parallel, each of the two systems computes its own checksum (a XOR byte by byte), denoted by chksum1 and chksum2. Even if the size of the application code is less than the capacity of the main area of the Flash, the interrupt vectors are however located in the range 0xFFDD-0xFFFD. Thus, the number of bytes which are sent from the PC to the MCU is exactly 16 kB minus 2 bytes. The 2 missing bytes thus prevent overwrite of the RESET vector, which has already been written at the locations 0xFFFE and 0xFFFF. After the PC sends the last byte of the application code, it sends the final value of checksum to the MCU. This, in turn compares the received byte with its own checksum. If the two bytes are the same, the 249 byte is sent to the PC. Otherwise, the 254 byte is sent to the PC. Thus, the PC can know

if the writing process was a success or if it failed and displays a message according to this.

In the following, some important things about the generation of the bootloader code and the application code are presented. It is known that the information memory, where the bootloader code is written, contains in the last locations some constants, for instance to calibrate the MCU internal oscillator. These must be hand written before the erasing and writing. Furthermore, the SMCLK signal, an output of the oscillator, must be calibrated to configure the UART for a right value of Baud Rate. This part is included in the MCU algorithm of Fig.4 after the Start bloc but is not presented due to the lack of space.

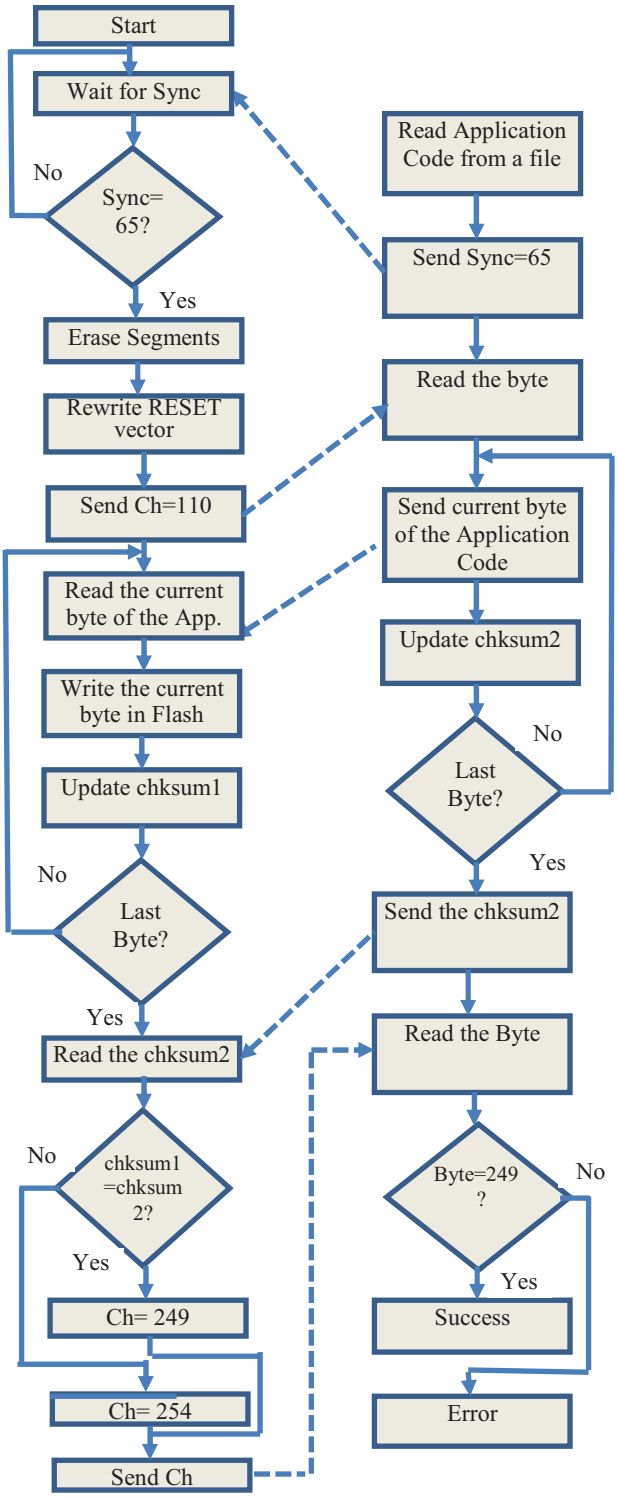

Figure 4. MCU (left) and PC-MATLAB (right) algorithms

As a part of IAR Embedded Workbench, the linker is used to create the executable code for the MCU. Thus, when the bootloader code and first version of the application code are written, both information and main parts of Flash are used. Therefore, the default form of *linker configuration file* should be modified as follows:

All the entries of the *Information memory* section must be removed and the following must be introduced instead:

-P(CODE) BSL=1000-10FD

On the other hand, the *Read-only memory* section must be unchanged. The next statement specifies where the code will be written:

-P(CODE)CODE = C000-FFFD

The assembly source of the bootloader must contain the following directives:

RSEG CODE, before the part of the program that corresponds to application.

RSEG BSL, before the part of the program that corresponds to bootloader.

Thus, it follows that the application part, denoted by CODE will be written starting with the address 0xC000, while the part of the program that corresponds to bootloader, denoted by BSL, will start with the address 0x1000.

Furthermore, to specify where the program will start, the label RESET will be introduced in front of the first line of the bootloader program.

RESET MOV.B #08, P1REN

In this way, the address 0x1000 of this instruction, will be written to the RESET location in the area of the interrupt vectors (0xFFFE and 0xFFFF), see also Fig. 2.

To enable writing in both information and main memory, the following options should be checked in IAR Embedded workbench (Project/Options /Debugger/ FET Debugger/Download): *Allow erase/write access to locked flash memory* and *Erase main and Information memory*.

Regarding the generation of application code, two elements must be presented. First, it is known that this code must start at address 0xC000. If it contains declarations of some global variables, their values are by default written starting with the address 0xC000. Then, through the execution of the program they are moved to RAM. In order to keep the start of the program at 0xC000, the following entry of the *Read only memory* section of the linker configuration file is changed

-Z(CONST)DATA16-C,DATA16-ID,TLS16_ID, DIFUNCT,CHECKSUM=C800-FFFD

where the value 0xC800 was introduced instead of 0xC000. In this way, the global variables are written starting with address 0xC800. This value has been

chosen because in our implementation the application code does not exceed this range. Then, once the linker configuration file was changed, the following option in IAR Embedded Workbench, Project/Options/Linker /Output must be checked: *Other, Output format: msp430-txt*.

Secondly, in order to generate the application code, the following steps are executed: 1. Download and Debug. 2. Select View/Memory and then Flash. 3. Right Click and Memory Save, where the ranges of addresses should be 0xC000 to 0xFFFF. In this way a file containing the application code is generated and can be open in MATLAB (see Fig. 4). This code has at the end the numbers 0x00 and 0xC0, that by concatenation represent the start address of the application. However, these numbers are ignored when the application code is written in main flash area.

IV. Implementation and Experimental Results

A bootloader according to those in previous sections was implemented. First, the MCU software having the structure of Fig. 1 (bootloader code and a simple application code) was written in the flash using IAR Embedded Workbench and the MSP-EXP430G2. The Baud Rate of the UART communication was set to 56000. We implemented then a more advanced application that uses the most important peripherals of MSP430G2553 microcontroller. This application mainly does the following: executes periodic measurements of either the voltage applied to pin P1.4 or the temperature corresponding to the internal sensor of ADC; a LED connected to pin P1.0 is configured to blink at each measurement (i); sends the measurement result to the PC by USB to TTL converter of Fig.3 (ii); the user can cyclically change the period of measurement between 3 values by typing the + key of the PC keyboard; in addition, the switching between voltage and temperature measurement can be made by typing the keys V and T, respectively (iii); the periodic measurement can be stopped or restarted by successive pressing of S2 pushbutton connected to P1.3(iv).

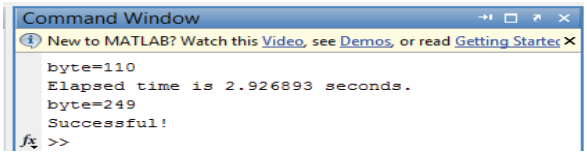

Figure 5. A screenshot from MATLAB

After generating the application code using the IAR Embedded Workbench, the MCU software is forced to enter in bootloader mode and the MATLAB application is started. A screenshot of the MATLAB command window is presented in Fig. 5 to show the progress of the writing process, according to the algorithm of Fig. 4. The time of 2.92 sec. represents the time required for sending the 16382 bytes of the application code to MCU, and as a consequence the time for it to be written. After displaying the message Successful!, the LED of the pin P1.0 starts to blink to notify us that the application code was written and started.

To conduct a further validation of the writing in Flash, another tool can be used: Flasher [12]. It can be invoked by command prompt and can read the content of the Flash (main or info) and saves it in a file. However, the programmer debugger of MSP-EXP430G2 is needed to use Flasher. We compared the content of the file read by Flasher with the file generated by IAR Embedded Workbench (application code) and they are the same.

V. Conclusions

The paper presented the design and implementation of a bootloader for a MSP430G2553 microcontroller. The method is easy to use. After the bootloader code is written in the information part of the microcontroller Flash memory it requires only two steps: generating the application code and using the PC MATLAB application to send the code to the MCU that is in bootloader mode. In can easily be changed to be used for other microcontrollers.

In terms of future work, we want to replace the MATLAB application to one implemented in a free environment such as SciLab or Octave. In addition, we could connect a Bluetooth adapter to the microcontroller to allow an over the air update of the application code.

References

[1] Y. Kang, J.Chen and B. Li, Generic Bootloader Architecture Based on Automatic Update Mechanism, 2018 IEEE 3rd International Conference on Signal and Image Processing, pp.586-590.

[2] I. Pratt and S. Zhong, Bootloader design considerations for resource-constrained microcontrollers in RFID reader designs, 2014 IEEE RFID Technology and Applications Conference, pp. 50-55.

[3] S. Nuratch, "A Serial Bootloader with IDE Extension Tools Design and Implemntation Technique based on Rapid Embedded Firmware Development for Developers," 2017 12th IEEE Conference on Industrial Electronics and Applications (ICIEA), pp. 1865–1869.

[4] M.Lewandowski, T. Orczyk and P. Porwik, "Dedicated AVR Bootloader for Performance Improvement of Prototyping Process," 2017 MIXDES – 24th Int. Conference "Mixed Design of Integrated Circuits and Systems", pp.553–557.

[5] C. Sha and Z. Ying Lin, "Design Optimization and Implementation of Bootloaders in Embedded System Development,", 2015 International Conference on Computer Science and Applications, pp. 151–156.

[6] D. Bogdan, R. Bogdan and M. Popa, "Design and Implementation of a Bootloader in the Context of Intelligent Vehicle Systems,"2017 IEEE Conference on Technologies for Sustainability, pp 1-5.

[7] D. Bogdan, R. Bogdan and M. Popa, "Delta Flahing of an ECU in the Automotive Industry," 11th IEEE Int. Symposyum on Applied Computational Intelligence and Informatics, May 12-14 2016, Timisoara, Romania, pp. 503-508.

[8] P. Lajsner, P. Krenek, P. Gargulak, Developer' Serial Bootloader, Freescale Semiconductor Application Note, AN2295, 2013.

[9] www.ti.com, SLAU319X, MSP430 Flash Devices Bootloader (BSL), July 2010, revised April 2019.

[10] www.ti.com, SLAA 450f, Creating a Custom Flash-Based Bootloader (BSL), april 2010, revised July 2018.

[11] www.ti.com, SLAA 650d, MSPBoot-Main Memory Bootloader for MSP430 Flash Microcontroller, June 2013, revised February 2018.

[12] www.ti.com, SLAU 654D, MSP Flasher, November 2015, revised November 2017.