v1.0 April 15, 2019

SD Card Bootloader for MSP430

This is a system for updating the firmware of MSP430 microcontrollers using an SD or SDHC memory card. If an SD or microSD port is built into the circuit, the user can copy a downloaded update file to the SD card, insert it into the port, and power up the device. The bootloader will detect the card, then perform the update as it reads the data from the file using SPI. This eliminates the need for interface devices, cables or jumpers, and the need for any installed driver or software on the user's computer. However, the user will need an SD or microSD card slot on his computer, or a USB SD card reader.

The bootloader is flashed to the MSP430 at the beginning of MAIN memory using standard JTAG or BSL flashing. It occupies 1K of memory. The reset vector at 0xFFFE must always point to the beginning of the bootloader. Thereafter, on power up, if no card is detected, the bootloader jumps to the application, which must begin at MAIN + 0x400 (the "NewMain" address).

Included here in addition to the bootloader is the Windows program SDBSL.exe, which takes in a standard Intel-HEX or TI-TXT file and creates the update file in the binary form needed by the bootloader. The C source code is included.

Bootloader examples included here are specifically for the MSP430G2452's USI module and the MSP430G2553's USCI-B module. It may be possible to adapt the code to other processors with those modules. The assembler source code is provided.

Requirements and Limitations

To achieve the smallest footprint for the bootloader, and thereby make it useable in processors with little memory, practical limitations were adopted. This is what the bootloader supports:

1. **SD and SDHC Cards:** The bootloader works only with standard SD/SDHC and microSD/SDHC cards, not MMC or SDXC. Standard SD cards are available in various sizes from 128MB to 2GB, any of which is more than big enough for this process, but a larger SDHC card can also be used.

2. **Partition Format:** The SD card must be formatted with a Master Boot Record ("MBR"). The partition containing the update file must be the first entry in the partition table beginning at 0x01BE in the MBR, and the partition must be formatted as FAT16 or FAT32.

3. **File Storage:** Each file containing a firmware update must be stored in the Root Directory of the partition, and the file contents must be stored in consecutive sectors. Consecutive sectors will automatically result if the formatted cluster size is greater than the file size. For example, the G2553 update file is about 15KB. If the cluster size is 16KB, all update files will automatically be saved in the consecutive sectors of a single cluster. If a smaller cluster size is used, file sectors may still be

consecutive if files are copied to a newly formatted card in sequence and nothing is ever deleted. With small (128MB and 256MB) SD cards being available for US$1.00 or less, it makes sense to include such a card with each copy of the project device. The developer can then make sure the card is formatted properly.

4. **File Format:** The update file must contain in binary form an image of the entire contents of MAIN memory from the beginning of the flash segment following the end of the bootloader, known as the NewMain address, up to, but NOT including, the reset vector at 0xFFFE, plus a one-byte XOR checksum. So for the G2553, the file must always be exactly 15,359 bytes long, and a G2452 file will always be 7167 bytes long.

5. **Firmware Format:** The firmware application must always begin with executable code at NewMain. The bootloader will always jump to that address on power up if no card is detected.

6. **File Naming:** Files can be named anything, but the bootloader only looks at the short 8.3 DOS version of the filename, which always has uppercase letters. It searches the entire Root Directory, and selects the file, if any, which (1) is the right size, and (2) has the highest "value" in the filename's first three characters. The characters are compared using their ASCII values, and interpreted as a three-digit number, with the first character being the most significant. So best practice would be to put the version number at the beginning of the filename. This system allows the old versions to remain on the card in case it becomes necessary to reinstall an earlier version (the later version can be marked Hidden and the bootloader will not see it).

SD Card Formatting

The bootloader does not implement any FAT traversal, and cannot follow cluster chains. Cards should therefore be formatted with the largest practical cluster size. The "Formatter" provided by the SD Association formats 2GB and smaller SD cards as FAT 16 with 16K clusters, and larger SDHC cards as FAT32 with 32K clusters. Use of the Formatter is strongly recommended. A card's current cluster size is reported by the CHKDSK function in Windows.

https://www.sdcard.org/downloads/formatter/index.html

The Windows Application

The SDBSL.exe program converts a typical Intel-HEX or TI-TXT file output by a compiler or assembler into a binary file with the correct format for the bootloader. Your source code should be written so that the G2553 application begins execution at the NewMain address C400, and should also set the reset vector at FFFE to NewMain. NewMain for the G2452 is E400. SDBSL will not actually include that reset vector in the binary file, but will check to make sure it's there in the input file, and that something other than FFs are stored at NewMain - it will error out if that's not the case. There should be no content specified for Information Memory because the bootloader only deals with MAIN memory. SDBSL ignores the checksum byte found at the end of each Intel-HEX line. SDBSL is a command line utility with the following format:

SDBSL.exe {Infile} {Outfile} {NewMain}

Update Procedure

1. Compile the new firmware version to a .hex or .txt file.

2. Run SDBSL.exe to produce the corresponding binary output file (a .bin perhaps).

3. Distribute the .bin file to users.

4. The user copies the .bin file to the SD card, inserts the card into the SD port, and powers up the device.

5. The bootloader detects the card, selects the file, and then, segment by segment, it erases flash memory and writes the new contents to it. It also saves its own entry point (C000 or E000) as the Reset Vector. Then it shuts down the processor.

6. The user should remove the card from the port before power cycling the processor. Otherwise the bootloader will install the new firmware again. If it is desireable to leave the card in the device, then there should be a switch or jumper on the card's Vdd line which turns off power to the card except when updating is needed.

Soft Power Switch

A developer may may wish to have his device power up by pushing a momentary button, then have a GPIO pin take over maintaining power and eventually automatically shutting it off. Of course the bootloader has no code to support that, and there may not be enough room left to add it. So it appears the best solution is to have the user simply hold down the momentary switch until updating as completed - about 3 seconds for the G2553, or half that for the G2452. Then the button would be released to turn off the power, at which time the user can remove the SD card.

Other MSP430 Parts

The code presented here should also work with minor changes for other MSP430 parts which have USI or USCI modules. It may also work with modifications for parts with USART modules. However, the examples included here fit into 1K of flash memory with only a few bytes to spare, so it is not certain that any other version would fit within the two-segment space.

The bootloader is written in assembler for the Naken Assembler, which is a free, small, simple command-line assembler written by Michael Kohn. Many thanks to Mike.

https://www.mikekohn.net/micro/naken_asm.php

https://github.com/mikeakohn/naken_asm

Circuit Design

SD cards are 3.3V devices, and are therefore compatible with MSP430 parts. Four GPIO pins are used to connect the processor to the card. Communication is via SPI. For the G2553, the USCI-B0 module was chosen, which uses P1.5, P1.6 and P1.7 for SPI . The additional Card Select ("CS") pin can be any other I/O pin - P1.4 is used in this example. For the G2452, the USI module is used. It uses the same pins, but the functions of the P1.6 and P1.7 data pins are reversed.

The processor's CS pin is connected to the card's Card Detect ("CD") pin. In the beginning, CS is an input pin. The card contains a nominal 50K internal pullup resistor on CD, so the voltage on that pin can be used to detect the presence of the card. To prevent the CS pin from floating when a card is not present, an external 1Meg pulldown resistor must be used. After the card is detected, the CS pin becomes an output that is used when sending commands to the card.

A 47K external pullup resistor is also needed on the card's DataOut pin. Apparently some cards configure this pin as an open-drain output, which must be externally pulled high.

Current drawn by the cards tested here ranged from 10mA for an Ebay $1 no-name 256MB microSD card, to 20mA for a Transcend 4GB microSDHC card. It appears most cards draw 30ma or less during reading (the bootloader does not write to the card).

Below is a diagram showing how the card and processor are connected. The mosfet and LED portion is optional, but if present it will light up for 1/4 second at the beginning of the update process, then light up permanently about three seconds later to indicate that everything updated successfully, including a match on the XOR checksum.

SD CARD BOOTLOADER