August 6, 2018

BootStrap Loader Flashing for MSP430G Value Line Processors Using an Embedded USB-to-Serial Adapter

The Texas Instruments MSP430 processors can be flash programmed in two ways. The most familiar is JTAG programming, which uses a Launchpad or similar programming device. The timing requirements of the JTAG protocol are quite demanding, and in virtually every case another processor is used as the JTAG master to program the ultimate target device.

The second flashing method uses the "BSL" (boot-strap loader) through which the target device participates in a more normal UART-type conversation directly with the PC master, and performs erasing and flashing operations as directed. Most MSP430 parts, but not all, have embedded BSL firmware which is entered through a special triggering pattern transmitted to certain pins. In some chips the BSL code is contained in ROM which cannot be modified, but in others the code is in flash memory. G2xx3 parts such as the G2553 have BSL code in ROM beginning at 0x0C00. The G2xx1 and G2xx2 parts, including the popular G2231 and G2452, contain no BSL at all, but they can be fitted with a "custom BSL".

[Chips with no built-in BSL will be referred to here for simplicity as "G2xx12", which means G2xx1 and G2xx2, and actually includes the G21x1, G22x1, G2x02, G2x12, G2x32 and G2x52 parts. They have 1K to 8K of flash MAIN memory, INFOA calibration data ranging from just the 1 MHz DCO settings found in the G2231 to all four DCO speed settings plus ADC10 values as found in the G2452, but they have no other calibration data. All have TimerA.]

Computers don't have serial ports anymore, but BSL flashing can be carried out using a USB-to-Serial adapter, with its driver installed, so the flashing software on the PC can converse with what pretends to be a COM port, while the target chip communicates with a real COM port provided by the adapter.

The cost of USB-to-Serial adapter parts has dropped to the point that it is now practical to embed one of them, along with a mini- or micro-USB socket, on a project's PC board. That would make it possible to flash firmware revisions to the target chip without a Launchpad. The user would only need a USB cable and the appropriate software installed on his PC. There would be no need to set up any special wiring or jumpers. The adapter and related parts would cost less than $3, whereas a Launchpad, with shipping, would cost around $15. Moreover, all the issues involved with connecting the Launchpad to the project for in-circuit flashing, which can be troubling for non-technical users, would be avoided. The discussion which follows explores the use of an embedded adapter for flashing the G2xx3 and G2xx12 Value Line processors via BSL.

USB-to-Serial Adapters

There are three adapter brands currently in widespread use in the market. The original and most widely used has been the FT232RL from FTDI. It is relatively expensive. At a lower cost are the CP2102 from Silicon Labs and the CH340G from WinChipHead. The CH340G is the least expensive, but it requires a crystal. Overall, it appears the CP2102 seems to be the best choice, and it was used for this project:

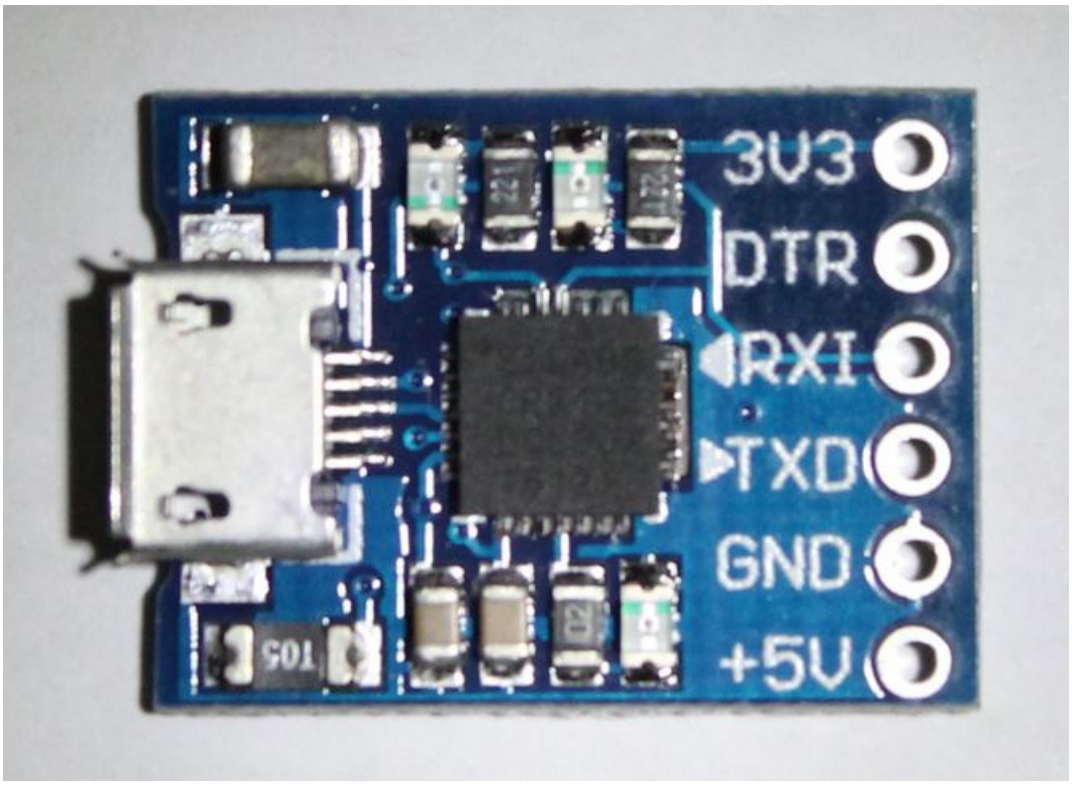
https://www.silabs.com/documents/public/data-sheets/CP2102-9.pdf

http://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers

http://www.silabs.com/documents/public/application-notes/an197.pdf

The CP2102 is available only in a QFN package, but if a through-hole board or hand soldering will be used for a project, that chip is also available on Ebay and elsewhere mounted on a very small module board, with a micro-USB connector at one end, and a six-pin UART header at the other, for around $1.36 each, delivered to the US. The module board can be mounted directly on the project board via matching through-holes. Or if board space is in short supply, the module could overlay other parts of the project circuit. The module could even be electronically connected to the project board via wires, but left lying loose in the battery compartment if there is one, to be used only when flashing is needed. It measures 3/4 inch by 5/8 inch, provides through-hole connectors on the serial side, including one for a 3.3V regulated output that can be used to power the target chip. This module provides DTR, but not RTS. (Also, see the warning at the end of this paper about bad versions of this module.)

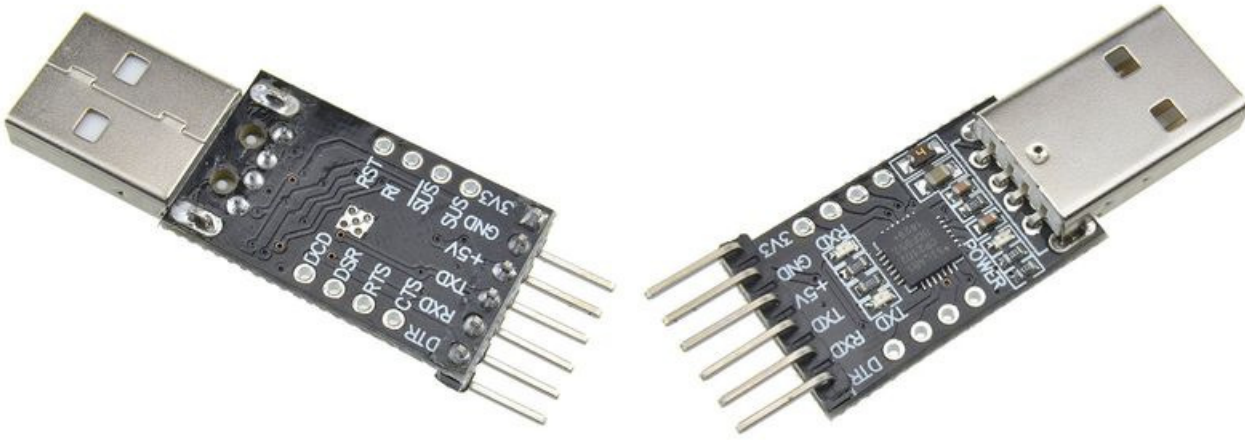
http://www.ebay.com/itm/162217753561



Update: Ebay seller erizh-51 has stopped selling the "good" small CP2102 module. But he says he could get more if a larger quantity is needed. But as of now there is no confirmed source of good modules, at least in small quantities. See the warning on page 14 about "bad" versions of this module, and the fix for converting bad to good.

Another module is larger and would require removal of its full-size USB connector, but it provides the RTS output needed for the special G2xx3 BSL triggering pattern. The existing trace to the +5V output pin would be cut, and a jumper installed from that pin back to the RTS through-hole on the side of the module. The current price is $1.60 delivered to the US.

https://www.ebay.com/itm/182603235304



G2xx3 Problem: The BSL Password

Looking first at TI's ROM-based BSL code in the G2xx3, we find a "feature" which was not well thought out, and which can effectively destroy a target chip if not handled correctly. That feature is the BSL password. Most instructions to the BSL are password protected, and will not be performed unless preceded by transmission of the password by the PC. The password is 32 bytes long, and is equal to the current contents of the interrupt vector table beginning at 0xFFE0. It must be provided in a Ti-TXT format file. If an incorrect password is given, the operation will not be performed - moreover, the entire flash memory of the device, including the calibration data in INFOA, will be automatically erased. It is possible to send an unprotected "mass erase" command, which would set the password to a known state of all FF's, but the mass erase command also erases INFOA. Unfortunately, the "segment erase" command, which permits selective erases, is a protected instruction requiring the password. So without knowing the password, by default it's impossible to do anything without destroying the calibration data first.

The G2xx3 parts have a "key word" located at 0xFFDE, immediately below the interrupt vector table, which adds additional granularity to BSL flashing options. If the value stored there, normally 0xFFFF, is programmed with 0xAA55, the entire BSL process will be disabled. If it contains a null word, then an incorrect password will NOT result in an automatic mass erase. But note that the mass erase command will still erase INFOA. While the key word offers no magic bullet, it certainly would be prudent to program a null word at that location on every application for the G2xx3. It does protect the calibration data from a bad password, but it still doesn't let you do anything useful in BSL without knowing the correct password.

In a situation where the user may be trying out different versions of firmware, or possibly skipping an update version, keeping track of the multitude of interrupt vector instances becomes increasingly difficult, and it becomes more likely that a user will provide the wrong password and be unable to perform the update, or at the worst a soldered through-hole or surface mount processor may be effectively destroyed by having its calibration data erased.

Password Solution: Use an Unchanging Password

For any particular project, one solution to the password feature may be to have an intermediate jump table to which the interrupt vectors always point, no matter what firmware version is installed. The contents of the jump table would change from version to version, but the vectors beginning at 0xFFE0 would be exactly the same for all versions. So the correct password would always be known. To allow for all possible vectors to be used, a jump table 64 bytes long could be defined immediately below the interrupt vectors (a 4-byte Branch instruction for each vector). But if only a few vectors are ever actually used, a much shorter jump table would suffice. So for example, if the reset vector at 0xFFFE is supposed to point to bootup code at 0xC000, then the vector/jump code setup would look like this:

```
.org    0xFFFE      ; reset vector
.dw     0xFFDA      ; points to the jump table entry - this never changes
 .
 .
 .
.org    0xFFDA      ; jump table entry - this location never changes
BR      #0xC000     ; branch to actual code - may change with different verions
```

The advantage of this solution is that the BSL system operates completely normally, with no special handling. Moreover, BSLDEMO2, the Windows program provided by TI to flash G2xx3 via BSL, has the simple command line switch +a to "restore INFOA after mass erase" which reads in the contents of INFOA, performs the mass erase, then writes the data back to INFOA - all using the correct password of course. So with the correct password, you can effectively perform a mass erase of everything but INFOA, which is what you normally want.

Enter BSL After Startup with LOCKA set

Instead of relying on the special triggering pattern on /Reset and Test to initiate BSL activity, it is possible to let the processor boot up in normal mode in special code located down in INFOA memory, then enter BSL if it detects that USB is powered up, or jump to the application if not. This method would not require any connection from the CP2102 to the /Reset or Test pins of the G2xx3, so the smaller module board shown above could be used, and it would already have a micro-USB jack installed.

This option would allow you to enter BSL at a different point, which could let BSL run with LOCKA remaining set, thus protecting INFOA. This would permit using the mass erase command, which converts the interrupt vectors, and hence the password, to all FFs, but would leave INFOA intact. So in effect, you can update firmware without knowing the password. However, leaving LOCKA set also leaves *ALL* INFO segments intact during a mass erase, which may not be desireable.

The BSL code of the G2xx3 is located in ROM beginning at 0x0C00, with vectors to "cold" and "warm" starts. The first instruction of a cold start is to toggle the LOCKA bit. Since the chip powers up with LOCKA set, that toggle instruction actually clears LOCKA, which allows INFOA to be erased. But the special boot code could jump into BSL at the point right after the toggle instruction, thus leaving LOCKA set.

When the G2xx3 powers up, the special boot code first determines whether there is power at the USB socket, which would indicate that it should enter BSL flashing mode. That is done by enabling the pulldown resistor on the pin to be used for receiving serial data (the one connected to the TXD pin of the CP2102, which is normally high when powered up but idle) and then reading the input value on that pin. If that pin is high on boot, it means the USB is connected to the PC, and a BSL session is intended. But if USB is not connected, the CP2102 will be powered down, and cannot drive the pin high. That means the processor must have been booted normally by the user pressing the power-on button, and the special boot code simply jumps to the beginning of the application in MAIN memory.

The special boot code is very short, and fits within the unused space in INFOA. But note that using it would require that the interrupt vector at 0xFFFE must always point to that code at 0x10C0. Moreover, the address the special boot code would jump to if there is no USB would be hard-coded. That means all applications would have to start at the beginning of MAIN memory.

Here is the beginning of the G2xx3's ROM BSL code:

```
0c00: 0c06  .dw 0x0c06                ;Cold start vector
0c02: 0c1e  .dw 0x0c1e                ;Warm start vector
0c04: 3fff  jmp 0x0c04                ;Endless loop

0c06: 40b2  mov.w #0xa540, &FCTL3     ;Cold start: toggle LOCKA, reset LOCK
0c08: a540
0c0a: 012c
0c0c: 90b2  cmp.w #0xaa55, &0xffde    ;check key word - is BSL disabled?
0c0e: aa55
0c10: ffde
0c12: 27ff  jeq 0x0c12                ;yes - infinite loop
0c14: 4031  mov.w #0x0220, SP         ;no - set stack pointer, etc.
0c16: 0220
0c18: 430b  mov.w #0, r11
0c1a: 43c0  mov.b #0, 0x0216
0c1c: f5fa

0c1e: c232  dint  --  bic.w #8, SR    ;Warm start - clear interrupts, etc.
0c20: c0f2  bic.b #0x32, &IE1
0c22: 0032
0c24: 0000
-etc-
```

And here is the special boot code to be flashed into INFOA (G2553 shown here):

```
ColdStart       equ     0x0C00            ; BSL cold start entry vector
BestWarmStart   equ     0x0C0C            ; skips LOCKA toggle and LOCK reset
BSLPIN          equ     0x20              ; BSL entry on P1.5=HIGH (Use pulldown)
MAIN            equ     0xC000            ; the application starts here on G2553
```

```
.org        0x10C0                          ; BSL bootup code. - Bottom of INFOA
Part1:                                      ; reset vector points here

            mov.b   #0,      &P1OUT         ; will be pull-DOWN resistor
            mov.b   #BSLPIN, &P1REN         ; enable resistor
            bit.b   #BSLPIN, &P1IN          ; test current state, pin high invokes BSL
            mov.b   #0,      &P1REN         ; restore P1REN - pd resistor disabled

            jnz     Part2                   ; Pin P1.5 was high - USB is connected

            br      #MAIN                   ; Pin P1.5 was low - jump to app

Part1End:

.org        0x10EC                          ; jump over ADC10 calibration data
Part2:                      ; No room for both Cold and Warm.  Comment out one.

ColdStart:                  ;Comment out this section if using Warm start

            br      &ColdVector             ; enter BSL - indirect address.

WarmStart:                  ;Comment out this section if using Cold start

;            mov.w   #FWKEY,  &FCTL3         ; must clear LOCK, but no change to LOCKA
;            br      #BestWarmStart          ; to leave LOCKA set - direct address

Part2End:
```

This project provides a universal installer for the special boot code for all G2xx3 parts, called "Installer-2xx3-Entry", that determines the chip's MAIN address and installs the boot code in INFOA. Both Cold and Warm versions are provided. This is for those who want to use the small module and forego using the special signaling pattern to enter BSL, and for those who want to run BSL with LOCKA set so they can flash new firmware without knowing the password.

G2xx3 Flashing Software for the PC

BSLDEMO2.EXE is the program provided by TI to flash 1xxx, 2xxx and 4xxx MSP430 parts which have embedded BSL functions. It is found, along with its source code, in the "Deprecated" folder of TI's MSPBSL_Scripter_win.zip. It is a command line utility for Windows with a number of options and switches. The file containing the new firmware to be flashed must be in TI-TXT format, not IntelHEX, as must the file containing the password. The update process would provide a batch file that calls this program with the correct command line options and switches. The user would only have to make sure the COM port is correct in the command line, and that it's set to 8/E/1. In addition, the driver for the CP2102 must be installed. It is available from Silicon Labs.

Unfortunately, v2.01 of this program drives the DTR line at the opposite polarity from what is needed for use with the currently available USB-to-Serial adapters. So if DTR is to be used to drive the special triggering pattern to invoke BSL, or just as a reset source, it will be necessary to use a slightly modified version of the TI software, referred to here as v2.01c of BSLDEMO2.EXE. That version, including its source code, is included with this project. It has an "-i" option which inverts DTR, and a "-j" option which inverts RTS. The "-j" option might be needed for MSP430 parts with dedicated JTAG pins, but is not used with the G2xx3. But the -i option is needed for G2xx3.

Here's the BSLDEMO2 v2.01c help screen:

```
MSP430 Bootstrap Loader Communication Program (Version 2.01c)
BSLDEMO-2.01c [-h][-c{port}][-p{file}][-w][-1][-m{num}][+aecpvruw] {file}


Options:
-h       Shows this help screen.
-c{port} Specifies the communication port to be used (e.g. -cCOM2).
-a{file} Filename of workaround patch (e.g. -aWAROUND.TXT).
-b{file} Filename of complete loader to be loaded into RAM (e.g. -bBSL.TXT).
-e{startnum}
         Erase Segment where address does point to.
-i       Invert polarity of DTR line
-j       Invert polarity of RTS line
-m{num}  Number of mass erase cycles (e.g. -m20).
-p{file} Specifies a TI-TXT file with the interrupt vectors that are
         used as password (e.g. -pINT_VECT.TXT).
-r{startnum} {lennum} {file}
         Read memory from startnum till lennum and write to file as TI.TXT.
         (Values in hex format.)
-s{num}  Changes the baudrate; num=0:9600, 1:19200, 2:38400 (e.g. -s2).
-w       Waits for <ENTER> before closing serial port.
-x       Enable MSP430X Extended Memory support.
-1       Programming and verification is done in one pass through the file.

Program Flow Specifiers [+aecipvruw]
 a       Restore InfoA after mass erase (only with Mass Erase)
 e       Mass Erase
 c       Erase Check by file {file}
 p       Program file {file}
 v       Verify by file {file}
 r       Reset connected MSP430. Starts application.
 u       User Called - Skip entry Sequence.
 w       Wait for <ENTER> before closing serial port.
 Only the specified actions are executed!

Default Program Flow Specifiers (if not explicitly given): +ecpvr
```

BSL for the G2xx12 Parts

The G2xx12 parts have no BSL code built in. But TI provides a primitive custom BSL framework that can be flashed into the chip. It occupies most of INFOD through INFOA, but preserves the calibration bytes near the top of INFOA. The reset/powerup interrupt vector at 0xFFFE always points to the BSL code in INFO memory, which then jumps to the application in MAIN memory if BSL isn't being invoked. There are no passwords to deal with, and flashing is limited to MAIN memory. INFO memory is never touched. For this system to work properly, the new application or update must always start at the beginning of MAIN memory.

If INFO memory is used to save operator settings between sessions, a 512-byte segment of MAIN memory could be used instead for that purpose if available, and erased separately when full. If that's too much space to dedicate to that purpose, then, with some additional overhead, the BSL code can be split so that it begins with 96 bytes in MAIN memory, then branches to INFOC. INFOD could then be used to store settings or do other things. This is referred to below as the "Split" version of the custom BSL, as opposed to the normal "INFO" version.

The G2xx12 Custom Protocol

Space is so limited in most G2xx12 parts that TI reduced the BSL function to doing just one thing. It will flash new firmware to MAIN memory. The BSL expects to receive an image of the entire MAIN memory (**excluding** the reset vector) even if mostly FFs, and it expects the data to be in binary form. The TI protocol has been modified somewhat here to implement syncing between the PC and the G2xx12 at the beginning. Here's what now happens:

1. The G2xx12 powers up, and the reset vector at 0xFFFE points to the BSL code.

2. The BSL enables the pulldown resistor on P1.1 (pin 3), then reads the pin. If it is high, BSL goes into flashing mode. Otherwise, it jumps to the application at MAIN. The application must always start at the start of MAIN memory - with an instruction, not a .db, etc.

3 In flashing mode, the BSL sets up for 9600 baud, 8/N/1, and waits for a byte to be received.

4. The PC begins by toggling the DTR line low, then back high, which resets the G2xx12 if that line is connected to its /Reset pin, but that's not required. The PC then sends some character other than 0xBA. BSL responds with a one-byte code that tells the PC where MAIN memory begins, and whether the INFO or Split version of BSL is installed. Of course it also tells the PC that the G2xx12 is present and the lines are working.

4. The PC then sends the correct Command Byte 0xBA to the G2xx12. Receipt of 0xBA from the PC causes BSL to erase all of MAIN memory, then re-write the reset vector pointing to itself, then wait for the data stream to begin.

5. Meanwhile, after sending the Command Byte, the PC waits a second to allow for the erase to complete, then simply sends the data to be flashed - in binary. As it does that, it calculates an XOR checksum.

6. The PC sends exactly (MAINsize - 2) bytes, excluding the reset vector value which will continue to point to the BSL, followed by a single XOR checksum byte.

7. The BSL writes each byte to flash memory as received beginning at MAIN. There is no buffering.

8. After receipt of the correct number of bytes, the BSL sends back 0xF8 as an ACK if the checksum matches its own calculation, or 0xFE as a NACK if it differs. Then the BSL restarts.

For the Split version, flashing will total (MAINsize - 96 - 2) bytes beginning at (MAIN + 0x60). The 96 bytes of BSL located at MAIN are copied to RAM before the erase of MAIN, and copied back after. Applications must always begin at MAIN + 0x60. The reset vector will actually point to MAIN, which is now the beginning of the BSL code.

Installing either BSL directly using JTAG is not practical because the existing calibration data would be destroyed in the process of erasing INFOA. A better method is to flash to the G2xx12 an application which will itself install BSL, and also develop 8/12/16 MHz calibrations if needed, and that application is flashed to the chip with a Launchpad via JTAG. It brings with it an image of the BSL code. It performs the calibrations, and saves those DCO values in RAM, along with the factory calibration data from INFOA. Then after erasing all of INFO, it copies the BSL image into INFO memory, then copies the saved calibration bytes from RAM back to the top of INFOA. It also erases the top segment of MAIN memory, and writes the correct reset vector at 0xFFFE pointing to BSL. Note that flashing the installer to the G2xx12 does not install BSL. The installer application must run on the first powerup after being flashed, and it performs all of these things at that time. After this "formattng" of the chip, future applications and their updates can be flashed via BSL. Changes to the BSL code itself would have to be done via a revised installer.

Universal installers for the INFO and Split BSL versions, both source code and hex files, are included in this project. The two .hex files work with any of the G2xx12 chips without the need to compile new versions. The idea is that each G2xx12 used in a project would be formatted for this process by flashing the installer .hex file using a Launchpad, then repowering the chip one time to run the installer. If through-hole versions of the G parts are used, then that process could be performed before the chips are installed on the project board. Otherwise, the board would have to provide a JTAG header for formatting purposes. Remember that the 16 MHz calibration requires a Vcc of 3.3V or higher.

G2xx12 Flashing Software for the PC

TI does not provide software for the PC that corresponds to BSLDEMO2.exe, which is not compatible with the G2xx12 protocol presented here. Included in this project is "BSLG2xx12.exe", and its source code. It is a new Windows console application, written in C for the LCC compiler, that implements this project's version of the TI minimalist protocol for G2xx12. Its command line inputs the COM port being used and the filename containing the firmware to be flashed, which can be either IntelHEX or TI-TXT. Use 8/N/1 for this.

```
Get BSL info:      bslg2xx12.exe COMn
Flash firmware:    bslg2xx12.exe COMn filename
```

In preparing the firmware file for use by BSLG2xx12.exe, the reset vector at 0xFFFE should be left at its normal value as though BSL were not involved. In other words, the IntelHEX or TI-TXT file should have the reset vector pointing to MAIN for the INFO version of BSL, or MAIN + 0x60 for the Split version, which is where execution actually begins. The program will not actually write such value at 0xFFFE because the vector there must point to the BSL entry point. But the vector shown in the firmware file will be used by the program to make sure the firmware matches the requirements of the BSL version being used. Note that this is different from using the special boot code in INFOA on the G2xx3 parts, which requires that you specify 0x10C0 as the reset vector in the firmware file. BSLG2xx12.exe will perform error checks on the firmware file, then display each step of the process as the new firmware is written to the target G2xx12 chip.

Calibration

TI only provides calibration for a 1 MHz clock in the G2231 and some other G2xx12 parts. But the higher calibrated frequencies can be derived from that calibrated clock, and a crystal is not required. Steve Gibson came up with the idea several years ago, and the code included here, with his permission, is based on his original code. It has been expanded from the original 8 MHz calibration to include 12 and 16 MHz, and a successive approximation algorithm added that allows the entire process to complete in less than one second. The calibration is done automatically by the installer for any G2xx12 part with only the 1 MHz calibration. The 16 MHz calibration requires a Vcc of at least 3.3V. A Vcc below that will produce unpredictable results.

Circuit Design

This project is about embedding a USB-to-Serial adapter within the project's circuit. However, the more typical option has been to include an adapter as a separate item which is connected through a header only when flashing new firmware is needed. And that certainly remains a viable option. Embedding, however, does provide advantages to the ultimate user, including greater simplicity, and avoiding the risk of losing the adapter or connecting it incorrectly, all at the cost of a bit more complexity in circuit design.

The biggest challenge to embedding results from the fact that all of the adapters tested behave badly when they are powered down. Whether through protection diodes or other internal pathways, all of the adapter's input and output lines sink current when the chip is powered down - as it would be when the USB cable is not connected and the project is running normally. Whether this matters depends on what else is going on in the project circuit, and on choices made by the designer.

For example, if the MCU pins connected to the adapter's TXD and RXD lines are dedicated solely to that purpose, and have no function when the project is operating normally, then those pins can be left as unsed inputs which are conveniently pulled to ground by the adapter when it is off. Similarly, if the project's own power supply is to be used during flashing, then the adapter's 3.3V regulated output need not even be connected to the project. And in the case of the G2xx3, if the special triggering pattern on /Reset and Test is not being used, then the DTR and RTS lines need not be connected either.

But if such simplifying decisions are not feasible, then using an embedded adapter like the CP2102 involves connecting its pins to the project's processor pins in a shared fashion. The processor's pins function in one way in normal operation, but in another way during a BSL session, and both sets of connections remain in place. So the circuit must be designed, and pins selected for various functions, so that when the circuit is operating in one mode, the presence of any other-mode connections to those pins will not interfere.

The Power Source

During a BSL session, the MSP430G device can be powered from the project's normal power supply, or from the 3.3V regulated output of the CP2102. Factors which may affect this decision are the following:

1. The CP2102's TXD, DTR and RTS outputs will all be at 3.3V when high. If the project's power supply normally provides 3.3V or 3.6V, that should work fine. However, if a lower voltage is being used in the project, then the voltage on the CP2102's outputs would exceed the permitted voltage on the processor's input pins. In that case, the processor would need to be temporarily powered by the higher 3.3V of the CP2102 for BSL sessions.

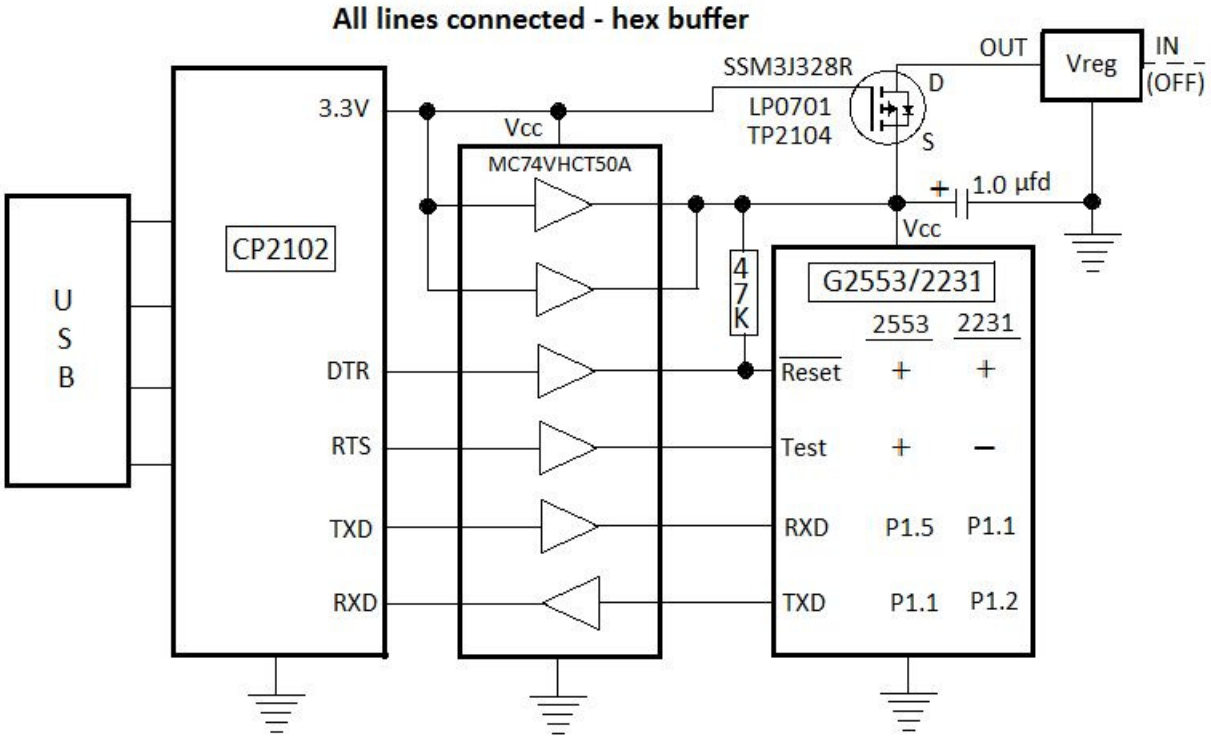
2. If the project is powered down when the active USB cable is plugged in and the adapter begins providing power to the processor, the custom BSL code will automatically detect that its RXD pin is high when it comes out of reset, and it will begin a BSL session. But if the project's own power is to be used, then it must be powered up before USB is connected, and there must be some way to initiate a reset to begin the BSL session after USB is connected - a reset button of some kind, or possibly connecting DTR to /Reset via a buffer or R/C circuit, and allowing the PC software to trigger a reset. Both BSL2xx12.exe and BSLDEMO-2.01c.exe perform such a DTR reset pulse.

3. As mentioned above, when the CP2102 is powered down, its 3.3V regulated output pin will sink significant current. But the same may be true of the project's voltage regulator - depending on what is on the other side of that regulator. If there is nothing on the IN pin of the regulator but a battery and an On/Off switch, then there should not be a problem. But if the IN pin is connected to other circuitry in the project, then excessive current may flow back through the regulator when power is supplied from the CP2102 and normal project power is off.

(Note that all of these issues apply even if a non-embedded adapter is temporarily connected via a header.)

CP2102 Isolation when Powered Down

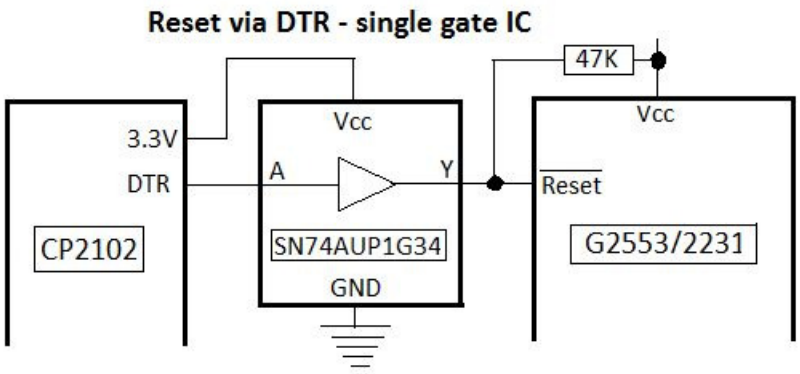
While it is certainly possible to use diodes to prevent the CP2102 pins from sinking current when it is powered down, at some point the number of diodes and pullup resistors, and in particular the difficulty of dealing with the DTR-to-/Reset connection through a diode, argue for a simpler solution. Below is a schematic showing the use of an MC74VHCT50A hex buffer to isolate the CP2102 when it is powered down:



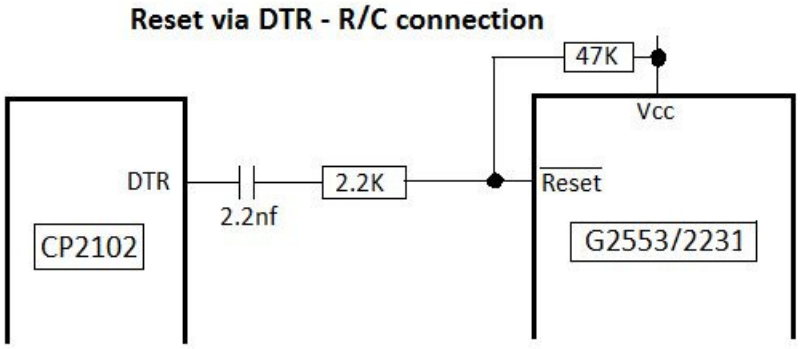
The buffer is a 40-cent item at Digikey, and comes in either SOIC, which can be manually soldered, or TSSOP. When it is powered down, all of its inputs and outputs become high-Z, and thus have no effect on the project circuit. Also shown is the use of a P-channel MOSFET in reverse orientation to block any current from flowing back through the project's regulator, with near-zero voltage drop in normal operation. Of course a diode on the IN line of the regulator may work just as well if the voltage drop is not material, and neither may be needed if measurements show the backwards current flow is minimal to start with.

The schematic presents pretty much the worst case of what may need to be done to embed the adapter if all the lines are used and the adapter provides power to the processor. Even so, it's just the adapter module and the buffer, and possibly a MOSFET or diode, most of which would also be needed even if the adapter isn't embedded.

If the special signalling pattern on /Reset and Test is not being used, but the processor needs to be reset from the PC software, the DTR line can be connected to the processor's /Reset pin using either of two methods. The first uses a single-gate version of the hex buffer shown above. The SN74AUP1G34 is a 36-cent item at Digikey, and comes in several packages, the largest being SOT 23-5, which can be hand-soldered.



The second way is to use a simple R/C circuit. The capacitor provides DC isolation, but allows a very brief reset pulse when DTR goes low. The 2.2K resistor is needed to limit current when DTR goes back high.

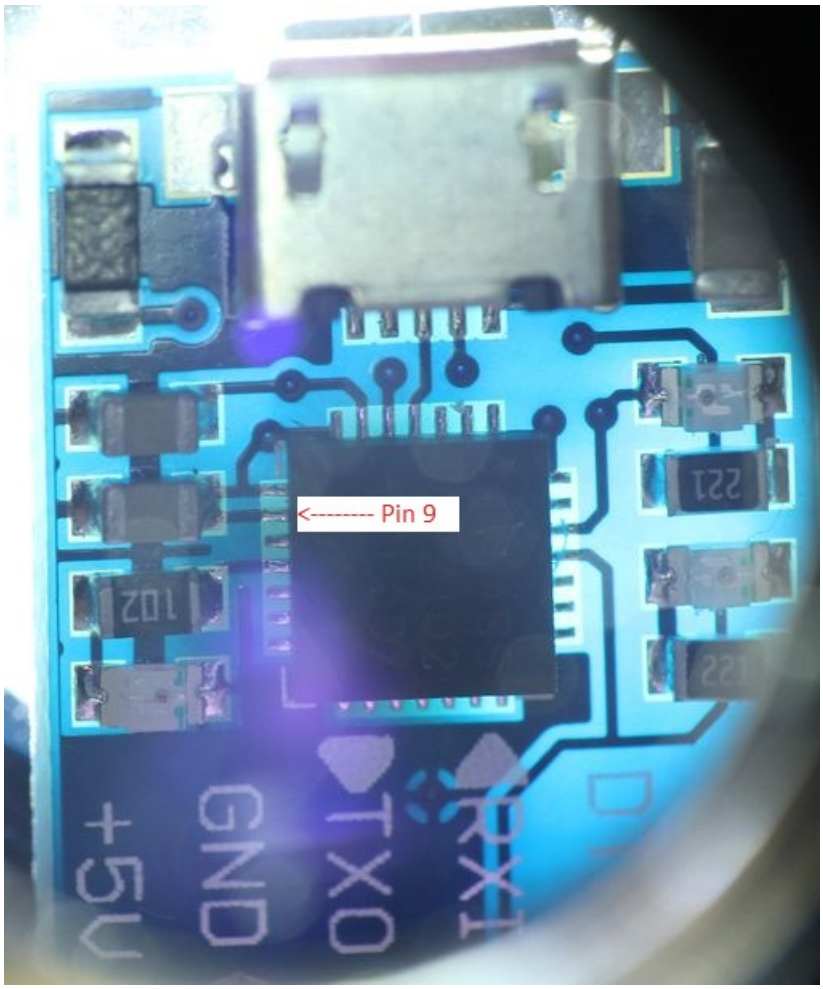


Warning About "CJMCU" and Other Adapter Modules

The small adapter module with micro-USB referred to at the beginning comes in two versions, one good and one bad. The bad one, sold by Banggood and others, outputs 4.3V at the "3.3V" output pin and on DTR and TXD. This voltage exceeds the Absolute Maximum ratings for MSP430G, so this version must not be used. This version may have the TXD output labeled as "TXO", and have a "000" jumper installed next to the USB connector. Modules marked "CJMCU" are all bad, but not all bad modules have that marking.

The good version outputs 3.3V everywhere, calls TXD "TXD", and may have a resettable fuse marked "T05" next to the USB connector. Despite the pictures shown in the linked Ebay listing, the items shipped by Ebay seller erizh-51 were actually the good version, and he claims to have access to more of that version from his supplier. Just be sure to measure the "3.3V" output voltage of every module before use.

If a module with 4.3V outputs is received, it can be converted to a "good" module with 3.3V ouputs by very carefully cutting the trace connected to pin 9 of the CP2102. That pin should be left floating.



Files Included

This project includes the following executables and documentation:

1. Installers (Cold and Warm) for the G2xx3 special Entry code in INFOA, with source code.
2. Installer for the "INFO" version of the G2xx12 BSL code, with source code.
3. Installer for the "Split" version of the G2xx12 BSL code, with source code.
4. BSLG2xx12.exe for Windows console for the G2xx12 protocol, with C source.
5. BSLDEMO-2.01c.exe for the G2xx3, with C source - allows inversion of DTR line.
6. HEX2TXT.vbs for Windows - converts IntelHEX to TI-TXT needed for BSLDEMO2.
7. An example showing use of an intermediate jump table for the G2553 interrupt vectors.
8. MSP430G BSL with Embedded Adapter.pdf (this file).
9. README.txt - brief description of the project and its files.

The source code files for the various MSP430 executables contain extended explanations, instructions and comments. BSLG2xx12.exe requires a command line - run with -h to see it.

In addition, you will need the Windows driver for the CP2102, available from Silicon Labs, or the equivalent driver for whatever other USB adapter you use:

http://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers

Conclusion

Experience has shown that asking non-tech users of a project to connect the circuit to the Launchpad to flash new firmware, or even to move the chip over to the Launchpad, is asking for trouble. And there is the cost of the Launchpad itself. USB-to-serial interface chips are now so inexpensive that it makes sense to embed BSL capability in projects for flashing firmware updates. But there can be pitfalls in the BSL process, so careful planning and testing is required.

Note: A special thanks to Michael Kohn, the author of the NAKEN ASSEMBLER used for all MSP430 code in this project. It is a very small, extremely fast assembler for MSP430 and for 31 other processors/microprocessors. The assembler, along with many other wonderful things, can be found at Mike's website and Github repositories:

https://www.mikekohn.net/
https://github.com/mikeakohn