Comprehensive Guide to BSLs

Luis Reynoso

MSP430 Applications

June, 2014

Agenda

- Introduction to BSL
- BSL Ecosystem
  - ROM BSL
  - Flash BSL
  - vBoot
  - vBSL
- Using the Factory BSL
  - SW tools
  - HW tools

Introduction to BSL

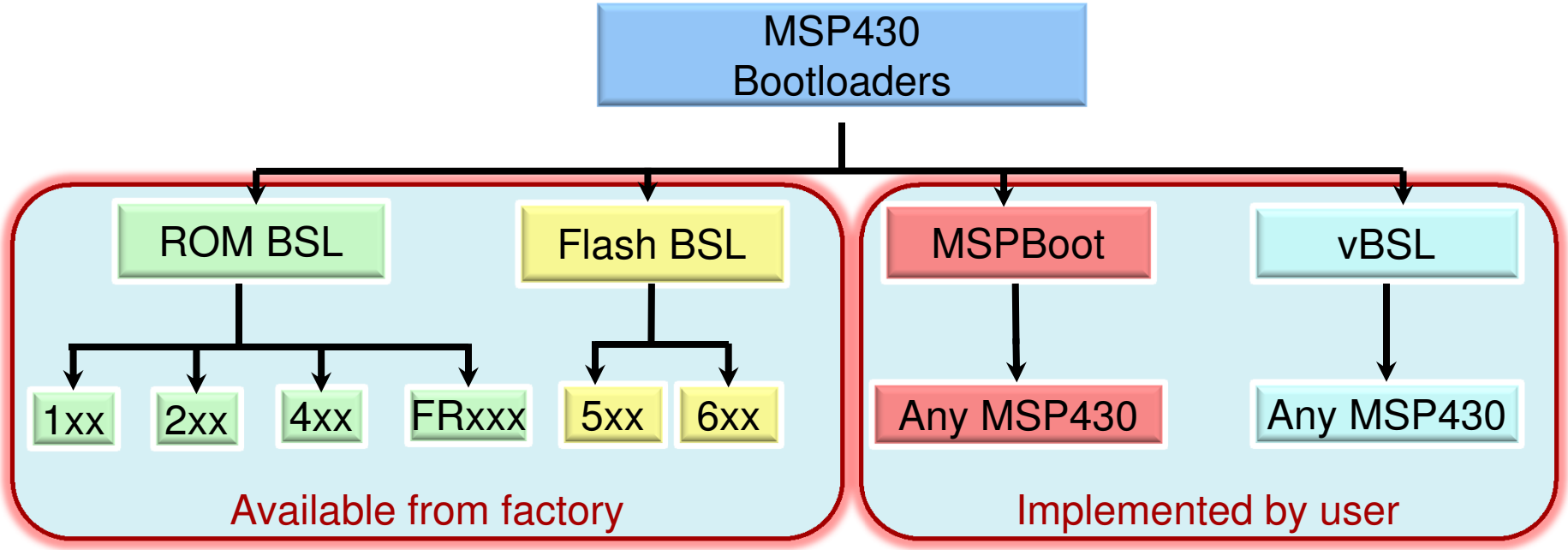
What is a bootloader?

- An application which executes before the main program
- Typically used to allow field firmware updates, but can also be used for device initialization, self-check, etc.
- Uses communication interfaces which are common and easier to implement from the host side

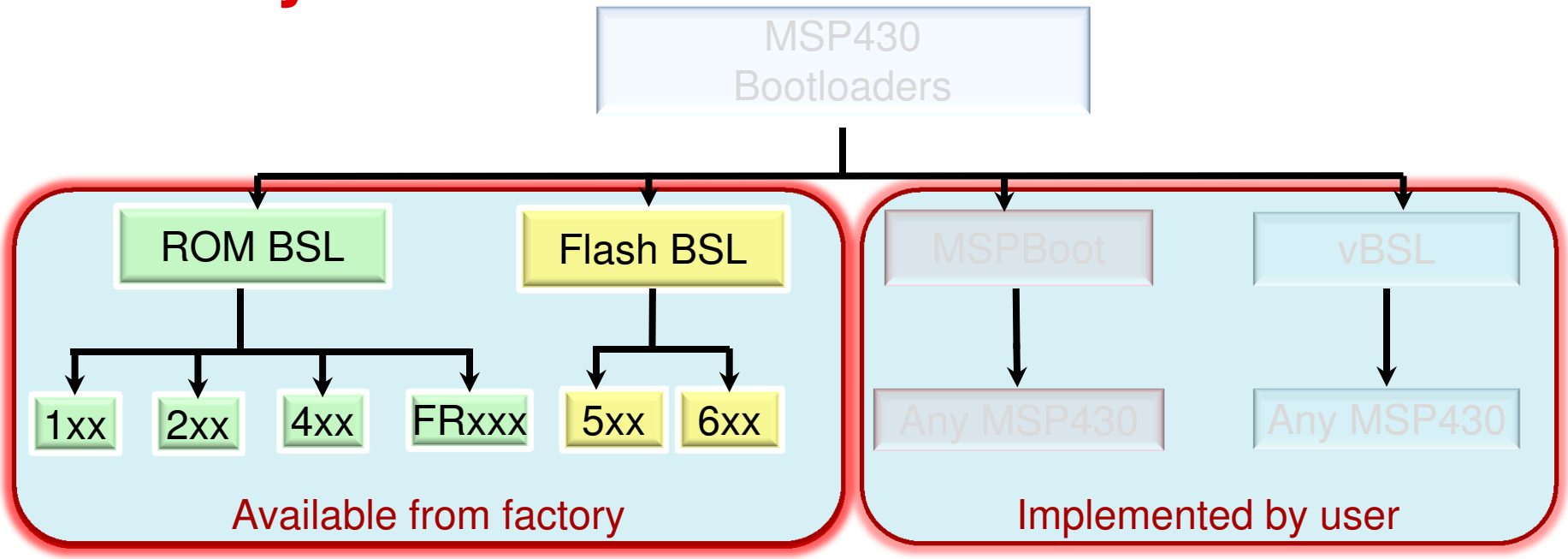
Advantages of using a bootloader

- Production-line programming can use common interfaces like UART, USB or I2C
- Fewer connectors and external signals: the same interface used for application can be used to program the device
- Problems discovered after product release can be fixed
- Reduced need for tech support, because problems can often be solved by instructing the user to upgrade firmware
- Product returns can be reduced
- End users have a more positive experience with the product

Bootloaders for different MSP430 families



Factory BSL


MSP430
Bootloaders
ROM BSL
Flash BSL
MSPBoot
vBSL
1xx
2xx
4xx
FRxxx
5xx
6xx
Any MSP430
Any MSP430
Available from factory
Implemented by user

Factory BSL

The Factory MSP430 BSL is:

- ✓A TI supplied bootstrap loader which provides a method to program the MSP430 during development and in the field
- ✓Designed to protect customer's IP from read-out
- ✓Designed to allow reading and writing to MSP430 memory
- ✓Always available, when device is blank or corrupted
- ✓Usable with little or no external peripheral requirements
- ✓Source available for some derivatives…customers can customize their own

Factory BSL

The Factory MSP430 BSL is NOT:

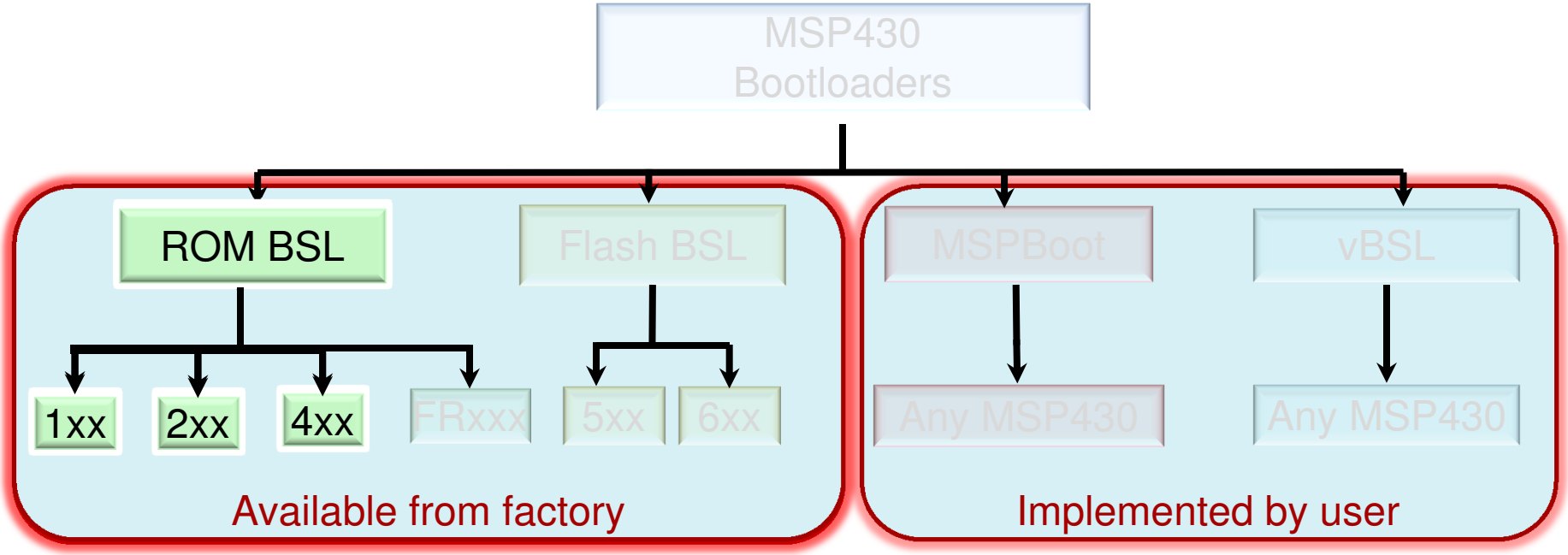
- The same between MSP430 families
- Designed to stop foreign code from being written
- Resource unlimited

Factory BSL - Security

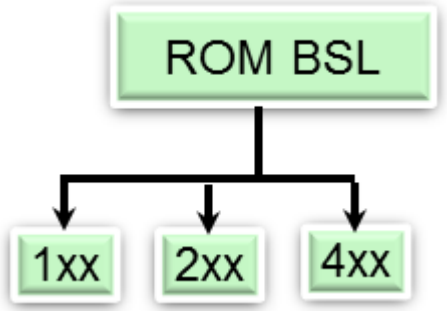
- Access to the BSL is granted via a BSL Password
  - 32 byte password is contents of 0xFFE0 to 0xFFFF
  - Some commands (such as MassErase) are available without the password
  - See SLAU319 for additional details on which commands require password

- BSL access is not impacted by the JTAG Fuse status
  - To render a device inaccessible, disable the BSL and blow the JTAG Fuse

ROM BSL- 1xx-2xx-4xx

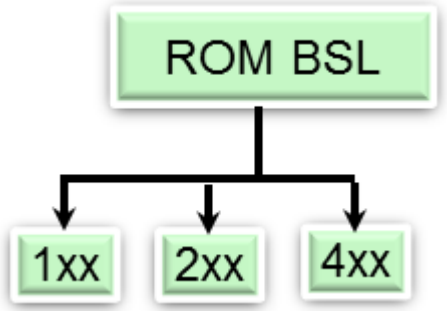


ROM BSL- 1xx-2xx-4xx



- Original Bootstrap Loader (BSL)
- UART based
- Hardware entry sequence is fixed
- Same protocol but different versions and capabilities depending on MSP430 derivative
- Cannot be modified, but patches can be applied to fix known bugs or upgrade BSL version
- Can be called from application software
- Not available in all devices (i.e. G2xx1/G2xx2)
- Can be disabled in some versions

ROM BSL- 1xx-2xx-4xx- Common issues/questions



- Entry sequence is different between devices with dedicated vs shared JTAG:

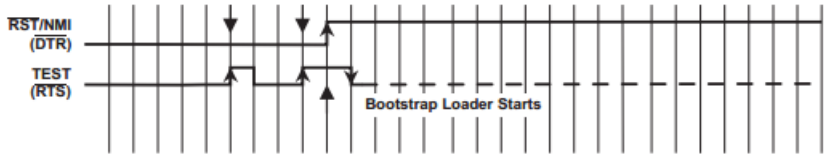


Figure 1-2. BSL Entry Sequence at Shared JTAG Pins

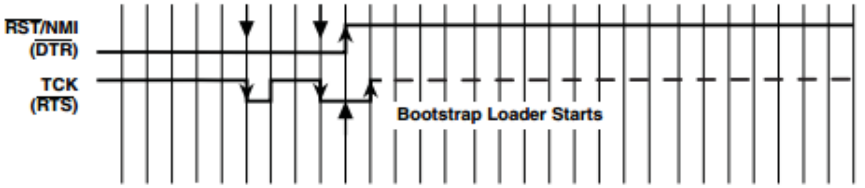
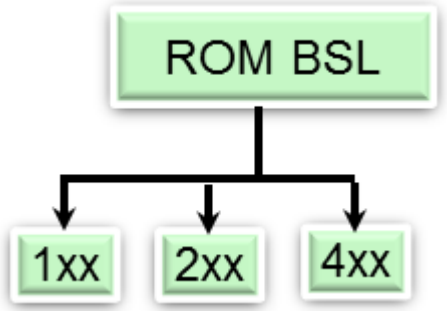


Figure 1-3. BSL Entry Sequence at Dedicated JTAG Pins

- Initial baudrate is fixed to 9600bps and following format:
  - Start bit, 8-bits (LSB first), **even** parity, 1 stop bit
- A SYNC character (0x80) must be sent before any and every command
- Handshake is performed by an acknowledge character:
  - A DATA_ACK (0x90) is sent back to confirm successful reception and execution
  - DATA_NACK (0xA0) indicates an error

ROM BSL- 1xx-2xx-4xx- More Common issues/questions



- Some commands (i.e. BSL Version, Change Baudrate) are not supported in some versions

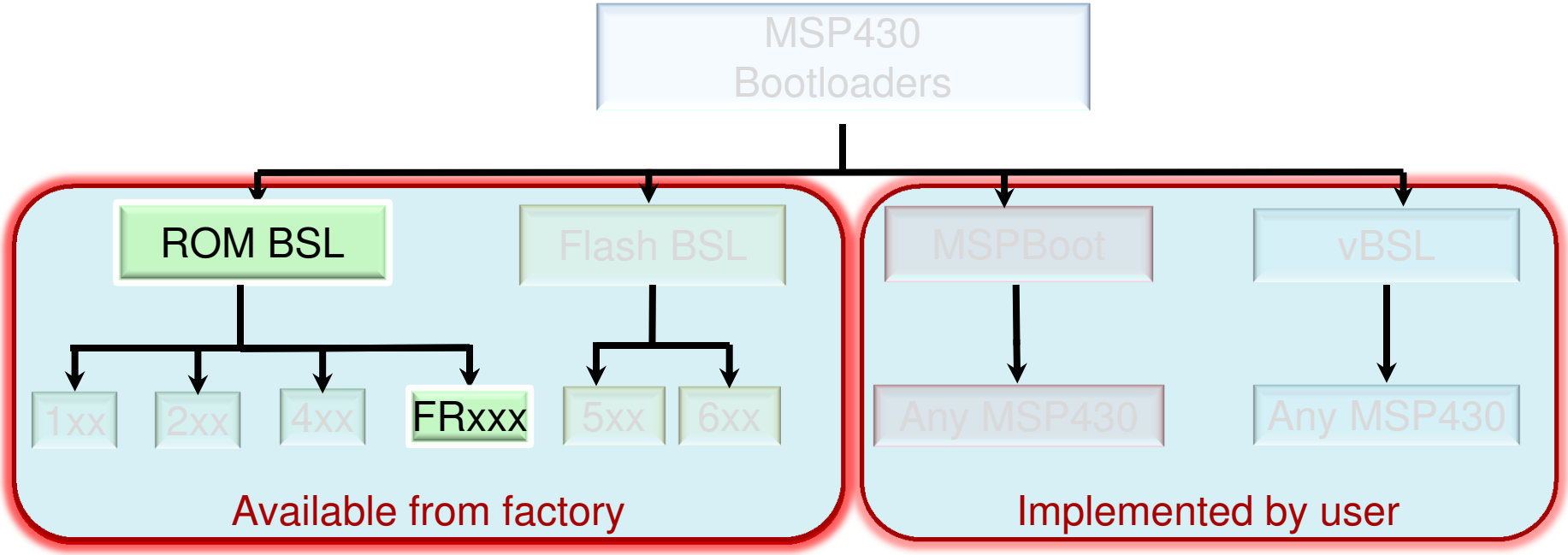
- BSL can be patched but some versions **must** be patched to eliminate ROM bugs

From SLAU319:

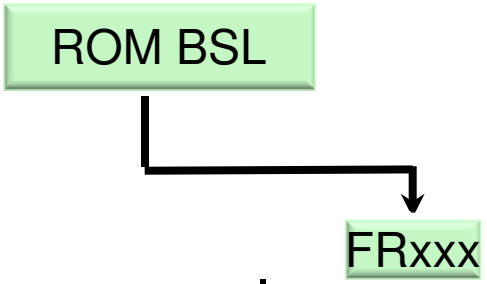
Device	F13x F14x(1) up to Rev N	F11x (obsolete) F11x1 (obsolete)
BSL Version	1.10	
Comment 1 Workaround mandatory	Load PATCH.TXT to eliminate ROM bug (see Section 5.2 and Section 2.5).	
Comment 2 Optional for F148, F149 only: Use loadable BSL (>1 KB RAM required)	Load BL_150S_14x.txt to get all features of V1.60 plus valid erase segment command (see Section 2.5).	
Comment 3 Optional for F1x4 to F1x9: Use small loadable BSL (<512B RAM required)	Load BS_150S_14x.txt to get some features of V1.60 (see Section 2.5).	

- The UART BSL is implemented as a Timer-based UART. This often means that the pins used for UART are not in the same location as Hardware UART (i.e. USCI RXD/TXD)

ROM BSL- FR5xx

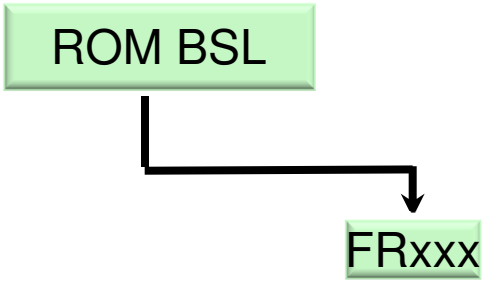


ROM BSL- FR5xx



- UART-based, except for some derivatives (for example, MSP430FR59xx1) which have I2C-based BSLs
- Hardware entry sequence is fixed (same as ROM BSL)
- **Same protocol as Flash BSL**
- Cannot be modified, but can be patched
- Can be called from application software

ROM BSL- FR5xx- Common issues/questions



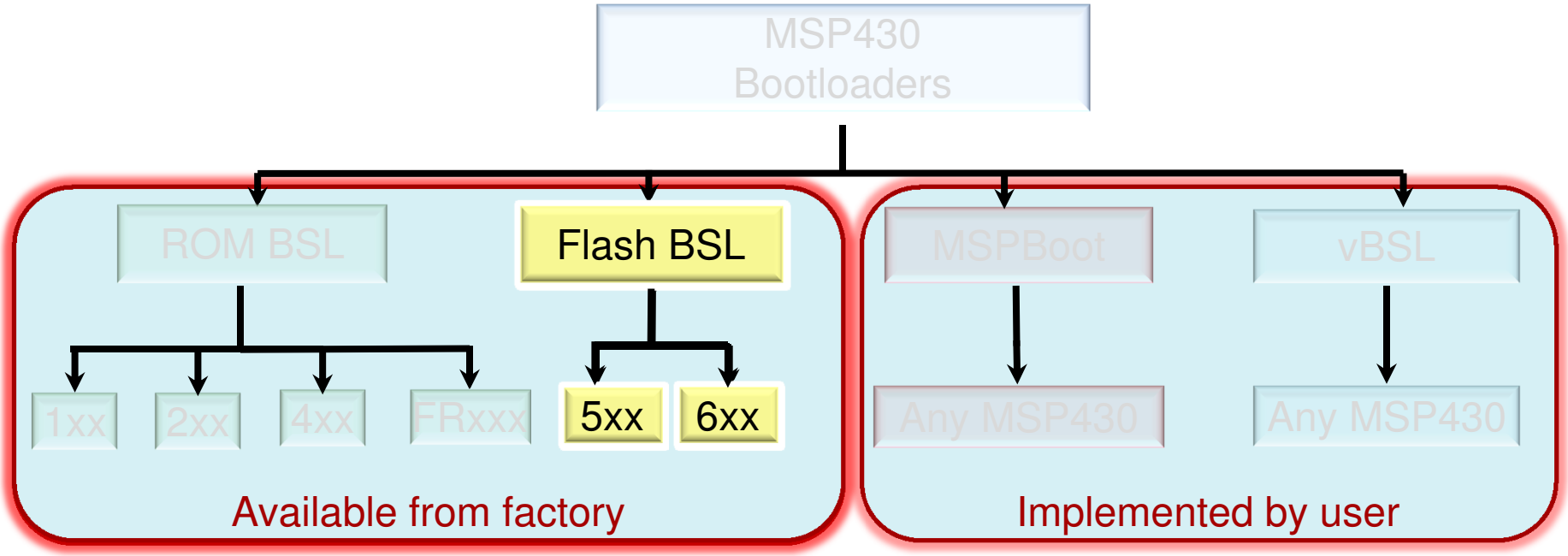
- The protocol for FR5xx BSL is the same as F5xx, not the 1xx-4xx ROM BSL
- The datasheet shows which devices support I2C BSL or UART BSL:

Device	FRAM (KB)	SRAM (KB)	Clock System	ADC12_B	Comp_E	Timer_A[3]	Timer_B[4]	eUSCI A[5]	eUSCI B[6]	AES	BSL	I/O	Package Type
MSP430FR5969	64	2	DCO **HFXT LFXT**	16 ext, 2 int ch.	16 ch.	3, 3[7] 2, 2[8]	7	2	1	yes	UART	40	48 RGZ
MSP430FR59691	64	2	DCO **HFXT LFXT**	16 ext, 2 int ch.	16 ch.	3, 3[7] 2, 2[8]	7	2	1	yes	I2C	40	48 RGZ

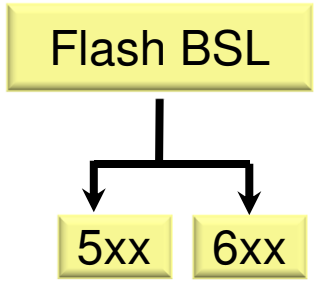
- Different pins are used for UART vs I2C
- MSP430FR59xx has Tags in TLV showing BSL information:

BSL Configuration	TAG	1Ch
	Length	Depends on the BSL_COM_IF value (actual: 02h for UART or I2C)
	Low Byte	BSL_COM_IF
	High Byte	BSL_CIF_CONFIG[0]
	Low Byte	BSL_CIF_CONFIG[1] (optional)
	High Byte	BSL_CIF_CONFIG[2] (optional)
	Low Byte	BSL_CIF_CONFIG[3] (optional)
	High Byte	BSL_CIF_CONFIG[4] (optional)
	⋮	⋮
	⋮	⋮
	High Byte	BSL_CIF_CONFIG[n] (optional)

Flash BSL

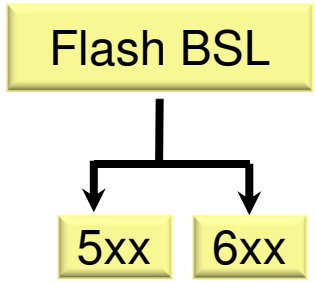

MSP430
Bootloaders
ROM BSL
Flash BSL
MSPBoot
vBSL
1xx
2xx
4xx
FRxxx
5xx
6xx
Any MSP430
Any MSP430
Available from factory
Implemented by user

Flash BSL



- Resides in a reserved, protected Flash area (2KB)
- Default entry sequence depends on BSL version
- Source code is available to allow for customizations
- Resource limited
- Can be disabled
- Uses UART, I2C or USB by default depending on the device
- Devices with dual voltage rails can use the default DVCC-based BSL, or a DVIO-based BSL.

Flash BSL - Customization



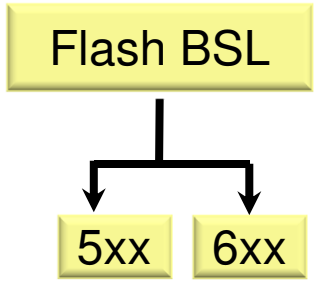
- Flash BSLs in F5xx/6xx devices can be over-written with a custom BSL

- BSL code space is limited
  - 2KB BSL space on F5xx/6xx BSLs
  - New features implemented must be limited to fit into this space
  - Typically requires a tradeoff between feature set and code size

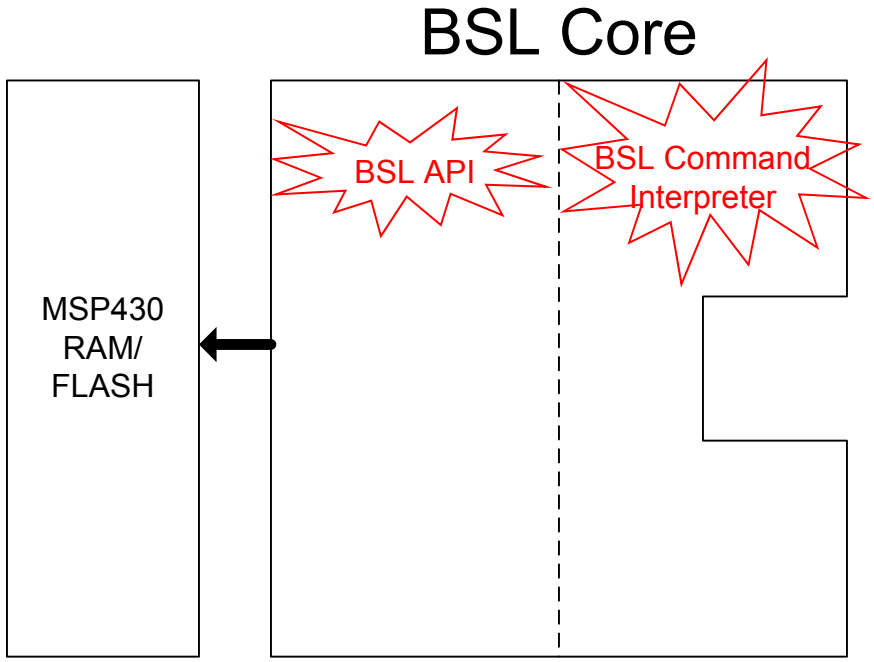
- Application Note SLAA450 contains additional information on customizing a Flash-based BSL
  - http://www.ti.com/lit/pdf/slaa450\

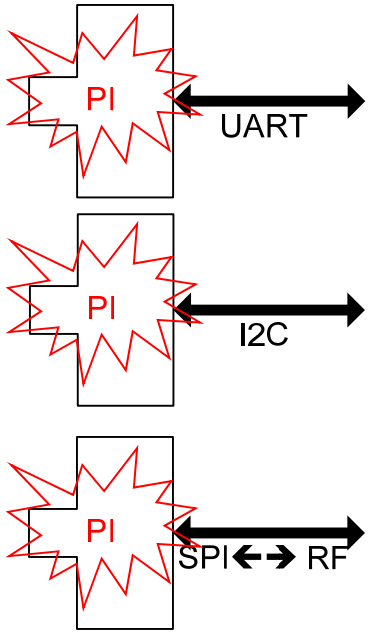
- Project available in IAR (kickstart edition), but a CCS example is included

The Modular Structure of BSL

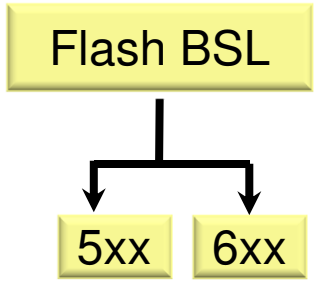


- The TI supplied Flash BSLs are very modular
- Makes customization easier as files can be reused or are easily replaced
- The 3 main sections of the BSL code are:





Flash BSL – Entry sequence



- The default entry sequence for non-USB BSLs is the same as ROM BSL
  - Software checks a flag (SYSCTL.SYSBSLIND)
- The entry sequence for DVIO-based BSL is different:

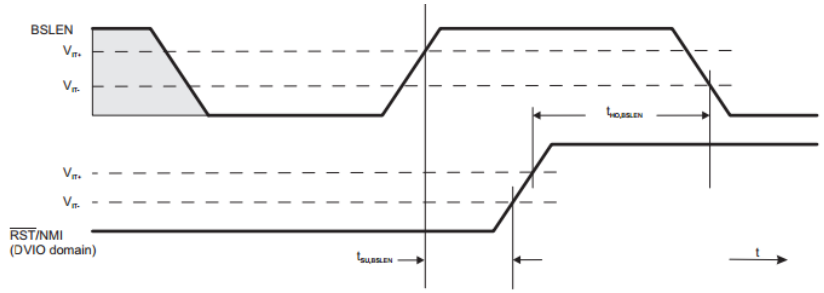
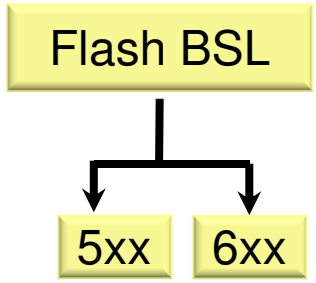


Figure 5-18. DVIO BSL Entry Timing

- The entry sequence for USB BSL requires:
  - Device is powered by USB AND reset vector is blank; OR
  - The device powers up with PUR tied to VUSB
- The entry sequence can be customized as needed. Some examples:
  - Check a GPIO
  - Check Reset vector
  - Validate the application by checking its CRC

Flash BSL – Peripheral Interface

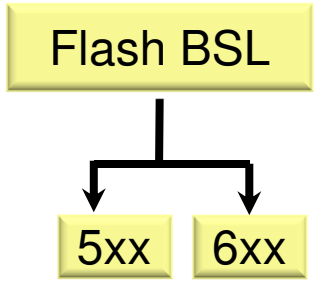


- Non-USB devices are typically shipped with UART
  - Dual-rail devices are shipped with DVCC-based BSL
  - Typically, the UART BSL is implemented as a Timer-based UART. This often means that the pins used for BSL are not in the same location as the Hardware UART (i.e. USCI RXD, TXD)
  - **Some devices (i.e. MSP430F5259) are shipped with I2C BSL**

- USB devices are shipped with USB BSL
  - Due to the larger size of the USB stack, the Flash BSL contains a smaller set of instructions
  - This BSL is used to download a fully-functional BSL to RAM

- Some possible customizations include:
  - USCI-based UART (available in SLAA450)
  - I2C (available in SLAA450)
  - SPI Master or SPI Slave
  - RF

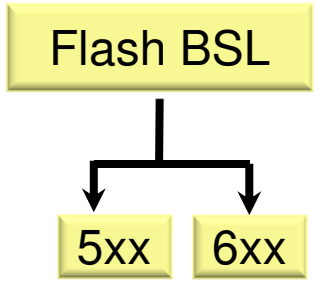
Flash BSL – Command Interpreter



- The Flash BSL includes several protected and unprotected commands, such as:
  - Mass Erase
  - RX Data Block (Write data to memory)
  - TX Data Block (Read data from memory)
  - CRC Check
  - Load PC
  - etc.

- Some possible customizations include:
  - Reset Device (force a BOR)
  - Disable Read operations
  - Change default password (i.e. use a predefined one instead of interrupt vectors)
  - Avoid mass erasing device on incorrect password

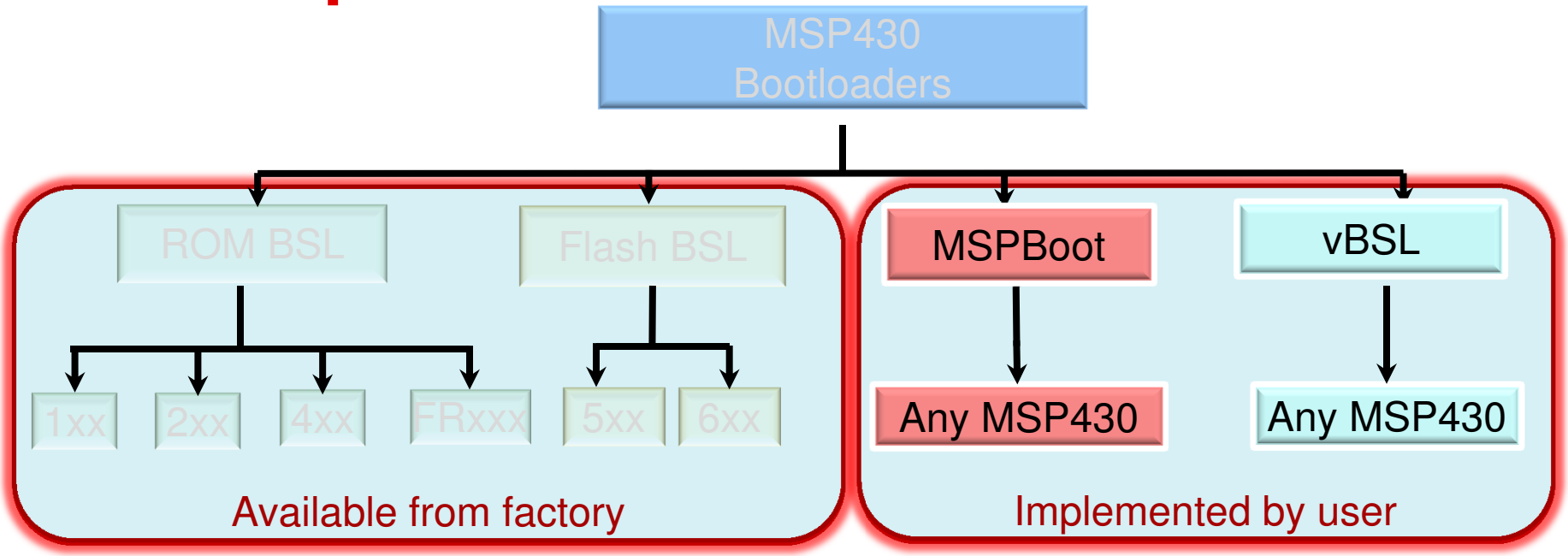
Flash BSL – API



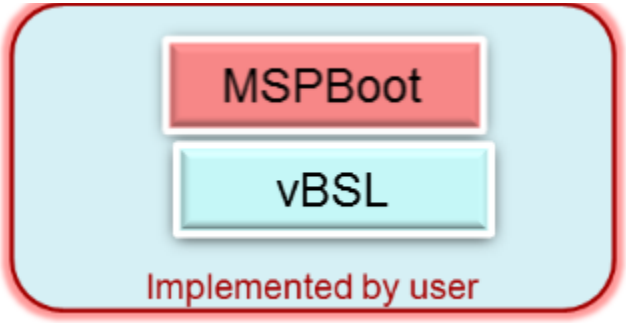
- The BSL API module takes care of reading/writing memory and calculating CRC

- Some possible customizations include:
  - Implement a dual-image approach
  - Prevent read/writes to some areas
  - Decrypt data from host

User-implemented BSLs



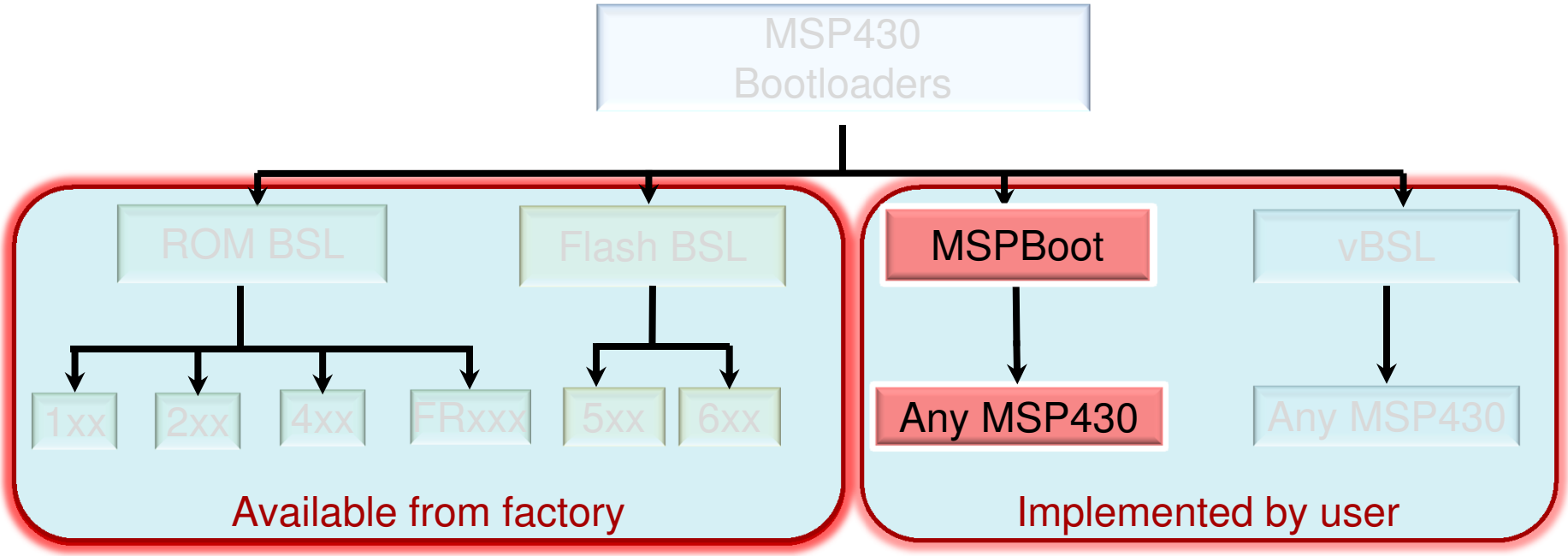
User-implemented bootloaders



- In some cases, the factory BSL is not an option:
  - Some devices don't have a factory BSL
  - ROM BSL can't be customized
  - Flash BSL has a limit of 2KB

- MSPBoot and vBSL are two options which can be implemented in practically any MSP430
- Both options have a small footprint but they consume user space (info memory or main memory)
- There's no hardware protection for main memory

MSPBoot


MSP430
Bootloaders
ROM BSL
Flash BSL
MSPBoot
vBSL
1xx
2xx
4xx
FRxxx
5xx
6xx
Any MSP430
Any MSP430
Available from factory
Implemented by user

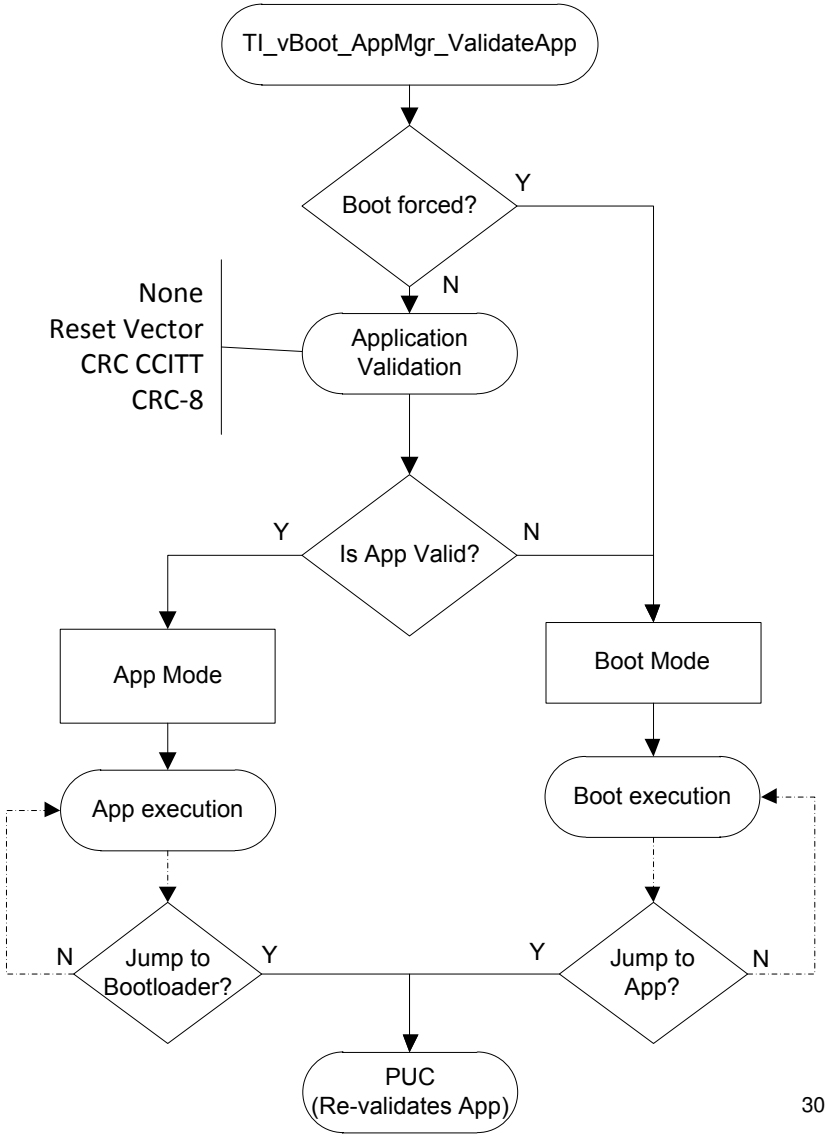
MSPBoot

MSPBoot is an main-memory resident bootloader for MSP430

- Small footprint typically from 1 to 3 Flash sectors
  - Footprint can grow as needed
- Very flexible/customizable
  - Configurable entry sequence
  - Optional Dual-image support
  - Optional CRC check of application
  - Optional support for SMBus
- Capability to debug MSPBoot with the rest of the application
- Currently implemented on G2xx and FR57xx and using UART, I2C or SPI Slave
  - Easily portable to any device or peripheral
- Bootloader is software-protected
  - Can't protect Bootloader area from accidental corruption generated by application
- Host and application examples provided
- Projects available for IAR and CCS

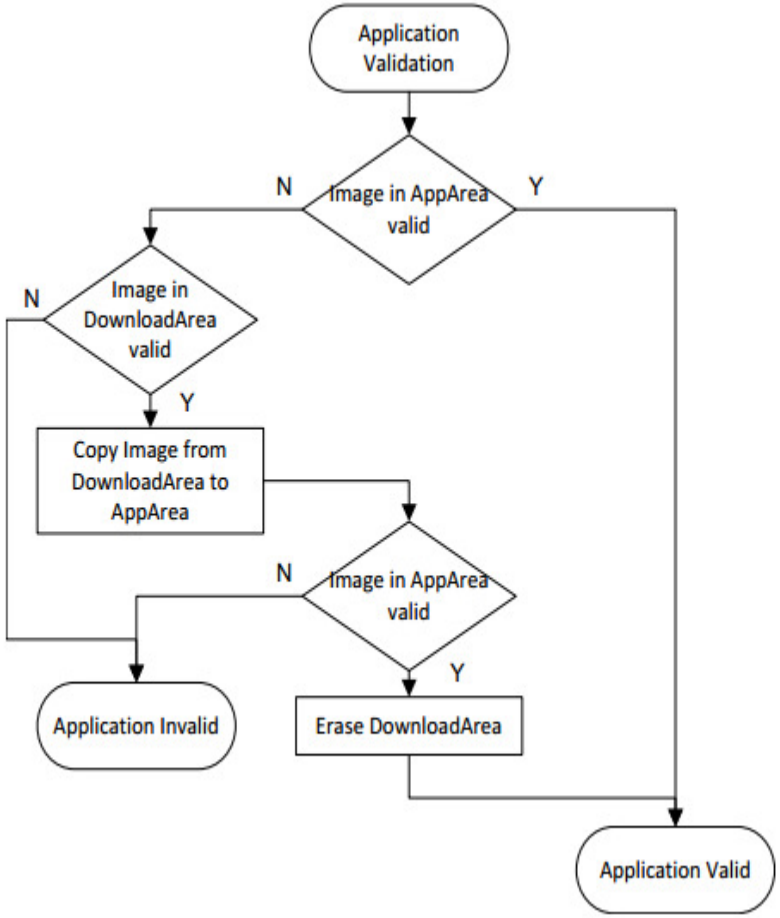
MSPBoot – entry sequence

- After reset, MSPBoot determines if bootloader or application should run
- Bootloader can be forced
  - Entry sequence
  - Called by application
- Bootloader runs if application is not valid
  - Reset vector invalid
  - CRC CCITT checksum fails
  - CRC-8 checksum fails

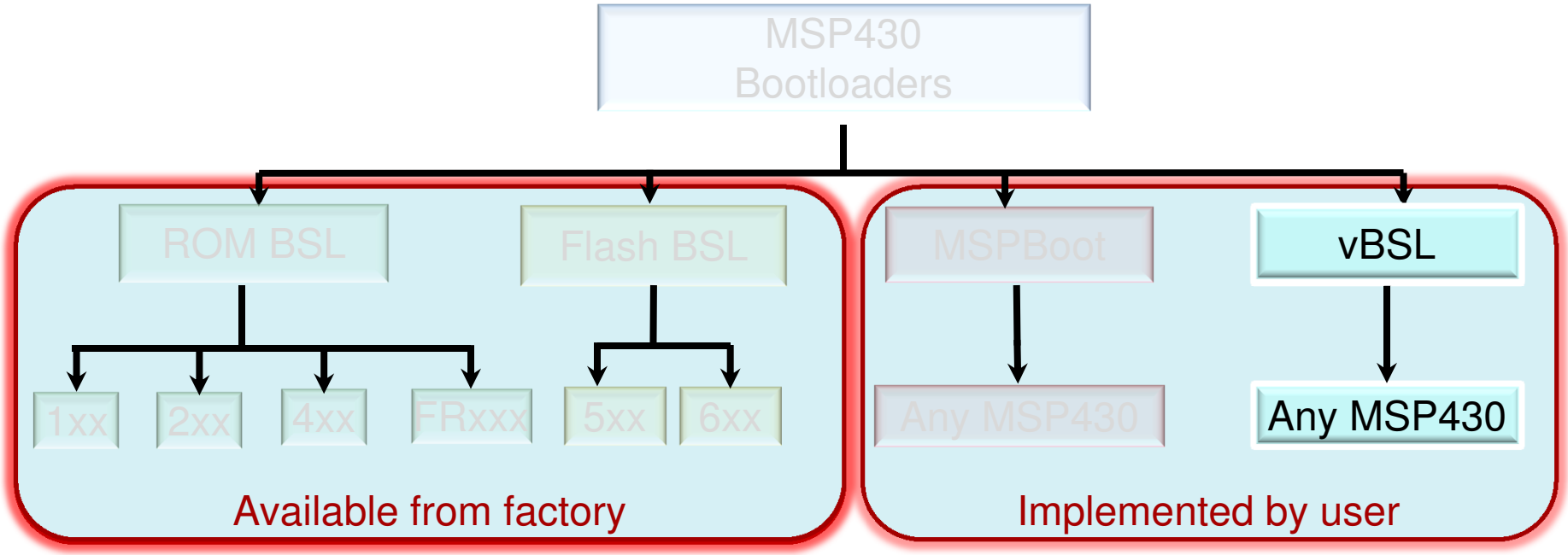


MSPBoot – dual image

- MSPBoot supports dual-image (aka image-copier)
- Code always run from the same “App Area”
- Updates are downloaded to a different “Download Area”
- Host can’t write directly to “App Area”
- After an initial download, a valid image is expected to be available always



vBSL


MSP430
Bootloaders
ROM BSL
Flash BSL
MSPBoot
vBSL
1xx
2xx
4xx
FRxxx
5xx
6xx
Any MSP430
Any MSP430
Available from factory
Implemented by user

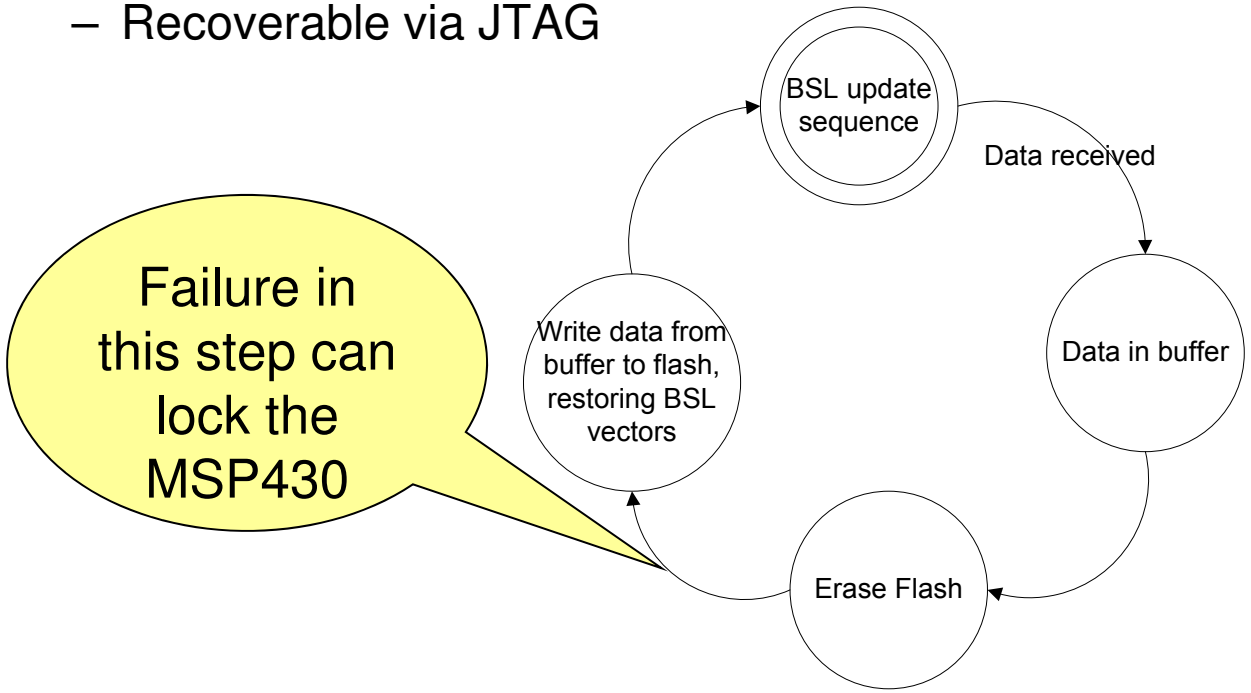
vBSL

The vBSL is a simple Bootloader targeted for the smallest G2xx value line devices

- Resides in Info Memory (only 256 Bytes)
- Customizable
- UART Based (software UART using TA0)
- Not compatible with other BSLs
- Limited functionality and commands
- Hardware entry sequence can be customized (i.e. push button)
- Uses main flash to store TA0CCR0 and Reset vectors
- The device can potentially get locked
- Projects available in CCS and IAR
- See AppNote slaa450 for more details
  - http://www.ti.com/lit/pdf/slaa450

vBSL

- vBSL is extremely space constrained- 256 bytes
- vBSL needs to erase flash vectors, which erases Reset and TA0CCR0 vectors
- Failure during this step (i.e. power cycle) will lock the MCU since the device can't enter bootloader and there is no user code
  - Recoverable via JTAG



Comparison Between Bootloaders

BSL Type	Available from Factory	Customizable	Location	BSL Area Protected	Default HW Entry	SW Entry
ROM	Yes	No	Dedicated ROM Block	Yes	TEST/RST or TCK/RST	Yes
Flash	Yes	Yes	Dedicated BSL Flash	Yes	Depends on device	Yes
vBSL	No	Yes (limited)	Info Flash	Yes (limited)	GPIO on reset	No
MSPBoot	No	Yes	Main Flash	No	GPIO on reset or invalid application	Yes

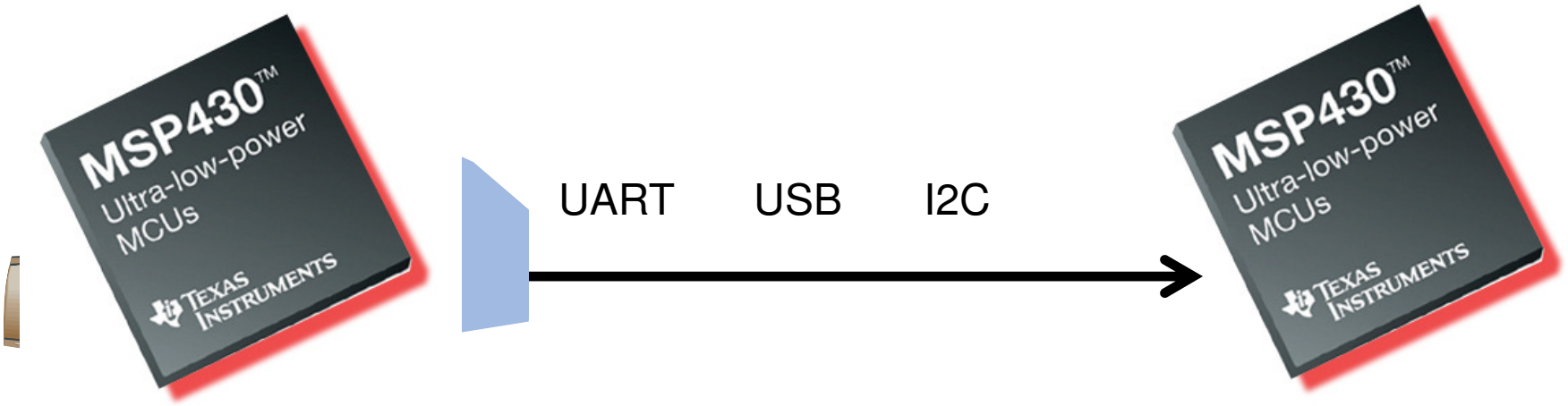
Other options

- OpenBSL https://code.google.com/p/ti-txt-parser/downloads/list
  - An open source, light-weight, customizable bootloader for MSP430G2xx devices
  - Currently supports MSP430G2553 using UART (USCI)
  - Resides in main memory (~2KB)
  - Project available for CCS

- Flash Monitor http://www.ti.com/lit/pdf/slaa341
  - Small program for F1xx-F4xx which provides the capability to examine and modify memory via UART
  - Resides in main memory (~1KB)
  - Project available for IAR

Using the Factory BSL

BSL System Overview



BSL Host

BSL Communication Interface

BSL Target Device

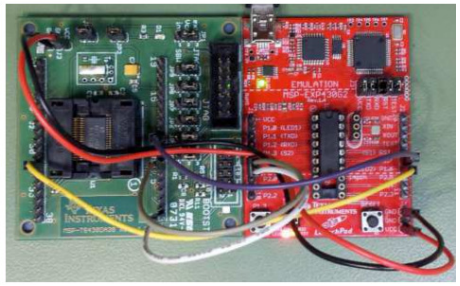
BSL Host – PC SW tools

- MSPBSL_Library
  - https://github.com/MSP-EricLoeffler/MSPBSL_Library
  - An Open Source, Cross Platform, Object Oriented (C++) Library designed to allow for easy PC-based communication with standard MSP430 BSLs.
- BSL_Scripter
  - http://www.ti.com/lit/zip/slau319
  - PC-based application and source code supporting Flash BSL and FRAM
- BSLDEMO2
  - http://www.ti.com/lit/zip/slau319
  - PC-based application and source code supporting F1xx/2xx/4xx BSL.
- USB Field Firmware Updater
  - http://www.ti.com/tool/msp430usbdevpack
  - This application allows for GUI controlled firmware download to an MSP430 device via the USB BSL.
- Python MSP430 tools
  - https://launchpad.net/python-msp430-tools
  - An open source collection of Python MSP430 tools. Includes scripts to interface with standard MSP430 BSLs.

BSL Host – HW tools

- Olimex BSL Rocket tool
  - https://www.olimex.com/Products/MSP430/BSL/MSP430-BSL/
  - Small, affordable programmer
  - Has support for UART and I2C
  - Supports Flash BSL by default but can be modified to support ROM BSL

- Launchpad-based MSP430 UART BSL interface
  - http://www.ti.com/lit/pdf/slaa535
  - Low cost option using MSP-EXP430G2 acting as UART bridge



- Flying Camp MSP430 BSL Programmer
  - http://www.flyingcampdesign.com/msp430-bsl-programmer.html
  - Open source USB to UART bridge
  - Also available from Moteware (http://www.moteware.com/products.php)

BSL Host – More HW tools

- MSP-GANG
  - http://www.ti.com/tool/msp-gang
  - JTAG/SBW/BSL programmer capable of programming 8 target devices simultaneously
  - Standalone or PC-based
  - Currently only supports UART, but I2C support is planned

- GangPro430
  - http://www.elprotronic.com/gangpro430.html
  - JTAG/SBW/BSL programmer capable of programming 6 target devices simultaneously
  - Currently only supports UART, but I2C support is planned

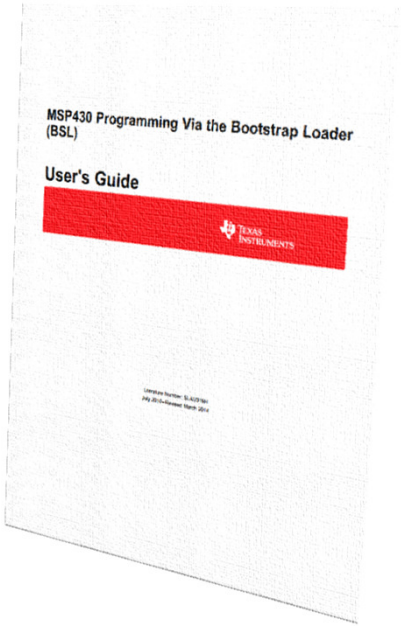
- FlashPro430
  - http://www.elprotronic.com/flashpro430.html
  - JTAG/SBW/BSL programmer
  - Currently only supports UART, but I2C support is planned

BSL documentation

SLAU319

- Explains factory BSLs (ROM and Flash) :
  - Entry sequence
  - Commands
  - Password protection
  - Patches for ROM BSL
  - BSL Versions for all devices
  - Known bugs

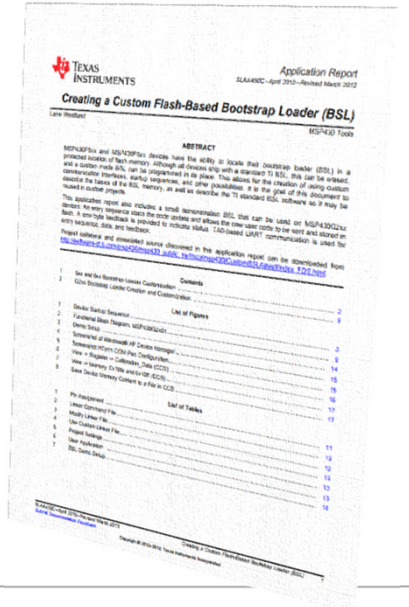
http://www.ti.com/lit/pdf/slau319



SLAA450

- Explains how to customize the Flash BSL in F5xx/F6xx:
  - Memory organization in BSL including Z-Area, JTAG Key, BSL signature
  - Startup and entry sequence
  - BSL_Protect function
  - Relevant functions and definitions for BSL modules (API, PI, CI)
- Also Includes vBSL:
  - How to build it and use it
  - Connection to host

http://www.ti.com/lit/pdf/slaa450

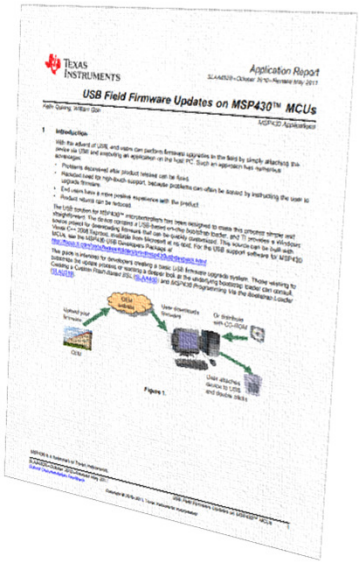


BSL documentation

SLAA452:

- Explains the implementation of USB BSL:
  - Invoking USB BSL
  - Using the GUI

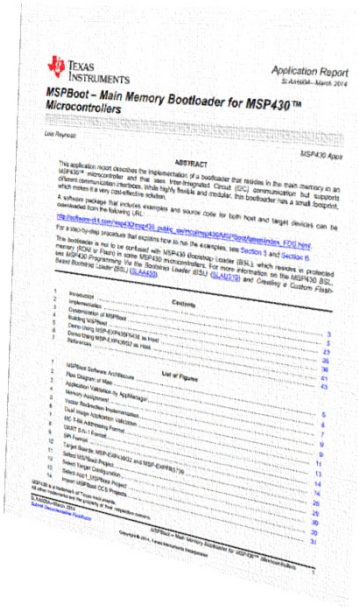
http://www.ti.com/lit/pdf/slaa452



SLAA600:

- Explains MSPBoot:
  - Vector redirection
  - Dual image implementation
  - Protocol, peripherals
  - Host implementation

http://www.ti.com/lit/pdf/slaa600



BSL documentation

SLAU550:

- Explains the implementation of FR59xx BSL.

  New format including:
  - Entry sequence
  - Protocol
  - BSL versions, known bugs
  - Examples for all commands

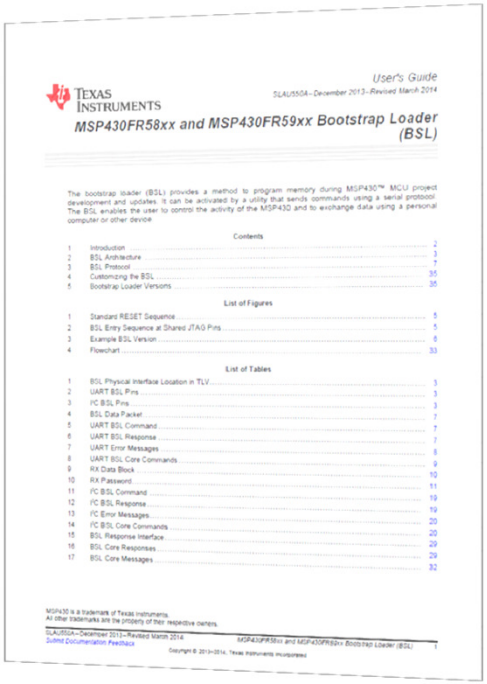
Command Example

Request the BSL version:

I2C	Header	Length	Length	CMD	CKL	CKH
S/A/W	0x80	0x01	0x00	0x19	0xE8	0x62

BSL response (version 00.07.34.B2 of the BSL):

I2C	ACK	Header	Length	Length	CMD	D1	D2	D3	D4	CKL	CKH	I2C
S/A/R	0x00	0x80	0x05	0x00	0x3A	0x00	0x07	0x34	0xB2	0x14	0x90	STOP


User's Guide
SLAU550A–December 2013–Revised March 2014

TEXAS INSTRUMENTS

MSP430FR58xx and MSP430FR59xx Bootstrap Loader (BSL)

The bootstrap loader (BSL) provides a method to program memory during MSP430™ MCU project development and updates. It can be activated by a utility that sends commands using a serial protocol. The BSL enables the user to control the activity of the MSP430 and to exchange data using a personal computer or other device.

Contents

1 Introduction ... 2
2 BSL Architecture ... 3
3 BSL Protocol ... 7
4 Customizing the BSL ... 35
5 Bootstrap Loader Versions ... 35

List of Figures

1 Standard RESET Sequence ... 5
2 BSL Entry Sequence at Shared JTAG Pins ... 5
3 Example BSL Version ... 6
4 Flowchart ... 33

List of Tables

1 BSL Physical Interface Location in TLV ... 3
2 UART BSL Pins ... 3
3 I²C BSL Pins ... 3
4 BSL Data Packet ... 7
5 UART BSL Command ... 7
6 UART BSL Response ... 7
7 UART Error Messages ... 8
8 UART BSL Core Commands ... 9
9 RX Data Block ... 10
10 RX Password ... 11
11 I²C BSL Command ... 19
12 I²C BSL Response ... 19
13 I²C Error Messages ... 20
14 I²C BSL Core Commands ... 20
15 BSL Response Interface ... 29
16 BSL Core Responses ... 29
17 BSL Core Messages ... 32

MSP430 is a trademark of Texas Instruments.
All other trademarks are the property of their respective owners.

SLAU550A–December 2013–Revised March 2014
Submit Documentation Feedback

MSP430FR58xx and MSP430FR59xx Bootstrap Loader (BSL) 1

Copyright © 2013–2014, Texas Instruments Incorporated

http://www.ti.com/lit/pdf/slau550

Questions?