MSP430 Boot Loader Project Details

The goal of this project is to be able to re-program the MSP430F2132 from the top-board ARM CPU using SPI communication.

**MSP430 Code Outline:**

The boot loader program is a <1 KB program that is stored in the first two segments of the MSP430's Flash memory (E000-E400]; each segment is 512 bytes. This program ...

1. Erases all other code segments in flash memory (non-info segments), removing the previous Bottom Board program and preparing flash to be re-written to.

2.Begins accepting data over SPI in ~64 byte blocks of information. The program validates the 64 byte block with a 16 bit checksum appended to the end of the block. Each block of data has a starting address specified by the top board before a block of data is sent, indicating where in MSP430 memory the block will be written.

3. If the block is validated, the boot loader moves the information from a buffer in RAM to flash memory at the starting address specified by the ARM. Else, the boot loader program will send a RESEND message back to the ARM processor.

4. The boot loader will continue to accept blocks of data to reprogram the MSP430 flash until the Arm sends a TRANSMISSION\_COMPLETE signal, at which time the boot loader exits and branches to the first instruction of the new Bottom Board Program.

There are two new files created within the main BottomBoard project for the MSP430 called "bootStrapLoader.c" and it's respective header file. bootStrapLoader.c is responsible for reprogramming the boot loader program if a new version of the boot loader is created. The data of a new boot loader program version is embedded into a c structure within bootStrapLoader.c in the MSP430 BottomBoard project and is transferred into the first two segments of flash memory. Essentially, segments 0 and 1 are erased and re-written via word-writes of the new data.

In the event that the MSP430 receives a REPROGRAM command from the ARM processor, the boot loader is branched to. This is done by storing the address of the first boot loader program instruction in a know word of flash memory (word beginning at E002). An assembly command is then used to set the program counter to this memory address.

**asm**(" mov &0E002h ,PC ; Branch to bootloader");

**Communication Protocol**

Both the MSP430 and the ARM have message bytes with the following structure

**MSP430 MESSAGE BYTE**

0 VALID MESSAGE BIT

1 CONFIRM (used to confirm messages and data sent by ARM)

2 RESEND (sent to the ARM if a block of data fails validation)

3 READY (general message)

4 UNUSED

5 UNUSED

6 UNUSED

7 VALID MESSAGE BIT

**ARM MESSAGE BYTE**

0 VALID MESSAGE BIT

1 NEW TRANSMISSION (prepare for set-up)

2 START DATA (program data)

3 READY

4 FINISH TRANSMISSION

5 UNUSED

6 UNUSED

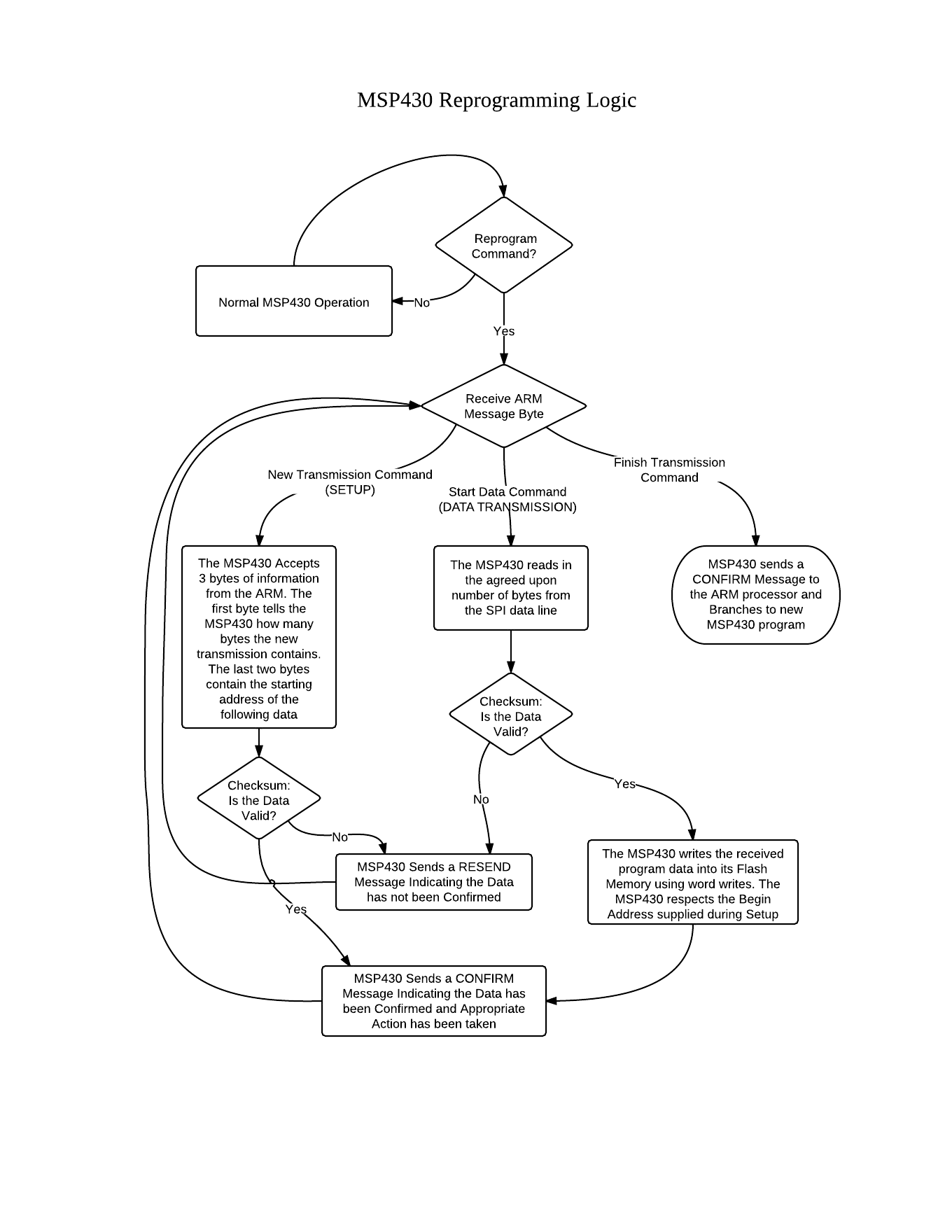
7 VALID MESSAGE BIT

The Arm processor sends the data of the new MSP430 program over the SPI bus in up to 64 byte segments (the last segment of data will most likely be shorter than 64 bytes). Each of these segments of data will have specified starting addresses that determine where in MSP430 Flash the data needs to be written.

To initiate a transmission, the ARM sends a NEW TRANSMISSION message over the SPI communication bus. The MSP430 confirms this command by sending a CONFIRM message and then prepares to receive setup information from the ARM. The ARM then sends 3 bytes of transmission parameter information along with a 16 bit checksum to validate that information. The first byte holds the length of the incoming segment and the last two bytes hold the 16-bit starting address of the segment.

If the setup parameters are validated, the MSP430 then confirms the transmission setup by sending a CONFIRM message back to the ARM. If it fails validation, the MSP430 sends a RESEND message over SPI, indicating that the setup has failed. Upon receiving confirmation from the MSP430, the ARM will then send a START DATA message to indicate the data is ready to be transmitted. The MSP430 sends a confirmation and then proceeds to read the established number of bytes from the SPI bus. This program data is validated with a 16 bit checksum that is appended to the end of the program data.

If the program data is validated, the MSP430 then transfers the data from a temporary buffer in RAM into flash memory, starting at the established start address. When this process has been completed it sends a confirmation to the ARM and the process is repeated until there is no more information left to be sent. At this point the ARM sends a FINISH TRANSMISSION message to the MSP430 and the MSP430 then branches to the first instruction of the new program (stored at 0xFFFE).



**ARM Code Outline:**

The top board code for the boot loader depends on the same timer interrupt that is used for normal SPI communication (\_\_cs3\_isr\_timer1b in system.c). The function that handles boot loader operation is called msp430BSLHandler and is located in msp430Bootloader.c. This function's structure is very similar to that of msp430InterruptHandler().

1. This function follows the protocol described by the flow diagram on the previous page. Each time the ISR is called, msp430BSLHandler will be called. When spiState is set to SPI\_STATE\_XMIT, a single byte of data will be sent over SPI. When the spiState is set to SPI\_STATE\_DESELECT, the msp430 is deselected on the Spi line.

2. The boot loader has 4 states: announce new trans, setup transmission parameters, transmit data, and finish transmission. These states are used to control the flow of data from the top to the bottom board.

3. The data of the new program is stored in a file named msp430ProgramData.h. When the bottom board project is compiled, this file should be modified to account for the changes to the bottom board project.

To initialize a transmission, the top board project must call the function msp430SystemReprogramCommand() located in spi\_message.c. This sets the SPI command to be a reprogram command. After sending the reprogram command, the ARM sets the msp430SpiOperationState to boot loader mode. At the same time, the msp430 branches to its own boot loader and begins receiving bytes.