C28x Boot ROM

Introduction

In chapter 10 we already discussed the option to start our embedded control program directly from the C28x internal Flash memory. We also looked briefly into other options for starting the code execution. We saw that it is also possible to start from H0 – SARAM, OTP and that we can select a 'boot load' operating mode that engages a serial or parallel download of the control code before it is actually executed.

In module 13 we will have a closer look into what is going on in these different modes and into the sequence of activities that is performed by the C28x boot firmware before your very own first instruction is touched. This chapter will help you to understand the start-up procedures of the C28x and the power-on problems of an embedded system in general.

We start with a summary of the six options to start the C28x out of RESET, followed by a look into the firmware structure inside the C28x Boot-ROM. This includes some lookup tables for mathematical operations, a generic interrupt vector table and the code that is used to select one of the six start options.

Because we have already dealt with the Flash start option in chapter 10, we can now focus on the serial boot loader options. Two options are available: Serial Communication Interface (SCI) and Serial Peripheral Interface (SPI). Both interfaces were discussed in detail in Modules 7 and 8. If you have finished the lab exercises of these two modules successfully, you should be able to develop your own code to download code from a PC as host into the SARAM of the C28x and start it from there.

A typical application for the serial download of new code into the C28x is a field update of the internal Flash memory that contains the control code for the embedded system. It would be much too expensive to use the JTAG-Emulator to download the new code. Instead, Texas Instruments offers a Flash API that uses exactly the same SCI boot load option to transmit the new code and/or data into the C28x. This API - a portion of code that will be part of your project will take care of the code update. For more details refer to "TMS320F2810, TMS320F2811 and TMS320F2812 Flash API v1.00", document number: SPRC125 on TI's website.

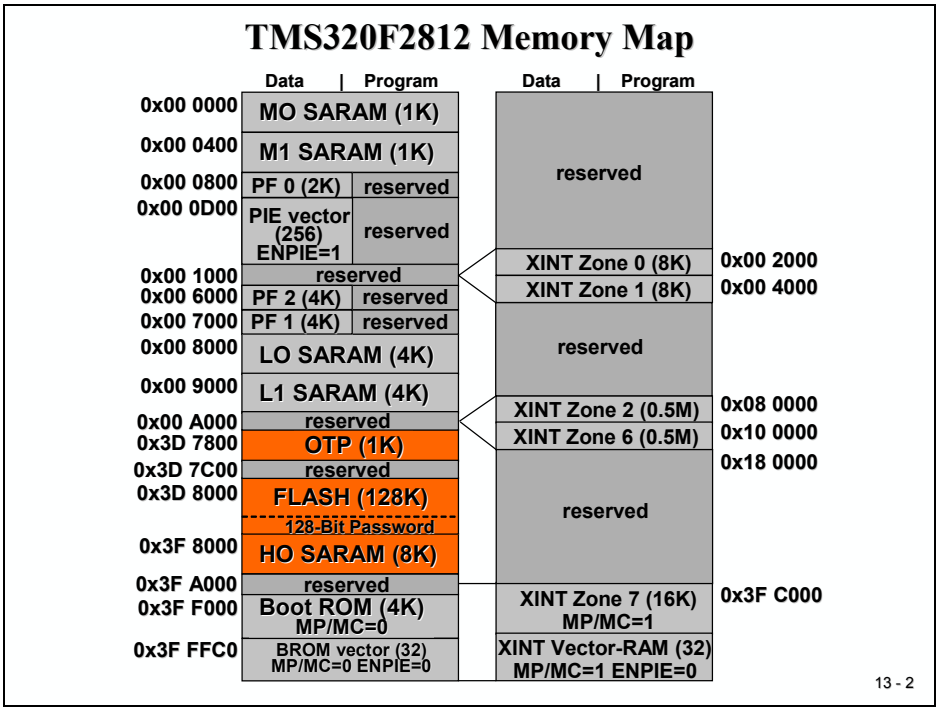
Another typical application is the use of the SPI boot load option. In this case, an external serial SPI-EEPROM of Flash holds the actual code. Before it is executed on the C28x, it is downloaded into the C28x. This is a useful option for the ROM version of the C28x or for an R28x, which do not have any internal non-volatile memory at all.

Finally, we will discuss a parallel boot load option that uses the GPIO port B to download code and/or data into the C28x.

Module Topics

C28x Memory Map

To begin with, let us recall the C28x memory map. We have a choice of starting our program from Flash, OTP and H0-SARAM, as highlighted in the slide:



We have six different options to start the C28x out of power-on. The options are hard-coded by 4 GPIO-Inputs of Port F (F4, F12, F3 and F2). The 4 pins are sampled during power-on; depending on the status one of the following options is selected:

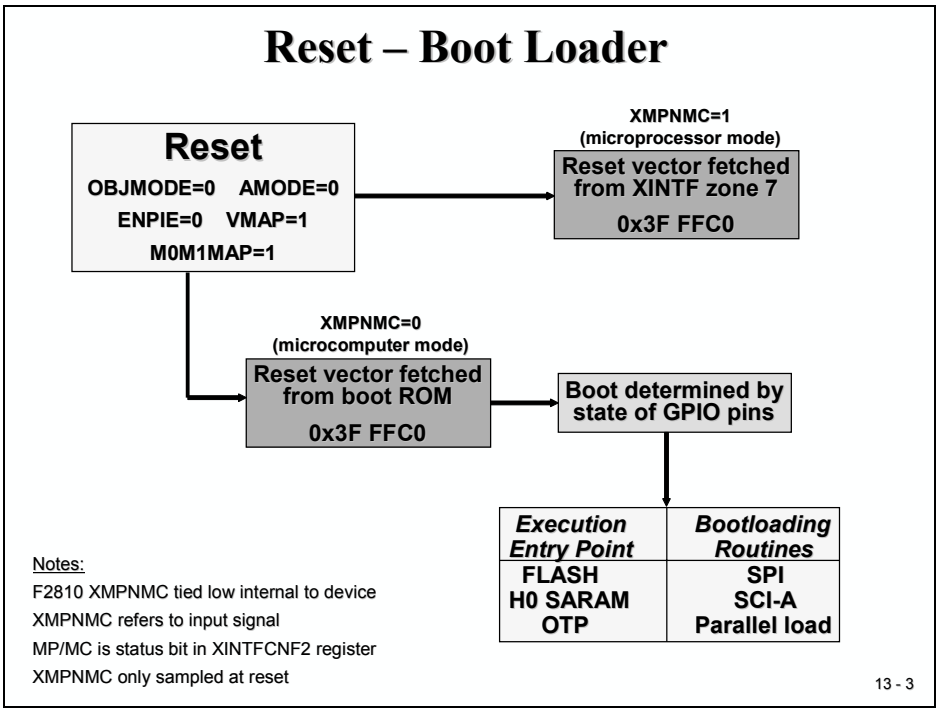
F4	F12	F3	F2		
1	x	x	x	:	FLASH address 0x3F 7FF6 (see slide 10-2)
0	0	1	0	:	H0 – SARAM address 0x3F 8000
0	0	0	1	:	OTP address 0x3D 7800
0	1	x	x	:	boot load from SPI
0	0	1	1	:	boot load from SCI-A
0	0	0	0	:	boot load from parallel GPIO – Port B

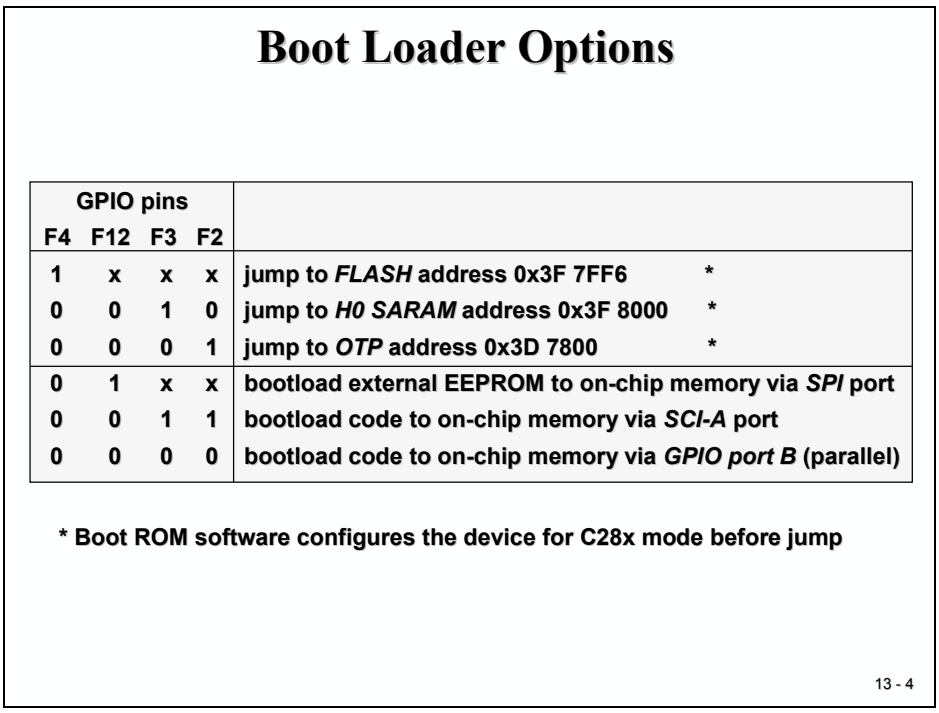
The F2812eZdsp controls the four lines F2, F3, F4 and F12 by four jumpers: JP7 (F4), JP8 (F12), JP11 (F3) and JP12 (F2). A ‘1’ in the table above is coded as 1-2 and a ‘0’ as 2-3 jumper set.

Jumper JP1 selects “Microcomputer-Mode” (2-3) or “Microprocessor-Mode” (1-2).

C28x Reset Boot Loader

The next two slides summarize the RESET options of the C28x.





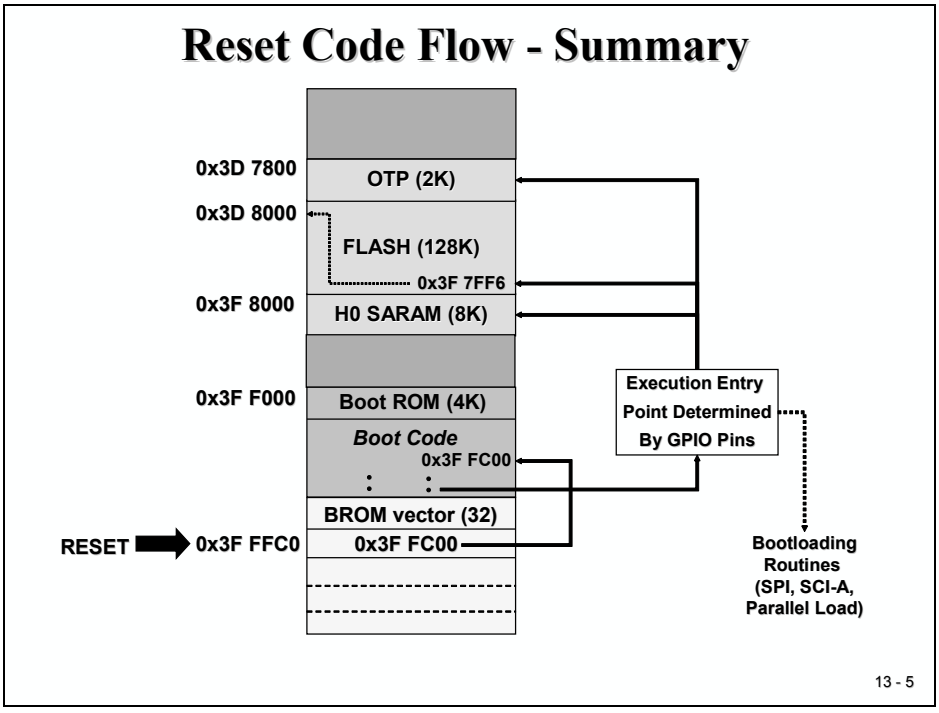
Timeline for Boot Load:

1. RESET-address is always 0x3F FFC0. This is part of TI's internal BOOT-ROM.

2. BOOT-ROM executes a jump to address 0x3F FC00 (the Boot Code). Here basic initialization tasks are performed and the type of the boot sequence is selected.

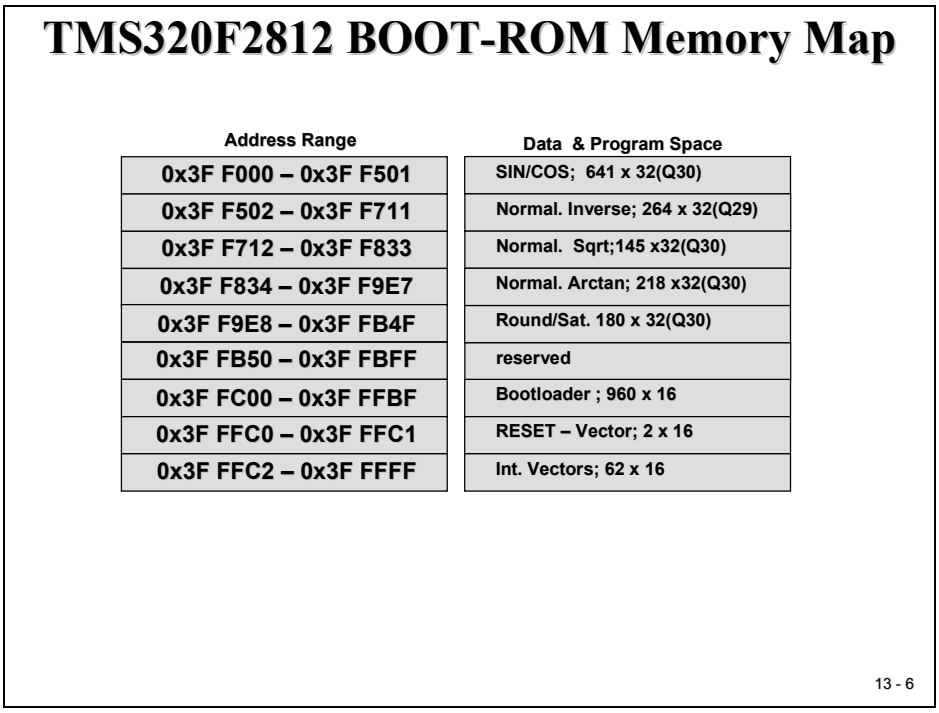
3. Next, still as part of the Boot Code, the execution entry point is determined by the status of the four GPIO-pins.

4. If one of the three boot loading options is selected, another dedicated part of the Boot Code is executed to establish a standard communication path for SCI, SPI or parallel port B. We will have a closer look into the three options in later slides.



Boot – ROM Memory Map

Before we go into the boot load options let us have a closer look into the partitioning of the boot-ROM area. The size of the area is 4K x 16bit and it is mapped both into code and data memory, using a unified memory map.



You can look at this memory with Code Composer Studio, providing you have started your eZdsp-board in "Microcomputer-Mode":

➔ View ➔ Memory

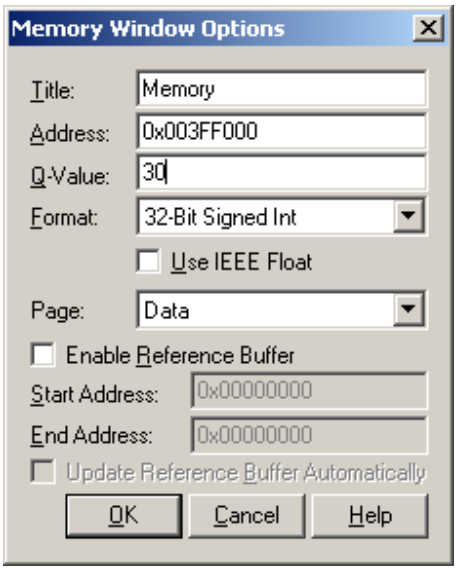
SINE / COSINE Lookup Table

Let's begin with the first 1282 addresses (0x3F F000 to 0x3F F501). This area includes a SIN/COS – Lookup-table and consists of 641 32bit-numbers. The first 512 numbers are for a 360-degrees unit circle with an increment angle of 360/512 = 0.703 degree. The remaining 128 elements repeat the first 90-degree angle.

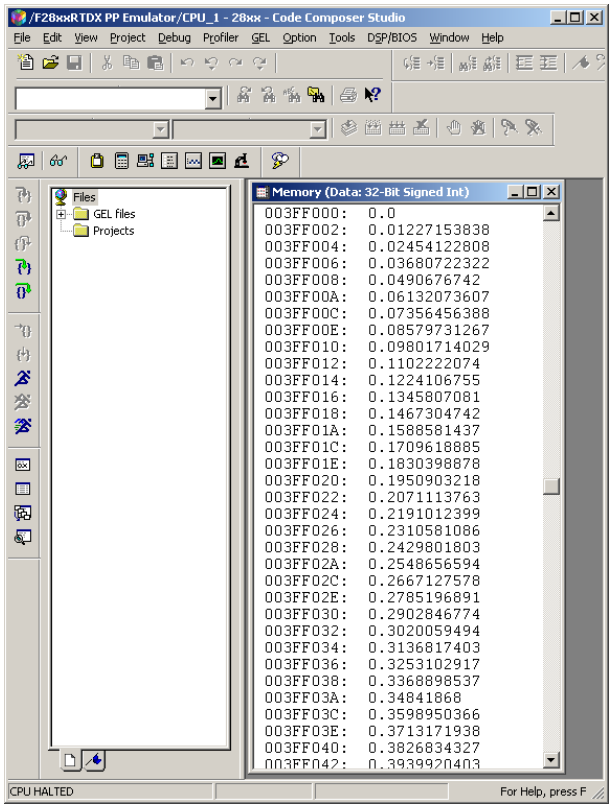
To visualize the SIN/COS -values setup the memory window properties like this:

➔ View ➔ Memory

Memory Window Options:



Numbers are in “IQ-Format” with 2 Integer and 30 Fractional Bits. CCS uses the binary content of the memory to display it in the correct format:



Compare: sin (1* 360/512) = 0.012271538285719926079408261951003

sin (2* 360/512) = 0.024541228522912288031734529459283

Normalized Inverse Table

The next section of the Boot-ROM includes a lookup table for the Newton-Raphson inverse algorithm. It spans 528 addresses (0x3F F502 to 0x3F F711) and covers 264 32-bit numbers in IQ29-Format.

Normalized Square Root Table

From address 0x3F F712 to 0x3F F833 145 32-bit numbers are stored as a lookup table for estimates of the Newton-Raphson square root algorithm. Data format is IQ30.

Normalized ArcTan Table

A lookup table for the iterative estimation of the Normalized Arc Tangent follows from 0x3F F834 to 0x3F F9E7 in IQ30-format.

Rounding and Saturation Table

Finally memory area 0x3F F9E8 to 0x3F FB4F is used for rounding and saturation subroutines of Texas Instrument library function, like IQ-math or digital motor control libraries (dmclib). The format is also of IQ30.

Boot Loader Code

The last (1K – 64) of memory addresses is used for the Boot Loader Code. When the C28x is coming out of RESET and is running in “Microcomputer Mode” this portion of code will be executed first. As mentioned earlier it derives the actual entry point or the boot loader option from the status of four input pins.

C28x Vector Table

The very last 64 addresses are reserved for 32 Entries of 32-bit address information for interrupt service routine entry points. The layout is shown at the following slide. Each interrupt core line is hard linked to its individual entry in this memory area. In the case where an interrupt is acknowledged by the C28x the assigned 32-bit-information (shown in the next slide as "Content") is used as entry point for the dedicated interrupt service routine. Because we can't change the content of this TI-ROM we have to use the fixed entry points in M0-SARAM (0x00 0040 to 0x00 007F) to place a 32-bit assembly branch instruction into our dedicated interrupt service routines. If we come out of RESET, all interrupts are disabled, so we don't have to do anything. If we decide to use interrupts, which is a wise decision for embedded control, we can use M0-SARAM as vector table – or – we use the Peripheral Interrupt Expansion (PIE) Unit – see Chapter 4.

C28x BOOT-ROM Vector Table

Vector	Address	Content
RESET	0x3F FFC0	0x3F FC00
INT1	0x3F FFC2	0x00 0042
INT2	0x3F FFC4	0x00 0044
INT3	0x3F FFC6	0x00 0046
INT4	0x3F FFC8	0x00 0048
INT5	0x3F FFCA	0x00 004A
INT6	0x3F FFCC	0x00 004C
INT7	0x3F FFCE	0x00 004E
INT8	0x3F FFD0	0x00 0050
INT9	0x3F FFD2	0x00 0052
INT10	0x3F FFD4	0x00 0054
INT11	0x3F FFD6	0x00 0056
INT12	0x3F FFD8	0x00 0058
INT13	0x3F FFDA	0x00 005A
INT14	0x3F FFDC	0x00 005C
DLOGINT	0x3F FFDE	0x00 005E

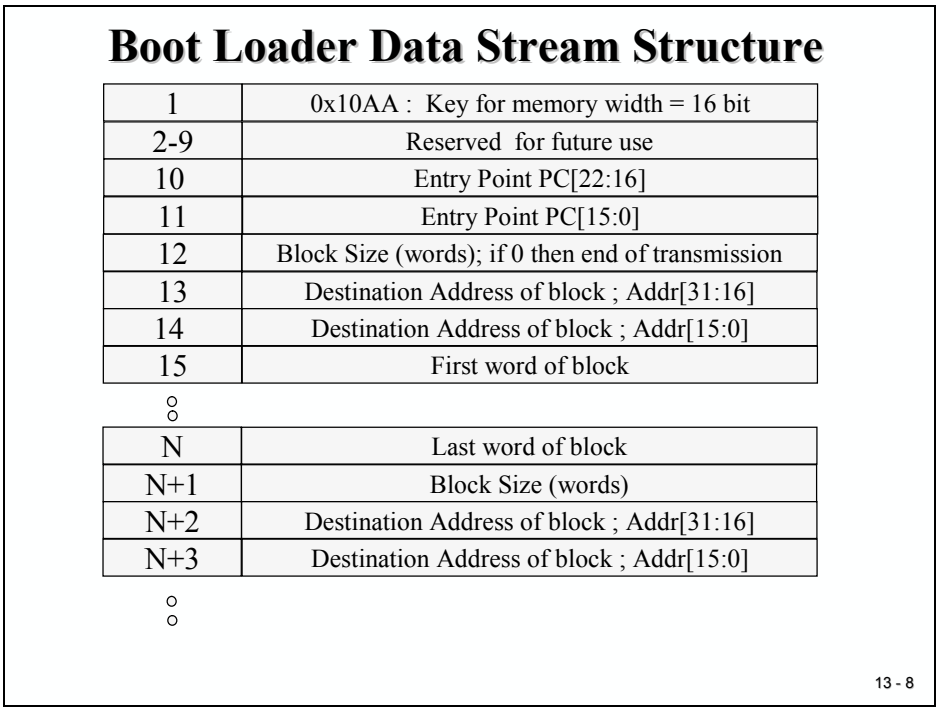
Vector	Address	Content
RTOSINT	0x3F FFE0	0x00 0060
reserved	0x3F FFE2	0x00 0062
NMI	0x3F FFE4	0x00 0064
ILLEGAL	0x3F FFE6	0x00 0066
USER 1	0x3F FFE8	0x00 0068
USER 2	0x3F FFEA	0x00 006A
USER 3	0x3F FFEC	0x00 006C
USER 4	0x3F FFEE	0x00 006E
USER 5	0x3F FFF0	0x00 0070
USER 6	0x3F FFF2	0x00 0072
USER 7	0x3F FFF4	0x00 0074
USER 8	0x3F FFF6	0x00 0076
USER 9	0x3F FFF8	0x00 0078
USER 10	0x3F FFFA	0x00 007A
USER 11	0x3F FFFC	0x00 007C
USER 12	0x3F FFFE	0x00 007E

13 - 7

Boot Loader Data Stream

The following two slides show the structure of the data stream incoming into the boot loader. The basic structure is the same for all the boot loaders and is based on the C28x hex utility. The tool is called “hex2000.exe (C:\ti\c2000\cgtools\bin)” and is used to convert the project’s out-file, which is in “COFF”-format into the necessary hex-format.

The first 16-bit word in the data stream is known as the key value. The key value is used to tell the boot loader the width of the incoming stream: 8 or 16 bits. Note that not all boot loaders will accept both 8 and 16-bit streams. The SPI boot loader is 8-bit only. Please refer to the detailed information on each loader for the valid data stream width. For an 8-bit data stream, the key value is 0x08AA and for a 16-bit stream it is 0x10AA. If a boot loader receives an invalid key value, then the load is aborted. In this case, the entry point for the Flash memory will be used.



The next eight words are used to initialize register values or otherwise enhance the boot loader by passing values to it. If a boot loader does not use these values then they are reserved for future use and the boot loader simply reads the value and then discards them. Currently, only the SPI boot loader uses one word to initialize registers.

The next 10^{th} and 11^{th} words comprise the 22-bit entry point address. This address is used to initialize the PC after the boot load is complete. This address is most likely the entry point of the program downloaded by the boot loader.

The twelfth word in the data stream is the size of the first data block to be transferred. The size of the block is defined for both 8 and 16-bit data stream formats as the number of 16-bit words in the block. For example, to transfer a block of twenty 8-bit data values from an 8-bit data stream, the block size would be 0x000A to indicate ten 16-bit words.

The next two words tell the loader the destination address of the block of data. Following the size and address will be the 16-bit words that makeup that block of data.

This pattern of block size/destination address repeats for each block of data to be transferred. Once all the blocks have been transferred, a block size of 0x0000 signals to the loader that the transfer is complete. At this point, the loader will return the entry point address to the calling routine, which in turn will cleanup and exit. Execution will then continue at the entry point address as determined by the input data stream contents.

.

Boot Loader Data Stream Example

Next is an example of a boot loader data stream that is used to load two blocks of data into two different memory locations of the C28x. Five words (1,2,3,4,5) are loaded into address 0x3F 9010 and two words are loaded into address 0x3F 8000.

Boot Loader Data Stream Example

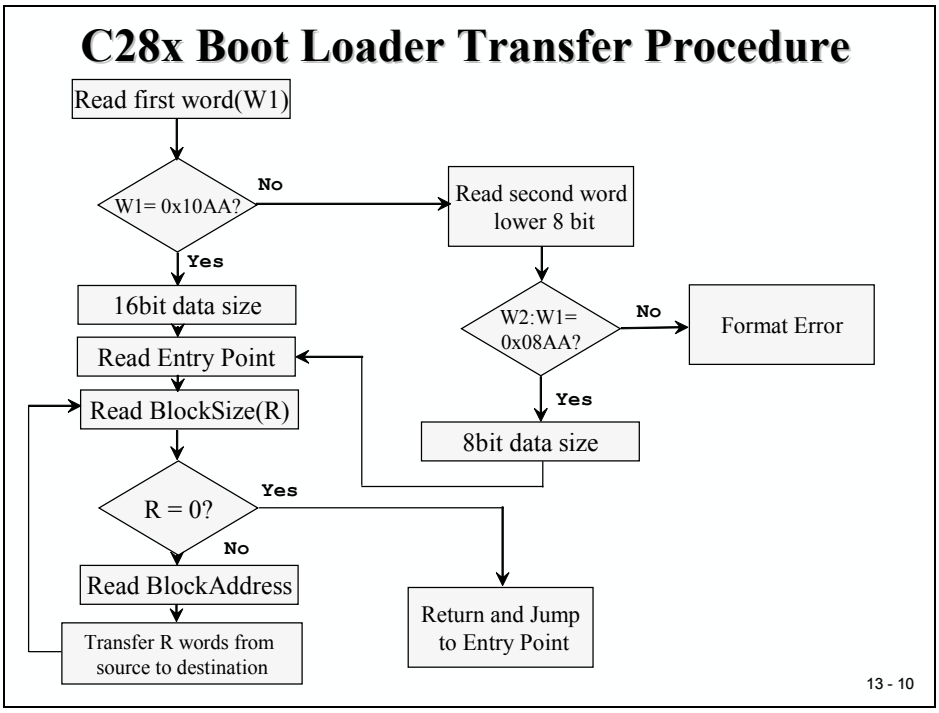
```
10AA    ; Key for 16-Bit memory stream
0000
0000
0000
0000
0000
0000
0000
0000
003F    ; PC – starting point after load is complete: 0x3F 8000
8000
0005    ; 5 words in block 1
003F
9010    ; First block is loaded into 0x3F 9010
0001    ; first data word
0002
0003
0004
0005    ; last data
0002    ; Second block is two words long
003F    ; Second block is loaded into 0x3F 8000
8000
7700    ; first data
7625    ; last data
0000    ; next block zero length = end of transmission
```

13-9

Boot Loader Transfer Function

The next flowchart illustrates the basic process a boot loader uses to determine whether 8-bit or 16-bit data stream has been selected, transfer that data, and start program execution. This process occurs after the boot loader finds the valid boot mode selected by the state of the GPIO pins.

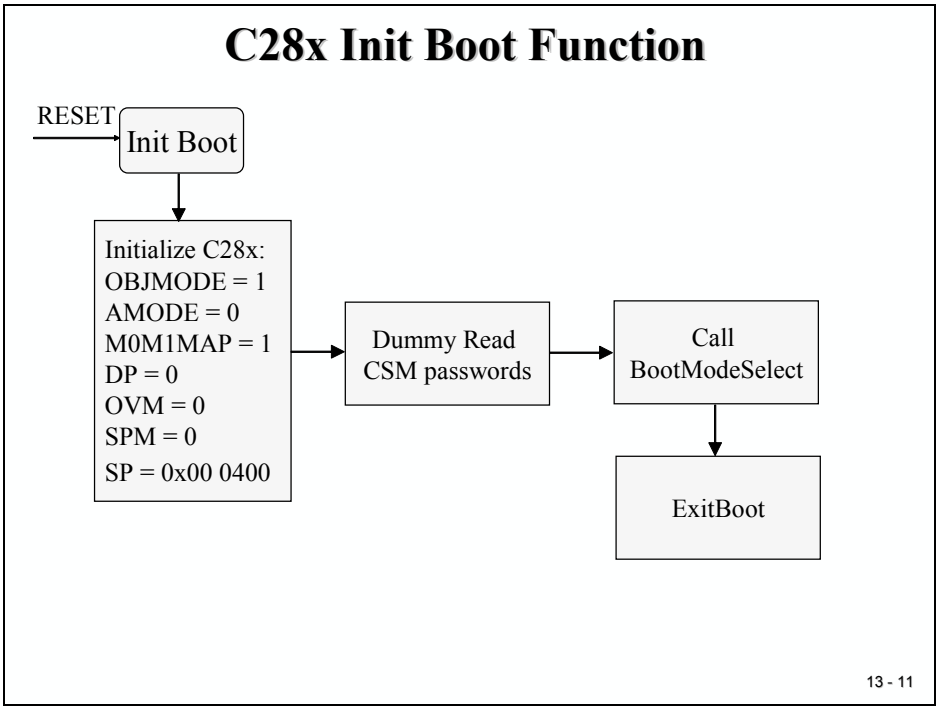
The loader compares the first value sent by the host against the 16-bit key value of 0x10AA. If the value fetched does not match then the loader will read a second value. This value will be combined with the first value to form a word. This will then be checked against the 8-bit key value of 0x08AA. If the loader finds that the header does not match either the 8-bit or 16-bit key value, or if the value is not valid for the given boot mode then the load will abort. In this case the loader will return the entry point address for the flash to the calling routine.



Init Boot Assembly Function

The first routine of the Boot-ROM that is called after RESET is the InitBoot assembly routine. This routine initializes the device for operation in C28x object mode. Next it performs a dummy read of the Code Security Module (CSM) password locations. If the CSM passwords are erased (all 0xFFFFs) then this has the effect of unlocking the CSM. Otherwise, the CSM will remain locked and this dummy read of the password locations will have no effect. This can be useful if you have a new device that you want to boot load.

After the dummy read of the CSM password locations, the InitBoot routine calls the SelectBootMode function. This function will then determine the type of boot mode desired by the state of certain GPIO pins. Once the boot is complete, the SelectBootMode function passes back the EntryAddr to the InitBoot function. InitBoot then calls the ExitBoot routine that then restores CPU registers to their reset state and exits to the EntryAddr that was determined by the boot mode.



SCI Boot Load

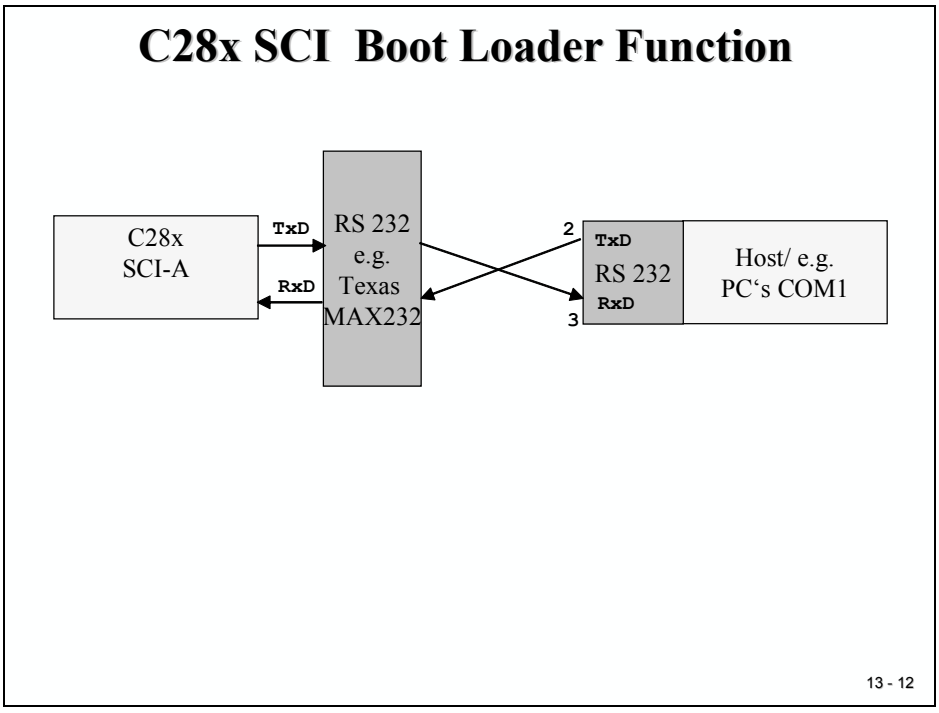
SCI Hardware Connection

The SCI boot mode asynchronously transfers code from SCI-A to the C28x. It only supports an incoming 8-bit data stream and follows the same data flow as outlined before.

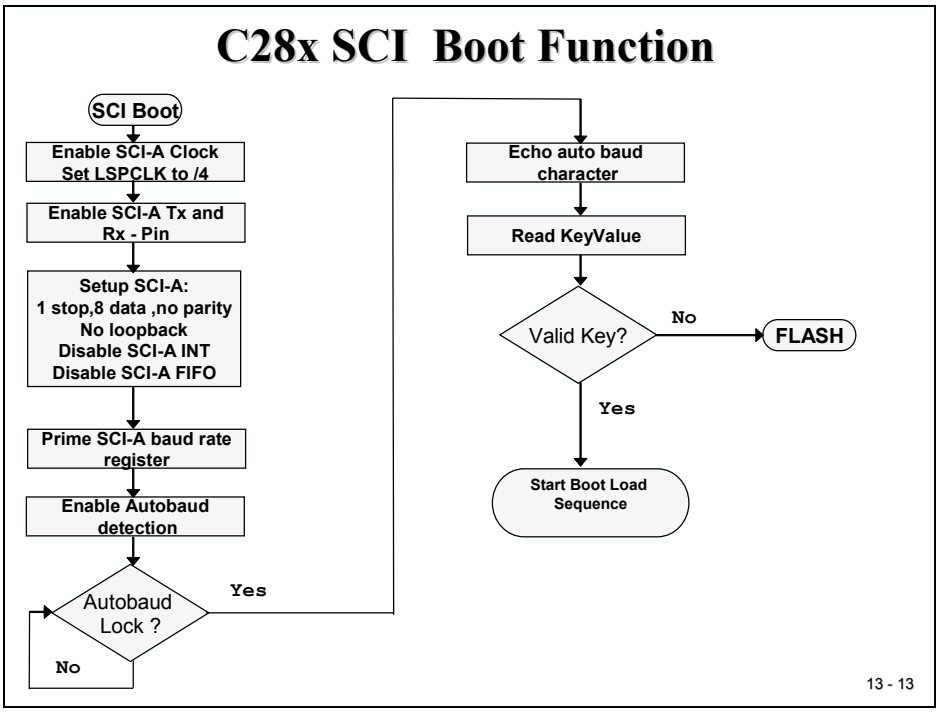
Note:

It is important to understand that, if you want to connect a PC via its serial COM-port to a C28x you will need to have a RS-232 transceiver device in front of the C28x to generate the necessary voltages. If you connect the C28x direct into the 2 PC-COM lines you will eventually destroy the C28x!

The 2812eZdsp does NOT have such a device on the board. The Zwickau adapter board, which was used in Module 8, is equipped with a TI MAX232. Ask your teacher about the actual set up in your laboratory!



SCI Boot Loader Function



The F2810/12 communicates with the external host device by communication through the SCI-A Peripheral. The auto baud feature of the SCI port is used to lock baud rates with the host. For this reason the SCI loader is very flexible and the user can use a number of different baud rates to communicate with the DSP.

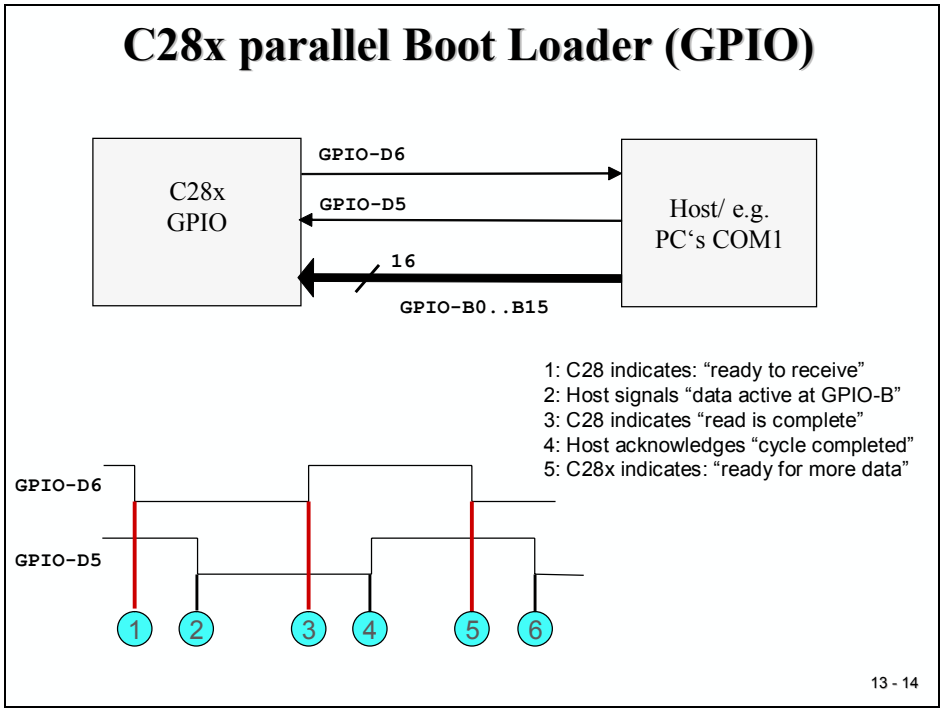
After each data transfer, the DSP will echo back the 8-bit character received to the host. In this manner, the host can perform checks that each character was received by the DSP.

At higher baud rates, the slew rate of the incoming data bits can be affected by transceiver and connector performance. While normal serial communications may work well, this slew rate may limit reliable auto-baud detection at higher baud rates (typically beyond 100 kbaud) and cause the auto-baud lock feature to fail.

Parallel Boot Loader

Hardware Connection

The parallel general purpose I/O (GPIO) boot mode asynchronously transfers code from GPIO port B to internal or XINTF memory. Each value can be 16 bits or 8 bits long and follows the same data flow as outlined in Data Stream Structure.



The F2810/12 communicates with the external host device by polling/driving the GPIOD5 and GPIOD6 lines. The handshake protocol shown above must be used to successfully transfer each word via GPIO port B. This protocol is very robust and allows for a slower or faster host to communicate with the F2810/12 device.

If the 8-bit mode is selected, two consecutive 8-bit words are read to form a single 16-bit word. The most significant byte (MSB) is read first followed by the least significant byte (LSB). In this case, data is read from the lower eight lines of GPIO port B ignoring the higher byte.

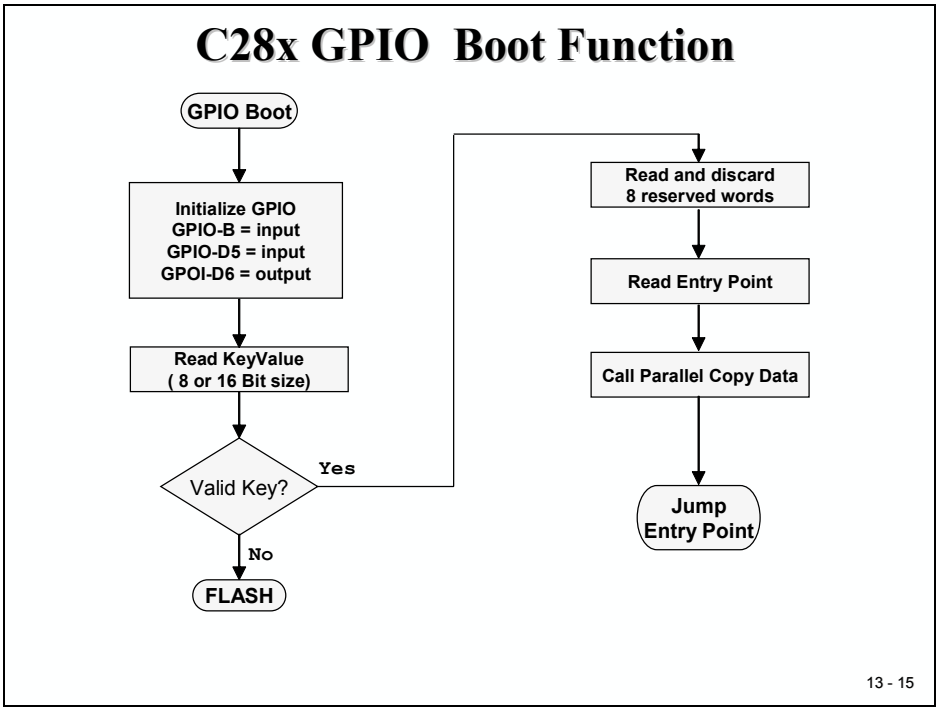
The DSP first signals to the host that the DSP is ready to begin data transfer by pulling the GPIOD6 pin low. The host load then initiates the data transfer by pulling the GPIOD5 pin low. The complete protocol is shown in the slide above.

C28x Software Flow

Slide 13-15 shows a flowchart for the Parallel GPIO boot loader inside the C28x. After parallel boot is selected during RESET, GPIO-port B is initialized as an input port. The two handshake lines GPIO-D5 and D6 are initialized as input and output respectively.

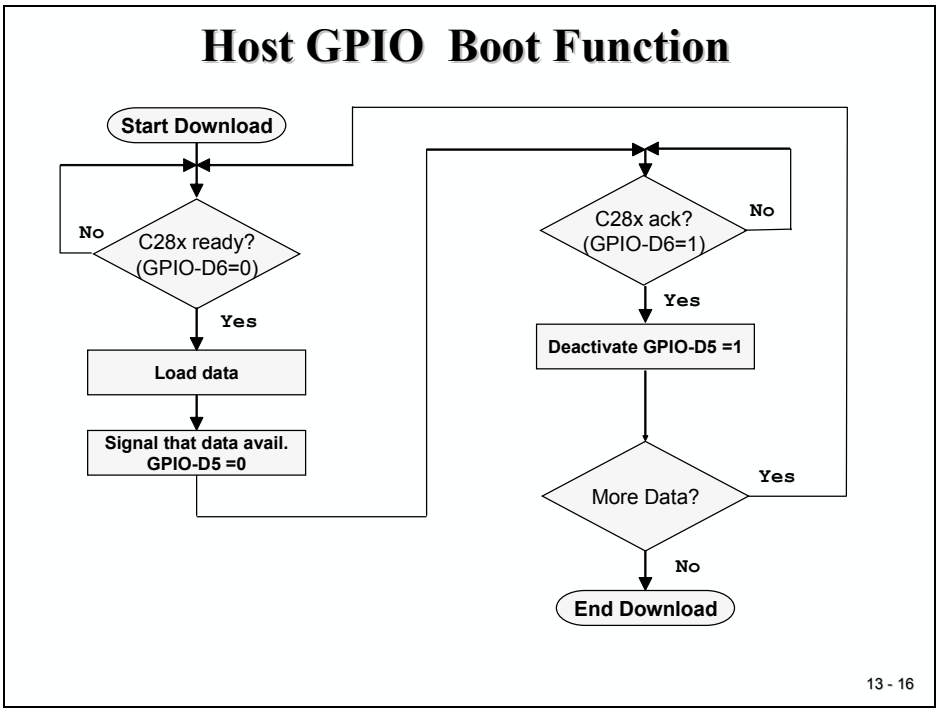
Next, the first character is polled from GPIO-port B. If it was a valid 8- (0x08AA) or 16-bit (0x10AA) key, the procedure continues to read eight more reserved words and discards them. Next, the code entry point and all following blocks are polled according to the diagram at slide 13-14.

If all blocks are received successfully, the routine jumps to the entry point address that was received during the boot load sequence.



Host Software Flow

Slide 13-16 shows the transfer flow from the Host side. The operating speed of the C28x and Host are not critical in this mode as the host will wait for the C28x and the C28x will in turn wait for the host. In this manner the protocol will work with both a host running faster and a host running slower then the C28x.



First, the host waits for a handshake signal (GPIO-D6) to be activated (= 0) by the C28x.

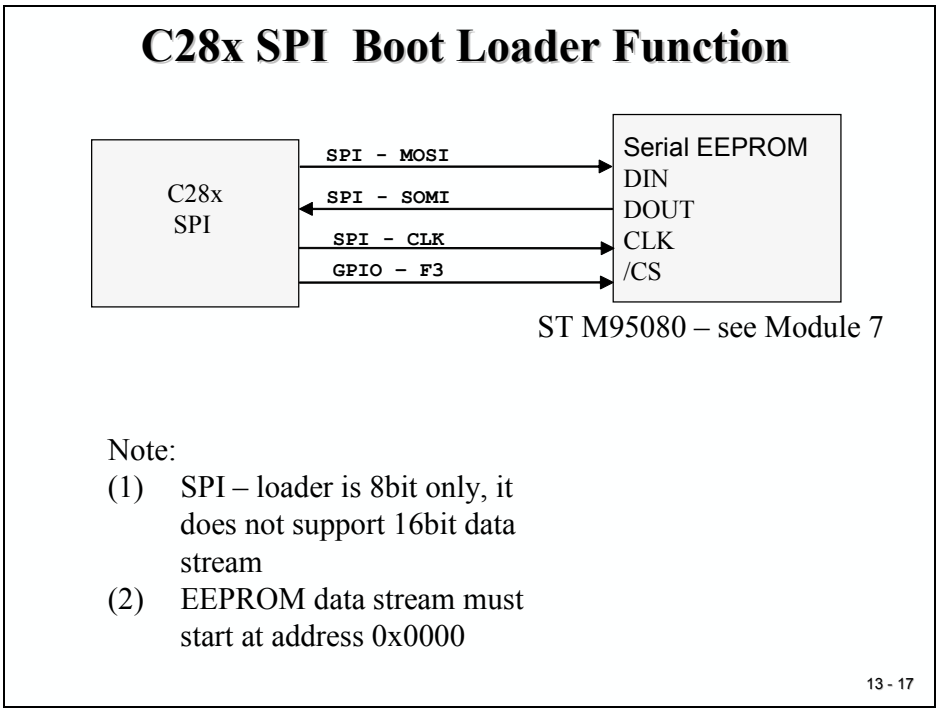
Next, the host has to load the next character onto its parallel output port. A valid character is then acknowledged by the host by activating (=0) a signal that is connected to the C28x GPIO-D5 input line.

The C28x has now all the time it requires to read the data from GPIO-port B. Once this is done, the C28x deactivates its output line GPIO-D6 to inform the host that the transfer cycle is completed.

The host acknowledges this situation by deactivating its handshake line (D5). If the algorithm has more data to transmit to the C28x, the procedure is repeated once more. If not, the download is finished.

SPI Boot Loader

The SPI loader expects an 8-bit wide SPI-compatible serial EEPROM device to be present on the SPI pins as indicated in Figure 21. The SPI boot loader does not support a 16-bit data stream.



The SPI boot ROM loader initializes the SPI module to interface to a serial SPI EEPROM. Devices of this type include, but are not limited to, the Microchip M95080 (1K x 8), the Xicor X25320 (4Kx8) and Xicor X25256 (32Kx8). The Zwickau adapter board is equipped with a M95080, which can be used to experiment with the SPI boot load mode.

The SPI boot ROM loader initializes the SPI to the following settings: FIFO enabled, 8-bit character, internal SPICLK master mode and talk mode, clock phase = 0, polarity = 0 and slowest baud rate.

If the download is to be preformed from an SPI port on another device, then that device must be setup to operate in the slave mode and mimic a serial SPI EEPROM. Immediately after entering the SPI Boot function, the pin functions for the SPI pins are set to primary and the SPI is initialized. The initialization is done at the slowest speed possible. Once the SPI is initialized and the key value read, the user could specify a change in baud rate or low speed peripheral clock.

SPI Boot Loader Data Stream

The following slide shows the sequence of 8-bit data expected by the Boot Loader.

C28x SPI Boot Loader Data Stream

Byte	Content
1	LSB = 0xAA (Key for 8bit transfer)
2	MSB = 0x08 (Key for 8bit transfer)
3	LSB = LSPCLK value
4	MSB = SPIBRR value
5-18	reserved
19	Entry Point [23:16]
20	Entry Point [31:24]
21	Entry Point [7:0]
22	Entry Point [15:8]
23 ...	Blocks of data: block size/destination/data as shown

13 - 18

SPI Boot Loader Flowchart

The chart is shown on the next page. The data transfer is done in “burst” mode from the serial SPI EEPROM. The transfer is carried out entirely in byte mode (SPI at 8 bits/character). A step-by step description of the sequence follows:

1) The SPI-A port is initialized
2) The GPIOF3 pin is now used as a chip-select for the serial SPI EEPROM
3) The SPI-A outputs a read command to the serial SPI EEPROM

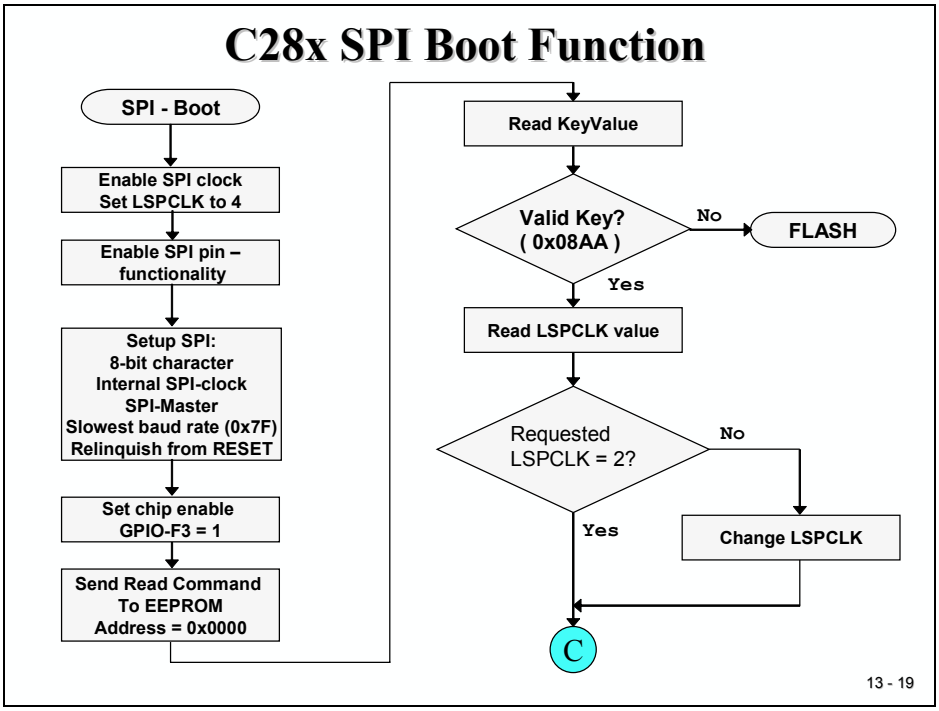
4) The SPI-A sends the serial SPI EEPROM address 0x0000; that is, the host requires that the EEPROM must have the downloadable packet starting at address 0x0000 in the EEPROM.

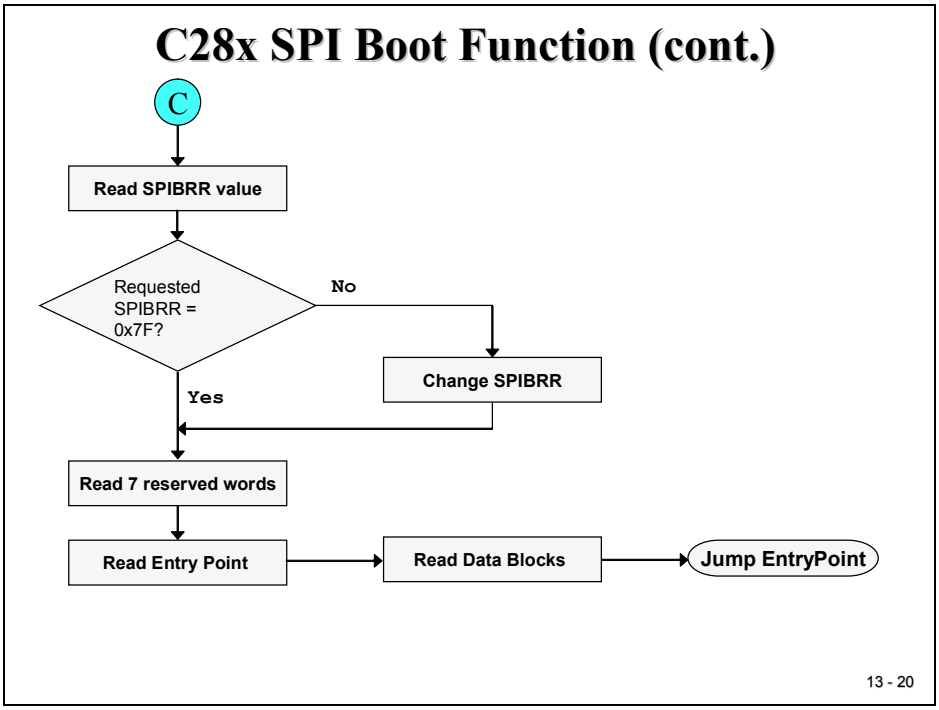
5) The next word fetched must match the key value for an 8-bit data stream (0x08AA). The most significant byte of this word is the byte read first and the least significant byte is the next byte fetched. This is true of all word transfers on the SPI. If the key value does not match then the load is aborted and the entry point for the Flash (0x3F 7FF6) is returned to the calling routine.

6) The next two bytes fetched can be used to change the value of the low speed peripheral clock register (LOSPCP) and the SPI Baud rate register (SPIBRR). The first byte read is the LOSPCP value and the second byte read is the SPIBRR value. The next seven words are reserved for future enhancements. The SPI boot loader reads these seven words and discards them.

7) The next two words makeup the 32-bit entry point address where execution will continue after the boot load process is complete. This is typically the entry point for the program being downloaded through the SPI port.

8) Multiple blocks of code and data are then copied into memory from the external serial SPI EEPROM through the SPI port. The blocks of code are organized in the standard data stream structure presented earlier. This is done until a block size of 0x0000 is encountered. At that point in time, the entry point address is returned to the calling routine that then exits the boot loader and resumes execution at the address specified.




C28x SPI Boot Function (cont.)
C
Read SPIBRR value
Requested SPIBRR = 0x7F?
No
Change SPIBRR
Yes
Read 7 reserved words
Read Entry Point
Read Data Blocks
Jump EntryPoint
13 - 20