F2833x Boot ROM

Introduction

In Chapter 14 we discussed the option of starting our embedded control program directly from the internal Flash memory of the F2833x. We also looked briefly into other options for starting the code execution. We saw that it is also possible to start up from M0 - 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 Chapter 15 we will take a closer look into what is going on in these different modes and into the sequence of activities that are performed by the F2833x boot firmware before the first instruction of your program is reached. This chapter will help you to understand the start-up procedures of the F2833x and the power-on problems of an embedded system in general.

We start with a summary of the 16 options to start the F2833x after a RESET, followed by a look into the firmware structure inside the F2833x 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 14, we can now focus on the serial boot loader options. Five options are available: Serial Communication Interface (SCI), Serial Peripheral Interface (SPI), Inter Integrated Circuit (I2C), Controller Area Network (CAN) used for motor vehicles and Multi Channel Buffered Serial Port (McBSP). All five interfaces were discussed in detail in Chapters 9, 10, 11, 12 and 13. If you have finished the lab exercises of some of these five modules successfully, you should be able to develop your own code to download code from a PC as host into the SARAM of the F2833x and start it from there.

A typical application for the serial download of new code into the F2833x 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 F2833x. 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 “TMS320F2833xFlash API v2.10”, document number: SPRC539 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 F2833x, it is downloaded into the F2833x. This is a useful option for members of the TMS320C34x - family, which do not have any internal non-volatile memory at all.

Finally, we will discuss a parallel boot load option that uses some GPIO lines to download code and/or data into the F2833x.

Module Topics

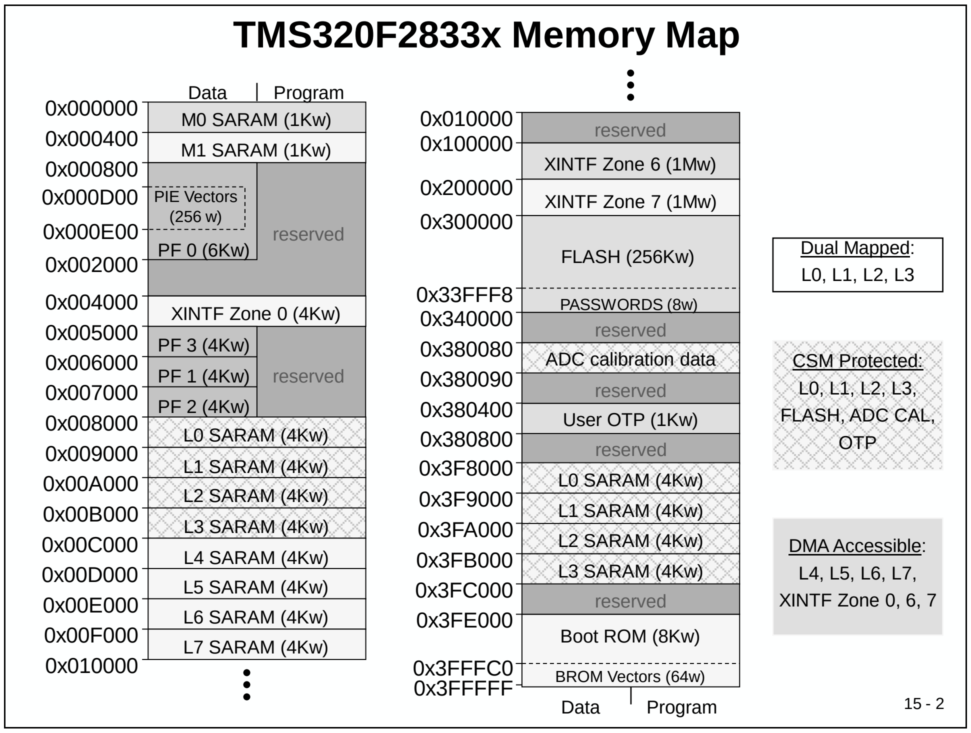
F2833x Memory Map

To begin with, let us recall the F2833x memory map.

Direct start of code execution

We have a choice of directly starting our program from fixed code entry points in the following memory sections:

- FLASH
- OTP
- M0-SARAM or
- XINTF - Zone 6

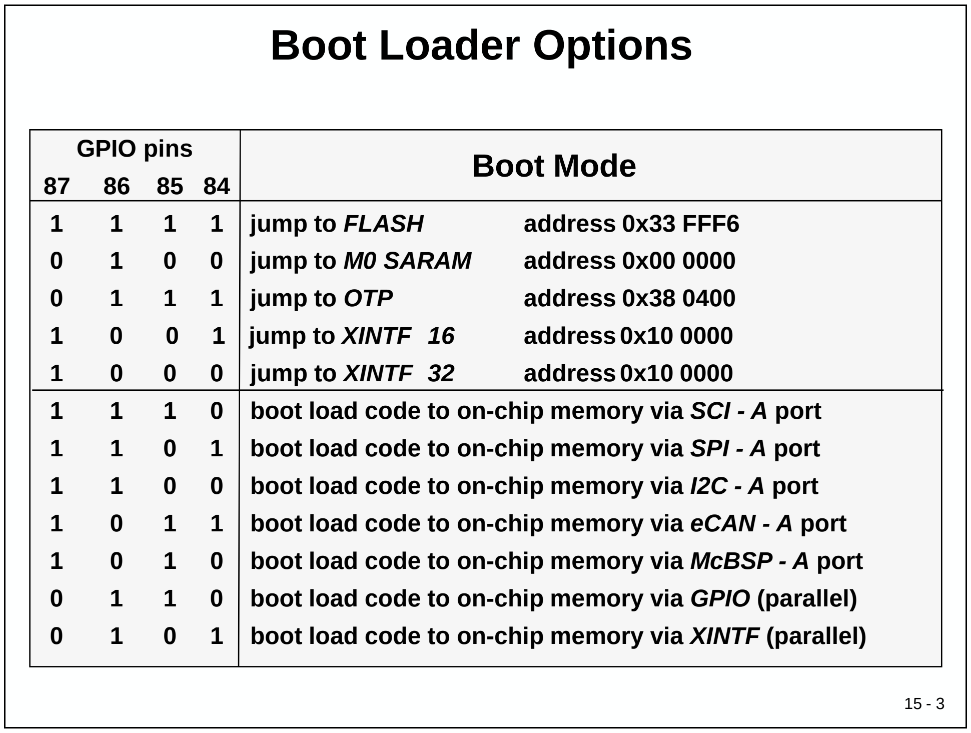


The options are hard-coded by 4 GPIO-lines (87, 86, 85 and 84). The 4 pins are always sampled during power-on. Depending on the status one of the options is selected and the code is executed immediately.

Start of a boot loader protocol

Instead of starting customer code directly after reset, we can engage a serial or parallel communication protocol between the F2833x and a host (e.g. a PC) or between the F2833x and an external non-volatile memory device. Such a communication link can be used (a) to download the control code before it is actually executed or (b) to update the control code by a new revision.

In addition to the 4 direct start options for code execution, we have 7 more options to open a serial or parallel communication protocol after power-on as shown on Slide 15-3:



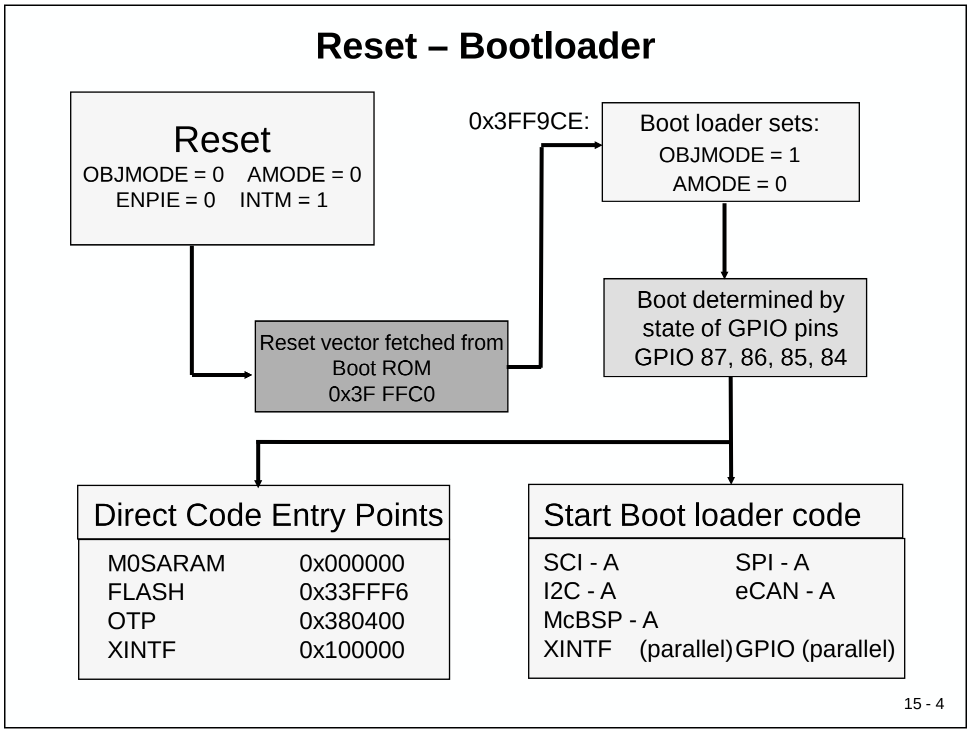
Boot Loader Options

GPIO pins 87	86	85	84	Boot Mode	
1	1	1	1	jump to *FLASH*	address 0x33 FFF6
0	1	0	0	jump to *M0 SARAM*	address 0x00 0000
0	1	1	1	jump to *OTP*	address 0x38 0400
1	0	0	1	jump to *XINTF 16*	address 0x10 0000
1	0	0	0	jump to *XINTF 32*	address 0x10 0000
1	1	1	0	boot load code to on-chip memory via *SCI - A* port	
1	1	0	1	boot load code to on-chip memory via *SPI - A* port	
1	1	0	0	boot load code to on-chip memory via *I2C - A* port	
1	0	1	1	boot load code to on-chip memory via *eCAN - A* port	
1	0	1	0	boot load code to on-chip memory via *McBSP - A* port	
0	1	1	0	boot load code to on-chip memory via *GPIO* (parallel)	
0	1	0	1	boot load code to on-chip memory via *XINTF* (parallel)	

15 - 3

F2833x Reset Boot Loader

The next two slides summarize the RESET options of the F2833x.



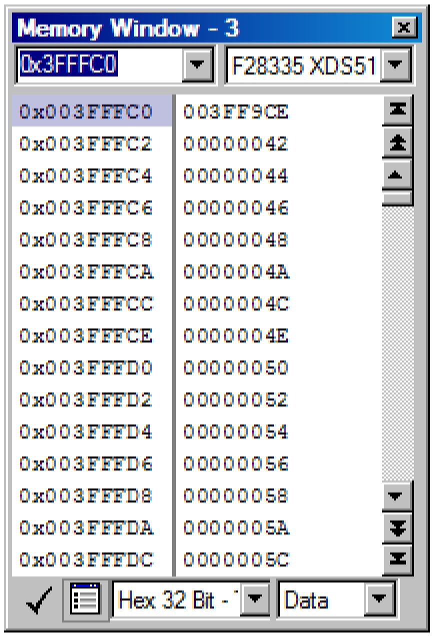
Reset – Bootloader

15 - 4

Timeline for Boot Loader

1. RESET-address is always 0x3F FFC0. This is part of Texas Instruments internal ROM.

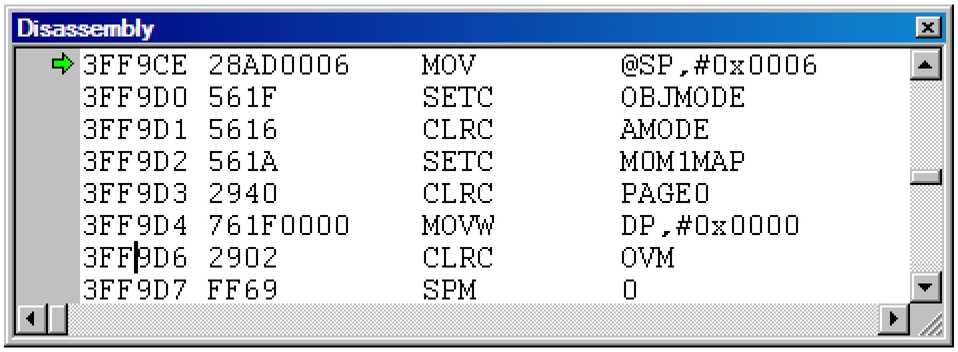
   Before we continue, let us inspect this part of the memory. In Code Composer Studio, open a memory window and enter the address 0x3F FFC0:



```
0x003FFFC0  003FF9CE
0x003FFFC2  00000042
0x003FFFC4  00000044
0x003FFFC6  00000046
0x003FFFC8  00000048
0x003FFFCA  0000004A
0x003FFFCC  0000004C
0x003FFFCE  0000004E
0x003FFFD0  00000050
0x003FFFD2  00000052
0x003FFFD4  00000054
0x003FFFD6  00000056
0x003FFFD8  00000058
0x003FFFDA  0000005A
0x003FFFDC  0000005C
```

   This is a direct view of the ROM-vector table at the end of this memory section. Address 0x3FFFC0 is loaded with the start address of the RESET-vector (0x3FF9CE); the following entries are vectors for interrupt INT1 (0x000042), INT2 (0x000044) and so on.

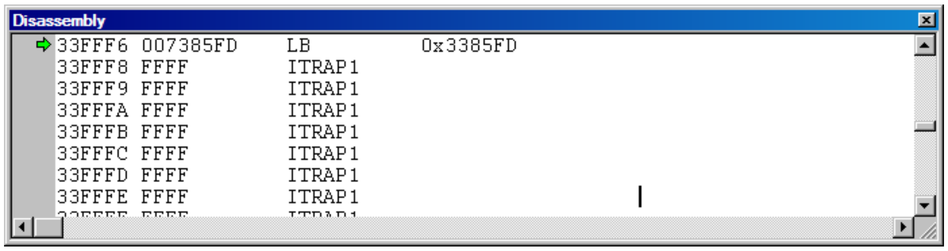
2. The F2833x reads the RESET-vector from the table and loads its program counter (PC) with this 32-bit value. If you perform a RESET in Code Composer Studio, the disassembly window will pop up and the green arrow will point to the first machine code instruction at address 0x3FF9CE, which is the first instruction of the boot code. Here basic initialization tasks are performed and the type of the boot sequence is selected.



```
3FF9CE  28AD0006    MOV      @SP,#0x0006
3FF9D0  561F        SETC     OBJMODE
3FF9D1  5616        CLRC     AMODE
3FF9D2  561A        SETC     M0M1MAP
3FF9D3  2940        CLRC     PAGE0
3FF9D4  761F0000    MOVW     DP,#0x0000
3FF9D6  2902        CLRC     OVM
3FF9D7  FF69        SPM      0
```

3. Next, still as part of the boot code function, the execution entry point is determined by the status of the four pins (GPIO87...84).

   In Code Composer Studio, if you use "single step over (F10)" from RESET a few times, you can inspect the sequence. If all four GPIOs (87...84) are '1' (in case of the Peripheral Explorer Board leave jumper J3 "SCI-boot 84" open), the FLASH-entry point is selected. After a few hits of function key F10 you will reach this entry point:

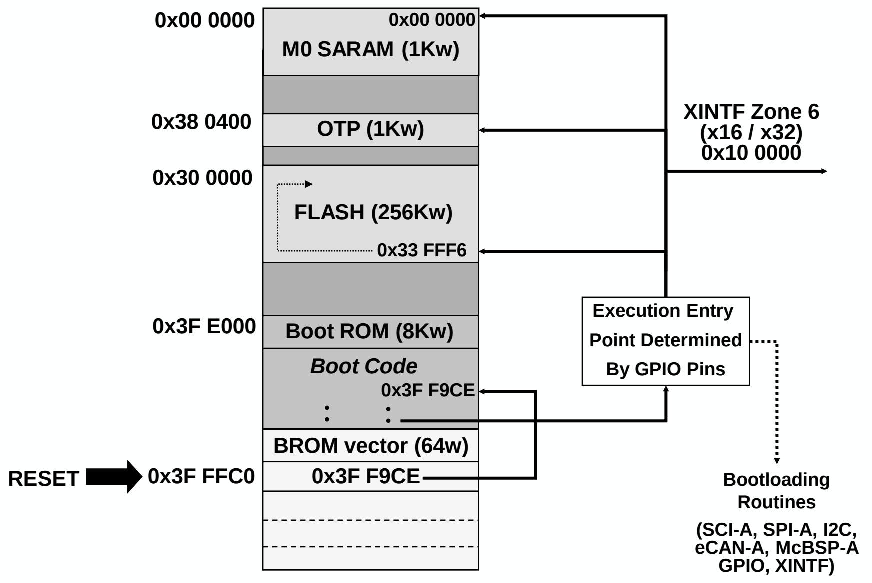


```
33FFF6 007385FD     LB          0x3385FD
33FFF8 FFFF         ITRAP1
33FFF9 FFFF         ITRAP1
33FFFA FFFF         ITRAP1
33FFFB FFFF         ITRAP1
33FFFC FFFF         ITRAP1
33FFFD FFFF         ITRAP1
33FFFE FFFF         ITRAP1
```

4. If one of the serial or parallel boot loading options is selected, another part of the boot code function is executed to establish a standard communication protocol for the pre-selected channel. We will have a closer look into these communication protocols in later slides. In case of the Peripheral Explorer Board we can close jumper J3 "SCI-boot 84" to select the "SCI-A boot loader". But before we will do that, we need to discuss the part of the communication host side.

Here is a graphical summary of the code flow after a RESET:

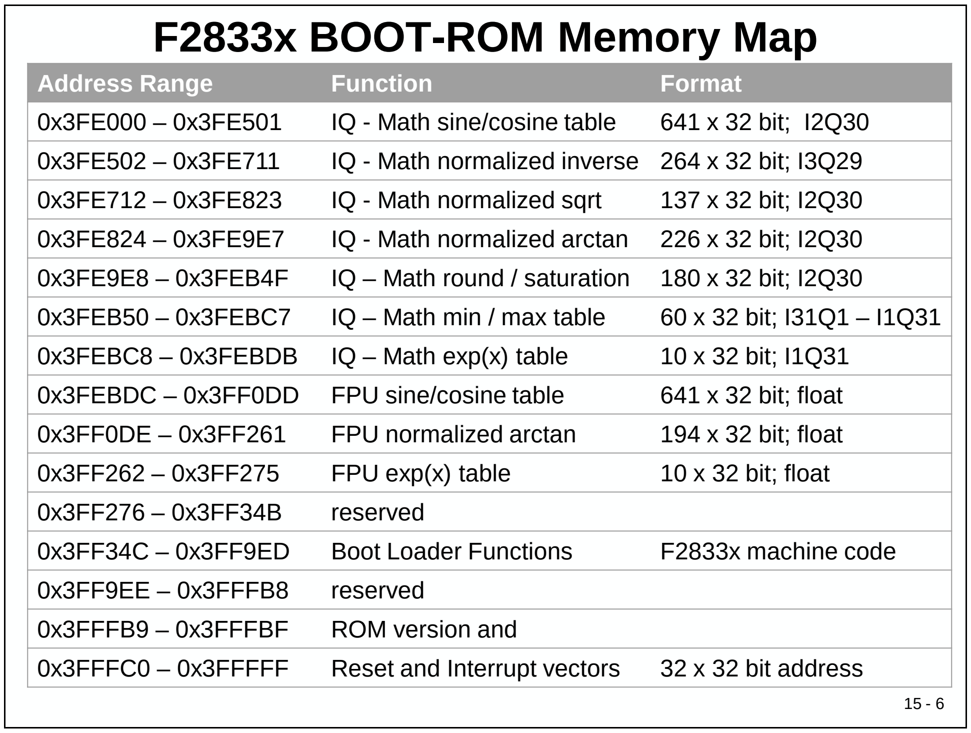
Reset Code Flow - Summary



15 - 5

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 8k x 16- bit and it is mapped both into code and data memory, using a unified memory map.



F2833x BOOT-ROM Memory Map

Address Range	Function	Format
0x3FE000 – 0x3FE501	IQ - Math sine/cosine table	641 x 32 bit; I2Q30
0x3FE502 – 0x3FE711	IQ - Math normalized inverse	264 x 32 bit; I3Q29
0x3FE712 – 0x3FE823	IQ - Math normalized sqrt	137 x 32 bit; I2Q30
0x3FE824 – 0x3FE9E7	IQ - Math normalized arctan	226 x 32 bit; I2Q30
0x3FE9E8 – 0x3FEB4F	IQ – Math round / saturation	180 x 32 bit; I2Q30
0x3FEB50 – 0x3FEBC7	IQ – Math min / max table	60 x 32 bit; I31Q1 – I1Q31
0x3FEBC8 – 0x3FEBDB	IQ – Math exp(x) table	10 x 32 bit; I1Q31
0x3FEBDC – 0x3FF0DD	FPU sine/cosine table	641 x 32 bit; float
0x3FF0DE – 0x3FF261	FPU normalized arctan	194 x 32 bit; float
0x3FF262 – 0x3FF275	FPU exp(x) table	10 x 32 bit; float
0x3FF276 – 0x3FF34B	reserved	
0x3FF34C – 0x3FF9ED	Boot Loader Functions	F2833x machine code
0x3FF9EE – 0x3FFFB8	reserved	
0x3FFFB9 – 0x3FFFBF	ROM version and	
0x3FFFC0 – 0x3FFFFF	Reset and Interrupt vectors	32 x 32 bit address

15 - 6

SINE / COSINE Lookup Tables

IQ-Math - Table

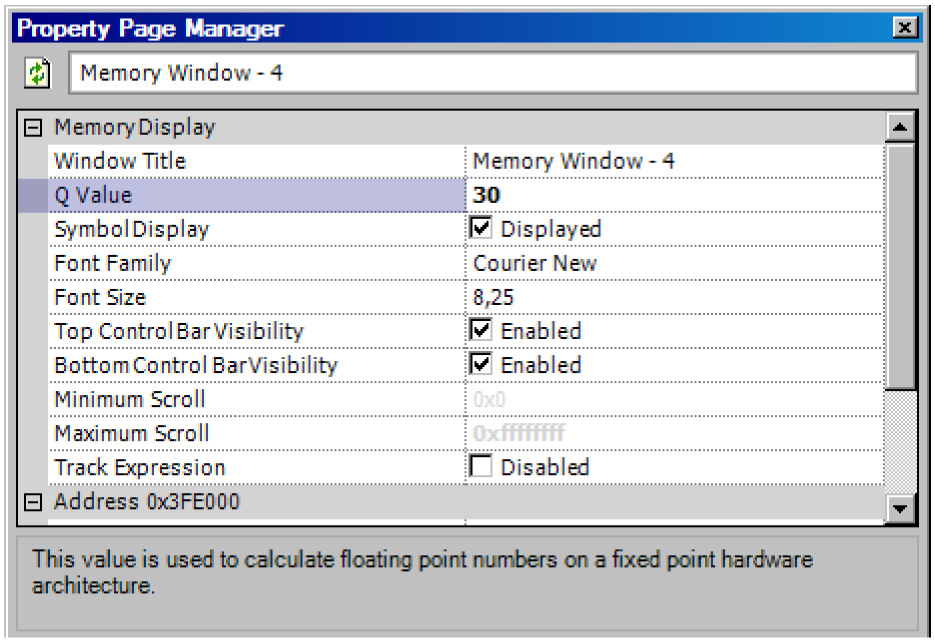
The ROM offers two different sine/cosine-tables; one for fixed-point math (IQ-numbers) and one for floating-point numbers (32-bit IEEE float format).

Since the look-up tables for sine and cosine are very useful tools in mathematic computations, we should inspect them now:

➔ View ➔ Memory

Let’s begin with the first 1282 addresses (0x3F E000 to 0x3F E501). This area includes an IQ-Math sine/cosine look-up table and consists of 641 32-bit numbers in I2Q30-format. The first 512 numbers are for a 360-degrees unit circle with an increment angle of 360/512 = 0.703 degree. The remaining 128 values are a repetition of the first 90-degree quarter.

To visualize the sine/cosine-values open a memory window and set up the properties to 32 - bit signed integer and Q-value to 30:



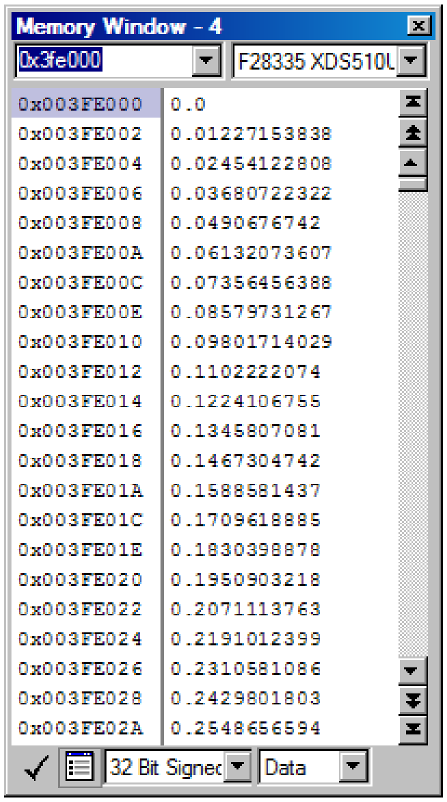
Property Page Manager

Memory Window - 4

⊟ Memory Display	
Window Title	Memory Window - 4
Q Value	**30**
Symbol Display	☑ Displayed
Font Family	Courier New
Font Size	8,25
Top Control Bar Visibility	☑ Enabled
Bottom Control Bar Visibility	☑ Enabled
Minimum Scroll	0x0
Maximum Scroll	0xffffffff
Track Expression	☐ Disabled
⊟ Address 0x3FE000	

This value is used to calculate floating point numbers on a fixed point hardware architecture.

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:



Memory Window - 4

0x3fe000 | F28335 XDS510U

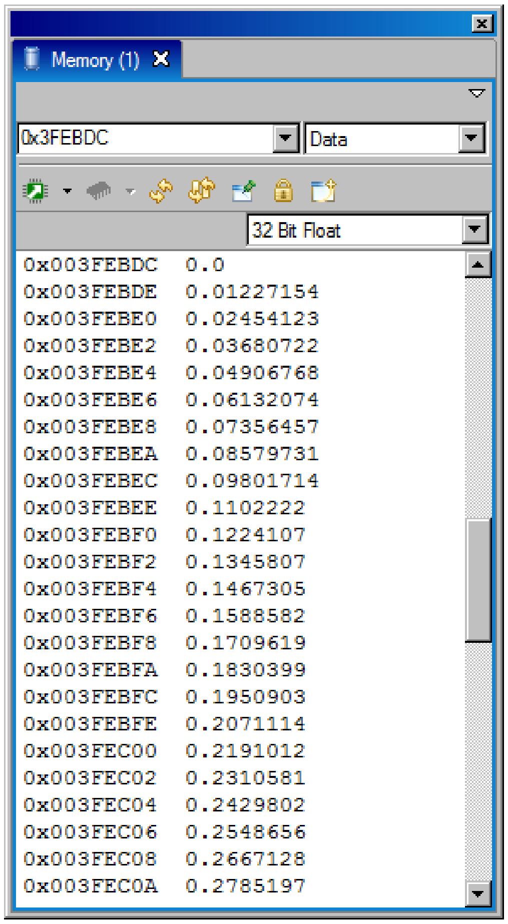
```
0x003FE000  0.0
0x003FE002  0.01227153838
0x003FE004  0.02454122808
0x003FE006  0.03680722322
0x003FE008  0.0490676742
0x003FE00A  0.06132073607
0x003FE00C  0.07356456388
0x003FE00E  0.08579731267
0x003FE010  0.09801714029
0x003FE012  0.1102222074
0x003FE014  0.1224106755
0x003FE016  0.1345807081
0x003FE018  0.1467304742
0x003FE01A  0.1588581437
0x003FE01C  0.1709618885
0x003FE01E  0.1830398878
0x003FE020  0.1950903218
0x003FE022  0.2071113763
0x003FE024  0.2191012399
0x003FE026  0.2310581086
0x003FE028  0.2429801803
0x003FE02A  0.2548656594
```

32 Bit Signed | Data

Compare:	sin (1* 360/512) = 0.012271538285719926079408261951003
	sin (2* 360/512) = 0.024541228522912288031734529459283

Floating-Point sine/cosine - Table

The floating-point look-up table at address 0x3FEBDC consists also of 640 entries for 5 quarters of sine-values, but now in IEEE 754 single precision floating point format. To inspect this region, change the memory windows start address to 0x3FEBDC and the data type to “32-bit Float”:



```
0x003FEBDC  0.0
0x003FEBDE  0.01227154
0x003FEBE0  0.02454123
0x003FEBE2  0.03680722
0x003FEBE4  0.04906768
0x003FEBE6  0.06132074
0x003FEBE8  0.07356457
0x003FEBEA  0.08579731
0x003FEBEC  0.09801714
0x003FEBEE  0.1102222
0x003FEBF0  0.1224107
0x003FEBF2  0.1345807
0x003FEBF4  0.1467305
0x003FEBF6  0.1588582
0x003FEBF8  0.1709619
0x003FEBFA  0.1830399
0x003FEBFC  0.1950903
0x003FEBFE  0.2071114
0x003FEC00  0.2191012
0x003FEC02  0.2310581
0x003FEC04  0.2429802
0x003FEC06  0.2548656
0x003FEC08  0.2667128
0x003FEC0A  0.2785197
```

You should always remember that there are these two tables available in the ROM. If you need to calculate trigonometric numbers, all you have to do is to set a pointer at the beginning of these memory arrays. In your control code you can then easily access sine/cosine-values.

Normalized Inverse Table

Another section of the Boot-ROM includes a lookup table for the Newton-Raphson inverse algorithm. It spans 528 addresses (0x3F E502 to 0x3F E711) and covers 264 32-bit numbers in I3Q29-Format.

Normalized Square Root Table

From address 0x3F E712 to 0x3F E823 137 32-bit numbers are stored as a look-up table for estimates of the Newton-Raphson square root algorithm. Data format is I2Q30.

Normalized ArcTan Table

A lookup table for the iterative estimation of the Normalized Arc Tangent follows from 0x3F E824 to 0x3F E9E7 in I2Q30-format.

Rounding and Saturation Table

The memory area 0x3F E9E8 to 0x3F EB4F 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 I2Q30.

Min / Max Table

A section with minimum and maximum fixed point numbers follows from address 0x3F EB50 to 0x3F EBC7. This array is used to define the number ranges from I31Q1 to I1Q31.

Exp(x) Table

A table for coefficients to calculate y = exp(x) using a Taylor series follows at address 0x3FEBC8. The numbering system is I1Q31.

Floating-point normalized ArcTan Table

A floating-point normalized arcos-tangents table in 32-bit float format is available from address 0x3F F0DE.

Floating-point Exp(x) Table

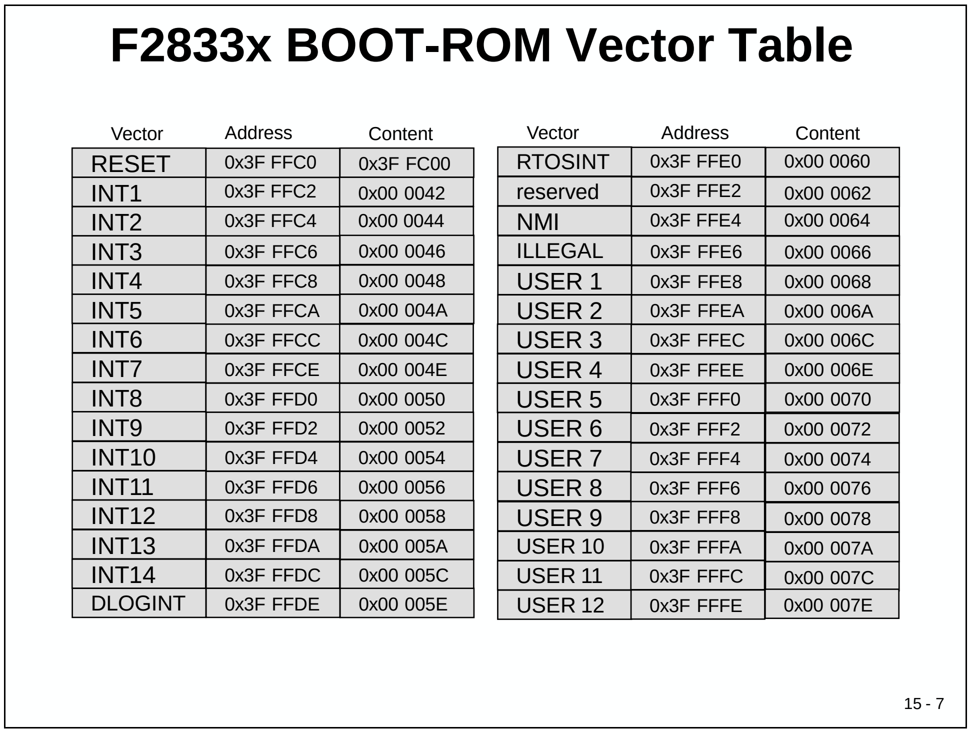
A table for coefficients to calculate y = exp(x) using a Taylor series follows at address 0x3FF262. The numbering system is single precision floating point.

Boot Loader Code

The memory space 0x3FF34C - 0x3FF9ED is used for the boot loader machine code. When the F2833x is coming out of RESET this 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.

F2833x 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 F2833x, the assigned 32-bit-information (shown in the next slide as “Content”) is used as the entry-point for the dedicated interrupt service routine. Because we cannot change the content of this TI-ROM, we have to use the fixed entry points in M0-SARAM (0x00 0042 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 better - we use the Peripheral Interrupt Expansion (PIE) Unit - see Chapter 6.



F2833x BOOT-ROM Vector Table

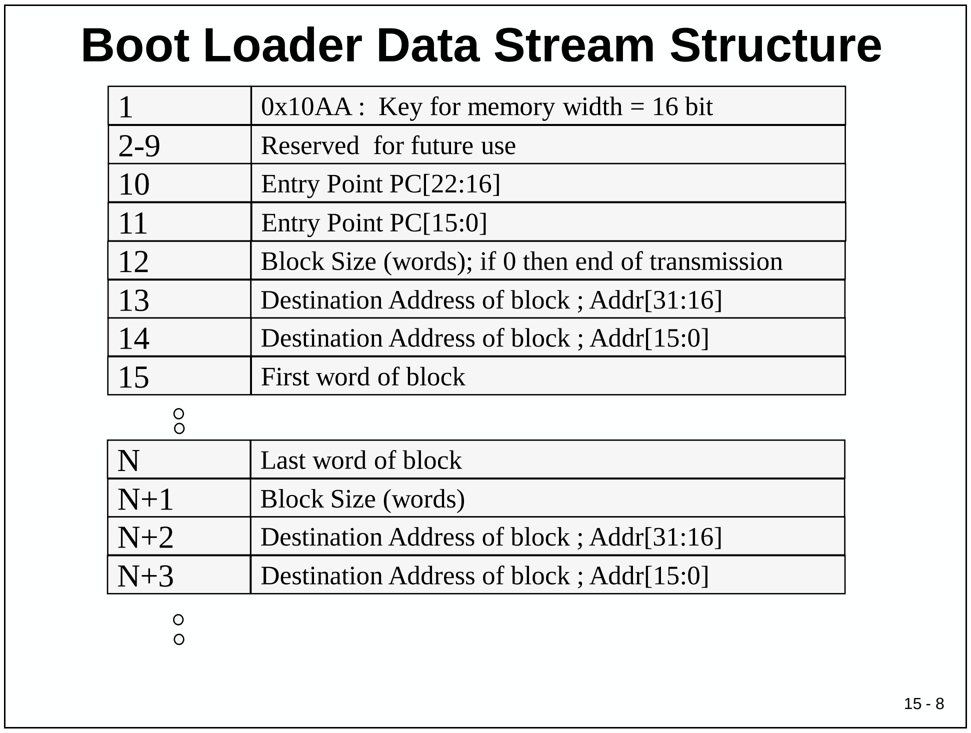
Vector	Address	Content	Vector	Address	Content
RESET	0x3F FFC0	0x3F FC00	RTOSINT	0x3F FFE0	0x00 0060
INT1	0x3F FFC2	0x00 0042	reserved	0x3F FFE2	0x00 0062
INT2	0x3F FFC4	0x00 0044	NMI	0x3F FFE4	0x00 0064
INT3	0x3F FFC6	0x00 0046	ILLEGAL	0x3F FFE6	0x00 0066
INT4	0x3F FFC8	0x00 0048	USER 1	0x3F FFE8	0x00 0068
INT5	0x3F FFCA	0x00 004A	USER 2	0x3F FFEA	0x00 006A
INT6	0x3F FFCC	0x00 004C	USER 3	0x3F FFEC	0x00 006C
INT7	0x3F FFCE	0x00 004E	USER 4	0x3F FFEE	0x00 006E
INT8	0x3F FFD0	0x00 0050	USER 5	0x3F FFF0	0x00 0070
INT9	0x3F FFD2	0x00 0052	USER 6	0x3F FFF2	0x00 0072
INT10	0x3F FFD4	0x00 0054	USER 7	0x3F FFF4	0x00 0074
INT11	0x3F FFD6	0x00 0056	USER 8	0x3F FFF6	0x00 0076
INT12	0x3F FFD8	0x00 0058	USER 9	0x3F FFF8	0x00 0078
INT13	0x3F FFDA	0x00 005A	USER 10	0x3F FFFA	0x00 007A
INT14	0x3F FFDC	0x00 005C	USER 11	0x3F FFFC	0x00 007C
DLOGINT	0x3F FFDE	0x00 005E	USER 12	0x3F FFFE	0x00 007E

15 - 7

Boot Loader Data Stream

The following two slides show the structure of the incoming data stream to the boot loader. The basic structure is the same for all the boot loaders and is based on the F2833x hex utility tool. The tool is en executable file called "hex2000.exe (C:\CCStudio_v3.3\C2000\cgtools\bin)" and is used to convert the project's out-file from "COFF"- format to 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.



Boot Loader Data Stream Structure

1	0x10AA : Key for memory width = 16 bit
2-9	Reserved for future use
10	Entry Point PC[22:16]
11	Entry Point PC[15:0]
12	Block Size (words); if 0 then end of transmission
13	Destination Address of block ; Addr[31:16]
14	Destination Address of block ; Addr[15:0]
15	First word of block

○
○

N	Last word of block
N+1	Block Size (words)
N+2	Destination Address of block ; Addr[31:16]
N+3	Destination Address of block ; Addr[15:0]

○
○

15 - 8

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 a register value.

The next 10th and 11th 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.

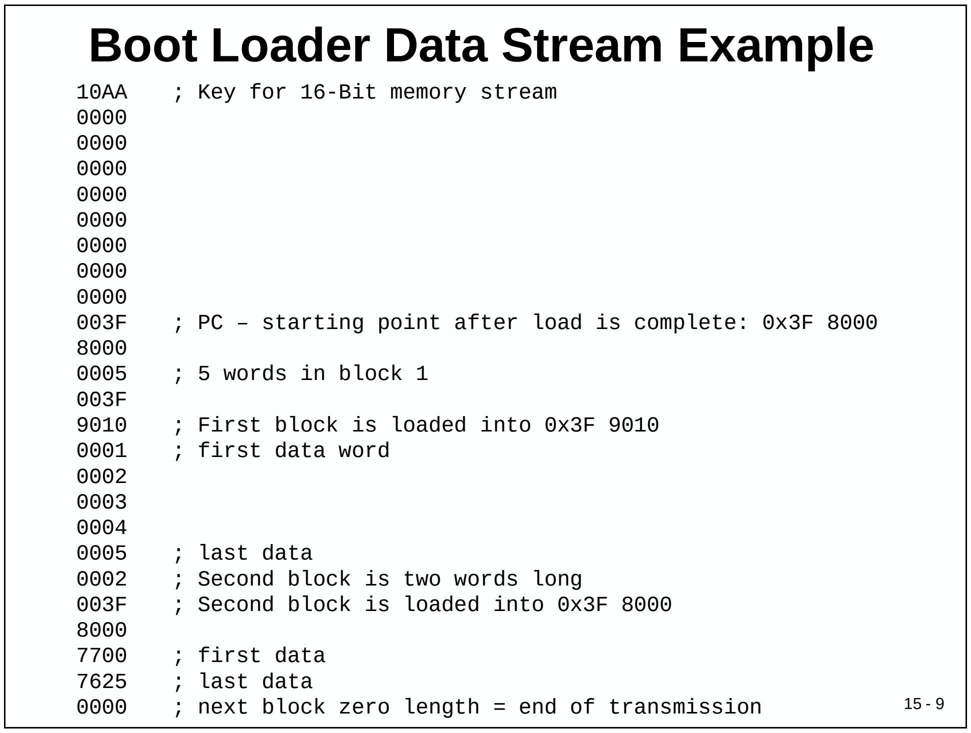
The twelfth word of the data stream is the size of the first data block to be transferred. The size of the block is defined for both 8 and 16-bit data stream formats as the number of 16-bit words in the block. For example, to transfer a block of twenty 8-bit data values from an 8-bit data stream, the 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 make up the corresponding 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 clean up 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 F2833x. 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
```

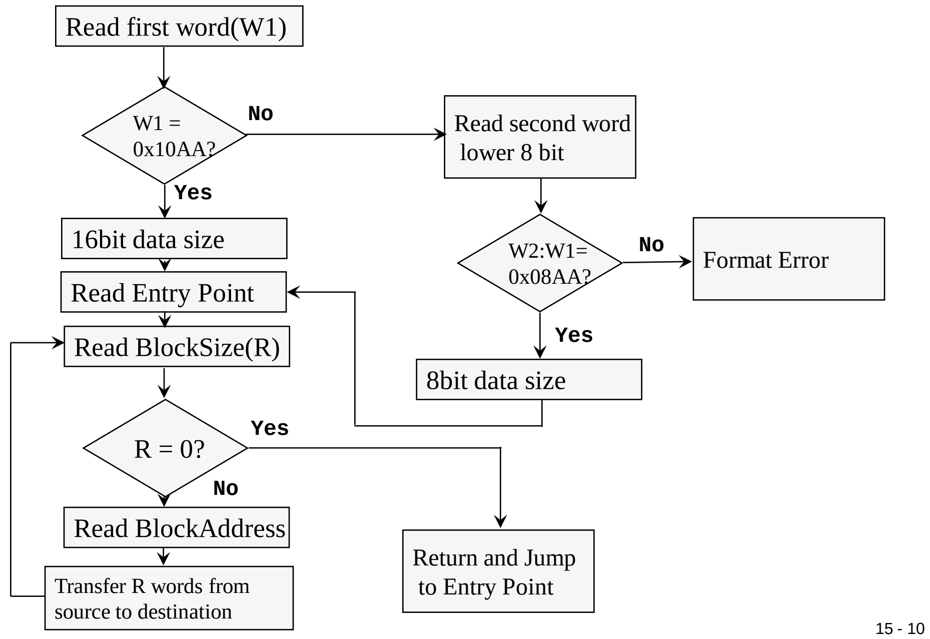
15 - 9

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 detects 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, it 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 the 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.

F2833x Boot Loader Transfer Procedure

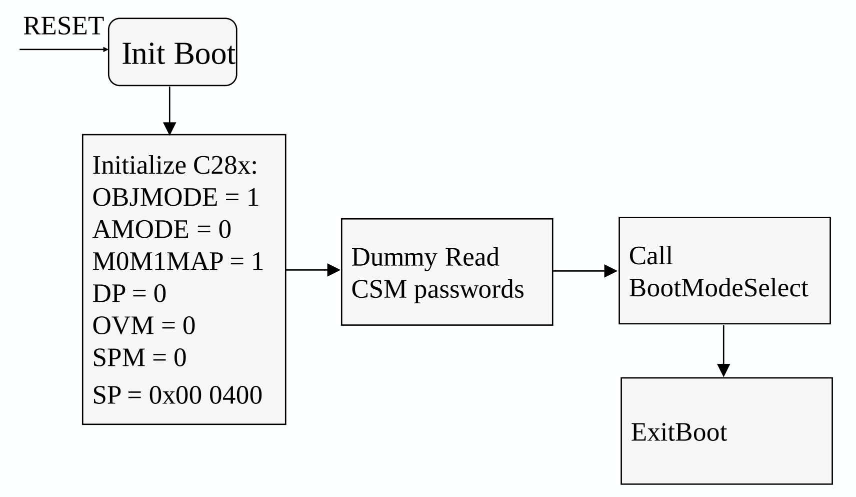


Init Boot Assembly Function

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 F2833x 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 specific 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 entry address that was determined by the boot mode.

F2833x Init Boot Function



15 - 11

SCI Boot Load

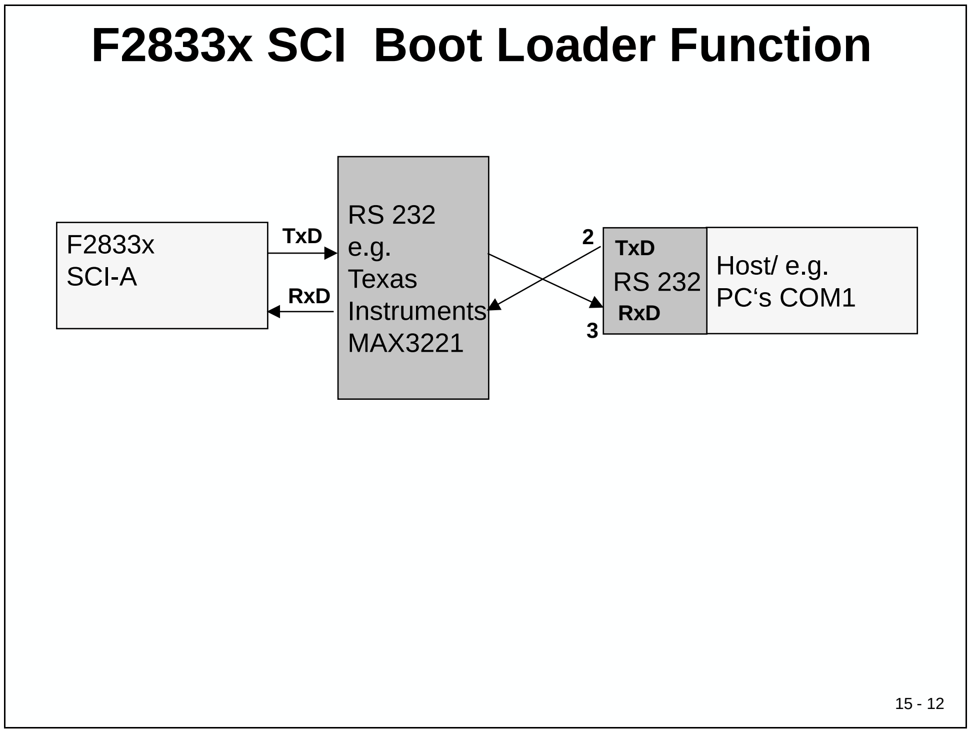
SCI Hardware Connection

The SCI boot mode asynchronously transfers code from SCI-A to the F2833x. 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 an F2833x, you will need to have a RS-232 transceiver interface between the F2833x and the PC to generate the necessary voltages. Fortunately the F28335ControlCard provides such a transceiver, a Texas Instruments MAX2332. If you connect the F2833x directly to the two PC-COM lines you will eventually destroy the F2833x!

If you are not sure about the hardware set up, ask your teacher before you continue with the laboratory exercise at the end of this chapter!



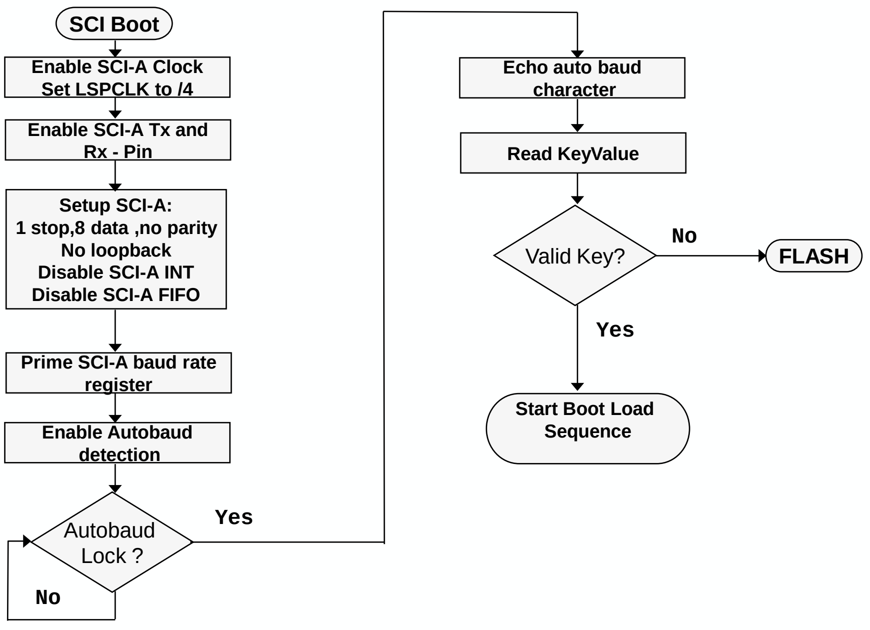
F2833x SCI Boot Loader Function

15 - 12

SCI Boot Loader Function

The flowchart for the SCI interface is shown in Slide 15-13 (below).

F2833x SCI Boot Function


SCI Boot
Enable SCI-A Clock Set LSPCLK to /4
Enable SCI-A Tx and Rx - Pin
Setup SCI-A: 1 stop,8 data ,no parity No loopback Disable SCI-A INT Disable SCI-A FIFO
Prime SCI-A baud rate register
Enable Autobaud detection
Autobaud Lock ?
Yes
No
Echo auto baud character
Read KeyValue
Valid Key?
No
FLASH
Yes
Start Boot Load Sequence

15 - 13

The F2833x 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 to the host. For this reason, the SCI loader is very flexible and the user can select a number of different baud rates to communicate with the DSP.

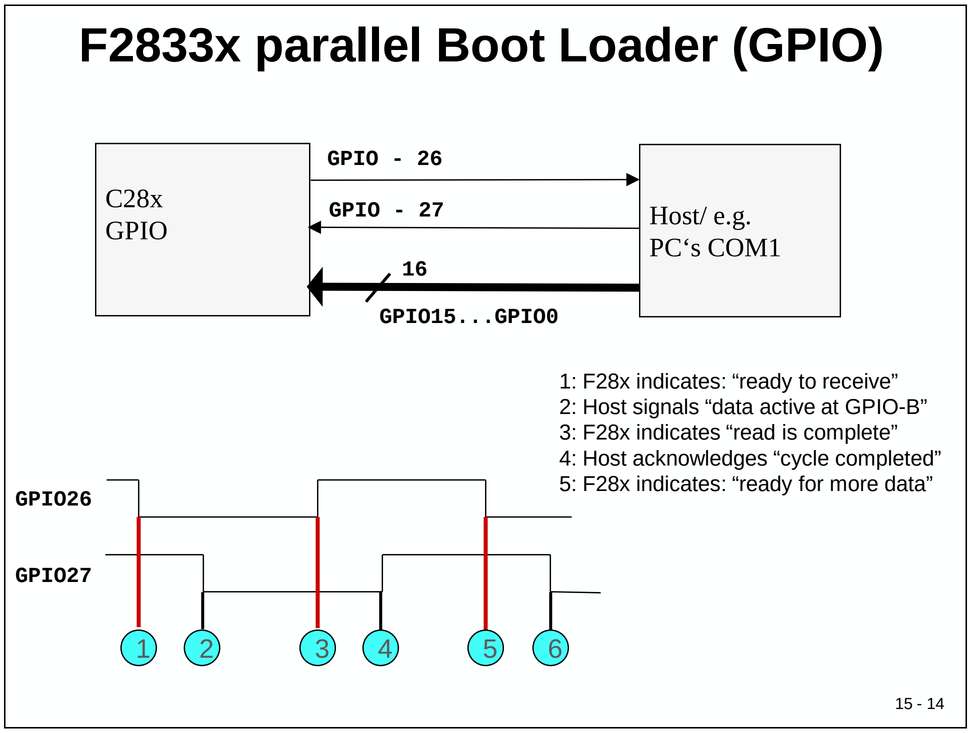
After each data transfer, the DSP will echo back the 8-bit character received to the host. In this manner, the host can perform checks that each character was received correctly by the DSP.

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 k baud) 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 GPIO0 to GPIO15 to internal or XINTF memory. Each value can be 16 bits or 8 bits wide and follows the same data flow as outlined in the data stream structure.



F2833x parallel Boot Loader (GPIO)

1: F28x indicates: "ready to receive"
2: Host signals "data active at GPIO-B"
3: F28x indicates "read is complete"
4: Host acknowledges "cycle completed"
5: F28x indicates: "ready for more data"

The F2833x communicates with the external host device by polling/driving the GPIO26 and GPIO27 lines. The handshake protocol shown in Slide 15-14(above) must be used to successfully transfer each word via GPIO0...GPIO15. This protocol is very robust and allows for a slower or faster host to communicate with the F2833x 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 bytes are read from GPIO0...GPIO7 only.

The DSP first signals to the host that the DSP is ready to start a data transfer by pulling the GPIOD27 pin low. The host load then initiates the data transfer by pulling the GPIOD26 pin low. The complete protocol is shown in the Slide 15-14 (above).

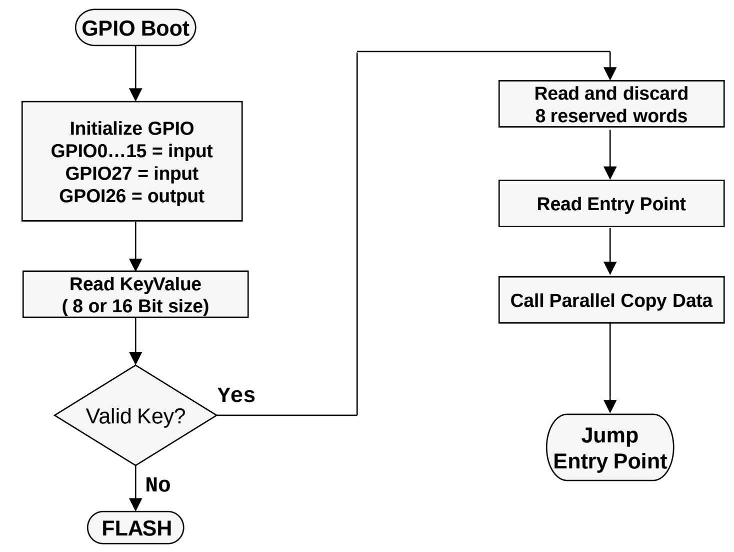
F2833x Software Flow

Slide 15-15 shows a flowchart for the Parallel GPIO boot loader inside the F2833x. After parallel boot has been selected at RESET, GPIO0...GPIO15 are initialized as an input port. The two handshake lines GPIO26 and GPIO27 are initialized as input and output respectively.

Next, the first character is polled from GPIO0...GPIO15. If it is a valid 8-bit (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 15-14.

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

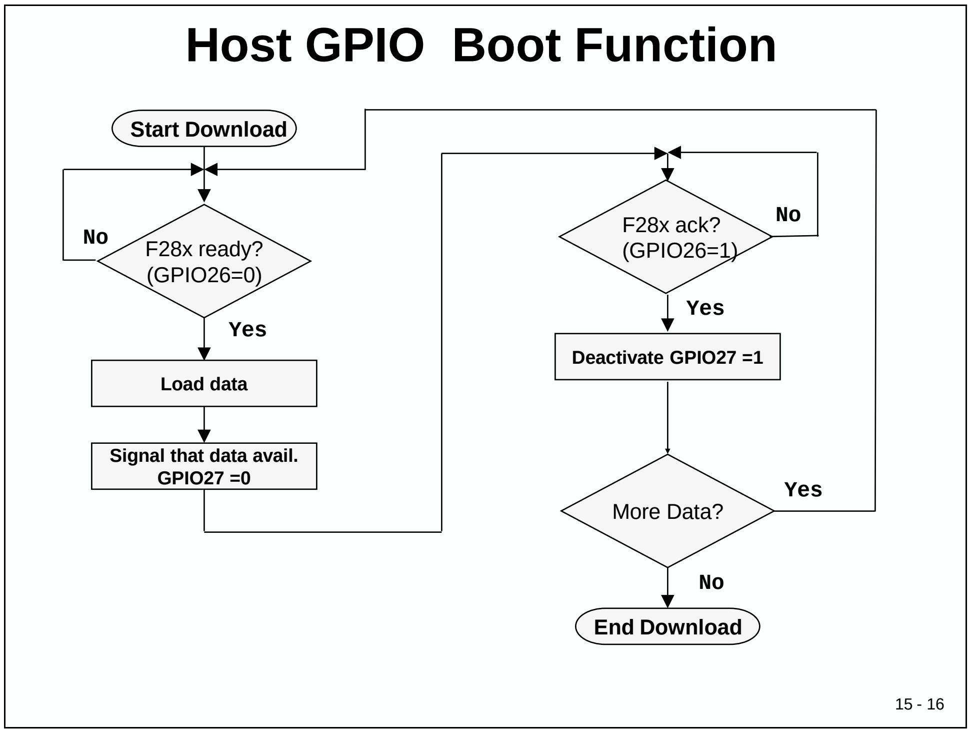
F2833x GPIO Boot Function



15 - 15

Host Software Flow

Slide 15-16 (below) shows the transfer flow from the Host side. The operating speed of the F2833x and Host are not critical in this mode as the host will wait for the F2833x and the F2833x 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 than the F2833x.


Host GPIO Boot Function

First, the host waits for a handshake signal (GPIO26) to be activated (= 0) by the F2833x.

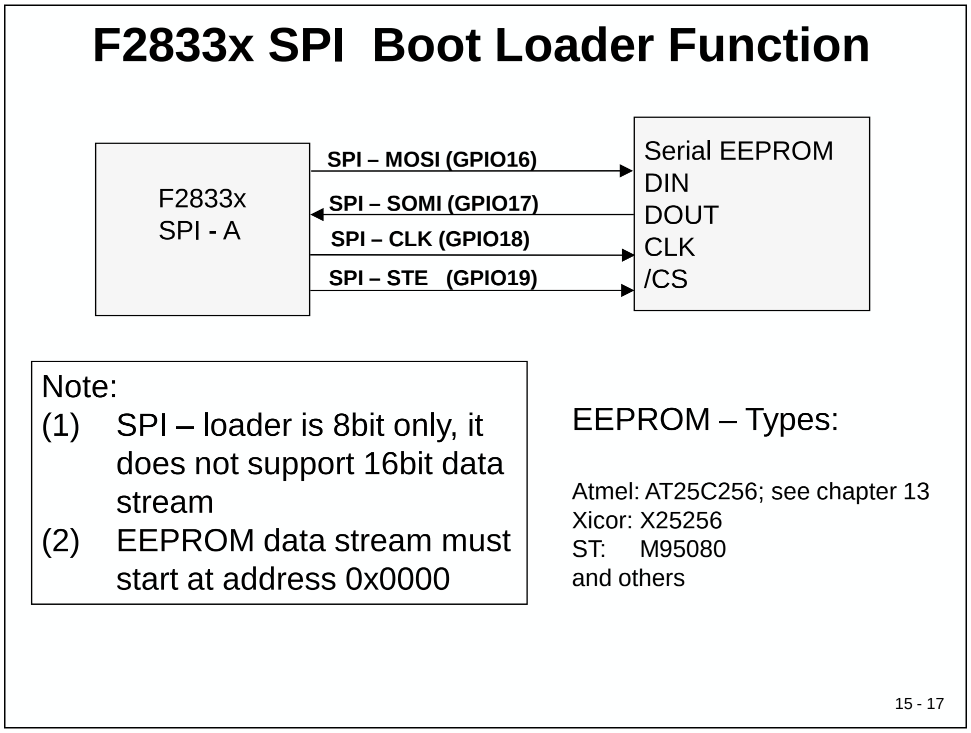
Next, the host has to load the next character onto its parallel output port. The host then acknowledges a valid character by activating (=0) the signal that is connected to the F2833x GPIO27 input line.

The F2833x has now all the time it requires to read the data from GPIO0…GPIO15. Once this has been performed, the F2833x deactivates its output line GPIO26 to inform the host that the transfer cycle is completed.

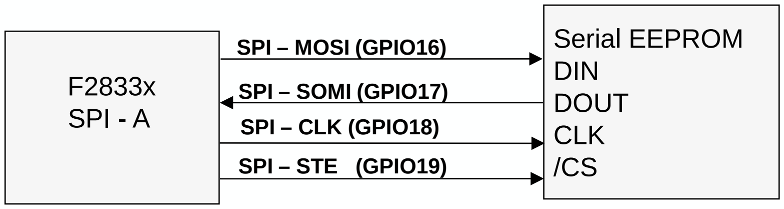
The host acknowledges this situation by deactivating its handshake line (GPIO27). If the algorithm has more data to transmit to the F2833x, the procedure is repeated once more. If not, the download is finished.

SPI Boot Loader

The SPI loader expects an 8-bit wide SPI-compatible serial EEPROM device to be present on the SPI pins as shown in Slide 15-17. The SPI boot loader does not support a 16-bit data stream.



F2833x SPI Boot Loader Function



Note:
(1) SPI – loader is 8bit only, it does not support 16bit data stream
(2) EEPROM data stream must start at address 0x0000

EEPROM – Types:

Atmel: AT25C256; see chapter 13
Xicor: X25256
ST: M95080
and others

15 - 17

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). At the Peripheral Explorer Board, the interface SPI-A is used for the control channel of the audio codec AIC23B, so we cannot experiment directly with the SPI-A boot loader. To do so, we would have to add an external EEPROM to the hardware. Again, ask you teacher, if your university classroom equipment has been enhanced.

An SPI boot loader is widely used in real world projects. Therefore let us discuss the software flow.

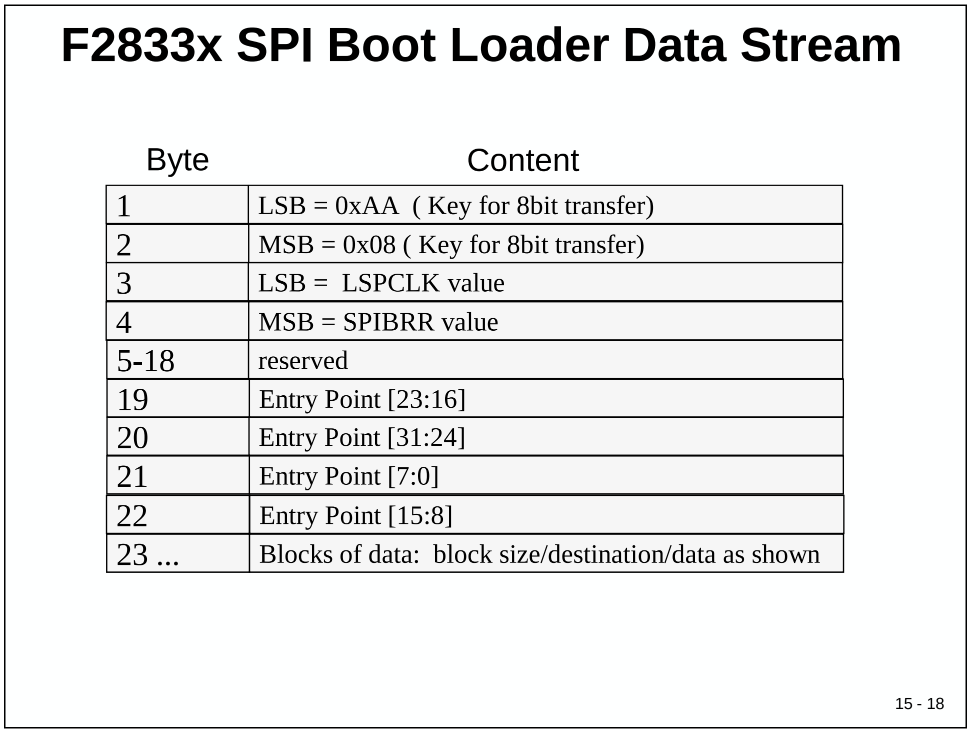
The SPI boot ROM loader initializes the SPI with the following settings: FIFO enabled, 8-bit character, internal SPICLK master mode and talk mode, clock phase = 0, polarity = 0 and slowest baud rate.

If the download is to be preformed from an SPI port on another device, then that device must be set up to operate in 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 function 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 (Slide 15-18) shows the sequence of 8-bit data expected by the Boot Loader.



F2833x 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

15 - 18

SPI Boot Loader Flowchart

The flowchart is shown in Slide 15-18 and Slide 15-19. The data transfer is performed 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 now follows:

1) The SPI-A port is initialized

2) The GPIO19 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

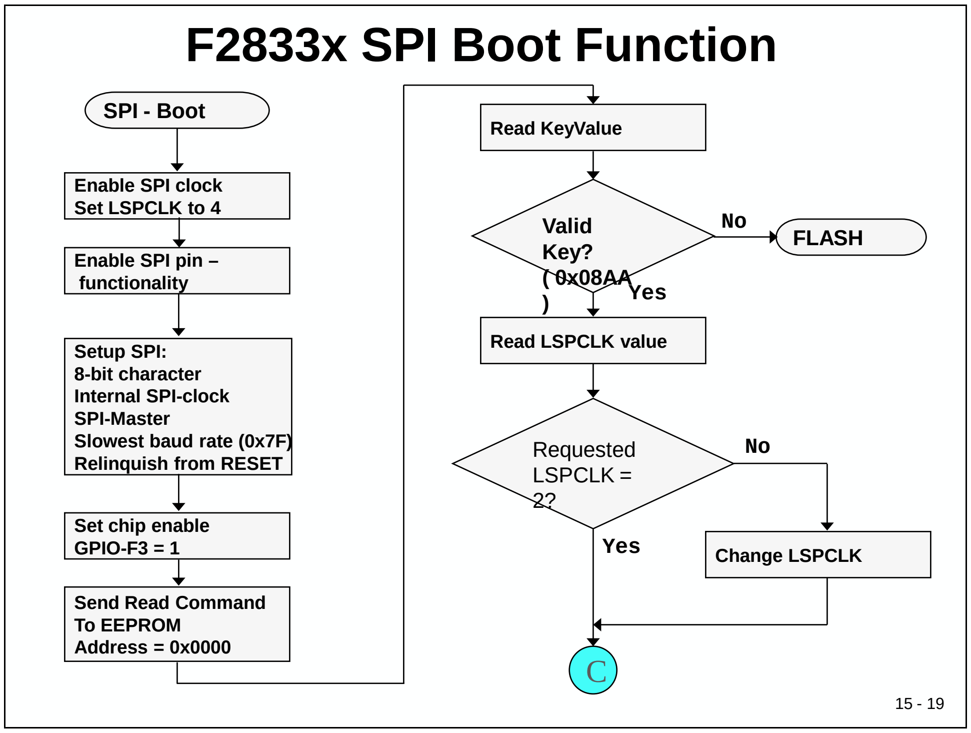
4) The SPI-A interface sends the serial SPI EEPROM address 0x0000; that is, the host requires that the EEPROM must have the downloadable package starting at internal address 0x0000 of the EEPROM.

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.

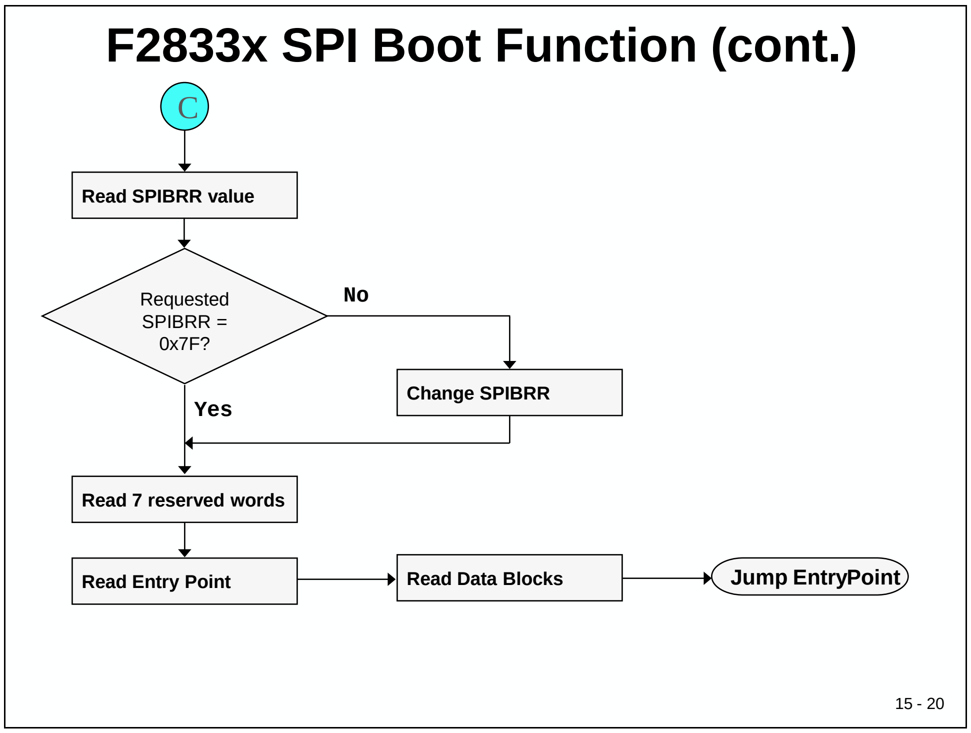
6) The next two bytes fetched can be used to change the value of the low speed peripheral clock register (LOSPCP) and the SPI Baud rate register (SPIBRR). The first byte read is the LOSPCP value and the second byte read is the SPIBRR value. The next seven words are reserved for future enhancements. The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



F2833x SPI Boot Function



F2833x SPI Boot Function (cont.)

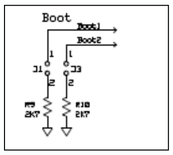
Lab 15_1: Serial Boot Loader SCI-A

Objective

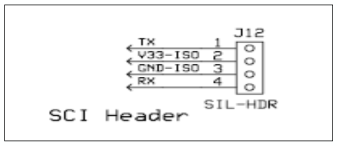
The objective of this laboratory exercise is to practice using the F2833x internal serial boot loader options. In Chapter 9 we discussed the SCI-interface of the F2833x and experimented with some transmit and receive laboratory examples. Let us now use the SCI-A interface to download control code on power ON from a host into the internal RAM of the F2833x and execute this code after the download is completed. This is a typical scenario for distributed control systems, in which a master-node sends control code to slave-nodes.

Lab15_1: SCI – A boot loader

- Use Lab6 ("binary counter") as starting point
- Change the Hardware Boot Sequence from FLASH to SCI -A (close Jumper J3)



- Use a SCI – connection to your PC:



- Use tool "hex2000.exe" to generate download stream
- Use a PC terminal program to download code

15 - 21

Again we will use our solution from Lab 6, the binary counter at LEDs LD1…LD4, as a starting point.

Procedure

Open Project

1. Open your project "Lab6.pjt" from C:\DSP2833x_V4\Labs\Lab6.
2. Open the file "Lab6.c", save it in "C:\DSP2833x_V4\Labs\Lab6" as "Lab15.c".
3. Exclude file "Lab6.c" from Build
4. Open the file "Lab15.c" to edit.

Although there is no need to change the control code of Lab15.c, we should generate a considerably slower frequency for the control code. Remember, that in Lab14 we have programmed the binary counter code, running at 100 milliseconds time steps into the FLASH of the F2833x. Now, to be able to distinguish between the FLASH-code (which should not be active in Lab15) and the RAM-downloaded code, the simplest way is to change the step size of our control code from 100 milliseconds to 1 second. Also, recall that the watchdog unit is active. In "main()", change the code-section to wait for the next control step into:

```
while(CpuTimer0.InterruptCount < 10)
{
        EALLOW;
        SysCtrlRegs.WDKEY = 0x55;       // service WD #1
        EDIS;
}
CpuTimer0.InterruptCount = 0;
```

Build, Load and Run

5. Click the "Rebuild Active Project " button or perform:

   Project → Rebuild All (Alt +B)

6. Load the output file in the debugger session:

   Target → Debug Active Project

   and switch into the "Debug" perspective.

7. Perform a real time run.

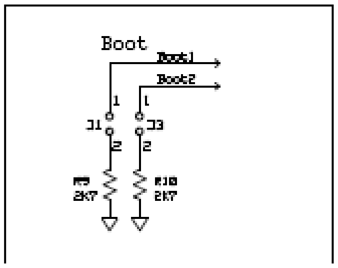
   Target ➔ Run

Verify that the control code (binary counter at LD1…LD4) is now running at a step size of 1 second. If this is true, we have a working code example, which can be used to extract the necessary download modules.

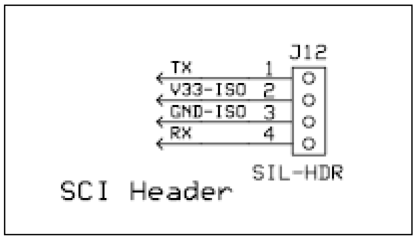
Verify that the file "Lab6.out" exists in subdirectory '\Debug', and that it is up to date.

Change Hardware set up

8. Close Code Composer Studio and switch off the power supply of the Peripheral Explorer Board.
9. Close jumper J3 ("SCI-BOOT 84") at the Peripheral Explorer Board. This will select the SCI-A boot loader option of the F2833x.



10. Connect the Peripheral Explorer Board SCI-A (header J12) with a serial COM-channel of your PC. Plug in the serial cable provided to header J12 making sure the red wire aligns with the Rx pin on the peripheral explorer kit.



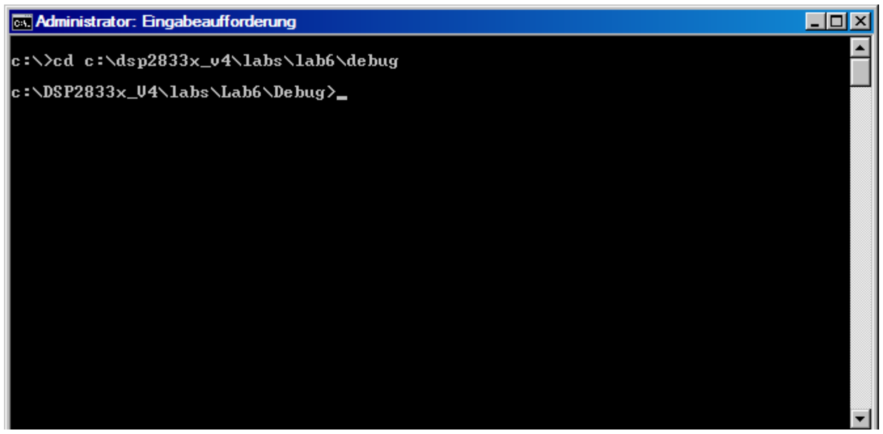
11. Re-power ON the Peripheral Explorer Board. The binary counter code at LEDs LD1…LD4 should not run.

Generate download data stream

12. Texas Instruments provides a very useful tool, called "hex2000.exe", to convert data from COFF-format ("Lab6.out") into any other format, including binary images, Intel-hex file format and many others. We can use this tool to generate the data stream accordingly to the serial download format, which we discussed at the beginning of this chapter. Unfortunately, this tool is available as a command line version only. If you recall the old days from Microsoft-DOS, you should be familiar with the command line window control. As a modern-day student, you are probably too young for such cryptic syntax, so here is an explanation of what to do:

- In Windows-XP or Windows-Vista, open a command line window ("cmd.exe").
- In this window, enter the path to the location of file "Lab6.out". The actual path depends on your PC-installation. Note: use the Windows-Explorer to locate the location on your PC. On my computer it is:

  cd C:\DSP2833x_V4\labs\Lab6\Debug

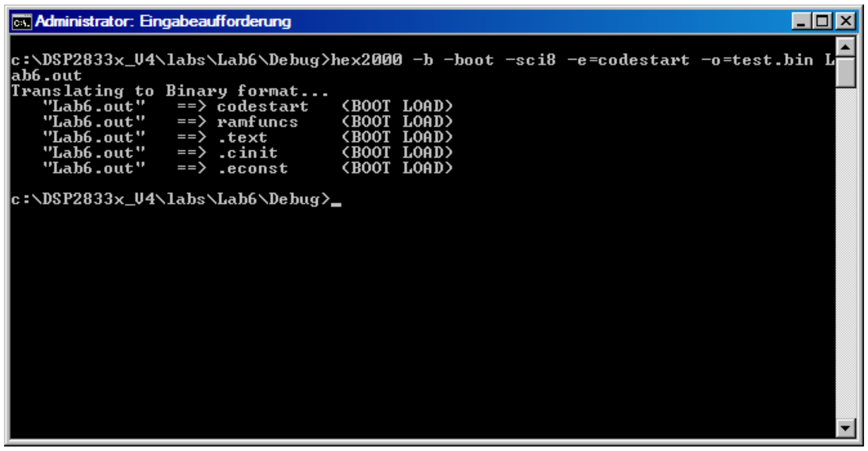


```
c:\>cd c:\dsp2833x_v4\labs\lab6\debug

c:\DSP2833x_V4\labs\Lab6\Debug>_
```

- To start the “hex2000” - tool, first search the harddisk location of the file “hex2000.exe”
- Next, using the Windows Explorer, copy the file “hex2000.exe” into the directory of your project, e.g. **C:\DSP2833x_V4\labs\Lab6\Debug.**
- Now enter the following command as a single line into the command window:

hex2000 -b -boot -sci8 -e=codestart -o=test.bin Lab6.out



```
c:\DSP2833x_V4\labs\Lab6\Debug>hex2000 -b -boot -sci8 -e=codestart -o=test.bin L
ab6.out
Translating to Binary format...
   "Lab6.out"   ==> codestart    (BOOT LOAD)
   "Lab6.out"   ==> ramfuncs     (BOOT LOAD)
   "Lab6.out"   ==> .text        (BOOT LOAD)
   "Lab6.out"   ==> .cinit       (BOOT LOAD)
   "Lab6.out"   ==> .econst      (BOOT LOAD)

c:\DSP2833x_V4\labs\Lab6\Debug>_
```

Here is an explanation of what we did:

The tool “hex2000.exe” used the input file “Lab6.out” to generate a new output file “test.bin”. Both the name “test” and extension “bin” are arbitrary and used just as examples. The switch “-b” told the tool to perform a binary extraction. The switches “-boot” and “-sci8” have been used to generate a file structure according the boot loader sequence for SCI-mode. The option “-e =codestart” has been used to specify the start address after the download sequence is completed. We used the symbol “codestart” from file “DSP2833x_CodeStartBranch.asm”.

The screenshot above tells us that the tool has generated binary sections for

- the codestart - section (codestart)

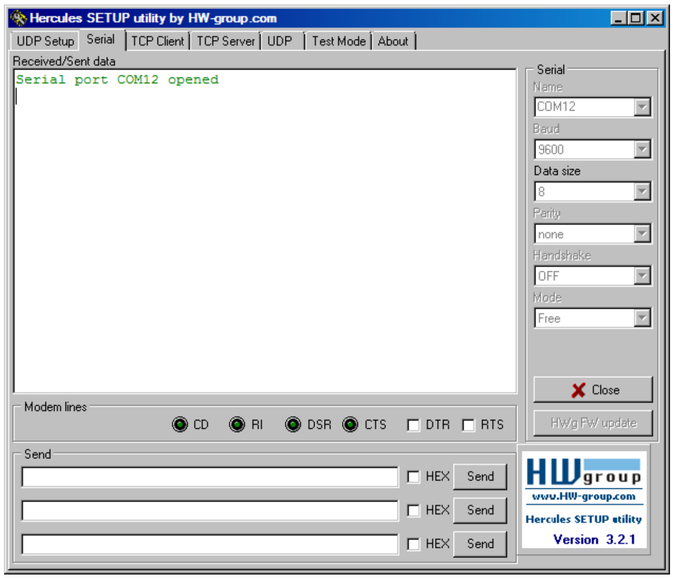
- the default machine-code section (.text)
- a section for functions with different Load and Run-addresses (ramfuncs)
- a section for global initialization constants (.cinit)
- a section for constant variables (.econst)

Verify, that you have a new file "test.bin" in your "Debug"-directory!

Download Image into the target

13. For this step we need a serial terminal program running under windows. If you still use XP, you can use the program "Windows-HyperTerminal". Under Windows-Vista, there is no such a tool. However, there are a few similar freeware tools available, such as "Hercules" (www.HW-group.com)
    - Open your terminal program and enter the following initial parameters:
        - 9600 bit/s
        - 8 data bits
        - no parity bit
        - 1 stop bit
        - no hardware handshake or flow control

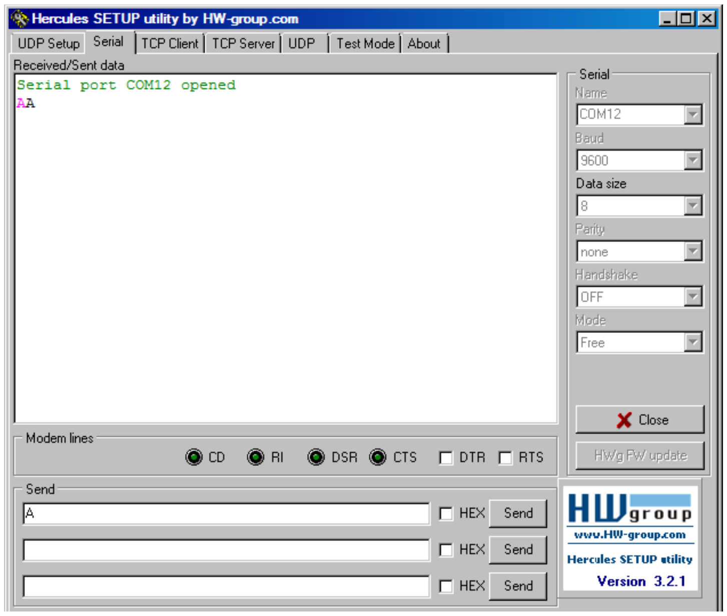
    Here is an example for "Hercules":



```
Serial port COM12 opened
```

Serial
Name: COM12
Baud: 9600
Data size: 8
Parity: none
Handshake: OFF
Mode: Free

    - Next, send a single character 'A', which is used for auto baud rate detection in the serial boot loader code of the F2833x. The F2833x will immediately respond with an echo of 'A' to indicate successful auto baud rate detection.

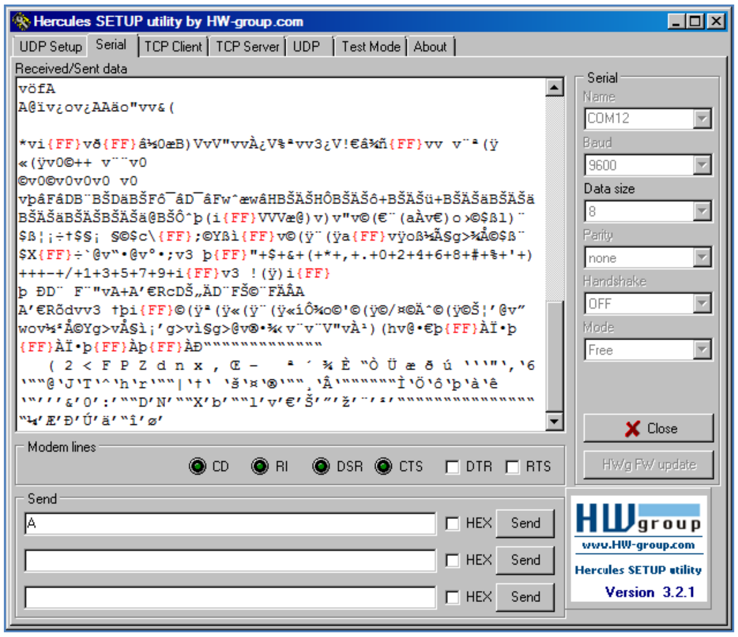


```
Serial port COM12 opened
AA
```

Serial: Name COM12, Baud 9600, Data size 8, Parity none, Handshake OFF, Mode Free

Send: A

- Finally, send the file “test.bin” to the F2833x. Right mouse click in “Hercules”, select “Send File” and browse to the location of “test.bin”. Do not worry about the strange output, the F2833x echoes back all bytes and since we are transmitting a binary image, only a few of them are printable:



```
vöfA
A@ïv¿ov¿AAäo"vv&(

*vi{FF}vð{FF}â½0æB)VvV"vvÀ¿V%ªvv3¿V!€â¾ñ{FF}vv v¨ª(ÿ
«(ÿv0©++ v¨¨v0
©v0©v0v0v0 v0
vþâFâDB¨BŠDäBŠFô¯âD¯âFw^æwâHBŠÄŠHÔBŠÄŠô+BŠÄŠü+BŠÄŠäBŠÄŠä
BŠÄŠäBŠÄŠBŠÄŠä@BŠÔ^þ(i{FF}VVVæ@)v)v"v©(€¨(aÀv€)o›©$ß1)¨
$ß¦¡÷†$§¡ §©$c\{FF};©Yßì{FF}v©(ÿ¨(ÿa{FF}vÿoß¼Ã§g>¾Å©$ß¨
$X{FF}÷`@v"•@v°•;v3 þ{FF}"+$+&+(+*+,+.+0+2+4+6+8+#+%+'+)
+++-+/+1+3+5+7+9+i{FF}v3 !(ÿ)i{FF}
þ ÐD¨ F¨"vA+A'€RcDŠ„ÄD¨FŠ©¨FÄÂA
A'€Rõdvv3 †þi{FF}©(ÿª(ÿ«(ÿ¨(ÿ«íÔ‰o©'©(ÿ©/¤©Ä^©(ÿ©Š¦'@v"
wov¼ªÅ©Yg>vÅ§i¡'g>vi§g>@v®•¾‹v¨v¨V"vÀ¹)(hv@•€þ{FF}ÀÏ•þ
{FF}ÀÏ•þ{FF}Àþ{FF}ÀÐ"""""""""""""""
   ( 2 < F P Z d n x , Œ –   ª ´ ¾ È "Ò Ü æ ð ú ''''"',‘6
'""@'J'T'^'h'r'""|'†' 'ă'¤'®'"",'Â'"""""""Ì'Ö'ô'þ'à'ê
'"'''&'0':'""D'N'""X'b'""l'v'€'Š'"'ž'¨'ª'"""""""""""""""""
"¼'Æ'Ð'Ú'ä'"î'ø'
```

Serial: Name COM12, Baud 9600, Data size 8, Parity none, Handshake OFF, Mode Free

Send: A

- At the end of the download sequence, the boot-loader code will branch directly into the code entry point “codestart”; our downloaded control code is running!

END of Lab15_1