Notes on using Analog Devices' DSP components and development tools
Phone: (800) ANALOG-D or (781) 461-3881, FAX: (781) 461-3010, EMAIL: dsp.support@analog.com

Contributed by T. Lorenzen, European DSP Applications (Sept. 01)

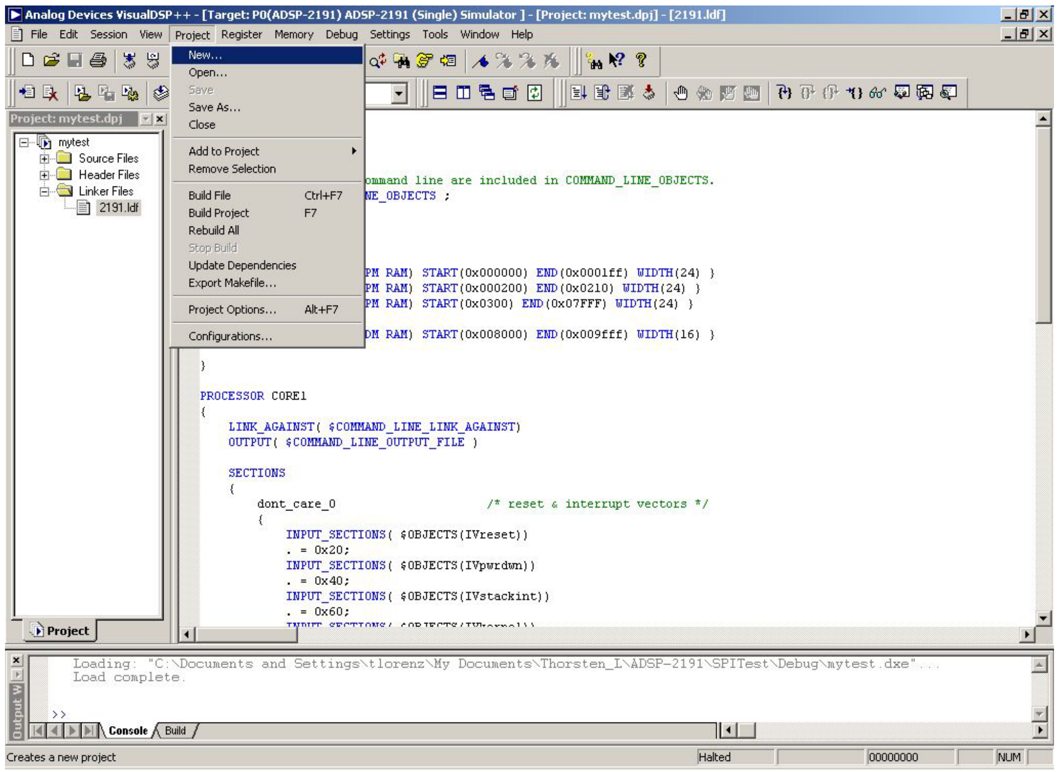
SPI Booting of the ADSP-2191 using the Atmel AT25020N on an EZ-KIT LITE Evaluation Board

Introduction:

Analog Devices ADSP-2191 is the first DSP that provides booting via **SPI**. This note shows how to interface an ATMEL EEPROM (AT25020) in order to boot the ADSP-2191 via the SPI interface. With the help of a little project, this app note will describe how to create a loader file which is stored in the EEPROM and used to boot the DSP. The loader file format as well as the Hardware will be covered in order to build the whole project.

VISUALDSP++™ 2.0:

The latest version of VisualDSP++2.0 creates loader files suitable for SPI booting (Serial Peripheral Interface) automatically. Open Analog Devices VisualDSP++2.0 and make a new project. Add the linker description file (LDF) of the ADSP-2191 to the project first. This file defines the entire address range of the DSP and identifiers individual sections of memory with labels.



```
ommand line are included in COMMAND_LINE_OBJECTS.
NE_OBJECTS ;

PM RAM) START(0x000000) END(0x0001ff) WIDTH(24) }
PM RAM) START(0x000200) END(0x0210) WIDTH(24) }
PM RAM) START(0x0300) END(0x07FFF) WIDTH(24) }

DM RAM) START(0x008000) END(0x009fff) WIDTH(16) }

}

PROCESSOR CORE1
{
    LINK_AGAINST( $COMMAND_LINE_LINK_AGAINST)
    OUTPUT( $COMMAND_LINE_OUTPUT_FILE )

    SECTIONS
    {
        dont_care_0                     /* reset & interrupt vectors */
        {
            INPUT_SECTIONS( $OBJECTS(IVreset))
            . = 0x20;
            INPUT_SECTIONS( $OBJECTS(IVpwrdwn))
            . = 0x40;
            INPUT_SECTIONS( $OBJECTS(IVstackint))
            . = 0x60;
```

```
Loading: "C:\Documents and Settings\tlorenz\My Documents\Thorsten_L\ADSP-2191\SPITest\Debug\mytest.dxe"...
Load complete.

>>
```

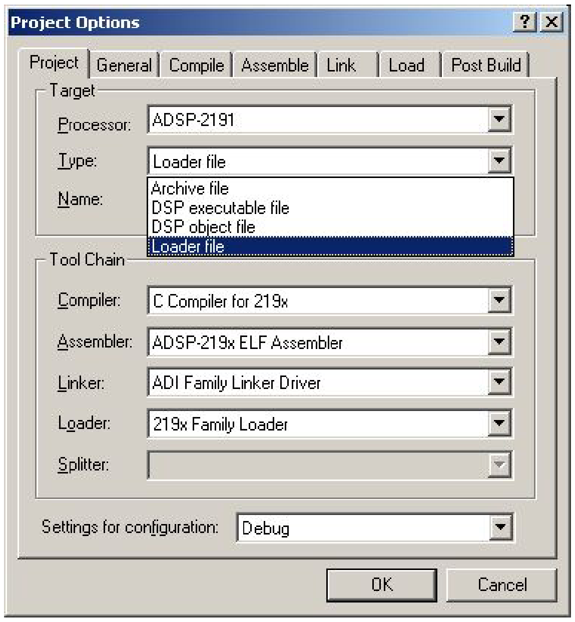
Figure 1 create a new project with VisualDSP++ 2.0

Finally, add an existing assembly file of your choice (or create a new asm file to add) and the header file "def2191.h" which defines all of the ADSP-2191's memory-mapped registers. If the ADSP-2191 processor is selected the project can now be build.

This way VisualDSP++2.0 will build an executable file "filename.dxe" which is required for debugging your program on the simulator or loading it to and debugging it on the DSP via the emulator.

Note:
Ensure you have the most recent loader "elfloader.exe". Find one in the zip file.

To prepare VisualDSP++ 2.0 SPI booting please open the "Project Options" dialog box from the "Project" menu. As shown in figure 2, on the project option tab change the option "type" in the "target" frame from "DSP executable file" to "Loader file". This causes VisualDSP++ 2.0 to create a loader file which can be stored in non-volatile memory of your choice. The ADSP-2191 is able to boot from different memory types. To select the appropriate booting device, select the "Loader" tab control in the "Project Options" box and set the Boot Type" to "SPI". Select binary format in the "Format" frame to create a binary file that is suitable for the most common programmers as shown in figure 3. The ASCII format lets the VisualDSP++ 2.0 loader "elfloader.exe" build a loader file in ASCII format. This file can be accessed by any text editor easily. The binary file can be accessed by using a binary editor only. Finally, press the "OK" button on the dialog "Project Options" and again rebuild the whole project again. A file with the same name as the project and the extension .ldr will be generated. This .ldr file can be loaded by a programmer and written to the booting device (in this case to the AT25020 EEPROM) straight forward.

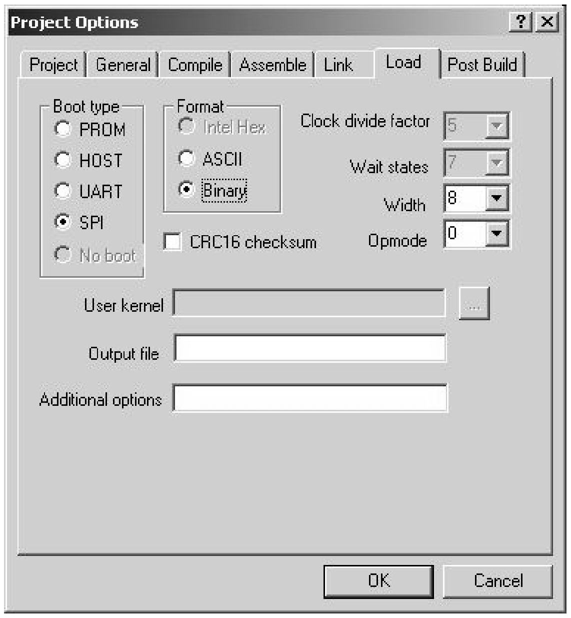

Project Options
Tabs: Project | General | Compile | Assemble | Link | Load | Post Build

Target
Processor: ADSP-2191
Type: Loader file
Name:
Type list: Archive file, DSP executable file, DSP object file, Loader file

Tool Chain
Compiler: C Compiler for 219x
Assembler: ADSP-219x ELF Assembler
Linker: ADI Family Linker Driver
Loader: 219x Family Loader
Splitter:

Settings for configuration: Debug

Figure 2 "Project" options dedicated to your project


Project Options
Tabs: Project | General | Compile | Assemble | Link | Load | Post Build

Boot type: PROM / HOST / UART / [✓ SPI] / No boot
Format: Intel Hex / ASCII / [✓ Binary]
CRC16 checksum: unchecked
Clock divide factor: 5
Wait states: 7
Width: 8
Opmode: 0
User kernel:
Output file:
Additional options:

Figure 3 "Loader" options dedicated to your project

ADSP-2191 Startup basics:

The ADSP-2191 has a booting scheme that is different from former ADI DSPs. The boot kernel is located on-chip and stored in a 24-bit wide, 1K ROM. After a hardware reset the processor starts running at address FF0000h. Automatically the boot kernel will be processed. There are three input pins on the ADSP-2191 whose input state upon hard /RESET determines the booting mode. The state of the three pins (OPMODE, BMODE0 and BMODE1) are sampled on the rising edge of /RESET and are captured into the corresponding bits (0,1 and 2) of the System Configuration Register by software (boot kernel). In the case of SPI booting the kernel branches to the SPI booting subroutine. It initializes the SPI port using SPISS0 and sends the first commands to the SPI EEPOM. Referring to the AT25020 data sheet the first byte sent to the EEPROM in order to receive data is the value "0x3" (Read Timing). Followed by this command the starting address is sent (0x0 in this case). For more information see, section titled *Hardware investigations*. Received data coming from the serial EEPROM will then be unpacked and stored in the appropriate memory locations as the following section will explain.

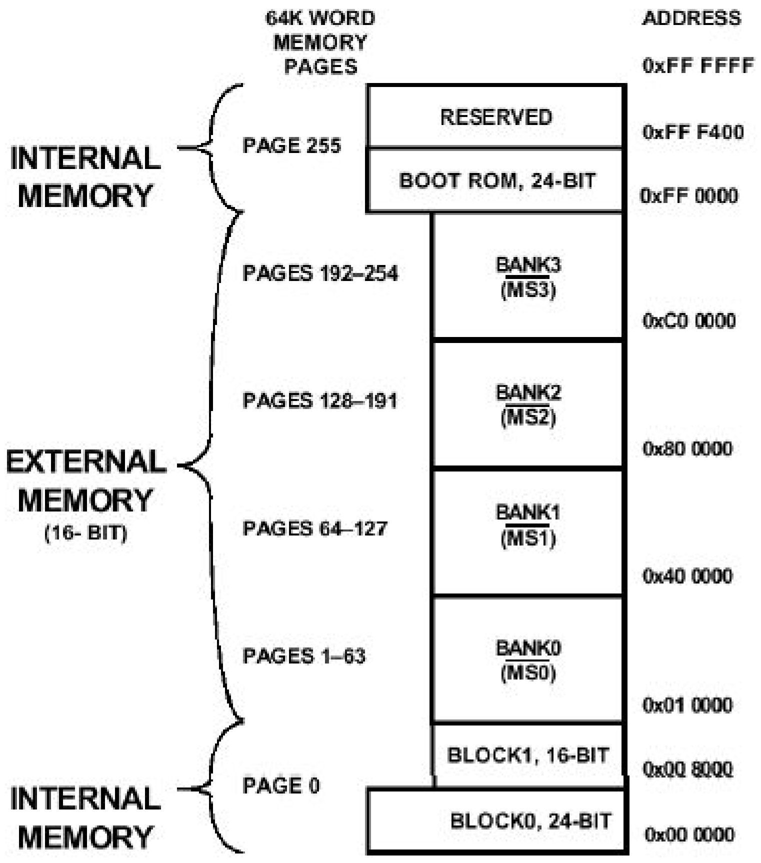


Figure 4 Memory Map of the ADSP2191

Serial Peripheral Interface (SPI):

The ADSP-2191 serial peripheral interface is an industry standard synchronous serial link that helps the DSP communicate with multiple SPI-compatible devices. The SPI peripheral is a synchronous, 4-wire inter-face consisting of two data pins, MOSI (Master Out Slave In) and MISO (Master In Slave Out); one device select pin, SPISS (SPI Slave Select); and a gated clock pin, SCK (Serial Clock). With the two data pins, it allows for full-duplex operation to other SPI-compatible devices. The SPI also includes programmable baud rates, clock phase, and clock polarity. The SPI can operate in a multi-master environment by interfacing with several other devices, acting as either a master device or a slave device. In a multi-master environment, the SPI interface uses open drain data pad driver outputs to avoid data bus contention. Figure 5 provides a block diagram of the ADSP-2191 SPI Interface. The interface is essentially a shift register that serially transmits and receives data bits, one bit a time at the SCK rate, to/from other SPI devices. SPI data is transmitted and received at the same time through the use of a shift register. When an SPI transfer occurs, data is simultaneously transmitted, or shifted out serially via the shift register, as new data is received, or shifted in serially at the other end of the same shift register. The SCK synchronizes the shifting and sampling of the data on the two serial data pins, MOSI and MISO.

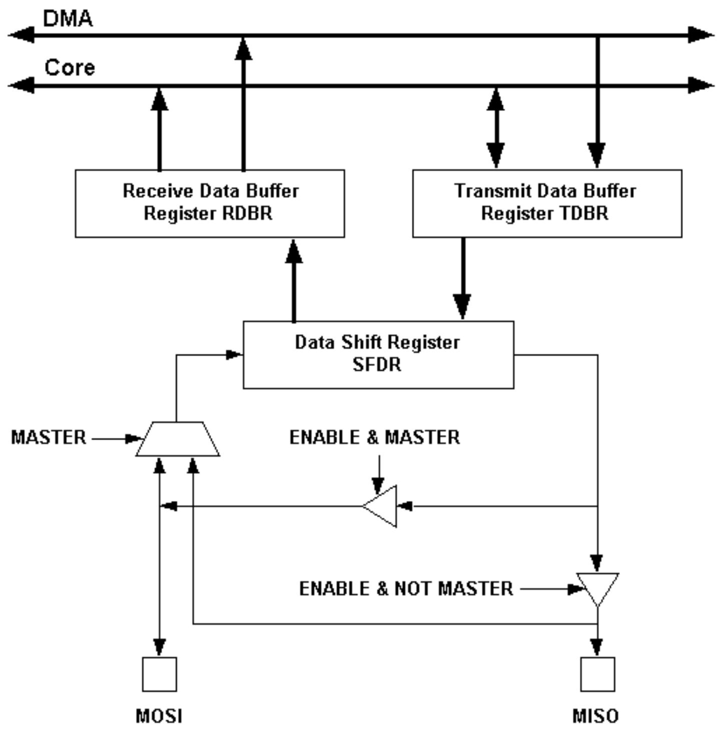


Figure 5 ADSP-2191 SPI Block Diagram

Boot stream format:

During booting the kernel follows a protocol to decode the loader file as shown below. The data received from the SPI EEPROM is eight bits wide. Each byte received will be checked, shifted together and than stored in memory. The processor stores the Header (Control Word, Flag (or known as **TAG**) , Start Address, Page and Word Count) in data memory at first. With help of these contents the processor knows where to store the instructions or data.

The Control Word will be stored in memory at the beginning of the boot sequence, only. But these values are ignored completely during SPI booting. It can just be booted in 8 bit mode as well as with having no wait states are required.

Each instruction or data block starts with a flag. This flag includes whether to store the instr./data in DM or PM. As well as zero filling or final block detection.

The start address indicates the location of the first DM or PM word in memory. The address will be incremented by 1 for all following words automatically.

The page extents the address in order to access more memory.

Instructions are placed in PM . one by one, consecutively. Word Count indicates the number of instructions following this header.

Finally the instructions, just as many as instr. are countered in Word Count as many 24 bits Data Words are expected. Beginning after Word Count and placed in memory dedicated to the Flag Word and starting at the address held in Start Address.

Flag	*Function*
0x00	24 bit PM
0x01	16 bit DM
0x02	Final PM
0x03	Final DM
0x04	Zero-init PM
0x05	Zero-init DM
0x06	Zero-init Final PM
0x07	Zero-init Final DM
0x08	Reserved

Table 1 Boot Flags (Tags)

The following Flag indicates the next instr./data words storing in different areas.

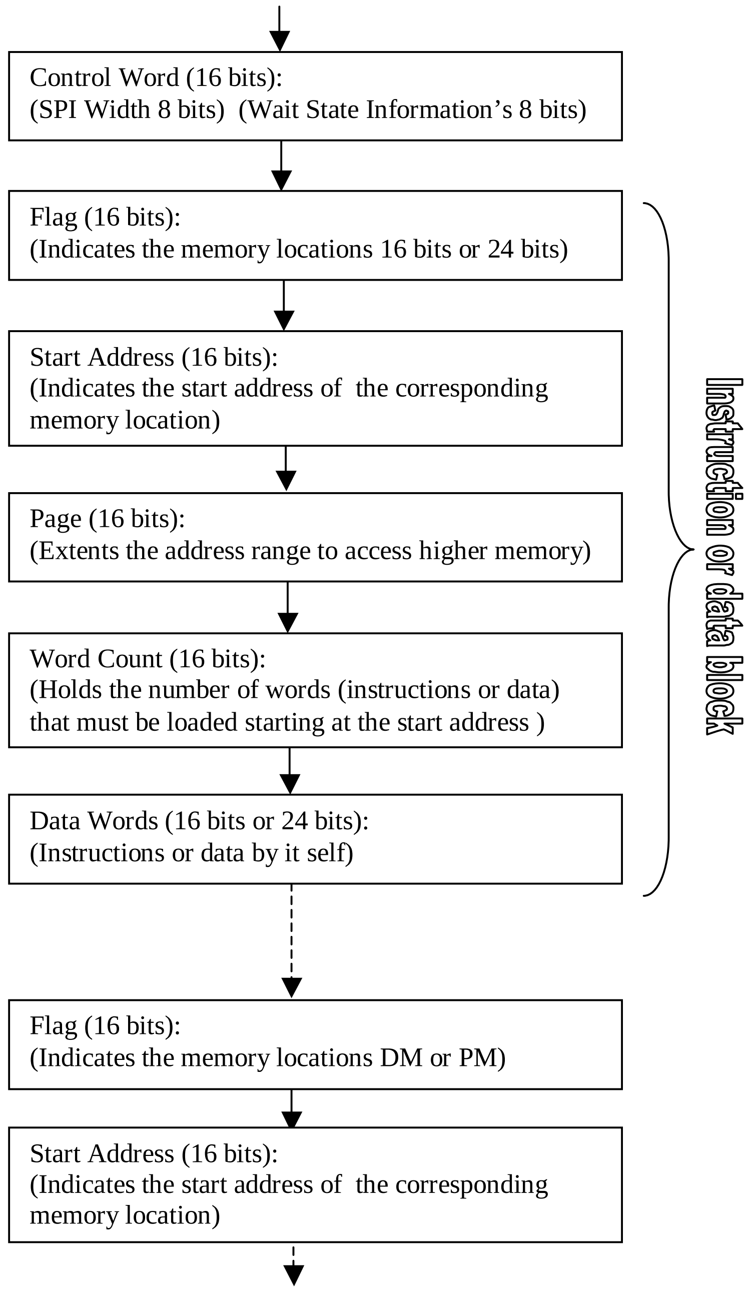

Control Word (16 bits):
(SPI Width 8 bits) (Wait State Information's 8 bits)
Flag (16 bits):
(Indicates the memory locations 16 bits or 24 bits)
Start Address (16 bits):
(Indicates the start address of the corresponding memory location)
Page (16 bits):
(Extents the address range to access higher memory)
Word Count (16 bits):
(Holds the number of words (instructions or data) that must be loaded starting at the start address)
Data Words (16 bits or 24 bits):
(Instructions or data by it self)
Instruction or data block
Flag (16 bits):
(Indicates the memory locations DM or PM)
Start Address (16 bits):
(Indicates the start address of the corresponding memory location)

Figure 6 Boot stream format

Format of the loader file "Filename.ldr":

This section will describe the loader file that has been build by VisualDSP++ 2.0. The whole file will be analyzed and will explain how the processor does unpacks the contents in order to place the code and data to the right location in DSPs memory.

Figure 7 shows the file in ASCII format. The programmer has to place the data starting at address 0x0 in the EEPROM. Each line (address) contains one byte (byte wise organized). The next job is to unpack the data by the boot kernel as shown in table 2.

```
Address 0x0:
2F          0E          C0
00          00          15
00          00          00
00          30          C0
00          50          15
00          14          00
00          00          C0
00          50          15
01          08          00
00          00          C0
00          50          15
20          00          00
1C          00          00
04          40          00
00          00          F3
01          10          FF
00          06          1F
00          00          07
00          C0          00
DF          15          F0
01          00          FE
00          C0          00
00          15          00
00          00          10
02          C0          01
00          15
00          00
```

Figure 7 Loader file created by VisualDSP++ 2.0

Table 2, figure 7 and figure 8 show how the booted data will be shifted together and placed in memory correctly. The first byte received from the EEPROM is 0x02F (Wait State) using the loader file above. This byte will be stored in a 16 bit memory space. The following zero (Width) at the next address. The first word of the header (Flag) will be loaded now as it can be seen in table 2. Two bytes must be shifted together in order to form the first sixteen bit value. The LSByte is received first and placed into the shifter of the DSP . The MSByte at next and shifted to the LSB. The 16 bit result is stored into a place in data memory. The whole header is unpacked the same way. After the header is completely transferred the first instruction following the header is unpacked and placed in memory. It works as follows.

Decode:

Flag:	data is 24 bits wide and must be placed in PM.
Address:	start address is 0x0
Page:	Internal memory space
Word Count:	only one instruction

The same procedure starts at the next flag again.

Received Bytes	
D7 D0	
Wait States	
Data Width	
LSB of the Flag	
MSB of the Flag	
LSB of the Address	
MSB of the Address	Header
LSB of the Page	
MSB of the Page	
LSB of Word Count	
MSB of Word Count	
LSB of Instr./data	
8-15 of Instr./data	Data
MSB of Instr./data	

Table 2 Unpacking

2F	Wait State
00	Width
00 00	Flag 24 bit data PM
00 00	Address
00 00	Page
00 01	Word count
1C 20 00	Instruction //Jump Start
00 04	Flag Zero init PM
00 01	Address
00 00	Page
01 DF	Word count
00 00	Flag 24 bit data PM
02 00	Address
00 00	Page
00 0E	Word count
50 30 00	Instruction //i0 = 0x300;
50 00 14	Instruction //m0 = 0x0001;
50 00 08	Instruction //l0 = 0x000;
40 00 00	Instruction //ax0 = 0x0000;
06 10 00	Instruction //reg(b0) = ax0;
15 C0 00	Instruction //ax0 = pm(i0+=m0);
15 C0 00	Instruction //ax0 = pm(i0+=m0);
15 C0 00	Instruction //ax0 = pm(i0+=m0);
15 C0 00	Instruction //ax0 = pm(i0+=m0);
15 C0 00	Instruction //ax0 = pm(i0+=m0);
15 C0 00	Instruction //ax0 = pm(i0+=m0);
15 C0 00	Instruction //ax0 = pm(i0+=m0);
00 00 00	Instruction //NOP;
1F FF F3	Instruction //Jump Loop1;
00 07	Flag Zero init final DM
FE F0	Address
00 00	Page
01 10	Word count //272 locations

Figure 8 Loader file decoded (DSP format)

The next block is implemented to fill a defined area with zeros. In order to save EEPROM memory space zero filling will be done automatically. As shown in Table 1 flags will force the boot kernel to fill a certain memory area by zeros. No data must be added to the header. The header includes start address and word count to specify the area to be filled. After this is done the next header can be processed.

By convention, the final block is always of the type "zero fill DM". This and the contents of the flag (0x07) lets the DSP jump to zero at the internal PM space to start executing the loaded code.

Hardware investigations:
The AT25020 can be connected to the DSP easily as shown in figure 9. This EE-Note is dedicated to the ADDS-2191M-EZLITE to connect the SPI EEPROM. The Bread Board Connector P9 offers all the signals are required for the connection.

Note:
A ceramic capacitor (100nF) connected to the power supply and placed very close to the device (AT25020) to decouple.
Four pull up resistors (10k) connected to the pins ensure a defined state of the DSP pins.

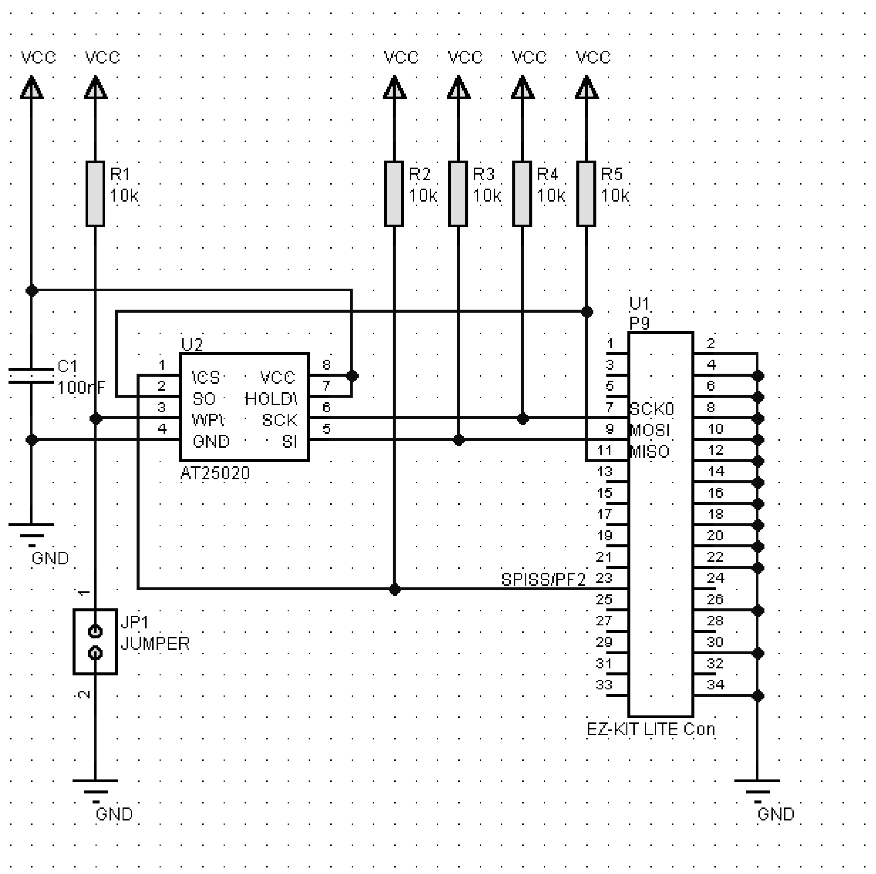


Figure 9 Schematic of the circuit placed on the Bread Board Area

The diagrams illustrates how the EEPROM can be accessed. After reset the DSP initializes the SPI Port 0, asserts the chip select line (SPISS0) and starts the transfer. As it can be seen at the clock line the DSP is configured to transfer data eight bits wide. The Bit rate is set to run at 122.0 kHz.

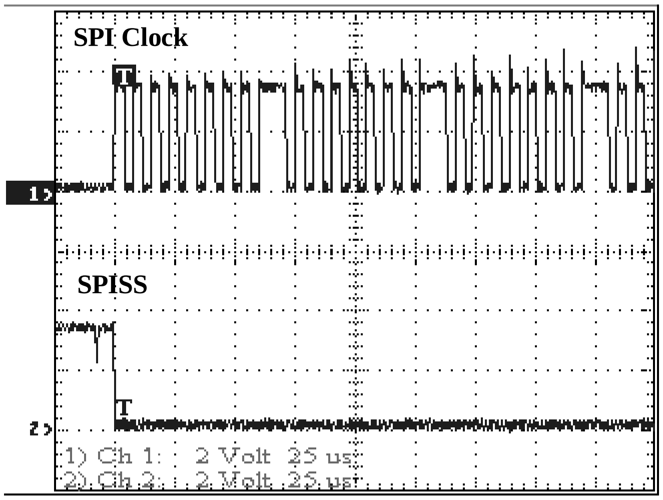


Diagram 1 Timing of the SPI Clock and Chip select line(SPISS)

Diagram 2 shows the Transfer line to the EEPROM (MOSI) "Master Out Slave In". The first sent byte puts the EEPROM in read mode (referring to AT25020 Data Sheet). The next one sets the start address which is zero in this case. Starting from this point on the EEPROM sends the data continuously as long as the SPI port is requesting (Clock runs). The MOSI line will be held to low for ever after the start address has been sent.

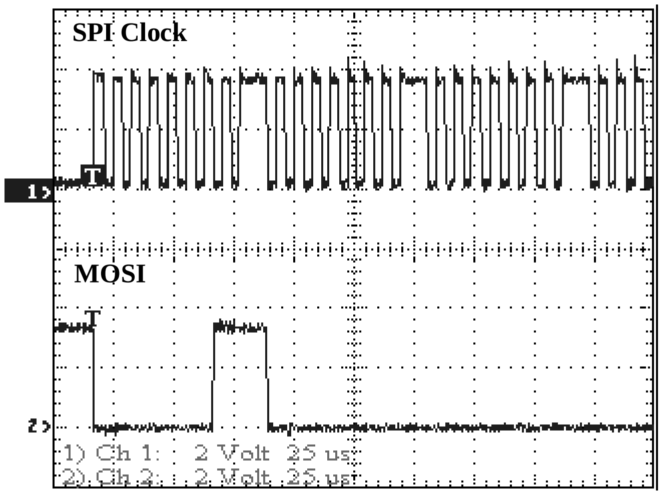


Diagram 2 Timing of the SPI Clock and Transfer line (MOSI)

Once the read instruction and the start address received by the EEPROM the first data will be sent immediately on the next clock cycles. Diagram 3 shows the first byte coming from address zero of the EEPROM. The EEPROM increments the address pointer automatically and sends the second data on the next clock cycles. This way the whole program code will be transferred to the DSP. After transfer the boot kernel disables the SPI port again and starts executing the code placed in program memory of the DSP by jumping to
PM address 0.

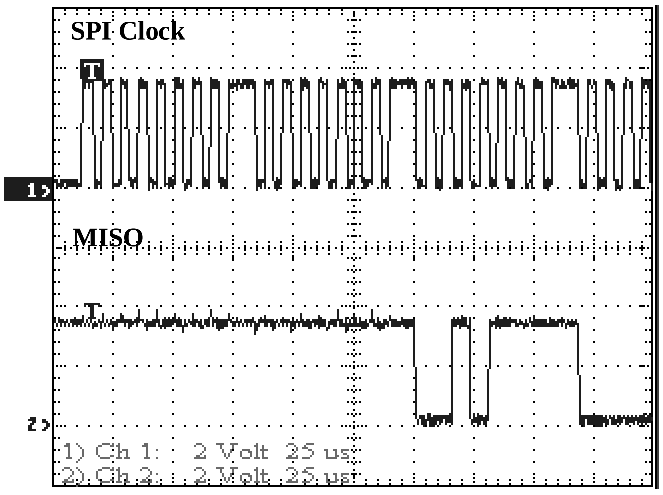


Diagram 3 Timing of the SPI Clock and Receiver line (MISO)

Finally the diagram 4 shows a part of the data transfer to the DSP.

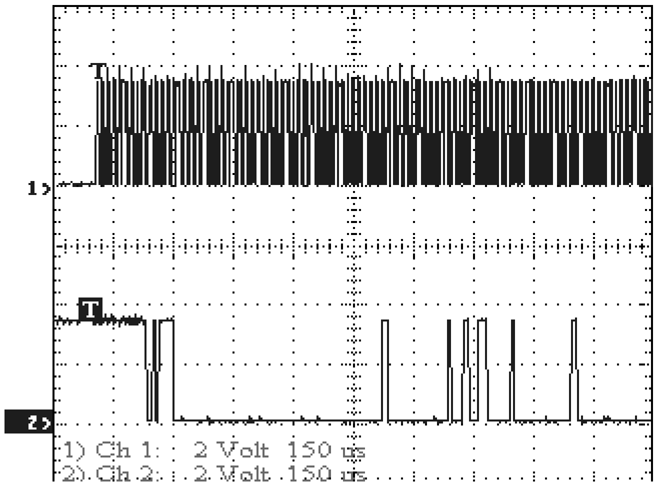


Diagram 4 Timing of the SPI Clock and Receiver line (MISO)

Conclusion:

This note should have given an idea on how a DSP can be booted via the SPI interface. Furthermore it has been described the hole project from beginning to the end of booting via SPI in hard- and software. Attached to this document please find the DSP software project used to create this file and the latest loader patch. (elfldr2191.dll).

References:

- VisualDSP++ 2.0 Getting Started Guide for ADSP-21xx DSPs
- VisualDSP++ 2.0 Linker and Utilities Manual for ADSP-21xx DSPs
- ADSP-2191 EZ –KIT LITE Manual
- ADSP-2191 Hardware Reference Manual
- All documents: http://www.analog.com/industry/dsp/tech_doc/gen_purpose.html
- ATMEL AT25020 data sheet
- http://www.atmel.com