Section 15

ADSP-BF533 Booting

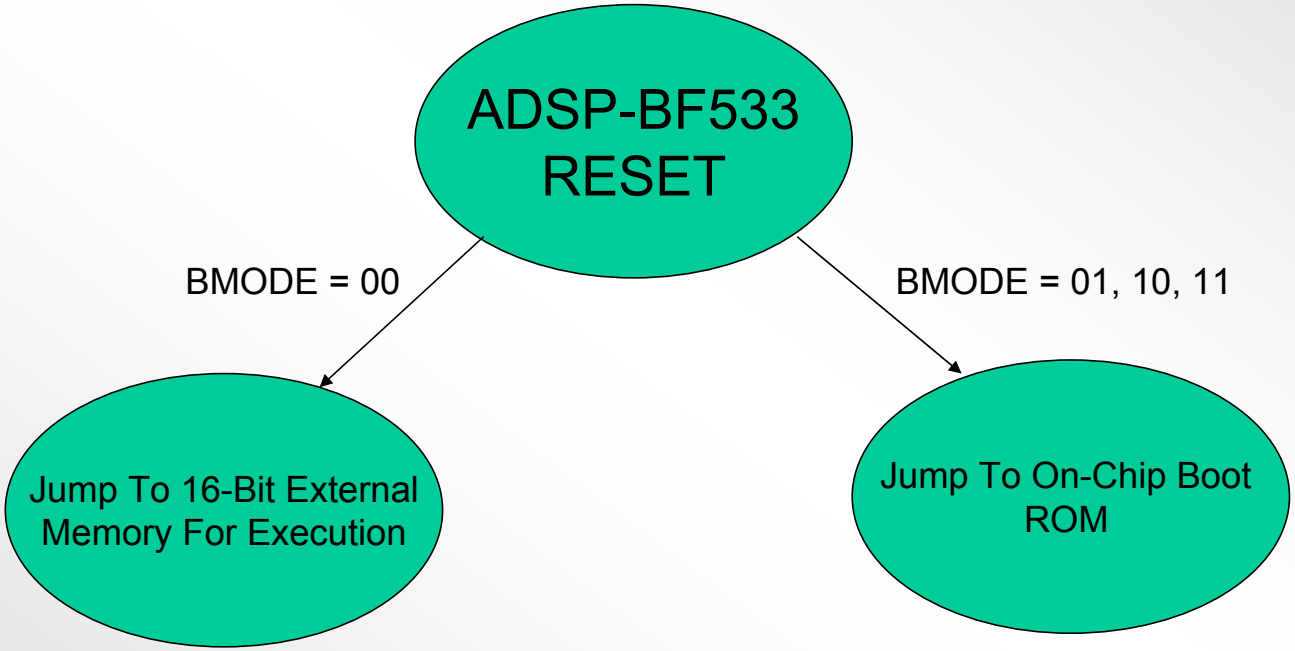
What is Booting?

- Booting is the process of loading application code, stored in an external memory device, into the various internal and external memories of the Blackfin Processor.
- Booting is done via the On-Chip Boot Rom located at 0xEF00 0000.
- This presentation describes the following:
  - Booting Methods
  - On-Chip Boot Rom

Booting Methods

BMODE Pins [1:0]	Description
00	Execute from 16-Bit External ASYNC Bank0 memory (Bypass Boot ROM)
01	Boot from 8/16-Bit Prom/Flash
10	Slave boot from SPI Master (see EE-240)
11	Boot from a 8- / 16- / 24-Bit Addressable SPI Device

Behavior Upon RESET



On-Chip Boot ROM Flow

1. Set up Supervisor Mode (doesn't apply for bypass mode)
   - Exits the Reset ISR and uses IVG15 (lowest priority interrupt)
2. Check to see if this boot request was from a software reset
   - Check bit 4 of the Reset Configuration Register
     - If 1, bypass normal boot sequence, jump to start of L1 memory (0xFFA0 0000 for ADSP-BF533 or 0xFFA0 8000 for ADSP-BF531 / ADSP-BF532) for execution
     - If 0, run full boot sequence (BMODE pins determines boot type)

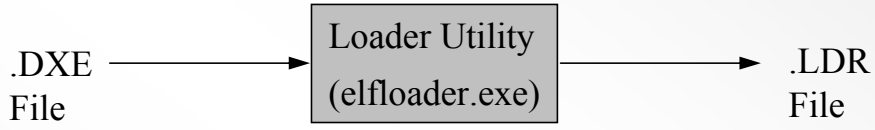
Boot From 8/16-bit Prom/Flash

- The On-Chip Boot Rom sets the following:
  - Enable Asynchronous Memory Bank 0 (ASYNC Bank 0)
  - Set Bank 0 hold time (R/W deasserted to AOE deasserted)
    - 3 cycles
  - Set Bank 0 Read/Write Access (Wait States) times
    - 15 cycles

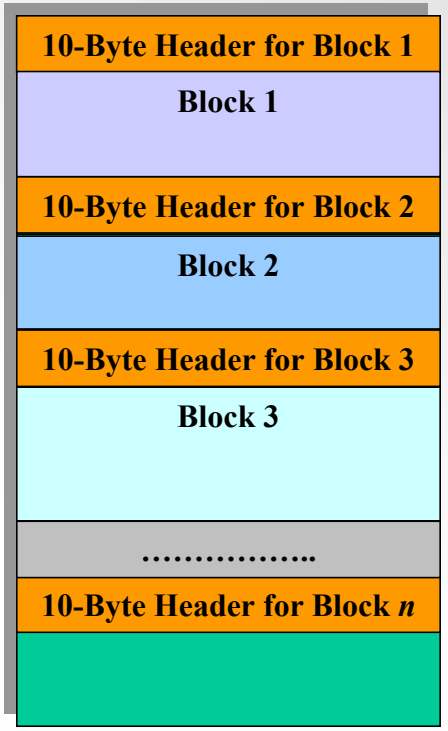
Boot from SPI Device

- Master Mode
  - Uses Slave Select 2 which maps to PF2
  - On-Chip Boot Rom sets the Baud Rate Register to 133
    - Which, based on a 133MHz system clock, will result in a 133MHz/(2*133) = 500kHz Baud Rate
  - Support for 8-,16-, and 24-bit addressable parts
- Slave Mode
  - Host downloads boot sequence through SPI port
  - PFx pin provides handshake to Host to pace transfers

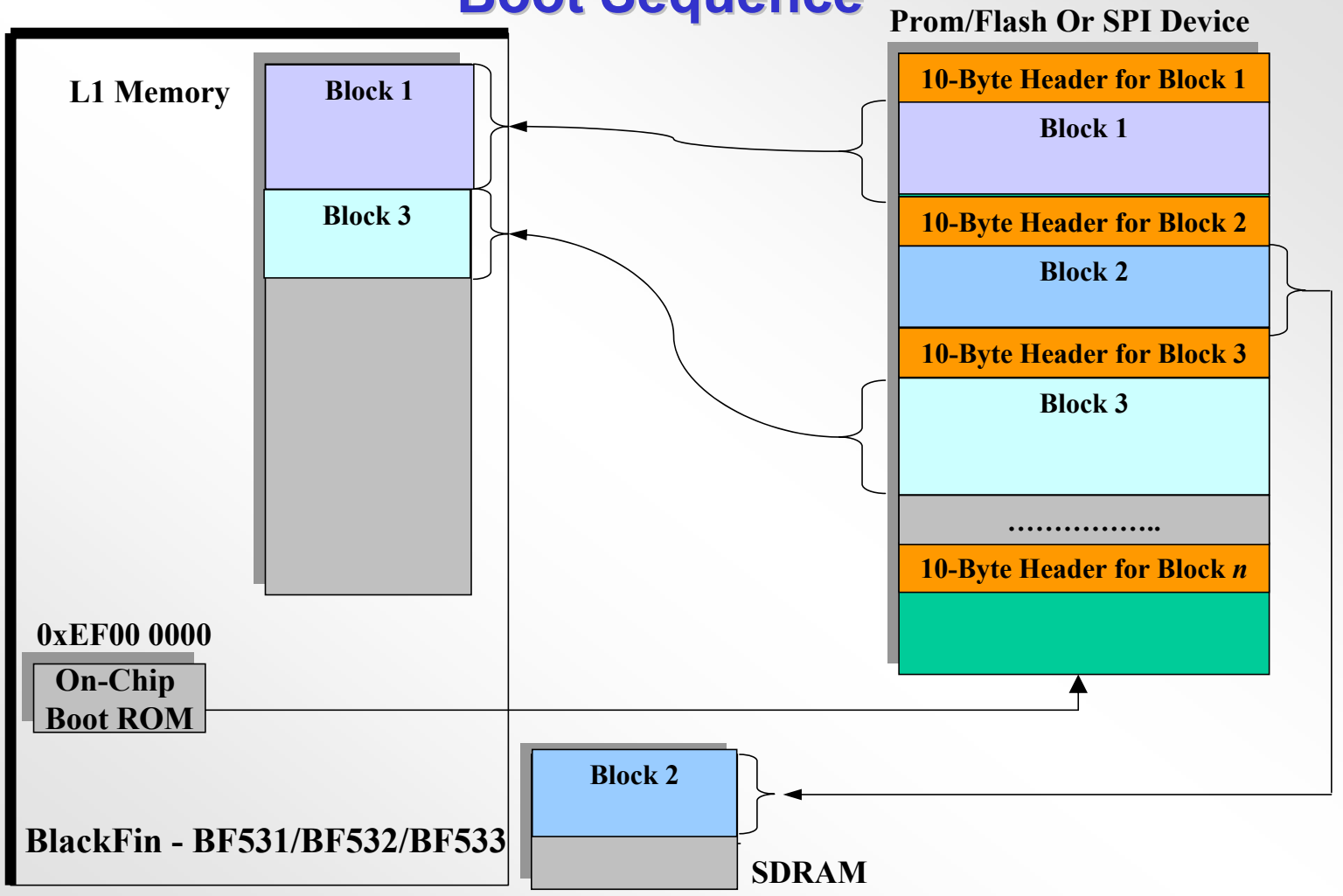
Executable → Loader File Conversion



- **The Loader Utility (elfloader.exe) converts executables (.DXE) into loader files (.LDR).**
- **The loader utility parses the input .DXE file and creates a loader file which consists of different blocks preceded by headers.**
- **These headers are, in turn, read and parsed by the On-Chip Boot Rom during booting.**

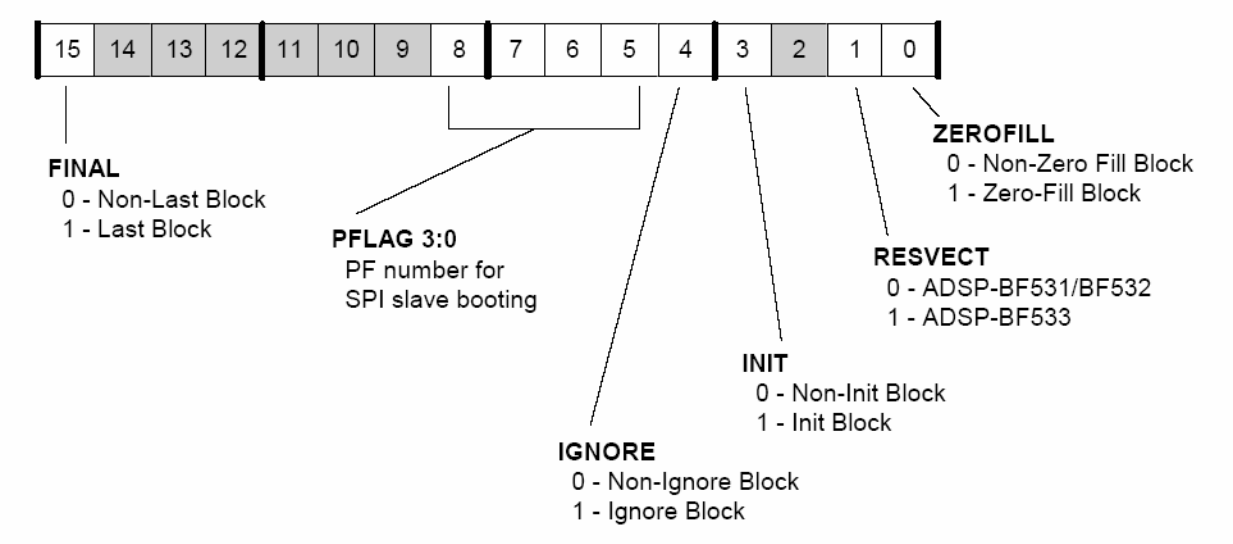


Boot Sequence



Header Information

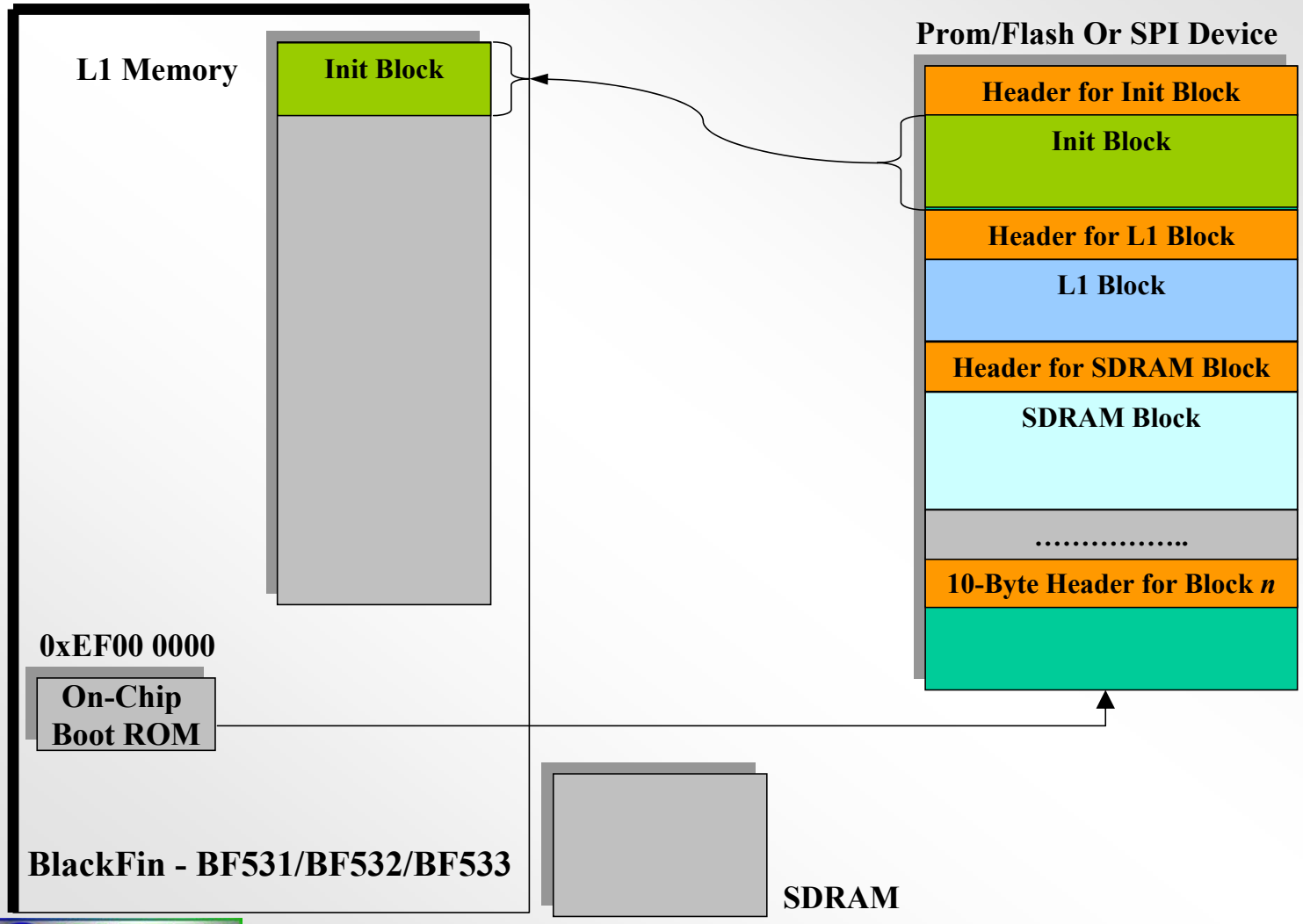
- **The elfloader utility converts the input .DXE file into various blocks. Each block is preceded by a 10-byte header:**
  - **Address (4 bytes) – where the block resides within memory**
  - **Count (4 bytes) – how many bytes to boot in**
  - **Flag (2 bytes) – information about the block:**



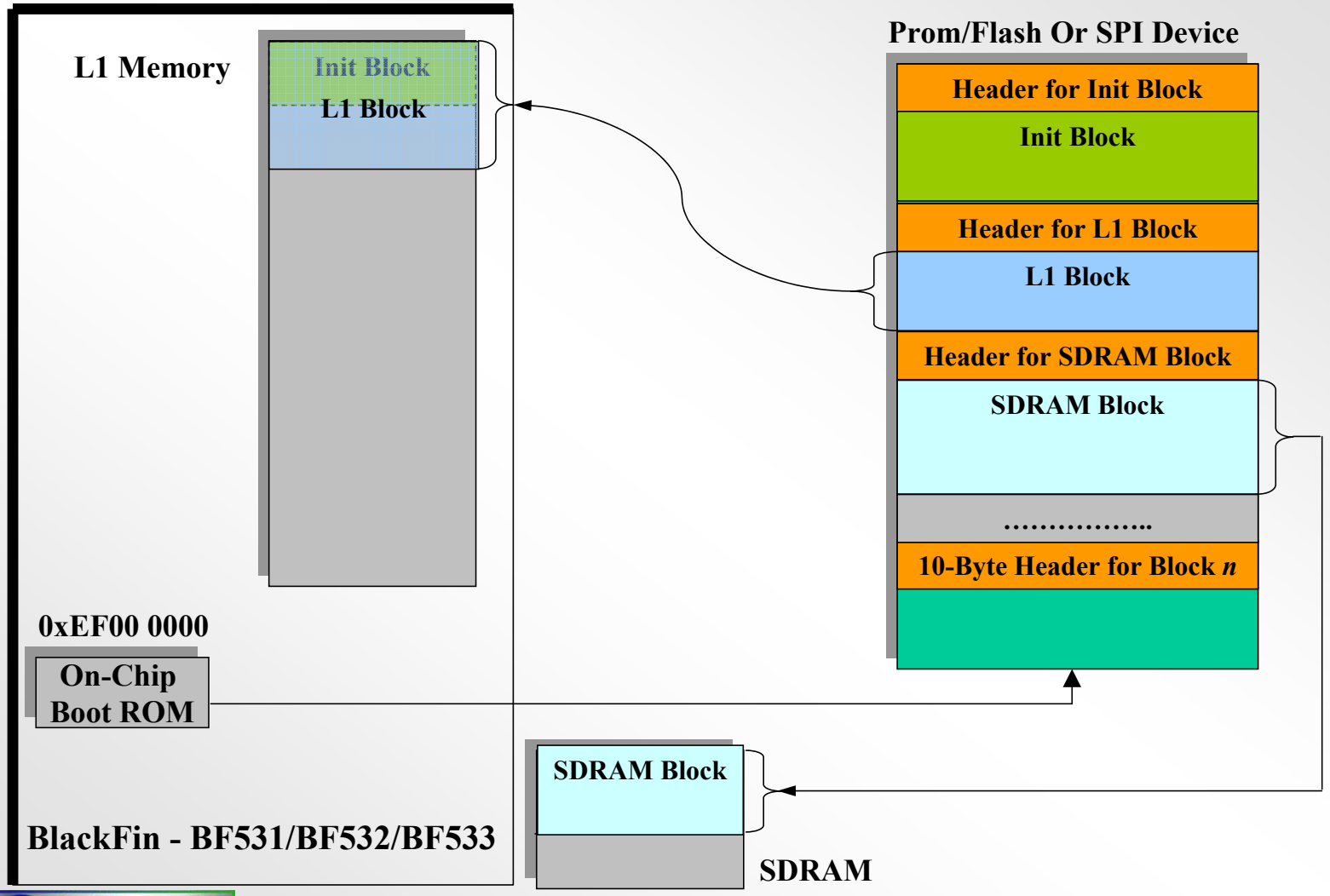
Flag Information

- ZEROFILL Block
  - indicates that the block is a buffer with zeros.
  - Zero Block is not included within loader file.
- RESVCT (Reset Vector)
  - Identifies the processor (0 for ADSP-BF532/1, 1 for ADSP-BF533)
    - Boot ROM jumps to start of L1 Instruction Memory after booting
- INIT (Initialization) Block
  - Block of code (i.e. subroutine) which executes before the actual application code boots over it.
    - When the On-Chip Boot Rom detects an Init Block, it boots the block into internal memory and makes a CALL to it. After the initialization code is executed, it gets overwritten with application code.
- IGNORE Block
  - Indicates a block that is not booted into memory.
    - Currently not implemented for application code.
- FINAL Block
  - Indicates boot process is complete after this block.
  - On-Chip Boot Rom jumps to the start of L1 memory for application code execution. The processor is left in Supervisor Mode (at IVG15).
- PFLAG
  - 4 bit code indicates which PFx to use for handshake during SPI slave boot.

Initialization Block Execution



Initialization Block Execution (cont.)



Initialization Code Example (Init Block)

```
/*********************************************************************************/
/*  This file contains 3 sections:                                               */
/*          1) A Pre-Init Section - this section saves off all the registers of the DSP onto the stack.  */
/*          2) A Init Code Section - this section is the customer initialization code which can be modified by the */
/*                  customer.  As an example, an SDRAM initialization code is supplied.  */
/*          3) A Post-Init Section - this section restores all the register from the stack.  Customers should not */
/*                  modify the Pre-Init and Post-Init Sections.  The Init Code Section can be modified for */
/*                  application use.                                             */
/*********************************************************************************/

#include <defBF532.h>

.section program;

/*******Pre-Init Section*********************************************/
        [--SP] = ASTAT;         // The Stack Pointer, SP, is set to the end of
        [--SP] = RETS;          //    scratchpad memory (0xFFB00FFC)
        [--SP] = (r7:0);        //     by the On-Chip Boot Rom
        [--SP] = (p5:0);
        [--SP] = I0;  [--SP] = I1;  [--SP] = I2;  [--SP] = I3;
        [--SP] = B0; [--SP] = B1; [--SP] = B2; [--SP] = B3;
        [--SP] = M0;[--SP] = M1;[--SP] = M2;[--SP] = M3;
        [--SP] = L0; [--SP] = L1; [--SP] = L2; [--SP] = L3;
/*************************************************************/
```

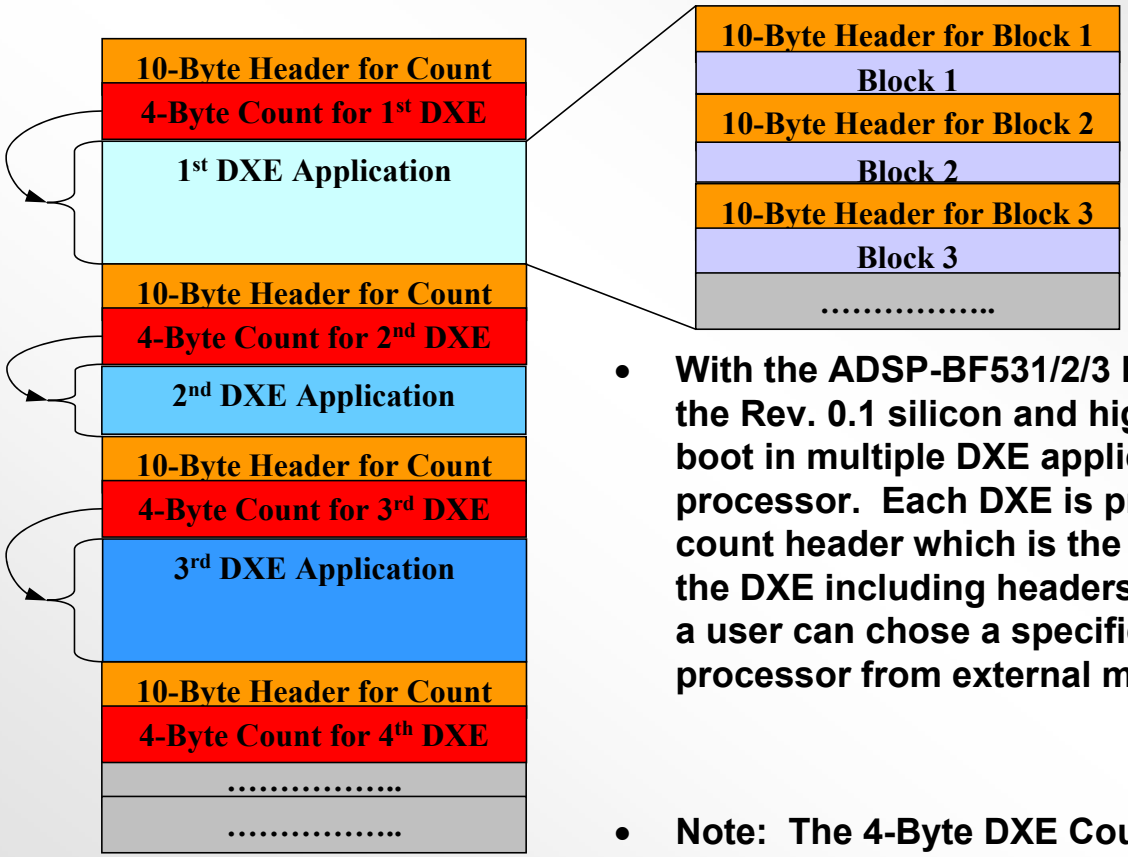
Initialization Code Example (cont.)

```
/*******Init Code Section*****************************************/

/**Please insert Initialization code in this section************/

/*******SDRAM Setup************/
Setup_SDRAM:
        P0.L = EBIU_SDRRC & 0xFFFF;
        P0.H = (EBIU_SDRRC >> 16) & 0xFFFF;      //SDRAM Refresh Rate Control Register
        R0 = 0x074A(Z);
        W[P0] = R0;

        SSYNC;

        P0.L = EBIU_SDBCTL & 0xFFFF;
        P0.H = (EBIU_SDBCTL >> 16) & 0xFFFF;     //SDRAM Memory Bank Control Register
        R0 = 0x0001(Z);
        W[P0] = R0;
        SSYNC;

        P0.L = EBIU_SDGCTL & 0xFFFF;
        P0.H = (EBIU_SDGCTL >> 16) & 0xFFFF;     //SDRAM Memory Global Control Register
        R0.L = 0x998D;
        R0.H = 0x0091;
        [P0] = R0;
        SSYNC;
/*****************************/
```

Initialization Code Example (cont.)

```
/*******Post-Init Section*********************************************/
                L3 = [SP++]; L2 = [SP++]; L1 = [SP++]; L0 = [SP++];
                M3 = [SP++]; M2 = [SP++]; M1 = [SP++]; M0 = [SP++];
                B3 = [SP++]; B2 = [SP++]; B1 = [SP++]; B0 = [SP++];
                I3 = [SP++]; I2 = [SP++]; I1 = [SP++]; I0 = [SP++];
                (p5:0) = [SP++];
                (r7:0) = [SP++];
                RETS = [SP++];
                ASTAT = [SP++];
/*********************************************************************/

                RTS;
```

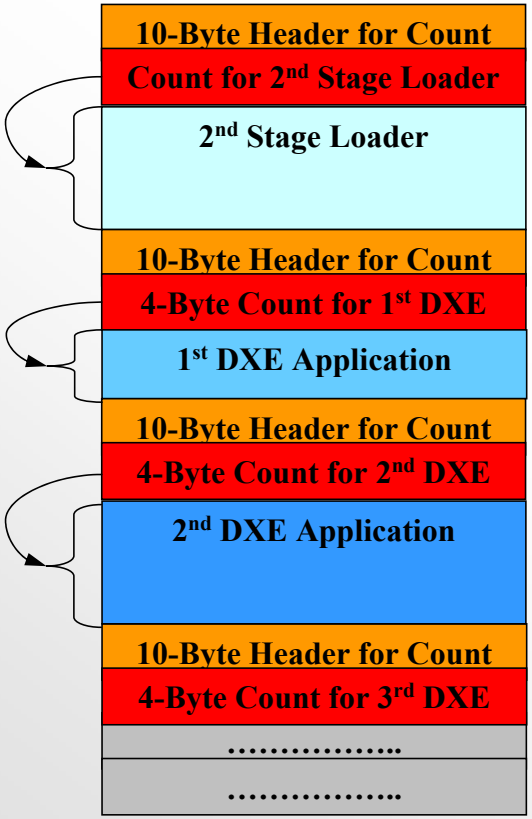
Multi-Application Boot Option



- With the ADSP-BF531/2/3 loader file structure and the Rev. 0.1 silicon and higher, it is possible to boot in multiple DXE applications into the processor. Each DXE is preceded by a 4-Byte DXE count header which is the number of bytes within the DXE including headers. With this information, a user can chose a specific DXE to boot into the processor from external memory.

- Note: The 4-Byte DXE Count Block is encapsulated within a 10-byte header to be compatible with the Rev. 0.0 Silicon.

How to Boot in Multiple Application DXEs



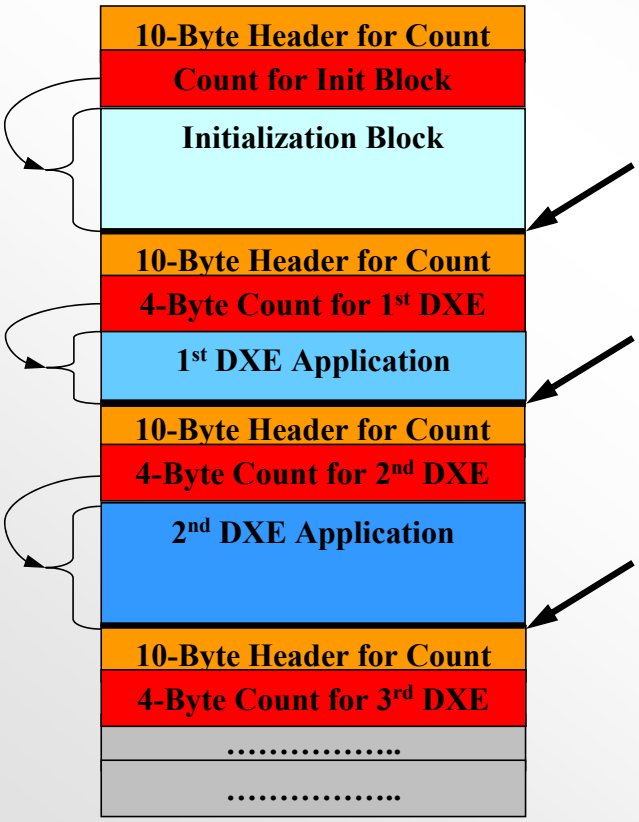
1. Use the 2nd Stage Loader option (-l 2ndStageLoaderName.dxe). This option allows you to use a custom 2nd Stage Loader (not included) to boot in specific DXEs from external memory.

   – After the the 2nd Stage Loader gets booted into internal memory via the On-Chip Boot Rom, it has full control of the boot process. It can use the DXE byte counts to boot in specific DXEs from external memory.

KAZTEK ENGINEERING

ANALOG DEVICES

How to Boot in Multiple Application DXEs



2. Use the Initialization Block option (-init InitBlock.dxe). This option allows you to change the external memory pointer and boot in a specific DXE via the On-Chip Boot Rom.
   - R0 and R3 are used as external memory pointers by the On-Chip Boot Rom. R0 is for Flash/Prom Boot and R3 is for SPI Memory Boot.
   - Within the Init Block code, change the value of R0 for Flash/Prom Boot or R3 for SPI boot to point to the external memory location of where the specific application DXE starts. After the processor returns from the Init Block code to the On-Chip Boot Rom, the On-Chip Boot Rom will continue to boot in bytes from the location specified in the R0 or R3 register.

Note: This option requires that the user know the starting locations of specific DXEs within external memory. R0 or R3 must point to the 10-byte Count Header as indicated on the figure.

Initialization Code for Multi-Application Boot

```
#include <defBF532.h>
.section program;
/*******Pre-Init Section*********************************************/
/*******************************************************************/
/*******Init Code Section*********************************************/
R0.H = High Address of DXE Location (R0 for Flash/Prom Boot;  R3 for SPI boot)
R0.L = Low Address of DXE Location. (R0 for Flash/Prom Boot;  R3 for SPI boot)
/*******************************************************************/
/*******Post-Init Section*********************************************/
/*******************************************************************/
                RTS;
```

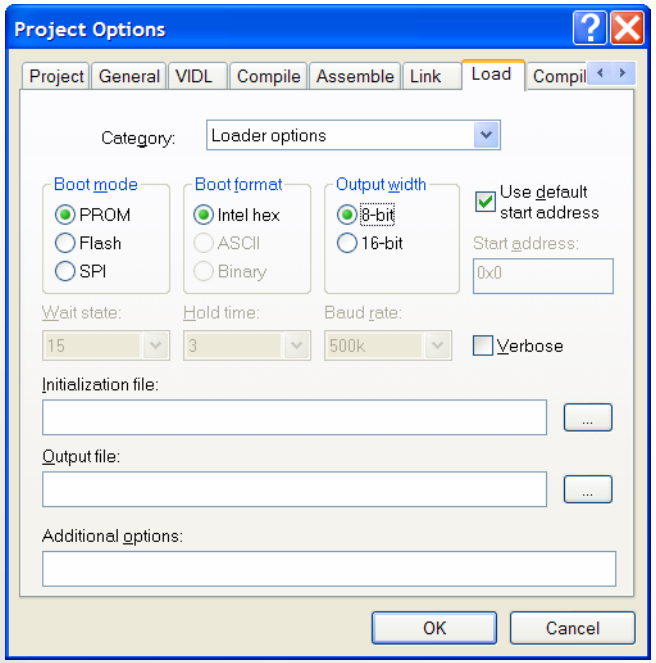
Supported Boot Memories

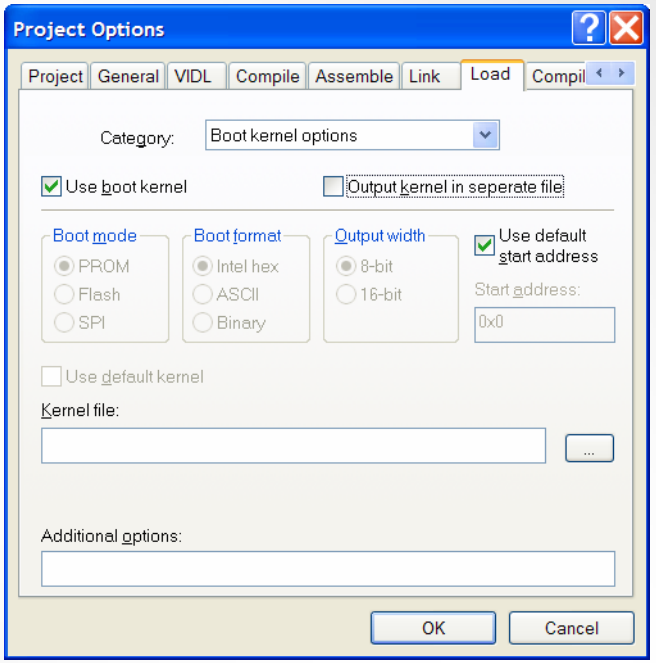
- On-Chip Boot Rom Allows Booting To All Memory Ranges*:
  - L1 Memory
    - ADSP-BF531
      - Data Bank A SRAM (0xFF80 4000 – 0xFF80 7FFF)
      - Instruction SRAM (0xFFA0 8000 – FFA0 BFFF)
    - ADSP-BF532
      - Data Bank A SRAM (0xFF80 4000 – 0xFF80 7FFF)
      - Data Bank B SRAM (0xFF90 4000 – 0xFF90 7FFF)
      - Instruction SRAM (0xFFA0 8000 – FFA1 3FFF)
    - ADSP-BF533
      - Data Bank A SRAM (0xFF80 0000 – 0xFF80 7FFF)
      - Data Bank B SRAM (0xFF90 0000 – 0xFF90 7FFF)
      - Instruction SRAM (0xFFA0 0000 – FFA1 3FFF)
  - SDRAM
    - Bank 0 (0x0000 0000 – 0x07FF FFFF)

*NOTE: Booting to Scratchpad Memory (0xFFB0 0000) is not supported.

Creating a Loader File

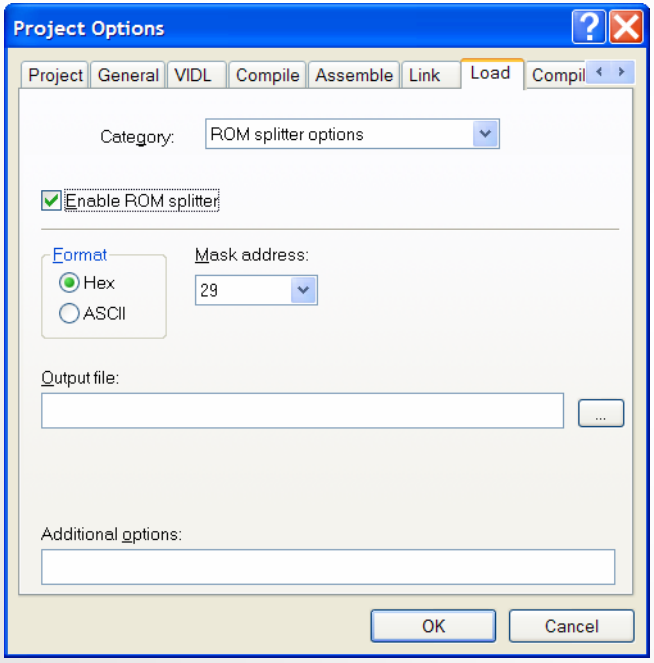
- **Using the Loader Property Page under Project Options**





Creating a Bypass Mode File

For 16-bit External Execution (BMODE = 00)



- **The Mask Address field masks all EPROM/Flash address bits above or equal to the number specified. For example, Mask Address = 29 masks all the bits above and including A29 (ANDed by 0x1FFF FFFF). For example, 0x2000 0000 becomes 0x0000 0000. The valid #s are integers 0 through 32.**

NOTE: The ROM splitter only processes ROM sections. Segments should be declared as 'ROM' within the .LDF file.