TMS320VC5470/5471 Bootloader

Bill Winderweedle — *DSP Catalog, C5000 Hardware Applications*
Jack Rosenzweig — *Software Development Systems*
Denise Ombres — *Software Development Systems*

ABSTRACT

The TMS320VC547x devices, TMS320VC5470 and TMS320VC5471, have a master TMS470R1x™ (ARM7TDMIE) microcontroller (MCU) central processing unit (CPU) with a slave TMS320C54x™ digital signal processor (DSP) CPU. Bootload is accomplished through the MCU while the DSP is held in reset. Since no ROM resides on these devices, the bootloader code is transferred on-chip through one of several interfaces to the MCU and sent through an internal memory interface to the DSP. This document describes a method for boot loading through the MCU external memory interface.

Contents

List of Figures

List of Tables

1 TMS320VC5470/TMS320VC5471 Bootloader

This document describes the process by which the TMS470R1x (ARM7TDMIE) MCU master processor loads code into the C54x™ DSP slave processor after a power-on reset.

1.1 Supported Configurations

- Both little- and big-endian loading are supported.
- Supports loading into flash or RAM memory
- Spectrum Digital TMS320VC5470/5471 EVM board
- Code Composer Studio™ OMAP™ v 2.1
- MS Visual Studio v6.0 (optional)

1.2 Bootloading Process

The MCU is responsible for setting up the DSP memory map register bits (MP/MC, OVLY, DROM and APIBN), and copying the bootloader into internal memory shared by the DSP and MCU via the arm port interface, before releasing the DSP from reset. The MCU clock, DSP clock, and arm port interface internal-memory wait states must be initialized before the bootload process is run. Manipulating the DSP memory map must be done carefully, because it is possible to swap out the memory that the DSP is currently executing from. With this consideration, the bootloader program has three distinct phases, each with its own memory map configuration. Following is a description of these phases.

1.2.1 Bootloader Phase 1

Phase 1 starts with MP/MC = 0, OVLY = 0, DROM = 0 in arm port interface boot mode (see Figure 1). When OVLY and DROM are cleared, the DSP expects to find external program memory space. In arm-port-interface boot mode, the DSP shadows 0x3F80 – 0x3FFF into the normal reset vector space at 0xFF80 – 0xFFFF. The following steps are executed from code at the shadowed reset vector, starting at 0x3F80:

- Write 0xFFA8 to the DSP PMST register to set the OVLY and DROM bits for proper enabling of internal DSP program memory space.
- Branch to address 0x3810, which is the start of the bootloader program phase 2.

C54x, Code Composer Studio, and OMAP are trademarks of Texas Instruments.

<table>
<tr><th>Hex</th><th colspan="2">Page 0 Program, MP/MC = 1 (Microprocessor Mode)</th><th>Hex</th><th colspan="2">Page 0 Program, MP/MC = 0 (Microcomputer Mode)</th><th>Hex</th><th colspan="2">Data</th></tr>
<tr><td>0000</td><td>OVLY = 1</td><td>OVLY = 0</td><td>0000</td><td>OVLY = 1</td><td>OVLY = 0</td><td>0000</td><td colspan="2" rowspan="2">Memory-Mapped Registers, Scratch-Pad RAM</td></tr>
<tr><td>007F</td><td>Reserved</td><td>External Program Space Memory</td><td>007F</td><td>Reserved</td><td>External Program Space Memory</td><td>007F</td></tr>
<tr><td>0080
1FFF</td><td rowspan="2">On-Chip Data DARAM</td><td rowspan="4">External Program Space Memory</td><td>0080
1FFF</td><td rowspan="2">On-Chip Data DARAM</td><td rowspan="4">External Program Space Memory</td><td>0080
1FFF</td><td colspan="2">On-Chip Data DARAM (8K-0x80 words)</td></tr>
<tr><td>2000
37FF</td><td rowspan="2">On-chip Data DARAM, Arm Port Interface-Accessible</td><td>2000
37FF</td><td rowspan="2">On-Chip Data DARAM, Arm Port Interface-Accessible</td><td>2000
37FF</td><td colspan="2">On-Chip Data DARAM, Arm Port Interface-Accessible (8K words)</td></tr>
<tr><td>3800
3FFF</td><td>3800
3FFF</td><td>3800
3FFF</td><td colspan="2">(Shadowed portion)</td></tr>
<tr><td>4000
5FFF</td><td>On-Chip Data SARAM</td><td>4000
5FFF</td><td>On-Chip Data SARAM</td><td>4000
5FFF</td><td colspan="2">On-Chip Data SARAM (8K words, data only)</td></tr>
<tr><td rowspan="5">6000

F7FF</td><td colspan="2" rowspan="5">External Program Space Memory</td><td rowspan="5">6000

F7FF</td><td colspan="2" rowspan="5">External Program Space Memory</td><td>6000
7FFF</td><td colspan="2">On-Chip Data SARAM (8K words)</td></tr>
<tr><td>8000
BFFF</td><td colspan="2">External Data Space Memory</td></tr>
<tr><td>C000</td><td>DROM=1</td><td>DROM=0</td></tr>
<tr><td>DFFF</td><td>On-Chip Program SARAM (8K words)</td><td rowspan="2">External Data Space Memory</td></tr>
<tr><td>E000

F7FF</td><td>On-Chip Program SARAM (6K words)</td></tr>
<tr><td>F800
FFFF</td><td colspan="2">Shadowed Arm Port Interface DARAM (2K)</td><td>F800
FFFF</td><td colspan="2">Shadowed Arm Port Interface DARAM (2K)</td><td>F800
FFFF</td><td colspan="2">External Data Space Memory</td></tr>
</table>

NOTE: When APIBN = 0 and ABMDIS = 0, 2K words of the arm port interface DARAM are re-mapped to program-space. All other internal program-space RAMs are disabled in program space. Data-space RAMs may be dual-mapped to program-space via OVLY.

Figure 1. DSP Sub-System Memory Map for Arm Port Interface Boot Mode (APIBN = 0 and ABMDIS = 0)

1.2.2 Bootloader Phase 2

Phase 2 starts with MP/MC = 0, OVLY = 1, DROM = 1 in arm port interface boot mode (see Figure 2). When OVLY and DROM are set, the DSP expects to find internal program memory space. In arm port interface boot mode, the DSP shadows 0x3F80–0x3FFF into the normal reset vector space at 0xFF80–0xFFFF. The following step is executed from code in arm port interface-accessible memory starting at 0x3810.

- Write 0x0010 to the BSCR register to clear the arm port interface boot mode bit. This takes the DSP out of arm port interface boot mode and places it into normal boot mode, which allows access to the C54x reset vector at the DSP program space address range 0xFF80–0xFFFF.

Hex	Page 0 Program, MP/MC = 1 (Microprocessor Mode)		Hex	Page 0 Program, MP/MC = 0 (Microcomputer Mode)		Hex	Data	
0000	OVLY = 1	OVLY = 0	0000	OVLY = 1	OVLY = 0	0000	Memory-Mapped Registers, Scratch-Pad RAM	
007F	Reserved	External Program Space Memory	007F	Reserved	External Program Space Memory	007F		
0080	On-Chip Data DARAM	External Program Space Memory	0080	On-Chip Data DARAM	External Program Space Memory	0080	On-Chip Data DARAM (8K-0x80 words)	
1FFF			1FFF			1FFF		
2000	On-Chip Data DARAM, Arm Port Interface-Accessible	External Program Space Memory	2000	On-Chip Data DARAM, Arm Port Interface-Accessible	External Program Space Memory	2000	On-Chip Data DARAM, Arm Port Interface-Accessible (8K Words)	
3FFF			3FFF			3FFF		
4000	On-Chip Data SARAM	External Program Space Memory	4000	On-Chip Data SARAM	External Program Space Memory	4000	On-Chip Data SARAM (8K words)	
5FFF			5FFF			5FFF		
6000	External Program Space Memory		6000	On-Chip Program SARAM (8K words, program only)		6000	On-Chip Data SARAM, (8K words, data only)	
7FFF			7FFF			7FFF		
8000	External Program Space Memory		8000	On-Chip Program SARAM (8K words, program only)		8000	External Data Space Memory	
			9FFF					
			A000	On-Chip Program SARAM (8K words, program only)				
			BFFF			BFFF		
			C000	On-Chip Program SARAM (8K Words)		C000	DROM=1	DROM=0
			DFFF				On-Chip Program SARAM	External Data-Space Memory
			E000	On-Chip Program SARAM (8K words)				
FFFF			FFFF			FFFF		

Figure 2. DSP Sub-System Memory Map for Normal Boot Mode (APIBN = 1 or ABMDIS = 1)

1.2.3 Bootloader Phase 3

Phase 3 starts with MP/MC = 0, OVLY = 1, DROM = 1 in normal boot mode. Now the DSP expects all of its program memory space to be internal. In normal boot mode the reset vector space from 0xFF80–0xFFFF is accessible. Currently, none of the program or data memory space has been initialized, except that the bootloader DSP program has been loaded into arm port interface-accessible memory space from 0x3810–0x3900.

There are four handshake variables held in arm port interface memory from 0x3800 to 0x3803 used in the boot-copy process. Also, the PMST value to be used by the DSP is written to location 0x3804. Table 1 summarizes the software handshake memory locations.

Table 1. Software Handshake Memory Locations

DSP Address	S/W Handshake	Description
0x3800	DSP_Ready	When set, signals to the MCU that the DSP is in normal boot mode.
0x3801	Prog_Buf_Ready	When set, signals to the DSP that the MCU has transferred code to arm port interface memory. This code is now ready to be transferred to runtime DSP memory.
0x3802	Data_Buf_Ready	When set, signals to the DSP that the MCU has transferred data to arm port interface memory. This data is now ready to be transferred to runtime DSP memory.
0x3803	Copy_Done	When set, signals to the DSP that the MCU has completed transfer of all code and data to arm port interface memory. The DSP can now boot from its reset vector at 0xFF80.
0x3804	PMST_VAL	PMST value used by the DSP for proper configuration of its memory map
0x3900	API_BUF_START	Starting address for arm port interface buffer written to by MCU for temporary storage of code/data to be transferred by the DSP.
0x3F80	API_BUF_END	Starting address for arm port interface memory shadowed to the DSP reset vector of 0xFF80 when in arm port interface boot mode.

The following steps are executed by the DSP Bootloader, which resides in arm port interface-accessible memory starting at 0x3810:

- The DSP sets the DSP_Ready word to a value of 0x0001, which tells the MCU that the DSP is now in normal boot mode and is ready to start the boot-copy process.
- MCU copies the code sections of the DSP program into the arm port interface transfer buffer. When the buffer becomes full, the DSP moves the buffer contents into the DSP program memory space. The MCU and DSP synchronize their access to the arm port interface buffer via the Prog_Buf_Ready handshake flag. The process is repeated until the last section of the DSP program code is copied, which is indicated by a section of zero length.
- Using the same algorithm, the data sections of the DSP program are loaded. The Data_Buf_Ready handshake flag is used for synchronization between MCU and DSP.
- The MCU will now set the Copy_Done word to a value of ‘0x0001’, which signals to the DSP that the boot-copy process has completed.
- Once the DSP sees the Copy_Done word equal to 0x0001, it will branch to the reset vector at 0xFF80 and begin normal execution.

NOTE: The reset vector will now be valid since it was initialized during the program memory boot-copy process.

1.2.4 Bootloader Execution Flow Chart

Figure 3 shows the order of actions for each processor, as well as the synchronization that needs to occur between them to control the timing of the execution.

NOTE: The following items have been simplified for better readability of the flow chart:

- Some of the configuration steps are not shown.
- The separate loadings of the DSP program and data sections are shown as a single loop in the flow chart; however, in reality these are done in two consecutive loops. The “arm port interface buffer ready” handshake flag shown in the flow chart represents two separate flags: Prog_Buf_Ready, and Data_Buf_Ready.

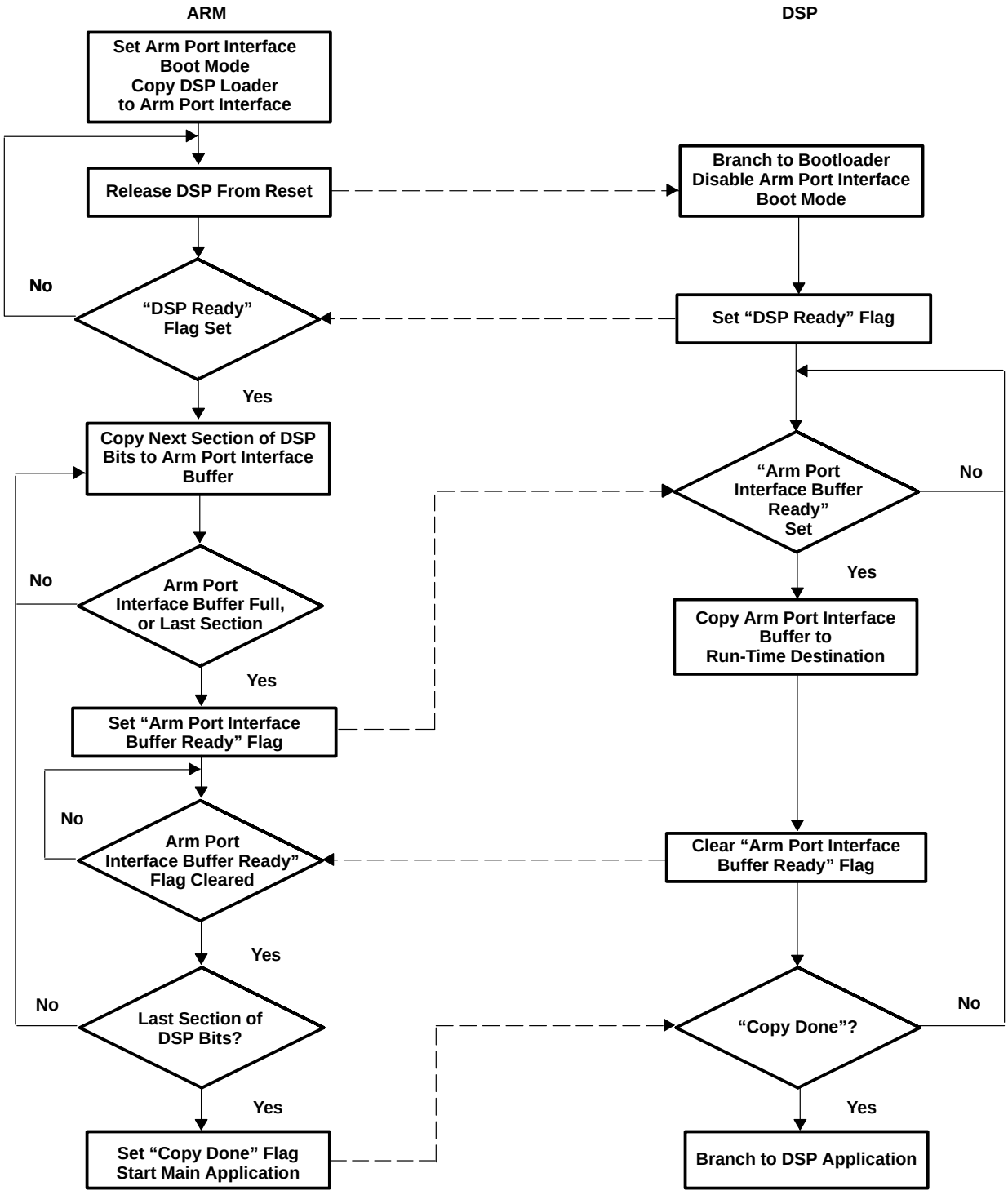


Figure 3. Bootloader Execution Flow Chart

1.2.5 Converting DSP COFF Files Into ARM C-Language Header Files

Since the MCU and the DSP are different types of processors, the DSP code needs to be converted into a format usable by the MCU. To do this, the DSP COFF file output is converted into MCU C language header files, which declare arrays initialized with the contents of the COFF file. This conversion can be accomplished by using the converter program *OUT2BOOT*. This converter uses the TI assembly language tool *HEX500* to extract the DSP code from the COFF file. See the *TMS320C54x Assembly Language Tools User's Guide* (SPRU102) for information on the common object file format (COFF) and the *HEX500* conversion utility. The *OUT2BOOT* program calls *HEX500* to convert the COFF file into two binary output files: one for program memory code and the other for data memory contents. Then, *OUT2BOOT* converts these two binary files into the C language header files, which are used as include files with the MCU C code. The combined MCU and DSP code can then be placed into flash or ROM to be accessed from the MCU external memory interface.

The header file consists of an array that holds the sections of code/data (see Figure 4). A record consisting of the following fields represents each section:

- The section length in 16-bit words
- The DSP destination run-time address for the section.
- The code/data words for the section.

```
Array [] = {
length, address, data, …
length, address, data, …
length, address, data, …
0  // '0' in the final line signifies the end of program code or data
}
```

Figure 4. OUT2BOOT Format for DSP Code and Data

1.3 Bootloader Example

Included with this application report is an example of how to load a program into the DSP processor from the MCU. This DSP application continuously toggles the XF pin, which flashes an LED. Listed below are the MS VC++ Studio and Code Composer Studio projects, which build the individual components of this example.

NOTE: The compressed file which accompanies this application report, *c547x_bootloader.zip*, should be copied to the Code Composer Studio *<drive>:\ti\myprojects* directory and extracted there into *<drive>:\ti\myprojects\c547x_bootloader.* All directories referenced below and in the following build example are relative to the c547x_bootloader directory.

- .**out2boot:** MS VC++ Studio project which builds the *out2boot.exe* conversion utility. This tool converts a DSP COFF File into an ARM C-language header. The source code is provided, as well as the executable*, .\out2boot\out2boot.exe*.
- .**dsp_proj:** Code Composer Studio project which builds a simple DSP application to toggle the XF pin. The DSP application can be modified or replaced to include new code.
- .**dsp_boot:** Code Composer Studio project which builds the C54x-side bootloader program, *dsp_boot.out.* This program copies the DSP application from arm port interface memory to its runtime location in memory. This project should not be modified.

- .**\arm_boot:** Code Composer Studio project which builds the ARM-side bootloader program, *arm_boot_le_flash.out* for little-endian or *arm_boot_be_flash.out* for big-endian. The program initializes the device and copies the DSP bootloader and the DSP application into arm port interface memory. Once this is completed, the ARM main program simply loops indefinitely, while the DSP application continuously flashes the LED. This project can be modified to include other arm or DSP code (as modified by out2boot) to be loaded by the MCU.

1.3.1 Building the Bootloader Example

To build the bootloader example, follow these steps (Figure 5):

1. If needed, use the MS VC++ Studio workspace file, .*out2boot**out2boot.dsw,* to build the conversion utility **.\out2boot\out2boot.exe.** Otherwise, just use the existing executable and skip this step.
2. Configure Code Composer Studio using provided configuration files:

- Run Code Composer Studio Setup program
- Choose File–>Import menu option.
- Click *Advanced* button.
- Browse to the .*config* directory, and select a .Code Composer Studio file that is appropriate for your memory setup. Different configuration files are provided to support various memory layouts. For example, choose *c5471_le_ramlow.ccs* for little-endian, RAM-low memory.

3. Using the C54x Code Composer Studio project file, .*dsp_proj**dsp_proj.prj,* build the sample DSP application **.\dsp_proj\dsp_proj.out**. The post-link step of this project will invoke the out2boot utility on the **dsp_proj.out** image, and copy the resulting C-header files, *dsp_proj_code.h* and *dsp_proj_data.h,* into the. *arm_boot* directory.
4. Using the C54x Code Composer Studio project file, .*dsp_boot**dsp_boot.prj,* build the DSP bootloader program **.\dsp_boot\dsp_boot.out**. The post-link step of this project will invoke the out2boot utility on the **dsp_boot.out** image, and copy the resulting C-header file, *dsp_boot_code.h* into the. *arm_boot* directory.
5. Choose the ARM Code Composer Studio project file appropriate for your memory byte ordering, .*arm_boot**arm_boot_le.prj* or .*arm_boot**arm_boot_be.prj,* and use it to build the ARM bootloader program, **.\arm_boot\arm_boot_le_flash.out** for little-endian or **.\arm_boot\arm_boot_be_flash.out** for big-endian. When switching between big- and little-endian modes, do a *Project–>Rebuild All* within Code Composer Studio because most of the files within each project are the same and will not be re-compiled with a *Project–>Build*.

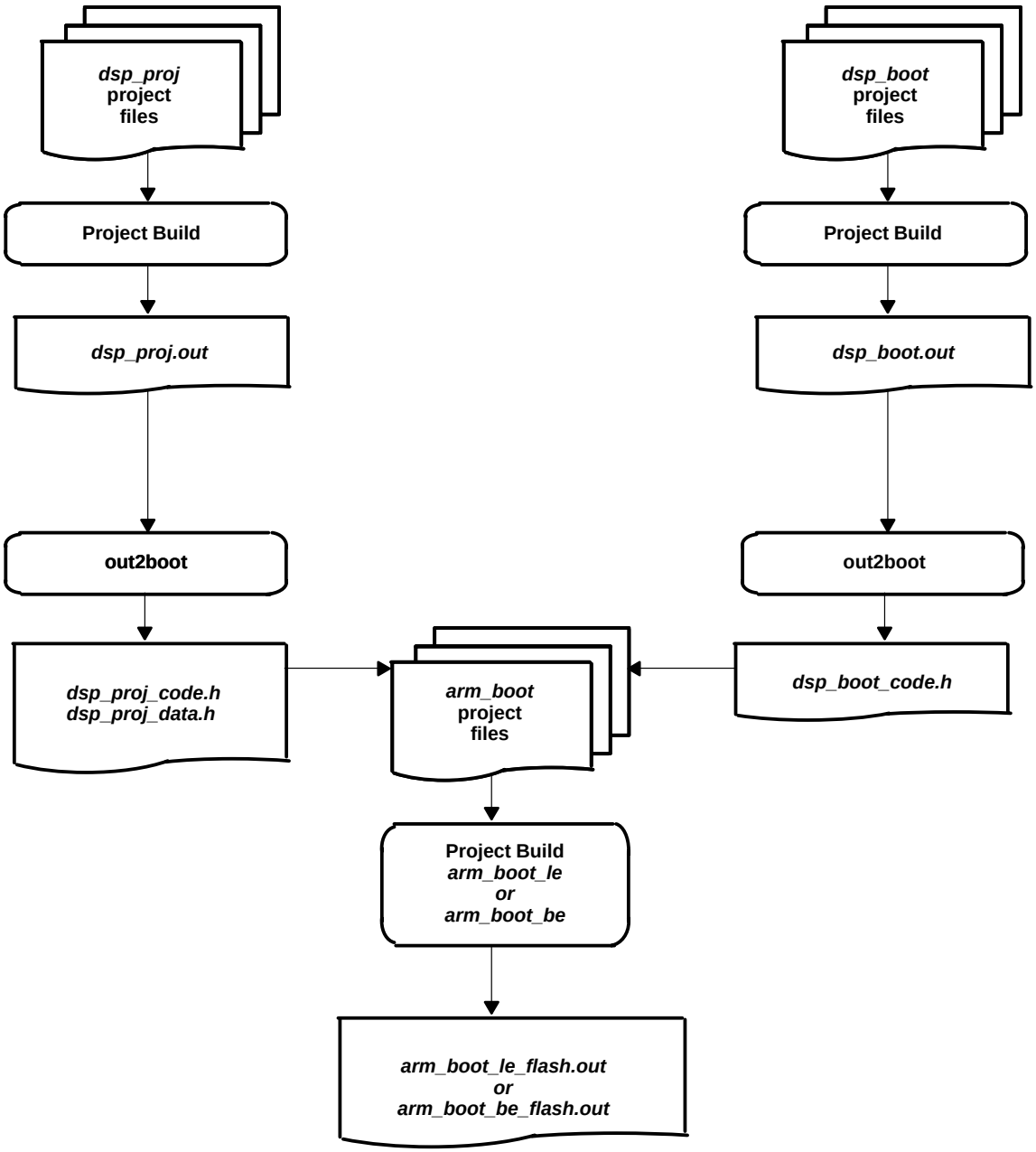


Figure 5. Bootloader Build Flowchart

1.3.2 Executing the Bootloader Example

To execute the bootloader example, follow these steps (NOTE: All directories referenced below are relative to the Code Composer Studio root installation directory and project sub-directory):

1. On the C5471 EVM board, configure JP19 for big-endian (1–2) or little-endian (2–3) mode. Also, ensure that JP20 is set to the 32-bit ROM (Flash) Size (1–2). The arm_boot code examples included with the bootloader use ARM 32-BIS. The chosen byte ordering and ARM instruction set must match those followed during the build of the bootloader example.
2. Start Code Composer Studio. This should run the Parallel Debug Manager.
3. From the Parallel Debug Manager open ARM Code Composer Studio.
4. From the Parallel Debug Manager open C54x Code Composer Studio.
5. In the C54x Code Composer Studio debug window, make sure that the DSP is in the running state. This will be indicated at the bottom of the window. If it is not, then execute Debug–>Reset CPU, followed by Debug–>Run Free.
6. In the ARM Code Composer Studio debug window, load the executable: **.\arm_boot\arm_boot_le_flash.out** for little-endian or **.\arm_boot\arm_boot_be_flash.out** for big-endian.
7. In the ARM Code Composer Studio debug window, execute Debug–>Run. The DS1 LED should begin to blink.

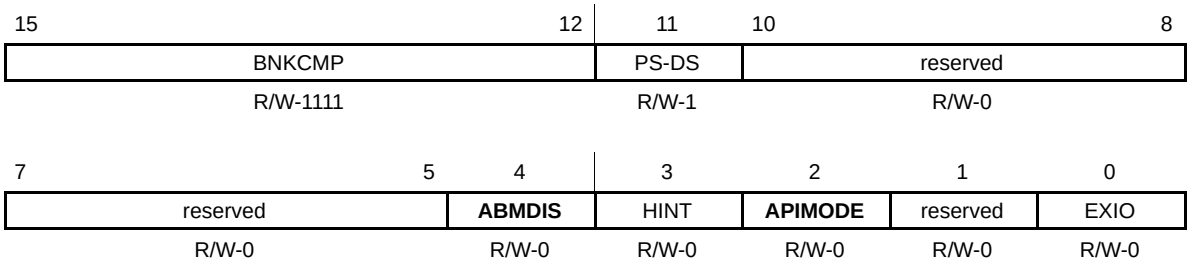
1.4 External Flash vs. External SRAM Considerations

Debug is difficult within flash or EPROM devices because of the lack of support for S/W breakpoints due to the inability to do instantaneous writes to these types of memories. Writes to flash must be performed through the use of custom algorithms developed for use with the 547x devices or other means. Spectrum Digital has created algorithms for use with their SD Flash utility to program the flash memories on their 5470/5471 EVM boards. The bootloader example code has some modifications to allow for booting from external flash devices and compatibility with debug within external SRAM. These special modifications are in the form of embedded assembly code instructions within the c boot vector program file, **.\arm_boot\intvectors_be_flash.c** for big-endian or **.\arm_boot\intvectors_le_flash.c** for little-endian, to properly configure the ARM MEMINT, API_REG, and the DSP. This code is linked at the ARM reset vector 0x0000:0000. Similar code is replicated within the Code Composer Studio Gel file, **.\gel\c5471_be_ramlow_mcu.gel** for big-endian or **.\gel\c5471_le_ramlow_mcu.gel** for little-endian, to allow for proper setup when debugging from external SRAM mapped to the ARM reset vector. The executable produced from the Code Composer Studio project build is compatible for both external SRAM loading and external flash image building and booting. Within the Code Composer Studio external SRAM debug session, the bootloader COFF will be loaded at the C entry point label “c_int00”. When executing the bootloader from external flash, the C boot vector program will take care of the necessary device configuration, as previously mentioned, before branching to the C entry point.

2 References

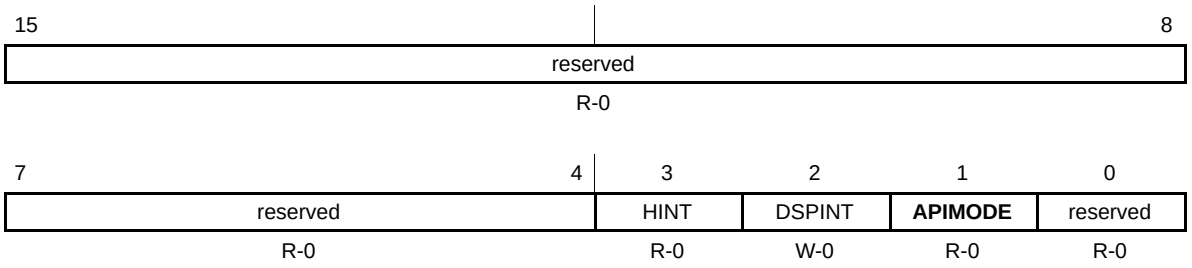
1. *TMS320C54x Assembly Language Tools User's Guide* (SPRU102).
2. *TMS320C54x DSP CPU and Peripherals Reference Set Volume 1* (SPRU131).
3. *TMS320VC547x CPU and Peripherals Reference Guide* (SPRU038).
4. *TMS320VC5471 Fixed-Point DSP Data Manua*l (SPRS180).
5. *TMS320VC5470 Fixed-Point Digital Signal Processor* (SPRS017).
6. *TMS470R1x 32-Bit RISC Microcontroller Family User's Guide* (SPNU134).
7. *TMS470R1x Assembly Language Tools User's Guide* (SPNU118).
8. *TMS320VC5470/5471 Evaluation Module Technical Reference* (SPRU135).
9. Spectrum Digital SD Flash Utility (http://www.spectrumdigital.com)

Appendix A DSP and MCU Registers



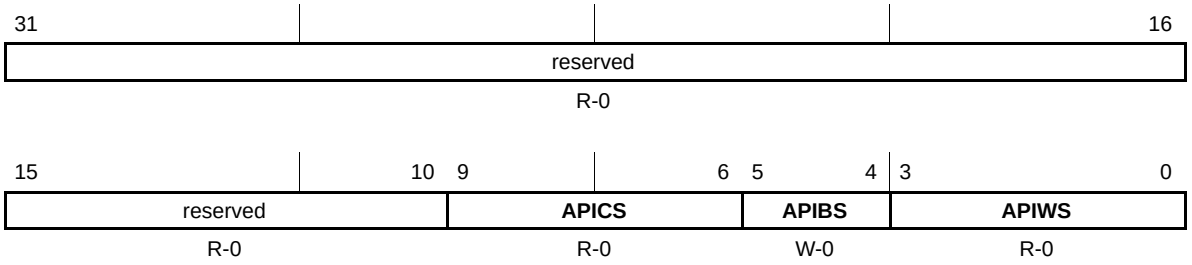
Legend: R/W = Read/Write

Figure A–1. DSP Bank-Switching Control Register (BSCR)



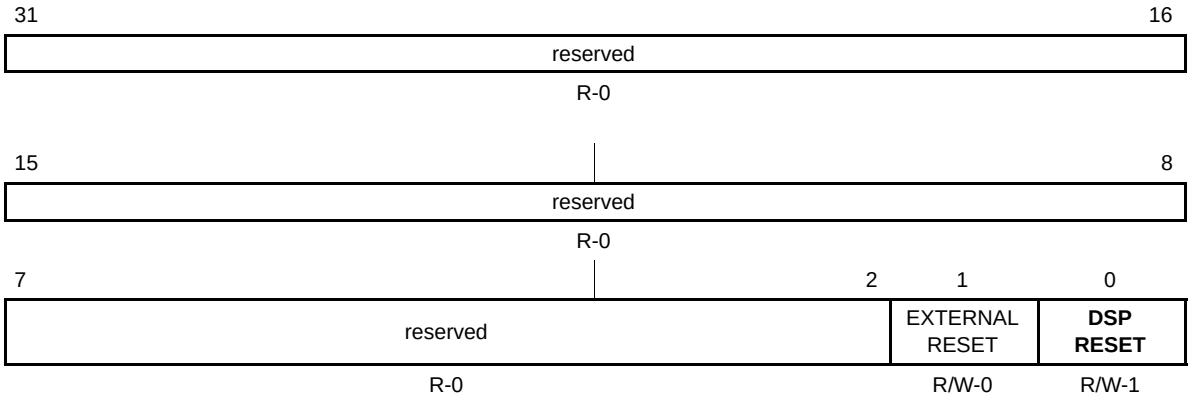
Legend: R = Read only; W = Write only

Figure A–2. MCU API Control Register (APIC)



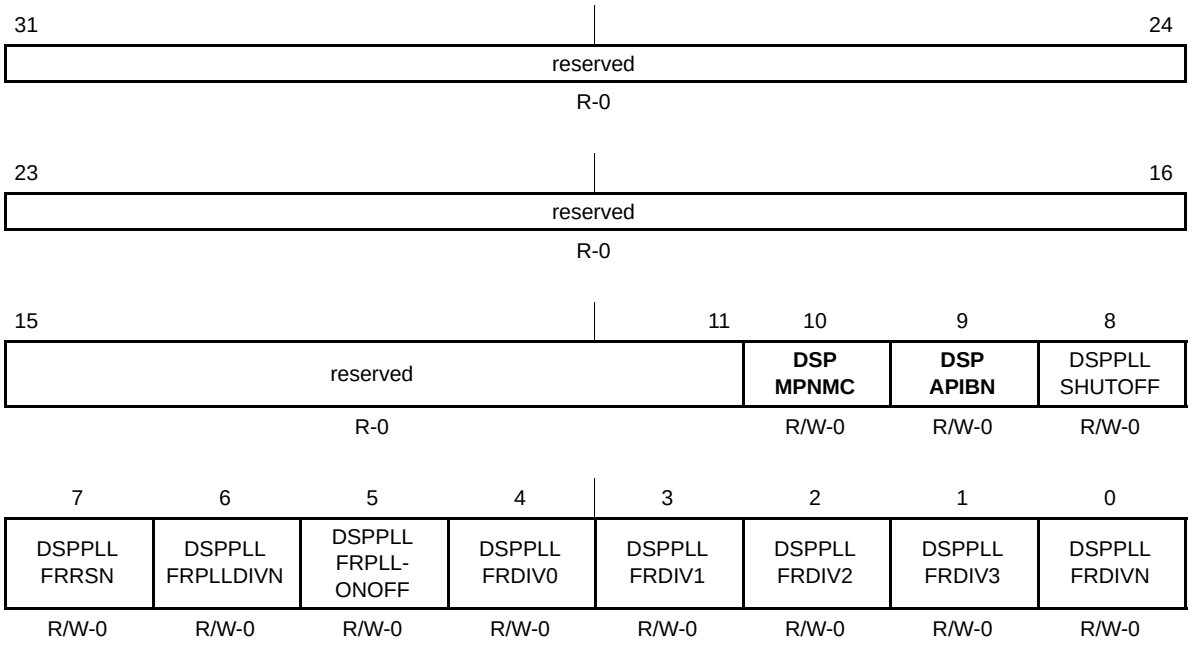
Legend: R = Read only; W = Write only

Figure A–3. MCU Arm Port Interface Wait-State Configuration Register (API_REG)



Legend: R = Read only; R/W = Read/Write

Figure A–4. MCU Reset Control Register (CLKM_CNTL_RESET)



Legend: R = Read only; R/W = Read/Write

Figure A–5. MCU DSP Phase-Locked Loop Register (DSP_REG)

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third–party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265