CIRRUS LOGIC®

AN177

Application Note

USING JTAG FOR DEBUGGING EP72XX MICROCONTROLLERS

Maverick™

ARM POWERED™

P.O. Box 17847, Austin, Texas 78760
(512) 445 7222 FAX: (512) 445 7581
http://www.cirrus.com

TABLE OF CONTENTS

LIST OF TABLES

Contacting Cirrus Logic Support

For a complete listing of Direct Sales, Distributor, and Sales Representative contacts, visit the Cirrus Logic web site at:
http://www.cirrus.com/corporate/contacts/

1. INTRODUCTION

The Cirrus Logic EP72xx microcontrollers use the ARM7TDMI cores that contain hardware debugging support via the JTAG interface. This interface has the following benefits:

- Allows connection of JTAG debugging units, such as Multi-ICE™ from ARM® and JEENI™ from Embedded Performance, Inc. to gain direct access to the ARM processor;
- Does not need separate debugging software on the target board;
- Requires no special hardware on the device;
- Needs no special memory to be set aside for debugging;
- Supports breakpoints via hardware (great for debugging code in ROM).

1.1 Organization of this Application Note

This application note covers the steps needed to configure the 72XX development boards for JTAG support, and on how to set up the software. (See Sections 2 and 3) You will also learn how to use JTAG debugging in the following memory models:

1) A "debug" memory model (ICE_BOOT) (See Section 4)
2) With Angel running in FLASH memory (See Section 5)
3) With ROMed applications installed, such as the Player demo provided by Cirrus Logic. (See Section 6)

Of the three memory models we explain, the ICE_BOOT memory model is the most flexible for use in a debugging environment. Cirrus Logic, Inc. has provided the files necessary to build ICE_BOOT along with this application note.

2. DEVELOPMENT BOARD HARDWARE SETUP

In order to use the JTAG port for debugging, it is necessary to connect TEST0 and TEST1 of the EP72xx to ground. On the EDB7209-2X.0 board, this is accomplished by placing jumpers on JP47 and JP48. In the case with the EDP7111-2 (with EP7211 installed), grounding these two signals is a bit more involved. You need to solder a wire from ground to TP8 and TP9. Fortunately, there is a ground nearby at JP33.

3. SOFTWARE SETUP

3.1 Using JEENI

With JEENI, the software required should be available in the ARM Debugger. Select "remote_a" from the ***Options→Configure Debugger…*** menu (serial port option only). Refer to the JEENI *User's Manual* for instructions on how to install the Ethernet interface.

3.2 Using ARM Tools

Here are a few hints to remember when you work with ARM Tools.

- You must run the Multi-ICE Server before launching the Multi-ICE debugger. Use the configuration file "720.cfg".
- Make sure you are using the ARM Tools Version 2.5. Multi-ICE will not operate correctly with the Version. 2.11a debugger (JEENI is compatible with 2.11a).
- When using Multi-ICE, make sure that the file "Multi-ICE.dll" resides in the ARM250\bin directory
- Confirm that "autoexec.bat" contains the following (your drive letter or ARM directory name may be different depending on how you installed the software):

  SET PATH=C:\ARM250\BIN

  SETARMLIB=C:\ARM250\LIB

  SET ARMINC=C:\ARM250\INCLUDE

- The processor must be running before Multi-ICE can access the processor core — it cannot be in the Standby or Idle modes. Otherwise you will get a "processor not stopped" error. This error is also displayed if TEST0 and TEST1 are not grounded.

4. DEBUGGING WITH ICE_BOOT

ICE_BOOT is a program that sets up a memory configuration that is "ideal" for general purpose debugging with a JTAG environment. In ICE_BOOT, DRAM is placed at the beginning of the memory starting at location 0x0000.0000. This memory location is where the exception vectors reside. It also is placed into a state that is compatible with the default settings of the ARM debugger and ARMulator.

NOTE: When using the memory model defined by ICE_BOOT, set the entry location used in the demonstration programs from 0x0802.8000 (as used by Angel) to 0x8000 (the ARM "default"). This is accomplished by setting the ARMLINK entry points in the Project Manager.

The program compiles as a straight binary object that loads into FLASH memory starting at location at 0x0. The DOWNLOAD.EXE program that is included with the Developer's Kit is used to program the FLASH. After programming the FLASH, you need to recycle the power on the board, causing the program to carry out the following actions:

NOTE: If you are using a Development Board from Cirrus Logic, consult the appropriate *Hardware User's Guide* for instructions on how to recycle power to the board.

1) Sets up the jump locations for the exception vectors.
2) Sets up the ARM debugger for little-endian 32-bit code and data
3) Disables all interrupts
4) Enables the DRAM memory.
5) Sets the CPU for highest clock speed
6) Sets up the chip selects and the attributes of the memory regions.
7) Creates the MMU translation tables
8) Copies the program from FLASH to DRAM
9) Enables the MMU
10) Goes into an endless loop while flashing the LED. Code is now running out of cached DRAM. The processor must be running before invoking a JTAG-ICE debugger.

The memory map that the program ICE_BOOT defines is shown in Table 1 on page 5:

In order to minimize the size of the translation tables, the memory region from 0x9000.0000 to 0xFFFF.FFFF is undefined and will cause a "data abort" error if that region is accessed. The translation table has over 2300 entries (>9 Kbytes), and this is small enough so that it will not interfere with user programs loaded at the default address of 0x8000. Most all of the memory is defined in the Level I translation table. The only Level II entry is support of the internal registers starting at 0x8000.0000.

The region from 0x1000.0000 to 0x5FFF.FFFF is mapped as one contiguous region for Chip Selects CS1, CS2, CS3, CS4, and CS5. You can determine the functions of these chip selects from the hardware configuration of the development board and the comments in the source code.

When using the debugger with this program running, you can use the default values of the debugger settings. For programs larger than about 450 Kbytes, the $top_of_Memory value should be changed from the default (0x80000 or 512 Kbytes). The debugger uses the default value as the top of the stack area. If this value is not increased with large programs, there is the possibility that the code will collide with the stack. Possible values to use are the top of internal SRAM (0x6000.9600) or the end of DRAM (0x00FF.FFFF). If you use the LCD controller, make sure the frame buffer is clear of the stack area.

From	To	Description
0x9000.0000	0xFFFF.FFFF	Invalid Memory Region
0x8000.E000	0x8FFF.FFFF	255 MB of inaccessible memory
0x8000.0000	0x8000.DFFF	Internal Register space
0x7010.0000	0x7FFF.FFFF	255 MB of inaccessible memory
0x7000.0000	0x700F.FFFF	Internal ROM (only first 128 bytes is valid)
0x6010.0000	0x6FFF.FFFF	255 MB of inaccessible memory
0x6000.0000	0x600F.FFFF	On-chip SRAM (only first 37.5 KB is valid)
0x10000000	0x5FFF.FFFF	1.25 GB of non-cacheable mapped r/w memory (parallel port, NAND FLASH, Ethernet, USB, PCMCIA, touch screen, SmartMedia, etc.)
0x0200.0000	0x0FFF.FFFF	224 MB of inaccessible memory
0x0100:0000	0x01FF:FFFF	16 MB NOR FLASH
0x0000:0000	0x00FF:FFFF	16 MB DRAM

Table 1. ICE_BOOT Memory Map

5. RUNNING ICE WITH ANGEL INSTALLED

Angel programs the MMU as shown in Table 2.

If you use Multi-ICE or JEENI, Angel becomes irrelevant. However, Angel does set up the MMU to provide enough DRAM memory to load and debug some large programs and data arrays. Notice that this memory map differs from that of ICE_BOOT since FLASH occupies the first 8 Mbytes. So it is not possible to alter the exception vectors directly. Also, user programs must be linked starting at location 0x0802.8000. This means that it is necessary to change some settings in the ADW (ARM Debugger for Windows) when using Multi-ICE or JEENI. Make the following changes:

1) The debugger variable $top_of_memory must be set to 0xC100.0000 or 0x0900.0000. The default value of 0x80000 will generate "data abort" errors since there is no virtual DRAM there.

2) Set the variable $vector_catch = 0. The new setting allows you to debug and view ROM starting at location zero. This prevents Multi-ICE from trying to set software break-points on the vector table.

The debugger variables can be changed by clicking on ***View→Debugger Internals***.

Please note that each time that Multi-ICE or ADW is restarted, these variables will need to be changed again. Any changes you make to ADW settings are not saved from one session to the next.

From	To	Description
0xC000.0000	0xC0FF.FFFF	DRAM Bank 0
0x8000.0000	0x8000.1800	EP7211/09 registers
0x7000.0000	0x7000.007F	On-chip boot ROM
0x6000.0000	0x6000.95FF	On-chip SRAM
0x5000.0000	0x5FFF.FFFF	Expansion Header
0x4000.0000	0x4FFF.FFFF	USB Interface Device
0x3001.0000	0x3001.FFFF	Keyboard scan latch and Touch Screen
0x3000.0000	0x3000.FFFF	Parallel Port
0x2000.0000	0x2FFF.FFFF	Ethernet controller CS8900A
0x1000.0000	0x1FFF.FFFF	NAND FLASH
0x0802.8000	0x08FF.FFFF	User program space (DRAM)
0x0802.0000	0x0802.7FFF	Angel workspace (DRAM)
0x0800.0000	0x0801.FFFF	LCD frame buffer (DRAM)
0x0000.0000	0x007F.FFFF	FLASH Bank 0

Table 2. Angel Memory Map

6. RUNNING MULTI-ICE WITH THE PLAYER DEMO.

Player is the MP3/WMA audio playback program that is available from Cirrus Logic. Unlike Angel or ICE_BOOT, Player does not use DRAM. The only RAM available is on-chip.

The following table illustrates how memory is set up for the Player Demo.

Note how much more compact this memory model is. The size of the Translation table is relatively small since there are fewer entries (five Level 1 entries for a total of 20 bytes). This is the preferred method in an embedded environment where memory is at a premium.

Despite the lack of DRAM, it is still possible to do source-level debugging of Player. The steps to do this are as follows:

1) Compile the Player program using the Debug Variant in the Project Window. This will create Player.rom in the \Debug directory.
2) Use the Download utility to program the Flash memory with Player.rom
3) Start the Player
4) Start the debugger with MultiICE
5) Set $vector_catch = 0
6) Load the symbols for the player. Note that under File, there is a “Load” and “Load symbols only” menu option. Use the second one. Select Player.axf in the \Debug directory.
7) Set the PC to 0. This simulates a reset (Click on ***View*** → ***Registers*** → ***Current Mode***).
8) Click on Run or press the F5 key.
9) Press the User 1 or User 2 key (depending on release number) on the evaluation board to start the music playing.
10) At this point you can stop and restart the program using the debugger Stop and Run commands.

From	To	Description
0x0050.0000	0xFFFF.FFFF	Undefined. Accesses generate Data Abort
0x0040.0000	0x004F.FFFF	USB interface (nCS4)
0x0030.0000	0x003F.FFFF	NAND FLASH interface (nCS1)
0x0020.0000	0x002F.FFFF	EP7209 internal registers
0x0010.0000	0x001F.FFF	1 Mb Internal SRAM (only 38.4K exists)
0x0000.0000	0x000F.FFFF	1Mb of program ROM (nCS0)

Table 3. Player Demo Memory Map

7. ALTERNATIVE DEBUGGING METHOD

The previous method uses the *native* environment for source level debugging of Player. It has the advantage of being easy to set up and use. However, there are some inherent disadvantages:

- Since code resides and Flash, you cannot change program memory (although you can change registers and data in SRAM)
- Downloading the program to Flash takes some time, which can be inconvenient if you need to make many changes to the code.

An alternative is to build the Player program in the ICE_Boot *debugging* environment. In this case, code is loaded into DRAM and debugged there. Code downloads are via MultiICE, which is very fast. And since all code is in DRAM, you can modify the code, change interrupt vectors, and do other debugging tricks that may not be possible using the *native* mode.

NOTE: This should only be performed by experienced or daring programmers.

Building the project for ICE_Boot mode requires that you make some changes to the source code. In particular:

- All registers need to be redefined with a base address of 0x8000.0000 (instead of 0x0002.C000).
- The base address of the internal SRAM needs to be changed to 0x6000.0000 (instead of 0x0002.0000).
- The base address of the NAND Flash needs to be changed to 0x1000.000 (instead of 0x0030.000).
- The base address of the USB port needs to be changed to 0x4000.0000 (instead of 0x0040.0000).

The easiest way to manage this is to create a new project file and add #defines (for C) and :DEF: (for assembly) to the source files. The files affected are: `ep7209.h`, `ep7209.inc`, and `vectors.s`. The changes to `vectors.s` are such that all code that sets up the MMU and the memory configuration registers are bypassed, as this is already set up in ICE_BOOT. (These tasks are performed in the Reset Handler routine.)

8. GENERAL HINTS

Here are some tips or hints that will assist you in carrying out your debugging activities:

1) As mentioned earlier, you can click on ***View →Debugger Internals*** to set the debugger's internal variables. But, here are two alternative ways of accomplishing the same task
   - In the Command Window of the ADW, you can assign the values directly, for example, type $vector_catch=0 then press the Enter key.
   - You can also create a text file with a list of debugger commands. Then in the Command Window, type Obey filename, where filename is the ASCII file (created with Notepad) that contains the commands.
2) Under Windows® 95/98, you can view the AUTOEXEC.BAT file by running SYSEDIT. Go to the START button, click on RUN, then type SYSEDIT and click on OK.
3) Any "Data Abort" errors that the debugger displays are due to either a bad value for $top_of_memory, or the memory region being examined is invalid.
4) A "Processor not stopped" message indicates that either TEST0 and TEST 1 are not grounded or that the processor is in the Idle or the Standby state
5) If you use SWI calls in your code, be sure to use the -fz compiler option before compiling the project in the ARM Project Manager. To choose this option go to ***Project → Tools Configuration for xxx.apj → <cc>=armcc → set***; then check "Inline SWI's may overwrite link register" under the *Code Generation* tab.

6) With Multi-ICE, you need to cycle the power on the development board in order to restart the ROM program in your development board.

7) An updated ARM debugger for Multi-ICE (ADW or MDW) is installed by Multi-ICE setup. Use this debugger as it supports Multi-ICE in ***Options → Configure Debugger***.


SMART
Analog
™