i.MX53 On Board Diagnostics Suite (OBDS) Users Guide

Revision History

VERSION	DATE	CHANGE DESCRIPTION
1.0.0	27-Apr-2010	Initial draft
1.0.1	05-May-2010	Updated doc with info on obtaining GNU tool chain from internal Compass site and several other edits from team review (references to Redboot).
1.0.2	21-May-2010	Updated doc with Figure 2 for Rev B EVK, added info regarding board ID check under the I2C device test, removed registers.h file, addition of programming SD card using Linux, and various minor editing.
1.1	18-June-2010	Updated doc with changes related to OBDS now being bootable, addition of tests and support of the Smart Mobile Reference Design (i.e. Quicksilver) and various other doc fixes. Removed support and references to Rev A EVK.

Table of Contents

1 Overview

The On Board Diagnostics Suite (OBDS) is a suite of tests designed to ensure the hardware connectivity between the SoC and on board peripherals to help ensure working and bug-free board systems. The i.MX53 OBDS is specifically tailored to test the peripherals on the i.MX53 EVK or the Smart Mobile Reference Design development board with little user interaction.

This document provides the user with a detailed overview of the OBDS including
- An overview of the OBDS directory structure
- The use of makefiles to build the OBDS project
- Descriptions of each test
- Installation of the GNU Toolchain
- How to build OBDS using GCC under Linux and Windows
- How to run the OBDS test suite

Note that the OBDS test can be executed from either a debugger (like ARM RVD) or from an SD card. The OBDS binary (.bin file) is a bootable binary that can be programmed into an SD card without the need for an additional bootloader. This allows the OBDS test to execute immediately after booting from SD. In addition, Redboot may also be used as the bootloader that allows the development board to boot from the SD card and then execute the OBDS test suite.

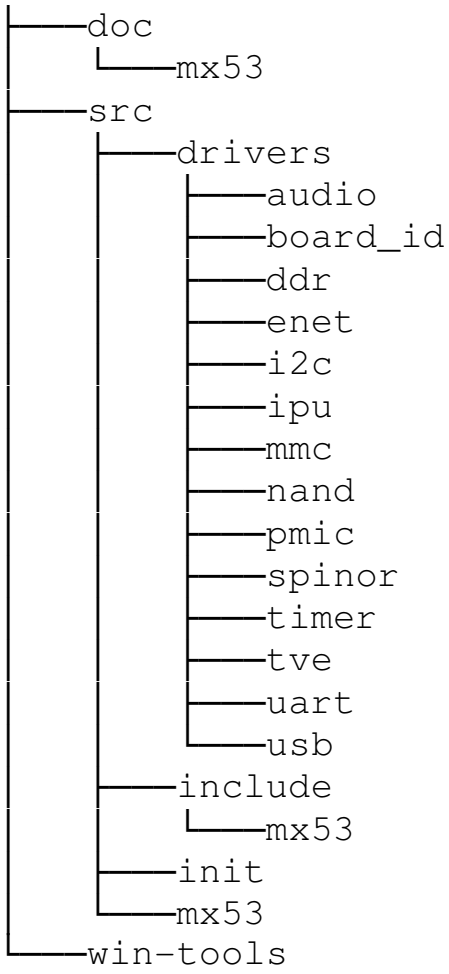
The OBDS release for the i.MX53 provides:

1. Source code for the OBDS test suite
2. The latest Redboot binary at the time of the OBDS release
3. The cfimager.exe tool for programming the Redboot binary and OBDS test suite onto an SD card
4. The RVD include file (`.inc`) for the i.MX53 development system
5. Pre-built OBDS binary and elf file for immediate testing

It is recommended that the user familairizes himself with the development system under test by reviewing the schematic and the board component placement.

2 OBDS Directory Structure

The following provides an illustration of the OBDS directory structure under `./diag-obds`:



In general, each test and driver source code is listed under the sub-directory `./diag-obds/src/driver`. These tests and drivers are "generically" written to allow them to be easily ported to other i.MX processors. For hardware specific (both SoC and board) set up, refer to file `hardware.c` under the directory `./diag-obds/src/mx53`. Other generic code such as the C runtime file and retarget file can be found under `./diag-obds/src/init`. Most of the header files can be found under `./diag-obds/src/include`.

The following sections provide more details into each directory.

2.1 Directory “mx53” Under “doc”

This directory contains the following files:

- `MX53_OBDS_UserGuide.pdf` – The user guide for the MX53 OBDS.
- `Rita_init_DDR2_EVK.inc` – This is the RVD include file to initialize the DDR2 memory for use with the ARM RVD debugger and the EVK.
- `Rita_init_DDR3_QS_CMOS.inc` – This is the RVD include file to initialize the DDR3 memory for use with the ARM RVD debugger and the Smart Mobile Reference Design development system.

2.2 Directory “mx53” Under “src”

SoC and board specific code and set up are found in the directory `./diag-obds/src/mx53`. The following provides a brief description of the files listed under this directory.

- `functions.c` – This file contains various generic function definitions like `reg32_write mask`.
- `hardware.c` – This file contains hardware/board specific setup, like the IOMUX setup and frequency setup.
- `i2c_dev_tests.c` – This file contains the I2C test function calls for the particular board under test. For the EVK, this will read the device ID of the MC13892 PMIC and the board ID chip, while for the SMRD development board, the device ID of the DA9053 is read. For the SMRD development board, there is also an optional test to evaluate the voltage output programmability of certain regulators should the user wish to run this. Note that this test does not generate a pass/fail condition to the OBDS output.
- `main.c` – This file contains the `main()` function, and from `main()` all other tests are called.
- `mx53.ld` – This is the GCC linker definition file, analogous to the ARM scatter file.
- `prog_pll.c` – This is where all the PLLs are initialized to their nominal values using an input clock of 24 MHz. The PLLs are programmed as follows:
  PLL1: 800 MHz
  PLL2: 400 MHz (the DDR is programmed to run at 400 MHz)
  PLL3: 216 MHz
  PLL4: 595 MHz

2.3 Directory “init” Under “src”

The directory `./diag-obds/src/init` contains the C runtime file and vector table (`vectors.S`), the retarget library (`retarget.c`), and the version control file (`version.c`). The following provides a brief description of these files listed under this directory.

- `retarget.c` – This file contains re-implementations of functions whose C library implementations rely on semi-hosting, such as re-directing the printf function to use the serial port.

- `vectors.S` – This is the so-called C-run-time file and sets up the stack. The image entry point label "`Reset_Handler`" is located here and this is where the program starts when using RVD or Redboot to execute the program. When booting from SD, the `PLATFORM_INIT` macro is run which contains the Device Configuration Data (DCD). The DCD is configuration information contained in a Program Image, external to the ROM, that the ROM interprets to configure various peripherals on the SoC. This marco and DCD are located in the file `plat_startup.inc`. The file `vectors.S` also contains the vector table for the ARM core. This file is referenced in the linker file `mx53.ld` which references `Reset_Handler` as the entry point of the code.
- `version.c` – This file contains the function to print out the version information of the OBDS. This outputs the data and time of the current build of the OBDS that is running on the target system.

2.4 Directory "include" Under "src"

Unless otherwise stated, header files are listed under the directory `./diag-obds/src/include`. In this directory, non-SoC specific header files can be found along with i.MX53 specific header files (which are found in the sub-directory `./mx53`). The following provides a brief description of these files listed under this directory.

- `io.h` – This header file contains basic defines for data types, register reads and writes, various structure definitions and various function prototypes.
- `pmic.h` – This header file contains the function prototype for the pmic driver.
- `version.h` – This header file contains the function prototype for the version output function.
- `imx_i2c.h` – This header file contains generic I2C defines, function prototypes and structure, all of which are relevant to the I2C device driver.
- `imx_nfc.h` – This header file contains specifc defines for the NAND flash test driver.
- `mx53 (folder)` – This sub-directory contains header files relevant to the i.MX53 SoC, including:
    - `soc_memory_map.h` – file which contains the i.MX53 module memory map defines
    - `hardware.h` – file which contains chip and board specific defines
    - `iomux_define.h` – file which contains IOMUX register offsets (note that these are relative offsets from the IOMUX_OBSERVE_MUX registers and are not absolute offsets)
    - `iomux_register.h` – file which contains IOMUX module register defines
    - `functions.h` – file which contains general register and memory read/write functions
    - `enet.h` – file which contains FEC module specific defines
    - `gpio_define.h` – file which contains GPIO module specific defines
    - `ipu_reg_def.h` – file which contains IPU module register defines
    - `tve_regs.h` – file which contains TVE module register defines
    - `ccm_pll_reg_defines.h` – file which contains the PLL and CCM module register defines

- `plat_startup.inc` – file which contains the Device Configuration Data (DCD) used by the SoC ROM when booting from SD. The DCD is used to setup and initialize the DDR memory.

2.5 Directory "drivers" Under "src"

The drivers directory contain all of the drivers both for module utility use (like the UART and timers) and for the module test drivers (like the DDR test). The files listed in each sub-directory are abstracted from a particular SoC or board, hence allowing ease of porting to another system.

- `audio` – This folder contains the files necessary to test the audio functionality of the board. The audio driver uses the SSI module to transmit audio information to the audio CODEC and the I2C interface for setting up the CODEC. The SSI and I2C IOMUX set up can be found in `hardware.c`.
- `ddr` – This folder contains the DDR test. The test is called from `main()` and two parameters are passed to the `ddr_test` function – `density` (this is the density per chip select available on the board) and `number_of_cs` (the number of chip selects used on the board).
- `enet` – This folder contains the Ethernet of FEC loopback test. Refer to `hardware.c` for IOMUX setup.
- `i2c` – This folder contains the driver for the I2C module. Refer to `hardware.c` for the IOMUX setup of the I2C.
- `ipu` – This folder contains the files necessary to test the display feature of the development board. Refer to `hardware.c` for IOMUX setup.
- `mmc` – This folder contains the files necessary to test the MMC/SD port, SD slot 2. This test configures and uses the SDHC3 module on i.MX53 for use with SD slot 2. SD slot1, which uses the i.MX53 SDHC1 module, is reserved for booting from an MMC/SD card.
- `nand` – This folder contains the files necessary to test the NAND flash device by reading the NAND flash device ID and prompting the user to confirm if this is the expected device ID. Note that when building OBDS for the SMRD development board, this test is not called as this board does not contain NAND flash.
- `pmic` – This folder contains the files necessary to test the PMIC device by reading the device ID. When building the OBDS for the EVK, the MC13892 device ID test is called. When building for the SMRD development board, the DA9053 device ID test is called.
- `spinor` – This folder contains the files necessary to test the SPI-NOR flash available on the EVK, using the i.MX53 eCSPI1 module and eCSPI1 SS1. Note that when building OBDS for the SMRD development board, this test is not called as this board does not contain SPI-NOR flash.
- `timer` – This folder contains the driver for the EPIT1 timer module. Note that the EPIT base address is defined in `soc_memory_map.h`. This folder also contains a generic test for the SRTC module. The SRTC is module is a special case test as it requires it's own voltage supply rail. The test enables the SRTC and verifies that the counter is running thus confirming that the voltage supply is correctly connected.
- `tve` – This folder contains the driver for the VGA display out test.
- `uart` – This folder contains the driver for the UART1 module. Refer to `hardware.c` for IOMUX setup and the clock setup for the UART module. The IOMUX is configured to mux out UART1 TXD and RXD on to the CSI0_DAT10 and CSI0_DAT11 pins. In addition, the

file `main.c` contains the definition for the debug_uart by defining it as UART1 (refer to "`static struct hw_module *debug_uart = &uart1;`" in `main.c`).

- `usb` – This folder contains the files necessary to test the USB port and HUB on the EVK. Note that when building OBDS for the SMRD development board, this test is not called as this board does not contain a USB HUB.

2.6 Directory "win-tools"

This folder contains the `cfimager.exe` tool for program an SD card. Note that the user will need an SD card reader tool connected to the Windows host machine. Note that the user can also use a Linux machine to program the SD card. Both methods are described in this document.

2.7 Description of the makefile Utility

The OBDS uses makefiles to manage the build process of the OBDS project. Each directory folder contains a makefile that defines which folders or files are included in the make-build process as well as defines the location of the output directory during the build process (where the `.bin` and `.elf` files are located) along with the location of the target library.

The top level directory (under the `diag-obds` folder) contains three noteworthy makefiles describe below.

- `makefile` – This file defines what sub-directory is included in the make-build process (in this case, only the `src` directory is built.
- `make.rules` – This file contains the make rules for linking in the targeted library file.
- `make.def` – This file contains the defines used in the makefile utility. This is also where pre-processor defines are located (for example, the defines based on board type "`evk`" or "`smrd`") along with flags and locations of include directories.

The makefiles in all of the folders are very similar. However, the makefile located in the `./diag-obds/src/init` folder is especially unique as it contains information critical to the build process. This is where the compiler is called to complete the built process and link the object files to produce a `.bin` and `.elf` file. The build procedure is later described in this document.

3 Test Descriptions

The following list describes each test performed by the OBDS. For hardware connectivity details, please refer to Section 4. Hardware Setup.

- UART (serial port) test: The UART port (labeled DEBUG UART on the EVK) is the primary communications channel between the EVK and Host PC and this implicitly tests the transmission capabilities of the serial port. However, the UART test also verifies the receive capabilities of this port by prompting the user to input a character from the Host PC to the serial port. Typing the character "X" (capital "X") will exit this test and move on to the next test.
- DDR test: The DDR test verifies the interface connectivity between the i.MX53 and the DDR memory. This test should not be confused with a stress test that validates robust signal integrity of the interface. Instead, this test focuses on ensuring properly assembly of the memory and i.MX53 by testing for opens and shorts on the interface.
- AUDIO test: The audio test first performs an I2C communications test between the i.MX53 and the SGTL5000 audio CODEC. Once completed, the test will then proceed to output audio data via the SSI/I2S interface to the audio CODEC. The user is prompted to listen for and verify the audio output on the headphones plugged into the HEADPHONE OUT jack.
- IPU (display) test: This test outputs an image to the WVGA display (Chungwa CLAA070VC01 7-inch WVGA TFT LCD). The test also gives the user two options to test an LVDS display (either the AUO T150XG01V02 15-Inch XGA Panel or CHIMEI M216H1-L01 21-Inch HD1080 Panel) and the VGA output. If no LVDS or VGA display is connected to the EVK, the user has the option to bypass these tests.
- I2C (connectivity) test: This tests performs an I2C communications test to the MC13892 PMIC (read back of device ID) and to the LTC2495 (read back board ID) for the EVK. For the SMRD development board, the DA9053 device ID is verified and the user is given the option to run a DA9053 evaluation test.
- MMC/SD port test: This test performs a read/write test to the MMC/SD card plugged into SD slot 2. This test configures and uses the SDHC3 module on i.MX53 for use with SD slot 2. SD slot1, which uses the i.MX53 SDHC1 module, is reserved for booting from an MMC/SD card.
- LED test: This test verifies the functionality of the on-board debug LED (D22) by prompting the user to visually inspect the LED to verify that the LED has been turned ON then OFF. The test function is located in `main.c`.
- ETHERNET (FEC) loopback test: This test performs a loopback test on the Ethernet connector using the i.MX53 FEC module. This test requires the use of a simple loopback Ethernet cable, which is described in Section 4. Hardware Setup, Table 1.
- SPINOR test: This test verifies the interface between the i.MX53 eCSPI1 module and the SPI-NOR flash. Note that when building OBDS for the SMRD development board, this test is not called as this board does not contain SPI-NOR flash.
- USB (HUB interface) test: In this test, the interface between the i.MX53 USBH1 and the USB2517 HUB is verified. This is accomplished by obtaining the device descriptor from the

USB HUB. Note that when building OBDS for the SMRD development board, this test is not called as this board does not contain a USB HUB.

- NAND Flash (device ID) test: In this test, the deice ID of the NAND flash is read and the suer is asked to confirm if this is the correct device ID. Note that this test is only available for the EVK and the SMRD development board does not contain NAND flash. Further note that on the EVK, only the NAND flash socket is available. It is up to the user to populate the socket with the desired NAND flash. If there is no NAND flash in the socket, this test may be bypassed.

4 Hardware Setup

This section describes how to setup the i.MX53 EVK for OBDS testing.

Figure 1 below illustrates the hardware connection locations for the OBDS test for the Rev B EVK. Following the figure is a list of hardware items to be connected to the i.MX53 EVK. For the SMRD development board, as this is not a production board, it is recommended to contact the factory for further details.

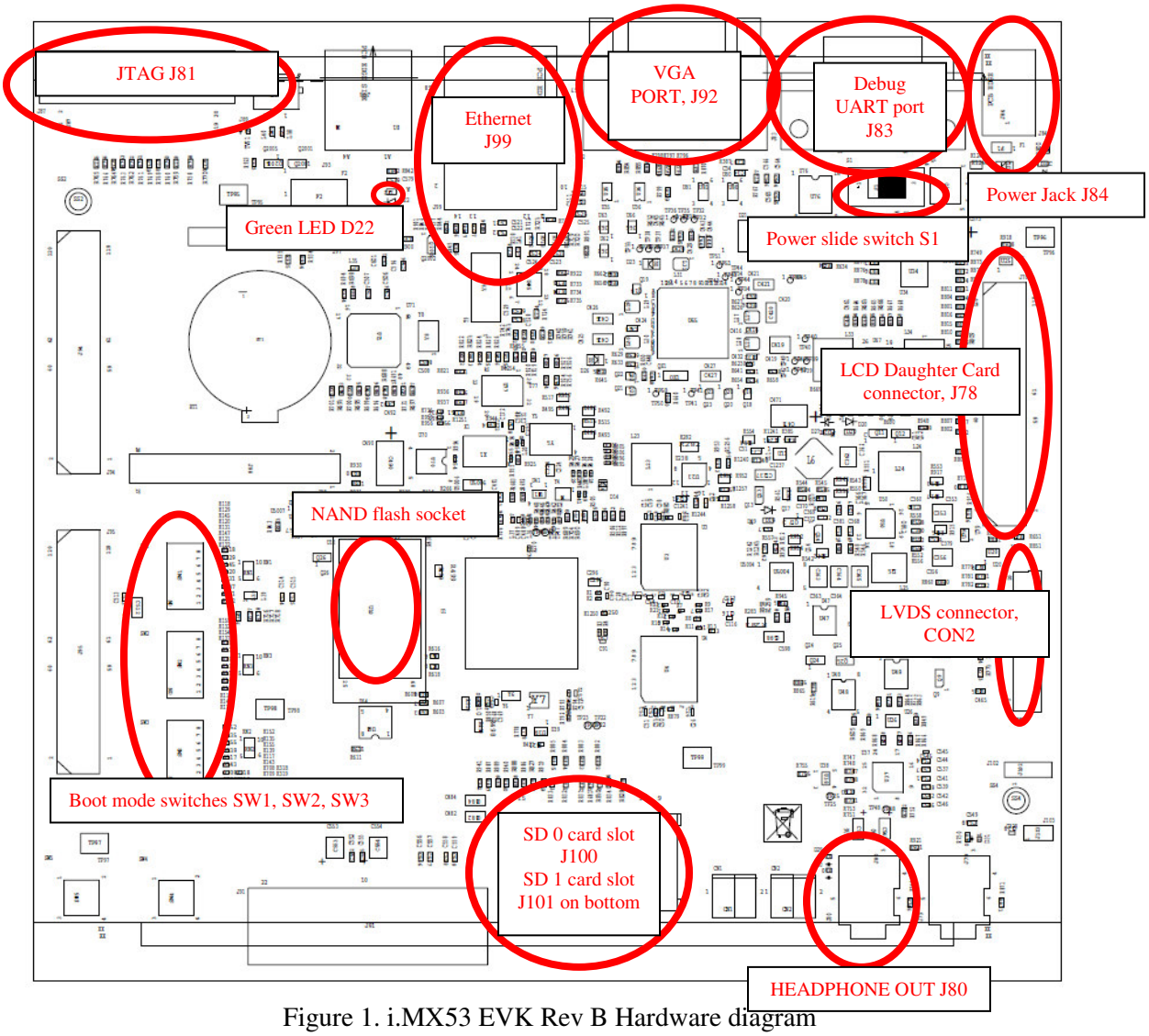


Figure 1. i.MX53 EVK Rev B Hardware diagram

Table 1. i.MX53 EVK Rev B Hardware Connectivity List

Hardware Item and location	Purpose
5V Power supply to Power Jack J84	Supplies power to the board
WVGA LCD daughter card to LCD daughter card connector J78	For WVGA display test
Headphones to HEADPHONE OUT J80	For the audio test
Generic SD or MMC card to SD 1 card slot J101	For the MMC/SD connectivity test
SD card with Redboot and OBDS programmed onto it	Optional: for SD boot testing OBDS
Serial cable to DEBUG UART port J83 and the other end to the Host PC (see note 1)	Needed for serial communication to the Host PC and for the UART test
RealView ICE to JTAG connector J81	Optional: for RVD testing of OBDS
Ethernet loopback cable to Ethernet jack J99	For Ethernet test (see note 2)
LVDS connector CON2 Voltage selection jumper J102 (see note 3)	For the AUO XGA panel, set J102 to short pins 1-2, the 3.3V setting
	For the CHIMEI 1080p panel, set J102 to short pins 2-3, the 5.0V setting
VGA port connector J92	For VGA display
NAND flash socket U10	Optional: for NAND flash testing. NAND flash si not populated by default.

Notes:

1. For the host PC serial terminal program, configure with `BAUD RATE – 115200, DATA – 8 bit, PARITY– none, STOP – 1bit, FLOW CONTROL – none` settings.
2. If an Ethernet loopback cable is not readily available, then one can simply be made. First locate an unused Ethernet cable (eight position RJ-45). Cut the cable to within three inches from the RJ-45 plug and strip back 1 inch of the outer insulator. Then simply short pin 1 (striped green wire) to pin 3 (striped orange wire) and short pin 2 (solid green wire) to pin 6 (solid orange wire).
3. Available on the Rev B EVK.

For SD boot, configure the boot switches per table 1 below.

Table 2. BOOT Switch settings for SD port 0 boot

Switch	1	2	3	4	5	6	7	8
SW1	OFF	OFF	OFF	OFF	OFF	OFF	ON	OFF
SW2	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF

W3	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF

5 Procedure to build and run the OBDS

This section describes the process for building the OBDS test suite and provides instructions on the various options to run the OBDS test suite.

5.1 Image Install Instructions

Unzip the Freescale OBDS release package. You will find the source files for the OBDS software along with the pre-built binary. When you uncompress the release package, the source files will uncompress into the following directory structure:

```
c:/diag-obds
  +---doc
     +---mx53
  +---src
     +---drivers
     +---include
     +---init
     +---mx53
  +---win-tools
```

Note

It is important to ensure the top level directory is labeled as "`diag-obds`" using the "hyphen" character [-] and not "`diag_obds`" which uses an underscore [_].

5.2 Setting up the Tool Chain

The next phase in building the OBDS is to set up the tool chain. The GNU Toolchain for ARM processors can be obtained from CodeSourcery. The GNU Toolchain is also available on the internal Freescale Compass site at `http://compass.freescale.net/go/217844329`. For customers having problems downloading the GNU Toolchain from CodeSourcery, they can contact their local FAE or factory applications engineer for assistance in obtaining the tool chain from Freescale's internal network. To obtain the tool chain used for this release from CodeSourcery, do the following:

- Go to http://www.codesourcery.com/sgpp/lite/arm
- Click on Downloads at the top of the page
- Go to the EABI row and click on "All versions…" under the Download column
- Click on "Sourcery G++ Lite 2008q3-66". This is the release version used for building OBDS.
- Go to Advanced packages and click and save both the Linux and Windows TAR files
- The file extensions will need to be changed on these files, from `*.tar.tar` to `*.tar.bz2`. After changing the file extension, the file names should be:


```
arm-2008q3-66-arm-none-eabi-i686-pc-linux-gnu.tar.bz2
arm-2008q3-66-arm-none-eabi-i686-mingw32.tar.bz2
```

Note

As the link provided above is a link to a third party, Freescale cannot guarantee the above link is maintained or notify users if the link as been changed. If in the event the link has been changed, a simple internet search for "Sourcery G++ Lite 2008q3-66" can be performed to locate the desired tool chain.

Once the tool chain has been obtained, the next step is to install it on either a Linux system or Windows system. Note that building OBDS can be achieved under either Windows using Cygwin or Linux. The next sections discuss setting up the tool chain and how to build OBDS.

5.2.1 Setting up the OBDS GCC Build Environment under Linux

The following discusses the process of installing the tool chain under Linux. It is assumed the user has already downloaded the necessary tool chain discussed previously.

1. Locate `arm-2008q3-66-arm-none-eabi-i686-pc-linux-gnu.tar.bz2`.
2. Under Linux, with root user privilege, go to `/opt` directory (create one if it is not there),
3. Under `/opt` directory, untar `-2008q3-66-arm-none-eabi-i686-pc-linux-gnu.tar.bz2` with:
   `tar jxvf -2008q3-66-arm-none-eabi-i686-pc-linux-gnu.tar.bz2`
   This creates `/opt/arm-2008q3` directory with the tool chain to build ecos under Linux
4. Add `/opt/tools/bin` and `/opt/arm-2008q3/bin` to your environment `PATH` variable.

5.2.2 Setting up the OBDS GCC Build Environment under Windows

To build OBDS under Windows, Cygwin has to be setup first (to obtain the latest version of Cygwin, visit http://www.cygwin.com). Follow these steps to have the tools installed under Cygwin:

1. Locate `arm-2008q3-66-arm-none-eabi-i686-mingw32.tar.bz2`.
2. Unzip `arm-2008q3-66-arm-none-eabi-i686-mingw32.tar.bz2` into the Cygwin root directory using the following command:
   `tar jxvf arm-2008q3-66-arm-none-eabi-i686-mingw32.tar.bz2`
3. After unzipping the tool chain, make sure the directory containing "`arm-none-eabi-gcc.exe`" is in your PATH environment variable.
4. Before using arm-none-eabi tool chain, do the following:
   - Set a environment variable using Cygwin as follows
     `export CYGPATH=cygpath`

5.3 Building OBDS

After the tools are setup properly, follow these instructions to build the OBDS binary:

1. Open Cygwin (for building under Windows) or a Terminal application (for building under Linux).
2. Uncompress the OBDS source package as described earlier.
3. Change the directory to the ./`diag-obds` folder (`cd diag-obds`).
4. Type the following make command to build for the EVK:
   `make TARGET=mx53 BOARD=evk`
5. Type the following make command to build for the Smart Mobile Reference Design (SMRD) dvelopment board:
   `make TARGET=mx53 BOARD=smrd`
6. This creates the OBDS images (`diag-obds-mx53-evk.bin` and `diag-obds-mx53-evk.elf` or `diag-obds-mx53-smrd.bin` and `diag-obds-mx53-smrd.elf` depending on the make option) under the directory `.\diag-obds\output\mx53\bin`
7. The `.elf` file can be used with a debugger like ARM RVD. The `.bin` file may be programmed onto an SD card in conjunction with Redboot.

5.4 How to run OBDS from the debugger

Once built, the OBDS test can be run using a debugger such as ARM RealView Debugger (RVD). This section will assume the use of RVD and will also assume that the user has familiarity with RVD.

The following describes how to configure the RV-ICE for first time connection to i.MX53.

1. Connect RV-ICE to EVK and power on EVK and RV-ICE
2. Open RVD (RealView Debugger)
3. Go to Target pull down menu and select Connect to Target…
4. In the rvdebug.brd pop up window, right click on RealView ICE
   a. Click on Add Configuration
5. A new pop up window will appear, select desired RealView ICE box and hit connect – a new pop up window will appear
6. Configure scan chain as follows (see Figure 2):
   a. Click on Add Device and select Custom Device in pop up window
   b. Under IR Length, select 5 and hit OK
   c. Click on Add Device and select Custom Device in pop up window
   d. Under IR Length, select 4 and hit OK
   e. Click on Add Device and select Registered Devices in pop up window
   f. Expand CoreSight and select ARMCS-DP and hit OK
   g. Click on Add Device and select Registered Devices in pop up window

h. Expand Cortex and select Cortex-A8 and hit OK

i. Right click on Cortex-A8 and select Configuration. Replace the CoreSight base address with 0xC0008000 (see Figure 3)

7. Go to File and Save, then File and Exit
8. In the rvdebug.brd pop up window, double click on Cortex-A8
9. RVD should now be connected to the i.MX53 and ready for debug

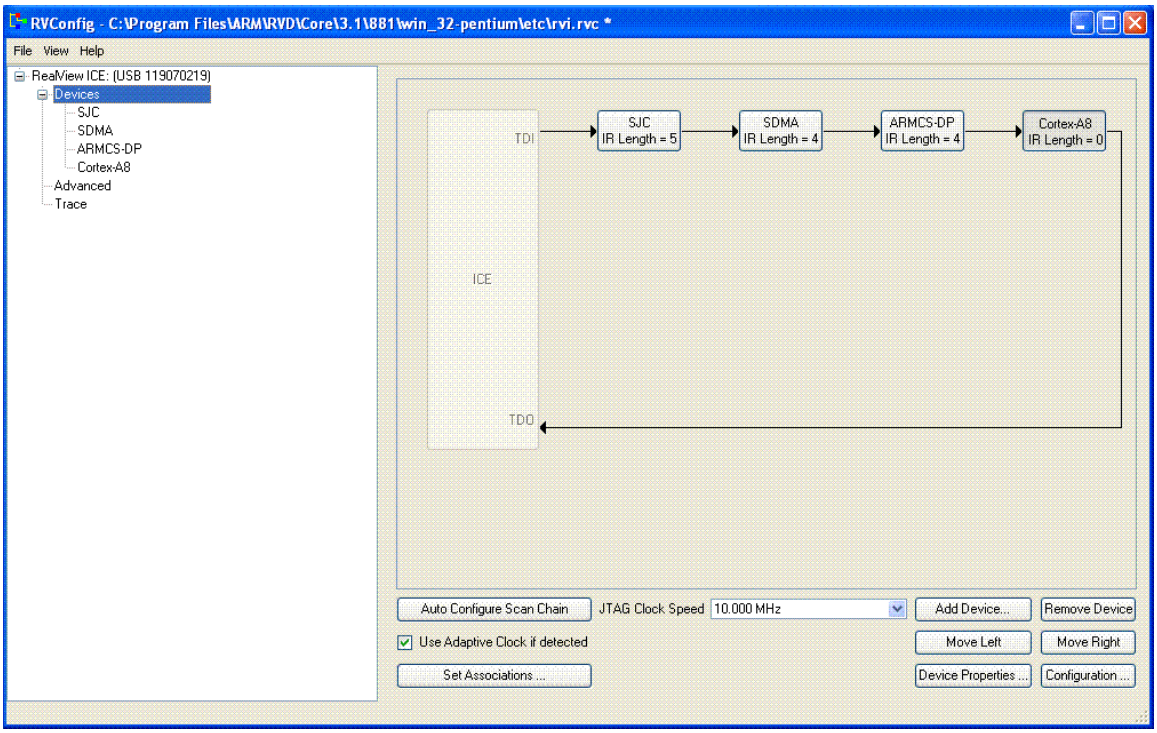


Figure 2. i.MX53 JTAG Scan Chain

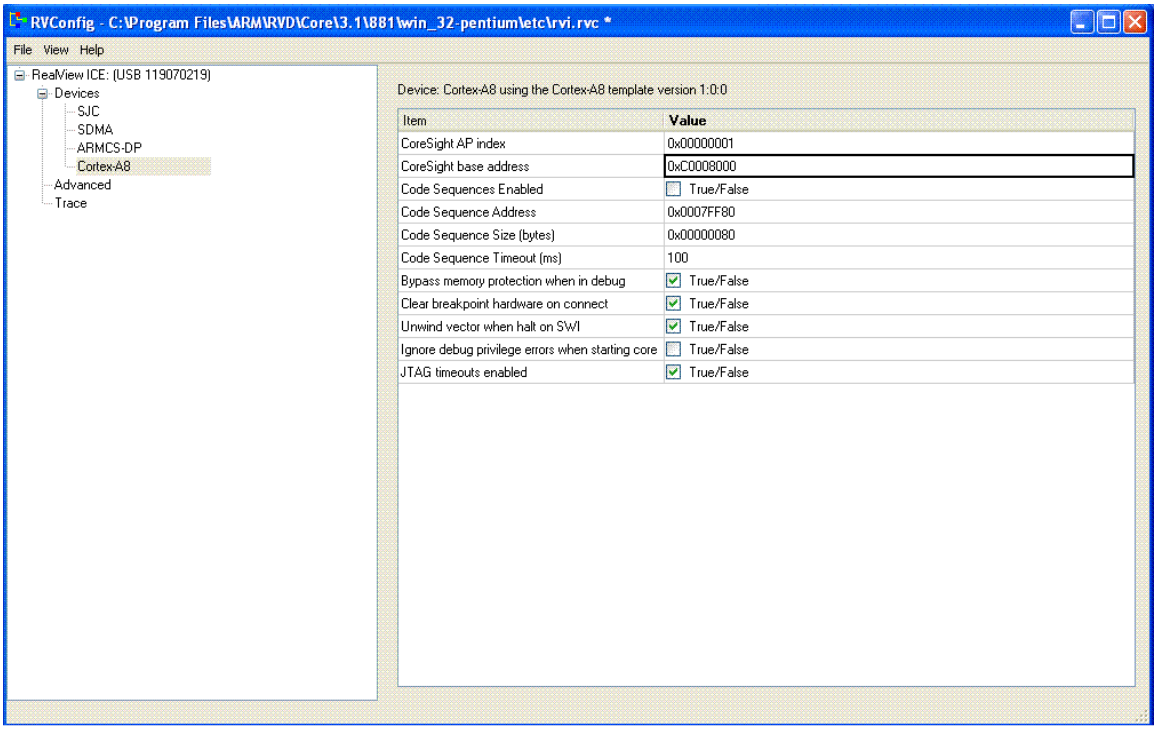


Figure 3. CoreSight base address modification

The next set of steps describe how to run the OBDS test under RVD.

1. Ensure all of the desired hardware components are connected as per section 3 (note that there is no need to install an SD card into SD 0 slot).
2. Start the terminal program on the Host PC per table 1 above.
3. Connect power and RV-ICE to the board and power up the board.
4. Connect to the board via RVD.
5. Run the include file (*.`inc`) for the target board located under the ./`diag-obds/doc/mx53` folder.
6. Load the elf file (`*.elf`) and run. A pre-built version of "`diag-obds-mx53-evk.elf`" and "`diag-obds-mx53-smrd.elf`" can be found under the ./`diag-obds` folder, or for the recently built elf file, go to ./`diag-obds/output/mx53/bin` folder.
7. Watch the host PC serial terminal for OBDS messages.

5.5 How to program and run OBDS from an SD card

The following sections describe two methods with which to launch and run OBDS from an SD card. The first method is to use Redboot as the bootloader to lauch OBDS. The second method is to program OBDS onto the SD card and then boot from SD where OBDS will lauch immediately.

5.5.1 Using Redboot to launch OBDS from an SD card

Redboot is a bootloader that can be configured to launch and run OBDS.

The latest Redboot release package can be found at an internal Freescale Compass site `http://compass.freescale.net/go/redboot`. The OBDS release package also contains the Redboot binary (`mx53_evk_redboot.bin`).

Next, use the `cfimager.exe` tool or Linux to program Redboot onto an SD card. The `cfimager.exe` tool comes with the OBDS release and can be found in the `./diag-obds/win-tools` folder. Before describing the procedure on how to program Redboot onto an SD card using `cfimager.exe`, it is assumed that the user has installed on their PC an MMC/SD card reader and that Windows has assigned this a drive letter. Take note of the drive letter assigned by Windows after inserting the SD card reader into the PC.

To program Redboot onto the SD card using a Windows machine, perform the following steps:

1. If not already present, copy the Redboot binary `mx53_evk_redboot.bin` into the `./diag-obds` folder
2. Open a Windows Command prompt or Cygwin under the `./diag-obds` folder
3. Take note of the `<drive_letter>` assigned by Windows after inserting the SD card reader into the PC, and issue the following command:

Cygwin:

```
$ ./win-tools/cfimager.exe -o 0x0 -f mx53_evk_redboot.bin -d <drive_letter> -a
```

Windows command prompt:

```
> .\win-tools\cfimager.exe -o 0x0 -f mx53_evk_redboot.bin -d <drive_letter> -a
```

Upon successful completion, you should see something like this:

```
drive = G
vendor = Generic
product = STORAGE DEVICE
removable = yes
device block size = 512 bytes
device block count = 0x762c00

firmware size = 0x306ec bytes (0x184 blocks)
extra blocks = 1659
Writing firmware...
writeFirmware from 0 or sector 0
Writing FAT partition...
```



```
Writing MBR...
setPartitionEntry m_blockCount = 730c00, m_startBlock = 32000,
deviceBlockCount
= 762c00
done!
```

If using a Linux machine, perform the following steps to program the SD card with Redboot:

1. Ensure an SD card reader is plugged into the Linux machine, take note of the SD card DEV entry when the card is plugged in (you can type `cat /proc/partitions` to see it)
2. Type the following command (the text "`<YOUR SD card DEV ENTRY>`" indicates the DEV entry of the CD card reader for your machine, for example this may be "`/dev/sdb`" assuming "`sdb`" is the DEV entry):
   `sudo dd if=mx53_evk_redboot.bin of=<YOUR SD card DEV ENTRY> bs=1024`

Once Redboot has been successfully programmed onto the SD card, the next step is to program the OBDS binary onto the SD card. To program OBDS onto the SD card using a Windows machine, perform the following steps:

1. Locate the pre-built copy of `diag-obds-mx53-evk.bin` (or `diag-obds-mx53-smrd.bin`) under the folder ./`diag-obds`. Note that you can replace this binary anytime with a newer binary you build, which can be found in `./diag-obds/output/mx53/bin` (make sure to copy this to the ./`diag-obds` folder)
2. Open a Windows Command prompt or Cygwin under the ./`diag-obds` folder
3. Take note of the `<drive_letter>` assigned by Windows after inserting the SD card reader into the PC, and issue the following command (note that the EVK binary is used for illustration purposes and can be replaced with the SMRD binary):

Cygwin:

```
$ ./win-tools/cfimager.exe -o 0x100000 -f diag-obds-mx53-evk.bin -d
<drive_letter> -a
```

Windows command prompt:

```
> .\win-tools\cfimager.exe -o 0x100000 -f diag-obds-mx53-evk.bin -d
<drive_letter> -a
```

Upon successful completion, you should see something like this:

```
drive = G
vendor = Generic
product = STORAGE DEVICE
removable = yes
device block size = 512 bytes
device block count = 0x762c00

firmware size = 0x5c560 bytes (0x2e3 blocks)
extra blocks = 1308
Writing firmware...
writeFirmware from 100000 or sector 2048
Writing FAT partition...
```



```
Writing MBR...
setPartitionEntry m_blockCount = 730c00, m_startBlock = 32000,
deviceBlockCount
= 762c00
done!
```

If using a Linux machine, perform the following steps to program the SD card with OBDS:

1. Ensure an SD card reader is plugged into the Linux machine, take note of the SD card DEV entry when the card is plugged in (you can type `cat /proc/partitions` to see it)
2. Type the following command (the text "`<YOUR SD card DEV ENTRY>`" indicates the DEV entry of the CD card reader for your machine, for example this may be "`/dev/sdb`" assuming "`sdb`" is the DEV entry):
   ```
   sudo dd if=diag-obds-mx53-evk.bin of=<YOUR SD card DEV ENTRY> bs=1024
   seek=1024
   ```

Note that for the above command the EVK binary is used for illustration purposes and can be replaced with the SMRD binary.

5.5.1.1 Configure Redboot to Run OBDS

After Redboot and OBDS have been successfully programmed onto the SD card, the next step is to configure Redboot to launch and run OBDS. The following describes how to configure Redboot.

1. First ensure that the hardware is setup to boot from MMC/SD as per Section 3 above.
2. Insert the SD card (with Redboot and OBDS programmed onto it) into SD slot 1
3. Power on the EVK and verify you can see messages output to the host PC serial terminal
4. In the host PC serial terminal, wait some time for the `Redboot>` prompt, or simply type `<CTRL> c` to get to the prompt quicker
5. At the `Redboot>` prompt, type `fconfig`. In the `fconfig` structure, type `true` for the "`Run script at boot:`" option, followed by the run script instructions (see below for script in **bold text** and note the empty line following the "`run 0x100000`" command – the empty line is created simply by hitting return), and then type `false` for "`Use BOOTP for network configuration:`" (note, when "`Gateway IP address:`" simply type "`.`" then enter to exit `fconfig`). The following illustrates these settings:

```
RedBoot> fconfig
Run script at boot: true
Boot script:
Enter script, terminate with empty line
>> sd 0 0x100000 0x100000 0x100000 r
>> run 0x100000
>>
Boot script timeout (1000ms resolution): 2
Use BOOTP for network configuration: false
Gateway IP address: .
```



```
Update RedBoot non-volatile configuration - continue (y/n)? y
... Read from 0x1dbe0000-0x1dbff000 at 0x00040000: .
... Erase from 0x00040000-0x00060000:
... Program from 0x1dbe0000-0x1dc00000 at 0x00040000: .
RedBoot>
```

Now that Redboot has been configured to launch and run OBDS, simply power cycle the board with the SD card installed and watch the host PC serial terminal for OBDS messages.

5.5.2 Booting and running OBDS directly from an SD card without using Redboot

To lauch OBDS from an SD, the first step is to program the SD card with the `.bin` binary. The user can simply follow the steps above to program the SD card using Windows (cfimager) or Linux. However, note that in this case, do not use an offset for programming into the SD card. The following examples illustrate this.

For Windows (cfimager under Cygwin), do the following (using the EVK binary as an example):

```
$./win-tools/cfimager.exe -o 0x0 -f diag-obds-mx53-evk.bin -d <drive_letter> -a
```

For Linux, do the following (using the SMRD binary as an example):

```
sudo dd if=diag-obds-mx53-smrd.bin of=<YOUR SD card DEV ENTRY> bs=1024
```

5.6 OBDS Test Output

Refer to Section 3. Test Descriptions for a complete list of modules being testing.

During the OBDS test run, the user will be prompted to interact with the host PC serial terminal answering whether or not they observed specific events such as seeing an image on the LCD or hearing audio through the headphones. Below (Figure 4) is an example screen shot of the OBDS test output as seen on the host PC serial terminal.

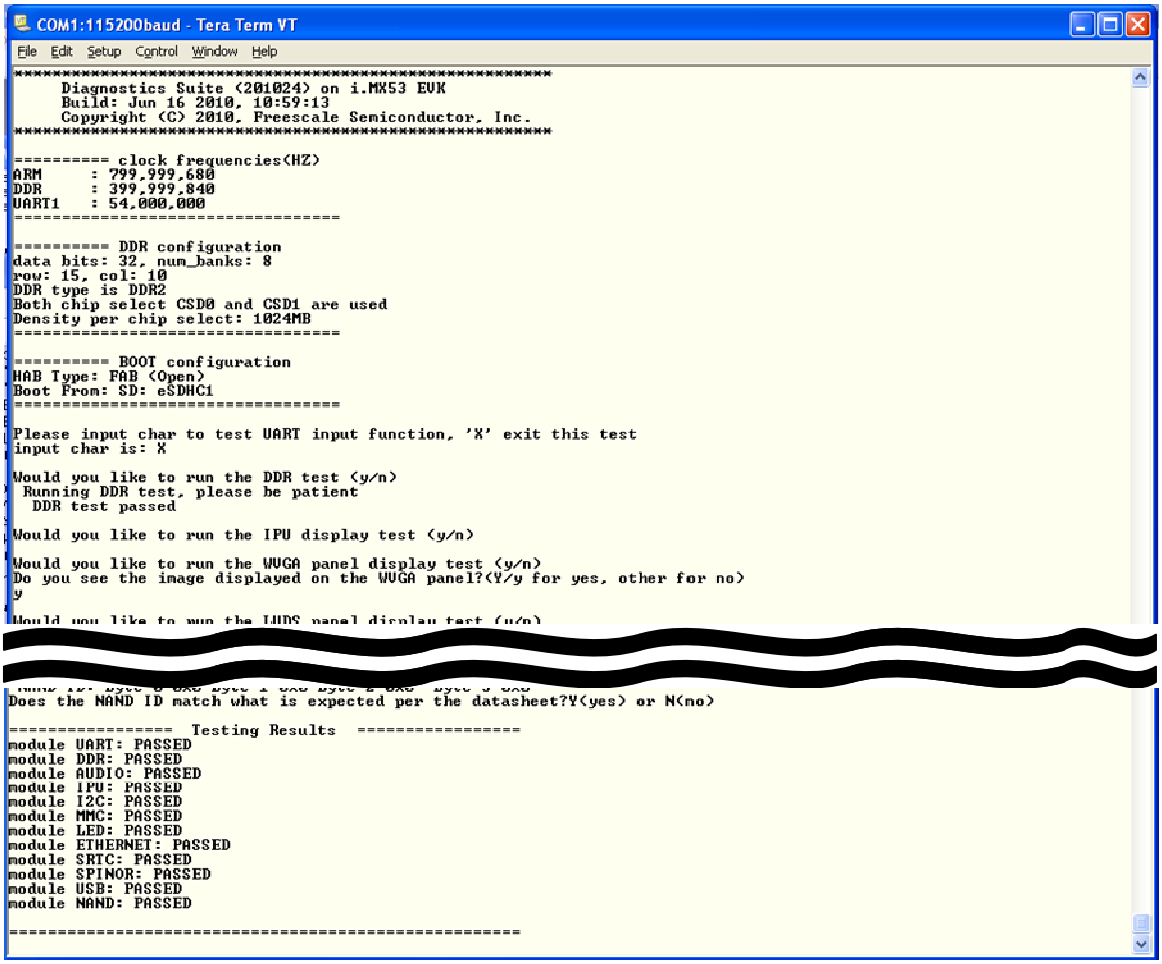


Figure 4. Example screen shot of the OBDS test output

6 Definitions, Acronyms, and Abbreviations

List of Acronyms/Abbreviations used in this document:

TERM/ACRONYM	DEFINITION
GCC	GNU C Compiler
SoC	System on Chip – refers to the microcontroller
i.MX	Generic term for Freescale's multi-media processors
OS	Operating System
OBDS	On Board Diagnostics Suite
RVD	ARM RealView Debugger
RV-ICE	ARM RealView ICE
SMRD	Smart Mobile Reference Design
EVK	Evaluation Kit Development System