Application Note 133

AN133 - VTOS Tools i.MX6 Processor Addendum

May 11, 2017

Table of Contents

Overview

This addendum describes the VTOS features specific to the i.MX6 processor family, manufactured by NXP.

Supported Processors

The VTOS package for the i.MX6 processor family supports the following processor types:

- i.MX6 Solo (i.MX6S)
- i.MX6 DualLite (i.MX6DL)
- i.MX6Dual (i.MX6D)
- i.MX6Quad (i.MX6Q)
- i.MX6 UltraLite (i.MX6UL)

The following processor types are not supported at this time.

- i.MX6 SoloLite (i.MX6SL)
- i.MX6 SoloX (i.MX6SX)

Contact sales@kozio.com if you have a design using an unsupported processor type.

Loading the VTOS firmware images onto your board

The VTOS firmware images are configured to run directly from the on-chip memory of the i.MX6. One of the i.MX6 UARTs is used for communication with the VTOS Tools applications.

VTOS i.MX6 Firmware Files

The VTOS Tools installer provides VTOS i.MX6 firmware files under the %KOZIO_HOME% directory.

Directory	Description
%KOZIO_USER_HOME%\VTOS_DDR\fw	Firmware image files for VTOS DDR.
%KOZIO_USER_HOME%\VTOS_Program\fw	Firmware image files for VTOS Program
%KOZIO_USER_HOME%\VTOS_Scan\fw	Firmware image files for VTOS Scan and VTOS ScanPlus

Tip: You can quickly traverse to the above folders by opening Windows File Exporer entering the folder name (e.g. %KOZIO_HOME%\VTOS_Program\fw) as the location.

All VTOS i.MX6 firmware files conform to the following naming convention:

```
vtos.imx6_<product_type>.<uart>.<filetype>
```

Where the <product_type>, <uart>, and <filetype> fields are each replaced with one of the following specifiers.

Firmware filename parameter	Value	Description
<product_type>	ddr	VTOS DDR firmware image file
	scan	VTOS Scan Base firmware image file
	scan_plus	VTOS Scan Plus firmware image file
	program	VTOS Program firmware image file
<uart>	See below	Specifies the processor type, UART interface

		number, and the specific pins on the i.MX6 package used for communication with VTOS Tools.
<filetype>	bin	Firmware image file, raw binary format. For loading using a JTAG debugger or the i.MX6 manufacturing today.
	bin.ivt	Firmware image file, USB serial download and bootable SD card binary format.
	elf	Firmware image file, ELF object file format. For loading using a JTAG debugger.
	srec	Firmware image file, SREC file format. For loading using a JTAG debugger.

UART Communication Variants

Processor Type	<uart> string	UART Interface	Pins (RxD, TxD)
i.MX6S	uart1_e13_f13	UART1	E13, F13
i.MX6DL	uart1_m3_m1	UART1	M3, M1
i.MX6D	uart2_r5_e24	UART2	R5,E24
i.MX6Q	uart4_l1_m2	UART4	L1,M2
	uart4_v6_w5	UART4	V6,W5
	uart5_m5_m4	UART5	M5,M4
i.MX6UL	ul_uart1_k16_k14	UART1	K16,K14
	ul_uart6_c16_c17	UART6	C16,C17

Select the appropriate VTOS firmware image based on the VTOS product licensed, the pins used for the UART connection on you board, and the desired mechanism for loading the image.

VTOS i.MX6 firmware file examples:

Filename	Description
vtos.imx6_ddr.uart1_e13_f13.bin	VTOS DDR firmware image file using UART1 for communication on pins E13 and F13. Use the i.MX6 manufacturing tool to load this image over the USB interface or use a JTAG debugger to load directly into the on-chip RAM of the i.MX6.
vtos.imx6_scan.uart4_l1_m2.bin.ivt	VTOS Scan firmware image file using UART4 for communication on pins L1 and M2. Program this file onto an SD card, following the procedure detailed in Loading the VTOS firmware from an SD card.
vtos.imx6_scan_plus.uart2_r5_e24.elf	VTOS ScanPlus firmware image file using UART2 for communication on pins R5 and E24. Load this image using a JTAG debugger.
vtos.imx6_program.uart5_m5_m4.srec	VTOS Program firmware image file using UART5 for communication on pins M5 and M4. Load this image using a JTAG debugger.

Using Serial Downloader to load VTOS i.MX6 Firmware Images

The ROM code for the i.MX6 includes a Serial Downloader which provides a mechanism for downloading a firmware image directly into the on-chip RAM over a USB connection.

Board requirements for using the Serial Downloader

The i.MX6 ROM code imposes the following requirements in order to use the Serial Downloader with a USB connection.

- USB OTG1 port is connected on your platform.
- BOOT_MODE[1:0] pins are set to 01b on your platform: enabling the Serial Downloader.

imx_usb_loader Utility

The VTOS Tools installer includes a utility called “imx_usb_loader.exe” that communicates with the i.MX6 ROM code to download and execute VTOS firmware images.

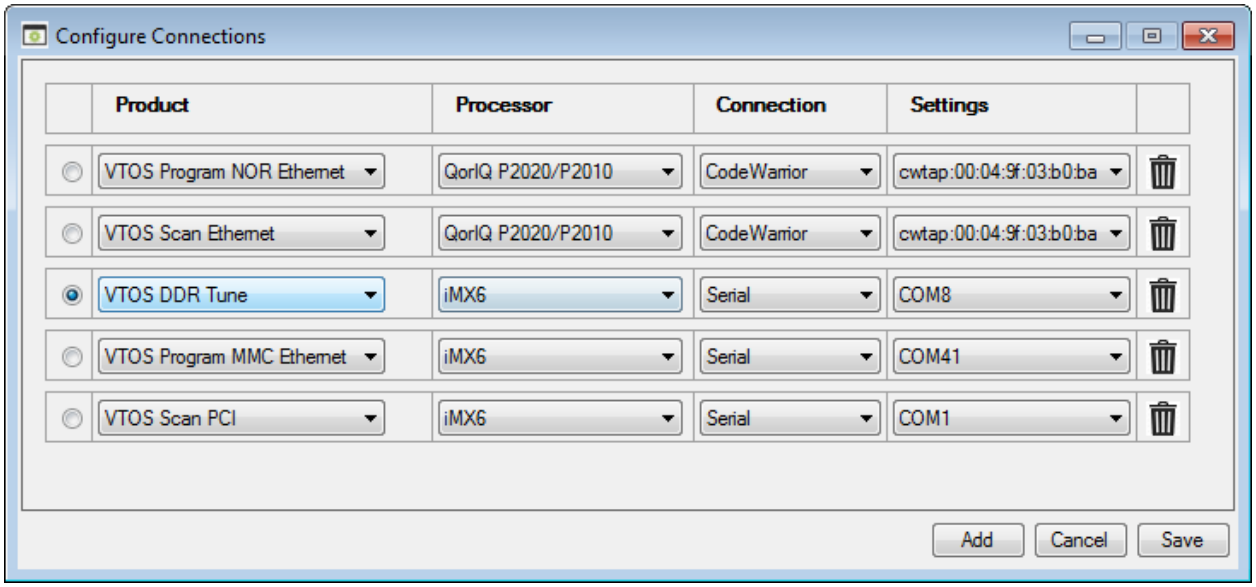
You execute the imx_usb_loader utility from a Windows command shell. To execute the utility, run the command as shown below. Note that you must include the quotes around the environment variable %KOZIO_VTOS_TOOLS%.

```
“%KOZIO_VTOS_TOOLS%”\imx_usb_loader <FILENAME>
```

The <FILENAME> parameter specifies the path and filename of the firmware file that will be loaded and executed. Always use a firmware file with the extension “bin.ivt” as these file types include the header and Image Vector Table as required by the i.MX6 ROM code Serial Downloader.

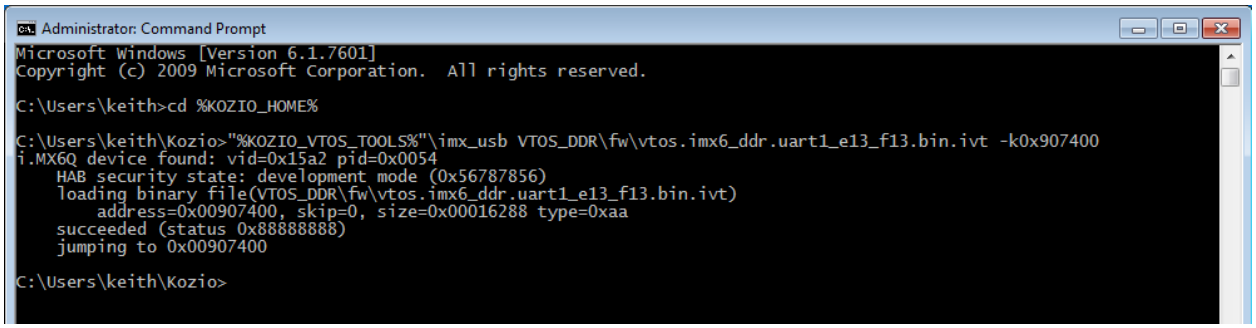
Loading and executing a VTOS Firmware image

1. Connect a USB cable between your Windows host and the USB OTG1 connection on your board.
2. Power up your board (ensure that that the BOOT_MODE[1:0] pins are set to 01b).
3. Connect a serial cable between the UART on your board and the Windows host.
4. Launch the VTOS Tools application (VTOS DDR, VTOS Scan, or VTOS Program)
5. Using the Connection Dialog, configure the serial port on the Windows host that is connected to your board. The example below shows the active connection is configured for VTOS DDR Tune, i.MX6 processor type, using COM8 on the Windows host.

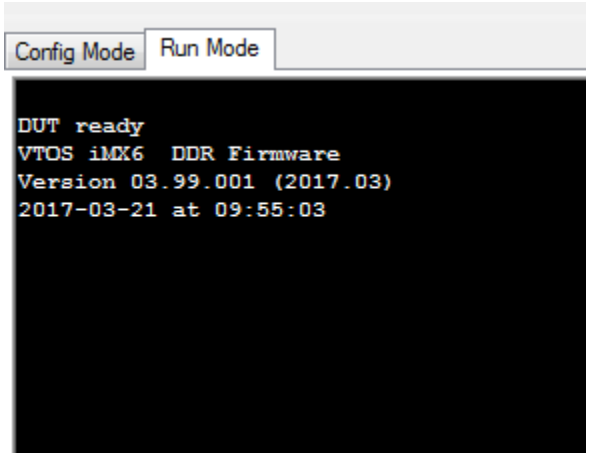


6. In the VTOS Tools application, click the Connect button. Note that cannot yet run commands or tests until you download the VTOS firmware to your board.
7. Open the Windows command shell, change into the %KOZIO_HOME% directory.
8. Use the utility `imx_usb_loader` to transfer VTOS firmware binary and start it executing. Note that you must select the VTOS DDR firmware binary that matches the UART pin configuration of your board. The example below downloads the VTOS DDR firmware image with communication configured for UART1 from the i.MX6 processor.

```
"%KOZIO_VTOS_TOOLS%"\imx_usb_loader  VTOS_DDR\fw\vtos.imx6_ddr.uart1_e13_f13.bin.ivt
```



9. The output from the `imx_usb_loader` utility confirms that transfer and execution of the VTOS DDR firmware binary was successful.
10. Check the VTOS DDR application window, if the COM port settings are correct, then the VTOS DDR console window will display “DUT Ready” as shown below.



Using a JTAG debugger to load ELF and SREC images

The instructions below assume that you are starting with a board that does not have any boot media programmed (such as NOR Flash, NAND flash, SD Card, eMMC, etc).

1. Connect the JTAG debugger to your board
2. Power up the JTAG debugger
3. Power up your board
4. Using the JTAG vendor supplied interface, issue a CPU reset to the i.MX6 core and allow the i.MX6 ROM code to execute.
5. Halt the i.MX6 processor.
    a. Confirm that the processor program counter register (labelled as PC, or R15) points to the I.MX6 ROM code area: 0x00000 – 0x17FFF.
    b. Confirm that the processor execution state is THUMB mode. THUMB mode operation is indicated by CPSR (Current Processor State Register) bit number 5. If bit 5 is set, the execution state is THUMB mode. If bit 5 is clear, the execution state is ARM mode.
6. Load the VTOS firmware ELF or SREC file using the JTAG vendor supplied interface. ELF and SREC files contain both the load address information, and the entry point location. Note that you must select the ELF/SREC file that matches the UART pin configuration of for your board.
    a. After loading the ELF/SREC file, confirm that the processor program counter has been changed to the THUMB mode entry point. This must be an address in the range 0x00907000 – 0x00937FFF.
7. Resume operation of the i.MX6 processor. The VTOS DDR application will display “DUT Ready” when the VTOS DDR firmware image boots.

Using a JTAG debugger to load binary images

For reference the load address and entry point information for VTOS firmware images is shown below.

VTOS Firmware Image Information: i.MX6	
Load Address	0x00907800
Entry Point (ARM mode)	0x00907800
Entry Point (THUMB mode)	Specified by the ELF image file

1. Connect the JTAG debugger to your board
2. Power up the JTAG debugger

3. Power up your board
4. Using the JTAG vendor supplied interface, issue a CPU reset to the i.MX6 core and allow the i.MX6 ROM code to execute.
5. Halt the i.MX6 processor.
    a. Confirm that the processor program counter register (labelled as PC, or R15) points to the i.MX6 ROM code area: 0x00000 – 0x17FFF.
6. Load the VTOS firmware binary file using the JTAG vendor supplied interface. The load address for the file is 0x00907800.
    a. After loading the file, modify the processor state to set ARM execution mode. Set the CPSR register to the value 0x00000193.
    b. Modify the program counter to point to the ARM execution mode entry point. Set the PC register to the value 0x00907800.
8. Resume operation of the i.MX6 processor. The VTOS DDR application will display "DUT Ready" when the VTOS DDR firmware image boots.

Loading the VTOS firmware from an SD card

If your board provides an SD card slot, you can program the VTOS firmware image onto an SD card and have the i.MX6 ROM code automatically load the VTOS firmware image into on-chip memory at power on.

The i.MX6 ROM code requires the following layout of data on an SD card to successfully boot a program image.

Description	Starting address
Reserved for MBR (optional)	0x00000000
Reserved for Secondary Image Table (optional)	0x00000200
Program Image `vtos.imx6_<product_type>.<uart>.bin.ivt`	0x00000400
Other partitions (optional)	0x00040000 (or greater)

The Linux dd utility is best suited for preparing an SD card and programing the VTOS DDR firmware image. Follow these steps to prepare a new SD card for booting the VTOS DDR firmware image. Note that the commands below require write permission to the SD card device. You made need to prepend each command with `sudo` for correct operation. Your SD card device will

Tip: Exercise caution when running these commands and ensure that you only direct commands to the SD card interface.

1. Use the `fdisk` command to determine the device assigned to your SD card interface. In the example below, the SD card is mapped to `/dev/sdc`. This is indicated both by the size of the disk (1967 MB).

```
$ fdisk –l

Disk /dev/sda: 85.9 GB, 85899280384 bytes
255 heads, 63 sectors/track, 10443 cylinders, total 167772032 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
```

```
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x0005fe56

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *        2048   165675007    82836480   83  Linux
/dev/sda2       165677054   167770111     1046529    5  Extended
/dev/sda5       165677056   167770111     1046528   82  Linux swap / Solaris

Disk /dev/sdc: 1967 MB, 1967128576 bytes
61 heads, 62 sectors/track, 1015 cylinders, total 3842048 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

Disk /dev/sdc doesn't contain a valid partition table
```

2. Delete the master boot record (MBR) and the secondary image table information by writing to the first two blocks (1024 bytes) of the SD card. Note that you must replace the text `<your-sd-card>` with the correct device name as given by the `fdisk` command.

```
dd if=/dev/zero of=/dev/<your-sd-card> bs=512 count=2
```

   Example output from the `dd` operation is shown below.

```
$ dd if=/dev/zero of=/dev/sdc bs=512 count=2
2+0 records in
2+0 records out
1024 bytes (1.0 kB) copied, 0.0159314 s, 64.3 kB/s
```

3. Copy the VTOS DDR firmware image file that matches your UART configuration to the SD card.

```
dd if=vtos.imx6_<product_type>.<uart>.bin.ivt of=/dev/<your-sd-card>
bs=512 seek=2
```

   Example output from the `dd` operation is shown below. This example programs the VTOS DDR firmware image for UART1, using the pins M3 and M1 for communication, to the SD card.

```
$ dd if=vtos.imx6_ddr.uart1_m3_m1.bin.ivt of=/dev/sdc bs=512 seek=2
158+1 records in
158+1 records out
81080 bytes (81 kB) copied, 0.223758 s, 362 kB/s
```

Using VTOS DDR

Board Specific settings for i.MX6

Automatic DDR Configuration

Set the **DDR Configuration Setting** to ***Automatic*** to have VTOS DDR automatically calculate the register settings for the MMDC (Multi Mode DDR Controller). Register settings are based on the memory part file selected and the configured speed of the DDR clock.

Tip: See the Empirical Software Calibration section for details on starting with a new timing based board file from scratch.

When configuring a new board, or a new memory part, you must configure the following items:

- From the **DDR Settings** node, set the **DDR Configuration Setting** to ***Automatic***.
- From the DDR Settings node, select the **Memory Type**. The i.MX6 supports LPDDR2 and DDR3 memory types.
- From the **Board Specific IMX6** node, configure the following:
    - DDR clock frequency
    - Number of chip selects used
    - Number of memory chips connected to each chip select
    - On-die termination value
    - Write additional latency
    - Read additional latency
    - Set **Fixed PHY Calibration** to 0 as an initial setting. This utilizes the hardware based PHY calibration. This setting can be used for quick initial memory qualification. However you should still run the software based PHY calibration as described in the Empirical Software Calibration section.
- From the **DDR Memory Part** node, right-click and select **Import** to configure the memory part timing values.
    - Lookup the speed bin table in the datasheet for your memory part. Adjust both the CL (CAS Latency) and CWL (CAS Write Latency) fields to match the configured DDR Clock Period.

Manual DDR Configuration

Set the **DDR Configuration Setting** to ***Manual*** to have VTOS DDR use explicit MMDC register settings. This is useful if you already have MMDC register settings from your own bootloader or software application and you want to run the VTOS DDR memory tests against these known settings.

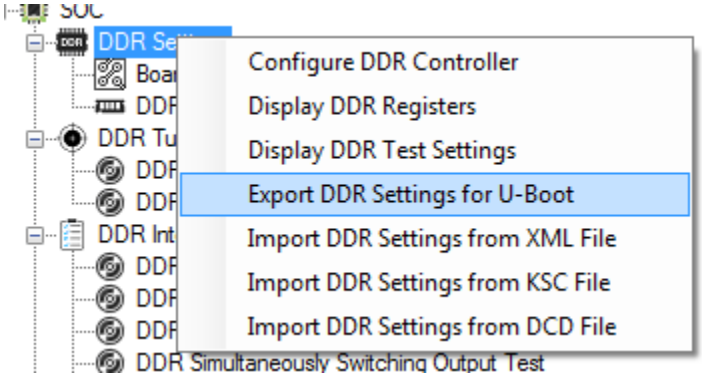
- From the **DDR Settings** node, set the **DDR Configuration Setting** to ***Manual***.
- When using ***Manual*** mode, the **Memory Type** field is not used.
- Fill out the **Board Specific IMX6** node as follows:
    - Set the DDR clock frequency for reference only.
    - The number of chip selects and DDR chips per chip select fields are not used in ***Manual*** mode.
    - Under the **Board Specific IMX6->DDR Manual** section, specify the exact 32-bit register value for all MMDC registers.
    - Under the **Board Specific IMX6->DDR Auto & Manual** section, specify the exact 32-bit register value for all MMDC PHY calibration registers.

Exporting DDR Settings

After configuring the DDR settings for your board, and successfully running the DDR memory tests, VTOS DDR supports exporting the DDR register settings.

To export the DDR settings using the Device Configuration Data (DCD) format, follow these steps:

- Select the **DDR Settings** node
- Right-click and select ***Export DDR Settings*** for U-Boot as shown below.



- The VTOS DDR run console displays the DCD formatted data in plain text. Copy and paste this data into the appropriate configuration file in your U-Boot source. Example output is shown below.

```
/* i.MX6 Dual/Quad DCD data */ cr cr
/* IOMUX configuration: DDR DQS signals */
DATA 4 0x020E05A8 0x000000B1
DATA 4 0x020E05B0 0x00000060
DATA 4 0x020E0524 0x000100B0
DATA 4 0x020E051C 0x000100B0

DATA 4 0x020E0518 0x000100B0
DATA 4 0x020E050C 0x000000B1
DATA 4 0x020E05B8 0x000100B0
DATA 4 0x020E05C0 0x000100B0

/* IOMUX configuration: DDR DQ signals */
DATA 4 0x020E05AC 0x0000B0B1
DATA 4 0x020E05B4 0x0001B0B0
DATA 4 0x020E0528 0x0001B0B0
DATA 4 0x020E0520 0x0001B0B0

DATA 4 0x020E0514 0x0001B0B0
DATA 4 0x020E0510 0x0000B0B1
DATA 4 0x020E05BC 0x0001B0B0
DATA 4 0x020E05C4 0x0001B0B0

/* IOMUX configuration: DDR control signals */
DATA 4 0x020E056C 0x0000B0B1
DATA 4 0x020E0578 0x0000B0B1
DATA 4 0x020E0588 0x0000B0B1
DATA 4 0x020E0594 0x0000B0B1

DATA 4 0x020E057C 0x0000B0B1
DATA 4 0x020E0590 0x0000B0B1
DATA 4 0x020E0598 0x0000B0B1
DATA 4 0x020E058C 0x0000B0B1

DATA 4 0x020E059C 0x0001B0B0
DATA 4 0x020E05A0 0x0001B0B0

/* IOMUX configuration: DDR group control */
DATA 4 0x020E0784 0x00000030
DATA 4 0x020E0788 0x00000000
DATA 4 0x020E0794 0x00000000
DATA 4 0x020E079C 0x00000000
```

```
DATA 4 0x020E07A0 0x00000000
DATA 4 0x020E07A4 0x00000000
DATA 4 0x020E07A8 0x00000000
DATA 4 0x020E0748 0x00000030

DATA 4 0x020E074C 0x00000030
DATA 4 0x020E0750 0x00020000
DATA 4 0x020E0758 0x00002000
DATA 4 0x020E0774 0x000C0000

DATA 4 0x020E078C 0x00000030
DATA 4 0x020E0798 0x00000000

/* MMDC: PHY 1 Read Delay Registers */
DATA 4 0x021B081C 0x00000000
DATA 4 0x021B0820 0x00000000
DATA 4 0x021B0824 0x00000000
DATA 4 0x021B0828 0x00000000

/* MMDC: PHY 2 Read Delay Registers */
DATA 4 0x021B481C 0x00000000
DATA 4 0x021B4820 0x00000000
DATA 4 0x021B4824 0x00000000
DATA 4 0x021B4828 0x00000000

DATA 4 0x021B0018 0x40011700

/* MMMDC: initialization sequence */
DATA 4 0x021B001C 0x00008000
DATA 4 0x021B000C 0x545978F4
DATA 4 0x021B0010 0xFD328F64
DATA 4 0x021B0014 0x01FF00DB
DATA 4 0x021B002C 0x0F9F26D2

DATA 4 0x021B0030 0x00591023
DATA 4 0x021B0008 0x09444040
DATA 4 0x021B0004 0x00020036
DATA 4 0x021B0040 0x00000027
DATA 4 0x021B0000 0x831A0000

/* MMDC: write mode registers: 2,3,1,0 */
DATA 4 0x021B001C 0x00088032
DATA 4 0x021B001C 0x00008033
DATA 4 0x021B001C 0x00448031
DATA 4 0x021B001C 0x09308030
/* MMDC: Enable ZQ calibration */
DATA 4 0x021B001C 0x04008040
DATA 4 0x021B0800 0xA1380003
DATA 4 0x021B4800 0xA1380003
DATA 4 0x021B0020 0x10168000
DATA 4 0x021B0818 0x00033337
DATA 4 0x021B4818 0x00033337

DATA 4 0x021B083C 0x4205017D
DATA 4 0x021B0840 0x0174017A
DATA 4 0x021B483C 0x015C0162
DATA 4 0x021B4840 0x0156014D
DATA 4 0x021B0848 0x40434742
DATA 4 0x021B4848 0x443F443C
DATA 4 0x021B0850 0x403F3537
DATA 4 0x021B4850 0x3538393F

DATA 4 0x021B080C 0x005A005F
DATA 4 0x021B0810 0x00590059

DATA 4 0x021B480C 0x0045004A
DATA 4 0x021B4810 0x004A0056

DATA 4 0x021B08B8 0x00000800
DATA 4 0x021B48B8 0x00000800

DATA 4 0x021B001C 0x00000000
DATA 4 0x021B0404 0x00011006
/* DDR configuration complete */
```

The output from VTOS DDR only includes the IOMUX configuration for the DDR signals, and the DDR memory controller initialization sequence. If your configuration file includes other sections, including boot options and clock setup, you must retain those sections.

An example DCD file from NXP is shown below. Note the marker that indicates where the exported data from VTOS DDR should be added.

```
/*
 * Copyright (C) 2012 Freescale Semiconductor, Inc.
 *
 * SPDX-License-Identifier:      GPL-2.0+
 *
 * Refer doc/README.imximage for more details about how-to configure
 * and create imximage boot image
 *
 * The syntax is taken as close as possible with the kwbimage
 */
/* image version */

IMAGE_VERSION 2

/*
 * Boot Device : one of
 * spi, sd (the board has no nand neither onenand)
 */
BOOT_FROM    sd

/*
 * Device Configuration Data (DCD)
 *
 * Each entry must have the format:
 * Addr-type           Address        Value
 *
 * where:
 *     Addr-type register length (1,2 or 4 bytes)
 *     Address        absolute address of the register
 *     value    value to be stored in the register
 */

/*
 * Copy and paste the exported output from VTOS DDR here
 */

/* set the default clock gate to save power */
DATA 4 0x020c4068 0x00C03F3F
DATA 4 0x020c406c 0x0030FC03
DATA 4 0x020c4070 0x0FFFC000
DATA 4 0x020c4074 0x3FF00000
DATA 4 0x020c4078 0xFFFFF300
DATA 4 0x020c407c 0x0F0000F3
DATA 4 0x020c4080 0x00000FFF

/* enable AXI cache for VDOA/VPU/IPU */
DATA 4 0x020e0010 0xF00000CF
/* set IPU AXI-id0 Qos=0xf(bypass) AXI-id1 Qos=0x7 */
DATA 4 0x020e0018 0x007F007F
DATA 4 0x020e001c 0x007F007F
/* end of file */
```

DDR3 PHY Calibration

The DDR PHY on the i.MX6 requires calibration. The calibration process fine tunes various parameters including:

- ZQ calibration
- Write leveling calibration
- DQS gating calibration
- Read data DQS delay calibration
- Write data DQS delay calibration

VTOS DDR supports two modes for calibrating the DDR3 PHY.

1. Empirical software calibration
2. Automatic hardware calibration (no longer recommended by NXP)

Empirical Software Calibration

VTOS DDR provides an empirical software calibration procedure for the DDR3 PHY on the i.MX6 processor family. This procedure calibrates the DDR3 PHY by executing the following steps:

1. Set the write leveling delay for one byte lane to a fixed value.
2. Calibrate of the DQS gating
3. Calibrate the read data DQS delay
4. Calibrate the write data DQS delay
5. Run a memory cell test
6. Run a memory bus noise test
7. Increment the write leveling delay value and repeat at step 2.

The procedure is repeated for all valid byte lanes for the design. For example, if your board uses a 64-bit wide data bus to the DDR memory, the procedure is repeated a total of 8 times.

Unlike hardware based calibration, the empirical software calibration procedure does not use DQ prime bits to obtain the leveling feedback. Instead, the results of the memory tests determine whether a specific write leveling setting is valid. At the successfully completion of the procedure, VTOS DDR reports the window of valid write leveling delay values for all byte lanes and sets the calibrated result to the middle of the window.

Running the Empirical Software Calibration

The empirical software calibration procedure can take significant time to run, as much as 20 minutes or more to complete. The procedure is typically run on new board designs, when changing to a new memory part, or changing the DDR interface speed.

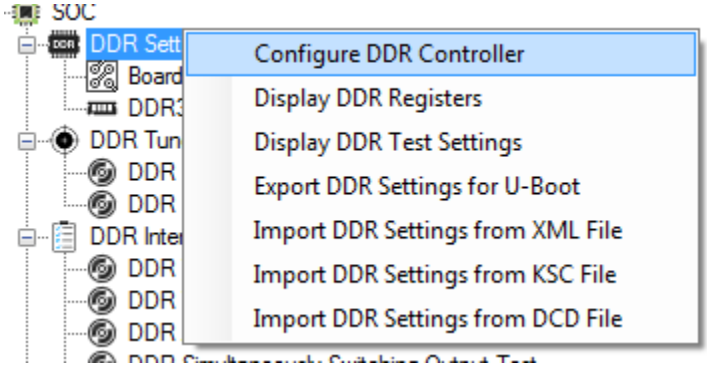
Follow this procedure when running the empirical software calibration procedure:

- In you board file, set the following parameters:
  - Set Write Additional Latency to the maximum value of 3.
  - Set Read Additional Latency to the maximum value of 7.
  - Enable Fixed PHY calibration
  - Set the initial value of all PHY calibration registers to zero.
    Example board file settings are shown below.

```
3 -> $ddr.memctrl.walat // Write Additional Latency
7 -> $ddr.memctrl.ralat // Read Additional Latency
TRUE -> $ddr.memctrl.fixed_calibration
0 -> $mmdc_mpwldectrl0
0 -> $mmdc_mpwldectrl1
```

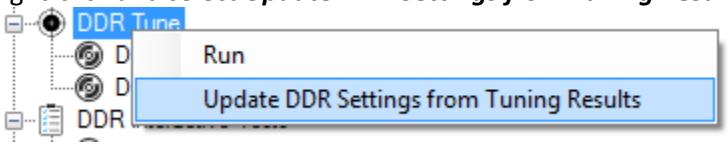
```
0 -> $mmdc_mpdgctrl0
0 -> $mmdc_mpdgctrl1
0 -> $mmdc_mprddlctl
0 -> $mmdc_mpwrdlctl
0 -> $mmdc2_mpwldectrl0
0 -> $mmdc2_mpwldectrl1
0 -> $mmdc2_mpdgctrl0
0 -> $mmdc2_mpdgctrl1
0 -> $mmdc2_mprddlctl
0 -> $mmdc2_mpwrdlctl
```

- Configure the DDR controller by right clicking on the **DDR Settings** node and selecting ***Configure DDR Controller*** as shown below.



Note that at this point, the DDR3 PHY is programmed with reset values and the DDR memory tests are not expected to pass.

- Run the empirical software calibration. Double click on the **DDR Tune Command: DDR3 PHY Calibration** node to start the PHY calibration. Note that for designs using 64-bit wide memory, this procedure can take 20 minutes or more to complete.
- At the successful completion of the empirical software calibration, update your VTOS DDR project file with the fixed PHY calibration settings displayed.
  - Select the **DDR Tune** node.
  - Right-click and select ***Update DDR Settings from Tuning Results***



  - Follow the prompts to update your project file with the tuning results
  - Save your VTOS DDR project file.

Example output from the empirical software calibration executed on a Sabre-Lite reference design is shown below.

```
Searching for write level window on byte lane 0
------------------------------------------------
MPWLDECTRL = 0x00000000
  Starting calibration of DQS gating, read delay, and write delay
    Warning: PHY0 reported errors during read DQS gating
    Warning: PHY1 reported errors during read DQS gating
    Warning: PHY0 reported errors during read delay line calibration
    Warning: PHY1 reported errors during read delay line calibration
    Warning: PHY0 reported errors during write delay line calibration
    Warning: PHY1 reported errors during write delay line calibration
  Calibration of DQS gating, read delay, and write delay completed

SDRAM: Full burst (64 bit) [00000000_10000000 - 00000000_100fffff]
Warning: only comparing data bus mask bits 0x00000000_000000ff
................Bad value at address 0x00000000_10000008
```

```
    expected 0x00000000_10000008, actual 0xffdfefff_ffffffff

//
// output truncated
//

Write level window (1/256 cycles) for byte 0
    lower  = 0x00000001
    upper  = 0x00000077
    middle = 0x0000003C
Searching for write level window on byte lane 1
------------------------------------------------
MPWLDECTRL = 0x0000003C
  Starting calibration of DQS gating, read delay, and write delay
  Calibration of DQS gating, read delay, and write delay completed

SDRAM: Full burst (64 bit) [00000000_10000000 - 00000000_100fffff]
Warning: only comparing data bus mask bits 0x00000000_0000ff00
............................................................
Passed!
SDRAM: Memory bus noise (burst) [00000000_10000000 - 00000000_100fffff]
Warning: only comparing data bus mask bits 0x00000000_0000ff00
............................................................
Passed!

//
// output truncated
//


Final Write Leveling results (1/256 cycles):
           lower   upper   middle  MPWLDECTRL setting
  Byte 0 : 0x0001  0x0077  0x003C     0x003C
  Byte 1 : 0x0000  0x006C  0x0036     0x0036
  Byte 2 : 0x0000  0x005E  0x002F     0x002F
  Byte 3 : 0x0000  0x0074  0x003A     0x003A
  Byte 4 : 0x0000  0x0067  0x0033     0x0033
  Byte 5 : 0x0000  0x0076  0x003B     0x003B
  Byte 6 : 0x0000  0x005F  0x002F     0x002F
  Byte 7 : 0x0000  0x0062  0x0031     0x0031

    Calibration completed.  Update your board file with the
    following settings.

        TRUE -> $ddr.memctrl.fixed_calibration
        0x0036003C -> $mmdc_mpwldectrl0
        0x003A002F -> $mmdc_mpwldectrl1
        0x4302031C -> $mmdc_mpdgctrl0
        0x027D026E -> $mmdc_mpdgctrl1
        0x40303436 -> $mmdc_mprddlctl
        0x34474A44 -> $mmdc_mpwrdlctl
        0x003B0033 -> $mmdc2_mpwldectrl0
        0x0031002F -> $mmdc2_mpwldectrl1
        0x03070321 -> $mmdc2_mpdgctrl0
        0x03010255 -> $mmdc2_mpdgctrl1
        0x37363047 -> $mmdc2_mprddlctl
        0x493A4D3D -> $mmdc2_mpwrdlctl
```

Tip: The empirical software calibration may generate warnings regarding PHY errors during the calibration process. These warnings are expected, especially during the first phases of the empirical software calibration. If you see the message “Final Write Leveling results”, then the empirical software calibration completed successfully.

Command Summary

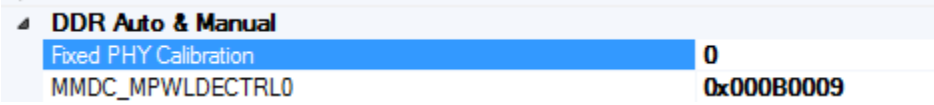
VTOS DDR provides the following commands related to empirical software calibration on the i.MX6 processor. These commands can be run manually using the VTOS DDR Advance Mode console.

Command	**ddr.tune.phy**	
Description	Searches for valid write leveling values and calibrates all required DDR3 parameters. The process is automatically repeated for all available byte lanes on the design. The current write leveling value on each byte lane is used as a seed value to begin the search. If the search fails for any byte lane, set the corresponding MPWLDECTRL register to zero and retry. Note that you must run the command **ddr.configure** before running this command.	
Usage	**ddr.tune.phy**	
Parameters	None	
Command	**tune.wrlvl**	
Description	Searches for valid write leveling values and calibrates all required DDR3 parameters on a single byte lane only. While this command runs, any data errors on other byte lanes are ignored. Note that you must run the command **ddr.configure** before running this command.	
Usage	**<byte_lane> tune.wrlvl**	
Parameters	None	
	<byte_lane>	The byte lane to run calibration against. Valid values are 0 – 7.
	<ddr_dqsx_length>	Average trace length of the DDR_DQSx signals, specified in thousandths of an inch.
Command	**ddr.tuning.results**	
Description	Display the results of the last **ddr.tune.phy** calibration performed.	
Usage	**ddr.tuning.results**	
Parameters	None	

Automatic Hardware Calibration

VTOS DDR executes the automatic hardware calibration for all the required PHY parameters. Note that using automatic hardware calibration is no longer recommended by NXP. However, the automatic hardware calibration runs very quickly compared to the empirical software calibration. It can be useful to initially run the automatic hardware calibration to find stable, but not necessarily optimal DDR PHY settings.

Automatic hardware calibration is enabled by setting the Board Specific IMX6->DDR Auto & Manual->Fixed PHY Calirbation field to 0.



When automatic hardware calibration is enabled, the command DDR configuration process displays the calibration results. The output from the hardware calibration executed on a Sabre-Lite reference design from NXP is shown below.

```
Configuring MMDC channel 1 for DDR3
Resetting MMDC channel 1
DDR total size = 1 GiB
Performing HW based calibration for DDR3
  Starting ZQ calibration
  ZQ calibration complete
  Starting write leveling calibration
  Write leveling complete
  Preparing DQS gating and delay line calibration
  Starting DQS gating calibration
  DQS gating calibration completed.
  Starting read delay line calibration.
  Read calibration completed.
  Starting write delay line calibration.
  Write calibration completed.
HW based calibration completed successfully

    Calibration completed.  Update your board file with the
    following settings.

        TRUE -> $ddr.memctrl.fixed_calibration
        0x001B0016 -> $mmdc_mpwldectrl0
        0x001D001B -> $mmdc_mpwldectrl1
        0x4313032C -> $mmdc_mpdgctrl0
        0x0310027E -> $mmdc_mpdgctrl1
        0x473B3E40 -> $mmdc_mprddlctl
        0x36444741 -> $mmdc_mpwrdlctl
        0x000C001D -> $mmdc2_mpwldectrl0
        0x00070018 -> $mmdc2_mpwldectrl1
        0x0307031D -> $mmdc2_mpdgctrl0
        0x03060252 -> $mmdc2_mpdgctrl1
        0x41383848 -> $mmdc2_mprddlctl
        0x45344C3E -> $mmdc2_mpwrdlctl
Enabling MMDC 1 controller for DDR3 operation
```

At this point you should run all the DDR tests to confirm the DDR settings and PHY calibration values are good. After confirming calibration values on several boards, you can use fixed calibration values by copying and pasting the calibration results back into your board file.

The example below shows the output from the DDR configuration process when fixed calibration is used.

```
Configuring MMDC channel 1 for DDR3
Resetting MMDC channel 1
DDR total size = 1 GiB
Enabling MMDC 1 controller for DDR3 operation
```

Manually Starting Hardware Calibration

You can execute the DDR PHY calibration manually with command **imx6.ddr3.calib.hw**. You may run this command even if your board file has enabled fixed calibration values. Repeatedly execute the DDR PHY calibration to verify that the calibration results are stable.

```
kozio> imx6.ddr3.calib.hw
Performing HW based calibration for DDR3
  Starting ZQ calibration
  ZQ calibration complete
  Starting write leveling calibration
  Write leveling complete
  Preparing DQS gating and delay line calibration
  Starting DQS gating calibration
  DQS gating calibration completed.
  Starting read delay line calibration.
  Read calibration completed.
  Starting write delay line calibration.
  Write calibration completed.
HW based calibration completed successfully

    To skip hardware driven calibration, copy and paste the following
    lines into your board file.

        TRUE -> $ddr.memctrl.fixed_calibration
        0x001B0016 -> $mmdc_mpwldectrl0
        0x001E001B -> $mmdc_mpwldectrl1
        0x4313032C -> $mmdc_mpdgctrl0
        0x03120300 -> $mmdc_mpdgctrl1
        0x473A3C40 -> $mmdc_mprddlctl
        0x36444741 -> $mmdc_mpwrdlctl
        0x000D001D -> $mmdc2_mpwldectrl0
        0x00070018 -> $mmdc2_mpwldectrl1
        0x0309031D -> $mmdc2_mpdgctrl0
        0x0301024D -> $mmdc2_mpdgctrl1
        0x3F363848 -> $mmdc2_mprddlctl
        0x45344B3D -> $mmdc2_mpwrdlctl
```

Determining optimal read and write latency

The i.MX6 DDR controller provides the settings RALAT (Read Additional Latency) and WALAT (Write Additional Latency) that add extra delay to read and write operations to the DDR memory. During initial board bring up, it is recommended that both RALAT and WALAT are set to the maximum possible values. However, this initial setting can impact the performance of the DDR memory.

To run the read and write latency calibration double click on the **DDR Tune Command: Read/Write Latency Calibration** node. This command sets with the RALAT/WALAT settings to the minimal value, incrementally tries new combinations for the RALAT/WALAT parameters, and determines a range of values that are functional on your board. At the completion of all tests, the command displays the middle range values found.

Example output from this command is shown below.

```
Existing settings: RALAT = 7   WALAT = 3

------------------------------------------------------------------------
Testing: RALAT = 0   WALAT = 0 (MDMISC = 0x40001600)


SDRAM: Full burst (64 bit) [00000000_10000000 - 00000000_100fffff]
................Bad value at address 0x00000000_10000000
    expected 0x00000000_10000000, actual 0xffffffff_effffffff
Bad value at address 0x00000000_10000008
    expected 0x00000000_10000008, actual 0xffffffff_effffff7
Bad value at address 0x00000000_10000010
    expected 0x00000000_10000010, actual 0xffffffff_effffffef

// output truncated

------------------------------------------------------------------------
Testing: RALAT = 3   WALAT = 0 (MDMISC = 0x400016C0)


SDRAM: Full burst (64 bit) [00000000_10000000 - 00000000_100fffff]
................................................................
Passed!

// output truncated

Testing: RALAT = 7   WALAT = 3 (MDMISC = 0x400317C0)


SDRAM: Full burst (64 bit) [00000000_10000000 - 00000000_100fffff]
................................................................
Passed!
              WALAT
          xxxx 3210
 RALAT    ---------
  0x00  | 0000 0000
  0x01  | 0000 0000
  0x02  | 0000 1111
  0x03  | 0000 1111
  0x04  | 0000 1111
  0x05  | 0000 1111
  0x06  | 0000 1111
  0x07  | 0000 1111
```

```
Optimal setting: RALAT = 4  WALAT = 1

If using a timing based file, copy and paste the following lines into your board file
    4  -> $ddr.memctrl.ralat
    1  -> $ddr.memctrl.walat

If using a register based file, copy and paste the following lines into your board
file
    0x00011700 -> $mmdc_mdmisc
```

Tip: At the completion of the window search for the RALAT and WALAT parameters, VTOS DDR applies the optimal results and re-configures the DDR controller.

Interpreting the latency tuning results

Run the command **latency.map** to display the results from the last execution of the command Read/Write Latency Calibration.

```
kozio> latency.map

             WALAT
          xxxx 3210
 RALAT    ---------
  0x00  | 0000 0000
  0x01  | 0000 0000
  0x02  | 0000 1111
  0x03  | 0000 1111
  0x04  | 0000 1111
  0x05  | 0000 1111
  0x06  | 0000 1111
  0x07  | 0000 1111

Optimal setting: RALAT = 4  WALAT = 1

If using a timing based file, copy and paste the following lines into your board file
    4  -> $ddr.memctrl.ralat
    1  -> $ddr.memctrl.walat

If using a register based file, copy and paste the following lines into your board
file
    0x00011700 -> $mmdc_mdmisc
```

Each row of the tuning map corresponds to a single setting of the RALAT parameter. At each RALAT setting, VTOS DDR tries all possible settings for the WALAT parameter. If the memory test passed for specific combination of RALAT/WALAT, the map displays a “1”. If the memory test failed, the map displays a “0”.

In the example above, VTOS DDR determined that at RALAT settings of 2 and higher, all settings for WALAT passed all tests.

Additional Contact Information

Kozio, Inc., +1 (303) 776-1356 x1, sales@kozio.com, www.kozio.comhttp://www.kozio.com/
http://www.kozio.com/

About Kozio, Inc.

Kozio, Inc. is a software technology company focused on providing superior embedded tools solving a variety of challenges during the design, production, and support of embedded devices. With its wide range of embedded tools, Kozio offers solutions to aerospace, automotive, consumer, industrial, military, medical, networking, and wireless markets. Kozio has been crafting embedded software since 2003 and has served the needs of thousands of engineers working for hundreds of companies, from the smallest to the largest.

Kozio's line of embedded tools include everything you need to configure, test, and tune DDR memory; identify interconnect faults and component placement problems on your printed circuit board; program on-board devices such as NAND Flash, NOR Flash, eMMC, FPGAs, and other programmable devices; and control, monitor, and calibrate power settings and usage. Kozio's tools are reusable across a family of SoC-based designs and reusable across teams.