StrataDNX™

HBM Configuration and Debugging Guide

Application Note
BCM88690
BCM88800
BCM88830

Table of Contents

Chapter 1: Overview

High-bandwidth memory (HBM) is a new type of memory that features an ultra-wide communication bus and low-power consumption per bit. The HBM standard was adopted by JEDEC (JESD235A).

The BCM88690 package includes two integrated HBM devices from leading memory industry vendors. The BCM88800 and BCM88830 packages include one HBM device.

This document describes the HBM configuration and provides information about how to debug the HBM devices. It also describes the relevant SOC properties, diagnostics commands, and APIs.

NOTE:

- The information required for configuring and debugging the HBM is available in the BCM88690 *Traffic Manager* Programming Guide (88690-PG2xx). This HBM application note document provides additional configuration information and examples and should be considered as supplemental to the 88690-PG2xx. Any information in the 88690-PG2xx document takes precedence over the information in this application note.
- This document is aligned with SDK 6.5.23 and later versions.

1.1 HBM Features

The following list summarizes the key features of the HBM devices used in the BCM88690 BCM88830, and BCM88800 packages:

- HBM Gen2 (HBM2) version, as defined by JEDEC (JESD235A).
- 4 GB of memory per HBM device.
  - The BCM88690 has two HBM2 instances for a total of 8 GB of memory
  - The BCM88800 has one HBM2 instance for a total of 4 GB of memory
  - The BCM88830 has one HBM2 instance for a total of 4 GB of memory
- Four stacked memories per HBM instance. Every stacked memory has two channels, which equals eight channels per device. Every stacked memory holds 1 GB.
- Each channel width is 128 bits of data and 16 bits of ECC.
- The channel default mode of operation is pseudo channel. In this mode, a 128-bit channel works as two pseudo channels of 64 bits each.
- The pseudo channel burst length is 4. Every read/write transaction is 4 × 64 bits = 256 bits.

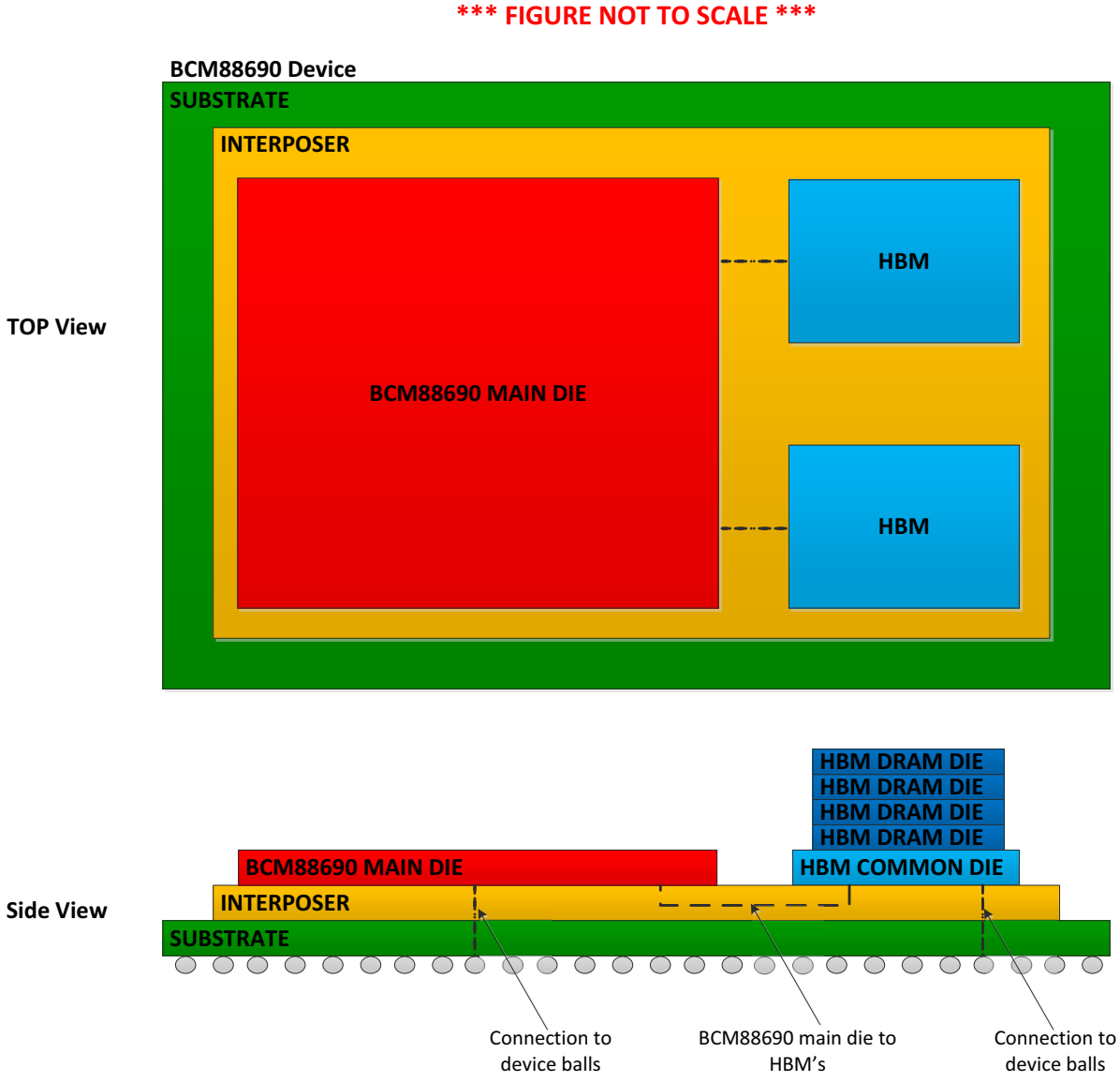
Each BCM88690 device has two HBM instances with a total bandwidth of about 4 Tb/s (2 Gb/s × 128 bits × 8 channels × 2 HBM) for read or write operations. The total bandwidth for the BCM88800 is 2 Tb/s for read or write operations.

The BCM88830 device has one HBM instance with a total bandwidth of about 2.45 Tb/s (2.4 Gb/s × 128 bits × 8 channels × 1 HBM) for read or write operations.

NOTE: Because packets are written to and read from the HBM, each packet consumes the HBM bandwidth twice. For packet rate calculations, divide the total maximum HBM bandwidth by 2.

The following figure shows an HBM device connected to the main die of the BCM88690 and to the device balls through the interposer. The device ball connection includes HBM power and a direct access (DA) test port for factory tests.

Figure 1: BCM88690 and HBM Instances



1.2 HBM Memory Power-Up and Initialization Sequence

Before the first memory access to HBM memory, customer software should perform the following sequence:

- Implemented by the customer:
  a. Verification that temperature is within the operational range
  b. Power-up of HBM-related rails
- Performed by the driver:
- PHY PLL initialization and lock status verification
  a. HBM PHY initialization
  b. HBM controller initialization and HBM initialization
  c. HBM tuning (Shmoo) for the BCM88690 and BCM88800 devices (not required for the BCM88830 device)
  d. DRAM built-in self test (BIST)
  e. HBM write or read to memory

If the HBM is powered down after a full power-up and initialization, and the BCM88690 main die is not powered down, it is assumed that the HBM PHY PLL is running and configured, and the HBM controller and HBM PHY have already initialized. In this case, only partial configuration is required using the following sequence:

1. Verify that the temperature is within the operational range (implemented by the customer).
2. Power up HBM-related rails (implemented by the customer).
3. Use an API to run a partial initialization.

1.2.1 Check Temperature before HBM Powers-Up

Before powering up an HBM device, verify that the HBM die is within its operational temperature range. For more information, refer to the Thermal Aspects section in the *Hardware Design Guidelines for StrataDNX 16-nm Devices* (DNX16-AN1xx) or the *Hardware Design Guidelines for StrataDNX 7-nm Devices* (DNX07-DG1xx).

1.2.2 HBM Power-Up

For the HBM power-up sequence, refer to the Power-Up and Reset Sequence section in the device data sheet.

1.2.3 Configure PHY PLL and Check PLL Lock

During the first initialization sequence, the HBM PHY PLL is initialized.

Every PHY includes a PLL that uses the HBM REFCLK and generates the HBM memory operating frequency. Initialize the PLL and check the PLL lock.

1.2.4 HBM PHY Initialization

In this step, the main die HBM PHY is initialized.

1.2.5 HBM Controller Initialization and HBM Initialization

In this step, the main die HBM controller is initialized, and the HBM mode register (MR) initialization occurs.

1.2.6 HBM Tuning and BIST (BCM88690 and BCM88800 Only)

NOTE: This section is applicable only to the BCM88690 and BCM88800 devices. The BCM88830 HBM PHYs are implemented differently and do not require tuning, restoring from OTP, or restoring from file mechanisms as described in this section.

The BCM88690 and BCM88800 are One-Time Programmable (OTP) devices and are programmed during manufacturing. Tuning is not required, and the devices can be tuned by restoring the OTP tune settings. This is the recommended approach for production. The other options of *tune* or *restore from file* described in the following table are still available.

As part of the power-up initialization, HBM tuning (Shmoo) must take place if HBM is used. The following table describes the tuning options.

Mode	System Conditions	Boot Time	Note
Tune	Nominal conditions	Slow	During the tune, the device should operate at nominal temperature and voltage conditions. In other words, it should not operate at the extremes (low or high voltage or temperature).
Restore from OTP	Per the device data sheet requirements	Fast	The tune is performed during device manufacturing and burned into device internal memory (OTP). When a restore is performed, OTP parameters are used.
Restore from file[a]	Per the device data sheet requirements	Fast	The tune is performed at nominal conditions, and the results are saved to a file for next boot. Saved parameters should be managed per device.

a. Perform the tune on each device assembled on the board. To restore the tune, each device assembled on the board should have a set of tune results specific to the assembled device.

For all tuning modes, when the system power-up and tune are complete, a DRAM BIST operation performs (by default) a basic check of HBM interfaces and memory. If DRAM BIST fails, a debug message is generated. To debug such a case, boot with `dram_phy_tune_mode_on_init` set to `SKIP_TUNE`, power up the system, and continue the debug by running a manual tune and different BIST options.

The SOC property `dram_phy_tune_mode_on_init` defines the tune mode of operation during power-up. For details about SOC property options and how to generate a tune file for restoring, see Section 2.2, HBM Tune (BCM88690 and BCM88800 Only).

1.2.6.1 Tune Mode

Before using tune mode, the system must be operating under nominal power and temperature conditions to enable the full span of operation per the device specification. Set `dram_phy_tune_mode_on_init` to `RUN_TUNE`.

1.2.6.2 Restore from OTP Mode

To enable OTP tune, set `dram_phy_tune_mode_on_init` to one of the following values:

- `RESTORE_TUNE_PARAMS_FROM_SOC_PROPS_OR_OTP_OR_TUNE`
- `RESTORE_TUNE_PARAMETERS_FROM_OTP`

For details, see Section 2.2, HBM Tune (BCM88690 and BCM88800 Only). This option does not have any prerequisites because the tuning and parameters are performed during device manufacturing.

1.2.6.3 Restore from File Mode

The prerequisite for this mode is to generate tune parameters during manufacturing (or when required) by using the following procedure:

1. Set `dram_phy_tune_mode_on_init` to `RUN_TUNE`.
2. Power up the device at nominal power and temperature conditions.
3. Tune the device to nominal conditions. (If more than one device is on the board, tune all devices.)
4. Because BIST is run after power-up tuning, make sure no DRAM BIST errors exist.
5. Save the tune results to a file.

Use the following steps to restore the tune parameters on each device during system boot at any condition (per specification):

1. Set `dram_phy_tune_mode_on_init` to `RESTORE_TUNE_PARAMETERS_FROM_SOC_PROPERTIES`.
   For details, see Section 2.2, HBM Tune (BCM88690 and BCM88800 Only).
2. Set the tune parameters for the specific device (based on the saved file or other method used).
3. Power up the system.

When restoring the tuning parameters, it might be necessary to repeat the steps to regenerate the tune parameter file in any of the following cases:

- When reassembling or replacing the device.
- When an SDK update requires retuning (the tune algorithm changed).
- When changing the HBM frequency operation (for debug or if required).
- When changing the voltage of one of the following supplies:
  - HBM[1:0]*
  - DRAM_PHY*
  - VDDC

1.2.7 Write to or Read from HBM Memory

When packet queues are built and packets need to move to deep buffering in the HBM memory, packets are aggregated into bundles of packets. Bundles from different cores are the data sources to the HBM controller. The HBM controller uses all current available HBM channels to spread each data source into the HBM memory at maximum bandwidth. For read operations, the reverse sequence is performed.

1.3 Temperature Monitor Thread and Temperature Thresholds

HBM temperature monitoring is performed in the background by periodically sampling the HBM die temperature and checking the high HBM die temperature group thresholds and low HBM die temperature group thresholds. Based on the threshold values and the HBM temperature, the following actions can be taken:

- Stop traffic to HBM
- Restore traffic to HBM
- Power down HBM

The following table describes the two threshold groups.

No.	Threshold Group	Usage	Affected Devices
1	High HBM die temperature	Protect HBM operation at the maximum (high) temperature operation range as defined in the data sheet.	All
2	Low HBM die temperature	Protect HBM operation at the minimum (low) operation range, below or close to 0°C, as defined in data sheet.	Industrial grade

For each threshold group, three thresholds are defined.

No.	Threshold Type	Usage	Note
1	Stop	Prevent packet flow through the HBM when the temperature is not stable	The threshold value considers the following factors: ■ Temperature change rate ■ Time until the HBM is empty ■ Temperature reading accuracy
2	Restore	Reenable traffic to the HBM because the temperature moved to a safe operation range	This threshold should not be the same as the Stop threshold to allow some hysteresis between the two thresholds.
3	Power-down	Prevent immediate damage to the HBM die	—

NOTE: The SDK includes default values for each threshold. Customers should review and, if required, modify thresholds per platform to keep HBM operation in compliance with the device data sheet (based on simulation and system thermal behavior). For more information, see Chapter 2, HBM Configuration – SOC Properties.

1.4 Industrial Devices and Low-Temperature Power-Up

DNX industrial devices support powering up in temperatures below 0°C; however, the HBM minimum power-up and operating temperature is 0°C. If the device boot process occurs when the ambient temperature is below 0°C, HBM power-up and initialization should be performed at a later stage. For this type of case, use a special SOC property to skip HBM activation during power-up and wait until the device temperature is above 0°C. Based on simulations, wait until the estimated HBM temperature is above 0°C with safe margins, power up the HBM, and call the HBM init API.

For information about the SOC property that supports power-up with the HBM powered down, see Section 2.6, HBM Freeze Boot.

For HBM initialization information, see Section 3.2, API: Partial Initialization after HBM Power-Down.

For additional information regarding industrial devices and low-temperature power-up, refer to the Thermal Aspects chapter in the following documents:

- BCM88690 and BCM88800: *Hardware Design Guidelines for StrataDNX 16-nm Devices* (DNX16-AN1xx)
- BCM88830: *Hardware Design Guidelines for StrataDNX 7-nm Devices* (DNX07-DG1xx)

1.5 CATTRIP Interrupt Assertion

Avoid the CATTRIP point in all cases. When the CATTRIP point is reached, the CATTRIP interrupt is asserted, and the user call-back (CB) function is activated to immediately shut down the HBM power supplies. For more information, see Section 3.5, Call-Back Function: HBM Power-Down. In parallel, the HBM_THERM_ALARM device pin is also asserted.

Restore power to the HBM power supplies only when the temperature has returned to normal operational conditions, and after reinitialization. For more information, see Section 3.2, API: Partial Initialization after HBM Power-Down.

Chapter 2: HBM Configuration – SOC Properties

This chapter describes SOC properties to define for HBM configuration as well as SOC properties for special HBM debug capabilities.

2.1 HBM Usage Configuration

The `ext_ram_enabled_bitmap` SOC property defines the HBM usage. The SDK default value may be disabled (set to 0). The bold rows in the following table describe how to enable the HBM on each device.

Table 1: HBM ext_ram_enabled_bitmap SOC Settings

Device	SOC Setting	Enabled HBMs	Comment
BCM88690	**`ext_ram_enabled_bitmap.BCM8869X=0x3`**	**2**	**Enable**
	`ext_ram_enabled_bitmap.BCM8869X=0x2`	1	Debug only, HBM1
	`ext_ram_enabled_bitmap.BCM8869X=0x1`	1	Debug only, HBM0
	`ext_ram_enabled_bitmap.BCM8869X=0x0`	0	Disable
BCM88800	**`ext_ram_enabled_bitmap.BCM8880X=0x1`**	**1**	**Enable**
	`ext_ram_enabled_bitmap.BCM8880X=0x0`	0	Disable
BCM88830	**`ext_ram_enabled_bitmap.BCM8830X=0x1`**	**1**	**Enable**
	`ext_ram_enabled_bitmap.BCM8830X=0x0`	0	Disable

2.2 HBM Tune (BCM88690 and BCM88800 Only)

The SOC property in this section defines how an HBM tune is performed. For more information, see Section 1.2.6, HBM Tuning and BIST (BCM88690 and BCM88800 Only).

NOTE: This section is applicable only to the BCM88690 and BCM88800 devices. The BCM88830 HBM PHYs are implemented differently and do not require the tuning, restoring from OTP, or restoring from file mechanisms described in this section.

The following table is a summary of available tune property sets (`dram_phy_tune_mode_on_init`).

Property	Tune Mode
`RUN_TUNE`	Run HBM tune on initialization.
`RESTORE_TUNE_PARAMETERS_FROM_SOC_PROPERTIES`	Restore HBM tune parameters from SOC.
`RESTORE_TUNE_PARAMETERS_FROM_OTP` `RESTORE_TUNE_PARAMS_FROM_SOC_PROPS_OR_OTP_OR_TUNE`	**Restore HBM tune parameters from OTP.**
`SKIP_TUNE`	Skip HBM tune on initialization.

Run HBM Tune on Initialization

Example: `dram_phy_tune_mode_on_init.BCM8869X=RUN_TUNE`

This property runs the tune after each device power-up. Running the tuning process determines and sets the optimal working point for current PVT conditions. To be able to work in the full temperature span, perform the tune under nominal voltage and temperature conditions.

Restore HBM Tune Parameters from SOC

Example: `dram_phy_tune_mode_on_init.BCM8869X=RESTORE_TUNE_PARAMETERS_FROM_SOC_PROPERTIES`

When this property is set, the driver assumes the HBM tune parameters (`hbm_tune_*`) are already loaded and a tune has been performed, and it skips the relevant tuning steps. Using this method is faster than using `RUN_TUNE`, but it requires management of the SOC properties for each BCM8869X assembled on the board. Additionally, the SOC tune properties must be loaded per device during the initialization phase.

Restore HBM Tune Parameters from OTP

Option 1 example:

```
dram_phy_tune_mode_on_init.BCM8869X= RESTORE_TUNE_PARAMETERS_FROM_OTP
```

Option 2 example:

```
dram_phy_tune_mode_on_init.BCM8869X= RESTORE_TUNE_PARAMS_FROM_SOC_PROPS_OR_OTP_OR_TUNE
```

When the option 2 property is set, the driver performs the following steps:

1. The driver attempts to restore tune values from SOC properties.
2. If Step 1 fails, the driver attempts to restore tune values from OTP.
3. If Step 2 fails, use `RUN_TUNE`.

Devices that have passed the product release authorization (PRA) stage are assumed to be OTP-tune programmed. To check if the device is OTP programmed, set the SOC property to `RESTORE_TUNE_PARAMETERS_FROM_OTP`. If the device fails to tune during power-up, it is not programmed. If it passed, use `RESTORE_TUNE_PARAMS_FROM_SOC_PROPS_OR_OTP_OR_TUNE` to support cases with no OTP.

Skip HBM Tune on Initialization

Example: `dram_phy_tune_mode_on_init.BCM8869X=SKIP_TUNE`

When tuning on initialization is skipped, the tune process is not performed. When this occurs, any transaction to the DRAM is not optimal and may have errors (possibly fatal errors). A full tune is required prior to accessing the DRAM. Do not use `SKIP_TUNE` unless it is specifically required. Do not disable the HBM in this case.

2.2.1 Generating the HBM Tune File

The following steps show how to generate HBM tune SOC properties and save them to a file. The generated file is specific to the assembled device on which the tune was run and cannot be shared with other devices. Every device should have its own tune file when assembled on a board.

1. Set the device to nominal conditions.
2. Tune all channels on the HBM. For information, see Section 4.2.2, Diagnostic Tuning.
3. Save the tune result to a file named (for example) `./BCM8869X_dram_tune.soc`.

   The following BCM shell command saves the HBM tune parameters from the last tune to a file.

   ```
   BCM> config save filename=./BCM8869X_dram_tune.soc pattern=hbm_tune
   ```
4. Add the text `config add` to every line in the `./BCM8869X_dram_tune.soc` file created Step 3.

   The following example uses the Linux shell `sed` command to add the text:

   ```
   Shell> sed -i -e "s/^/config add /g" ./BCM8869X_dram_tune.soc
   ```
5. Sort the file. This step is not required for proper operation but makes the file more readable.

   The following example uses the Linux shell `sort` command:

   ```
   Shell> sort -o ./BCM8869X_dram_tune.soc ./BCM8869X_dram_tune.soc
   ```

The following output is an example of the `BCM8869X_dram_tune.soc` file contents. It is not the complete output and shows only a few lines as an example of the output format:

```
config add hbm_tune_AWORD_DAC_CONFIG__DAC_DATA_0_0.0=0x00000004
config add hbm_tune_AWORD_DAC_CONFIG__DAC_DATA_0_1.0=0x00000004
config add hbm_tune_AWORD_DAC_CONFIG__DAC_DATA_0_2.0=0x00000004
config add hbm_tune_AWORD_DAC_CONFIG__DAC_DATA_0_3.0=0x00000004
config add hbm_tune_AWORD_DAC_CONFIG__DAC_DATA_0_4.0=0x00000004
config add hbm_tune_AWORD_DAC_CONFIG__DAC_DATA_0_5.0=0x00000004
config add hbm_tune_AWORD_DAC_CONFIG__DAC_DATA_0_6.0=0x00000004
config add hbm_tune_AWORD_DAC_CONFIG__DAC_DATA_0_7.0=0x00000004
config add hbm_tune_AWORD_DAC_CONFIG__DAC_DATA_1_0.0=0x00000004
config add hbm_tune_AWORD_DAC_CONFIG__DAC_DATA_1_1.0=0x00000004
config add hbm_tune_AWORD_DAC_CONFIG__DAC_DATA_1_2.0=0x00000004
config add hbm_tune_AWORD_DAC_CONFIG__DAC_DATA_1_3.0=0x00000004
config add hbm_tune_AWORD_DAC_CONFIG__DAC_DATA_1_4.0=0x00000004
config add hbm_tune_AWORD_DAC_CONFIG__DAC_DATA_1_5.0=0x00000004
config add hbm_tune_AWORD_DAC_CONFIG__DAC_DATA_1_6.0=0x00000004
config add hbm_tune_AWORD_DAC_CONFIG__DAC_DATA_1_7.0=0x00000004
config add hbm_tune_AWORD_IO_CONFIG__DIFF_PORT_CK_DRIVE_STRENGTH_0_0.0=0x00000004
config add hbm_tune_AWORD_IO_CONFIG__DIFF_PORT_CK_DRIVE_STRENGTH_0_1.0=0x00000004
config add hbm_tune_AWORD_IO_CONFIG__DIFF_PORT_CK_DRIVE_STRENGTH_0_2.0=0x00000004
config add hbm_tune_AWORD_IO_CONFIG__DIFF_PORT_CK_DRIVE_STRENGTH_0_3.0=0x00000004
config add hbm_tune_AWORD_IO_CONFIG__DIFF_PORT_CK_DRIVE_STRENGTH_0_4.0=0x00000004
config add hbm_tune_AWORD_IO_CONFIG__DIFF_PORT_CK_DRIVE_STRENGTH_0_5.0=0x00000004
config add hbm_tune_AWORD_IO_CONFIG__DIFF_PORT_CK_DRIVE_STRENGTH_0_6.0=0x00000004
config add hbm_tune_AWORD_IO_CONFIG__DIFF_PORT_CK_DRIVE_STRENGTH_0_7.0=0x00000004
config add hbm_tune_AWORD_IO_CONFIG__DIFF_PORT_CK_DRIVE_STRENGTH_1_0.0=0x00000004
config add hbm_tune_AWORD_IO_CONFIG__DIFF_PORT_CK_DRIVE_STRENGTH_1_1.0=0x00000004
config add hbm_tune_AWORD_IO_CONFIG__DIFF_PORT_CK_DRIVE_STRENGTH_1_2.0=0x00000004
config add hbm_tune_AWORD_IO_CONFIG__DIFF_PORT_CK_DRIVE_STRENGTH_1_3.0=0x00000004
config add hbm_tune_AWORD_IO_CONFIG__DIFF_PORT_CK_DRIVE_STRENGTH_1_4.0=0x00000004
config add hbm_tune_AWORD_IO_CONFIG__DIFF_PORT_CK_DRIVE_STRENGTH_1_5.0=0x00000004
config add hbm_tune_AWORD_IO_CONFIG__DIFF_PORT_CK_DRIVE_STRENGTH_1_6.0=0x00000004
config add hbm_tune_AWORD_IO_CONFIG__DIFF_PORT_CK_DRIVE_STRENGTH_1_7.0=0x00000004
```

2.2.2 Load HBM Tune SOC Properties from the File

The following lines are embedded in the SDK `dnx.soc` file. These lines enable the loading of the tune file for the BCM8869X (Jericho 2) device:

```
if $?BCM_JR2_DEV \
       rcload ./BCM8869X_dram_tune.soc'
```

In this example, the `BCM8869X_dram_tune.soc` file is specific to one device. If two or more BCM8869X devices exist on the same board, the file should include all devices tune parameters, or different files should be loaded, one per device.

2.3 HBM External Frequency (For Debug Only)

NOTE: For normal operation, do not use this SOC property.

The `ext_ram_freq` SOC property sets the HBM interface speed to run at the specified frequency. Do not use this property unless it is specifically required for debugging by Broadcom support.

BCM88690: `ext_ram_freq.BCM8869X=1000`

BCM88800: `ext_ram_freq.BCM8880X=1000`

BCM88830: `ext_ram_freq.BCM8883X=1200`

The BCM88690 example shows the frequency set to 1000, which means the interface speed operates at 1000 MHz, and the bit rate per DQ bit is 2000 Mb/s.

2.4 HBM Temperature Monitor Enable

To enable the default HBM temperature management that the SDK performs, set the `dram_temperature_monitor_enable` SOC property to *enabled*. To implement a different mechanism to handle HBM temperature management, set the SOC to *disabled* and ensure the mechanism performs the following:

- Monitors the HBM junction temperature and triggers APIs based on the condition.
- Determines when to disable or enable data to the DRAM by calling APIs.
- Determines when to power up or down the HBM power by calling APIs.

When enabled, the SDK monitors and triggers the activities listed previously.

For more information about temperature monitoring, refer to the *Hardware Design Guidelines for StrataDNX 16-nm Devices* (DNX16-AN1xx) or *Hardware Design Guidelines for StrataDNX 7-nm Devices* document (DNX07-DG1xx). For more information about related APIs, see Chapter 3, HBM Operation – API and Call-Back Functions.

Temperature Monitor Enable

Example: `dram_temperature_monitor_enable.BCM8869X=1`

Temperature Monitor Disable

Example: `dram_temperature_monitor_enable.BCM8869X=0`

2.5 HBM Temperature Thresholds

High and low temperature thresholds are per-device settings. Update the threshold values based on simulation results and the planned margins.

High Temperature Thresholds

All devices use the following thresholds:

- `dram_temperature_threshold_stop_traffic`
- `dram_temperature_threshold_restore_traffic`
- `dram_temperature_threshold_power_down`

Low Temperature Thresholds

Industrial devices use high and low temperature thresholds.

The following list shows additional industrial devices thresholds:

- `dram_low_temperature_threshold_stop_traffic`
- `dram_low_temperature_threshold_restore_traffic`
- `dram_low_temperature_threshold_power_down`

HBM Stop Traffic Threshold

Example: `dram_temperature_threshold_stop_traffic.BCM8869X=90`

Traffic stops flowing to the HBM when the HBM junction temperature reaches 90°C.

HBM Restore Traffic Threshold

Example: `dram_temperature_threshold_restore_traffic.BCM8869X=85`

Traffic resumes flowing to the HBM when the HBM junction temperature reaches 85°C.

HBM Power-Down Temperature

Example: `dram_temperature_threshold_power_down.BCM8869X=95`

The power-down process is initiated when the HBM junction temperature reaches 95°C to prevent immediate damage.

HBM Low Stop Traffic Threshold

Example: `dram_low_temperature_threshold_stop_traffic.BCM8869X=5`

Traffic stops flowing to the HBM when the HBM junction temperature reaches 5°C.

HBM Low Restore Traffic Threshold

Example: `dram_low_temperature_threshold_restore_traffic.BCM8869X=10`

Traffic resumes flowing to the HBM when the HBM junction temperature reaches 10°C.

HBM Low Power-Down Temperature

Example: `dram_low_temperature_threshold_power_down.BCM8869X=0`

The power-down process is initiated when the HBM junction temperature reaches 0°C to prevent immediate damage.

2.6 HBM Freeze Boot

Use the following SOC property when loading the SDK at a temperature below 0°C without enabling the HBM:

```
ext_ram_freeze_boot=1
```

When the temperature is above 0°C, use the following SOC property (default, no user intervention required):

```
ext_ram_freeze_boot=0
```

For additional information, see Section 1.4, Industrial Devices and Low-Temperature Power-Up.

Chapter 3: HBM Operation – API and Call-Back Functions

You can use the API when implementing custom temperature monitors.

3.1 API: HBM Junction Temperature

This API returns the HBM junction temperature.

NOTE: Use this API if the HBM temperature monitor is disabled.

API

```
bcm_switch_thermo_sensor_read(int unit, bcm_switch_thermo_sensor_type_t sensor_type,
int interface_id, bcm_switch_thermo_sensor_t* sensor_data)
```

In the following CINT run and BCM shell diagnostics examples, the HBM thermal sensor read gives the same junction temperature because the tests are performed at almost the same time (30°C for HBM0 and 32°C for HBM1).

CINT Example

```
BCM.0> cint
Entering C Interpreter. Type 'exit;' to quit.
cint> bcm_switch_thermo_sensor_type_t sensor_type;
cint> sensor_type = bcmSwitchThermoSensorDram;
cint> bcm_switch_thermo_sensor_t sensor_data;
cint> print bcm_switch_thermo_sensor_read(0, sensor_type, 0,&sensor_data);
int $$ = 0 (0x0)
cint> print sensor_data;
bcm_switch_thermo_sensor_t sensor_data = {
int thermo_sensor_min = 0 (0x0)
int thermo_sensor_max = 0 (0x0)
int thermo_sensor_current = 30 (0x1e)
}
cint>
cint> print bcm_switch_thermo_sensor_read(0, sensor_type, 1,&sensor_data);
int $$ = 0 (0x0)
cint> print sensor_data;
bcm_switch_thermo_sensor_t sensor_data = {
int thermo_sensor_min = 0 (0x0)
int thermo_sensor_max = 0 (0x0)
int thermo_sensor_current = 32 (0x20)
}
cint>
```

BCM Shell Example

```
BCM.0> dnx dram debug temp
========================
| Dram Temperature |
========================
| Dram | Temperature |
========================
| 0x0 | 30 C |
| 0x1 | 32 C |
========================
```

3.2 API: Partial Initialization after HBM Power-Down

This API supports a partial initialization after assuming the first initialization to the memory subsystem has already been performed. For example, the BCM88690 main die and HBM power and initialization are performed, but due to an unexpected thermal event, an HBM power-down occurs. When the temperature has returned to normal operating conditions and is stable, perform an HBM power-up. After the power-up, the API performs a partial initialization before the HBM can be accessed again. It is a *partial* initialization because it is a subset of the first main die and HBM initialization that occurred in the first device power-up.

API

```
int bcm_switch_dram_init(int unit, uint32 flags)
```

3.3 API: HBM Power-Down

This API prepares the driver and handles updates before powering down the HBM power. Call this API before the actual power-down is performed.

NOTE: Use this API if the HBM temperature monitor is disabled.

API

```
int bcm_switch_dram_power_down(int unit, uint32 flags)
```

3.4 API: Enable or Disable Traffic to DRAM

This API enables or disables traffic flowing through the DRAM.

NOTE: Use this API if the HBM temperature monitor is disabled.

API

```
int bcm_switch_dram_traffic_enable_set(int unit, uint32 flags, uint32 enable)
```

Enable example:

```
BCM.0> cint
Entering C Interpreter. Type 'exit;' to quit.

cint> print bcm_switch_dram_traffic_enable_set(0,0,1);
int $$ = 0 (0x0)
cint> exit;
BCM.0> dnx dram debug redirectToOCB
currently traffic can be directed to both dram and OCB
```

Disable example:

```
BCM.0> cint
Entering C Interpreter. Type 'exit;' to quit.

cint> print bcm_switch_dram_traffic_enable_set(0,0,0);
int $$ = 0 (0x0)
cint> exit;
BCM.0> dnx dram debug redirectToOCB
currently traffic is restricted to OCB
```

3.5 Call-Back Function: HBM Power-Down

This call-back (CB) function is board-related and is customer implemented. Make sure the code implementation powers down all HBM power supplies.

CB

```
bcm_switch_dram_power_down_cb_register()
bcm_switch_dram_power_down_cb_unregister()
```

Chapter 4: Diagnostics

Use diagnostics commands to get information from the HBM memory subsystem and to perform various operations, such as tuning, BIST, and so on.

All diagnostics operations can be run during normal operation unless otherwise noted.

4.1 Enable or Disable Traffic to Memory

Use these commands to disable or enable traffic to HBM memory. When moving traffic to OCB only, it may take a few seconds until the DRAM is empty and diagnostics operations, such as BIST or tune, can be performed without data loss.

Traffic to HBM Disabled (OCB Only)

```
BCM.0> dnx dram debug redirectToOCB enable=1
```

Traffic to HBM Enabled (with OCB)

```
BCM.0> dnx dram debug redirectToOCB enable=0
```

Using OCB only or OCB + HBM might require a different queuing configuration to be set. This type of configuration is beyond the scope of this document.

Check Current Status Example with Output

Use the `dnx dram debug redirectToOCB` command to check the current status.

Verify that the HBM is enabled, then run the BCM shell command and view the output:

```
BCM.0> dnx dram debug redirectToOCB enable=1
BCM.0> dnx dram debug redirectToOCB
currently traffic is restricted to OCB
BCM.0>
```

NOTE: The highlighted line in the previous output indicates that traffic to the HBM is disabled.

Verify that the HBM is disabled, then run the BCM shell command and view the output:

```
BCM.0> dnx dram debug redirectToOCB enable=0
BCM.0> dnx dram debug redirectToOCB
currently traffic can be directed to both dram and OCB
BCM.0>
```

NOTE: The highlighted line in the previous output indicates that traffic to the HBM is enabled.

4.2 Tuning (BCM88690 and BCM88800 Only)

Diagnostic tuning is primarily run for debug purposes. For more information on tuning, see Section 1.2.6, HBM Tuning and BIST (BCM88690 and BCM88800 Only), and Section 2.2, HBM Tune (BCM88690 and BCM88800 Only).

Tuning works incrementally per each HBM channel for address, read, and write. Tuning results show the selected working point.

NOTE: This section is applicable only to the BCM88690 and BCM88800 devices. The BCM88830 HBM PHY is implemented differently and does not require the tuning, restoring from OTP, or restoring from file mechanisms described in this section.

4.2.1 Read Tuning Information

The following command shows current tuning parameters:

```
BCM.0> config show hbm_tune
```

4.2.2 Diagnostic Tuning

Tuning can be run manually if it is required (for example, for debug purposes).

NOTE: Running tuning while the device handles traffic is not allowed. Disable traffic to the DRAM before tuning.

Diagnostic Tuning Sequence

The commands in this example run tuning on a specific channel.

1. Enable DDR debugging in verbose mode.
2. Disable new traffic to the HBM and wait until all data from the HBM is cleared. (New traffic will be at the OCB only.)
3. Run tuning on DRAM-0 (HBM0) channel-0.
4. Enable HBM usage and redirect data back to the HBM.

Example:

```
BCM.0> debug soc ddr verb
BCM.0> dnx dram debug redirectToOCB enable=1
BCM.0> dnx dram debug eyeScan dram=0 channel=0 type=all
BCM.0> dnx dram debug redirectToOCB enable=0
```

The commands in the following example run tuning on all channels.

Example (all DRAM and all channels):

```
BCM.0> debug soc ddr verb
BCM.0> dnx dram debug redirectToOCB enable=1
BCM.0> dnx dram debug eyeScan
BCM.0> dnx dram debug redirectToOCB enable=0
```

4.2.2.1 Tune Log Phases and Output

Running tune with the verbose option (`dram soc ddr verb`) creates a tune log.

The tune/Shmoo runs in phases:

- RD_EXTENDED Shmoo
- WR_EXTENDED Shmoo
- ADDR_CTL_EXTENDED Shmoo

RD_EXTENDED Shmoo

This phase trains the read path and selects the optimal working point position. On the vertical axis, the Shmoo 2D plot represents the bit trained (0 to 127), and the other axis shows the VDL (variable delay) in which the tune was performed.

The per-bit output generally contains the following pattern:

```
------++++++++++++++++++++++X+++++++++++++++++++-----
```

In the output, the hyphen (`-`) represents a training run that fails, the plus symbol (`+`) represents a training run that passes, and the `X` is the best point of operation.

Each bit should have `X` in the middle with large passing margins.

The following patterns are not good and can be considered as failing:

- Very small margins:
  ```
  ----++++-----+++++-----++++-----+++X+++------+++---------------
  ```
- No margins and no working point:
  ```
  ----------------------------------------------------------------------------------
  ```

WR_EXTENDED Shmoo

This phase trains the write path. The plot is similar to the output from the RD_EXTENDED phase.

ADDR_CTL_EXTENDED Shmoo

This phase trains the address and control bus. It contains one line for all address and control signals. All other output is similar to the RD_EXTENDED output.

4.3 BIST

BIST is a functional test used to stress the interface between the DDR PHY and the HBM. BIST runs at operation speed and imitates the way the controller reads packets from and writes packets to the HBM. On the first power-up after tuning, BIST also validates the tuning of selected working condition points.

NOTE: Running BIST while the device handles traffic is not allowed. Disable all traffic to DRAM before running BIST.

Run HBM BIST on All Memory Array

The commands in this example run memory array BIST on one channel.

1. Enable DDR debugging in verbose mode.
2. Disable HBM usage and redirect new data to OCB only.
3. Run BIST on all memory arrays on DRAM-1 (HBM1) channel-6.
4. Show the results.
5. Enable HBM usage and redirect data back to the HBM.

Example:

```
BCM.0> debug soc ddr verb
BCM.0> dnx dram debug redirectToOCB enable=1
BCM.0> dnx dram debug bist dram=1 channel=6 type=all_address
BCM.0> dnx dram bist print
BCM.0> dnx dram debug redirectToOCB enable=0
```

The commands in this example run memory array BIST on all channels.

Example (all DRAM and all channels):

```
BCM.0> debug soc ddr verb
BCM.0> dnx dram debug redirectToOCB enable=1
BCM.0> dnx dram debug bist type=all_address
BCM.0> dnx dram debug redirectToOCB enable=0
```

Example of BIST Print Output with No Errors

When no errors exist, the values in the *Data error counter* and *Data error bits* columns are all 0. If values other than 0 are present, BIST has failed. The following output shows an example of the BIST print output with no errors.

```
BCM.0> dnx dram debug bist  type=all_address
Dram: 0 | Channel: 0 | Number of Dram BIST errors: 0x00000000
Dram: 0 | Channel: 1 | Number of Dram BIST errors: 0x00000000
Dram: 0 | Channel: 2 | Number of Dram BIST errors: 0x00000000
Dram: 0 | Channel: 3 | Number of Dram BIST errors: 0x00000000
Dram: 0 | Channel: 4 | Number of Dram BIST errors: 0x00000000
Dram: 0 | Channel: 5 | Number of Dram BIST errors: 0x00000000
Dram: 0 | Channel: 6 | Number of Dram BIST errors: 0x00000000
Dram: 0 | Channel: 7 | Number of Dram BIST errors: 0x00000000
Dram: 1 | Channel: 0 | Number of Dram BIST errors: 0x00000000
Dram: 1 | Channel: 1 | Number of Dram BIST errors: 0x00000000
Dram: 1 | Channel: 2 | Number of Dram BIST errors: 0x00000000
Dram: 1 | Channel: 3 | Number of Dram BIST errors: 0x00000000
Dram: 1 | Channel: 4 | Number of Dram BIST errors: 0x00000000
Dram: 1 | Channel: 5 | Number of Dram BIST errors: 0x00000000
Dram: 1 | Channel: 6 | Number of Dram BIST errors: 0x00000000
Dram: 1 | Channel: 7 | Number of Dram BIST errors: 0x00000000
BCM.0> dnx dram bist print
==========================================================================================================================================================
| Dram BIST results                                                                                                                                      |
==========================================================================================================================================================
| Dram index | Channel | Write command counter | Read command counter | Read data counter  | Data error counter | Data error bits                    |
==========================================================================================================================================================
| 0          | 0       | 0x0000000080000000    | 0x0000000080000000   | 0x0000000080000000 | 0x00000000         | 0x00000000000000000000000000000000 |
| 0          | 1       | 0x0000000080000000    | 0x0000000080000000   | 0x0000000080000000 | 0x00000000         | 0x00000000000000000000000000000000 |
| 0          | 2       | 0x0000000080000000    | 0x0000000080000000   | 0x0000000080000000 | 0x00000000         | 0x00000000000000000000000000000000 |
| 0          | 3       | 0x0000000080000000    | 0x0000000080000000   | 0x0000000080000000 | 0x00000000         | 0x00000000000000000000000000000000 |
| 0          | 4       | 0x0000000080000000    | 0x0000000080000000   | 0x0000000080000000 | 0x00000000         | 0x00000000000000000000000000000000 |
| 0          | 5       | 0x0000000080000000    | 0x0000000080000000   | 0x0000000080000000 | 0x00000000         | 0x00000000000000000000000000000000 |
| 0          | 6       | 0x0000000080000000    | 0x0000000080000000   | 0x0000000080000000 | 0x00000000         | 0x00000000000000000000000000000000 |
| 0          | 7       | 0x0000000080000000    | 0x0000000080000000   | 0x0000000080000000 | 0x00000000         | 0x00000000000000000000000000000000 |
| 1          | 0       | 0x0000000080000000    | 0x0000000080000000   | 0x0000000080000000 | 0x00000000         | 0x00000000000000000000000000000000 |
| 1          | 1       | 0x0000000080000000    | 0x0000000080000000   | 0x0000000080000000 | 0x00000000         | 0x00000000000000000000000000000000 |
| 1          | 2       | 0x0000000080000000    | 0x0000000080000000   | 0x0000000080000000 | 0x00000000         | 0x00000000000000000000000000000000 |
| 1          | 3       | 0x0000000080000000    | 0x0000000080000000   | 0x0000000080000000 | 0x00000000         | 0x00000000000000000000000000000000 |
| 1          | 4       | 0x0000000080000000    | 0x0000000080000000   | 0x0000000080000000 | 0x00000000         | 0x00000000000000000000000000000000 |
| 1          | 5       | 0x0000000080000000    | 0x0000000080000000   | 0x0000000080000000 | 0x00000000         | 0x00000000000000000000000000000000 |
| 1          | 6       | 0x0000000080000000    | 0x0000000080000000   | 0x0000000080000000 | 0x00000000         | 0x00000000000000000000000000000000 |
| 1          | 7       | 0x0000000080000000    | 0x0000000080000000   | 0x0000000080000000 | 0x00000000         | 0x00000000000000000000000000000000 |
|            |         |                       |                      |                    |                    |                                    |
==========================================================================================================================================================
BCM.0>
```

Example of BIST Print Output with Errors

When no errors exist, the values in the *Data error counter* and *Data error bits* columns are all 0. If values other than 0 are present, BIST has failed. The following code output shows an example of the BIST print output with errors on HBM0 CH2 and CH6.

```
BCM.0> dnx dram bist start count=0
BCM.0> dnx dram bist print
```

Dram BIST results						
Dram index	Channel	Write command counter	Read command counter	Read data counter	Data error counter	Data error bits
0	0	0x000000002e4e9f00	0x000000002e4e9ec6	0x000000002e4e9e69	0x00000000	0x00000000000000000000000000000000
0	1	0x000000002e48b73b	0x000000002e48b700	0x000000002e48b700	0x00000000	0x00000000000000000000000000000000
0	2	0x000000002e42bc43	0x000000002e42bc00	0x000000002e42bc00	0x0000045f	0x0000000000000000ffffffffffffff04
0	3	0x000000002e3cd886	0x000000002e3cd800	0x000000002e3cd800	0x00000000	0x00000000000000000000000000000000
0	4	0x000000002e36e83f	0x000000002e36e800	0x000000002e36e800	0x00000000	0x00000000000000000000000000000000
0	5	0x000000002e30f7bd	0x000000002e30f700	0x000000002e30f700	0x00000000	0x00000000000000000000000000000000
0	6	0x000000002e2b0e00	0x000000002e2b0d0f	0x000000002e2b0d00	0x2e21d990	0xffffffffffffffffffffffffffffffff
0	7	0x000000002e251db5	0x000000002e251d00	0x000000002e251d00	0x00000000	0x00000000000000000000000000000000
1	0	0x000000002e1f3be3	0x000000002e1f3b00	0x000000002e1f3b00	0x00000000	0x00000000000000000000000000000000
1	1	0x000000002e194b34	0x000000002e194b00	0x000000002e194afd	0x00000000	0x00000000000000000000000000000000
1	2	0x000000002e136c28	0x000000002e136c00	0x000000002e136bdc	0x00000000	0x00000000000000000000000000000000
1	3	0x000000002e0d7a00	0x000000002e0d792f	0x000000002e0d7900	0x00000000	0x00000000000000000000000000000000
1	4	0x000000002e079800	0x000000002e079710	0x000000002e079700	0x00000000	0x00000000000000000000000000000000
1	5	0x000000002e01b500	0x000000002e01b428	0x000000002e01b400	0x00000000	0x00000000000000000000000000000000
1	6	0x000000002dfbc2eb	0x000000002dfbc200	0x000000002dfbc200	0x00000000	0x00000000000000000000000000000000
1	7	0x000000002df5d200	0x000000002df5d1d1	0x000000002df5d175	0x00000000	0x00000000000000000000000000000000

```
BCM.0>
```

Run HBM BIST for IO Stress

The commands in this example run IO stress BIST on one channel.

1. Enable DDR debugging in verbose mode.
2. Disable HBM usage and redirect new data to OCB only.
3. Run BIST on all memory arrays on DRAM-1 (HBM1) channel-6.
4. Show the results.
5. Enable HBM usage and redirect data back to the HBM.

Example:

```
BCM.0> debug soc ddr verb      // Enable DDR debugging in verbose mode.
BCM.0> dnx dram debug redirectToOCB enable=1
BCM.0> dnx dram debug bist dram=1 channel=6 type=IO_stress
BCM.0> dnx dram bist print
BCM.0> dnx dram debug redirectToOCB enable=0
```

For an example of the BIST print results, see Example of BIST Print Output with No Errors. The maximum HBM current is expected to be produced by IO stress BIST.

The commands in this example run IO stress BIST on all channels.

Example (all DRAM and all channels):

```
BCM.0> debug soc ddr verb
BCM.0> dnx dram debug redirectToOCB enable=1
BCM.0> dnx dram debug bist type=IO_stress
BCM.0> dnx dram bist print
BCM.0> dnx dram debug redirectToOCB enable=0
```

Run HBM BIST on all DRAM and Channels Simultaneously

The commands in this example run the following sequence:

1. Enable DDR debugging in verbose mode.
2. Disable HBM usage and redirect new data to OCB only.
3. Run BIST on all DRAMs and channels.
4. Wait 60 seconds (while BIST runs in the background).
5. Stop BIST from running.
6. Print the status.
7. Enable HBM usage and redirect data back to the HBM.

Example:

```
BCM.0> debug soc ddr verb
BCM.0> dnx dram debug redirectToOCB enable=1
BCM.0> dnx dram bist start count=0 mode=prbs
BCM.0> sleep 60
BCM.0> dnx dram bist stop
BCM.0> dnx dram bist print
BCM.0> dnx dram debug redirectToOCB enable=0
```

For an example of the BIST print results, see Example of BIST Print Output with No Errors.

4.4 Read/Write MR Register

CAUTION! Do not modify MR registers unless you must assert CATTRIP for debugging purposes. Modifying the MR registers can lead to misconfiguration and to a driver fatal error.

Use the commands in this section to read or write HBM MR registers by the HBM controller. HBC0 to HBC7 are HBM controllers for the HBM0 device, when present. HBC8 to HBC15 are HBM controllers for the HBM1 device, when present.

Read All HBM MR Registers Short

```
BCM> dnx access read name=HBC_HBM_MODE_REGISTERS
```

Read All HBM MR Registers with Field Information

```
BCM> dnx access read name=^HBC_HBM_MODE_REGISTERS field=1
```

Modify MR Register

The following command modifies HBM controller-3 mode-register-0 with a value of 0x0:

```
BCM> m HBC_HBM_MODE_REGISTERS.HBC3 MODE_REGISTER_0=0x0
```

The following figures show examples of running the BCM shell for read.

Figure 2: BCM Shell Read MR Register Example

```
BCM.0> dnx access read name=HBC_HBM_MODE_REGISTERS
+-----------------------------------------------------------------------------------------------------------------+
| access read                                                                                                     |
+-----------------------------+-----------+-------+-----------+-----------------------------------+-------------+
| Object                      | Block     | Index | Address   | Value                             | Property    |
+-----------------------------+-----------+-------+-----------+-----------------------------------+-------------+
| BRDC_HBC_HBM_MODE_REGISTERS | BRDC_HBC0 | 0     | 0000010b | 00000000000000000700003e1963003 | reg         |
| HBC_HBM_MODE_REGISTERS      | HBC0      | 0     | 0000010b | 00000000000000080700003e1963003 | reg         |
| HBC_HBM_MODE_REGISTERS      | HBC1      | 0     | 0000010b | 00000000000000000700003e1963003 | reg         |
| HBC_HBM_MODE_REGISTERS      | HBC2      | 0     | 0000010b | 00000000000000000700003e1963003 | reg         |
| HBC_HBM_MODE_REGISTERS      | HBC3      | 0     | 0000010b | 00000000000000080700003e1963003 | reg         |
| HBC_HBM_MODE_REGISTERS      | HBC4      | 0     | 0000010b | 00000000000000000700003e1963003 | reg         |
| HBC_HBM_MODE_REGISTERS      | HBC5      | 0     | 0000010b | 00000000000000000700003e1963003 | reg         |
| HBC_HBM_MODE_REGISTERS      | HBC6      | 0     | 0000010b | 00000000000000000700003e1963003 | reg         |
| HBC_HBM_MODE_REGISTERS      | HBC7      | 0     | 0000010b | 00000000000000000700003e1963003 | reg         |
| HBC_HBM_MODE_REGISTERS      | HBC8      | 0     | 0000010b | 00000000000000000700003e1963003 | reg         |
| HBC_HBM_MODE_REGISTERS      | HBC9      | 0     | 0000010b | 00000000000000000700003e1963003 | reg         |
| HBC_HBM_MODE_REGISTERS      | HBC10     | 0     | 0000010b | 00000000000000000700003e1963003 | reg         |
| HBC_HBM_MODE_REGISTERS      | HBC11     | 0     | 0000010b | 00000000000000000700003e1963003 | reg         |
| HBC_HBM_MODE_REGISTERS      | HBC12     | 0     | 0000010b | 00000000000000000700003e1963003 | reg         |
| HBC_HBM_MODE_REGISTERS      | HBC13     | 0     | 0000010b | 00000000000000000700003e1963003 | reg         |
| HBC_HBM_MODE_REGISTERS      | HBC14     | 0     | 0000010b | 00000000000000000700003e1963003 | reg         |
| HBC_HBM_MODE_REGISTERS      | HBC15     | 0     | 0000010b | 00000000000000000700003e1963003 | reg         |
+-----------------------------+-----------+-------+-----------+-----------------------------------+-------------+
```

Figure 3: BCM Shell Read MR Register Details Example

```
BCM.0> dnx access read name=^HBC_HBM_MODE_REGISTERS field=1
=============================================================================================================
| access read                                                                                               |
=============================================================================================================
| Object                 | Block | Index | Address  | Value                                 | Property |
=============================================================================================================
| HBC_HBM_MODE_REGISTERS | HBC0  | 0     | 0000010b | 0000000000000000080700003e1963003 | reg      |
|   MODE_REGISTER_0      |       |       |          |   03                                  |          |
|   MODE_REGISTER_1      |       |       |          |   30                                  |          |
|   MODE_REGISTER_2      |       |       |          |   96                                  |          |
|   MODE_REGISTER_3      |       |       |          |   e1                                  |          |
|   MODE_REGISTER_4      |       |       |          |   03                                  |          |
|   MODE_REGISTER_5      |       |       |          |   00                                  |          |
|   MODE_REGISTER_6      |       |       |          |   70                                  |          |
|   MODE_REGISTER_7      |       |       |          |   80                                  |          |
|   MODE_REGISTER_8      |       |       |          |   00                                  |          |
|   MODE_REGISTER_9      |       |       |          |   00                                  |          |
|   MODE_REGISTER_10     |       |       |          |   00                                  |          |
|   MODE_REGISTER_11     |       |       |          |   00                                  |          |
|   MODE_REGISTER_12     |       |       |          |   00                                  |          |
|   MODE_REGISTER_13     |       |       |          |   00                                  |          |
|   MODE_REGISTER_14     |       |       |          |   00                                  |          |
|   MODE_REGISTER_15     |       |       |          |   00                                  |          |
| HBC_HBM_MODE_REGISTERS | HBC1  | 0     | 0000010b | 0000000000000000000700003e1963003 | reg      |
```

4.4.1 CATTRIP Debug (Assertion Using MR Write)

To simulate a CATTRIP assertion for board testing, access the HBM MR register to assert CATTRIP (resulting in a HBM_THERM_ALARM condition), which should trigger a power-down of the HBM supply.

- To force a CATTRIP, write 1 to HBM MR7 OP[7].
- To clear a CATTRIP, write 0 to HBM MR7 OP[7].

CATTRIP Assertion for HBM0

The following command modifies HBM controller 0 mode-register 7 with a value of 0x80:

```
BCM> m HBC_HBM_MODE_REGISTERS.HBC0 MODE_REGISTER_7=0x80
```

CATTRIP Assertion for HBM1

The following command modifies HBM controller 8 mode-register 7 with a value of 0x80:

```
BCM> m HBC_HBM_MODE_REGISTERS.HBC8 MODE_REGISTER_7=0x80
```

CATTRIP Clear for HBM0

The following command modifies HBM controller 0 mode-register 7 with a value of 0x0:

```
BCM> m HBC_HBM_MODE_REGISTERS.HBC0 MODE_REGISTER_7=0x00
```

CATTRIP Clear for HBM1

The following command modifies HBM controller 8 mode-register 7 with a value of 0x00:

```
BCM> m HBC_HBM_MODE_REGISTERS.HBC8 MODE_REGISTER_7=0x00
```

4.5 HBM Loopback Test Modes (JEDEC-Defined)

JEDEC-defined loopback modes are unidirectional tests that help diagnose DDR issues during HBM debug. All loopback modes test only the interface I/Os, and not the memory array. Aword and Dword loopback modes are controlled using the IEEE 1500 standard. When running the Dword loopback test, DBI and DM signals are treated as pure data signals.

Aword Loopback Test

```
BCM> DRAM DeBuG LooPBack
```

Dword Loopback Test

```
BCM> DRAM DeBuG LooPBack
```

4.6 Read HBM Junction Temperature

This command reads the HBM temperature by accessing the HBM IEEE-1500 module.

The following example reads the temperature of all DRAM devices.

Example:

```
BCM> dnx dram debug temperature
```

Command output example:

```
BCM.0> dnx dram debug temperature
========================
| Dram Temperature     |
========================
| Dram | Temperature   |
========================
| 0x0  | 35 C          |
| 0x1  | 37 C          |
========================
BCM.0>
```

NOTE: If an HBM is disabled when this command is executed, an error message prints, and the temperature for the disabled HBM is reported as *Invalid*.

4.7 HBM PHY and Controller Register Initialization Logs

Use this command to dump debug registers for the HBM PHY and controller in a unified format.

HBM PHY and Controller Register Initialization Log

```
BCM.0> log file=hbm_register_init_log.txt on
BCM.0> dnx dram debug register
BCM.0> log file=hbm_register_init_log.txt off
```

Log notes:

- The *Channels symmetric configuration* statement means that all other channels are configured the same (which makes the output more compact).
- The *Channels A-symmetric configuration* statement means that channels are configured differently.
- The log includes the following configurations:
  - Channels symmetric configuration
  - Controller A-symmetric configuration
  - Controller symmetric configuration
  - Channels A-symmetric configuration

4.8 HBM Counter and Status Logs

Use this command to show block-level counters and events.

Show HBM Controller Indications

```
BCM.0> log file=hbm_counter_and_status_log.txt on
BCM.0> dnx dram debug status
BCM.0> log file=hbm_counter_and_status_log.txt off
```

Log notes:

- Check the status at the beginning of the log for general indications.

  Log header example:

  ```
  No Interrupts were triggered for dram channels
  No content for table:Channels interrupt registers
  HBM error counters are cleared
  Pipeline debug info is cleared
  No content for table:Channels debug registers
  ```

- Log information and counter registers include the following:
  - Channel counter registers
  - Address translation counter registers
  - Controller information registers
  - Address translation counter registers
- Counter status example:

  ```
  HBM error counters are cleared
  HBM error counters are cleared
  No content for table:Channels debug registers
  Channels additional infoHistogram for register HBC_WPBQ_STATUS, block HBC0, index 0
  ```

- Log interrupt status example:

  ```
  No Interrupts were triggered for dram channels
  No content for table:Channels interrupt registers
  No Interrupts were triggered for dram controllers
  No content for table:controllers interrupt registers
  No Interrupts were triggered for address translation
  No content for table:address translation interrupt registers
  ```

Appendix A: HBM CSP Debug Checklist and Tips

A.1 CSP Case Checklist

When opening an HBM-related CSP case, generate the files in Table 2 and complete the checklist in Table 3. Attach these files to the CSP case. Providing the requested information can help expedite the troubleshooting and resolution of the issue.

When attaching log files or JPG files to the CSP case, be sure to add the board name and serial number—or any other unique number—to ease the tracking of the issue per board and of the specific device on the board. For example, `dnx_dram_<platform name>_<board_name>_JR2_<device number>.log`.

Table 2: HBM CSP Debugging Files

Information	Details
Tune logs (BCM88690 and BCM88800 only)	Power-up tune logs with full shmoo and tune information from three passing and three failing boards.
Default settings	From BCM shell, issue the `config show` command and attach the output in a text file.
AC/DC measurements	Provide snapshots and measurements from the AC/DC parameters of the HBM and PHY power rails, as specified in the device data sheet. Perform DC accuracy and AC noise measurements on Broadcom device pads with power probes per the data sheet specification. When sense pins are present, measure them with a differential power probe. BCM88690 and BCM88800: ■ HBM* (VDDC, VDDO, and VPP) ■ DRAM_PHY* (PVDD and VDDC)
Reference clock	Snapshots and measurements of HBM reference clock AC parameters and jitter, as specified in the data sheet. Perform all measurement on Broadcom device pads.
Power-up sequence	Snapshot and measurement from the power-up sequence as it relates to HBM and PHY power rails, as specified in the data sheet.
Failure rate	Provide the following information in a separate Excel file or other table format: ■ % pass vs. fail boards ■ % pass vs. fail devices – Number of boards tested – Number of different platforms tested

Table 3: HBM CSP Debugging Checklist

Information	Details
Customer information	Company name:
	Project name:
	Board name:
Device revision	Select one of the following options: ■ A0 ■ A1 ■ B0 ■ B1 ■ Other: _____
Device package marking	■ 4S ■ 4S2
	Date code:
Board phase:	Select one of the following options: ■ Bring-up ■ Production ■ Other: _____
Symptoms	Select one of the following options: ■ Failures present in logs ■ Signal integrity issues ■ Other: _____
Issue frequency	Select one of the following options: ■ Intermittent ■ Constant
Error conditions	Select one of the following options: ■ In tune ■ When running BIST ■ With traffic ■ Temperature ■ Voltage ■ Other: _____
Channel	Select one of the following options: ■ All channels ■ Specific channels ■ One channel
HBM	Select one of the following options: ■ HBM0 ■ HBM1 ■ Both
SDK	Is the issue SDK related:_____
	SDK versions used when issue is present:_____
Schematics review	CSP case for schematics review:_____ **NOTE:** If the schematics are more advanced than CSP, include schematics in PDF format to the case.
Layout review	CSP case for layout review:_____ **NOTE:** If the layout is more advanced than CSP, include the layout in `.brd` format to the case.

A.2 BIST Run Examples

Use the following BCM shell commands to run random BIST for 60 seconds:

```
BCM> show pvt
BCM> debug soc ddr verb
BCM> dnx dram debug redirectToOCB enable=1
BCM> dnx dnx dram bist print
BCM> dram bist start count=0
BCM> sleep=60
BCM> dnx dram bist stop
BCM> dnx dram bist print
BCM> show pvt
BCM> dnx dram debug redirectToOCB enable=0
```

Run the following tests on all channels in verbose mode:

- all_address mode
- IO_stress

Use the following BCM shell commands to run the tests:

```
BCM> show pvt
BCM> debug soc ddr verb
BCM> dnx dram debug redirectToOCB enable=1
BCM> dnx dnx dram bist print
BCM> dram debug type=all_address
BCM> dram debug type=IO_stress
BCM> show pvt
BCM> dnx dram debug redirectToOCB enable=0
```

Appendix B: HBM Simple Throughput Test

This appendix provides guidelines for performing a very limited test on HBM memory. Other methods are possible but are not included in this document.

The throughput test involves the following three steps:

1. Configure the port modes.

   Set the traffic generator port mode to 100G, and set the loopback ports to 100G also. Perform this step using a configuration file.

2. Redirect all traffic to DRAM and configure MAC loopback snakes.

   Perform this step using the BCM shell.

3. Inject the data from the data generator, sample the number of packets received, and monitor error registers.

   No errors are expected, and all data should be received unless the amount of data generated by the snake is above the DRAM bandwidth. The DRAM bandwidth is split 50/50 between writing the packet and reading the packet.

The following sections provide additional details and examples for each step.

B.1 Configure the Port Modes

To set the port modes, do the following:

1. Configure the input port, which is connected to the traffic generator.
2. Configure loopback ports

The following output shows an example of how to perform this step using the `config-jer2.bcm` file:

```
# ixia input port 100G
ucode_port_1.BCM8869X=CGE20:core_1.1
# snake port 2x50G=100G x 45
ucode_port_2.BCM8869X=CGE2_42:core_1.2
ucode_port_3.BCM8869X=CGE2_0:core_0.3
ucode_port_4.BCM8869X=CGE2_1:core_0.4
ucode_port_5.BCM8869X=CGE2_2:core_0.5
ucode_port_6.BCM8869X=CGE2_3:core_0.6
ucode_port_7.BCM8869X=CGE2_4:core_0.7
ucode_port_8.BCM8869X=CGE2_5:core_0.8
ucode_port_9.BCM8869X=CGE2_6:core_0.9
ucode_port_10.BCM8869X=CGE2_7:core_0.10
ucode_port_11.BCM8869X=CGE2_8:core_0.11
ucode_port_12.BCM8869X=CGE2_9:core_0.12
ucode_port_13.BCM8869X=CGE2_10:core_0.13
ucode_port_14.BCM8869X=CGE2_11:core_0.14
ucode_port_15.BCM8869X=CGE2_12:core_0.15
ucode_port_16.BCM8869X=CGE2_13:core_0.16
ucode_port_17.BCM8869X=CGE2_14:core_0.17
ucode_port_18.BCM8869X=CGE2_15:core_0.18
ucode_port_19.BCM8869X=CGE2_16:core_0.19
ucode_port_20.BCM8869X=CGE2_17:core_0.20
ucode_port_21.BCM8869X=CGE2_18:core_0.21
ucode_port_22.BCM8869X=CGE2_19:core_0.22
ucode_port_23.BCM8869X=CGE2_20:core_0.23
```

```
ucode_port_24.BCM8869X=CGE2_21:core_0.24
ucode_port_25.BCM8869X=CGE2_22:core_0.25
ucode_port_26.BCM8869X=CGE2_24:core_1.26
ucode_port_27.BCM8869X=CGE2_25:core_1.27
ucode_port_28.BCM8869X=CGE2_26:core_1.28
ucode_port_29.BCM8869X=CGE2_27:core_1.29
ucode_port_30.BCM8869X=CGE2_28:core_1.30
ucode_port_31.BCM8869X=CGE2_29:core_1.31
ucode_port_32.BCM8869X=CGE2_30:core_1.32
ucode_port_33.BCM8869X=CGE2_31:core_1.33
ucode_port_34.BCM8869X=CGE2_32:core_1.34
ucode_port_35.BCM8869X=CGE2_33:core_1.35
ucode_port_36.BCM8869X=CGE2_34:core_1.36
ucode_port_37.BCM8869X=CGE2_35:core_1.37
ucode_port_38.BCM8869X=CGE2_36:core_1.38
ucode_port_39.BCM8869X=CGE2_37:core_1.39
ucode_port_40.BCM8869X=CGE2_38:core_1.40
ucode_port_41.BCM8869X=CGE2_39:core_1.41
ucode_port_42.BCM8869X=CGE2_44:core_1.42
ucode_port_43.BCM8869X=CGE2_45:core_1.43
ucode_port_44.BCM8869X=CGE2_46:core_1.44
ucode_port_45.BCM8869X=CGE2_47:core_1.45
ucode_port_46.BCM8869X=CGE2_43:core_1.46
```

B.2 Redirect Data and Configure Snakes

To perform this step, do the following:

1. To redirect data to DRAM, use the `mod CGM_VOQ_SRAM_DRAM_ONLY_MODE` command.
2. Use a CINT script to configure 14-port snakes (14 × 100G throughput to DRAM).
   You can increase or decrease the throughput to DRAM.
   Note that in following example, port 14 routed back to port 1.

The following output shows an example of the BCM shell commands required for this step:

```
BCM.0> mod CGM_VOQ_SRAM_DRAM_ONLY_MODE 0 64 SRAM_DRAM_ONLY_MODE=2
BCM.0> cint
//bcm_port_loopback_set(0, 1, BCM_PORT_LOOPBACK_MAC);
bcm_port_loopback_set(0, 2, BCM_PORT_LOOPBACK_MAC);
bcm_port_loopback_set(0, 3, BCM_PORT_LOOPBACK_MAC);
bcm_port_loopback_set(0, 4, BCM_PORT_LOOPBACK_MAC);
bcm_port_loopback_set(0, 5, BCM_PORT_LOOPBACK_MAC);
bcm_port_loopback_set(0, 6, BCM_PORT_LOOPBACK_MAC);
bcm_port_loopback_set(0, 7, BCM_PORT_LOOPBACK_MAC);
bcm_port_loopback_set(0, 8, BCM_PORT_LOOPBACK_MAC);
bcm_port_loopback_set(0, 9, BCM_PORT_LOOPBACK_MAC);
bcm_port_loopback_set(0, 10, BCM_PORT_LOOPBACK_MAC);
bcm_port_loopback_set(0, 11, BCM_PORT_LOOPBACK_MAC);
bcm_port_loopback_set(0, 12, BCM_PORT_LOOPBACK_MAC);
bcm_port_loopback_set(0, 13, BCM_PORT_LOOPBACK_MAC);
bcm_port_loopback_set(0, 14, BCM_PORT_LOOPBACK_MAC);
bcm_port_force_forward_set(0,1,2,1);
bcm_port_force_forward_set(0,2,3,1);
bcm_port_force_forward_set(0,3,4,1);
bcm_port_force_forward_set(0,4,5,1);
bcm_port_force_forward_set(0,5,6,1);
```

```
bcm_port_force_forward_set(0,6,7,1);
bcm_port_force_forward_set(0,7,8,1);
bcm_port_force_forward_set(0,8,9,1);
bcm_port_force_forward_set(0,9,10,1);
bcm_port_force_forward_set(0,10,11,1);
bcm_port_force_forward_set(0,11,12,1);
bcm_port_force_forward_set(0,12,13,1);
bcm_port_force_forward_set(0,13,14,1);
bcm_port_force_forward_set(0,14,1,1);

exit;
```

B.3 Inject Data and Monitor Results

To perform this step, do the following:

1. Activate the data generator at a certain bandwidth and monitor RX and TX traffic.
   a. Using the 14-port snake loopback mode described in Appendix B.2, Redirect Data and Configure Snakes, produces 14 × 100G bandwidth to DRAM (assuming the data generator is producing 100G). To reduce the bandwidth to DRAM, reduce the generator throughput to a rate lower than 100G or change the loopback configuration.
   b. To increase the bandwidth to DRAM, change the 14-port snake loopback to a longer snake with more ports.
   c. Monitor the data received at the data generator. All data sent should be received. However, data might be dropped for various reasons, such as if the data generator and snake generate more bandwidth than the DRAM can sustain.
   d. Monitor error registers.
2. Monitor device errors.

The following BCM shell commands show an example of how to sample some of the interrupt registers and error counter registers:

```
BCM.0> g HBC_INTERRUPT_REGISTER
BCM.0> g HBMC_ECC_INTERRUPT_REGISTER
BCM.0> g HBC_ECC_2B_ERR_CNT
BCM.0> g HBC_ECC_1B_ERR_CNT
BCM.0> g HBC_PARITY_ERR_CNT
BCM.0> g HBC_HBM_ERR_COUNTERS
```