

# Keystone II DDR3 Debug Guide

Keegan Garcia

## ABSTRACT

This guide provides tools for use when debugging a failing DDR3 interface on a KeyStone II device.

Project collateral and source code discussed in this application report can be downloaded from the following URL: <http://www.ti.com/lit/zip/sprac04>.

## Contents

1	Introduction .....	1
2	TI-Provided Tools for Debug .....	2
3	How to Use TI-Provided tools to Interpret Common DDR3 Issues.....	5
4	Other Common DDR3 Issues.....	10
5	References .....	11

## List of Figures

1	Variables That Can be Modified in the <code>ddr3_memory_test()</code> Function of the DDR3 Debug GEL.....	3
2	Variables that can be Modified in the <code>ddr3_memory_test_address_test()</code> Function of the DDR3 Debug GEL .	4
3	An Example DDR3-1600 Clocking Configuration Through the Cascaded PLL System on Keystone 2 Devices.....	5

## 1 Introduction

DDR3 is a complex peripheral that, in order to work correctly, requires a combination of the correct software PHY and controller configuration, multiple low noise supplies, correct clocking, and robust PCB channel design. If the peripheral does not operate as expected, this combination of requirements can prove tricky to debug!

This application report serves as a short guide to outline the tools that TI provides to help in the DDR3 debug process, review how the TI-provided tools can be used to identify issues, and discuss common issues that can often come up in a design bring-up.

In addition, TI provides documents and tools to help you gain a more fundamental understanding of the DDR3 peripheral and interface.

- *KeyStone II Architecture DDR3 Memory Controller User's Guide* ([SPRUHN7](#)) – for detailed explanation of DDR3 peripheral registers and their functionality.
- *DDR3 Design Requirements for KeyStone Devices* ([SPRABI1](#)) – for DDR3 design and board layout guidance.
- *Keystone II DDR3 Initialization* ([SPRABX7](#)) – for step-by-step initialization guide of the DDR3 peripheral.
- *Keystone II DDR3 Initialization Calculation Spreadsheet* ([SPRABX7](#)) – for calculating how to program the DDR3 peripheral registers for a given use-case or topology.

## 2 TI-Provided Tools for Debug

TI provides a number of tools to help identify and debug DDR3 issues in a board design. These tools are discussed in detail in the following section.

### 2.1 DDR3 Debug GEL

The DDR3 Debug GEL is a script that can be loaded and executed on a C66x DSP core of a Keystone II device using Code Composer Studio™ (CCS) and a IEEE Standard 1149.1-1990, IEEE Standard Test Access Port and Boundary-Scan Architecture (JTAG) connection. The GEL contains pre-made functions that can be run in order to provide register and debug reports specific to the DDR3 peripheral. The GEL also contains some basic memory tests to exercise the DDR3 interface.

---

**NOTE:** The DDR3 Debug GEL functions are intended to only be run after the DDR3 peripheral has been initialized on your Keystone II system.

---

#### 2.1.1 DDR3 Peripheral Register Reporting Functions

The DDR Debug GEL includes a number of functions that help to verify the operation of the DDR3 peripheral by reading and parsing some of the important registers and their contents after initialization.

The GEL register reporting functions attempt to make reading the register contents as easy as possible. The functions print the registers and register contents in an easy-to-read format that can be used to compare against the desired result after initialization.

Some of the register reporting functions that can be run and their purposes are listed below:

- **DDR3n PLL Configuration → Complete\_Report\_DDR3n\_PLL\_Configuration()**  
The DDR3A SoC PLL and DDR3B SoC PLL generate interface clocks for the DDR3A , DDR3B Controller and PHY blocks. When coming out of power-on reset, DDR3A PLL and DDR3B PLL should be programmed to a valid frequency during the boot configuration process before the DDR3 peripheral is enabled and used. The reporting function serves to output the status registers from the DDR3 SoC PLLs and allows you to verify the configured input divider, multiplier, and output divider.
- **DDR3n Controller Configuration → Complete\_Report\_DDR3n\_EMIF\_Configuration()**  
The DDR3 EMIF is the memory controller within the DDR3 peripheral. It controls all command, data prioritization, and arbitration for the DDR3 memory interface. The EMIF must be programmed according to a design's memory topology, address, and command timing parameters. Most of these values can be found in the memory vendor's data sheet. TI provides the [Keystone 2 DDR3 Register Calculation Spreadsheet](#) to help populate the DDR3 EMIF registers. The reporting function outputs the values from some of the Controller registers to help verify the programming for their configuration.
- **DDR3n PHY Configuration → Complete\_Report\_DDR3n\_PHY\_Configuration()**  
The DDR3 PHY handles the physical interface within the DDR3 peripheral. It controls all physical interface timing and operation of the I/O's. The PHY also configures and controls all leveling and training functionality. The PHY must also be programmed according to a design's DRAM timing parameters; most of these values can be found in the memory vendor's spreadsheet. TI's [Keystone 2 DDR3 Register Calculation Spreadsheet](#) can also be used to help populate the DDR3 PHY registers. The reporting function outputs the values from some of the PHY registers to help verify the programming for their configuration.

## 2.1.2 Leveling Status Reporting Functions

Keystone II devices implement a 'leveling' mechanism that aligns the DQS in time such that it arrives at each SDRAM edge-aligned with the clock for writes and aligns the DQS Gate and DQ sample timing for reads. This leveling mechanism is handled internally by the DDR3 PHY, and the status and timing values from the leveling process can be read from various registers within the DDR3 PHY.

The leveling registers can be difficult to interpret, so the Leveling Status Reporting Functions have been created to help parse the register values into a more human readable form.

- **DDR3n Leveling → Report\_Leveling\_Errors\_DDR3n()**

This function reports the leveling status and error bits given in the DDR3 PHY General Status Register (PGSR0) and the DDR3 PHY Data Lane Status Registers (DXnGSR0-2). Depending on which registers are read, the status and errors can provide information on the whole interface or by byte lane.

- **DDR3n Leveling → Report\_Leveling\_Values\_DDR3n()**

This function reports the leveling values found by the DDR3 PHY hardware after the leveling and training initialization sequence has been performed. These leveling values can be analyzed to gain insight into the layout of the DDR3 interface on a byte-lane by byte-lane basis.

## 2.1.3 Memory Tests

The Debug GEL includes two functions that allow you to perform basic functionality tests on DDR3 memory. The available tests include a basic DDR3 access test with CPU-driven writes/reads across a limited address range as well as an address spanning access test that attempts to access multiple memory windows across a larger memory region.

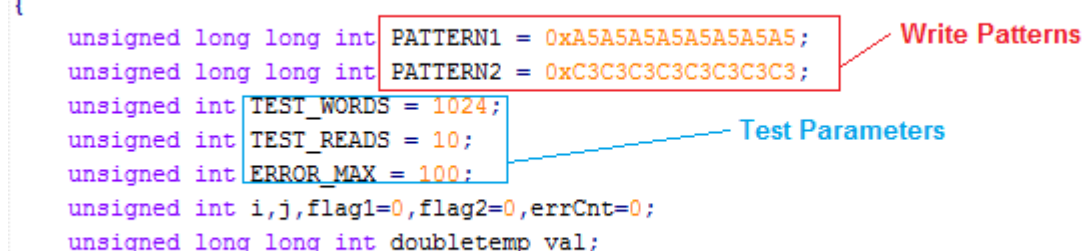
Depending on the size of the DDR3 in use on your design, a different function can be called to test the appropriate range of addresses in the address range test. All memory tests in the DDR3 Debug GEL can be called using GEL functions listed below. A more detailed description of the test operation is also provided below:

- **DDR3n Tests → Write\_Read\_Test\_DDR3n()** – Basic write-read test

This Write/Read test consists of two iterations of write and read cycles as performed by the *ddr3\_memory\_test()* GEL function. By default, each iteration first performs 64b writes via the CPU across a limited range of 64Kb. After all writes are completed, the test then performs 64b reads via the CPU across the 64Kb range. The test performs these reads 10 times in total.

By default, the first cycle iteration writes an incrementing pattern that uses 0xA5A5A5A5A5A5A5A5 as a seed value and the second iteration writes an incrementing pattern that uses 0xC3C3C3C3C3C3C3C3 as a seed value. You can modify the PATTERN1 and PATTERN2 seed variables in the *ddr3\_memory\_test()* function, but any replacement must also be a 64b pattern. The testing range, number of reads, and maximum errors to display can also be modified through the TEST\_WORDS, TEST\_READS, and ERROR\_MAX variables, respectively. Figure 1 shows the variables that can be modified in the *ddr3\_memory\_test()* function of the DDR3 Debug GEL.

```
ddr3_memory_test(unsigned int DDR3_TEST_START_ADDRESS)
{
    unsigned long long int PATTERN1 = 0xA5A5A5A5A5A5A5A5;
    unsigned long long int PATTERN2 = 0xC3C3C3C3C3C3C3C3;
    unsigned int TEST_WORDS = 1024;
    unsigned int TEST_READS = 10;
    unsigned int ERROR_MAX = 100;
    unsigned int i,j,flag1=0,flag2=0,errCnt=0;
    unsigned long long int doubletemp_val;
```



**Figure 1. Variables That Can be Modified in the *ddr3\_memory\_test()* Function of the DDR3 Debug GEL**

- **DDR3n Tests → Test\_Address\_RangeXGB\_DDR3n()** – Write-Read Test across Address Range, where X is 2, 4, or 8 GB

This Write/Read test consists of calls to the `ddr3_mpax_setup()` and `ddr3_memory_test_address_test()` functions in the DDR3 Debug GEL. Both functions are necessary components for testing across the entire 8GB address range available to some Keystone 2 DDR3 interfaces.

A C66X DSP core is limited to logically addressing 2GB of DDR3 memory at a time, so by default is unable to access greater than 2GB of DDR3 at one time. The `ddr3_mpax_setup()` function serves to configure the C66x DSP core MPAX for different 2GB address regions during testing and permit testing across the entire available 8GB memory range.

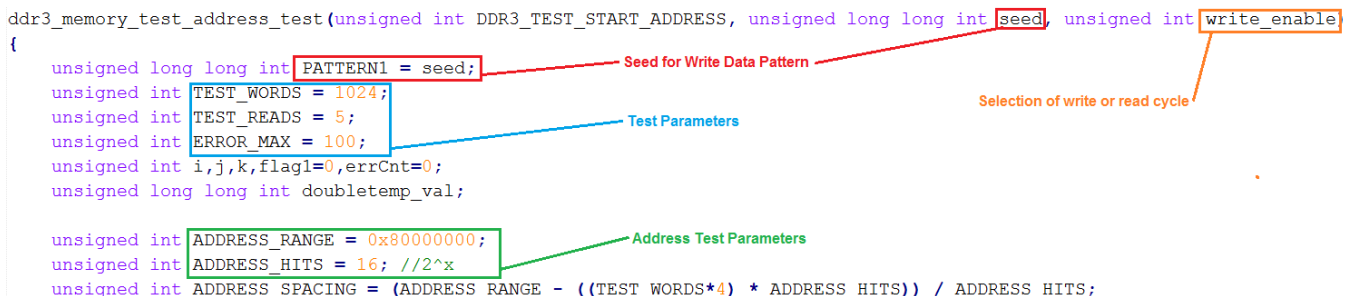
The `ddr3_memory_test_address_test()` function consists of only one iteration of either write or read cycles as chosen by the `write_enable` function parameter. To further save time, the write and read cycles are performed in small regions of the available 2GB addressable space. At each small region, 64b writes across a 64Kb range are performed via the CPU. After all writes are completed, the test then performs 64b reads via the CPU across the 64Kb range. The test performs these reads 5 times in total.

The address regions to test are defined by the `ADDRESS_RANGE`, `ADDRESS_HITS` and `ADDRESS_SPACING` variables. The `ADDRESS_RANGE` variable defines the total DDR3 range addressable by the C66x core at a time (2GB). The `ADDRESS_HITS` defines the number of regions to test in the addressable DDR3 range. The `ADDRESS_SPACING` is a variable calculated to find the spacing between each address testing range.

As with the basic write/read test, the test parameters can also be controlled using the `TEST_WORDS`, `TEST_READS`, and `ERROR_MAX` variables.

```
ddr3_memory_test_address_test(unsigned int DDR3_TEST_START_ADDRESS, unsigned long long int seed, unsigned int write_enable)
{
    unsigned long long int PATTERN1 = seed;
    unsigned int TEST_WORDS = 1024;
    unsigned int TEST_READS = 5;
    unsigned int ERROR_MAX = 100;
    unsigned int i,j,k,flag1=0,errCnt=0;
    unsigned long long int doubletemp_val;

    unsigned int ADDRESS_RANGE = 0x80000000;
    unsigned int ADDRESS_HITS = 16; //2^x
    unsigned int ADDRESS_SPACING = (ADDRESS_RANGE - ((TEST_WORDS*4) * ADDRESS_HITS)) / ADDRESS_HITS;
```



**Figure 2. Variables that can be Modified in the `ddr3_memory_test_address_test()` Function of the DDR3 Debug GEL**

An address test over a large memory range is performed by interleaving calls to `ddr3_mpax_setup()` and `ddr3_memory_test_address_test()` functions. The test works by first writing patterns into the full range of expected addressable DDR3 memory. The test writes patterns using different seed values for each 2GB range that is selected via the `ddr3_mpax_setup()` function. Once the full range of expected addressable DDR3 memory has been written, the test then iterates through each 2GB range and confirms that the correct data pattern is read back. Since unique seed patterns are used for each data, the test sequence ensures that you have access to the correct amount of memory range.

## 2.2 DDR3 EDMA Test

The DDR3 EDMA test is a Code Composer Studio (CCS) project that can be built, loaded, then executed on a device. The test contains multiple cycles of DMA-driven DDR3 accesses in order to maximize throughput on the DDR3 data bus and stress the physical interface. No DDR3 initialization, setup, or configuration is performed during this DDR3 EDMA test. Pre-built \*.out files (for K2K/K2H devices) are provided for convenience, but you can generate your own out file from this CCS project and use it to run the DDR3 EDMA test. A recommended use-case might be to load the \*.out file on an unused C66x core0 after initialization of your system. Like the DDR3 Debug GEL, the DDR3 EDMA test requires that your system already have fully initialized the DDR3 peripheral.

**NOTE:** The DDR3 EDMA test was designed to be run on c66x DSP core and only after the DDR3 peripheral has been initialized on your Keystone II system.

The DDR3 EDMA test consists of multiple iterations of back-to-back write → read → write → read cycles. Each alternating cycle writes a different data pattern followed by 2x read cycles. The first write pattern is an incrementing pattern over a span of 256 64-bit words, while the second write pattern is just zeroes over the same span. The buffers used for writes/reads in the EDMA test are located within the Multicore Shared Memory (MSM).

The write and read cycles can be configured to perform either CPU-driven accesses or EDMA-driven accesses on either the first or second cycle or both cycles. The test is defaulted to EDMA-based accesses, but can be changed by modifying the #defines in the beginning of the main.c file and rebuilding the CCS project. The project can also be configured to access either DDR3A or DDR3B memory space (if applicable for device). The #defines that can be modified are provided below.

```
// Project Defines (User Configurable)
#define EDMA_ON_WRITE1      1
#define EDMA_ON_WRITE2      1

#define EDMA_ON_READ1       1
#define EDMA_ON_READ2       1

#define DDR3A_TEST           1 Select either DDR3A or DDR3B, not both
#define DDR3B_TEST           0 Select either DDR3A or DDR3B, not both
```

### 2.2.1 Debug Spreadsheet

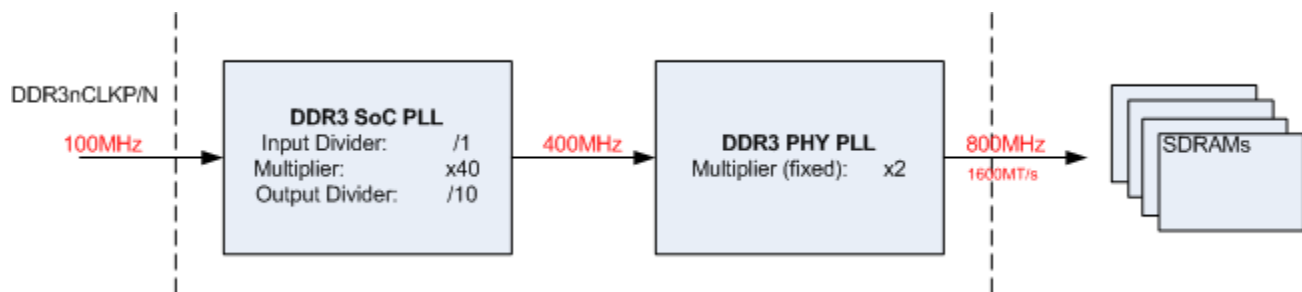
The [Keystone 2 DDR3 Register Calculation Spreadsheet](#) is a simple tool to help document and report any issues found when bringing up the DDR interface on your design. The spreadsheet guides you through the GEL functions that can be used for testing and logging as well as common issues that may need to be verified or addressed in a non-functional design.

When requesting TI support on a DDR3 issue, be sure to provide a completed Debug Spreadsheet and a completed [Keystone 2 DDR3 Register Calculation Spreadsheet](#). This will help the TI support member to understand the context of this issue.

## 3 How to Use TI-Provided tools to Interpret Common DDR3 Issues

### 3.1 PLL/PHY Rate Config

One of the first things that should be checked or confirmed when debugging a DDR3 interface is that all clock rates, both external and internal to the device, appear correct. The DDR3 peripheral operates on a cascaded PLL system that depends on all PLLs being properly programmed for the desired operational frequency. The most important configuration in this clocking system is the DDR3 SoC PLL, which creates the clock source for the DDR3 PHY and its PLL. [Figure 3](#) illustrates an example of a DDR3-1600 clocking configuration from an external reference clock of 100 MHz.



**Figure 3. An Example DDR3-1600 Clocking Configuration Through the Cascaded PLL System on Keystone 2 Devices**

**NOTE:** When debugging the DDR3 operation on a board, it may be worthwhile to try operating the peripheral at different frequencies in order to determine whether or not there is a dependency on the DDR3 clock rate. In general, slower clock frequencies are easier to get working on most boards.

The DDR3 Debug GEL includes a PLL reporting function that can be used to verify some aspects of the DDR3 PLL/PHY Clock Rate Configuration. *DDR3n PLL Configuration* → *Complete\_Report\_DDR3n\_PLL\_Configuration()* that was reviewed earlier in this document, provides an output log that can be reviewed to verify that the DDR3 SoC PLL was configured and initialized correctly. The most important parameters to check are the input divider, multiplier, and output divider of the DDR3 SoC PLL. The PLL Configuration report in the DDR3 Debug GEL outputs these key parameters for verification. An example log from the *DDR3n PLL Configuration* → *Complete\_Report\_DDR3n\_PLL\_Configuration()* function after configuring for DDR3-1600 is provided below.

```
C66xx_0: GEL Output:
*****
C66xx_0: GEL Output: *****
C66xx_0: GEL Output: DDR PLL Registers:

C66xx_0: GEL Output:  DDR3A_PLL_CTL0 register:  0x071803C0 (0x02620360)
C66xx_0: GEL Output:  PLLD[5:0]:                0 (Pre-Divide value of 1)
C66xx_0: GEL Output:  PLLM[18:6]:                39 (Multiplier value of 40)
C66xx_0: GEL Output:  CLKOD[22:19]:              9 (Output Divide value of 10)
C66xx_0: GEL Output:  BYPASS[23]:                0
C66xx_0: GEL Output:  BWADJ-lower[31:24]:         7
C66xx_0: GEL Output:  DDR3A_PLL_CTL1 register:  0x00000040
C66xx_0: GEL Output:  PLLRESET[14]: Reset ** DEASSERTED ** to PLL
C66xx_0: GEL Output:  ENSAT[6]: ENSAT is SET - (GOOD)
C66xx_0: GEL Output:  BWADJ-upper[3:0]:          0
C66xx_0: GEL Output:  BWADJ[11:0] (combined):    7
C66xx_0: GEL Output:
*****
```

With an input reference clock of 100 MHz, the above log tells us that the output of the DDR3 SoC PLL is set to 100 MHz / 1 \* 40 / 10, which is the correct DDR3 SoC PLL frequency for DDR3-1600 operation.

## 3.2 Leveling Errors

During DDR3 initialization, the DDR3 PHY will perform several different leveling and training steps in order to ensure optimal operating timing margins. After the initialization sequence has completed, and the leveling/training has finished, you can check the status of the leveling and training steps using the *DDR3n Leveling* → *Report\_Leveling\_Errors\_DDR3n()* function. You can see part of an example log from this function below:

```
C66xx_0: GEL Output:
*****
C66xx_0: GEL Output: ***** DDR3A Leveling Errors *****
C66xx_0: GEL Output:  PGSR0[27]:  WEERR has  ** No Error **
C66xx_0: GEL Output:  PGSR0[26]:  REERR has  ** No Error **
C66xx_0: GEL Output:  PGSR0[25]:  WDERR has  ** No Error **
C66xx_0: GEL Output:  PGSR0[24]:  RDERR has  ** No Error **
C66xx_0: GEL Output:  PGSR0[23]:  WLAERR has ** No Error **
C66xx_0: GEL Output:  PGSR0[22]:  QSGERR has ** No Error **
C66xx_0: GEL Output:  PGSR0[21]:  WLERR has  ** No Error **
C66xx_0: GEL Output:  PGSR0[20]:  ZCERR has  ** No Error **

C66xx_0: GEL Output:  PGSR0[11]:  WEDONE is  ** Set **
C66xx_0: GEL Output:  PGSR0[10]:  REDONE is  ** Set **
C66xx_0: GEL Output:  PGSR0[9]:   WDDONE is  ** Set **
C66xx_0: GEL Output:  PGSR0[8]:   RDDONE is  ** Set **
C66xx_0: GEL Output:  PGSR0[7]:   WLADONE is ** Set **
C66xx_0: GEL Output:  PGSR0[6]:   QSGDONE is ** Set **
C66xx_0: GEL Output:  PGSR0[5]:   WLDONE is  ** Set **
C66xx_0: GEL Output:  PGSR0[4]:   DIDONE is  ** Set **
```



```
C66xx_0: GEL Output: PGSR0[3]: ZCDONE is ** Set **
C66xx_0: GEL Output: PGSR0[2]: DCDONE is ** Set **
C66xx_0: GEL Output: PGSR0[1]: PLDONE is ** Set **
C66xx_0: GEL Output: PGSR0[0]: IDONE is ** Set **

C66xx_0: GEL Output: *****
C66xx_0: GEL Output: Leveling Errors by Byte Lane:
[...]
```

Only part of the function output is shown above. This section provides an overall impression of the leveling errors across the entire interface. A breakdown of any errors on each byte lane can be found in the later part of the function log. If consistent across bring-ups, errors on a specific byte lane could point to a physical issue on that byte lane.

Nominally, no leveling errors should be seen after initialization. If an error is seen, it is generally an indication that something is fundamentally wrong with the interface or with the software configuration. TI recommends that if an error is seen in the leveling status, start by double checking that the clocks and all DDR3-related power rails are as expected.

### 3.3 Write Errors

The write/read test and DDR3 EDMA test use multiple iterations of write and read cycles to help identify whether a write error or read error is being seen where an error is reported by the test. During these DDR3 tests, errors will only be flagged and printed if the data that was intended to be written to the DRAMs at a specific memory address does not match what was read back.

A write error means that the data that was written to the DRAMs was not what was intended. Write errors can take a few forms. One form of error is represented by a complete 'miss' write to the DRAMs, where stale data remains at the memory address, and not the intended data. In case of a 'miss' write, the stale data is shown across an entire byte lane. Below is an example log (with inline comments) from a 'miss' write on byte lane 3 during the second iteration of the basic write/read test. A 'miss' write error such as this implies that a write command or address may not have been correctly received by the DRAM on that byte lane. A clear indication of a write error is that after the error occurs every subsequent read back of data is incorrect, and the data pattern from each read does not change between reads. The example shows that each of the x10 reads from a single address results in the same read error pattern.

```
C66xx_0: GEL Output: Beginning DDR3 Memory Write/Read Test
C66xx_0: GEL Output: Two iterations to be run...
//>> First iteration, test writes a pattern of 0xA5A5A5A5A5A5A5A5 <<
C66xx_0: GEL Output: First Iteration: Starting Writes
C66xx_0: GEL Output: Write Iteration Pulse: 0 Cycles
C66xx_0: GEL Output: Write Iteration Pulse: 500 Cycles
C66xx_0: GEL Output: Write Iteration Pulse: 1000 Cycles
//>> Test reads back pattern, finds no error <<
C66xx_0: GEL Output: First Iteration: Starting Reads... May take some time
C66xx_0: GEL Output: Read Iteration Pulse: 0 Cycles
C66xx_0: GEL Output: Read Iteration Pulse: 500 Cycles
C66xx_0: GEL Output: Read Iteration Pulse: 1000 Cycles
//>> Second iteration, test writes a pattern of 0xC3C3C3C3C3C3C3C3 <<
C66xx_0: GEL Output: Second Iteration: Starting Writes
C66xx_0: GEL Output: Write Iteration Pulse: 0 Cycles
C66xx_0: GEL Output: Write Iteration Pulse: 500 Cycles
C66xx_0: GEL Output: Write Iteration Pulse: 1000 Cycles
//>> Test reads back pattern, finds error at specific address when reading back data <<
C66xx_0: GEL Output: Second Iteration: Starting Reads... May take some time
C66xx_0: GEL Output: Read Iteration Pulse: 0 Cycles
Read Error @ 80001040 iter 0. Expected: 0xC3C3C3C3C3C3C3C3, Got: 0xC3C3C3C3A5C3C3C3
Read Error @ 80001040 iter 1. Expected: 0xC3C3C3C3C3C3C3C3, Got: 0xC3C3C3C3A5C3C3C3
Read Error @ 80001040 iter 2. Expected: 0xC3C3C3C3C3C3C3C3, Got: 0xC3C3C3C3A5C3C3C3
Read Error @ 80001040 iter 3. Expected: 0xC3C3C3C3C3C3C3C3, Got: 0xC3C3C3C3A5C3C3C3
Read Error @ 80001040 iter 4. Expected: 0xC3C3C3C3C3C3C3C3, Got: 0xC3C3C3C3A5C3C3C3
Read Error @ 80001040 iter 5. Expected: 0xC3C3C3C3C3C3C3C3, Got: 0xC3C3C3C3A5C3C3C3
Read Error @ 80001040 iter 6. Expected: 0xC3C3C3C3C3C3C3C3, Got: 0xC3C3C3C3A5C3C3C3
Read Error @ 80001040 iter 7. Expected: 0xC3C3C3C3C3C3C3C3, Got: 0xC3C3C3C3A5C3C3C3
Read Error @ 80001040 iter 8. Expected: 0xC3C3C3C3C3C3C3C3, Got: 0xC3C3C3C3A5C3C3C3
Read Error @ 80001040 iter 9. Expected: 0xC3C3C3C3C3C3C3C3, Got: 0xC3C3C3C3A5C3C3C3
```

```
Read Error @ 80001040 iter 10. Expected: 0xC3C3C3C3C3C3C3C3, Got: 0xC3C3C3C3A5C3C3C3
C66xx_0: GEL Output: Read Iteration Pulse: 500 Cycles
C66xx_0: GEL Output: Read Iteration Pulse: 1000 Cycles
```

A write error can also take the form of a bit error where a single bit of data (or more) that is written to the DRAMs is incorrect versus what was intended. The erroneous bits may be in a single byte lane, or spread out across multiple byte lanes on the interface. Below is an example log (with inline comments) from bit errors due to a write error on the second iteration of the basic write/read test. Bit errors such as this are indicative of errors in the data signals upon reaching the DRAMs. Once again, a clear indication of a write error is that after the error occurs every subsequent read back of data is incorrect and the data pattern from each read does not change between reads. The example below shows that each of the x10 reads from a single address results in the same read error pattern.

```
C66xx_0: GEL Output: Beginning DDR3 Memory Write/Read Test
C66xx_0: GEL Output: Two iterations to be run...
//>> First iteration, test writes a pattern of 0xA5A5A5A5A5A5A5A5 <<
C66xx_0: GEL Output: First Iteration: Starting Writes
C66xx_0: GEL Output: Write Iteration Pulse: 0 Cycles
C66xx_0: GEL Output: Write Iteration Pulse: 500 Cycles
C66xx_0: GEL Output: Write Iteration Pulse: 1000 Cycles
//>> Test reads back pattern, finds no error <<
C66xx_0: GEL Output: First Iteration: Starting Reads... May take some time
C66xx_0: GEL Output: Read Iteration Pulse: 0 Cycles
C66xx_0: GEL Output: Read Iteration Pulse: 500 Cycles
C66xx_0: GEL Output: Read Iteration Pulse: 1000 Cycles
//>> Second iteration, test writes a pattern of 0xC3C3C3C3C3C3C3C3 <<
C66xx_0: GEL Output: Second Iteration: Starting Writes
C66xx_0: GEL Output: Write Iteration Pulse: 0 Cycles
C66xx_0: GEL Output: Write Iteration Pulse: 500 Cycles
C66xx_0: GEL Output: Write Iteration Pulse: 1000 Cycles
//>> Test reads back pattern, finds error at specific address when reading back data <<
C66xx_0: GEL Output: Second Iteration: Starting Reads... May take some time
C66xx_0: GEL Output: Read Iteration Pulse: 0 Cycles
Read Error @ 80001040 iter 0. Expected: 0xC3C3C3C3C3C3C3C3, Got: 0xC3C2C3C3C3C3D3C3
Read Error @ 80001040 iter 1. Expected: 0xC3C3C3C3C3C3C3C3, Got: 0xC3C2C3C3C3C3D3C3
Read Error @ 80001040 iter 2. Expected: 0xC3C3C3C3C3C3C3C3, Got: 0xC3C2C3C3C3C3D3C3
Read Error @ 80001040 iter 3. Expected: 0xC3C3C3C3C3C3C3C3, Got: 0xC3C2C3C3C3C3D3C3
Read Error @ 80001040 iter 4. Expected: 0xC3C3C3C3C3C3C3C3, Got: 0xC3C2C3C3C3C3D3C3
Read Error @ 80001040 iter 5. Expected: 0xC3C3C3C3C3C3C3C3, Got: 0xC3C2C3C3C3C3D3C3
Read Error @ 80001040 iter 6. Expected: 0xC3C3C3C3C3C3C3C3, Got: 0xC3C2C3C3C3C3D3C3
Read Error @ 80001040 iter 7. Expected: 0xC3C3C3C3C3C3C3C3, Got: 0xC3C2C3C3C3C3D3C3
Read Error @ 80001040 iter 8. Expected: 0xC3C3C3C3C3C3C3C3, Got: 0xC3C2C3C3C3C3D3C3
Read Error @ 80001040 iter 9. Expected: 0xC3C3C3C3C3C3C3C3, Got: 0xC3C2C3C3C3C3D3C3
Read Error @ 80001040 iter 10. Expected: 0xC3C3C3C3C3C3C3C3, Got: 0xC3C2C3C3C3C3D3C3
C66xx_0: GEL Output: Read Iteration Pulse: 500 Cycles
C66xx_0: GEL Output: Read Iteration Pulse: 1000 Cycles
```



### 3.4 Read Errors

A read error means that data that was read and interpreted by the SoC device was misrepresented versus the actual data on the DRAMs. Read errors can also take a few forms similar to write errors. One form is a complete 'miss' read, where an entire byte lane shows incorrect or no data. Below is an example log (with inline comments) from a 'miss' read error on byte lane 6 during the second read iteration. A 'miss' read error such as this implies that a read command or address may not have been correctly received by the DRAM on that byte lane. Note that unlike a write error, the read error will generally only appear once in a series of x10 reads from the same address; subsequent reads from that interface will appear to be correct.

```
C66xx_0: GEL Output: Beginning DDR3 Memory Write/Read Test
C66xx_0: GEL Output: Two iterations to be run...
//>> First iteration, test writes a pattern of 0xA5A5A5A5A5A5A5A5 <<
C66xx_0: GEL Output: First Iteration: Starting Writes
C66xx_0: GEL Output: Write Iteration Pulse: 0 Cycles
C66xx_0: GEL Output: Write Iteration Pulse: 500 Cycles
C66xx_0: GEL Output: Write Iteration Pulse: 1000 Cycles
//>> Test reads back pattern, finds no error <<
C66xx_0: GEL Output: First Iteration: Starting Reads... May take some time
C66xx_0: GEL Output: Read Iteration Pulse: 0 Cycles
C66xx_0: GEL Output: Read Iteration Pulse: 500 Cycles
C66xx_0: GEL Output: Read Iteration Pulse: 1000 Cycles
//>> Second iteration, test writes a pattern of 0xC3C3C3C3C3C3C3C3 <<
C66xx_0: GEL Output: Second Iteration: Starting Writes
C66xx_0: GEL Output: Write Iteration Pulse: 0 Cycles
C66xx_0: GEL Output: Write Iteration Pulse: 500 Cycles
C66xx_0: GEL Output: Write Iteration Pulse: 1000 Cycles
//>> Test reads back pattern, finds error at specific address when reading back data <<
C66xx_0: GEL Output: Second Iteration: Starting Reads... May take some time
C66xx_0: GEL Output: Read Iteration Pulse: 0 Cycles
Read Error @ 80001040 iter 2. Expected: 0xC3C3C3C3C3C3C3C3, Got: 0xC300C3C3C3C3C3C3
C66xx_0: GEL Output: Read Iteration Pulse: 500 Cycles
C66xx_0: GEL Output: Read Iteration Pulse: 1000 Cycles
```

Another form of read error is an individual bit error where a single bit or more of data that is read back from the DRAMs does not match what was written to the DRAM at that address. Below is an example log (with inline comments) from a bit read error on byte lane 6 during the second read iteration. Bit errors such as this are indicative of errors in the data signals upon reaching the DRAMs. Once again, unlike a write error, note that the read error only appears once in a series of x10 reads from the same address; subsequent reads from that interface appear to be correct.

```
C66xx_0: GEL Output: Beginning DDR3 Memory Write/Read Test
C66xx_0: GEL Output: Two iterations to be run...
//>> First iteration, test writes a pattern of 0xA5A5A5A5A5A5A5A5 <<
C66xx_0: GEL Output: First Iteration: Starting Writes
C66xx_0: GEL Output: Write Iteration Pulse: 0 Cycles
C66xx_0: GEL Output: Write Iteration Pulse: 500 Cycles
C66xx_0: GEL Output: Write Iteration Pulse: 1000 Cycles
//>> Test reads back pattern, finds no error <<
C66xx_0: GEL Output: First Iteration: Starting Reads... May take some time
C66xx_0: GEL Output: Read Iteration Pulse: 0 Cycles
C66xx_0: GEL Output: Read Iteration Pulse: 500 Cycles
C66xx_0: GEL Output: Read Iteration Pulse: 1000 Cycles
//>> Second iteration, test writes a pattern of 0xC3C3C3C3C3C3C3C3 <<
C66xx_0: GEL Output: Second Iteration: Starting Writes
C66xx_0: GEL Output: Write Iteration Pulse: 0 Cycles
C66xx_0: GEL Output: Write Iteration Pulse: 500 Cycles
C66xx_0: GEL Output: Write Iteration Pulse: 1000 Cycles
//>> Test reads back pattern, finds error at specific address when reading back data <<
C66xx_0: GEL Output: Second Iteration: Starting Reads... May take some time
C66xx_0: GEL Output: Read Iteration Pulse: 0 Cycles
Read Error @ 80001040 iter 4. Expected: 0xC3C3C3C3C3C3C3C3, Got: 0xC3C1C3C3C3C3C3C3
C66xx_0: GEL Output: Read Iteration Pulse: 500 Cycles
C66xx_0: GEL Output: Read Iteration Pulse: 1000 Cycles
```

## 4 Other Common DDR3 Issues

This section lists other common DDR3 issues that may impact a DDR3 design. These issues are broken down between software and hardware configuration.

### 4.1 Software Configuration

- **Follow the TI-Suggested Initialization Sequence**

Ensure that the correct initialization sequence is being followed per the TI-provided GEL or u-boot initialization sequences. For a step-by-step breakdown of the TI-recommended sequence, or if in doubt whether or not a followed sequence is correct, see *Keystone II DDR3 Initialization* ([SPRABX7](#)).

- **Ensure that all workarounds and fixes related to DDR3 have been implemented per the device-specific Errata and Usage Notes**

Each Keystone 2 device is released with a document listing the errata, workarounds, and usage notes. Some of these items may be of concern to you when implementing the DDR3 peripheral. Please make sure that all workarounds or usage notes are followed according to the device-specific errata document.

- **Verify Correct Programming of DDR3 Registers**

One of the most common issues when using the DDR3 peripheral is misconfiguration for a desired use case or DDR3 topology. Before using the DDR3 peripheral, TI recommends using the [Keystone 2 DDR3 Register Calculation Spreadsheet](#) to ensure that all DDR3 registers are properly configured for the desired use-case.

- **Verify Correct Impedance and ODT Settings**

For optimal performance on the physical DDR3 interface, the output impedance and drive strength and on-die-termination (ODT) settings must be properly configured on the host K2 device as well as on the DRAMs devices. These settings will be determined by the board design chosen in your design. You can choose to set the impedances to the TI recommended or default parameters, or more optimal settings can be chosen through the use of s-parameter extractions of a board model combined with IBIS simulations. Configure these settings using the [Keystone 2 DDR3 Register Calculation Spreadsheet](#).

### 4.2 Hardware Configuration

- Ensure power supplies are correct per the JEDEC specification. Before DDR3 initialization, ensure that all power supplies that pertain to DDR3 are correct and meet the correct JEDEC specifications. These supplies include:
  - VDDQ – VDDQ/DVDD15 is the main DC power supply for the DDR3 DRAMs as well as the DDR3 PHY. This voltage should be at 1.5 V for DDR3 operation or 1.35 V for DDR3L operation. The output voltage from this supply should also meet all noise and stability requirements as outlined by the JEDEC JESD-79 specification.
  - VTT – VTT is the supply voltage that is used to terminate the DDR3 fly-by signals. This voltage should track VDDQ/2, and is generally provided by a dedicated regulator. The output voltage from this supply should meet all noise and stability requirements as outlined by the JEDEC JESD-79 specification.
  - VREF – VREF is the reference voltage used by both the DRAMs and the DDR3 peripheral for referencing I/O signals. This voltage should also track VDDQ/2, and can be created through a dedicated reference supply or a precision resistive divider. The output voltage from this supply should meet all noise and stability requirements as outlined by the JEDEC JESD-79 specification.
- Ensure that all clock signals appear correct. After DDR3 initialization is completed, use an oscilloscope to verify that the DDR3 clocks are being configured correctly for the desired DDR3 data rate. Both the input reference clock can be probed to confirm the DDR3 SoC PLL input is correct, as well as the output clocks to the DRAMs. For a dual rank design, note that there are two sets of clock output pins, one for each rank.
  - Pins/Signals to Check:
    - Input Reference Clock (DDR3nCLKP/N)
    - Output clock to DRAMs (DDR3nCLKOUT0P/N, DDR3nCLKOUT1P/N)

- Follow the DDR3 Routing Guidelines. TI provides a set of DDR3 routing guidelines in *DDR3 Design Requirements for KeyStone Devices* ([SPRABI1](#)). These guidelines should be followed by anyone designing with a TI Keystone device. It is recommended to review the routing guidelines once again to make sure no critical recommendations were missed. Below is an abridged list of routing guidelines that should be verified on your board:
  - Ensure all PCB routes have been length matched appropriately. This includes length matching of the data signals within a byte lane.
  - Ensure all fly-by routes are correctly terminated per the routing guidelines.
  - Ensure that no DDR3 signals are routed across cuts or interruptions in reference planes.
  - Ensure that correct reference planes are used. Data and clock routes must reference to GND only, while ACC routes can be referenced to GND or DVDD15.

## 5 References

- *KeyStone II Architecture DDR3 Memory Controller User's Guide* ([SPRUHN7](#))
- *DDR3 Design Requirements for KeyStone Devices* ([SPRABI1](#))
- *Keystone II DDR3 Initialization* ([SPRABX7](#))
- *Keystone II DDR3 Initialization Calculation Spreadsheet* ([SPRABX7](#))

## IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as "components") are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

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

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

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

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

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

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI's goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or "enhanced plastic" are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have **not** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

### Products

Audio	<a href="http://www.ti.com/audio">www.ti.com/audio</a>
Amplifiers	<a href="http://amplifier.ti.com">amplifier.ti.com</a>
Data Converters	<a href="http://dataconverter.ti.com">dataconverter.ti.com</a>
DLP® Products	<a href="http://www.dlp.com">www.dlp.com</a>
DSP	<a href="http://dsp.ti.com">dsp.ti.com</a>
Clocks and Timers	<a href="http://www.ti.com/clocks">www.ti.com/clocks</a>
Interface	<a href="http://interface.ti.com">interface.ti.com</a>
Logic	<a href="http://logic.ti.com">logic.ti.com</a>
Power Mgmt	<a href="http://power.ti.com">power.ti.com</a>
Microcontrollers	<a href="http://microcontroller.ti.com">microcontroller.ti.com</a>
RFID	<a href="http://www.ti-rfid.com">www.ti-rfid.com</a>
OMAP Applications Processors	<a href="http://www.ti.com/omap">www.ti.com/omap</a>
Wireless Connectivity	<a href="http://www.ti.com/wirelessconnectivity">www.ti.com/wirelessconnectivity</a>

### Applications

Automotive and Transportation	<a href="http://www.ti.com/automotive">www.ti.com/automotive</a>
Communications and Telecom	<a href="http://www.ti.com/communications">www.ti.com/communications</a>
Computers and Peripherals	<a href="http://www.ti.com/computers">www.ti.com/computers</a>
Consumer Electronics	<a href="http://www.ti.com/consumer-apps">www.ti.com/consumer-apps</a>
Energy and Lighting	<a href="http://www.ti.com/energy">www.ti.com/energy</a>
Industrial	<a href="http://www.ti.com/industrial">www.ti.com/industrial</a>
Medical	<a href="http://www.ti.com/medical">www.ti.com/medical</a>
Security	<a href="http://www.ti.com/security">www.ti.com/security</a>
Space, Avionics and Defense	<a href="http://www.ti.com/space-avionics-defense">www.ti.com/space-avionics-defense</a>
Video and Imaging	<a href="http://www.ti.com/video">www.ti.com/video</a>

### TI E2E Community

[e2e.ti.com](http://e2e.ti.com)