Application Report
Lit. Number – March 2011

KeyStone DSP Memory System Test

Brighton Feng *DSP System*

ABSTRACT

Memory related issues are very common on DSP application. This application note introduces some methods to test memory. Example code based on KeyStone DSP is provided.

Contents

Figures

Tables

1 KeyStone DSP memory system introduction

Figure 1 shows the common KeyStone DSP memory architecture.

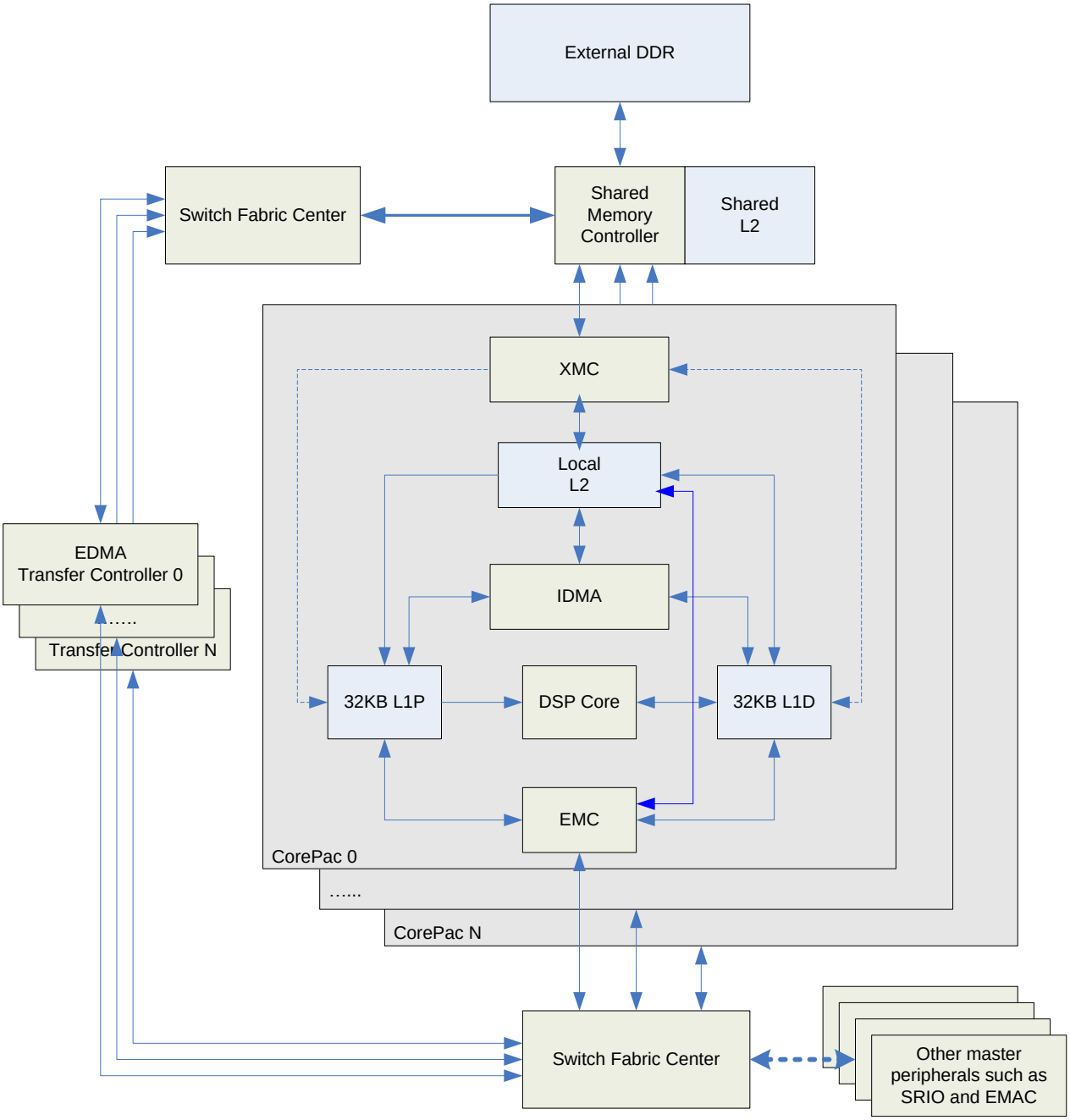

Figure 1. KeyStone DSP memory architecture

For different DSP, the size of memory are different, the number of DSP core and the number of EDMA transfer controller are also different. Table 1 summarizes these differences of KeyStone DSPs.

Table 1. KeyStone Memory system comparison

	C6678	C6670
L1D(KB/core)	32	32
L1P(KB/core)	32	32
Local L2 (KB/core)	512	1024
Shared L2 (KB)	4096	2048
DSP core number	8	4
EDMA transfer controller number	10	10

Memory tests should be implemented to cover:

- All memories in the system;
- All DSP cores and DMA transfer controllers which can access memory
- All data paths for accessing memory

2 Memory test algorithms

Three memory test algorithms are introduced in this section, and the purpose of these test algorithms is discussed.

2.1 Data filling Test

The pseudocode for data pattern filling test is:

```
for(memory range under test)
 fill the memory with a value;
for(memory range under test)
 read back the memory and compare the readback value to the written value
```

Normally, this test is done several times with different test data value. The common values used for test include 0, 0xFFFFFFFF, 0x55555555, 0xAAAAAAAA.

This test can detect stuck data bit, for example, if

written value = 0, readback value = 0x8,

It indicates bit 3 sticks to 1. If

written value = 0xFFFFFFFF, readback value = 0xFFFFFFFE,

It indicates bit 0 sticks to 0.

2.2 Addressing test

The pseudocode for addressing test is:

```
for(memory range under test)
 fill each memory unit with its address value;
for(memory range under test)
 read back the memory and compare the readback value to the written value
```

This test can be done for two times. Firstly, from lower address to higher address; and then, from higher address to lower address.

This test can detect stuck bit on address bus, for example, if

written value = 0 at address 0

written value = 1 at address 1

written value = 2 at address 2

written value = 3 at address 3

……

readback value = 2 at address 0

readback value = 3 at address 1

readback value = x at address 2

readback value = x at address 3

……

The readback value x means don't care the value.

For this case, it indicates bit 1 of address bus sticks to 0. When you write 2 to address 2, it actually writes to address 0 because the bit 1 of address bus sticks to 0; when you write 3 to address 3, it actually writes to address 1 because the bit 1 of the address bus sticks to 0…

Similarly, if

……

written value = 3 at address 3

written value = 2 at address 2

written value = 1 at address 1

written value = 0 at address 0

……

readback value = 1 at address 3

readback value = 0 at address 2

readback value = x at address 1

readback value = x at address 0

For this case, it indicates bit 1 of address bus sticks to 1.

For external memory bus, once we identify the stuck bit. Normally, there are three possible causes:

1. PCB issue. Normally, it is soldering issue, for example, one bus bit is connected to power or ground. We can use multimeter to measure the resistance between the bus signal and the power or ground. If the resistance to power or ground closes to 0, it normally indicates soldering issue.
2. external memory failure. We can use oscilloscope to monitor the bus, if the bit only sticks when read back, it normally indicates external memory failure.
3. DSP memory controller failure. If we exclude above possibilities, then we can swap the DSPs on a good board and the problematic board, if the problem goes with the DSP, then it normally indicates it is DSP memory controller failure.

2.3 Bit walking test

The pseudocode for bit walking test is:

```
test_value = 1;
for(0 to 31)
{
 write test_value to memory;
 readback the data and compared to written value;
 inversed_test_value = bit inverse of test_value;
 write inversed_test_value to memory;
 readback the data and compared to written value;
 test_value = test_value<<1;        //bit walking
}
```

This test is used to identify cross-talk issue. The test intentionally make one bit on the bus or memory cells different from all other bits, thus the negative cross-talk from all other bits is strongest. If there is cross-talk issue, it should be exposed by this test.

For example, if

written value = 0x00000010, readback value = 0

written value = 0xFFFFFFEF, readback value = 0xFFFFFFFF

It indicates the bit 4 is affected by cross-talk issue.

To indentify cross-talk issue on a data bus, this test only need be executed at one address; to identify cross-talk of memory bit cells, this test need be executed for whole memory range under test.

For external memory interface, once we identify this, we should use oscilloscope to check the signal integrity of all related signals to find the source of cross-talk, and then PCB re-layout may be necessary to solve it.

3 Memory Test Coverage

Memory tests should be implemented to cover:

- All memories in the system;
- All DSP cores and DMA transfer controllers which can access memory
- All paths for memory access

3.1 Test by different masters

As shown in figure 1, there are mainly three types of masters can be used to access memory:

- DSP core, can access all memories
- EDMA, can access all memories
- IDMA, can only access Local L1 and L2 memories

All masters should be covered by memory test. However, if all masters are used to test all available memories, there are too much combinations, the test time may be very long. So, memory test should be optimized to use proper master to test proper memories and make sure all of them are covered.

Normally, DMA is much faster than DSP for memory access. So, for big external memory test, EDMA is preferred.

To use DMA to test memory, we need an extra DMA buffer for the test, the test flow is:

DSP core wirtes test data to the DMA buffer

DMA copy the contents of the DMA buffer to the memory under test

DMA copy the contents of the memory under test back to the DMA buffer

DSP core read the contents in DMA buffer and compare to expected value.

Normally, the DMA buffer is a small buffer allocated in L2 memory, the memory under test may be much larger than the DMA buffer, so, above operation need be repeated for multiple times to complete the test.

EDMA has multiple transfer controller, they should be used alternately to make sure all of them are covered. For example, we can use EDMA TC1 to executed data filling test with the first data pattern, and then we use EDMA TC2 to executed data filling test with the second data pattern, and so on…

Please note, on multi-core DSP, EDMA must use global address to access L1 and L2 memory.

3.2 Test all memories in the system

As shown in figure 1, there are mainly four types of memories in the system, all of them should be covered by memory test.

3.2.1 internal L1 memory

Normally, it is recommended use IDMA to test local L1 memory. Before L1 memory is tested, the L1 memory needs to be configured as normal memory instead of cache.

On multi-core DSP, if we want to run a test program on one DSP core to test other core's L1 memory, IDMA can not be used, since IDMA can only access local memory. EDMA should be used in this case. The other cores must be halted when this test is been executing.

DSP core can be used to test L1D memory, but L1P can not be tested with DSP core.

3.2.2 Internal Local L2 memory

L2 may be tested by all masters in the system. According to common use cases, both DSP core and EDMA should be used to test local L2 memory.

Normally, test code and test data buffer are allocated at the beginning of local L2 memory. So, the test program should determine how much memory is occupied by test code and test data buffer, only the remainder part of the L2 memory can be tested by normal memory test algorithms.

Though the part of memory for test code can not be covered by normal memory test algorithms, it is actually used by the test program, if it has problem, the test program will fail somehow. So, to some extent, this part of memory is actually covered by the test. If you really want to cover this part of memory by normal memory test algorithms, you need implement two test programs. One program has the test code in the first half of the memory and test the other part of the memory; and the other program has the test code in the second half of the memory and test the first half of the memory.

On multi-core DSP, if we want to run a test program on one DSP core to test other core's L2 memory, both DSP core and EDMA can be used as well. The other cores must be halted when this test is been executing.

3.2.3 Internal Shared L2 memory

Each DSP core and DMA has separated bus to access Shared L2 memory, so each DSP core and DMA should be used to test shared L2 memory.

DSP core accesses to SL2 at its default space at 0x0C000000 will always go through cache and prefetch buffer. To access SL2 without cache and prefetch buffer, the SL2 can be remapped to another region with can be configured through MAR registers. Both these cases should be covered.

3.2.4 External DDR SDRAM

External DDR SDRAM should be configured properly before test. If DDR test failed on some boards but passed on some other boards, you may try to relax some timing configuration, if it makes some failed board passes, then it may indicate timing issue on DDR interface.

Both DSP cores and EDMA should be used to test DDR SDRAM. For DSP core test, DDR SDRAM should be test with both cache enabled and disabled.

DDR size may be up to 8GB, DDR test consumes most of the test time. So, we should use EDMA to test full space of DDR, since EDMA is the fastest master to access DDR. And them, for other masters, such as DSP core or SRIO, they do not need test the full space of DDR again, they only need test a relative small part of the DDR, the purpose is just to cover the data path from those masters to DDR memory since the DDR memory itself has been fully tested by EDMA. For example, to test DSP core access DDR through L2 cache, we only need test a DDR space larger than L2 cache, thus cover the data path and L2 cache.

3.3 Test through different paths

Since each master has its own bus, same memory may be accessed by different masters from different data paths. For example, DSP core and EDMA access Local L2 through different path.

Same memory may be accessed thought different data path by same master. For example, DDR SDRAM can be accessed by DSP core through cache or directly without cache.

All these different paths should be covered by memory test.

4 Example code based on KeyStone DSP

This section introduces the implementation of these memory tests on KeyStone DSP.

4.1 CCS project

Following figure shows the directory structure of the example projects

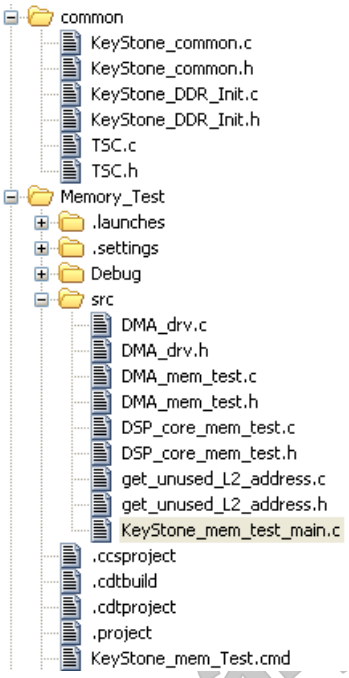

Figure 2. Directory structure of example projects

The project files are in “Memory_Test” folder. There is some commonly used initialization code put in “common” folder. The main source code files are in the “Memory_Test\src” subfolder. Below table describes these source files.

Table 2. Source files of the example codes

Source files	Descriptions
DMA_drv	EDMA and IDMA initialization; implementation code for EDMA and IDMA copy
DMA_mem_test	Implementation of memory test algorithms by EDMA and IDMA, the key API include: int EDMA_MEM_Test(unsigned int uiStartAddress, unsigned int uiStopAddress, unsigned int uiDmaBufAddress, unsigned int uiDmaBufSize); int IDMA_MEM_Test(unsigned int uiStartAddress, unsigned int uiStopAddress, unsigned int uiDmaBufAddress, unsigned int uiDmaBufSize);
DSP_core_mem_test	Implementation of memory test algorithms by DSP core. The key API is: int DSP_core_MEM_Test(unsigned int uiStartAddress, unsigned int uiStopAddress,

	unsigned int uiStep);
get_unused_L2_address	parser .map file to determine the start address of unused L2 for test

The test flow is controlled by the code in “KeyStone_mem_test_main.c”. The basic test flow is:

Disable all caches

Test Local L1 with IDMA

Test other core’s L1 with EDMA

Enable 32KB L1P cache

Test Local L2 with DSP core

Test other core’s L2 with DSP core

Test Shared L2 with DSP core

Test 1KB of external memory with DSP core (just cover the data path)

Enable 32KB L1D cache

Test Local L2 with EDMA

Test Local L2 with DSP core

Test other core’s L2 with EDMA

Test other core’s L2 with DSP core

Test Shared L2 with EDMA

Test Shared L2 with DSP core

Test external memory with EDMA (cover full external memory space)

Test 64KB of external memory with DSP core (just cover the data path and L1D cache)

Enable 256KB L2 cache

Test other core’s L2 with DSP core

Test 512KB external memory with DSP core (just cover the data path and L1D, L2 caches)

Test Shared L2 with DSP core through remmapped noncacheable nonprefetchable window

4.2 Test time

The test time depends on memory size, DSP and memory speed. Following table shows the time measured on demo boards for full memory test introduced in above section.

Table 3. Typical Test time on EVMs

	Shannon EVM	**Nyquist EVM**

DSP speed	1GHz x 8 cores	1GHz x 4 cores
DDR speed	1333MHz	1333MHz
DDR size	512MB	1GB
Test time (second)	287	419

The proportion of test time between data filling test, addressing test and bit walking test is about 4:2:128, because the data filling test is done for four patterns; the addressing test is done for two times; the bit walking test is done on 64 bits for two times.

Most of the time is spent on DDR SDRAM test. For example, on Shannon EVM, 270 seconds is spent on external DDR SDRAM test; about 17 seconds is spent on internal memory test.

The EDMA test is faster than DSP core test. For example, on Shannon, to test the 4MB SL2, the EDMA takes about 1.5 seconds, while the DSP core takes about 2 seconds.

To reduce test time with DSP core, the DSP core memory test function has a parameter “uiStep”:

```
int DSP_core_MEM_Test(unsigned int uiStartAddress, unsigned int uiStopAddress, unsigned int uiStep)
```

Normally, it is called with uiStep= 1 for full coverage like below:

```
DSP_core_MEM_Test(DDR_TEST_START_ARRD, DDR_TEST_STOP_ADDR, 1)
```

uiStep= 1 means to test all DDR memory sequentially. This parameter can be set to a bigger value which means to test one memory unit for every “uiStep” units, thus save time.

The EDMA memory test function does not have this parameter because the efficiency of EDMA index transfer is not good.

4.3 Test configuration

There are multiple macros defined in the source code to configure the test.

The data pattern used for data filling test is defined in “DSP_core_mem_test.c” as below. You can add, remove or modify the values.

```
unsigned int uiDataPatternTable[] = {
0x0000000000000000,
0xffffffffffffffff,
0xaaaaaaaaaaaaaaaa,
0x5555555555555555 };
```

The three memory test algorithms introduced in above section can be enabled or disabled separately through following macros in “DSP_core_mem_test.c” and “DMA_mem_test.c”. “1” means enable, “0” means disable.

```
#define BIT_PATTERN_FILLING_TEST        1
#define ADDRESS_TEST                    1
```

```
#define BIT_WALKING_TEST        1
```

Normally, disable bit walking test or only executing bit walking test on several memory address is a good option to save test time.

Each memory can be configured to be tested or not through following macros in "KeyStone_mem_test_main.c"

```
#define LL1_MEM_TEST            1
#define OTHER_L1_TEST           1
#define LL2_MEM_TEST            1
#define OTHER_L2_TEST           1
#define SL2_MEM_TEST            1
#define EXTERNAL_MEM_TEST       1
```

Whether DSP core or DMA be used for test can be configured through following macros in "KeyStone_mem_test_main.c"

```
#define TEST_BY_DSP_CORE        1
#define TEST_BY_DMA             1
```

These examples are implemented based on TI's demo board. In your real system, the DDR configuration may be changed according to your hardware design in "KeyStone_DDR_Init.c".

The PLL multiplier may also be changed through KeyStone_main_PLL_init() function in "KeyStone_common.c"

To make your own configurations take effect, you must rebuild the project. Since CSL (Chip Support Library) header files are used by these projects, you may need change CSL including path in your system before you rebuild the project.

References

1. *TMS320C66x DSP CorePac User's Guide* (SPRUGW0A)
2. *KeyStone Multicore Shared Memory Controller (MSMC) User's Guide (*SRPUGW7*)*
3. *KeyStone DDR3 Memory Controller User's Guide (*SPRUGV8*)*
4. KeyStone Architecture Enhanced Direct Memory Access (EDMA3) Controller User Guide (SPRUGS5)