

Keystone Architecture Inter-core Data Exchange

*Brighton Feng
Vincent Han*

Communication Infrastructure

ABSTRACT

This application note introduces various methods for inter-core data exchange on the Keystone architecture, the benchmark data for different methods are also provided.

Preliminary

Contents

1	Introduction to Keystone	4
2	Inter-core data exchange on Keystone.....	7
2.1	Inter-core flag exchange	7
2.1.1	Flag in the LL2 of transmitter	7
2.1.2	Flag in the LL2 of receiver	8
2.1.3	Flag in shared memory	9
2.1.4	Use IPC register for inter-core flag exchange	10
2.1.5	Comparison of the methods for inter-core flag exchange	11
2.1.6	Benchmark for different inter-core flag exchange methods	12
2.2	Inter-core data block exchange	14
2.2.1	inter-core data block exchange with shared memory buffer	14
2.2.2	Inter-core data block exchange with shared memory buffer + IPC	16
2.2.3	Inter-core data block exchange with the EDMA.....	17
2.2.4	inter-core data exchange with hardware queue.....	18
2.2.5	Inter-core data block exchange with packet DMA	21
2.2.6	Comparison of the methods for inter-core data block exchange.....	22
2.2.7	Benchmark for different inter-core data block exchange methods.....	23
3	Summary	29
	References	32

Figures

Figure 1.	Keystone devices Block Diagram	4
Figure 2.	Keystone Architecture Memory System	6
Figure 3.	Flag in the LL2 of transmitter	7
Figure 4.	Flag in the LL2 of receiver	8
Figure 5.	Flag in shared memory	9
Figure 6.	Use IPC register for inter-core flag exchange	10
Figure 7.	IPC Generation Registers (IPCGR0-IPCGR2).....	11
Figure 8.	IPC Acknowledgment Registers (IPCAR0-IPCAR2)	11
Figure 9.	Delay for different inter-core flag exchange methods	12
Figure 10.	Simple data structure for the inter-core data block exchange	14
Figure 11.	inter-core data block exchange with shared memory buffer.....	15
Figure 12.	Inter-core data block exchange with shared memory buffer + IPC.....	16
Figure 13.	Inter-core data block exchange with the EDMA	17
Figure 14.	monolithic packet supported by Multicore navigator	18
Figure 15.	host packet supported by Multicore navigator	19
Figure 16.	inter-core data exchange with hardware queue (packet in LL2 of transmitter).....	19
Figure 17.	inter-core data exchange with hardware queue (packet in LL2 of receiver)	20
Figure 18.	inter-core data exchange with hardware queue (packet in LL2 of shared memory).	20
Figure 19.	Inter-core data block exchange with the Packet DMA	22

Tables

Table 1.	Comparison of the methods for inter-core flag exchange.....	12
Table 2.	Delay for different inter-core flag exchange methods	13
Table 3.	Comparison of the methods for inter-core flag exchange.....	23

Table 4.	Delay for inter-core data block exchange with shared memory buffer.....	23
Table 5.	Delay for inter-core data block exchange with shared memory buffer + IPC.....	24
Table 6.	Delay for inter-core data block (monolithic packet) exchange with hardware queue.....	26
Table 7.	Delay for inter-core data block (host packet) exchange with hardware queue.....	27
Table 8.	Delay for inter-core data block exchange with DMA.....	28

Preliminary

1 Introduction to Keystone

Keystone is a DSP architecture designed to aim at the requirements for communication infrastructure systems. The following figure shows a general block diagram of the Keystone devices. Different device in Keystone series may have different size of internal memory, number of cores, coprocessor or peripherals, refer to data sheet of a device for details.

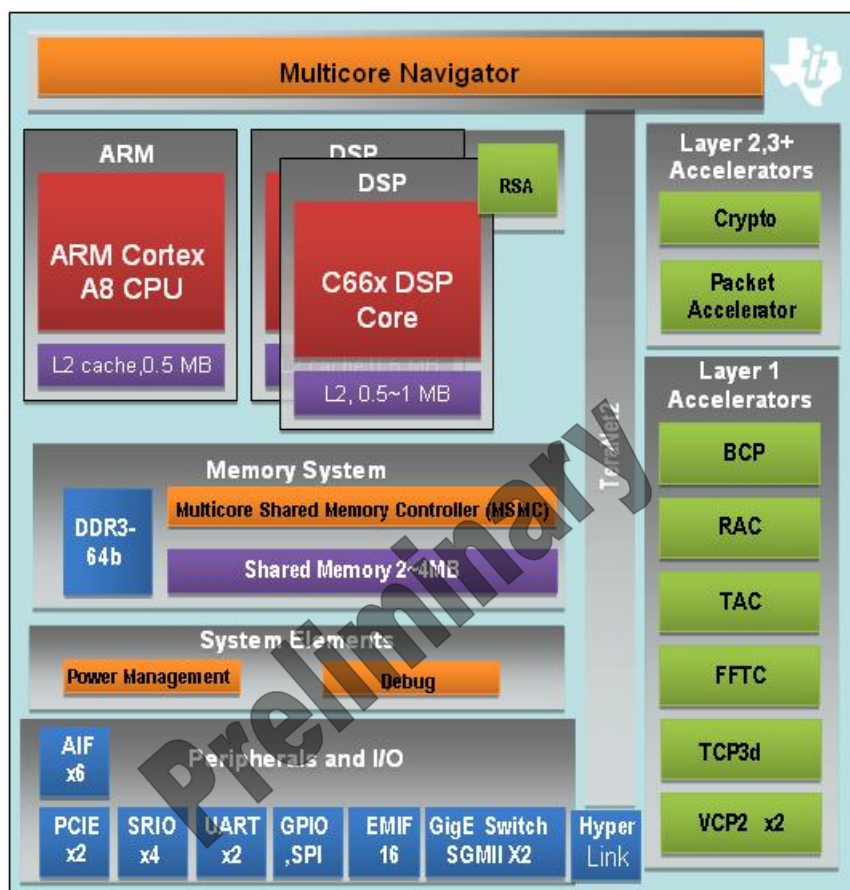


Figure 1. Keystone devices Block Diagram

The Keystone devices may include 1~8 C66x cores, each of them has:

- ⌋ 32KB L1D (Level 1 Data) SRAM, which runs at the DSP Core speed, can be used as normal data memory or cache.
- ⌋ 32KB L1P (Level 1 Program) SRAM, which runs at the DSP Core speed, can be used as normal program memory or cache
- ⌋ 0.5~1MB L2 (Level 2) unified SRAM, which runs at the DSP Core speed divided by two, and can be used as normal memo

Some Keystone devices integrate one Cortex A8 ARM core, 1~8 C66x cores, each of them has:

- ⌋ 32KB Level 1 Data cache.

- └ 32KB Level 1 Program cache
- └ 256KB Level 2 cache

All DSP cores and ARM core share 2~4MB SL2 (Shared Level 2) SRAM, which runs at the DSP Core speed divided by two, can be used as data or code memory.

A 64-bit 1333M DDR SDRAM interface is provided to support up to 8GB external memory, which can be used as data or program memory. The interface can also be configured to only use 32 bits or 16 bits data bus.

The TeraNet switch fabric, which provides the interconnection between the C66x cores (and their local memories), ARM core, external memory, the enhanced DMA v3 (EDMA3) controllers, and on-chip peripherals, there are two main TeraNet switch fabrics, one has 128 bit access bus to each end point, runs at DSP core frequency divided by three, so, in theory, capable of sustaining up to 5.333GB/second at 1GHz core clock frequency; the other TeraNet switch fabric has 256 bit access bus to each end point, runs at DSP core frequency divided by two, so, in theory, capable of sustaining up to 16GB/second at 1GHz core clock frequency.

There are ten EDMA transfer controllers that can be programmed to move data, concurrently, in the background of DSP and ARM core activity, between the on-chip level-two (L2) memory of DSP core, external memory, and the peripherals on the device, two of them connect to the 256-bit TeraNet switch fabric at DSP core clock divided by 2, the other eight connect to the 128-bit TeraNet switch fabric at DSP core clock divided by 3. The EDMA3 architecture has many features designed to facilitate simultaneous multiple high-speed data transfers. With a working knowledge of this architecture and the way in which data transfers interact and are performed, it is possible to create an efficient system and maximize the bandwidth utilization of the EDMA3.

Multicore Navigator is a new architecture for high-speed data packet movement within KeyStone DSPs. The Multicore Navigator uses a Queue Manager Subsystem (QMSS) and a Packet DMA (PKTDMA) to control and implement high-speed data packet movement within the device. This reduces the traditional internal communication load on the DSP core significantly, increases overall system performance.

Following figure shows the memory system of Keystone Architecture. The number on the line is the bus width. Most modules run at CoreClock/n, the DDR typically runs at 1333M.

Overwrite this text with the Lit. Number

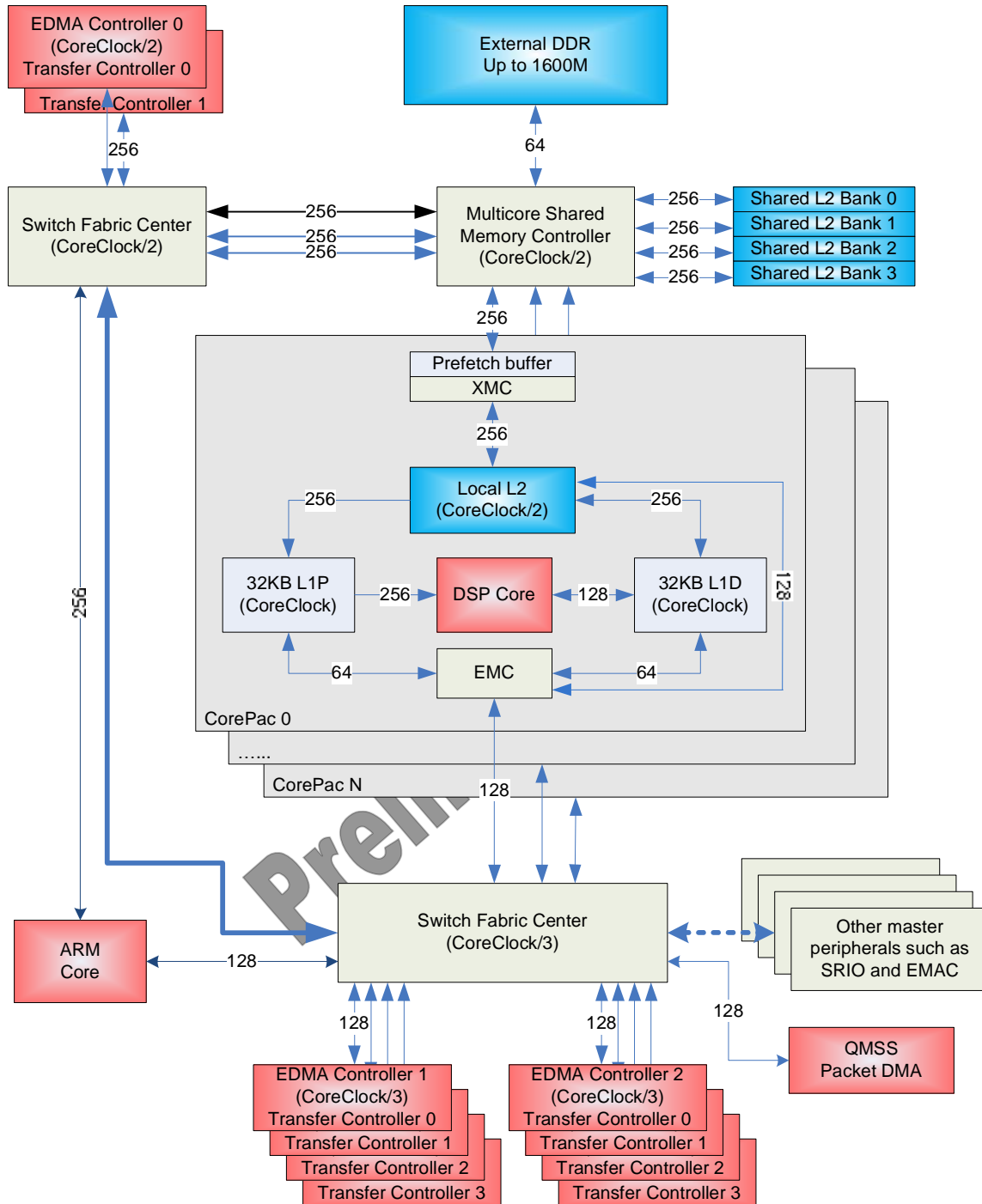


Figure 2. Keystone Architecture Memory System

Inter-core data exchange is one of the biggest challenges of multi-core system programming. There are various ways for inter-core data exchange on Keystone architecture; each has advantages and disadvantages. It is important for a designer to understand these methods, know the differences between them, and choose the best one for his application. This application note discusses these methods and provides basic benchmark data for them.

2 Inter-core data exchange on Keystone

There are two basic types for inter-core data exchange:

- 1) Flag: one core sets a flag, and the other core detects it and takes corresponding actions. Normally, the flag is a data type that is a 32-bit word or smaller.
- 2) Data block: one core transfers data (typically more than one word) to the other core, and the other core processes the received data.

2.1 Inter-core flag exchange

Normally, the flag can be a memory unit of 8 bits, 16 bits or 32 bits. The location of the flag from memory to memory will affect the flag exchange performance. Figures 3 to 5 show different locations of the flag, where Core X sets flag; Core Y detects the flag.

2.1.1 Flag in the LL2 of transmitter

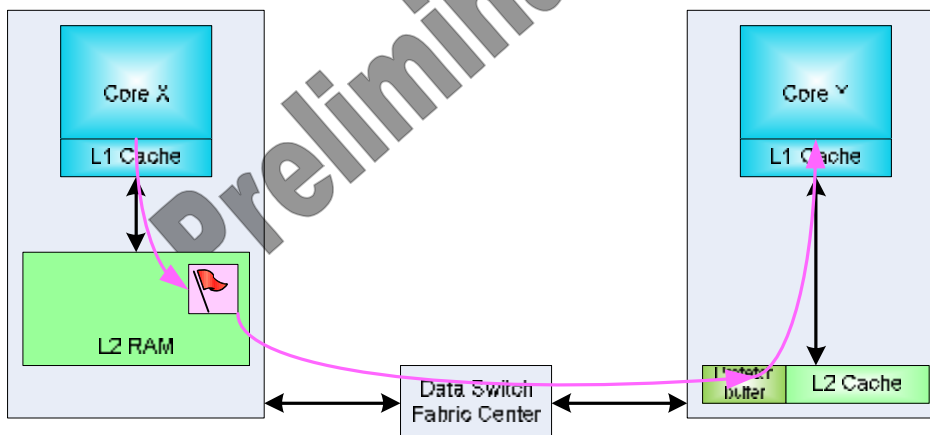


Figure 3. Flag in the LL2 of transmitter

The first case is the flag in Core X's L2 RAM (core X must be a DSP core, all ARM core's internal memory are used as cache). When Core X writes to the flag in L2 RAM, if the flag is cached in the L1D, the DSP hardware will maintain the coherency between the L1D and the L2 RAM (L2 cache of Core X does not take effect for this case). So, the operation on the Core X is:

Write to Flag at L2 RAM

When Core Y reads the flag at Core X's L2 RAM, if the cache or prefetch buffer of Core Y is enabled for accessing memory in Core X, the hardware can not maintain the coherency between Core Y's cache or prefetch buffer and Core X's L2 RAM, Core Y's software needs to maintain the coherency. So, the operations on Core Y are:

Overwrite this text with the Lit. Number

```

If(Cache is enabled for access Core X's L2 RAM)
{
    If(L2 Cache Size>0)
    {
        CACHE_invL2(flag address, size of flag);
    }
    Else if(L1D Cache Size>0)
    {
        CACHE_invL1D(flag address, size of flag);
    }
}
If(Prefetch buffer is enabled for access Core X's L2 RAM)
{
    Invalidate Prefetch Buffer;
}
Read Flag at Core X's L2 RAM

```

Please note that the CACHE_invL2/ CACHE_invL1D functions will invalidate the whole cache line (128/64 bytes) including the flag. All data in the cache line is discarded. However, there may be other data than the flag in the same cache line which should not be discarded. To avoid discarding other valuable data in the line, CACHE_wbInvL2/CACHE_wbInvL1D should be used instead. The other simple way to avoid it is to make sure the whole cache line is only used as flag, which can be implemented with compiler option like: #pragma DATA_ALIGN(flag, 128).

For more details about cache and prefetch buffer, please refer to "TMS320C66x DSP CorePac User's Guide" (SPRUGW0)"

Please note that, ARM core does not support prefetch buffer, and all ARM core's internal memory are used as cache.

2.1.2 Flag in the LL2 of receiver

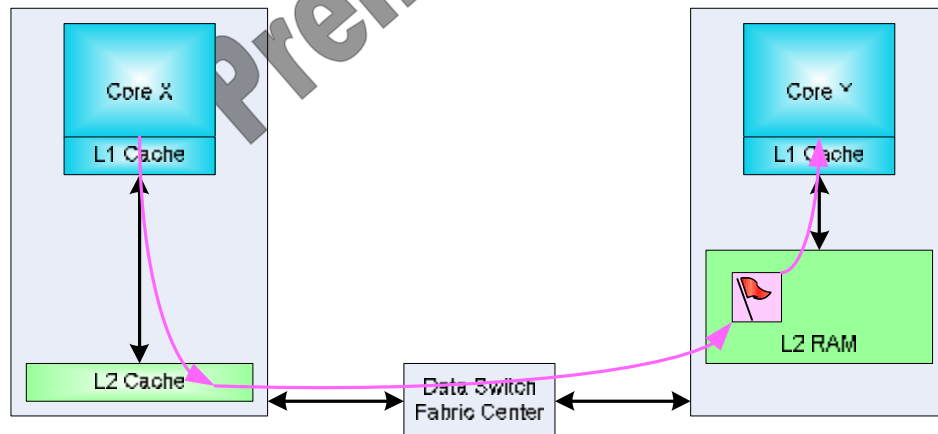


Figure 4. Flag in the LL2 of receiver

The second case is the flag in Core Y's L2 RAM (core Y must be a DSP core, all ARM core's internal memory are used as cache). When Core X writes the flag to Core Y's L2 RAM, if the cache of the Core X is enabled for accessing internal memory of Core Y, the hardware can not maintain the cache coherency between Core X's cache and Core Y's L2 RAM, Core X's software needs to maintain the cache coherency. So, the operations on Core X are:


```

Write Flag at Core Y's L2 RAM
If(Cache is enabled for access Core Y's L2 RAM)
{
    If(L2 Cache Size>0)
    {
        CACHE_wbL2(flag address, size of flag);
    }
    Else if(L1D Cache Size>0)
    {
        CACHE_wbL1D(flag address, size of flag);
    }
}

```

When Core Y reads the flag in Core Y's L2 RAM, if the unit is cached in L1D, the hardware will maintain the coherency between the L1D and the L2 RAM (core Y's prefetch buffer and L2 cache does not take effect for this case). So, the operation on Core Y is:

Read Flag at L2 RAM

2.1.3 Flag in shared memory

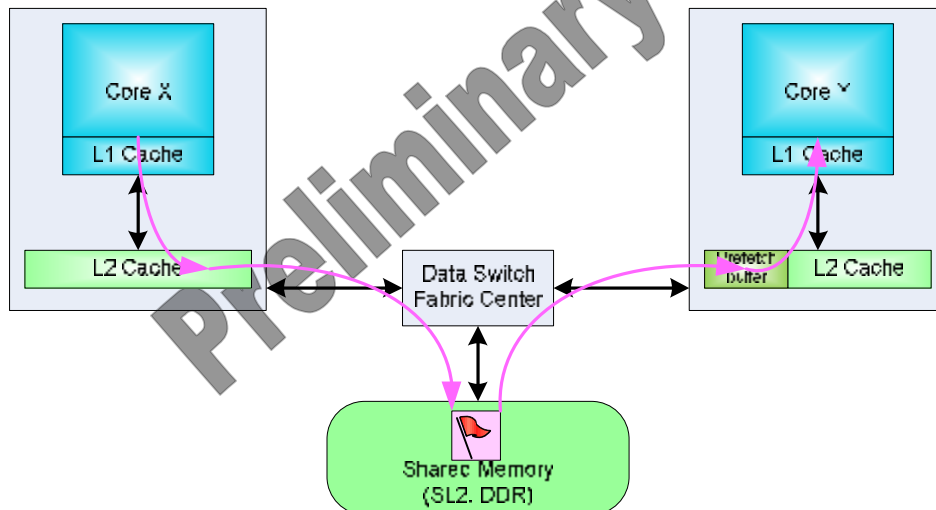


Figure 5. Flag in shared memory

The third case is the flag in the shared memory. If the caches or prefetch buffer are enabled for the core to access the flag in the shared memory, the hardware can not maintain the coherency, software need maintain the coherency. The operations on Core X are:

Overwrite this text with the Lit. Number

```

Write Flag at shared memory
If(Cache is enabled for access shared memory)
{
    If(L2 Cache Size>0)
    {
        CACHE_wbL2(flag address, size of flag);
    }
    Else if(L1D Cache Size>0)
    {
        CACHE_wbL1D(flag address, size of flag);
    }
}

```

The operations on Core Y are:

```

If(Cache is enabled for access shared memory)
{
    If(L2 Cache Size>0)
    {
        CACHE_invL2(flag address, size of flag);
    }
    Else if(L1D Cache Size>0)
    {
        CACHE_invL1D(flag address, size of flag);
    }
}
If(Prefetch buffer is enabled for access Core X's L2 RAM)
{
    Invalidate Prefetch Buffer;
}
Read Flag at shared memory

```

2.1.4 Use IPC register for inter-core flag exchange

The Keystone device has a dedicated hardware module for inter-core communication; it is named IPC (Inter-Processor Communication). The following figure describes the usage of it.

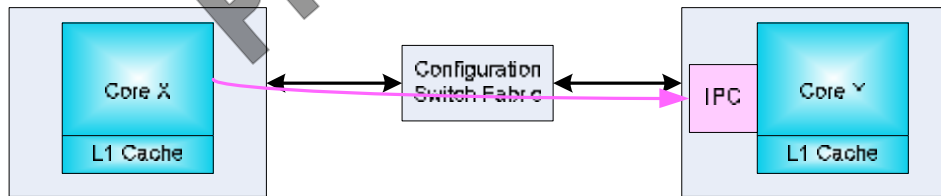


Figure 6. Use IPC register for inter-core flag exchange

Core X directly writes to IPCGRy register of Core Y. Figure 7 describes the register.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SRCS27	SRCS26	SRCS25	SRCS24	SRCS23	SRCS22	SRCS21	SRCS20	SRCS19	SRCS18	SRCS17	SRCS16	SRCS15	SRCS14	SRCS13	SRCS12
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
15	14	13	12	11	10	9	8	7	6	5	4	3		1	0
SRCS11	SRCS10	SRCS9	SRCS8	SRCS7	SRCS6	SRCS5	SRCS4	SRCS3	SRCS2	SRCS1	SRCS0	Reserved			IPCG
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-000			R/W-0

Bit	Field	Value	Description
31:4	SRCS[27:0]	0 1	Write: No effect Set register bit Read: Returns current value of internal register bit
3:1	Reserved		Reserved
0	IPCG	0 1	Write: No effect Create an inter-DSP interrupt pulse to the corresponding C64x+ megamodule (C64x+ Megamodule0 for IPCGR0, etc.) Read: Returns 0, no effect

Figure 7. IPC Generation Registers (IPCGR0-IPCGR2)

The write goes directly through the configuration switch fabric, so there is no cache coherency issue. Bits 4~31 of the register are used as the flag, it can be set to any value by the host application. Writing 1 to the bit 0 can trigger an interrupt to Core Y. In the interrupt service routine of Core Y, it should check the flag (Bits 4~31 of the register) and take an appropriate action. The flag in the register can be cleared with another register as follows:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SRCC27	SRCC26	SRCC25	SRCC24	SRCC23	SRCC22	SRCC21	SRCC20	SRCC19	SRCC18	SRCC17	SRCC16	SRCC15	SRCC14	SRCC13	SRCC12
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
15	14	13	12	11	10	9	8	7	6	5	4	3			0
SRCC11	SRCC10	SRCC9	SRCC8	SRCC7	SRCC6	SRCC5	SRCC4	SRCC3	SRCC2	SRCC1	SRCC0	Reserved			
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0000			

Bit	Field	Value	Description
31:4	SRCC[27:0]	0 1	Write: No effect Clear register bit Read: Returns current value of internal register bit
3:0	Reserved		Reserved

Figure 8. IPC Acknowledgment Registers (IPCAR0-IPCAR2)

2.1.5 Comparison of the methods for inter-core flag exchange

Table 1 compares the above methods.

Table 1. Comparison of the methods for inter-core flag exchange

Flag	Cache coherency Operation on transmitter	Cache coherency Operation on receiver	Interrupt Generation
in the LL2 of transmitter	No	Yes	No
in the LL2 of receiver	Yes	No	No
in the shared memory	Yes	Yes	No
In IPC Generation Register	No	No	Yes

The delay for flag exchange is defined as the time or cycles between the flag set and the flag read. The delay for different methods can be partitioned as following:

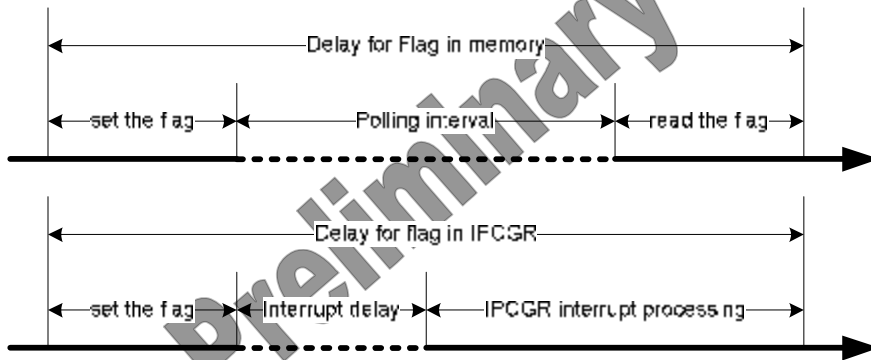


Figure 9. Delay for different inter-core flag exchange methods

For the cases with the flag in memory, the receiver does not know when the flag will be set, it can only poll it periodically. So, the total delay depends on the polling interval, which is indicated as a dash line in figure 9. The polling period is determined by the system designer, which is beyond the scope of this application note. In the test for this application note, the polling interval is almost zero.

For the case utilizing the IPCGR register, normally, if the interrupt is not blocked, the receiver will respond the interrupt immediately after the flag is set, but it depends in a real system. In the test for this application note, the interrupt is never blocked.

2.1.6 Benchmark for different inter-core flag exchange methods

Table 2 shows the delay measured on a 1GHz TCI6618 EVM with 1333MHz 64-bit DDR2. A 32-bit variable is used as a flag for those tests. The cache is flushed before any test.

Table 2. Delay for different inter-core flag exchange methods

Flag	Src->Dst	Cache Configuration	Minimal delay (Cycles)	Notes
in the LL2 of transmitter	DSP->DSP	Non-cacheable, non-prefetch	294	(1)
		32KB L1D + prefetch	378	(2)
		32KB L1D + 256KB L2 + prefetch	414	
	DSP->ARM	32KB L1D + 256KB L2		
in the LL2 of receiver	DSP->DSP	Non-cacheable, non-prefetch	204	(1)
		32KB L1D + prefetch	216	(3)
		32KB L1D + 256KB L2 + prefetch	324	
	ARM->DSP	32KB L1D + 256KB L2		
in the SL2	DSP->DSP	Non-cacheable, non-prefetch	270	(1)
		32KB L1D + prefetch	306	(4)
		32KB L1D + 256KB L2 + prefetch	336	
	DSP->ARM	32KB L1D + 256KB L2 + prefetch for DSP only		
	ARM->DSP	32KB L1D + 256KB L2 + prefetch for DSP only		
in the DDR	DSP->DSP	Non-cacheable, non-prefetch	234	(1)
		32KB L1D + prefetch	372	(4)
		32KB L1D + 256KB L2 + prefetch	420	
	DSP->ARM	32KB L1D + 256KB L2 + prefetch for DSP only		
	ARM->DSP	32KB L1D + 256KB L2 + prefetch for DSP only		
In IPC Generation Register		N/A	666	

Notes:

- (1) The flag writing goes through the L1 Write buffer, which is faster than go through the cache.
- (2) Most cycles consumed by the cache and prefetch buffer invalidation.
- (3) Most cycles consumed by the cache write-back.
- (4) Most cycles consumed by the cache write-back, cache and prefetch buffer invalidation.

The above data shows that the cache is not helpful for inter-core flag exchange.

2.2 Inter-core data block exchange

The inter-core flag exchange exchanges one data unit; we define inter-core data block exchange as exchanging multiple data units. So, the inter-core data block exchange is actually extension of the inter-core flag exchange.

2.2.1 inter-core data block exchange with shared memory buffer

The methods of inter-core flag exchange with the flag in memory can be used for inter-core data block exchange as well. Normally, the inter-core data block exchange also needs additional flag for data availability notification.

A simple data structure for inter-core data block exchange is as below:

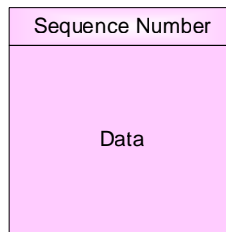


Figure 10. Simple data structure for the inter-core data block exchange

The first data unit is used as sequence number (flag); the other units are the data to be exchanged. The transmitter fills the data and then increases the sequence number to indicate new data is available; the receiver detects the increasing of the sequence number and then read the data. This is only the simplest case of many possibilities; many systems use a more complex data header instead of sequence number.

Following figure shows three different cases with the data exchange buffers in three different places.

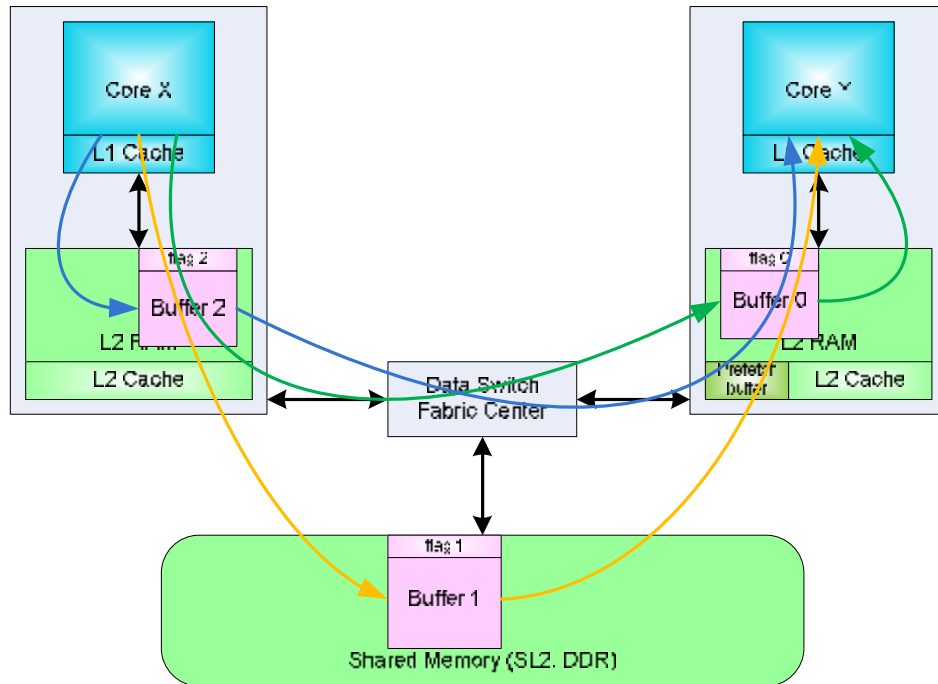


Figure 11. inter-core data block exchange with shared memory buffer

Please note that, the cache coherency should also be taken into account just like the flag exchange introduced in the above sections. So, the operations on Core X should be:

```
The transmitter writes the data buffer
The transmitter increase the sequence number
If(Cache is enabled for access the data block)
{
    If(L2 Cache Size>0)
    {
        CACHE_wbL2(address of the data block, size of the data block);
    }
    Else if(L1D Cache Size>0)
    {
        CACHE_wbL1D(address of the data block, size of the data block);
    }
}
```

The operations on Core Y are:

```

Do
{
  If(Cache is enabled for access the data block)
  {
    If(L2 Cache Size>0)
    {
      CACHE_invL2(address of the data block, size of the data block);
    }
    Else if(L1D Cache Size>0)
    {
      CACHE_invL1D(address of the data block, size of the data block);
    }
  }
  If(Prefetch buffer is enabled for access the data block)
  {
    Invalidate Prefetch Buffer;
  }
}while(Sequence number is not changed)
The receiver read the data buffer

```

2.2.2 Inter-core data block exchange with shared memory buffer + IPC

IPC can also be used as flag for data availability notification for inter-core data block exchange. Following figure shows the cases.

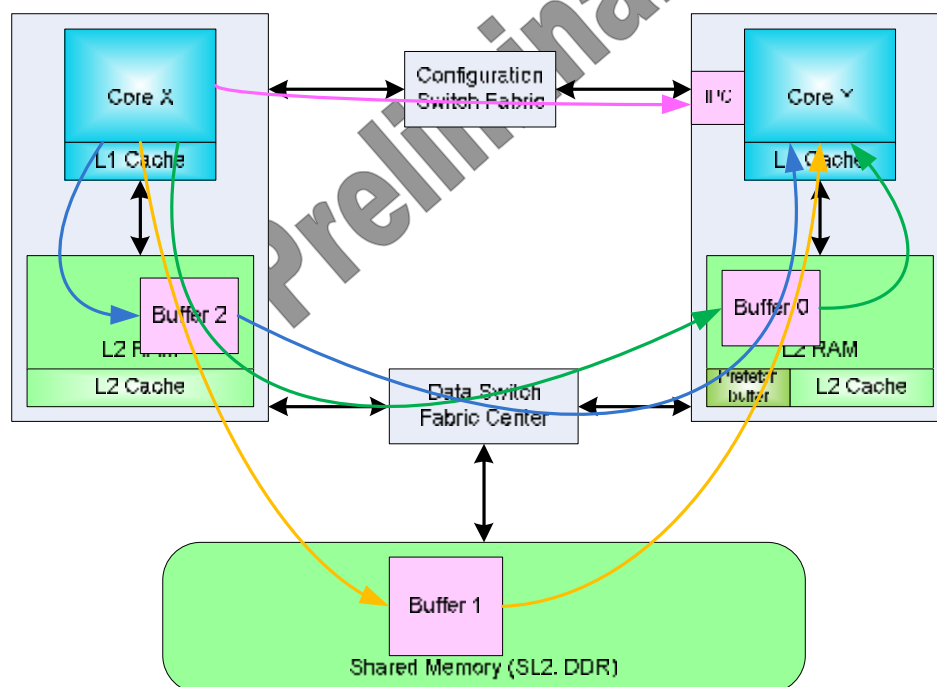


Figure 12. Inter-core data block exchange with shared memory buffer + IPC

The operations on Core X should be:


```

The transmitter writes the data buffer
If(Cache is enabled for access the data block)
{
    If(L2 Cache Size>0)
    {
        CACHE_wbL2(address of the data block, size of the data block);
    }
    Else if(L1D Cache Size>0)
    {
        CACHE_wbL1D(address of the data block, size of the data block);
    }
}
Write IPC register of core Y

```

The operations on Core Y are:

```

Polling the IPC register or waiting for the IPC interrupt.
If(Cache is enabled for access the data block)
{
    If(L2 Cache Size>0)
    {
        CACHE_invL2(address of the data block, size of the data block);
    }
    Else if(L1D Cache Size>0)
    {
        CACHE_invL1D(address of the data block, size of the data block);
    }
}
If(Prefetch buffer is enabled for access the data block)
{
    Invalidate Prefetch Buffer;
}
The receiver read the data buffer

```

2.2.3 Inter-core data block exchange with the EDMA

If the data size is relative large, it is a good idea to transfer the data between the Cores with the EDMA as depicted following figure.

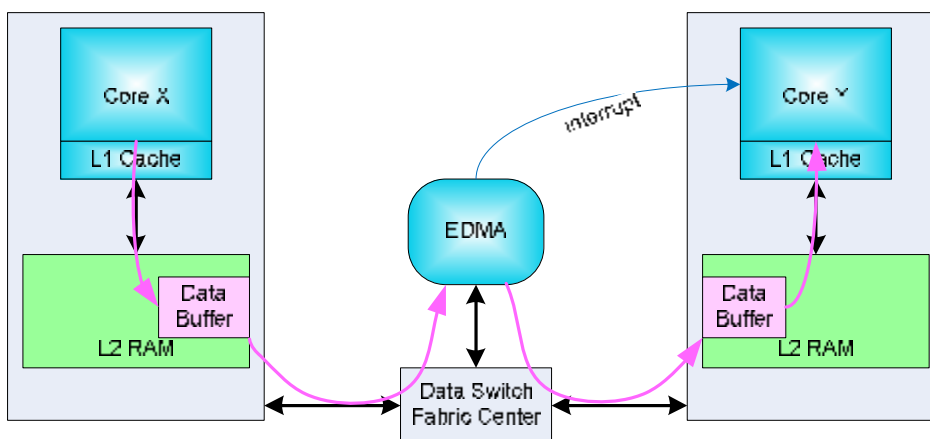


Figure 13. Inter-core data block exchange with the EDMA

Overwrite this text with the Lit. Number

For this case, every Core has its own data buffer in the L2 RAM, the EDMA copies the data from one buffer to another. There is no cache coherency issue for this case since both Cores only access local memory. The operations on Core X are:

```
Write the Data to the local buffer
Configure the EDMA (the EDMA completion interrupt should be enabled)
Start the EDMA to transfer the data
```

When the EDMA completes the data transfer, it will generate an interrupt to both Core X and Core Y. There should be an interrupt service routine on the Core Y. The operations in it are:

```
Read the data from local buffer
Clear the EDMA completion interrupt flag
```

On Core X, it can also have an interrupt service routine to trigger appropriate action, such as preparing the next data block.

2.2.4 inter-core data exchange with hardware queue

The monolithic packet supported by Multicore navigator on Keystone devices is an extension of the data block structure introduced in section 2.2.1.

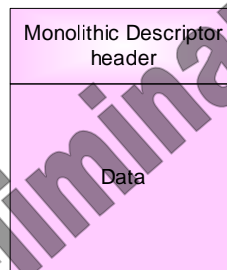


Figure 14. monolithic packet supported by Multicore navigator

The header and data block of monolithic packet are consecutive in memory space, it is simple and high efficient, but it is not flexible for some cases.

Multicore navigator supports a more flexible packet type, host packet, the header and data for it can be placed in different place, there is a pointer in the header to specify the position of data buffer, following figure shows two host packets.

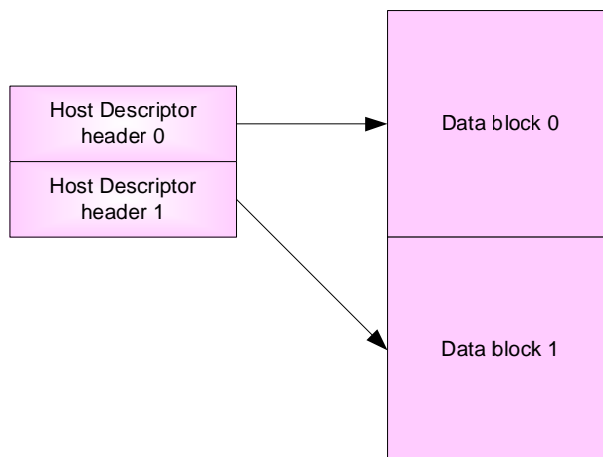


Figure 15. host packet supported by Multicore navigator

For more details about Multicore navigator, refer to “*KeyStone Architecture Multicore Navigator User's Guide (sprugr9)*”

Following figures show three different cases for inter-core packet exchange with hardware queue.

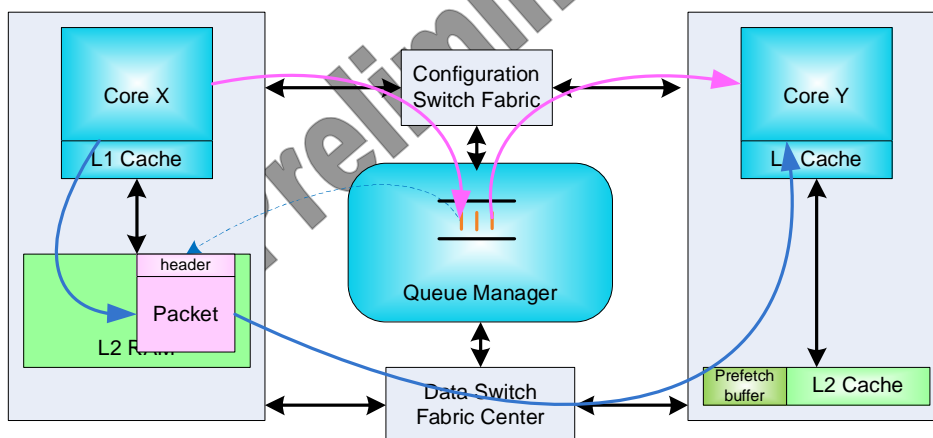


Figure 16. inter-core data exchange with hardware queue (packet in LL2 of transmitter)

Overwrite this text with the Lit. Number

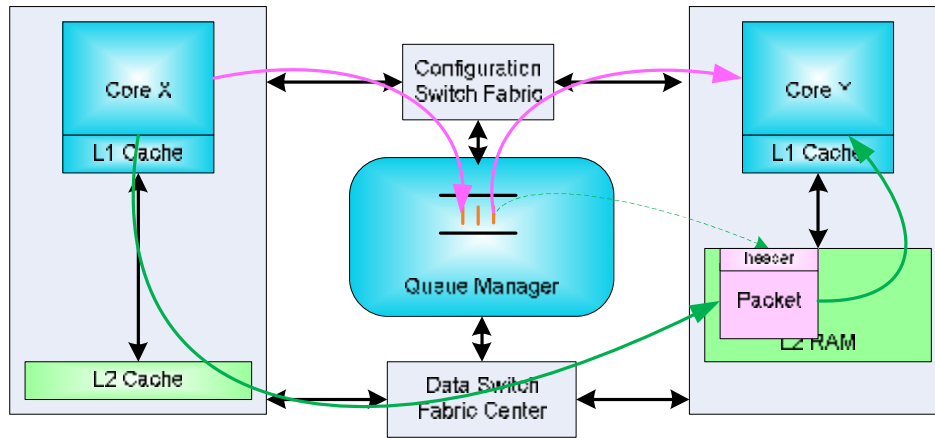


Figure 17. inter-core data exchange with hardware queue (packet in LL2 of receiver)

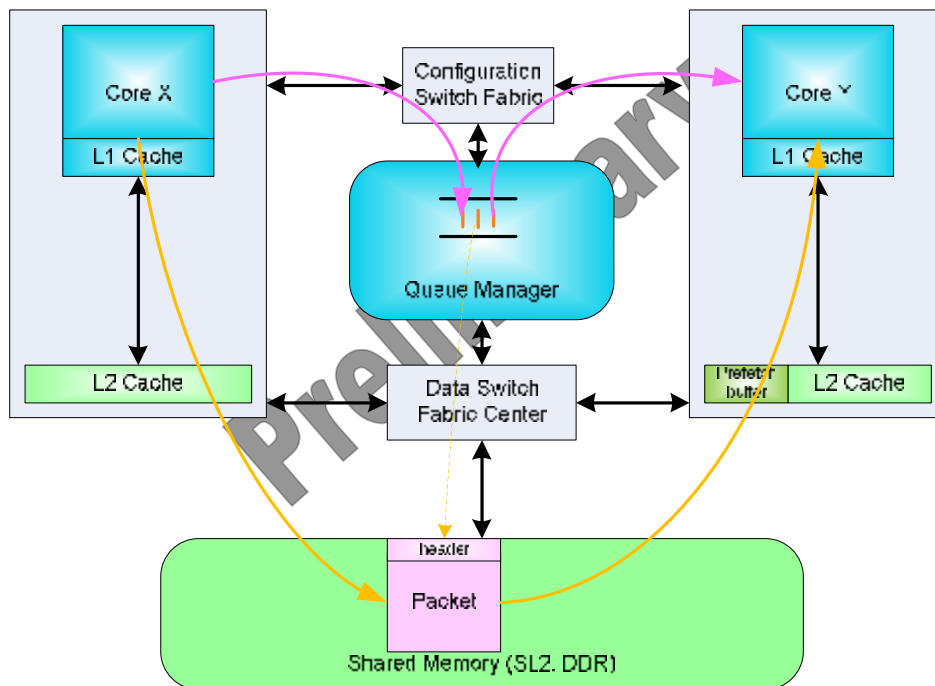


Figure 18. inter-core data exchange with hardware queue (packet in LL2 of shared memory)

Normally, the queue operations go through configuration switch fabric, cache is not used. The cache coherency for packet read/write is still a problem. So, the operations on Core X should be:

```

The transmitter fills the data buffer
The transmitter fills the packet header
If(Cache is enabled for access the packet)
{
    If(L2 Cache Size>0)
    {
        CACHE_wbL2(address of the packet, size of the packet);
    }
    Else if(L1D Cache Size>0)
    {
        CACHE_wbL1D(address of the packet, size of the packet);
    }
}
The transmitter pushes the packet header pointer to the receiver's queue

```

The operations on Core Y are:

```

Polling the receiver queue or waiting for the queue pending interrupt.
If(Cache is enabled for access the packet)
{
    If(L2 Cache Size>0)
    {
        CACHE_invL2(address of the packet, size of the packet);
    }
    Else if(L1D Cache Size>0)
    {
        CACHE_invL1D(address of the packet, size of the packet);
    }
}
If(Prefetch buffer is enabled for access the packet)
{
    Invalidate Prefetch Buffer;
}
The receiver read the packet header
The receiver read the data buffer

```

Please note, for host packet, if the packet header and packet buffer are in different places, above cache coherency operation should be done separately, that is the reason we say the host packet is not as efficient as monolithic packet.

2.2.5 Inter-core data block exchange with packet DMA

To simplify the cache operations, packet DMA can be used to transfer packet from one core's LL2 to another core's LL2.

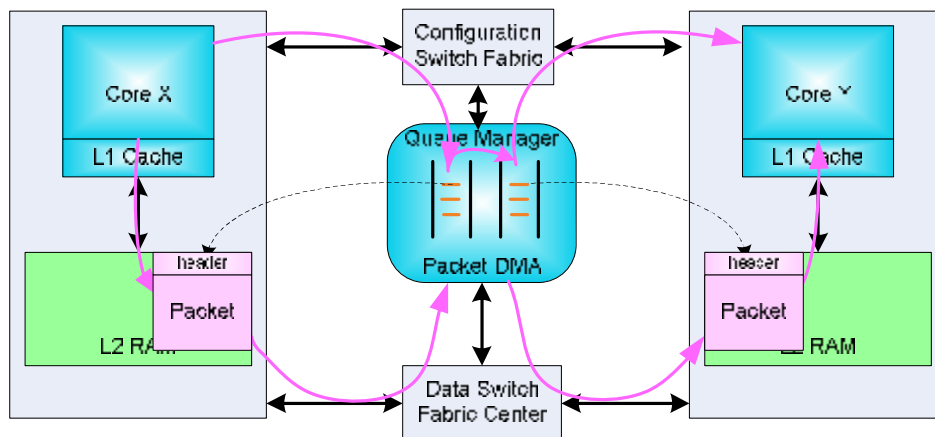


Figure 19. Inter-core data block exchange with the Packet DMA

For this case, every Core has its own data buffer in the L2 RAM. There is no cache coherency issue for this case since both Cores only access local memory. So, the operations on Core X should be:

The transmitter fills the data buffer
The transmitter fills the packet header
The transmitter pushes the packet header pointer to the transmit queue

The pushing to the transmit queue will trigger a corresponding packet DMA transfer. The packet DMA copies the contents of the packet in Core X's L2 to a packet buffer in Core Y's L2, and then push the descriptor header pointer to receiver's queue.

The operations on Core Y are:

Polling the receiver queue or waiting for the queue pending interrupt.
The receiver read the packet header
The receiver read the data buffer

2.2.6 Comparison of the methods for inter-core data block exchange

Following table compares the above methods.

Table 3. Comparison of the methods for inter-core flag exchange

Flag	Cache coherency Operation	Interrupt Generation
shared memory buffer	Yes	No
shared memory buffer + IPC	Yes	Yes
EDMA	No	Yes
Hardware queue	Yes	Yes
Packet DMA	No	Yes

2.2.7 Benchmark for different inter-core data block exchange methods

The delay for data block exchange is defined as the time or cycles between the first data unit write and the last data unit read.

Following tables show the data measured on a 1GHz TCI6618 EVM with 1333MHz 64-bit DDR2. The cache is flushed before each test.

Table 4. Delay for inter-core data block exchange with shared memory buffer

Buffer location	Src->Dst	Cache Configuration	Minimal delay (Cycles) for different data size (Bytes)						
			16B	64B	256B	1KB	4KB	16KB	64KB
LL2 of Transmitter	DSP->DSP	Non-cacheable, non-prefetch	336	588	1680	5712	22026	87336	345720
		32KB L1D + prefetch	414	414	624	1320	4110	15246	59340
		32KB L1D + 256KB L2 + prefetch	450	450	582	1176	3072	10668	40854
	DSP->ARM	32KB L1D + 256KB L2							
LL2 of receiver		Non-cacheable, non-prefetch	270	270	420	1032	3480	13278	52056
		32KB L1D + prefetch	264	270	426	660	1812	6420	24666
		32KB L1D +							
			378	588	756	1428	4320	15846	61494

Overwrite this text with the Lit. Number

		256KB L2 + prefetch							
	ARM->DSP	32KB L1D + 256KB L2							
SL2	DSP->DSP	Non-cacheable, non-prefetch	294	366	654	1818	6456	25032	98820
		32KB L1D + prefetch	342	342	402	672	1766	6084	23214
		32KB L1D + 256KB L2 + prefetch	372	378	438	714	1824	6246	23916
	DSP->ARM	32KB L1D + 256KB L2 + prefetch for DSP only							
	ARM->DSP	32KB L1D + 256KB L2 + prefetch for DSP only							
DDR	DSP->DSP	Non-cacheable, non-prefetch	420	684	1914	6720	26340	106698	424764
		32KB L1D + prefetch	408	408	594	1278	4602	16590	64752
		32KB L1D + 256KB L2 + prefetch	438	642	762	1674	5070	17514	76848
	DSP->ARM	32KB L1D + 256KB L2 + prefetch for DSP only							
	ARM->DSP	32KB L1D + 256KB L2 + prefetch for DSP only							

Table 5. Delay for inter-core data block exchange with shared memory buffer + IPC

Buffer location	Src->Dst	Cache Configuration	Minimal delay (Cycles) for different data size (Bytes)						
			16B	64B	256B	1KB	4KB	16KB	64KB
LL2 of Transmitter	DSP->DSP	Non-cacheable, non-prefetch	1002	1254	2346	6378	22692	88002	346386

		32KB L1D + prefetch	1080	1080	1290	1986	4776	15912	60006
		32KB L1D + 256KB L2 + prefetch	1116	1116	1248	1842	3738	11334	41520
	DSP->ARM	32KB L1D + 256KB L2							
LL2 of receiver		Non-cacheable, non-prefetch	936	936	1086	1698	4146	13944	52722
		32KB L1D + prefetch	930	936	1092	1326	2478	7086	25332
		32KB L1D + 256KB L2 + prefetch	1044	1254	1422	2094	4986	16512	62160
	ARM->DSP	32KB L1D + 256KB L2							
SL2	DSP->DSP	Non-cacheable, non-prefetch	960	1032	1320	2484	7122	25698	99486
		32KB L1D + prefetch	1008	1008	1068	1338	2442	6750	23880
		32KB L1D + 256KB L2 + prefetch	1038	1044	1104	1380	2490	6912	24582
	DSP->ARM	32KB L1D + 256KB L2 + prefetch for DSP only							
	ARM->DSP	32KB L1D + 256KB L2 + prefetch for DSP only							
DDR	DSP->DSP	Non-cacheable, non-prefetch	1086	1350	2580	7386	27006	107364	425430
		32KB L1D + prefetch	1074	1074	1260	1944	5268	17256	65418
		32KB L1D + 256KB L2 + prefetch	1104	1308	1428	2340	5736	18180	77514
	DSP->ARM	32KB L1D + 256KB L2 + prefetch for DSP							

Overwrite this text with the Lit. Number

		only							
	ARM->DSP	32KB L1D + 256KB L2 + prefetch for DSP only							

Table 6. Delay for inter-core data block (monolithic packet) exchange with hardware queue

Buffer location	Src->Dst	Cache Configuration	Minimal delay (Cycles) for different data size (Bytes)						
			16B	64B	256B	1KB	4KB	16KB	64KB
LL2 of Transmitter	DSP->DSP	Non-cacheable, non-prefetch	527	803	1809	5977	22655	89001	354609
		32KB L1D + prefetch	920	1014	1201	1988	5270	18338	64383
		32KB L1D + 256KB L2 + prefetch	987	1039	1242	1833	4211	13740	51453
	DSP->ARM	32KB L1D + 256KB L2							
LL2 of receiver		Non-cacheable, non-prefetch	789	712	911	1466	3791	13068	50049
		32KB L1D + prefetch	1073	993	1173	1848	4532	15403	34041
		32KB L1D + 256KB L2 + prefetch	1355	1386	1626	2388	5435	17676	66403
	ARM->DSP	32KB L1D + 256KB L2							
SL2	DSP->DSP	Non-cacheable, non-prefetch	395	467	793	2357	7803	32173	119287
		32KB L1D + prefetch	1020	964	1077	1699	4072	13459	51153
		32KB L1D + 256KB L2 + prefetch	1025	1045	1152	1782	4153	13775	39732
	DSP->ARM	32KB L1D + 256KB L2 + prefetch for DSP only							

	ARM->DSP	32KB L1D + 256KB L2 + prefetch for DSP only							
DDR	DSP->DSP	Non-cacheable, non-prefetch	426	796	2270	9076	36533	149068	598300
		32KB L1D + prefetch	1469	1810	3489	10472	38684	154074	613918
		32KB L1D + 256KB L2 + prefetch	1481	1925	3609	10582	38996	154674	616120
	DSP->ARM	32KB L1D + 256KB L2 + prefetch for DSP only							
	ARM->DSP	32KB L1D + 256KB L2 + prefetch for DSP only							

Table 7. Delay for inter-core data block (host packet) exchange with hardware queue

Buffer location	Src->Dst	Cache Configuration	Minimal delay (Cycles) for different data size (Bytes)						
			16B	64B	256B	1KB	4KB	16KB	64KB
LL2 of Transmitter	DSP->DSP	Non-cacheable, non-prefetch	388	772	1780	5860	22270	87490	348558
		32KB L1D + prefetch	925	1018	1183	1909	4916	16789	64399
		32KB L1D + 256KB L2 + prefetch	963	975	1142	1700	3768	12154	45317
	DSP->ARM	32KB L1D + 256KB L2							
LL2 of receiver		Non-cacheable, non-prefetch	913	968	1074	1658	4020	13396	50861
		32KB L1D + prefetch	1075	1074	1176	1545	3108	9352	34071
		32KB L1D + 256KB L2 + prefetch	1487	1467	1770	2578	5368	16889	62622

Overwrite this text with the Lit. Number

	ARM->DSP	32KB L1D + 256KB L2							
SL2	DSP->DSP	Non-cacheable, non-prefetch	438	520	888	2400	8374	32202	127754
		32KB L1D + prefetch	972	987	1042	1565	3287	10414	38867
		32KB L1D + 256KB L2 + prefetch	1061	1050	1158	1615	3438	10732	39694
	DSP->ARM	32KB L1D + 256KB L2 + prefetch for DSP only							
	ARM->DSP	32KB L1D + 256KB L2 + prefetch for DSP only							
DDR	DSP->DSP	Non-cacheable, non-prefetch	1038	1406	3158	10008	37411	149990	599300
		32KB L1D + prefetch	1671	2094	3830	10767	39062	154372	614346
		32KB L1D + 256KB L2 + prefetch	1840	2331	3949	10919	39345	155121	616483
	DSP->ARM	32KB L1D + 256KB L2 + prefetch for DSP only							
	ARM->DSP	32KB L1D + 256KB L2 + prefetch for DSP only							

Table 8. Delay for inter-core data block exchange with DMA

Method	Minimal delay (Cycles) for different data size (Bytes)						
	16B	64B	256B	1KB	4KB	16KB	64KB
EDMA3 (TPCC0)+ interrupt	1392	1422	1512	1842	3186	8568	29862
EDMA3 (TPCC1) + interrupt	1332	1356	1440	1776	3120	8496	29790
Packet DMA + interrupt							

(monolithic packet)							
Packet DMA + interrupt (host packet)							

The above data shows, for data sizes larger than 64 bytes, the cache is helpful. Generally speaking, the EDMA is a good choice for the inter-core data block exchange.

3 Summary

The Keystone device provides various ways for inter-core data exchange, each has its advantages and disadvantages, the system designer should choose the best one according to his application.

Generally speaking, for simple flag exchange, the IPC module is a good choice; for big data block exchange, the EDMA or packet DMA is a good choice.

4 Test data for 2K and 75K bytes cases

Delay for inter-core data block (monolithic packet) exchange with hardware queue + Qpend interrupt

Buffer location	Src->Dst	Cache Configuration	Minimal delay (Cycles) for different data size (Bytes)	
			2K	64K*
LL2 of Transmitter	DSP->DSP	Non-cacheable, non-prefetch	11838	354960
		32KB L1D + prefetch	3480	64934
		32KB L1D + 256KB L2 + prefetch	2988	51804
LL2 of receiver	DSP->DSP	Non-cacheable, non-prefetch	2607	50400
		32KB L1D + prefetch	3111	34392
		32KB L1D + 256KB L2 + prefetch	3915	66754
SL2	DSP->DSP	Non-cacheable, non-prefetch	4392	119638
		32KB L1D + prefetch	2814	51504
		32KB L1D + 256KB L2 + prefetch	2983	34083

Overwrite this text with the Lit. Number

DDR	DSP->DSP	Non-cacheable, non-prefetch	19589 *	598654
		32KB L1D + prefetch	20074	614269
		32KB L1D + 256KB L2 + prefetch	20417	616474

* The data with double Strikethrough should be confirm later

* The monolithic descriptor's maximum length is 65536 bytes, which means the maximum length of payload is 65536 - 12 bytes

Delay for inter-core data block (host packet) exchange with hardware queue + Qpend interrupt

Buffer location	Src->Dst	Cache Configuration	Minimal delay (Cycles) for different data size (Bytes)	
			2K	75K
LL2 of Transmitter	DSP->DSP	Non-cacheable, non-prefetch	11407	403853
		32KB L1D + prefetch	3250	75670
		32KB L1D + 256KB L2 + prefetch	2722	53411
LL2 of receiver	DSP->DSP	Non-cacheable, non-prefetch	2792 *	59850
		32KB L1D + prefetch	2438	40158
		32KB L1D + 256KB L2 + prefetch	2787	73602
SL2	DSP->DSP	Non-cacheable, non-prefetch	4742	150171
		32KB L1D + prefetch	2432	45712
		32KB L1D + 256KB L2 + prefetch	2546	46763
DDR	DSP->DSP	Non-cacheable, non-prefetch	19474	702525
		32KB L1D + prefetch	20429	720735
		32KB L1D + 256KB L2 + prefetch	20748	723274

* The data with double Strikethrough should be confirm later

Delay for inter-core data block exchange with EDMA3 + CP-INTC

Method	Minimal delay (Cycles) for different data size (Bytes)
--------	--

	2K	75K
EDMA3 (TPCC0) + interrupt	2298	33846
EDMA3 (TPCC1) + interrupt	2220	33660

Delay for inter-core data block exchange with shared memory buffer + IPC

Buffer location	Src->Dst	Cache Configuration	Minimal delay (Cycles) for different data size (Bytes)	
			2K	75K
LL2 of Transmitter	DSP->DSP	Non-cacheable, non-prefetch	11856	408918
		32KB L1D + prefetch	2916	70680
		32KB L1D + 256KB L2 + prefetch	2466	48786
LL2 of receiver	DSP->DSP	Non-cacheable, non-prefetch	2514	62100
		32KB L1D + prefetch	1710	29748
		32KB L1D + 256KB L2 + prefetch	3060	73212
SL2	DSP->DSP	Non-cacheable, non-prefetch	4884	117360
		32KB L1D + prefetch	1698	28026
		32KB L1D + 256KB L2 + prefetch	1746	28830
DDR	DSP->DSP	Non-cacheable, non-prefetch	19812	746652
		32KB L1D + prefetch	3972	110268
		32KB L1D + 256KB L2 + prefetch	4134	114882

References

1. *TMS320C66x DSP CorePac User's Guide (SPRUGW0)*
2. *Keystone Architecture Enhanced DMA (EDMA3) Controller User's Guide (sprugs5)*
3. *KeyStone Architecture Multicore Navigator User's Guide (sprugr9)*

Preliminary