Texas Instruments

Application Report

Lit. Number – June 2011

Performance of Multicore Navigator on KeyStone DSPs

Brighton Feng *Communication Infrastructure*

ABSTRACT

Multicore Navigator is a new architecture for high-speed data packet movement within KeyStone DSPs.

This application report discusses the performance of Multicore Navigator, provides measured performance data achieved under various operating conditions. Some factors affecting Multicore Navigator performance are discussed.

Contents

Figures

Tables

1 Introduction to Multicore Navigator

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.

The following figure shows the block diagram of the Multicore Navigator on KeyStone DSP family.

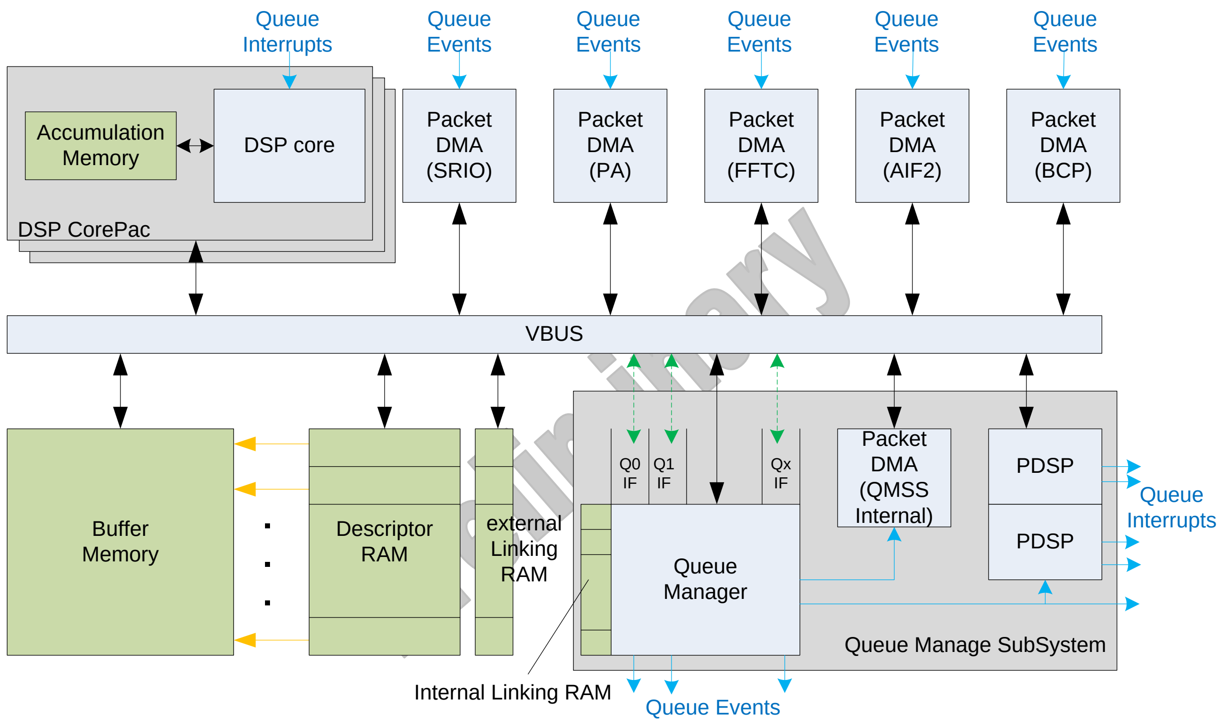

Figure 1. Multicore Navigator Architecture

Hardware queues are the base of the system. Queue manager manages all 8192 queues and provides services including pushing to and popping from all queues.

The key data structure the queue manager maintains is a linking table. Each linking entry occupies 64 bits, which represents the linking information of a packet descriptor, i.e. what is the next descriptor of current descriptor.

The typical activities of a queue PUSH operation are:

1. A master in the system writes the address of a descriptor to a register for a queue, which actually generates a PUSH request to the Queue Manager.
2. Queue Manager reads the tail pointer of the queue to get the last descriptor in the queue.
3. Queue Manager modifies the linking entry of the last descriptor; make it point to the new descriptor pushed in.

4. Queue Manager modifies the tail pointer of the queue; make it point to the new descriptor pushed in.
5. Queue manager modifies the linking entry of the new descriptor, make it NULL.

The typical activities of a queue POP operation are:

1. A master in the system reads a register for a queue, which actually generates a POP request to the Queue Manager.
2. Queue Manager reads the header pointer of the queue to get the first descriptor in the queue, the address of the first descriptor is returned to the master.
3. Queue Manager reads the linking entry of the first descriptor to find the second descriptor in the queue.
4. Queue Manager modifies the header pointer of the queue, make it point to the second descriptor in the queue, and then it becomes the first descriptor.

The queue manager subsystem includes an internal linking RAM for 16K entries. If your system has more than 16K descriptors, any memory including LL2 (Local Level 2 memory), SL2 (Shared Level 2 memory), DDR (Double Data Rate external memory) in the DSP system can be used as external linking RAM of Queue Manager to store other linking entries.

PUSH operation normally takes few cycles of a master, because it is normally a write operation of the master, normally, it does not stall master; While POP operation stalls the master because it is actually a read operation, and master need wait response from the queue manager.

To minimize the stall of DSP cores by POP operation, two embedded micro-controller (PDSP) is integrated into the QMSS (Queue Manager SubSystem). According to user’s configuration, the PDSP monitors one or more specific queues called accumulation queues, once there is a descriptor in the queue, PDSP pops it from the queue, and write the descriptor pointer to some scratch RAM, which is call accumulation buffer, normally it is in the local L2 of a DSP core. Each accumulation can be configured with specific accumulation buffer size. The PDSP generates an event for the accumulation queue when the buffer is full. This event can be routed to either DSP or other system masters to process those packets. Since all the descriptors have been accumulated to the scratch RAM close to the system master, it can significantly reduce the processing overhead. The masters can access the hardware queues include:

- All DSP cores
- All modules including Packet DMA
    - Queue Manager Subsystem
    - Serial RapidIO
    - Packet Accelerator
    - FFT Coprocessor (only available on some DSPs)
    - Antenna Interface 2 (only available on some DSPs)
    - Bit CoProcessor (only available on some DSPs)

The Packet DMA is a DMA (Direct Memory Access) engine designed for packet based data transfer. The traditional EDMA transfer request is defined in the PaRAM (Parameter RAM); while Packet DMA transfer request is defined in a descriptor, which can be popped from or pushed to hardware queues. Additionally, EDMA supports 3 dimensions data block and indexing, while Packet DMA only supports one dimension linear data block.

This application report discusses the performance of QMSS and Packet DMA, provides measured performance data achieved under various operating conditions. Some factors affecting Multicore Navigator performance are discussed.

Without special note, all data in this application report are measured on 1GHz C6678 EVM with 1333M 64-bit DDR3.

2 Performance of QMSS

The Key performance data of QMSS include the delay for PUSH, POP operations, and the delay of queue pending interrupt and descriptor accumulation interrupt.

2.1 DSP core cycles for PUSH operation

The performance of PUSH operation is tested with code like following pseudo code:

```
preTSC= TimeStampCount;
for(i=0; i< Number_of_Descriptors; i++)
{
        queueRegs->REG_D_Descriptor= uiDescriptor[i];
}
AverageCycles= (TimeStampCount - preTSC)/ Number_of_Descriptors;
```

The result measured on C6678 is shown in following table.

Table 1. Average DSP core cycles for different PUSH operation

number of descriptors	Push through different regions			
	Queue Manage Register	queue manage proxy register	Queue manage VBUSM space	Queue manage proxy VBUSM space
512	15	15	14	14
256	15	15	13	13
128	14	14	11	11
64	14	14	7	7
32	12	12	1	1
16	10	9	1	1
8	6	5	1	1
4	7	8	1	1
2	7	6	1	1
1	6	6	1	1

According to the test result, a PUSH operation takes about **1 to 15** DSP core cycles for different cases.

PUSH operation is actually write operation, DSP core may return after it posts the write data into write buffer. Before the write buffer full, the DSP core may not encounter any stall, which is the case when few descriptors are pushed; while if a lot of descriptors were pushed, after the write buffer full with several descriptors, the DSP core will be stalled until there is room in the write buffer for the later descriptor. The number of cycles be stalled is actually the time the queue manager handles one PUSH operation, according to above test result, it is about 15 cycles.

So, as a conclusion, if masters PUSH descriptors with an interval larger than 15 cycles, it will not see any stall, otherwise, it may be stalled 1 to 15 DSP core cycles.

2.2 DSP core cycles for POP operation

The performance of POP operation is tested with code like following pseudo code:

```
preTSC= TimeStampCount;
for(i=0; i< Number_of_Descriptors; i++)
{
        uiDescriptor[i]= queueRegs->REG_D_Descriptor;
}
AverageCycles= (TimeStampCount - preTSC)/ Number_of_Descriptors;
```

The result measured on C6678 is shown in following table.

Table 2. Average DSP core cycles for different POP operation

number of descriptors	Pop through different regions	
	Queue Manage Register	Queue manage VBUSM space
512	45	87
256	45	87
128	45	87
64	45	87
32	45	87
16	45	87
8	45	87
4	46	87
2	47	88
1	47	88

According to the test result, a POP operation takes at least **45** cycles. The cycles for POP are much larger than PUSH because POP is actually a read operation. The master must wait until the read data returned from queue manager.

If there are multiple descriptors to be popped, accumulator may be used to accumulate multiple descriptors from queue manager into DSP core's local memory, then DSP core can read descriptors in its local memory, it only takes about 5 cycles to read a descriptor in LL2, which saves at lease 40 cycles for each descriptor read.

2.3 Access queue through different access regions

The queue manager provides four regions or windows for masters to access a queue, they are:

1. Normal registers through VBUSP configuration bus (only for DSP core)
2. Proxy registers through VBUSP configuration bus (only for DSP core PUSH)
3. Normal data space through VBUSM data bus
4. Proxy data space through VBUSM data bus (only for PUSH)

They are mapped to different address, from master's point view, to access through different region is actually access different address.

Proxy region is used for PUSH only. It is used for PUSH operations with additional information such as packet size. For this case, the master need write the additional information to the proxy region first, and then write the descriptor pointer to the proxy region lastly. The proxy ensures that all these writes are atomic, i.e. may not be interrupted by other masters, which eliminates the need for locking mechanisms in the device.

If only descriptor need be pushed to queue, i.e. only one write operation, both normal region and proxy region can be used.

According to above test result, pushing to proxy region takes same cycles as pushing to normal region.

VBUSP bus is much different from VBUSM bus. DSP core can use both VBUSP and VBUSM, while all other Packet DMAs only use VBUSM.

According to above test result, pushing to the VBUSM address is faster than the VBUSP address due to the deeper write buffer (and no stalls before the buffer full), but popping is faster via the VBUSP address due to the higher VBUSM latency for read operations.

If DSP core need push to a queue which will be popped by a Packet DMA, the DSP core should use VBUSM region for the pushing to avoid potential race condition. One of the possible scenarios is,

1. DSP core writes data to packet buffer in DDR3 through VBUSM,
2. DSP core pushes descriptor to PktDMA through VBUSP
3. PktDMA reads the descriptor
4. PktDMA reads data from DDR3

Since descriptor and payload data are go through different VBUS, it is possible that PktDMA reads the descriptor before the payload data lands the DDR3 memory. If the descriptor is also pushed through VBUSM, then this issue can be avoided.

The other way to avoid race condition is to use MFENCE instruction to make sure the data is landed before pushing the descriptors to the QM. Refer to “TMS320C66x CPU and Instruction Set Reference Guide (sprugh7)” for more details about MFENCE instruction.

Without special note, all tests for this application report use VBUSM region for PUSH, and use VBUSP region for POP for DSP

2.4 Using external linking RAM

The queue manager subsystem includes internal linking RAM for 16K entries. If your system has more than 16K descriptors, any memory including LL2 (Local Level 2 memory), SL2 (Shared Level 2 memory), DDR (Double Data Rate external memory) in the DSP system can be used as external linking RAM of Queue Manager to store other linking entries.

Queue manager access linking entry in external linking RAM consumes more cycle than access its internal linking RAM. Generally speaking, DDR is very inefficient to be used as external linking RAM because linking entry access is normally discrete; to use LL2 as external linking RAM should achieve good performance, but LL2 is relative small. So, SL2 is a good trade-off. In the test for this application report, SL2 is used as external linking RAM for test.

Following table compares the average cycles consumed to PUSH/POP descriptors with external linking RAM and internal linking RAM.

Table 3. Average PUSH/POP cycles with External linking RAM v.s. internal linking RAM

number of descriptors	PUSH		POP	
	Internal linking RAM	External linking RAM	Internal linking RAM	External linking RAM
512	14	14	45	100
256	13	13	45	100
128	11	11	45	99
64	7	7	45	99
32	1	1	45	98
16	1	1	45	96
8	1	1	45	92
4	1	1	46	86
2	1	1	47	74
1	1	1	50	48

According to the test result, we can not see any effect of external linking RAM for PUSH operation since it is post operation; while external linking RAM increases average cycles for POP operation about 50 cycles. As mentioned in above sections, accumulation can also be used to save these additional cycles for POP operation.

Without special note, all results in this application report are tested with internal linking RAM.

2.5 Delay for queue pending interrupt

Queue manager monitors some queues, if any of these queues is not empty, a queue pending interrupt is sent to DSP core. The delay for queue pending interrupt is measured with following pseudo code:

```
    ……
    preTSC= TimeStampCount;
    queueRegs->REG_D_Descriptor= uiDescriptor;  //push to an empty queue
    asm(" IDLE");    //wait for the queue pending interrupt
    delay= intTSC - preTSC;
    ……
interrupt void QueuePendISR()        //queue pending Interrupt Service Routine
{
    intTSC= TimeStampCount;          //save the Time Stamp Count when the interrupt happens
    ……
}
```

This delay measured on C6678 is about **130** DSP core cycles.

2.6 Delay for descriptor accumulation interrupt

To save the cycles of DSP cores for POP operation, a small programmable controller (PDSP) is integrated into the QMSS. According to user's configuration, the PDSP monitors one or more specific queues, once there is a descriptor in the queue, PDSP pops it from the queue, and writes the pointer of the descriptor to a buffer, which is call accumulation buffer; normally it is in the local L2 of a DSP core. When PDSP accumulates a specific number of descriptors to the accumulation buffer, it triggers an interrupt to the DSP core, and then the DSP core reads the descriptors in its local L2 instead of from Queue Manager, which saves about 40 cycles of DSP core for each descriptor read according to the analysis in above section.

The features of PDSP are different with different firmware. The Acc48 firmware monitors up to 32 queues with high priority, and monitor up to 512 (16x32) queues with low priority. The Acc32 firmware monitors up to 32 queues. The Acc16 firmware monitors up to 512 (16x32) queues.

The delay between queue push operation and accumulation completion interrupt depends on how many queues the PDSP monitors, and how busy these queues are.

For the simplest case, that is, PDSP only monitors one queue, and accumulates one descriptor to DSP core's LL2, the average delay measured on C6678 is show in following table.

Table 4. Average accumulation delay

Firmware		Cycles
Acc 48	High priority channel	2953
	Low priority channel	7875
Acc 32		2841
Acc 16		1862

Though the delay is relative large, the DSP core is free to do anything else during this delay period. So, this method is good for transfer large quantity of packets which can tolerate high latency. For packet transfer with very tight timing, polling method or queue pending interrupt should be used instead of accumulation.

2.7 Delay for descriptor reclamation

Normally, to return a used descriptor to a free descriptor queue, DSP core software need

1, parse the “return queue number”, “return policy” and “return push policy” fields in the descriptor.

2, push the descriptor to the specified return queue.

To simplify these operations, PDSP provides a feature, reclamation queue, to save DSP core cycles by the first step above.

With the reclamation queue, DSP core software just pushes any used descriptor into the reclamation queue; the PDSP monitors any incoming descriptor in the reclamation queue, and return them to proper free descriptor queue according to the “return queue number”, “return policy” and “return push policy” fields in the descriptor.

Since reclamation is done by PDSP firmware, the delay between pushing a used descriptor to reclamation queue and getting the descriptor in the free descriptor queue depends on how busy the PDSP is.

For the simplest case, that is, PDSP only does reclamation, the delay measured on C6678 is about **900** cycles.

So, this method is good for recycling large quantity of descriptors with relative flexible timing requirement. If you want a descriptor to be recycled immediately, you should use normal method.

2.8 Other performance considerations for queue operation

DSP core write/read full descriptor content may consume many cycles. For most applications, DSP core may initialize all descriptors during initialization, and only write/read few fields (such as packet size) of the descriptor during run time.

Since host packet has separate descriptor and packet buffer, they are normally not continuous in memory. Access descriptor and packet buffer separately introduces extra overhead, especially when the descriptor or packet buffer is in cacheable memory space. So, host packet provides more flexibility, but monolithic packet may achieve better memory access performance. Designer should carefully consider these facts to choose between host packet and monolithic packet.

If descriptor is in cacheable SL2 or DDR2 memory space, software need maintain cache coherency for it. Host packet descriptor is relative small, normally, 32~64 bytes, it may not fully utilize 64 bytes L1D cache line or 128 bytes L2 cache line, and it is relative costly to maintain cache coherency for small descriptor. So, for small descriptors, you should try to put them in LL2 to avoid these cache related penalty.

By default, we get a descriptor from the head of a queue, and return a descriptor to the tail of a queue. But we can change the "Return Push Policy" field of a descriptor to make it return to the head of a queue. This policy makes recently used descriptor be reused more quickly; this may utilize cache better for some cases.

3 Performance of Packet DMA

The performance of Packet DMA engines is measured with loopback mode (infrastructure mode), i.e. the TX packet is looped back to RX side.

3.1 Packet DMA Transfer overhead

In this application report, Packet DMA transfer overhead is defined as the time to transfer minimum data element (1 word), it is measured as the time between pushing packet to the TX queue and getting the packet in the RX queue.

Following table shows the average measured Packet DMA transfer overhead on C6678.

Table 5. Packet DMA Transfer overhead

Packet DMA in	**Average overhead**
QMSS	550
PASS	715
SRIO	545

Transfer overhead is a big concern for short transfers and need to be included when scheduling DMA traffic in a system. Single-element transfer performance is overhead-dominated. So, for small transfer inside DSP, you should make the trade off between using DMA and using DSP core.

3.2 Packet DMA bandwidth

Following figure shows throughput measured on single Packet DMA channel in QMSS for transferring 4 bytes to 8K bytes.

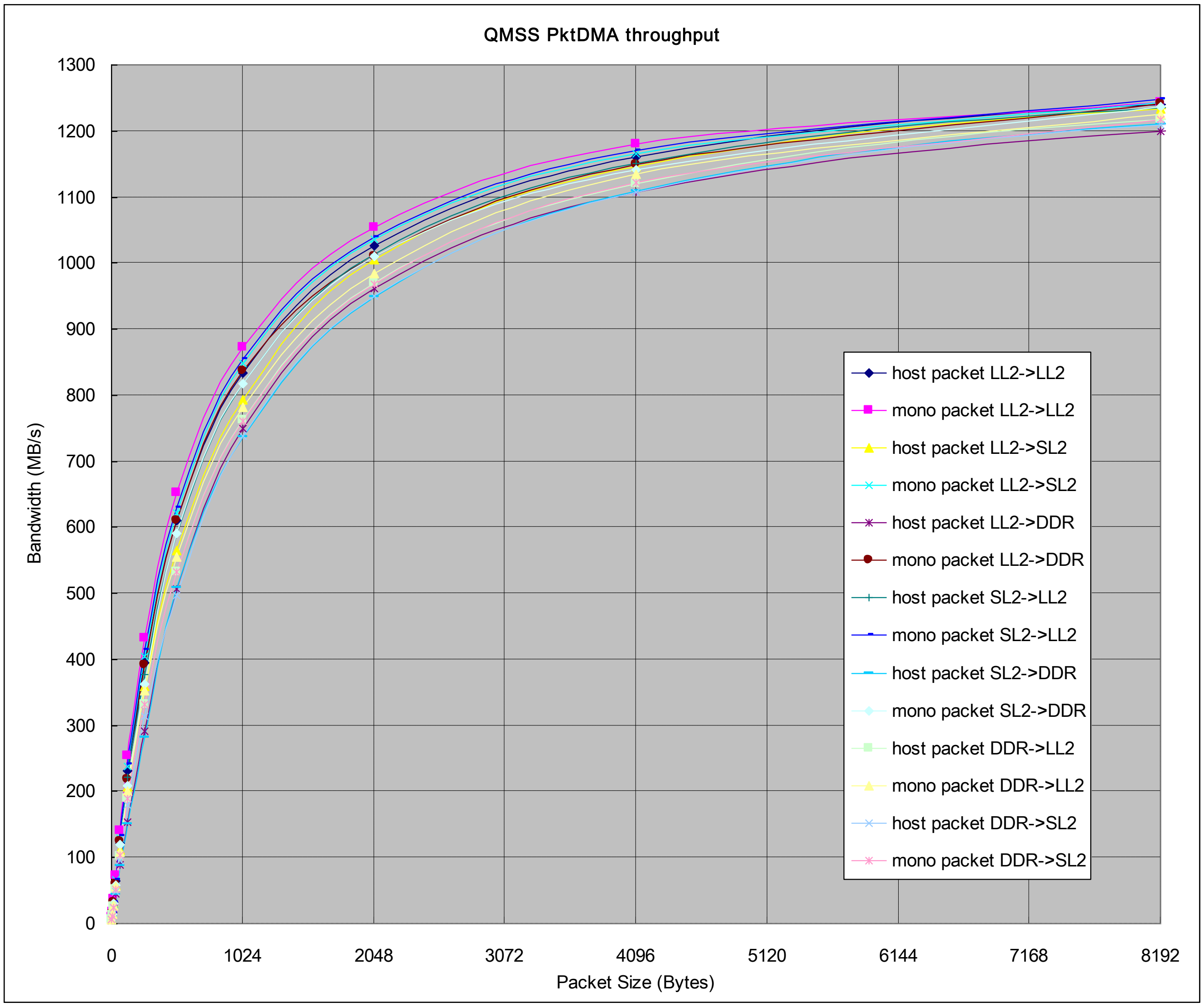

Figure 2. Throughput of single packet DMA channel in QMSS

Since overhead is fixed, the bigger the packet size, the more bandwidth can be utilized.

The different source or destination affects the throughput a little bit. Packet DMA accesses LL2 a little bit faster than SL2; and SL2 a little bit faster than DDR.

The packet type affects the throughput a little bit, Packet DMA achieves a little bit higher throughput for monolithic packet transfer.

The Packet DMA has a full-duplex, 128-bit, 1/3 DSP core speed bus. The theoretical bandwidth on 1GHz DSP is 128/8*1000/3= 5333MB/s. Above figure shows much less bandwidth because only one channel is used for the tests.

The Packet DMA has a priority-based scheduler to manage the bandwidth among multiple channels. Since the Packet DMA needs to continuously fetch data and descriptor information for multiple channels to construct packets, one channel can not fully utilize the bandwidth of the Packet DMA (that is, the bus will be idle during Packet DMA reads as there can only be one pending transaction per channel). If multiple channels are used then multiple pending transactions can exist, making more use of the bandwidth capabilities. The following figure shows the total bandwidth achieved with multiple channels.

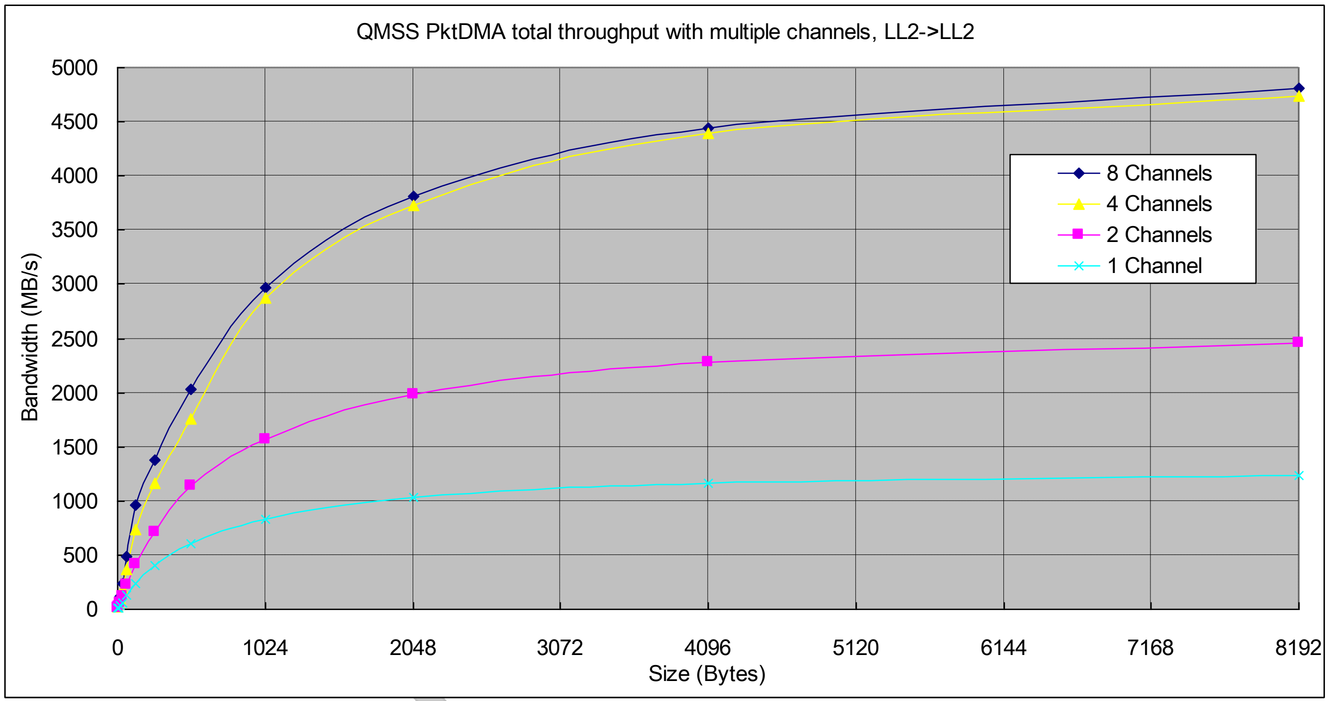

Figure 3. Total Throughput of multiple packet DMA channels in QMSS

Above figure shows 4 channels can fully utilize the bandwidth of QMSS Packet DMA bus. For more than 4 channels, the total throughput is limited by the bus.

3.3 Packet DMA v.s. EDMA

There are multiple Packet DMA engines on each KeyStone DSP; these packet DMA engines are not exactly same.

Following figure compares the throughput of different Packet DMA channel to an EDMA channel for data transfer between two cores' LL2 memory.

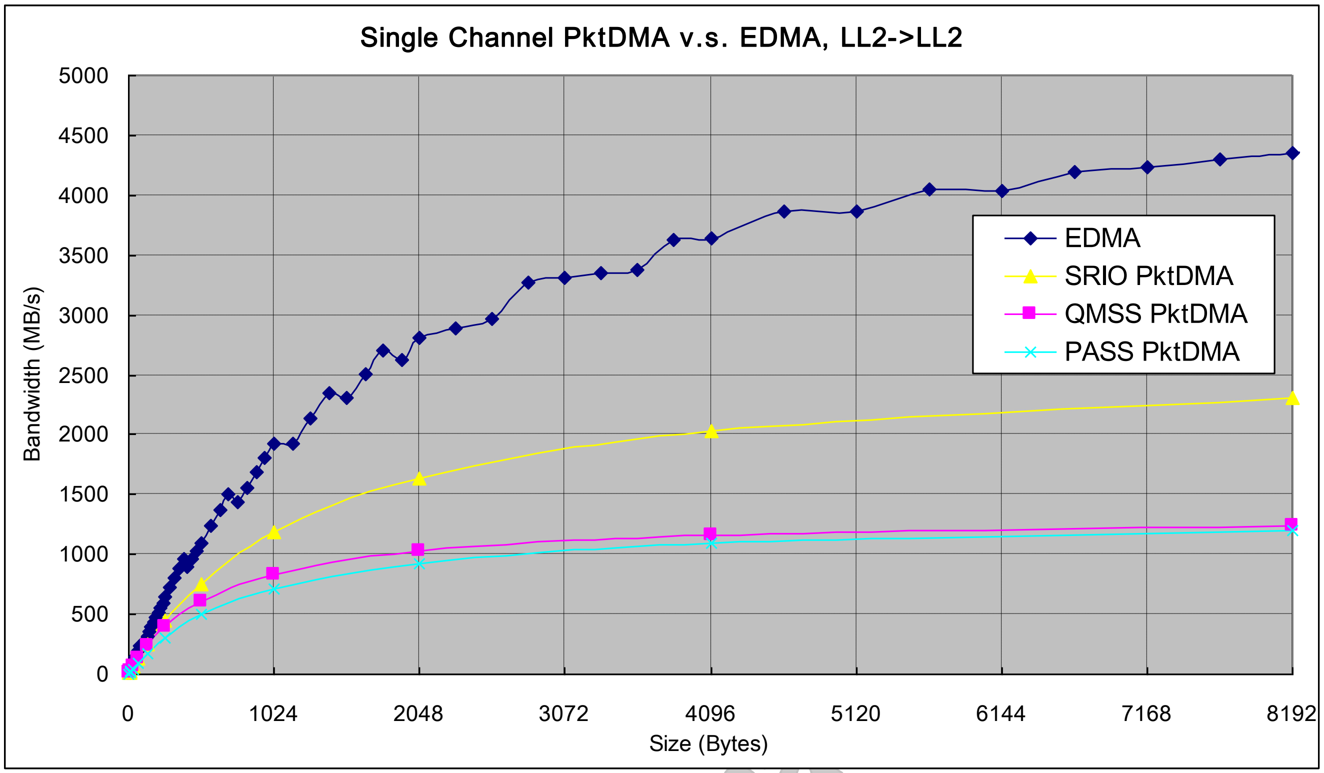

Figure 4. Single Channel Packet DMA v.s. EDMA

The figure shows the throughput of a QMSS packet DMA channel is similar as the PASS (Packet Accelerator SubSystem) packet DMA channel; they are almost half of the throughput of a SRIO packet DMA channel. The throughput of those packet DMA channels are much less than an EDMA channel, which can fully utilize the bus bandwidth with one channel.

As a conclusion, Packet DMA provides convenient features for packet transfer; its throughput meets the requirement of most applications. Single packet DMA channel does not fully utilize the bandwidth of memory and bus system. For data transfer who requires huge bandwidth through single channel, EDMA is a good choice.

References

1. *KeyStone Architecture Multicore Navigator User Guide* (SPRUGR9)
2. *TMS320C66x DSP CorePac User Guide* (SPRUGW0)
3. *KeyStone Architecture Enhanced Direct Memory Access (EDMA3) Controller User Guide (*SPRUGS5*)*
4. *TMS320C6678 data manual (*SPRS691*)*
5. *TMS320C66x CPU and Instruction Set Reference Guide* (sprugh7)