

---

## Implementing a Queue Manager in Traffic Management Systems

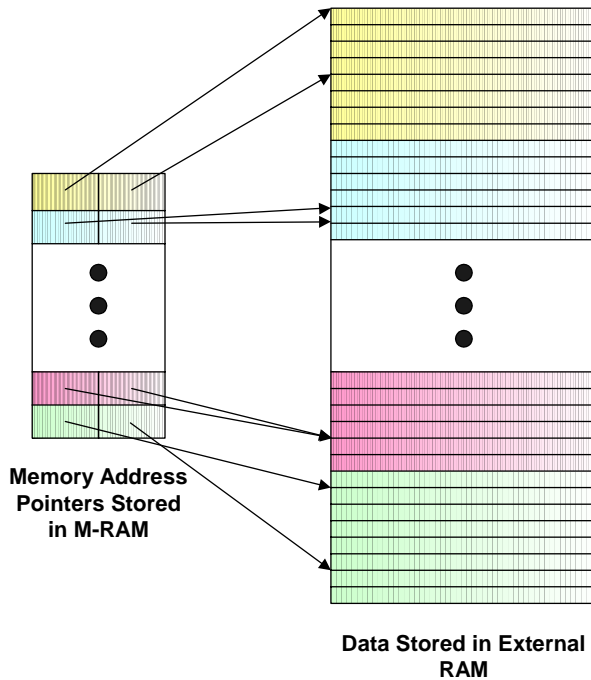
You can implement a queue manager for traffic management within Stratix™ II, Stratix, and Stratix GX devices using the M-RAM memory structure and an external memory component. The large amount of memory available in the M-RAM blocks (512K bits in each block) makes it ideal for performing address mapping to external memories. The flexibility of programmable logic allows you to define the number and depth of queues or multi-queues for the queue manager, and interface to external memory to store the actual data. The control logic and addresses required to segment the external memory into a multi-queue architecture are implemented in the Stratix II, Stratix, and Stratix GX device.

This white paper describes how you can use Stratix II, Stratix, and Stratix GX M-RAM blocks to implement a queue management block for traffic managers. The system is able to provide quality of service (QoS) by prioritizing the traffic based on rules and constraints enforced by the policer and scheduler. It is not the scope of this white paper to discuss queuing algorithms, or how to implement other parts of a traffic management system such as the policer and scheduler. An overview on how to construct the queue manager in Stratix II, Stratix, and Stratix GX devices as well as some challenges and solutions will be discussed.

### *Overview*

To implement the queue manager, the internal M-RAM memory block must be address mapped to the external memory. You can dynamically map addresses by creating a linked-list structure in hardware. An alternative is to statically allocate memory by dividing the external memory into fixed-sized, sub-memory blocks. There are advantages and disadvantages to both approaches. The dynamic approach is more flexible and allows for a better utilization of memory. The static approach does not incur the overhead of a linked-list structure, allowing simpler handling of status signals. This white paper describes the static memory allocation approach. Figure 1 shows an example of a statically allocated memory implementation.

Figure 1. Static Memory Allocation



Each queue has a single entry in the M-RAM block to describe the queue with the following information (see Table 1):

Table 1. Queue Entries

Status Flags	Head Pointer (Read)	Tail Pointer (Write)
--------------	---------------------	----------------------

The status flags can contain the empty, full, almost empty, and almost full flags for each queue. The head pointer stores the address location for the next read for the queue in the external memory. The tail pointer stores the address location for the next write for the queue in the external memory. Depending on the depth of the queue required, you can segment the external memory into sub-memory blocks, which are controlled by each entry in the M-RAM block representing a single queue and first-in first-out (FIFO) buffer.

For example, with an address width of 25 bits, you can configure the M-RAM block in 8k×64-bit wide mode. The 64 bits include two 25-bit addresses and additional status flag bits. This configuration could manage up to 8,000 queues. Larger queue managers can be built by merging multiple M-RAM blocks together. The depth of the queues is determined by the size of the external memory. For example, a read or write process to a multi-queue FIFO buffer has the following memory configuration specifications:

- Sixty four queues
- Frame size of 64 bytes
- Queue depth of 128 frames

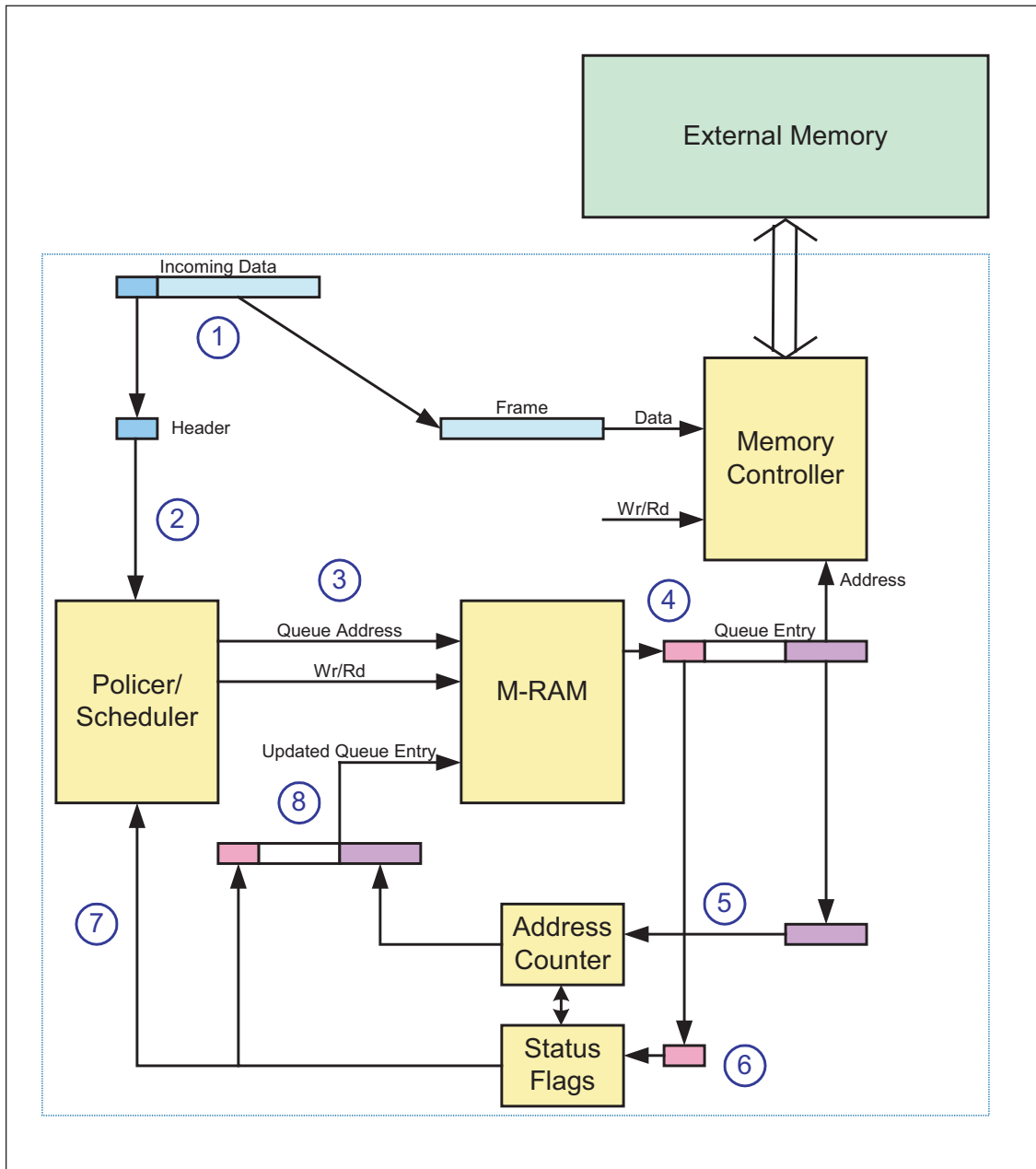
This example requires each queue to be 8,192 bytes (64 bytes×128 frames), and the entire memory to be 524,288 bytes (8,192 bytes×64 queues) or 4 Mbits. The first queue allocates the memory from 0 to 8,191 bytes, the second queue from 8,192 to 16,384 bytes, and so on. When the pointers reach the upper limit of the allocated memory section, they loop back to the lower limit. The M-RAM block is configured in an 8k×64 mode to store a 64-bit wide queue entry.

### *Read & Write Operations*

When a packet arrives at the queue, the policer/scheduler determines in which queue (i.e., queue 0 to 63) the packet must be stored. If a write to queue 3 was requested, the M-RAM block accesses the queue entry at the address location 3. After the first clock cycle, the tail pointer is masked out and sent to the external memory controller along with the frame to be stored. The tail pointer is incremented by one frame size and operations are performed on the head and tail pointers to update the status flags. The updated pointers and status flag bits are written back into the M-RAM on the next cycle. The same process occurs for a read. The updated status flag bits for queue 3 are also sent to the queue manager for processing. If the last write to queue 3 causes the queue to become full, a read operation for that queue can be executed before servicing other queues.

Status signal generation and processing occurs immediately after a read or write request, resulting from the embedding of the status signals into the queue entry. The alternative would be to register individual status signals for each queue. If only empty and full flags are required with 8,000 queues, that would require 16k registers, which is not efficient. Instead, a simple arithmetic operation is performed where the head pointer value is subtracted from the absolute value of the tail pointer. If the difference is zero, the queue is empty. If the difference is equal to the maximum depth of the queue, the queue is full. The queue manager must control the pointers such that the head (read) pointer never leads the tail (write) pointer. The queue manager monitors the queues when they become full and empty. If the queue is empty, the queue manager ignores all reads from the external memory for that queue. Other intermediate status signals such as almost full and almost empty flags can be generated. See Figures 2 and 3.

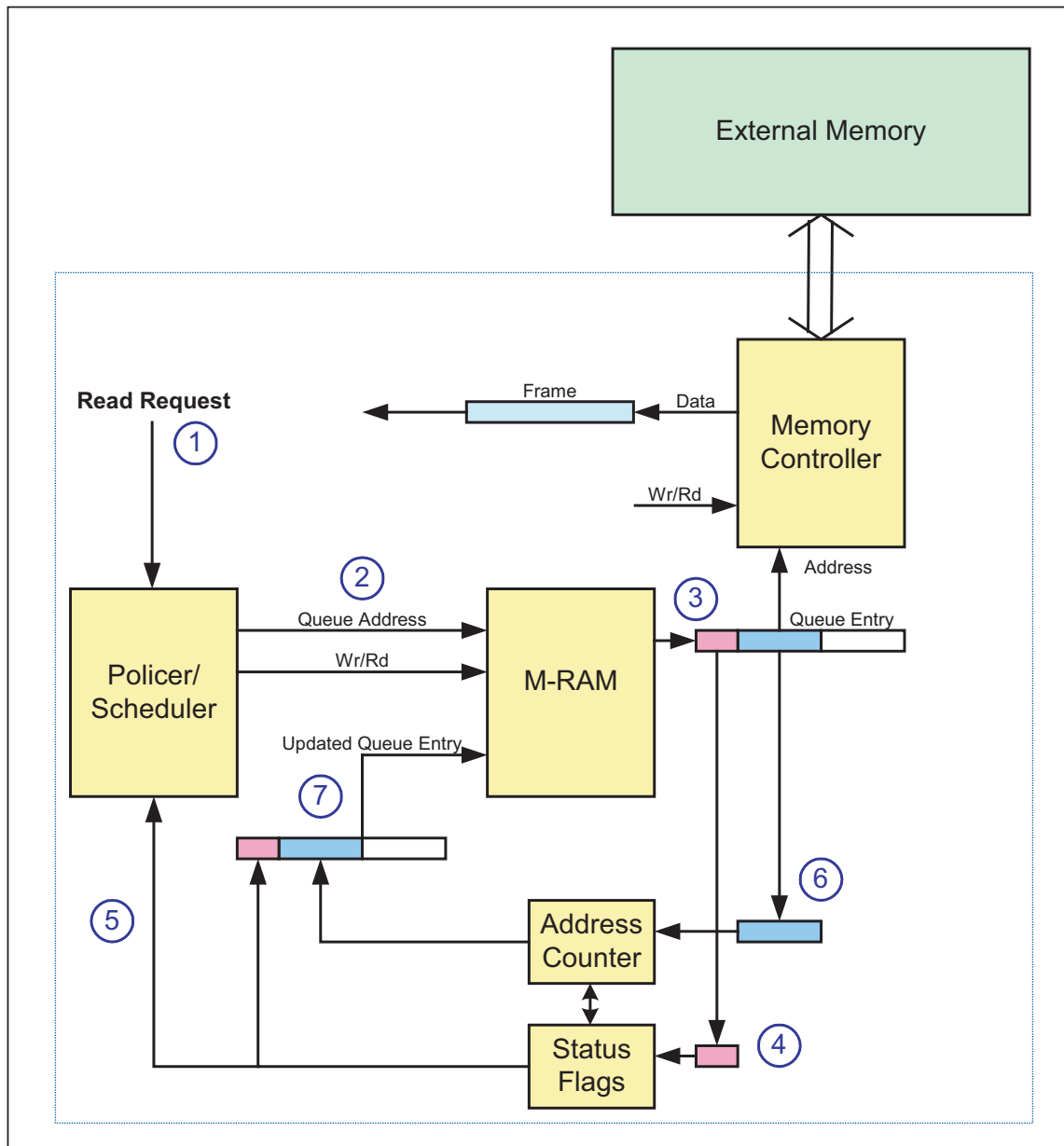
Figure 2. Write Operation



Notes to Figure 2:

- (1) Incoming data from traffic. Mask out header to policer/scheduler and frame to the memory controller.
- (2) Policer/scheduler parses the header information and determines which queue to place the frame.
- (3) Sends a read request to the M-RAM block and the queue address to access.
- (4) Mask out the tail pointer which contains the address in the external memory and sends it to the memory controller.
- (5) Sends the tail pointer to the address counter to increment to the next memory location.
- (6) Calculates the appropriate status flags for queue with the pointer information.
- (7) Checks the status flags to determine if immediate action is required, (i.e., queue is full).
- (8) Rebuilds the queue entry and writes into the M-RAM block.

Figure 3. Read Operation

**Notes to Figure 3:**

- (1) Reads the request to policer/scheduler or from within policer/scheduler.
- (2) Policer/scheduler sends the appropriate queue address to read from the M-RAM block.
- (3) The status flag is masked out and the head pointer is sent to the memory controller.
- (4) Calculates the appropriate status flags for queue with the pointer information.
- (5) Checks the status flags to determine if immediate action is required (i.e., queue is empty). If the queue is empty, a read from the external memory is not required.
- (6) Sends the head pointer to the address counter to increment to the next memory location.
- (7) Rebuilds the queue entry and write into the M-RAM block.

*Static Memory Allocation*

For statically allocated memory, the M-RAM block must be initialized with the sub-memory block’s starting addresses for each queue before start up. The M-RAM block requires an initialization circuit to write the starting addresses for each queue, which is performed by using a state machine and a counter. The counter increments by the depth of each queue. Once the M-RAM block initializes, the state machine sends a flag to the queue manager. Alternatively, an external look-up table (LUT) can be used to initialize the M-RAM block. The external LUT has the starting address for each queue, which is read into the M-RAM block to initialize the queue manager.

For determining the memory space for each queue (see Table 2), it is advantageous to allocate memory space of  $2^x$  for each queue, where the memory space is divisible by the frame size. This process simplifies the pointer address operations, as the counter can just increment by the frame size. When the counter reaches the upper memory space limit, it will automatically roll over to 0 or the lower limit. For example, if the frame size is 64 bytes ( $2^6$ ) and the depth is 128 frames, each memory space is 8,192 bytes ( $2^{13}$ ). The address can then be broken up into two parts: the static most significant bit (MSB) portion which denotes a specific queue; and a dynamic least significant bit (LSB) portion which changes as that specific queue is filled.

*Table 2. Memory Space*

<b>Static Queue Identifier</b>	<b>Dynamic Frame Queue Counter/Pointer</b>
0000000000000	0000000000000
0000000000001	0000000000000
...	...
...	...
1111111111110	0000000000000
1111111111111	0000000000000

The upper MSB remains the same for a specific queue, only the lower LSB changes by the address counter. In this manner, efficiencies are gained by keeping the address counter operation small and uniform for all queues.

The alternative would be to have a special function for each queue to handle the pointers once it reached its upper limit. An LUT would have to be used to reset the pointer to the lower limit.

### External Memory

The amount of external memory required depends on the application governed by the number of queues required, the depth of the queues, and the throughput of the system. The type of external memory is governed by the cost and performance. Table 3 describes different external memories used in traffic management systems:

Table 3. Exernal Memory Used in Traffic Management

	DRAM	SRAM
<b>Latency</b>	High	Low
<b>Density</b>	High	Low
<b>Cost</b>	Low	High
<b>Power</b>	Low	Medium
<b>Applications</b>	Packet Buffer	Pointers, Flow Tables, and Rate Tables

Given the various options, and the typically large amounts of memory required to buffer packets in systems, SDRAM is typically the best choice. Due to the high latencies associated with SDRAM memories, the effective throughput of the system is lower when compared to other types of memories.

### Performance

The performance of the system is determined by the effective throughput of the external memory device and the queue manager. The latency in writing to and reading from the external memory dictates the overall system performance as it tends to be slower than the internal M-RAM operations. For the queue manager, the throughput requirement will be 2× the line rate. The data is written into and read out of the FIFO buffer before a new frame is processed. The header operation and queue entry update is pipelined, which has no impact on the performance. This coupled with the latency through the external memory determines the effective throughput the queue manager is able to achieve.

The following example shows a typical performance calculation to determine if a DRAM interface with a 64-bit wide bus at 200 MHz can meet the bandwidth requirements of an OC-48 application (with a line rate of 2.4 gigabits per second (Gbps)). Given a typical Internet protocol (IP) packet of 40 bytes plus a 20-byte header, the DRAM interface requires eight cycles to read or write the 60 bytes of data (i.e., 8 cycles×8 bytes (64 bits) = 64 bytes).

For write into DRAM, the bandwidth required is  $2.4 \text{ Gbps} \times 60 \text{ bytes} / 40 \text{ bytes} = 3.6 \text{ Gbps}$  (since the 20 bytes of header is overhead that needs to be written). For read from a DRAM, the bandwidth required is generally 1.5× to 2× the write bandwidth, depending on the fabric interface. Assuming 1.5×, the required read bandwidth is  $2.4 \text{ Gbps (no header)} \times 1.5 = 3.6 \text{ Gbps}$ . The total required bandwidth is the addition of the write and read bandwidth which totals 7.2 Gbps.

The maximum bus utilization possible in the DRAM is  $64 \text{ bits} \times 200 \text{ MHz} = 12.8 \text{ Gbps}$ . When transitioning from a write to a read, the number of cycles lost is three assuming a CAS latency of three. Now the bus utilization becomes  $8 \text{ cycles} / (8 \text{ cycles} + 3 \text{ cycles CAS latency}) \times$  the maximum bus utilization  $= 8/11 \times 12.8 \text{ Gbps} = 9.3 \text{ Gbps}$ . 15% of the bus utilization generally is consumed by refreshing and other overhead. Therefore, the actual available bandwidth is  $9.3 \text{ Gbps} \times 85\% \approx 8 \text{ Gbps}$ . Since the realized bandwidth (8 Gbps) > the required bandwidth (7.2 Gbps), a 64-bit DRAM interface operating at 200 MHz can meet the bandwidth requirements of an OC-48 application.

## Conclusion

The large amount of internal memory available in Stratix II, Stratix, and Stratix GX M-RAM block structures make them an ideal candidate for performing queue management functions for traffic managers. M-RAM blocks allow external memory address pointers to be stored inside the FPGA, which partitions the external memory into a large multi-queue FIFO buffer. By implementing a queue manager, you can realize better memory utilization and provide QoS capability in traffic management systems.



101 Innovation Drive  
San Jose, CA 95134  
(408) 544-7000  
[www.altera.com](http://www.altera.com)

Copyright © 2004 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries.\* All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.