Messaging

HMM-148-0

Messaging Components

A message is a packet of information that one PE sends to another PE. After receiving a message, the support circuitry in the PE stores the message in a designated location in local memory called the message queue. When the message queue contains more than a specified amount of messages (this number is specified by software), the support circuitry interrupts the microprocessor (if software enables the interrupt). The microprocessor then reads the messages from the message queue.

Messaging uses several hardware and software components. These components include: the message queue, the message queue control word (MQCW), the software-maintained head pointer, the SEND E-register command, the interrupt status (IR_STATUS) register, and the message address (MSG_ADDR) register.

The following subsections provide brief descriptions of these components.

Message Queue

A message queue is a portion of local memory that software designates as the destination for messages. Software may define one, several, or many individual message queues for each PE.

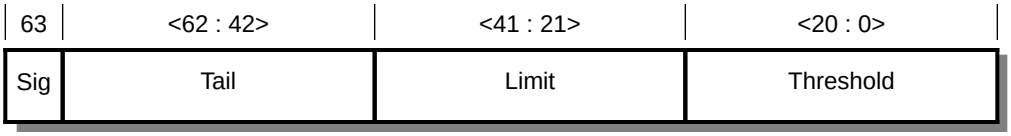
Software also defines the size of each message queue. A message queue may store from 1 to 2,097,151 messages. Each message contains 64 bytes of information. Each 64-byte address location in the message queue is called a slot.

Message Queue Control Word

The message queue control word (MQCW) is a 64-bit word of information that software stores in bits <63 : 0> of the first slot in a message queue. The support circuitry and software use the information stored in the MQCW to perform messaging functions.

Figure 1 shows the format of the MQCW. The MQCW contains four fields: a limit field, a tail field, a threshold field, and a signal (Sig) bit. All of the fields except the signal bit are offset values with respect to the address of the MQCW.

Figure 1. MQCW Bit Format



The limit field indicates the number of slots in the message queue. For example, when the limit field is set to 50, the message queue contains 50 slots for 50 messages. Software may set the limit field to any value between 1 and 2,097,151.

The tail field indicates the slot in the message queue that will receive the next message. When the support circuitry receives a message, it stores the message in the slot that is indicated by the tail field of the MQCW. After storing the message in the slot, the support circuitry increments the tail field by 1.

The threshold field indicates which slot will cause the support circuitry to set an interrupt for the microprocessor. For example, when the threshold field is set to 30, the support circuitry sets an interrupt for the microprocessor when the tail field is equal to 30.

The signal bit indicates that the tail field is equal to the threshold value. Although the support circuitry sets the signal bit to 1, only software can reset the signal bit to 0 by writing a 0 to the MQCW.

NOTE: Hardware modifies the MQCW in an atomic read-modify-write operation. Because of this characteristic, software manipulation of the MQCW does not conflict with hardware updates of the MQCW.

Head Pointer

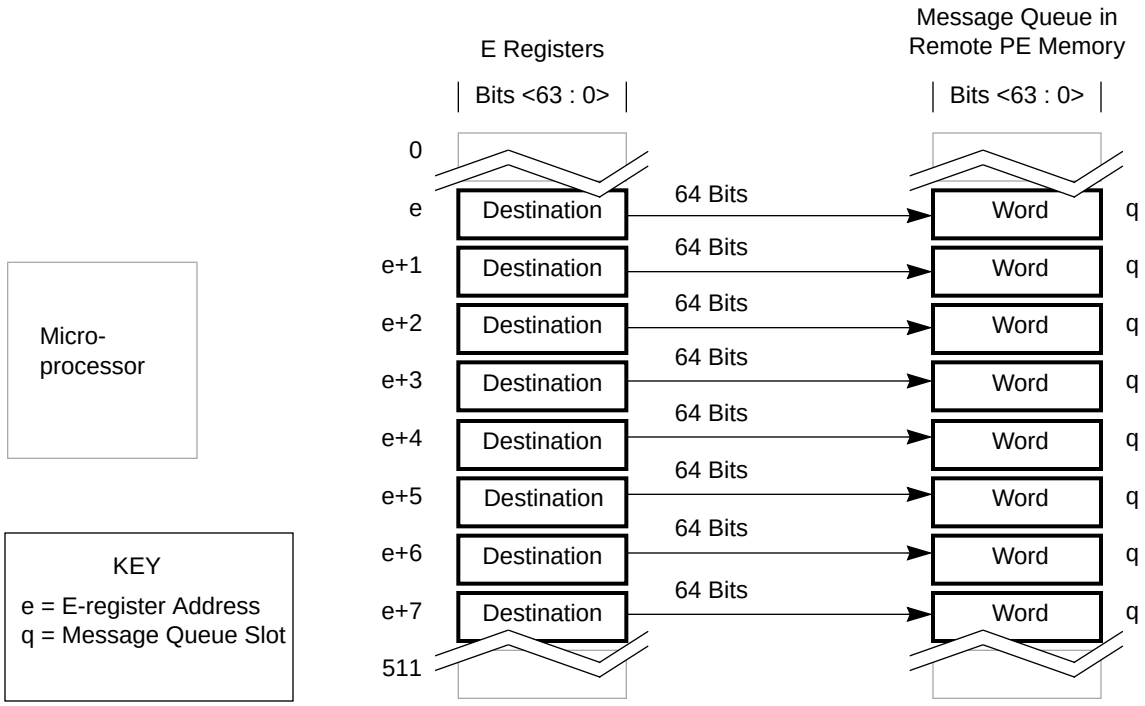
The head pointer is a software-maintained value that indicates the slot of the message queue that software will read next. System hardware does not use, modify, or provide storage for the head pointer.

After reading a message from the message queue, software increments the head pointer to point to the next slot in the message queue. When the head pointer points to the same slot as the tail pointer, software has read all of the messages out of the message queue.

SEND E-register Command

The message send (SEND) E-register command signals the support circuitry to transfer a 64-byte (8-word) message from 8 contiguous source E registers to a 64-byte slot in a destination PE message queue (refer to Figure 2). The *CRAY T3E System Commands* document provides detailed information on the SEND E-register command.

Figure 2. SEND E-register Command



IR_STATUS Register

The interrupt status (IR_STATUS) register contains two bits that are used for messaging: bit 32 (MSG_ADDR_VALID bit) and bit 33 (MSG_MULTIPLE bit).

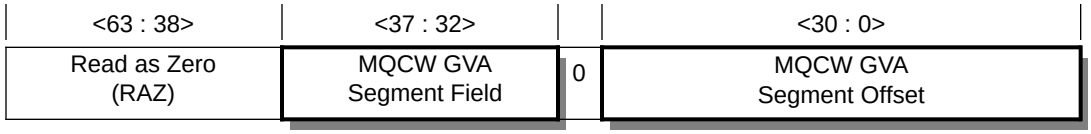
The message address valid (MSG_ADDR_VALID) bit of the IR_STATUS register indicates that the tail value for a message queue is the same as the threshold value for that message queue. [The message queue is identified by the message address (MSG_ADDR) register.] The MSG_ADDR_VALID bit remains set to 1 until software resets it. Software resets this bit by writing a 1 to the MSG_ADDR_VALID bit of the IR_STATUS register.

The multiple messages (MSG_MULTIPLE) bit indicates that the tail value is equal to the threshold value in more than one message queue. The MSG_MULTIPLE bit remains set to 1 until software resets the bit. Software resets this bit by writing a 1 to the MSG_MULTIPLE bit of the IR_STATUS register.

MSG_ADDR Register

The message address (MSG_ADDR) register contains the global virtual address (GVA) of the MQCW for the message queue that set the MSG_ADDR_VALID bit of the IR_STATUS register to 1 (refer to Figure 3). The GVA in the MSG_ADDR register remains locked in the register until software resets the MSG_ADDR_VALID bit of the IR_STATUS register. Then, the support circuitry stores a new GVA into the register the next time the support circuitry sets the MSG_ADDR_VALID bit to 1.

Figure 3. MSG_ADDR Register Format

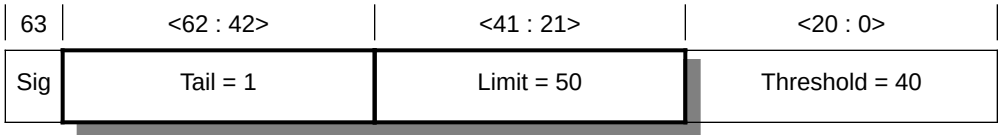


Defining a Message Queue

Before a PE can receive messages, software must define at least one message queue for the PE. To do this, software stores a MQCW to a location in local memory. As an example, the following paragraphs describe sample MQCW field values.

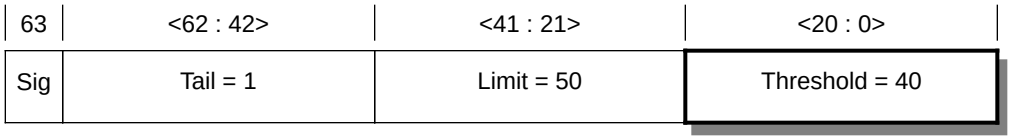
To define a message queue that stores as many as 50 messages, software sets the limit field of the MQCW to 50 (refer to Figure 4). Software also sets the tail value to a number greater than 0. Normally, software initially sets the tail value to 1. If software set the tail value to 0, the first message would overwrite the MQCW, because the tail value is an offset with respect to the MQCW address.

Figure 4. Sample Tail and Limit Values



When software defines a message queue, it may set the threshold value to any number greater than 0. For the following example, software set the threshold value to slot 40 (refer to Figure 5). The threshold value must be at least one slot larger than the tail value to generate an interrupt.

Figure 5. Sample Threshold Value



Sending a Message

Before sending a message, software stores data for the 64-byte message in 8 contiguous E registers that align on a 64-byte boundary in the E-register address space. To do this, software may use STORE E-register commands, which transfer data from the microprocessor to the E registers.

After storing data for the message in the 8 source E registers, software issues a SEND E-register command. The SEND command signals the support circuitry to send the message to the destination PE.

SEND Command Address Arguments

Table 1 describes the address arguments for the SEND command. The "SEND" subsection in the CRAY T3E system *Commands Booklet* provides more information on the format of the address arguments for a SEND command.

Table 1. SEND E-register Command Address Arguments

Argument	Description
Opcode	Indicates to the support circuitry that the E-register command is a SEND. Also indicates whether the SEND command references the segment translation table (STT) and whether the ordering of multiple SEND commands to the same address is fixed.
Source E Register Reference	Indicates to the support circuitry which E registers are the 8 contiguous source E registers for the message.
Global Index	A software-formatted value that indicates to the support circuitry which PE is the destination PE and provides the support circuitry with the index offset for the MQCW.
MO Pointer	Indicates to the support circuitry which E registers are the more operands (MO) block of E registers for the SEND command.
Centrifuge Mask	Defines the format of the global index.
Base Virtual Address	The support circuitry adds the base virtual address to the index offset to generate a user virtual address or a global virtual address (GVA).

NOTE: The "E-register Global-memory Commands" subsection in the CRAY T3E system *Addressing* document provides more information on the global index, index offset, MO block, centrifuge mask, base virtual address, user virtual address, and global virtual address.

GVA Values

When software sets the opcode to indicate that the SEND command will reference the segment translation table (STT), the support circuitry adds the index offset to the base virtual address to create a user virtual address. The support circuitry then references the STT to convert the user virtual address into a GVA. When it sends a message, software sets these address arguments so that the GVA is the GVA for the MQCW in the destination PE.

When software sets the opcode to indicate that the SEND command will not reference the STT, the support circuitry adds the index offset to the base virtual address to create the GVA. When it sends a message, software sets these address arguments so that the GVA is the GVA for the MQCW in the destination PE.

ERS Codes

After it receives all of the SEND command address arguments from the microprocessor, the support circuitry sets the E-register state (ERS) codes for the 8 contiguous source E registers to EMPTY (refer to Table 2). The support circuitry and network interface then create a message packet and send the packet to the destination PE.

The support circuitry does not change the value of the ERS codes again until it receives a SEND-accepted or a SEND-rejected response packet. When the support circuitry receives a SEND-accepted packet, it sets the ERS codes to FULL, which indicates that the message was accepted by the destination PE. When the support circuitry receives a SEND-rejected packet, it sets the ERS codes to FULL_SR, which indicates that the message queue was full and the message was not accepted by the destination PE (when this occurs, software may want to resend the message).

Table 2. ERS Codes

Bit 1	Bit 0	Name	Description
0	0	EMPTY	The E register is empty.
0	1	FULL_F	The E register is full; however, a deferred software error was detected during the transfer.
1	0	FULL	The E register is full and no errors occurred during the transfer.
1	1	FULL_SR	The E register is full; however, a previous message send (SEND) command that used the E register was rejected.

Receiving and Storing a Message

When the support circuitry in a PE receives a message, the support circuitry references a location in local memory that the message packet identifies as the MQCW location. After reading the MQCW, the support circuitry compares the tail value of the MQCW to the limit value. When the tail value is equal to the limit value, the support circuitry and network interface create a SEND-rejected packet and return it to the PE that created the message. The SEND-rejected packet indicates that the message queue is full and cannot accept another message.

When the tail value is less than the limit value, the support circuitry returns a SEND-accepted packet to the PE that created the message. The support circuitry then stores the message in the message queue at the slot pointed to by the tail pointer in the MQCW. The support circuitry then increments the tail pointer and updates the MQCW in memory.

The following subsections describe scenarios in which the support circuitry accepts a message and stores the message in the message queue. The scenarios describe activity under three conditions:

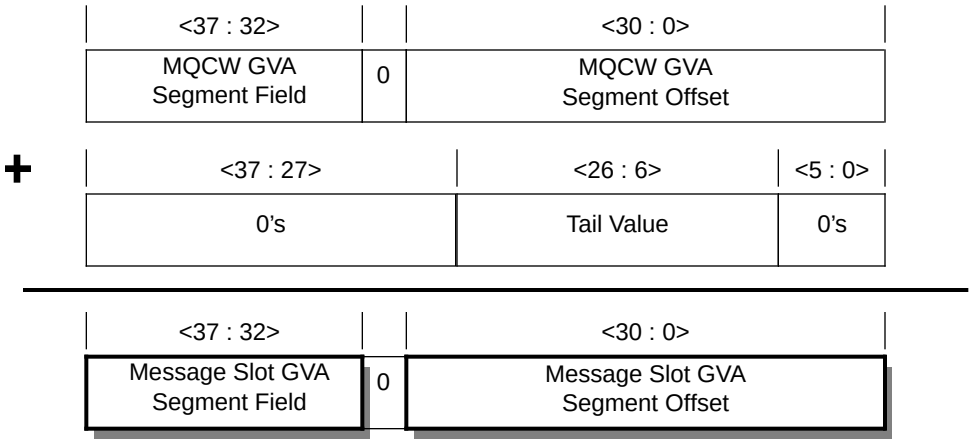
- The tail value is less than the threshold value
- The tail value is equal to the threshold value
- The tail value is greater than the threshold value

Tail Value < Threshold Value

The following steps describe the process that the support circuitry performs when the tail value is less than the threshold value. In this example, the limit of the message queue is 4, the tail is 1, and the threshold is 3.

1. The support circuitry shifts the tail value 6 bit positions to the left and adds this value to the GVA of the MQCW. The sum is the byte-addressed GVA for the destination slot in the message queue (refer to Figure 6).

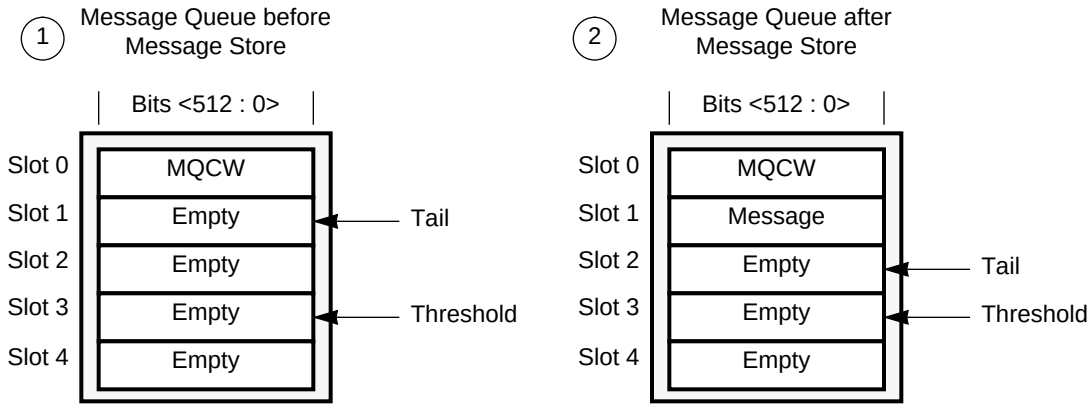
Figure 6. Generating the Message Slot GVA



2. After adding the MQCW GVA to the shifted tail value, the support circuitry sets bits <5 : 0> of the resulting address to 0's. This creates a 64-byte aligned GVA for the destination slot in the message queue.

3. The support circuitry stores the message in the slot that is identified by the message slot GVA and increments the tail value by 1 (refer to Figure 7).

Figure 7. Tail Value Less Than the Threshold Value



4. The support circuitry compares the tail value of the MQCW to the threshold value. For this example, the tail value is less than the threshold value. When this occurs, the support circuitry does not set a message interrupt for the microprocessor.

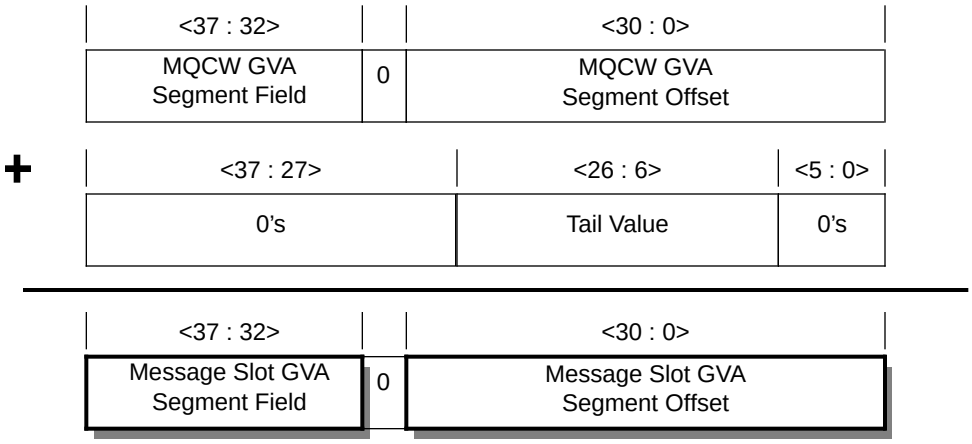
5. The message store operation is complete.

Tail Value = Threshold Value

The following steps describe the process that the support circuitry performs when the tail value is incremented and is equal to the threshold value. In this example, the limit of the message queue is 4, the tail is 2, and the threshold is 3.

1. The support circuitry shifts the tail value 6 bit positions to the left and adds this value to the GVA of the MQCW. The sum is the GVA for the destination slot in the message queue (refer to Figure 8).

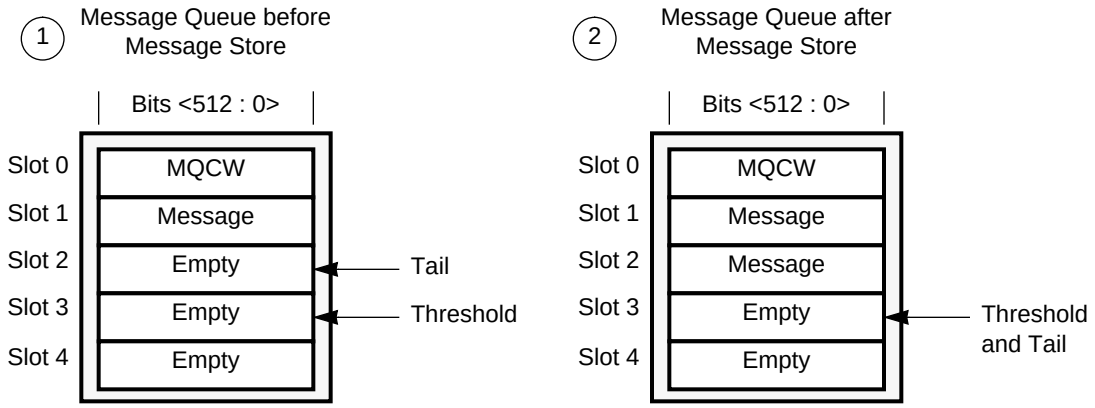
Figure 8. Generating the Message Slot GVA



2. After adding the MQCW GVA to the shifted tail value, the support circuitry sets bits <5 : 0> of the resulting address to 0's. This creates a 64-byte aligned GVA for the destination slot in the message queue.

3. The support circuitry stores the message in the slot that is identified by the message slot GVA and increments the tail value by 1 (refer to Figure 9).

Figure 9. Tail Value Equal to the Threshold Value



4. The support circuitry compares the tail value of the MQCW to the threshold value. For this example, the tail value is equal to the threshold value.

5. The support circuitry sets the signal bit (bit 63) of the MQCW for the message queue to 1. This bit indicates that the tail value is equal to the threshold value.

6. The support circuitry examines the value of the MSG_ADDR_VALID bit of the IR_STATUS register.

7. When the MSG_ADDR_VALID bit is set to 0, the support circuitry sets the MSG_ADDR_VALID bit to 1, stores the GVA for the MQCW in the MSG_ADDR register, and locks the MSG_ADDR register. If software enables the interrupt, the MSG_ADDR_VALID =1 condition sets an interrupt for the microprocessor.

8. When the MSG_ADDR_VALID bit is set to 1, the support circuitry checks the value of the MSG_MULTIPLE bit of the IR_STATUS register. If the MSG_MULTIPLE bit is set to 0, the support circuitry sets this bit to 1.

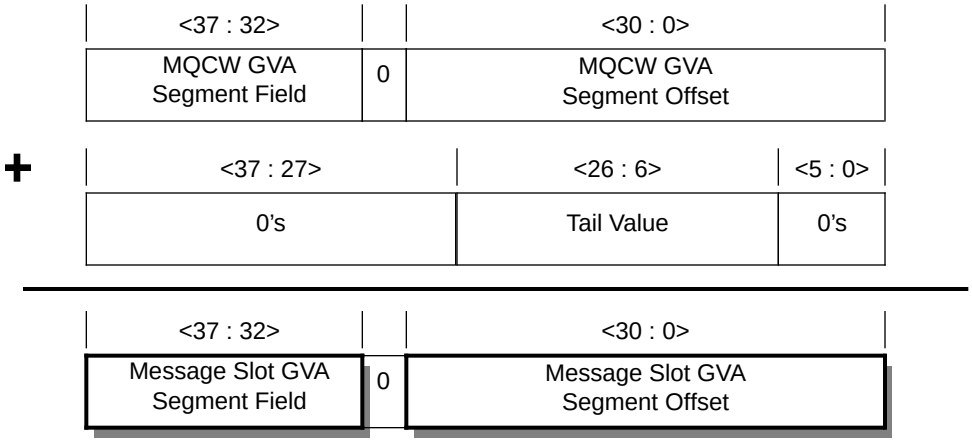
9. The message store operation is complete.

Tail Value > Threshold Value

The following steps describe the process that the support circuitry performs when the tail value is greater than the threshold value. In this example, the limit of the message queue is 4, the tail is 3, and the threshold is 2.

1. The support circuitry shifts the tail value 6 bit positions to the left and adds this value to the GVA of the MQCW. The sum is the GVA for the destination slot in the message queue (refer to Figure 10).

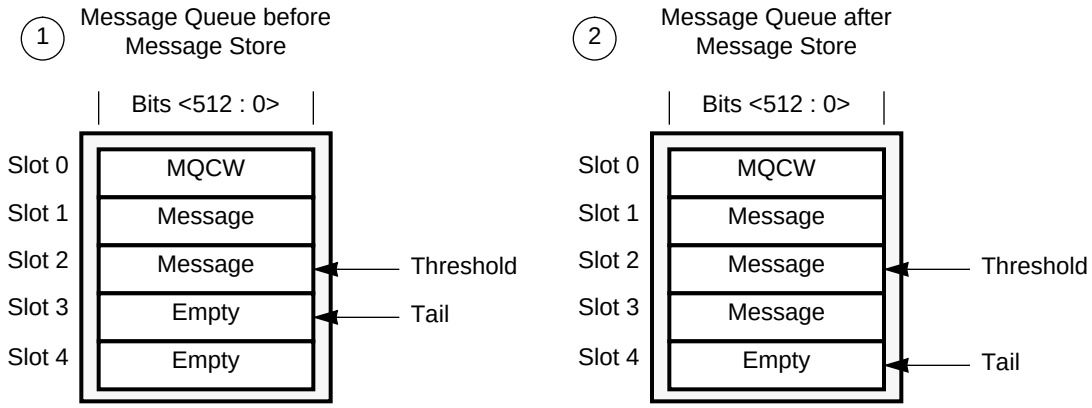
Figure 10. Generating the Message Slot GVA



2. After adding the MQCW GVA to the shifted tail value, the support circuitry sets bits <5 : 0> of the resulting address to 0's. This creates a 64-byte aligned GVA for the destination slot in the message queue.

3. The support circuitry stores the message in the slot that is identified by the message slot GVA and increments the tail value by 1 (refer to Figure 11).

Figure 11. Tail Value Greater Than the Threshold Value



4. The support circuitry compares the tail value of the MQCW to the threshold value. For this example, the tail value is greater than the threshold value. When this occurs, the support circuitry does not set a message interrupt for the microprocessor.

5. The support circuitry does not modify the signal bit (bit 63) of the MQCW. Unless it has been reset by software, the signal bit (bit 63) of the MQCW for the message queue remains set to 1. If the signal bit of the MQCW is reset by software, the bit remains set to 0.

6. The support circuitry does not modify the MSG_ADDR_VALID bit or the MSG_MULTIPLE bit of the IR_STATUS register. Unless they are reset by software, the MSG_ADDR_VALID bit and/or the MSG_MULTIPLE bit of the IR_STATUS register remain set to 1. If these bits have been reset by software, these bits remain set to 0.

7. The message store operation is complete.

Reading a Message

The following subsections show scenarios of how software may read messages out of the message queues. These samples are for illustration only and may not reflect the actual methods used by software. In these examples, software enables the MSG_ADDR_VALID and MSG_MULTIPLE interrupts in the IR_STATUS register. (Software may also poll the MQCW tail pointer to determine when messages have arrived.)

When software enables the MSG_ADDR_VALID and MSG_MULTIPLE interrupts in the IR_STATUS register, three scenarios may exist when the microprocessor receives an interrupt from the support circuitry.

- The MSG_ADDR_VALID bit is 1 and the MSG_MULTIPLE bit is 0.
- The MSG_ADDR_VALID bit is 1 and the MSG_MULTIPLE bit is 1.
- The MSG_ADDR_VALID bit is 0 and the MSG_MULTIPLE bit is 1.

The following subsections describe these scenarios.

MSG_ADDR_VALID = 1 and MSG_MULTIPLE = 0

The following steps describe a process that software may use to read messages out of a message queue. In this example, the user software defined a message queue with the limit set to 30, the threshold set to 25, and the tail set to 1.

1. The microprocessor receives an interrupt from the support circuitry.

2. The operating system (OS) software running in the microprocessor reads the value of the IR_STATUS register. For this example, the MSG_ADDR_VALID interrupt is set to 1 and the MSG_MULTIPLE interrupt is set to 0.

3. The OS software reads the value stored in the MSG_ADDR register. This value is the GVA of the message queue that caused the support circuitry to set the MSG_ADDR_VALID interrupt.

4. The OS software resets the MSG_ADDR_VALID interrupt to 0. To do this, software writes a 1 to the MSG_ADDR_VALID bit of the IR_STATUS register.

5. The OS software signals the user software that a message queue contains messages and provides the user software with the GVA of the MQCW.

6. The user software performs an atomic swap (SWAP) command on the MQCW. The swaperand value for the SWAP command contains new MQCW values that are set to a limit of 60, a threshold of 55, a tail of 31, and a signal bit of 0.

7. The user software retrieves the value of the software-maintained head pointer for the message queue. The head pointer indicates the slot location of the next message to be read from the message queue.

8. The user software sends a command to the support circuitry to read the message that is referenced by the head pointer. The command may be any command that accesses local memory. For example, the command may be a direct local-memory load command or a get vector (GETV) E-register command.

9. The user software processes the message.

10. The user software increments the value of the head pointer by one slot and compares the value of the head pointer to the tail pointer. When the head pointer is less than the tail pointer, the user software repeats steps 8 through 10. When the head pointer is equal to the tail pointer, the user software has read all of the messages out of the message queue.

The next time the microprocessor receives an interrupt because this message queue exceeded the threshold value, the OS and user software repeat steps 1 through 10; however, in Step 6, the user software sets the new MQCW values to a limit of 30, a threshold of 25, a tail of 1, and a signal bit of 0. This process creates two virtual message queues that use the same MQCW.

MSG_ADDR_VALID = 1 and MSG_MULTIPLE = 1

The following steps describe a process that software may use to read messages out of a message queue. In this example, the user software defined multiple message queues and each message queue is initially set to a limit of 30 and a threshold of 25.

1. The microprocessor receives an interrupt from the support circuitry.

2. The OS software running in the microprocessor reads the value of the IR_STATUS register. For this example, the MSG_ADDR_VALID interrupt is set to 1 and the MSG_MULTIPLE interrupt is set to 1.

3. The OS software reads the value stored in the MSG_ADDR register. This value is the GVA of the first message queue that caused the support circuitry to set the message interrupt.

4. The OS software resets the MSG_ADDR_VALID and MSG_MULTIPLE interrupts to 0. To do this, software writes a 1 to the MSG_ADDR_VALID and MSG_MULITPLE bits of the IR_STATUS register.

5. The OS software signals the user software that multiple message queues contain messages and provides the user software with the GVA of the first MQCW.

   NOTE: For the first MQCW, the signal bit is set to 1 in the next step.

6. The user software reads the MQCW from memory and examines the signal bit. When the signal bit is set to 0, software does not read messages from that message queue, but instead retrieves the GVA for the MQCW of another message queue. When the signal bit is set to 1, software continues with the following steps.

7. The user software performs an atomic swap (SWAP) command on the MQCW. The swaperand value for the SWAP command contains new MQCW values that are set to a limit of 60, a threshold of 55, a tail of 31, and a signal bit of 0.

8. The user software retrieves the value of the software-maintained head pointer for this message queue. The head pointer indicates the slot location of the next message to be read from the message queue.

9. The user software sends a command to the support circuitry to read the message that is referenced by the head pointer. The command may be any command that accesses local memory. For example, the command may be a direct local-memory load command or a get vector (GETV) E-register command.

10. The user software processes the message.

11. The user software increments the value of the head pointer by one slot and compares the value of the head pointer to the tail pointer. When the head pointer is less than the tail pointer, software repeats Steps 9 through 11. When the head pointer is equal to the tail pointer, user software has read all of the messages out of this message queue.

12. The user software repeats steps 6 through 12 for each message queue that it defined.

MSG_ADDR_VALID = 0 and MSG_MULTIPLE = 1

When software enables the MSG_ADDR_VALID and MSG_MULTIPLE interrupts in the IR_STATUS register, a possible result is that the MSG_ADDR_VALID bit is set to 0 and the MSG_MULTIPLE bit is set to 1.

If this scenario occurs, OS and user software running in the microprocessor perform the same functions listed in the steps under “MSG_ADDR_VALID=1 and MSG_MULTIPLE=1” in this document; however, the MSG_ADDR register does not contain a valid GVA. Because of this characteristic, the OS software does not perform Step 3, and Step 5 is replaced with the following step.

5. The OS software signals the user software that multiple message queues contain messages.

Index