```
xVersion" 2011.11.5.1"

module queues

public

  #Document queues
  //
  //  Generic queue (fifo) support
  //
  //    Version 2, 2008-03-05
  //
  //
  //  When including this file in make.4th and using this module by means of
  //  the uses directive (uses queues endUses)) Queues can be defined in the
  //  following way :
  //
  //    [[queue]] QueueName QueueLength // ( -- )
  //
  //  QueueLength must be a power of 2 and can't exceed 64 k
  //
  //
  //  Some X4th words will be auto defined (but be only compiled if actually
  //  used) for each timer :
  //
  //  Constants and variables, the variables are grouped and will all end up in
  //  one RAM bank.
  //
  //    QueueLength Constant QueueNameLength  // Holds the queue's length.
  //    QueueLength Array QueueName           // Holds the queue's data.
  //    Variable QueueNameFreeCount           // Holds the queue's number of
  //                                          //   free locations.
  //    Variable QueueNameFillCount           // Holds the queue's number of
  //                                          //   filled locations.
  //    Variable QueueNameInPtr               // Holds the queue's data put
  //                                          //   pointer.
  //    Variable QueueNameOutPtr              // Holds the queue's data get
  //                                          // pointer.
  //    Variable QueueNameOverflows           // Current number of queue overflows
  //    Variable QueueNameUnderflows          // Current number of queue underflows
  //    Variable QueueNameTmp                 // A temporary storage location.
  //
  //
  //  Interface, as a set of colon definitions
  //
  //    QueueName.hasData     (   -- b )      // Returns the number of bytes
  //                                          //   present in the queue.
  //    QueueName.hasRoom     (   -- b )      // Returns the number of bytes
  //                                          //   free in the queue.
  //    QueueName.read        (   -- b )      // Reads a byte from the queue,
  //                                          //   undefined if no data
  //                                          //   available.
  //    QueueName.write       ( b --   )      // Writes a byte on the queue, or
  //                                          //   dumps it when the queue is
  //                                          //   full.
  //    QueueName.overflows   (   -- b )      // Returns the number of overflows
  //                                          //   since the last time checked
  //    QueueName.underflows  (   -- b )      // Returns the number of underflows
  //                                          //   since the last time checked
  //    QueueName.clear       (   --   )      // Flushes the queue's contents
  //
  //
  //  Interrupt interface, as a set of interrupt callable colon definitions
  //
  //  These words take a value from or deliver it to wreg.
  //
  //    QueueNameIntRead      (   --   )      // a read to be used from
  //                                          //   interrupts.
  //    QueueNameIntWrite     (   --   )      // a write to be used from
  //                                          //   interrupts.
  //
  //  Before the interrupt handler can use the above words it should check for
  //  data to be available or free space to be available in the queue by using
  //  the variables :
  //
  //    Variable QueueNameFreeCount           // Holds the queue's number of
  //                                          //   free locations.
  //    Variable QueueNameFillCount           // Holds the queue's number of
  //                                          //   filled locations.
  #endDoc


  { // Start meta compiler


  // ///////////////////////////////////////////////////////////////////////
  // Support macro's for queues, all meta words

  macro queue.hasData ( aName ) ( -- bCount )
    PushReg <@" $aName$">FillCount
  endMacro
  #doc
  //  [[Queues]] Get number of data bytes available
  #endDoc


  // ////////////////////////////////////////////////////////////////////////

  macro queue.hasRoom ( aName ) ( -- bCount )
    PushReg <@" $aName$">FreeCount
  endMacro
  #doc
  //  [[Queues]] Get number of free entries
  #endDoc


  // ////////////////////////////////////////////////////////////////////////

  macro queue.read ( aName ) ( -- bData )

    PushNothing                                 ; // Make room for result

    mov     #<@" $aName$">FillCount, w3         ; // Point w3 to fill count
    dec     [ w3], [ w3]                        ; // Claim data, one less filled
    bra     n, <@" $aName$">_underflow          ; // B/ no data, queue underflow

    mov     #<@" $aName$">, w1                  ; // Set up buffer pointer in w1
    mov     <@" $aName$">OutPtr, w2

    clr     w0                                  ; // Clear upper byte
    mov.b   [ w1 + w2], w0                      ; // get byte

    inc     w2, w2                              ; // Data read, advance out ptr
    and     #<@" $aName$">Length - 1, w2        ; // Wrap at length
    mov     w2, <@" $aName$">OutPtr             ; // write back to memory

    inc     <@" $aName$">FreeCount              ; // One more free
    bra     <@" $aName$">_rd_done               ; // B/ done

<@" $aName$">_underflow:

    inc     [ w3], [ w3]                        ; // Unclaim data
    inc     <@" $aName$">Underflows             ; // Increment underflow count

    ; // Fall through to done

<@" $aName$">_rd_done:
  endMacro
  #doc
  //  [[Queues]] Read first byte from queue.
  //  When no bytes are available ([[queue.HasData]] returns 0) the result
  //  is undefined and the underflow count is incremented by one.
  #endDoc


  // ////////////////////////////////////////////////////////////////////////

  macro queue.write ( aName ) ( bData -- )

    mov     #<@" $aName$">FreeCount, w3         ; // Point to free count
    dec     [ w3], [ w3]                        ; // Claim room, one less free
    bra     n, <@" $aName$">_overflow           ; // B/ no room, queue overflow

    mov     #<@" $aName$">, w1                  ; // Set up pointer
    mov     <@" $aName$">InPtr, w2

    mov.b   w0, [ w1 + w2]                      ; // Store byte

    inc     w2, w2                              ; // Advance in ptr
    and     #<@" $aName$">Length - 1, w2
    mov     w2, <@" $aName$">InPtr

    inc     <@" $aName$">FillCount              ; // One more filled
    bra     <@" $aName$">_wr_done               ; // B/ done

<@" $aName$">_overflow:

    inc     [ w3], [ w3]                        ; // Unclaim room
    inc     <@" $aName$">Overflows              ; // Increment overflow count

    ; // Fall through to done

<@" $aName$">_wr_done:

    Drop                                        ; // Drop a stack entry
  endMacro
  #doc
  //  [[Queues]] Write byte to queue.
  //  When no empty cells are available ([[queue.HasRoom]] returns 0) the data
  //  byte is discarded and the overflow count is incremented by one.
  #endDoc


  // ////////////////////////////////////////////////////////////////////////
```

```
macro queue.overflows ( aName ) ( -- bCount )
  mov     #<@" $aName$">Overflows, w1         ; // Point to count
  PushReg [ w1]                               ; // Push current count
  subr    w0, [ w1], [ w1]                    ; // Subtract returned count from current
endMacro
#doc
//  [[Queues]] Get number of overflows since last read
#endDoc


// ////////////////////////////////////////////////////////////////////////

macro queue.underflows ( aName ) ( -- bCount )
  mov     #<@" $aName$">Underflows, w1        ; // Point to count
  PushReg [ w1]                               ; // Push current count
  subr    w0, [ w1], [ w1]                    ; // Subtract returned count from current
endMacro
#doc
//  [[Queues]] Get number of underflows since last read
#endDoc


// ////////////////////////////////////////////////////////////////////////
// Parameterized high level definitions
//
// Define the following as meta code so no target code can directly be
// created from this

macro queue ( aName aSize ) ( -- )
<!
  $aSize$ constant $aName$Length
  $aSize$ array $aName$       #//  [[Queues]] Queue data storage
  variable $aName$FreeCount   #//  [[Queues]] Number of free cells in the queue
  variable $aName$FillCount   #//  [[Queues]] Number of filled cells in the queue
  variable $aName$InPtr       #//  [[Queues]] Location of the first free cell in the queue
  variable $aName$OutPtr      #//  [[Queues]] Location of the first filled cell in the queue
  variable $aName$Overflows   #//  [[Queues]] Current number of queue oveflows
  variable $aName$Underflows  #//  [[Queues]] Current number of queue underflows

  : $aName$.hasData ( -- b )

    #//  [[Queues]] Returns number of data items available in the queue

    queue.hasData $aName$
  ;
    resources
      $aName$FillCount
    endResources

  : $aName$.hasRoom ( -- b )

    #//  [[Queues]] Returns the number of free cells available in the queue

    queue.hasRoom $aName$
  ;
    resources
      $aName$FreeCount
    endResources

  : $aName$.read ( -- b )

    #//  [[Queues]] Read first available item from the queue.
    #//  When none is available ([[$aName$.hasData]] returns 0) the result is undefined.

    queue.read $aName$
  ;
    resources
      $aName$
      $aName$FillCount
      $aName$FreeCount
      $aName$OutPtr
    endResources

  : $aName$.write ( b -- )

    #//  [[Queues]] Write data byte in first free location of queue
    #//  When no room is available ([[$aName$.hasRoom]] returns 0) the data byte is discarded.

    queue.write $aName$
  ;
    resources
      $aName$
      $aName$FillCount
      $aName$FreeCount
      $aName$InPtr
    endResources

  : $aName$.overFlows ( -- b )

    #//  [[Queues]] Returns number of queue ovweflows since last read

    queue.overflows $aName$
  ;
```

```
      resources
        $aName$Overflows
      endResources

    : $aName$.underflows ( -- b )

      #//  [[Queues]] Returns the number of queue underflows since last read

      queue.underflows $aName$
    ;
      resources
        $aName$Underflows
      endResources

    : $aName$.clear ( -- )

      #//  [[Queues]] Clear the queue such that it holds no data
      #//  and that all cells are available.
      #//  [[$aName$FreeCount]] is set to the queue size, the other queue variables are set to zero.

      0             $aName$FreeCount  !         // Lock queue for writes
      0             $aName$FillCount  !         // Lock queue for reads
      0             $aName$InPtr      !         // Clear in  pointer
      0             $aName$OutPtr     !         // Clear out pointer
      0             $aName$Overflows  !         // Clear overflows
      0             $aName$Underflows !         // Clear underflows
      $aName$length $aName$FreeCount  !         // Give write access
    ;
      resources
        $aName$FillCount
        $aName$FreeCount
        $aName$InPtr
        $aName$OutPtr
      endResources
  >
  endMacro
  #doc
  // [[Queues]] Used to define a queue (fifo)
  //
  // with :
  //
  //   [[queue]] QueueName QueueLength // ( -- )
  //
  // QueueLength must be a power of 2 and should not exceed 64k
  #endDoc

  } // end meta compiler

// end
// ///////////////////////////////////////////////////////////////////////////


endModule
```