INTERFACING IDT's 3.3V MULTI-QUEUE FLOW-CONTROL DEVICE TO THE VIRTEX™ II FPGA

APPLICATION NOTE AN-349

By Stewart Speed

CONTENTS

ABSTRACT

The Virtex™ II family of FPGA's provides access and interfacing to a variety of memory resources, both off-chip and on-chip. In addition to the on-chip distributed RAM and block RAM features, Virtex-II FPGA's can interface to a variety of external high speed memory devices. One such device is a new family of memories manufactured by "Integrated Device Technology", IDT. This is the multi-queue flow-control device family, which introduces a new memory architecture to the arena of memory.

INTRODUCTION

Data buffering and queuing are common challenges in high performance data acquisition and data communication applications. If we look in particular at communication systems there is real need for buffering and queuing functions. Data streams/paths have typically been buffered by traditional FIFO's, these provide short-term buffering and also provide an effective means of coupling between two different clock domains. The multi-queue flow-control device provides the traditional benefits and features of a standard FIFO, but now with the added function of having discrete queues that can be accessed independently by the write port and read port, each of these queues being a storage buffer. These devices are based on a new industry architecture that combines high-speed queuing logic with an embedded memory core. The multi-queue flow-control device is a fully programmable device, the number of queues, depth of each queue and flag offset positions are all programmable.

The multi-queue flow-control devices are available in densities up to 2Megabits and clock speeds up to 200MHz. By integrating high-speed logic and memory on silicon, they are able to achieve sustained throughput rates up to 7.2 Gbps

Since the device is programmable and queues are addressable on both the write and read port, there is some control involved in the operation of the ports. This application note provides the reader with some general guidelines and suggestions for the control and interfacing requirements between a Virtex II device and a multi-queue flow-control device. This application note is based on using the 3.3V multi-queue, (the timing for a 2.5V multi-queue is slightly different). For more information on the Virtex II devices please refer to the data sheet. More information and data sheets for the Multi-Queue devices can be found on the IDT website at: www.idt.com

MULTI-QUEUE FLOW-CONTROL DEVICE AT A GLANCE

- 2.5V and 3.3V Families
- Q-Number options: 4Q, 8Q, 16Q, 32Q
- Memory density options: 256Kbit, 512Kbit, 1Mbit, 2Mbit
- Up to 200MHz high speed operation
- x9, x18, x36 bit wide data port options
- Individual, active queue flags ($\overline{OV}$, $\overline{FF}$, $\overline{AE}$, $\overline{AF}$) and Programmable Almost Full and Almost Empty Flags ($\overline{PAF}$/$\overline{PAE}$)
- 8-bit dedicated flag busses to monitor all queues ($\overline{PAF}$n/$\overline{PAE}$n)
- Packet Ready mode of operation
- 256-pin Ball Grid Array (BGA) with JTAG Functionality
- Bus-width matching on both ports
- Available with selectable LVTTL or HSTL I/O (2.5V family)
- Echo Read Clock and Enable is available for 'Source Synchronous Clocking' (2.5V family)
- Up to 8 Multi-Queue devices can be cascaded for depth and queue expansions (256 queues)

MULTI-QUEUE FLOW-CONTROL DEVICE ARCHITECTURE

BASIC CONCEPT

The multi-queue flow-control device is a single chip within which anywhere between 1 and 32 discrete queues can be setup. All queues within the device have a common data input bus, (write port) and a common data output bus, (read port). Data written into the write port is directed to a respective queue via an internal de-multiplex operation, addressed by the user. Data read from the read port is accessed from a respective queue via an internal multiplex operation, addressed by the user. Data writes and reads can be performed at high speeds up to 200MHz. Data write and read operations are totally independent of each other, a queue maybe selected on the write port and a different queue on the read port or both ports may select the same queue simultaneously.

QUEUE CONFIGURATION

The user has full flexibility configuring queues within the device, being able to program the total number of queues between 1 and 32 and the individual queue depths. If the user does not wish to program the multi-queue device, a default option is available that configures the device in a predetermined manner. Each multi-queue device has a total available memory within it, queues can be configured within the device using some or all of this available memory. Queues of varying depths can be configured within a single device.

JULY 2003

QUEUE FLAGS

For the selected queue, the device provides Full flag ($\overline{FF}$) status for write operations and Output Valid ($\overline{OV}$) flag status for read operations.

Also, the user can set independent Almost Full and Almost Empty values for each queue. These values can be monitored via the Programmable Almost Full ($\overline{PAF}$) flag and Programmable Almost Empty ($\overline{PAE}$) flag for the selected queue.

To retrieve the Almost Full and Almost Empty status of queues not selected for read or write operations the device provides two 8-bit flag busses ($\overline{PAF}$n/$\overline{PAE}$n). When 8 or less queues are configured in the device each bit of these flag busses represents an individual flag per queue. When more than 8 queues are used, a block of 8 queues can be monitored at a time.

A Packet Ready mode of operation is also available on the Multi-Queue family. Here the device provides status of whether or not one or more 'full' packets of data are present within respective queue's. A Packet Ready flag status can be obtained for each queue within a device.

PACKET MODE

A Packet Ready mode of operation is also available on the Multi-Queue family. Here the device provides status of whether or not one or more 'full' packets of data are present within respective queue's. A Packet Ready flag status can be obtained for each queue within a device.

BUS-WIDTH MATCHING

Bus-Width Matching is available on this device. Either port can be 9-bit, 18-bit or 36-bit provided that at least one port is 36 bits wide. This allows to interface busses of different width, seamlessly.

DEPTH/QUEUE EXPANSION

Expansion of Multi-Queue devices is also possible. Up to 8 devices can be connected providing the possibility of both depth expansion and queue expansion. Depth Expansion means expanding the depths of individual queues, queue expansion means increasing the total number of queues available.

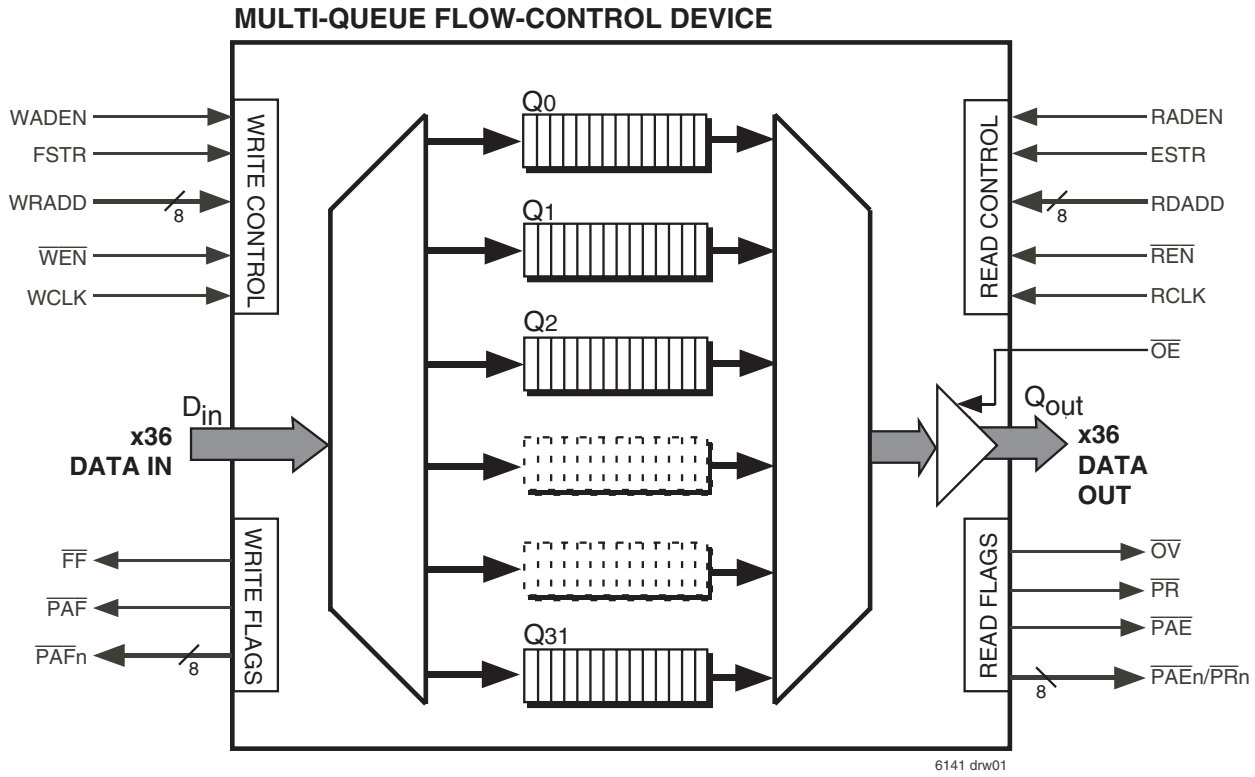


Figure 1. Block Diagram

INTERFACE CONNECTIONS BETWEEN THE VIRTEX™ II AND MULTI-QUEUE

The diagram below shows the connections between the Virtex™ II and Multi-Queue devices based on a single FPGA controlling both the Write and Read ports. It can seen that the write port control operates on a separate clock than the read port control. This illustrates an advantage of using the Virtex II and Multi-Queue combination in that both devices can provide data transfer between different clock domains. For example the Write Clock can be running totally independently and even at a different speed than the Read Clock.

The Virtex II family and Multi-Queue family also complement each other in that the I/O of each device can be configured by the user for different I/O standards. The 2.5V Multi-Queue family supports 2.5V LVTTL, 1.5V HSTL and 1.8V eHSTL, all of which (and many more) are supported by the Virtex II family.

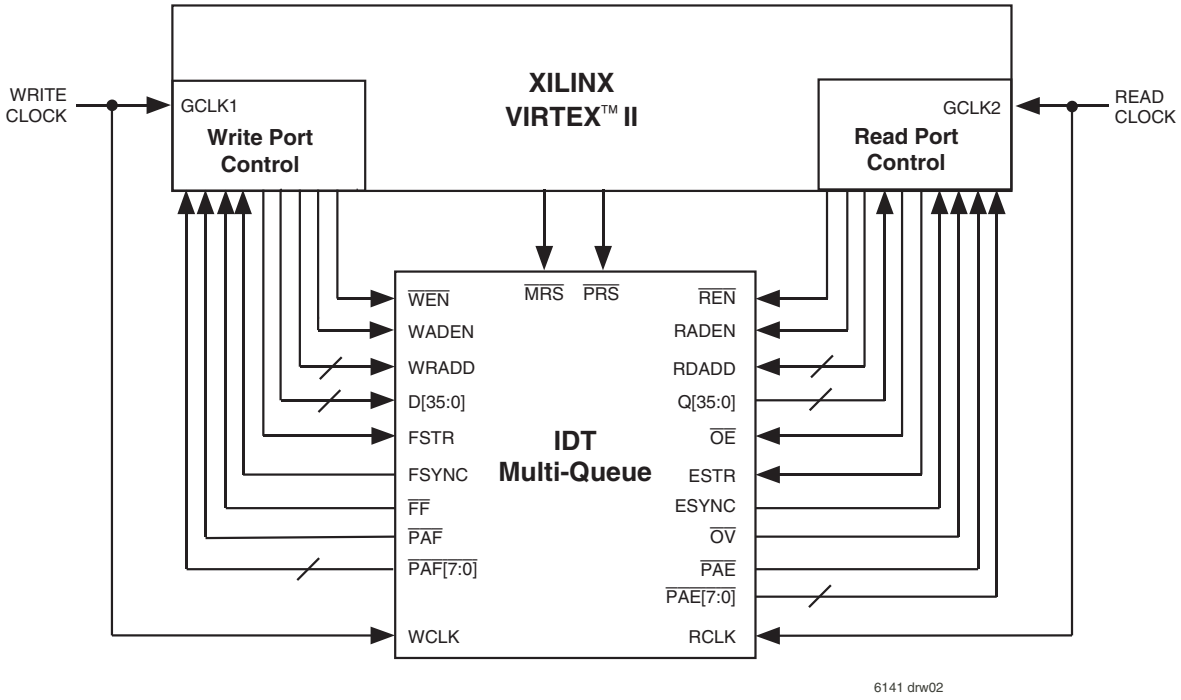


Figure 2. Interconnect Diagram

SET-UP, CONFIGURATION & BASIC OPERATION OF THE 3.3V MULTI-QUEUE

There are three main stages to using an IDT Multi-Queue Flow-Control Device(s):

1. Master Reset
2. Programming
3. Normal Operation (Writes & Reads)

This Application Note guides the user through each of these stages and makes suggestions as to how each stage may be controlled.

RESET, PROGRAMMING AND CONFIGURATION OPERATION

For the purpose of this application note, the following block of verilog code shows both the write and read ports being controlled by the same device/module. However it is possible that a separate device will control the read port and that a separate device again may even control the reset logic.

Any input that does not toggle can be tied HIGH or LOW by the user. For example, if a Partial Reset is never required, the $\overline{PRS}$ input does not need to be controlled by the controller and can be tied in-active, HIGH. This discussion also requires that the JTAG port is idle.

Control Pins Involved:

$\overline{MRS}$, $\overline{PRS}$, $\overline{WEN}$, WCLK, WRADDn, WADEN, FSTR, $\overline{REN}$, RDADDn, RADEN, ESTR, $\overline{SENI}$, DFM, DF.

MASTER RESET & PROGRAMMING

The Master Reset described below is initiated by a "System Reset". When a System Reset occurs a master reset of the Multi-Queue follows.

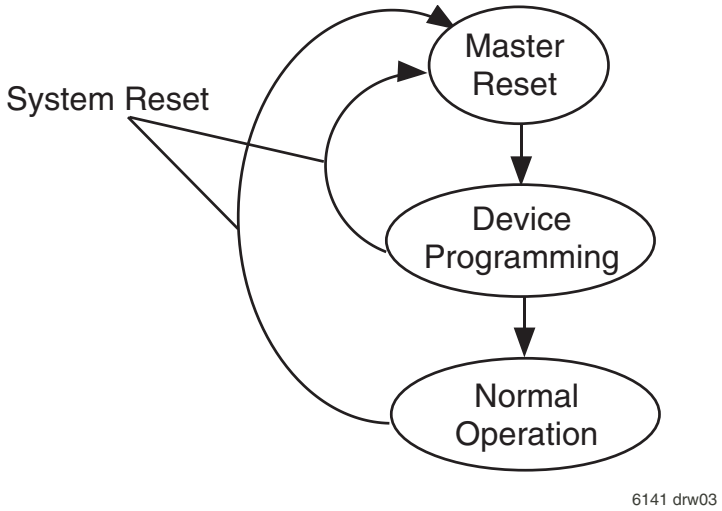


Figure 3. Reset Flow Chart

Following is an extract of sample Verilog code that controls the Master Reset function. This code is based on the user performing default programming of the Multi-Queue device to setup the part in a pre-determined configuration. If the user wishes to customize the configuration of the Multi-Queue they should refer to the IDT Application Note, AN-303 "Multi-Queue Flow-Control Device- Serial Programming", for a detailed example.

VERILOG CODE - MASTER RESET & PROGRAMMING

```
// Master Reset & Programming Function

// The input & output ports are not listed in this example
// Multi-Queue inputs: MAST, OW, IW, ID0, ID1 and ID2 can be controlled by the FPGA, however it is
// assumed that they will be externally tied HIGH or LOW.
// The DFM input to the Multi-Queue must be tied HIGH to enable default programming. If it is LOW then // serial programming should be performed.
// Registers required for the master reset & programming function
reg [2:0] state;
reg [1:0] count;

// State Machine labels
parameter reset = 3'b000;
parameter res_rec = 3'b001;
parameter dfprog = 3'b010;
parameter progrec = 3'b011;
parameter normop = 3'b100;

always @ (posedge wclk or posedge system_reset)
begin  // sync to wclk

    if (system_reset) begin      // System Reset is High, therefore perform Master Reset on MQ
        mrs_ <= 1;
        prs_ <= 1;
        df <= 0;        // df = 0 gives PAE & PAF offset values of 8 for all queues. df = 1 is 128
        wen_ <= 1;   ren_ <= 1;
        fstr <= 0;      estr <= 0;
        waden <= 0;  raden <= 0;
        seni_ <= 1;

        id0 <= 0; // id0, id1 and id2 are device ID and may be set up as 1 based on
        id1 <= 0; // the user requirements. These pins are typically tied High or
        id2 <= 0; // LOW

        count <= 0;   // This is a general purpose counter.

        State <= reset;
    end  // IF (system reset is high)

    else begin  // System Reset goes Low, reset complete.

        case (state)

        reset:    if (count < 3) begin          // Perform mrs_ LOW
                      mrs_ <= 0;
                      state <= reset;           // remain in reset for 3 wclk cycles
                      count <= count + 1;
                  end
                  else begin                    // when counter = 3 reset is complete
                      mrs_ <= 1;                // Take mrs_ HIGH, reset complete
```

```
                state <= res_rec;       // The delay ensures reset recovery period
                count <= 0;             // Reset counter
            end

res_rec:    if (count < 3) begin        // Delay for Reset Recovery
                state <= res_rec;       // Delay for 3 wclk cycles
                count <= count + 1;
            end
            else begin
                state <= dfprog;        // Reset recovery complete goto programming
                count <= 0;             // reset counter
                seni_ <= 0;             // Take seni_ LOW so default prog can begin
            end

dfprog:     if (seno_) begin            // wclk cycles load the default settings while
                state <= dfprog;        // seno_ is HIGH and seni_ is LOW
            end
            else begin
                seni_ <= 1;             // Loading of default settings is complete
                state <= progrec        // Normal Operations may begin after a Prog Recovery
            end

progrec:    if (count < 3) begin        // Delay for Programming Recovery
                state <= progrec;       // Delay for 3 wclk cycles
                count <= count + 1;
                end
                else begin
                    state <= normop;        // Programming recovery complete goto Normal Op
                    count <= 0;             // reset counter
                end

            normop:
            // Once the state machine has reached this state, normal operations of the
            // Multi-Queue may begin. This includes addressing of queues, writing & reading
            // of queues, direct accessing of the flag busses etc.

        end // IF System Reset is LOW

end  // sync to wclk
```

PARTIAL RESET

A Partial Reset can be performed on any individual queue within the device. A partial reset will reset the read and write memory pointers to the first location of the queue. To perform a partial reset the respective queue must be selected on both the write and the read ports before the partial reset is initiated. The following verilog code illustrates how this may be achieved. The code assumes that a single Virtex II is used to control both the write and read ports and that the same clock is applied to the write and read ports. The code also assumes that a Partial Reset request is provided when a Partial reset is required.

VERILOG CODE - PARTIAL RESET

```
// State Machine labels
parameter waittwo = 3'b000;
parameter pares = 3'b001;
parameter done = 3'b010;
parameter recov = 3'b011;
parameter normop = 3'b100;

always @ (posedge wclk)
begin  // sync to wclk

    if (partial_reset) begin // System Reset is High, therefore perform Master Reset on MQ
        wradd <= qnum;   //Set-up the queue to be partial reset on the write port.
        waden <= 1;
        wen_ <= 1;          // disable writes during Partial Reset
        rdadd <= qnum;   //Set-up the queue to be partial reset on the read port.
        raden <= 1;
        ren_ <= 1;          // disable reads during Partial Reset
        state <= waittwo;
        count <= 0;
    end

    else begin

        case (state)

        waittwo: begin     // wait two cycles before performing partial reset
            if (count == 1) begin
                state <= pares;
                count <= 0;
            end

            else begin
                count <= count + 1;
                state <= waittwo;
            end

        pares: begin  // perform reset, take prs_ LOW,
            prs_ <= 0;
            state <= done;
            end

        done:    begin     // Partial Reset complete, take prs_ HIGH
            prs <= 1;
            state <= recov;
            end

        recov:    begin     // Wait 3 cycles before normal operations can resume
            if (count == 3) begin
                state <= normop;
```

```
                count <= 0;
            end

            else begin
                count <= count + 1;
                state <= recov;
            end

        normop: // Here normal operations may resume, queues may be addressed, written to and read from


        end case
    end
end
```

NORMAL OPERATION

WRITE PORT CONTROL

The write timing port is illustrated in the timing diagram below. Here we can see that a queue to be written into is first selected ant then 2 write clock cycles later an actual write operation on that queue may occur and data on the Din input bus is written into the queue on a rising edge of the write clock. It can also be seen that on the previous 2 clock cycles data can be written into the previously selected queue, this illustrates what is called "100% bus utilization". The timing diagram assumes that all queues are NOT full and are therefore available to accept data.

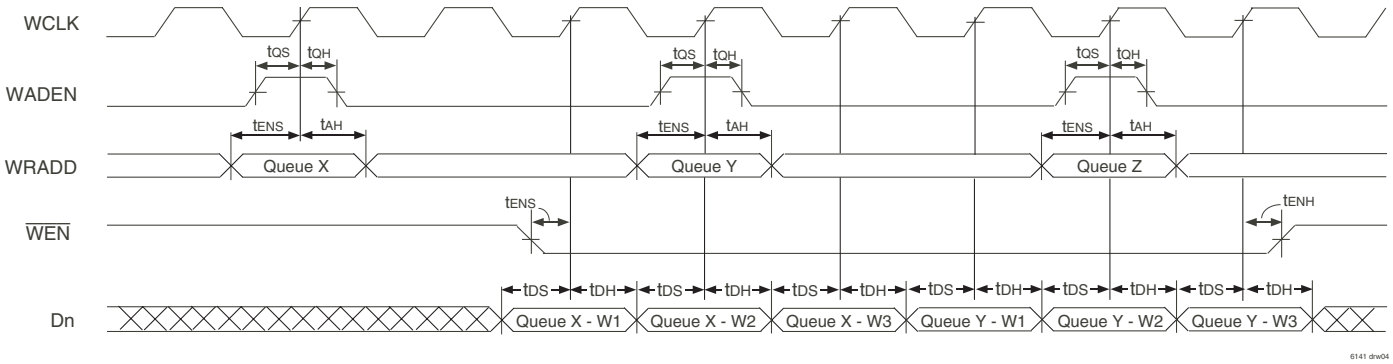


Figure 4. Write Control

The FPGA controls the write port and all write operations. The write control of the FPGA must take into consideration the 2 cycle latency between a queue selection and data being written to that queue. The FPGA write control must also detect when a queue being selected is full by monitoring a "full flag" output from the IDT Multi-Queue device. The Full flag output will provide status of the queue selected on the next WCLK clock cycle after the queue selection is made.

The following section of Verilog code should provide the reader with a simple example of how the VirtexTM II FPGA can provide effective control of the Multi-Queue write port. The code below assumes the following:

1. A Master Reset has been performed on the Multi-Queue device.
2. Programming of Multi-Queue device has occurred, (either default or serial).
3. The FPGA will determine when a queue is going to be written into, the initial part of the code is waiting for a queue request to be made.
4. The Multi-Queue contains 32 queues that can be addressed.

VERILOG CODE – WRITE PORT CONTROL

```
output  MQdata_in;
output  wen_;
output  waden;
output [4:0] wradd;
input ff;

reg [1:0] state;

parameter qwait = 2'b00;
parameter cycle1 = 2'b01;
parameter cycle2 = 2'b10;
parameter cycle3 = 2'b11;

always @ (posedge wclk)
begin

case  (state)

qwait:    begin     // Wait for first queue request
                    if (qreq) begin      // qreq = 1, a queue request is made
                              state = cycle1;
                    end
                    else begin           // qreq = 0, No request yet, wait for request
                              state = qwait;
                    end // qwait

cycle1:   begin
                    if (!qreq) begin     // No Queue change request - write to current queue
                              if (!ff_) begin       // Current queue is full, stop writing
                                        wen_ = 1;           // Writes are disabled, Data on queue inputs are
                                                            // NOT written into the current queue on this cycle.
                                        state = cycle1;
                                        MQdata_in = MQdata_in; // No change in data
                                        // New data should not be placed on the Multi-Queue inputs here.
                              end
                              else begin            //Current queue is not full, writes may continue
                                        state = cycle1;
                                        wen_ = 0;
                                        // New data maybe placed on to the Data Inputs of the Multi-Queue
                                        // at this point.
                                        // Note also that data on the MQdata_in inputs is written into the
                                        // new queue on this cycle if wen_ is LOW
                                        // It may be desirable to add further control here if writes
                                        // are not simply of function of a queue being full.
                              end
                    else begin           // A queue change has been requested
```

```
                                        // wen_ can be Low on this cycle
                                        // Note also that data on the MQdata_in inputs is written into the
                                        // previous queue on this cycle if wen_ is LOW
                                        // Write operations can occur during a queue switch
                                wradd = newqnum;            // Queue address gets new queue
                                waden = 1;          // set WADEN so address is made on next cycle
                                state = cycle2;
                                qreq = 0;// a queue change will occur reset the change
                                                    // request until next queue change request
                        end // cycle1

cycle2:   begin     // The address on WRADD is loaded into the Multi-Queue
                                // Note here that the Full Flag still shows status of the previous queue
                                // Note also that data on the MQdata_in inputs is written into the
                                // previous queue on this cycle if wen_ is LOW
                                // of the previous queue for this cycle.
                        waden = 0;          // WADEN goes LOW, address is now loaded
                        state = cycle3;
                        end // cycle2

cycle3:   begin
                        state = cycle1;
                        // On this cycle the Full Flag to change to show new queue status
                        // status to the new queue.The Full Flag will update to new queue
                        // on this cycle.
                        // Note also that data on the MQdata_in inputs is written into the
                        // previous queue on this cycle if wen_ is LOW
                        // A write to the previous queue may occur, wen_ may be LOW
                        end // cycle3
endcase
```

READ PORT CONTROL

The control algorithm/ logic controlling the read port must be able to recognize that a valid data read, (that is to say that a new data word may have been accessed and placed onto the Multi-Queue output bus), may have occurred due to at least one of four possible events. Here is a description of the four possible events:

1. A read may occur from a previously selected queue by taking the $\overline{REN}$ input LOW. Note, here that a new word is accessed on the rising edge of read clock and is available for the reading device to process that word on the following rising edge of read clock.

2. The Multi-Queue device operates in a "First Word Fall Through", FWFT mode. One implication of this is that if the same queue is selected on both the write and read ports the first word written into the queue will automatically fall through to the outputs, regardless of the state of $\overline{REN}$.

3. During a queue switch on the read port a final read will occur from the old queue. Data from the old queue will be accessed and placed onto the Multi-Queue data outputs regardless of the state of $\overline{REN}$. This is due to the "Next Word Fall Through" nature of the device. This word from the old is forced out to allow an automatic read from the new queue, where the next word available in the new queue falls through to the outputs of the Multi-Queue. This leads to event four.

4. Also during a queue switch on the read port, the first word from the newly selected queue will be accessed and placed onto the Multi-Queue data outputs regardless of the state of $\overline{REN}$. Again this is due to the "Next Word Fall Through" operation, where the next word available in the new queue falls through to the outputs of the Multi-Queue.

This Application Note reviews these four events and shows an overall solution that may be implemented into the control logic of the FPGA to handle any event.

Read Operations on the Current Queue

Figure 5 illustrates the potential event outlined in item 1 above. The diagram shows the situation where a queue has previously been selected on the read port and read operations are occurring based on valid words being available in the queue and the $\overline{REN}$ input being toggled LOW to read out data words.

The Output Valid flag is essentially provided to indicate when a queue is empty or has been read to empty. It does not indicate when a new word has been placed on to the data outputs.

This diagram illustrates the implementation in the FPGA's control logic of a flip-flop called "lastren", that indicates when an enabled read has been performed and the word on the output bus should be processed by the reading device. For example, if we look at cycle ***C*,** we can see an enabled read is performed and word "LW-5" is placed onto the outputs. On the next the cycle, cycle ***D*** the reading device should process this word "LW-5". The state of "lastren" is used to determine whether the word is a new word and therefore whether it should be processed.

Based on this set-up the reading device will process words on the following cycles:

D*, *G*, *H*, *I*, *J* & *K*.** Note, that on cycle ***L the output valid flag, $\overline{OV}$ is HIGH, therefore the queue has essentially been read to empty.

In summary, when a normal read event occurs on the current queue, a new word should be processed by the reading device on an RCLK cycle where the $\overline{OV}$ flag is true, LOW and the "lastren" flip-flop is true, LOW.

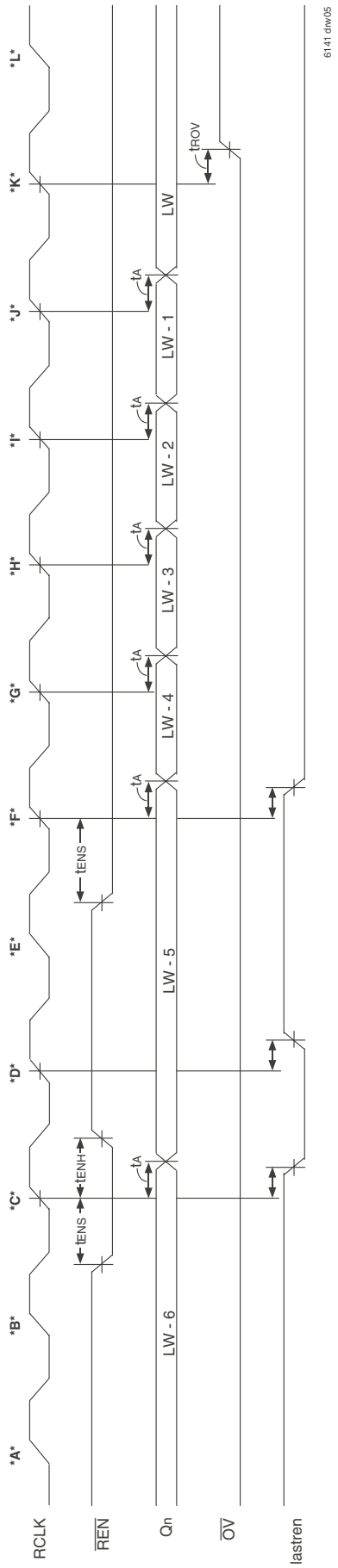


Figure 5. Read Control- Normal Read

Read Operation due to FWFT in Current Queue

Figure 6 below illustrates the "First Word Fall Through" (FWFT) effect, this the second possible read event mentioned earlier. In this diagram both the Write and Read ports are selected for the same queue. A new word, "W1" is written into that queue, which before the write operation was an empty queue, the $\overline{OV}$ flag is HIGH. This first word written into the queue automatically falls through to the data outputs, causing the $\overline{OV}$ flag to go active, LOW. Note, that this first word has fallen through regardless of the state of $\overline{REN}$, in fact $\overline{REN}$ is HIGH throughout this diagram. Therefore, the use of the "lastren" flip-flop mentioned above is no longer an option to determine if a new word is available for the reading device to process. We have now included a second flip-flop, "lastov" into the control logic of the reading device. This flip-flop provides the reading device with a status of the $\overline{OV}$ flag delayed by one read clock cycle. Therefore, the reading device monitors both $\overline{OV}$ and "lastov" to determine whether a new word is available, here we can see that on read clock cycle *C* the first word, W1 should be processed by the reading device.

In summary, in the case of the FWFT read event a new word should be processed by the reading device on an RCLK cycle where the $\overline{OV}$ flag is true, LOW and the "lastov" flip-flop is false, HIGH

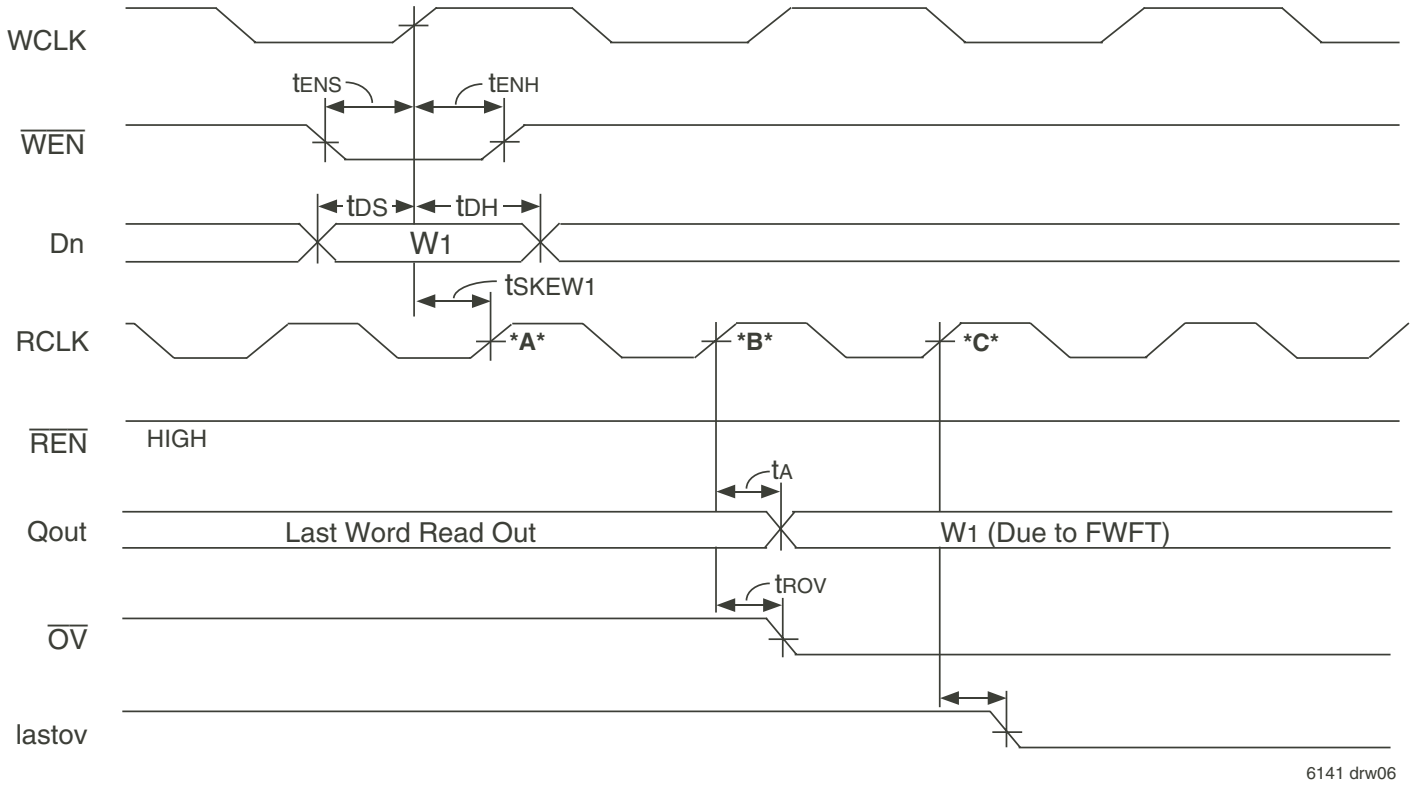


Figure 6. Read Control - FWFT

Read Operation due to a Queue Switch on the Read Port

Figure 7 below illustrates the third and fourth possible read events, which is indirectly due to the "First Word Fall Through" effect caused by a queue switch on the read port. In the diagram we can see that a "New Q" is selected on the read port on RCLK cycle ***A***. Here RADEN is HIGH and the RDADD address bus is addressing the new queue. When a queue selection is made the first word of the new queue will fall through to the data outputs of the read port, forcing a final word to be read from the previous (old) queue. This occurs regardless of the state of $\overline{REN}$, in fact the $\overline{REN}$ input is HIGH throughout this diagram.

During a queue switch the $\overline{REN}$ input can remain HIGH and the $\overline{OV}$ flag can remain LOW indicating that both the old queue and the new queue have not been read to empty. Therefore, we introduce a register "sinceraden" into the control logic of the reading device. Monitoring of the "lastren" and "lastov" flip-flops alone does not indicate to the reading device that a new word is available in this example, hence the introduction of the register, "sinceraden". This register is a 2 bit register and in the event of a queue switch is simply used to provide a count from 0 through 3. Upon RADEN going HIGH this register should be reset to a value 0. On the following three RCLK cycles this count will be incremented up to a value of 3, at 3 the count will cease incrementing and wait until RADEN is toggled HIGH once more.

In the event of a queue switch a minimum of 2 new words will be automatically read out from the Multi-Queue device and will need to be processed by the reading device, (this again is an effect of the "Next Word Fall Through" operation). From the diagram we can see that on RCLK cycle *B* "Word N" is read from the "Old Queue". This word must be processed by the reading device on RCLK cycle *C*. The same is true for the first word of the New Queue, "Word 1" which must be processed by the reading device on RCLK cycle *D*. The reading device should monitor the "sinceraden" register and when the register has a value of either 1 or 2 the data word on the bus is valid, providing of course that the $\overline{OV}$ flag is LOW. It is also very important during a queue switch on the read port, that the reading device realize that the new word processed with "sinceraden" at value 1, is data from the old queue, and therefore processed appropriately. Secondly, the new word processed with "sinceraden" at value 2, is data from the new queue, and therefore processed appropriately.

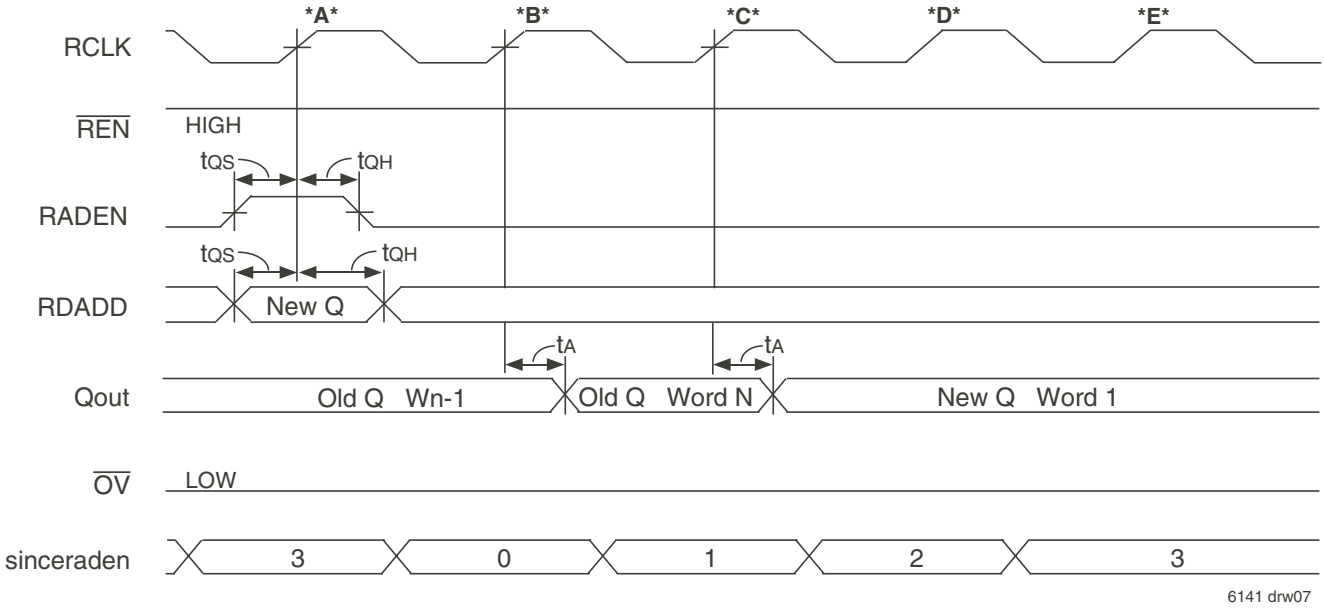


Figure 7. Read Control - Queue Switch

An Example of Consecutive Read Events

Figure 8 below shows an example where are number of read events occur consecutively. From Figure 8 we can see that an enabled read has occurred on the same RCLK cycle as the queue switch, cycle ***A***. In this example the "sinceraden" register serves exactly the same purpose as before, indicating in conjunction with the $\overline{OV}$ flag that a new word must be processed on RCLK cycles ***C*** and ***D***, ("sinceraden" = 1 or 2). However, in this event we must also process a word from the old queue that was accessed on RCLK ***A***. This word from the old queue, word "N-1" must be process by the reading device on RCLK *B*. The reading device control logic processes this word due to the fact that the "lastren" flip-flop is active, LOW on this cycle, and in conjunction with $\overline{OV}$ being true processes the word "N-1". As with the previous example, it is important to note that the data words processed by the reading device on RCLK cycles *B* and *C* are from the old queue and are therefore processed appropriately. In summary, when a queue switch is made it is possible that up to three words will need to be processed by the reading device. The first word will be processed by virtue of the fact that an enabled read was performed at the queue switch and therefore, the state of "lastren" in conjunction with the $\overline{OV}$ flag will ensure this word is processed. The following 2 words will be processed by virtue of the fact that register "sinceraden" has the value of 1 or 2, and that the $\overline{OV}$ flag is active, LOW. Again, it is important to note that a valid word read with "sinceraden" equal to '1', is a word from the old queue and therefore processed appropriately.

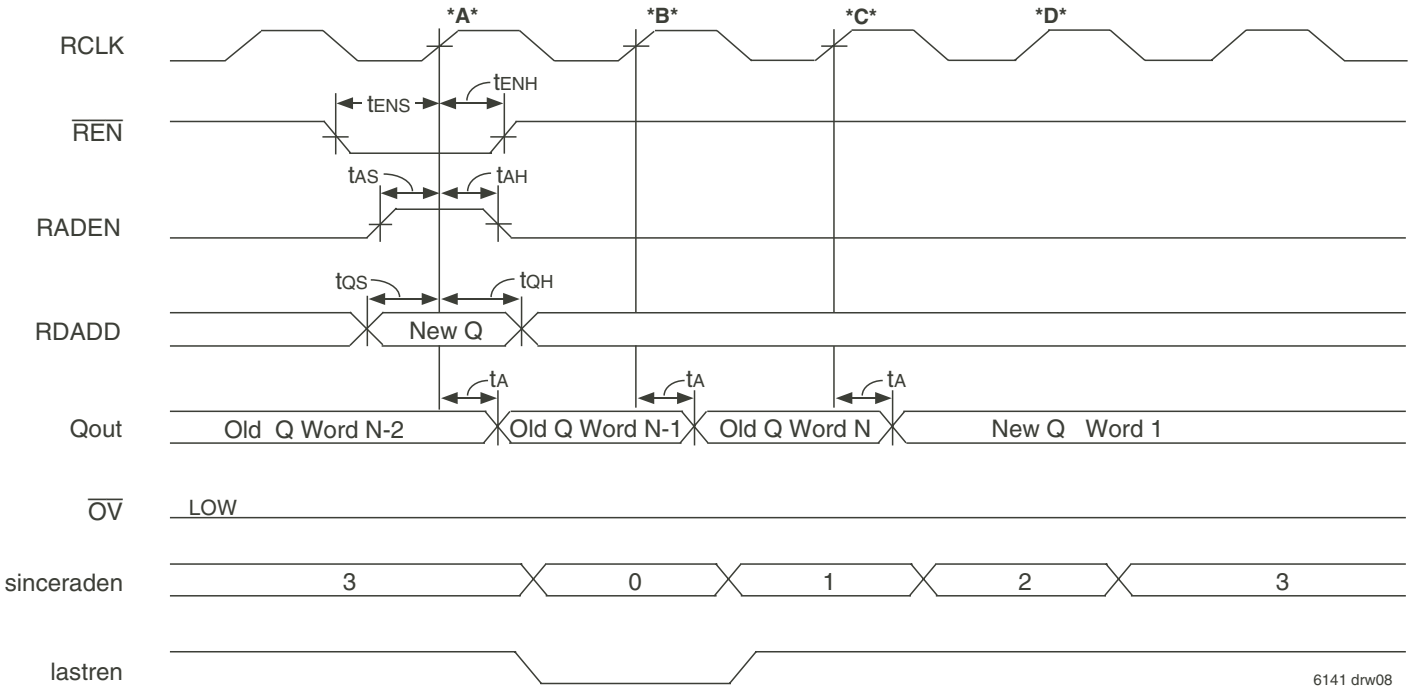


Figure 8. Read Control - Multiple Reads

Read port verilog Implementation

The control logic required to handled all of the events outlined above can be easily implemented in the FPGA, below is sample of Verilog code that produces the control logic required to detect when a word out of the Multi-Queue read port is a new, valid word that must be processed.

FPGA - Verilog Code Sample

```
// IDT 3.3V Multi-Queue Read Port Control Code Example

always @ (posedge rclk) begin

        if( (!ov_) && (
        (  !lastren_          //Check for normal read
        || lastov_            //Check for FWFT independent of queue switch
        || (sinceraden == 1)    //Check for queue purge after queue change (LWFT)
        || (sinceraden == 2) ))) //Check for next word from new queue after queue change (NWFT)

                begin
                /*
                Here the user should implement code that handles a valid data word
                according to their application requirements.
                */
                end
        end

        //Update counter that checks which queue (new or old) a word was read from
        if( raden ) begin
                sinceraden <= 0;
                //Reset the sinceraden 2 bit register to 0 when a new queue switch occurs
                end
        else begin
                if( sinceraden < 3 ) sinceraden <= sinceraden + 1;
                // sinceraden counter maximum is 3
        end

        lastren_ <= ren_;
        // The lastren_ register always provides stauts of the ren_ on the last RCLK rising edge
        lastov_ <= ov_;
        // The lastov_ register always provides stauts of the ov_ flag on the last RCLK rising edge

end

/*
The code outlined above only interprets when a new, valid word has been read out of
the Multi-Queue device and the queue the word was read from.
This code implements a 2 bit counter, "sinceraden" and two flip-flops, "lastren_"
and  "lastov_"
The counter, sinceraden is utilized during a read port queue switch. The value of
sinceraden determines from which queue a word was read during a queue switch. When the
read port pipeline is purged (regardless of ren_), a final word will be read from the
"old" queue (if a word is available) and the next word will be read from the "new"
queue (if a word is available).
If sinceraden = 1 then the word is from the old queue.
If sinceraden = 2 then the word is from the new queue.
If sinceraden = 0 or 3 then the lastren_ or lastov_ is used to determine whether a
read was performed.
*/
```

FLAG BUS OPERATION

There are two flag busses provided on the Multi-Queue device. This flag bus may be 4 bits wide or 8 bits wide, depending on the device selected. If a 4 queue device is used the flag busses are 4 bits wide and provide a constant status of "Almost Full" and "Almost Empty" conditions for each queue. An 8 queue, 16 queue or 32 queue device has 8 bit wide busses. For the 8 queue part the 8 bit busses again provide continuous status of each queue. Now for the 16 queue and 32 queue parts the flag busses can provide status of all queues in a multiplex manner. There are two modes of operation for the flag bus in a 16 queue or 32 queue part, these modes are:

1. Direct Mode – Here the user address the queues required on the flag bus.
2. Polled Mode – Here the queues are cycled on the flag bus.

In the following discussion we provide an example for utilizing a 32 queue part with the flag bus operating in "Polled" mode.

For a 32 queue device both the Almost Full flag bus, $\overline{PAF}$n and Almost Empty flag bus, $\overline{PAE}$n are 8 bits wide. The status of all 32 queues can be obtained over the period of 4 cycles of 8 bits, each 8 bit status is called a "Quadrant". The diagram below illustrates the timing of the "Almost Full" flag bus and the relationship between the FSYNC pulse & flag bus data.

The diagram above illustrates the timing for the "Almost Full" flag bus. Each quadrant is placed onto the bus with respect to a WCLK cycle, the first quadrant is marked with a logic HIGH on the FSYNC output. The "Almost Empty" flag bus has equivalent timing based on an RCLK input and providing and ESYNC output.

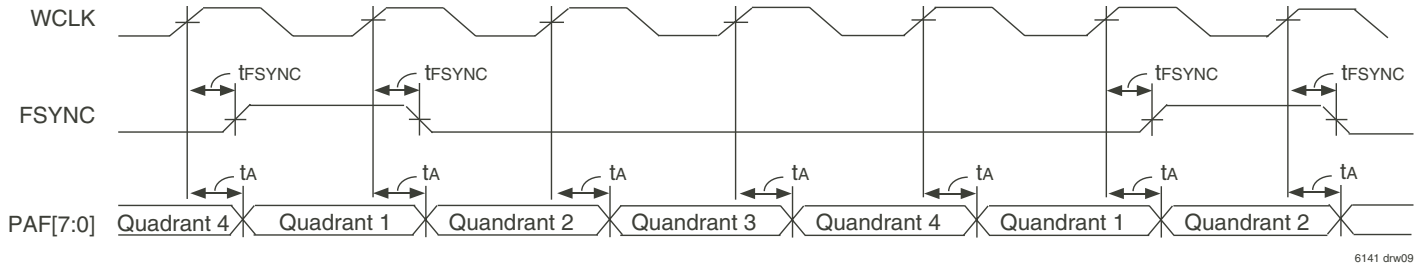


Figure 9. Flag Bus Control

Below is some example Verilog code that shows how the Virtex™ II FPGA can keep track of all 32 queues by updating a 32 bit flag bus register. This code assumes a 32 queue device is being used in Polled mode. The same code may applied to the "Almost Empty" flag bus operation.

FLAG BUS CONTROL CODE EXAMPLE

```
input [7:0] pafn;   // Almost Full Flag bus inputs
input  fsync;       // Almost Full Flag bus Sync pulse

reg [31:0] flagreg; // 32 bit register maintains a status of all queues Almost Full levels
reg [1:0] count;            // Counter

always @ (posedge wclk)
begin

        if (fsync) begin
                flagreg[7:0] <= pafn[7:0];
                count <= 0;
                end

        else begin
                if (count == 2'b00) begin
                        flagreg[15:8] <= pafn[7:0];
                        count <= count + 1;
                end

        else if (count == 2'b01) begin
                        flagreg[23:16] <= pafn[7:0];
                        count <= count + 1;
                end
        else if (count == 2'b10) begin
                        flagreg[31:24] <= pafn[7:0];
                        count <= count + 1;
                end
        end
end
```

PACKET MODE OPERATION

APPLICABLE DEVICES

IDT72V51236, IDT72V51246, IDT72V51256
IDT72V51336, IDT72V51346, IDT72V51356
IDT72V51436, IDT72V51446, IDT72V51456
IDT72V51546, IDT72V51556

INTRODUCTION

Packet Mode operation of the multi-queue flow-control device is only available on 36 bit wide devices (listed above). Packet Mode is a selectable mode of the part and must be selected during a Master Reset. When in Packet Mode the Packet Ready, $\overline{PR}$ flag becomes available and the $\overline{PAE}$n/$\overline{PR}$n bus operates in Packet Mode, providing Packet Ready status of queues. Note also, that when in Packet mode the Output Valid, $\overline{OV}$ flag is still available.

When in Packet Mode the user must utilize the Most Significant 3 bits of the data bus as Packet markers. D35/Q35 is the "End Of Packet", EOP marker, D34/Q34 is the "Start Of Packet", SOP marker and D33/Q33 is the "Almost End Of Packet", AEOP marker. When writing packets into a queue the user must include these markers at the appropriate positions. The 36 bit wide family of Multi-Queue devices contains logic that monitors these markers as they enter and leave the Multi-Queue. This device then provides Packet Ready status for both the queue selected on the read port as well all other queues within the device. Please refer to the data sheets for more details on the Packet Mode operation.

The purpose of this application note is to provide the reader with a more clear understanding of the control requirements of the Multi-Queue when operate in Packet mode. It is also makes suggestions/recommendations as to how the device controlling the read port should operate.

OPERATION

When using the multi-queue in Packet Mode the user should utilize a "Start of Packet" marker (SOP), an "End of Packet" marker (EOP) and an "Almost End of Packet" marker (AEOP).

The purpose of the SOP and EOP markers is apparent from reviewing the data sheet in that the Multi-Queue Packet Ready logic must track these markers as they both enter and exit a queue so as to provide accurate "Packet Ready" (packet availability) status for a respective queue. The purpose of the AEOP marker is not so apparent, but still important to the efficient operation of the Multi-Queue.

When in Packet Mode appropriate control of the read port is very important to maintain effective switching between queues and maintain packet integrity. Due to the "pipelining" structure of the Read port, one or two words will be automatically read out when a queue switch is made (on the read port). When a read port queue switch is made from Queue 1 to Queue 2, the next available in Queue 1 followed by the next available word in Queue 2 will be read out consecutively regardless of the state of the $\overline{REN}$ input, provided that the queues have complete packets available. It is important to note that a word cannot be read from a queue unless a complete packet is available in that queue.

There are a number of scenario's when considering the operation of the read port in Packet Mode, these are:

A. Reading out one (or more) packets until the last 'complete' Packet of a queue is read.

B. Reading out a Packet and then switching to a queue with another packet waiting.

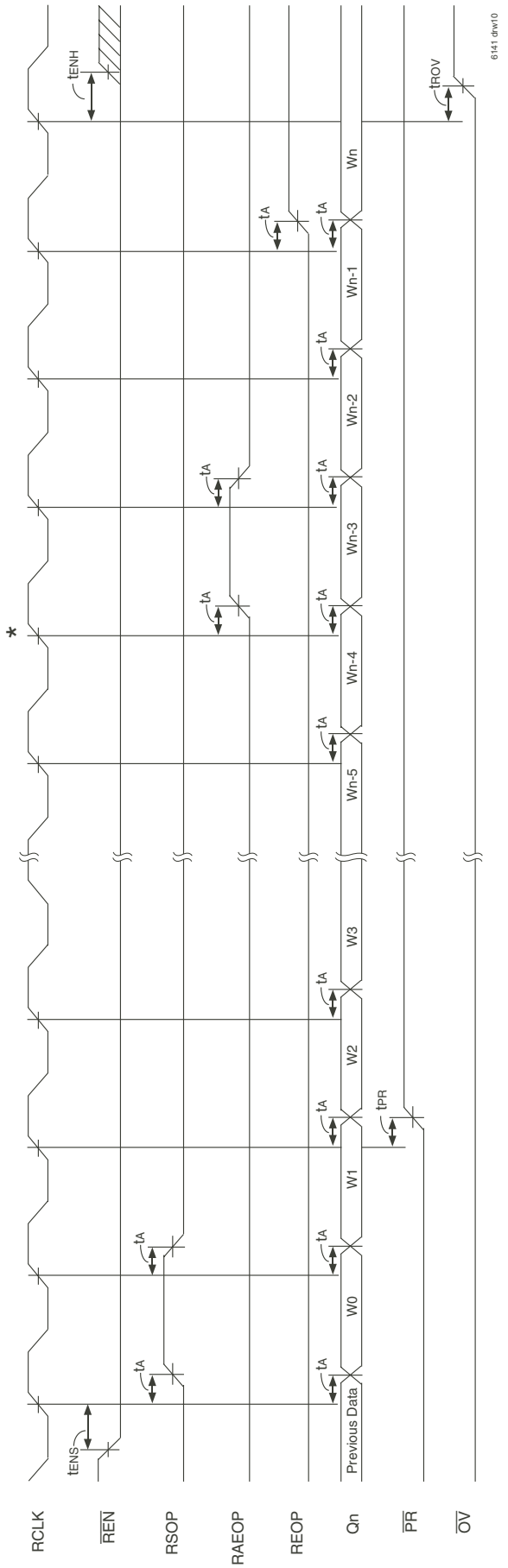
C. Reading out a Packet with another 'complete' packet behind it in the same queue and stopping reads without accessing the next packet.

D. Switching from a queue with no packets available (maybe empty or containing just a partial packet), to a queue with a packet ready.

Let us take a look at these scenario's in turn and highlight the control considerations for each.

A.

This is shown in Figure 10. Essentially when the packet is completely read out and the last word in the queue has been accessed, the Output Valid Flag will go HIGH thus preventing any further reads. This also covers the situation where multiple packets are read from the same queue until the queue has gone empty, or there are no more complete packets available.



NOTE:
1. This diagram shows a Queue being emptied of it's final "complete" packet.
2. On cycle "*" a decision must be made whether there is another packet to be read and where it is located. In this diagram the current queue is read to empty.

Figure 10.

B.

This operation brings about the possible need for the implementation of a "Null-Queue". As explained in the data sheet for the multi-queue flow-control device, the Null-Queue is a default queue to be selected at the end of read operations on a respective queue to prevent the pipeline being filled with the next word available in the current queue, this is also applicable in Packet Ready mode. If the user requires to read a packet from a queue and NOT begin reading the next "complete' packet in that same queue, a queue switch should be made out of that queue, either into a queue that has no complete packets available or into a 'Null-Queue". The timing of the queue switch is important here for 2 reasons:

1. So as to ensure the last word of the required packet is read out, this word marked with EOP. Let us call this word, "EOP of Packet A in Queue 1".

2. To ensure the first word of the next packet (marked with SOP) is NOT pushed into the read pipeline. Let us call this word "SOP of Packet B in Queue 1".

If there is a queue available with no complete packets ready, a queue switch can be made to this queue by virtue of the fact that data cannot be read from a queue that does not have a complete packet available. However, there may be an instance where this is not possible, i.e. all queues have packets available, but reading these packets is not currently desired. In this case a Null-Queue must be selected, to effectively, 'flush' the last EOP word of Packet A from Queue 1 and not read any further words or push the SOP word of Packet B from Queue 1 into the pipeline.

From reading the Multi-Queue Data Sheet, the definition of a Null-Queue is a queue that can never be written to or read from. To the Write port this queue always behaves as though it is full (Full flag LOW) and to the read port this queue always behaves as though it is empty (Output Valid flag HIGH).

Please see Figure 11.

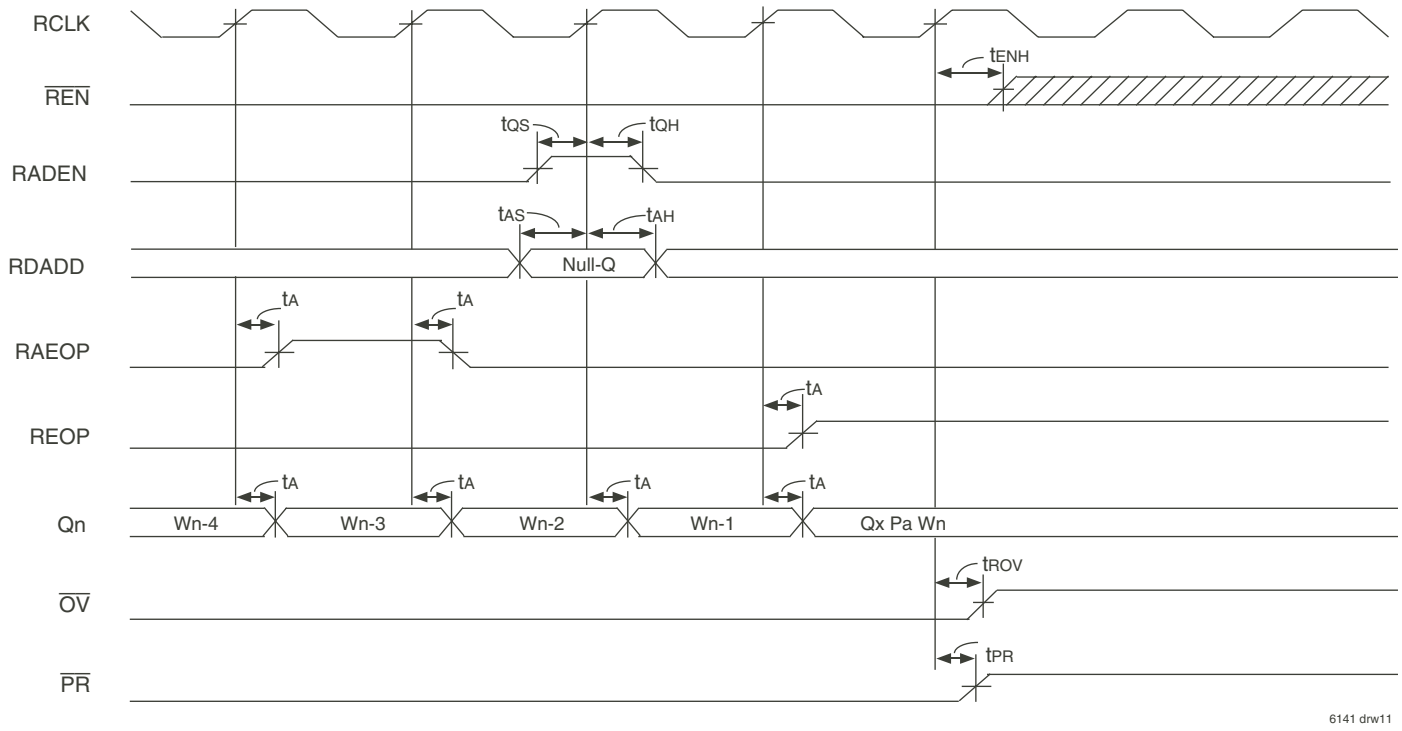


Figure 11.

C.

When a queue switch is required at the end of a packet, such that the next read will be a packet from the new queue, the device reading data from the Multi-Queue must ensure a queue switch occurs two RCLK cycles ahead of the EOP and to do this the user should utilize the AEOP marker.

Please see Figure 12.

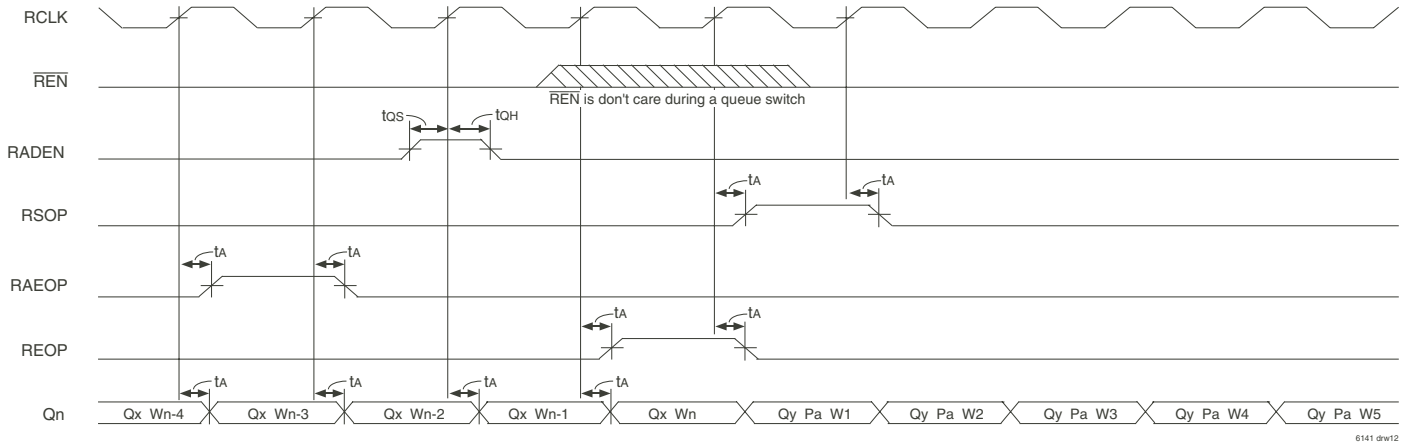


Figure 12.

D.

This instance may occur when the read port has previously been selected for a queue that either contains no packets available (or only a partial packet available), or the read port has been selected for the Null-Queue. A packet has now become available within another queue, thus requiring a queue switch to service that packet. This operation is quite straightforward, the first word of the new packet will be accessed 2 RCLK cycles after the queue selection is made, this can be seen in Figure 13 below.

This Figure 13 shows a Null-Queue selection being made, followed at a later time by a Selection of Queue Y being made. Here we can see that when the Null-Queue was selected we completed the reading of Packet 1 from Queue X. This last word from Queue X will remain on the output bus until a new queue (with a complete packet) is selected.

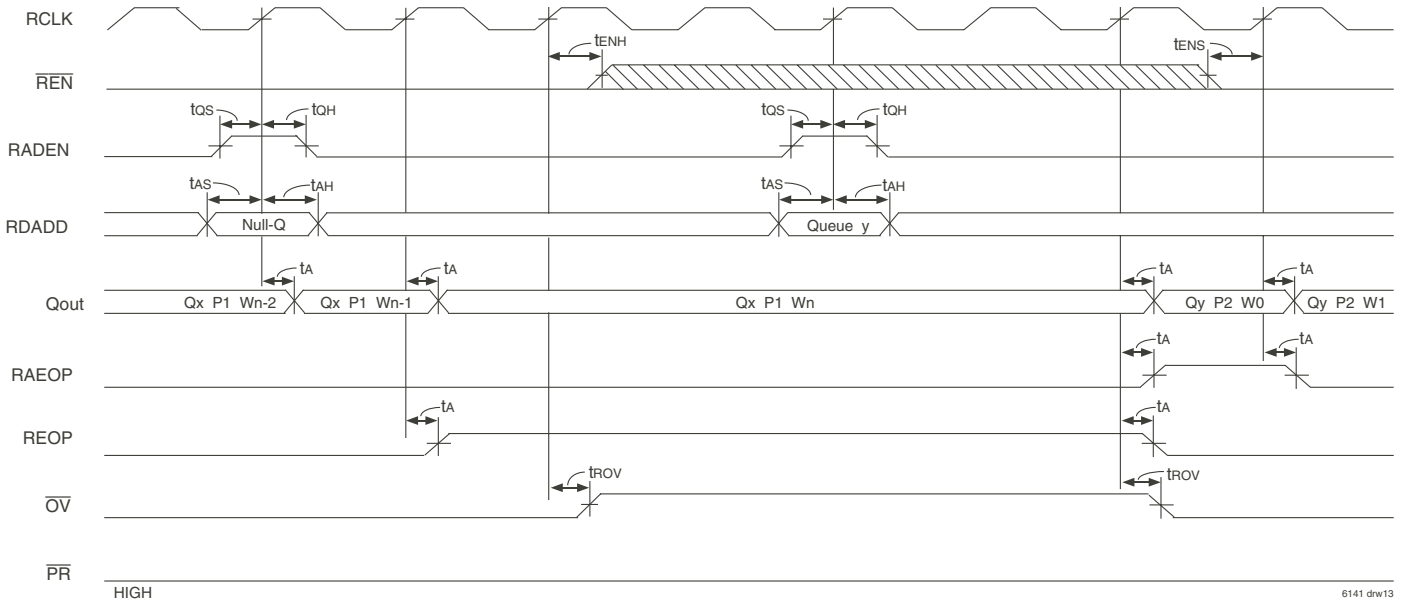


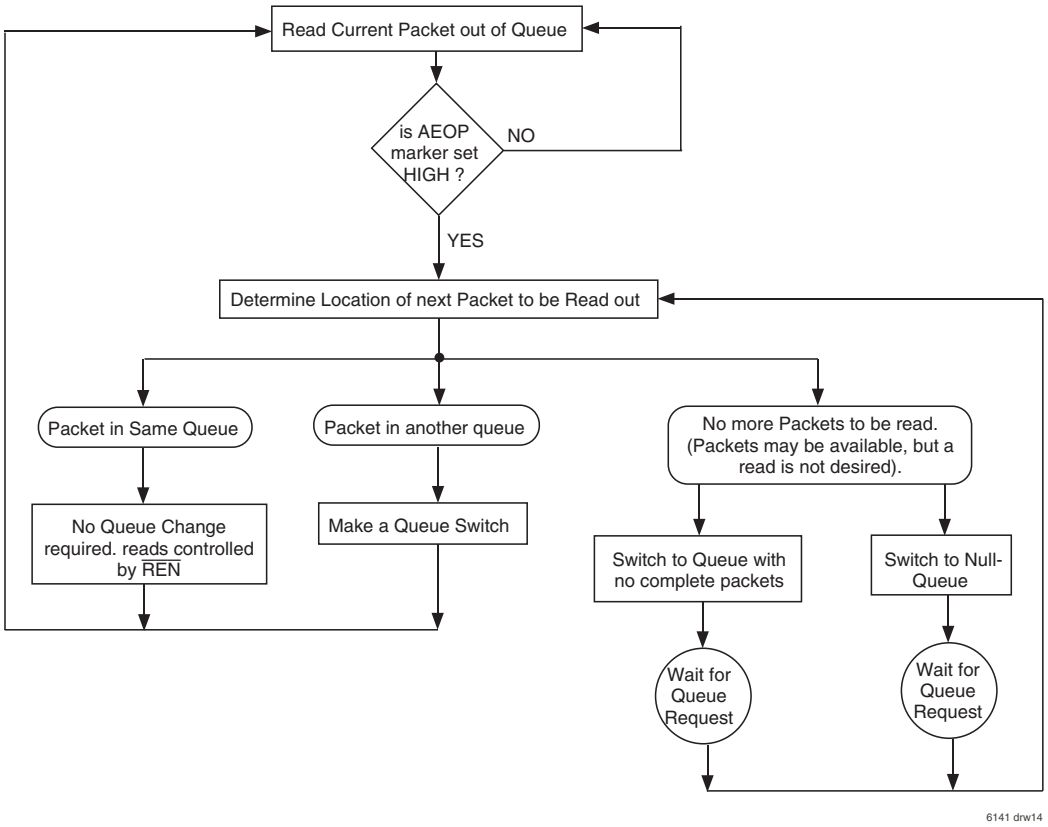
Figure 13.

PACKET MODE SUMMARY

In summary there is an important guideline when using the multi-queue flow-control device in Packet Mode, this is that the use of an AEOP marker is very important. The AEOP marker must be set a minimum of 4 locations away from the EOP. The controlling device must always determine what action must be taken before the EOP marker (last word of the packet), has been accessed. For the reasons discussed above, when an AEOP is detected the read port controlling device needs to make a decision on where the next packet will be accessed from and if no further packets are to be accessed, whether the selection of an empty queue (or even Null-Queue) is required to 'flush the pipe'.

To help determine which queue to switch to (if any), the multi-queue flow-control device provides a Packet Ready flag bus, in packet mode this $\overline{PR}$n bus becomes available. This bus provides a packet ready status for all queues, whether they are selected or not. The user should monitor this bus and use it to determine which queue a packet will be read from at any given time.

An example block diagram of process when an AEOP marker is detected on the read port is shown below.



NOTE:
That if there are any switching latencies within the read port controlling device when making a queue switch decision, the AEOP marker may need to be set more than 4 locations away from EOP to allow for this additional delay.

Figure 14. Packet Mode Flow Diagram

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers who are designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only to develop an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third-party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising from your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.01)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit www.renesas.com/contact-us/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.