PM7322

RCMP EGRESS ROUTING LOGIC IN VHDL

Issue 1: March, 1997

TABLE OF CONTENTS

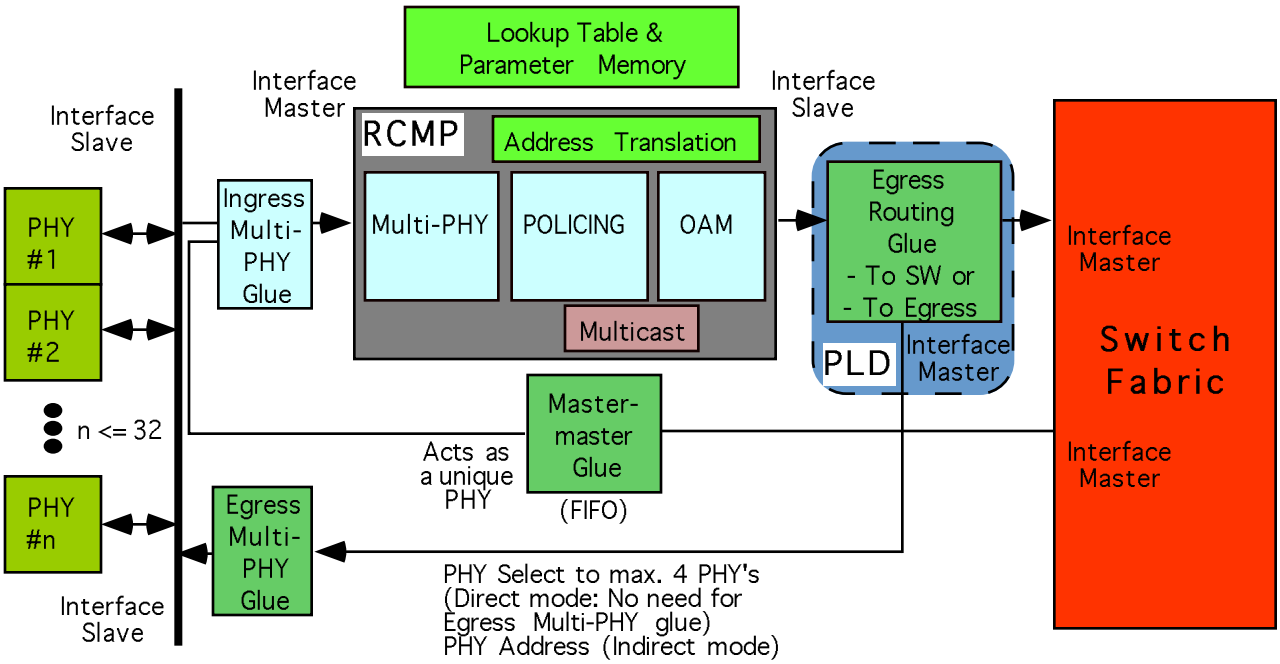
INTRODUCTION

The RCMP-800[1] can be configured to process both the Ingress and Egress cell traffic in a port card, where the bandwidth in either direction is less than half of the maximum bandwidth of the RCMP (ie. less than 400Mbps for the RCMP-800). As shown in Figure 1, with the RCMP positioned on the Ingress side, Egress cell traffic can be "looped" into the RCMP input as if it is sourced from another PHY. The output of the RCMP will thus consists of both Ingress and Egress cell traffic. As such, external glue logic is needed to route these cells to the appropriate destination (ie. to the switch or to the PHY).

This appnote describes a VHDL implementation of the Egress Routing glue logic as described above. The functionality, the timing aspects and a reference VHDL code implementation (included in Appendix A) are included. This glue logic can be implemented in a simple PLD.

This implementation is based on Appendix A of the appnote, PMC-951205, Issue 2, "RCMP in Egress Operation".

Figure 1 Egress "Loop" Port Card System



Although Figure 1 shows only the Egress "loop" application of the Egress Routing glue logic, this design can be easily modified to be used to interface with an existing Egress

[1]Hereafter referred to as "RCMP"

device. In this case, the Egress cell traffic from the switch will need to be muxed together with the "backwards" cell traffic from the RCMP output. This is especially useful for applications where only certain OAM cells from the RCMP are to be routed "backwards" to the Egress side, thus bypassing the switch fabric.

This design has been implemented and simulated in a Cypress CY7C374 PLD. This PLD contains 128 macrocells, which is roughly 4K gates, of which approximately 60% is utilized in this design. This PLD has 84 pins, 63 of which are I/O pins. This design uses 58 I/O pins.

This design applies equally to PM7323, the RCMP-200. The only difference with the RCMP-200 is that it has an 8-bit interface that runs at a maximum clock frequency of 25MHz.

FEATURES

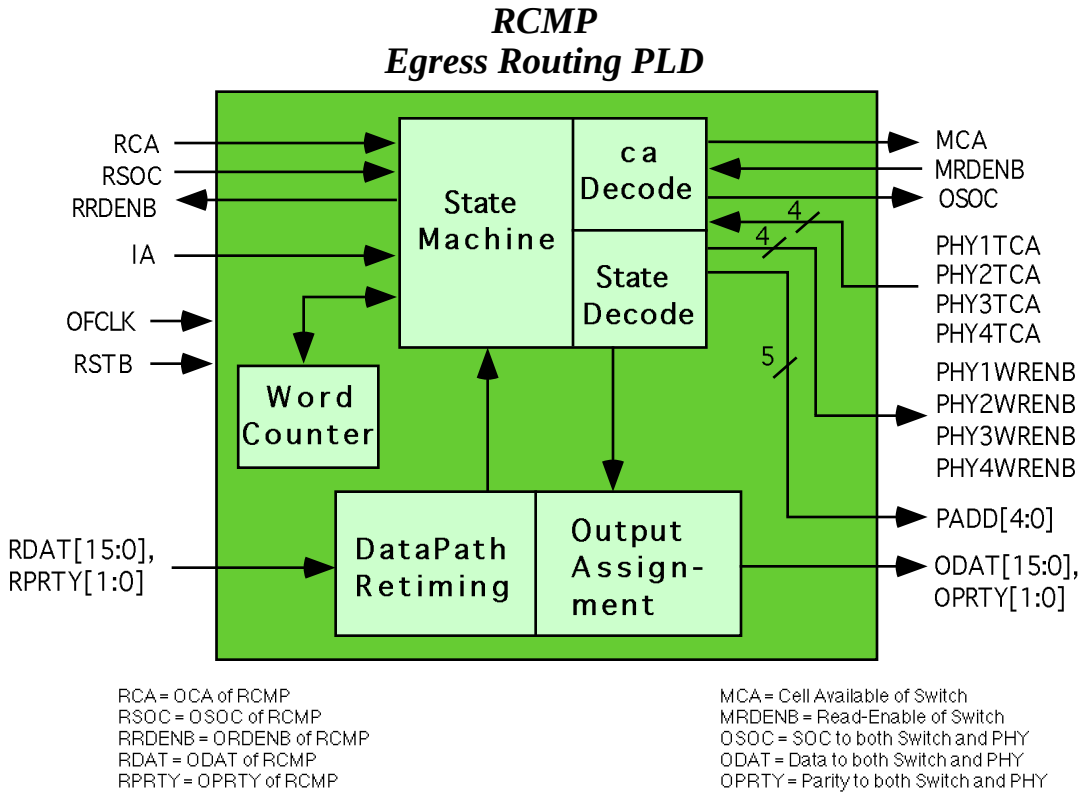
- Supports Backwards Routing of cells from the RCMP output . Decodes the prepend word to determine whether a cell is destined to the switch fabric or one of the PHY devices in the Transmit direction.
- Supports either Direct Multi-PHY interface or Indirect-addressing Multi-PHY interface for up to 32 PHY's.
- Supports cell format with one prepend and 27 words (16-bit wide). Prepend word is stripped off.
- Supports back-pressure flow control (referred to as stuttering in the code) from the switch.

ARCHITECTURE

Figure 2 is a block diagram of the Egress Routing Logic. The State machine handles all the cell-transfer handshaking. The interface to the RCMP consists of the control signals RCA, RSOC and RRDENB, plus the 16-bit data RDAT and the parity RPRTY. RCMP is a UTOPIA "slave", which receives the read-enable signal, RRDENB. The interface to the switch includes MCA and MRDENB. The switch is assumed to be a UTOPIA "master", which sources the read-enable signal, MRDENB. The interface to the PHY devices consists of the 4 PHYTCA's, the 4 PHYWRENB's (in Direct mode), and the 5-bit PHY address, called PADD (in Indirect-addressing mode). All PHY devices are assumed to be UTOPIA "slaves". The retimed data and parity output (ODAT and OPRTY) are connected to both the switch and the PHY devices.

Note that signals that end with a 'B' are active-low (ie. asserted if equal to 0).

Figure 2 Egress Routing Design Block Diagram

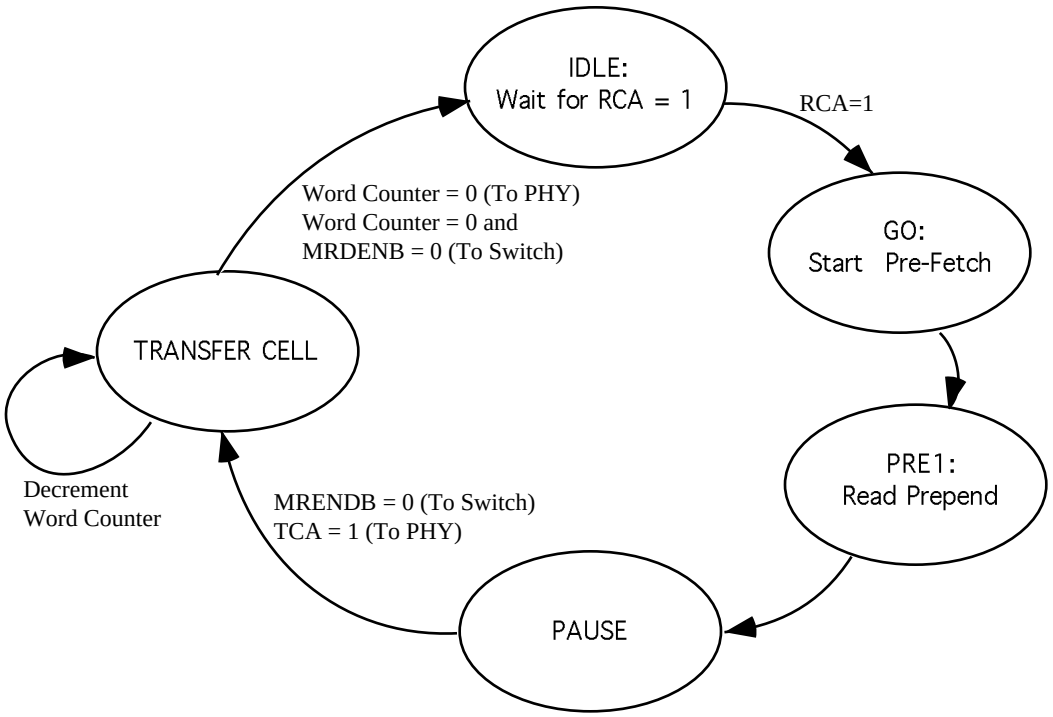


OPERATION

There are two modes of operation, direct and indirect mode, which correspond to the interface mode to the PHY devices. The interface to the switch is identical in both modes.

The state diagram is shown in Figure 3 as a reference.

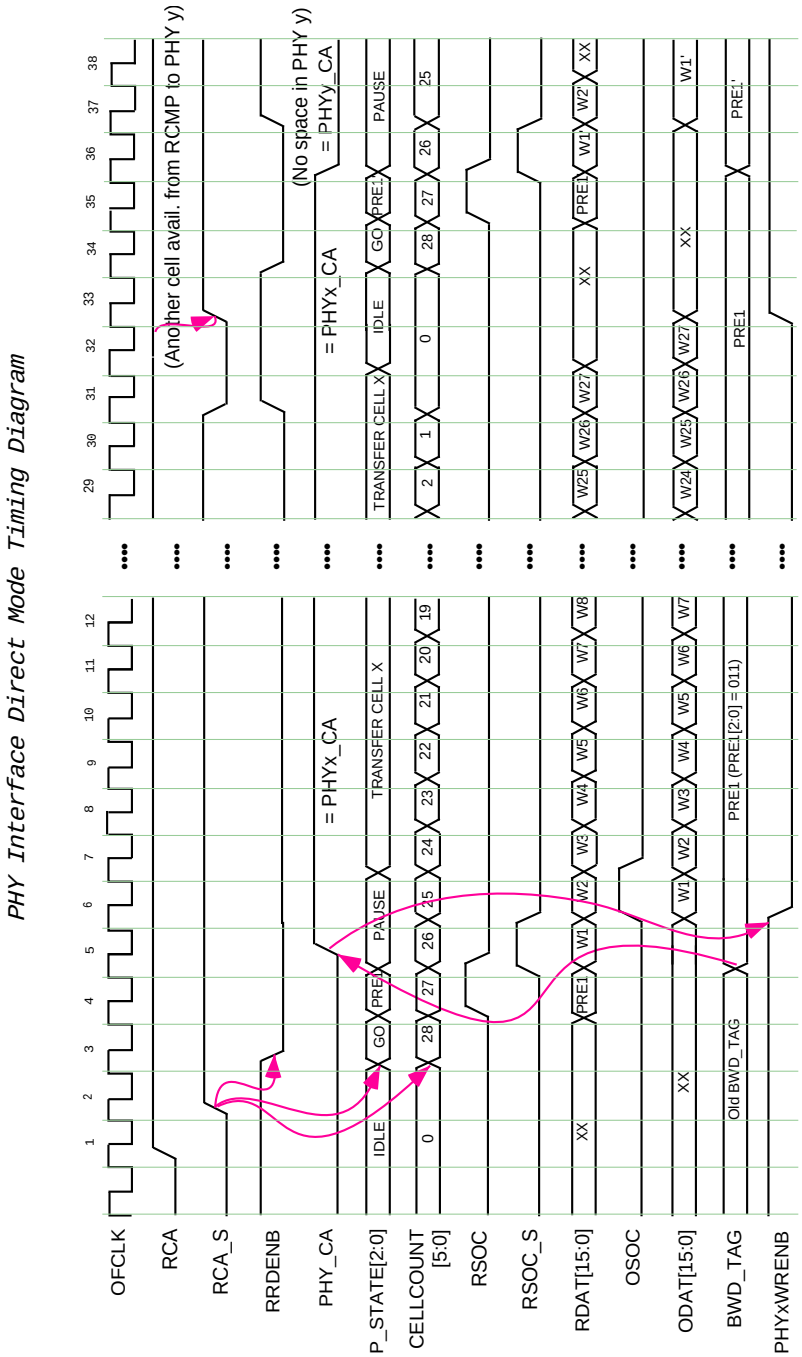
Figure 3 Egress Routing State Machine Diagram



Direct Mode

The operation in the Direct mode for **cell transfer to the PHY devices** is shown in the timing diagram in Figure 4. The state machine starts off in the IDLE mode, waiting for RCA to be asserted by the RCMP, indicating a cell is available to be transferred. Once RCA is sampled high, 3 words are "pre-read" from the RCMP, with the first word containing the backward routing tag. The following describes in detail the sequence of events for each cell transfer:

Figure 4 PHY Interface Direct Mode Timing Diagram



Clock cycle 1: RCMP indicates a cell is available by asserting RCA.

Clock cycle 2: Glue logic samples the RCA.

Clock cycle 3: Glue logic asserts RRDENB to read out the first prepend word. P_STATE becomes GO, and the CELL_COUNT initializes to 28 (which is the number of words to be read from the RCMP).

Clock cycle 4: The first prepend word (RDAT[15:0]) and parity (not shown), along with RSOC, are read out from the RCMP. P_STATE becomes PRE1, which aligns with RDAT.

Clock cycle 5: Glue logic samples the prepend word, the parity (not shown) and RSOC(as RSOC_S). Word 1 of the 27-word cell is read out from the RCMP. The BWD_TAG is decoded from the prepend word (Bits 15 to 8), which indicates that the cell is destined to one out of eight PHY devices. P_STATE enters PAUSE state, to wait for the PHY device to indicate that space is available in its transmit FIFO. Here the PHY_CA signal is shown to be asserted in this cycle.

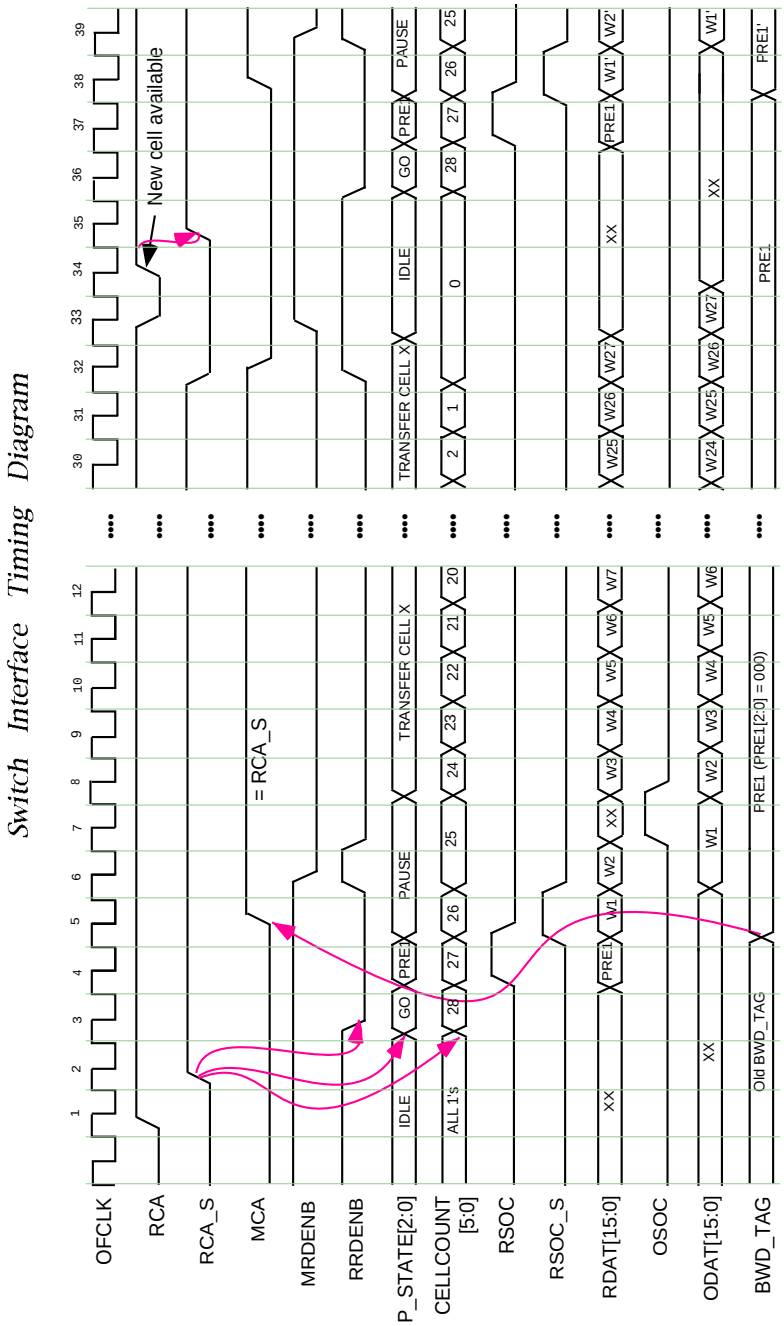
Clock cycle 6: The PHY_CA signal from the appropriate PHY device is sampled. If the PHY_CA signal is sampled to be high (as shown in the figure), RRDENB will continued to be asserted to read the rest of the 27-word cell from the RCMP. Otherwise, the glue logic will stay in the PAUSE state until PHY_CA is sampled high. Word 2 of the 27-word cell is read out from the RCMP. Word 1 and OSOC are presented on the output to the PHY device. PHYWRENB to the appropriate PHY device is asserted to write the cell into the PHY transmit FIFO.

Clock cycle 7-31: P_STATE enters the TRANSFER CELL state, and the RCMP to outputs the remaining 26 words of the cell. In Cycle 34, RRDENB is deasserted to end the cell transfer. RCA_S is forced low for the cell-transfer state machine to reset itself.

Clock cycle 32: P_STATE resets to the IDLE state after the cell transfer. Word 27 of the cell is output.

Clock cycle 33: Glue logic samples RCA. If RCA is sampled high (which is the case shown), indicating another cell is available from the RCMP, the entire cell transfer process begins again. Note that in this figure, the new prepend, PRE1', indicates a cell transfer to PHYy. However, PHYy does not have space in its FIFO, hence the low PHY_CA. The glue logic will stay in the PAUSE state until PHYy_CA is asserted.

Figure 5 Switch Interface Timing Diagram



The operation in the Direct mode for a **cell transfer to the switch** is shown in the timing diagram in Figure 5. The state machine starts off in the IDLE mode, waiting for RCA to be asserted by the RCMP, indicating a cell is available to be transferred. The prepend decode process is identical to the process when interfacing with the PHY devices. The difference here is that the cell transfer control is basically handed over to the switch (the switch controls the read-enable). The following describes the sequence of events for each cell transfer:

Clock cycle 1: RCMP indicates a cell is available by asserting RCA.

Clock cycle 2: Glue logic samples the RCA.

Clock cycle 3: Glue logic asserts RRDENB to read out the first prepend word. P_STATE becomes GO, and the CELL_COUNT initializes to 28.

Clock cycle 4: The first prepend word (RDAT[15:0]) and parity (not shown), along with RSOC, are read out from the RCMP. P_STATE becomes PRE1, which aligns with RDAT.

Clock cycle 5: Glue logic samples the prepend word, the parity (not shown) and RSOC(as RSOC_S). Word 1 of the 27-word cell is read out from the RCMP. The BWD_TAG is decoded from the prepend word (Bits 15 to 8), which indicates that the cell is destined to the switch. MCA is thus asserted. P_STATE enters PAUSE state, to wait for the switch to sample MCA and assert MRDENB.

Clock cycle 6: The switch samples MCA, asserts MRDENB and starts the cell transfer. Note that the switch now has full control of the cell transfer process. If it decides to postpone MRDENB assertion, the glue logic will stay in the PAUSE state. Word 2 is read from the RCMP. RRDENB is deasserted.

Clock cycle 7: MRDENB is sampled. RRDENB asserted as a result. Word 1 and OSOC is output to the switch.

Clock cycle 8: P_STATE enters the TRANSFER CELL state, and the RCMP outputs Word 2 of the cell.

Clock cycle 9-32: The rest of the ATM cell is transferred. RRDENB is deasserted in Cycle 32. MCA is forced to 0. RCA_S is forced low for the cell-transfer state machine to reset itself.

Clock cycle 33: P_STATE enters the IDLE state. MRDENB is deasserted by the switch to end the cell transfer. Word 27 is output. Meanwhile, RCA from the RCMP is shown to have deasserted.

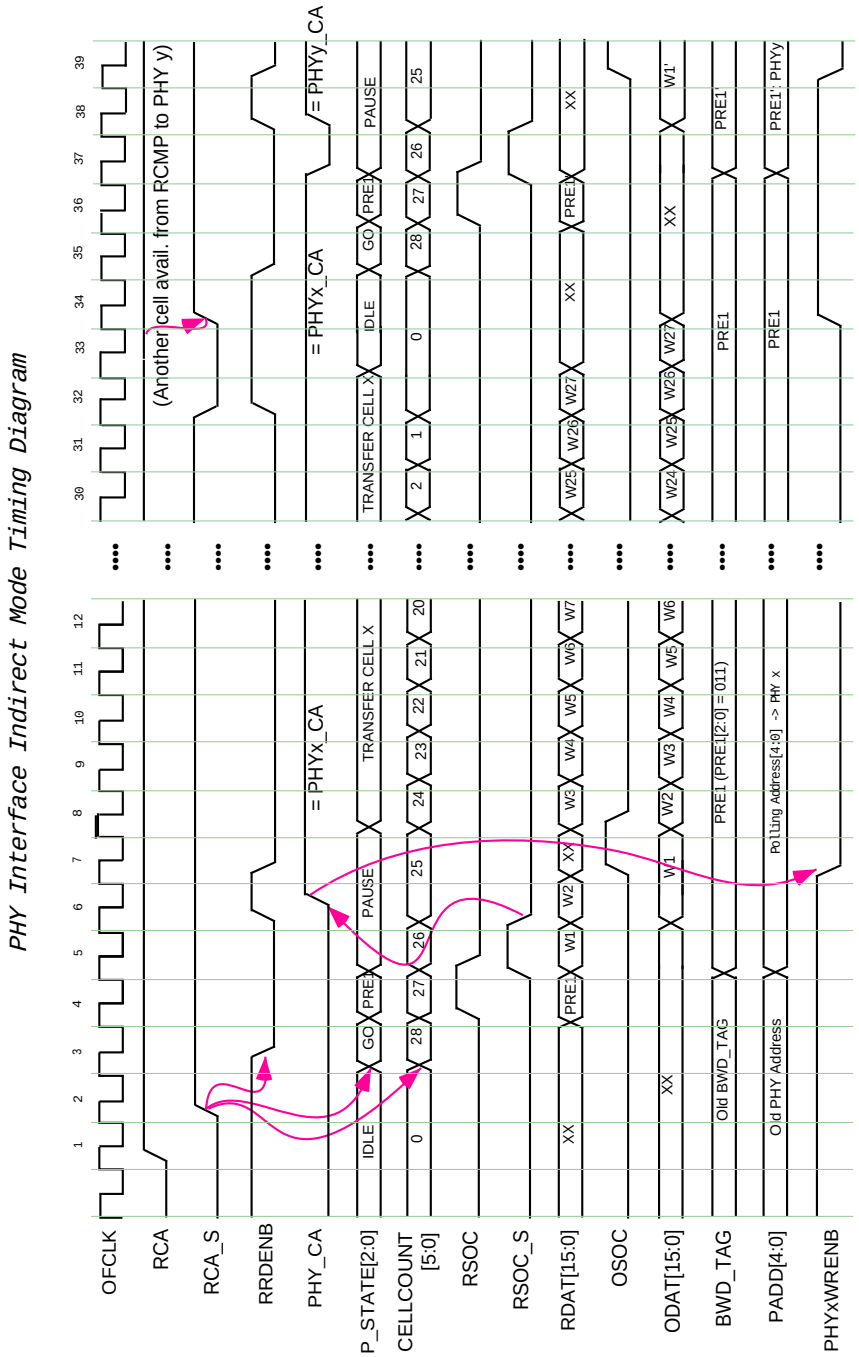
Clock cycle 34: Glue logic samples RCA. If RCA is sampled high (not in cycle 34), indicating another cell is available from the RCMP, the entire cell transfer process begins again. Otherwise, the glue logic will stay in the IDLE state until RCA is sampled high (as in Cycle 35).

Note that at any time during the cell transfer, the switch can deassert MRDENB to pause the transfer and resume at a later time. The cell count will stop decrementing until MRDENB is asserted again. This is to support the switch back-pressuring mechanism, and is called "stuttering" here. This mechanism will be described in a latter section.

Indirect Addressing Mode

The operation in the Indirect-addressing mode for a **cell transfer to the switch** is identical to that in the Direct mode. The operation in the Indirect-addressing mode for **cell transfer to the PHY device** is shown in the timing diagram in Figure 6. The state machine starts off in the IDLE mode, waiting for RCA to be asserted by the RCMP, indicating a cell is available to be transferred. The prepend decode process is identical to the process when interfacing with the PHY devices. The difference here is that, instead of the 4 direct PHYWRENB signals, a 5-bit PHY address (PADD) is output. PADD can be decoded to address a maximum of 32 PHY devices that are interfaced to this PLD. The PHY device samples this address and the selected PHY asserts the cell-available (PHY_CA) signal in the next cycle. The remaining sequence of events for the cell transfer is identical to the Direct Mode case.

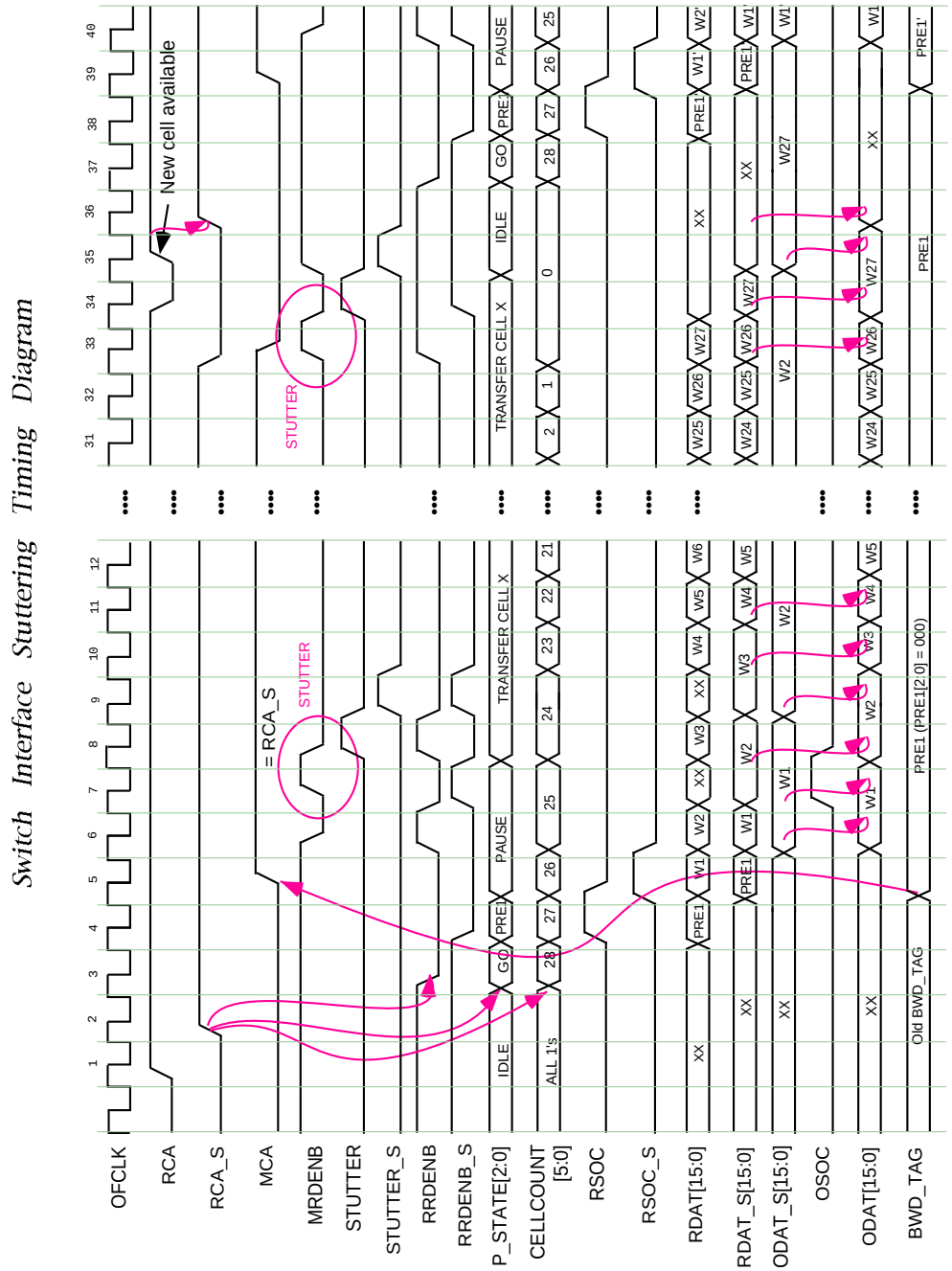
Figure 6 PHY Interface Indirect Mode Timing Diagram



Stuttering

Figure 7 shows two representative stuttering situations. The first case is when the switch deasserts read-enable just after the first word. The second case is when the switch deasserts read-enable just before the last word. In both cases, a special mechanism in the PLD effectively "buffers" the data read from the RCMP, and holds the data until the switch is ready to transfer again. Specifically, the internal registers RDAT_S and ODAT_S provides the "buffering" to read and hold the data in a stuttering situation.

Figure 7 Switch Interface Stuttering Timing Diagram



DESIGN CONSIDERATIONS

- The same clock, OFCLK, is assumed to be used by all interfaces.
- Signals on the RCMP interface, ie. rca, rsoc, data/parity, are all sampled and retimed in the PLD due to the propagation delay and setup times in the RCMP, so that this design can be used in 50MHz operation.
- With this design, it is transparent to the PHY device and the switch in terms of the UTOPIA protocol, since there is no extra delay between the cell-available signals, the read-enable signals and the data.
- With this PLD, some extra bandwidth is required to transition from cell to cell and to decode the routing tag. If the cell is going to a PHY, 3 extra cycles are required. If the cell is going to the switch, 4 extra cycles are required.
- This design can be modified to support an 8-bit interface, by changing the counter initial value.
- This design supports only one prepend. To support more prepend words, the counter initial value can be modified, and simply treat the additional prepend words as part of the cell, assuming the first prepend word contains the routing tag.
- The reset signal (RSTB), which is not shown in the timing diagrams, is active low and should be asserted for at least 2 clock cycles for a proper device reset.
- AC timing: This VHDL design can be implemented in any programmable device that contains adequate internal logic and pins. For 50MHz operation, the UTOPIA interface requirements are such that the maximum setup and hold times are 4ns and 1ns respectively, and the maximum propagation delay from the active clock edge is 16ns. The AC characteristics of the device must be such that it satisfies the timing requirements at the operating clock frequencies, with enough margins to cover routing delay and clock skew.

SUMMARY

A VHDL design for the RCMP Egress cell routing was described in this appnote. The design acts as the routing function which sits in between the RCMP output, the switch input, and the PHY inputs. It provides both the direct and indirect (using PHY addresses) interface modes with the PHY devices. It supports stuttering or back-pressuring by the switch input interface.

APPENDIX A: VHDL CODE

The following is the RCMP Egress Routing Logic design in VHDL.

```
--------------------------------------------------------------------------------
-- Name          : RCMP Egress Routing Logic for Appnote (AN6)
-- File Name     : RCMP_MPE.VHD
-- Company       : PMC Sierra, Inc
-- Date          :  Feb. 10, 1997
--
--
-- Description   :
-- Performs the multi-PHY Egress Routing logic for the RCMP-800/200.
--
-- The main function that this logic supports is Backward Routing.
-- It decodes the Backward Tag in the first prepend word (bits 15 to 8)
-- to decide whether the cell is destined for the ATM switch or
-- should be routed backwards to a PHY.
-- If the cell is going to a PHY, it then selects 1 out of 32 PHY devices to receive
-- the cell as decoded from the Backward Tag.
--
-- Supports 2 modes of PHY select:  1) Direct mode, 2) Indirect-addressing mode.
-- In Direct mode:
-- It monitors the cell-available (PHY_CA) signal from the correct PHY and provides
-- the appropriate control signal (PHYxWRENB) to the correct PHY.
-- In Indirect-addressing mode:
-- It outputs a 2-bit address (PADD[1:0]) extracted from the Backward Tag
-- (bits 1 to 0).  It monitors the cell-available (PHYxCA) signal from the
-- correct PHY,  as selected based on the Backward Tag (bits 4 to 2).
-- Then it provides the control signal (PHYxWRENB) to the correct PHY device.
-- The Indirect mode is used to interface with "Quad-PHY" devices, which have
-- PHY address-decoding built-in.  For example, the PM7344, the S/UNI-MPH
-- device, has 4 T1/E1 ports, which are selected based on a 2-bit address.
--
-- If the cell is going to the switch, it then proceed with the regular cell transfer,
-- using MCA and MRDENB signals for control.  "Stuttering", or deassertion of
-- MRDENB, during cell transfer by the switch is supported.
--
-- Note that RSOC from the RCMP must align with the first prepend word.  It is
-- not optional.
--
-- Only 16-bit interface supported.  Cellcounts will need to be modified for
-- 8-bit interface.
--
-- Supports prepend only.   Prepend word is stripped off at output.
--
-- Supports only 27-word cells (plus the prepend).
```

```
--
--  The output data, ODAT[15:0], and OSOC should be connected to BOTH
--  the switch and the PHY's.
--
--  Note that the this logic always "preread" the cell words, so that there will not be
--  any extra clock cycle latency from the point of view of the switch.  (ie. when the
--  switch asserts MRDENB, cell data will be valid to be sampled at the end of  the
--  NEXT clock cycle).
--
--   DISCLAIMER:  PMC does not guarantee that this VHDL code is correct and
--   functional in all applications.  It is intended to be used as a reference only.
-------------------------------------------------------------------------------------------

-- Compiler specific directives
USE work.int_math.all;
USE work.bv_math.all;

ENTITY rcmp_mpe IS PORT (
      rdat:       IN  BIT_VECTOR(15 downto 0); -- rcmp data
      rprty:      IN  BIT_VECTOR(1 downto 0);  -- rcmp data partity
      odat:      OUT BIT_VECTOR(15 downto 0); -- output data
      oprty:     OUT BIT_VECTOR(1 downto 0);  -- output data partity
      ia:             IN BIT; -- 1: Indirect-addressing mode, 0: Direct mode
      padd:          OUT BIT_VECTOR(4 downto 0);  -- PHY address in Indirect mode
      rsoc:          IN  BIT;  -- rcmp osoc
      osoc:          OUT BIT; -- output soc
      rca:            IN  BIT; -- rcmp oca
      rrdenb:        OUT BIT;
      mrdenb:       IN BIT;  -- ATM Switch read-enable
      mca:           OUT BIT;   -- cell-available to ATM Switch
      phy1tca:       IN  BIT; -- From PHY 1    in Direct mode; Shared phy_ca in Indirect mode
      phy1wrenb: OUT BIT; -- To PHY 1 in Direct mode; Shared phywrenb in Indirect mode
      phy2tca:       IN  BIT; -- From PHY 2
      phy2wrenb: OUT BIT;
      phy3tca:       IN  BIT; -- From PHY 3
      phy3wrenb: OUT BIT;
      phy4tca:       IN  BIT; -- From PHY 4
      phy4wrenb: OUT BIT;
      ofclk:          IN  BIT;
      rstb:           IN  BIT);

ATTRIBUTE order_code of rcmp_mpe:ENTITY is "CY7C374-83JC";
ATTRIBUTE part_name of rcmp_mpe:ENTITY IS "C374";

ATTRIBUTE pin_numbers of rcmp_mpe:ENTITY IS
              "rdat(0):76    rdat(1):77    rdat(2):78   rdat(3):79   "
            & "rdat(4):80    rdat(5):81    rdat(6):82   rdat(7):83   "
            & "rdat(8):3     rdat(9):4     rdat(10):5   rdat(11):6   "
            & "rdat(12):7    rdat(13):8    rdat(14):9   rdat(15):10  "
            & "rprty(0):73  rprty(1):75                    "
```

```
          & "odat(0):51   odat(1):50   odat(2):49  odat(3):48  "
          & "odat(4):47   odat(5):46   odat(6):45  odat(7):58  "
          & "odat(8):40   odat(9):39   odat(10):38 odat(11):37 "
          & "odat(12):36  odat(13):35  odat(14):34 odat(15):33 "
          & "oprty(0):54  oprty(1):52                          "
          & "ia:27          "
         & "padd(0):15   padd(1):16   padd(2):17   padd(3):18  padd(4):19  "
          & "rrdenb:70     rca:71       rsoc:72               "
          & "mrdenb:12    mca:13               "
          & "phy3tca:60    phy3wrenb:61               "
          & "phy2wrenb:56  phy4tca:57                 "
          & "phy1wrenb:30    phy1tca:29      osoc:31              "
          & "phy2tca:28    phy4wrenb:66                         "
          & "ofclk:65      rstb:41";
END rcmp_mpe;

ARCHITECTURE behav OF rcmp_mpe IS

  SUBTYPE wcount IS integer range 0 to 63;

  SIGNAL phywrenb:     bit;
  SIGNAL phy_ca:       bit;
  SIGNAL cellcount:    wcount;
  SIGNAL bwd_tag:      bit_vector(7 downto 0);
  SIGNAL rrdenbi:      bit;  -- internal version of rrdenb (output to RCMP)
  SIGNAL rrdenbi_s:    bit;  -- sampled version of rrdenbi
  SIGNAL stutter:      bit;  -- indicates mrdenb deassertion during cell transfer
  SIGNAL stutter_s:    bit;  -- stutter retimed

  TYPE state_type IS (idle, go,  transfer_pre1, transfer_cell, pause);
   signal n_state : state_type;
   signal p_state : state_type;

-- Sampled versions of rcmp outputs
   SIGNAL rdat_s:  bit_vector(15 downto 0);
   SIGNAL odat_s:  bit_vector(15 downto 0);
   SIGNAL rprty_s : bit_vector(1 downto 0);
   SIGNAL oprty_s : bit_vector(1 downto 0);
   SIGNAL rca_s : bit;
   SIGNAL rsoc_s: bit;

--  "to_switch" is set to "000", so that a default prepend of 0's will mean
--  cell transfer to switch
   CONSTANT to_switch:bit_vector(7 downto 0) := "00000000";
   CONSTANT to_phy:bit_vector(7 downto 0) := "00000011";

BEGIN
```

```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--  Process:  output_assignment
--  Purpose:  Assigns the output data, parity, soc, and other
--  control signals.
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

 output_assignment: PROCESS (rdat_s, rprty_s)
 BEGIN
    IF (p_state = pause) THEN
      odat <= odat_s; -- for Word 1 only
      oprty <= oprty_s;
    ELSIF (stutter_s = '1') THEN
      odat <= odat_s; -- Hold the word during stutter
      oprty <= oprty_s;
    ELSE
      odat <= rdat_s; -- 1-stage pipeline
      oprty <= rprty_s;
    END IF;

  rrdenb <= rrdenbi;
  padd <= bwd_tag(7 downto 3);
  IF (rrdenbi = '0' and cellcount = 25) THEN
    osoc <= '1';
  ELSE
    osoc <= '0';
  END IF;
 END PROCESS output_assignment;

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--  Process:  state_transition, state_assign
--  Purpose:  This process implements the main state machine that
--  controls the cell transfer/routing process.  It supports only prepends.
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

 state_transition: PROCESS (rca_s, p_state, cellcount)
 BEGIN
  CASE p_state IS
    WHEN idle =>
      -- The RCMP has a cell available, so get the first word and
      -- see which PHY it is destined to.
      IF (rca_s = '1') THEN
        n_state <= go;
      ELSE
        n_state <= idle;
      END IF;
    WHEN  go  =>
```

```
       --  Sample the first word.
           n_state <= transfer_pre1;
-- "Pause" to wait for bwd_tag to be decoded
      WHEN transfer_pre1 =>
         n_state <= pause;
      WHEN pause =>
        IF (rrdenbi = '0' and cellcount = 25) THEN
         n_state <= transfer_cell;
        ELSE
         n_state <= pause;
        END IF;
 -- End of "Pause" logic
      WHEN transfer_cell =>
        IF (cellcount = 0) THEN
          IF (bwd_tag(2 downto 0) = to_switch(2 downto 0)
               and mrdenb = '0') THEN
              n_state  <= idle;
          ELSIF (bwd_tag(2 downto 0) = to_phy(2 downto 0)) THEN
              n_state <= idle;
          ELSE
              n_state  <= transfer_cell;
          END IF;
        ELSIF (cellcount = 63) THEN
             n_state <= idle;
        ELSE
             n_state <= transfer_cell;
        END IF;
      WHEN others =>
        -- Included to handle any unknown case out of reset
        n_state <= idle;
    END CASE;
   END PROCESS state_transition;

    state_assign: PROCESS (rstb, ofclk)
    BEGIN
      IF (rstb = '0') THEN
        p_state <= idle;
      ELSIF (ofclk'EVENT and ofclk = '1') THEN
        p_state <= n_state;
      END IF;
    END PROCESS state_assign;

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--  Process:  ca_decode
--  Purpose:  This processdecodes the phy_ca.  Also controls
--  mca.
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

```
   ca_decode: PROCESS (p_state, bwd_tag)
   BEGIN
      IF (bwd_tag(2 downto 0) = to_switch(2 downto 0)
      and (p_state = pause or p_state = transfer_cell)) THEN
         mca <= rca_s;    -- cell is headed for Switch
      ELSE
         mca <= '0';
      END IF;

      IF (bwd_tag(2 downto 0) = to_phy(2 downto 0)) THEN
       IF(ia = '1') THEN
         IF (rsoc_s = '1') THEN
           phy_ca <= '0';  -- Null phy_ca until PHY device have
                           -- time to output valid tca
         ELSE
           phy_ca <= phy1tca; -- All PHY's share phy1tca
         END IF;
       ELSE
        CASE bwd_tag(4 downto 3) IS
         WHEN "00" =>
           phy_ca <= phy1tca;
         WHEN "01" =>
           phy_ca <= phy2tca;
         WHEN "10" =>
           phy_ca <= phy3tca;
         WHEN others =>
           phy_ca <= phy4tca;
        END CASE;
       END IF;
      ELSE
        phy_ca <= '0';
      END IF;
   END PROCESS ca_decode;

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
--  Process:  state_decode
--  Purpose:  This process implements UTOPIA signal processing and
--  Backward routing processing functions.
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

   state_decode: PROCESS (rstb, ofclk)
    BEGIN
     IF (rstb = '0') THEN
        rdat_s <= "0000000000000000";
        rprty_s <= "00";
        rsoc_s <= '0';
        bwd_tag <= to_switch;
     ELSIF (ofclk'EVENT and ofclk = '1') THEN
```

```
--------------------------------------------------------------------------------
-- Section:  UTOPIA signal processing
-- Purpose:  This section samples the RDAT, RPRTY, RSOC, RCA signals
-- from the RCMP.  It samples the prepend word that contains the
-- Backward Routing Information.
-- It generates the correct SOC, and thus strips the
-- prepend word if the cell is going to a PHY.
-- It generates the RRENDB signal to the RCMP, which controls the
-- actual cell transfer out of the RCMP.
--
-- Preread circuit used to eliminate extra latency to switch.  When RCA=1,
-- the prepend, Word 1, Word 2 will always be preread.
--
-- "Stutter" is used to handle the case where MRDENB is deasserted
-- during cell transfer.
--------------------------------------------------------------------------------

      IF (rrdenbi_s = '0') THEN
         rdat_s <= rdat;
         rprty_s <= rprty;
      ELSE
         rdat_s <= rdat_s;
         rprty_s <= rprty_s;
      END IF;

      IF (rsoc_s = '1') THEN
        odat_s <= rdat;
        oprty_s <= rprty;
      ELSIF (stutter = '1' and stutter_s = '0') THEN
        odat_s <= rdat_s;  -- Hold the data during stutter
        oprty_s <= rprty_s;
      ELSE
        odat_s <= odat_s;
        oprty_s <= oprty_s;
      END IF;

      rsoc_s <= rsoc;

      IF (p_state = transfer_cell
       and (cellcount = 1 or cellcount = 0)) THEN
       -- This is done just to force a transition on the sampled version of rca.
       -- so that the state machine can go to idle after a cell transfer
       rca_s <= '0';
      ELSE
       rca_s <= rca;
      END IF;

      IF (rsoc = '1') THEN
       bwd_tag  <= rdat(15 downto 8); -- Get Backward Tag from RDAT[15:8]
```

```
ELSE
  bwd_tag  <= bwd_tag;
END IF;

rrdenbi_s <= rrdenbi; -- for odat retime control
stutter_s <= stutter; -- for odat retime control during stutter

IF (rca_s = '1' and  p_state = idle) THEN
   rrdenbi <= '0';  -- Read prepend
ELSIF (bwd_tag(2 downto 0) = to_phy(2 downto 0)) THEN
   IF (p_state = go) THEN
     rrdenbi <= '0'; -- preread Word 1
   ELSIF (p_state = transfer_pre1) THEN
     rrdenbi <= '0'; -- preread Word 2
   ELSIF (p_state = pause and phy_ca = '1') THEN
     rrdenbi <= '0'; -- Start cell transfer to PHY; read Word 3
   ELSIF (p_state = transfer_cell and cellcount /= 1
         and cellcount /= 0) THEN
     rrdenbi <= '0';
   ELSE
     rrdenbi <= '1';
   END IF;
ELSIF (bwd_tag(2 downto 0) = to_switch(2 downto 0)) THEN
   IF (p_state = go) THEN
     rrdenbi <= '0'; -- preread Word 1
   ELSIF (p_state = transfer_pre1) THEN
     rrdenbi <= '0'; -- preread Word 2
   ELSIF (p_state = pause) THEN
     rrdenbi <= mrdenb;  -- transfer control to switch
   ELSIF (p_state = transfer_cell and cellcount /= 0
            and cellcount /= 1 and cellcount /= 61 and
            cellcount /= 62 and cellcount /= 63) THEN
     rrdenbi  <= mrdenb;
   ELSIF (p_state = transfer_cell and cellcount = 1) THEN
      IF (stutter = '1') THEN
        rrdenbi <= mrdenb;
       ELSE
        rrdenbi <= '1';
       END IF;
  ELSE
     rrdenbi  <= '1';
  END IF;
ELSE
     rrdenbi <= '1';
END IF;

IF (rca_s = '1' and  p_state = idle) THEN
   stutter <= '0';  -- initialize stutter
ELSIF (bwd_tag(2 downto 0) = to_switch(2 downto 0)) THEN
  IF (rrdenbi = '0' and cellcount = 25) THEN
```

```
          stutter <= mrdenb; -- To handle stuttering right after Word1
        ELSIF (p_state = transfer_cell) THEN
          stutter <= mrdenb;
        ELSE
          stutter  <= '0';
        END IF;
      ELSE
          stutter <= '0';
      END IF;

-------------------------------------------------------------------------------
-- Section:  Backward Tag Decoding for PHY
-- Purpose:  This section decodes the backward tag to determine
-- if the cell is to be routed to the switch or to a PHY.
-- If bwd_tag[2:0] = to_phy[2:0], then use bwd_tag[4:3] to determine PHY#
-- else the cell is going to the switch.
-- The decoding circuit to handle cells going to the switch is found
-- in the ca_decode section.
-------------------------------------------------------------------------------

      IF ((p_state = transfer_cell) or
          (p_state = pause and phy_ca = '1')) THEN
        IF (bwd_tag(2 downto 0) = to_phy(2 downto 0)) THEN
         IF (ia = '1') THEN   -- Indirect Addressing Mode
            phy1wrenb <= '0';
            phy2wrenb <= '1';
            phy3wrenb <= '1';
            phy4wrenb <= '1';
         ELSE  -- Direct Mode
          CASE bwd_tag(4 downto 3) IS
           WHEN "00" =>
            phy1wrenb <= '0';
            phy2wrenb <= '1';
            phy3wrenb <= '1';
            phy4wrenb <= '1';
           WHEN "01" =>
            phy1wrenb <= '1';
            phy2wrenb <= '0';
            phy3wrenb <= '1';
            phy4wrenb <= '1';
           WHEN "10" =>
            phy1wrenb <= '1';
            phy2wrenb <= '1';
            phy3wrenb <= '0';
            phy4wrenb <= '1';
           WHEN others =>
            phy1wrenb <= '1';
            phy2wrenb <= '1';
            phy3wrenb <= '1';
            phy4wrenb <= '0';
```

```
          END CASE;
         END IF; -- ia
        ELSE
           phy1wrenb <= '1';
           phy2wrenb <= '1';
           phy3wrenb <= '1';
           phy4wrenb <= '1';
       END IF; -- bwd_tag
      ELSE
       phy1wrenb <= '1';
       phy2wrenb <= '1';
       phy3wrenb <= '1';
       phy4wrenb <= '1';
      END IF; -- p_state
    END IF; -- rstb

   END PROCESS state_decode;


-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
--  Process:  counter
--  Purpose:  This process implements a cell transfer counter.
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

   counter : PROCESS (rstb, ofclk)

   BEGIN  --  PROCESS counter
    IF (rstb = '0') THEN
     cellcount  <= 0;
    ELSIF (ofclk'EVENT and (ofclk = '1')) THEN
     IF (rca_s = '1' and p_state = idle) THEN
          cellcount <= 28; -- 28 = prepend + (27-word cell) + 1
     ELSIF ((bwd_tag(2 downto 0) = to_switch(2 downto 0))
               and p_state = transfer_cell and
              ((cellcount = 0) or (cellcount = 1))
               and mrdenb = '1' and stutter = '0') THEN
             cellcount <= 63 - cellcount - 1;
     ELSIF ((bwd_tag(2 downto 0) = to_switch(2 downto 0))
               and p_state = transfer_cell and
              ((cellcount = 61) or (cellcount = 62)) and mrdenb = '0') THEN
             cellcount <= cellcount + 1;
     ELSIF ((bwd_tag(2 downto 0) = to_switch(2 downto 0))
               and cellcount = 63) THEN
             cellcount <= 0;
     ELSIF (rrdenbi = '0') THEN
        cellcount <= cellcount - 1;
     ELSE
      cellcount <= cellcount;
```

```
      END IF;
    END IF;
  END PROCESS counter;

END behav;
```