Texas Instruments

Accessing the TNETA1500 Control Registers With the TNETA1570 SAR

Application Report

1996

Networking Products Group

Texas Instruments

Printed in U.S.A.
0196–AL

SDNA002

Texas Instruments

Accessing the TNETA1500 Control Registers With the TNETA1570 SAR

1996

Networking Products Group

Printed in U.S.A.
0196–AL

SDNA002

Application Report

Accessing the TNETA1500 Control Registers With the TNETA1570 SAR

SDNA002
January 1996

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Contents

List of Illustrations

Introduction

This application report assists the system, board, and software designers in the use of the SAR2PHY field-programmable gate array (FPGA) described herein to access the TNETA1500 control registers via the TNETA1570 control-memory interface. Descriptions of the TNETA1570 control-memory interface and the TNETA1500 controller interface are presented, followed by a hardware description of the SAR2PHY FPGA. Finally, the TNETA1500 controller interface is presented as it appears to the programmer.

TNETA1570 Control-Memory Interface

The TNETA1570 control-memory interface is an asynchronous interface and is used by the TNETA1570 to access its local-control memory. Read and write waveforms are shown in Figures 1 and 2 along with timing specifications and requirements.

timing requirements

NO.			MIN	MAX	UNIT
1	$t_{su(CMDATA)}$	Setup time, CMDATA31–CMDATA0 valid before $\overline{CMOE}\uparrow$	10		ns
2	$t_{h(CMDATA)}$	Hold time, CMDATA31–CMDATA0 valid after $\overline{CMOE}\uparrow$	0		ns

operating characteristics

NO.			MIN	MAX	UNIT
3	$t_{d(CMDATA)}$	Delay time, CMDATA17–CMDATA0 valid to $\overline{CMOE}\downarrow$		3	ns

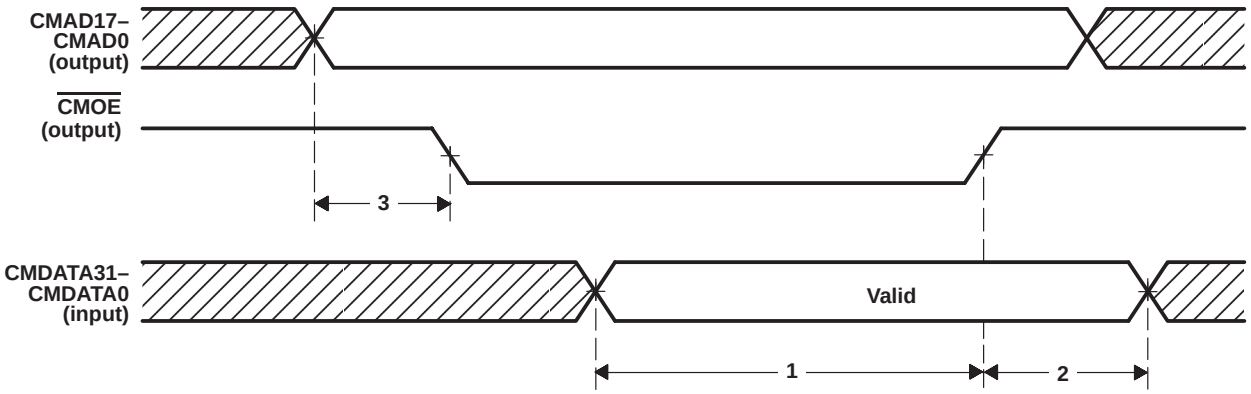

Figure 1. TNETA1570 Control-Memory Interface Read-Cycle Timing

operating characteristics

NO.			MIN	MAX	UNIT
1	$t_{w(CMWEL)}$	Pulse duration, $\overline{\text{CMWE}}$ low	28	31	ns
2	$t_{d(CMWE)1}$	Delay time, CMAD17–CMAD0 valid to $\overline{\text{CMWE}}\downarrow$	12	15	ns
3	$t_{d(CMWE)2}$	Delay time, CMDATA31–CMDATA0 valid to $\overline{\text{CMWE}}\downarrow$	10	14	ns
4	$t_{d(CMDATA)}$	Delay time, $\overline{\text{CMWE}}\uparrow$ to CMDATA31–CMDATA0 invalid	15	19	ns

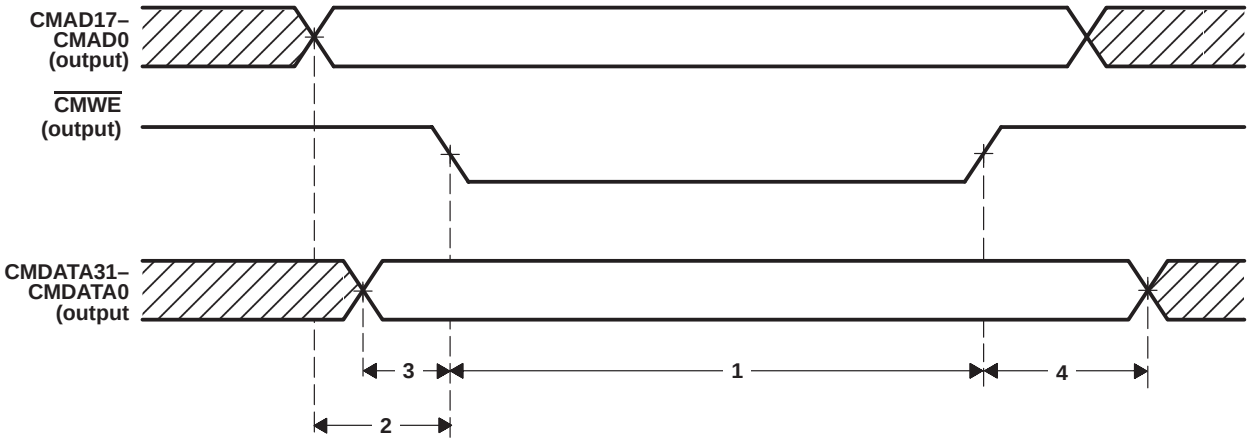

Figure 2. TNETA1570 Control-Memory Interface Write-Cycle Timing

TNETA1500 Controller Interface

The TNETA1500 controller interface allows access to the TNETA1500 internal registers. Timing specifications and requirements for the interface are shown in Figures 3 and 4.

timing requirements

NO.			MIN	MAX	UNIT
1	$t_{su(A0–A7)}$	Setup time, A0–A7 valid before $\overline{SEL}\downarrow$	0		ns
2	$t_{su(RD/WR)}$	Setup time, RD/$\overline{WR}$ high before $\overline{SEL}\downarrow$	3		ns
3	$t_{h(A0–A7)}$	Hold time, A0–A7 valid after $\overline{SEL}\uparrow$	4		ns
4	$t_{h(RD/WR)}$	Hold time, RD/$\overline{WR}$ high after $\overline{SEL}\uparrow$	35		ns
5	$t_{w(SEL)}$	Pulse duration, $\overline{SEL}$ low	35		ns

operating characteristics

NO.			MIN	MAX	UNIT
6	$t_{d(D0–D7)1}$	Delay time, $\overline{SEL}\downarrow$ to D0–D7 valid	7	25	ns
7	$t_{d(D0–D7)2}$	Delay time, $\overline{SEL}\uparrow$ to D0–D7 invalid	5	18	ns
8	t_{PHL}	Propagation delay time, $\overline{SEL}\downarrow$ to $\overline{READY}\downarrow$	7	26	ns
9	t_{PLH}	Propagation delay time, $\overline{SEL}\uparrow$ to $\overline{READY}\uparrow$	3	15	ns

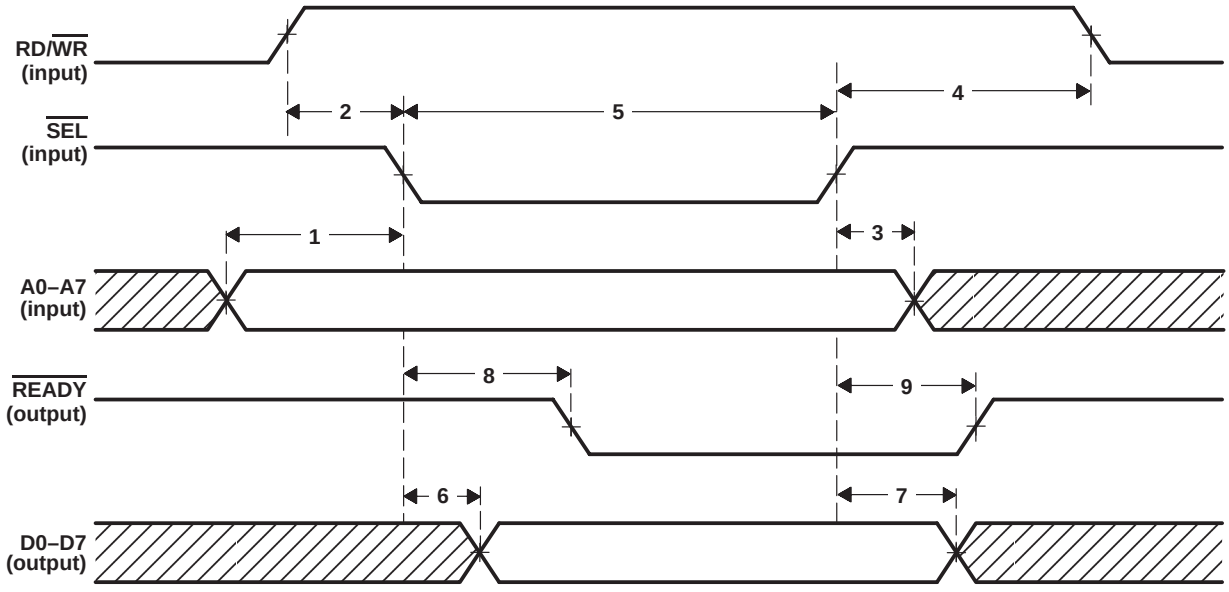

Figure 3. TNETA1500 Controller-Interface Read-Cycle Timing

timing requirements

NO.			MIN	MAX	UNIT
1	$t_{su(A0–A7)}$	Setup time, A0–A7 valid before $\overline{SEL}\downarrow$	0		ns
2	$t_{su(D0–D7)}$	Setup time, D0–D7 valid before $\overline{SEL}\uparrow$	5		ns
3	$t_{su(RD/WR)}$	Setup time, RD/$\overline{WR}$ low before $\overline{SEL}\downarrow$	1		ns
4	$t_{h(D0–D7)}$	Hold time, D0–D7 valid after $\overline{SEL}\uparrow$	0		ns
5	$t_{h(RD/WR)}$	Hold time, RD/$\overline{WR}$ low after $\overline{SEL}\downarrow$	35		ns
6	$t_{w(SEL)}$	Pulse duration, $\overline{SEL}$ low	35		ns

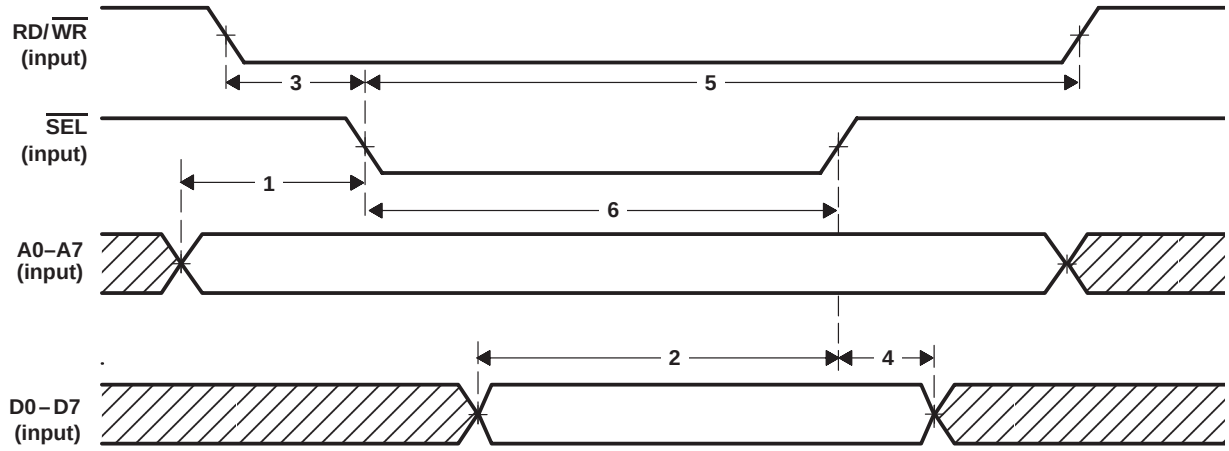

Figure 4. TNETA1500 Controller-Interface Write-Cycle Timing

SAR2PHY Field-Programmable Gate Array (FPGA)

The SAR2PHY (segmentation and reassembly to physical layer) describes the function of the FPGA as it bridges the TNETA1570 to the TNETA1500.

The SAR2PHY FPGA acts as a bridge between the TNETA1570 control-memory bus and the TNETA1500 controller interface previously described. The FPGA maps the TNETA1500 internal registers into the upper (unused) portion of the TNETA1570 control-memory space. It also issues high-priority segmentation requests to the TNETA1570 on command. A block diagram of the SAR2PHY connection to the TNETA1570 and TNETA1500 is shown in Figure 5.

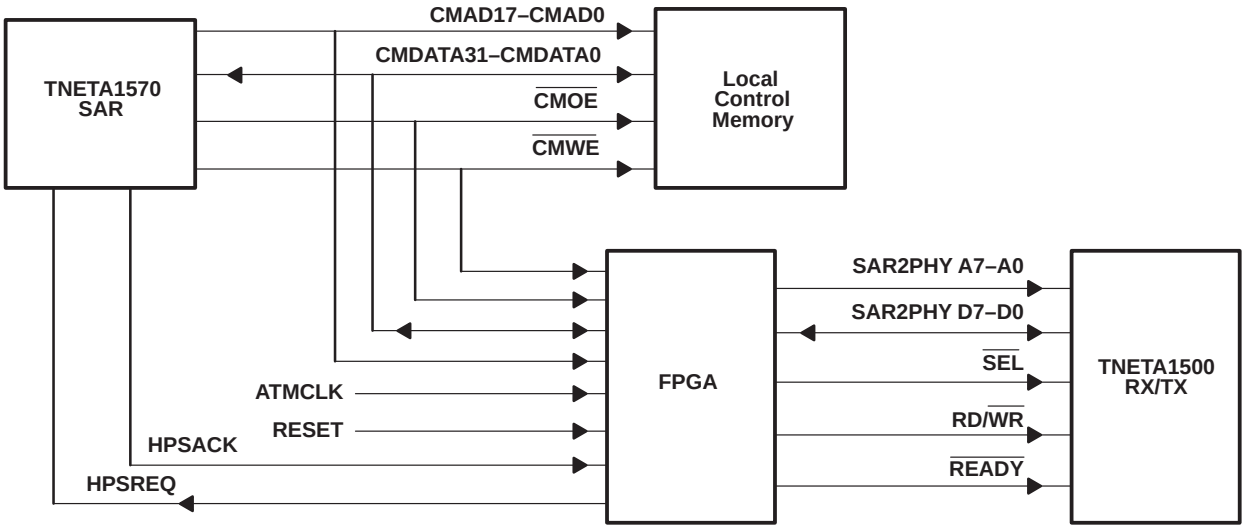

Figure 5. SAR2PHY FPGA Connection to the TNETA1570 and TNETA1500

The SAR2PHY FPGA is designed to respond to control-memory accesses in which CMAD17 is high, giving the TNETA1500 an effective control-memory base address of 0x20000h. When a control-memory access with CMAD17 high is detected by the SAR2PHY FPGA, a read or write to the TNETA1500 controller interface is executed using CMAD7-CMAD0 as the address. Read and write cycles generated by the SAR2PHY FPGA are similar to those shown in Figures 3 and 4.

The command set of the SAR2PHY FPGA is shown in the following table. In addition to CMAD17 enabling the FPGA, control-memory address bits 5, 6, and 7 also have special meaning when CMAD17 is also asserted.

COMMAND	CONTROL-MEMORY ADDRESS (17–0)	NOTES
FPGA enabled	1xxxxxxxxxxxxxxxx (bits 17 high)	
Software reset of FPGA	1xxxxxxxx1xxxxxx (bits 17 and 7 high)	$\overline{\text{CMOE}}$ must be low.
Toggle FPGA control-memory data-bus output buffers	1xxxxxxxxx1xxxxx (bits 17 and 6 high)	$\overline{\text{CMOE}}$ must be low.
Issue high-priority segmentation request	1xxxxxxxxxx1xxxx (bits 17 and 5 high)	$\overline{\text{CMOE}}$ must be low.

A brief discussion of the above commands and their purpose follows.

Software Reset of the FPGA

A software-reset function is included in the FPGA placing it in a known state. The SAR2PHY FPGA resets itself on any control-memory read ($\overline{\text{CMOE}}$ low) in which address bits 17 and 7 are high. Software reset clears the FPGA data and address latches and disables its control-memory data-bus output buffers.

Toggling and Resetting the Control-Memory Data Bus

To facilitate reads from the TNETA1500 through the FPGA, which is not capable of speeds implied by a 15-ns SRAM interface, a bus on/off feature is included in the SAR2PHY FPGA. To read an internal register from the TNETA1500, control of the TNETA1570 control-memory data bus is given to the SAR2PHY FPGA by enabling its data-bus output buffers. Two subsequent reads from the appropriate address yields the correct data and the FPGA data-bus output buffers then can be disabled.

Any control-memory read ($\overline{\text{CMOE}}$ low) in which control-memory address bits 17 and 6 are high, toggles the SAR2PHY FPGA control-memory data-bus drivers on or off depending on which state they are currently in.

Issuing a High-Priority Segmentation Request (HPSREQ)

Any control-memory read ($\overline{\text{CMOE}}$ low) in which control-memory address bits 17 and 5 are high, causes the SAR2PHY FPGA to issue a high-priority segmentation request to the TNETA1570 by asserting HPSREQ. The FPGA continues to assert HPSREQ until the TNETA1570 asserts high-priority segmentation acknowledge (HPSACK) or until the FPGA is reset.

WARNING:

Control-memory accesses directed to the FPGA (address bit 17 set) when address bits 7, 6, and 5 are simultaneously set can cause unpredictable operation and should be avoided.

Programmer's Reference

The following table is a map of the TNETA1570 control-memory space. Included in the map are the TNETA1500 internal registers and the SAR2PHY FPGA command set.

MEMORY BLOCK	CONTROL-MEMORY ADDRESS	PCI OFFSET
Scheduler table	00000–00C1B	00000–0306C
Reserved	00C1C–00DFF	03070–037FC
Free buffer pointers	00E00–00FFF	03800–03FFC
Rx VPI/VCI DMA pointer table	01000–01FFF	04000–04FFC
Reserved	02000	08000
Tx DMA state table	02001–03FFF	08004–0FFFC
Rx DMA state table	04000–3FFFF	10000–FFFFC
TNETA1500 INTERNAL REGISTERS		
Interrupt register 1	20000	80000
Interrupt register 2	20001	80004
Interrupt register 3	20002	80008
ID register	20003	8000C
Control register 1	20005	80014
Control register 2	20006	80018
Interrupt-mask register 1	20007	8001C
Interrupt-mask register 2	20008	80020
Interrupt-mask register 3	20009	80024
Multierrored cell counter	2000A	80028
Path-RDI soak counter	2000B	8002C
B1-block error counter	2000D	80034
B2-block error counter	2000F	8003C
B3-block error counter	20011	80044
B1 coding-violation counter (LSB)	20012	80048
B1 coding-violation counter (MSB)	20013	8004C
B2 coding-violation counter (LSB)	20014	80050
B2 coding-violation counter	20015	80054
B2 coding-violation counter (MSB)	20016	80058
B3 coding-violation counter (LSB)	20017	8005C
B3 coding-violation counter (MSB)	20018	80060
SAR2PHY FPGA COMMAND SET		
Issue HPSREQ	20020	80080 (read only)
Toggle data-bus drivers	20040	80100 (read only)
FPGA software reset	20080	80200 (read only)

Reading From the TNETA1500

As previously noted, the SAR2PHY control-memory data-bus output buffers must be enabled before executing a read from the TNETA1500. Care should be taken when enabling these buffers, since contention on the control-memory data bus could result if the TNETA1570 is executing control-memory accesses at the same time. The best way to avoid this is to disable the TNETA1570 transmit and receive by clearing the appropriate bits in its internal-configuration register. This can be done by turning on the FPGA control-memory data-bus buffers (read to PCI offset 0x80100), executing two reads to the desired address (second read returns valid data), and turning off the control-memory data-bus output buffers (read to PCI offset 0x80100 or 0x80200).

Turning off the TNETA1570 before reading the FPGA is only necessary because the FPGA used in this example is too slow to meet the TNETA1570 control-memory read-cycle timing. This requirement does not exist for an implementation that meets the timing parameters.

Writing to the TNETA1500

Writing to the TNETA1500 through the SAR2PHY FPGA requires only a single PCI write to the appropriate offset.

Resetting the SAR2PHY FPGA

A PCI read from offset 0x80200 resets (turns off) the FPGA control-memory data-bus drivers and clears all its internal registers.

Issuing a High-Priority Segmentation Request

A PCI read from offset 0x80080 causes the FPGA to issue a high-priority segmentation request to the TNETA1570.

WARNING:

As previously discussed, control-memory accesses when address bit 17, 7, 6, or 5 are simultaneously set should be avoided; therefore, the programmer should be careful not to initiate transactions to the following PCI offsets:

- 0x80180 – 0x801ff
- 0x80280 – 0x802ff
- 0x80300 – 0x803ff

Appendix A

Example of Verilog Code for PHY Interface

This appendix contains an example of Verilog code for logic that proves a method for accessing the registers on a physical-layer device using the control-memory interface on the TNETA1570. The target device for this example is a TPC1200 FPGA device; however, other devices could be used.

Example Code

This example code consists of the following three modules:

1. sar2phy.v is the Verilog description of the FPGA logic.
2. S2P_TOP is the top-level module that ties iobuf.v and sar2phy.v together.
3. iobuf.v is the Verilog description for the TPC1200 series I/O wrapper (instantiates I/O buffers).

```
/* This file provides the HDL code for the FPGA Interface device
 * (SAR2PHY) which provides control logic for the TNETA1500's controller
 * interface. The controller interface provides access to the internal
 * memory locations which contain the control registers, interrupt
 * registers, interrupt mask registers, and the ID register.
 *
 * Name      : SAR2PHY.v
 *
 * Interface Summary :
 *  The SAR2PHY block interfaces with the TNETA1570 and the TNETA1500.
 *
 * External Memory Interface :
 *  Control Memory is set up in a 512KX32 Configuration,with 32
 *  bidirectional data pins and 18 address pins. The bidirectional data
 *  pins are accessed via separate32–bit input and output buses, with an
 *  output enable signal to the I/O driver.
 *
 *  A read/write_ signal provides active low write or read, with an
 *  output enable signal to enable data output from the memory.
 *
 * General Inputs and Outputs from TNETA1570
 *  This module uses the inverted PCIbus reset signal and PCI clock.
 *                            -------
 *
 *
 */
'timescale 1ns / 1ns


module sar2phy(
       // TNETA1570 Control Pins
       cm_we_i_, cm_oe_i_,
       // TNETA1500 Control Pins
       rd_wr_, sel_o_, ready,
       // Generic Inputs and Outputs
       atm_clk, reset,
       // Data Bus
       cm_data_in, cm_data_out, cmdata_en,
       sar2phyif_phy_d_in, sar2phyif_phy_d_out,
       sar2phyif_en,
       // Address Buses
       cm_ad, sar2phyif_phy_a,
       // High Priority Segmentation
       hpsreq, hpsack
       ) ;

// Buses from TNETA1570/SRAM

  input [17:0]
       cm_ad            // Control Memory Address Bits
             ;

                        // BIDIRECTIONAL CONTROL MEMORY DATA BUS
  input [31:0]
       cm_data_in       // Data from TNETA1570 to TNETA1500
             ;

  output [31:0]
       cm_data_out      // Data from TNETA1500 to TNETA1570
             ;

// Buses from SAR2PHY

  output [7:0]
         sar2phyif_phy_a  // Address Bus to TNETA1500
             ;

                          // BIDIRECTIONAL SAR2PHY(FPGA) DATA BUS

  input [7:0] sar2phyif_phy_d_in;       // Data from TNETA1500 to FPGA

  output [7:0] sar2phyif_phy_d_out;  // Data from FPGA to TNETA1500

  input   cm_we_i_,        // TNETA1570 Active Low Write Enable Signal
          cm_oe_i_         // TNETA1570 Active Low Output Enable Signal
             ;
```

```
  output sar2phyif_en,
       cmdata_en        // Bidirectional Bus Enables
              ;

// General Inputs and Outputs
  input   atm_clk,      // PCIbus Clock
                        //*****************************************
          reset         // PCIbus Reset *NOTE: ACTUAL BOARD DESIGN
                                    RESET ACTIVE HIGH
                        //*****************************************
              ;
  output  sel_o_        // SAR2PHY Active Low Select Enable Signal
              ;
  output  rd_wr_        // TNETA1500 rd/wr_ signal from fpga
              ;
  input   ready         // TNETA1500 Ready Signal to TNETA1570
              ;
  output  hpsreq        // High Priority Segmentation Request
              ;
  input   hpsack        // High Priority Segmentation Acknowledge
              ;

// Register Definitions for SAR2PHYIF
  reg [7:0]
     sar2phyif_phy_d_out,  // Data Bus to TNETA1500
     phy_data_in_lat
           ;
  reg [31:0]
       cm_data_out         // Data Bus to TNETA1500
              ;
  reg [7:0]
       sar2phyif_phy_a     // Address Bus to TNETA1500
              ;
  reg  rd_wr_,
       hpsreq,             // High Priority Request
       sel_o_del,          // CM_DATA Buffer Enables
       sel_o_del_1,
       sel_o_del_2,
       sw_reset_del3,
       sw_reset_del2,
       sw_reset_del1,
       sw_reset,
       cmdata_en           // CM_DATA Buffer Enables
              ;

// GENERIC DELAY VALUES are in NS
‘define latdel   3         // Propagation Delay through D–Latch



// TRUE, FALSE,  NULL etc.
‘define TRUE 1             // Definition of TRUE
‘define FALSE 0            // Definition of FALSE
‘define NULL 0             // Definition of NULL
‘define actlowTRUE 0       // Definition of active low TRUE
‘define actlowFALSE 1      // Definition of active low FALSE

// Wire Declarations

wire int_reset = (reset || sw_reset );

wire   sel_o_;
wire   sel_o_set_ = ~((~ready && ~sel_o_del) || (int_reset) ||
                     (~rd_wr_ && ~ sel_o_del));
wire   #5 sel_o_read_reset = (cm_ad[17] && !cm_oe_i_);
wire   sel_o_writ_reset = (cm_ad[17] && !cm_we_i_);
wire   sel_o_reset_ = (~(sel_o_read_reset || sel_o_writ_reset) ||
                       ~sel_o_set_);
```

```
// S-R latch inference
wire   sr_lat_qb;
wire   sr_lat_q;

assign #1 sr_lat_qb = ~(sel_o_reset_ && sel_o_);
assign #1 sel_o_  = ~(sel_o_set_ && sr_lat_qb);

// Definition of Bidirectional DATA BUS

 assign sar2phyif_en = ~rd_wr_;  // Write Operation

// CODE STARTS HERE

// INFER A LATCH FOR WRITE OPERATION
always @ (sel_o_ or  cm_data_in or int_reset )
   if (int_reset )
          begin
            sar2phyif_phy_d_out <= #'latdel 'NULL;
          end
   else if (sel_o_)
          begin
            sar2phyif_phy_d_out <= #'latdel cm_data_in[7:0];
          end

// INFER A LATCH FOR READ OPERATION
always @ (cm_oe_i_ or phy_data_in_lat or int_reset )
   if (int_reset )
          begin
            cm_data_out <= #'latdel 'NULL;
          end
   else if (cm_oe_i_)
          begin
            cm_data_out <= #'latdel {24'd0,phy_data_in_lat[7:0]};
          end

// INFER A LATCH FOR DATA FROM PHY
always @ (sel_o_ or sar2phyif_phy_d_in or rd_wr_ or int_reset )
   if (int_reset )
          begin
            phy_data_in_lat <= #'latdel 'NULL;
          end
   else if (rd_wr_ && !sel_o_)
          begin
            phy_data_in_lat <= #'latdel sar2phyif_phy_d_in[7:0];
          end
```

```
// SEL_O_ is low for 3 ATM_CLK
always @ (negedge atm_clk or posedge int_reset )
   if (int_reset )
          begin
            sel_o_del <= #‘latdel ‘TRUE;
            sel_o_del_1 <= #‘latdel ‘TRUE;
            sel_o_del_2 <= #‘latdel ‘TRUE;
          end
   else
          begin
            sel_o_del <= #‘latdel sel_o_del_2;
            sel_o_del_1 <= #‘latdel sel_o_;
            sel_o_del_2 <= #‘latdel sel_o_del_1;
          end

// RD_WR_ is low for duration of SEL_O_ low
// TNETA1500 Address from FPGA
always @ (cm_oe_i_ or sel_o_ or cm_ad or int_reset or sel_o_del_1 )
   if (int_reset )
          begin
            rd_wr_ <= #‘latdel ‘FALSE;
            sar2phyif_phy_a <= #‘latdel ‘NULL;
          end
   else if (sel_o_ && sel_o_del_1)
          begin
            rd_wr_ <= #‘latdel ~cm_oe_i;
            sar2phyif_phy_a <= #‘latdel cm_ad[7:0];
          end

// Toggle the output enable for the FPGA SAR–interface data when
// a read is executed to the SAR2PHY and cm_ad[6]=1.
always @ (posedge cm_oe_i_ or posedge int_reset ) if (int_reset )
          begin
            cmdata_en <= #‘latdel ‘FALSE;
          end
   else if (cm_ad[17] && cm_ad[6])
          begin
            cmdata_en <= #‘latdel ~cmdata_en;
          end

// HIGH PRIORITY SEGMENTATION PROCESSING
always @ (posedge atm_clk or posedge int_reset )
   if (int_reset )
          begin
            hpsreq <= #‘latdel ‘FALSE;
          end
   else if (!hpsack && (cm_ad[5] == 1’d1))
          begin
            hpsreq <= #‘latdel ‘TRUE;
          end
   else if (hpsack)
          begin
            hpsreq <= #‘latdel ‘FALSE;
          end
```

```
// SOFTWARE RESET IS HIGH FOR 3 ATM_CLK
always @ (posedge atm_clk or posedge int_reset)
   if (int_reset)
          begin
            sw_reset_del1 <= #‘latdel ‘FALSE;
            sw_reset_del2 <= #‘latdel
            sw_reset_del3 <= #‘latdel ‘FALSE;
          end
   else if (cm_ad[17] && cm_ad[7] && !cm_oe_i_)
          begin
            sw_reset_del1 <= #‘latdel sw_reset_del2;
            sw_reset_del2 <= #‘latdel sw_reset_del3;
            sw_reset_del3 <= #‘latdel ‘TRUE;
          end
   else
          begin
            sw_reset_del1 <= #‘latdel sw_reset_del2;
            sw_reset_del2 <= #‘latdel sw_reset_del3;
            sw_reset_del3 <= #‘latdel ‘FALSE;
          end

// SOFTWARE RESET
always @ (sw_reset_del1 or sw_reset_del2 or sw_reset_del3 )
          begin
   if (sw_reset_del1 || sw_reset_del2 || sw_reset_del3 )
          begin
            sw_reset <= 1’b1;
          end
   else
          begin
            sw_reset <= 1’b0;
          end
   end

endmodule
```

```
module SAR2PHY
       (CM_WE_I_, CM_OE_I_, RD_WR_, SEL_O_, READY,ATM_CLK, RESET,
       CM_DATA0, CM_DATA1, CM_DATA2, CM_DATA3, CM_DATA4, CM_DATA5, CM_DATA6, CM_DATA7,
       CM_DATA8, CM_DATA9, CM_DATA10, CM_DATA11, CM_DATA12, CM_DATA13, CM_DATA14,
       CM_DATA15, CM_DATA16, CM_DATA17, CM_DATA18, CM_DATA19, CM_DATA20, CM_DATA21,
       CM_DATA22, CM_DATA23, CM_DATA24, CM_DATA25, CM_DATA26, CM_DATA27, CM_DATA28,
       CM_DATA29, CM_DATA30, CM_DATA31, SAR2PHYIF_PHY_D0, SAR2PHYIF_PHY_D1,
       SAR2PHYIF_PHY_D2, SAR2PHYIF_PHY_D3, SAR2PHYIF_PHY_D4, SAR2PHYIF_PHY_D5,
       SAR2PHYIF_PHY_D6, SAR2PHYIF_PHY_D7, CM_AD0, CM_AD1, CM_AD2, CM_AD3, CM_AD4,
       CM_AD5, CM_AD6, CM_AD7, CM_AD8, CM_AD9, CM_AD10, CM_AD11, CM_AD12, CM_AD13,
       CM_AD14, CM_AD15, CM_AD16, CM_AD17, SAR2PHYIF_PHY_A0, SAR2PHYIF_PHY_A1,
       SAR2PHYIF_PHY_A2, SAR2PHYIF_PHY_A3, SAR2PHYIF_PHY_A4, SAR2PHYIF_PHY_A5,
       SAR2PHYIF_PHY_A6, SAR2PHYIF_PHY_A7, HPSREQ, HPSACK);

input     CM_WE_I_;
input     CM_OE_I_;

output    RD_WR_;
output    SEL_O_;
input     READY;

input     ATM_CLK;
input     RESET;

inout     CM_DATA0, CM_DATA1, CM_DATA2, CM_DATA3, CM_DATA4, CM_DATA5,
          CM_DATA6, CM_DATA7, CM_DATA8, CM_DATA9, CM_DATA10, CM_DATA11, CM_DATA12,
          CM_DATA13, CM_DATA14, CM_DATA15, CM_DATA16, CM_DATA17, CM_DATA18, CM_DATA19,
          CM_DATA20, CM_DATA21, CM_DATA22, CM_DATA23, CM_DATA24, CM_DATA25, CM_DATA26,
          CM_DATA27, CM_DATA28, CM_DATA29, CM_DATA30, CM_DATA31;

inout     SAR2PHYIF_PHY_D0, SAR2PHYIF_PHY_D1, SAR2PHYIF_PHY_D2,
          SAR2PHYIF_PHY_D3, SAR2PHYIF_PHY_D4, SAR2PHYIF_PHY_D5,
          SAR2PHYIF_PHY_D6, SAR2PHYIF_PHY_D7;

input     CM_AD0, CM_AD1, CM_AD2, CM_AD3, CM_AD4, CM_AD5, CM_AD6,
          CM_AD7, CM_AD8, CM_AD9, CM_AD10, CM_AD11, CM_AD12,
          CM_AD13, CM_AD14, CM_AD15, CM_AD16, CM_AD17;

output    SAR2PHYIF_PHY_A0, SAR2PHYIF_PHY_A1, SAR2PHYIF_PHY_A2,
          SAR2PHYIF_PHY_A3,SAR2PHYIF_PHY_A4, SAR2PHYIF_PHY_A5,
          SAR2PHYIF_PHY_A6, SAR2PHYIF_PHY_A7;

output    HPSREQ;
input     HPSACK;

// wire declarations
wire      CM_WE_I_, CM_OE_I_, RD_WR_, SEL_O_, READY,ATM_CLK, RESET,
          CM_DATA0, CM_DATA1, CM_DATA2, CM_DATA3, CM_DATA4, CM_DATA5, CM_DATA6,
          CM_DATA7, CM_DATA8, CM_DATA9, CM_DATA10, CM_DATA11, CM_DATA12, CM_DATA13,
          CM_DATA14, CM_DATA15, CM_DATA16, CM_DATA17, CM_DATA18, CM_DATA19, CM_DATA20,
          CM_DATA21, CM_DATA22, CM_DATA23, CM_DATA24, CM_DATA25, CM_DATA26,
          CM_DATA27,CM_DATA28, CM_DATA29, CM_DATA30, CM_DATA31, SAR2PHYIF_PHY_D0,
          SAR2PHYIF_PHY_D1, SAR2PHYIF_PHY_D2, SAR2PHYIF_PHY_D3, SAR2PHYIF_PHY_D4,
          SAR2PHYIF_PHY_D5, SAR2PHYIF_PHY_D6, SAR2PHYIF_PHY_D7, CM_AD0, CM_AD1,
          CM_AD2, CM_AD3, CM_AD4, CM_AD5, CM_AD6, CM_AD7, CM_AD8, CM_AD9, CM_AD10,
          CM_AD11, CM_AD12, CM_AD13, CM_AD14, CM_AD15, CM_AD16, CM_AD17, SAR2PHY-
          IF_PHY_A0, SAR2PHYIF_PHY_A1, SAR2PHYIF_PHY_A2, SAR2PHYIF_PHY_A3,SAR2PHY-
          IF_PHY_A4, SAR2PHYIF_PHY_A5, SAR2PHYIF_PHY_A6, SAR2PHYIF_PHY_A7, HPSREQ,
          HPSACK;
```

```
wire          CM_WE_I_pad;
wire          CM_OE_I_pad;

wire          RD_WR_pad;
wire          SEL_O_pad;
wire          READY_pad;

wire          ATM_CLK_pad;
wire          RESET_pad;

wire [31:0]   cm_data_out;
wire [31:0]   cm_data_in;
wire          cmdata_en;

wire [7:0]    sar2phyif_phy_d_out;
wire [7:0]    sar2phyif_phy_d_in;
wire          sar2phyif_en;

wire [17:0]   CM_AD_pad;

wire [7:0]    SAR2PHYIF_PHY_A_pad;

wire          HPSREQ_pad;
wire          HPSACK_pad;

// Instantiate the Core Design

sar2phy FPGA (
              .cm_we_i_(CM_WE_I_pad),
              .cm_oe_i_(CM_OE_I_pad),
              .rd_wr_(RD_WR_pad),
              .sel_o_(SEL_O_pad),
              .ready(READY_pad),
              .atm_clk(ATM_CLK_pad),
              .reset(RESET_pad),

              .cm_data_in(cm_data_in),

              .cm_data_out(cm_data_out),
              .cmdata_en(cmdata_en),

              .sar2phyif_phy_d_in(sar2phyif_phy_d_in),
              .sar2phyif_phy_d_out(sar2phyif_phy_d_out),
              .sar2phyif_en(sar2phyif_en),

              .cm_ad(CM_AD_pad),

              .sar2phyif_phy_a(SAR2PHYIF_PHY_A_pad),

              .hpsreq(HPSREQ_pad),
              .hpsack(HPSACK_pad)
               );
```

```
/*Instantiate Buffers*/

iobuffer IOS (
             .CM_WE_I_(CM_WE_I_),
             .CM_OE_I_(CM_OE_I_),
             .RD_WR_(RD_WR_),
             .SEL_O_(SEL_O_),
             .READY(READY),
             .ATM_CLK(ATM_CLK),
             .RESET(RESET),

             .CM_DATA({CM_DATA0, CM_DATA1, CM_DATA2, CM_DATA3,
              CM_DATA4, CM_DATA5, CM_DATA6, CM_DATA7, CM_DATA8,
             CM_DATA9, CM_DATA10, CM_DATA11, CM_DATA12, CM_DATA13,
             CM_DATA14, CM_DATA15, CM_DATA16, CM_DATA17, CM_DATA18,
             CM_DATA19, CM_DATA20, CM_DATA21, CM_DATA22, CM_DATA23,
             CM_DATA24, CM_DATA25, CM_DATA26, CM_DATA27, CM_DATA28,
             CM_DATA29, CM_DATA30, CM_DATA31}),

             .SAR2PHYIF_PHY_D({SAR2PHYIF_PHY_D0, SAR2PHYIF_PHY_D1,
             SAR2PHYIF_PHY_D2, SAR2PHYIF_PHY_D3, SAR2PHYIF_PHY_D4,
             SAR2PHYIF_PHY_D5, SAR2PHYIF_PHY_D6, SAR2PHYIF_PHY_D7}),

             .CM_AD({ CM_AD0, CM_AD1, CM_AD2, CM_AD3, CM_AD4,
             CM_AD5, CM_AD6, CM_AD7, CM_AD8, CM_AD9,
             CM_AD10, CM_AD11, CM_AD12, CM_AD13, CM_AD14,
             CM_AD15, CM_AD16, CM_AD17 }),

             .SAR2PHYIF_PHY_A({SAR2PHYIF_PHY_A0, SAR2PHYIF_PHY_A1,
             SAR2PHYIF_PHY_A2, SAR2PHYIF_PHY_A3,SAR2PHYIF_PHY_A4,
             SAR2PHYIF_PHY_A5, SAR2PHYIF_PHY_A6, SAR2PHYIF_PHY_A7}),

             .HPSREQ(HPSREQ),
             .HPSACK(HPSACK),

             .CM_WE_I_pad(CM_WE_I_pad),
             .CM_OE_I_pad(CM_OE_I_pad),
             .RD_WR_pad(RD_WR_pad),
             .SEL_O_pad(SEL_O_pad),
             .READY_pad(READY_pad),
             .ATM_CLK_pad(ATM_CLK_pad),
             .RESET_pad(RESET_pad),
             .cm_data_out(cm_data_out),
             .cm_data_in(cm_data_in),
             .cmdata_en(cmdata_en),
             .sar2phyif_phy_d_out(sar2phyif_phy_d_out),
             .sar2phyif_phy_d_in(sar2phyif_phy_d_in),
             .sar2phyif_en(sar2phyif_en),
             .CM_AD_pad(CM_AD_pad),
             .SAR2PHYIF_PHY_A_pad(SAR2PHYIF_PHY_A_pad),
             .HPSREQ_pad(HPSREQ_pad),
             .HPSACK_pad(HPSACK_pad)
             );

endmodule
```

```
module iobuffer
       (CM_WE_I_, CM_OE_I_, RD_WR_, SEL_O_, READY, ATM_CLK, RESET,
       CM_DATA, SAR2PHYIF_PHY_D, CM_AD, SAR2PHYIF_PHY_A, HPSREQ, HPSACK,
       CM_WE_I_pad,  CM_OE_I_pad, RD_WR_pad, SEL_O_pad,  READY_pad,
       ATM_CLK_pad, RESET_pad, cm_data_out, cm_data_in, cmdata_en,
       sar2phyif_phy_d_out, sar2phyif_phy_d_in, sar2phyif_en,
       CM_AD_pad, SAR2PHYIF_PHY_A_pad, HPSREQ_pad, HPSACK_pad
       );

input         CM_WE_I_;
input         CM_OE_I_;

output        RD_WR_;
output        SEL_O_;
input         READY;

input         ATM_CLK;
input         RESET;

inout [0:31]  CM_DATA;

inout [0:7]   SAR2PHYIF_PHY_D;

input [0:17]  CM_AD;

output [0:7]  SAR2PHYIF_PHY_A;

output        HPSREQ;
input         HPSACK;

output        CM_WE_I_pad;
output        CM_OE_I_pad;

input         RD_WR_pad;
input         SEL_O_pad;
output        READY_pad;

output        ATM_CLK_pad;
output        RESET_pad;

input [31:0]  cm_data_out;
output [31:0] cm_data_in;
input         cmdata_en;

input [7:0]   sar2phyif_phy_d_out;
output [7:0]  sar2phyif_phy_d_in;
input         sar2phyif_en;

output [17:0] CM_AD_pad;

input [7:0]   SAR2PHYIF_PHY_A_pad;

input         HPSREQ_pad;
output        HPSACK_pad;

// wire declarations

wire          CM_WE_I_;
wire          CM_OE_I_;

wire          RD_WR_;
wire          SEL_O_;
wire          READY;

wire          ATM_CLK;
wire          RESET;

wire [0:31]   CM_DATA;

wire [0:7]    SAR2PHYIF_PHY_D;

wire [0:17]   CM_AD;

wire [0:7]    SAR2PHYIF_PHY_A;

wire          HPSREQ;
wire          HPSACK;

wire          CM_WE_I_pad;
wire          CM_OE_I_pad;
```

```
wire          RD_WR_pad;
wire          SEL_O_pad;
wire          READY_pad;

wire          ATM_CLK_pad;
wire          RESET_pad;

wire [31:0]   cm_data_out;
wire [31:0]   cm_data_in;
wire          cmdata_en;

wire [7:0]    sar2phyif_phy_d_out;
wire [7:0]    sar2phyif_phy_d_in;
wire          sar2phyif_en;

wire [17:0]   CM_AD_pad;

wire [7:0]    SAR2PHYIF_PHY_A_pad;

wire          HPSREQ_pad;
wire          HPSACK_pad;

/*Instantiate Input Buffers*/
INBUF P_01 ( .PAD( CM_WE_I_ ), .Y( CM_WE_I_pad));
INBUF P_02 ( .PAD( CM_OE_I_ ), .Y( CM_OE_I_pad));
INBUF P_03 ( .PAD( READY ), .Y( READY_pad));
INBUF P_04 ( .PAD( ATM_CLK ), .Y( ATM_CLK_pad));
INBUF P_05 ( .PAD( RESET ), .Y( RESET_pad));
INBUF P_06 ( .PAD( HPSACK ), .Y( HPSACK_pad));
INBUF P_07 ( .PAD( CM_AD[0] ), .Y( CM_AD_pad[0]));
INBUF P_08 ( .PAD( CM_AD[1] ), .Y( CM_AD_pad[1]));
INBUF P_09 ( .PAD( CM_AD[2] ), .Y( CM_AD_pad[2]));
INBUF P_10 ( .PAD( CM_AD[3] ), .Y( CM_AD_pad[3]));
INBUF P_11 ( .PAD( CM_AD[4] ), .Y( CM_AD_pad[4]));
INBUF P_12 ( .PAD( CM_AD[5] ), .Y( CM_AD_pad[5]));
INBUF P_13 ( .PAD( CM_AD[6] ), .Y( CM_AD_pad[6]));
INBUF P_14 ( .PAD( CM_AD[7] ), .Y( CM_AD_pad[7]));
INBUF P_15 ( .PAD( CM_AD[8] ), .Y( CM_AD_pad[8]));
INBUF P_16 ( .PAD( CM_AD[9] ), .Y( CM_AD_pad[9]));
INBUF P_17 ( .PAD( CM_AD[10] ), .Y( CM_AD_pad[10]));
INBUF P_18 ( .PAD( CM_AD[11] ), .Y( CM_AD_pad[11]));
INBUF P_19 ( .PAD( CM_AD[12] ), .Y( CM_AD_pad[12]));
INBUF P_20 ( .PAD( CM_AD[13] ), .Y( CM_AD_pad[13]));
INBUF P_21 ( .PAD( CM_AD[14] ), .Y( CM_AD_pad[14]));
INBUF P_22 ( .PAD( CM_AD[15] ), .Y( CM_AD_pad[15]));
INBUF P_23 ( .PAD( CM_AD[16] ), .Y( CM_AD_pad[16]));
INBUF P_24 ( .PAD( CM_AD[17] ), .Y( CM_AD_pad[17]));

/*Instantiate Output Buffers*/
OUTBUF P_25 ( .D( RD_WR_pad ), .PAD( RD_WR_));
OUTBUF P_26 ( .D( SEL_O_pad ), .PAD( SEL_O_));
OUTBUF P_27 ( .D( HPSREQ_pad ), .PAD( HPSREQ));
OUTBUF P_28 ( .D( SAR2PHYIF_PHY_A_pad[0] ), .PAD( SAR2PHYIF_PHY_A[0]));
OUTBUF P_29 ( .D( SAR2PHYIF_PHY_A_pad[1] ), .PAD( SAR2PHYIF_PHY_A[1]));
OUTBUF P_30 ( .D( SAR2PHYIF_PHY_A_pad[2] ), .PAD( SAR2PHYIF_PHY_A[2]));
```

```
OUTBUF P_31 ( .D( SAR2PHYIF_PHY_A_pad[3] ), .PAD( SAR2PHYIF_PHY_A[3]));
OUTBUF P_32 ( .D( SAR2PHYIF_PHY_A_pad[4] ), .PAD( SAR2PHYIF_PHY_A[4]));
OUTBUF P_33 ( .D( SAR2PHYIF_PHY_A_pad[5] ), .PAD( SAR2PHYIF_PHY_A[5]));
OUTBUF P_34 ( .D( SAR2PHYIF_PHY_A_pad[6] ), .PAD( SAR2PHYIF_PHY_A[6]));
OUTBUF P_35 ( .D( SAR2PHYIF_PHY_A_pad[7] ), .PAD( SAR2PHYIF_PHY_A[7]));
BBHS P_50io( .PAD( CM_DATA[0]), .Y( cm_data_in[0]), .D( cm_data_out[0]),
.E( cmdata_en ));
BBHS P_51io( .PAD( CM_DATA[1]), .Y( cm_data_in[1]), .D( cm_data_out[1]),
.E( cmdata_en ));
BBHS P_52io( .PAD( CM_DATA[2]), .Y( cm_data_in[2]), .D( cm_data_out[2]),
.E( cmdata_en ));
BBHS P_53io( .PAD( CM_DATA[3]), .Y( cm_data_in[3]), .D( cm_data_out[3]) ,
.E( cmdata_en ));
BBHS P_54io( .PAD( CM_DATA[4]), .Y( cm_data_in[4]), .D( cm_data_out[4]),
.E( cmdata_en ));
BBHS P_55io( .PAD( CM_DATA[5]), .Y( cm_data_in[5]), .D( cm_data_out[5]),
.E( cmdata_en ));
BBHS P_56io( .PAD( CM_DATA[6]), .Y( cm_data_in[6]), .D( cm_data_out[6]),
.E( cmdata_en ));
BBHS P_57io( .PAD( CM_DATA[7]), .Y( cm_data_in[7]), .D( cm_data_out[7]),
.E( cmdata_en ));
BBHS P_58io( .PAD( CM_DATA[8]), .Y( cm_data_in[8]), .D( cm_data_out[8]),
.E( cmdata_en ));
BBHS P_59io( .PAD( CM_DATA[9]), .Y( cm_data_in[9]), .D( cm_data_out[9]),
.E( cmdata_en ));
BBHS P_60io( .PAD( CM_DATA[10]), .Y( cm_data_in[10]), .D( cm_data_out[10]),
.E( cmdata_en ));
BBHS P_61io( .PAD( CM_DATA[11]), .Y( cm_data_in[11]), .D( cm_data_out[11]),
.E( cmdata_en ));
BBHS P_62io( .PAD( CM_DATA[12]), .Y( cm_data_in[12]), .D( cm_data_out[12]),
.E( cmdata_en ));
BBHS P_63io( .PAD( CM_DATA[13]), .Y( cm_data_in[13]), .D( cm_data_out[13]),
.E( cmdata_en ));
BBHS P_64io( .PAD( CM_DATA[14]), .Y( cm_data_in[14]), .D( cm_data_out[14]),
.E( cmdata_en ));
BBHS P_65io( .PAD( CM_DATA[15] ), .Y( cm_data_in[15]),.D( cm_data_out[15]),
.E( cmdata_en ));
BBHS P_66io( .PAD( CM_DATA[16]), .Y( cm_data_in[16]), .D( cm_data_out[16]),
.E( cmdata_en ));
BBHS P_67io( .PAD( CM_DATA[17]), .Y( cm_data_in[17]),.D( cm_data_out[17]),
.E( cmdata_en ));
BBHS P_68io( .PAD( CM_DATA[18]), .Y( cm_data_in[18]),.D( cm_data_out[18]),
.E( cmdata_en ));
BBHS P_69io( .PAD( CM_DATA[19]), .Y( cm_data_in[19]), .D( cm_data_out[19]),
.E( cmdata_en ));
BBHS P_70io( .PAD( CM_DATA[20]), .Y( cm_data_in[20]), .D( cm_data_out[20]),
.E( cmdata_en ));
BBHS P_71io( .PAD( CM_DATA[21]), .Y( cm_data_in[21]), .D( cm_data_out[21]),
.E( cmdata_en ));
BBHS P_72io( .PAD( CM_DATA[22]), .Y( cm_data_in[22]), .D( cm_data_out[22]),
.E( cmdata_en ));
BBHS P_73io( .PAD( CM_DATA[23]), .Y( cm_data_in[23]), .D( cm_data_out[23]),
.E( cmdata_en ));
BBHS P_74io( .PAD( CM_DATA[24]), .Y( cm_data_in[24]), .D( cm_data_out[24]),
.E( cmdata_en ));
```

```
BBHS P_75io( .PAD( CM_DATA[25]), .Y( cm_data_in[25]), .D( cm_data_out[25]),
.E( cmdata_en ));

BBHS P_76io( .PAD( CM_DATA[26]), .Y( cm_data_in[26]), .D( cm_data_out[26]),
.E( cmdata_en ));

BBHS P_77io( .PAD( CM_DATA[27]), .Y( cm_data_in[27]), .D( cm_data_out[27]),
.E( cmdata_en ));

BBHS P_78io( .PAD( CM_DATA[28]), .Y( cm_data_in[28]), .D( cm_data_out[28]),
.E( cmdata_en ));

BBHS P_79io( .PAD( CM_DATA[29]), .Y( cm_data_in[29]), .D( cm_data_out[29]),
.E( cmdata_en ));

BBHS P_80io( .PAD( CM_DATA[30]), .Y( cm_data_in[30]), .D( cm_data_out[30]),
.E( cmdata_en ));

BBHS P_81io( .PAD( CM_DATA[31]), .Y( cm_data_in[31]), .D( cm_data_out[31]),
.E( cmdata_en ));

BBHS P_82io( .PAD( SAR2PHYIF_PHY_D[0]), .Y( sar2phyif_phy_d_in[0]),
.D( sar2phyif_phy_d_out[0]), .E(sar2phyif_en));

BBHS P_83io( .PAD( SAR2PHYIF_PHY_D[1]), .Y( sar2phyif_phy_d_in[1]),
.D( sar2phyif_phy_d_out[1]), .E(sar2phyif_en));

BBHS P_84io( .PAD( SAR2PHYIF_PHY_D[2]), .Y( sar2phyif_phy_d_in[2]),
.D( sar2phyif_phy_d_out[2]), .E(sar2phyif_en));

BBHS P_85io( .PAD( SAR2PHYIF_PHY_D[3]), .Y( sar2phyif_phy_d_in[3]),
.D( sar2phyif_phy_d_out[3]), .E(sar2phyif_en));

BBHS P_86io( .PAD( SAR2PHYIF_PHY_D[4]), .Y( sar2phyif_phy_d_in[4]),
.D( sar2phyif_phy_d_out[4]), .E(sar2phyif_en));

BBHS P_87io( .PAD( SAR2PHYIF_PHY_D[5]), .Y( sar2phyif_phy_d_in[5]),
.D( sar2phyif_phy_d_out[5]), .E(sar2phyif_en));

BBHS P_88io( .PAD( SAR2PHYIF_PHY_D[6]), .Y( sar2phyif_phy_d_in[6]),
.D( sar2phyif_phy_d_out[6]), .E(sar2phyif_en));

BBHS P_89io( .PAD( SAR2PHYIF_PHY_D[7]), .Y( sar2phyif_phy_d_in[7]),
.D( sar2phyif_phy_d_out[7]), .E(sar2phyif_en));

endmodule
```