Scalable Centralized Power Management with Field Upgrade Support

June 2012 Application Note AN6089

Introduction

The Platform Manager™ device family is a single-chip, fully-integrated solution for supervisory and control designs encountered when implementing on-board power conversion and distribution systems. It provides several types of on-chip resources which can be used to meet the requirements of an application. A Platform Manager device is a combination of FPGA LUTs, CPLD logic cells and analog features for voltage monitoring, power supply trimming, reset generation, I/O control and more. See Table 1 for the Platform Manager devices applicable to this application note.

Table 1. Applicable Platform Manager Devices Summary

Device	VMONs	Digital Inputs	HVOUTs	Open Drain Outputs	Trim DAC Outputs	Digital I/O	CPLD Macrocells	FPGA LUTs	Package
LPTM10-1247	12	4	4	12	6	31	48	640	128-pin TQFP
LPTM10-12107	12	4	4	12	8	91	48	640	208-ball ftBGA

Lattice provides Windows-based, PAC-Designer® software, which can be used to generate JEDEC or SVF programming files for the Platform Manager device. LogiBuilder is a design environment within PAC-Designer that can be used to design and simulate the CPLD with a custom state machine based on the specific requirements of the application. To design the FPGA portion of a Platform Manager device, either LogiBuilder or Lattice Diamond® software can be used. PAC-Designer creates a single merged JEDEC file for Platform Manager. The file is a combination of the JEDEC files for the FPGA and CPLD portions. The JEDEC or SVF file can be downloaded into the Platform Manager device using Lattice ispVM™ System software.

Application of Scalable Centralized Power Management

In some applications it is necessary to monitor and sequence many voltage supplies for different system devices. An example of this is a server board with multiple CPUs. Lattice provides different Power Manager II products that can monitor up to 12 voltages and sequence up to 20 devices. Systems that are required to monitor more than 12 supplies often use multiple Power Manager II devices on a board. However, distributed supervisory control has limited synchronization and presents design and maintenance issues.

An alternative approach to sequencing and monitoring more than 12 supplies is to use the Lattice Platform Manager device along with Lattice Power Manager II device(s). The advantage of this approach is the Platform Manager can monitor up to 12 voltage supplies itself while also acting as the centralized sequence controller for all the supplies in the system. The Platform Manager has more logic resources so that it can accommodate the increased number of supplies which need to be monitored and controlled. The Platform Manager can also provide logic for other ancillary functions such as reset generation, alarms, or other status signals.

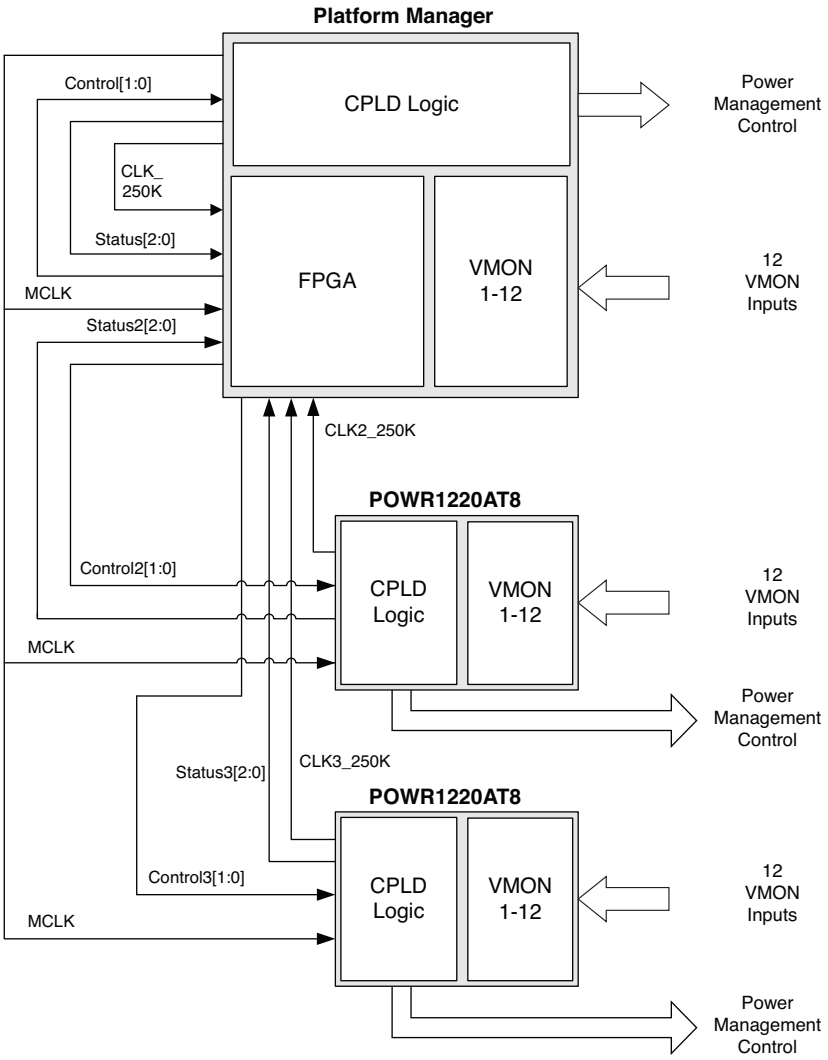
The power supplies which are not connected to the Platform Manager can be monitored and switched by a separate Power Manager II device. Using a full duplex serial link, the Power Manager II sends the voltage status back to the Platform Manager for use in the sequence control. This provides the Platform Manager with all the system operation parameters so it can determine which supplies to sequence. The Platform Manager then sends control signals to turn On or Off the power supplies using the serial link.

Another serial link of the same design is used to send the voltage status from the CPLD of the Platform Manager to the FPGA since there is no internal connections between these two sections of the Platform Manager device.

This application note demonstrates how a system of up to 36 power supplies can be successfully monitored and sequenced using a Platform Manager and two Power Manager II devices.

A block diagram of this concept and the connections between the Platform Manager and Power Manager devices is shown in Figure 1.

Figure 1. Scalable Centralized Power Management Block Diagram



CPLD Design Description

The MCLK from the CPLD in the Platform Manager is used as the master clock for the entire design in order to synchronize the CPLD sections and the FPGA. In the two external Power Manager devices, the MCLK pin is set as an input using the PAC-Designer software.

The CLK_250K signal is derived from the MCLK signal in each CPLD section, thus ensuring that these are synchronized also. The CLK_250K signals are used to drive the serial link in each CPLD section so this is sent to the FPGA by each CPLD section to synchronize the respective serial links.

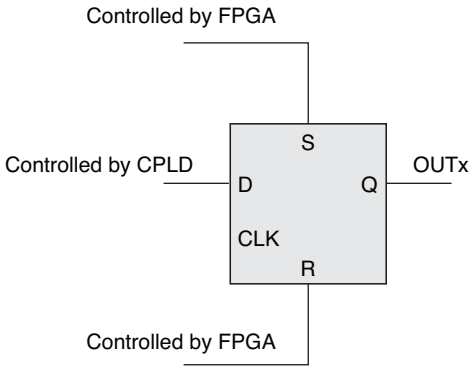
The logic required to implement the serial link is contained in the Supervisory equations section of LogiBuilder. The serial link sends the VMON status bits and internal AGOOD signal to the FPGA and also de-codes the control signals received from the FPGA to turn on or off the power supplies. This is described in more detail below:

1. Transmitting the VMON status bits to the FPGA section starts with the rising edge of the Data Start line, called Start, to indicate the beginning of the serial packet. The Start signal is one of the three serial data lines used to send data from the CPLD section to the FPGA section in the Platform Manager. The CPLD

continues sending information on the three data lines for the next 13 clock cycles after which it repeats the process. The FPGA section de-serializes this serial stream to parallel and presents it to the central sequencer logic within the FPGA.

2. The output control signals from the FPGA are received through two CPLD digital input signals. One input receives the output control bits (14 in each packet) and the second input receives a strobe pulse for each data bit. The supervisory logic uses the first two bits to detect if the FPGA is present and the remaining 12 bits are used to control outputs. The FPGA present status signal is used in the CPLD section to enable a backup power sequence if the FPGA is not present at power-up. The basic idea for controlling each output register is represented in Figure 2.

Figure 2. Control of the Outputs

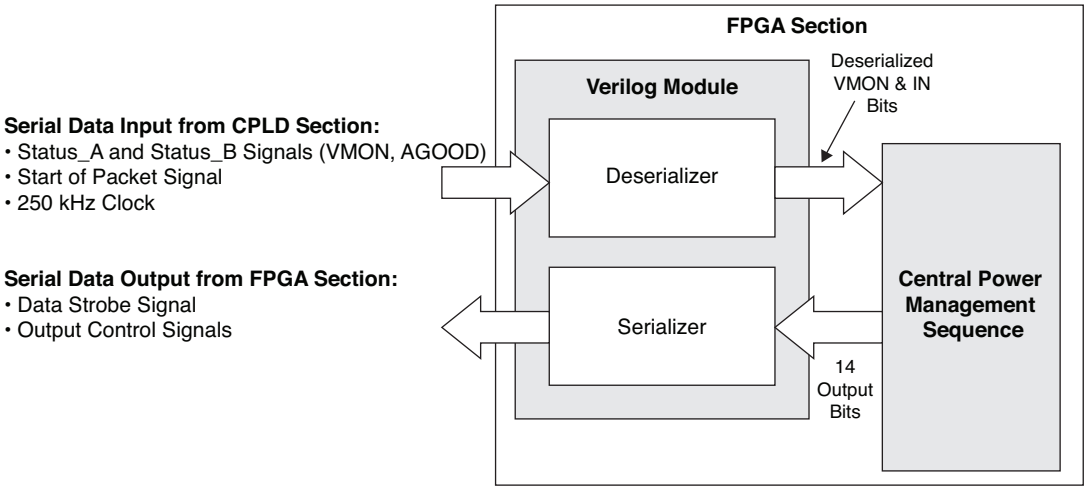


The FPGA will turn ON the output using the preset input and turn OFF the output using the reset input of a D type flip-flop. The CPLD will control the D input to the flip-flop. The reset and preset are asynchronous and will override the D input from the CPLD thus giving the FPGA control. The backup sequence in the CPLD will take over only at startup if the FPGA fails to establish communication with the CPLD which is indicated by the FPGA present status.

FPGA Design Description

The FPGA design includes the serial-deserializer block for communication to the CPLD and a central power management sequence. The FPGA receives the VMON status bits from the deserializer as inputs to the central power management sequence. The central power management sequence provides output control bits to the serializer which are sent to the CPLD for control of the outputs. A block diagram of the logic is shown in Figure 3. The Verilog code for the serializer/deserializer logic is shown in "Appendix B. Listing of FPGA Code" on page 10. The central power management sequence is application specific and not discussed in this application note.

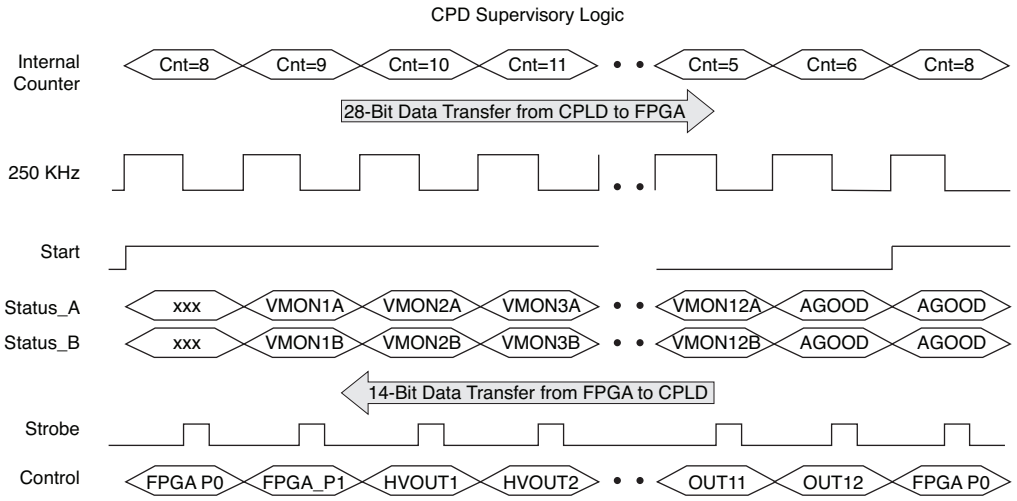
Figure 3. FPGA Section of the Power Management Design



Serial Protocol

The supervisory section of the CPLD implements a 4-bit counter that controls the 28 bits sent in each data transmission frame. The counter begins at 8 then counts up to 15, rolls over, and continues counting from 0 to 6. When the counter reaches 6 it resets to 8. Start is set to the counter's most significant bit so that when the count is between 8 and 15 the Start is high. The rising edge of the Start signal is used to indicate the beginning of a new data frame. The FPGA Verilog module also implements a counter that counts from 0 to 13 and the rising edge of the Start signal resets this FPGA counter to 0 and synchronizes it to the CPLD.

Figure 4. Serial Protocol Implementation



Initialization of the protocol occurs when the counter is at 8 with CPLD supervisory logic outputting high on the Start signal. In the FPGA portion, serializer/deserializer logic will send the output data on the Control signal along with a strobe signal. The Control data packet has a logic '1' for the first bit and logic '0' for the second bit and this bit combination is used by the CPLD to determine whether the FPGA is present. The remaining bits in the data stream are the output control bits. FPGA logic uses an 8 MHz clock along with a 250 kHz clock to generate 0.5 us pulses at the falling edge of the 250 kHz clock signal. This generated pulse is sent out as the strobe bit.

During the next 13 counts, the CPLD section outputs values corresponding to VMON1A through VMON12B along with the CPLD internal AGOOD status bit though the Status_A and Status_B lines. At the same time, the FPGA portion outputs the strobe and control of the HVOUT1 through OUT12 signals as determined by the power management sequence. Waveforms describing this protocol are shown in Figure 4.

Supervisory logic equations are used to perform de-serialization functions in the CPLD section. These equations capture the status of the Control signal into the corresponding nodes when the counter value is reached. For example, when the counter value is 8, the Control value is used to set/reset the FPGA_Prsnt node. When the counter value is 10, the Control value is used to set/reset the HVOUT1 macrocell. The Control value is shown as DataIN in the code listing which is provided in "Appendix A. Listing of CPLD Supervisory Equations" on page 8. The logic implemented in the FPGA portion captures the CPLD outputs on the Status_A and Status_B lines and assigns them to nodes according to the count value. In the FPGA code listing the Control signal is named Data. Example Verilog code is shown in "Appendix B. Listing of FPGA Code" on page 10.

Working with the Design Templates

This application note is provided with a set of Design Template files which are available at the Lattice web site. The Design Templates are provided as building blocks for user projects. Project elements like voltage thresholds, sequence steps and input and output assignments are always specific to the application. The design templates leave this information set to default values and must be configured by the user.

The Platform Manager template uses the design flow of PAC-Designer and Diamond. This means the FPGA I/O and logic are all handled in Lattice Diamond software. Therefore the Design Template files for the Platform Manager are separated into a PAC-Designer project and a Diamond Project. The Diamond project includes Verilog files and project files used to build the FPGA JEDEC file with the Diamond software. The PAC-Project includes the CPLD Control receiver and VMON Status transmitter, analog configuration data, and a pointer to the Diamond generated FPGA JEDEC file. This information together is used to build the Platform Manager JEDEC file.

An additional PAC-Designer project is included for the external POWR1220AT8 devices used with the additional supplies. Each project is provided in separate directories for ease of use.

The Diamond Files directory includes two Verilog source code files which implement the design:

1. DT6089_FPGA_Top.V
2. DT6089_FPGA_250kSERDES.v.

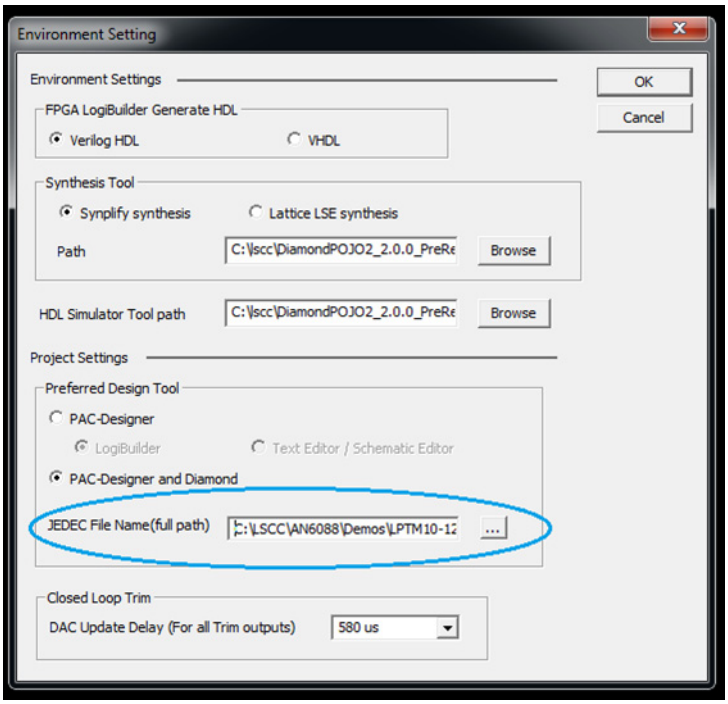
The DT6089_FPGA_250kSERDES.v contains the serial link sub-module. This sub-module should not be modified. DT6089_FPGA_Top.V is the top module. It instantiates the 250k serial sub-module and the minimum number of I/O ports in order to work with the serial link. It also includes a comment showing where the user can add their design specific sequence or other code. These two files are included in the Diamond project as a starting point. For additional details about working with the Lattice Diamond software see the Lattice Diamond Tutorial found on the Lattice Diamond start page. The Diamond project needs to be completed and compiled to create a JEDEC file before working with PAC-Designer.

PAC-Designer projects are provided for both the Platform Manager and the POWR1220AT8. In the .PAC files, the CPLD LogiBuilder code contains a set of supervisory equations. The templates also assign the inputs and outputs used by the serial link. All the analog settings and additional I/O assignments will need to be made before the file is usable.

In the Platform Manager design flow of PAC-Designer and Diamond, a combined JEDEC file is built in PAC-Designer based on a path reference to the FPGA JEDEC file which is generated by Diamond. The full path is required for this file and it is left blank in the design template. You will need to create and build your Diamond JEDEC file before starting in PAC-Designer and then specify the full path to this file in PAC-Designer.

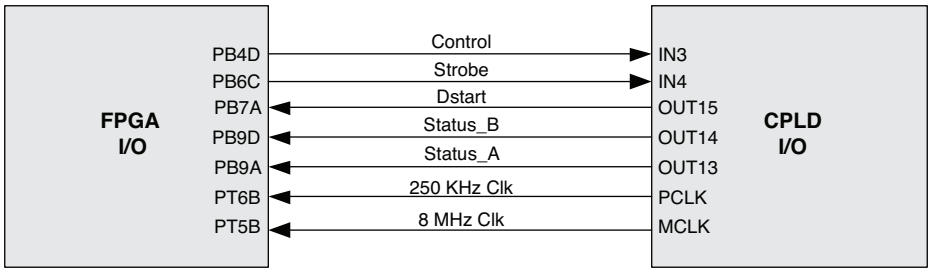
To supply the pointer to the Diamond generated FPGA JEDEC file, open PAC-Designer and from the Schematic window of PAC-Designer choose the **Options** menu, then the **FPGA Environment** selection. A dialog box will open as shown in Figure 5. The full path for the FPGA JEDEC file is entered in the location circled in dialog box.

Figure 5. PAC-Designer Environment Settings



The serial link used in the Platform Manager Design Template is comprised of the connections below. The PAC-Designer supervisory equations can be edited if the inputs or outputs need to be changed in the user's design. To change the outputs, edit only the output assignments in supervisory equations 8 through 10. To change the inputs edit the logic in supervisory equations 11 and 12.

Figure 6. Connections Used in the Platform Manager Design Template



In order to ensure reliable operation of the serial link it is highly recommended that the master clock (MCLK) and CLK_250K signals from the CPLD section and the CLK_250K signals from the Power Manager devices be connected to the Primary Clock pins (PCLKx_x) on the FPGA section of the Platform Manager device. It is also recommended that a series resistor be used for the CLK_250K signals from the Power Manager devices to the FPGA section to insure reliable operation of the Platform Manager device. These resistors should be located close to the FPGA and should be sized to match the trace impedance (typically 50 Ohms).

Implementation

Table 2. Performance and Resource Utilization[1]

Device	FPGA LUTs	FPGA Slices	FPGA I/Os	CPLD Macrocells[2]	CPLD Timers[2]	CPLD Product Terms[2]	CPLD I/Os[2]	VMONs
LPTM10-12107	163	100	19	29	1	119	19	12

1. These results were obtained using PAC-Designer 6.2 and Diamond 1.4 with the PAC-Designer and Diamond design tool setting in PAC-Designer. The PAC-Designer LogiBuilder Options were set to Binary Encoding and T-Type Flip-Flop synthesis.
2. Resources required in the CPLD section. The same resources are also required in the two external ispPAC®-POWR1220AT8 devices.

Summary

The combination of Platform Manager and Power Manager II devices offers the power supply system designer a powerful and flexible tool for controlling their system. This application note has demonstrated how a Platform Manager and a Power Manager II device can be combined to sequence and monitor up to 36 voltage rails in a system.

A method for transferring the VMON status bits from the Power Manager II and the CPLD section to the FPGA section using a serial communication link was described. This serial communication link was also used to transfer control bits from the FPGA to the Power Manager II and the CPLD. The serial communication link uses a minimum of logic resource in the FPGA so the remaining resources in the FPGA are available for implementation of a centralized power sequence as well as other ancillary functions such as reset generation, alarms, and other status signals.

References

- DS1036, Platform Manager Data Sheet
- DS1015, ispPAC-POWR1220AT8 Data Sheet
- TN1223, Using the Platform Manager Successfully

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-503-268-8001 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
May 2012	01.0	Initial release.
May 2012	01.1	Expanded design to control 36 power supplies. Updated figure 1. Added Working with the Design Templates section. Updated Appendix A.
June 2012	01.2	Updated Resource Utilization table. Updated the Working with the Design templates section and added clock pin recommendations paragraph.

Appendix A. Listing of CPLD Supervisory Equations

```
// Logic for Generating Counter
Eq 0: Cntr0.D = NOT Cntr0
Eq 1: Cntr1.D = ( NOT Cntr1 AND Cntr0 ) OR ( Cntr1 AND NOT Cntr0 )
Eq 2: Cntr2.D = ( NOT Cntr2 AND Cntr1 AND Cntr0 ) OR ( Cntr2 AND NOT ( Cntr1 AND Cntr0) )
Eq 3: Cntr3.D = ( NOT Cntr3 AND Cntr2 AND Cntr1 AND Cntr0 ) OR ( Cntr3 AND NOT ( Cntr2
AND Cntr1 AND Cntr0 ) )
Eq 4: Cntr0.ar = NOT Cntr3 AND Cntr2 AND Cntr1 AND NOT Cntr0
Eq 5: Cntr1.ar = NOT Cntr3 AND Cntr2 AND Cntr1 AND NOT Cntr0
Eq 6: Cntr2.ar = NOT Cntr3 AND Cntr2 AND Cntr1 AND NOT Cntr0
Eq 7: Cntr3.ap = NOT Cntr3 AND Cntr2 AND Cntr1 AND NOT Cntr0

//VMONA Status Output Logic
Eq 8: OUT13 = ( VMON1_A AND Cntr3 AND NOT Cntr2 AND NOT Cntr1 AND NOT Cntr0 ) OR (
VMON2_A AND Cntr3 AND NOT Cntr2 AND NOT Cntr1 AND Cntr0 ) OR ( VMON3_A AND Cntr3 AND NOT
Cntr2 AND Cntr1 AND NOT Cntr0 ) OR ( VMON4_A AND Cntr3 AND NOT Cntr2 AND Cntr1 AND Cntr0 )
OR ( VMON5_A AND Cntr3 AND Cntr2 AND NOT Cntr1 AND NOT Cntr0 ) OR ( VMON6_A AND Cntr3 AND
Cntr2 AND NOT Cntr1 AND Cntr0 ) OR ( VMON7_A AND Cntr3 AND Cntr2 AND Cntr1 AND NOT Cntr0 )
OR ( VMON8_A AND Cntr3 AND Cntr2 AND Cntr1 AND Cntr0 ) OR ( VMON9_A AND NOT Cntr3 AND NOT
Cntr2 AND NOT Cntr1 AND NOT Cntr0 ) OR ( VMON10_A AND NOT Cntr3 AND NOT Cntr2 AND NOT Cntr1
AND Cntr0 ) OR ( VMON11_A AND NOT Cntr3 AND NOT Cntr2 AND Cntr1 AND NOT Cntr0 ) OR (
VMON12_A AND NOT Cntr3 AND NOT Cntr2 AND Cntr1 AND Cntr0 ) OR ( AGOOD AND NOT Cntr3 AND
Cntr2 AND NOT Cntr1 AND NOT Cntr0 ) OR ( AGOOD AND NOT Cntr3 AND Cntr2 AND NOT Cntr1 AND
Cntr0 )

//VMONB Status Output Logic
Eq 9: OUT14 = ( VMON1_B AND Cntr3 AND NOT Cntr2 AND NOT Cntr1 AND NOT Cntr0 ) OR (
VMON2_B AND Cntr3 AND NOT Cntr2 AND NOT Cntr1 AND Cntr0 ) OR ( VMON3_B AND Cntr3 AND NOT
Cntr2 AND Cntr1 AND NOT Cntr0 ) OR ( VMON4_B AND Cntr3 AND NOT Cntr2 AND Cntr1 AND Cntr0 )
OR ( VMON5_B AND Cntr3 AND Cntr2 AND NOT Cntr1 AND NOT Cntr0 ) OR ( VMON6_B AND Cntr3 AND
Cntr2 AND NOT Cntr1 AND Cntr0 ) OR ( VMON7_B AND Cntr3 AND Cntr2 AND Cntr1 AND NOT Cntr0 )
OR ( VMON8_B AND Cntr3 AND Cntr2 AND Cntr1 AND Cntr0 ) OR ( VMON9_B AND NOT Cntr3 AND NOT
Cntr2 AND NOT Cntr1 AND NOT Cntr0 ) OR ( VMON10_B AND NOT Cntr3 AND NOT Cntr2 AND NOT Cntr1
AND Cntr0 ) OR ( VMON11_B AND NOT Cntr3 AND NOT Cntr2 AND Cntr1 AND NOT Cntr0 ) OR (
VMON12_B AND NOT Cntr3 AND NOT Cntr2 AND Cntr1 AND Cntr0 ) OR ( AGOOD AND NOT Cntr3 AND
Cntr2 AND NOT Cntr1 AND NOT Cntr0 ) OR ( AGOOD AND NOT Cntr3 AND Cntr2 AND NOT Cntr1 AND
Cntr0 )

//Start, DataIN, and Strobe IO Assignments
Eq 10: Start = Cntr3
Eq 11: DataIN = IN3
Eq 12: Strobe = IN4

//Logic for Output Controls
Eq 13: HVOUT1.ap = ( Strobe AND DataIN AND Cntr3 AND NOT Cntr2 AND Cntr1 AND Cntr0 ) AND
FPGA_prsnt
Eq 14: HVOUT1.ar = ( Strobe AND NOT DataIN AND Cntr3 AND NOT Cntr2 AND Cntr1 AND Cntr0 )
AND FPGA_prsnt
Eq 15: HVOUT2.ap = ( Strobe AND DataIN AND Cntr3 AND Cntr2 AND NOT Cntr1 AND NOT Cntr0 )
AND FPGA_prsnt
Eq 16: HVOUT2.ar = ( Strobe AND NOT DataIN AND Cntr3 AND Cntr2 AND NOT Cntr1 AND NOT
Cntr0 ) AND FPGA_prsnt
Eq 17: HVOUT3.ap = ( Strobe AND DataIN AND Cntr3 AND Cntr2 AND NOT Cntr1 AND Cntr0 ) AND
FPGA_prsnt
Eq 18: HVOUT3.ar = ( Strobe AND NOT DataIN AND Cntr3 AND Cntr2 AND NOT Cntr1 AND Cntr0 )
AND FPGA_prsnt
Eq 19: HVOUT4.ap = ( Strobe AND DataIN AND Cntr3 AND Cntr2 AND Cntr1 AND NOT Cntr0 ) AND
FPGA_prsnt
```

```
Eq 20: HVOUT4.ar = ( Strobe AND NOT DataIN AND Cntr3 AND Cntr2 AND Cntr1 AND NOT Cntr0 )
AND FPGA_prsnt
Eq 21: OUT5.ap = ( Strobe AND DataIN AND Cntr3 AND Cntr2 AND Cntr1 AND Cntr0 ) AND
FPGA_prsnt
Eq 22: OUT5.ar = ( Strobe AND NOT DataIN AND Cntr3 AND Cntr2 AND Cntr1 AND Cntr0 ) AND
FPGA_prsnt
Eq 23: OUT6.ap = ( Strobe AND DataIN AND NOT Cntr3 AND NOT Cntr2 AND NOT Cntr1 AND NOT
Cntr0 ) AND FPGA_prsnt
Eq 24: OUT6.ar = ( Strobe AND NOT DataIN AND NOT Cntr3 AND NOT Cntr2 AND NOT Cntr1 AND
NOT Cntr0 ) AND FPGA_prsnt
Eq 25: OUT7.ap = ( Strobe AND DataIN AND NOT Cntr3 AND NOT Cntr2 AND NOT Cntr1 AND Cntr0
) AND FPGA_prsnt
Eq 26: OUT7.ar = ( Strobe AND NOT DataIN AND NOT Cntr3 AND NOT Cntr2 AND NOT Cntr1 AND
Cntr0 ) AND FPGA_prsnt
Eq 27: OUT8.ap = ( Strobe AND DataIN AND NOT Cntr3 AND NOT Cntr2 AND Cntr1 AND NOT Cntr0
) AND FPGA_prsnt
Eq 28: OUT8.ar = ( Strobe AND NOT DataIN AND NOT Cntr3 AND NOT Cntr2 AND Cntr1 AND NOT
Cntr0 ) AND FPGA_prsnt
Eq 29: OUT9.ap = ( Strobe AND DataIN AND NOT Cntr3 AND NOT Cntr2 AND Cntr1 AND Cntr0 )
AND FPGA_prsnt
Eq 30: OUT9.ar = ( Strobe AND NOT DataIN AND NOT Cntr3 AND NOT Cntr2 AND Cntr1 AND Cntr0
) AND FPGA_prsnt
Eq 31: OUT10.ap = ( Strobe AND DataIN AND NOT Cntr3 AND Cntr2 AND NOT Cntr1 AND NOT
Cntr0 ) AND FPGA_prsnt
Eq 32: OUT10.ar = ( Strobe AND NOT DataIN AND NOT Cntr3 AND Cntr2 AND NOT Cntr1 AND NOT
Cntr0 ) AND FPGA_prsnt
Eq 33: OUT11.ap = ( Strobe AND DataIN AND NOT Cntr3 AND Cntr2 AND NOT Cntr1 AND Cntr0 )
AND FPGA_prsnt
Eq 34: OUT11.ar = ( Strobe AND NOT DataIN AND NOT Cntr3 AND Cntr2 AND NOT Cntr1 AND
Cntr0 ) AND FPGA_prsnt
Eq 35: OUT12.ap = ( Strobe AND DataIN AND Cntr3 AND NOT Cntr2 AND NOT Cntr1 AND NOT
Cntr0 ) AND FPGA_prsnt
Eq 36: OUT12.ar = ( Strobe AND NOT DataIN AND Cntr3 AND NOT Cntr2 AND NOT Cntr1 AND NOT
Cntr0 ) AND FPGA_prsnt

//FPGA_prsnt detection logic
Eq 37: FPGA_prsnt.D = FPGA_Prsnt
Eq 38: FPGA_prsnt.ar = ( Strobe AND DataIN AND Cntr3 AND NOT Cntr2 AND NOT Cntr1 AND
Cntr0 )
Eq 39: FPGA_prsnt.ap = ( Strobe AND NOT DataIN AND Cntr3 AND NOT Cntr2 AND NOT Cntr1 AND
Cntr0 ) OR ( Strobe AND DataIN AND Cntr3 AND NOT Cntr2 AND Cntr1 AND NOT Cntr0 )
```

Appendix B. Listing of FPGA Code

```
module PtM_250kSERDES(mclk, pclk, resetn, dstart, statA, statB,
              OUT1, OUT2, OUT3, OUT4, OUT5, OUT6, OUT7, OUT8, OUT9, OUT10, OUT11, OUT12,
              Data, Strobe, VMON1A, VMON2A, VMON3A, VMON4A, VMON5A, VMON6A, VMON7A,
              VMON8A, VMON9A, VMON10A, VMON11A, VMON12A, VMON1B, VMON2B, VMON3B, VMON4B,
              VMON5B, VMON6B, VMON7B, VMON8B, VMON9B, VMON10B, VMON11B, VMON12B, AGOOD );

input  mclk, pclk, resetn ;                    //  mclk = 8 MHz,  pclk = 250 kHz
input  dstart, statA, statB ;                  //  data bus signals from the CPLD

//  Inputs from Sequence code for transmission to Power Manager CPLD
input   OUT1, OUT2, OUT3, OUT4, OUT5, OUT6 ;
input   OUT7, OUT8, OUT9, OUT10, OUT11, OUT12 ;

output  Data, Strobe ;     //  Data & strobe outputs to Power Manager CPLD

output VMON1A, VMON2A, VMON3A, VMON4A, VMON5A, VMON6A ;
output VMON7A, VMON8A, VMON9A, VMON10A, VMON11A, VMON12A ;
output VMON1B, VMON2B, VMON3B, VMON4B, VMON5B, VMON6B ;
output VMON7B, VMON8B, VMON9B, VMON10B, VMON11B, VMON12B, AGOOD ;

reg      Data,  Strobe ;

reg    VMON1A, VMON2A, VMON3A, VMON4A, VMON5A, VMON6A ;
reg    VMON7A, VMON8A, VMON9A, VMON10A, VMON11A, VMON12A ;
reg    VMON1B, VMON2B, VMON3B, VMON4B, VMON5B, VMON6B ;
reg    VMON7B, VMON8B, VMON9B, VMON10B, VMON11B, VMON12B, AGOOD ;

reg    [3:0] statecntr ;                       //  state machine counter for data decode
reg    [2:0] pulsedelay ;               //  Pulse delay counter for Pulse output

//  internal variables for detecting start of data transmission
reg    start, detect ;

reg    pm_spare1, pm_spare2 ;//  spare bits for use with data stream

//  Start of data transfer to/from Power Manager device  >>>>>>>>>>>>>>>>>

//  detect rising edge of dstart signal to reset state counter
       always @ (negedge mclk or negedge resetn)
              begin
              if ( !resetn )
                     start <= 1'b0 ;
              else
                     if ( dstart & !detect )
                            start <= 1'b1 ;
                     else
                            start <= 1'b0 ;
              end
```

```
//  latch to insure only the rising edge of dstart is detected
      always @ (negedge pclk or negedge resetn)
            begin
            if ( !resetn )
                  detect <= 1'b0 ;
            else
                  if ( dstart & start )
                        detect <= 1'b1 ;
                  else
                        if ( !dstart )
                              detect <= 1'b0 ;
            end


//  assign state machine counter for data read
      always @ (negedge pclk or negedge resetn)
            begin
            if ( !resetn )
                  statecntr <= 4'b0000 ;
            else
                  if ( start )
                        statecntr <= 4'b0000 ;
                  else
                        statecntr <= statecntr + 1 ;
            end


//  capture VMON status data from data bus (statA, statB)

      always @ (negedge pclk or negedge resetn)
            if ( !resetn )
                  begin
                   {VMON1A, VMON2A, VMON3A, VMON4A, VMON5A, VMON6A}    <= 6'b000000 ;
                   {VMON7A, VMON8A, VMON9A, VMON10A, VMON11A, VMON12A, AGOOD} <= 7'b0000000 ;
                   {VMON1B, VMON2B, VMON3B, VMON4B, VMON5B, VMON6B}    <= 6'b000000 ;
                   {VMON7B, VMON8B, VMON9B, VMON10B, VMON11B, VMON12B} <= 6'b000000 ;
                  end
            else
                  case (statecntr)
                        4'b0000 : {VMON1A,  VMON1B}  <= {statA, statB} ;
                        4'b0001 : {VMON2A,  VMON2B}  <= {statA, statB} ;
                        4'b0010 : {VMON3A,  VMON3B}  <= {statA, statB} ;
                        4'b0011 : {VMON4A,  VMON4B}  <= {statA, statB} ;
                        4'b0100 : {VMON5A,  VMON5B}  <= {statA, statB} ;
                        4'b0101 : {VMON6A,  VMON6B}  <= {statA, statB} ;
                        4'b0110 : {VMON7A,  VMON7B}  <= {statA, statB} ;
                        4'b0111 : {VMON8A,  VMON8B}  <= {statA, statB} ;
                        4'b1000 : {VMON9A,  VMON9B}  <= {statA, statB} ;
                        4'b1001 : {VMON10A, VMON10B} <= {statA, statB} ;
                        4'b1010 : {VMON11A, VMON11B} <= {statA, statB} ;
                        4'b1011 : {VMON12A, VMON12B} <= {statA, statB} ;
                        4'b1100 : {AGOOD, pm_spare2} <= {statA, statB} ;
```

```
                        default : {pm_spare1, pm_spare2}     <= {statA, statB} ;
                        endcase

//  send OUTPUT data to Power Manager data out pin
      always @ (posedge pclk or negedge resetn)
            if ( !resetn )
                  begin
                        Data <= 1'b0 ;
                  end
            else
                  case (statecntr)
                        4'b0000 : Data <= 1'b1 ;//  FPGA Presnt to Power Manager
                        4'b0001 : Data <= 1'b0 ;//  FPGA Presnt to Power Manager -

//  CPLD looks for this bit to toggle.
                        4'b0010 : Data <= OUT1 ;//  OUT1  from FPGA to Power Manager CPLD
                        4'b0011 : Data <= OUT2 ;//  OUT2  from FPGA to Power Manager CPLD
                        4'b0100 : Data <= OUT3 ;//  OUT3  from FPGA to Power Manager CPLD
                        4'b0101 : Data <= OUT4 ;//  OUT4  from FPGA to Power Manager CPLD
                        4'b0110 : Data <= OUT5 ;//  OUT5  from FPGA to Power Manager CPLD
                        4'b0111 : Data <= OUT6 ;//  OUT6  from FPGA to Power Manager CPLD
                        4'b1000 : Data <= OUT7 ;//  OUT7  from FPGA to Power Manager CPLD
                        4'b1001 : Data <= OUT8 ;//  OUT8  from FPGA to Power Manager CPLD
                        4'b1010 : Data <= OUT9 ;//  OUT9  from FPGA to Power Manager CPLD
                        4'b1011 : Data <= OUT10 ;//  OUT10 from FPGA to Power Manager CPLD
                        4'b1100 : Data <= OUT11 ;//  OUT11 from FPGA to Power Manager CPLD
                        4'b1101 : Data <= OUT12 ;//  OUT12 from FPGA to Power Manager CPLD

                        default : Data <= 1'b0 ;
                        endcase

//  Create pulse delay counter to set width of Strobe output to Power Manager
      always @ (posedge mclk or negedge resetn)
            begin                       //  reset pulse delay counter when pclk is high
            if ( !resetn || pclk )
                  pulsedelay <= 1'b0 ;
            else                        //  When pclk is low increment pulse delay counter
                  if ( !pclk && !pulsedelay[2] )
                        pulsedelay <= pulsedelay + 1 ;
                  else                  //  Stop pulse delay counter when counter value > '100'
                        pulsedelay <= pulsedelay ;
            end


//  Send Strobe output signal to Power Manager which will latch the inputs
//  when Strobe is high - Strobe will stay high for 0.5 us
      always @ (posedge mclk or negedge resetn)
            begin
            if ( !resetn || pclk )      //  reset Strobe when pclk is high
                  Strobe <= 1'b0 ;
            else                        //  set Strobe high on falling edge of pclk
                  if ( !pclk && !pulsedelay[2] )
                        Strobe <= 1'b1 ;
```

```
                    else
                           Strobe <= 1'b0 ;
             end


//  End of data transfer to/from Power Manager device   >>>>>>>>>>>>>>>>>>>

endmodule
```