Fail-Safe Startup Sequencing During Field Upgrades with Platform Manager

June 2012 Application Note AN6088

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. This document focuses on ensuring reliable startup of the Platform Manger device that supports in-system programming. 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.

The CPLD section is normally used for sequencing and monitoring of power supplies. The FPGA section can be used to implement enhanced power management functions like VID or Fault Logging. The FPGA section can also be used to sequence digital control functions like resets and Power Good signals, or used to control very complex power sequences. Executing the power sequencing and monitoring from the FPGA section provides the possibility to execute background updates to the FPGA configuration in-system (i.e. field upgrades).

Background Programming and the Need for Fail-Safe Startup Sequencing

Board power management functions such as supply sequencing and reset generation controlled by the FPGA portion of the device are very critical in nature and any interruption in FPGA start-up or FPGA operation can be fatal to the system.

The FPGA configuration is stored in SRAM during runtime. A non-volatile map of the configuration is stored in Flash memory and loaded during power-up. This provides the possibility to re-program the FPGA Flash in-system in the background (without interrupting FPGA operation from the SRAM). The next time the FPGA is powered up, it will be configured according to the new image inside the Flash memory. See “Demonstrating the Concept” on page 5 for more details on this process.

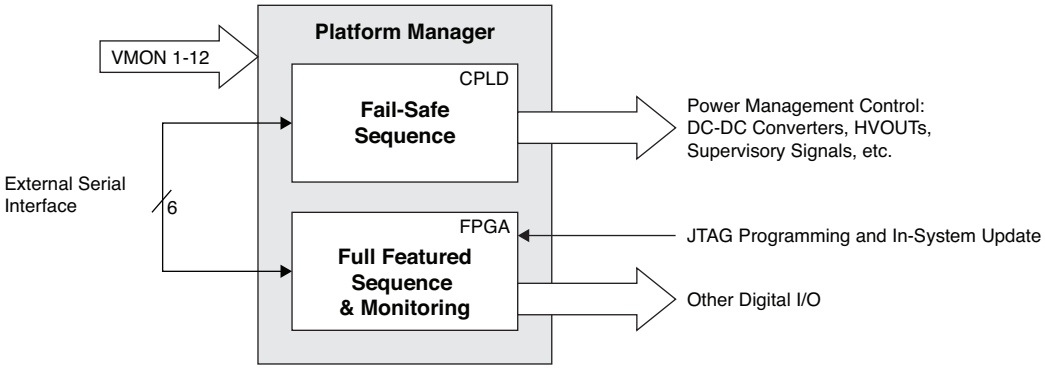
During background programming, the configuration Flash needs to be erased before it can be re-programmed. Power failures or other interruptions during these programming steps can result in incomplete or incorrect configuration of the FPGA. This application note provides a framework for safe background programming with a fall-back startup sequence in the CPLD of the Platform Manager to increase the reliability of field programming.

Fail-Safe Architecture Concept

In order to implement in-system upgradable sequencing and monitoring in the FPGA section of the Platform Manager, a dedicated communication link between the FPGA and CPLD sections must be established. This is accomplished using an external serial interface, consisting of six wires, which is implemented as a logic block in both the

CPLD and the FPGA. Under this concept, the primary sequence resides in the FPGA, while the CPLD contains a fall-back sequence. Figure 1 shows the system partitioning.

Figure 1. Fail-Safe Block Diagram



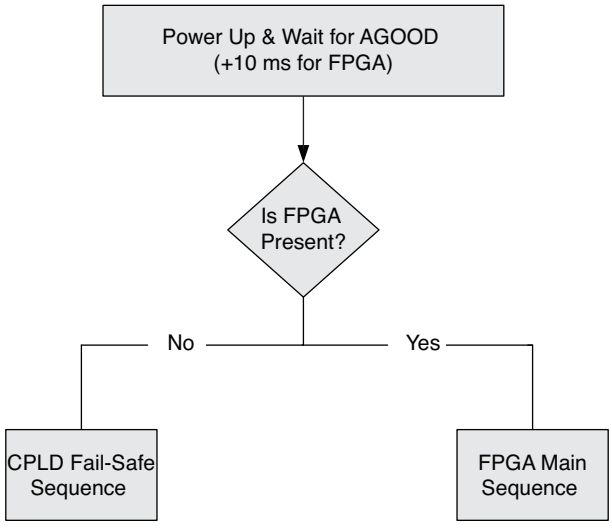
CPLD Design Description

Two designs are implemented within the CPLD:

- The fail-safe startup sequence
- A serializer-deserializer to allow the FPGA to control the outputs and monitor the inputs

In order to ensure safe operation, the CPLD includes the fail-safe startup sequence. This sequence begins with a short delay at power-up to allow the FPGA time to initialize and start up the serial interface. Next, the sequence will check the status of the FPGA Present node. This node is set true if the serial interface is operating properly and data packet is valid. If the FPGA node is not set then the CPLD fail-safe sequence takes control, as shown in Figure 2.

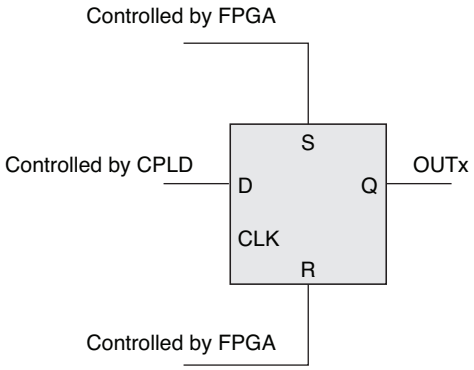
Figure 2. Fail-safe Startup Sequence



If the FPGA Present node is set then normal operation is in control. During normal operation, the CPLD portion transfers the VMON status to the FPGA portion, which implements the main sequence. The FPGA portion sends output control signals back to the CPLD. The serial link is implemented in supervisory equations in LogiBuilder (shown in "Appendix A. Listing of CPLD Supervisory Equations" on page 9). The serial link is presented in detail later in this document.

Implementation of the fail-safe sequence is application specific. The outputs can be controlled by both the CPLD fail safe sequence and the serial interface from the FPGA. The implementation is shown in Figure 3.

Figure 3. Control of the Outputs

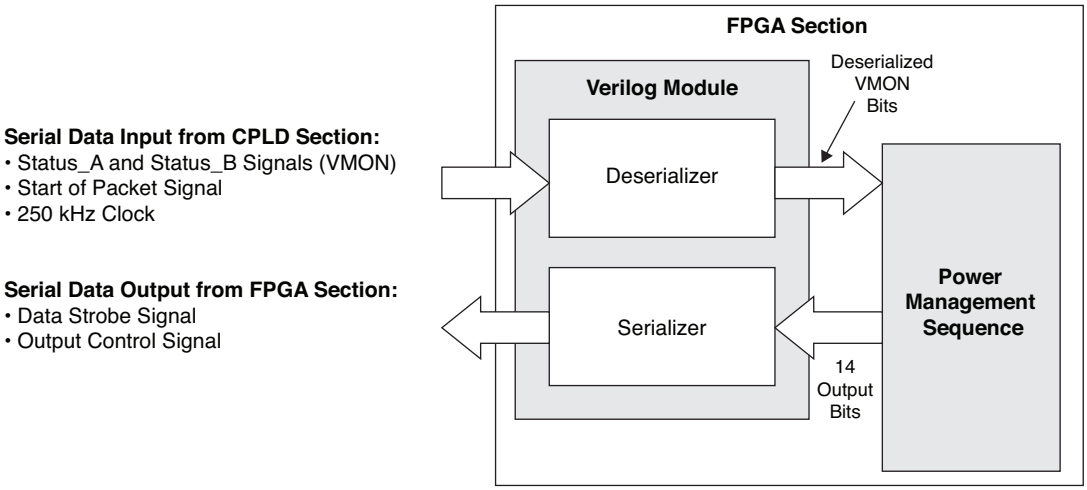


FPGA Design Description

The FPGA design includes the serial-deserializer block for communication to the CPLD and a power management sequence. The FPGA receives the VMON and input status from the deserializer as inputs to its power management sequence. The power management sequence provides output control bits to the serializer which are sent to the CPLD for control of the outputs. The concept is shown in Figure 4.

The Verilog code for the serializer/deserializer logic is shown in "Appendix B. Listing of FPGA Code" on page 11. The power management sequence in the FPGA is application-specific and not discussed in this application note.

Figure 4. FPGA Section of the Power Management Design



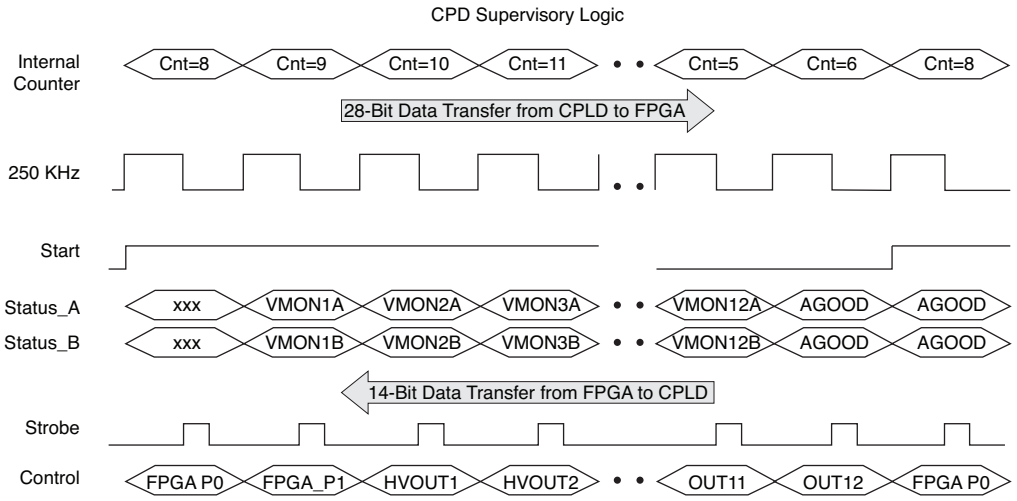
Serial Protocol

The logic required to implement the serial link is contained in the Supervisory equations section of the CPLD. The serial link sends the VMON status bits and internal AGOOD signal to the FPGA over two status signals. The Supervisory equations in the CPLD also decode the control signal received from the FPGA to turn on or off the power supplies. The data is communicated in 14 clock packets.

The supervisory section of the CPLD implements a 4-bit counter that controls the 28 bits sent in each data transmission frame (two status lines times 14 clocks). 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. The rising edge of the Start signal is used to indicate the beginning of a new data frame. 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 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 5. 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 output the send 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 5.

Supervisory logic equations are used to perform de-serialization functions in the CPLD section. This equation captures 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 9. 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. The Control signal is named data in the Verilog code shown in "Appendix B. Listing of FPGA Code" on page 11.

Guidelines for Background Programming

The fail-safe startup concept presented in this application note is designed to ensure a reliable system restart in case of a failure during re-programming. In order to implement this concept in a field re-programming scenario, a power cycle operation on the Platform Manager device is required. This will require that the entire system is shut down before the Platform Manager device is restarted with the new program.

The system is programmed with the combined FPGA/CPLD JEDEC prior to deployment in the field. This file should include the FPGA based sequence, the serial links, and a fail-safe sequence in the CPLD. Download this file during production using ispVM™ System or another programming tool.

After deployment in the field, upgrades are sometimes required. The Diamond portion of the design flow outputs an independent FPGA JEDEC which can be used to upgrade only the FPGA portion of the device. "Demonstrating the

Concept" on page 5 gives more details on generating an FPGA JEDEC only.

The procedure for background programming in field is shown below.

- **Phase1**: Execute the remote Background Flash Programming step for the FPGA only. The Demos/LPTM10-12107/ LPTM10_12107_FPGA_Background_Program.xcf file provided with the demo files gives a starting point for accessing the FPGA and bypassing the CPLD in Platform Manager. The FPGA and CPLD will continue operating normally during this step, with no interruptions.
- **Phase2**: Once background programming completes, the Platform Manager device and the system will need to be powered down. This can be accomplished either remotely in hardware or based on manual user intervention. This mechanism for system power-down and restart is application specific.
- **Phase3**: Power is restored to the Platform Manager device. The FPGA configuration stored in Flash is transferred to SRAM for operation.
- **Phase4**: After the power-on reset, the Platform Manager CPLD will confirm that the Platform Manager FPGA is transmitting properly. If the FPGA is properly configured, the upgraded sequence in the FPGA will be in control of the system.

  If the FPGA is not properly configured, the CPLD fail-safe sequence will take over. This fail-safe sequencing can support the re-programming hardware, in order to execute the field upgrade procedure again (restarting at step 1).

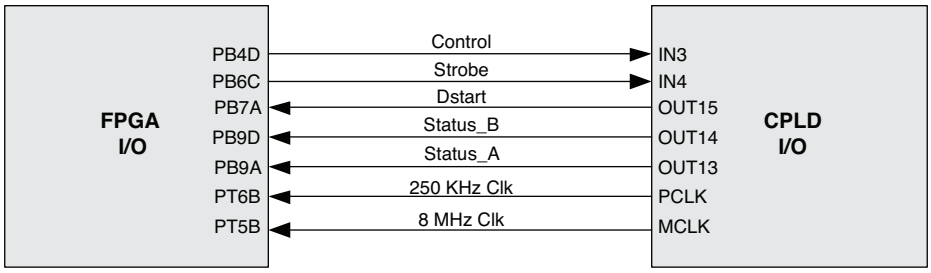
Demonstrating the Concept

This application note is provided with a set of design files which are available on the Lattice web site. A demo project can be found in the AN6088/Demos/LPTM10_12107 directory of the zip file. This demo project is separated into a PAC-Project directory and a Diamond Project directory.

The PAC-Project includes the CPLD Control receiver and VMON Status transmitter, a demonstration fail-safe sequence and the analog configuration data. The Diamond project includes a set of directories with Verilog files and project files used to build the FPGA JEDEC file inside Diamond. This information together is used to build the Platform Manager JEDEC file.

The serial link used in the demo is comprised of the connections below. These are all present on the Platform Manager Evaluation Board so no hardware updates are required for this demo.

Figure 6. Connections Used in Demonstration



To use this demo with the Platform Manager Evaluation Board, follow the steps listed below.

1. Move both slider pots to the lower end of their range (toward the DIP switches)
2. Using ispVM System, download the **Demos/LPTM10_12107/PAC_Project/Demo_AN6088_LPTM10-1207.JED** file.
3. The evaluation board should now be operating under an FPGA controlled sequence. Try moving the slider pots up and down. The LEDs on both sections of the board (D3-6 and D21-24) should be flashing in sync. LEDs D3-6 are controlled directly by the FPGA, while D21-24 are controlled by the CPLD via the serial link.

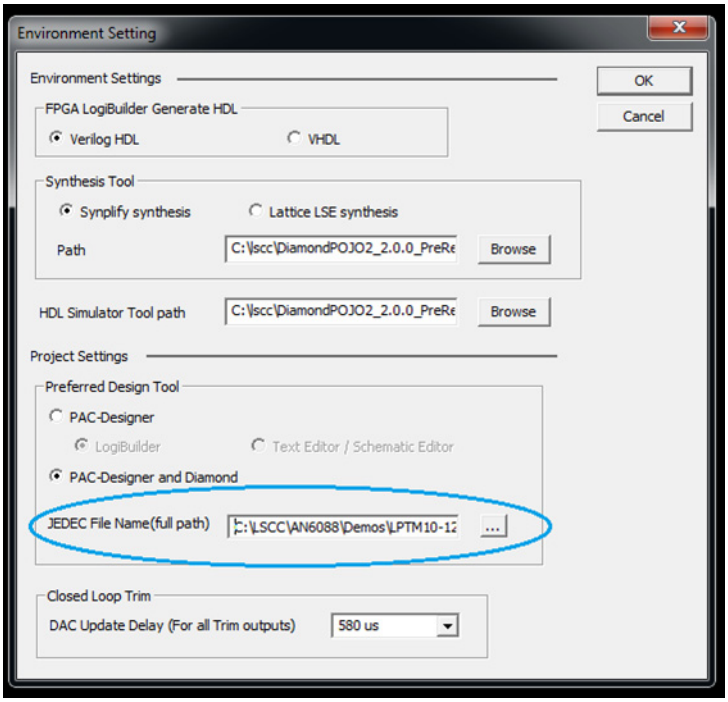
4. Open the .xcf configuration file **Demos/LPTM10_12107/LPTM_12107_FPGA_Background_Erase.xcf**. Click **GO** and this configuration file will background erase the Platform Manager FPGA.
5. The FPGA controlled sequence should still be running, with LEDs D3-6 in sync with LEDs D21-24. Power-down the evaluation board by removing the USB cable or external power source.
6. The FPGA SRAM is empty since the non-volatile Flash image was erased. LEDs D3-6 will not blink, however LEDs D21-D23 will be counting up. This is demonstrating a limited fail-safe sequence.
7. Open the .xcf configuration file **Demos/LPTM10_12107/LPTM_12107_FPGA_Background_Program.xcf**. You will need to update the FPGA background program operation to use the upgraded FPGA JEDEC. You can find this file at **Demos/LPTM10_12107/Diamond_Project/AN6088_implEnhanced.JED**. Click **GO** and this configuration file will background program the Platform Manager FPGA.[1]
8. Cycle the power again on the evaluation board. The evaluation board will now be operating under an enhanced sequence in the FPGA. Try moving the slide pots up and down again. Notice that D3-6 and D21-24 are now operating in complementary fashion.

Before modifying the demo files, the PAC-Designer project file needs to be updated. The pointer to the Diamond generated FPGA JEDEC file needs to be updated depending on the installation directory of the files. 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 7. The JEDEC File name should be updated to:

Installation_path\AN6088\demos\LPTM_12107\Diamond_Project\impl1\Demo_AN6088_FPGA_impl1.jed

To update only the FPGA portion of the Platform Manager, use the LPTM_12107_FPGA_Background_Program.xcf configuration, with the above FPGA JEDEC file used as the data file. This will update the FPGA Flash configuration, which will be copied to SRAM on the next power cycle, while leaving the CPLD configuration untouched.

Figure 7. Updating the FPGA JEDEC Pointer in PAC-Designer



1. Note: The Platform Manager evaluation board includes an FTDI USB-to-serial converter IC on-board, which is used for programming operations with the standard USB cable. This interface will issue a reset command to the CPLD, even when a bypass is requested. This might result in the CPLD LEDs blinking or dropping out during background FPGA operations. This will not occur when programming the FPGA through a dedicated programming cable or using ispVM embedded with a microcontroller in the field.

Working with the Design Templates

The Design Templates are provided as building blocks for user projects. Project elements like voltage thresholds, sequence steps and output assignments are always specific to the application. The design templates use default values to support configuration by the user.

The design templates are device specific and the Platform Manager design templates are provided in two separate directories (PAC_Files and Diamond_Project). The Platform Manager template uses the design flow of PAC-Designer plus Diamond. This means the FPGA I/O and logic are all handled in Diamond. The combined JEDEC 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 (for details on how to update this path, see "Demonstrating the Concept" on page 5). You will need to build your Diamond JEDEC before starting in PAC-Designer.

The Diamond_Project directory contains the minimum set of files to get started with building a design for the FPGA. Full descriptions of these files are found in the docs/readme.txt included with the files. Inside the Diamond_Project/impl1/source directory are the two Verilog source files: DT6088_FPGA_Top.v and DT6088_FPGA_250kSERDES.v.

The DT6088_FPGA_250kSERDES.v contains the serial link sub-module. This sub-module should not be modified. DT6088_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 placeholder for the sequence inclusion. These two files are included in the Diamond Project as a starting point. For additional details on designing in Diamond, 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 (for use in the 36-rail design described in AN6089, Scalable Centralized Power Management with Field Upgrade Support). In the .PAC files, the CPLD LogiBuilder code contains the supervisory equations for the serial link. The templates also assign the inputs and outputs used by the serial link. The only modification which should be made to these equations are the input and output pins used by the serial link. The output assignments can be modified in equations 8-10 (see "Appendix A. Listing of CPLD Supervisory Equations" on page 9), while the input assignments can be modified in equations 11-12.

The sequencer state machines contain a NOP placeholder at step 4. This can be replaced with the fail-safe sequence. All the analog settings and additional I/O assignments will need to be made before the file is usable.

Implementation

Table 2. Performance and Resource Utilization[1]

Device	FPGA LUTs	FPGA Slices	FPGA I/Os	CPLD Macrocells	CPLD Timers	CPLD Product Terms	CPLD I/Os	VMONs
LPTM-12107	70	39	16	32	1	141	18	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. PAC-Designer LogiBuilder options were set to defaults. Utilization numbers include demo program in addition to serial link logic.

Summary

This application note has provided a framework for implementing a fall-back startup sequence in the CPLD sequence of the Platform Manager to increase the reliability of field programming. The primary power management sequence is implemented in the FPGA while the analog data is shared between the CPLD and FPGA via serial link. A procedure for background programming has also been discussed. Supervisory code in the CPLD and serializer/deserializer Verilog module of the FPGA portion is provided in the appendices.

References

- DS1036, Platform Manager Data Sheet
- AN6089, Scalable Centralized Power Management with Field Upgrade Support
- 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	Updated with instructions for working with the design files.
June 2012	01.2	Clarifications on working with the design files. New section on guidelines for background programming.

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_12supply(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
```