
Mixed Signal Power Manager for SmartFusion2 Reference Design

User's Guide



Table of Contents

Introduction	6
Using MPM	8
Installation	8
Hardware Setup	8
MPM Reference Design Demo	14
MPM Design Files	20
MPM GUI Overview	24
MPM for SmartFusion2 Reference Design Flow	33
Purpose	33
Tools/Resources Used	33
Building the SoftConsole Firmware Image	34
MSS ENVM Data Storage Clients	35
Libero SoC Design Flow	38
STAPL File Generation and GUI Integration	38
MPM for SmartFusion2 Reference Design I²C	39
I ² C Operation	39
I ² C Register Map	42
SmartFusion2 MPM Data Logging	59
Data Logging	59
MPM for SmartFusion2 Reference Design Trimming	64
Operation	64
GUI Operation	66
MPM Firmware Notes	68
Introduction	68
MPM and FreeRTOS	68
Compile Time Options in the MPM Code	69
MPM Daughter Board Hardware Guide	71
Introduction	71
Kit Contents	72
Related Information	72
Board Description	72
Installation and Switch Settings	76
Manufacturing Information	79
Product Support	81

Customer Service.....	81
Customer Technical Support Center.....	81
Technical Support	81
Website	81
Contacting the Customer Technical Support Center	81
ITAR Technical Support.....	82
List of Changes	83
List of Changes	83

Introduction

This document explains how to use the mixed signal power manager (MPM) reference design for the SmartFusion®2 customizable System-on-Chip (cSoC) device, using the SmartFusion2 Digital MPM Daughter Board (SF2-DMPM-DB) connected to the SmartFusion2 Development Kit. It assumes the use of SF2 MPM 6.1.100 firmware or greater.

You should read the [SmartFusion2 Development Kit User's Guide](#) in conjunction with this guide for full details of the Development Kit.

MPM is a reference design that is programmed into the SmartFusion2 cSoC device which can be controlled and configured by the MPM graphical user interface (GUI) via a standard 2-wire Inter-Integrated Circuit (I²C) interface.

Based on Microsemi's SmartFusion2 cSoC, MPM delivers superior power monitoring, power sequencing, closed-loop trimming, and power-up and power-down control of up to 64 power supplies which can be a mix of Analog Points of Load (APOLs) and Digital Points of Load (DPOLs). This adds more flexibility, reduces total parts count on the board level, and increases system reliability by eliminating single points of failure.

You do not need to use FPGA design tools to configure power management sequencing, levels, or thresholds; the MPM design is programmed into the device through an easy-to-use standalone GUI tool. The GUI enables you to configure power management and drive output signals as the monitored voltages meet or deviate from the user-programmed operating limits, all without opening the Microsemi Libero System-on-Chip (SoC) tools.

The MPM GUI tool writes register values to on-chip embedded flash memory, which control power sequencing and monitoring functionality of the MPM reference design. The MPM GUI tool programs the board and configuration settings by launching FlashPro software which communicates with the target over a USB connected on board or separate FlashPro programmer device. In addition the MPM GUI can configure, control and monitor the MPM target over I²C using an appropriate USB to I²C Dongle.

This document assumes some knowledge of MPM or similar application-specific standard product (ASSP) applications.

Note: All photographs in this revision of the document are based on development boards and are not representative of the release hardware. The document is currently based on the Rev A SF2-DMPM-DB but will note any differences for the Rev B SF2-DMPM-DB.

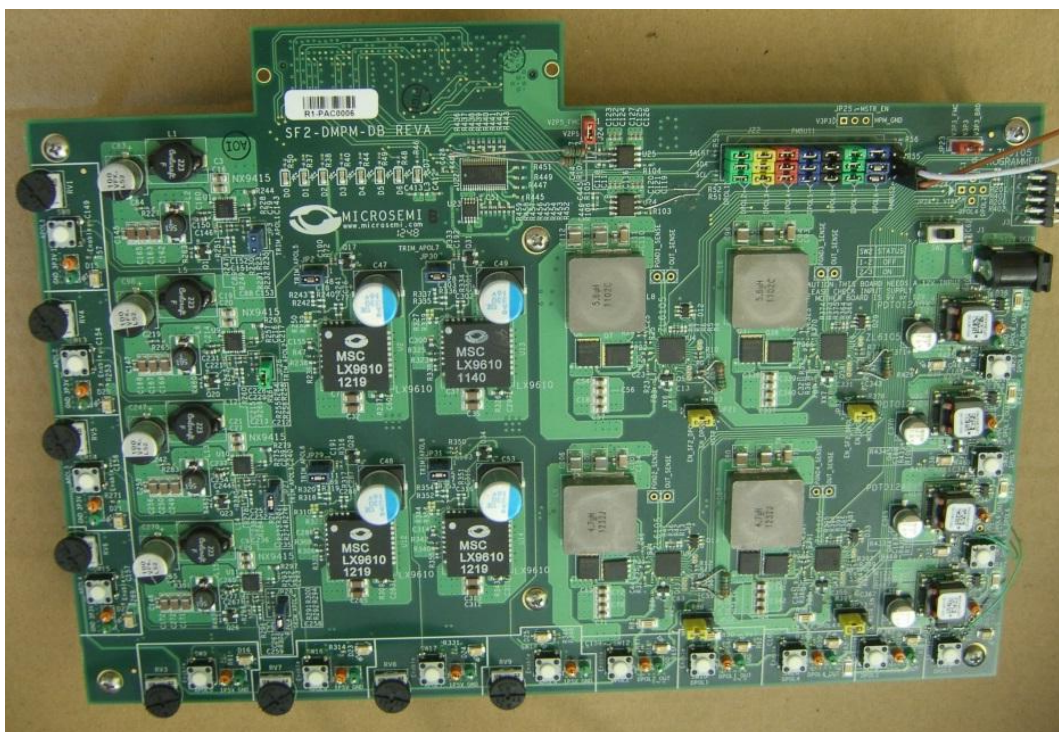


Figure 1 · SF2-DMPM-DB Rev A

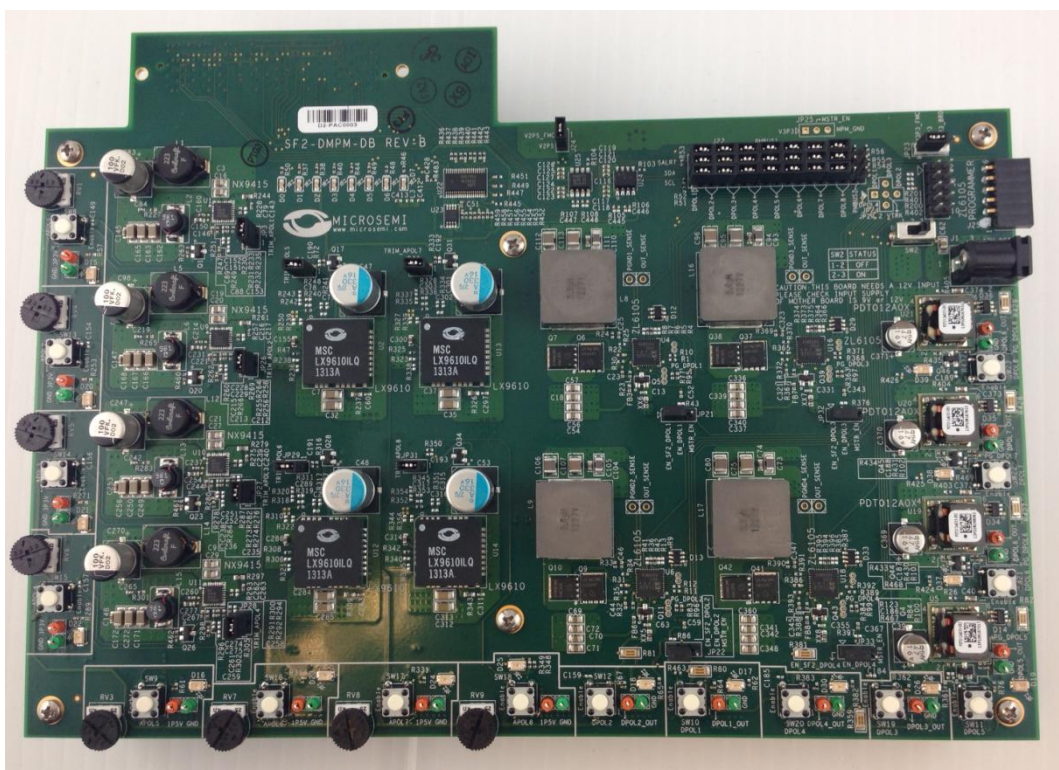


Figure 2 · SF2-DMPM-DB Rev B

Using MPM

Installation

Run the installer and follow the installation wizard instructions. By default MPM installs into the `C:\Microsemi\SF2_MPM_RefDesign_v6.1` folder and this is the recommended location for it. In particular avoid installing it in a folder than is very deeply nested or has a very long path name otherwise some tools may encounter problems accessing files with names longer than 259 characters¹.

Once installed MPM adds the following options to the Windows start menu:

- ✓ Start
 - ✓ All Programs
 - Microsemi SmartFusion2 MPM Reference Design v6.1
 - ❖ Browse Design Files
 - Opens in Windows Explorer the folder containing the MPM Libero SoC hardware and SoftConsole firmware projects
 - ❖ MPM GUI
 - Runs the MPM GUI
 - ❖ Uninstall
 - Uninstalls MPM

Hardware Setup

This section explains how to prepare the hardware for programming the MPM reference design and running the default demonstration.

You will need the following:

- SmartFusion2 Development Kit Board
(http://www.actel.com/products/hardware/devkits_boards/smartfusion2_dev.aspx)
and
- SF2 Digital MPM Daughter Board (SF2-DMPM-DB)
and
- Devantech/Robot Electronics USB-ISS communications module
(http://robot-electronics.co.uk/acatalog/USB_I2C.html)

¹. See this Microsoft MSDN article for more on path length issues: <http://msdn.microsoft.com/en-us/library/aa365247.aspx>.

SmartFusion2 Development Kit Board Setup

The following provides basic setup information for the hardware and firmware for the DMPM 6.1.100 Release. The setup is based on the initial porting of MPM to the SmartFusion2 platform with the Rev B Dev Kit and the Rev A SF2-DMPM-DB (see note below for Rev B SF2-DMPM-DB).

Before applying power to the MPM system, ensure that the following jumpers are installed:

Dev Kit Setup

The following jumper settings are required on the SmartFusion2 Dev Kit to route the signals required for the DMPM board to the FMC connector:

J174 pins 2 and 3 jumpered	J172 pins 2 and 3 jumpered	J184 pins 2 and 3 jumpered
J175 pins 2 and 3 jumpered	J179 pins 2 and 3 jumpered	J195 pins 2 and 3 jumpered
J200 pins 2 and 3 jumpered	J194 pins 2 and 3 jumpered	J202 pins 2 and 3 jumpered
J210 pins 2 and 3 jumpered	J201 pins 2 and 3 jumpered	J209 pins 2 and 3 jumpered
J155 pins 2 and 3 jumpered	J146 pins 2 and 3 jumpered	J140 pins 2 and 3 jumpered
J138 pins 2 and 3 jumpered	J158 pins 2 and 3 jumpered	J154 pins 2 and 3 jumpered
J143 pins 2 and 3 jumpered	J141 pins 2 and 3 jumpered	J111 pins 2 and 3 jumpered
J133 pins 1 and 2 jumpered	J214 pins 2 and 3 jumpered	J213 pins 2 and 3 jumpered
J178 pins 2 and 3 jumpered	J188 pins 2 and 3 jumpered	J187 pins 2 and 3 jumpered
J197 pins 2 and 3 jumpered	J196 pins 2 and 3 jumpered	
J145 pin 2 connected via flying lead to J199 pin 3		

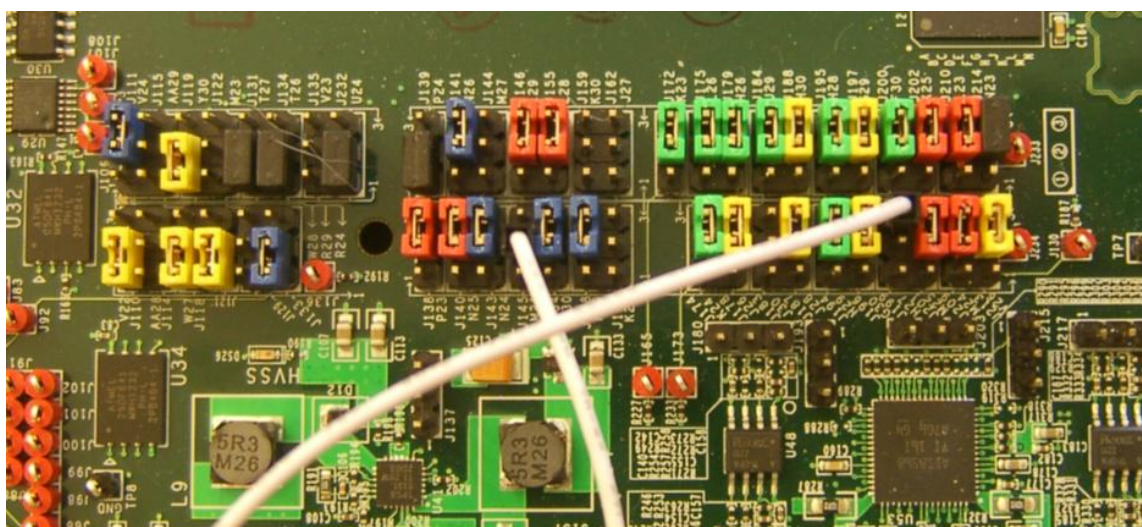


Figure 3 - SF2 Dev Kit Jumpers

The DIP switch SW10 is used to provide 4 of the MPM inputs and also to select optional operational modes for MPM.

The default settings for SW10 are:

- SW10.1 - off
- SW10.2 - off
- SW10.3 - off
- SW10.4 - off

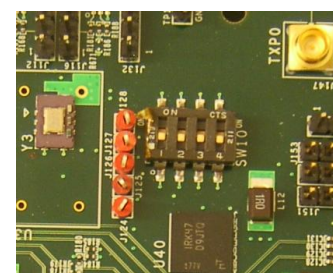


Figure 4 - SF2 Dev Kit SW10

Note: If you are using a Rev B SF2-DMPM-DB board the following changes apply:

1. J183 needs to be jumpered on pins 2 and 3.
2. Remove the jumper lead between J145 and J199

SF2 DMPM Daughter Board Setup

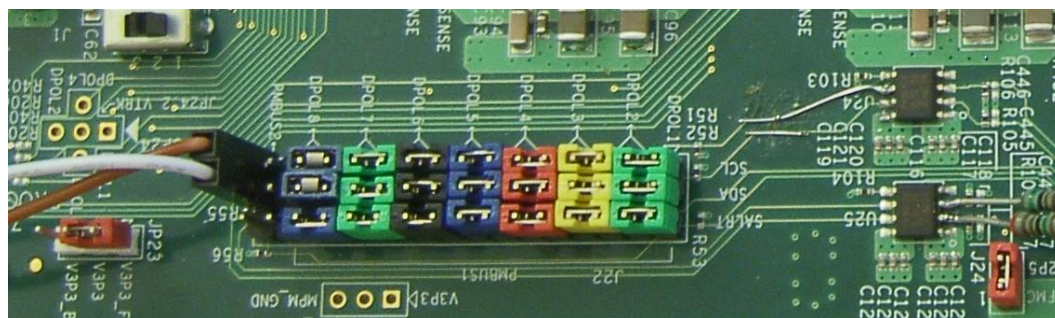


Figure 5 · SF2-DMPM-DB I2C Jumpers

JP23 should have pins 1 and 2 jumpered and J24 should be jumpered to provide 3.3V and 2.5V supplies to the SF2-DMPM-DB board from the FMC connector.

J22 should have 7 x 3 jumpers installed as shown above, to link the 8 DPOLs to PMBUS1. The last pair of pins labeled PMBUS2 are unjumpered but are used to connect the Devantech USB Adaptor as shown below. A suitable ground connection for this can be found on J3 - second pin from the left on the bottom row.

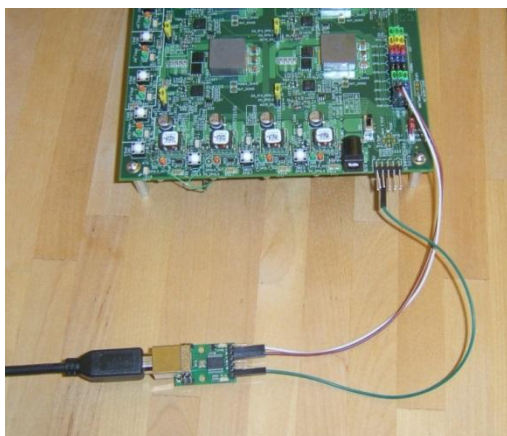


Figure 6 · SF2-DMPM-DB Rev A Slave I2C Connection

SF2-DMPM-DB Rev B boards have a dedicated connector for the Devantech USB Adaptor (J25).

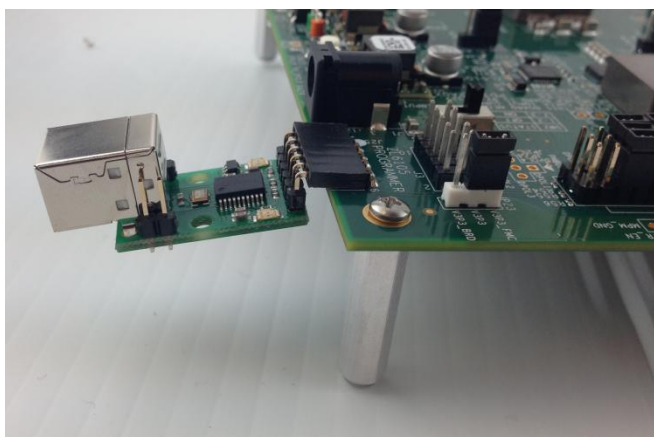


Figure 7 · SF2-DMPM-DB Rev B Slave I2C Connection

For the APOLs on the SF2 DMPM-DB JP3, JP26, JP27, JP28, JP2, JP29, JP30 and JP31 must be jumpered to enable trimming.

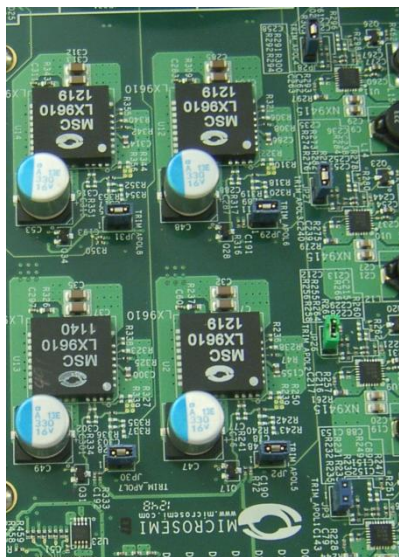


Figure 8 · SF2-DMPM-DB Trim Jumpers

For the DPOLs on the SF2-DMPM-DB board JP21, JP22, JP32 and JP33 must be jumpered on pins 1 and 2 to route the regulator enables of the first 4 DPOLs to the FMC connector.

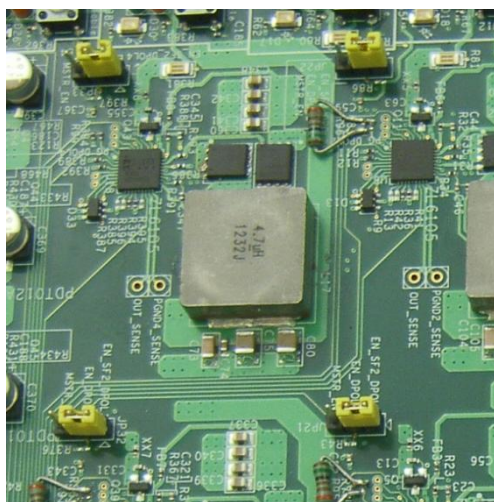


Figure 9 · SF2-DMPM-DB DPOL Enable Jumpers

Programming MPM

The first time you use MPM you need to use the MPM GUI to program the MPM design to the target hardware.

- For the SmartFusion2 Development Kit Board
 - ✓ Connect the 12V 6A power supply to the J18 12V INPUT SUPPLY connector
 - ✓ Connect a FlashPro4 programmer to the J59 – FP4 Header
 - ✓ Connect a mini USB cable between your PC and the FlashPro4
 - ✓ Power the board on using SW7
- Run the MPM GUI.
 - ✓ Select **Data > FlashPro > Choose STAPL template** and select the appropriate STAPL file for your target hardware from the `C:\Microsemi\SF2_MPM_RefDesign_v6.1\template` folder.¹
 - ✓ Select **Data > FlashPro > FlashPro Setup** and browse to and select the FlashPro software executable in your Libero SoC v11 or later installation.
 - ✓ Select **Data > FlashPro > Write NVM & Fabric** and the MPM GUI will launch FlashPro and program the full design (MSS configuration, MPM firmware, MPM configuration data in ENVN along with the FPGA fabric logic) to the target hardware. You will see a Command Shell “DOS Box” appear reporting progress. When prompted you can close this. At this stage the MPM target should be programmed with the MPM reference design.
- Power off the SmartFusion2 board by disconnecting the USB cable and power supply cables.
- Connect the SF2-DMPM-DB to the SmartFusion2 Development Kit Board by connecting their respective FMC connectors.
- Connect the 12V power supply to the SF2-DMPM-DB J1 12V DC IN connector
- Reconnect power and USB to the SmartFusion2 Dev Kit as described above.
- Power the SF2-DMPM-DB on using SW2
- Reset the SmartFusion2 board by pressing SW9 RESET.
- Once the initial programming of the MPM design to the target has been performed then MPM configuration setting changes alone can be programmed to the target's MSS ENVN over an I²C connection between the MPM GUI and the MPM target hardware using **Data > I²C > Read Config Via I²C** and **Write Config Via I²C**.

I²C Setup

Use the MPM GUI to connect to the MPM target via I²C with the included Devantech/Robot Electronics USB-ISS communications module. Ensure that the USB-ISS Power Link has a jumper removed for 3.3V operations for compatibility with the MPM target.

Use the included standard USB A/B cable to connect the USB-ISS to your PC. Install the drivers that are bundled with MPM GUI install in the `C:\Microsemi\SF2_MPM_RefDesign_v6.1\Devantech_USB-ISS_drivers` folder.

After installing the drivers, and plugging the USB-ISS module into a spare USB port, you need to know to which COM port it has been assigned to. This will vary from system to system depending on how many COM ports you currently have installed. To find out where it is, right-click on your **Computer** desktop icon and select **Properties > Device Manager**. Now scroll down and open the **Ports (COM & LPT)** tab. You should see the USB serial port listed - COM5 in the example below. If you want to change the COM port number - just right-click on it, select **properties > advanced > Port Settings > COM port number** from the available list. The COM port default settings are sufficient.

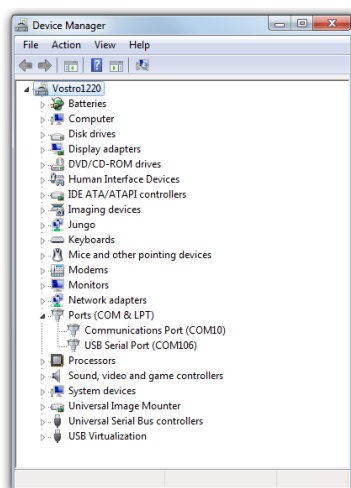


Figure 10 · Device Manager

To allow the MPM GUI to communicate with MPM, connect the Devantech USB-ISS to the SmartFusion2 board's MSS I2C_1 by installing female to female jumper cables between the following pins:

Table 1 · Devantech/Robot Electronics USB-ISS Connections

I ² C Signal Board	SCL	SDA	GND
Devantech/Robot Electronics USB-ISS http://www.robot-electronics.co.uk/htm/usb_iss_tech.htm	SCL (I/O 3)	SDA (I/O 4)	0V/Ground
SF2 DMPM Board Rev A	J22P pin 16	J22P pin 32	3 pin 8

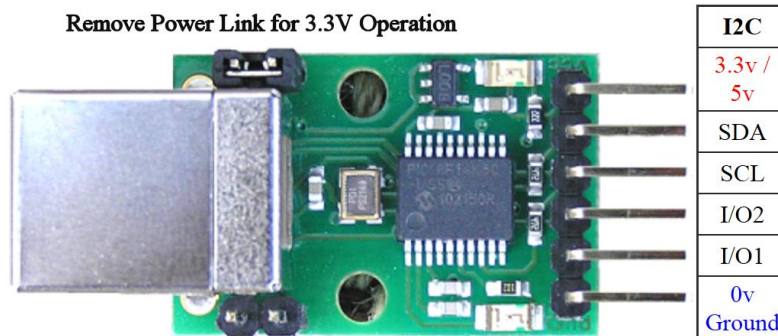


Figure 11 · Devantech/Robot Electronics USB-ISS

As explained earlier note the COM port assigned to the USB-ISS communications module. Run the MPM GUI, select **Data > I²C > Test I²C Dongle**. Select the USB-ISS COM port from the **Communications > Port**: drop-down list. Select **Test Dongle** from the **Activity > Action** drop-down list and click the **Go** button. If the test is successful then you should see the following in the **Status** text box.

Starting 'Test of Communications with Dongle'

I2C Speed set to 1 MHz

Communicated with dongle

Firmware Version: 07 02 80

Serial No: [00001425]

Test of Communications with Dongle Completed

If the test fails then you will see this and you need to review and correct the setup:

Starting 'Test of Communications with Dongle'

Failed to open I2C port

The DIP switch, SW10 is used to provide another 4 of the MPM inputs and also to select optional operational modes for MPM.

SW10.1 is connected to MPM input 5.

SW10.2 is connected to MPM input 6.

SW10.3 is connected to MPM input 7.

SW10.4 is connected to MPM input 8.

Additionally:

SW10.1 selects whether MPM shuts down if a POL goes to OV2 or UV2 or not. When off MPM does not shut down and when on MPM does shut down.

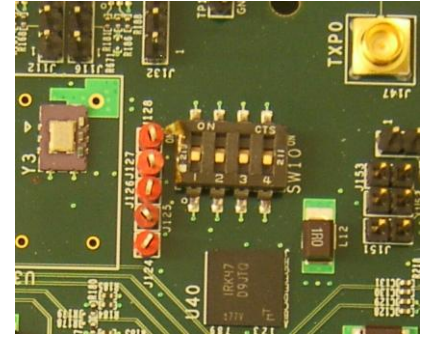


Figure 14 · SF2 Dev Kit SW10

SW10.2 selects whether MPM reloads the configuration in eNMV with known good values on power up or not.

SW10.3 and **SW10.4** allow adjustment of the DEADTIME parameter for DPOLs 1 to 4 as follows:

If **SW10.3** is off, DEADTIME is not set by MPM and will either have the default values as defined in the ZL6105 documentation if the DPOL has just powered up or the last value set by MPM if the DPOL has not been power cycled since the last write to DEADTIME.

If **SW10.3** is on, **SW10.4** off selects a DEADTIME of 40nS and **SW10.4** on selects a DEADTIME of 24nS.

The 8 LEDs on the DMPM board display the state of the first 8 MPM outputs. With the default configuration, these LEDs represent the state of the first 8 APOLs. When an LED is on, the corresponding APOL is in a non nominal state. When an LED is off, the corresponding channel is in the nominal state. This is useful when sequencing the APOLs as you can adjust the pot for a channel until it enters the nominal state by watching for the corresponding LED to turn off.

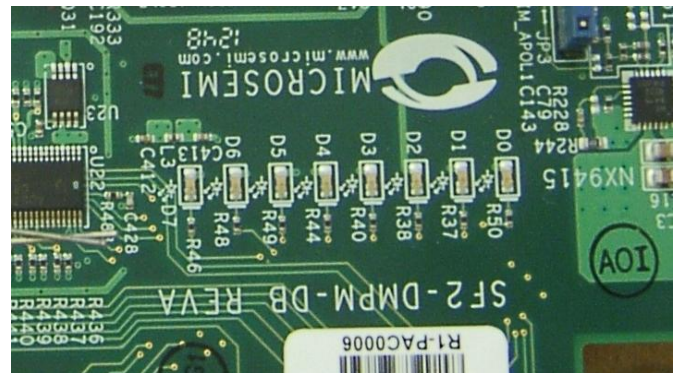


Figure 15 · SF2-DMPM-DB LEDs

If you press SW3 to initiate power-up sequencing, the MPM state changes to Starting and you can see the various regulator enabled LEDs on the SF2-DMPM-DB turning on in sequence. If the status does not change to Started and the power-on sequence restarts, you can adjust the voltage of the individual APOL channels via the associated potentiometers to complete the sequencing. If open-loop trimming is enabled, the open-loop trim pin voltage will only achieve nominal value if the potentiometer is suitably adjusted.

When power sequencing has completed and all regulators have reached nominal voltage, the status changes to Started and APOL closed-loop trimming is also enabled if applicable. Closed-loop trimming keeps the APOL channel output voltage at the nominal value specified in the GUI, even when the potentiometer is adjusted. You can disable trimming by removing the Trim jumper for a given regulator. The following list shows which jumper configures which APOL:

Table 3 · MPM Status Description

Jumper	APOL
JP3	1
JP26	2
JP27	3
JP28	4
JP2	5
JP29	6
JP30	7
JP31	8

Removing one of these jumpers disables closed-loop trimming for the associated APOL and you see that the output voltage can be varied using the potentiometer for that channel. Reinstalling the jumper reactivates closed-loop trimming and brings it back to nominal.

The characteristics of the channels configured in the reference design are as follows (**note**. The default demo setup for all APOLs is closed loop trimming):

- Channel A1
 - ✓ SF2-DMPM-DB APOL1
 - ✓ Microsemi NX9415 3.3 V nominal regulator
 - ✓ POT range when switched on approximately 3100 – 3600 mV
 - ✓ Optionally open and closed loop trimmed using CorePWM output 1 depending on MPM GUI configuration and SF2-DMPM-DB jumper settings.
- Channel A2
 - ✓ SF2-DMPM-DB APOL2
 - ✓ Microsemi NX9415 3.3 V nominal regulator
 - ✓ POT range when switched on approximately 3100 – 3600 mV
 - ✓ Optionally open and closed loop trimmed using CorePWM output 2 depending on MPM GUI configuration and SF2-DMPM-DB jumper settings.
- Channel A3
 - ✓ SF2-DMPM-DB APOL3
 - ✓ Microsemi NX9415 3.3 V nominal regulator
 - ✓ POT range when switched on approximately 3100 – 3600 mV
 - ✓ Optionally open and closed loop trimmed using CorePWM output 3 depending on MPM GUI configuration and SF2-DMPM-DB jumper settings.
- Channel A4
 - ✓ SF2-DMPM-DB APOL4
 - ✓ Microsemi NX9415 3.3 V nominal regulator
 - ✓ POT range when switched on approximately 3100 – 3600 mV
 - ✓ Optionally open and closed loop trimmed using CorePWM output 4 depending on MPM GUI configuration and SF2-DMPM-DB jumper settings.
- Channel A5
 - ✓ SF2-DMPM-DB APOL5
 - ✓ Microsemi LX9610 1.5 V nominal regulator
 - ✓ POT range when switched on approximately 1340 – 1650 mV
 - ✓ Optionally open and closed loop trimmed using CorePWM output 5 depending on MPM GUI configuration and SF2-DMPM-DB jumper settings.
- Channel A6
 - ✓ SF2-DMPM-DB APOL6
 - ✓ Microsemi LX9610 1.5 V nominal regulator
 - ✓ POT range when switched on approximately 1340 – 1650 mV
 - ✓ Optionally open and closed loop trimmed using CorePWM output 6 depending on MPM GUI configuration and SF2-DMPM-DB jumper settings.
- Channel A7
 - ✓ SF2-DMPM-DB APOL7
 - ✓ Microsemi LX9610 1.5 V nominal regulator
 - ✓ POT range when switched on approximately 1340 – 1650 mV
 - ✓ Optionally open and closed loop trimmed using CorePWM output 7 depending on MPM GUI configuration and SF2-DMPM-DB jumper settings.
- Channel A8
 - ✓ SF2-DMPM-DB APOL8
 - ✓ Microsemi LX9610 1.5 V nominal regulator
 - ✓ POT range when switched on approximately 1340 – 1650 mV
 - ✓ Optionally open and closed loop trimmed using CorePWM output 8 depending on MPM GUI configuration and SF2-DMPM-DB jumper settings.

- Channel A9
 - ✓ SF2 MPM-DB DPOL1
 - ✓ Intersil ZL6105 3.3V nominal DPOL
 - ✓ PMBus address 0x21
- Channel A10
 - ✓ SF2 MPM-DB DPOL2
 - ✓ Intersil ZL6105 3.3V nominal DPOL
 - ✓ PMBus address 0x20
- Channel A11
 - ✓ SF2 MPM-DB DPOL3
 - ✓ Intersil ZL6105 3.3V nominal DPOL
 - ✓ PMBus address 0x22
- Channel A12
 - ✓ SF2 MPM-DB DPOL4
 - ✓ Intersil ZL6105 3.3V nominal DPOL
 - ✓ PMBus address 0x23
- Channel A13
 - ✓ MPM-DC DPOL5
 - ✓ Lineage PDT012A0X 1.5V nominal DPOL
 - ✓ PMBus address 0x10
- Channel A14
 - ✓ MPM-DC DPOL6
 - ✓ Lineage PDT012A0X 1.5V nominal DPOL
 - ✓ PMBus address 0x11
- Channel A15
 - ✓ MPM-DC DPOL7
 - ✓ Lineage PDT012A0X 1.5V nominal DPOL
 - ✓ PMBus address 0x12
- Channel A16
 - ✓ MPM-DC DPOL8
 - ✓ Lineage PDT012A0X 1.5V nominal DPOL
 - ✓ PMBus address 0x13

Note: The MPM DPOL channels are programmed over PMBus, switched on/off using the DPOL discrete enable digital input and monitored using the DPOL Power Good/PG (or equivalent) digital output.

Note: It is the MPM demo program (main.c) which implements SW3 control, LED status display and the functionality underlying options selected with SW10.1 to SW10.4. The demo program is provided as a simple illustration of how MPM and the MPM API can be used but it is not part of the MPM reference design "core engine" per se. Refer to the demo program code to see how MPM can be used and deployed and feel free to adapt it to your own specific needs.

With MPM GUI

The MPM reference design demo can also be exercised using the MPM GUI communicating with the MPM target I2C slave via the Devantech/Robot Electronics USB-ISS I²C communications module. Ensure that the USB-ISS hardware and drivers are installed, configured and working as described earlier.

The default I2C slave address for MPM is 100 (decimal). All menu options under **Data > I²C** launch the **MPM I²C Communications** dialog with different default settings. For example **Data > I²C > Monitor Values ON/OFF** gives:

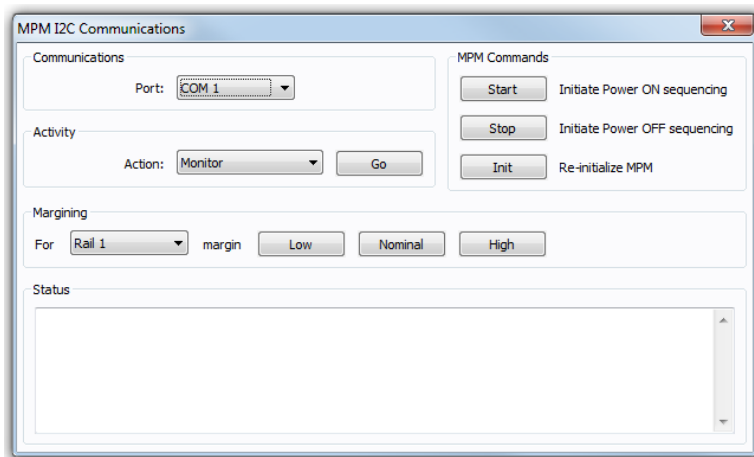


Figure 16 · MPM I²C Communications Dialog

As explained earlier select the appropriate COM port for the USB-ISS device from the **Communications > Port:** drop-down and then click the **Activity > Go** button to check that MPM GUI communication with the USB-ISS works. If not ensure that all hardware, driver and software configurations are correct.

Once MPM GUI communications with the USB-ISS device are working select **Find I²C Address** from the **Activity > Action** drop-down and click the **Activity > Go** button. For the default configuration with MPM I2C slave address 100, you should see:

```
Starting Looking for I2C Address
I2C Speed set to 1MHz
Trying I2C Address of 100
Success
```

If the **Find I²C Address** option is not used to dynamically scan for the MPM slave address then the GUI will by default try to use the I2C address specified under **Misc > Management Interface I²C > I²C Address**.

Once GUI communication with the USB-ISS and the MPM target I2C slave has been established it is possible to use the various other options in the MPM I2C Communications dialog to interact with the target:

- **Activity**
 - ✓ **Test Dongle:** tests communication with the USB-ISS dongle only. The USB-ISS does not need to be connected to the MPM target.
 - ✓ **Find I2C Address:** dynamically scans to search for the MPM I2C slave address.
 - ✓ **Read:** reads the configuration settings from the target via I2C and populates the MPM GUI settings using them.
 - ✓ **Write:** writes the configuration settings in the MPM GUI to the MPM target via I²C. These settings will take effect the next time MPM is reinitialized (in stopped mode) and then (re)started. Note you can **Read**, reconfigure the MPM configuration settings and **Write** in order to change the configuration of the MPM target.
 - ✓ **Monitor:** enters monitoring mode so that the GUI can display live updates of the target state using the Meters and Memory Map views.
 - ✓ **Go/Stop:** click the Go button to run the chosen activity. While running the Go button changes to a Stop button and can be pressed to prematurely terminate the active activity.

- **MPM Commands**
 - ✓ **Start:** initiate power on sequencing which is the same as pressing SW3 when MPM is in stopped mode. No effect if MPM is already started.
 - ✓ **Stop:** initiate power off sequencing which is the same as pressing SW3 when MPM is in started mode. No effect if MPM is already stopped. Note that you can start MPM using the GUI and stop it using SW3 and vice versa if you want.
 - ✓ **Init:** re-initialize MPM – i.e. reload the latest MPM configuration data from ENV. Only works when MPM is in stopped mode – no effect otherwise.
- **Margining**
 - ✓ **For:** select the channel/rail to which the following margining command will be applied.
 - ✓ **Margin:** the **Low**, **Nominal** and **High** buttons cause the selected channel/rail to margin to its low, nominal or high set voltage. No effect if MPM is not started or the relevant channel is not available.
- **Status**
 - ✓ Displays information about the progress and status of the most recent activity, command or margining operation.

MPM Design Files

SoftConsole Firmware Project

A SoftConsole workspace containing the reference design firmware and demonstration program application code is included in

C:\Microsemi\SF2_MPM_RefDesign_v6.1\design_files\SoftConsole_workspace\SF2_MPM_RefDesign.

To access this, do the following:

1. Make a backup or work copy of the original SoftConsole workspace
2. Run SoftConsole v3.4
3. Choose **File > Switch Workspace > Other...**
4. Browse to the reference design SoftConsole workspace folder containing the .metadata folder
5. Click OK.

SoftConsole reopens using the reference design firmware workspace, which contains a single *mpm_reference_design* project implementing the MPM driver and demo program.

The *main.c* file contains the implementation of the reference design demonstration program, which interacts with the MPM hardware design through the MPM driver bundled in the project's *mpm* folder.

The *mpm.h* and *mpm.c* files are also in the *mpm* folder.

Review *main.c* to see how the MPM driver is used by the demonstration program. Note that the demonstration program also includes other firmware cores used directly by the application code or MPM driver.

The *mpm/mpm.h* file describes the public interface to the MPM driver.

The *mpm/mpm.c* and various files in *mpm* implement the actual MPM driver functionality that interfaces with the underlying MPM hardware design.

Note that the project settings include a number of manifest constant/symbol definitions under **Properties > C/C++ Build > Settings > Tool Settings > GNU C Compiler > Symbols** which configure and tune the way that the firmware operates.

The MPM driver public interface is described by *mpm/mpm.h*:

```
/* void mpm_task_init()
 *
 * Initializes the tasks, queues, semaphores etc for the MPM engine
 * with FreeRTOS. This call leaves the MPM tasks suspended and should
 * be called before enabling the task scheduler.
 *
 * This should be called before vTaskStartScheduler() and before mpm_init()
 * which will usually be called in the main application task.
 */

void mpm_task_init(void);

/* void mpm_init()
 *
 * Initializes the MPM engine "driver" and must be called before any other
 * MPM driver methods below.
 */
void mpm_init();

/* void mpm_start()
 *
 * Starts the MPM engine.
 *
 * If MPM is not in state mpm_is_stopped then this method does nothing and
 * just returns immediately otherwise if MPM is in state mpm_is_stopped
```

```
* then this method:
*
* - puts MPM into state mpm_is_starting
* - starts channel threshold monitoring
* - starts channel open loop trimming where applicable
* - initiates power on sequencing
* - waits until power on sequencing has successfully completed
* - starts channel closed loop trimming where applicable
* - puts MPM into state mpm_is_running
* - channel threshold monitoring remains active
*/
void mpm_start();

/* void mpm_stop()
*
* Stops the MPM engine
*
* If MPM is not in state mpm_is_running then this method does nothing and
* just returns immediately otherwise if MPM is in state mpm_is_running
* then this method:
*
* - puts MPM into state mpm_is_stopping
* - stops closed loop trimming where applicable
* - initiates power down sequencing
* - waits until power down sequencing has successfully completed
* - stops open loop trimming where applicable
* - stops channel threshold monitoring
* - puts MPM into state mpm_is_stopped
*/
void mpm_stop();

/* void mpm_dev_show_state(void)
*
* Reflect the current state of MPM on the Dev Kit LEDs
*/
void mpm_dev_show_state(void);

void mpm_dev_debug_led_5(int state);
void mpm_dev_debug_led_6(int state);
void mpm_dev_debug_led_7(int state);
void mpm_dev_debug_led_8(int state);

/* uint32_t mpm_dev_get_inputs(void)
*
* return the input state of the internal mode switching and
* debug display GPIO.
*/

uint32_t mpm_dev_get_inputs(void);

/* mpm_state_t mpm_get_state()
*
* Returns the state that the MPM engine is currently operating in.
*/
mpm_state_t mpm_get_state();

/* mpm_channel_state_t mpm_get_channel_state(mpm_channel_number_t)
*
*/
```

```

    * Returns the current threshold relative state for the channel
    * identified by the channel number passed in. If the channel number
    * passed in does not refer to a valid channel then
    * mpm_channel_is_off is returned.
    */
mpm_channel_state_t mpm_get_channel_state(mpm_channel_number_t);

/* int32_t mpm_get_channel_voltage_mv(mpm_channel_number_t)
 *
 * Returns the current channel voltage in mV for the channel identified
 * by the channel number passed in. If the channel number passed in does
 * not refer to a valid channel then 0mV is returned.
 *
 * If MPM is in state mpm_is_stopped then mpm_channel_is_off is returned
 * for all channels.
 */
int32_t mpm_get_channel_voltage_mv(mpm_channel_number_t);

/* bool mpm_is_valid_channel(mpm_channel_number_t)
 *
 * Returns true if the channel identified by the channel number passed in is
 * a valid channel recognized by the MPM engine and false otherwise.
 *
 * For a channel to be valid it must meet the following criteria:
 *
 * - Signal name in ACE configuration must be "MPM_Channel_<n>..." where
 *   <n> is a unique identification number between 1 and min(64,
 *   MPM_MAX_NUMBER_OF_CHANNELS) and "..." can be any other text.
 * - ACE configuration for this channel must include one "UNDER" threshold
 *   flag named "DOWN" and one "OVER" threshold flag named "UP" with any
 *   valid voltage level (the threshold voltage levels are dynamically
 *   adjusted at runtime by the MPM engine)
 * - None of the threshold/hysteresis/nominal voltage values configured
 *   through the MPM GUI is out of range of the underlying ACE (ABPS or
 *   direct analog input) voltage channel.
 */
bool mpm_is_valid_channel(mpm_channel_number_t);

/* uint32_t mpm_get_digital_inputs(int bank)
 * uint32_t mpm_get_digital_outputs(int bank)
 * uint32_t mpm_get_regulator_enable_outputs(int bank)
 *
 * Returns a 32 bit bitmask [31:0] representing the current state of the
 * relevant digital I/Os. In 32 channel builds bank is ignored but in 64
 * channel builds, bank == 0 is the first 32 channels and bank == 1 is the
 * second 32 channels.
 */
uint32_t mpm_get_digital_inputs(int bank);
uint32_t mpm_get_digital_outputs(int bank);
uint32_t mpm_get_regulator_enable_outputs(int bank);

```

Libero SoC Hardware Project

A Libero SoC v11.0 project implementing the MPM hardware is included in
 C:\Microsemi\SF2_MPM_RefDesign_v6.1\design_files\Libero_project\SmartFusion2_MP
 M_RefDesign.

To use this:

1. Make a backup or work copy of the original Libero SoC project bundled with the package.

2. Run Libero SoC v11.0
 3. Browse to the Libero IDE project folder.
C:\Microsemi\SF2_MPM_RefDesign_v6.1\design_files\Libero_project\SmartFusion2_MPM_RefDesign
 4. Select SF2_MPM_RefDesign.prjx and Open.
 5. If you receive any warnings about missing IP cores, make sure to download them from the repository to the vault using the Libero SoC Catalog.
 6. The design comprises a top-level SmartDesign that instantiates the SmartFusion2 MSS and several fabric-based peripherals
- SmartFusion2 MSS resources used
 - ✓ MSS GPIOs used for interfacing to DPOL PG (Power Good) inputs.
 - ✓ 3 x MSS eNVM data storage client placeholders for MPM firmware, MPM configuration data and MPM logging
 - ✓ 2 x MSS I2Cs for PMBus(I2C_0) and MPM I2C slave interfacing (I2C_1)
 - ✓ Clock, reset and MSS/FPGA interrupt and AMBA configuration.
 - ✓ RTC for log record time stamping.
 - Fabric-based logic
 - ✓ CoreAPB3 for interfacing MSS to fabric DirectCore peripherals.
 - ✓ 2 x CoreGPIO (MPM_GPIO_Digital_IOs + MPM_GPIO_Digital_IOs_II) implementing up to 64 general digital inputs and 64 flag digital outputs. These connect to SW1,2,4,5 push-buttons, SW10 and the SF2-DMPM-DB LEDs
 - ✓ 2 x CoreGPIO (MPM_GPIO_Regulator_Enables + MPM_GPIO_Regulator_Enables_II) implementing up to 64 APOL/DPOL regulators enable digital outputs.
 - ✓ CorePWM (MPM_PWM_Trimming_Outputs) implementing up to 16 APOL channel trimming PWM DAC outputs.
 - ✓ CoreI2C for MPM to DPOL PMBus connectivity – not supported in initial release.
 - ✓ CoreGPIO (MPM_GPIO_Mode) for mode selection to allow mapping of some POL enables to upper 32 regulator enables, Dev Kit LEDs and SW3.
 - ✓ CoreInterrupt (MPM_CoreInterrupt) for routing I²C interrupts to fabric int.
 - ✓ 8 x Multiplexors to switch regulator enables between low and high 32 bit blocks.
 - ✓ BIBUFs for PMBus bidirectional signal support.

Note: In the reference design, some but not all output flag digital outputs are connected to LEDs, some but not all MPM digital inputs are connected to switches and some but not all regulator enable digital outputs are connected to regulator enables.

APOL Channels

APOL channels are assigned to ADC inputs on a first come first served basis so it is important that the order of APOLs in the configuration matches the physical order on the SF2-DMPM-DB.

DPOL Channels

MPM DPOL channels require the following:

1. The DPOL for the channel must be connected to the MPM PMBus
2. Each DPOL on the MPM PMBus must obviously have a unique I²C/PMBus slave address.
3. As with APOL channels the enable/switching input to the DPOL must be connected to the relevant MPM_GPIO_Regulator_Enable output (Channel <n> enable must be connected to MPM_GPIO_Regulator_Enable:GPIO_OUTS [n-1] where 1 ≤ n ≤ min(64, MPM_MAX_NUMBER_OF_CHANNELS). Note that in this release DPOLs are switched only using the discrete enable input and not using the PMBus OPERATION (0x01) command.
4. The Power Good (PG) DPOL output signal must be connected to a suitable MPM input and the MPM firmware modified to handle this. Refer to how the SF2-DMPM-DB Channels 9-16 (DPOL1-8) are connected in the reference design Libero project.

MPM GUI Overview

This section briefly summarizes the layout of the MPM GUI.

Property Page Tabs

Built-in Help

The main display of the Mixed Signal Power Manager (MPM) has four property page tabs:

- Power
- Outputs [1-32]
- Outputs [33-64]
- Miscellaneous

Numerous configuration settings can be set from these tabs. A built-in help panel can be shown using the **Help > Help** menu option.

When the title of any of the groups of configuration parameters is clicked, the background to that group of parameters changes to yellow and the text in the help panel updates to describe those options.

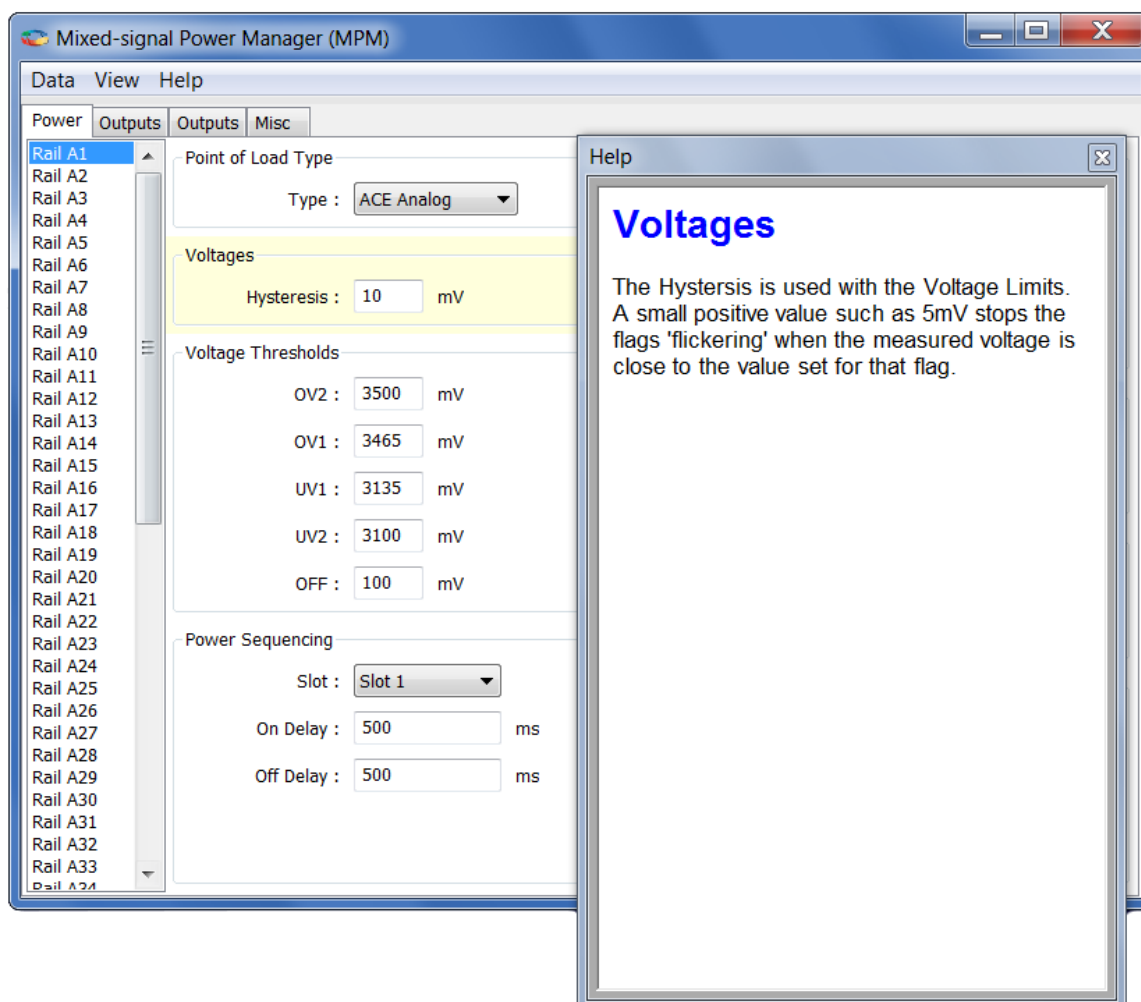


Figure 17 · Built-In Help Window

Power Property Page Tab

The Power tab is used to set parameters for each voltage rail including:

- Channel/point of load type (ACE Analog, Generic Digital, Zilker Labs, and Lineage)
- Hysteresis, nominal and threshold voltages
- Power sequencing slot and on/off delays
- Trimming/margining control

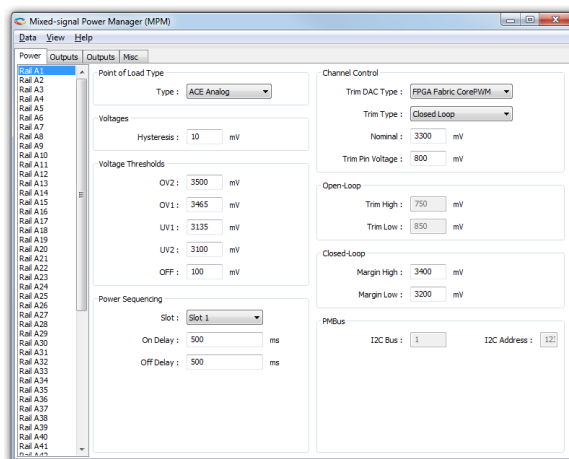


Figure 18 · Power Property Page Tab Window

Outputs Property Page Tabs

The Outputs tabs are used to define the conditions for and polarity of up to 64 digital outputs based on the states of various MPM channels. There are two such tabs, one for outputs 1 to 32 and one for outputs 33 to 64. The tabs support configuration of if/how each digital output is combined with a corresponding digital input, when the output is changed, and whether and how the output is logged.

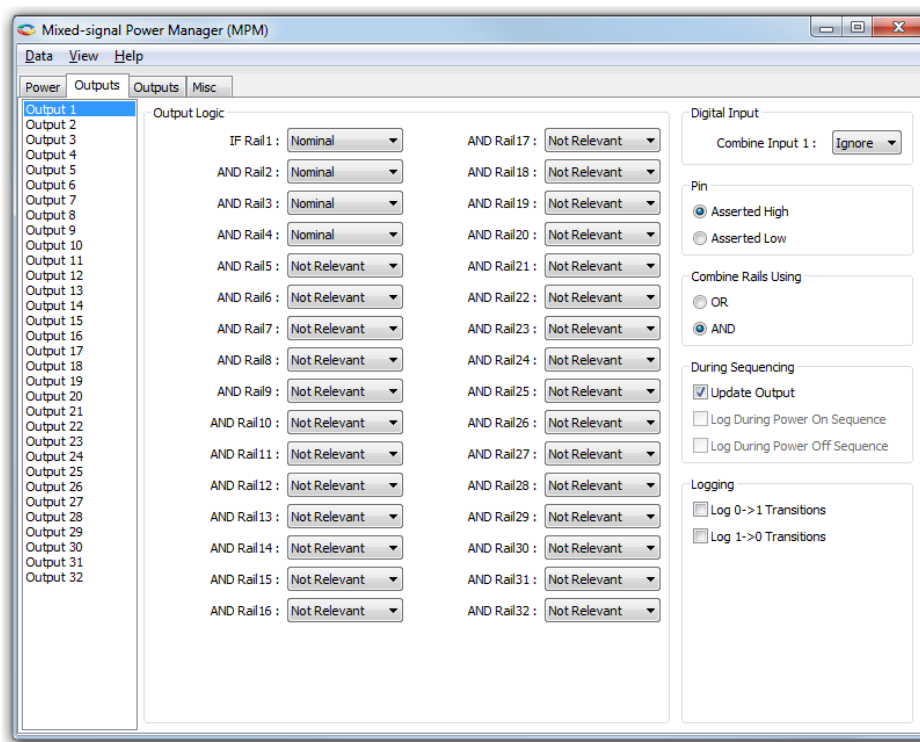


Figure 19 · Outputs Property Page Tab Window

Miscellaneous Property Page Tab

The Miscellaneous tab is used to configure various other aspects of MPM including:

- Power on sequence slot timeout and failure action
- Power off sequence direction
- MPM I²C slave interface address

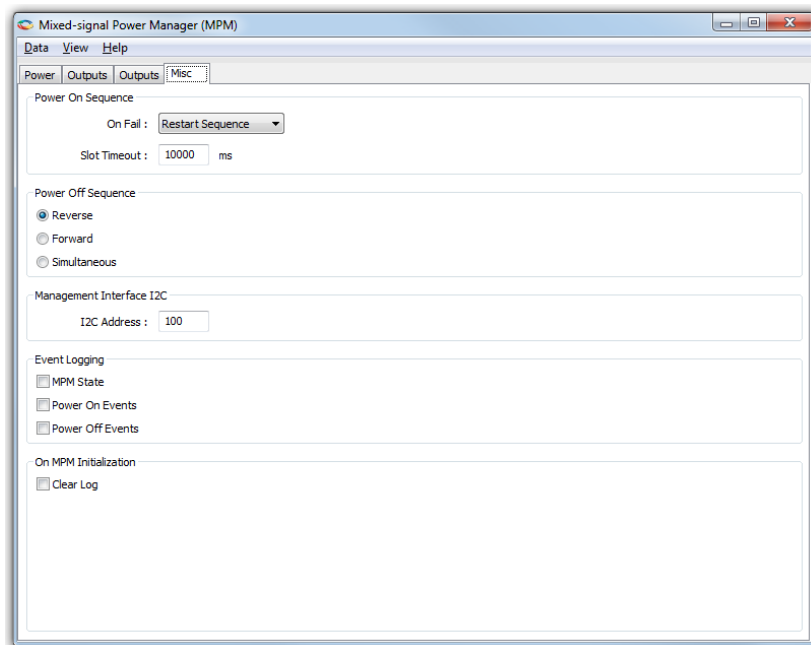


Figure 20 · Miscellaneous Property Page Tab Window

Menu Options

Figure 9 shows what activities can be initiated from the MPM GUI menu bar

- The Data sub menu has activities relating to transferring data to and from the MPM and to and from files.
- The View sub menu shows or hides additional views of the configuration data.
- The Help sub menu opens help on the application and gives information about the software version

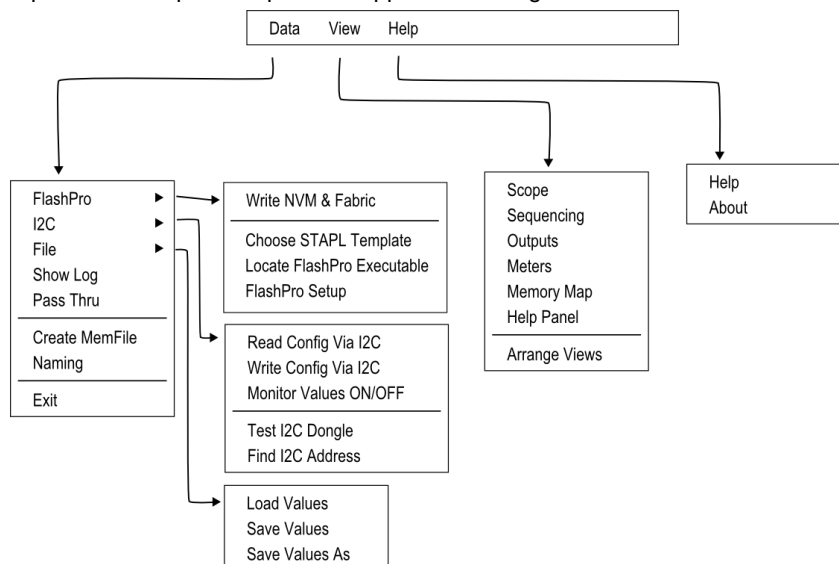


Figure 21 · Menu Options

I²C Communications

The MPM I²C Communications dialog is for communication from the GUI to MPM using I²C via the Devantech/Robot Electronics USB-ISS communications module/dongle. In the **Activity** box choosing an activity and pressing **Go** will start that activity. A log of what happens will be displayed in the Status view

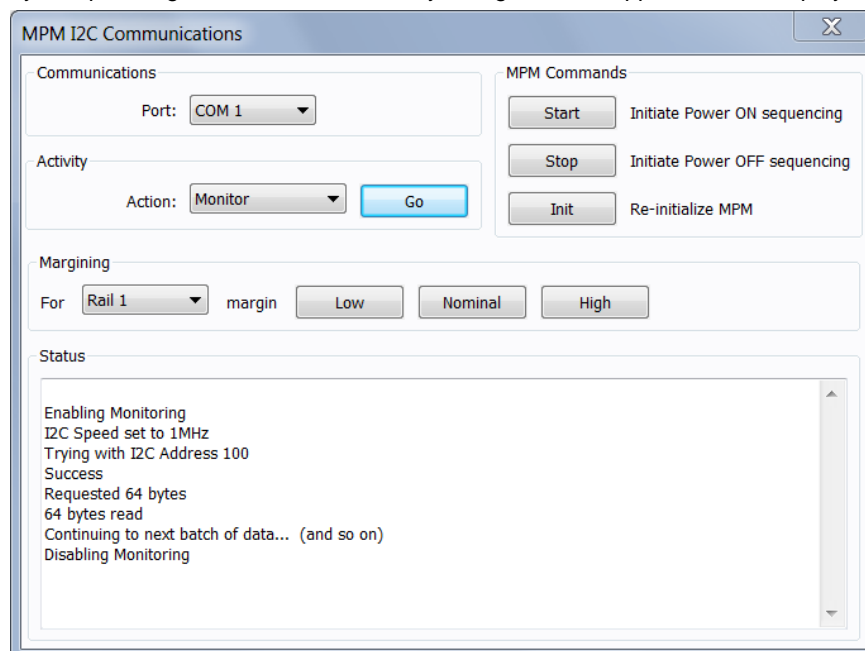


Figure 22 · I²C Communication

Table 4 · Status

Activity	Description
Test Dongle	Tests communications between the PC and the dongle only. The dongle does not need to be connected to the Mixed Signal Power Manager.
Read	Reads the entire MPM configuration. The configuration values in the main tabs are updated, as are the values in the GUIs copy of the memory map.
Write	Writes the entire MPM configuration using values in the main tab.
Monitor	Repeatedly read the live analog values at address 0x1900 to 0x197F. This activity continues until "Stop" is pressed.
Find I²C Address	This scans the I ² C bus looking for an MPM device. The I ² C address in the Misc tab is the first address the GUI tries.

The text on the **Go** button changes to **Stop** when there is I²C activity. The activity can be stopped by clicking the **Stop** button. The dialog also allows for single actions – to start power ON or power OFF sequencing, to reinitialize MPM and to perform margining on a particular rail.

Log Window

The log window shows the contents of the event log. Clicking the **Fetch** button retrieves the most recent 128 records from the log. Each row in the log window shows the status of one log record, the timestamp for it and associated data.

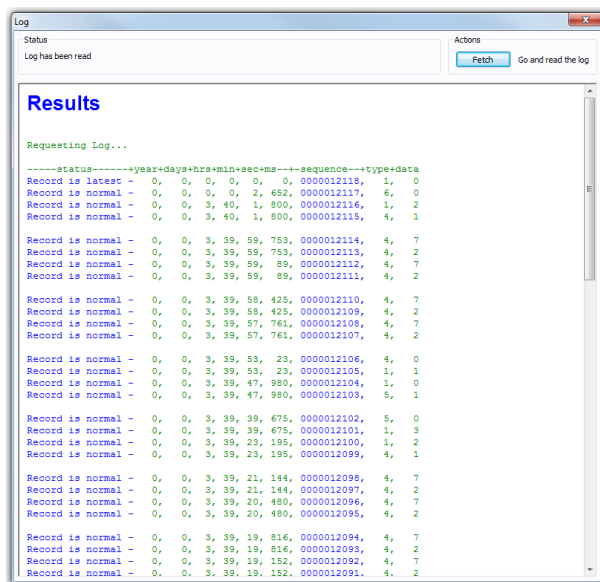


Figure 23 · Log Window

Views

Sequencing View

This gives a visual display of the configured power on and power off sequencing. Rails in the same slot are shown stacked above each other. The delay time for each rail within its slot is shown both numerically and as a bar below the rail's name. Clicking on a rail will open the main configuration dialog on the appropriate page.

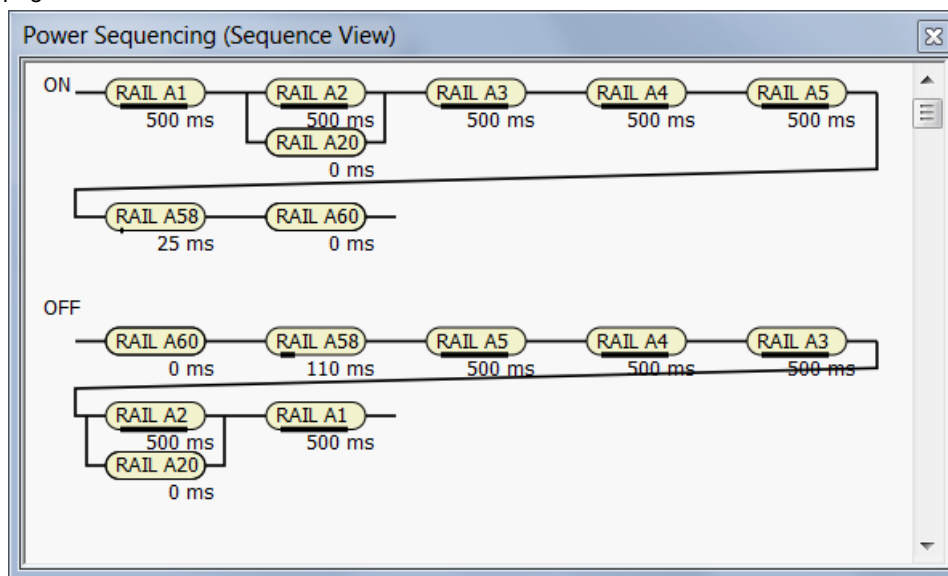


Figure 24 · Sequencing View

Scope View

This gives an alternative schematic display of the power on and off sequencing as a graph. It is meant to be used as a visual guideline only, displaying only relative time frames of power-up and power-down as a result of sequencing requirements entered on the **Power** tab. Rise and fall times are not accurate and voltage levels are not taken into account.

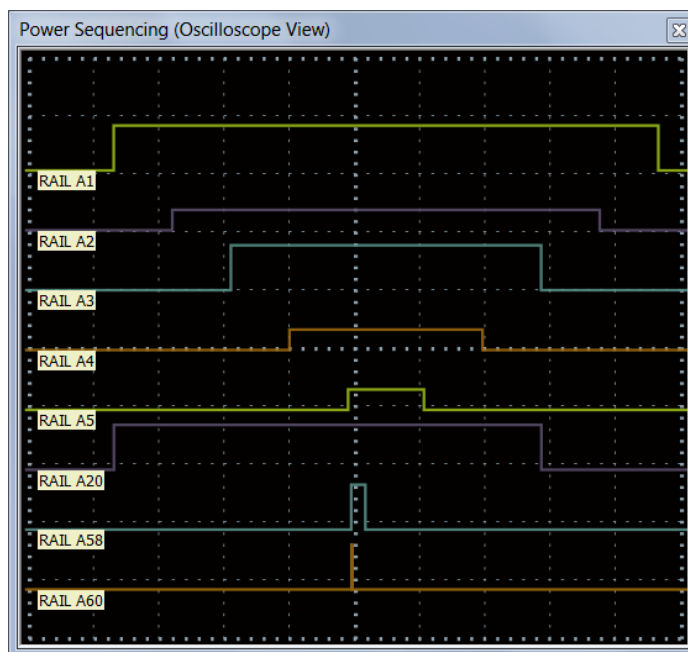


Figure 25 · Scope View

Output View

The Outputs view gives a schematic view of the output logic selected on the current outputs tab.

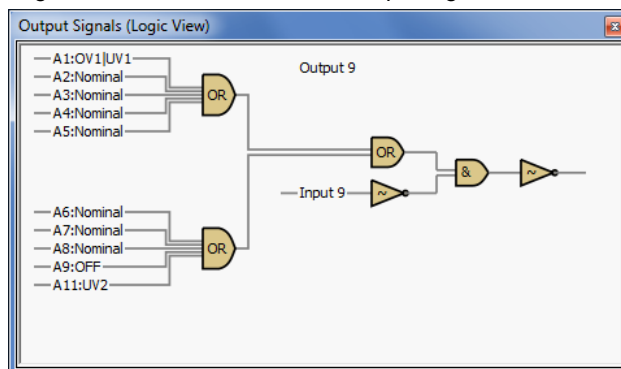


Figure 26 · Output View

Meters View

The meters view shows the values of the first twelve analog values read back by the MPM firmware at address 0x1900. For the meters to reflect live values the USB-ISS module must be connected to the SmartFusion2 board and monitoring must be active on the MPM I²C Communications Dialog. The meters ranges automatically adjust to your voltage conditions. Values up to 50 volts are shown. The actual value is shown numerically in Figure 15.

At the top of the Meters view are two rows of simulated LEDs. These indicate:

- (a) The state of each digital input that is used in generating the corresponding flag.
- (b) The current state of the generated flag.

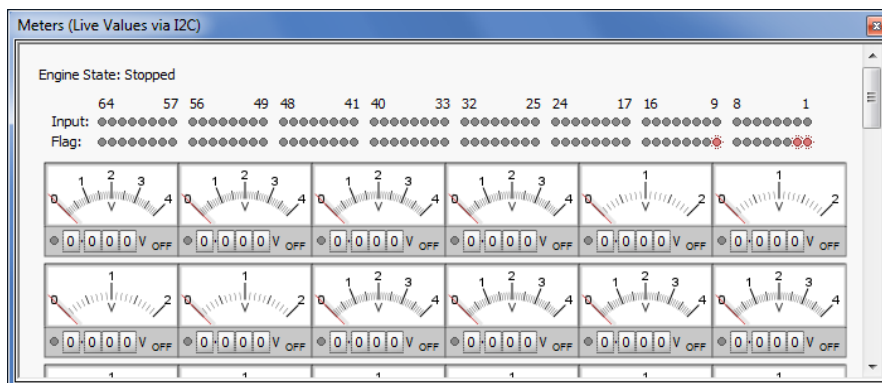


Figure 27 · Meters View

The details shown for each meter are annotated in Figure 16 below.

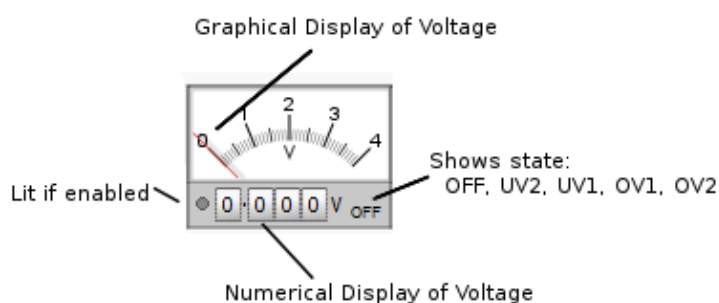


Figure 28 · Details of Meter information

Memory Map

The Memory Map view shows a hexadecimal representation of the configuration data. Values with dark yellow backgrounds are values that are relevant to configuration. Values with light yellow backgrounds are not configuration values or are unused. When the help panel is open, clicking on a value gives more information about that value in the help panel.

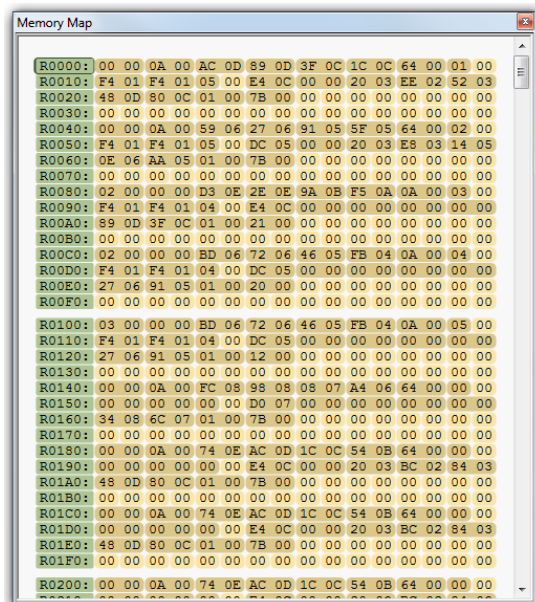


Figure 29 · Memory Map View

Pass Thru

Pass Thru is a facility that allows data to be sent from the GUI via the DMPM I2C through to the I2C for the DPOLs. The GUI currently supports a subset of configurable parameters of the Intersil DPOLs.

The Pass Thru Dialog is invoked from the Data Menu.

Clicking the **File** button opens a file browser to select a file that can be passed on through to a particular DPOL.

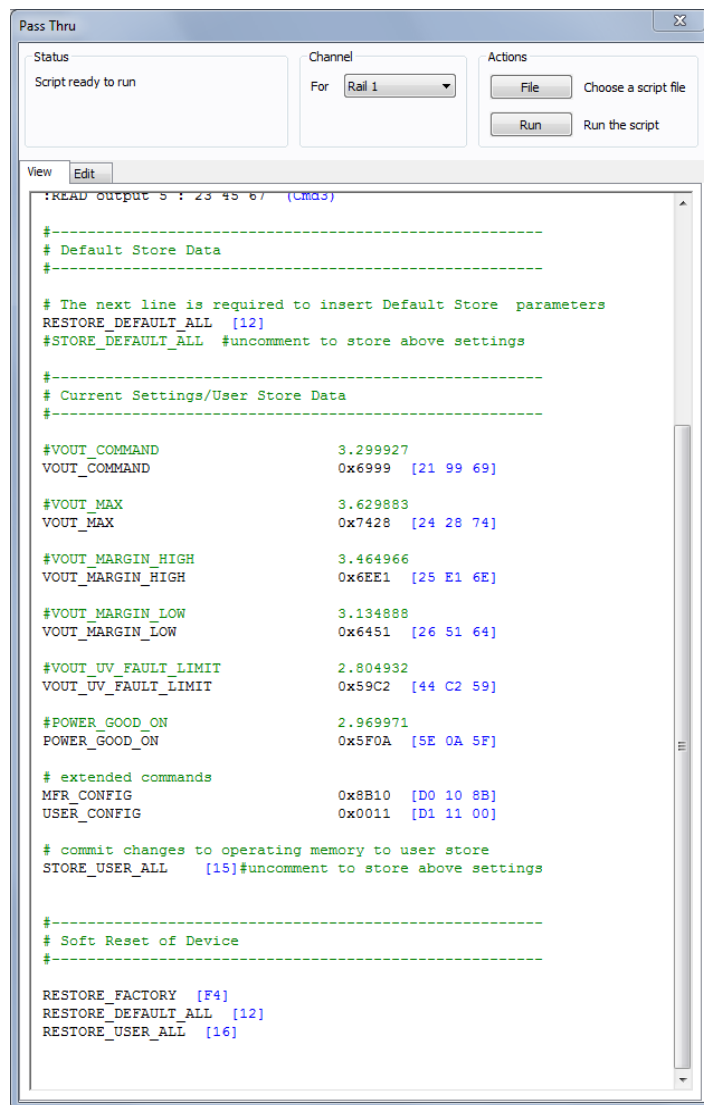


Figure 30 · Pass Thru Window

The file is parsed and the data to be sent is shown in blue. Invalid entries or entries which the software cannot parse are shown in red.

“Run” runs the configuration script sending the values to the target DPOL. The target DPOL can be changed by choosing a different rail in the drop-down in the 'channel' box.

Naming

The naming feature allows the default names of Rail A1, Rail A2... and Output 1, Output 2... to be changed.

The Naming Dialog can be invoked from the Data Menu.

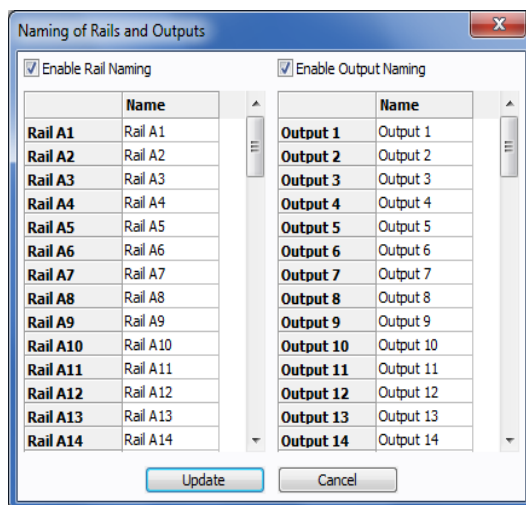


Figure 31 · Naming Dialog Showing Defaults

Each name is editable. After changing the names and clicking update the new names will be shown in the GUI, provided that the tick boxes to enable naming are ticked.

The names can also be saved and loaded to the configuration file and NVM along with the configuration.

There are some restrictions on naming.

- Names should be at most 14 characters long as no more than that will be saved/loaded from NVM. It's OK to use most ASCII characters including the characters 0..9 a..z and A..Z. The " character will be changed to a #. If the names include accented characters the entire renaming will revert to defaults.
- The outputs diagram does not update to take account of new naming. Also the space in diagrams for names is not increased if long names are used.

An example of the Naming Dialog after some names have been changed is shown below:

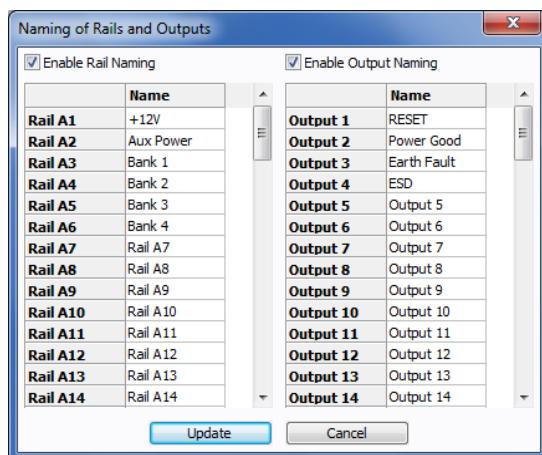


Figure 32 · Naming Dialog After Renaming

MPM for SmartFusion2 Reference Design Flow

Purpose

The purpose of this document is to provide the user with a detailed description of the flow of the SmartFusion2 MPM Reference Design: updating/compiling firmware, modifying the top-level FPGA design, synthesis/layout, updating NVM, generating STAPL/PDB programming file, and integrating the programming file into the GUI.

Tools/Resources Used

The following Microsemi resources were used to create the MPM Reference Design.

- Libero SoC v11(11.0.0.23)
- SoftConsole v3.4 (3.4.0.5-M20130329-1700)
- IP cores – MSS and DirectCores
 - ✓ SmartFusion2 MSS v1.0.100
 - ✓ CoreAPB3 v4.0.8
 - ✓ CoreGPIO v3.0.120
 - ✓ CoreI2C v7.0.102
 - ✓ CoreInterrupt v1.1.101
 - ✓ CorePWM v4.1.106
- Firmware drivers
 - ✓ SmartFusion2 MSS drivers
 - ✓ SmartFusion2 CMSIS-PAL v2.1.101 – With modifications to the startup_m2sxxx.s file to support execution of NVM related code from RAM when the rest of the code is executing from NVM.
 - ✓ SmartFusion2 MSS GPIO driver v2.0.101
 - ✓ SmartFusion2 MSS I2C driver v2.0.100 - This has been modified from the stock driver to integrate with Free RTOS when using a slave write handler
 - ✓ SmartFusion2 MSS NVM driver v2.0.103
 - ✓ SmartFusion2 MSS PDMA driver 2.0.102
 - ✓ SmartFusion2 MSS RTC driver 2.0.101
 - ✓ SmartFusion2 MSS SPI driver 2.0.103.
 - ✓ SmartFusion2 MSS Timer driver v2.0.101
 - ✓ SmartFusion2 MSS Watchdog driver v2.0.102
- DirectCore drivers
 - ✓ Hardware Abstraction Layer (HAL) v2.2.102
 - ✓ CoreGPIO driver v3.0.101
 - ✓ CoreI2C driver v2.0.103 (currently not used)
 - ✓ CorePWM driver v2.1.107
- Other drivers
 - ✓ Unified I2C driver abstraction layer (DAL) v1.0.101
 - ✓ PMBus driver v1.1.102 - This has been modified to fix a bug in the linked list handling

- ✓ SPI Flash driver - This has been modified from the version supplied with the SPI sample code to use dual channel DMA transfer.
- ✓ SPI ADS7953 driver – new SPI based driver for Texas Instruments ADS7953 16 channel 12bit ADC..
- Non Microsemi software
 - ✓ FreeRTOS version 7.1.1

Building the SoftConsole Firmware Image

The SmartFusion2 MPM reference design release contains a SoftConsole v3.4 project, which may be modified by the user, located in the SoftConsole workspace at

C:\Microsemi\SF2_MPM_RefDesign_v6.1\design_files\SoftConsole_workspace\SF2_MPM_RefDesign

For information on the MPM driver and how to use/modify it, refer to the "SoftConsole Firmware Project" section.

Once the project has been compiled and tested, the user can build a final version for deployment in NVM by **Project > Build Configurations > Set Active > Release**. The Release target is configured to link using the mpm/ mpm-production-run-from-envm.ld linker script. This linker script creates a firmware image suitable for storage in eNVM which has been mirrored to 0x00000000. Once stored in eNVM mirrored to 0x00000000 the program will start executing directly from eNVM. The Figure 18 shows a screenshot of this option

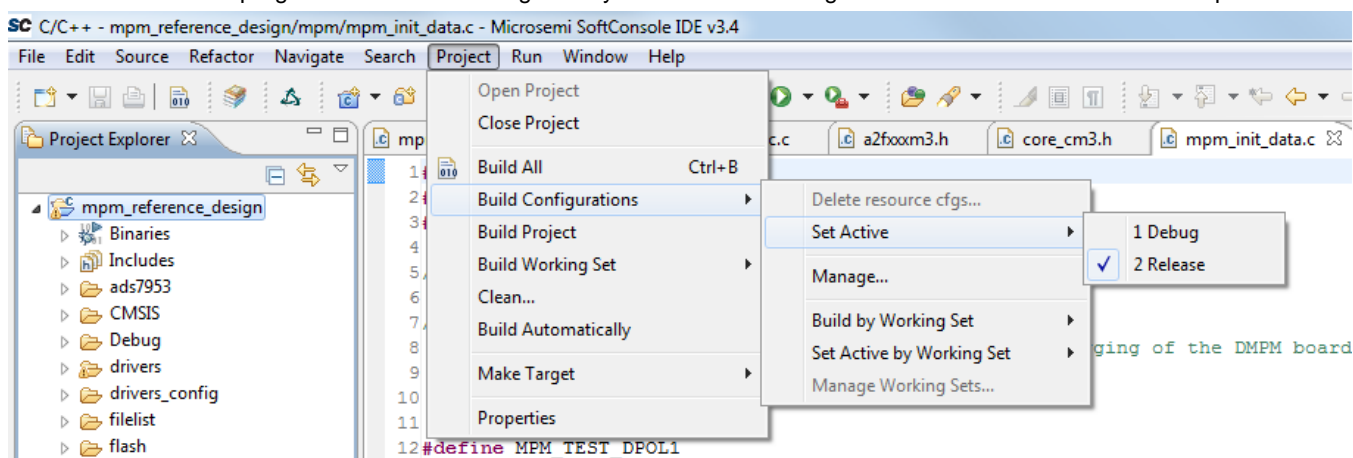


Figure 33 · Setting Active Build to Release in SoftConsole

At this point you may build the project. One of the steps of the MPM project build is to generate a *.hex file containing the firmware:

```
arm-none-eabi-objcopy -O ihex mpm_reference_design "mpm_reference_design.hex"
```

mpm_reference_design.hex will later be used to place the firmware into NVM. Refer to the "The SmartFusion2 MPM Reference design defines three MPM specific eNVM data storage clients as illustrated in Figure 19 for this step.

MSS ENVM Data Storage Clients

The SmartFusion2 MPM Reference design defines three MPM specific eNVN data storage clients as illustrated in Figure 19.

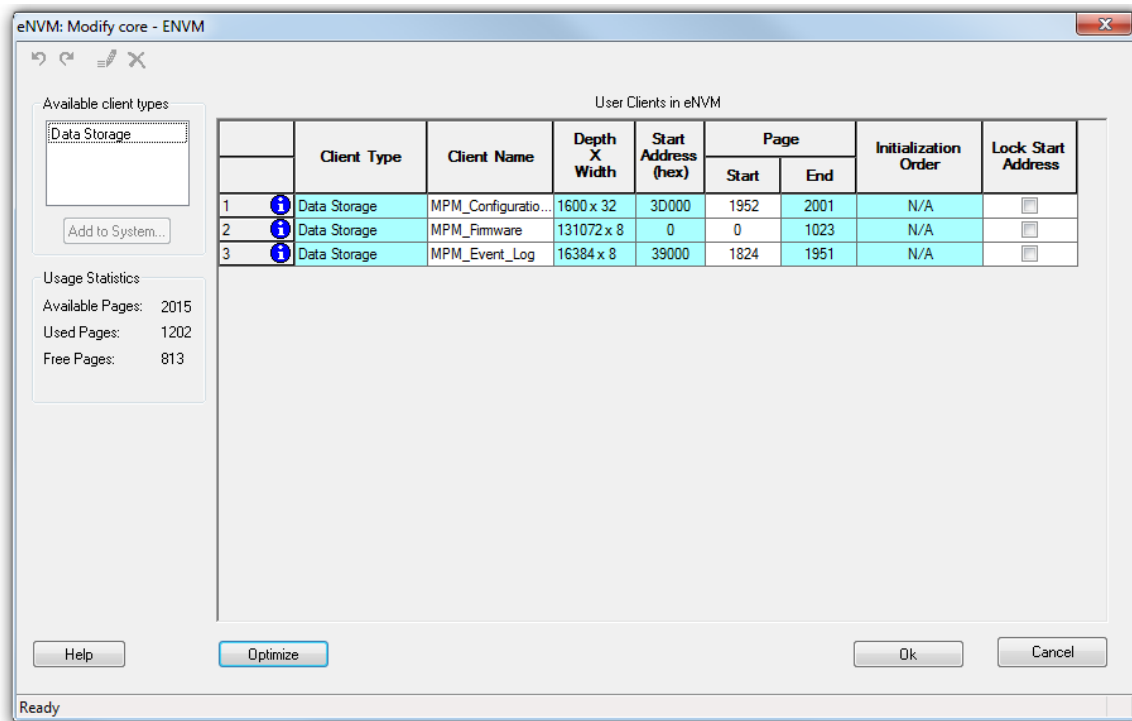


Figure 34 · MSS eNVN Configuration GUI

1. MPM_Configuration_Data

This is where the MPM configuration data is stored in NVN. By default, with the MPM firmware driver unchanged it has a base address of 0x3D000 from the start of eNVN (0x60000000) and consists of 1600 x 32-bit words or 6400 bytes.

The MPM_Configuration_Data ENVN data storage client configuration is as follows in the reference design:

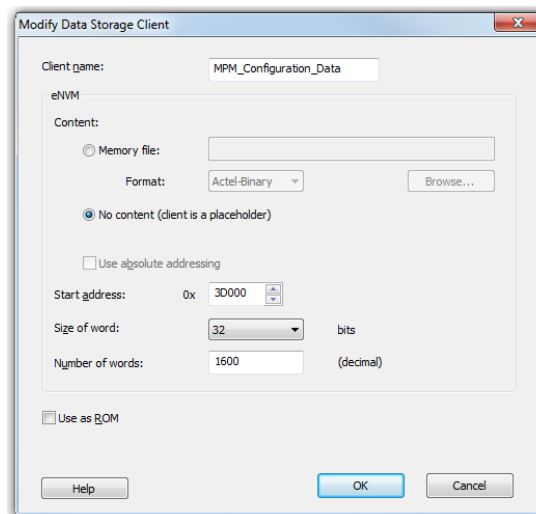


Figure 35 · Data Storage Client for MPM Configuration Data

Note that if the base address of the MPM_Configuration_Data eNVN data storage block needs to change (e.g. because the SmartFusion2 MSS etc. configuration data also stored in eNVN extends into this region)

then the address must be changed consistently in both the eNVM data storage client configurator and by changing this manifest constant in mpm.c:

```
/* Base address of MPM configuration data */
#ifndef MPM_CONFIGURATION_DATA_BASE_ADDRESS
#define MPM_CONFIGURATION_DATA_BASE_ADDRESS (0x6003D000)
#endif
```

2. MPM_Firmware

This is the region in NVM that stores the MPM firmware for execution out of reset. The MPM_Firmware eNVM data storage client configuration is as shown in Figure 21:

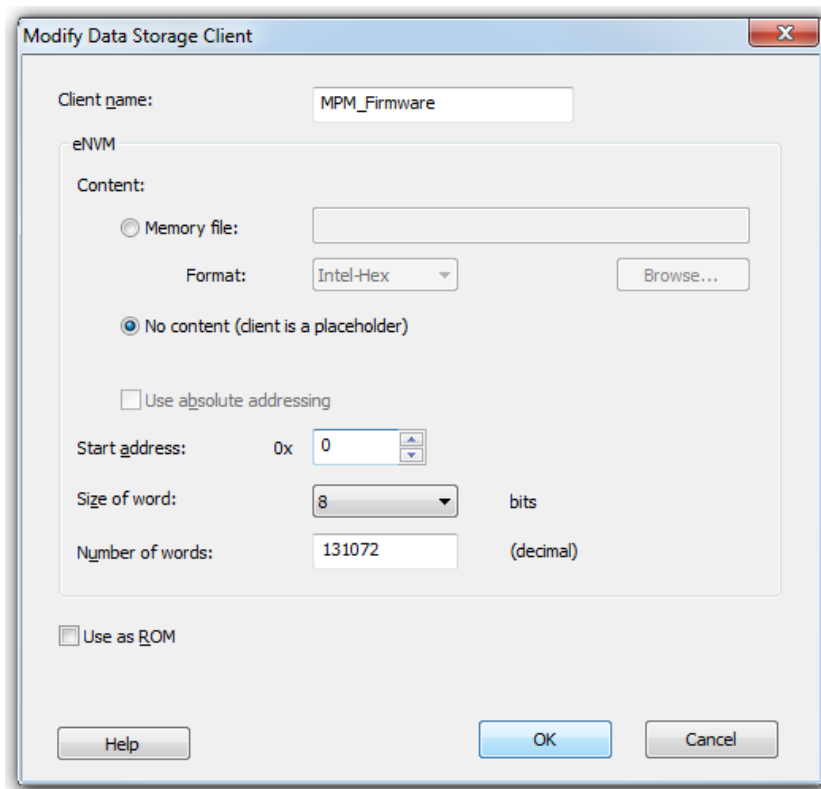


Figure 36 · Data Storage Client for MPM Firmware

To properly populate this region with the MPM firmware, the user should select “Memory file” in the above menu and browse to the Intel-Hex file generated in SoftConsole during the firmware build. For example:

C:\Microsemi\SF_MPM_RefDesign_v5.0\design_files\SoftConsole_workspace\SF_MPM_RefDesign\mpm_reference_design\Release\mpm_reference_design.hex

The **Start address**, **Size of word**, and **Number of words** parameters should be detected from the Hex file and auto-populated.

3. MPM_Event_Logging

This region is needed only if you compile MPM to use a data logging event log in internal NVM rather than the (default) external NVM.

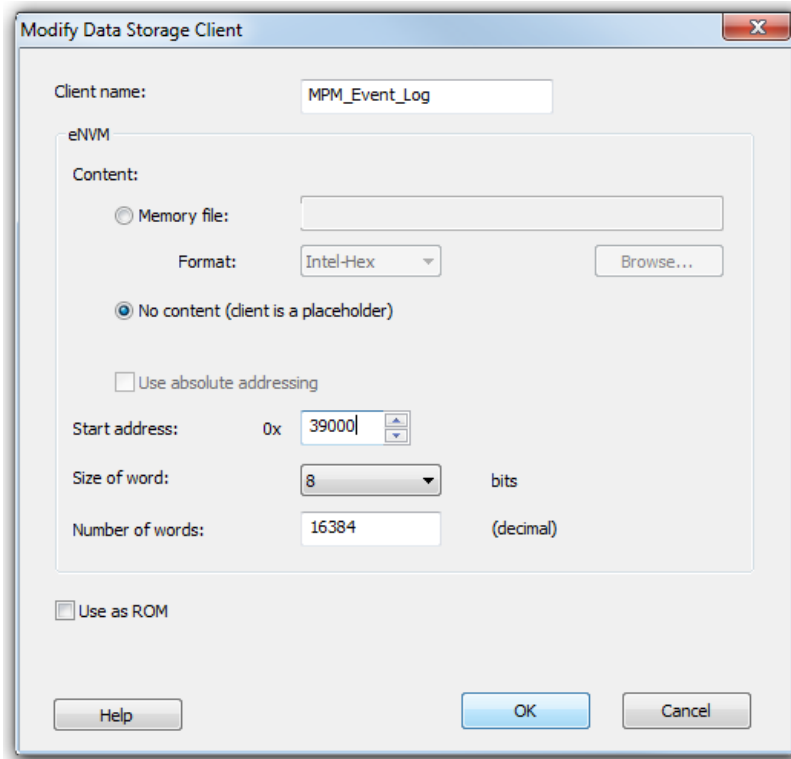


Figure 37 · Data Storage Client for MPM_Event_Logging

Note that, as with MPM_Configuration_Data, the base address and size of this memory region must be consistent in both the eNVM data storage client configurator and the mpm.c file:

```
/* Base address and extent of MPM event logging area in eNVM.
 * These must match the configuration of the MPM_Event_Logging MSS eNVM
 * data storage client in the Libero project */
#ifndef MPM_EVENT_LOG_BASE_ADDRESS
#define MPM_EVENT_LOG_BASE_ADDRESS (0x6003A000)
#endif

#ifndef MPM_EVENT_LOG_LENGTH
#define MPM_EVENT_LOG_LENGTH (0x0004000)
#endif
```

Libero SoC Design Flow

If any changes have been made to the MPM Reference Design Libero project (e.g. changing die, modification of MSS configuration settings, addition of FPGA fabric logic etc.) the project can be run through the normal Libero SoC design flow.

STAPL File Generation and GUI Integration

Once the design has been run through the design flow it is necessary to generate the STAPL file for the design for use with the MPM GUI.

To do this run FlashPro interactively from Libero SoC and export the STAPL file for the PDB file. Once generated this STAPL file can be copied to the MPM GUI

C:\Microsemi\SF2_MPM_RefDesign_v6.1\template folder where it can be selected from the GUI using the **Data > FlashPro > Choose STAPL** Template menu option.

MPM for SmartFusion2 Reference Design I²C

I²C Operation

Overview

SmartFusion2 MPM Reference Design uses a standard 2-wire I²C slave interface, which is by default implemented using the SmartFusion2 microcontroller subsystem (MSS) I2C_1 interface. The host can perform block reads and writes of up to 64 bytes at a time.

The SmartFusion2 MPM Reference Design I²C protocol conforms to the Philips Inter-Integrated Circuit (I²C) v2.1 Specification utilizing 7-bit addressing format at effective 100 kbps and 400 kbps data rates². For more information on I2C, please consult the [Philips/NXP I2C specification document](#).

For more information on how this pertains to the MSS I²C or CoreI²C, refer to the appropriate IP/hardware and firmware driver datasheet/handbook.

Refer to section "I²C Register Map" section for specific configuration register information.

Board

I²C access from an I²C master to the MPM I²C slave is provided via the MSS I2C_1 SCL and SDA pins on the SmartFusion2 DMPM-DB as detailed in Table 4. The MPM GUI can act as an I²C master communicating with the MPM I²C slave via the Devantech/Robot Electronics USB-ISS USB to I²C dongle and the corresponding pins on this dongle are also listed.

SF2-DMPM-DB Rev B boards have a dedicated connector for the Slave I²C interface which the USB-ISS can plug directly into.

Table 5 · Devantech/Robot Electronics USB-ISS Connections

I ² C Signal Board	SCL	SDA	GND
Devantech/Robot Electronics USB-ISS http://www.robot-electronics.co.uk/htm/usb_iss_tech.htm	SCL (I/O 3)	SDA (I/O 4)	0V/Ground
SF2-DMPM-DB Board Rev A	J22P pin 16	J22P pin 32	J3 pin 8
SF2-DMPM-DB Board Rev B	J25 pin 3	J25 pin 2	J25 pin 6

1. *Note that effective data rates may be lower due to clock stretching. Also note that the I²C clock is configurable to other clock rates as well and is dependent on the I²C input clock(s).*

The SF2-DMPM-DB MSS I²C SCL and SDA signals have 1K Ohm pull-up resistors and have an operating voltage of 3.3 V³. For more detailed board specifications, consult the SmartFusion2 Development Kit, SF2-DMPM-DB and Devantech/Robot Electronics USB-ISS documentation.

GUI

Unlike previous releases of MPM the GUI can only program the MPM design and original configuration from the Libero project by calling out to the FlashPro software and passing the relevant "template" STAPL file. It is not possible to include user specified MPM configuration settings in the STAPL file.

Once MPM has initially been programmed to the target, the GUI can connect via the Devantech/Robot Electronics USB-ISS USB to I2C dongle to the MPM I²C slave on MSS I2C_1 in order to reprogram, control, and monitor MPM.

I²C Protocol

This section describes the application-level protocol; that is, the meaning of specific transmitted and received bytes from / to the SmartFusion2 MPM I2C slave interface. For a description of low-level I2C protocol, please refer to the I2C Specification.

Block Write

Using block writes, a host can write up to 64 bytes of data to MPM registers starting at any base address. An overview of the block write protocol is shown in Figure 23.

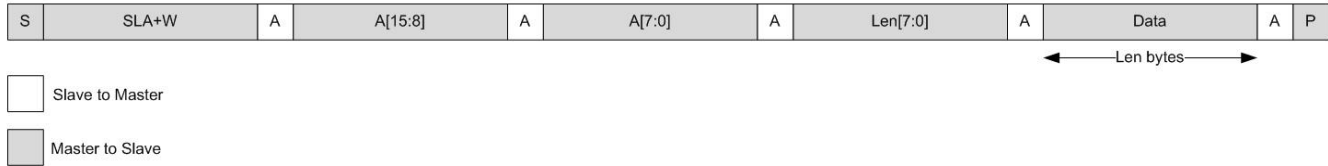


Figure 38 · I²C Block Write

Note:

- 'S' denotes a start condition, 'A' denotes acknowledge (one bit), and 'P' denotes a stop condition.
- SLA+W refers to 8 bits: the 7-bit I2C slave address and 1 bit ('0') to indicate a write transaction.
- A[15:0] refers to the two-byte (16-bit) address of the MPM register being written (see section "I2C Register Map" on page 38 for specific addresses). The first data byte is written to the above address, and all subsequent data is written to incremental addresses.
- Len [7:0] is the number of data bytes that follow; valid range is 1-64.

Block Read

A host can read up to 64 bytes at a time from SmartFusion2 MPM registers by issuing a block read, as shown in Figure 24.

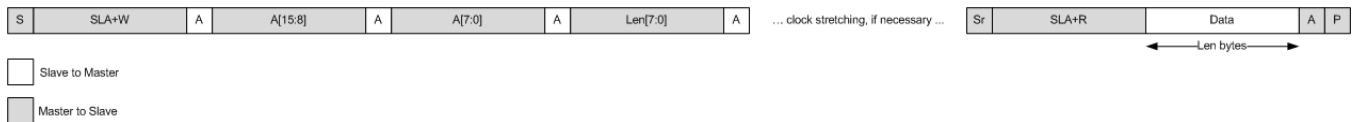


Figure 39 · I²C Block Read

3. When using the Devantech/Robot Electronics USB-ISS make sure to remove the Power Link jumper for 3.3 V rather than 5 V operations. See the "Connections" section in: www.robot-electronics.co.uk/htm/usb_iss_tech.htm

Note:

- 'S' denotes a start condition, 'A' denotes acknowledge (one bit), and 'P' denotes a stop condition, "Sr" denotes a repeated start condition (for change of direction).
- SLA+R refers to 8 bits: the 7-bit I2C slave address and 1 bit ('1') to indicate a read transaction
- A[15:0] refers to the two-byte (16-bit) address of the MPM register being read from (see section "I2C Register Map" on page 38 for specific addresses). The first data byte is read from the above address, and all subsequent data is read from incremental addresses.
- Len [7:0] is the number of data bytes that follow; valid range is 1-64.
- Clock stretching4 is employed by the I²C slave (MPM) for as long as is necessary to fetch the requested data.

Commands

Command transfers, implemented as I²C writes to a fixed address (refer to "I2C Register Map" section on page 38 for specific addresses) are used to issue real-time commands to MPM, as seen in Figure 25.

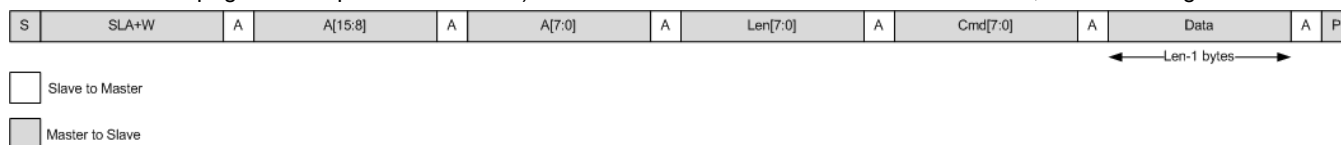


Figure 40 · I²C Command

Note:

- Cmd [7:0] is a command op-code.
- Len n[7:0] is the number of data bytes that follow the command op-code, valid range is 0-64.
- A [15:0] in this case is fixed at I2C_CMD_0 (0x8000), but this can be expanded in firmware to support multiple command addresses.

I²C Status

The I2C_STATUS register described in the section "I2C Register Map" reflects the result of the most recent I²C access.

Types of Access

All I²C access is read and/or write access to the real or pseudo registers defined in the MPM register map. Three types of access are possible:

Three specific types of I²C access are supported:

Read/write access to configuration registers - write access to configuration registers is only allowed when MPM is in the stopped state and each individual write operation is immediately committed to eNVM. Configuration registers changes do not take effect immediately but only when MPM is next reinitialized/restarted.

Read only access to monitoring pseudo registers - monitoring pseudo registers can be read any time no matter what state MPM is in. Reads of rail/channel voltage pseudo registers will map to runtime calls within the MPM engine that retrieve the current rail/channel voltages.

Write access to pseudo command registers - commands are executed by writing a command opcode and any associated data to the relevant I²C command pseudo register. Some commands may only be available in certain MPM states and ignored if MPM is not in the appropriate state. (for example attempting to stop MPM when it has not been started). Command register writes map to runtime calls inside the MPM engine to code that implements the relevant command functionality.

4. For more information on I2C clock stretching, refer to Section 3.9 of the I2C Rev 03 specification.

Size of Transfers

Block accesses (read or write) of up to 64 bytes of data are supported. In this case a read or write of up to 64 bytes starts at the specified MPM register map address and continues to subsequent register addresses. For example a block read of 64 bytes from MPM register address 0x1900 reads the 16 bit signed mV values for 64 of the MPM rails/channels and returns them in order of MPM rail/channel number (1..64)⁵. Individual 8 or 16 bit register reads or writes are also possible. Bear in mind that each individual configuration register write will be committed immediately to eNVM and will take effect next time MPM is reinitialized/restarted.

I²C Register Map

MPM Memory Map Overview

Table 6 · Memory Map Overview

Name	Description	Start Address	End Address	Persistent Storage Space Used
Channel configuration	64 channels x 64 bytes of configuration data each	0x0000	0x0FFF	MPM_Configuration_Data ENVm data storage client. This is a 0x1900 (6400 decimal) byte sized memory block at ENVm offset 0x003D000 which is 0x603D000 in the SmartFusion2 MSS Cortex-M3 memory map.
Output flags configuration	64 flags x 32 bytes of configuration data each	0x1000	0x17FF	
Miscellaneous configuration data	256 bytes of miscellaneous configuration data – only 8 bytes actually used right now	0x1800	0x18FF	
Channel voltage monitoring registers	64 channels x 2 bytes each	0x1900	0x197F	None – these are “pseudo” registers that do not occupy any persistent data storage space.
I ² C status, log status and RTC registers	7 bytes	0x1980	0x1986	
Input, Output, Regulator Enables and MPM state	28 bytes	0x1990	0x19AB	
MPM channel state values	256 bytes	0x19b0	0x1AAF	
I ² C status, log status and RTC registers	7 bytes	0x1980	0x1986	
I2C command pseudo register	MPM I ² C slave I ² C command register	0x8000	0x8001	None – these are “pseudo” registers that do not occupy any persistent data storage space.

5. Contiguous reads may not be possible across all register sub-blocks

Base Address

The MPM register map base address is set in two places which must always be kept in sync:

1. In the SoftConsole MPM firmware project via the `MPM_CONFIGURATION_DATA_BASE_ADDRESS` manifest constant in `mpm.c` which specifies the Cortex-M3 processor eNVM base address of the MPM register map. By default this is set to 0x6003D000 which is offset 0x0003D000 from the start of eNVM at Cortex-M3 processor address 0x60000000.

The default can be overridden by adding the following to the SoftConsole MPM firmware project properties:

`-DMPM_CONFIGURATION_DATA_BASE_ADDRESS=<some-eNVM-address>`

2. In the MPM Libero project's MSS configuration where the "MPM_Register_Map" eNVM data storage client's eNVM offset address is specified. By default this is set to 0x0003D000 which is implicitly with respect to the eNVM base address of 0x60000000 in the Cortex-M3 processor memory map and which obviously matches the firmware project's default setting above.

The MPM GUI writes MPM configuration register data to the target by reading in the template STAPL file, updating the "MPM_Register_Map" eNVM data storage client with the values loaded/entered by the user through GUI and writing a new STAPL file before invoking FlashPro to write this STAPL file to the target. Alternatively the GUI can connect to the MPM I2C slave in order to read/write MPM configuration registers for the purposes of configuration, control and monitoring.

The MPM register base address must be a 128 byte eNVM page aligned address (i.e. lower 7 bits are 0). If the MPM register map base address needs to be changed then both of the above must be changed consistently. If there is a mismatch then MPM will behave unexpectedly – e.g. due to reading garbage configuration data.

In the general case the MPM register map must be capable of being placed anywhere in otherwise unused eNVM address space - the amount of eNVM space available differs by die. Obviously the MPM register map must not clash with other demands on eNVM space (e.g. firmware usually at 0x60000000, MSS configuration data etc.).

Register Addresses

MPM registers are identified using 16 bit addresses allowing for up to 64 Kbytes of register space only a portion of which is used for MPM operation. Configuration registers are "real" registers insofar as they actually occupy SmartFusion2 eNVM memory in the Cortex-M3 processor address space. Other registers may be "pseudo" registers insofar as they have specific register addresses but do not occupy any SmartFusion2 eNVM (or other) memory in the Cortex-M3 processor address space. "Real" registers have their 16 bit MPM register map addresses translated internally into 32 bit Cortex-M3 processor addresses mapping to the relevant underlying eNVM addresses. This involves adding the 16 bit MPM register address to the MPM register map base address described above.

Register Map Overview

The Table 6 summarizes the SF MPM register map. Each register is described in further detail in subsequent sections of this document.

Table 7 · Register Map Overview

Configuration Registers - 'real' registers that use eNVM storage	
Per channel/rail configuration registers - 64 (0x40) bytes for each channel/rail	
Address	Description
0x0000	Rail A1 configuration registers
0x0040	Rail A2 configuration registers
.	
.	

0x0F80	Rail A63 configuration registers
0x0FC0	Rail A64 configuration registers
Per-trigger/flag output configuration registers – 32 (0x20) bytes for each output	
0x1000	Output 1 configuration registers
0x1020	Output 2 configuration registers
.	
.	
0x17C0	Output 63 configuration registers
0x17E0	Output 64 configuration registers
Miscellaneous configuration registers	
Address	Description
0x1800	Miscellaneous configuration registers
I²C monitoring and status registers	
Rail voltage monitoring, RTC registers, I/O states, MPM status and channel state registers – “pseudo” registers that do not occupy persistent eNVM memory storage.	
Address	Description
0x1900..0x197F	Rail voltage monitoring registers
0x1980..0x1985	I ² C status, log status and RTC registers
I²C command registers	
Address	Description
0x8000	MPM I ² C command register(s) – also “pseudo” register(s)

Per-rail Configuration Data

The per-channel/rail configuration data ‘sub-block’ base address for channel n ($1 \leq n \leq 64$) is $(0x0000 + ((n - 1) \times 0x40))$. The Table 7 details the per-channel/rail registers and their offsets with respect to this per-channel/rail configuration data ‘sub-block’ base address.

Table 8 · Pre-rail Configuration Data

Name	Offset	Access	Description	
POL_TYPE	0x0000	R/W	0 = APOL 1= DPOL (generic) 2= Intersil ZL6105 3= Lineage PDT012A0X	Specifies whether the channel is an APOL or a DPOL, and if DPOL, what type
VHL	0x0002	R/W	Threshold hysteresis VH[7:0]	Each pair of high/low 8 bit registers combines to make a 16 bit signed mV value. For positive voltage channels 0mV < OFF < UV2 < UV1 < OV1 < OV2. For negative channels 0mV > OFF > OV2 > OV1 > UV1 > UV2. Allowable ranges depend on nature of underlying ACE channel (e.g. direct analog input 0 to +2560mV or ABPS with a particular prescalar range).
VHH	0x0003	R/W	Threshold hysteresis VH[15:8]	
VOV2L	0x0004	R/W	OV2 threshold VOV2[7:0]	
VOV2H	0x0005	R/W	OV2 threshold VOV2[15:8]	
VOV1L	0x0006	R/W	OV1 threshold VOV1[7:0]	
VOV1H	0x0007	R/W	OV1 threshold VOV1[15:8]	
VUV1L	0x0008	R/W	UV1 threshold VUV1[7:0]	
VUV1H	0x0009	R/W	UV1 threshold VUV1[15:8]	
VUV2L	0x000A	R/W	UV2 threshold VUV2[7:0]	
VUV2H	0x000B	R/W	UV2 threshold VUV2[15:8]	
VOFFL	0x000C	R/W	OFF threshold OFF[7:0]	
VOFFH	0x000D	R/W	OFF threshold OFF[15:8]	
SSLO	0x000E	R/W	Power sequencing slot. Use an extra bit to accommodate 0..64, so Bit [7] reserved, Bits [6:0] power sequence slot number to use. Valid values are: 0: No slot/not sequenced 1-64: Slot in which this channel is sequenced	-
Unused	0x000F	N/A	Unused/reserved	-
SDONL	0x0010	R/W	SDON[7:0]	SDON[15:0] - per channel power on sequencing delay with respect to start of relevant sequencing slot. 16 bit unsigned time value in milliseconds.
SDONH	0x0011	R/W	SDON[15:8]	
SDOFFL	0x0012	R/W	SDOFF[7:0]	SDOFF[15:0] - per channel power off sequencing delay with respect to start of relevant sequencing slot. 16 bit unsigned time value in milliseconds.
SDOFFH	0x0013	R/W	SDOFF[15:8]	

TRCFG	0x0014	R/W	Trimming configuration Bits [7:3] reserved Bits [1:0] trimming mode 0 = None/disabled (default) 1 = Open Loop 2 = Closed Loop Bit [2] trimming control DAC type 0 = SmartFusion MSS OBD (not supported in MPM 6) 1 = CorePWM ⁶	Only relevant to trimming enabled positive voltage channels. 0 is default for all channels other than channels 1-4 for which the default is 2=Closed Loop in order to match the demo/reference design setup.
Unused	0x0015	N/A	Unused/reserved	-
VNOML	0x0016	R/W	VNOM[7:0]	16 bit signed mV value where $UV1 < VNOM < OV1$ (positive voltage channel) or $OV1 < VNOM < UV1$ (negative voltage channel) and $TRILO < VNOM < TRIHI$ for a trimming enabled channel. Used for closed loop trimming to requested nominal output voltage.
VNOMH	0x0017	R/W	VNOM[15:8]	
TRSDLL	0x0018	R/W	TRDEL[7:0]	(Closed loop) trimming startup delay with respect to completion of power on sequencing. Trimming starts after this delay has elapsed. 16 bit unsigned value in ms. Deprecated in v3.0, not configurable via MPM GUI.
TRSDLH	0x0019	R/W	TRDEL[15:8]	
DACOUT_NOML	0x001A	R/W	DACOUT_NOM[7:0]	For open loop trimming DACOUT_NOM is a 16 bit value between 0 and 3300 mV specifying the voltage required as input to the regulator's trim pin circuit in order to achieve the required trimmed nominal output voltage on this channel. Internally this mV value is converted to a DAC duty cycle as follows: CorePWM implementation: $DAC_DUTY_CYCLE = ((DACOUT_NOM / 3300) * 0xFFFF)$ SDD Hard Macro implementation; $DAC_DUTY_CYCLE = ((DACOUT_NOM / 3560) * 0xFFFF)$
DACOUT_NOMH	0x001B	R/W	DACOUT_NOM[15:8]	
DACOUT_HIL	0x001C	R/W	DACOUT_HI[7:0]	Similar to DACOUT_NOM except that this is the regulator trim pin input voltage required to achieve the required trimmed "high" output voltage on this channel.
DACOUT_HIH	0x001D	R/W	DACOUT_HI[15:8]	
DACOUT_LOL	0x001E	R/W	DACOUT_LO[7:0]	Similar to DACOUT_NOM except that this is the regulator trim pin input voltage required to achieve the required trimmed "low" output voltage on this channel.
DACOUT_LOH	0x001F	R/W	DACOUT_LO[15:8]	
TRIHIL	0x0020	R/W	TRIHIL[7:0]	Closed loop trimming high output voltage

TRIIH	0x0021	R/W	TRIIH[15:8]	target. Normally within 10% of VNOM. 16 bit signed mV value.
TRILOL	0x0022	R/W	TRILO[7:0]	Closed loop trimming low output voltage target. Normally within 10% of VNOM. 16 bit signed mV value.
TRILOH	0x0023	R/W	TRILO[15:8]	
DPOL_I2C_BUSL	0x0024	R/W	DPOL_I2C_BUS[7:0]	MPM I ² C bus identifier – currently ignored.
DPOL_I2C_BUSH	0x0025	R/W	DPOL_I2C_BUS[15:8]	
DPOL_I2C_ADDRL	0x0026	R/W	DPOL_I2C_ADDR[7:0]	DPOL only – PMBus I ² C slave address. Only bits [6:0] are significant and store the I ² C address in “0x7F” mode which is shifted left one bit internally to make room for I ² C R/W/GCA bit.
DPOL_I2C_ADDRH	0x0027	R/W	DPOL_I2C_ADDR[15:8]	
RAIL_NAME	0x0028-0x0035	R/W	Rail name as 14 byte array.	Text for MPM GUI to display – 14 bytes, nul padded if less than 14 chars are used
Unused	0x0036-0x003F	N/A	Unused/reserved	

Per-output Configuration Data, Outputs 1-32

The per-trigger/flag output configuration data 'sub-block' base address for outputs ($1 \leq n \leq 32$) is $(0x1000 + ((n-1) \times 0x20))$.

The Table 8 details the per-trigger/flag output registers and their offsets with respect to this per-trigger/flag output configuration data 'sub-block' base address.

Table 9 · Pre-output Configuration Data

Name	Offset	Access	Description
R1_R2_STATE	0x0000	R/W	Rail A1/A2 states used to generate this flag. Bits[7:4] Rail A2 state Bits[3:0] Rail A1 state 0 = rail does not impact flag 1 = OV2 2 = OV1 3 = UV1 4 = UV2 5 = Nominal ($UV1 < \text{nominal} < OV1$) 6 = OV1 or UV1 7 = OV1 or UV2 8 = OV2 or OV1 9 = OV2 or UV2 10 = OFF 11 = Not OFF <i>Note: The following registers use the same encoding.</i>
R3_R4_STATE	0x0001	R/W	Rail A3/A4 states used to generate this flag.
R5_R6_STATE	0x0002	R/W	Rail A5/A6 states used to generate this flag.
R7_R8_STATE	0x0003	R/W	Rail A7/A8 states used to generate this flag.
R9_R10_STATE	0x0004	R/W	Rail A9/A10 states used to generate this flag.
R11_R12_STATE	0x0005	R/W	Rail A11/A12 states used to generate this flag.
R13_R14_STATE	0x0006	R/W	Rail A13/A14 states used to generate this flag.
R15_R16_STATE	0x0007	R/W	Rail A15/A16 states used to generate this flag.
R17_R18_STATE	0x0008	R/W	Rail A17/A18 states used to generate this flag.
R19_R20_STATE	0x0009	R/W	Rail A19/A20 states used to generate this flag.
R21_R22_STATE	0x000A	R/W	Rail A21/A22 states used to generate this flag.
R23_R24_STATE	0x000B	R/W	Rail A23/A24 states used to generate this flag.

R25_R26_STATE	0x000C	R/W	Rail A25/A26 states used to generate this flag.
R27_R28_STATE	0x000D	R/W	Rail A27/A28 states used to generate this flag.
R29_R30_STATE	0x000E	R/W	Rail A29/A30 states used to generate this flag.
R31_R32_STATE	0x000F	R/W	Rail A31/A32 states used to generate this flag.
FLAG_CFG	0x0010	R/W	<p>Other flag configuration</p> <p>Bit[7:6] Output logging during power sequencing</p> <p>0 = None</p> <p>1 = During power on sequencing</p> <p>2 = During power down off sequencing</p> <p>3 = During power on and power off sequencing</p> <p>Bit[5] During power sequencing</p> <p>0 = Hold/don't update flag</p> <p>1 = Track/update flag</p> <p>Bit[4] combine rail states using:</p> <p>0 = OR</p> <p>1 = AND</p> <p>Bit[3] polarity</p> <p>0 = asserted high</p> <p>1 = asserted low</p> <p>Bits[2:0] combine digital input n using:</p> <p>0 = Ignore</p> <p>1 = OR</p> <p>2 = AND</p> <p>3 = XOR</p> <p>4 = OR (NOT input)</p> <p>5 = AND (NOT input)</p>
FLAG_LOGGING	0x0011	R/W	<p>Bits[2:0] Log 0 to 1 transitions</p> <p>0 = Never</p> <p>1..7 = Always</p> <p>Bits[5:3] Log 1 to 0 transitions</p> <p>0 = Never</p> <p>1..7 = Always</p>
Output Name	0x0012-0x001F	R/W	Text for MPM GUI to display – 14 bytes, nul padded if less than 14 characters are used

Per-output Configuration Data, Outputs 33-64

The per-trigger/flag output configuration data 'sub-block' base address for output n ($33 \leq n \leq 64$) is $(0x1400 + ((n - 1) \times 0x20))$. The Table 9 details the per-trigger/flag output registers and their offsets with respect to this per-trigger/flag output configuration data 'sub-block' base address.

Table 10 - Pre-output Configuration Data

Name	Offset	Access	Description
R33_R34_STATE	0x0000	R/W	Rail A33/A34 states used to generate this flag. Bits[7:4] Rail A2 state Bits[3:0] Rail A1 state 0 = rail does not impact flag 1 = OV2 2 = OV1 3 = UV1 4 = UV2 5 = Nominal ($UV1 < \text{nominal} < OV1$) 6 = OV1 or UV1 7 = OV1 or UV2 8 = OV2 or OV1 9 = OV2 or UV2 10 = OFF 11 = Not OFF <i>Note: The following registers use the same encoding.</i>
R35_R36_STATE	0x0001	R/W	Rail A35/A36 states used to generate this flag.
R37_R38_STATE	0x0002	R/W	Rail A37/A38 states used to generate this flag.
R39_R40_STATE	0x0003	R/W	Rail A39/A40 states used to generate this flag.
R41_R42_STATE	0x0004	R/W	Rail A41/A42 states used to generate this flag.
R43_R44_STATE	0x0005	R/W	Rail A43/A44 states used to generate this flag.
R45_R46_STATE	0x0006	R/W	Rail A45/A46 states used to generate this flag.
R47_R48_STATE	0x0007	R/W	Rail A47/A48 states used to generate this flag.
R49_R50_STATE	0x0008	R/W	Rail A49/A50 states used to generate this flag.
R51_R52_STATE	0x0009	R/W	Rail A51/A52 states used to generate this flag.
R53_R54_STATE	0x000A	R/W	Rail A53/A54 states used to generate this flag.
R55_R56_STATE	0x000B	R/W	Rail A55/A56 states used to generate this flag.
R57_R58_STATE	0x000C	R/W	Rail A57/A58 states used to generate this flag.
R59_R60_STATE	0x000D	R/W	Rail A59/A60 states used to generate this flag.
R61_R62_STATE	0x000E	R/W	Rail A61/A62 states used to generate this flag.
R63_R64_STATE	0x000F	R/W	Rail A63/A64 states used to generate this flag.

FLAG_CFG	0x0010	R/W	<p>Other flag configuration</p> <p>Bit[7:6] Output logging during power sequencing</p> <p>0 = None</p> <p>1 = During power on sequencing</p> <p>2 = During power down off sequencing</p> <p>3 = During power on and power off sequencing</p> <p>Bit[5] During power sequencing</p> <p>0 = Hold/don't update flag</p> <p>1 = Track/update flag</p> <p>Bit[4] combine rail states using:</p> <p>0 = OR</p> <p>1 = AND</p> <p>Bit[3] polarity</p> <p>0 = asserted high</p> <p>1 = asserted low</p> <p>Bits[2:0] combine digital input n using:</p> <p>0 = Ignore</p> <p>1 = OR</p> <p>2 = AND</p> <p>3 = XOR</p> <p>4 = OR (NOT input)</p> <p>5 = AND (NOT input)</p>
FLAG_LOGGING	0x0011	R/W	<p>Bits[2:0] Log 0 to 1 transitions</p> <p>0 = Never</p> <p>1..7 = Always</p> <p>Bits[5:3] Log 1 to 0 transitions</p> <p>0 = Never</p> <p>1..7 = Always</p>
Output Name	0x0012-0x001F	R/W	Text for MPM GUI to display – 14 bytes, nul padded if less than 14 characters are used

Miscellaneous/Global Configuration Data

The miscellaneous configuration data 'sub-block' base address is 0x1800.

Table 11 · Miscellaneous/Global Configuration Data

Name	Address	Access	Description	
PWR_ON_CTRL	0x1800	R/W	Power on sequence control. Bits[7:3] reserved Bits[2:0] power up sequence failure action 0 = hold 1 = shutdown 2 = restart slot 3 = restart sequence 4 = skip slot	-
Unused	0x1801	N/A	Unused/reserved	-
SLOT_TIMEOUT_L	0x1802	R/W	SLOT_TIMEOUT[7:0]	SLOT_TIMEOUT [15:0] - power on sequencing slot timeout. 16 bit unsigned time value in milliseconds.
SLOT_TIMEOUT_H	0x1803	R/W	SLOT_TIMEOUT[15:8]	
PWR_OFF_CTRL	0x1804	R/W	Power off sequence control. Bits[7:3] reserved Bit[2] post power off action (currently unused by SF MPM) 0 = stay off 1 = allow restart Bits[1:0] power off sequence 0 = reverse (slot n, n-1,...,1) 1 = forward (slot 1, 2,...,n) 2 = simultaneous (as if all channels in a single slot but off delays still accounted for)	Bit[2] is deprecated in SF MPM v3.0
I2C_SLAVE_ADDR	0x1805	R/W	Bits[6:0] I ² C slave address	MPM I ² C slave address
LOGGING	0x1806	R/W	Bit[0] MPM engine state changes 0 = don't log 1 = log Bit[1] Power on sequencing events 0 = don't log 1 = log Bit[2] Power off sequencing events 0 = don't log 1 = log	Optional logging of MPM engine state transitions and power sequencing events.

CLEAR_LOG	0x1807	R/W	Bit[0] 0 = don't clear log 1 = clear log	Clear log next time MPM reinitializes. A 0x00 "end of file" marker byte is written at the start of the log file so that the next time a log record is written to the log file it is written at the very beginning of the log file. Note that only the first byte of the log file is written (to 0x00) when clearing the log. Once the log file has been cleared MPM also clears the CLEAR_LOG register so that the log is not automatically cleared again next time MPM initializes.
Unused	0x1808	N/A	Unused/reserved	-

Rail Voltage Monitoring Registers

The rail monitoring 'pseudo' register 'sub-block' base address is 0x1900.

Table 12 · Rail Voltage Monitoring Registers

Name	Address	Access	Description	
RAIL1_VH	0x1900	R/O	RAIL1_V[15:8]	16 bit signed value representing the instantaneous output voltage for rail A1 in mV.
RAIL1_VL	0x1901	R/O	RAIL1_V[7:0]	
RAIL2_VH	0x1902	R/O	RAIL2_V[15:8]	16 bit signed value representing the instantaneous output voltage for rail A2 in mV.
RAIL2_VL	0x1903	R/O	RAIL2_V[7:0]	
.				
.				
RAIL63_VH	0x197C	R/O	RAIL63_V[15:8]	16 bit signed value representing the instantaneous output voltage for rail A63 in mV.
RAIL63_VL	0x197D	R/O	RAIL63_V[7:0]	
RAIL64_VH	0x197E	R/O	RAIL64_V[15:8]	16 bit signed value representing the instantaneous output voltage for rail A64 in mV.
RAIL64_VL	0x197F	R/O	RAIL64_V[7:0]	

I2C_STATUS	0x1980	R/C	<p>Status of most recent I²C protocol interaction:</p> <p>0 = OK</p> <p>1 = Invalid register address</p> <p>2 = Unrecognized command</p> <p>3 = Invalid data length (0 or > 64)</p> <p>4 = Data length mismatch (length value received did not match amount of data that followed)</p> <p>5 = NVM write failure (when writing config registers)</p> <p>6 = Bad packet format</p> <p>7 = MPM could not process the command as it was not in the correct state</p> <p>8 = Trim command failed.</p> <p>9 = Unknown PMBus slave device</p> <p>10 = PMBus write operation failed</p> <p>11 = PMBus read operation failed</p> <p>12 = PEC fail on PMBus data read</p> <p>13 = PMBus read data len too long</p>	Mainly for debugging I ² C interactions. Indicates the status of the most recent I ² C protocol interaction.
RTC4	0x1981	R/O	RTC byte 4 – i.e. bits[39:32]	40 bit RTC (Real Time Counter) value from the SmartFusion2 target. Read-only access is provided so that an I2C master can read this and synchronize with its own view of real (world) time. Normally all 5 bytes would be read via I ² C in one operation since reading individual bytes separately and recombining them will most likely yield an incorrect time value.
RTC3	0x1982	R/O	RTC byte 4 – i.e. bits[31:24]	-
RTC2	0x1983	R/O	RTC byte 4 – i.e. bits[23:16]	-
RTC1	0x1984	R/O	RTC byte 4 – i.e. bits[15:8]	-
RTC0	0x1985	R/O	RTC byte 4 – i.e. bits[7:0]	-
Unused	0x1986-0x198F	N/A	Unused/reserved	-
MPM Inputs 1-32	0x1990-0x1993	R/O	Bitmapped value state of the first 32 MPM input pins.	32bit unsigned value.
MPM Inputs 33-64	0x1994-0x1997	R/O	Bitmapped value state of the last 32 MPM input pins.	32bit unsigned value.

MPM Outputs 1-32	0x1998-0x199B	R/O	Bitmapped value state of the first 32 MPM output pins.	32bit unsigned value.
MPM Outputs 33-64	0x199C-0x199F	R/O	Bitmapped value state of the last 32 MPM output pins.	32bit unsigned value.
MPM Regulator Enables 1-32	0x19A0-0x19A3	R/O	Bitmapped value state of the first 32 MPM regulator enable pins.	32bit unsigned value.
MPM Regulator Enables 33-64	0x19A4-0x19A7	R/O	Bitmapped value state of the last 32 MPM regulator enable pins.	32bit unsigned value.
MPM State	0x19A8-0x19AB	R/O	Current state of the MPM Engine: 0 = stopped 1 = starting 2 = started 3 = stopping	32bit unsigned value.
Unused	0x19AC-0x19AF	N/A	Unused/reserved	-
MPM Channel States	0x19B0-0x1AAF	R/O	64 channel state registers with the contents indicating the state of the corresponding channel as follows: 0 = off 1 = under voltage level 2 2 = under voltage level 1 3 = nominal 4 = over voltage level 1 5 = over voltage level 2	64 x 32 bit unsigned values for channels 1 to 64 in ascending order.

I²C Command Registers

The I²C command 'pseudo' register 'sub-block' base address is 0x8000.

Table 13 · I²C Command Registers

Name	Address	Access	Description
I2C_CMD_0	0x8000	W/O	I ² C command pseudo register. See below for details about how this is used.
Unused	0x8001-0xFFFF	N/A	Unused/reserved

Commands are implemented as writes to an MPM I2C command pseudo register of an 8 bit opcode followed by an optional sequence of up to 63 command data bytes. The reference design implements a single command register at MPM register map address 0x8000 and a set of command opcodes that can be written to this command register. Opcodes that are currently not defined for command register 0x8000 are reserved by Microsemi SoC Products Group for future use. Additional command registers and command opcodes can be implemented by end users by adapting and extending the reference design firmware.

Table 13 describes the opcodes defined for command register 0x8000.

Table 14 · Command Register

Code	Name	Parameters	Description/Response
0x00	Unused/reserved	N/A	N/A
0x01	START	None	Initiate power on sequencing. Maps to the MPM driver API <code>mpm_start()</code> . Ignored if MPM is not in stopped state.
0x02	STOP	None	Initiate power off sequencing. Maps to the MPM driver API <code>mpm_stop()</code> . Ignored if MPM is not in started state.
0x03	MARGIN_LOW	One eight bit byte containing the channel to be trimmed/margined (1..64).	Set margin level to MARGIN_LOW for the specified channel (1..64). For Analog Point of Load (APOL) channels: In open-loop mode, this sets the trim pin voltage to DACOUT_LO. In closed-loop mode, the trim pin voltage is continually adjusted until the output voltage reaches TRLO. For Digital Point of Load (DPOL) channels: Uses PMBus OPERATION (0x01) command to program the DPOL to output the margin low voltage. Ignored if MPM is not in started state.

0x04	MARGIN_NOM	One eight bit byte containing the channel to be trimmed/margined (1..64)	<p>Set margin level to NOMINAL (default) for the channel</p> <p>For Analog Point of Load (APOL) channels:</p> <p>In open-loop mode, this sets the trim pin voltage to DACOUT_NOM.</p> <p>In closed-loop mode, the trim pin voltage is continually adjusted until the output voltage reaches V_{NOM}.</p> <p>For Digital Point of Load (DPOL) channels:</p> <p>Uses PMBus OPERATION (0x01) command to program the DPOL to output the normal nominal voltage.</p> <p>Ignored if MPM is not in started state.</p>
0x05	MARGIN_HIGH	One eight bit byte containing the channel to be trimmed/margined (1..64).	<p>Set margin level to MARGIN_HIGH for the channel</p> <p>For Analog Point of Load (APOL) channels:</p> <p>In open-loop mode, this sets the trim pin voltage to DACOUT_HI.</p> <p>In closed-loop mode, the trim pin voltage is continually adjusted until the output voltage reaches TR_{HI}.</p> <p>For Digital Point of Load (DPOL) channels:</p> <p>Uses PMBus OPERATION (0x01) command to program the DPOL to output the margin high voltage.</p> <p>Ignored if MPM is not in started state.</p>
0x06	INIT	None	<p>Initiate reinitialization of MPM. Maps to the MPM driver API mpm_init(). This causes MPM to (re)load the latest configuration settings. Any MPM configuration register changes made via I²C since the last call to mpm_init() take effect at this stage.</p> <p>Ignored if MPM is not in stopped state.</p>
0x07	LOG_INFO	None	<p>Returns basic status information about the log as follows:</p> <p>[0..3] Logging API Major number.</p> <p>[4..7] Max Records possible.</p> <p>[8..11] Max Guaranteed Records. In the event of a log roll-over the number of records that will be available.</p> <p>[12..15] Current number of entries in the log.</p> <p>[16..19] Current Sequence No.</p>
0x08	LOG_RESET	None	<p>Force a clear down of the log. All queued messages will be discarded, the log will be erased, the sequence number will be reset to 0 and a logging started entry will be placed in the log with sequence number 0.</p>
0x09	LOG_READ_LAST	None	<p>Reads the last 4 entries from the log (i.e. the newest entries). The response read back will contain 64 bytes of data with between 1 and 4 log records returned depending on the number of records in the log (external reading should never see less than 1 record returned as we insert a log started record into the new log before allowing read access). This command also resets the</p>

			internal sequential log pointer to the end of the log.
0x0A	LOG_READ_NEXT	None	Read the next 4 entries from the log (these should be the next oldest records if no new entries have been added since the last read operation). The response read back will contain 64 bytes of data with between 0 and 4 log records returned depending on the number of records in the log and our starting position. Reading less than 4 records indicates the end of the log has been reached.
0x0B	LOG_SYSTEM_INFO	None	Information about the MPM firmware in the device: [0..19] Zero terminated MPM version number string. [20..21] Version major number. [22..23] Version minor number. [24..25] Sub version number. [26..27] Device Id. This is a code to identify the hardware platform: we use 0 for MPM running on the SmartFusion Dev Kit board, 1 for MPM running on the SmartFusion Eval Kit board and 2 for MPM running on the SmartFusion2 DEv Kit board. Customers can assign their own numbers to allow them id different products or product revisions etc
0x0C	PMBUS_RAW_WR	3 x single byte fixed parameters followed by a variable length data block. Byte 1: I2C bus number, currently only 1 is allowed here. Byte 2: 7 bit address of PMBus device in low 7 bits. Byte 3: Flag byte, b0 is PEC presence flag and rest are 0. Byte 4: first byte of data block. Overall length is the I2C command length – 4.	Writes a block of data to the specified PMBus device. If a PEC byte is supplied and the device does not support it, MPM will remove the PEC byte and likewise, if the device supports PEC and now PEC byte is supplied, MPM will append the correct PEC byte before transmission.
0x0D	PMBUS_READ	4 x single byte parameters: Byte 1: I2C bus number, currently only 1 is allowed here. Byte 2: 7 bit address of PMBUS device in low 7 bits. Byte 3: operation - 0 = read byte, 1 = read word, 2-64 = read block of n-1 bytes. If b7 is set then read is extended command. Byte 4 - PMBUS command byte for read operation	This command allows data to be read from the selected PMBus device using byte, word or block reads up to 63 bytes in length. The returned response always has 1 + n bytes of data with the first byte being a status byte with 0 for success and one of the values described for I2C_STATUS on failure.
0x0E-0xFF	Unused/reserved	N/A	N/A

Notes

The MPM version 6 register map and tools are not backward compatible with previous versions of MPM.

SmartFusion2 MPM Data Logging

Data Logging

Events

The following events can be logged (configurable via the MPM GUI or I²C configuration registers).

MPM Engine State Changes

SF MPM consists of four states - stopped, starting, started, and stopping. Transitions between states are conditional on Rail statuses or can be initiated via MPM API functions (or I²C commands). These transitions can be logged. Figure 26 illustrates a state diagram of SF MPM.

Note: Can "loop" on starting/stopping state depending on timeout and on fail action config

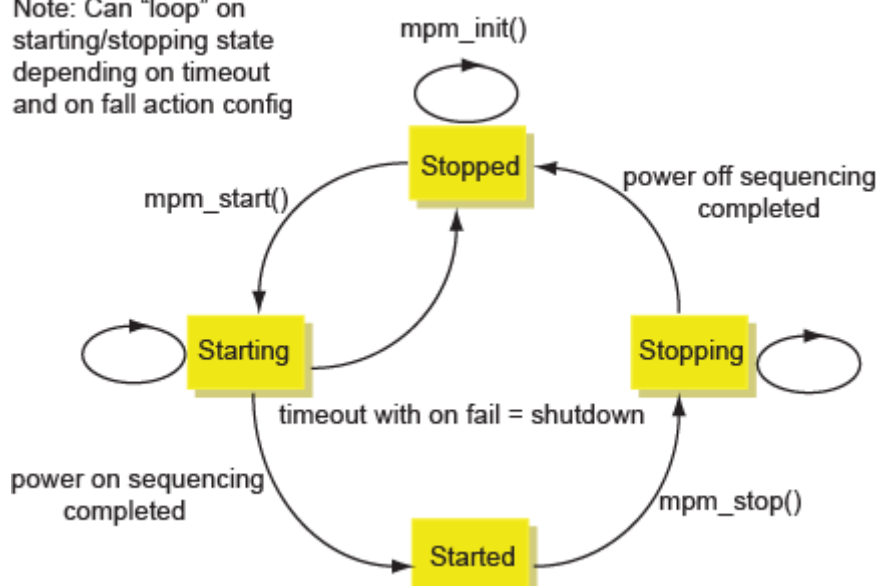


Figure 41 · MPM State Transition Diagram

Output Trigger/Flag State Change

Users may log low-to-high (0 to 1) or high-to-low (1 to 0) transitions on any configured output, which can in turn be configured as a combination of Rail flags and corresponding digital input.

Figure 27 shows the GUI configuration for output/trigger logging.

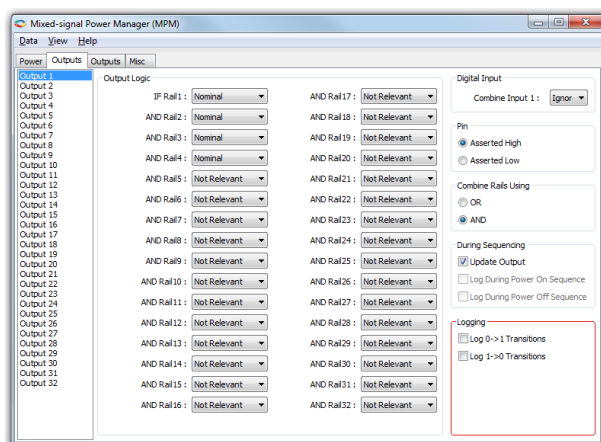


Figure 42 · GUI Configuration of Output/Trigger Data Logging

Power Sequencing Events

The user may log power-on sequencing events or power-off sequencing events. Figure 28 shows the GUI (global) configuration for power sequencing event logging.

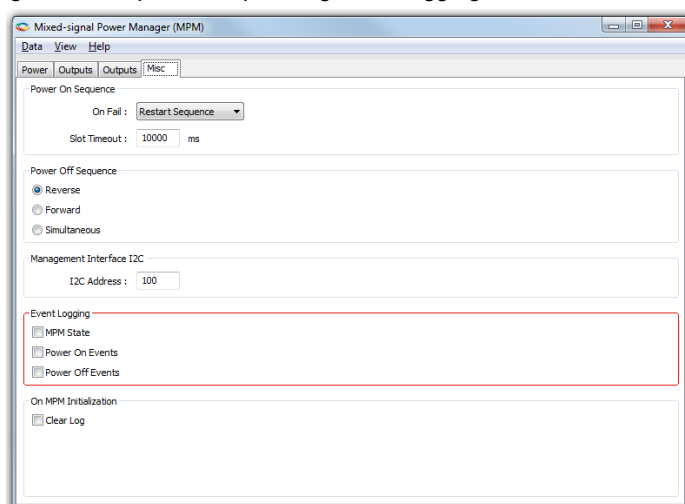


Figure 43 · Power Sequencing Events Logging

Log Record Format

Each log item consists of a record of 16 bytes as follows (LSB first):

- Byte 0: log record type
- Byte 1: log data Byte 1
- Byte 2: log data Byte 2 for Power on/off sequencing events 0x05/0x06 only – otherwise reserved
- Byte 3-4: Reserved
- Byte 5-8: Sequence number, Little-endian order
- Byte 9-13: log timestamp, Big-endian order
- Byte 14: Checksum of bytes 0-13 inclusive
- Byte 15: Flag Byte

More information on the above bytes is provided in subsequent sections.

Log Record Type Byte

Table 14 describes record types used for SF MPM data logging.

Table 15 · Data Logging Record Types

Value	Meaning
0x00	Null record type: Used to indicate the next free slot in the log (empty log item)
0x01	MPM state changed
0x02	Output flag/trigger transition occurred in the lower bank (flag outputs 1-32)
0x03	Output flag/trigger transition occurred in the upper bank (flag outputs 33-64)
0x04	Power on sequencing event
0x05	Power off sequencing event
0x06	Log status changed
0xFE	Extension record type
0x07 - 0xFD and 0xFF	Reserved

Log Data

Log data meaning is dependent on the corresponding log record type byte.

Table 16 · Log Data

Log record byte = 0x01 (MPM state changed)	
Bit(s)	Meaning
1:0	0 = stopped
	1 = starting
	2 = started
	3 = stopping
7:2	Unused
Log record byte = 0x02 (Output flag/trigger transition occurred (outputs 1-32))	
4:0	Output number N, minus 1 (valid values are 0-31, while output numbers range from 1-32)
5	Change to new value of output - 0 or 1
7:6	Unused

Log record byte = 0x03 (Output flag/trigger transition occurred (outputs 33-64))	
4:0	Output number N, minus 33 (valid values are 0-31, while output numbers range from 33-64)
5	Change to new value of output - 0 or 1
7:6	Unused
Log record byte = 0x04/0x05 (Power on/off sequencing event)	
Byte 1[2:0]	Event Type [0 = start; 1 = finish; 2 = timeout; 3 = hold; 4 = shutdown/terminate; 5 = restart slot; 6 = restart sequence; 7 = skip slot]
Byte 1[7:3]	Unused
Byte 2[5:0]	Slot number: For "restart slot events" (configuration 5) and "skip slot events" (configuration 7), this indicates the slot number, minus 1 (valid values are 0-63, while slot numbers range from 1-64)
Byte 2[7:6]	Unused
Log record byte = 0x06 (Log status changed event)	
2:0	0 = Normal Log Startup 1 = Abnormal Log Startup 2 = Log Erased
Log record byte = 0xFE (Extension record)	
	Bytes 2..13 in this log record are available for user defined additional data rather than timestamp and sequence. Bytes 14..15 and byte 1 are reserved.

Log Timestamp

The SmartFusion2 MSS Real Time Counter (RTC) 40 bit (5 byte) counter value is used to timestamp log records (the SmartFusion2 RTC is actually 43 bits but we ignore the top 3 bits in this application).. By default the RTC is clocked by a 1MHzclock with a prescaling divider of 3906 so that the resolution of the RTC timer is 1 second / (1MHz/3906) = 1/256th of a second = 3.90625ms and the timer can measure up to $0xFFFFFFFF * 0.00390625$ seconds = ~ 4,294,967,296 seconds = ~ 136 years. The RTC timestamp is reset by the power on reset and system reset and so cannot be used as a true RTC to allow calendar based time stamping unless an additional battery backed RTC is available in the system.

The timestamp measures the time (in 3.90625ms units) since the most recent reset of the MSS RTC.

Because of the way the MSS RTC operates, there is no guarantee that log records have monotonically incrementing timestamps. The presence of a log entry with timestamp greater than the immediately following log entry indicates logging across a SmartFusion2 cSoC device reset.

Due to the fact that the RTC range is 136 years there is no special handling of the situation in which the RTC overflows and wraps around from 0xFFFFFFFF to 0x000000000.

The log file is stored in the SmartFusion2 Development Kit 8 MB external FLASH memory accessed via SPI.

Log records are written in blocks of four, and the log wraps around when 'full'. Sequence numbers in the log record show the sequence in which the records were written. The timestamps only show times since power up as the clock resets to zero when the device is unpowered.

There is an API within the firmware for access to the log records, and they can be accessed using this API via I²C.

Log Access Via I²C

The commands supported to access the log via I²C are described in Table 14. These are the I²C commands with command codes 0x07 (LOG_INFO) through to 0x0B (LOG_SYSTEM_INFO).

The difference between the max number of records possible in the log and the max number guaranteed is down to a choice in the design to use records in blocks of 4K or 8K. Some records in a block might not be used.

The log retrieval scheme assumes that there is a single master reading the logs at a time, as there is a single pointer in the firmware that tracks which record to hand back with each 0x0A (LOG_READ_NEXT).

eNVM Logging

Defining MPM_LOG_TO_SPI configures MPM to use the external SPI Flash and leaving it undefined configures MPM to use the internal 16 KB eNVM memory.

When logging to eNVM, there are a maximum of 1022 records and a maximum guaranteed 894 records.

MPM for SmartFusion2 Reference Design Trimming

Operation

Overview

In general, the purpose of trimming is to perform small adjustments on the output voltage of a regulator or power supply (less than 10% of the output) by driving the trim, adjust, or feedback pin of the regulator. There are two main modes for trimming, open-loop and closed-loop, as described below.

Open-Loop Trimming

The hardware for open-loop trimming resembles the setup shown in Figure 29.

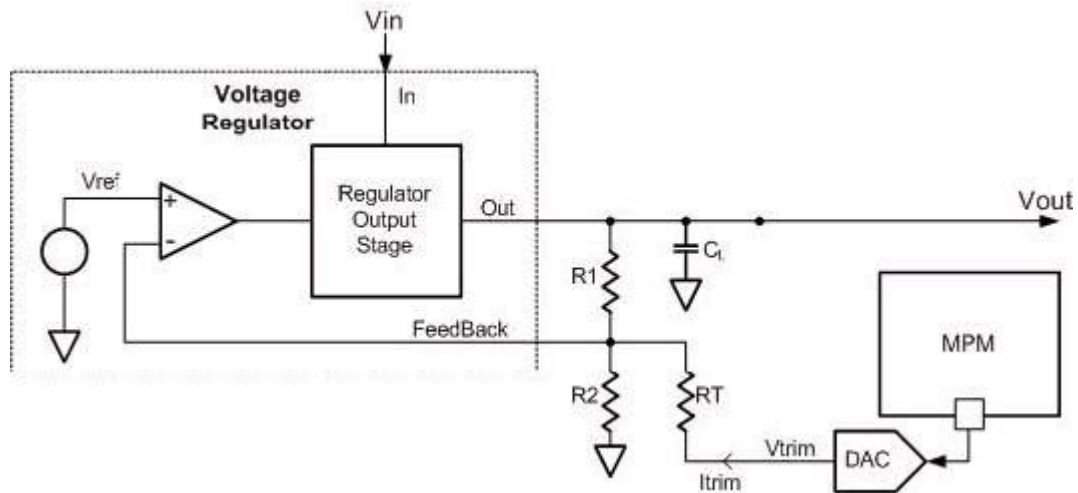


Figure 44 · Open-Loop Trimming Using SmartFusion2 MPM

The regulator feedback is controlled by SmartFusion2 MPM, which puts out a pulse-width modulated (PWM) signal that acts as a DAC when fed through a low pass filter such as an RC network. With open-loop trimming, you can adjust the regulator output voltage by driving this signal with different voltages of small variation.

Open-loop trimming is a passive mode. It is not run continually; the feedback pin value is never adjusted. SmartFusion2 MPM sets the feedback pin value at system initialization and leaves it at that fixed value until a reset or power cycle occurs, or if a MARGIN-HIGH or a MARGIN-LOW command is issued.

Closed-Loop Trimming

The hardware for closed-loop trimming is identical to that of open-loop trimming, except that there is feedback from the regulator output to SmartFusion2 MPM, as seen in Figure 30.

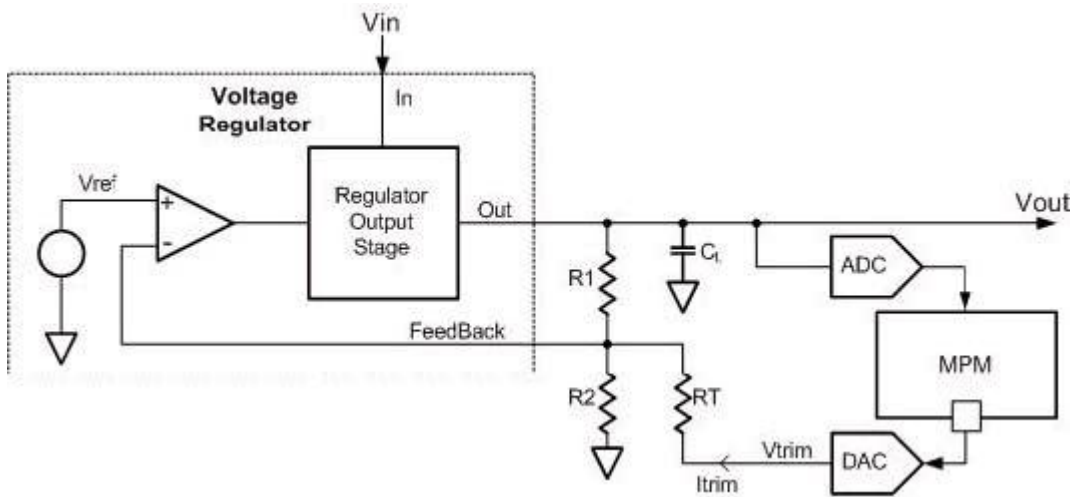


Figure 45 · Closed-Loop Trimming Using SmartFusion2 MPM

In closed-loop trimming, MPM constantly scans (once per loop) the output voltage of the regulator, and actively adjusts the regulator feedback voltage to drive the regulator to some target output voltage.

Closed-loop trimming is an active mode; it is continually operating. The algorithm for trimming is linear, and is done as follows:

1. Read V_{out} (rail value)
2. Compare V_{out} to $V_{outtarget}$
3. Set V_{trim} according to the following:
 - ✓ If $V_{out} > V_{outtarget}$, $V_{trim} = V_{trim} + 1$
 - ✓ If $V_{out} < V_{outtarget}$, $V_{trim} = V_{trim} - 1$

Refer to the "Trim Type" section on page 61 for information on how $V_{outtarget}$ is determined.

SmartFusion2 MPM is responsible for driving the V_{trim} pin of the above systems (both open-loop and closed-loop). However, varying the R_T and R_2 resistances (R_1 is typically internal to regulators) changes the regulator's sensitivity to variations on the R_{trim} pin. It is your responsibility to ensure that R_T is large enough so that V_{trim} variations will not put the regulator out of its operating range (typically only small variations on the feedback pin are allowed, on the order of mV's), and not so large as to make V_{trim} voltages changes negligible to the system.

Suggested resistances, as well as trim pin voltage ranges (typically around 0.8 V) can usually be found in regulator datasheets and associated application notes.

GUI Operation

The sections of the GUI highlighted in Figure 31 define configuration parameters used for trimming on a per-channel basis.

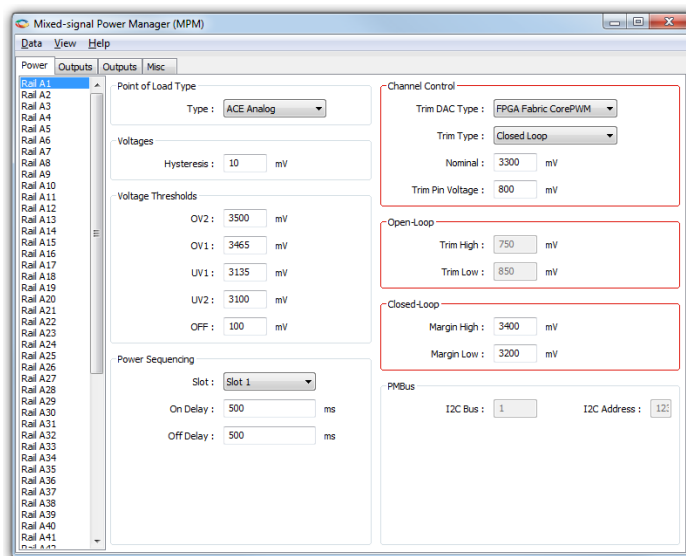


Figure 46 · Trimming GUI Section

Channel Control

The following section describes the parameters in the Channel Control pane of the Power tab of the GUI. Clicking on the pane displays online help on the right-side of the GUI.

For DPOL channels only the **Channel Control > Nominal and Closed Loop > Margin High and Closed Loop > Margin Low** settings are relevant. DPOL trimming is done using the PMBus OPERATION (command code 0x01) command. DPOLs do not support dynamic closed loop trimming by way of a trim voltage input signal.

Trim Type

This is the trimming type as described in the "Operation" section on page 59.

Disabled

If this option is selected, trimming is not performed for this rail. The portion of the MPM loop that would normally be executed for trimming is skipped for this particular rail.

Open-Loop

If this option is selected, it resets the trim pin driven with **Trim Pin Voltage**.

After startup, the trim pin voltage is not adjusted, unless a MARGIN_LOW or MARGIN_HIGH command is issued, in which case the trim pin voltage jumps to TRIM_LOW or TRIM_HIGH, respectively.

Closed-Loop

As with open-loop, on reset the trim pin is driven with one of three values:

However, after all rails have been powered up, the trim pin voltage is adjusted according to the scheme described in the "Closed-Loop Trimming" section on page 60. The target voltage ($V_{outtarget}$) is one of three values:

1. Nominal: By default, or if the MARGIN_NOM command has been received by MPM
2. Margin High: If the MARGIN_HIGH command has been received by MPM
3. Margin Low: If the MARGIN_LOW command has been received by MPM

Note: This is a target voltage, and you must ensure that the voltage is within the operating range of the system. That is, you must ensure that the voltage range on the DAC output, which ranges from 0 V to 3.3 V for soft CorePWM implementation is sufficient for the system (feedback resistor, voltage divider, trim pin operating range, etc.) and for the amount of trimming you expect to be performed.

Trim DAC Type

For trimming, this determines the type of DAC used for this rail. SmartFusion2 MPM only supports trimming via FPGA Fabric CorePWM.

FPGA Fabric CorePWM

This represents the CorePWM in low-ripple DAC mode. Requires a low-pass filter at the output of the digital I/O.

When using the DMPM-DC the MPM GUI configuration settings for trimming of Channel A1 to Channel A8 must be matched by the appropriate configuration of DMPM-DB trimming jumpers (Jumper fitted to enable trimming, removed to disable trimming):

- Channel A1 - JP3
- Channel A2 - JP26
- Channel A3 - JP27
- Channel A4 - JP28
- Channel A5 - JP2
- Channel A6 - JP29
- Channel A7 - JP30
- Channel A8 - JP31

MPM Firmware Notes

Introduction

This section provides additional information about the MPM firmware to assist implementers in understanding the code and modifying it for their designs. It provides information on how MPM is implemented using FreeRTOS and on the compile time defines which can be used to alter facets of MPMs operation.

MPM and FreeRTOS

MPM 6.1 uses a Real Time Operating System to coordinate the various subsystems that implement its functionality. MPM 6.1.100 uses FreeRTOS 7.1.1 (www.freertos.org) as its RTOS. The following describes the tasks that are used in MPM 6.1 to implement its functionality

The MPM reference application consists of a main application task which is responsible for overall system operation and a number of tasks which implement the MPM functionality. The following are the tasks and their relative priorities ranked from lowest to highest:

mpm_logging_task()

mpm_logging_task – This task is currently configured with a priority just above the FreeRTOS idle task and is responsible for initialising the logging system and writing log records into the event log. It has a low priority on the basis that writing to the log is a non-critical operation which should impact the other functionality as little as possible.

It reads the records to log from a queue and writes them to either the internal eNVM or an external SPI connected FLASH memory device. The length of the queue is determined by the `MPM_LOG_QUEUE_LEN` define in `mpm_logging.h` which is currently set to 10.

This task uses two mutual exclusion semaphores to control queue insertion (guaranteeing insertion and time stamping in the order of arrival) and exclusive access to the SPI interface.

mpm_main_task()

This is the example main application task which is responsible for the user interface via SW1 - SW2, SW10 and the 8 LEDs on the SmartFusion2 Development Kit. This task uses the MPM API to start and stop sequencing and to demonstrate how to monitor and control the MPM engine from an application.

This has a priority which is less than the remaining MPM tasks to ensure that it does not interfere with the operation of MPM. This task sleeps for 20mS after each pass through the main loop which allows the logging task get an opportunity to run.

mpm_i2c_slave_task()

This task is responsible for executing commands received over the slave I2C interface. It is linked to the I2C slave write handler ISR call back via a single entry queue which decouples the I2C command processing from the I2C ISR. This task relies on a mutual exclusion semaphore to arbitrate access to the PMBUS interface for some operations.

This task sleeps until an I2C request arrives, processes and responds to the request and goes back to sleep.

mpm_threshold_task()

This task is responsible for checking the state of the analog channels based on the voltages read via the ADS7953 ADC. It is also responsible for updating the MPM outputs based on any changes to the channel states.

This task sleeps for 2ms after each iteration through its checking.

mpm_timer2_task()

This task is responsible for handling all the sequencing and trimming for MPM. As such, it is the central task for managing the internal operation of MPM. It is also responsible for servicing the `MSS_I2C_system_tick()` functions for the 2 MSS I2C channels used by MPM to ensure timeout operations work correctly.

This task replaces a periodic interrupt in previous MPM implementations and the timing in the underlying sequencing code has been modified to work correctly even if this task is not called with complete regularity.

This task sleeps for 2 mS after each pass through.

Compile Time Options in the MPM Code

There are a number of build options in the MPM code which are controlled by the use of macros. The macro definitions can be used to control implementation of optional functionality and/or amounts of resources used by MPM. Many of these macros are defined in `MPM.H`.

This section describes the main options and how to configure them.

32 vs 64 Channel Operation

MPM 6.1 can support sequencing of a maximum of either 32 or 64 power channels. To ensure that the support for 64 channels does not impact the code and data usage of 32 channel builds, the **MPM_64_CHANNEL_SUPPORT** macro is used to exclude 64 channel specific code and data when it is not required.

MPM is only configured for 64 channel operation if **MPM_64_CHANNEL_SUPPORT** is defined in the project.

The **MPM_64_CHANNEL_REMAP** macro can be defined to make MPM reconfigure the I/O to allow APOLs 5 to 8 and DPOLs 5 to 8 appear as channels 33 to 40 respectively. This involves remapping the POL enable signals in hardware and remapping the DPOL PG to channel correspondence in the firmware. This allows MPM demonstrate proper 64 channel operation.

Note: For simplicity's sake, the hardware configuration shipped with the MPM 6.1 reference design has the support for 64 channel operation built in and the initialisation routines can control mapping of the POLs between the lower 32 channels and the upper 32 channels. The additional hardware resources for 64 channel mode can be removed by the user if they are not required and the fabric reconfigured.

Analog Monitoring of DPOL Voltages

The ADS7953 ADC used on the SF2-DMPM-DB has 16 analog channels. The first 8 of these channels are used to monitor the outputs of the APOLs and the remaining 8 are connected to the outputs of the DPOLs. Under normal circumstances these 8 additional channels are not used but it is possible to set up configurations in MPM which allow them be used to view the output voltage from the DPOLs via the ADC.

Defining the **MPM_SHADOW_APOLS** macro and the **MPM_RELOAD_ENVI** macro creates 8 dummy APOL channels numbered 17 to 23 which have identical nominal voltage and thresholds to the corresponding 8 DPOLs. For example DPOL1 and APOL 9 (channels 9 and 17 respectively) are paired and when channel 17 is read via the MPM GUI it should show the same voltages as channel 9. If **MPM_SHADOW_APOLS** is not defined, the user can still configure the dummy DPOLs via the MPM GUI to achieve the same result.

Known Good Configuration Setup

When modifying the hardware configuration of the MPM design, switching between 32 and 64 channel build modes or reprogramming the eNVM directly from Flashpro, the configuration in eNVM can become invalid and will usually cause sequencing to fail.

There is a convenient method to ensure that a known good configuration is written to the eNVM on power up which is controlled via the **MPM_RELOAD_ENVM** macro. Defining this macro in the project causes MPM to reload the eNVM configuration from a set of values defined in the `mpm_init_data.c` file if the current contents do not match the values stored there.

This reloading feature is further controlled by SW10.2 as described earlier.

It is important to undefine this macro or set SW10.2 to off when you are finished reloading the eNVM as otherwise any configuration changes made via the GUI will be undone at the next reset or power up.

There are a number of other macros associated with this configuration operation which are documented in the comments in `mpm_init_data.h`.

Logging Support

The logging of events to nonvolatile memory can be enabled by defining the **MPM_ENABLE_LOGGING** macro. Defining this macro causes the `mpm_logging_task()` to be created and scheduled and adds the logging access functionality to the I2C slave interface. This macro is usually defined in the project options and logging is enabled in the default build shipped with MPM 6.1.

Internal vs External Logging

When logging is enabled in MPM (via the **MPM_ENABLE_LOGGING** macro), it is possible to select logging to the internal eNVM memory or to an external SPI connected FLASH memory device.

This is controlled via the **MPM_LOG_TO_SPI** macro. Defining this macro in `mpm_logging.h` or in the project configuration, selects logging to an external FLASH device which is accessed via an SPI interface. On the SmartFusion2 Development Kit boards, this FLASH device is an Atmel AT25DF641 which has up to 8MB of storage available. By default we use the first 32KB of this for logging.

If this macro is undefined, the 16KB of eNVM reserved for logging is used by default.

The amount of memory used for logging and its location within the memory device are defined in a series of macros in the `mpm_logging.h` file. See the associated comments for details of this. If you increase the size of the eNVM logging memory allocation, you will need to ensure that the details for the associated storage client are also updated.

I2C Slave Interface Support

The I2C Slave Interface in MPM 6.1 can be enabled and disabled through the use of the **MPM_ENABLE_I2C** macro. Defining this macro causes the `mpm_i2c_slave_task()` to be created and scheduled and configures the MSS I2C 0 interface appropriately. This macro is usually defined in the project options and the Slave I2C interface is enabled in the default build shipped with MPM 6.1.

PMBus I2C Device Selection

The fabric design for MPM 6.1 has a Core I2C implemented in it which is currently unused. The **USE_MSS_PMBUS** macro must be defined to ensure that MPM uses MSS I2C 0 for the PMBus interface.

MPM Daughter Board Hardware Guide

Introduction

The SmartFusion2 Digital Mixed Signal Power Manager Daughter Board (SF2-DMPM-DB) enables system designers to evaluate and demonstrate the functionality of Microsemi's power management solutions in hardware, using a 16 regulator benchtop power management development system. The daughter board includes 8 analog regulators with adjustable bias voltage POTs, 8 digital regulators residing on a connectable PMBus (I²C) bus and a series of LED's that may be used to demonstrate and explore MPM digital output functionality. Figure 32 shows the SF2-DMPM-DB with its major components.

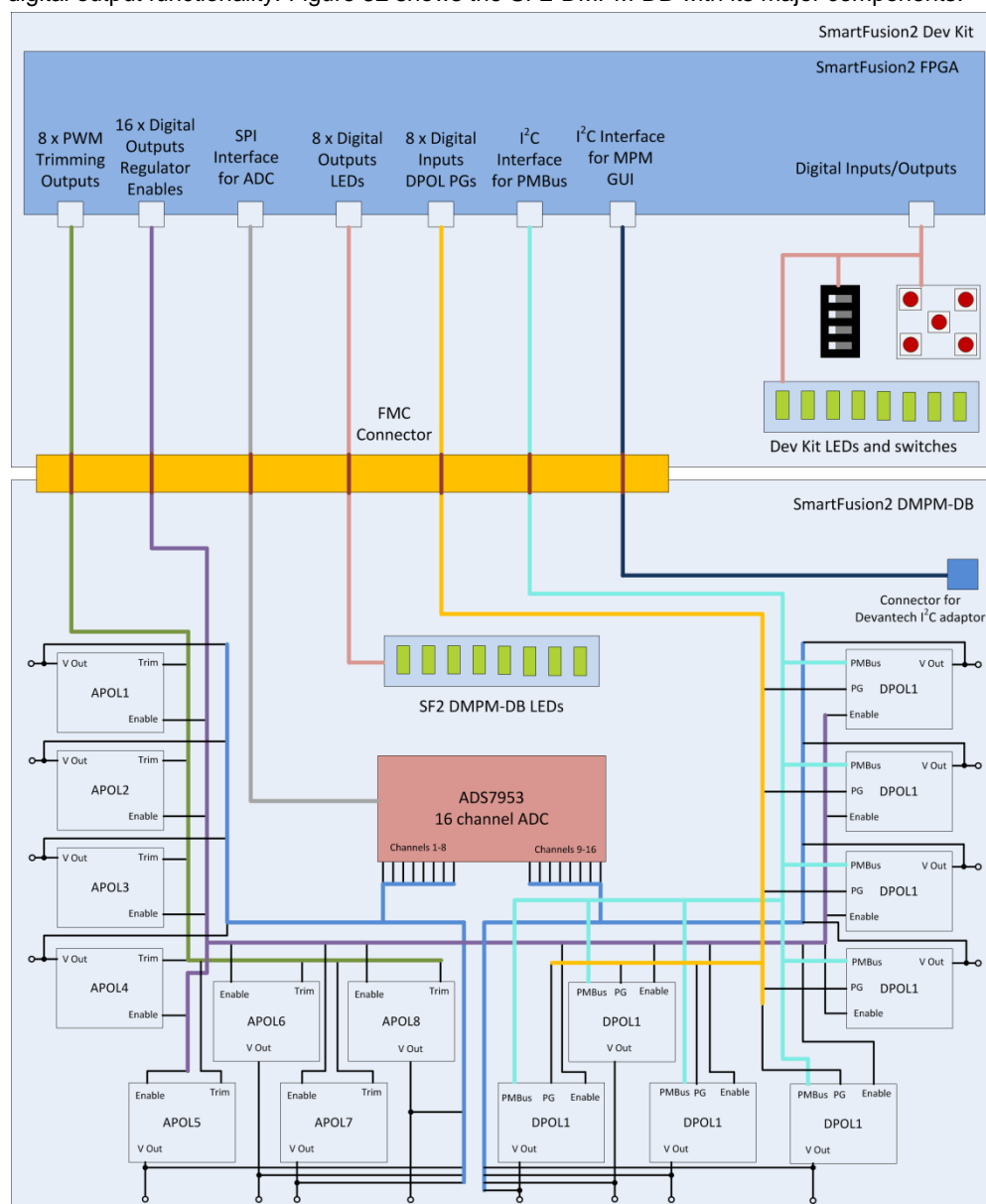


Figure 47 · SF2-DMPM Daughter Board Block Diagram

Kit Contents

Table 17 · SF2-DMPM Daughter Board Kit Contents

Quantity	Description
1	SF2-DMPM Daughter Board
1	12V, 5A AC Power Adapter
1	Any plug to Grounded 3 pin UK/Hong Kong Plug Adapter
1	Enhanced USB I ² C Module
1	USB Cable A-B 3'
50	Mini Jumpers
1	SF2-DMPM-Kit Quickstart Card

Related Information

- SmartFusion2 Development Kit Board
- SmartFusion2 Reference Documents
- Libero IDE Design Software

Board Description

The SF2 DMPM Daughter Board provides a benchtop demonstration and development platform for Microsemi's SF2 MPM reference design, specifically for version 6.1 of the design – which incorporates digital regulators (controlled and monitored via PMBus) in addition to the existing analog regulators from version 3.0 of the reference design which are monitored via an external ADC.

Figure 33 is the SF2 DMPM Daughter Board connected to the SmartFusion2 Development Kit through their respective FMC connectors.



Figure 48 · SF2-DMPM Daughter Board with SmartFusion2 Development Kit Board

Digital Mixed Signal Power Manager Board Components

Table 18 · DMPM-DB Components

Serial Number	Name	Description
1	APOL1,2,3,4	3.3 V regulated supply, Microsemi Analog Mixed Signal Group NX9415
2	APOL5,6,7,8	1.5 V regulated supply, Microsemi Analog Mixed Signal Group LX9610
3	DPOL1,2,3,4	3.3 V digital regulated supply, Intersil ZL6105
4	DPOL5,6,7,8	1.5 V digital regulated supply, Lineage Power PDT012A0X
5	SW8, SW13, SW14, SW15, SW9, SW16, SW17, SW18, SW10, SW12, SW19, SW20, SW11, SW21, SW22, SW23	Switch to enable/disable voltage regulators (1 to 16) to demonstrate regulator failure.
6	J3	ZL6105 programming header, used in conjunction with the Zilker programmer (ZLUSB EVAL1Z).
7	J23	FMC connector used to connect the SF2-DMPM-DB to the SmartFusion2 Development Kit Board.

Note: DPOL1 to DPOL8 may be programmed to output different voltages.

APOL1,2,3,4 - 3.3 V regulated supply, Microsemi Analog Mixed Signal Group NX9415

These synchronous buck switching voltage regulators provide a 3.3 V output (as configured) at up to 5 A. Different output voltages can be selected by changing the feedback voltage. For more information, please refer to the NX9415 datasheet.

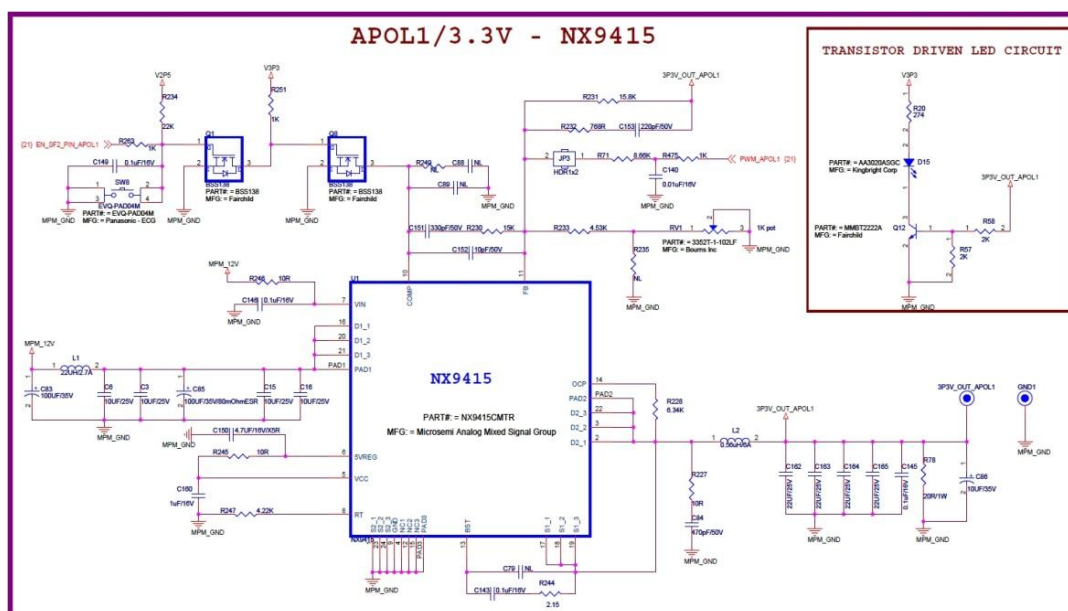


Figure 49 · APOL1 3.3 V Switching Power Supply

APOL5,6,7,8 - 1.5 V Regulated Supply, Microsemi Analog Mixed Signal Group LX9610

These synchronous buck switching voltage regulators provide a 1.5 V output (as configured). Different output voltages can be selected by changing the feedback voltage. For more information, please refer to the LX9610 datasheet.

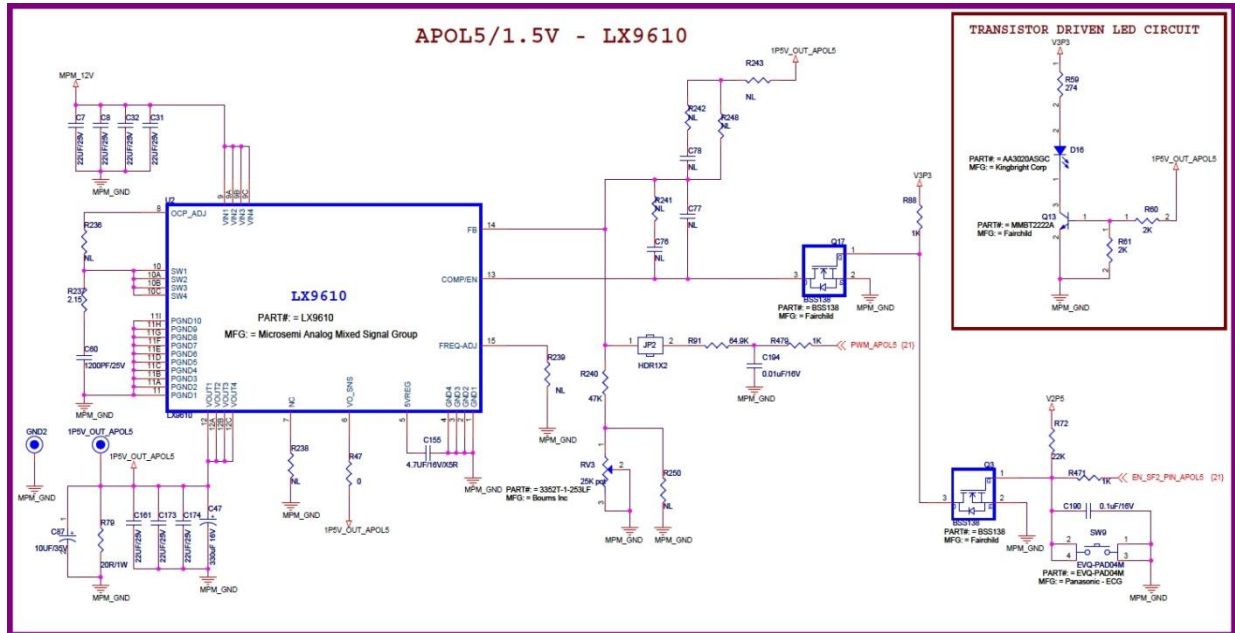


Figure 50 • APOL5 1.5 V Switching Power Supply

DPOL1,2,3,4 - 3.3 V Intersil ZL6105 Digital Power Supply

These regulators are Intersil ZL6105 digital power controllers. The ZL6105 uses the I2C/PMBus 3-wire communication bus to connect to the host controller (in this case, MPM via the FMC connector on the SF2-DMPM-DB). These devices can be programmed to output voltages of 0.54 V to 5.5 V with currents of up to 3A. The user may set the output voltage via the Zilker Labs Power Navigator. For more information, please refer to the Intersil website and the ZL6105 datasheet.

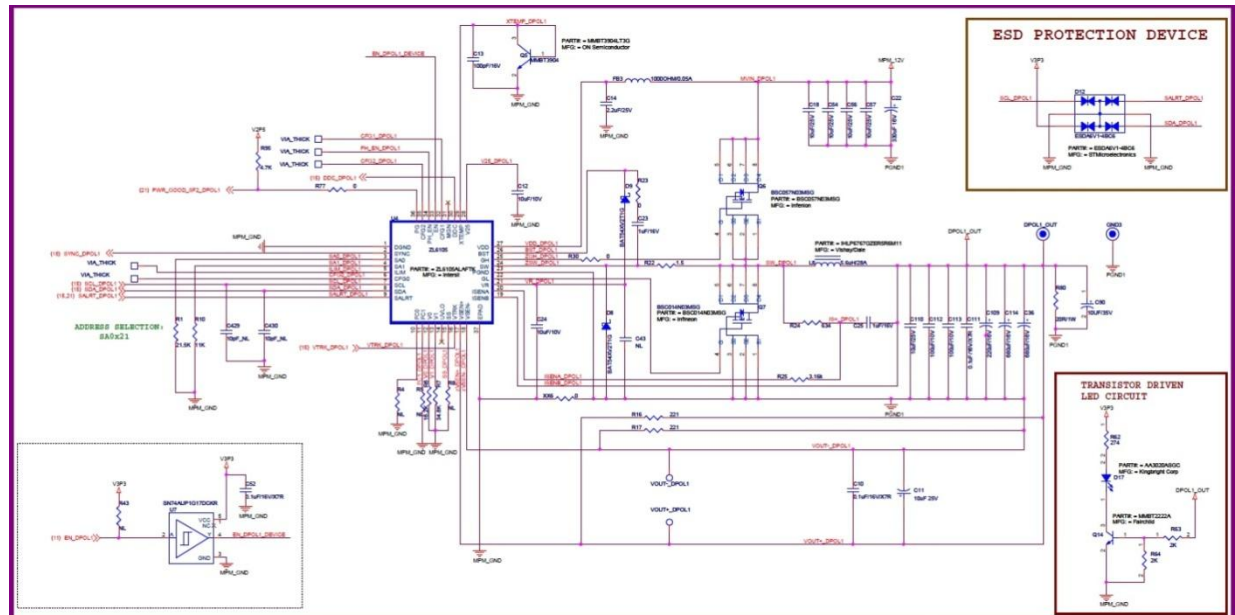


Figure 51 • DPOL1 ZL6105 Regulator

Installation and Switch Settings

Recommended default jumpers settings are show in Table 18. Connect jumpers in the default settings to enable the pre-programmed reference design to function correctly. Note that the user must program the Digital Points of Load (DPOL) regulators to the default output voltages and settings defined in "Digital Mixed Signal Power Manager Board Components" section, and in the MPM GUI's default settings.

Table 19 · Jumper Settings (DMPM-DB)

Jumper	Function	Default Setting
J22	Connects DPOL1 to DPOL8 to PMBus (PMBus1 to PMBus2)	All DPOL's connected to PMBus1, pins shorted: A-B, C-D, E-F, G-H, I-J, K-L, M-N
JP3, JP26, JP27, JP28, JP2, JP29, JP30 and JP31	Connects APOL1 to APOL8 trim pin for closed-loop trimming.	Closed
JP21, JP22, JP32 and JP33	Connects DPOL1 to DPOL4 enable pin to FPGA or to master enable	Pins 1-2 shorted
JP23	Connects the 3.3V rail on the SF2-DMPM-DB to either the FMC connector or the Zilker programming header.	Pins 1-2 shorted
JP25	Connects the 2.5V rail on the SF2-DMPM-DB to the FMC Connector	Closed

Table 20 · Switches (DMPM-DB)

Switch	Description
SW8	Push-button to disable APOL1
SW13	Push-button to disable APOL2
SW14	Push-button to disable APOL3
SW15	Push-button to disable APOL4
SW9	Push-button to disable APOL5
SW16	Push-button to disable APOL6
SW17	Push-button to disable APOL7
SW18	Push-button to disable APOL8
SW10	Push-button to disable DPOL1
SW12	Push-button to disable DPOL2

SW19	Push-button to disable DPOL3
SW20	Push-button to disable DPOL4
SW11	Push-button to disable DPOL5
SW21	Push-button to disable DPOL6
SW22	Push-button to disable DPOL7
SW23	Push-button to disable DPOL8
SW2	Switch ON 12 V DC into MPM

Note: The push-buttons detailed above ground the Enable pin of the corresponding regulator, thereby injecting failure in power subsystem. The Enable pin is also connected to the pin on the FPGA. The FPGA could be damaged if it is driving HIGH on the enable line and simultaneously shorted to ground. To avoid this, Microsemi recommends that the FPGA pin driving these enables should be either driving Low or Tristate. Please refer to the Libero design for details on how to do this.

Manufacturing Information

MPM DB Board

The full PCB design layout is provided on the SF2 DMPM-DC Kit webpage. To view the PCB design layout files, you can use Allegro Free Physical Viewer, which can be downloaded from the Cadence Allegro Download page.

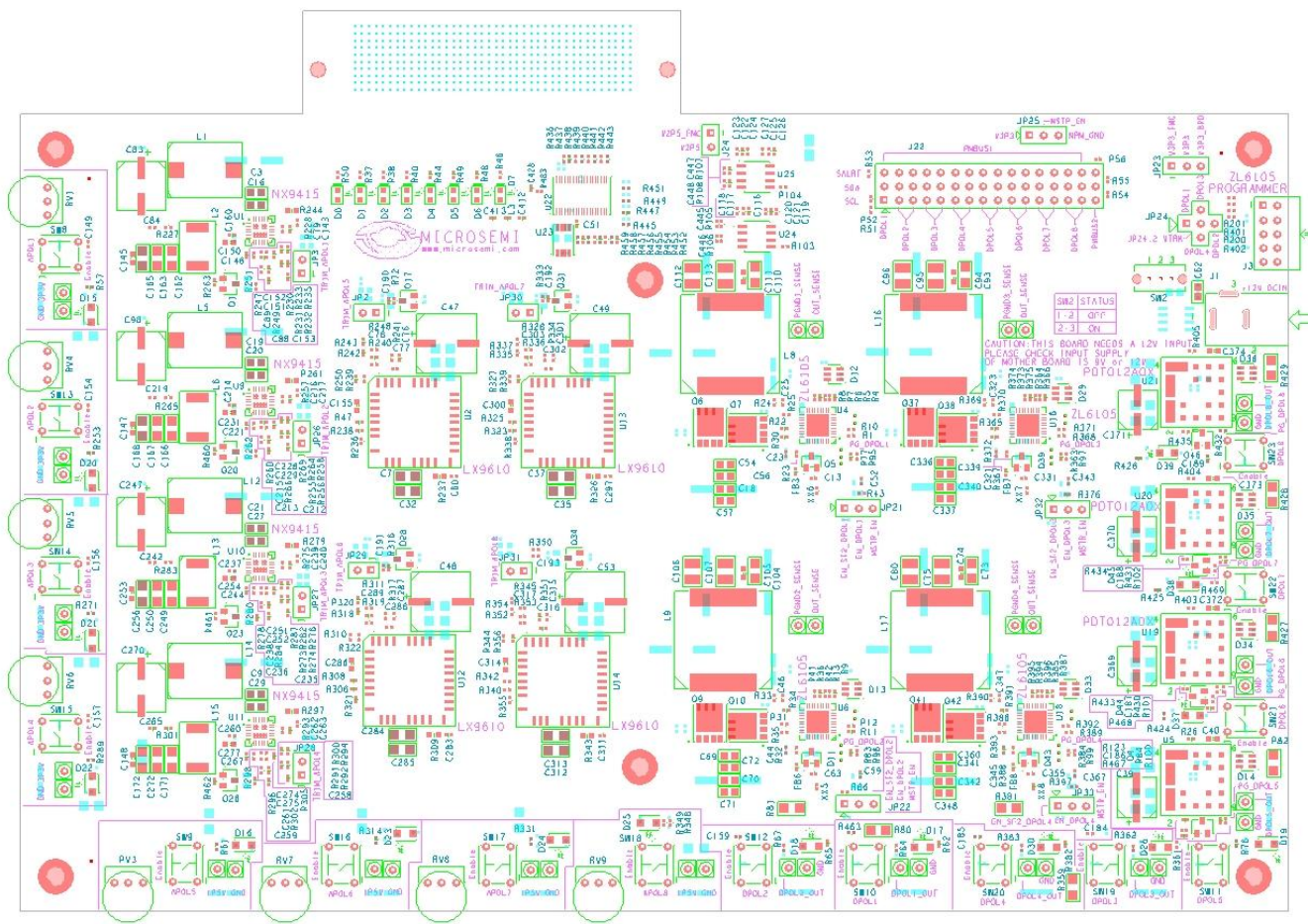


Figure 54 · Top Silk Screen for SF2 DMPM Daughter Board Rev A

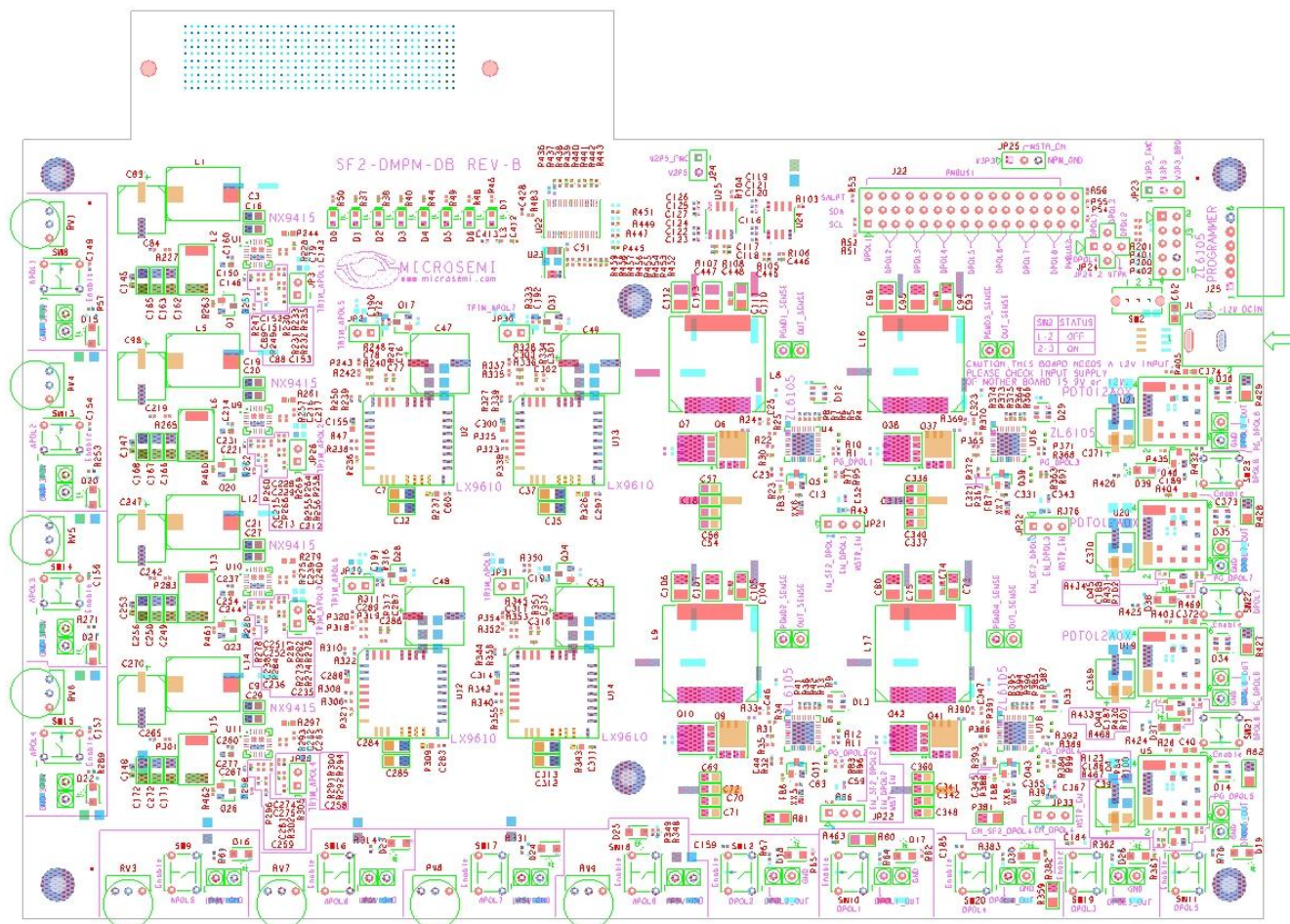


Figure 55 · Top Silk Screen for SF2 DPM Daughter Board Rev B

Product Support

Microsemi SoC Products Group backs its products with various support services, including Customer Service, Customer Technical Support Center, a website, electronic mail, and worldwide sales offices. This appendix contains information about contacting Microsemi SoC Products Group and using these support services.

Customer Service

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

From North America, call **800.262.1060**

From the rest of the world, call **650.318.4460**

Fax, from anywhere in the world **408.643.6913**

Customer Technical Support Center

Microsemi SoC Products Group staffs its Customer Technical Support Center with highly skilled engineers who can help answer your hardware, software, and design questions about Microsemi SoC Products. The Customer Technical Support Center spends a great deal of time creating application notes, answers to common design cycle questions, documentation of known issues and various FAQs. So, before you contact us, please visit our online resources. It is very likely we have already answered your questions.

Technical Support

Visit the Microsemi SoC Products Group Customer Support website for more information and support (<http://www.microsemi.com/soc/support/search/default.aspx>). Many answers available on the searchable web resource include diagrams, illustrations, and links to other resources on website.

Website

You can browse a variety of technical and non-technical information on the Microsemi SoC Products Group home page, at <http://www.microsemi.com/soc/>.

Contacting the Customer Technical Support Center

Highly skilled engineers staff the Technical Support Center. The Technical Support Center can be contacted by email or through the Microsemi SoC Products Group website.

Email

You can communicate your technical questions to our email address and receive answers back by email, fax, or phone. Also, if you have design problems, you can email your design files to receive assistance. We constantly monitor the email account throughout the day. When sending your request to us, please be sure to include your full name, company name, and your contact information for efficient processing of your request.

The technical support email address is soc_tech@microsemi.com.

My Cases

Microsemi SoC Products Group customers may submit and track technical cases online by going to [My Cases](#).

Outside the U.S.

Customers needing assistance outside the US time zones can either contact technical support via email (soc_tech@microsemi.com) or contact a local sales office. [Sales office listings](#) can be found at www.microsemi.com/soc/company/contact/default.aspx.

ITAR Technical Support

For technical support on RH and RT FPGAs that are regulated by International Traffic in Arms Regulations (ITAR), contact us via soc_tech_itar@microsemi.com. Alternatively, within [My Cases](#), select **Yes** in the ITAR drop-down list. For a complete list of ITAR-regulated Microsemi FPGAs, visit the [ITAR](#) web page.

List of Changes

List of Changes

The following table lists critical changes that were made in each revision of the chapter.

Date	Changes	Page
xxxxxxxx-1/04.13	Initial revision	All

**The part number is located on the last page of the document. The digits following the slash indicate the month and year of publication.*



Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo CA 92656 USA
Within the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996

Microsemi Corporation (NASDAQ: MSCC) offers a comprehensive portfolio of semiconductor solutions for: aerospace, defense and security; enterprise and communications; and industrial and alternative energy markets. Products include high-performance, high-reliability analog and RF devices, mixed signal and RF integrated circuits, customizable SoCs, FPGAs, and complete subsystems. Microsemi is headquartered in Aliso Viejo, Calif. Learn more at www.microsemi.com.

© 2013 Microsemi Corporation. All rights reserved. Microsemi and the Microsemi logo are trademarks of Microsemi Corporation. All other trademarks and service marks are the property of their respective owners.