AHEAD OF WHAT'S POSSIBLE™

MAX96793 Single CSI-2 to GMSL3 Serializer User Guide

Authored by: GMSL SerDes Applications Team

UG-2232

Rev 0; 5/24

Device Overview

This user guide is intended to be used in conjunction with other documents such as the MAX96793 data sheets, errata documents, and other user and design guides. It provides explanations, examples, and instructions to help set up video configurations and use various features.

Examples may be shown without errata writes that are necessary to ensure reliable operation in production. Contact the Analog Devices, Inc. field applications engineer or representative to obtain the errata documents. The user should include any relevant errata writes in the final production software. In addition to the errata, it is also important to have the latest revision of the GMSL device for testing.

The GMSL3 serial links use packet-based, bidirectional architecture with forward and reverse channels. The forward channel transfers data from the serializer to the deserializer; the reverse channel transfers data from the deserializer to the serializer. The GMSL3 devices have a forward channel serial bit rate of 3Gbps, 6Gbps, or 12Gbps and reverse channel serial bit rate of 187.5Mbps. See *Table 1* for supported data rates/features by part number.

The MAX96793 is a full featured GMSL3/2 serializer device. The MAX96793 is capable of 3Gbps, 6Gbps, or 12Gbps forward link rate (selectable with resistors connected to the CFG pin or with register writes) with a 187.5Mbps reverse direction rate.

Table 1. MAX96793 High-Level Features

Part Number	Forward Link Rate	Coax/STP	MIPI Input	GMSL Links	ASIL Rating
MAX96793	3Gbps, 6Gbps, or 12Gbps	Coax or STP	DPHY or CPHY	1	B

Note: This is not a complete list of device differences. See the device data sheets for all feature details.

Application Use Case

In a typical configuration, the MAX96793 serializer accepts mobile industry processor interface (MIPI) CSI-2 D-PHY or C-PHY video data from an image sensor in the 8MP to 17MP range, see *Figure 1*. The MAX96793 serializer then takes that data, converts it to GMSL, and sends it out over the link to a GMSL3 deserializer. In *Figure 1*, there are two MAX96793s operating independently and sending data out to a MAX96792A deserializer. Coax or shielded twisted-pair (STP) cables can be used for the GMSL link.

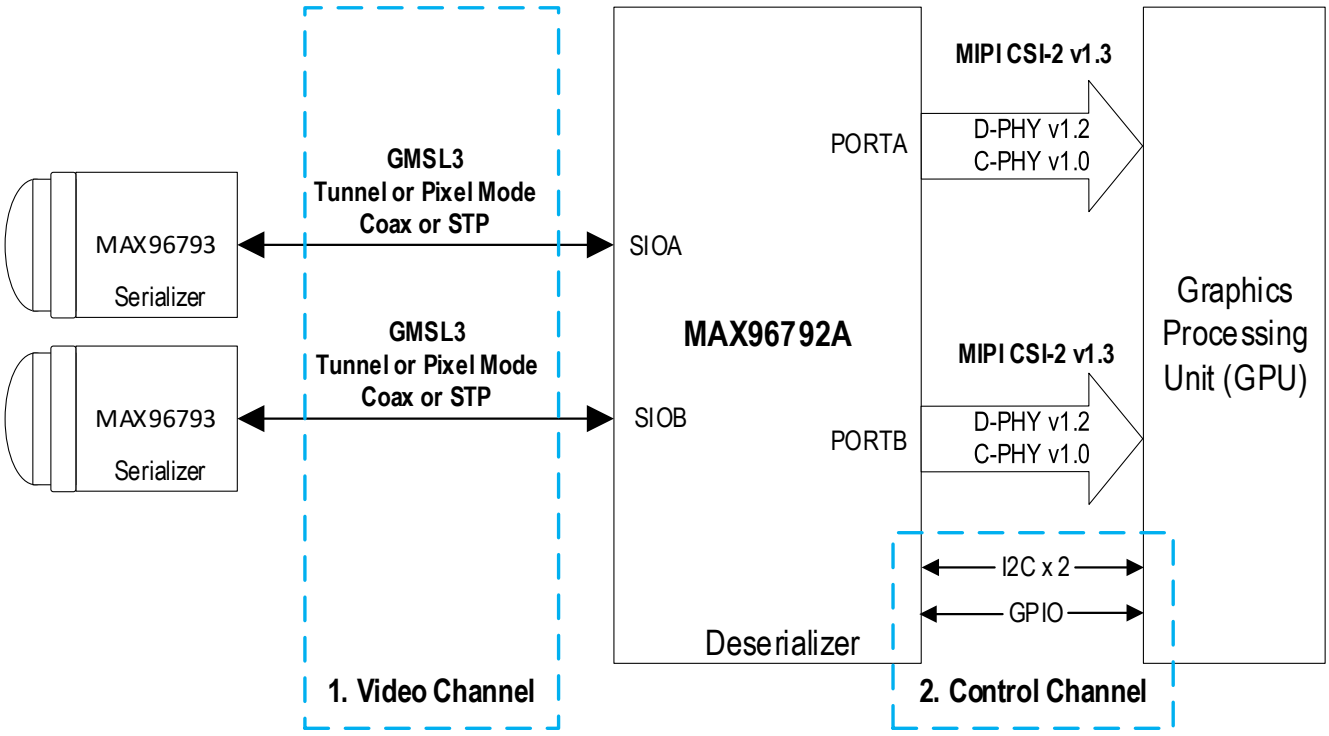

Figure 1. MAX96793 Two Camera Application Example

Architecture

The MAX96793 video path starts with one MIPI PHY that connects to a CSI-2 controller which then routes to internal Pipe Z and ends with a GMSL PHY. The default video path settings are shown in *Figure 2*.

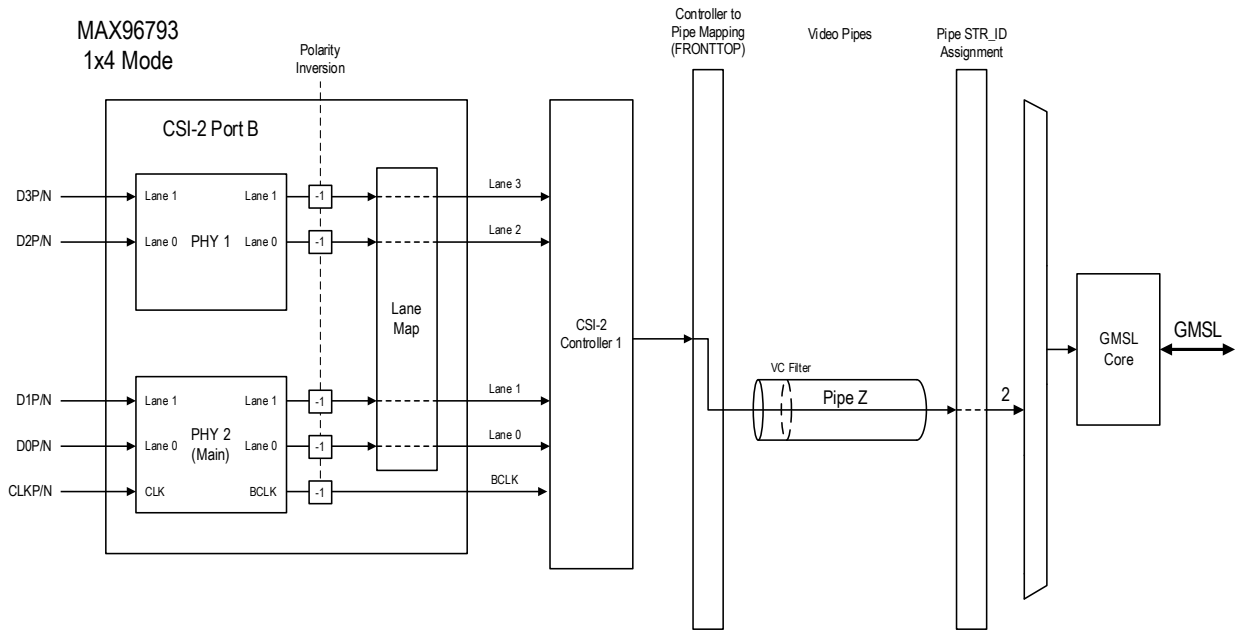

Figure 2.MAX96793 Video Path (1x4 D-PHY Mode)

Startup and Programming Sequence

The GMSL3 SerDes devices have many application use cases and features that work in conjunction with each other. To avoid feature and system sequencing issues, *Table 2* outlines the preferred sequence order. Features or configuration changes may not be required and may be skipped in the start-up sequence. This depends on system requirements and data configurations.

Table 2. GMSL3 SerDes Start-Up Sequence

GMSL3 Start-Up and Programming Sequence		
Sequence Number	**Configuration Setup**	**Notes**
0	Ramp-Up Voltage Power Supply	-No power supply voltage sequencing required; voltage rails are internally independent and managed by on-chip power management block.
1	Release PWNDB Pin (L→H) (If necessary)	
	I^2C Wake-Up Time	Time from power-up or rising edge of PWDNB for local register access. For remote register access, I^2C wakeup is the same as GMSL Link Lock time.
2	CFG Pins Automatically Set Link	-CFG pins sampled on every power-on reset (POR) and/or PWDNB L→H transition
3	Link Configuration (Single Link vs. Multilink Operation Setup)	Some deserializers power up in single-link mode while others in multilink modes. Read register map for correct registers to set up correct link operation mode. See *Multilink Operation* section for more information.
4	GMSL Link Lock is Established	-If GMSL Link Lock is not established, verify the following: 1) Voltage rails are correct per DS specification. 2) Datarate, Coax/STP mode, and GMSL settings match between serializer and deserializer.
5	I^2C Rate Adjustment (If necessary)	SerDes has I^2C rate register settings that need to match up to I^2C main.
6	SER I^2C Device Address Reassignment (If necessary)	Reassigning SER I^2C device address can help in multicamera systems
7	Disable DES CSI Output	-Set register bitfield CSI_OUT_EN=0
8	DES Errata Settings (If necessary)	-Ensure errata settings match DEV_REV and use case
9	DES MIPI TX Configuration	-MIPI Port Config -Lane Count -Lane Mapping/Polarity Swap -Pipe to Controller Mapping -Deskew (>1.5Gbps/Lane) -MIPI Data Rate
10	DES GPIO and Other Feature Configuration	-GPIO Forwarding -FSYNC

		-I²C/UART Pass-through Channels -Line Fault
11	DES Interrupt Handling (ERRB) and ASIL Configuration	-See *Error Flags* section and Safety Documents of the Deserializer for more information
12	SER Errata Settings (If necessary)	-Ensure Errata settings match DEV_REV and use case
13	SER MIPI RX Configuration	-Lane Count -Lane Mapping/Polarity Swap -Pipe to Controller Mapping -Deskew (>1.5Gbps/Lane) -MIPI Continuous vs. Noncontinuous Mode
14	SER GPIO and Other Feature Configuration	-GPIO Forwarding -FSYNC -I2C/UART Pass-through Channels -Reference Clock Out -ADC -Line Fault
15	SER Interrupt Handling (ERRB) and ASIL Configuration	-See *Error Flags* section and Safety Documents of the Serializer for more information.
16	RESET LINK=1	-Reset whole data path to allow configuration and errata settings to take effect. (While this bit is '1' remote access to link is not possible.)
17	RESET LINK=0	-Release of reset, link relocks. Remote access is possible after link is locked.
18	DES Enable CSI Output	-Set register bitfield CSI_OUT_EN=1
19	Enable Deserializer Register CRC Safety Mechanism (If necessary)	Review Safety documents of the Deserializer for more information. See *Register CRC* section for more information.
20	Enable Serializer Register CRC Safety Mechanism (If necessary)	Review Safety documents of the Serializer for more information. See *Register CRC* section for more information.
21	Start Video Source	

Note(s):

- Perform any configuration changes before video starts.
- If changes are needed after video has started; stop the video, then make changes, and restart the video.
- Dynamic configuration is not supported.

Video Configuration

Overview

The forward video paths of the MAX96793 serializers are configured with the following programming:

- Pixel and Tunneling Mode
- Link Initialization
- Link Lock Check
- MIPI Controller and PHY Settings
- Datatype (DT)/Virtual Channel (VC) Filtering and Overrides

Enable the video only after the video path is configured; dynamic configuration is not supported. The following subsections detail the operation of each of these steps with descriptions of relevant registers and programming examples.

Pixel and Tunneling Mode

The MAX96793 supports both pixel and tunneling modes. Always ensure that both serializer and deserializer are in pixel or tunneling mode.

Pixel mode provides the ability for systems to manipulate data types, bits per pixel (BPP), and virtual channels. This mode can be used when the incoming data must be manipulated over the serial link before outputting from the deserializer.

Tunneling mode can be used when data integrity is a major system concern as it ensures end-to-end data integrity. End-to-end data protection is a common requirement for Advanced Driver Assistance Systems (ADAS), where data may not be altered from the transmitter to the downstream receiver. In tunneling mode, data may not be changed as it is protected with an end-to-end cyclic redundancy check (CRC) and is passed from serializer to deserializer without any manipulation. In tunneling mode, any combination of data type, BPP, and virtual channel may be transmitted if the video bandwidth total does not exceed the link bandwidth.

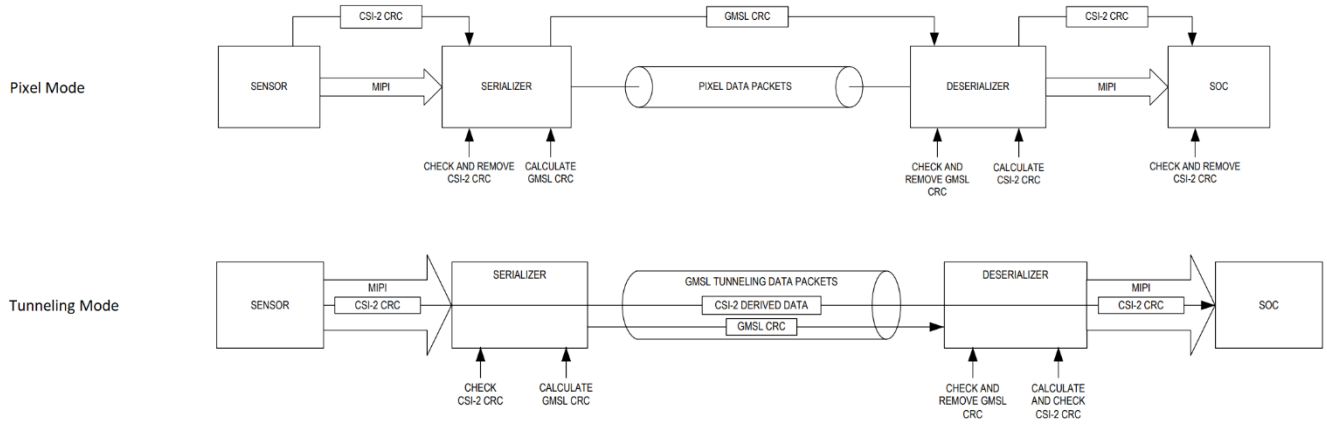

Figure 3. Pixel and Tunneling Mode Comparison

The following tables show MAX96793 registers to enable pixel or tunneling mode and the mode differences.

Table 3. MAX96793 Pixel and Tunneling Mode Register Settings

Register Address	Bitfield Name	Bits	POR	Decode
0x383	TUN_EN	7	0b0	0b0: Pixel Mode 0b1: Tunneling Mode

Table 4. MAX96793 Features Supported by Pixel Mode vs. Tunneling Mode

Feature	Pixel Mode	Tunneling Mode
Video Processing (Watermarking)*	Supported	Not Supported
Virtual Channel Reassignment	Supported	Not Supported
Synchronous Aggregation	Not Supported	Not Supported
First Come First Serve Aggregation	Supported	Supported**
Compatibility with Legacy GMSL2 Pixel Parts	Supported	Not Supported
D-PHY to C-PHY Translation	Supported	Not Supported
C-PHY to D-PHY Translation	Supported	Not Supported
Mixed Mode (One GMSL Link is C-PHY and other GMSL Link is D-PHY)	Supported	Supported
End-to-End CRC Coverage for Video	Not Supported	Supported
Video Line CRC (LCRC)	Supported	Supported
GMSL Packet CRC (VID_PXL_CRC)	Supported	Supported
Line Length >8k pixels at 24BPP	Not Supported	Supported
16-Channel Virtual Channel Support	Supported	Supported

*Check device data sheet to verify watermarking is supported.

**Video source needs to set different VCs, as VC reassignment is not supported in tunneling mode.

Link Initialization

Link initialization establishes the device link modes and speeds. The MAX96793 device family is a GMSL3 Single CSI-2 to GMSL3 Serializer that can support coax or STP cables. Using the following registers, select the GMSL link rate and coax or STP connectivity. Any changes to the GMSL link should be followed by a link reset to reinitialize the link (toggle RESET_LINK=1 and then RESET_LINK=0). The CFG pins are the preferred method of setting up the GMSL rate and transmission mode. The selected configuration becomes the new default on power-up once the CFG pins are set and the part is power cycled. The following tables show capabilities of MAX96793 devices and its link initialization registers.

Table 5. MAX96793 Basic Settings (CFG1 Pin)

CFG1 Value	Coax/STP	Data Rate	Transmission Mode
0	STP	6Gbps	Tunneling Mode
1	STP	12Gbps	Tunneling Mode
2	STP	3Gbps	Pixel Mode
3	STP	6Gbps	Pixel Mode
4	Coax	6Gbps	Tunneling Mode
5	Coax	12Gbps	Tunneling Mode
6	Coax	3Gbps	Pixel Mode
7	Coax	6Gbps	Pixel Mode

Table 6. MAX96793 Link Initialization Registers

Register Address	Bitfield Name	Bits	POR	Decode
0x0001	TX_RATE	3:2	0b10	0b00: RSVD 0b01: 3Gbps 0b10: 6Gbps 0b11: 12Gbps
0x0011	CXTP_A	0	0b1	0b0: Shielded twisted pair drive 0b1: Coax drive
0x0010	RESET_ALL	7	0b0	0b0: No action 0b1: Activate chip rest
0x0010	RESET_LINK	6	0b0	0b0: Release link A reset 0b1: Activate link A reset
0x0010	RESET_ONESHOT	5	0b0	0b0: No action 0b1: Reset data path

Note: A link reset on CSI-2 serializers resets the entire data path of any video connected to the PHY. Link resets should not be used when video is being fed to the device. Doing this can corrupt data and have unintended consequences.

Multilink Operation

The MAX96793 has only one GMSL link so it does not have multilink operation. Although multicamera systems can be created with multiple MAX96793 and single, dual, and quad CSI-2 deserializers.

Link Lock Check

If the MAX96793 device configuration is correct, the link automatically locks upon connection.

Pin #17 (MFP3) can be used as LOCK indication if enabled by register write. Bit 3 in register 0x0013 asserts if link is locked.

Video Pipe Selection

Video pipes must be configured to match video streams between the deserializer and serializer. This programming step is typically done following link initialization and ensures that the deserializer properly receives video data from the serializer. By default, the deserializer is programmed to accept the most common stream from the serializers and configuration is not usually needed.

The MAX96793 has one video pipe (video pipe Z) that can select one of the four stream IDs. By default, the MAX96793 pipe Z stream ID is set to 0b10.

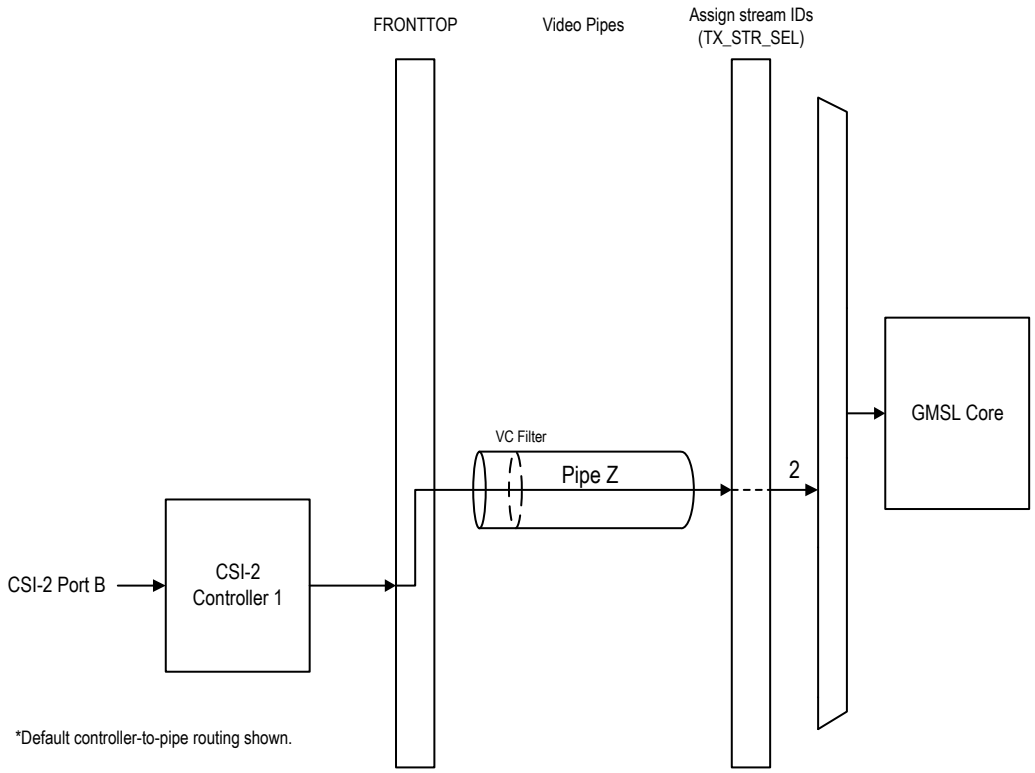

Figure 4. MAX96793 Video Pipe Path

Video Pipe Selection Registers

Select the serializer stream ID to match the video streams (STR_ID) from the serializers for each deserializer video pipe. The GMSL3 camera serializers can have up to four video pipes (X, Y, Z, and U). These are annotated as 2 bits representing the stream ID. Pipe X = 0b00, Pipe Y = 0b01, Pipe Z = 0b10, and Pipe U = 0b11.

By default, the MAX96793 serializer selects stream ID 0b10, which is the default for most CSI-2 deserializers.

Table 7. MAX96793 Video Pipe Selection Registers

Register Address	Bitfield Name	Bits	POR	Decode
0x5B	TX_STR_SEL (Pipe Z)	00	0b10	0b00: str_id=00 0b01: str_id=01 0b10: str_id=10 0b11: str_id=11

Video Pipe to MIPI Controller Mapping (VC/DT Mapping and Filtering)

As data comes in through the MIPI PHY on serializer input, the pixel data goes to a CSI-2 controller then is routed to internal video pipes. Video pipes are then mapped to a GMSL PHY that is sent over the GMSL link. Pixel data within video pipes can be filtered by DT or VC.

The MAX96793 has only one CSI-2 input port (B) and one video pipe (Z). This means the video routing is much simpler since there is only one path for video data to flow. Register 0x0308, shown in the Video Pipe and Filtering Registers table (see *Table 8*), contains the bits for enabling/disabling MIPI port B.

Default Mapping:

MIPI PHY1/2 → MIPI Controller 1 → Video Pipe Z

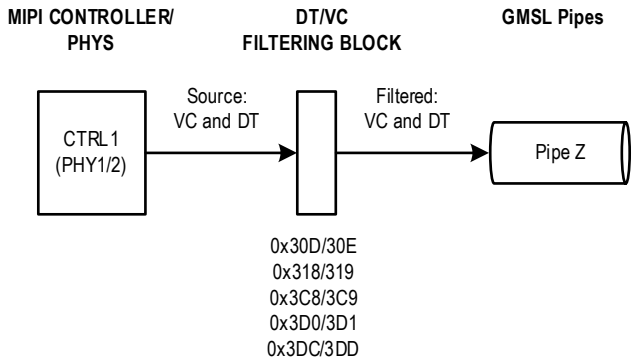

Figure 5. MAX96793 MIPI Controller to Video Pipe Block Diagram

Using the default video routing, any data received on the MIPI input port B is automatically routed onto video pipe Z, unless filtering by DT or VC is being used. Every CSI-2 packet includes a header that indicates the DT and VC. If pixel mode is being used, this information can be used to route the incoming data throughout the serial link system. The DT and VC filtering is not supported in tunneling mode.

The data within the serializer's controller can be filtered so that the user can control what data, if not all, gets serialized and sent across the GMSL link. The mem_dt_selz registers are used to filter the controller data by CSI-2 datatype code at the FRONTTOP before it reaches the video pipe. When using DT filtering, up to four datatypes can be routed from the controller to pipe Z. The pixel datatype codes to be routed must be set in mem_dt{1,2,7,8}_selz. Bits [5:0] in these registers must match the incoming datatype code. Bit 6 enables the filter.

Another form of filtering is by VC which is configured with the VC_SELZ bitfield. In pixel mode, when multiple data streams are transmitted over the same pipe with different virtual channels, the VC_SELZ_L and VC_SELZ_H bits must be set to represent the virtual channels present on that pipe. Each bit place represents a VC within these registers. For example, if VC_SELZ_L[0] = 1 and VC_SELZ_L[1] = 1, then pipe Z would expect to have VC 0 and 1 on the pipe. When a bit position is set to zero, that VC is not allowed to enter that pipe. The VC_SELZ_L/H registers on this part have a default value of 0xFF, meaning that all 16 VCs (0-15) are allowed onto the pipe unless programmed otherwise.

Table 8. MAX96793 Video Pipe and Filtering Registers

Register	Bitfield Name	Bits	Default Value	Decode
0x0308	START_PORTB	5	0b1	0 = CSI-2 on port B disabled 1 = CSI-2 on port B enabled
0x0308	CLK_SELZ	2	0b1	0 = Reserved (Port A does not exist) 1 = Port B selected for pipe Z
0x0002	VID_TX_EN_Z	6	0b1	0 = Video transmit on pipe Z disabled

				1 = Video transmit on pipe Z enabled
0x005B	TX_STR_SEL[1:0]	1:0	0b10	00 = Stream ID for pipe Z is 0 01 = Stream ID for pipe Z is 1 10 = Stream ID for pipe Z is 2 11 = Stream ID for pipe Z is 3
0x0318, 0x0319, 0x03DC, 0x03DD	mem_dt{1,2,7,8}_selz[6]	6	0b0	0 = Datatype filtering disabled 1 = Datatype filtering enabled
0x0318, 0x0319, 0x03DC, 0x03DD	mem_dt{1,2,7,8}_selz[5:0]	5:0	0b000000	The value of bits 5:0 in this register should equal the datatype ID of the datatype the user wishes to allow onto the video pipe (e.g., RAW12 = 0x2C).
0x03C8, 0x03C9	mem_dt{3,4}_selz[7:6]	7:6	0b00	These two bits select the two LSBs of the virtual channel that is to be filtered onto the video pipe
0x03C8, 0x03C9	mem_dt{3,4}_selz[5:0]	5:0	0b000000	The value of bits 5:0 in this register should equal the datatype ID of the datatype the user wishes to allow onto the video pipe.
0x03D1	mem_dt{3,4}_selz_en	1:0	0b00	0 = Disable filtering set in registers 0x3C8, 0x3C9 1 = Enable filtering set in registers 0x3C8, 0x3C9
0x030D	VC_SELZ_L	7:0	0xFF	Bits 0-7 represent VC0-VC7, respectively. If a bit is high, it means that VC is allowed onto the video pipe (e.g., if only bits 0 and 2 are HIGH, then only VC0 and VC2 are accepted).
0x030E	VC_SELZ_H	7:0	0xFF	This register works the same as register 0x30D except bits 0-7 represent VC8-VC15, respectively

Video Pipe to Controller Routing Example
This example filters video pipe Z for RAW12 datatype and virtual channel 1 on the MAX96793.

0x80, 0x0318, 0x6C, #enable DT filter for RAW12 (ID = 0x2C)

0x80, 0x030D, 0x02, #only VC1 allowed onto pipe

Video Lock Check

In pixel mode, the MAX96793 register 0x112 bit 7 (PCLKDET) asserts if it is receiving valid MIPI long packets (pixel data) and GMSL Link LOCK=1.

In tunneling mode, the MAX96793 register 0x112 bit 7 (PCLKDET) asserts if it is receiving valid CSI-2 clock and GMSL Link LOCK=1.

DT/VC Software Override

The software override manually overrides the video DT (i.e., packet header), VC number, or BPP. This operation affects the video data between the video pipe(s) and the MIPI controller(s). Overriding the DT and VC information is used for MIPI controller mapping. If the received video data is from a serializer in parallel mode (e.g., GMSL1 serializers), it is necessary to specify the desired DT, VC, and BPP with the software override.

Note: The VC can be changed individually. However, DT and BPP must be adjusted together to ensure settings compatibility.

Input DT BPP Manipulation (Pixel Mode Only)

One advantage of pixel mode is that the data can be manipulated, such as by doubling or zero padding. Doubling the BPP of a data type allows for more efficient bandwidth usage. Zero padding is used to match the BPP of two or more data types so that they can share a video pipe.

Double Mode

Double mode is a data arrangement available for data types with BPP = 8, 10, or 12. With double mode enabled, two input pixels are concatenated and processed as a single pixel within the video pipe. This concatenation reduces the internal PCLK and increases the GMSL3 bandwidth efficiency. Double mode is enabled on a BPP basis.

Further, user-defined 8-bit data types (UDP or UDT), which have header codes 0x30, 0x31–0x37, or 0x10–0x11 can alternatively be combined and transmitted by the serializer as 24-bit data. Set ctrl1_mode_UDT = 1 to treat these data types as 24BPP. This mode cannot be used simultaneously while bpp8dblz = 1, and tripled data types can only share a pipe with data types that use 24BPP or other tripled 8-bit data types.

When using double or triple mode, the new internal BPP must be programmed into the serializer in addition to enabling the mode. Video pipe Z has a soft_bppz bitfield that must be set to the new BPP (e.g., 8→16, 8→24, 10→20, 12→24) and a soft_bpp_en bit.

Table 9. MAX96793 Double Mode Registers

Register	Bitfield Name	Bits	Default Value	Decode
0x312	bpp8dblz	2	0b0	0: Send as 8-bit pixels 1: Send 8-bit pixels as 16-bit pixels
0x313	bpp10dblz	2	0b0	0: Send as 10-bit pixels 1: Send 10-bit pixels as 20-bit pixels
0x313	bpp12dbl{Z}	6	0b0	0: Send as 12-bit pixels 1: Send 12-bit pixels as 24-bit pixels
0x337	ctrl1_mode_UDT	5	0b0	0: Treat UDP as 8 bits 1: Treat UDP as 24 bits

Zero Padding

Pixel data being received by the MAX96793 can be zero padded as it enters a video pipe up to a resulting BPP of 16. With zero padding, an input with multiple BPP rates can be routed through its video pipe if the following conditions are met:

1. 8 ≤ bpp ≤ 16 for all incoming bpp rates that are routed to the video pipe
    a. Zero padding occurs after doubling. The 8 ≤ bpp ≤ 16 requirement applies to the resulting bpp after doubling.
2. Bandwidth is lost proportionally to the amount of zero padding. Some amount of GMSL3 bandwidth is dedicated to sending zeros instead of the original CSI-2 data. Ensure system bandwidth requirements can be met by using the calculations shown in the User Guide *Bandwidth Efficiency Optimization* section.
3. Video pipe Z's PCLK Drift detection must be disabled.

Zero padding applies to all data being routed in the pipe. When enabled, the pipe PCLK is set to the fastest incoming PCLK (smallest BPP) and all data within the pipe is treated as having a pixel width set by the BPP bitfield.

To enable zero padding, set AUTO_BPP = 0, BPP = largest bpp in the pipe (≤16), soft_bpp = smallest bpp in the pipe, and soft_bpp_en = b1. PCLK drift detection must also be disabled using the pipe's DRIFT_DET_EN bit. Using this method, all incoming data types with a BPP < BPP (Register) are zero padded so that all BPP rates within the pipe are equal. See *Table 10* for zero padding registers.

The zero-padded data is automatically recovered correctly on the deserializer based on the DT information that is automatically transmitted to the deserializer. But any DT that were "doubled" in the serializer must be "undoubled" in the deserializer.

Table 10. MAX96793 Zero Padding Registers

Register	Bitfield Name	Bits	Default Value	Decode
0x110	AUTO_BPP	3	0b1	0: Use BPP from BPP register 1: Use BPP from MIPI receiver
0x111	BPP	5:0	0b011000	Number of bits per pixel (AUTO_BPP must = 0)
0x112	DRIFT_DET_EN	1	0b1	Enables PCLK frequency drift detection, resets video pipeline upon error and reports it

Double Mode and Zero Padding Example

EMB8, RAW12, and RAW16 share Pipe Z. EMB8 is doubled to 16BPP. RAW12 is zero padded to 16BPP. RAW16 is unmodified. All data types are 16BPP inside of the pipe. EMB8 is doubled rather than zero padded because doubling is more efficient than zero padding, and EMB8 (DBL) has a BPP equal to the largest BPP in the pipe (RAW16).

1. AUTO_BPP = 0 – Do not set BPP based on CSI-2 header (Pipe Z).
2. BPP = 0x10 – Force Pipe Z BPP to 16 by zero-padding.
3. soft_bppz = 0x0C – This must be set to the smallest input BPP (before padding and after doubling).
4. soft_bppz_en = 1 – This enables software override of BPP.
5. bpp8dblz = 1 – This doubles all incoming BPP = 8 DTs.
6. DRIFT_DET_EN = 0 – PCLK frequency drift detection is disabled for Pipe Z.

Software Override

The data type, virtual channel, and bpp overrides must be enabled to take effect. The following table are the required registers on the MAX96793.

Table 11. MAX96793 Software Override Registers

Register	Bitfield Name	Bits	Default Value	Decode
0x031E	soft_dtz_en	7	0b0	0 = Data type software override disabled on pipe Z 1 = Data type software override enabled on pipe Z
0x031E	soft_vcz_en	6	0b0	0 = Virtual channel software override disabled on pipe Z 1 = Virtual channel software override enabled on pipe Z
0x031E	soft_bppz_en	5	0b0	0 = BPP software override disabled on pipe Z 1 = BPP software override enabled on pipe Z
0x031E	soft_bppz[4:0]	4:0	0b11000	These bits should be set to the smallest input BPP (before padding and after doubling).
0x0320	soft_vcz[1:0]	5:4	0b00	00 = VC0

				01 = VC1 10 = VC2 11 = VC3
0x0323	soft_dtz[5:0]	5:0	0b110000	These bits should be set to the appropriate data type ID.

Software Override Programming Examples

Some examples of software override settings are as follows:

• DT: soft_dtz[5:0]

DT = 0x24 = 0b100100 for RGB888

• VC: soft_vcz[1:0]

VC = 0x03 = 0b11 for VC3

• BPP: soft_bppz[4:0]

BPP = 0xC = 0b01100 for RAW12

Extended Virtual Channels

Virtual channels allow the serial link system to differentiate video inputs by the VC assigned to them. When extended VCs are enabled, the standard 2-bit VC selection is extended to 4-bit (D-PHY), increasing the number of available VCs to 16 for D-PHY applications. The increase in available VCs allows systems to support more camera inputs.

Extended VCs are enabled by the ctrl1_vcx_en bit. Once this feature is enabled, VCs 0-16 can be accommodated on the serializer input. If this bit is disabled, the VC is limited to the least significant 2 bits. Virtual channel remapping is available if the incoming video streams cannot be changed at the video source.

The VC remapping is typically done at the deserializer but can be remapped on the serializer using ctrl1_vc_map_en and ctrl1_vc_map0-15 bit fields. When the mapping is enabled, each ctrl1_vc_map field remaps its respective VC. For example, when ctrl1_vc_map0 bit field is set to 0101b, then the input VC0 would remap to VC5. Similarly, ctrl1_vc_map1 would allow for the remapping of input VC1.

Table 12. MAX96793 New Virtual Channel Mapping (Extended VC)

CTRL1_VC_Map 0-15						
VC Input	**Ctrl1_vc_map**	**VC Remap**	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**
0	ctrl1_vc_map0	VC = 5	0	1	0	1
1	ctrl1_vc_map1	VC = 4	0	1	0	0
3	ctrl1_vc_map3	VC = 2	0	0	1	0

Table 13. MAX96793 Extended Virtual Channel Registers

Register	Bits	Default Value	Description
0x0330	7	0	**Ctrl1_vcx_en:** 0 = VC Extension Disabled 1 = VC Extension Enabled
0x0331	5	0	**Ctrl1_vc_map_en:** 0 = VC Remapping Disabled

			1 = VC Remapping Enabled
0x0345	7:4	0x00	**Ctrl1_vc_map0:** Bits [7:4]: New Virtual Channel for VC0
0x0346	7:4	0x00	**Ctrl1_vc_map1:** Bits [7:4]: New Virtual Channel for VC1
0x0347	7:4	0x00	**Ctrl1_vc_map2:** Bits [7:4]: New Virtual Channel for VC2
0x036C	7:4	0x00	**Ctrl1_vc_map3:** Bits [7:4]: New Virtual Channel for VC3
0x036D	7:4	0x00	**Ctrl1_vc_map4:** Bits [7:4]: New Virtual Channel for VC4
0x036E	7:4	0x00	**Ctrl1_vc_map5:** Bits [7:4]: New Virtual Channel for VC5
0x036F	7:4	0x00	**Ctrl1_vc_map6:** Bits [7:4]: New Virtual Channel for VC6
0x0377	7:4	0x00	**Ctrl1_vc_map7:** Bits [7:4]: New Virtual Channel for VC7
0x0378	7:4	0x00	**Ctrl1_vc_map8:** Bits [7:4]: New Virtual Channel for VC8
0x0379	7:4	0x00	**Ctrl1_vc_map9:** Bits [7:4]: New Virtual Channel for VC9
0x037A	7:4	0x00	**Ctrl1_vc_map10:** Bits [7:4]: New Virtual Channel for VC10
0x037B	7:4	0x00	**Ctrl1_vc_map11:** Bits [7:4]: New Virtual Channel for VC11
0x037C	7:4	0x00	**Ctrl1_vc_map12:** Bits [7:4]: New Virtual Channel for VC12
0x037D	7:4	0x00	**Ctrl1_vc_map13:** Bits [7:4]: New Virtual Channel for VC13
0x037E	7:4	0x00	**Ctrl1_vc_map14:** Bits [7:4]: New Virtual Channel for VC14
0x037F	7:4	0x00	**Ctrl1_vc_map15:** Bits [7:4]: New Virtual Channel for VC15

Pixel Mode

Pixel mode supports VC extension as well as overriding VC in the deserializer. If the incoming video source is using extended VCs, VCX needs to be enabled on the serializer and deserializer.

See *Table 13* for extended virtual channel and VC override registers details.

Pixel Mode Programming Example

This example enables the extended VCs on the MAX96793 and remaps VCs 0 and 1 to be 5 and 6.

Turn on VC extension

0x80, 0x331, 0xB0

Enable VC remapping

0x80, 0x330, 0x20

VC remap of VC0 to VC5

0x80, 0x345, 0x50

VC remap of VC1 to VC6

0x80, 0x346, 0x60

Tunneling Mode
Tunnel mode does not support overriding of the VC; however, it supports VC extension. If the incoming video source is using extended VCs, VCX needs to be enabled on serializer and deserializer.

Tunneling Mode Programming Example
0x04,0x80,0x03,0x31,0xB0, // ctrl1_vcx_en=1, enable VCX on Serializer controller 1

0x04,0x98,0x04,0x4A,0xD8, // CSI_VCX_EN=1, enable VCX on Deserializer controller 1

MIPI Controller and PHY Settings

MIPI PHY Settings

The MIPI PHY settings contain programming options for continuous vs. noncontinuous clock modes, number of lanes, lane mapping, polarity swapping, and port selection.

The MAX96793 CSI-2 to GMSL3 Serializer has two 2-lane capable MIPI PHYs (PHY 1, 2) controlled by one MIPI controllers (Ctrl 1) to establish up to one 4-lane input port. The 0 contains the MIPI PHY setting registers.

The 1x1, 1x2, 1x3, and 1x4 configurations use lane count settings. See *Table 14* for register write details.

Table 14. MAX96793 MIPI PHY Setting Registers

Register	Bitfield Name	Bits	Default Value	Description
0x0330	mipi_noncontclk_en	6	0b0	0 = enables MIPI continuous clock 1 = enables MIPI noncontinuous clock
0x0330	ctrl1_vc_map_en	5	0b0	0 = disables VC mapping 1 = enables VC mapping
0x0330	mipi_rx_reset	3	0b0	0 = do not reset MIPI receiver 1 = resets MIPI receiver (This bit should be toggled HIGH and then LOW before any video is received—per errata.)
0x0330	phy_config[2:0]	2:0	0b000	0 = 1x4 (only available option)
0x0331	ctrl1_vcx_en	7	0b0	0 = extended VC disabled 1 = extended VC enabled
0x0331	ctrl1_deskewen	6	0b0	0 = deskew calibration disabled 1 = deskew calibration enabled
0x033C, 0x033E	phy{1,2}_hs_err[7:6]	7:6	0b0000	Bit 7 represents lane 0, Bit 6 represents lane 1 0 = Deskew calibration pattern flag not received 1 = Deskew calibration pattern flag received
0x033C, 0x033E	phy{1,2}_hs_err[5:4]	5:4	0b0000	Bit 5 represents lane 0, Bit 4 represents lane 1 0 = Default 1 = Deskew calibration failure
0x0331	ctrl1_num_lanes[1:0]	5:4	0b11	00 = 1 data lane 01 = 2 data lanes

				10 = 3 data lanes 11 = 4 data lanes
0x0332	phy1_lane_map[3:2]	7:6	0b11	00 = map lane0 to lane3 01 = map lane1 to lane3 10 = map lane2 to lane3 11 = map lane3 to lane3
0x0332	phy1_lane_map[1:0]	5:4	0b10	00 = map lane0 to lane2 01 = map lane1 to lane2 10 = map lane2 to lane2 11 = map lane3 to lane2
0x0333	phy2_lane_map[3:2]	3:2	0b01	00 = map lane0 to lane1 01 = map lane1 to lane1 10 = map lane2 to lane1 11 = map lane3 to lane1
0x0333	phy2_lane_map[1:0]	1:0	0b00	00 = map lane0 to lane0 01 = map lane1 to lane0 10 = map lane2 to lane0 11 = map lane3 to lane0
0x0334	phy1_pol_map[1]	5	0b0	0 = normal polarity for data lane 3 1 = inverse polarity for data lane 3
0x0334	phy1_pol_map[0]	4	0b0	0 = normal polarity for data lane 2 1 = inverse polarity for data lane 2
0x0335	phy2_pol_map[2]	2	0b0	0 = normal polarity for clock lane 1 = inverse polarity for clock lane
0x0335	phy2_pol_map[1]	1	0b0	0 = normal polarity for data lane 1 1 = inverse polarity for data lane 1
0x0335	phy2_pol_map[0]	0	0b0	0 = normal polarity for data lane 0 1 = inverse polarity for data lane 0
0x0345-0x0347, 0x036C-0x036F, 0x0377-0x037F	ctrl1_vc_map{0,15}	7:4	0b0000	VC reassignment registers. See the *Extended Virtual Channels* section for description.

MIPI Data Lane and Polarity Swap

The MAX96793 supports lane swapping for pins on the same MIPI port.

The data pins can be swapped within each port, but the clock location is fixed. For example, in 2x4 mode, the default mappings of the D0, D1, D2, and D3 pairs can be swapped to different output pins. Additionally, the polarity of each output data and clock pair can be inverted.

See *Table 14* for relevant registers. *Figure 6* shows the default lane mapping that matches device pinout. *Figure 7* demonstrates the polarity swap exatabe.

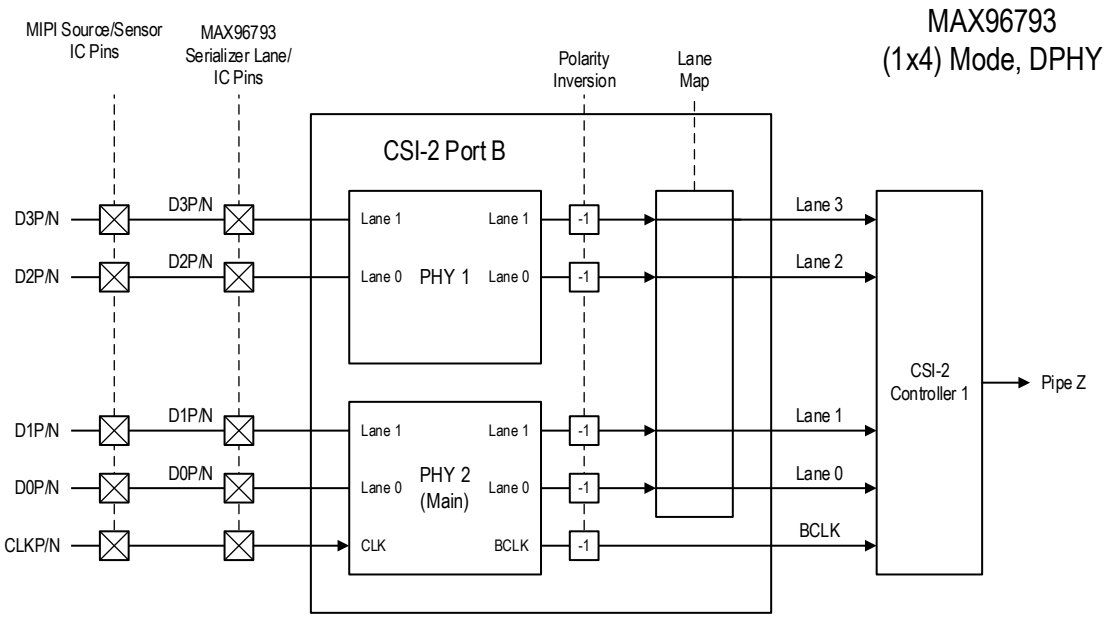

Figure 6. MAX96793 1x4 DPHY Default Lane Mapping

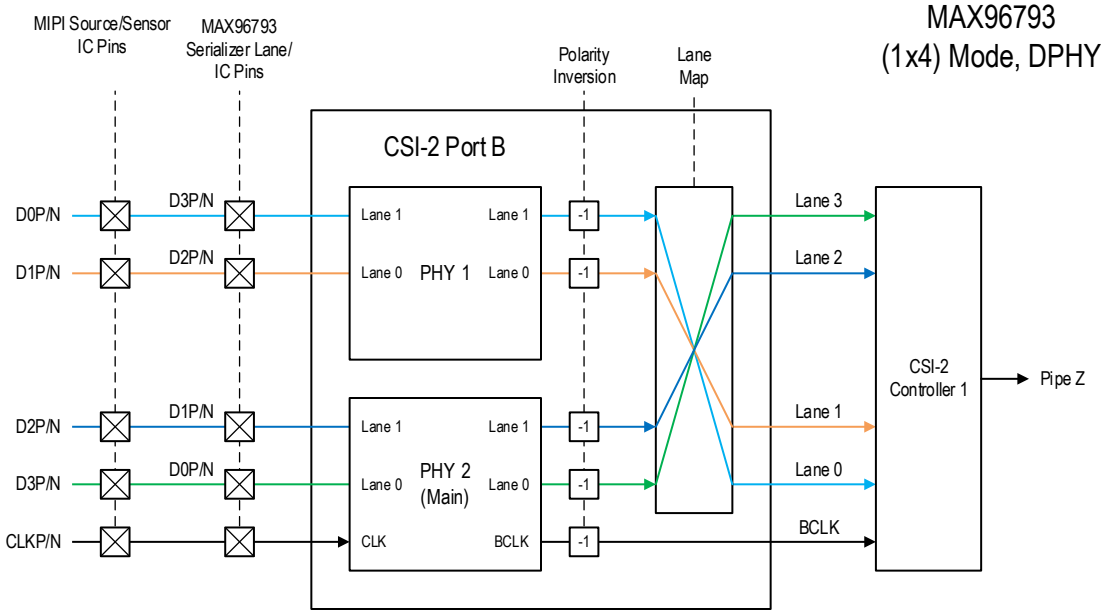

Figure 7. MAX96793 1x4 DPHY Lane Swap Example

Lane Swap Programming Example
Figure 7 lane swap example register writes are seen in the following description.

#SER I2C Address=0x80

Set lane mapping for all 4 lanes on ctrl 1. This is written to completely swap the device pinout from default as shown in the figure.

0x80,0x0332,0x10 #D0 mapped to D3, D1 mapped to D2

0x80,0x0333,0x0B #D2 mapped to D1, D3 mapped to D0

MIPI D-PHY Deskew Settings

The D-PHY deskew mechanism is only relevant to lane speeds greater than 1.5Gbps per lane (Deskew optional for data rates below 1.5Gbps/lane). Periodic deskew requires a continuous clock and the clock lane always in the high-speed mode. Deskew is initiated by the transmitter under CSI-PPI control.

The MAX96793 serializer only has one bit for enabling deskew calibration (ctrl1_deskewen) which can be found in REG 0x331. See *Table 14* for more information.

Deskew Register Example
#SER I2C Address=0x80

Enable deskew calibration on the serializer

0x80, 0x331, 0x70

I²C Control Channels

Overview

The primary I²C control channel is used to provide access to both serializer and deserializer registers across the GMSL link. This provides flexibility where the registers for both serializer and deserializer are accessible from whichever side the main microcontroller resides (for camera applications, the main microcontroller typically resides on the deserializer side).

The pass-through I²C channels are used to send I²C data across the GMSL Link. The pass-through channels can access the remote-side devices connected through the corresponding pass-through ports but cannot access the serializer or deserializer registers.

When making changes to any of the serializer's or deserializer's I²C configuration, such as enabling or disabling an I²C channels, a 10µs delay from the write acknowledgment (ACK) to the next transaction is required.

Port Access and Routing

The multifunction pins (MFPs) shown in *Table 15* are used for the I²C primary and pass-through channels.

Table 15. MAX96793 MFPs for I²C

MFP Pin	Default Function	I²C Function #1	I²C Function #2	Notes
MFP7	GPI7	SDA1_RX1	N/A	Enables I²C pass-through by register

MFP8	GPIO	SCL1_TX1	N/A	Enables I²C pass-through by register
MFP9	SDA_RX or RX	SDA_RX or RX	SDA2_RX2	I²C or UART function selected by CFG0 pin or Enables I²C pass-through by register
MFP10	SCL_TX or TX	SCL_TX or TX	SCL2_TX2	I²C or UART function selected by CFG0 pin or Enables I²C pass-through by register

On power-up, the device should be set to I²C mode by the CFG0 latch. The function names in the MFPs table and following I²C sections assume that the device has been configured for I²C mode.

By default, the primary I²C control channel lines are brought out on MFP9 and MFP10 for SDA and SCL, respectively. The user can disable the primary control channel's line access by setting field DIS_LOCAL_CC in register 0x1. Also, the access to remote device control can be disabled by setting field DIS_REM_CC in register 0x1.

CRC for I²C and Message Counter Transactions

The MAX96793 devices have the option to add CRC and a message counter to the I²C interface. The interface protocols are the same as in legacy designs, but there are new bytes introduced in the packets to support CRC and a message counter. The ECU and SerDes device each keep a copy of a message counter, which increments each transaction. The message counter and CRC bytes are sent in each transaction to ensure data integrity. Features are available only on primary I²C control channel, not supported for pass-through channels. Features are disabled by default and need to be enabled with register writes.

Note: These features can only be used with I²C host controller that supports addition of this CRC.

CRC for I²C Transactions

The CRC feature is used to detect corrupt data written on the serializer's primary I²C control channel. Each I²C transaction has a corresponding CRC packet associated with it. If a CRC is detected, the command is not executed, and the transaction is reported as Not Acknowledge (NACK).

The MAX96793 CRC feature is enabled by setting fields CC_CRC_EN and CC_CRC_MSGCNTR_OVR in register 0x4.

Message Counter for I²C Transactions

The message counter feature is used to detect missing or repeated I²C transactions. For every transaction, the message counter is incremented accordingly while a copy of the counter is maintained on both ends of the I²C link. If a mismatch between copies is found, then the transaction is rejected.

The MAX96793 message counter feature is enabled by setting fields CC_MSGCNTR_EN and CC_CRC_MSGCNTR_OVR in register 0x4.

Enabling CRC for I²C and Message Counter

Following are the register writes to enable CRC for I²C and Message counter for primary I²C control channel.

(This only works with I²C host controller that supports CRC/Message counter.)

#SER I2C Address=0x80

#Enable CRC and Message Counter on SER in XTAL Mode

0x80,0x4,0x1D

I²C Registers

Table 16 has registers that are needed to enable/disable primary and pass-through I²C channels.

Table 16. MAX96793 I²C Registers

Register	Bits	Default Value	Description
0x0001	7:4	0x08	**I²C Enable Register:** Bit [7]: Enables pass-through I²C Control Channel 2 (SDA2, SCL2) Bit [6]: Enables pass-through I²C Control Channel 1 (SDA1, SCL1) Bit [5]: Disables main I²C Control Channel connection to SDA and SCL pins Bit [4]: Disables access to remote device Control Channel over GMSL3 connection
0x0004	4:2	0x18	**Enable CRC and Message Counter Register:** Bit [4]: Enables I²C message counter. **Note: Only active when Bit [2] is also set to 1.** Bit [3]: Enables I²C CRC packeting. **Note: Only active when Bit [2] is also set to 1.** Bit [2]: Enables manual override of I²C CRC or message counter configuration. If set to a 0, then CRC and message counter features are disabled.
0x0006	4	0x80	**I²C Selection Register:** Bit [4]: Enables I²C when set to a 1 or UART when set to a 0 **Note: This bit is set according to the CFG0 pin value on power-up. Writing to this register is not recommended.**
0x1D00	3	0x00	**Enable CRC Computation Register:** Bit [3]: Computes register CRC after every I²C register write
0x1D08	0	0x00	**Message Counter Reset Register:** Bit [0]: Resets Message Counter value to 0
0x1D09	1:0	0x00	**CRC Reset Register:** Bit [1]: Resets Message Counter error count to 0 Bit [0]: Resets CRC error count to 0.
0x1D0A	7:0	0x00	**Read CRC Value Register:** Bits [7:0]: CRC value for the last write transaction
0x1D0B	7:0	0x00	**Read Message Counter Low Bits Register:** Bits [7:0]: Low bits of current message counter value
0x1D0C	7:0	0x00	**Read Message Counter High Bits Register:** Bits [7:0]: High bits of current message counter value

Enabling I²C Pass-Though Channels

When enabling an I²C pass-through channel, other MFP functions must be disabled first.

With the MAX96793, the user can bring out the first pass-through I²C channel (SDA1_RX1/SCL1_TX1) on MFP7/MFP8. The second pass-through I²C channel (SDA2_RX2/SCL2_TX2) can be programmed on MFP9/MFP10. Pass-through channels are enabled by setting the fields IIC_1_EN and IIC_2_EN in register 0x1.

Control Channel Programming Example

The following example shows the register writes needed to enable I²C pass-through channel 1 on the MAX96793.

#DES I2C Address=0x98

#SER I2C Address=0x80

#Enable I2C Pass-through channel 1 on DES

0x98,0x0001,0x42

#Disable UART Pass-through channel 1 on DES

0x98,0x0003,0x43

#Enable I2C Pass-through channel 1 on SER

0x80,0x0001,0x48

I²C Broadcasting

Overview

When transmitting to a multiple-link input deserializer or multiple deserializers, each device on the serializer side requires a unique address for individual programming and identification. Through I²C translation and address reassignment, each serializer and image sensor can have both a unique address and a broadcasting address. This allows for selective programming of each device and the ability to broadcast commands to all devices simultaneously. When broadcasting, if any remote GMSL I²C port ACKs the packet, it ACKs for all remote GMSL I²C ports.

Note: When making changes to any of the serializer or deserializer's I²C configuration, such as enabling or disabling an I²C port, at least a 10µs delay from the write ACK to the next transaction is required.

An example of I²C broadcasting is discussed in the following section. Two equivalent camera modules, including an image sensor and GMSL3 serializer with the same respective addresses, are connected to two GMSL3 deserializers with different device addresses. Each of the camera modules comprises a serializer at the default I²C address 0x80 and an image sensor at address 0x20.

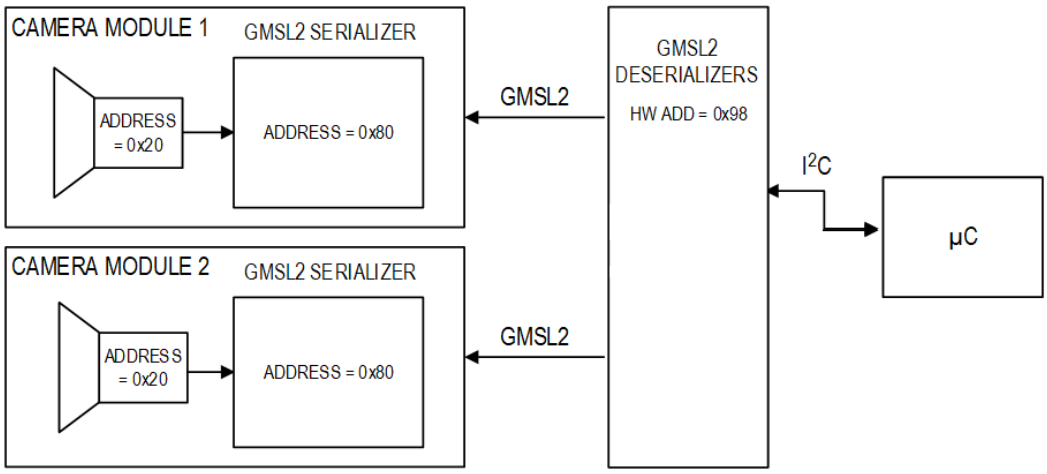

Figure 8. I²C Interfaced Camera-Module System with Default Address Settings and Dual Deserializer

I2C Broadcasting Technique

The I²C broadcasting technique helps to communicate with multiple camera-serializer modules with a single microcontroller, which in turn streamlines the transmission process.

The general procedure is to:

- Isolate a single camera/serializer module for remote I²C access, meaning no other device with the same address should be connected to the I²C data line.
- Change the serializer address to a unique address.
- Modify the first I²C address translation register with a common source address but the unique destination address. This is to streamline the interface with the serializer.
- Modify the second I²C address translation register with a unique source address but the default image sensor addresses for the destination address. This is to streamline the interface with the image sensor.
- Repeat this process for each camera serializer module.

Note: When making changes to any of the serializer or deserializer's I²C configuration, such as enabling or disabling an I²C port, at least a 10µs delay from the write ACK to the next transaction is required.

I²C Broadcasting GMSL3 Use Case Example

The procedure for the I²C broadcasting example is as follows:

- Isolate camera module 1 by disabling camera module 2's GMSL link (RESET_LINK = 1).
- Change the serializer device address in camera module 1 from 0x80 to 0x82. This is done with a register write to DEV_ADDR[6:0] located in REG0.
- Modify the first address translation register in this serializer to give a broadcast address (0xC4) to the serializer. Program 0xC4 into the source register SRC_A[6:0], and 0x82 in the destination register DST_A[6:0]. Thus, for the serializer in camera module 1, anything sent to address 0xC4 is sent to address 0x82 instead.
- Modify the second translation register in this serializer to give a unique address to the image sensor. Program 0x22 into the source register SRC_B[6:0] and 0x20 into the destination register DST_B[6:0]. Thus, for the serializer in camera module 1, anything sent to address 0x22 is sent to address 0x20 instead.
- Isolate camera module 2 by disabling camera module 1's GMSL link (RESET_LINK = 1) and enabling camera 2's GMSL link (RESET_LINK = 0).

- Change the serializer device address in camera module 2 from 0x80 to 0x84. This is done with a register write to DEV_ADDR[6:0] located in REG0.
- Modify the first address translation register in this serializer to give a broadcast address (0xC4) to the serializer. Program 0xC4 into the source register SRC_A[6:0] and 0x84 in the destination register DST_A[6:0]. Thus, for the serializer in camera module 2, anything sent to address 0xC4 is sent to address 0x84 instead.
- Modify the second translation register in this serializer to give a unique address to the image sensor. Program 0x24 into the source register SRC_B[6:0] and 0x20 into the destination register DST_B[6:0]. Thus, for the serializer in camera module 2, anything sent to address 0x24 is sent to address 0x20 instead.
- Now, enable all the links for remote primary I²C port access.
- All devices should be present on the I²C bus. Continue with any additional required system configuration.

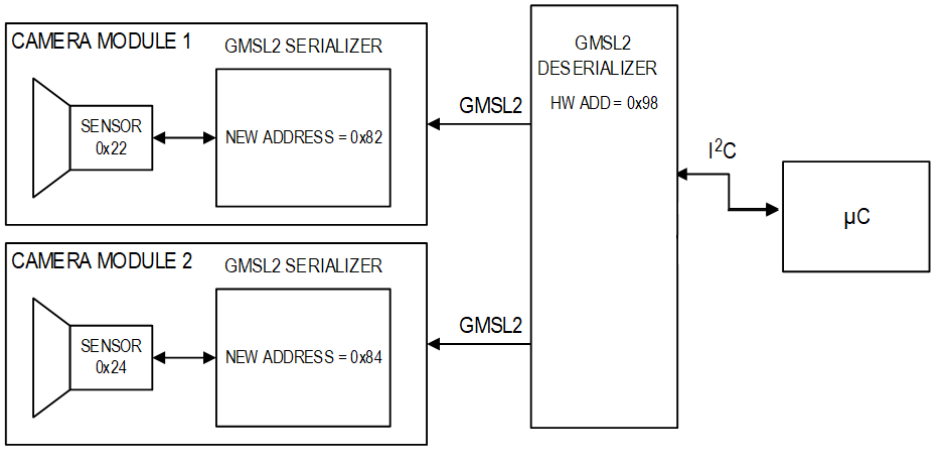

Figure 9. Two Camera-Module System with Translated Address Settings and Dual Deserializer

Table 17. I²C Broadcasting Example – Serializer

Old I²C Address	New I²C Address	SRC_A (SER, 0x42)	DST_A (SER, 0x43)	Sink Devices
0x80	0x82	0xC4	0x82	Serializer in Camera Module 1
0x80	0x84	0xC4	0x84	Serializer in Camera Module 2

0 shows how the serializers are assigned a single device address to allow writes to all devices as a broadcast. This allows I²C host controller to broadcast with address 0xC4.

Table 18. I²C Broadcasting Example - Image Sensor

Old I²C Address	New I²C Address	SRC_B (SER, 0x44)	DST_B (SER, 0x45)	Sink Devices
0x20	0x22	0x22	0x20	Serializer in Camera Module 1
0x20	0x24	0x24	0x20	Serializer in Camera Module 2

Table 18 shows how each image sensor is assigned a unique device address. This allows I²C host controller to isolate I²C commands to only one image sensor.

I²C Broadcasting Programming Example

This script sets up the I²C broadcasting as shown in *Figure 9*.

#DES I2C Address=0x98

Disable GMSL Link B

0x98,0x0013,0x1

Change I2C address for this Link A serializer

0x80,0x0000,0x82

Set Ser source to 0xC4

0x82,0x0042,0xC4

Set Ser destination to 0x82

0x82,0x0043,0x82

Set Image sensor source to 0x22

0x82,0x0044,0x22

Set Image sensor destination to 0x20

0x82,0x0045,0x20

Enable GMSL Link B

0x98,0x0013,0x10

Disable GMSL Link A

0x98,0x0010,0x51

Change I2C address for this Link B serializer

0x80,0x0000,0x84

Set Ser source to 0xC4

0x84,0x0042,0xC4

Set Ser destination to 0x84

0x84,0x0043,0x84

Set Image sensor source to 0x24

0x84,0x0044,0x24

Set Image sensor destination to 0x20

0x84,0x0045,0x20

Enable GMSL Link A

0x98,0x0010,0x31

UART Control Channel

Overview

The primary universal asynchronous receiver/transmitter (UART) control channel is used to provide access to both serializer and deserializer registers across the GMSL link. This provides flexibility where the registers for both serializer and deserializer are accessible from whichever side the main microcontroller resides (for camera applications, the main microcontroller typically resides on the deserializer side).

The pass-through UART channels are used to send UART data across the GMSL Link. The pass-through channels can access the remote-side devices connected through the corresponding pass-through ports but cannot access the serializer or deserializer registers.

Note: When making changes to any of the serializer or deserializer's UART configuration, such as enabling or disabling an UART channels, a 10µs delay from the write ACK to the next transaction is required.

Base Mode

Base mode allows the device registers of both the serializer and the deserializer to be accessed by the host microcontroller. It is the default mode for the primary UART control channel on power-up.

Bypass Mode

In the bypass mode, both the serializer and deserializer ignore all UART commands from the microcontroller. The serializer/deserializer registers are not accessible and the microcontroller can freely communicate with any peripherals using its defined UART protocol. In this mode, the UART commands are still sent over the GMSL3 link. This mode prevents inadvertent programming of the serializer/deserializer registers and can be switched in and out of during normal operation.

Port Access and Routing

The MFPs shown in *Table 19* are used for the UART primary and pass-through channels.

Table 19. MAX96793 MFP Pins for UART

MFP Pin	Default Function	UART Function #1	UART Function #2	Notes
MFP7	GPI7	RX1	N/A	Enables UART pass-through by register
MFP8	GPIO	TX1	N/A	Enables UART pass-through by register
MFP9	SDA_RX or RX	SDA_RX or RX	RX2	I^2C or UART function selected by CFG0 pin or Enables UART pass-through by register
MFP10	SCL_TX or TX	SCL_TX or TX	TX2	I^2C or UART function selected by CFG0 pin or Enables UART pass-through by register

On power-up, the device should be set to UART mode by the CFG0 latch. The function names in this MFP table and following UART sections assume the device has been configured for UART mode.

By default, the primary UART control channel lines are brought out on MFP9 and MFP10 for RX and TX, respectively. The user can disable the primary control channel's line access by setting field DIS_LOCAL_CC in register 0x01. One can also disable access to remote device control by setting field DIS_REM_CC in register 0x01.

CRC for UART and Message Counter Transactions

The MAX96793 devices have the option to add CRC and a message counter to the UART interface. The interface protocols are the same as in legacy designs, but there are new bytes introduced in the packets to support CRC and a message counter. The ECU and SerDes device each keep a copy of a message counter, which increments each transaction. The message counter and CRC bytes are sent in each transaction to ensure data integrity. Features are available only on primary UART control channel, not supported for pass-through channels. Features are disabled by default and need to be enabled with register writes.

Note: These features can only be used with UART Host controller that supports addition of this CRC.

CRC for UART Transactions

The CRC feature is used to detect corrupt data written on the UART control channel. Each UART transaction has a corresponding CRC packet associated with it. The host microcontroller must compute and send a CRC byte after each data byte.

- If the host microcontroller is writing to the serializer registers, then the serializer receives the data byte, calculates the CRC using an identical CRC engine, and verifies a match before accepting the data byte. If a mismatch is detected, then the write is not accepted, and the error counter is incremented.
- If the host microcontroller is reading the serializer registers, then the serializer calculates the CRC byte and appends it to the output data stream. The host microcontroller's CRC engine should then calculate its own CRC byte and compare it with the one received from the serializer to determine if there is a mismatch.

The MAX96793 CRC feature is enabled by setting fields CC_CRC_EN and CC_CRC_MSGCNTR_OVR in register 0x4.

Message Counter for UART Transactions

The message counter feature is used to detect missing or repeated UART transactions. For every transaction, the message counter is incremented accordingly while a copy of the counter is maintained on both ends of the UART transaction. If a mismatch between copies is found, then the transaction is rejected.

The MAX96793 message counter feature is enabled by setting fields CC_MSGCNTR_EN and CC_CRC_MSGCNTR_OVR in register 0x4.

Enabling CRC for UART and Message Counter

Following are the register writes to enable CRC for UART and Message counter for primary UART control channel. (It only works with UART Host controller that supports CRC/Message counter.)

#SER UART Address=0x80

#Enable CRC and Message Counter on SER in XTAL Mode

0x98,0x4,0x1D

UART Registers

The following table has registers that are needed to enable/disable primary and pass-through UART channels.

Table 20. MAX96793 UART Registers

Register	Bits	Default Value	Description
0x0001	5:4	0x08	**UART Control Channel Enable Register:** Bit [5]: Disables main UART Control Channel connection to TX/RX pins Bit [4]: Disables access to remote device control channel over GMSL3 connection
0x0003	5:4	0x00	**UART Pass-Through Channel Enable Register:** Bit [5]: Enables pass-through UART Channel 2 Bit [4]: Enables pass-through UART Channel 1
0x0004	4:2	0x18	**Enable CRC and Message Counter Register:** Bit [4]: Enables UART message counter. **Note: Only active when Bit [2] is also set to 1.** Bit [3]: Enables UART CRC packeting. **Note: Only active when Bit [2] is also set to 1.** Bit [2]: Enables manual override of UART CRC or message counter configuration. If set to a 0, then CRC and message counter features are disabled.
0x0006	4	0x80	**UART Selection Register:** Bit [4]: Enables UART when set to a 0. **Note: This bit is set according to the CFG0 pin value on power-up. Writing to this register is not recommended.**
0x0048	5:0	0x42	**UART Bypass Mode Control Register:** Bit[5]: Enables UART bypass mode control by remote GPIO pin (Function **MS** on **MFP8**) Bit[4]: Enables UART bypass mode control by local GPIO pin (**GPIO2)** Bit[3]: Enables or disables parity bit in bypass mode Bit[2]: UART soft-bypass timeout duration Bit[1]: Enables UART soft-bypass mode
0x004F	7:6 3:2	0x00	**UART Pass-Through Channels Config Register:** Bit[7]: Uses standard or custom bit rate Bit[6]: Enables parity bit Bit[3]: Uses standard or custom bit rate Bit[2]: Enables parity bit
0x1D08	0	0x00	**Message Counter Reset Register:** Bit [0]: Resets Message Counter value to 0
0x1D09	1:0	0x00	**CRC Reset Register:** Bit [1]: Resets Message Counter error count to 0 Bit [0]: Resets CRC error count to 0.
0x1D0A	7:0	0x00	**Read CRC Value Register:** Bits [7:0]: CRC value for the last write transaction
0x1D0B	7:0	0x00	**Read Message Counter Low Bits Register:** Bits [7:0]: Low bits of current message counter value
0x1D0C	7:0	0x00	**Read Message Counter High Bits Register:** Bits [7:0]: High bits of current message counter value

Serial Peripheral Interface

Overview

The SPI is available on the MAX96793. Unlike I²C and UART, SPI cannot be used to modify any registers in either serializer or deserializer, it is only used to transfer SPI data across the GMSL link. Typical SPI use cases are to send commands for other devices or to stream data other than video data (e.g., for sensors). GMSL devices support SPI transmission rate up to 25MHz.

Figure 10 shows the GMSL SPI architecture. On each side of the link, the GMSL devices become part of a main-subordinate pair and have transmit and receive buffers inside. On the local side, an internal SPI subordinate receives data from an external SPI main or microcontroller and transmits it across the serial link. On the remote side, the device receives the data from GMSL link and uses an internal SPI main to transmit the data to the external SPI main/subordinate devices.

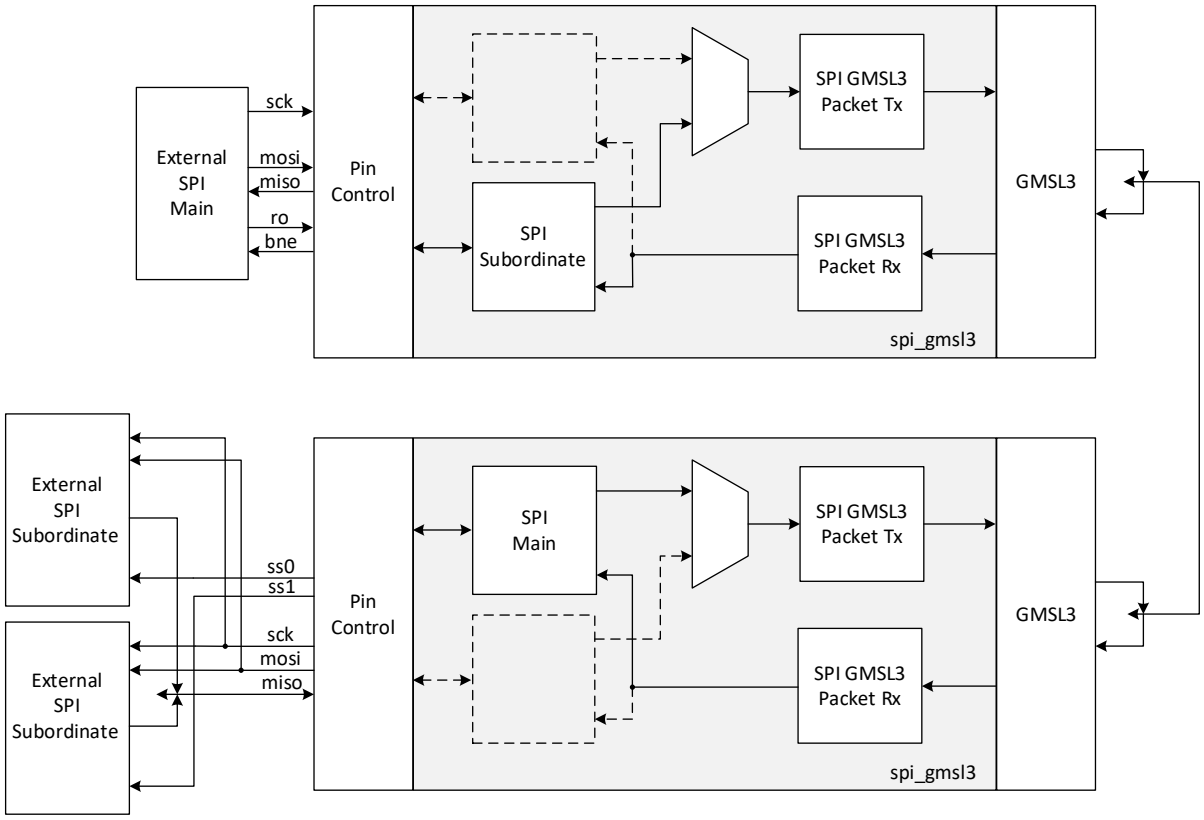

Figure 10. GMSL3 SPI Architecture

MFP/CFG Pin Setup for SPI

- Refer to the latest device data sheets for SPI MFP pins. Some MFP pins may have default alternate functions that must be disabled before enabling SPI. MFP status tool in the GMSL GUI can be used to verify the MFP functions that are enabled/disabled. If any of the SPI pins are also used as CFG pins, do not let any external SPI devices pull the CFG pins up or down until the GMSL devices power up and the CFG pins are latched. Power on the GMSL parts with the external SPI main device not connected, or not pulling on the CFG pins, otherwise, the GMSL part boots up into an unwanted configuration.
- RO (Read Only) is an input bit that determines if the SPI subordinate is in read or write mode.

- BNE (Buffer Not Empty) is an output bit that shows the receive FIFO state. BNE is low when the buffer is empty; BNE is high when there is data in the buffer. This bit is used to determine the status of the buffer for data transfers and avoiding buffer overflow.

SPI Setup Registers

The following table shows some of the important setup registers for enabling SPI. Register block in the data sheet has additional details for configuring the SPI main, subordinate, SCLK timings, etc.). See the programming script in the *SPI Example with Register Writes* section as an example.

Table 21. MAX96793 SPI Register Settings

Register	Bitfield Name	Bits	Default Value	Description
0x170	SPI_EN	0	0	0 = SPI not enabled 1 = SPI enabled
0x170	MST_SLVN	1	0	0 = SPI subordinate 1 = SPI main
0x170	SPI_IGNR_ID	2	1	0 = Accepts packets with proper ID 1 = Ignores ID and accepts all packets (recommended)
0x172	SPIM_SS1_ACT_H	0	1	0 = SS1 is active low 1 = SS1 is active high
0x172	SPIM_SS2_ACT_H	1	1	0 = SS2 is active low 1 = SS2 is active high
0x176	RWN_IO_EN	0	0	0 = Do not bring RO out to MFP pin 1 = Bring out RO to MFP pin
0x176	BNE_IO_EN	1	0	0 = Do not bring BNE out to MFP pin 1 = Bring out BNE to MFP pin
0x176	BNE	5	0	0 = No bytes to read 1 = Bytes ready to read
0x177	SPI_TX_OVRFLW	6	0	0 = No overflow 1 = Overflow
0x177	SPI_RX_OVRFLW	7	0	0 = No overflow 1 = Overflow

SPI Initialization

Configure Serializer and deserializer in the following order to initialize SPI (starting from the default values):

Configure SPI mode 0 or 3 on the serializer and deserializer

Set SS output polarity (remote side).

Set the clock delay and SCLK rate high/low times (in number of 300MHz clocks).

Program the IO pin enables (BNE/RO/SS1/SS2).

Configure internal GMSL Main/Subordiante mode, SPI ID (if needed), and enable SPI.

SPI Example with Register Writes

SPI Setup Example Script (0x80 is the serializer address, 0x98 is the deserializer address), assuming microcontroller or external SPI main is on serializer side.*

0x98,0x003,0x3 #disable pass-through UART

0x98,0x162,0x0 #select SPI link A on deserializer (SPI_LINK_SELECT).

0x80,0x170,0x9 #enable SPI, default set to subordinate, and ignore the SPI header ID

0x80,0x171,0x1D #default, sets SPI packet size and GMSL link scheduler priority.

0x80,0x172,0x0 #default, subordinate select (SS) is active low, SPI mode is 0.

0x80,0x173,0x0 #default, delay between assertion of subordinate select (SS) and SPI clock (SCLK) start

0x80,0x176,0x3 #enable RO and BNE

0x80,0x178,0x0 #default, timeout delay

0x98,0x170,0xB #Enable SPI channel, set as main

0x98,0x171,0x1D #default, sets SPI packet size and GMSL link scheduler priority.

0x98,0x172,0x0 #default, subordinate select is active low, SPI mode is 0.

0x98,0x173,0x1E #default, delay between assertion of subordinate select and clock start.

0x98,0x174,0x1E #SPI clock low time

0x98,0x175,0x1E #SPI clock high time

0x98,0x176,0xC #Enable subordinate select 1 (SS1) and 2 (SS2), RO and BNE not enabled

0x98,0x178,0x0 #SPI timeout delay

*If microcontroller or external SPI main is on deserializer side, swap the Ser and Des register writes in the script.

Figure 11 and *Figure 12* show MFP pins being used for SPI after running this script.

Serializer | Deserializer

Pin	MFP	Function	Function	Function	Function	Function	Function	Function	Function
2	0	PCLK	**SCLK**	VTG0	GPIO0				
3	1	**BNE**	SS1	VTG1	GPO1				
4	2	RCLKOUT(alt)	VTG2	**GPO2**					
17	3	VS	ERRB/LFLTB	LOCK	ADC0	VTG3	**GPIO3**		
18	4	HS	**RO**	SS2	RCLKOUT	VTG4	GPIO4		
21	5	LMNO	ADC1	**ODO5/GPI5**					
22	6	LMN1	ADC2	**ODO6/GPI6**					
31	7	SDA1_RX1	D12	**MOSI**	LOCK(alt)	VTG7	GPIO7		
32	8	SCL1_TX1	D13	**MISO**	ERRB(alt)	MS	VTG8	GPIO8	
5	9	**SDA_RX**	SDA2_RX2	ODO9/GPI9					
6	10	**SCL_TX**	SCL2_TX2	ODO10/GPI10					

Figure 11. SPI MFP Pin Settings for Serializer

Serializer | Deserializer

Pin	MFP	Function	Function	Function	Function	Function	Function	Function	Function
2	0	SDA1_RX1	SDA2_RX2	Fsync/VS2/H...	**SCLK**	CNTL1(GMSL...	MS	GPIO00	
3	1	SCL1_TX1	SCL2_TX2	**LOCK**	GPIO01				
4	2	BNE	CNTL0(GMS...	**SS1**	GPO02				
8	3	VS1	HS1	CNTL3(GMSL...	**SS2**	GPO03			
9	4	RO (ALT)	**LFLTB / ERRB**	GPIO04					
21	5	SDA2_RX2	SDA1_RX1	**MOSI**	CNTL2(GMS...	LOCK (alt)	GPI_1(GMSL...	GPIO05	
22	6	SCL2_TX2	SCL1_TX1	**MISO**	CNTL4(GMS...	GPI_0(GMSL1)	GPIO06		
27	7	**RO**	LMN0	GPI7					
28	8	LMN1	**GPI8**						
45	9	LMN2	**GPI9**						
46	10	LMN3	**GPI10**						
10	11	**SDA_RX**	ODO11/GPI11						
11	12	**SCL_TX**	ODO12/GPI12						

Figure 12. SPI MFP Pin Settings for Deserializer

SPI Example using GMSL GUI and Evaluation Boards

- It is recommended to set the SCLK output rate equal or more than the SCLK input rate to avoid buffer overflow in the ser or des. The SCLK rate can be set using register writes.
- Disconnect external SPI main device or do not pull CFG pins on Ser and Des.
- Power up EV boards (ensure that VDDIO on the external SPI main device matches the VDDIO on GMSL Ser and Des)
- Start GMSL GUI.
- Load GMSL script to enable SPI.
- Reconnect external SPI main device.
- Set RO high and write 0xA0, 0xA4(0xA0-0xA3 are used for SPI ID selection, 0xA4 asserts SS1, 0xA5 asserts SS2, and 0xA6 de-asserts both SS1 and SS2).
- As a best practice, before starting SPI data transfer, check BNE to ensure that the buffer is empty. If BNE is high, there is data in the RX buffer that is ready to be read by the external SPI main device. Set RO high and write FF until BNE = 0.
- Set RO low and start the SPI data transfer.

SPI With and Without Video Running

The SPI Tx FIFO is 16 bytes and Rx FIFO is 32 bytes. The following screenshots show the SPI oscilloscope probes on the SPI clock and data output (the receiving end at the deserializer). When there is no video through the GMSL link, the SPI data is transferred consistently without any delay, however when the GMSL link is utilized 90% by video data, there may be some intermittent pauses during the SPI data transmission. This is because video data has higher priority as compared to SPI data transfer. The 32-byte buffer compensates for this scheduling delay and makes continuous streaming of the data possible.

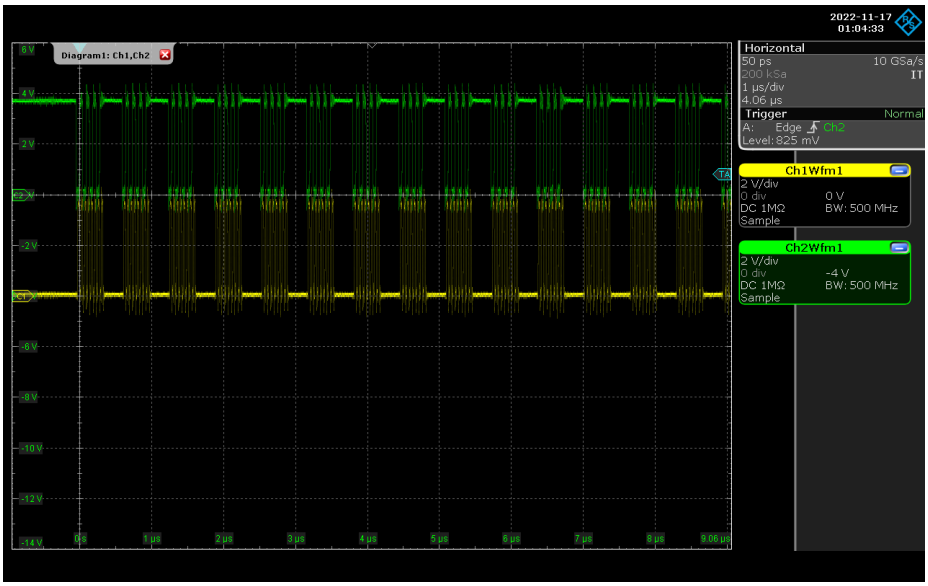

Figure 13. SPI Clock and Data at Final Output (at External SPI Subordinate), No Video on GMSL Link

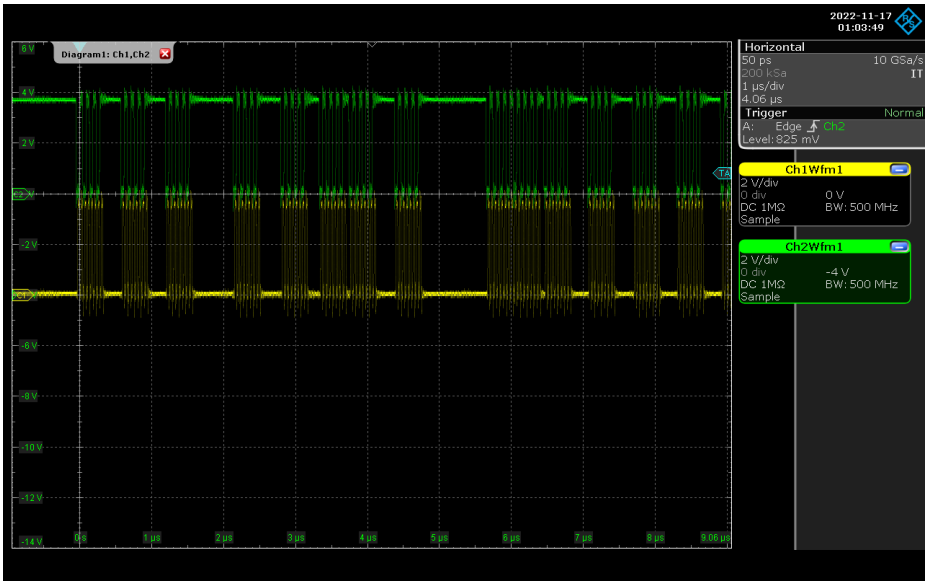

Figure 14. SPI Clock and Data at Final Output (at External SPI Subordinate), 92% Video on GMSL Link

Data Integrity and Avoiding Buffer Overflow

In general, SPI streams continuously and without having the SPI external main/generator read back any of the values. However, the techniques in this section are additional steps that are recommended to be implemented in the system to ensure that all the data is sent correctly.

After a SPI data byte is sent across the GMSL link, the GMSL device on the remote side will send the data out on the MFP pins. It will also send the data back across the link so that the external SPI main device can read back the data.

State of RO pin dictates direction of data movement.

• RO = 0: Data transmitted between main and subordinate by MOSI

• RO = 1: Data transmitted between subordinate and main by MISO, BNE is high if there are bytes in this buffer to be read back.

Note that the word “read” in the name of the RO pin does not mean that it is an output pin; it is in fact an input pin that is toggled externally high or low depending on read or write operation.

It is recommended to limit the amount of “Bytes in Transit” (bytes that have been sent but not received back) to 16. The external SPI main device can compute this value (= valid bytes sent – valid bytes read).

One way of doing this is to send the data in a group of 16 bytes or less. If more than 16 bytes are sent at a time, it is still possible (depending on timing) that all the data is sent properly but it is not possible to easily be sure that the data was sent properly.

Following is an example of transferring 4 bytes of SPI data across the GMSL link:

- RO is pulled high, and the A0 and A4 control commands are sent.
- RO is pulled low, and 4 bytes of SPI data (0x80, 0x04, 0x01, 0x47) are sent from the external SPI main device to the GMSL input side (serializer).
- The last three signals in the graph show SPI data output from the remote side of the link (Deserializer), and there may be some delay between the input and output of the SPI data across the GMSL link.
- RO is high, and the bytes are read back by the external SPI main device. **Note**: BNE is high which indicates that SPI data bytes from the external SPI subordinate are available to be read.

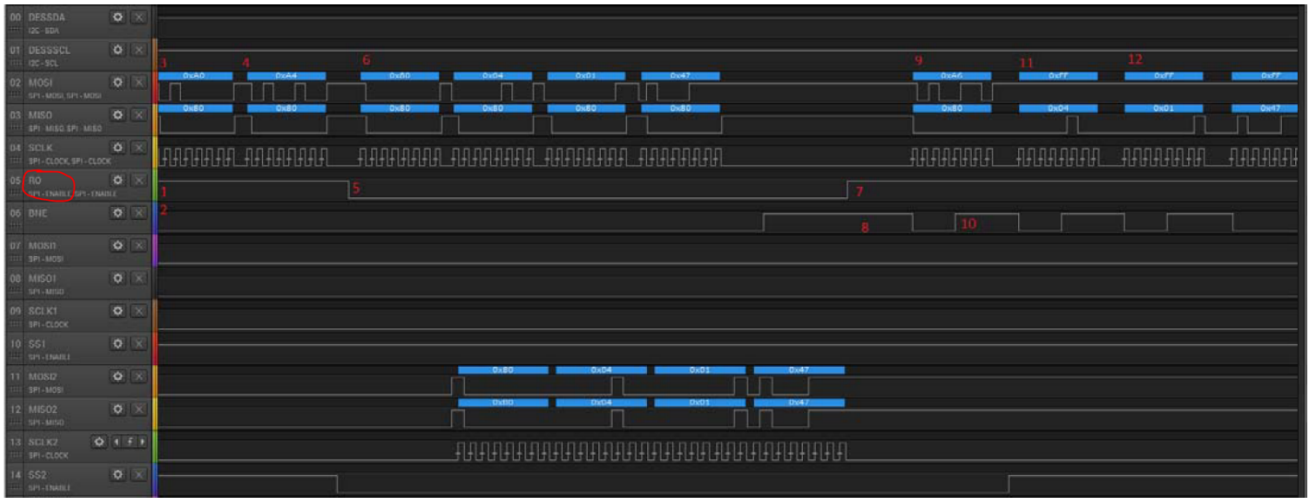

Figure 15. SPI Transmission Example

Each Rx and Tx SPI buffer has overflow detection logic with status bits that can be monitored by registers SPI_TX_OVRFLW and SPI_RX_OVRFLW.

Frame Synchronization

Overview

Frame synchronization (FSYNC) is used to align image frames sent from multiple sources in surround-view applications and is required for deserializer functions like concatenation. In FSYNC mode, the deserializer sends a sync signal to each serializer connected; the serializers then send the signal to the connected image sensor.

Video frame synchronization occurs by synchronizing the vertical sync (VS) signals of the various video streams at the image sensors. This is done on the serializer side of the link by enabling GPIO tunneling and selecting a GPIO to act as an output to the image sensor.

There are two types of FSYNC methods available on the GMSL3 CSI-2 deserializers: internal and external frame syncs. The internal frame sync indicates the GMSL3 CSI-2 deserializer generates the sync signal internally from its internal clock. The sync signal frequency must be specified in terms of the onboard crystal clock (25MHz) in the FSYNC period registers. The deserializer may be configured as a main that generates the FSYNC and outputs it on an MFP pin, or as a subordinate. If configured as a subordinate, it may accept an FSYNC output from another deserializer configured as the main.

With external frame sync, the GMSL3 CSI-2 deserializer forwards a sync signal generated by a system-on-chip (SoC). In the GMSL3 application, any of the serializer or deserializer general purpose inputs/outputs (GPIOs) can be used as a sync signal input/output by using GPIO forwarding.

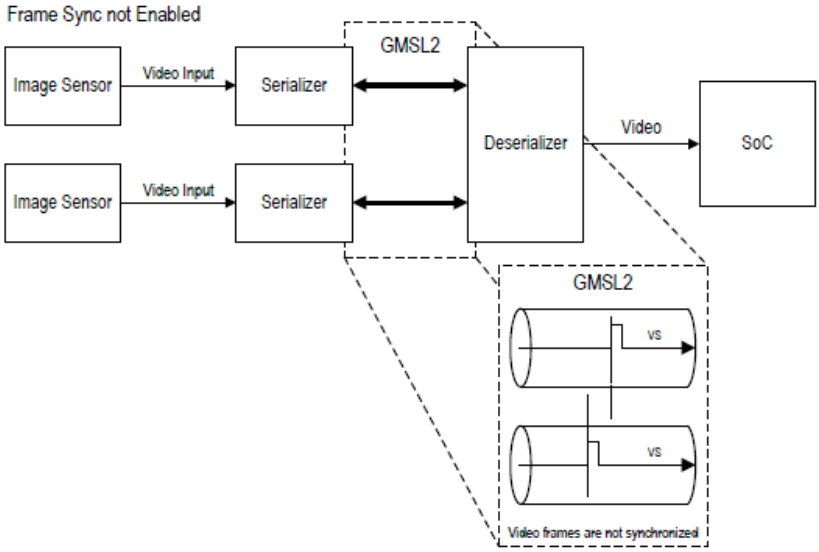

Figure 16. Frame Alignment (Without Frame Sync)

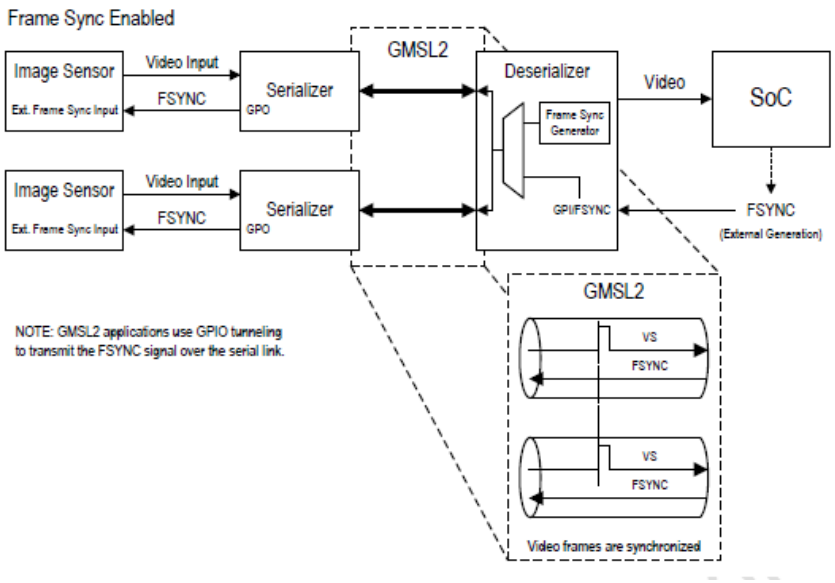

Figure 17. Frame Alignment (Frame Sync Enabled)

Configuration

The frame synchronization can be enabled on any MFP of the serializer. This is done by making the selected MFP a general-purpose output. The deserializer must be programmed to send the FSYNC signal to the selected MFP's RX_ID to ensure the FSYNC signal is transmitted across the link. The deserializer programming is determined by whether the frame sync is generated externally by an SoC, or internally by the deserializer.

Programming Example

External FSYNC

The following example sets up Deserializer MFP8 to accept FYSNC signal from SoC/ECU and outputs on Serializer MFP8.

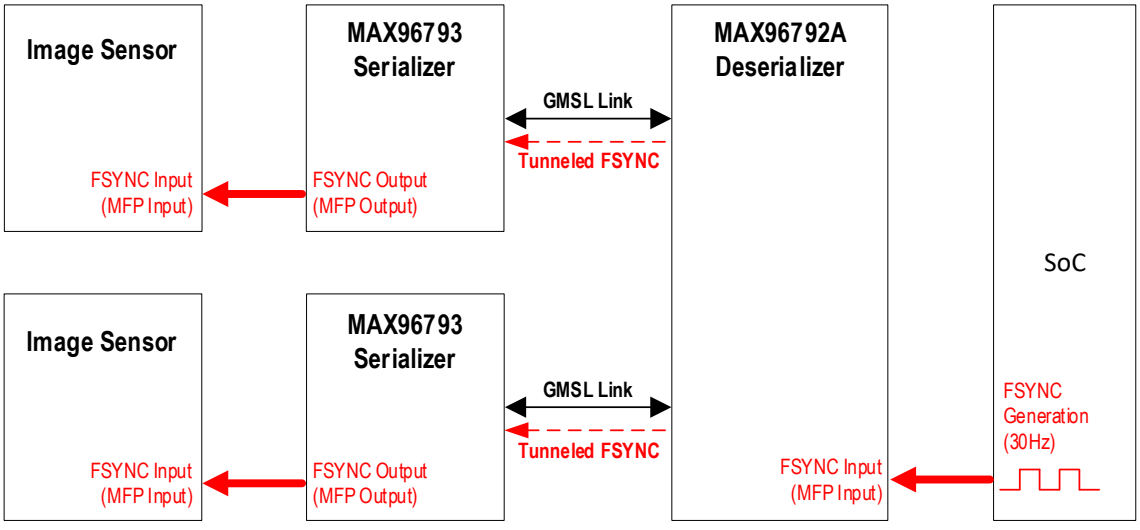

Figure 18. External Frame Sync Example

#DES I2C Address=0x98

#SER I2C Address=0x80

#SER MFP8 Outputs FSYNC

0x80,0x02D6,0x84 # GPIO RX Tunnel Enabled, 1MΩ Pull-Down

0x80,0x02D7,0x68 # Pull-up, Push-pull, TX_ID=0x08

0x80,0x02D8,0x48 # RX_ID=0x08

#DES MFP8 Accepts FYSNC Signal

0x98,0x03E0,0x08 # FSYNC mode set to external

0x98,0x03EF,0x86 # FYSNC enabled on Pipe Y and Z

0x98,0x02C8,0x83 # GPIO TX tunnel Enabled

0x98,0x02C9,0x68 # TX_ID=0x08

0x98,0x02CA,0x48 # RX_ID=0x08

Power Manager and Sleep Mode

Overview

The MAX96793 includes an integrated power manager that ensures the reliable and efficient operation of various power functions. The power manager controls the internal switched supply domains during the full sequence of power states so that the device powers up and down smoothly. During power-up, the power manager guards the device until the internal supplies have been validated and the digital core assumes normal operations. In all power modes, the power manger monitors power supplies for under and overvoltage conditions. In sleep mode, the power manager minimizes current consumption and can quickly restore device configurations after waking up.

Table 22. MAX96793 Power Manager and Sleep Mode Availability

Part Number	Power Manager	Sleep Mode
MAX96793	Supported	Supported

Device Power Operation

The power manager block minimizes required user interaction while providing extensive diagnostic indicators. Power manager status registers can be polled for valid supply levels, and a system-level interrupt (ERRB=0) can be generated in the case of a device power failure.

The MAX96793 uses common power rails (V_{DD}, V_{DD18}, and V_{DDIO}) and an integrated internal V_{DD_SW} regulator.

Note: If the power manager sends an ERRB interrupt due to a power fail condition, check PWR0, PWR1, and other diagnostic registers to identify the source of the failure. Refer to the *Voltage Monitoring* section of the respective data sheet for additional details.

Power Supplies

The MAX96793 shares a common set of power supply voltages that power universal functions such as the digital core, GMSL link circuitry, and GPIO. These power supplies are summarized as follows:

- V_{DD}: The input voltage to the V_{DD} rail can be between 1.0V and 1.2V. An Internal LDO regulates the voltage to 1.0V.
- V_{DD18}: 1.8 V power rail
- V_{DDIO}: 1.8 V or 3.3 V I/O power rail for I/O
- V_{DD_SW}: (CAP_VDD pin): Internal 1.0 V power rail that powers the digital core logic.

External power is supplied directly to V_{DD}, V_{DD18}, V_{TERM}, and V_{DDIO}, but the V_{DD_SW} (CAP_VDD pin) just has external capacitors connected.

Power Manager States

At device power-up, the power manager block automatically controls the power sequencing process. Power supplies can ramp in any order and do not need to be externally sequenced. When power is applied, the power manager senses the presence of each domain. When the voltage threshold is reached for all supplies, the power manager signals to the other device domains that power is stable and begins to transition into run mode.

The power manager state machine has four power states: boot, run, saved, and reset (power down/sleep). The power manager circuitry is in the “always-on” V_{DD18} domain so that all power domains may be managed and monitored during the full sequence of power states. This architecture allows for a seamless resume from the sleep to run mode and draws minimal current. Retention memories are also powered by the V_{DD18} domain so that device configuration and register settings can be saved and restored.

Figure 19 shows the state diagram for the power manager.

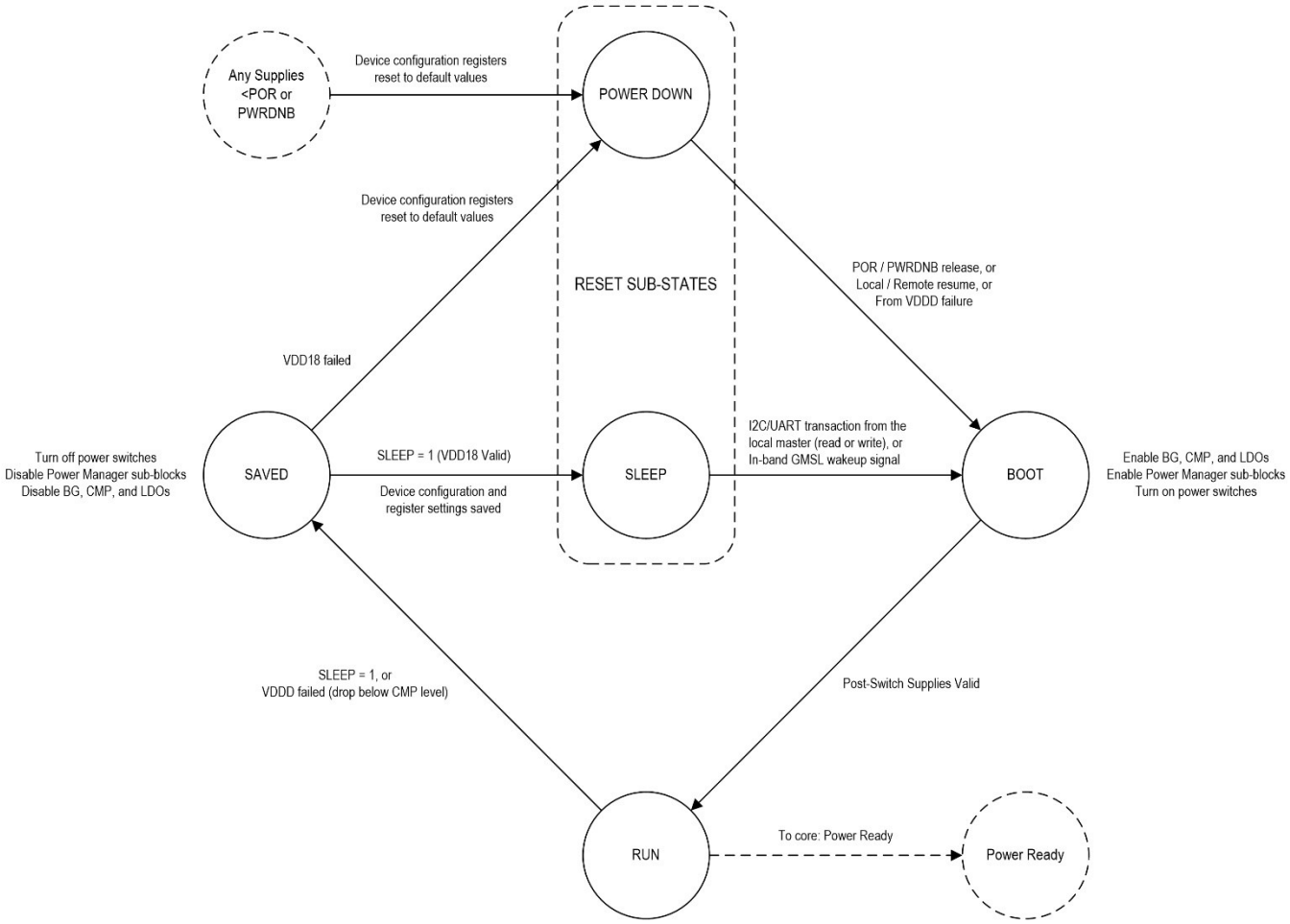

Figure 19. Power Manager State Diagrams

Reset (Power Down/Sleep)

Power down and sleep are two substates of the reset state.

The device enters the power-down state if the PWDNB pin is asserted (low), VDD_sw falls below the internally set threshold, or if any other supply falls below the associated POR value. In power down, all registers in the digital core revert to default reset values. Power failure latches are retained unless VDD18 falls too low. De-asserting PWDNB (high) releases the chip from the power-down state and into the boot state.

Sleep is a low-power consumption state that preserves the configurations and settings saved in the previous state and enables a much faster return to running operation than from power down. When the device is in the run state, the system (µC/SoC) can initiate sleep state with an I2C/UART command (SLEEP = 1). Sleep mode is entered automatically after the retention memory is loaded following the SLEEP=1 command. In the sleep state, the VDD18 supply must be continuously maintained to ensure that previous configurations and settings are preserved. It is recommended that all other supplies be maintained during sleep mode to simplify the sleep and wake-up sequences.

BOOT

The device can enter the boot state from reset after external supplies have ramped up or the device has resumed operation from sleep. In boot, all power switches are turned on, and all power manager subblocks are enabled. When all post-switch supplies are valid, the chip enters run state. The power manager has an inrush current control feature; in boot state, the core supply switches are turned on gradually.

RUN

The run state is the normal operating mode of the device. The device enters run when all power supplies to the chip are valid. Once entering this state, the crystal begins to warm up, on-board calibration is initiated, and the GMSL handshake begins the process of establishing link lock.

SAVED

Saved mode is initiated with an I^2C/UART command (SLEEP = 1) while the device is in run mode. Before the power manager enters the saved state, the core saves the current device configuration and register values to retention memory. In the saved state, all power switches are turned off and the power manager blocks are disabled. The device enters the sleep state.

Sleep Mode

Sleep mode provides a low power state from which prior configuration information is automatically loaded upon wakeup. This enables very fast recovery from low power sleep to full run operation by eliminating the need for the user to reprogram configuration registers as is required after a full power cycle.

In run mode (normal operation), writing (SLEEP=1) starts the process of saving device configuration and register settings. The power manager shuts down all internal power supplies, the clocks are disabled, and the chip enters the very low power consumption sleep state. V_{DD18} must remain stable to provide continuous power to the data retention memory, and it is recommended that all supplies be maintained in their nominal operating range.

Sleep and Wake-Up Sequences

There are two ways to enable and wake up from sleep mode. Depending on the device (remote or local to microcontroller) that is desired to be in sleep mode, the following procedures can be used.

Enable SLEEP Mode

- Remote device
  - Write SLEEP = 1 to remote device
  - Write RESET_LINK = 1 to local device (**Note**: Perform within 8ms after the SLEEP = 1 command)
- Local Device
  - Write register WAKE_EN_A = WAKE_EN_B = 0 to local device. (**Note**: This prevents the local device from being woken up from the remote side.)
  - Write RESET_LINK = 1 to local device
  - Write SLEEP = 1 to local device

Wake-Up (Exit Sleep Mode):

- Remote device
  - Write RESET_LINK = 0 to local device (or power-up/wake-up the local device)
  - Wait for LOCK = 1
  - Write SLEEP = 0 to remote device (**Note**: Perform within 8ms after LOCK = 1)
- Local device
  - Perform a dummy I2C/UART transaction (**Note**: This wakes up the device.)

- Wait 5ms
- Write SLEEP = 0 to local device
- Write RESET_LINK = 0 to local device to enable the link

If devices at both ends of a GMSL link are sleeping, the host processor must initiate the wake-up sequence by waking up the local device first and waiting for link lock. The host can then immediately disable sleep mode in the remote device.

The opposite sequence is used when transitioning devices at both ends of a GMSL link into sleep mode. The host must first configure sleep mode in the remote device while the link is locked. It can then immediately place the local device in sleep mode.

Sleep Mode Limitations

Sleep mode should not be used in conjunction with RESET_ALL.

The GMSL3 family includes a global soft reset function called RESET_ALL. This is a self-clearing reset command that is intended to reset all subsystems to their default configurations. However, if a device has previously gone through a sleep/wake cycle, issuing a RESET_ALL resets the device and erroneously loads the contents of the retention memory that had been stored when the most recent SLEEP command was executed. As a result, the device "resets" to the state that had been configured prior to entering sleep mode previously rather than recovering in a clean power-up default state. The most severe implication of this is that the (SLEEP=1) state is saved in the retention memory, so the device recovers from reset and immediately enters sleep mode.

Due to this described behavior, RESET_ALL and sleep should never be used together.

Not All Registers are Saved in Retention Memory

Most key registers corresponding to common device configuration are saved in retention memory. However, applications requiring extensive low-level configuration or infrequently used features may require writing to registers that are not saved in retention after resume from sleep. In these cases, full recovery from sleep mode to the pre-sleep device state requires some repeated register configuration following resume from sleep. Registers that are stored in retention memory are marked with "*" in the register map in the data sheet.

Register CRC

Overview

This device includes a register CRC to alert if the device is accidentally placed into an undesired state. This is done by calculating a CRC value based on the state of the specific control registers' values. These control registers program certain device and system parameters such as, but not limited to, GMSL link speed, MIPI port configuration, and GPIO configuration. If any of these parameters are changed mid-operation, the device configuration changes, and the ERRB output of the device is activated.

The period within the CRC calculations is programmable from 2ms to ~500ms.

Usage Models

There are two usage modes to employ register CRC (basic and rolling). There is also a register block to skip specific registers from CRC calculation.

Basic CRC

The basic mode simply calculates a CRC value during each 'CRC_PERIOD' and checks that this value remains unchanged. The calculated CRC value is deposited in the REGCRC_MSB/LSB registers. Errors are indicated on the interrupt pin, which can be enabled or disabled using the standard interrupt mechanism.

Rolling CRC

The rolling CRC mode incorporates a rotating 2-bit counter that is incremented each 'CRC_PERIOD' so that the CRC value changes through four values in a repeating fashion. The calculated CRC value is deposited in the REGCRC_MSB/LSB registers. In this mode, poll the CRC value to verify the CRC value is cycling through these four values and has not stopped working. ERRB is not indicated as the CRC value is changing periodically, and the usage mode is to have the user poll the CRC register.

Skipping Registers from CRC Calculation

If desired, registers can be removed from the CRC calculation using SKIPX_MSB[7:0] and SKIPX_LSB[7:0]; where X = 0 to 7.

Note: The register CRC protection mechanism must be enabled as the final function on the chip (including all interrupt ERRB flags); no register writes to CRC protected registers can occur after enabling the register CRC, or the CRC is corrupted.

System Implementation

Follow these steps to enable the Register CRC feature.

1. Configure SerDes link per use case. Program any 'SKIP' registers to avoid CRC calculation.
   Example: Skipping GPIO_C register 0x02D8: SKIP0_MSB=0x02, SKIP0_LSB=0xD8.
2. Enable video.
3. Enable the REG CRC feature.
    a. Basic CRC
        i. Set REG_CRC_ERR_OEN = 1b'1.
        ii. Set I2C_WR_COMPUTE = 0b'1.
        iii. Set CRC_PERIOD[7] = 1'b1 (corresponds to a CRC period of ~250 ms).
        iv. Set PERIODIC_COMPUTE = 1b'1.
        v. Set CHECK_CRC bit = 1b'1.
        vi. Read calculated CRC MSB/LSB and store.
        Example: 'REGCRC_Original', REGCRC_MSB[7:0], REGCRC_LSB[7:0]
    b. Rolling CRC
        i. Set GEN_ROLLING_CRC = 1'b1.
        ii. Follow steps for basic CRC mode.
            1. Read calculated CRC MSB/LSB and store.
            Example: 'REGCRC_Original', REGCRC_MSB[7:0], REGCRC_LSB[7:0].

2. There are four individual CRCs calculated automatically that should be checked per CRC_PERIOD.

4. *If REG_CRC_ERR_FLAG only asserts, do the following:
    a. Rewrite register values to get back to the original state.
    b. Read calculated CRC MSB/LSB.
        i. If new calculated CRC MSB/LSB = ‘REGCRC_Original’, registers are back to the original state.
            1. Do RESET_CRC = 1’b1.
            2. Continue system operation.
        ii. If new calculated CRC MSB/LSB ≠ ‘REGCRC_Original’, a different register must have been written.
            1. Restart the link.
            2. Reconfigure the SerDes link.

*Events that trigger REG_CRC_FLAG may also simultaneously trigger other ERRB flags. Prioritize error handling accordingly.

Reference over Reverse

Overview

All GMSL3 parts must receive an external clock for full function (typically through a crystal). Some serializers can get a reference clock through reference clock over reverse channel (RoR). In RoR, there is no need to connect a crystal next to the serializer because the deserializer is supplying the reference signal on the reverse channel of the GMSL link.

Using RoR instead of the crystal oscillator (XTAL) provides several advantages, including:

- Reduced system cost
- Increased reliability
- Reduced board area
- Simplified board layout

Note: RoR is a serializer function. Check the serializer user guide for more information.

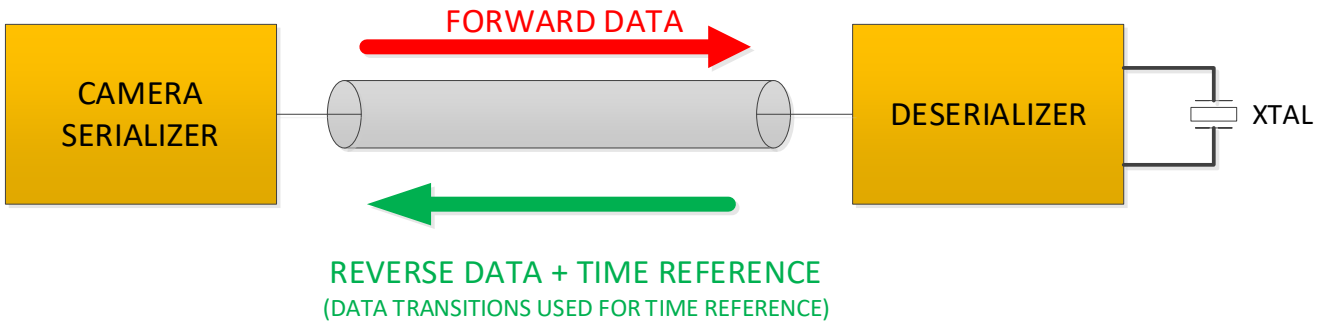

Figure 20. RoR Block Diagram

The MAX96793 supports RoR mode and can receive reference signal from a companion deserializer that also supports RoR mode.

Enabling RoR Mode by CFG Pins

The MAX96793 device selects XTAL or RoR mode by CFG0 setting. Deserializer automatically detects that RoR is enabled and sends reference signal to companion serializer, no additional deserializer configuration is required.

Enabling RoR with Register Writes

The RoR mode can be enabled by register writes. This is useful for evaluating performance in RoR mode if the board is already set up with a crystal. The procedure is as follows and assumes I^2C access on the deserializer side only.

To switch from crystal mode to RoR mode:

- Bring up the GMSL link in crystal mode per normal use case.
- Write serializer bit XTAL_PU=0 (bit 0 in register 0x4) to disable the serializer crystal driver. This results in the GMSL link losing lock and automatically relocking in RoR mode. It is not necessary to issue a reset link command.
- Reading of serializer bit ROR_CLK_DET (bit 5 in register 0x14AA) now shows a logic 1, indicating RoR mode is active.

To switch from RoR mode back to crystal mode:

- Write SER bit XTAL_PU=1 to enable the serializer crystal driver. This results in the GMSL link losing lock and automatically relocking in crystal mode. It is not necessary to issue a reset link command.
- Reading of serializer bit ROR_CLK_DET (bit 5 in register 0x14AA) now shows a logic 0, indicating RoR mode is no longer active.

GMSL Link Lock in RoR Mode

The GMSL link lock time in the RoR mode is same as crystal mode. Refer to the device data sheet for lock time specifications.

Hardware Considerations

When RoR mode is enabled and a crystal is not used, the X1 and X2 pins on the serializer may be left unconnected. It is not necessary to drive the X1 or X2 pins to any specific logic level because they are internally disabled when not used. However, for customers who prefer to connect the X1 and X2 pins, they may be tied directly to ground.

For the MAX96793 devices being operated in GMSL3-12Gbps and in RoR mode, Vref pin requires an 18nF or 22nF ±10% capacitor to ground.

RoR Jitter Considerations

While the RoR jitter does affect the serializer transmit signal, the jitter bandwidth is limited and can be tracked by the clock recovery circuit in the deserializer. The deserializer clock input source must meet the data sheet requirements for reference clock input jitter.

Spread Spectrum Clocking

Spread Spectrum Clocking (SSC) is an alternate configuration for the GMSL link that helps with EMI. SSC is supported in RoR mode. If it is used, it is enabled only in the deserializer. The serializer clock follows the deserializer clock modulation and consequently, the serializer operates with a spread spectrum clock as well.

Recovery After Loss of GMSL Link Lock

The PLL used to recover the clock from the reverse channel has a narrow bandwidth of approximately 1MHz. The narrow bandwidth makes it robust to glitches on the link. However, if the clock reference is lost, the link automatically resets and the RoR synchronization sequence is repeated to regain link lock. The time needed to relock is the same as the initial lock time.

RCLKOUT Setup

Overview

Instead of having separate crystal oscillators for the serializer and image sensor, it is recommended to just use one crystal or RoR and feed that clock signal from the serializer to the image sensor. RCLKOUT or DPLL_OUT is the name of a clock signal that gets sent from the serializer to the image sensor. RCLKOUT is 25MHz and can be divided by 2 or 4. DPLL_OUT is programmable from 1MHz to 75MHz.

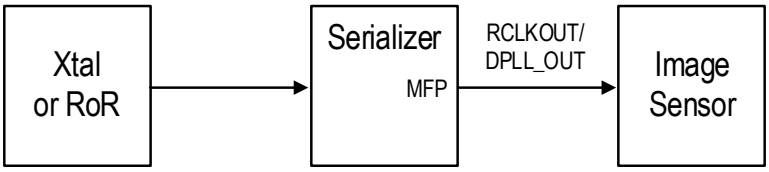

Figure 21. Serializer RCLKOUT/DPLL_OUT Clock Diagram

Following are the descriptions of the major blocks used for serializer clocking.

Xtal/OSC: In crystal mode, this block receives the 25MHz clock from a crystal by X1/OSC and X2 pins. The clock is then forwarded to the PLL/CMU block.

RoR CLOCK RX: In RoR mode, this block extracts the clock from the GMSL reverse channel. The clock is then forwarded to the PLL/CMU block.

DIVIDER/DPLL: This block has two methods of generating an output clock signal: Divider mode and DPLL mode. In Divider mode, RCLKOUT/DPLL_OUT pin can be programmed to /1, /2 or /4 of the 25MHz input reference. In DPLL mode, RCLKOUT/DPLL_OUT pin can be programmed to any frequency from 1MHz to 75MHz. Serializer RCLKOUT/DPLL_OUT is disabled by default and can be enabled with a register write.

RCLKOUT/DPLL_OUT frequency is always relative to the input reference frequency. If the clock source's frequency stays within the limits, then the RCLKOUT/DPLL_OUT frequency stays within the limits. (Refer to the specific device's data sheet for the limits.)

For the MAX96793, RCLKOUT/DPLL_OUT is available from MFP4, or MFP2 as an alternate.

Path from XTAL/RoR to RCLKOUT/DPLL_OUT

In *Figure 22*, a partial block diagram of the MAX96793 is shown. There are connections for the crystal oscillator, the clock signal output for the image sensor, and the GMSL link (SIOP and SION).

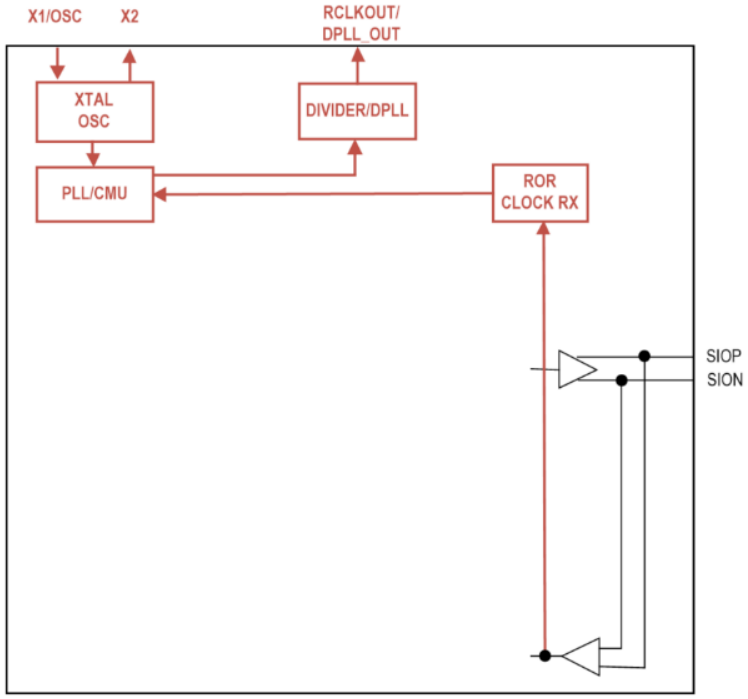

Figure 22. Functional Block Diagram of XTAL and RoR

There are four possible cases of using a crystal or RoR mode to generate the reference clock RCLKOUT/DPLL_OUT.

Figure 23 shows the path of using a crystal as the clock source and divider to generate the reference clock.

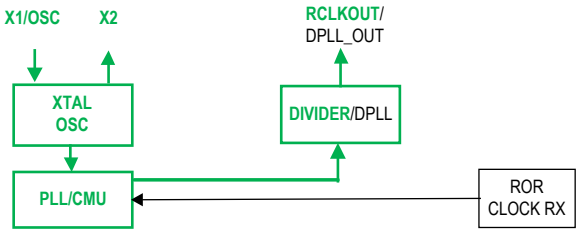

Figure 23. XTAL/OSC to RCLKOUT

The following figure shows the path of using a crystal as the clock source and DPLL to generate the reference clock.

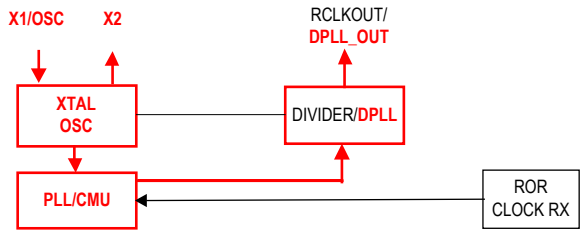

Figure 24. XTAL/OSC to DPLL_OUT

Another setup is RoR mode as the clock source and the divider generating the reference clock.

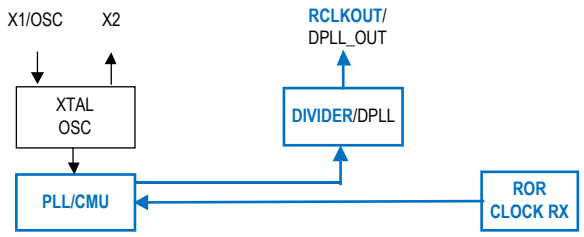

Figure 25. RoR to RCLKOUT

And, finally the setup of using RoR mode as the clock source and DPLL to generate the reference clock.

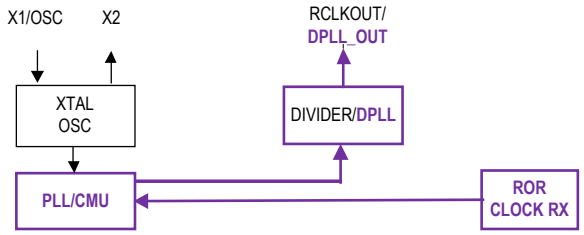

Figure 26. RoR to DPLL_OUT

Setting Serializer into XTAL or RoR Mode

To choose XTAL or RoR mode, choose the appropriate resistor values connected to the CFG0 pin as shown in *Table 23*. All deserializers that support RoR mode automatically detect and generate RoR with no additional deserializer configuration required.

If the deserializer has multiple PHYs connected to different serializers, it is possible to operate the system in a mixed configuration with some serializers in RoR mode and some in crystal mode. The RoR enabling procedure is the same as in a single PHY case.

Table 23. MAX96793 CFG0 RoR vs. XTAL Settings

CFG0 Value	RoR or XTAL
0	RoR

1	RoR
2	XTAL
3	XTAL
4	RoR
5	RoR
6	XTAL
7	XTAL

The MAX96793 RoR-mode status can be read back from serializer bit ROR_CLK_DET (bit 5 in register0x14AA). Logic 1 indicates RoR mode is active and logic 0 indicates RoR mode is not active.

Recovery After Loss of GMSL Link Lock

When the serializer is in RoR mode and the GMSL link lock is not established for any reason, RCLK/DPLLOUT frequency will be approximately 10% lower than the programmed frequency. For example, if RCLK/DPLLOUT is programmed to 25MHz, when the link is not locked the output will be 22.5MHz. The reduced frequency results from the Voltage Controlled Oscillator (VCO) input being held at a constant voltage allowing the CMU to continue oscillating at approximately a 10% reduced frequency. Once the GMSL link is locked in RoR mode, the RCLK frequency returns to the programmed 25MHz.

Upon loss of GMSL link lock, the RCLKOUT/DPLL_OUT frequency initially increases approximately 16% for roughly 150µs, and then settles to the unlocked 10% reduced frequency of 22.5MHz. This behavior is shown in *Figure 27*.

If the RCLKOUT frequency is out of the image sensor's tolerance range, it may impact the operation of the image sensor, and the image sensor may require reinitialization.

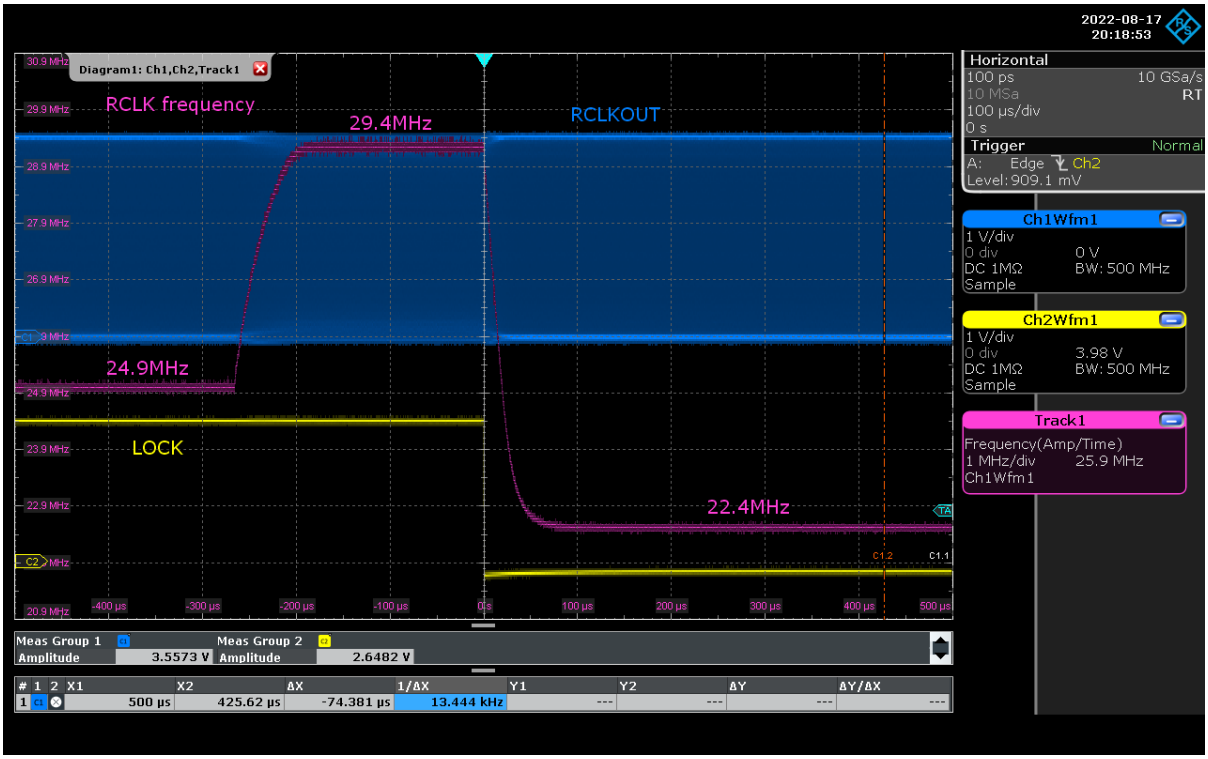

Figure 27. 25MHz RCLK to 16% Frequency Overshoot (29MHz) and then to 10% Reduced Frequency (22.5MHz)

SSC with RCLKOUT

The serializer RCLKOUT output reference is also the frequency modulated in the same manner as the GMSL link.

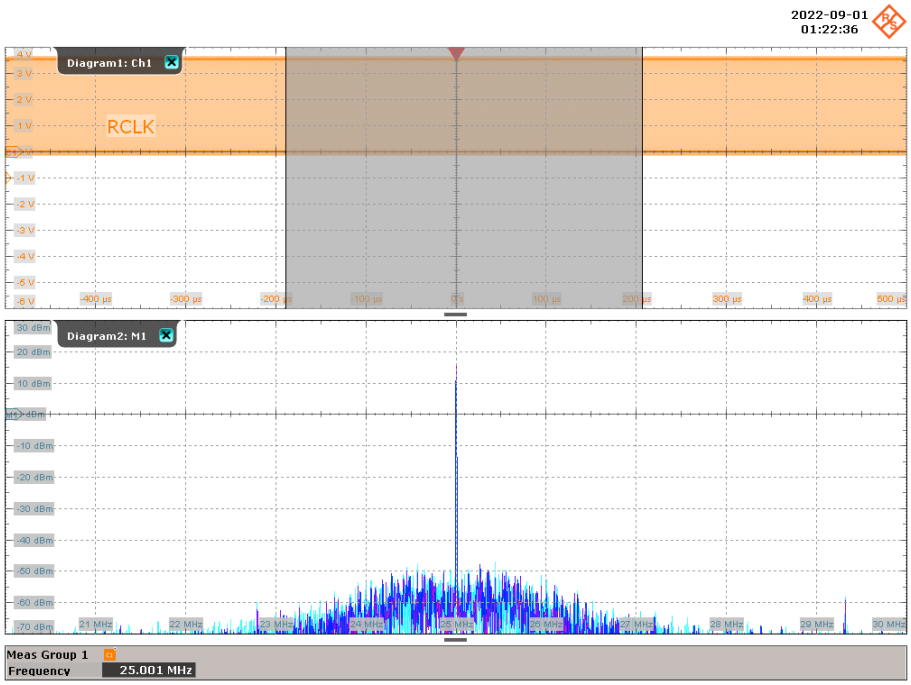

Figure 28. 25MHz RCLKOUT Signal using RoR Without SSC Feature Enabled

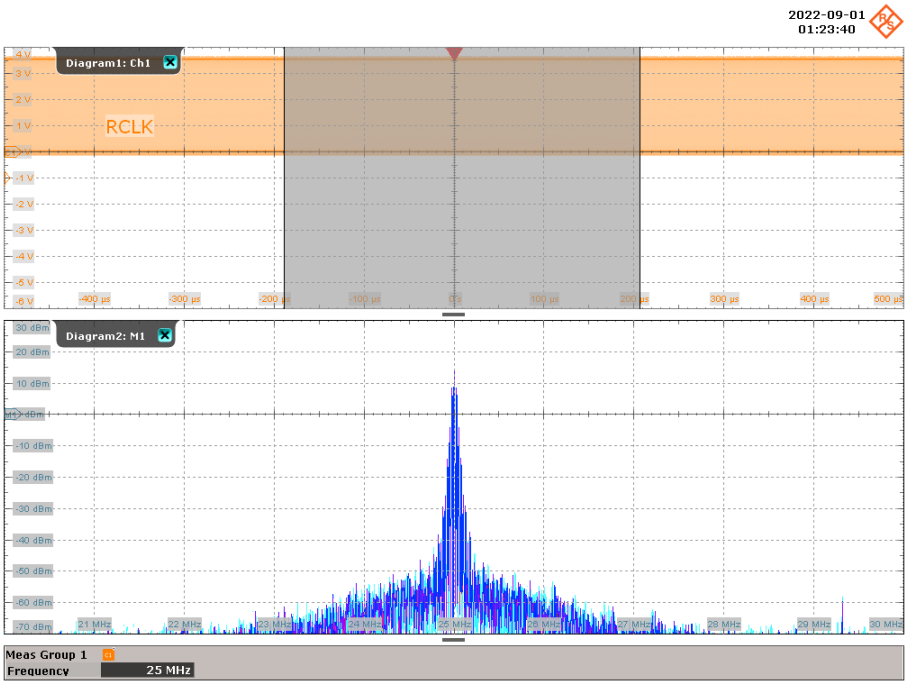

Figure 29. 25MHz RCLKOUT Signal using RoR with SSC Feature Enabled

Turning on RCLKOUT/DPLL_OUT

Set slew rate to maximum in register 0x56F if using MFP2 and in register 0x570 if using MFP4. In Register 0x03, set bit 2 low to use MFP4, or set it high to use MFP2. In register 0x06, set bit 5 high to enable RCLKOUT.

ADC Voltage Monitoring

GMSL Serializer features a 10-bit analog-to-digital converter (ADC) with an analog input multiplexer. The multiplexer selects a single-ended input channel from external input lines (MFP3/5/6), internal power supply monitors, and a temperature monitor.

High-Level Features

Following is a list of features of the ADC for the MAX96793.

- Programmable ADC voltage reference options
  - Internal thermally corrected 1.25V reference (preferred)
  - Internal VDD18 voltage rail
  - External voltage reference via the VREF pin
- Programmable input multiplexer
  - Three external voltages (MFP3/5/6)
  - Three internal voltages (CAP_VDD, VDDIO, VDD18)
  - Die temperature monitor within 0.5 Kelvin resolution
- Programmable modes of operation
  - On-demand monitoring of voltages and temperature
  - Continuous round-robin monitoring of voltage and temperature
- Programmable under and overvoltage/temperature limit thresholds (up to eight channels)
- Programmable internal and external voltage scaling
- ADC accuracy BIST and ability to verify GPIO Input MUX functionality

Typical ADC Flow of Operation

The general steps of setting up the ADC for operation are listed in the following section. Each portion is expanded on in the *Details of Operation* section.

- ADC Setup
  - Power up ADC
  - Select voltage reference
- Set up HI/LO channel limits (if desired)
- On-Demand ADC read or enable round-robin state machine
- Shutdown ADC

Details of Operation

This section is a descriptive overview. Actual use case examples with direct register writes appear past this section in the *Examples of ADC Operation* section.

ADC Setup

Setting up the ADC comprises shutting down the ADC, powering up the ADC, and selecting the voltage reference to be used.

ADC Shutdown

It is recommended to shut down the ADC prior to setup so the ADC is in a known state before programming the register map for the desired configuration. The steps in the following table are relevant for the shutdown of the round-robin state machine as well as the ADC.

Table 24. MAX96793 ADC Shutdown Flow

Step	Action	Register Address	Bitfield Name	Bits	POR	Decode
1	Disable round robin state machine	0x534	adc_rr_run	0	0b0	0b0 – Hold round-robin state machine in idle 0b1 – Run round-robin state machine
2	Power down the ADC	0x500	adc_pu	1	0b0	0b0: ADC powered off 0b1: ADC powered on
3	Power down the input buffer	0x500	buf_pu	2	0b0	0b0: ADC internal buffer off 0b1: ADC internal buffer on
4	Power down the reference buffer	0x500	adc_refbuf_pu	3	0b0	0b0: ADC reference buffer off 0b1: ADC reference buffer on
5	Power down the ADC charge pump	0x500	adc_chgpump_pu	4	0b0	0b0: ADC charge pump off 0b1: ADC charge pump on
6	Disable the clock	0x501	adc_clk_en	3	0b0	0b0: ADC clock disable 0b1: ADC clock enable

ADC Power Up

The following table shows the relevant registers necessary to apply power to the ADC and enable operation. Before the ADC can be utilized, the power manager for the serializer must have VDD, VDD18 and VDDIO supplies available and the link between the serializer and deserializer must be established.

After the power supplies have been validated, the ADC clock must be enabled, and power-up controls must be applied. Note that the ADC's charge pump requires a 10µs delay from initial power-up before it is in steady state and available for use. The interface detects when the ADC circuits are enabled and sets the ADC ready interrupt flag (ADC_INTRIE0.adc_ref_ready_ie, ADC_INTR0.adc_ref_ready_if); once the delay has passed, the ADC is ready for use.

Table 25. MAX96793 ADC Power Up Flow

Step	Action	Register Address	Bitfield Name	Bits	POR	Decode
1	Enable Global ADC Interrupt	0x1E	ADC_INT_OEN	2	0b0	0b0: Reporting disabled 0b1: Reporting enabled
2	Enable ADC clock	0x501	adc_clock_en	3	0b0	0b0: ADC clock disable 0b1: ADC clock enable

3	Select 1.25V or 1.8V internal reference (not used for external reference)	0x501	adc_refsel	2	0b0	0b0 – Use the internal 1.25V as reference 0b1 – Use VDD18 as reference
4	Select internal or external reference	0x502	adc_xref	1	0b0	0b0: Internal reference for ADC 0b1: External reference for ADC
5	If using VDD18/2 internal reference, enable voltage correction bypass (not used for external reference)	0x509	bypass_volttempt_corr	7	0b0	0b0 :Do not bypass 0b1: Bypass
6	Enable ADC ready interrupt	0x50C	adc_ref_ready_ie	1	0b0	0b0: Reporting disabled 0b1: Reporting enabled
7	Enable ADC done interrupt	0x50C	adc_done_ie	0	0b0	0b0: Reporting disabled 0b1: Reporting enabled
8	Enable ADC calibration done interrupt	0x50C	adc_calDone_ie	7	0b0	0b0: Reporting disabled 0b1: Reporting enabled
9	Clear ADC interrupts	0x510 0x511 0x512 0x513	ADC.INTR0 ADC.INTR1 ADC.INTR2 ADC.INTR3	7:0 7:07:07:0	0x000x00 0x00 0x00	ADC Interrupt reporting, clear on read
10	Power up the ADC charge pump	0x500	adc_chgpump_pu	4	0b0	0b0: ADC charge pump off 0b1: ADC charge pump on
11	Power up the ADC	0x500	adc_pu	1	0b0	0b0: ADC powered off 0b1: ADC powered on
12	Power up the ADC reference buffer (not needed for external reference)	0x500	adc_refbuf_pu	3	0b0	0b0: ADC reference buffer off 0b1: ADC reference buffer on
13	Power up the ADC internal buffer	0x500	buf_pu	2	0b0	0b0: ADC internal buffer off

						0b1: ADC internal buffer on
14	Wait for ready interrupt to be asserted	0x510	adc_ref_ready_if	1	0b0	0b0: Flag cleared 0b1: ADC reference ready, cleared on read
If using internal 1.25V thermally corrected reference (see the following sections)						
15	Initialize a temperature conversion	0x1D28	RUN_TMON_CAL	1	0b0	0b0: Do not run 0b1: Run, cleared on write
16	Wait for ADC done interrupt to assert	0x510	adc_done_if	0	0b0	0b0: Flag cleared 0b1: ADC conversion done
17	Wait for calibration done interrupt to assert	0x510	adc_calDone_if	7	0b0	0b0: Flag cleared 0b1: ADC accuracy calibration done

On-Demand Conversions

After the ADC is powered up and a voltage reference has been configured, the user can use the ADC in whichever configuration is desired. On-demand ADC readings allow the user to dictate what input is read and when to start an ADC conversion. To initiate an On-Demand read, the user needs to set up the input MUX, initiate the ADC conversion, and then read the output data from the conversion.

Temperature Readings

To take a die temperature measurement, the internal 1.25V thermally corrected voltage reference must be used. The relevant registers to read the internal die temperature are shown in *Table 26*.

Table 26. MAX96793 On-Demand Temperature Reading

Step	Action	Register Address	Bitfield Name	Bits	POR	Decode
1	Initiate temperature conversion	0x1D28	RUN_TMON_CAL	0	0b0	0b0: Do not run 0b1: Run, cleared on write
2	Wait for ADC done interrupt to assert	0x510	adc_done_if	0	0b0	0b0: Flag cleared 0b1: ADC conversion done
3	Wait for calibration done interrupt to assert	0x510	adc_calDone_if	7	0b0	0b0: Flag cleared 0b1: ADC accuracy calibration done
4	Read internal die temperature registers	0x1D3B 0x1D3C	T_EST_OUT_B0 T_EST_OUT_B1	7:0 7:6	0xFF 0b11	Bits 7:0 of TMON temp Bits 9:8 of TMON temp

5	Read alternate internal die temperature registers (optional)	0x1D3C 0x1D3D	ALT_T_EST_OUT_B1 ALT_T_EST_OUT_B0	1:0 7:0	0b11 0xFF	Bits 9:8 of alt TMON temp Bits 7:0 of alt TMON temp

Reading the registers 0x1D3B and 0x1D3C provide the 10-bit reading of the die temperature, which is split across these two registers. Use the following equation to convert the T_EST_OUT[9:0] value to die temperature:

$$\frac{T\_EST\_OUT[9:0](Decimal)}{2} = Die\ Temp\ (K) - 273.15\ = Die\ Temp\ (℃)$$

In addition to this die temperature reading, the redundant die temperature reading can be found by reading registers 0x1D3C and 0x1D3D. Using the following equation provides the die temperature as read by the redundant temperature monitor:

$$\frac{ALT\_T\_EST\_OUT[9:0](Decimal)}{2} = Die\ Temp\ (K) - 273.15\ = Die\ Temp\ (℃)$$

Internal Voltage Reading

The MAX96793 provides the ability to measure the three internal voltage rails: VDDIO, VDD18 and CAP_VDD with the ADC. To take these measurements, the MUX must be set up to read the desired internal voltage prior to initiating the ADC conversion. After taking the ADC conversion, it is recommended to disable the input MUX to create a "Break before Make" functionality that avoids shorting two inputs momentarily.

Table 27. MAX96793 On-Demand Internal Voltage Reading

Step	Action	Register Address	Bitfield Name	Bits	POR	Decode
1	Set input channel to desired input channel	0x501	adc_chsel	7:4	0x0	0x8 – VDDIO/4 0x9 – VDD18/2 0xA – CAP_VDD/2
2	Enable channel multiplexer	0x502	Inmux_en	0	0b0	0b0: Input MUX is open 0b1: MUX selected by adc_chsel field
3	Clear ADC interrupts	0x510 0x511 0x512 0x513	ADC.INTR0 ADC.INTR1 ADC.INTR2 ADC.INTR3	7:0 7:07:07:0	0x00 0x00 0x00 0x00	ADC interrupt reporting, clear on read
4	Start ADC conversion	0x500	cpu_acd_start	0	0b0	0b0: Conversion complete 0b1: Start ADC conversion
5	Wait for ADC done interrupt	0x510	adc_done_if	0	0b0	0b0: Flag cleared 0b1: ADC conversion done
6	Read ADC result	0x508 0x509	adc_data_l adc_data_h	7:0 1:0	0x00 0b00	Bits 7:0 of 10b ADC data Bits 9:8 of 10b ADC data
7	Open input multiplexer	0x502	Inmux_en	0	0b0	0b0: Input MUX is open

						0b1: MUX selected by adc_chsel field

External Voltage Reading

On the MAX96793 external voltages can be monitored on MFP3, MFP5 and MFP6 using the inputs ACD0, ADC1 and ADC2, respectively. The process to take an external voltage reading is like that of an internal voltage, but it is important to note that the ADC0/1/2 must also be enabled by writing to 0x53E. In addition to this, ensure that the voltage divider is used if attempting to measure a voltage above the reference voltage used. *Table 28* shows the sequence to monitor the voltage at any of the three MFPs:

Table 28. MAX96793 On-Demand External Voltage Reading

Step	Action	Register Address	Bitfield Name	Bits	POR	Decode
1	Set input channel to desired input channel	0x501	adc_chsel	7:4	0x0	0x0 – ADC0 (MFP3) 0x1 – ADC1 (MFP5) 0x2 – ADC2 (MFP6)
2	Enable channel multiplexer	0x502	Inmux_en	0	0b0	0b0: Input MUX is open 0b1: MUX selected by adc_chsel field
3	Set voltage divider as needed	0x502	adc_div	3:2	0b00	0b00 – Divide by 1 0b01 – Divide by 2 0b10 – Divide by 3 0b11 – Divide by 4
4	Enable monitoring of MFP with ADC	0x53E	adc_pin_en	2:0	0b000	0bXX1 – Enable ADC0 0bX1X – Enable ADC1 0b1XX – Enable ADC2
5	Clear ADC interrupts	0x510 0x511 0x512 0x513	ADC.INTR0 ADC.INTR1 ADC.INTR2 ADC.INTR3	7:0 7:0 7:0 7:0	0x00 0x00 0x00 0x00	ADC interrupt reporting, clear on read
6	Start ADC conversion	0x500	cpu_acd_start	0	0b0	0b0: Conversion complete 0b1: Start ADC conversion
7	Wait for ADC done interrupt	0x510	adc_done_if	0	0b0	0b0: Flag cleared 0b1: ADC conversion done
8	Read ADC result	0x508 0x509	adc_data_l adc_data_h	7:0 1:0	0x00 0b00	Bits 7:0 of 10b ADC data Bits 9:8 of 10b ADC data
9	Open input multiplexer	0x502	Inmux_en	0	0b0	0b0: Input MUX is open 0b1: MUX selected by adc_chsel field

Converting from the 10-bit ADC reading to the corresponding voltage uses the same equation as internal voltages. Again, it is important to use the correct reference voltage value and divider value. For external voltages, the Divider value is what was set in the adc_div bitfield (1, 2, 3 or 4). If using the VDD18 voltage rail as the ADC reference, use 900mV as the value for $V_{ref.}$ The equation is as follows:

$$Voltage = \frac{ADC\ value\ (Decimal) \times V_{ref}}{1023} \times Divider$$

Round-Robin Conversion

To use the round-robin state machine, first shut down the ADC and then configure it to use the desired voltage reference. Next, HI/LO channel thresholds should be configured for the channels that are going to be monitored. These thresholds can be set for the three internal voltages, three external voltages, and the die temperature. After configuring the HI/LO thresholds for the desired channels, follow these steps to enable the round-robin state machine.

Table 29. MAX96793 Round-Robin Setup Flow

Step	Action	Register Address	Bitfield Name	Bits	POR	Decode
After configuring HI/LO limits for all desired channels (shown in "Channel HI/LO Limits" section)						
1	*(Optional)* Run ADC Accuracy BIST	0x1D28	RUN_ACCURACY	2	0b0	0b0 – Do not run 0b1 - Run
2	Set number of conversion cycles between ADC conversion	0x536 0x537	adc_rr_sleep_l adc_rr_sleep_h	7:0 7:0	0x00 0x00	Bits 7:0 of ADC conversion cycles Bits 15:8 of ADC conversion cycles
3	Enable round robin state machine	0x534	adc_rr_run	0	0b0	0b0 – Hold round robin state machine in idle 0b1 – Run round robin state machine

Once the round-robin state machine is enabled, monitor the 0x510 register for a HI_LIMIT or LO_LIMIT interrupt flag. For a HI_LIMIT flag, use register 0x511 to see the specific channel(s) with an over limit detected. For a LO_LIMIT flag, use register 0x512 to see the specific channel(s) with an under limit detected. These over/under limits assert the ADC interrupt in the 0x50C register, if configured to do so.

The order in which the round-robin state machine executes its conversions is shown in the following table:

Table 30. MAX96793 Round-Robin Timing

Step	Action	Estimated Time (µs)	Comment
1	Temperature Monitor Update	860	
2	Channel 0	430	
3	ADC Accuracy	1750	If enabled
4	Channel 1 through 7	430	
5	Sleep Mode	SleepCount * 430	SleepCount from 0 to 65535, programmed in registers 0x536 and 0x537

Channel HI/LO Limits

To simplify and minimize power consumption during power supply monitoring, the ADC supports programmable data limits and interrupt enables for up to eight (8) channels. If the converted level for the programmed channel has crossed the enabled threshold, an interrupt is generated.

The ADC output is compared to a limit (ChxHiLimit) when a selected channel (ChxSel) is sampled. If the detector is enabled (ChxHiLimitEn) and the over-limit interrupt is enabled, an interrupt is generated. This same capability is provided for the under-range limit detector.

Setting up the channel HI/LO limits is the first step to programming the ADC for round robin operation. Start by determining the desired voltage or temperature limits and using the following equations to determine the hexadecimal equivalent for the HI/LO channel limit registers.

Internal/External Voltage Threshold Formula

To calculate the thresholds for internal or external voltages, use the following formula to get the HI/LO threshold value:

$$\frac{Voltage\ Threshold\ \times 1023}{V_{ref} \times Divider} = ADC\ value\ (Decimal)$$

Internal Die Temperature Threshold Formula

Calculating the temperature thresholds is device specific and based on six EFUSE registers. The following equations are required and require the device's EFUSE registers to get the correct HI and LO threshold values.

EFUSE registers to read: 0x1C2D, 0x1C2E, 0x1C26, 0x1C27, 0x1C2A, 0x1C2B.

Using the previously read EFUSE registers, calculate the values of DADC_THOT, VBG_THOT and THOT:

$$DADC\_THOT = 0x1C2D[7:0] + (0x1C2E[6:0] \times 256)$$

$$VBG\_THOT = 0x1C26[7:0] + (0x1C27[6:0] \times 256)$$

$$THOT = 0x1C2A[7:0] + (0x1C2B[3:0] \times 256)$$

Next values of ADC_HOT and TRIM_TEMP are calculated as follows:

$$ADC\_HOT = Round\left(\frac{2^{24}}{DADC\_THOT} \times \frac{VBG\_THOT}{20{,}480}\right)$$

$$TRIM\_TEMP\ (Kelvin) = \frac{THOT}{4}$$

Finally, the HI/LO threshold values (in decimal) can be calculated using the previous values and the value of the adc_scale[0] bit found in register 0x501:

$$ADC\ TEMP\ REG = Round\left(\frac{ADC\_HOT}{TRIM\_TEMP} \times Desired\ Temperature\ Limit\ in\ Kelvin\right)/2^{adc\_scale}$$

Writing 10-Bit HI/LO Thresholds

The 10-bit HI/LO voltage thresholds are stored across multiple registers, so it is necessary to correctly break up each threshold when writing to the ADC_LIMIT_0, ADC_LIMIT_1 and ADC_LIMIT_2 registers. The low 8 bits of the LO threshold are written to the ADC_LIMIT_0 register. The ADC_LIMIT_1 register comprises the upper 2 bits of the LO threshold in the [1:0] bit position and the lower 4 bits of the HI threshold in the [7:4] position. Lastly, the ADC_LIMIT_2 register comprises the upper 6 bits of the HI threshold in the [5:0] positions. An example is shown in the following lines:

Calculated LO threshold = 0x260

Calculated HI threshold = 0x2E6

ADC_LIMIT_0 register = 0xXX

ADC_LIMIT_1 register = 0bXXXXXXXX

ADC_LIMIT_2 register = 0bXXXXXXXX

Table 31. Channel HI/LO Limit Register Setup

Step	Action	Register Address	Bitfield Name	Bits	POR	Decode
1	Program the 10b LO threshold	ADC_LIMIT<***CH***>_0 ADC_LIMIT<***CH***>_1	chLoLimit_l<***CH***> chLoLimit_h<***CH***>	7:0 1:0	0x00 0b00	Bits 7:0 of 10b LO limit Bits 9:8 of 10b LO limit
2	Program the 10b HI threshold	ADC_LIMIT<***CH***>_1 ADC_LIMIT<***CH***>_2	chHiLimit_l<***CH***> chHiLimit_h<***CH***>	7:4 5:0	0x0 0x00	Bits 3:0 of 10b HI limit Bits 9:4 of 10b HI limit
3	Program the multiplexer input channel	ADC_LIMIT<***CH***>_3	ch_sel<***CH***>	3:0	0x00	0x0 – ADC0 (MFP3) 0x1 – ADC1 (MFP5) 0x2 – ADC2 (MFP6) 0x8 – VDDIO/4 0x9 – VDD18/2 0xA – CAP_VDD/2 0xB – Die Temperature
3a	Enable monitoring of ADC0, ADC1 or ADC2 (if monitoring MFP3/5/6)	0x53E	adc_pin_en	2:0	0x00	0bXX1 – Enable ADC0 0bX1X – Enable ADC1 0b1XX – Enable ADC2
4	Program the GPIO divider setting if needed	ADC_LIMIT<***CH***>_3	div_sel<***CH***>	4:5	0b00	0b00 – Divide by 1 0b01 – Divide by 2 0b10 – Divide by 3 0b11 – Divide by 4

5	Enable the channel HI interrupt	0x50D	ch<***CH***>_hi_limit_ie	7:0	0x00	0b1 – Enable Channel X interrupt in the [X] bit position
6	Enable the channel LO interrupt	0x50E	ch<***CH***>_lo_limit_ie	7:0	0x00	0b1 – Enable Channel X interrupt in the [X] bit position
7	Enable the global HI interrupt	0x50C	adc_hi_limit_ie	2	0b0	0b0 – ADC interrupt source disabled 0b1 – HI limit monitor asserts ADC interrupt
8	Enable the global LO interrupt	0x50C	adc_lo_limit_ie	3	0b0	0b0 – ADC interrupt source disabled 0b1 – LO limit monitor asserts ADC interrupt

Internal Testing

The MAX96793 has the capability to internally test various portions of the ADC. The checks include an ADC accuracy BIST and GPIO input MUX verification. The ADC accuracy is a set of internal tests which comprises manipulating the input voltage and divider settings to test the accuracy of the ADC. Each of these functions can be run as an "on-demand" (upon start-up or as required during operation), or the ADC accuracy BIST can be run as part of the round-robin functionality.

The accuracy checks only work when using the ADC's internal voltage reference.

ADC Accuracy BIST

The checks performed during the ADC accuracy BIST are described in the following table. The different modes of the ADC's internal dividers are checked during the accuracy test.

Table 32. MAX96793 ADC Accuracy Tests

ADC Input Voltage (V)	adc_scale	adc_refscl	Low Code Limit	High Code Limit	Comments
1.0	0	0	838-REFLIM	838+REFLIM	This ADC code is stored as REFTSTCODE for following tests.
0.5	0	1	REFTSTCODE -REFLIMSCL1	REFTSTCODE -REFLIMSCL1	
0.25	1	0	REFTSTCODE/8 -REFLIMSCL2	REFTSTCODE/8 +REFLIMSCL2	
0.125	0	0	REFTSTCODE/8 -REFLIMSCL3	REFTSTCODE/8 +REFLIMSCL3	

- REFLIM is a 5b register that controls the limits for accuracy of the reference comparisons (default: 17).
- REFLIMISCL1 is a 4b register that controls the limits for the accuracy of the second test (default: 5).

- REFLIMISCL2 is a 4b register that controls the limits for the accuracy of the third test (default: 5).
- REFLIMISCL3 is a 4b register that controls the limits for the accuracy of the fourth test (default: 5).

The following register reads/writes initiate the ADC accuracy checks as an on-demand operation.

Table 33. MAX96793 ADC Accuracy BIST Setup

Step	Action	Register Address	Bitfield Name	Bits	POR	Decode
1	Set the desired REFLIM accuracy (limit between the two BG references)	0x1D31	REFLIM	7:0	0x0F	0xXX: LSBs of accuracy (Default: 17)
2	Set the reference scale /2 accuracy limit	0x1D32	REFLIMSCL1	7:0	0x0F	0xXX: LSBs of accuracy (Default: 5)
3	Set the ADC scale /2 accuracy limit	0x1D33	REFLIMSCL2	7:0	0x07	0xXX: LSBs of accuracy (Default: 5)
4	Set the ADC input /8 accuracy limit	0x1D34	REFLIMSCL3	7:0	0x07	0xXX: LSBs of accuracy (Default: 5)
5	Enable the ADC calibration done interrupt	0x50C	adc_calDone	7	0b0	0b0: Reporting disabled 0b1: Reporting enabled
6	Enable the interrupts for each of the REF limits	0x50F	reflim_ie reflimscl1_ie reflimscl2_ie reflimscl3_ie	6 5 4 3	0b0 0b0 0b0 0b0	0b0: Reporting disabled 0b1: Reporting enabled
7	Clear the interrupt status flags by reading registers	0x510 0x513	ADC_INTR0 ADC_INTR3	7:0 7:0	0x00 0x00	ADC Interrupt reporting, clear on read
8	Execute the ADC accuracy BIST	0x1D28	RUN_ACCURACY	2	0b0	0b0: Do not run 0b1: Run
9	Wait for calibration done and ADC done	0x510	adc_done_if adc_calDone_if	0 7	0b0 0b0	0b0: Flag cleared 0b1: Flag set
10	Read status interrupt registers	0x513	reflim_if reflimscl1_if reflimscl2_if reflimscl3_if	6 5 4 3	0b0 0b0 0b0 0b0	0b0: Flag cleared 0b1: Flag set

ADC GPIO Input Verification Test (Errata)

GPIO input MUX functionality is not tested using the built-in-self-test of the ADC. This functionality can be manually verified via software. Verification is only required for the MFP channels that will be monitored using the ADC. If no MFP3/5/6 voltages are being monitored using the ADC, then this test is not necessary. The steps to complete this test are shown in the following sections and are also described in the MAX96793 Errata Sheet.

ADC Power Up

The ADC should be powered up using the procedure in the following table prior to running the verification test on each GPIO input that is used. This ADC setup is common to each of the input tests and only needs to be done once at the beginning.

Table 34. MAX96793 ADC GPIO Input Verification Power-Up Flow

Step	Action	Register Address	Bitfield Name	Bits	POR	Decode
1	Reset the device	0x10	RESET_ALL	7	0b0	0b0: No action 0b1: Activate chip reset
2	Enable ADC clock	0x501	adc_clock_en	3	0b0	0b0: ADC clock disable 0b1: ADC clock enable
3	Enable ADC ready interrupt	0x50C	adc_ref_ready_ie	1	0b0	0b0: Reporting disabled 0b1: Reporting enabled
4	Enable ADC done interrupt	0x50C	adc_done_ie	0	0b0	0b0: Reporting disabled 0b1: Reporting enabled
5	Power up the ADC charge pump	0x500	adc_chgpump_pu	4	0b0	0b0: ADC charge pump off 0b1: ADC charge pump on
6	Power up the ADC	0x500	adc_pu	1	0b0	0b0: ADC powered off 0b1: ADC powered on
7	Power up the ADC reference buffer	0x500	adc_refbuf_pu	3	0b0	0b0: ADC reference buffer off 0b1: ADC reference buffer on
8	Power up the ADC internal buffer	0x500	buf_pu	2	0b0	0b0: ADC internal buffer off 0b1: ADC internal buffer on
9	Turn on multiplexer input enable	0x502	Inmux_en[0]	0	0b0	0b0: Input MUX is open 0b1: MUX selected by adc_chsel field
10	Select ADC0 for input MUX	0x501	adc_chsel	7:4	0x0	0x0 – ADC0 (MFP3)
11	Initiate temperature conversion	0x1D28	RUN_TMON_CAL	0	0b0	0b0: Do not run 0b1: Run
12	Enable the MUX verification bit	0x1D28	MUXVER_EN	4	0b0	0b0: Disable 0b1: Enable

Input Verification Test

Once the ADC has been set up, the following test can be run for each GPIO input that is being used.

Table 35. MAX96793 ADC GPIO Input Verification Test Flow

Step	Action	Register Address	Bitfield Name	Bits	POR	Decode
1	Enable ADC0/1/2 Input	0x53E	adc_pin_en	2:0	0b000	0bXX1: Enable ADC0 0bX1X: Enable ADC1 0b1XX: Enable ADC2
2	Set channel for GPIO being tested to low and all other channels to high CH0 – ADC0 CH1 – ADC1	0x1D37	MUXV_CTRL	7:0	0x00	0bXXXXXXX0: CH0 driven low 0bXXXXXXX1: CH0 driven high

	CH2 – ADC2					…….. 0b0XXXXXXX: CH7 driven low 0b1XXXXXXX: CH7 driven high
3	Start an ADC conversion	0x500	cpu_adc_start	0	0b0	0b0: Conversion complete 0b1: Start ADC conversion
4	Read the ADC data registers and check that result is within 15LSB of 0x000	0x508 0x509	adc_data_l adc_data_h	7:0 1:0	0x00 0b00	Bits 7:0 of 10b ADC data Bits 9:8 of 10b ADC data
5	Set channel for GPIO being tested to high and all other channels to low CH0 – ADC0 CH1 – ADC1 CH2 – ADC2	0x1D37	MUXV_CTRL	7:0	0x00	0bXXXXXXX0: CH0 driven low 0bXXXXXXX1: CH0 driven high …….. 0b0XXXXXXX: CH7 driven low 0b1XXXXXXX: CH7 driven high
6	Start an ADC conversion	0x500	cpu_adc_start	0	0b0	0b0: Conversion complete 0b1: Start ADC conversion
7	Read the ADC data registers and check that result is within 35LSB of 0x3E6 (0x3C3 – 0x409)	0x508 0x509	adc_data_l adc_data_h	7:0 1:0	0x00 0b00	Bits 7:0 of 10b ADC data Bits 9:8 of 10b ADC data

Reset for Normal Operation

After running the GPIO Input Verification test for all the desired GPIO inputs, disable the input MUX test bit to return the part to normal operation.

Table 36. MAX96793 Returning to Normal Operation

Step	Action	Register Address	Bitfield Name	Bits	POR	Decode
1	Disable input MUX test	0x1D28	MUXVER_EN	4	0b0	0b0: Disable 0b1: Enable

Examples of ADC Operation

ADC Shutdown Example

Prior to setting up the ADC, it is assumed that the ADC Shutdown sequence is used to ensure that the ADC is in a known state.

Table 37. MAX96793 ADC Shutdown

Step	Action	Read/Write	Register Address	Value	Comments
1	Disable round-robin state machine	W	0x534	0x00	
2	Power down ADC, input buffer, reference buffer and charge pump	W	0x500	0x00	
3	Disable ADC clock	W	0x501	0x00	

ADC Setup Example

Prior to setting up the ADC, it is assumed that the ADC Shutdown sequence is used to ensure that the ADC is in a known state.

Table 38. MAX96793 ADC Setup

Step	Action	Read/Write	Register Address	Value	Comments
1	Enable Global ADC Interrupt	W	0x1E	0xFF	
2	Enable ADC clock and select 1.25V internal reference	W	0x501	0x08	
3	Enable ADC ready interrupt, done interrupt and calibration done interrupt	W	0x50C	0x83	
4	Clear ADC interrupts	R	0x510 0x511 0x512 0x513	-	Read to clear
5	Select internal or external reference	W	0x502	0x00	
6	Power up the ADC, charge pump, internal buffer, and reference buffer	W	0x500	0x1E	
7	Wait for ready interrupt to be asserted	R	0x510	-	Waiting to read 0x02
8	Initialize a temperature conversion	W	0x1D28	0x01	
9	Wait for ADC done, and calibration done interrupt to assert	R	0x510	-	Waiting to read 0x81

On-Demand Read Examples

It is assumed that the ADC has been properly configured using the “ADC Setup” example prior to doing any on-demand reads.

Die Temperature

Note: The 1.25V thermally corrected voltage reference must be used to perform a die temperature reading.

Table 39. MAX96793 On-Demand Die Temperature Reading

Step	Action	Read/Write	Register Address	Value	Comments
1	Initiate temperature conversion	W	0x1D28	0x01	

2	Wait for ADC done, and calibration done interrupt to assert	R	0x510	-	Waiting for 0x81
3	Read internal die temperature registers	R	0x1D3B 0x1D3C	-	
4	Read alternate internal die temperature registers (optional)	R	0x1D3C 0x1D3D	-	

Internal Voltage

In this example, the VDD18 rail is monitored using the ADC.

Table 40. MAX96793 On-Demand Internal Voltage Reading

Step	Action	Read/Write	Register Address	Value	Comments
1	Set input channel to VDD18	W	0x501	0x98	Keep adc_clk_en = 1
2	Enable channel multiplexer	W	0x502	0x01	
3	Clear ADC Interrupts	R	0x510 0x511 0x512 0x513	-	Read to clear
4	Start ADC conversion	W	0x500	0x1F	Keep power up bits enabled
5	Wait for ADC done interrupt	R	0x510	-	Waiting for 0x01
6	Read ADC result	R	0x508 0x509	-	
7	Open input multiplexer	W	0x502	0x00	

External Voltage (MFP3)

In this example, MFP3 is being monitored and 2.0V is the expected voltage.

Table 41. MAX96793 On-Demand External Voltage Reading

Step	Action	Read/Write	Register Address	Value	Comments
1	Set input channel to ADC0	W	0x501	0x08	Keep adc_clk_en = 1
2	Enable channel to multiplexer & set internal divider to divide by 2	W	0x502	0x05	
3	Enable monitoring of MFP with ADC	W	0x53E	0x01	
4	Clear ADC Interrupts	R	0x510 0x511 0x512 0x513	-	Read to clear
5	Start ADC conversion	W	0x500	0x1F	Keep power up bits enabled
6	Wait for ADC done interrupt	R	0x510	-	Waiting for 0x01
7	Read ADC result	R	0x508 0x509	-	
8	Open input multiplexer	W	0x502	0x00	

HI/LO Channel Limit Example

Voltage Threshold Calculations

As an example, the thresholds are calculated for a ±10% HI/LO threshold on the 3.3V VDDIO rail. The LO threshold is 2.97V, and the HI threshold is 3.63V.

$$\frac{LO\ Threshold\ \times 1023}{V_{ref} \times Divider} = \frac{2.97V\ \times 1023}{1.25V\ \times 4} = 608\ (decimal) \rightarrow 0x260$$

$$\frac{HI\ Threshold\ \times 1023}{V_{ref} \times Divider} = \frac{3.63V\ \times 1023}{1.25V\ \times 4} = 742\ (decimal) \rightarrow 0x2E6$$

It is important to use the correct values for V_{REF} and Divider as they can be different from this example. V_{REF} is 1.25V if using the thermally corrected internal voltage reference, but this reference can be configured to different voltages (internal 1.8V which uses 900mV in this equation, or an external voltage reference). The divider value is different depending on which internal voltage is monitored or what GPIO divider setting is used.

Temperature Threshold Calculations

An example of calculating HI/LO thresholds for die temperature is shown in this section. In this example, EFUSE registers must first be read to calculate device specific values. After reading these values, convert only the bits used in the equations to decimal for use in the DADC_THOT, VBG_THOT, and THOT equations.

Read 0x1C2D = 0x67

Read 0x1C2E = 0x74

Read 0x1C26 = 0xEC

Read 0x1C27 = 0x4D

Read 0x1C2A = 0x06

Read 0x1C2E = 0x86

0x1C2D [7:0] = 0x67 -> 103

0x1C2E [6:0] = 0x74 -> 116

0x1C26 [7:0] = 0xEC -> 236

0x1C27 [6:0] = 0x4D -> 77

0x1C2A [7:0] = 0x06 -> 6

0x1C2E [3:0] = 0x06 -> 6

With these EFUSE register values, DADC_THOT, VBG_THOT, and THOT can be calculated:

$$DADC\_THOT = 0x1C2D[7:0] + (0x1C2E[6:0] \times 256) = 103 + (116 \times 256) = 29{,}799$$

$$VBG\_THOT = 0x1C26[7:0] + (0x1C27[6:0] \times 256) = 236 + (77 \times 256) = 19{,}948$$

$$THOT = 0x1C2A[7:0] + (0x1C2B[3:0] \times 256) = 6 + (6 \times 256) = 1{,}542$$

Next, ADC_HOT and TRIM_TEMP are calculated:

$$ADC\_HOT = Round\left(\frac{2^{24}}{DADC\_THOT} \times \frac{VBG\_THOT}{20{,}480}\right) = Round\left(\frac{2^{24}}{29{,}799} \times \frac{19{,}948}{20{,}480}\right) = 548$$

$$TRIM_{TEMP}(Kelvin) = \frac{THOT}{4} = \frac{1{,}542}{4} = 385.5$$

Now, the hexadecimal HI/LO thresholds can be calculated. In this example, the values calculated are for -37°C and 102°C:

$$ADC\ TEMP\ REG = Round\left(\frac{ADC\_HOT}{TRIM\_TEMP} \times Desired\ Temperature\ Limit\ in\ Kelvin\right)/2^{adc\_scale[0]}$$

$$ADC\ TEMP\ LOW = \frac{Round\left(\frac{548}{385.5} \times (-37°C + 273.15)\right)}{2^0} = 336 \rightarrow 0x150$$

$$ADC\ TEMP\ HIGH = \frac{Round\left(\frac{548}{385.5} \times (102°C + 273.15)\right)}{2^0} = 533 \rightarrow 0x215$$

Note that in this example adc_scale[0] = 0, but this is dependent on what is set in register 0x501.

These hexadecimal values are what is written to ADC_LIMIT<CH>_0, ADC_LIMIT<CH>_2 and ADC_LIMIT<CH>_2.

Round-Robin Example

In the following example, 5 HI/LO channel limits are set for VDDIO, VDD18, CAP_VDD, die temperature, and MFP3. The limit channels and thresholds are shown as follows:

CH0: +/-10% of 3.3V for VDDIO

CH1: +/-5% of VDD18

CH2: +/-5% of 1.0V for CAP_VDD

CH3: -37°C to 100°C for die temperature

CH4: +/-10% of 2.0V for MFP3

Note: These channels and thresholds can be changed depending on the specific application.

Table 42. MAX96793 ADC Round-Robin Example

Step	Action	Read/Write	Register Address	Value	Comments
1	Program the 10b HI & LO thresholds for CH0 (VDDIO)	W	0x514 0x515 0x516	0x60 0x62 0x2E	LO threshold – 0x260 HI threshold – 0x2E6
2	Program the multiplexer input channel to VDDIO	W	0x517	0x08	
3	Program the 10b HI & LO thresholds for CH1 (VDD18)	W	0x518 0x519	0xBC 0x52	LO threshold – 0x2BC HI threshold – 0x305

			0x51A	0x30	
4	Program the multiplexer input channel to VDD18	W	0x51B	0x09	
5	Program the 10b HI & LO thresholds for CH2 (CAP_VDD)	W	0x51C 0x51D 0x51E	0x71 0x21 0x1C	LO threshold – 0x171 HI threshold – 0x1C2
6	Program the multiplexer channel 2 to CAP_VDD	W	0x51F	0x0A	
7	Read EFUSE registers used to calculate temperature thresholds	R	0x1C2D 0x1C2E 0x1C26 0x1C27 0x1C2A 0x1C2B	0x67 0x74 0xEC 0x4D 0x06 0x86	
8	Program the 10b HI & LO thresholds for CH3 (TMON)	W	0x520 0x521 0x522	0x50 0x51 0x21	LO threshold – 0x150 HI threshold – 0x215
9	Program the multiplexer input channel to TMON	W	0x523	0x0B	
10	Program the 10b HI & LO thresholds for CH4 (ADC0/MFP3)	W	0x524 0x525 0x526	0xE1 0x42 0x38	LO threshold – 0x2E1 HI threshold – 0x384
11	Enable ADC0 (MFP3)	W	0x53E	0x01	
12	Program the multiplexer input channel to ADC0 (MFP3) and set divider to Divide by 2	W	0x527	0x10	
13	Enable the channel HI interrupt	W	0x50D	0x1F	CH0-CH4 enabled
14	Enable the channel LO interrupt	W	0x50E	0x1F	CH0-CH4 enabled
15	Enable the global HI and LO interrupts	W	0x50C	0x8F	Keeps adc_done_ie, adc_ref_ready_ie and adc_calDone enabled
16	*(Optional)* Run ADC Accuracy test	W	0x1D28	0x80	0x80 will run ADC accuracy test in round robin state machine
17	Set number of conversion cycles between ADC conversion	W	0x536 0x537	0x6B 0x2D	5 second sleep time = 0x2D6B
18	Enable round robin state machine	W	0x534	0x01	

Internal Testing Examples

ADC Accuracy BIST

Table 43. MAX96793 ADC Accuracy BIST Example

Step	Action	Read/Write	Register Address	Value	Comments
1	Set the desired REFLIM accuracy (limit between the two BG references)	W	0x1D31	0x11	Default 17 LSBs
2	Set the reference scale /2 accuracy limit	W	0x1D32	0x05	Default 5 LSBs

3	Set the ADC scale /2 accuracy limit	W	0x1D33	0x05	Default 5 LSBs
4	Set the ADC input /8 accuracy limit	W	0x1D34	0x05	Default 5 LSBs
5	Enable the ADC calibration done interrupt	W	0x50C	0x83	Preserves ADC done and ADC ref ready interrupt
6	Enable the interrupts for each of the REF limits	W	0x50F	0x78	Enables all interrupts for ADC accuracy tests
7	Clear the interrupt status flags by reading registers	R	0x510 0x513	-	Clear on read
8	Execute the ADC accuracy tests	W	0x1D28	0x04	
9	Wait for Calibration done and ADC done	R	0x510	-	Waiting for 0x81
10	Read status interrupt registers	R	0x513	-	0x00 for no failures

ADC GPIO Input Verification Test (Errata)

In this example, the GPIO input verification test is run for MFP3. Full scripts can also be found in the MAX96793 Errata Sheet.

Table 44. MAX96793 ADC GPIO Input Verification Test Example

Step	Action	Read/Write	Register Address	Value	Comments
1	Reset the device	W	0x10	0x80	Cleared on write
2	Enable ADC clock	W	0x501	0x08	
3	Enable ADC ready, ADC done interrupts	W	0x50C	0x03	
4	Power up the ADC, ADC charge pump, input buffer and internal reference	W	0x500	0x1E	
5	Turn on multiplexer input enable	W	0x502	0x01	
6	Select ADC0 for input MUX	W	0x501	0x08	Keep ADC clock enable on
7	Initiate temperature conversion	W	0x1D28	0x01	Cleared on write
8	Enable the MUX verification bit	W	0x1D28	0x10	
9	Enable ADC0/1/2 Input	W	0x53E	0x01	
10	Set channel for GPIO being tested to low and all other channels to high CH0 – ADC0 CH1 – ADC1 CH2 – ADC2	W	0x1D37	0x0FE	
11	Start an ADC conversion	W	0x500	0x1F	
12	Read the ADC data registers and check that result is within 15LSB of 0x000	R	0x508 0x509	-	
13	Set channel for GPIO being tested to high and all other channels to low CH0 – ADC0 CH1 – ADC1 CH2 – ADC2	W	0x1D37	0x01	
14	Start an ADC conversion	W	0x500	0x1F	
15	Read the ADC data registers and check that result is within 35LSB of 0x3E6	R	0x508 0x509	-	

16	Disable input MUX test bit for normal operation	W	0x1D28	0x00	

Power on Self-Test (LBIST/MBIST)

Overview

POST runs on the chip during the power-up sequence and is turned off afterwards. During POST, a portion of the logic is tested using the logic built-in self-test (LBIST) and memories are tested using the memory built-in self-test (MBIST).

The runtime for POST is under 10ms and cannot be bypassed. Following POST, a status register contains a bit indicating pass/fail for LBIST and a bit for MBIST pass/fail. Although LBIST/MBIST testing may fail, the chip continues the bring-up operation, enabling continued functionality.

As LBIST and MBIST operate on multiple blocks, LBIST and MBIST pass/fail bits indicate that all LBIST sub-blocks or all MBIST memories passed. Therefore, a failure in one of the LBIST sub-blocks or one of the MBIST memories indicates that the entire LBIST or MBIST operation failed.

Operation

POST cannot be bypassed. There are no configuration bits for this function. During start-up, do not attempt to configure the part until 15ms after power is applied. The status of the POST step is checked by reading the following table's register bit fields.

Note: If LBIST or MBIST fails, do not use the device (shut down the application).

Table 45. MAX96793 POST Status Registers

Register	Bitfield Name	Bits	Default Value	Decode
0x1D20	POST_DONE	7	0	0b0= POST did not run 0b1= POST is run
	POST_MBIST_PASSED	6	0	0b0= POST MBIST has failed 0b1= POST MBIST has passed
	POST_LBIST_PASSED	5	0	0b0= POST LBIST has failed 0b1= POST LBIST has passed

Bandwidth Efficiency Optimization

Overview

Before implementing a new design, it is critical to do bandwidth (BW) calculations to verify that the right devices and settings are used. If this is not done, it is possible that data is lost or corrupted. Although the MAX96793 family can transmit at 3Gbps, 6Gbps, or 12Gbps depending on part number and configuration, the maximum allowable video payload is smaller due to the overhead of the GMSL link. The video payload should not exceed the values shown in this table.

Table 46. GMSL3 Maximum Video Payloads

GMSL3 Mode	Maximum Video Payload

3Gbps Mode (NRZ)	2.6Gbps (2600Mbps)
6Gbps Mode (NRZ)	5.2Gbps (5200Mbps)
12Gbps Mode (PAM4*)	9.7Gbps (9700Mbps)

*While operating in 12Gbps PAM4 mode, FEC is required (enabled by default).

Calculating Bandwidth

The basic video payload can be calculated using the following equations:

$$PCLK = H_{total} \times V_{total} \times frame\ rate \quad OR \quad PCLK = \frac{LANE\_CNT \times LANE\_RATE}{bpp}$$

$$Video\ Payload = PCLK \times bpp$$

Note: (1) Htotal and Vtotal must include the horizontal and vertical blanking. (2) Use a BPP of 9 when calculating the BW for 8 BPP datatypes. This is the minimum BPP required for the video pipe.

If the lane speeds on the MIPI receiver are fast enough to handle the video payload, the part should not overflow. After calculating the video payload, overhead is added to calculate the total GMSL bandwidth.

$$Video\ BW = [(video\ payload) + (video\ packet\ header) + (video\ pixel\ CRC)] \times (9b10b\ encoding) \times (sync\ words) \times (forward\ error\ correction)^*$$

$$Video\ BW = PCLK \times \left[(bpp) + \left(\frac{1}{2}\right) + \left(\frac{1}{2}\right)\right] \times \left(\frac{10}{9}\right) \times \left(\frac{2048}{2047}\right) \times \left(\frac{128}{120}\right)^*$$

The majority of GMSL3 serial link bandwidth comprises video transmission. The total link bandwidth consumed by video is derived from the incoming video stream and calculated by multiplying the pixel clock (PCLK) expressed in MHz, BPP, and GMSL3 link overhead factors. Note that control channel features (e.g., GPIO, SPI) affect link bandwidth consumption and must be considered if enabled.

*Only applies when GMSL FEC is enabled. Bandwidth utilization for all control channel and side channel communication increases by 6.7% when FEC is enabled. FEC is optional for GMSL2 operation.

Note: FEC is mandatory when operating in GMSL3 12Gbps mode.

Optimizing Bandwidth

Data can be manipulated over the GMSL link to utilize the bandwidth more efficiently. The easiest way to optimize the bandwidth consumption is by using tunneling mode. In tunneling mode, all BPPs are forced to 24 regardless of datatype and this allows for the PCLK and overall bandwidth to be reduced. Another common way of optimizing the bandwidth is by double or tripling the BPP. When transmitting an 8BPP datatype at 16BPP or 24BPP, less bandwidth is consumed for the same reason as when using tunneling mode. With regards to bandwidth, zero padding has the opposite effect compared to what doubling or tripling has. Zero padding always consumes more bandwidth so this method of data manipulation should only be used when necessary.

Bandwidth Optimization Example

The GMSL GUI has a very useful tool called Bandwidth Calculator that is used for calculating the total GMSL bandwidth consumed by the link. The following example compares the tool's calculations between identical pixel mode links with and without utilizing double mode.

Case 1: Transmitting 4-lanes of RAW8 data at 1100Mbps/lane without doubling the BPP

- RAW8 datatype, without doubling, gives a total GMSL BW of roughly 5,829Mbps.
- Note that the pipe BPP is 9 instead of 8. This is the minimum BPP supported on the pipe.

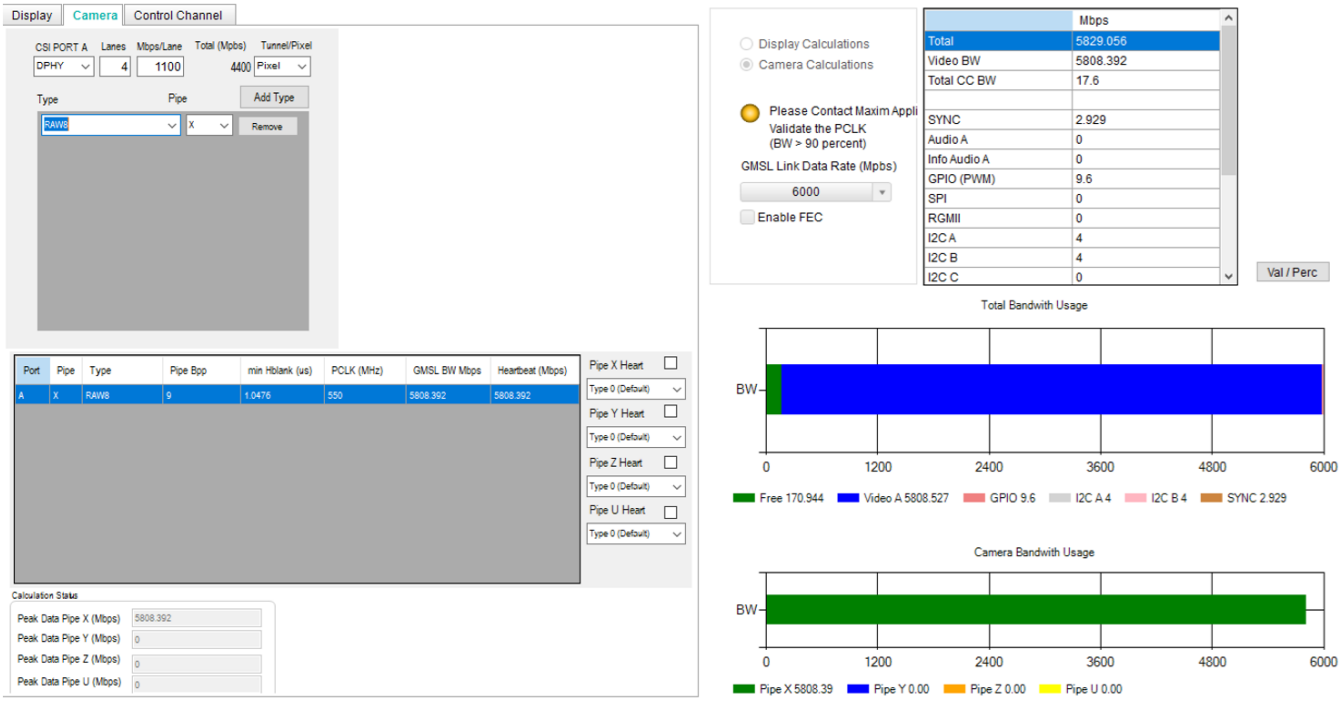

Figure 30. Bandwidth Calculations without Doubling

Case 2: Transmitting 4-lanes of RAW8 data at 1100Mbps/lane with doubling the BPP to 16

- RAW8 datatype, with doubling, gives a total GMSL BW of roughly 5,064Mbps.
  - Doubling the BPP saved over 750Mbps worth of BW
- Note that the pipe BPP is 16, confirming that RAW8 was doubled.

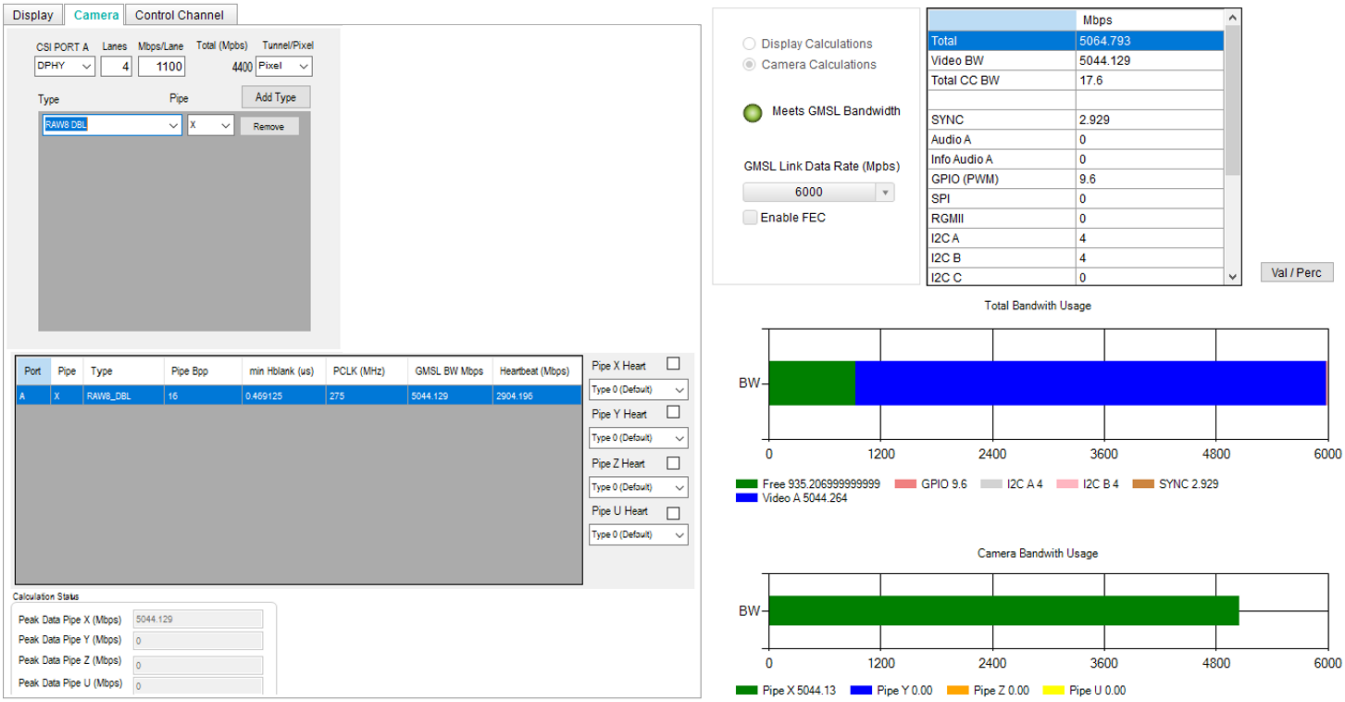

Figure 31. Bandwidth Calculations with Doubling

MIPI Packet Counter

Overview

The MIPI packet count registers are used to determine whether MIPI data is flowing. This is usually done as a debugging step if the video is not being received.

MIPI Packet Counter Registers

Each GMSL link has a built-in, 8-bit received packet counter. Within the MIPI PHY/Controller there are packet counters implemented that can be used to verify data is being transferred within the serializer and deserializer. Sequential reads returning different values indicate that data is being transmitted by the associated MIPI controller, PHY, or function.

The MAX96793 has packet counters for MIPI PHY, Controller, DPHY Clock, and tunneling mode packets.

Table 47. MAX96793 MIPI Counter Registers

Register	Bits	Default Value	Description	Decode
0x38D	7:0	0x00	Packet count of MIPI PHY1	**Phy1_pkt_cnt registers:** 0bXXXx: Toggling bits indicate MIPI data is active on the PHY
0x38E	7:0	0x00	Packet count of CSI-2 Controller	**csi1_pkt_cnt registers**: 0bXXXX: Toggling bits indicate MIPI data is active on the controller. (Packets Processed)

0x38F	7:0	0x00	Packet Count of Tunnel Packet Processed	**tun_pkt_cnt register:** 0bXXXx: Toggling bits indicate MIPI data is active on the PHY
0x390	7:0	0x00	MIPI PHY Clock Counter	**phy_clk_cnt register:** 0bXXXx: Toggling bits indicate MIPI clock is detected

Packet Counter Example

The following example demonstrates the use case of the MIPI Packet counters.

Setup: MAX96793 and MAX96792A with SV4E Introspect Generator and Analyzer.

- Connect the SV4E Introspect generator to the MAX96793.
- Connect Link A of the MAX96792A to the MAX96793.
- Generate a CSI_RGB888 Pattern on the Introspect Generator.
- Check PCLKDET on the MAX96793 and video lock on the MAX96792A.
- Now, check registers 0x38D, 0x38E, 0x390 on the MAX96793 to check if bits are toggling.
- Now, check registers 0x342, 0x344, and 0x345 on the MAX96792A to check if bits are toggling.
- If the bits are not toggling, then check the parts configuration.

Error Flags

Overview

The device contains many internal error detection mechanisms to alert the system or user of any issues. Error flags are register bits that can be checked to see if an error has occurred.

The error bar (ERRB) pin is an MFP pin that logically NORs many of the errors. Whether an error is included in the ERRB output depends on if its output-enable (OEN) bitfield is set high. Most OENs are high by default.

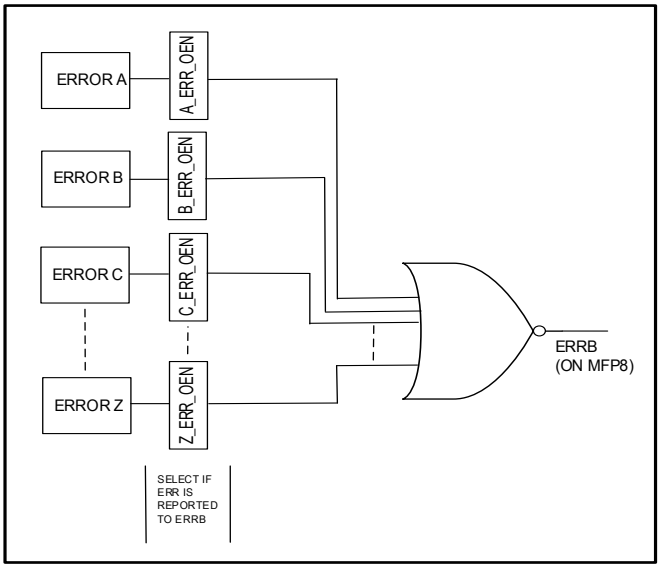

Figure 32. MAX96793 ERRB Reporting Flow

Device Error Flags

The MAX96793 ERRB pin (active low) is available on MFP3 and needs to be enabled by register writes. In addition, register bitfield ERROR (register 0x13 bit 2) goes HIGH when device has an error. *Table 48* lists all the error flags in the MAX96793.

Table 48. MAX96793 Error Flags Table

Error Flag	Error Description
VREG_OV_FLAG	CAP_VDD overvoltage indication
EOM_ERR_FLAG_A	Eye Opening is below the configured threshold
VDD_OV_FLAG	VDD overvoltage indication
VDD18_OV_FLAG	VDD18 overvoltage indication
MAX_RT_FLAG	Combined ARQ maximum retransmission limit error flag
RT_CNT_FL AG	Combined ARQ retransmission event flag
PKT_CNT_F LAG	Packet Count Flag
VDDCMP_INT_FLAG	Combined undervoltage comparator output. Asserted when GMP_SATATUS is asserted.
PORZ_INT_FLAG	PORZ interrupt flag. PORZ is Monitoring of undervoltage levels of VDD18 and VDDIO.
VDDBAD_INT_FLAG	Combined VDD bad indicator. Asserted when either with VDDBAD_STATUS or CAP_VDD <0.82V.
EFUSE_CRC_ERR	EFUSE CRC error indicator. An issue with the device EFUSE has occurred, the device should no longer be used.
RTTN_CRC_INT	Retention memory restore CRC error interrupt
ADC_INT_FLAG	ADC Interrupt
MIPI_ERR_FLAG	MIPI RX Error Flag
REFGEN_UNLOCKED	Reference DPLL generating RCLKOUT is not locked.
REM_ERR_FLAG	Received remote side error status
LFLT_INT	Line-Fault Interrupt Asserted when either one of line-fault monitors indicates a fault status
IDLE_ERR_FLAG	Idle-Word Error Flag
DEC_ERR_FLAG	Errors detected in GMSL packet
I2C_UART_ MSGCNTR_ ERR_INT	I2C/UART message counter error flag. Asserted when two message counter bytes sent do not match expected count value.
I2C_UART_ CRC_ERR_I NT	I2C/UART CRC error flag. Asserted when CRC byte sent does not match calculated value.
MEM_ECC_ ERR2_INT	Error flag for 2-bit uncorrectable memory ECC errors seen in any memories
MEM_ECC_ ERR1_INT	Error flag for 1-bit uncorrectable memory ECC errors seen in any memories
REG_CRC_ ERR_FLAG	Error occurred on the register CRC calculation.
SPI_RX_OVRFLW	SPI Rx Buffer Overflow Flag
SPI_TX_OVRFLW	SPI Tx buffer overflow flag
adc_overRange_ie	ADC Digital Correction Overrange enabled
adc_tmon_cal_ood_ie	Enable temperature sensor out-of-date interrupt
adc_lo_limit_ie	Enable ADC low limit monitor interrupt
adc_hi_limit_ie	Enable ADC high limit monitor interrupt
adc_ref_read y_ie	Enable ADC ready interrupt
adc_done_ie	Enable ADC conversion completed Interrupt
adc_calDone_ie	Signal that ADC accuracy/temperature sensor calibration is completed
DRIFT_ERR	VID_TX_PCLK drift error detected

FIFO_WARN	Transmitted video (VID_TX_FIFO) is more than half full
OVERFLOW	VID_TX FIFO overflow occurred
tun_fifo_overflow	Tunnel FIFO overflow occurred
CMP_STATUS	VDD18, VDDIO, and CAP_VDD supply voltage comparator status bits. Latched when the supply voltages are not in range.
phy*1_lp_err	Unrecognized commands or Invalid line sequences are detected. *May be replaced with phy1/phy2.
phy*1_hs_err	HS Synch pattern with error detected on data lanes. *May be replaced with phy1/phy2.
ctrl1_csi_err_ l	ECC or CRC errors detected in CSI-2 controller, low byte
ctrl1_csi_err_ h	Packets terminated early or/and Frame count error detected in CSI-2 controller

General-Purpose Input and Output

Overview

This section explains the GPIO function of MFP pins. The GPIO blocks communicate and regenerate state changes of GPIO pins from one side of the serial link to the other. An input GPIO value on one side of the GMSL link may be sent to any of the GPIO outputs on the opposite side of the link.

Depending on the pin, they can be used as either full or partial GPIO pins or for other functionality (e.g., I^2C, LOCK, ERRB, etc.). Refer to the data sheets for additional details on GPIO capabilities and default states after power-up.

The MAX96793 has 11 MFPs.

Operation

GPIO pin mapping is coordinated across the serial link through GPIO "pin ID" assignments. Each GPIO input is assigned a pin ID that is included in the packet sent across the serial link and corresponds with a GPIO output. By default, the GPIO mapping is GPIO0 to GPIO0, GPIO1 to GPIO1, GPIO2 to GPIO2, etc. The GPIO mappings can be changed through registers.

The MAX96793 uses 5-bit pin IDs that can support mapping up to 32 GPIO pins. Note that the usable number of GPIOs is limited by the specific GPIO pinout. Each GPIO is controlled by three registers: GPIO_A, GPIO_B, and GPIO_C. In the register documentation, the GPIO mapping is sequential (i.e., the first three GPIO registers correspond to GPIO0, then next three to GPIO1, etc.). Additional details related to these registers can be found in the "GPIO Registers" section of the respective data sheet.

When programming GPIOs, it is important to program the GPIO Rx before the GPIO Tx to avoid asynchronous initial states. For example, if Tx is low but Rx is high, the first transition of Tx from low to high is ignored by Rx as Rx is already high. All subsequent transitions are correctly observed.

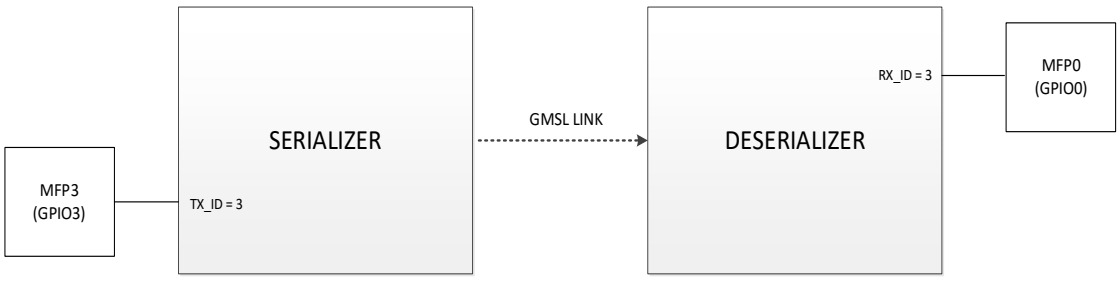

Figure 33. GPIO Forwarding Example with a Transition from MFP3 to MFP0

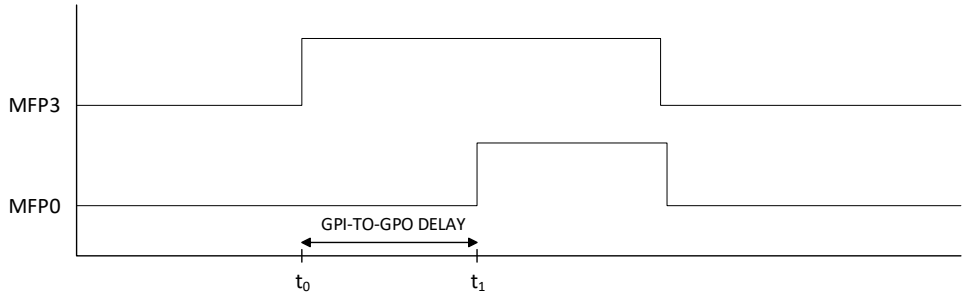

Figure 34. GPIO Forwarding Timing Diagram

MFP Capabilities: GPIO, GPI, GPO and ODO

The MAX96793 have eleven MFPs; five of these MFPs are GPIO, two are GPO (general-purpose output) and four are ODO_GPI (open-drain output, general-purpose input) which are reserved for the primary control channel. *Table 49* shows the GPIO capabilities of each MFP.

Table 49. MAX96793 Multifunction (MFP) Capabilities

Pin Name	Capability
MFP0	GPIO0
MFP1	GPO1
MFP2	GPO2
MFP3	GPIO3
MFP4	GPIO4
MFP5	ODO5_GPI5
MFP6	ODO6_GPI6
MFP7	GPIO7
MFP8	GPIO8
MFP9	ODO9_GPI9
MFP10	ODO10_GPI10

GPIO Pull-Up and Pull-Down Resistor Setup

Each GPIO can be programmed to have either a pull-up, pull-down, or no resistor. The pull-up or pull-down resistance can be set to either 40kΩ or 1MΩ.

The resistor is configured with the PULL_UPDN_SEL[1:0] register:

- 00: No resistor
- 01: Pull-up resistor
- 10: Pull-down resistor
- 11: Reserved

The resistance value of the resistor is set using the RES_CFG register:

- 0: 40 kΩ
- 1: 1 MΩ

GPIO Output Driver Setup

The GPIO output driver can be enabled or disabled. When enabled, the output driver can be configured to be either open-drain or push-pull. The output driver is enabled by writing GPIO_OUT_DIS = 0 and disabled by writing GPIO_OUT_DIS = 1. The output driver is configured for the open-drain mode (i.e., NMOS output driver enabled) by writing OUT_TYPE = 0 and for push-pull mode (i.e., both NMOS and PMOS output driver enabled) by writing OUT_TYPE = 1.

Configuring GPIO Forwarding

GPIO forwarding is the transmission and regeneration of state changes of GPIO pins on the local side of the serial link to the corresponding GPIO pins on the remote side. To forward the pin value, the local and remote side GPIOs must be properly configured. Each GPIO has configurable registers GPIO_TX_ID and GPIO_RX_ID used for mapping GPIO pins across the serial link. Note that this configuration applies to both the serializer-to-deserializer and deserializer-to-serializer communications.

Configuring Input GPIO:

- Set GPIO_TX_ID with a value from 0 to 31 to assign the GPIO pin ID.
- Write GPIO_TX_EN = 1 to enable the GPIO transmit block.

Configuring Output GPIO:

- Set GPIO_RX_ID with a value from 0 to 31 to assign the GPIO pin ID. This must be the same value used for GPIO_TX_ID to map the input and output GPIO pins.
- Write GPIO_RX_EN = 1 to enable the GPIO receive block for the GPIO pin.

By default, the GPIO_TX_ID and GPIO_RX_ID are the same value as the GPIO number. For example, the default GPIO_TX_ID and GPIO_RX_ID values for GPIO1 is 1. Accordingly, GPIO1 is mapped to GPIO1 on the opposite side of the serial link by default.

GPIO Broadcasting

The same concept of GPIO forwarding can be configured so that a transition on a single GPIO input is mapped to multiple GPIO outputs (broadcasting). To do this, set the GPIO_TX_ID of the input GPIO to the same GPIO_RX_ID of multiple output GPIO pins. *Figure 35* is an example of this configuration.

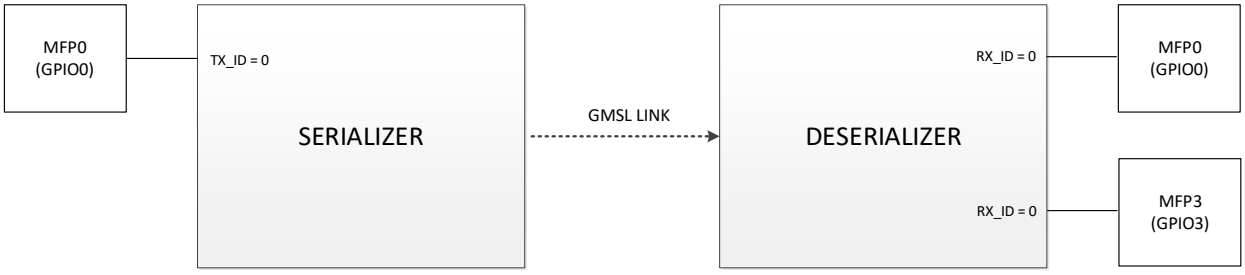

Figure 35. GPIO Broadcasting

GPIO Delay Compensation

In the nondelay-compensated mode (default), the GPI transition is sent as fast as possible across the link, based on priority and available link bandwidth. As a result, there is a variable delay between an input transition and the subsequent transition on the other side of the GMSL3 link. Delay compensation can be used to ensure that the timing delay between input transition and output transition is constant. *Table 50* shows the typical values and the next table's registers show how to set delay compensation.

Table 50. GPIO (with/without) Delay Compensation Values

Direction	Delay Compensation	Delay
GPIO Forwarding from Serializer to Deserializer	0	1us
	1	3.5us (default)
GPIO Forwarding from Deserializer to Serializer	0	6us
	1	15us (default)

Table 51. MAX96793 GPIO Delay Compensation Delay Registers

Register	Bitfield Name	Bits	Default Value	Decode
0x30	GPIO_FWD_CDLY	5:0	0b000001	Bit [5:0]: 0bxxxxxx Compensation delay multiplier for the forward direction. This must be the same value as GPIO_FWD_CDLY of the chip on the other side of the link. Total delay is the (value + 1) multiplied by 1.7µs. Default delay is 3.4µs.
0x31		5:0	0b001000	Bit [5:0]: 0bxxxxxx Compensation delay multiplier for the reverse direction. This must be the same value as GPIO_REV_CDLY of the chip on the other side of the link. Total delay is the (value + 1) multiplied by 1.7µs. Default delay is 3.4µs.

Toggling GPIO Manually with Registers

GPIO pins can be manually controlled through I²C or UART register writes. Write to the local device to toggle local GPIO pins; write to the remote device using the control channel to toggle remote GPIO pins.

- Set GPIO_OUT_DIS = 0 to enable the output driver and configure OUT_TYPE to the desired output mode (open-drain or push-pull).
- Set GPIO_RX_EN = 0 to disable the GPIO receive block for the GPIO pin. This sets the GPIO to receive its value from the bitfield GPIO_OUT instead of from the value being transmitted across the GMSL3 link.
- Set GPIO_OUT to the desired value.

Table 52. MAX96793 GPIO Registers

Register	Bits	Default Value	Decode
0x2BE	7:0	0x83	**GPIO0 A:** Bit 7: RES_CFG 0=40kΩ 1=1MΩ

			Bit 6: RSVD Bit 5: TX_COMP_EN 0=Jitter compensation disabled 1=Jitter compensation enabled Bit 4: GPIO_OUT 0=Drive output to LOW (0) 1=Drive output to HIGH (1) Bit 3: GPIO_IN, 0=GPIO input level LOW (0) 0=GPIO input level HIGH (1) Bit 2: GPIO_RX_EN 0=Disable receiving from the link. 1=Enable receiving from the link Bit 1: GPIO_TX_EN 0=Disable transmitting to the link. 1=Enable transmitting to the link Bit 0: GPIO_OUT_DIS 0=Output driver enabled 1=Output driver disabled
0x2BF	7:0	0xA0	**GPIO0 B:** Bit [7:6]: Resistor Configuration 00=None 01=Pull-up 10=Pull-down Bit 5: OUT_TYPE 0 = Open-drain 1 = Push-pull Bit [4:0]: GPIO_TX_ID Address=0-31
0x2C0	7:0	0x40	**GPIO0 C:** Bit 6: GPIO_RECVED 0=Received GPIO Value 0 1=Received GPIO Value 1 Bit [4:0]: GPIO_RX_ID Address=0 to 31
0x2C1 to 0x2DE	…	…	Repeat Registers A, B, C for GPIO1 to GPIO10.

GPIO Programming Example

In this example, GPIO0 on a MAX96793 serializer is forwarded across the link to GPIO0 on a GMSL3 deserializer. This example can be adjusted to use different GPIO pins or forward a GPIO on the local side to the remote side, depending on the desired application. An important note is to set up the GPIO Rx side before setting up the GPIO Tx side to prevent asynchronous states.

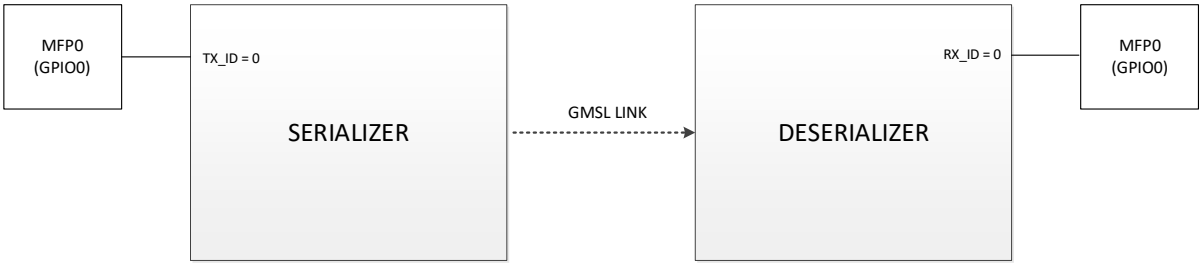

Figure 36. GPIO Forwarding Programming Example

#DES I2C Address=0x98

#SER I2C Address=0x80

#Setup SER MFP0 Pin

#1MΩ Pull-Up, Open Drain

0x80,0x02BE,0x83

0x80,0x02BF,0x40

0x80,0x02C0,0x40

#Setup DES MFP0 Pin

#1MΩ Pull-Up, Open Drain

0x98,0x02B0,0x84

0x98,0x02B1,0x40

0x98,0x02B2,0x40

MFP Slew Rate

The MAX96793 MFPs have configurable rise and fall times (slew rate). This parameter may be referred to as the I/O "speed (control)", "slew (rate)", or "edge rate" in register control bit names. Note that the MFP slew rate cannot be adjusted independently on a per-pin basis. MFPs are divided into separate speed groups; the slew rate adjustment register contains a bitfield for each group that configures the rise and fall time to all pins in the group. Refer to the data sheet for the relevant register map and MFP speed grouping details.

The MFP edge transitions must be fast enough to meet the application's requirement; however, the high-speed I/O of the GMSL link and video protocols (e.g., MIPI) are sensitive to coupling and crosstalk from MFP transitions. Take care at a system level to prevent high edge rates and high frequencies on the MFP inputs close to these I/O. In general, the MFP pins should be configured to the slowest slew rate that allows proper function to mitigate I/O interference.

Note: Coupling refers to both inductive and capacitive coupling. Higher V_{DDIO} supply values increase the MFP edge rate and energy. This can introduce additional noise into the high-speed I/O.

High MFP slew rates, especially combined with high toggle frequencies, near the GMSL or high-speed video pins may adversely affect performance of the data path, including CRC errors, 9b10b code or disparity errors, reduction of link margin, and/or loss of link lock.

Configuration

MFPs are divided into speed groups by digital function. The slew rate adjustment register configures the rise and fall times for each MFP in the group simultaneously. The MFP slew rate can be adjusted at any time, and the changes are applied immediately.

The MFP slew rate configuration applies to all pins in the speed group regardless of the enabled function of the pin. For example, the speed setting is applied to a GPIO and a dedicated pin function if both are in the same MFP speed group.

Refer to the corresponding device's data sheet *Control- and Side-Channel Typical Rise and Fall Times* section for V_{DDIO} timing details. The following tables present Slew Rate Registers and the typical rise and fall times for the MAX96793.

Table 53. MAX96793 MFP Slew Rate Registers

Register	Bitfield Name	Bits	Default Value	Description
0x56F	PIO00_SLEW	1:0	0x10	MFP0 Slew Rate 0b00=Fastest Slew Rate 0b11=Slowest Slew Rate
	PIO01_SLEW	3:2	0x11	MFP1 Slew Rate 0b00=Fastest Slew Rate 0b11=Slowest Slew Rate
	PIO02_SLEW	5:4	0x11	MFP2 Slew Rate 0b00=Fastest Slew Rate 0b11=Slowest Slew Rate
0x570	PIO05_SLEW	3:2	0x11	MFP5 Slew Rate 0b00=Fastest Slew Rate 0b11=Slowest Slew Rate
	PIO06_SLEW	5:4	0x11	MFP6 Slew Rate 0b00=Fastest Slew Rate 0b11=Slowest Slew Rate
0x571	PIO09_SLEW	3:2	0x11	MFP9 Slew Rate 0b00=Fastest Slew Rate 0b11=Slowest Slew Rate
	PIO10_SLEW	5:4	0x11	MFP10 Slew Rate 0b00=Fastest Slew Rate 0b11=Slowest Slew Rate
	PIO11_SLEW	7:6	0x11	MFP11 Slew Rate 0b00=Fastest Slew Rate 0b11=Slowest Slew Rate

Table 54. MAX96793 Typical MFP Rise/Fall Times

Register Value	Rise Time		Fall Time	
	VDDIO=1.8V	VDDIO=3.3V	VDDIO=1.8V	VDDIO=3.3V
0x00	1ns	0.6ns	0.8ns	0.5ns
0x01	2.1ns	1.1ns	2ns	1.1ns
0x10	4ns	2.3ns	4.3ns	2.3ns
0x11	9ns	5ns	10ns	5ns
I2C	N/A	N/A	40ns	30ns

VPRBS Generator and Checker

Overview

One way to verify the link is functional and check for issues is to run a pseudorandom binary sequence (PRBS) test. During this testing, the serializer generates the video PRBS signal and the deserializer checks it.

In conjunction with the VPRBS generator/checker, the serializer error generator feature can also be used to validate errors can be seen on the deserializer. The MIPI input to the serializer must be disabled before running this test.

Table 55. Serializer Video PRBS Generator and Checker Registers

Register	Bits	Default Value	Description	Decode
0x110	3	0b1	Select BPP source	0b0: Use BPP from register (note 1) 0b1: Use BPP from MIPI RX
0x24F	3:1	0b000	Pattern generator clock source for video PRBS, checkerboard and gradient patterns.	0b0xx: Use external PCLK 0b100: Use 25MHz internal CLK 0b101: Use 75MHz internal CLK 0b110: Use 150MHz internal CLK 0b111: Use 375MHz internal CLK
0x26B	7	0b0	Enable Video PRBS generator	0b0: Video PRBS generator disabled 0b1: Video PRBS generator enabled
0x112	7	0b0	PCLKDET	0b0: Video transmit PCLK not detected 0b1: Video transmit PCLK detected

Note 1: When VPRBS is enabled, the default BPP = 24 when register 0x111 is used.

Programming Example

The following script configures a MAX96793 and a MAX96792A to conduct the standard VPRBS test. In this test, the serializer generates a PRBS pattern and the deserializer checks it. For this test, the serializer must have PCLK and no video must be running into the serializer.

#DES I2C Address=0x98

#SER I2C Address=0x80

Connect Serializer GMSL link to Deserializer GMSL link A

#Disable auto bpp on serializer

0x80,0x0110,0x60

#Enable serializer internal PCLK generation, PCLK = 150MHz

0x80,0x024F,0x0D

#Enable serializer pattern generator

0x80,0x026B,0x01

#Delay for 3ms

#Enable PRBS checker for deserializer

0x98,0x01FC,0x90

#Delay for 3ms

#Enable serializer VPRBS generator

0x80,0x026B,0x81

#Verify serializer has PCLKDET = 1

0x80,0x0112,0x8A

#Verify deserializer has VIDEO_LOCK = 1 for Pipe Y

0x98,0x01FC,0x91

#Verify deserializer does not have PRBS errors VPBRB_ERR = 0x00

0x98,0x01FB,0x00

#Optional Step: Enable Serializer errors to verify setup and PRBS error detection

#Note: Error generation also creates decode and idle errors, so these must be cleared as well.

#Enable serializer error generator

0x80,0x0029,0x18

#Delay for a short time to accumulate errors.

#Disable serializer error generator

0x80,0x0029,0x08

#Verify deserializer has PRBS errors VPBRB_ERR > 0x00, 0xFF is used as the read value in this example, but errors may vary. VPRBS Errors should clear after reading this register.

0x98,0x01FB,0xFF

Video Pattern Generator

Overview

The video pattern generator (VPG) creates either a checkerboard or gradient pattern with programable parameters. These patterns can be used to replace the incoming video or in conjunction with the VTG to create an RGB888 video pattern when no video is present on the serializer input.

The serializer and deserializer have an internal VPG that accommodates a wide range of resolutions and frame rates. The VPG does not require an external PCLK source from the CSI-2 input and uses the external 25MHz crystal clock (i.e., REFCLK input) to derive four different pixel clock (PCLK) options. Link lock is not required to use the VPG when being outputted on deserializer, link lock is required if using serializer as VPG source.

The VPG has its own register block settings for timing configurations. See the following tables for register settings required.

There are two clock configuration registers that set the PCLK value for the video pipes. The video pattern PCLK frequency can optionally be set to 25MHz/75MHz/150MHz/375MHz. This internal PCLK is not related to the MIPI CSI-2 port clock rate, which must be set to accommodate the VPG data stream.

The GMSL SerDes GUI can be used to set up the VPG and to generate VPG register write example codes.

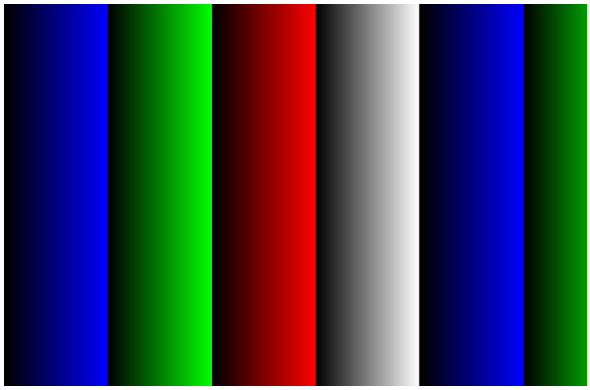

Figure 37. VPG - Gradient Pattern

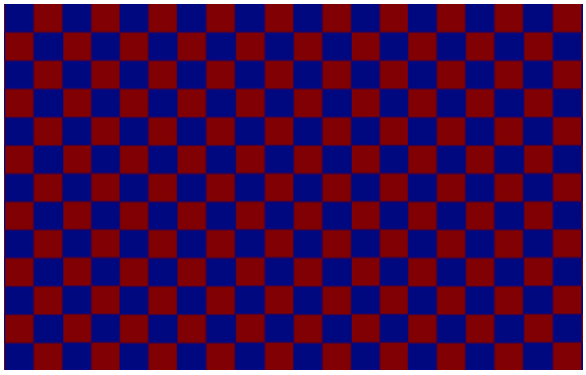

Figure 38. VPG - Checkerboard Pattern

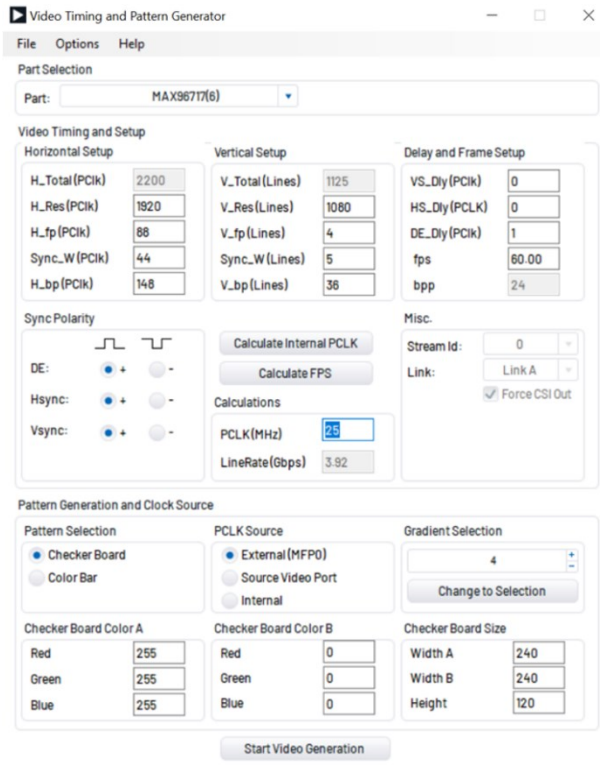

Figure 39. GMSL VTG and VPG Checkerboard Example

Table 56. MAX96793 Video Pattern Registers

Parameter	Register	Bitfield	Decode and Description
Select the pattern type	VTX29	PATGEN_MODE[1:0]	Select the VPG pattern 00: Pattern generator disabled; use video from the serializer input (default) 01: Generate checkerboard pattern 10: Generate gradient pattern 11: Reserved
In gradient mode: Select the gradient direction	VTX29	GRAD_MODE[0]	0: Gradient mode increasing. Each gradient color starts from a value of 0x00 and increases to 0xFF. 1: Gradient mode decreasing. Each gradient color starts from a value of 0xFF and decreases to 0x00.
In gradient mode: Select the gradient pattern length	VTX30	GRAD_INC[7:0]	Selects the value each pixel increments, program to the desired increment amount multiplied by 4. The default value of 4 results in each pixel incrementing by 1, resulting in a pattern length of 256 pixels per color.
In checkerboard mode: Set the value of color A	VTX31 VTX32 VTX33	CHKR_A_L[7:0] CHKR_A_M[7:0] CHKR_A_H[7:0]	Sets the red component of color A Sets the green component of color A Sets the blue component of color A

In checkerboard mode: Set the value of color B	VTX34 VTX35 VTX36	CHKR_B_L[7:0] CHKR_B_M[7:0] CHKR_B_H[7:0]	Sets the red component of color B Sets the green component of color B Sets the blue component of color B
In checkerboard mode: Set the length of color A	VTX37	CHKR_RPT_A[7:0]	Sets the number of pixels of color A. The first line outputs color A first.
In checkerboard mode: Set the length of color B	VTX38	CHKR_RPT_B[7:0]	Sets the number of pixels of color B. The first line outputs color B after CHKR_RPT_A pixels. Set equal to CHKR_RPT_A for a square checkerboard pattern.
In checkerboard mode: Set the height of the checkerboard	VTX39	CHKR_ALT[7:0]	After CHKR_ALT lines, the pattern switches to output color B before color A. Set equal to CHKR_RPT_A and CHKR_RPT_B for a square checkerboard pattern.

Table 57. MAX96793 VPG PCLK Setting Registers

Register	Bits	Default Value	Description
0x24F	3:1	0x000	0XX=Use external clock* (MFP0 can be used) 100=25MHz internal clock 101=75MHz internal clock 110=150MHz internal clock 111=375MHz internal clock

Note: When using MFP0 as external clock source, valid PCLK frequency must be used.

Programming Example

Script that sets up pattern generator in MAX96793 out on Pipe Z

Settings: 8MP @8fps, RGB888

#SER I2C Address=0x80

0x80,0x024E,0x03

0x80,0x0250,0x00

0x80,0x0251,0x00

0x80,0x0252,0x00

0x80,0x0253,0x00

0x80,0x0254,0x50

0x80,0x0255,0x78

0x80,0x0256,0x8A

0x80,0x0257,0x4E

0x80,0x0258,0x40

0x80,0x0259,0x00

0x80,0x025A,0x00

0x80,0x025B,0x00

0x80,0x025C,0x00

0x80,0x025D,0x2C

0x80,0x025E,0x0F

0x80,0x025F,0xEC

0x80,0x0260,0x08

0x80,0x0261,0x9D

0x80,0x0262,0x02

0x80,0x0263,0x94

0x80,0x0264,0x98

0x80,0x0265,0x0F

0x80,0x0266,0x00

0x80,0x0267,0x01

0x80,0x0268,0x18

0x80,0x0269,0x08

0x80,0x026A,0x70

0x80,0x026B,0x01

0x80,0x026D,0xFF

0x80,0x026E,0xFF

0x80,0x026F,0xFF

0x80,0x0270,0x00

0x80,0x0271,0x00

0x80,0x0272,0x00

0x80,0x0273,0xF0

0x80,0x0274,0xF0

0x80,0x0275,0x78

0x80,0x0007,0xF6

0x80,0x024F,0x0B

0x80,0x0110,0x62

0x80,0x0383,0x00

0x80,0x024E,0xE3

Video Timing Generator

Overview

The Video Timing Generator (VTG) generates or adjusts video sync signals. It may be used to modify the timing details of incoming video streams, or it may be used with the Video Pattern Generator (VPG) to generate test images. The VTG is implemented within the video pipe of the serializer. The MAX96793 all have the same VTG functionality.

The VTG provides user-configurable options for the video sync signals: Vertical Sync (VS), Horizontal Sync (HS), and Data Enable (DE). If enabled, the VTG regenerates the video sync signals in accordance with user defined timing parameters. These parameters offer flexibility to customize the sync polarity, pulse width, and timing of the regenerated video sync signals.

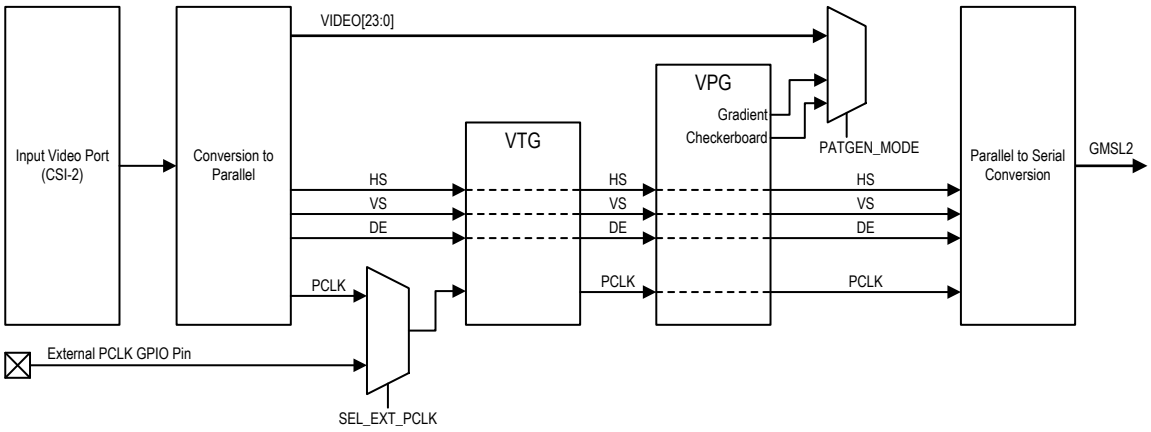

Figure 40. MAX96793 Video Timing Generator Block Diagram

VTG Operation

The core function of the VTG block is to generate VS, HS, and DE signals based on a trigger. The VTG can be configured to generate these signals internally or if the VTG is not enabled then the VS/HS/DE signals at the input of the VTG drive the output signals. This selection is made individually for each sync signal by the GEN_VS, GEN_HS, and GEN_DE register bits.

The VTG can be triggered by either a VS input transition (i.e., the external trigger) or an internally generated VS trigger (i.e., the tracking VS signal). In the case of the external trigger, the VS transition trigger is selected to be either the rising edge or the falling edge of the input VS signal with the VS_TRIG bit. Note that the selected edge for the input transition is referred to as the active edge. The polarity, start timing (i.e., delay from the trigger), periodicity, duty-cycle, and the number of HS and DE pulses per frame are all programmable.

Note: After the VTG is enabled, the first and/or second frame sync output pulses (VS/HS/DE) may be invalid.

VTG Configuration

Configuring the VTG consists of two steps: selecting the VTG operation mode and configuring the timing parameters for the VS, HS, and DE generation as shown in the following table.

Table 58. MAX96793 VTG Operation Mode

Parameter	Register	Bitfield	Decode and Description
VS Generation Enable	VTX0	GEN_VS	0: Bypass VTG and use the VS signal from the video input 1: Generate VS signal from the VTG with specified timing
HS Generation Enable	VTX0	GEN_HS	0: Bypass VTG and use the HS signal from the video input 1: Generate HS signal from the VTG with specified timing
DE Generation Enable	VTX0	GEN_DE	0: Bypass VTG and use the DE signal from the video input 1: Generate DE signal from the VTG with specified timing
VS trigger mode	VTX0	VTG_MODE	00: VS tracking mode 01: VS one-trigger mode 10: Auto-repeat mode 11: Free-running mode (default)
Select PCLK source	VTX1	PATGEN_CLK_SRC	Pattern generator clock source for video PRBS, checkerboard, and gradient patterns. decode: 0XX: Use external clock 100: Use 25MHz internal clock 101: Use 75MHz internal clock 110: Use 150MHz internal clock 111: Use 375MHz internal clock
VS trigger polarity	VTX0	VS_TRIG	0: Falling edge 1: Rising edge (default)
Sync signal polarity	VTX0	VS_INV	The VTG timing configuration instructions assume positive sync polarity (sync pulse active high), so if negative VS polarity is desired, use this bit to invert 0: Do not invert VS signal 1: Invert VS signal **Note**: This bit is active even when GEN_VS=0 and can be used to invert the VS polarity without configuring the VTG timing parameters.
Sync signal polarity	VTX0	HS_INV	The VTG timing configuration instructions assume positive sync polarity (sync pulse active high), so if negative HS polarity is desired, use this bit to invert 0: Do not invert HS signal 1: Invert HS signal **Note**: This bit is active even when GEN_HS=0 and can be used to invert the HS polarity without configuring the VTG timing parameters.

Sync signal polarity	VTX0	DE_INV	The VTG timing configuration instructions assume positive sync polarity (sync pulse active high), so if negative DE polarity is desired, use this bit to invert 0: Do not invert DE signal 1: Invert DE signal Note: This bit is active even when GEN_DE=0 and can be used to invert the DE polarity without configuring the VTG timing parameters

VTG Trigger Modes

There are four different VTG trigger modes:

1. **VS tracking mode** – This mode is used to reduce the glitches and jitters of the input VS signal. In this mode, the input VS period (VS_HIGH + VS_LOW) is tracked. After the VS tracking has locked, any VS input edge not in the expected PCLK cycle (e.g., glitch) is ignored. VS tracking is locked upon three consecutive periodic matches; VS tracking is unlocked following three consecutive periodic mismatches. At power-up or if VS tracking is unlocked, the next VS input edge is assumed to be the correct VS edge.
2. **VS one-trigger mode** – In this mode, only one frame of VS, HS, and DE output signals is generated per VS input trigger. The polarity, timing (delay from the VS input trigger), and period/duty cycle of the generated VS, HS, and DE signals are in accordance with the user-programmed parameters.
3. **Auto-repeat mode** – This mode uses the VS input trigger to generate VS, HS, and DE signals as with VS one-trigger mode. However, instead of one frame per VS input trigger, auto-repeat mode generates continuous frames of VS, HS, and DE output signals following a VS input trigger. If the next VS input edge occurs earlier or later than expected by the VS period (VS_HIGH + VS_LOW), the newly generated frame is considered correct. The previous VS/HS/DE signals are cut or extended at the time point of the rising edge of the newly generated VS, HS, and DE signals.
4. **Free-running mode (default)** – This mode is based on auto-repeat mode. In this mode, the VS input signal is not needed to generate continuous frames of VS, HS, and DE output signals. The VTG automatically starts generating a continuous stream of frames, consisting of VS, HS, and DE signals in accordance with the user-programmed parameters.

VTG Timing Parameters

The sync pulse timing parameters for the VTG are shown in the following figure and configuring the timing parameters for the VS, HS, and DE generation is shown in the following table.

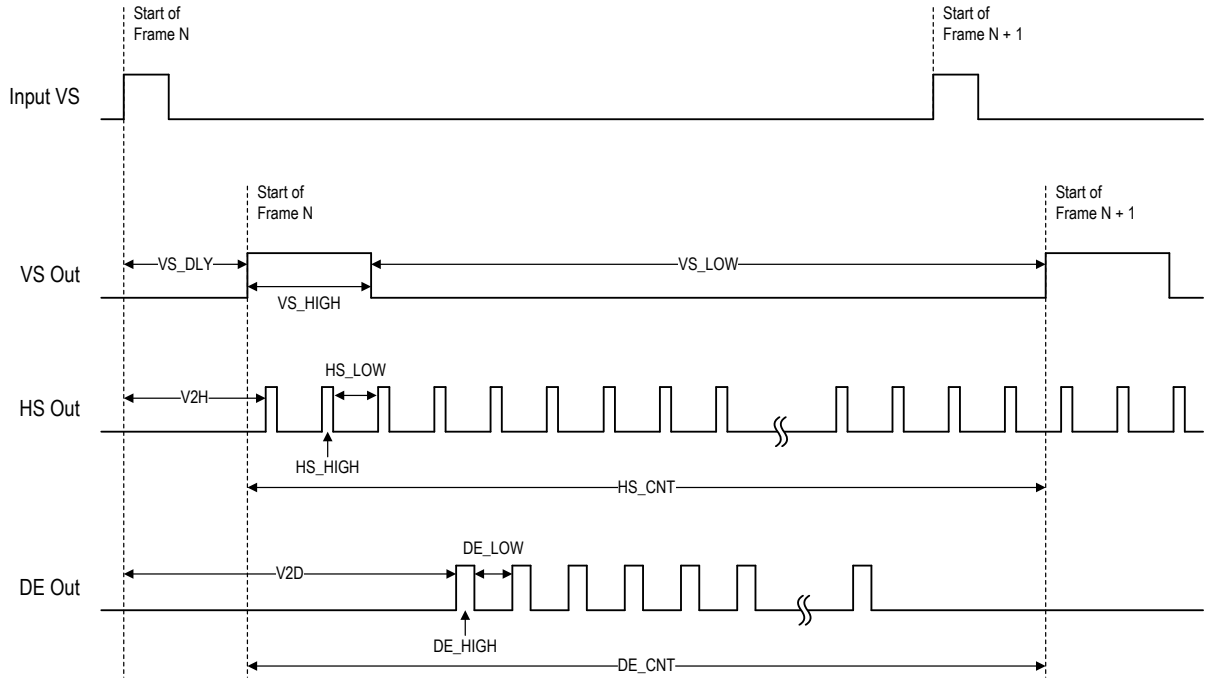

Figure 41. MAX96793 VTG Timing Parameters Diagram

All parameters are defined in terms of PCLKs. Vertical timing parameters must be converted from lines to pixel clocks by multiplying by H_{total}.

Only the parameters associated with the sync pulses that are enabled must be configured. For example, if only VS is being generated by the VTG (GEN_VS=1, GEN_HS=0, GEN_DE=0), only VS_DLY, VS_HIGH, and VS_LOW must be configured.

Note: In cases where the VTG is used to regenerate DE in combination with incoming video data, the incoming data is not retimed to DE if it is adjusted from the original timing. This will result in lost video data. E.g., If DE is delayed four PCLKs, the first four pixels from the incoming data are lost and the last four are converted to zeros (i.e., padded).

Table 59. MAX96793 Timing Parameter Configuration Registers

Parameter	Register	Bitfield	Description
VS_DLY	VTX2 VTX3 VTX4	VS_DLY_2[7:0] VS_DLY_1[7:0] VS_DLY_0[7:0]	The delay from the VS trigger to the generated VS signal in terms of PCLKs. Note, if using the input video stream, this will delay the output sync signals relative to the video data.
VS_HIGH	VTX5 VTX6 VTX7	VS_HIGH_2[7:0] VS_HIGH_1[7:0] VS_HIGH_0[7:0]	The high duration of the generated VS output signal in terms of PCLKs
VS_LOW	VTX8 VTX9 VTX10	VS_LOW_2[7:0] VS_LOW_1[7:0] VS_LOW_0[7:0]	The low duration of the generated VS output signal in terms of PCLKs

V2H	VTX11 VTX12 VTX13	V2H_2[7:0] V2H_1[7:0] V2H_0[7:0]	The delay from the VS trigger to the rising edge of the generated HS signal
HS_HIGH	VTX14 VTX15	HS_HIGH_1[7:0] HS_HIGH_0[7:0]	The high duration of the generated HS output signal in terms of PCLKs
HS_LOW	VTX16 VTX17	HS_LOW_1[7:0] HS_LOW_0[7:0]	The low duration of the generated HS output signal in terms of PCLKs
HS_CNT	VTX18 VTX19	HS_CNT_1[7:0] HS_CNT_0[7:0]	The number of HS output pulses generated per video frame
V2D	VTX20 VTX21 VTX22	V2D_2[7:0] V2D_1[7:0] V2D_0[7:0]	The delay from the VS trigger to the rising edge of the generated DE signal
DE_HIGH	VTX23 VTX24	DE_HIGH_1[7:0] DE_HIGH_0[7:0]	The high duration of the generated DE output signal in terms of PCLKs
DE_LOW	VTX25 VTX26	DE_LOW_1[7:0] DE_LOW_0[7:0]	The low duration of the generated DE output signal in terms of PCLKs
DE_CNT	VTX27 VTX28	DE_CNT_1[7:0] DE_CNT_0[7:0]	The number of DE pulses generated per video frame

Use Case Programming Examples

Overview

The following use case examples demonstrate how many of the features described throughout this document can be used together to program a SerDes system. These examples may need to be manipulated or completely changed for more specific use cases. The basic flow of programming and important steps is annotated to give a broad picture of the requirements users can expect to get a system working.

The format of the programming examples throughout this section follow the format allowable by the GMSL GUI for .CPP files, so that users may copy them for use immediately.

Use Case Examples

The following examples (Description, Block Diagram, and Script) shown are with the MAX96793 serializer and MAX96792A deserializer.

Use Case Example #1

Example #1 is a common use case that that takes two image sensors data and aggregates them out of one DES MIPI output port. In this example the image sensor outputted data on the same virtual channel (VC0), but one image data stream VC was renumbered in the DES. Now, the SoC can filter data between the two image sensors.

-Image Sensor #1 (Input to SER):

Virtual Channel: VC0

Data Type(s): RAW12 & EMB8

-Image Sensor #2 (Input to SER):

Virtual Channel: VC0

Data Type(s): RAW16 & EMB8

-DES MIPI Output A (Input to SoC):

VC0, RAW12 & EMB8

VC1, RAW16 & EMB8

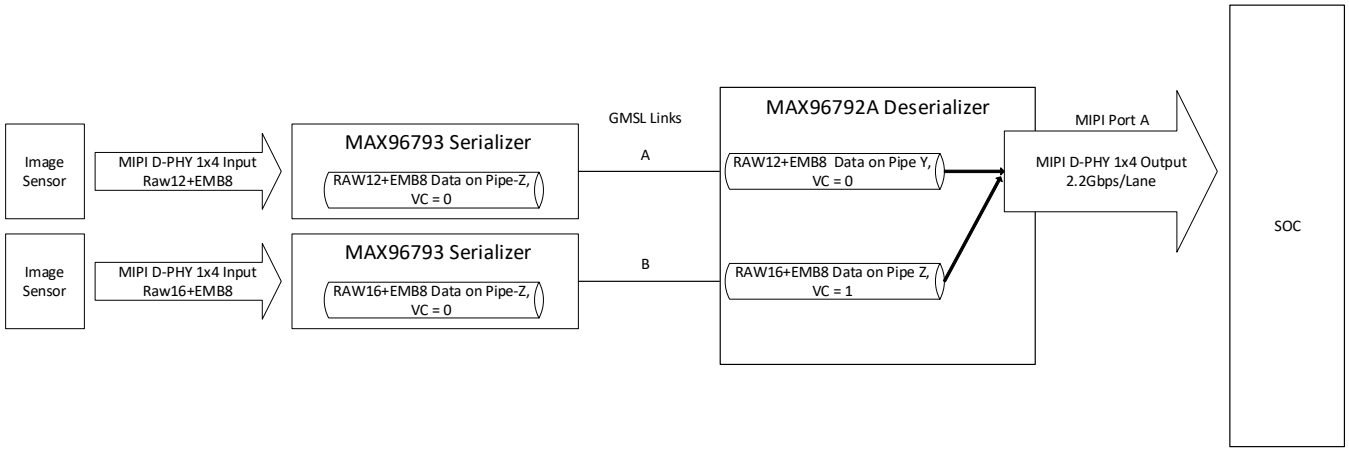

Figure 42. MAX96793 Use Case Example #1

// GMSL-A / Serializer: MAX96793 (Pixel Mode) / Mode: 1x4 / Device Address: 0x80 / Multiple-VC Case: Single VC / Multiple-VC Pipe Sharing: N/A

// PipeZ:

// Input Stream: VC0 RAW12 PortB (D-PHY)

// Input Stream: VC0 EMB8 PortB (D-PHY)

// GMSL-B / Serializer: MAX96793 (Pixel Mode) / Mode: 1x4 / Device Address: 0x80 / Multiple-VC Case: Single VC / Multiple-VC Pipe Sharing: N/A

// PipeZ:

// Input Stream: VC0 RAW16 PortB (D-PHY)

// Input Stream: VC0 EMB8 PortB (D-PHY)

// Deserializer: MAX96792A / Mode: 2 (1x4) / Device Address: 0x98

// PipeY:

// GMSL-A Input Stream: VC0 RAW12 PortB - Output Stream: VC0 RAW12 PortA (D-PHY)

// GMSL-A Input Stream: VC0 EMB8 PortB - Output Stream: VC0 EMB8 PortA (D-PHY)

// PipeZ:

// GMSL-B Input Stream: VC0 RAW16 PortB - Output Stream: VC1 RAW16 PortA (D-PHY)

// GMSL-B Input Stream: VC0 EMB8 PortB - Output Stream: VC1 EMB8 PortA (D-PHY)

0x04,0x98,0x03,0x13,0x00, // (CSI_OUT_EN): CSI output disabled

// Single Link Initialization Before Serializer Device Address Change

0x04,0x98,0x00,0x10,0x02, // (AUTO_LINK): Disabled | (LINK_CFG): 0x2

0x04,0x98,0x0F,0x00,0x02, // (LINK_EN_A): Disabled | (Default) (LINK_EN_B): Enabled

0x04,0x98,0x00,0x12,0x24, // (RESET_ONESHOT LINK B): Activated

0x00,0x78,

// GMSL-B Serializer Address Change from 0x80 to 0x82

0x04,0x80,0x00,0x00,0x82, // DEV : REG0 | DEV_ADDR (DEV_ADDR): 0x41

// Link Initialization for Deserializer

0x04,0x98,0x00,0x10,0x23, // (Default) (AUTO_LINK): Disabled | (LINK_CFG): 0x3 | (RESET_ONESHOT LINK A): Activated

0x04,0x98,0x00,0x12,0x24, // (Default) (RESET_ONESHOT LINK B): Activated

0x00,0x78,

// Video Transmit Configuration for Serializer(s)

0x04,0x80,0x00,0x02,0x03, // DEV : REG2 | VID_TX_EN_Z (VID_TX_EN_Z): Disabled

0x04,0x82,0x00,0x02,0x03, // DEV : REG2 | VID_TX_EN_Z (VID_TX_EN_Z): Disabled

//

// INSTRUCTIONS FOR GMSL-A SERIALIZER MAX96793

//

// MIPI DPHY Configuration

0x04,0x80,0x03,0x30,0x00, // MIPI_RX : MIPI_RX0 | (Default) RSVD (Port Configuration): 1x4

0x04,0x80,0x03,0x83,0x00, // MIPI_RX_EXT : EXT11 | Tun_Mode (Tunnel Mode): Disabled

0x04,0x80,0x03,0x31,0x30, // MIPI_RX : MIPI_RX1 | (Default) ctrl1_num_lanes (Port B - Lane Count): 4

0x04,0x80,0x03,0x32,0xE0, // MIPI_RX : MIPI_RX2 | (Default) phy1_lane_map (Lane Map - PHY1 D0): Lane 2 | (Default) phy1_lane_map (Lane Map - PHY1 D1): Lane 3

0x04,0x80,0x03,0x33,0x04, // MIPI_RX : MIPI_RX3 | (Default) phy2_lane_map (Lane Map - PHY2 D0): Lane 0 | (Default) phy2_lane_map (Lane Map - PHY2 D1): Lane 1

0x04,0x80,0x03,0x34,0x00, // MIPI_RX : MIPI_RX4 | (Default) phy1_pol_map (Polarity - PHY1 Lane 0): Normal | (Default) phy1_pol_map (Polarity - PHY1 Lane 1): Normal

0x04,0x80,0x03,0x35,0x00, // MIPI_RX : MIPI_RX5 | (Default) phy2_pol_map (Polarity - PHY2 Lane 0): Normal | (Default) phy2_pol_map (Polarity - PHY2 Lane 1): Normal | (Default) phy2_pol_map (Polarity - PHY2 Clock Lane): Normal

// Controller to Pipe Mapping Configuration

0x04,0x80,0x03,0x08,0x64, // FRONTTOP : FRONTTOP_0 | (Default) RSVD (CLK_SELZ): Port B | (Default) START_PORTB (START_PORTB): Enabled

0x04,0x80,0x03,0x11,0x40, // FRONTTOP : FRONTTOP_9 | (Default) START_PORTBZ (START_PORTBZ): Start Video

0x04,0x80,0x03,0x18,0x6C, // FRONTTOP : FRONTTOP_16 | mem_dt1_selz (mem_dt1_selz): 0x6C

0x04,0x80,0x03,0x19,0x52, // FRONTTOP : FRONTTOP_17 | mem_dt2_selz (mem_dt2_selz): 0x52

0x04,0x80,0x03,0x15,0x80, // (independent_vs_mode): Enabled

0x04,0x80,0x03,0x0D,0x01, // FRONTTOP : FRONTTOP_5 | VC_SELZ_L (VC_SELZ_L): 0x1

// Double Mode Configuration

0x04,0x80,0x03,0x12,0x04, // FRONTTOP : FRONTTOP_10 | bpp8dblz (bpp8dblz): Send 8-bit pixels as 16-bit

0x04,0x80,0x03,0x1E,0x2C, // FRONTTOP : FRONTTOP_22 | soft_bppz (soft_bppz): 0xC | soft_bppz_en (soft_bppz_en): Software override enabled

// Pipe Configuration

0x04,0x80,0x00,0x5B,0x01, // CFGV__VIDEO_Z : TX3 | TX_STR_SEL (TX_STR_SEL Pipe Z): 0x1

//

// INSTRUCTIONS FOR GMSL-B SERIALIZER MAX96793

//

// MIPI DPHY Configuration

0x04,0x82,0x03,0x30,0x00, // MIPI_RX : MIPI_RX0 | (Default) RSVD (Port Configuration): 1x4

0x04,0x82,0x03,0x83,0x00, // MIPI_RX_EXT : EXT11 | Tun_Mode (Tunnel Mode): Disabled

0x04,0x82,0x03,0x31,0x30, // MIPI_RX : MIPI_RX1 | (Default) ctrl1_num_lanes (Port B - Lane Count): 4

0x04,0x82,0x03,0x32,0xE0, // MIPI_RX : MIPI_RX2 | (Default) phy1_lane_map (Lane Map - PHY1 D0): Lane 2 | (Default) phy1_lane_map (Lane Map - PHY1 D1): Lane 3

0x04,0x82,0x03,0x33,0x04, // MIPI_RX : MIPI_RX3 | (Default) phy2_lane_map (Lane Map - PHY2 D0): Lane 0 | (Default) phy2_lane_map (Lane Map - PHY2 D1): Lane 1

0x04,0x82,0x03,0x34,0x00, // MIPI_RX : MIPI_RX4 | (Default) phy1_pol_map (Polarity - PHY1 Lane 0): Normal | (Default) phy1_pol_map (Polarity - PHY1 Lane 1): Normal

0x04,0x82,0x03,0x35,0x00, // MIPI_RX : MIPI_RX5 | (Default) phy2_pol_map (Polarity - PHY2 Lane 0): Normal | (Default) phy2_pol_map (Polarity - PHY2 Lane 1): Normal | (Default) phy2_pol_map (Polarity - PHY2 Clock Lane): Normal

// Controller to Pipe Mapping Configuration

0x04,0x82,0x03,0x08,0x64, // FRONTTOP : FRONTTOP_0 | (Default) RSVD (CLK_SELZ): Port B | (Default) START_PORTB (START_PORTB): Enabled

0x04,0x82,0x03,0x11,0x40, // FRONTTOP : FRONTTOP_9 | (Default) START_PORTBZ (START_PORTBZ): Start Video

0x04,0x82,0x03,0x18,0x6E, // FRONTTOP : FRONTTOP_16 | mem_dt1_selz (mem_dt1_selz): 0x6E

0x04,0x82,0x03,0x19,0x52, // FRONTTOP : FRONTTOP_17 | mem_dt2_selz (mem_dt2_selz): 0x52

0x04,0x82,0x03,0x15,0x80, // (independent_vs_mode): Enabled

0x04,0x82,0x03,0x0D,0x01, // FRONTTOP : FRONTTOP_5 | VC_SELZ_L (VC_SELZ_L): 0x1

// Double Mode Configuration

0x04,0x82,0x03,0x12,0x04, // FRONTTOP : FRONTTOP_10 | bpp8dblz (bpp8dblz): Send 8-bit pixels as 16-bit

0x04,0x82,0x03,0x1E,0x30, // FRONTTOP : FRONTTOP_22 | soft_bppz (soft_bppz): 0x10 | soft_bppz_en (soft_bppz_en): Software override enabled

// Pipe Configuration

0x04,0x82,0x00,0x5B,0x02, // CFGV__VIDEO_Z : TX3 | (Default) TX_STR_SEL (TX_STR_SEL Pipe Z): 0x2

// INSTRUCTIONS FOR DESERIALIZER MAX96792A

// Video Pipes And Routing Configuration

0x04,0x98,0x01,0x61,0x31, // (STR_SELY): Link A Stream Id 1 | (Default) (STR_SELZ): Link B Stream Id 2

// Pipe to Controller Mapping Configuration

0x04,0x98,0x04,0x4B,0x0F, // (MAP_EN_L Pipe Y): 0xF

0x04,0x98,0x04,0x4C,0x00, // (Default) (MAP_EN_H Pipe Y): 0x0

0x04,0x98,0x04,0x4D,0x2C, // (MAP_SRC_0 Pipe Y DT): 0x2C | (Default) (MAP_SRC_0 Pipe Y VC): 0x0

0x04,0x98,0x04,0x4E,0x2C, // (MAP_DST_0 Pipe Y DT): 0x2C | (Default) (MAP_DST_0 Pipe Y VC): 0x0

0x04,0x98,0x04,0x4F,0x00, // (Default) (MAP_SRC_1 Pipe Y DT): 0x0 | (Default) (MAP_SRC_1 Pipe Y VC): 0x0

0x04,0x98,0x04,0x50,0x00, // (Default) (MAP_DST_1 Pipe Y DT): 0x0 | (Default) (MAP_DST_1 Pipe Y VC): 0x0

0x04,0x98,0x04,0x51,0x01, // (MAP_SRC_2 Pipe Y DT): 0x1 | (Default) (MAP_SRC_2 Pipe Y VC): 0x0

0x04,0x98,0x04,0x52,0x01, // (MAP_DST_2 Pipe Y DT): 0x1 | (Default) (MAP_DST_2 Pipe Y VC): 0x0

0x04,0x98,0x04,0x53,0x12, // (MAP_SRC_3 Pipe Y DT): 0x12 | (Default) (MAP_SRC_3 Pipe Y VC): 0x0

0x04,0x98,0x04,0x54,0x12, // (MAP_DST_3 Pipe Y DT): 0x12 | (Default) (MAP_DST_3 Pipe Y VC): 0x0

0x04,0x98,0x04,0x6D,0x55, // (MAP_DPHY_DST_0 Pipe Y): 0x1 | (MAP_DPHY_DST_1 Pipe Y): 0x1 | (MAP_DPHY_DST_2 Pipe Y): 0x1 | (MAP_DPHY_DST_3 Pipe Y): 0x1

0x04,0x98,0x04,0x8B,0x0F, // (MAP_EN_L Pipe Z): 0xF

0x04,0x98,0x04,0x8C,0x00, // (Default) (MAP_EN_H Pipe Z): 0x0

0x04,0x98,0x04,0x8D,0x2E, // (MAP_SRC_0 Pipe Z DT): 0x2E | (Default) (MAP_SRC_0 Pipe Z VC): 0x0

0x04,0x98,0x04,0x8E,0x6E, // (MAP_DST_0 Pipe Z DT): 0x2E | (MAP_DST_0 Pipe Z VC): 0x1

0x04,0x98,0x04,0x8F,0x00, // (Default) (MAP_SRC_1 Pipe Z DT): 0x0 | (Default) (MAP_SRC_1 Pipe Z VC): 0x0

0x04,0x98,0x04,0x90,0x40, // (Default) (MAP_DST_1 Pipe Z DT): 0x0 | (MAP_DST_1 Pipe Z VC): 0x1

0x04,0x98,0x04,0x91,0x01, // (MAP_SRC_2 Pipe Z DT): 0x1 | (Default) (MAP_SRC_2 Pipe Z VC): 0x0

0x04,0x98,0x04,0x92,0x41, // (MAP_DST_2 Pipe Z DT): 0x1 | (MAP_DST_2 Pipe Z VC): 0x1

0x04,0x98,0x04,0x93,0x12, // (MAP_SRC_3 Pipe Z DT): 0x12 | (Default) (MAP_SRC_3 Pipe Z VC): 0x0

0x04,0x98,0x04,0x94,0x52, // (MAP_DST_3 Pipe Z DT): 0x12 | (MAP_DST_3 Pipe Z VC): 0x1

0x04,0x98,0x04,0xAD,0x55, // (MAP_DPHY_DST_0 Pipe Z): 0x1 | (MAP_DPHY_DST_1 Pipe Z): 0x1 |
(MAP_DPHY_DST_2 Pipe Z): 0x1 | (MAP_DPHY_DST_3 Pipe Z): 0x1

// Double Mode Configuration

0x04,0x98,0x04,0x73,0x10, // (ALT2_MEM_MAP8 CTRL1): Alternate memory map enabled

// MIPI DPHY Configuration

0x04,0x98,0x03,0x30,0x04, // (Default) (Port Configuration): 2 (1x4)

0x04,0x98,0x04,0x4A,0xD0, // (Default) (Port A - Lane Count): 4

0x04,0x98,0x03,0x33,0x4E, // (Default) (Lane Map - PHY0 D0): Lane 2 | (Default) (Lane Map - PHY0 D1): Lane 3 |
(Default) (Lane Map - PHY1 D0): Lane 0 | (Default) (Lane Map - PHY1 D1): Lane 1

0x04,0x98,0x03,0x35,0x00, // (Default) (Polarity - PHY0 Lane 0): Normal | (Default) (Polarity - PHY0 Lane 1): Normal |
(Default) (Polarity - PHY1 Lane 0): Normal | (Default) (Polarity - PHY1 Lane 1): Normal | (Default) (Polarity - PHY1
Clock Lane): Normal

0x04,0x98,0x04,0x43,0x81, // (Controller 1 Auto Initial Deskew): Enabled

// This is to set predefined (coarse) CSI output frequency

// CSI Phy 1 is 2200 Mbps/lane.

0x04,0x98,0x1D,0x00,0xF4,

0x04,0x98,0x03,0x20,0x36,

0x04,0x98,0x1D,0x00,0xF5,

0x04,0x98,0x03,0x32,0x34, // (phy_Stdby_2): Put PHY2 in standby mode | (phy_Stdby_3): Put PHY3 in standby
mode

0x04,0x98,0x03,0x13,0x02, // (CSI_OUT_EN): CSI output enabled

// Video Transmit Configuration for Serializer(s)

0x04,0x80,0x00,0x02,0x43, // DEV : REG2 | VID_TX_EN_Z (VID_TX_EN_Z): Enabled

0x04,0x82,0x00,0x02,0x43, // DEV : REG2 | VID_TX_EN_Z (VID_TX_EN_Z): Enabled

Use Case Example #2

Example #2 is a use case that that takes two image sensors data and sends them independently out of two DES MIPI outputs.

-Image Sensor #1 (Input to SER):

Virtual Channel: VC0

Data Type(s): RAW12 & EMB8

-Image Sensor #2 (Input to SER):

Virtual Channel: VC0

Data Type(s): RAW10 & EMB8

-DES MIPI Output A (Input to SoC):

VC0, RAW12 & EMB8

-DES MIPI Output B (Input to SoC):

VC0, RAW10 & EMB8

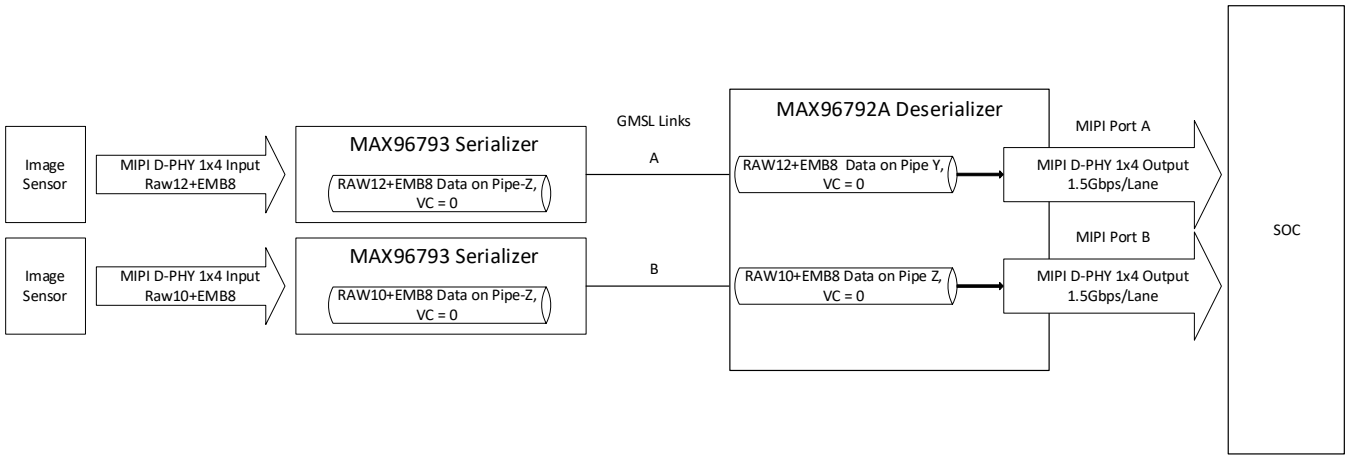

Figure 43. MAX96793 Use Case Example #2

// GMSL-A / Serializer: MAX96793 (Tunnel Mode) / Mode: 1x4 / Device Address: 0x80 / Multiple-VC Case: Single VC / Multiple-VC Pipe Sharing: N/A

// PipeZ:

// Input Stream: VC0 RAW12 PortB (D-PHY)

// Input Stream: VC0 EMB8 PortB (D-PHY)

// GMSL-B / Serializer: MAX96793 (Tunnel Mode) / Mode: 1x4 / Device Address: 0x80 / Multiple-VC Case: Single VC / Multiple-VC Pipe Sharing: N/A

// PipeZ:

// Input Stream: VC0 RAW10 PortB (D-PHY)

// Input Stream: VC0 EMB8 PortB (D-PHY)

// Deserializer: MAX96792A / Mode: 2 (1x4) / Device Address: 0x98

// PipeY:

// GMSL-A Input Stream: VC0 RAW12 PortB - Output Stream: VC0 RAW12 PortA (D-PHY)

// GMSL-A Input Stream: VC0 EMB8 PortB - Output Stream: VC0 EMB8 PortA (D-PHY)

// PipeZ:

// GMSL-B Input Stream: VC0 RAW10 PortB - Output Stream: VC0 RAW10 PortB (D-PHY)

// GMSL-B Input Stream: VC0 EMB8 PortB - Output Stream: VC0 EMB8 PortB (D-PHY)

0x04,0x98,0x03,0x13,0x00, // (CSI_OUT_EN): CSI output disabled

// Single Link Initialization Before Serializer Device Address Change

0x04,0x98,0x00,0x10,0x02, // (AUTO_LINK): Disabled | (LINK_CFG): 0x2

0x04,0x98,0x0F,0x00,0x02, // (LINK_EN_A): Disabled | (Default) (LINK_EN_B): Enabled

0x04,0x98,0x00,0x12,0x24, // (RESET_ONESHOT LINK B): Activated

0x00,0x78,

// GMSL-B Serializer Address Change from 0x80 to 0x82

0x04,0x80,0x00,0x00,0x82, // DEV : REG0 | DEV_ADDR (DEV_ADDR): 0x41

// Link Initialization for Deserializer

0x04,0x98,0x00,0x10,0x23, // (Default) (AUTO_LINK): Disabled | (LINK_CFG): 0x3 | (RESET_ONESHOT LINK A): Activated

0x04,0x98,0x00,0x12,0x24, // (Default) (RESET_ONESHOT LINK B): Activated

0x00,0x78,

// Video Transmit Configuration for Serializer(s)

0x04,0x80,0x00,0x02,0x03, // DEV : REG2 | VID_TX_EN_Z (VID_TX_EN_Z): Disabled

0x04,0x82,0x00,0x02,0x03, // DEV : REG2 | VID_TX_EN_Z (VID_TX_EN_Z): Disabled

//

// INSTRUCTIONS FOR GMSL-A SERIALIZER MAX96793

//

// MIPI DPHY Configuration

0x04,0x80,0x03,0x30,0x00, // MIPI_RX : MIPI_RX0 | (Default) RSVD (Port Configuration): 1x4

0x04,0x80,0x03,0x83,0x80, // MIPI_RX_EXT : EXT11 | (Default) Tun_Mode (Tunnel Mode): Enabled

0x04,0x80,0x03,0x31,0x30, // MIPI_RX : MIPI_RX1 | (Default) ctrl1_num_lanes (Port B - Lane Count): 4

0x04,0x80,0x03,0x32,0xE0, // MIPI_RX : MIPI_RX2 | (Default) phy1_lane_map (Lane Map - PHY1 D0): Lane 2 | (Default) phy1_lane_map (Lane Map - PHY1 D1): Lane 3

0x04,0x80,0x03,0x33,0x04, // MIPI_RX : MIPI_RX3 | (Default) phy2_lane_map (Lane Map - PHY2 D0): Lane 0 | (Default) phy2_lane_map (Lane Map - PHY2 D1): Lane 1

0x04,0x80,0x03,0x34,0x00, // MIPI_RX : MIPI_RX4 | (Default) phy1_pol_map (Polarity - PHY1 Lane 0): Normal | (Default) phy1_pol_map (Polarity - PHY1 Lane 1): Normal

0x04,0x80,0x03,0x35,0x00, // MIPI_RX : MIPI_RX5 | (Default) phy2_pol_map (Polarity - PHY2 Lane 0): Normal | (Default) phy2_pol_map (Polarity - PHY2 Lane 1): Normal | (Default) phy2_pol_map (Polarity - PHY2 Clock Lane): Normal

// Controller to Pipe Mapping Configuration

0x04,0x80,0x03,0x08,0x64, // FRONTTOP : FRONTTOP_0 | (Default) RSVD (CLK_SELZ): Port B | (Default) START_PORTB (START_PORTB): Enabled

0x04,0x80,0x03,0x11,0x40, // FRONTTOP : FRONTTOP_9 | (Default) START_PORTBZ (START_PORTBZ): Start Video

0x04,0x80,0x03,0x15,0x00, // (Default) (independent_vs_mode): Disabled

// Pipe Configuration

0x04,0x80,0x00,0x5B,0x00, // CFGV__VIDEO_Z : TX3 | TX_STR_SEL (TX_STR_SEL Pipe Z): 0x0

//

// INSTRUCTIONS FOR GMSL-B SERIALIZER MAX96793

//

// MIPI DPHY Configuration

0x04,0x82,0x03,0x30,0x00, // MIPI_RX : MIPI_RX0 | (Default) RSVD (Port Configuration): 1x4

0x04,0x82,0x03,0x83,0x80, // MIPI_RX_EXT : EXT11 | (Default) Tun_Mode (Tunnel Mode): Enabled

0x04,0x82,0x03,0x31,0x30, // MIPI_RX : MIPI_RX1 | (Default) ctrl1_num_lanes (Port B - Lane Count): 4

0x04,0x82,0x03,0x32,0xE0, // MIPI_RX : MIPI_RX2 | (Default) phy1_lane_map (Lane Map - PHY1 D0): Lane 2 | (Default) phy1_lane_map (Lane Map - PHY1 D1): Lane 3

0x04,0x82,0x03,0x33,0x04, // MIPI_RX : MIPI_RX3 | (Default) phy2_lane_map (Lane Map - PHY2 D0): Lane 0 | (Default) phy2_lane_map (Lane Map - PHY2 D1): Lane 1

0x04,0x82,0x03,0x34,0x00, // MIPI_RX : MIPI_RX4 | (Default) phy1_pol_map (Polarity - PHY1 Lane 0): Normal | (Default) phy1_pol_map (Polarity - PHY1 Lane 1): Normal

0x04,0x82,0x03,0x35,0x00, // MIPI_RX : MIPI_RX5 | (Default) phy2_pol_map (Polarity - PHY2 Lane 0): Normal | (Default) phy2_pol_map (Polarity - PHY2 Lane 1): Normal | (Default) phy2_pol_map (Polarity - PHY2 Clock Lane): Normal

// Controller to Pipe Mapping Configuration

0x04,0x82,0x03,0x08,0x64, // FRONTTOP : FRONTTOP_0 | (Default) RSVD (CLK_SELZ): Port B | (Default) START_PORTB (START_PORTB): Enabled

0x04,0x82,0x03,0x11,0x40, // FRONTTOP : FRONTTOP_9 | (Default) START_PORTBZ (START_PORTBZ): Start Video

0x04,0x82,0x03,0x15,0x00, // (Default) (independent_vs_mode): Disabled

// Pipe Configuration

0x04,0x82,0x00,0x5B,0x03, // CFGV__VIDEO_Z : TX3 | TX_STR_SEL (TX_STR_SEL Pipe Z): 0x3

// INSTRUCTIONS FOR DESERIALIZER MAX96792A

// Video Pipes And Routing Configuration

0x04,0x98,0x01,0x61,0x38, // (STR_SELY): Link A Stream Id 0 | (STR_SELZ): Link B Stream Id 3

// Double Mode Configuration

// MIPI DPHY Configuration

0x04,0x98,0x03,0x30,0x04, // (Default) (Port Configuration): 2 (1x4)

0x04,0x98,0x04,0x74,0x09, // (Port A Tunnel Mode): Enabled

0x04,0x98,0x04,0x4A,0xD0, // (Default) (Port A - Lane Count): 4

0x04,0x98,0x03,0x33,0x4E, // (Default) (Lane Map - PHY0 D0): Lane 2 | (Default) (Lane Map - PHY0 D1): Lane 3 | (Default) (Lane Map - PHY1 D0): Lane 0 | (Default) (Lane Map - PHY1 D1): Lane 1

0x04,0x98,0x03,0x35,0x00, // (Default) (Polarity - PHY0 Lane 0): Normal | (Default) (Polarity - PHY0 Lane 1): Normal | (Default) (Polarity - PHY1 Lane 0): Normal | (Default) (Polarity - PHY1 Lane 1): Normal | (Default) (Polarity - PHY1 Clock Lane): Normal

// This is to set predefined (coarse) CSI output frequency

// CSI Phy 1 is 1500 Mbps/lane.

0x04,0x98,0x1D,0x00,0xF4,

0x04,0x98,0x03,0x20,0x2F, // (Default)

0x04,0x98,0x1D,0x00,0xF5,

0x04,0x98,0x04,0xB4,0x0F, // (Port B Tunnel Mode): Enabled

0x04,0x98,0x04,0x8A,0xD0, // (Default) (Port B - Lane Count): 4

0x04,0x98,0x03,0x34,0xE4, // (Default) (Lane Map - PHY2 D0): Lane 0 | (Default) (Lane Map - PHY2 D1): Lane 1 | (Default) (Lane Map - PHY3 D0): Lane 2 | (Default) (Lane Map - PHY3 D1): Lane 3

0x04,0x98,0x03,0x36,0x00, // (Default) (Polarity - PHY2 Lane 0): Normal | (Default) (Polarity - PHY2 Lane 1): Normal | (Default) (Polarity - PHY3 Lane 0): Normal | (Default) (Polarity - PHY3 Lane 1): Normal | (Default) (Polarity - PHY2 Clock Lane): Normal

// This is to set predefined (coarse) CSI output frequency

// CSI Phy 2 is 1500 Mbps/lane.

0x04,0x98,0x1E,0x00,0xF4,

0x04,0x98,0x03,0x23,0x2F, // (Default)

0x04,0x98,0x1E,0x00,0xF5,

// Tunnel Mode Configuration

0x04,0x98,0x03,0x13,0x02, // (CSI_OUT_EN): CSI output enabled

// Video Transmit Configuration for Serializer(s)

0x04,0x80,0x00,0x02,0x43, // DEV : REG2 | VID_TX_EN_Z (VID_TX_EN_Z): Enabled

0x04,0x82,0x00,0x02,0x43, // DEV : REG2 | VID_TX_EN_Z (VID_TX_EN_Z): Enabled

Revision History

REVISION NUMBER	REVISION DATE	DESCRIPTION
0	5/24	Initial release

#

#