AHEAD OF WHAT'S POSSIBLE™

MAX96716A/MAX96716F User Guide

GMSL SerDes Applications Team

Version 0

11/23

Table of Contents

MAX96716A/MAX96716F Dual Deserializers

Device Overview

This user guide is intended for use in conjunction with other documents such as the MAX96716A/MAX96716F 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 necessary to ensure reliable operation in production. Be sure to contact your Analog Devices field applications engineer or representative to obtain the errata documents. Make sure to 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 MAX96716A/MAX96716F device for testing.

The MAX96716A and MAX96716F are both full featured GMSL2 deserializer devices; only exception is that MAX96716F only supports 3Gbps operation.

Table 1. Comparing the MAX96716A/MAX96716F Family

Part Number	Forward Link Rate	Coax/STP	MIPI Output	GMSL Links	ASIL Rating
MAX96716A	3Gbps or 6Gbps	Coax or STP	DPHY/CPHY	2	B
MAX96716F	3Gbps	Coax or STP	DPHY/CPHY	2	B

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

GMSL2 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 MAX96716A is capable of 3Gbps or 6Gbps forward link rate (selectable with resistors connected to the CFG pin or with register writes) while the MAX96716F is capable of 3Gbps. Both variants have a 187.5 Mbps reverse direction rate.

Application Use Case

In a typical configuration, the MAX96716A/MAX96716F deserializers accept two GMSL2 links used to support two cameras in the 1MP to 8MP range (*Figure 1*). Typically, each camera's image sensor feeds the video into the D-PHY CSI input port of a serializer. The serializer then takes that data, converts it to GMSL, and sends it over the link to a MAX96716A/MAX96716F deserializer. In *Figure 1*, there are two MAX96717 devices operating independently and sending data out to a MAX96716A/MAX96716F deserializer. Coax or shielded twisted-pair (STP) cables can be used for the GMSL link.

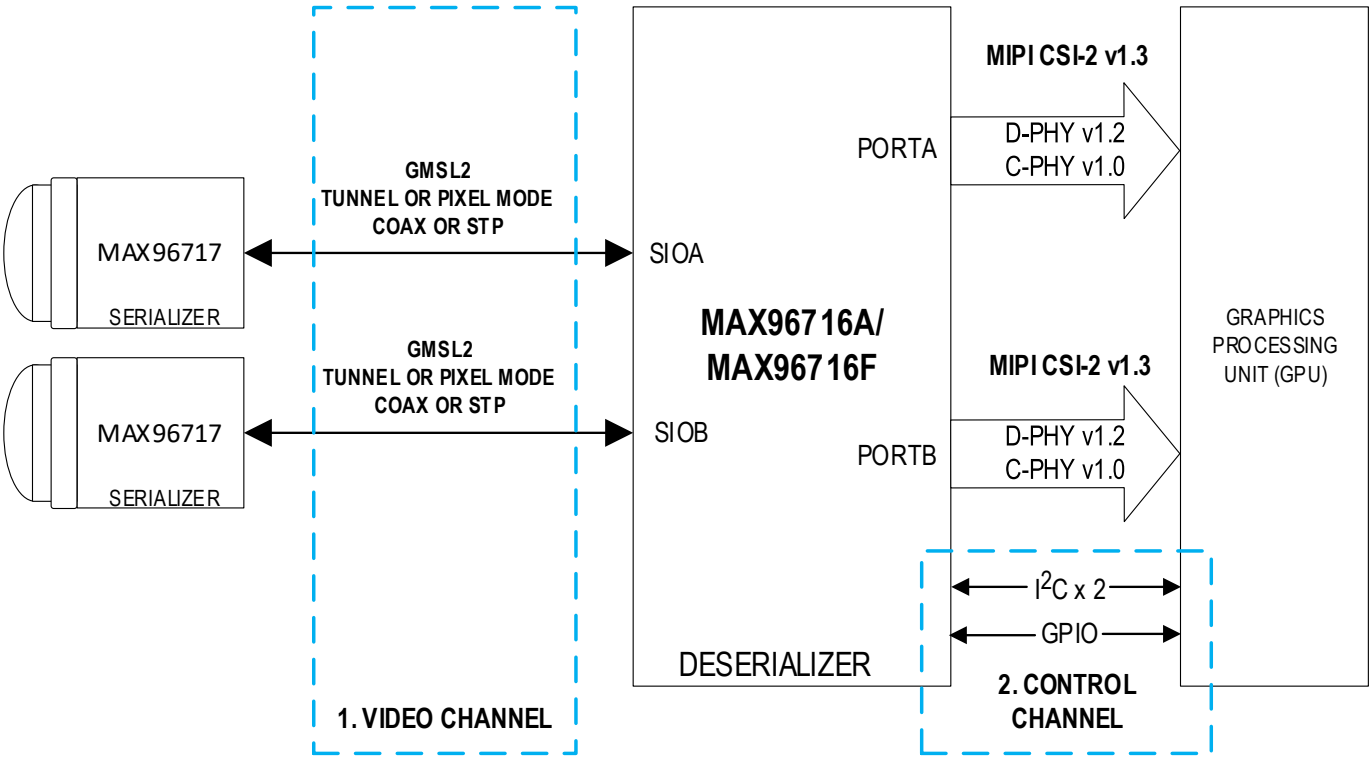

Figure 1. MAX96716A/MAX96716F Two Camera Application Example

Architecture

The MAX96716A/F video path starts with two GMSL PHYs that route to two internal video pipes (Pipe Y and Z), which connect to a CSI-2 controller and end with MIPI PHYs. *Figure 2* shows the default video path settings. Depending on the use case, PHY to pipe, pipe to controller, and controller to MIPI PHY can be altered.

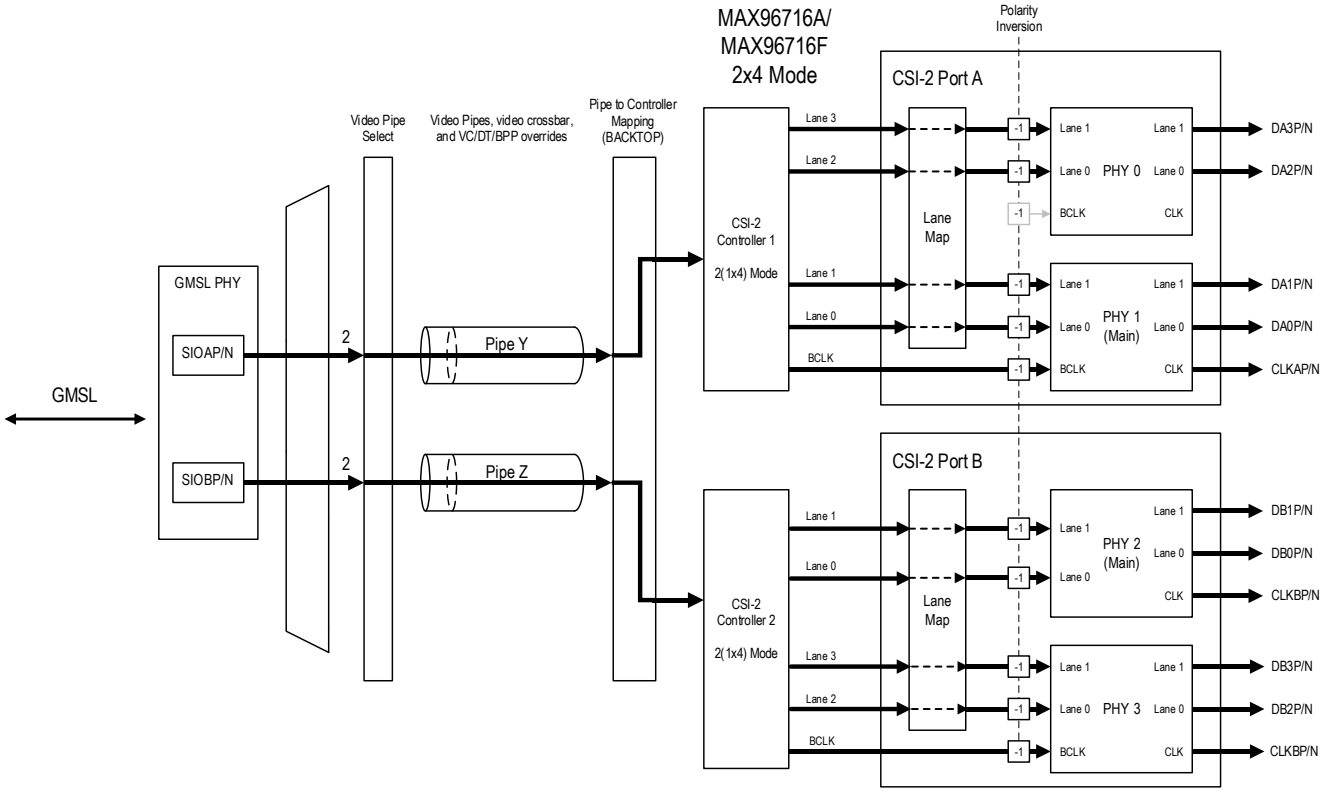

Figure 2. MAX96716A/MAX96716F Video Path (2x4 D-PHY Mode)

Start-Up and Programming Sequence

Overview

The MAX96716A/MAX96716F devices have many applications 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. MAX96716A/MAX96716F Start-Up Sequence

MAX96716A/MAX96716F 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)	
2	I^2C Wakeup 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.
3	CFG Pins Automatically Set Link	CFG pins sampled on every POR and/or PWDNB L→H transition.
4	Link Configuration (Single Link vs. Multilink Operation Setup)	Some deserilaizers power-up in single link mode while others in multilink modes. Read the register map for the correct registers to set up the correct link operation mode. See the *Single vs. Dual Link Operation* section for more information.
5	GMSL Link Lock is Established	If GMSL link lock is not established, verify the following: a) Voltage rails are correct per data sheet specification. b) Data rate, Coax/STP mode and GMSL settings match between serializer and deserializer.
6	I^2C Rate Adjustment (If Necessary)	SerDes has I^2C rate register settings that need to match up to I^2C main.
7	SER I^2C Device Address Reassignment (If Necessary)	Reassigning SER I^2C device address can help in multi-camera systems.
8	Disable DES CSI Output	Set register bitfield CSI_OUT_EN = 0.
9	DES Errata Settings (If Necessary)	Ensure errata settings match DEV_REV and use case.
10	DES MIPI TX Configuration	• MIPI Port Config • Lane Count • Lane Mapping/Polarity Swap • Pipe to Controller Mapping

		• Deskew (>1.5Gbps/Lane) • MIPI Data Rate
11	DES GPIO and Other Feature Configuration	• GPIO Forwarding • FSYNC • I^2C/UART Pass-through Channels • Line Fault
12	DES Interrupt Handling (ERRB) and ASIL Configuration	See the **Error Flags** section and safety documents of the deserializer for more information.
13	SER Errata Settings (If Necessary)	Ensure errata settings match DEV_REV and use case.
14	SER MIPI RX Configuration	• Lane Count • Lane Mapping/Polarity Swap • Pipe to Controller Mapping • Deskew (>1.5Gbps/Lane) • MIPI Continuous vs. Non-Continuous Mode
15	SER GPIO and Other Feature Configuration	• GPIO Forwarding • FSYNC • I^2C/UART Pass-through Channels • Reference Clock Out • ADC • Line Fault
16	SER Interrupt Handling (ERRB) and ASIL Configuration	See the **Error Flags** section and safety documents of the deserializer for more information.
17	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).
18	RESET LINK = 0	Release of reset, link will relock. Remote access is possible after link is locked.
19	DES Enable CSI Output	Set register bitfield CSI_OUT_EN = 1.
20	Enable Deserializer Register CRC Safety Mechanism (If Necessary)	Review the safety documents of the deserializer for more information. See *Register CRC* section for more information.
21	Enable Serializer Register CRC Safety Mechanism (If Necessary)	Review the safety documents of the serializer for more information. Refer to the serializer user guide **Register CRC** section for more information.
22	Start Video Source	

Notes:

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

Configuration

Overview

The forward video paths of the MAX96716A/MAX96716F deserializers are configured with the following programming:

- Pixel and Tunneling Mode
- Link Initialization
- Link Lock Check
- Video Pipe Selection
- MIPI PHY Settings
- Datatype (DT)/Virtual Channel (VC) Overrides (Including Extended Virtual Channels)

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

Pixel and Tunneling Modes

The MAX96716A/MAX(6716F support both pixel and tunneling modes. Always ensure the serializer and deserializer are both in pixel or tunneling mode.

Pixel mode provides the ability for systems to manipulate data types, bits per pixel, 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 CRC and is passed from serializer to deserializer without any manipulation. In tunneling mode, any combination of data type, bits per pixel (bpp), and virtual channel may be transmitted if the video bandwidth total does not exceed the link bandwidth.

Table 3 shows the registers to enable pixel or tunneling mode, while *Table 4* shows the differences between mode features.

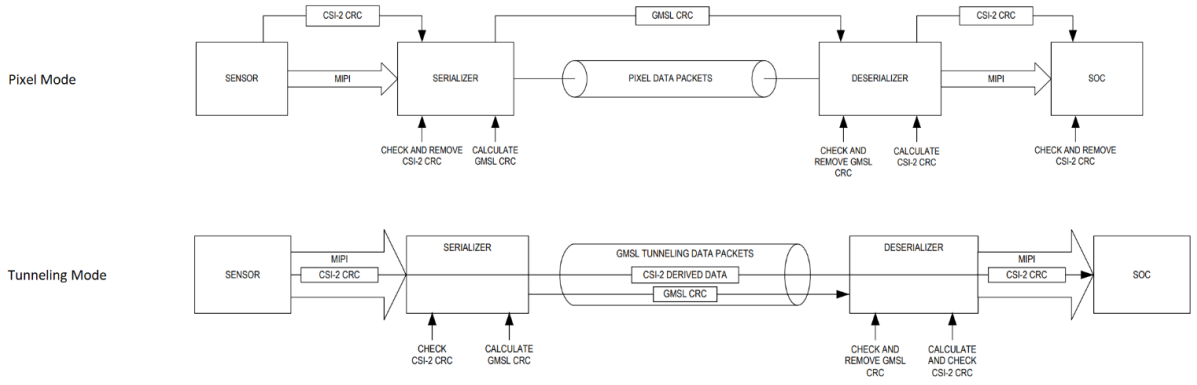

Figure 3. Pixel and Tunneling Modes Comparison

Register Settings

Table 3. MAX96716A/MAX96716F Pixel and Tunneling Mode Register Settings

Register Address	Bitfield Name	Bits	POR	Decode
0x474	TUN_EN	0	0b0	0b0: Pixel Mode 0b1: Tunneling Mode
0x4B4	TUN_EN	0	0b0	0b0: Pixel Mode 0b1: Tunneling Mode

Pixel vs. Tunneling Mode Differences

Table 4. MAX96716A/MAX96716F 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 24 BPP	Not Supported	Supported
16-Channel Virtual Channel Support	Supported	Supported

Note*: Check device data sheet to verify watermarking is supported.
Note**: 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 MAX96716A/MAX96716F device family is a GMSL2 dual-port deserializer that can support coax or shielded-twisted pair (STP) cables. The MAX96716A variant can receive 3Gbps or 6Gbps GMSL2 data in the forward direction over the GMSL link while the MAX96716F is only capable of 3Gbps GMSL2. Using the following registers, select the GMSL link rate and coax or STP cabling. Any changes to the GMSL link should be followed by a link reset to reinitialize the link (toggle RESET_LINK HIGH and then LOW). 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. *Table 5* and *Table 6* show the capabilities of the MAX96716A and MAX96716F, respectively. *Table 7* describes the link initialization registers for both devices.

Table 5. MAX96716A Basic Settings (CFG1 Pin)

CFG1 Value	Coax/STP	Data Rate	Transmission Mode
0	STP	3Gbps	Tunneling Mode
1	STP	6Gbps	Tunneling Mode

2	STP	3Gbps	Pixel Mode
3	STP	6Gbps	Pixel Mode
4	Coax	3Gbps	Tunneling Mode
5	Coax	6Gbps	Tunneling Mode
6	Coax	3Gbps	Pixel Mode
7	Coax	6Gbps	Pixel Mode

Table 6. MAX96716F Basic Settings (CFG1 Pin)

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

Table 7. Link Initialization Registers

Register Address	Bitfield Name	Bits	POR*	Decode
0x0001	RX_RATE	1:0	0b10	0b01: 3Gbps 0b10: 6Gbps
0x0004	RX_RATE_B	1:0	0b10	0b01: 3Gbps 0b10: 6Gbps
0x0011	CXTP_A	0	0b1	0b0: Shielded twisted pair drive 0b1: Coax drive
0x0011	CXTP_B	2	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
0x0013	RESET_LINK_B	0	0b0	0b0: Release link B reset 0b1: Activate link B 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 reset PHY. Link resets should not be used when video is being fed to the device. Doing this may corrupt data and have unintended consequences.
*POR value depends on CFG0 and CFG1 settings.

Single vs. Dual Link Operation

The MAX96716A/MAX96716F have AUTO_LINK = 1 enabled by default. This means that whichever link powers up first (GMSL link A or link B), it 'automatically' establishes link lock and has control channel access (I^2C or

UART). To access both GMSL links for dual link operation, the MAX96716A/MAX96716F must be placed in reverse splitter mode. See *Table 8* for register settings required.

Table 8. MAX96716A/MAX96716F Single vs. Dual Link Operation

Link Access	AUTO_LINK (Reg 0x10)	LINK_CFG (Reg 0x10)	LINK_EN_A/B (Reg 0xF00)
Link A Only	0x1	0x01 or 0x10	0x01
Link B Only	0x1	0x01 or 0x10	0x10
Links A and B (Reverse Splitter Mode)	0x1	0x11	0x11

Link Lock Check

If the device configuration is correct, the link automatically locks upon connection. Pin #3 (MFP1) is used as LOCK indication by default. Bit 3 in register 0x0013 asserts if link A is locked. Bit 3 in register 0x5009 asserts if link B is locked.

Video Pipe Selection

The video pipe must be configured to match the video streams received from the connected serializer. This programming step, typically performed following link initialization, ensures that the GMSL2 CSI-2 deserializer properly receives video data from the serializers. There are two video pipes (Y and Z) that can select one of four streams from the serializer. By default, the deserializer is programmed to accept the most common stream from the CSI serializers (stream 0b10) for both GMSL links A and B, and configuration is not usually needed.

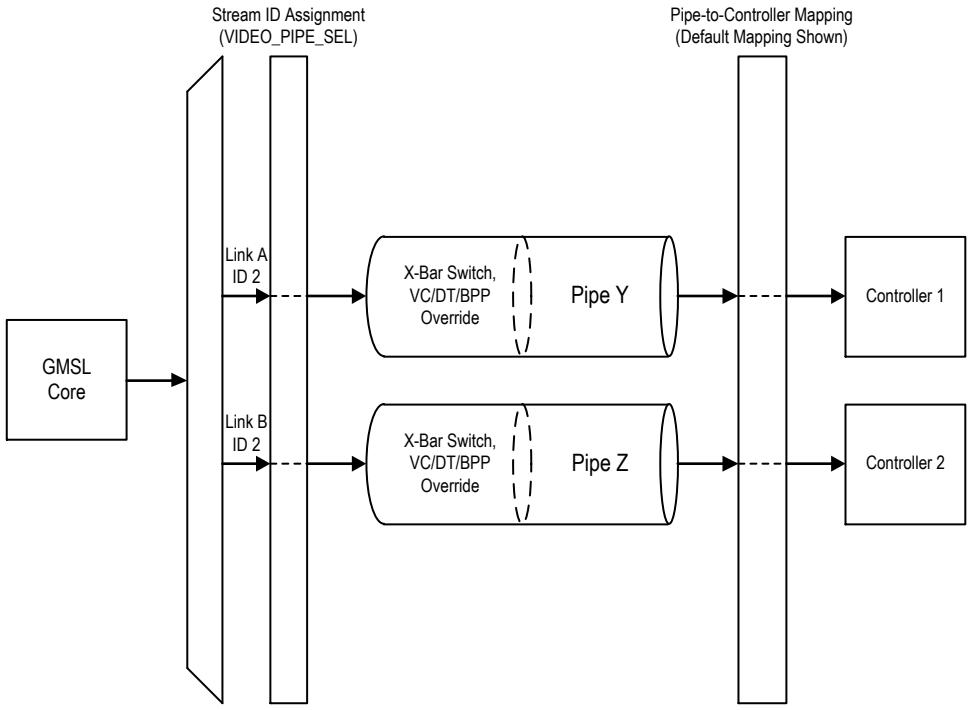

Figure 4. MAX96716A/MAX96716F Video Pipe Path

Video Pipe Selection Register

Select the serializer stream ID to match the video streams (STR_ID) from the serializers for each deserializer video pipe. Most GMSL2 camera serializers have 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 deserializer selects stream 0b10, which is the default for most CSI serializers.

Table 9. MAX96716A/F Video Pipe Selection

Register Address	Bitfield Name	Bits	POR	Decode
0x160	VIDEO_PIPE_EN (Pipe Y)	0	0b1	0b0: Video Pipe Disabled 0b1: Video Pipe Enabled
0x160	VIDEO_PIPE_EN (Pipe Z)	1	0b1	0b0: Video Pipe Disabled 0b1: Video Pipe Enabled
0x161	VIDEO_PIPE_SEL_Y	2:0	0b010	0b000: GMSL Link A, str_id=00 0b001: GMSL Link A, str_id=01 0b010: GMSL Link A, str_id=10 0b011: GMSL Link A, str_id=11 0b100: GMSL Link B, str_id=00 0b101: GMSL Link B, str_id=01 0b110: GMSL Link B, str_id=10 0b111: GMSL Link B, str_id=11
0x161	VIDEO_PIPE_SEL_Z	5:3	0b110	0b000: GMSL Link A, str_id=00 0b001: GMSL Link A, str_id=01 0b010: GMSL Link A, str_id=10 0b011: GMSL Link A, str_id=11 0b100: GMSL Link B, str_id=00 0b101: GMSL Link B, str_id=01 0b110: GMSL Link B, str_id=10 0b111: GMSL Link B, str_id=11

Video Lock Check

Register 0x01FC bit 0 (VIDEO_LOCK) asserts if it is receiving valid video data on pipe Y. Register 0x21C bit 0 (VIDEO_LOCK) asserts if it is receiving valid video data on pipe Z.

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

Video pipe to mobile industry processor interface (MIPI) controller mapping can be configured to send video pipe data from one GMSL link to a specific MIPI controller and MIPI PHY output of the deserializer.

In addition, MIPI controller mapping can also be used to change the VC mapping of individual MIPI data types and aggregate data to one output.

Changing VC/DT Source Destination Registers (Pixel Mode Only)

Video data is mapped from the video pipes to the two CSI-2 Tx controllers. Each video pipe has dedicated register address space for pipe-to-controller mapping. This is used to aggregate data from multiple video pipes

to a single CSI-2 output port. A video pipe can only be routed to a single controller. The configuration registers are contained in *Table 10*.

The pipe-to-controller mapping registers can be used to overwrite any previously assigned virtual channel to data within a video pipe. This is done by setting the 2 MSBs in the MAP_SRC register to the existing virtual channel and then setting the 2 MSBs in the MAP_DST register to the target virtual channel.

Pipe-to-controller register space:

- Pipe Y: MIPI_TX 1
- Pipe Z: MIPI_TX 2

Default Mapping:

- Pipe Y → Controller 1
- Pipe Z → Controller 2

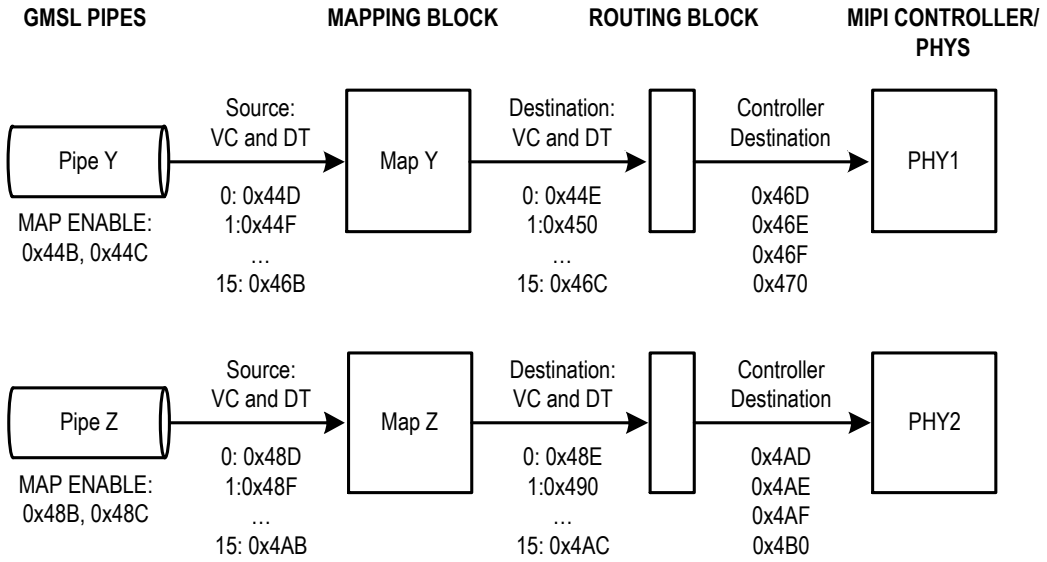

Figure 5. Video Pipe to MIPI Controller Mapping Block Diagram

Table 10. Video Pipe to MIPI PHY Controller Register Table

Register Name	Bitfield Name	Bits	POR	Decode
MIPI_TX11	MAP_EN_L[7:0]	7:0	0x00	Mapping enables low byte [7:0]. Each bit enables 1 of 8 mapping and distribution entries (defined in MAP_SRC_x, MAP_DST_x and MAP_DPHY_DST_x) for the current video stream. Non-matched virtual channel (VC) and data types (DT) pass to the corresponding CSI2 controller. 0bxxxxxxx1: Enable MAP_SRC_0 and MAP_DST_0 registers. … 0b1xxxxxxx: Enable MAP_SRC_7 and MAP_DST_7 registers.
MIPI_TX12	MAP_EN_H[7:0]	7:0	0x00	Mapping enables high byte [15:8]. Each bit enables 1 of 8 mapping and distribution entries (defined in MAP_SRC_x, MAP_DST_x and MAP_DPHY_DST_x) for the current video stream. Non-matched virtual channel (VC) and data types (DT) pass to the corresponding CSI2 controller. 0bxxxxxxx1: Enable MAP_SRC_8 and MAP_DST_8 registers. … 0b1xxxxxxx: Enable MAP_SRC_15 and MAP_DST_15 registers.
MIPI_TX13-43 (Odd Registers)	MAP_SRC_{0-15}	7:0	0x00	Source DT and VC (present in the corresponding pipe) that are mapped to the corresponding destination DT and VC and routed to the PHY configured in MIPI_TX45-48. Bit [7:6]: Virtual Channel Bit [5:0]: Data type
MIPI_TX14-44 (Even Registers)	MAP_DST_{0-15}	7:0	0x00	Destination DT and VC that are routed to the PHY configured in MIPI_TX45-48. Bit [7:6]: Virtual Channel Bit [5:0]: Data type
MIPI_TX45	MAP_DPHY_DEST_{0-3}[1:0]	7:0	0x00	CSI2 Controller Destination for the corresponding MAP_SRC_x and MAP_DST_x registers. Bits [7:6] = Destination controller for Map 3 Bits [5:4] = Destination controller for Map 2 Bits [3:2] = Destination controller for Map 1 Bits [1:0] = Destination controller for Map 0 0b00: Not supported.

				0b01: Map to MIPI PHY1 0b10: Map to MIPI PHY2 0b11: Not supported.
MIPI_TX46	MAP_DPHY_DEST_{4-7}[1:0]	7:0	0x00	CSI2 Controller Destination for the corresponding MAP_SRC_x and MAP_DST_x registers. Bits [7:6] = Destination controller for Map 7 Bits [5:4] = Destination controller for Map 6 Bits [3:2] = Destination controller for Map 5 Bits [1:0] = Destination controller for Map 4 0b00: Not supported. 0b01: Map to MIPI PHY1 0b10: Map to MIPI PHY2 0b11: Not supported.
MIPI_TX47	MAP_DPHY_DEST_{8-11}[1:0]	7:0	0x00	CSI2 Controller Destination for the corresponding MAP_SRC_x and MAP_DST_x registers. Bits [7:6] = Destination controller for Map 11 Bits [5:4] = Destination controller for Map 10 Bits [3:2] = Destination controller for Map 9 Bits [1:0] = Destination controller for Map 8 0b00: Not supported. 0b01: Map to MIPI PHY1 0b10: Map to MIPI PHY2 0b11: Not supported.
MIPI_TX48	MAP_DPHY_DEST_{12-15}[1:0]	7:0	0x00	CSI2 Controller Destination for the corresponding MAP_SRC_x and MAP_DST_x registers. Bits [7:6] = Destination controller for Map 15 Bits [5:4] = Destination controller for Map 14 Bits [3:2] = Destination controller for Map 13 Bits [1:0] = Destination controller for Map 12 0b00: Not supported. 0b01: Map to MIPI PHY1 0b10: Map to MIPI PHY2 0b11: Not supported.

Example Pipe to Controller Routing

This example takes RAW12 + EMB8 from VCO on Pipe Y and outputs it on DES MIPI Port A. To maximize the GMSL bandwidth, the EMB8 bpp is doubled from 8bpp to 16bpp and the RAW12 is zero padded to become 16bpp.

```
#Serializer settings, I2C address=0x80
0x04,0x80,0x03,0x12,0x04, // bpp8dblz: Send 8-bit pixels as 16-bit (for EMB8)
0x04,0x80,0x03,0x1E,0x2C, // Set minimum bpp for Pipe Z to 12 (for RAW12) and soft_bppz_en=1
0x04,0x80,0x01,0x11,0x50, // Set maximum bpp for Pipe Z to 16
0x04,0x80,0x01,0x10,0x60, // Disable auto bpp for Pipe Z

#Deserializer settings, I2C address=0x98
// un-double 8 bit data type
0x04,0x98,0x04,0x73,0x10, // ALT2_MEM_MAP8 CTRL1: Alternate memory map enabled
0x04,0x98,0x04,0xB3,0x10, // ALT2_MEM_MAP8 CTRL1: Alternate memory map enabled

0x04,0x98,0x04,0x4B,0x0F, // Enable 4 mappings for Pipe Y
0x04,0x98,0x04,0x4D,0x2C, // Map RAW12, VC0
0x04,0x98,0x04,0x4E,0x2C,
0x04,0x98,0x04,0x4F,0x00, // Map frame start, VC0
0x04,0x98,0x04,0x50,0x00,
0x04,0x98,0x04,0x51,0x01, // Map frame end, VC0
0x04,0x98,0x04,0x52,0x01,
0x04,0x98,0x04,0x53,0x12, // Map EMB8, VC0
0x04,0x98,0x04,0x54,0x12,
0x04,0x98,0x04,0x6D,0x55,// All mappings to PHY1 (MIPI Port A)
```

MIPI PHY Settings

The MIPI PHY settings contain programming options for output data rate, number of lanes, and port selection. The MAX96716A/MAX96716F deserializers have four 2-lane capable MIPI PHYs (PHY 0, 1, 2, 3) controlled by two MIPI controllers (Ctrl 1 and 2) to establish up to dual 4-lane output ports. *Table 11* contains the MIPI PHY settings registers.

The 1x1, 1x2, 1x3, and 1x4 configurations per port use lane count settings. See *Table 11* for register write details.

Table 11. MIPI PHY Setting Registers

Register Address	Bitfield Name	Bits	POR	Decode
0x330	phy_1x4	1	0b0	0b0: 1x4 configuration not selected 0b1: MIPI output configured in one 4-Lane MIPI Ports
0x330	phy_2x4	2	0b1	0b0: 2x4 configuration not selected 0b1: MIPI output configured in two 4-Lane MIPI Ports
0x332	phy_stdby_n	7:4	0b1111	0bxxx0: Put PHY0 in standby mode. 0bxx0x: Put PHY1 in standby mode. 0bx0xx: Put PHY2 in standby mode. 0b0xxx: Put PHY3 in standby mode.
0x333	phy0_lane_map	3:0	0b0100	MIPI PHY0 to Port A lane mapping in 2x4 Bits[1:0] map PHY0 data lane 0 Bits[3:2] map PHY0 data lane 1 Decode: 0bXX00Map PHY0 D0 to Port A D0 0bXX01Map PHY0 D0 to Port A D1 0bXX10Map PHY0 D0 to Port A D2 0bXX11Map PHY0 D0 to Port A D3 0b00XXMap PHY0 D1 to Port A D0 0b01XXMap PHY0 D1 to Port A D1 0b10XXMap PHY0 D1 to Port A D2 0b11XXMap PHY0 D1 to Port A D3
0x333	phy1_lane_map	7:4	0b0100	MIPI PHY1 to Port A lane mapping in 2x4 Bits[1:0] map PHY1 data lane 0 Bits[3:2] map PHY1 data lane 1 Decode: 0bXX00Map PHY1 D0 to Port A D0 0bXX01Map PHY1 D0 to Port A D1 0bXX10Map PHY1 D0 to Port A D2 0bXX11Map PHY1 D0 to Port A D3

				0b00XXMap PHY1 D1 to Port A D0 0b01XXMap PHY1 D1 to Port A D1 0b10XXMap PHY1 D1 to Port A D2 0b11XXMap PHY1 D1 to Port A D3
0x334	phy2_ lane_map	3:0	0b0100	MIPI PHY2 to Port B Lane mapping in 2x4 Bits[1:0] map PHY2 data lane 0 Bits[3:2] map PHY2 data lane 1 Decode: 0bXX00Map PHY2 D0 to Port B D0 0bXX01Map PHY2 D0 to Port B D1 0bXX10Map PHY2 D0 to Port B D2 0bXX11Map PHY2 D0 to Port B D3 0b00XXMap PHY2 D1 to Port B D0 0b01XXMap PHY2 D1 to Port B D1 0b10XXMap PHY2 D1 to Port B D2 0b11XXMap PHY2 D1 to Port B D3
0x334	phy3_lane_map	7:4	0b0100	MIPI PHY3 to Port B Lane mapping in 2x4 Bits[1:0] map PHY3 data lane 0 Bits[3:2] map PHY3 data lane 1 Decode: 0bXX00Map PHY3 D0 to Port B D0 0bXX01Map PHY3 D0 to Port B D1 0bXX10Map PHY3 D0 to Port B D2 0bXX11Map PHY3 D0 to Port B D3 0b00XXMap PHY3 D1 to Port B D0 0b01XXMap PHY3 D1 to Port B D1 0b10XXMap PHY3 D1 to Port B D2 0b11XXMap PHY3 D1 to Port B D3
0x335	phy0_pol_map	2:0	0b000	0bxx0: Normal polarity for Lane 0 0bxx1: Inverse polarity for Lane 0 0bx0x: Normal polarity for Lane 0 0bx1x: Inverse polarity for Lane 1 0b0xx: Normal polarity for Clock Lane 0b1xx: Inverse polarity for Clock Lane
0x335	phy1_pol_map	5:3	0b000	0bxx0: Normal polarity for Lane 0 0bxx1: Inverse polarity for Lane 0 0bx0x: Normal polarity for Lane 0 0bx1x: Inverse polarity for Lane 1 0b0xx: Normal polarity for Clock Lane 0b1xx: Inverse polarity for Clock Lane
0x336	phy2_pol_map	2:0	0b000	0bxx0: Normal polarity for Lane 0

				0bxx1: Inverse polarity for Lane 0 0bx0x: Normal polarity for Lane 0 0bx1x: Inverse polarity for Lane 1 0b0xx: Normal polarity for Clock Lane 0b1xx: Inverse polarity for Clock Lane
0x336	phy3_pol_map	5:3	0b000	0bxx0: Normal polarity for Lane 0 0bxx1: Inverse polarity for Lane 0 0bx0x: Normal polarity for Lane 0 0bx1x: Inverse polarity for Lane 1 0b0xx: Normal polarity for Clock Lane 0b1xx: Inverse polarity for Clock Lane
0x33F	RST_MIPITX_LOC	2:1	0b00	Bit 1 for controller 1 Bit 2 for controller 2 0b0: Do not reset 0b1: Reset the MIPI Controller
0x44A	CSI2_LANE_CNT	7:6	0b11	0b00: One data lane 0b01: Two data lanes 0b10: Three data lanes 0b11: Four data lanes
0x48A	CSI2_LANE_CNT	7:6	0b11	0b00: One data lane 0b01: Two data lanes 0b10: Three data lanes 0b11: Four data lanes
0x320	phy1_csi_tx_dpll_predef_freq	4:0	0x01111	Set DPLL frequency on multiples of 100Mbps 0b00001 = 100 Mbps/lane data rate … 0b11001 = 2500Mbps/Lane data rate
0x323	phy2_csi_tx_dpll_predef_freq	4:0	0x01111	Set DPLL frequency on multiples of 100Mbps 0b00001 = 100 Mbps/lane data rate … 0b11001 = 2500Mbps/Lane data rate

MIPI Data Lane and Polarity Swap

Lane swapping is available for pins on the same port. See *Table 11* for relevant registers.

The data pins can be swapped within each port, but the clock location is fixed. For example, in the 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 pairs, and the clock lane supports polarity inversion. *Figure 6* shows the default lane mapping that matches device pinout. *Figure 7* demonstrates the polarity swap example.

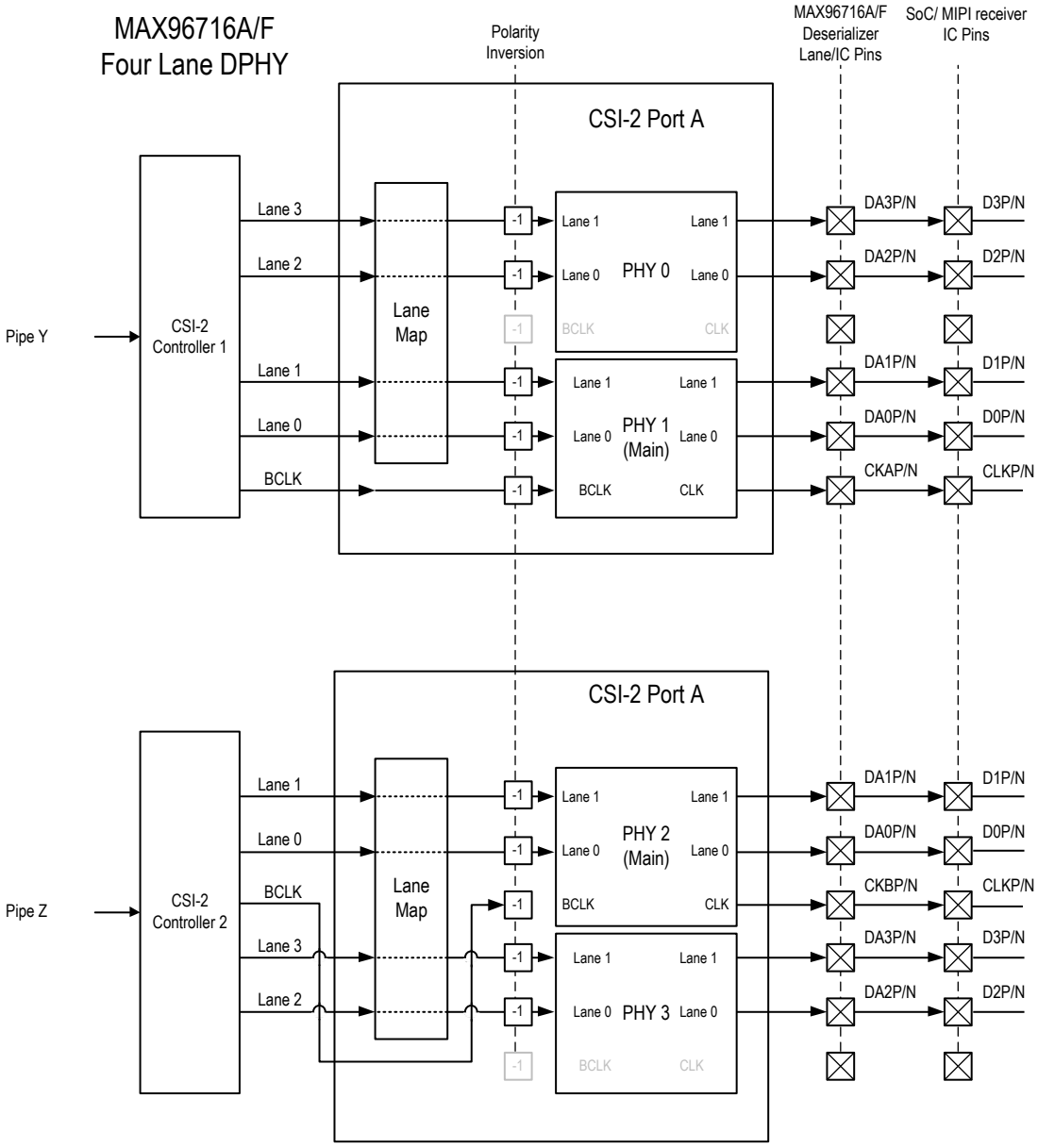

Figure 6. MAX96716A/MAX96716F Default 2x4 DPHY Lane Mapping

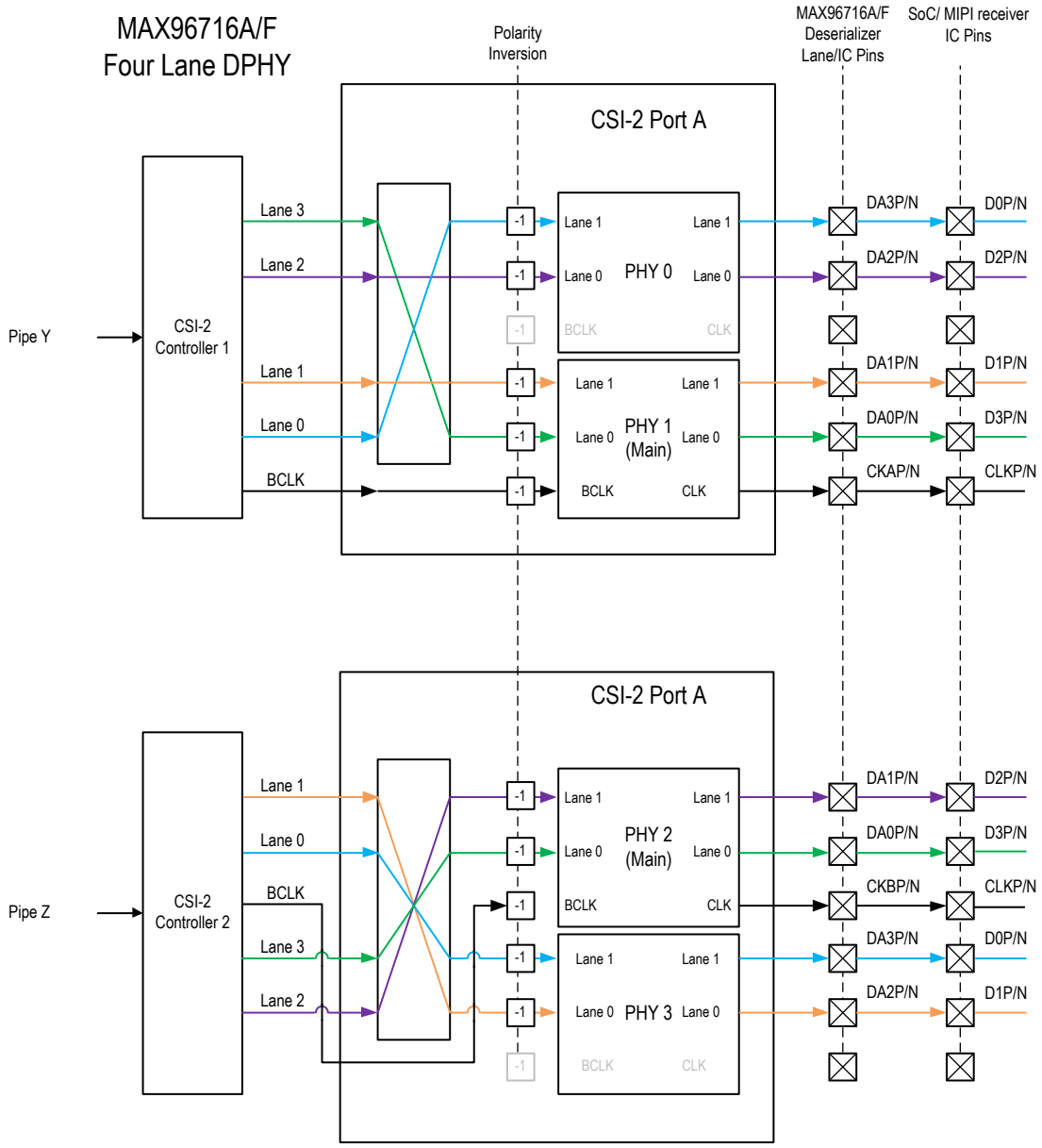

Figure 7. MAX96716A/MAX96716F 2x4 DPHY Lane Swap Example

Lane Swap Programming Example

Figure 7 lane swap example register writes are as following. Register 0x333 is for MIPI Port A and 0x334 is for MIPI Port B.

```
#DES I2C Address=0x98
0x04,0x98,0x03,0x33,0x72, // (Default) (Lane Map - PHY0 D0): Lane 2 | (Lane Map - PHY0 D1): Lane 0 | (Lane Map - PHY1 D0): Lane 3 | (Default)  (Lane Map - PHY1 D1): Lane 1
0x04,0x98,0x03,0x34,0x1B, // (Lane Map - PHY2 D0): Lane 3 | (Lane Map - PHY2 D1): Lane 2 |  (Lane Map - PHY3 D0): Lane 1 |  (Lane Map - PHY3 D1): Lane 0
```

MIPI D-PHY Deskew Settings

The D-PHY deskew mechanism is only relevant to lane speeds greater than 1.5Gbps per lane. Periodic deskew requires a continuous clock and the clock lane always in the high-speed mode.

The initial transmit deskew is set with DESKEW_INIT[7:0] in register 0x0443. All settings must be configured before video streams are received. After video lock, the MIPI Tx clock lane starts automatically and an automatic deskew pattern is generated before the first HS data transmission. For system flexibility, an additional initial deskew pattern can be inserted by changing DESKEW_INIT bit 5 any time after the clock lane is enabled. Note that DESKEW_INIT bit 4 must be set to trigger a manual deskew.

The periodic deskew is initiated by setting bit 7 of DESKEW_PER[7:0] in register 0x0444. The period interval has a programmable range from every 1 frame to 128 frames. *Table 12* contains the relevant MIPI D-PHY deskew registers.

Table 12. MAX96716A/MAX96716F MIPI D-PHY Deskew Registers

Register Address	Bitfield Name	Bits	POR	Decode
0x443	DESKEW_INT		0b00000001	Bit [7]: Sets auto-initial deskew calibration on or off 0b0=Auto initial deskew off 0b1=Auto initial deskew on Bit [6]: RSVD Bit [5]: Any bit change initiates an initial calibration if "Bit [4]=1" Bit[4] Selects manual initial on or off Bit [3} RSVD Bits [2:0] Selects initial deskew width 0b000=RSVD 0b001= 2 x 32k UI -(t_lpx+t_hs_przero)[UI] - 16 UI 0b010= 3 x 32k UI -(t_lpx+t_hs_przero)[UI] - 16 UI 0b011= 4 x 32k UI -(t_lpx+t_hs_przero)[UI] - 16 UI 0b100= 5 x 32k UI -(t_lpx+t_hs_przero)[UI] - 16 UI 0b101= 6 x 32k UI -(t_lpx+t_hs_przero)[UI] - 16 UI 0b110= 7 x 32k UI -(t_lpx+t_hs_przero)[UI] - 16 UI 0b111= 8 x 32k UI -(t_lpx+t_hs_przero)[UI] - 16 UI
0x444	DESKEW_PER	7:0	0b00000001	Bit [7]: Enable periodic deskew calibration 0b0=Deskew off 0b1=Deskew enabled Bit [6]: Selects generation on rising or falling edge of VS 0b0=Falling Edge 0b1=Rising Edge Bits [5:3]: Selects periodic interval 0b001= Every frame

				0b010= Every 2 frames 0b011= Every 4 frames 0b100= Every 8 frames 0b101= Every 16 frames 0b110= Every 32 frames 0b111= Every 64 frames 0b000= Every 128 frames Bits [2:0] Select initial deskew width 0b000=RSVD 0b001= 2 x 32k UI -(t_lpx+t_hs_przero)[UI] - 16 UI 0b010= 3 x 32k UI -(t_lpx+t_hs_przero)[UI] - 16 UI 0b011= 4 x 32k UI -(t_lpx+t_hs_przero)[UI] - 16 UI 0b100= 5 x 32k UI -(t_lpx+t_hs_przero)[UI] - 16 UI 0b101= 6 x 32k UI -(t_lpx+t_hs_przero)[UI] - 16 UI 0b110= 7 x 32k UI -(t_lpx+t_hs_przero)[UI] - 16 UI 0b111= 8 x 32k UI -(t_lpx+t_hs_przero)[UI] - 16 UI

Deskew Register Example

```
#DES I2C Address=0x98
# Enable initial deskew packets with the minimum width 32K UI on controller 1 for port A
0x98, 0x443, 0x80
# Enable periodic deskew packets with width 2K UI every 2 frames
0x98, 0x444, 0x91
```

Extended Virtual Channels

GMSL2 deserializers can use CSI-2 extended virtual channels to accommodate larger serial link systems. Virtual channels allow the serial link system to differentiate video inputs by the virtual channel assigned to them. When extended virtual channels are enabled, the standard 2-bit virtual channel selection is extended to 4-bit (D-PHY), increasing the number of available virtual channels to 16 for D-PHY applications. The increase in available virtual channels allows systems to support more camera inputs.

Extended virtual channels are enabled individually for each controller (1 or 2) on the deserializer. Any video pipe routed to a controller with virtual channel extension enabled uses the extended virtual channels. *Table 13* describes the registers for virtual channel extension.

Table 13. Extended Virtual Channels Registers

Register Address	Bits	Default Value	Decode
0x044A/0x048A	3	0x00	**CSI_VCX_EN:** 0 = VC extension disabled 1 = VC extension disabled Controller 1 and 2 virtual channel enable
0x510	7:2	0x00	**Pipe Y, Extended VC MAP_SRC/DST_0 Register:** Bits [7:5]: SRC Virtual Channel

			Bits [4:2]: DST Virtual Channel
0x51F	7:2	0x00	**Pipe Y, Extended VC MAP_SRC/DST_15 Register:** Bits [7:5]: SRC Virtual Channel Bits [4:2]: DST Virtual Channel
0x520	7:2	0x00	**Pipe Z, Extended VC MAP_SRC/DST_0 Register:** Bits [7:5]: SRC Virtual Channel Bits [4:2]: DST Virtual Channel
0x52F	7:2	0x00	**Pipe Z, Extended VC MAP_SRC/DST_15 Register:** Bits [7:5]: SRC Virtual Channel Bits [4:2]: DST Virtual Channel

Tunneling Mode

Tunnel mode does not support overriding of the virtual channel; however, it supports virtual channel extension. If the incoming video source is using extended VCs, VCX needs to be enabled on the 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
```

Pixel Mode

Pixel mode supports virtual channel extension as well as overriding virtual channel 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 channels and VC override registers details.

Pixel Mode Programming Example

This example routes video pipe Z to controller 1 and changes the source virtual channel (VC=0) to the extended virtual channel (VC = 5). DES I2C address=0x98.

```
// Pipe Z mapping to controller 1
0x04,0x98,0x04,0x8B,0x07, // Enable three mappings
0x04,0x98,0x04,0xAD,0x15, // Send all mapped data to Controller 1
0x04,0x98,0x04,0x4A,0xD8, // Enable VC extension on Controller 1

0x04,0x98,0x05,0x20,0x04, // Mapping 0 source data, extended VC = 0->5
0x04,0x98,0x04,0x8D,0x2C, // Mapping 0 source data. RAW12 pixels with VC = 0->5
0x04,0x98,0x04,0x8E,0x6C, // Mapping 0 destination data. RAW12 pixels with VC = 0->5

0x04,0x98,0x05,0x21,0x04, // Mapping 1 source data, extended VC = 0->5
0x04,0x98,0x04,0x8F,0x00, // Mapping 1 source data. RAW12 Frame start with VC = 0->5
0x04,0x98,0x04,0x90,0x40, // Mapping 1 destination data.RAW12 Frame start with VC = 0->5

0x04,0x98,0x05,0x22,0x04, // Mapping 2 source data, extended VC = 0->5
```

```
0x04,0x98,0x04,0x91,0x01, // Mapping 2 source data. RAW12 Frame end with VC = 0->5
0x04,0x98,0x04,0x92,0x41, // Mapping 2 destination data.RAW12 Frame end with VC = 0->5
```

Software Override

The software override manually overrides the video data type (DT) (that is, packet header), virtual channel (VC) number, or bits per pixel (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 (example, GMSL1 serializers), it is necessary to specify the desired DT, VC, and bpp with the software override. See *Table 14* for a list of software override registers.

The data type, virtual channel, and bpp overrides must be enabled to take effect. Enable override_bpp_vc_dt_y/z bit for video streams that require the override of the VC, DT, and bpp.

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

Table 14. Software Override Register Table

Register Address	Bitfield Name	Bits	POR	Decode
0x314	soft_vc_y	7:4	0b0000	Pipe Y Virtual Channel Override 0b0000=VC0 Assignment 0b1111=VC15 Assignment
0x315	soft_vc_z	3:0	0b0000	Pipe Z Virtual Channel Override 0b0000=VC0 Assignment 0b1111=VC15 Assignment
0x316	soft_dt_y_h	7:6	0b00	Pipe Y Data Type (DT) Override High Bits
0x317	soft_dt_y_l	3:0	0b0000	Pipe Y Data Type (DT) Override Low Bits
0x317	soft_dt_z_h	7:4	0b0000	Pipe Z Data Type (DT) Override High Bits
0x318	soft_dt_z_l	1:0	0b00	Pipe Z Data Type (DT) Override Low Bits
0x319	soft_bpp_y	4:0	0b00000	Pipe Y Software Defined bpp
0x319	soft_bpp_z_h	7:5	0b000	Pipe Z Software Defined bpp High Bits
0x31A	soft_bpp_z_l	1:0	0b00	Pipe Z Software Defined bpp Low Bits
0x31D	override_bpp_vc_dty	7	0b0	0b0=Disable software override for BPP, VC, & DT 0b1=Enable software override for BPP, VC & DT
0x320	override_bpp_vc_dtz	6	0b0	0b0=Disable software override for BPP, VC & DT 0b1=Enable software override for BPP, VC & DT

Software Override Programming Example

The following script performs a series of VC, DT, and bpp software overrides on video pipe Y.

```
#DES I2C Address=0x98
# VC/DT/BPP Software override for Pipe Y.
# VC 0 override for pipe Y
0x98,0x0314,0x00
# RAW12 (0x2C) DT override for pipe Y
0x98,0x0316,0x80
0x98,0x0317,0x0C
# 12 BPP override for pipe Y
0x98,0x0319,0x0C
# Enable the software override for pipe Y
0x98,0x031D,0x80
```

I²C Control Channels

Overview

The MAX96716A/MAX(6716F provide one primary I²C channel and two pass-through I²C channels.

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

Primary I²C Control Channel

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

Pass-Through I²C Channel

There are two pass-through I²C channels 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.

Port Access and Routing

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

Table 15. MFP Pins for I²C

MFP Pin	Default Function	I²C Function #1	I²C Function #2	Notes
MFP0	GPIO0	SDA1_RX1	SDA2_RX2	Enable I²C pass-through through register
MFP1	LOCK	SCL1_TX1	SCL2_TX2	Enable I²C pass-through through register
MFP5	RX1	SDA2_RX2	SDA1_RX1	Enable I²C pass-through through register
MFP6	TX1	SCL2_TX2	SCL1_TX1	Enable I²C pass-through through register
MFP11	SDA_RX	N/A	N/A	I²C or UART function selected through CFG0 pin

MFP12	SCL_TX	N/A	N/A	I²C or UART function selected through CFG0 pin

On power-up, the device should be set to I²C mode through the CFG0 latch. The function names in *Table 15* and ensuing I²C sections assume the device is configured for I²C mode.

By default, the primary I²C control channel lines are brought out on MFP11 and MFP12 for SDA and SCL, respectively. One can disable the primary control channel's line access by setting field DIS_LOCAL_CC in register 0x1. One can also disable access to remote device control by setting field DIS_REM_CC in register 0x1.
On power-up, the UART pass-through channel-1 is enabled on MFP5 (RX1) and MFP6 (TX1) as UART_PT_SWAP is ON by default.

CRC for I²C and Message Counter Transactions

The MAX96716A/MAX96716F 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 SER/DES 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 error is detected, the command is not executed, and the transaction is reported as NACK. The 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 message counter feature is enabled by setting fields CC_MSGCNTR_EN and CC_CRC_MSGCNTR_OVR in register 0x4.

Enable CRC and Message Counter on Primary I²C Control Channel

The following are the register writes to enable CRC for I²C and Message counter for primary I²C control channel. (Only works with I²C host controller that supports CRC/Message counter).

```
#DES I2C Address=0x98
#Enable CRC and Message Counter on DES
0x98,0x300F,0x07
```

I^2C Registers

Table 16 has registers needed to enable/disable primary and pass-through I^2C channels. *Table 16* also shows how to enable CRC for I^2C and Message Counter features.

Table 16. I^2C Registers

Register Address	Bits	POR	Decode
0x01	7:4	0x00	Bit [7]: Enable pass-through I^2C Control Channel 2 (SDA2, SCL2) Bit [6]: Enable pass-through I^2C Control Channel 1 (SDA1, SCL1) Bit [5]: Disable primary I^2C Control Channel connection to SDA and SCL pins Bit [4]: Disable access to remote device control-channel over GMSL2 Link A connection
0x03	6	0x09	Bit [6] Enable swap of pass-through 1 assignment for pass-through 2 assignment
0x03	2	0b0	Bit [2] Disable access to remote device control-channel over GMSL2 Link B connection
0x04	5:4	0b00	Bit [5] Disable pass-through channel 1 on GMSL2 Link B Bit [4] Disable pass-through channel 1 on GMSL2 Link A
0x05	4	0b0	Bit [4] Disable pass-through channel 2 on GMSL2 Link A
0x06	5:4	0b0	Bit [5] Disable pass-through channel 2 on GMSL2 Link B Bit [4] Enables I^2C when set to a 1 or UART when set to a 0***
0x300F	4:0	0x06	Bit [4:3] Message Counter Enable 0b00: Primary I^2C/UART Control Channel 0b01: Pass-through I^2C/UART Channel 1 0b10: Pass-through I^2C/UART Channel 2 0b11: RSVD Bit [2]: Enable I^2C/UART message counter* Bit [1]: Enable I^2C/UART CRC packeting* Bit [0]: Enable manual override of I^2C CRC or message counter configuration**
0x3008	0	0x00	Bit [0] Reset Message Counter Value
0x3009	1:0	0b00	Bit [1] Rest Reset Message Counter Error Count to 0 Bit [0] Reset CRC Error Count to 0
0x300A	7:0	0b00000000	Bits [7:0] Read I^2C/UART CRC Value
0x300B	7:0	0b00000000	Bits [7:0] Read I^2C/UART Message Counter Low Bits
0x300C	7:0	0b00000000	Bits [7:0] Read I^2C/UART Message Counter High Bits
0x300D	7:0	0b00000000	Bits [7:0] Read number of I^2C/UART CRC errors
0x300E	7:0	0b00000000	Bits [7:0] Read number I^2C/UART Message Counter errors

Note*: Only active when Bit [2] is also set to 1.
Note:** If set to a 0, then CRC and message counter features are disabled.
Note*:** This bit is set according to the CFG0 pin value on power-up. Writing to this register is not recommended.

Enabling I²C Pass-Through Channels

The user can bring out the first pass-through I²C channel (SDA1_RX1/SCL1_TX1) on MFP0/MFP1 or MFP5/6. The second pass-through I²C channel (SDA2_RX2/SCL2_TX2) can also be programmed on MFP0/MFP1 or MFP5/6. Pass-through channels are enabled by setting the fields IIC_1_EN and IIC_2_EN in register 0x1 and use bitfield UART_PT_SWAP in register 0x03 for enabling MFP0/1 or MFP5/6. When enabling a I²C pass-through channel, other MFP functions must be disabled first.

Control Channel Programming Example

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

```
#DES I2C Address=0x98
#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 at the same time. When broadcasting, if any remote GMSL I²C port ACKs the packet, it ACKs for all remote GMSL I²C ports.

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 acknowledgement (ACK) to the next transaction is required.

An example of I²C broadcasting is discussed in the ensuing section. Two equivalent camera modules, including an image sensor and GMSL2 serializer with the same respective addresses, are connected to two GMSL2 deserializers with different device addresses. Each of the camera modules comprise a serializer at the default I²C address 0x80 and an image sensor at address 0x20.

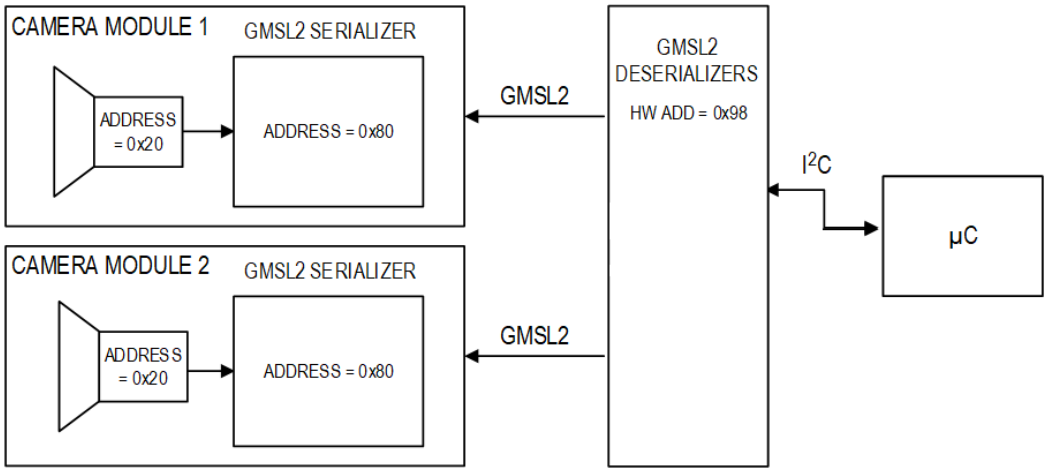

Figure 8. I²C Interfaced Camera-Module System with Default Address Settings

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.

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 acknowledgement (ACK) to the next transaction is required.

I²C Broadcasting GMSL2 Use Case Example

The procedure for the I²C broadcasting example is as follows.

1) Isolate camera module 1 by disabling camera module 2's GMSL link (RESET_LINK = 1).
2) 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.
3) 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.
4) 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.

5) 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).
6) 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.
7) 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.
8) 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.
9) Now enable all the links for remote primary I^2C port access.
10) All devices should be present on the I^2C bus. Continue with any additional required system configuration.

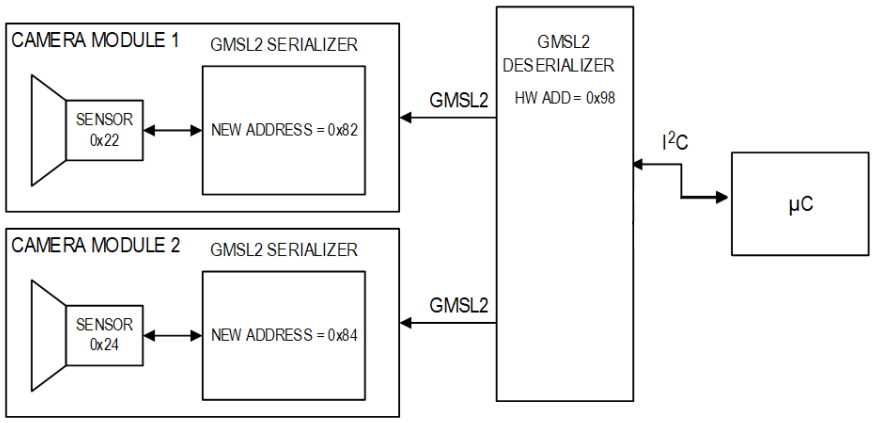

Figure 9. Two Camera-Module System with Translated Address Settings

Table 17. I^2C Broadcasting Example (Serializer)

Old I^2C Address	New I^2C 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

Table 17 shows how the serializers are assigned a single device address to allow writes to all devices as a broadcast. This allows the I^2C host controller to broadcast with address 0xC4.

Table 18. I^2C Broadcasting Example (Image Sensor)

Old I^2C Address	New I^2C 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 the 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 MAX96716A/MAX96716F provides one primary UART channel and two pass-through UART channels.

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 acknowledgement (ACK) to the next transaction is required.

Primary UART Control Channel

The primary UART control channel is used to provide access to both the 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).

Base Mode

Base mode allows the device registers of both the serializer and 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 GMSL2 link. This mode prevents inadvertent programming of the serializer/deserializer registers and can be switched in and out of during normal operation.

Pass-Through UART Control Channel

There are two pass-through UART channels to send data across the GMSL2 link. Serializer/deserializer registers are not accessible in this mode but any other peripherals on the link with compatible UART protocol are accessible. This makes either of the pass-through UART channels equivalent to running the primary UART control channel in the bypass mode as described above.

Port Access and Routing

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

Table 19. MFP Pins for UART

MFP Pin	Default Function	UART Function #1	UART Function #2	Notes
MFP0	GPIO0	RX1	RX2	Enable UART pass-through through register
MFP1	LOCK	TX1	TX2	Enable UART pass-through through register

MFP5	RX1	RX2	RX1	Enable UART pass-through through register
MFP6	TX1	TX2	TX1	Enable UART pass-through through register
MFP11	SDA_RX	N/A	N/A	I^2C or UART function selected through CFG0 pin
MFP12	SCL_TX	N/A	N/A	I^2C or UART function selected through CFG0 pin

On power-up, the device should be set to UART mode through the CFG0 latch. The function names in *Table 19* and ensuing UART sections assume the device is configured for UART mode.

By default, the primary UART control channel lines are brought out on MFP11 and MFP12 for RX and TX, respectively. Disable the primary control channel's line access by setting field DIS_LOCAL_CC in register 0x01. Also disable access to remote device control by setting field DIS_REM_CC in register 0x01.

Enabling UART Bypass Mode

Register Setting Change (Soft Bypass)

UART bypass mode can be enabled through register setting by first setting field BYPASS_EN in register 0x48. Next, configure a timeout (2ms, 8ms, 32ms, or no-timeout) by setting the field BYPASS_TO in register 0x48. Bypass mode is active only if there is UART activity. When there are no UART transitions detected for the selected timeout duration, the device exits bypass mode and re-enters base-mode. The timeout is optional. If field BYPASS_TO is set for no timeout, then the device remains in bypass mode until the next power-cycle.

Pin Setting Change (Hard-Bypass)

UART bypass mode can also be enabled by the mode select (MS) pin, which the user can bring out on MFP8. In this state, a high-voltage level on the MS pin enables bypass mode, while a low voltage level disables bypass mode. To enable this setting, set field REM_MS_EN in register 0x48.

Additionally, the MS pin can be set to use the GPIO2 pin instead of the function MS on MFP8. To enable this setting, set the field LOC_MS_EN in register 0x48. This setting might be needed if MFP8 is needed for another use.

Enabling UART Pass-Through Channels

The user can bring out the first pass-through UART channel (RX1/TX1) on MFP0/MFP1 or MFP5/6. The second pass-through UART channel (RX2/TX2) can also be programmed on MFP0/MFP1 or MFP5/6. Pass-through channels are enabled by setting the fields UART_1_EN and UART_2_EN in register 0x03. When enabling a UART pass-through channel, other MFP functions must be disabled first.

Programming Example

This example enables pass-through UART Channel 1.

```
#DES I2C Address=0x98
# Enable pass-through UART Channel 1 on DES MFP5/6
0x98,0x0003,0x53
# Enable pass-through UART Channel 1 on SER MFP7/8
0x80,0x0003,0x10
```

CRC for UART and Message Counter Transactions

The MAX96716A/MAX96716F 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 SER/DES 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 the 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, 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 CRC feature is enabled by setting fields CC_CRC_EN and CC_CRC_MSGCNTR_OVR in register 0x3004.

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

Enable CRC and Message Counter on Primary UART Control Channel

The following are the register writes to enable CRC for UART and Message Counter for primary UART control channel. (Only works with the UART host controller that supports CRC/Message Counter)

```
#DES I2C Address=0x98
#Enable CRC and Message Counter on DES
0x98,0x300F,0x07
```

UART Registers

Table 20 has registers needed to enable/disable primary and pass-through UART channels. *Table 20* also shows how to enable CRC for UART and Message Counter features.

Table 20. UART Registers

Register Address	Bits	POR	Decode
0x01	5:4	0x00	Bit [5]: Disable primary UART Control Channel connection to RX and TX pins Bit [4]: Disable access to remote device control-channel over GMSL2 Link A connection
0x03	6:4	0x05	Bit [6] Enable swap of pass-through 1 assignment for pass-through 2 assignment Bit [5] Enable UART pass-through channel 2 Bit [4] Enable UART pass-through channel 1
0x03	2	0b0	Bit [2] Disable access to remote device control-channel over GMSL2 Link B connection
0x04	5:4	0b00	Bit [5] Disable pass-through channel 1 on GMSL2 Link B Bit [4] Disable pass-through channel 1 on GMSL2 Link A
0x05	4	0b0	Bit [4] Disable pass-through channel 2 on GMSL2 Link A
0x06	5:4	0b0	Bit [5] Disable pass-through channel 2 on GMSL2 Link B Bit [4] UART enabled when set to a 0***
0x300F	4:0	0x06	Bit [4:3] Message Counter Enable 0b00: Primary I^2C/UART Control Channel 0b01: Pass-through I^2C /UART Channel 1 0b10: Pass-through I^2C /UART Channel 2 0b11: RSVD Bit [2]: Enable I^2C /UART message counter* Bit [1]: Enable I^2C /UART CRC packeting* Bit [0]: Enable manual override of UART CRC or message counter configuration**
0x3008	0	0x00	Bit [0] Reset Message Counter Value
0x3009	1:0	0b00	Bit [1] Rest Reset Message Counter Error Count to 0 Bit [0] Reset CRC Error Count to 0
0x300A	7:0	0b00000000	Bits [7:0] Read I^2C /UART CRC Value
0x300B	7:0	0b00000000	Bits [7:0] Read Message Counter Low Bits
0x300C	7:0	0b00000000	Bits [7:0] Read Message Counter High Bits

0x300D	7:0	0b00000000	Bits [7:0] Read number of I^2C/UART CRC errors
0x300E	7:0	0b00000000	Bits [7:0] Read number Message Counter errors

Note*: Only active when Bit [2] is also set to 1.
Note:** If set to a 0, then CRC and message counter features are disabled.
Note*:** This bit is set according to the CFG0 pin value on power-up. Writing to this register is not recommended.

Serial Peripheral Interface (SPI)

Overview

SPI is available on the MAX96716A/MAX96716F deserializers. Unlike I^2C and UART, SPI cannot be used to modify any registers in either the 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 (example, 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 peripheral/subordinate devices.

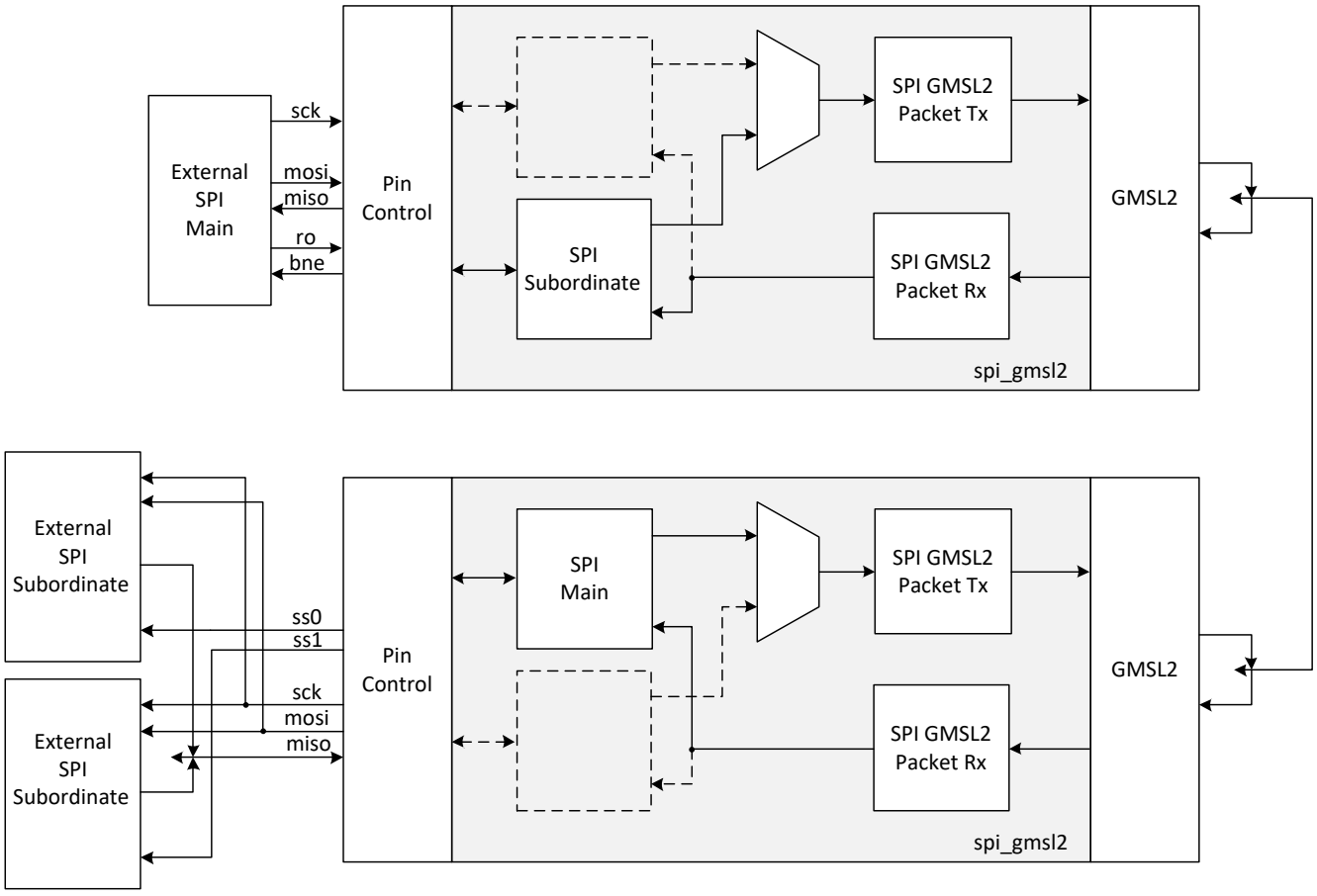

Figure 10. SPI Architecture

MFP/CFG Pin Setup for SPI

SPI setup considerations for GMSL devices:

- 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.
- Read only (RO) is an input bit that determines if the SPI subordinate is in read or write mode.
- Buffer not empty (BNE) 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 to avoid buffer overflow.

SPI Setup Registers

Table 21 shows some of the important setup registers for enabling SPI. Refer to the 0x170-0x178 register block in the data sheet for additional details on configuring the SPI main, subordinate, SCLK timings, etc. See the programming script in the following section as an example.

Table 21. 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 = Accept packets with proper ID 1 = Ignore ID and accept 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 the serializer and deserializer in the following order to initialize SPI (starting from the default values):

1. Configure SPI mode 0 or 3 on the serializer and deserializer.
2. Set SS output polarity (remote side).
3. Set the clock delay and SCLK rate high/low times (in number of 300MHz clocks).
4. Program the IO pin enables (BNE/RO/SS1/SS2).
5. Configure internal GMSL main/subordinate 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 the 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
```

*Note: If the 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 the above 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 V_{DDIO} on the external SPI main device matches the V_{DDIO} on GMSL Ser and Des).
- Start GMSL GUI.
- Load GMSL script to enable SPI.
- Reconnect external SPI main device.
- Set read only (RO) high and write 0xA0, 0xA4 *(0xA0 to 0xA3 are used for SPI ID selection, 0xA4 asserts SS1, 0xA5 asserts SS2, and 0xA6 deasserts both SS1 and SS2).*
- As a best practice, before starting SPI data transfer, check buffer not empty (BNE) to ensure that the buffer is empty. If BNE is high, there is data in the RX buffer 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. *Figure 13* and *Figure 14* show the SPI oscilloscope probes on the SPI clock and data output (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.

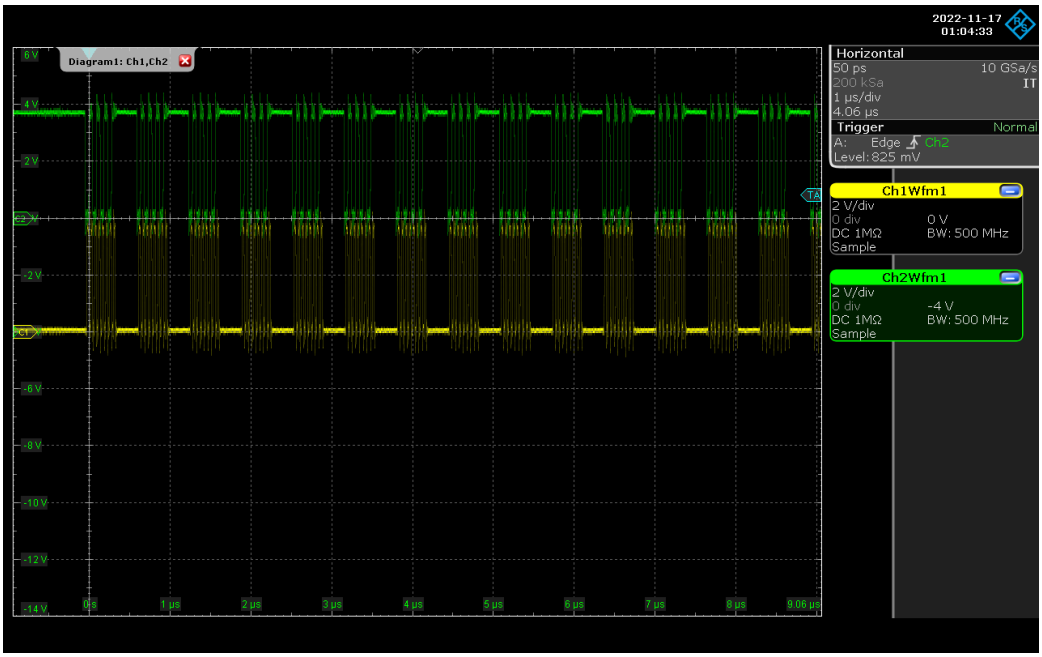

Figure 13. SPI Clock and Data at Final Output (at External SPI Subordinate), No Video on GMSL Link

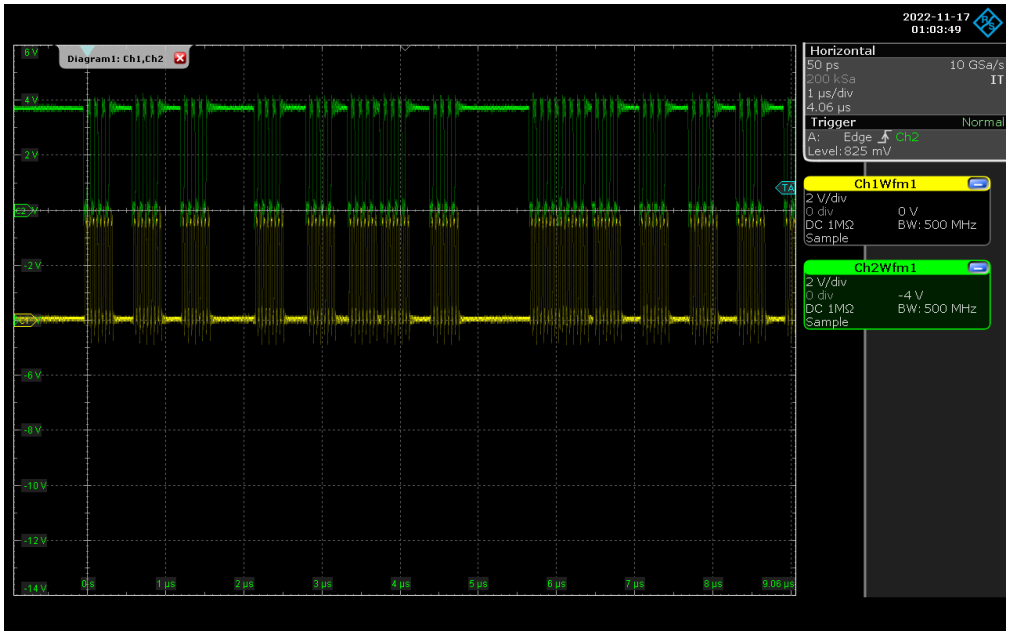

Figure 14. SPI Clock and Data at Final Output (at External SPI Subordinate), 92% Video on GMSL Link

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 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 sends the data out on the MFP pins. It also sends 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 through MOSI.
- RO = 1: Data transmitted between subordinate and main through 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 it is an output pin. It is in fact an input pin toggled externally high or low depending on the read or write operation.

It is recommended to limit the amount of “bytes in transit” (bytes 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 the data was sent properly.

The following is an example of transferring four bytes of SPI data across the GMSL link:

1) RO is pulled high, and the A0 and A4 control commands are sent.
2) RO is pulled low, and four bytes of SPI data (0x80, 0x04, 0x01, 0x47) are sent from the external SPI main device to the GMSL input side (serializer).
3) The last three signals in the graph show SPI data output from the remote side of the link (deserializer). There may be some delay between the input and output of the SPI data across the GMSL link.
4) 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.

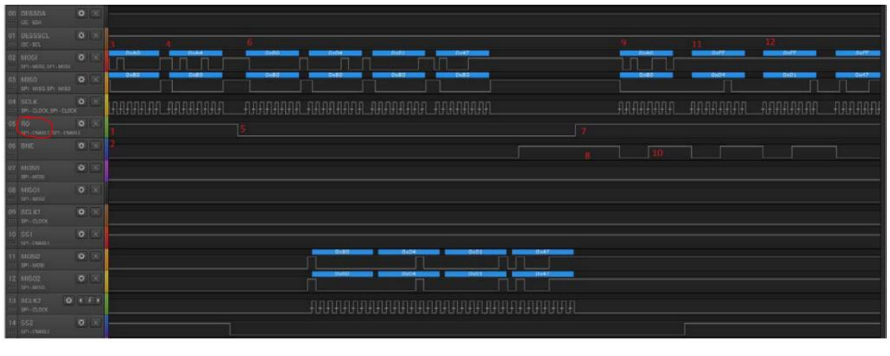

Figure 15. SPI Transmission Example

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 (FSYNC)

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. Frame synchronization has more configurable options on the deserializer side of the link (see the user guide of the deserializer for additional details).

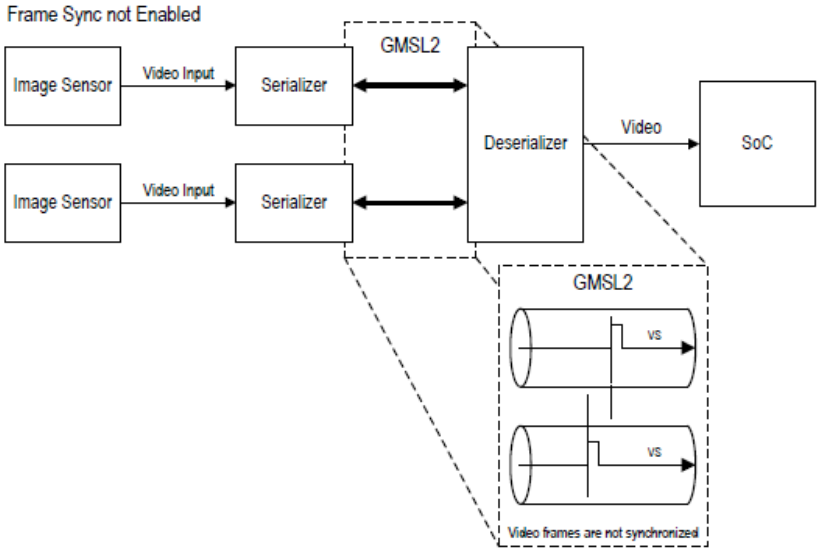

Figure 16. Frame Alignment (Without Frame Sync)

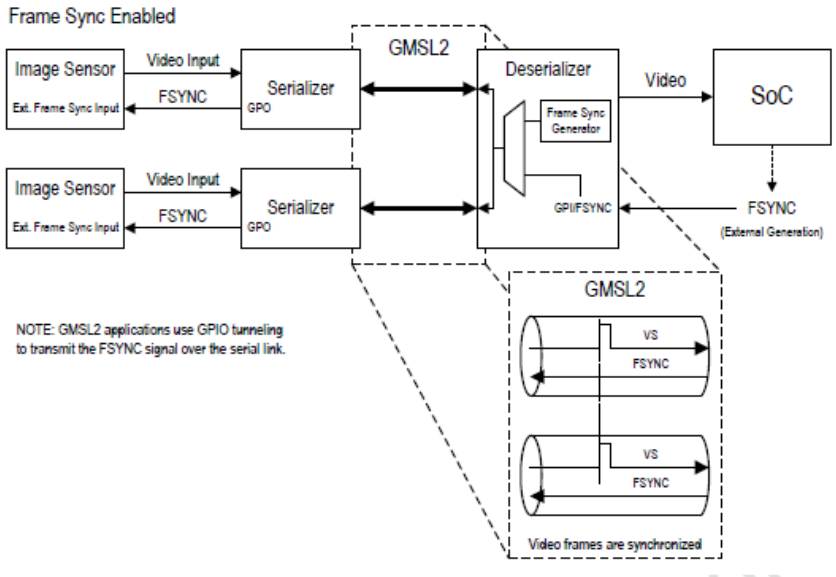

Figure 17. Frame Alignment (Frame Sync Enabled)

Configuration

The frame synchronization can be enabled on any multifunction pin (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 Examples

External FYSNC

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

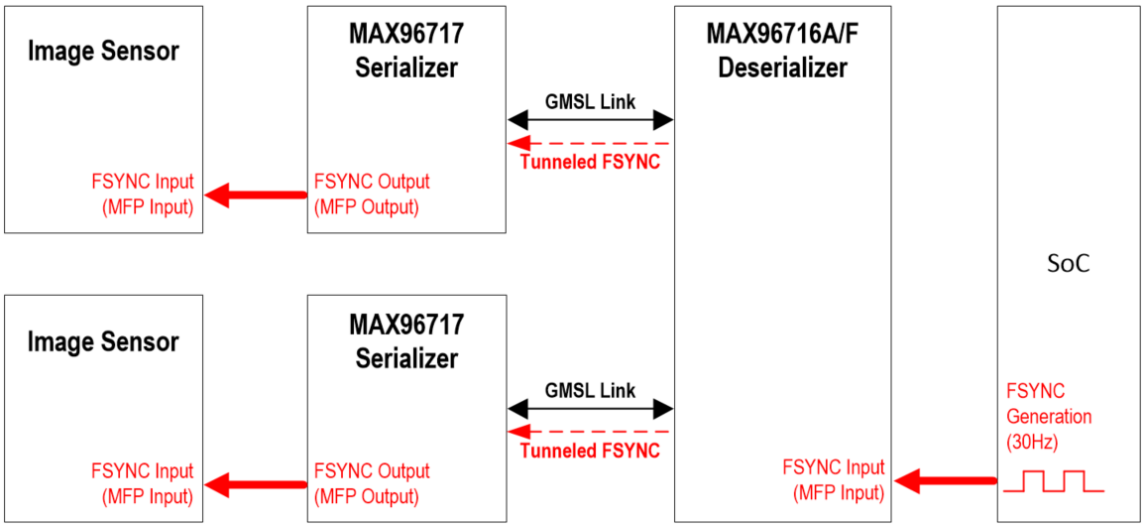

Figure 18. External Frame Sync Example

```
#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 MAX96716A/MAX96716F 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 are validated and the digital core assumes normal operations. In all power modes, the power manger monitors power supplies for undervoltage and overvoltage conditions. In sleep mode, the power manager minimizes current consumption and can quickly restore device configurations after waking up.

Table 22. Power Manager and Sleep Mode Availability

Part Number	Power Manager	Sleep Mode
MAX96716A	Supported	Supported
MAX96716F	Supported	Supported

Device Power Operation

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

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.

***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 MAX96716A/MAX86716F share 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.
    a. An internal LDO regulates the voltage to 1.0V.
- **V_{DD18}:** 1.8V power rail.
- **V_{TERM}**: 1.2V power rail.
- **V_{DDIO}**: 1.8V or 3.3V 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.

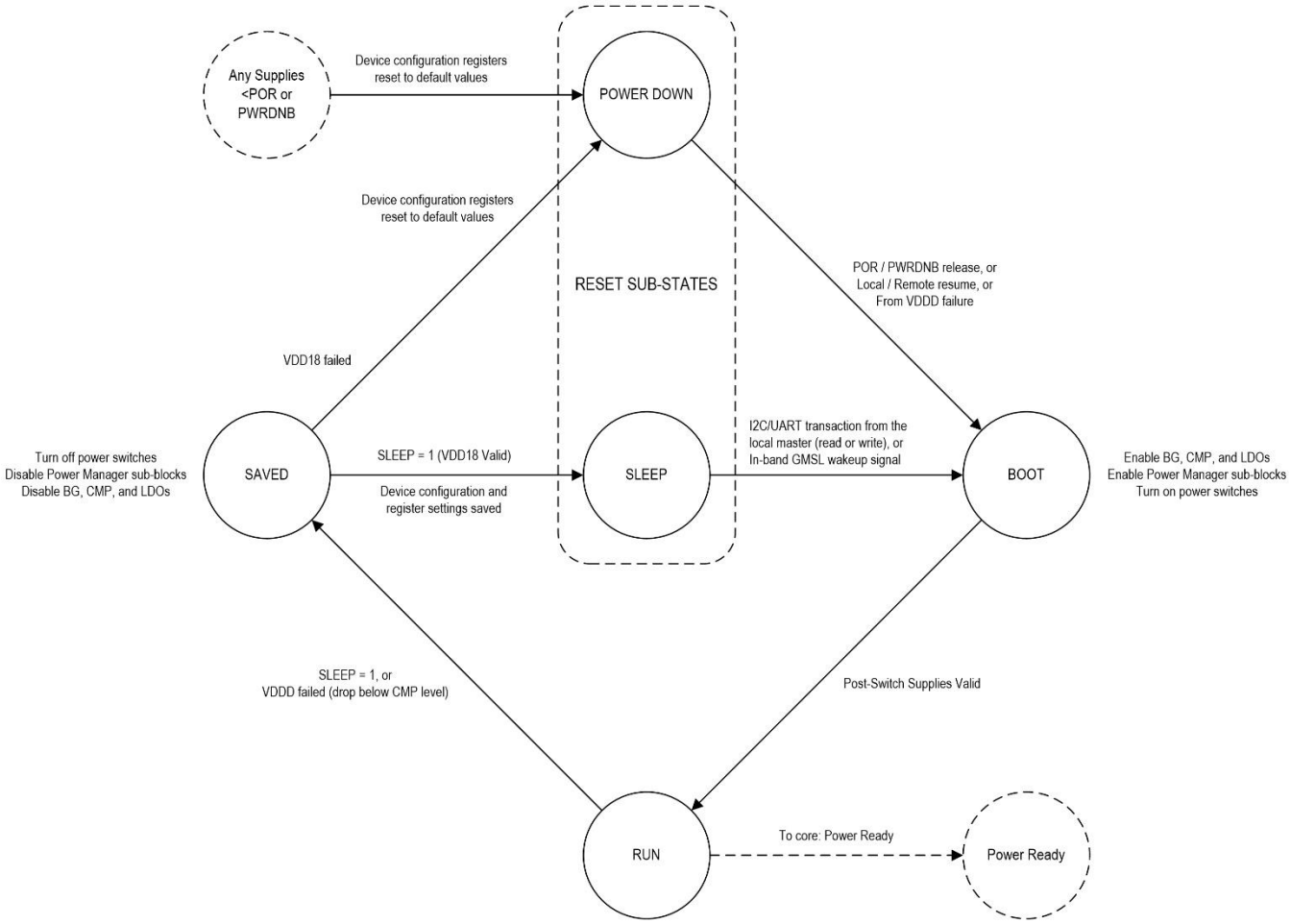

Figure 19. Power Manager State Diagrams

Reset (Power Down/Sleep)

Power down and sleep are two sub-states 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 V_{DD18} falls too low. Deasserting 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 I^2C /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 V_{DD18} 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 wakeup 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 sub-blocks 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. 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

The MAX96716A/MAX96716F supports sleep mode, which 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 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 Wakeup Sequences

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

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.

Wakeup (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 I^2C /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 wakeup 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 GMSL2 family includes a global soft reset function called RESET_ALL. This is a self-clearing reset command intended to reset all sub-systems 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 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 the above-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 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 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 between 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 CRC 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.

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

Register CRC Registers

Table 23. MAX96716A/MAX96716F Register CRC Registers

Register Name	Register Function	Register Address
FS_INTR0	Register CRC Error Flag to ERRB	0x3010
FS_INTR1	Register CRC Error Flag	0x3011
REGCRC0	Register CRC Feature Enable	0x3000
REGCRC1	Register CRC Period	0x3001
REGCRC2	Register CRC LSB	0x3002
REGCRC3	Register CRC MSB	0x3003
REGCRC8	Skip Register 0 LSB	0x3030
REGCRC9	Skip Register 0 MSB	0x3031
REGCRC10	Skip Register 1 LSB	0x3032
REGCRC11	Skip Register 1 MSB	0x3033
REGCRC12	Skip Register 2 LSB	0x3034

REGCRC13	Skip Register 2 MSB	0x3035
REGCRC14	Skip Register 3 LSB	0x3036
REGCRC15	Skip Register 3 MSB	0x3037
REGCRC16	Skip Register 4 LSB	0x3038
REGCRC17	Skip Register 4 MSB	0x3039
REGCRC18	Skip Register 5 LSB	0x303A
REGCRC19	Skip Register 5 MSB	0x303B
REGCRC20	Skip Register 6 LSB	0x303C
REGCRC21	Skip Register 6 MSB	0x303D
REGCRC22	Skip Register 7 LSB	0x303E
REGCRC23	Skip Register 7 MSB	0x303F

Register CRC Protected Status Registers

There are video status registers that change logic levels depending on whether there is input video. It is recommended to reset the CRC value when system changes states (that is, with or without video running). Another option is to 'SKIP' these status registers.

Table 24. MAX96716A/MAX96716F Video Status CRC Protected Registers

Register	Bitfield(s)
VPRBS Y	VIDEO_LOCK, Pipe Y
VPRBS Z	VIDEO_LOCK, Pipe Z
BACKTOP1	CSIPLLY_LOCK, Pipe Y
BACKTOP1	CSIPLLZ_LOCK, Pipe Z
GPIOX: GPIO C	GPIO RECVED

Note: *Where X is the GPIO number. It is recommended to skip GPIO_RECVED register from the CRC calculation.

Reg CRC Skip Example

Table 25. MAX96716A/MAX96716F Video Status Register CRC Skip Registers

Register	Bitfield(s)	Register Address	SKIPx_LSB	SKIPx_LSB
VPRBS Y	VIDEO_LOCK, Pipe Y	0x1FC	0xFC	0x01
VPRBS Z	VIDEO_LOCK, Pipe Z	0x21C	0x1C	0x02
BACKTOP1	CSIPLLY_LOCK, Pipe Y	0x308	0x08	0x03
BACKTOP1	CSIPLLZ_LOCK, Pipe Z	0x308	0x08	0x03
GPIOX: GPIO C	GPIO RECVED	Varies	Varies	Varies

Reference-over-Reverse (RoR)

Overview

All GMSL2 parts must receive an external clock for full function (typically through a crystal). Some serializers, such as the MAX96717/MAX96716F, 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 MAX96716A/MAX96716F is supplying the reference signal on the reverse channel of the GMSL link.

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

- 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

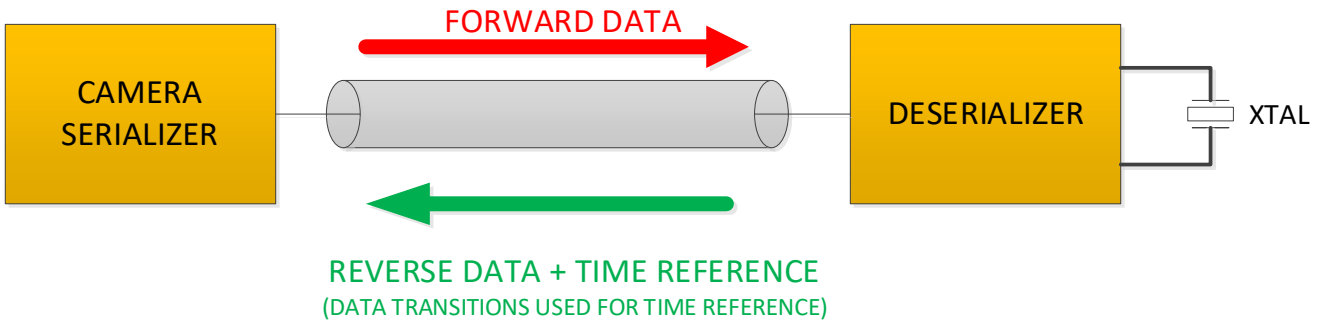

Figure 20. RoR Block Diagram

Enabling RoR Mode

The MAX96716A/MAX96716F devices automatically detect whether RoR feature is enabled. No additional deserializer configuration is required. The serializer selects between XTAL or RoR mode through CFG0. Refer to the serializer user guide for more RoR information.

GMSL Link Lock in RoR Mode

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

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 (SSC)

Spread spectrum clocking (SSC) is an alternate configuration for the GMSL link that helps with electromagnetic interference (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.

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 register bit fields.

Table 26. MAX96716A/MAX96716F POST Status Register

Register	Bitfield Name	Bits	Default Value	Decode
0x3020	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

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

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 gets lost or corrupted. Although the MAX96716A/MAX96716F family can transmit at 3Gbps or 6Gbps 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 *Table 27*.

Table 27. GMSL2 Maximum Video Payloads

GMSL2 Mode	Maximum Video Payload
3Gbps Mode	2.6Gbps (2600Mbps)
6Gbps Mode	5.2Gbps (5200Mbps)

Calculating Bandwidth

The basic video payload can be calculated using these equations.

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

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

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.

As long as 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.

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

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

The majority of GMSL2 serial link bandwidth comprises video transmission. The total link bandwidth consumed by the video is derived from the incoming video stream and calculated by multiplying the pixel clock (PCLK) expressed in MHz, bits per pixel (bpp), and GMSL2 link overhead factors. Note that control channel features (example, GPIO, SPI) affect link bandwidth consumption and must be considered, if enabled.

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 doubling or tripling the BPP. When transmitting an 8 BPP datatype at 16 BPP or 24 BPP, less bandwidth is consumed for the same reason as when using tunneling mode. With regards to

bandwidth, zero padding has the opposite effect of doubling or tripling. 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 to calculate 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.

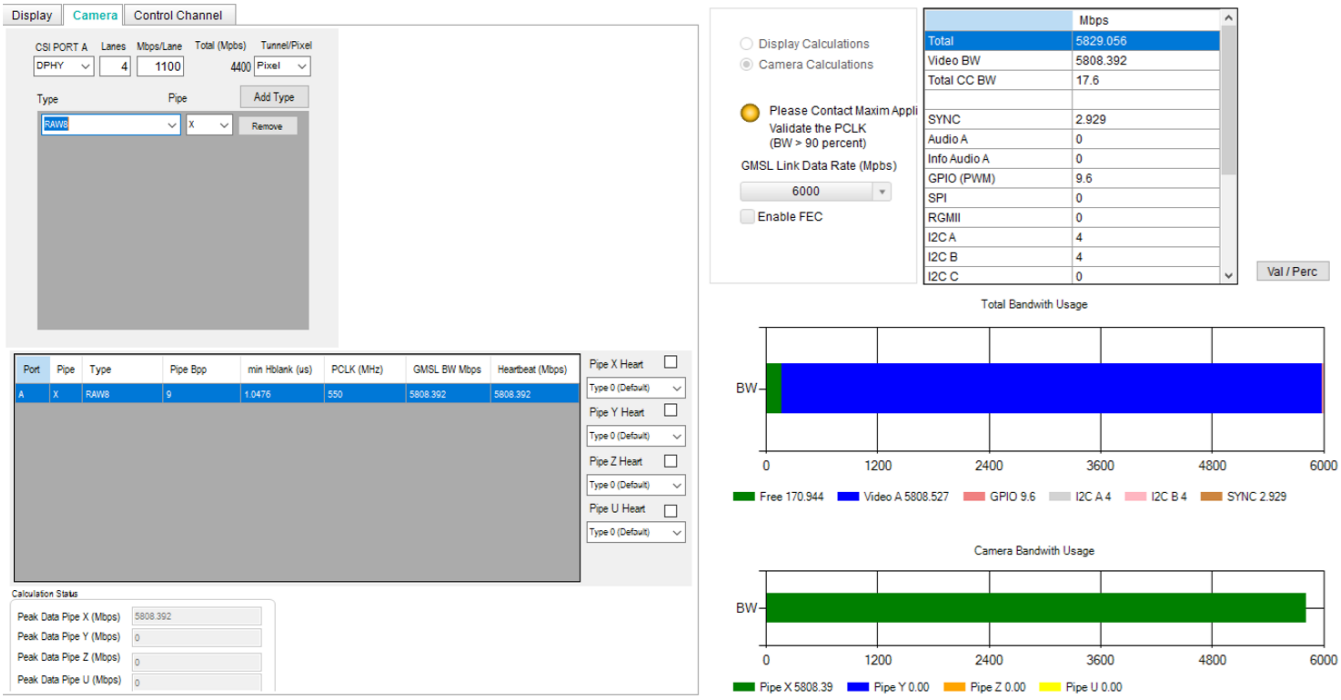

Figure 21. Bandwidth Calculations Without Doubling

Case 2: Transmitting 4-lanes of RAW8 data at 1100Mbps/lane by 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 is doubled.

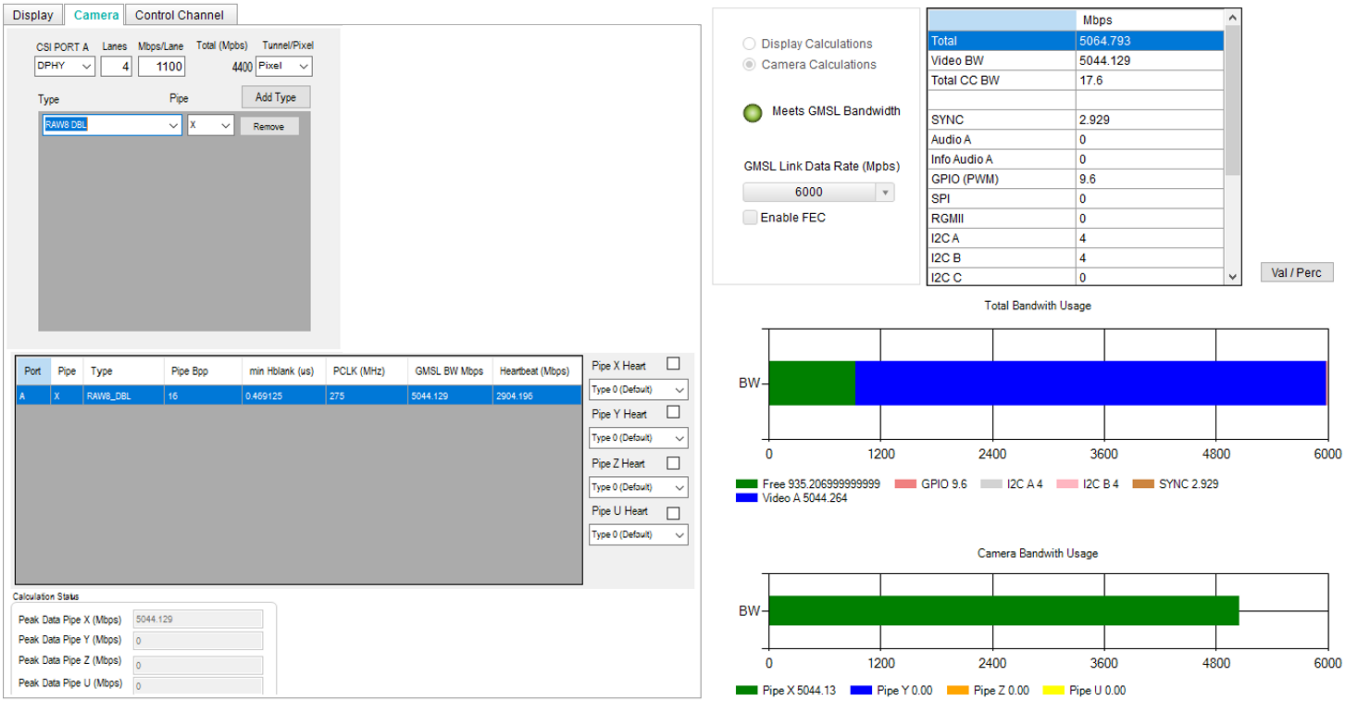

Figure 22. Bandwidth Calculations with Doubling

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 if data is being transferred within the deserializer. There are packet counters for MIPI controller and PHY.

Table 28. MIPI Counter Registers

Register	Bits	Default Value	Description	Decode
0x342	7:4	0x00	Packet count of CSI-2 Controller 2	0bXXXx: Toggling bits indicate MIPI data is active on the controller 2
	3:0	0x00	Packet count of CSI-2 Controller 1	0bXXXx: Toggling bits indicate MIPI data is active on the controller 1
0x343	7:4	0x00	Packet count of CSI-2 duplicate Controller 2	0bXXXx: Toggling bits indicate MIPI data is active on the duplicate controller 2
	3:0	0x00	Packet count of CSI-2 duplicate Controller 1	0bXXXx: Toggling bits indicate MIPI data is active on the duplicate controller 1
0x344	7:4	0x00	Packet count of MIPI PHY 1	0bXXXx: Toggling bits indicate MIPI data is active on PHY 1
	3:0	0x00	Packet count of MIPI PHY 0	0bXXXx: Toggling bits indicate MIPI data is active on PHY 0
0x345	7:4	0x00	Packet count of MIPI PHY 3	0bXXXx: Toggling bits indicate MIPI data is active on PHY 3
	3:0	0x00	Packet count of MIPI PHY 2	0bXXXx: Toggling bits indicate MIPI data is active on PHY 2

Packet Counter Example

The following example demonstrates the use case of the MIPI packet counters.
Setup: MAX96717 and MAX96716A with SV4E introspect generator and analyzer.

1) Connect the SV4E Introspect generator to MAX96717.
2) Connect Link A of MAX96716A to MAX96717.
3) Generate a CSI_RGB888 pattern on the introspect generator.
4) Check PCLKDET on the MAX96717 and video lock on the MAX96716A.
5) Now check registers 0x38D, 0x38E, 0x390 on MAX96717 to check if bits are toggling.
6) Now check registers 0x342, 0x344 and 0x345 on MAX96716A to check if bits are toggling.
7) If the bits are not toggling, then check the parts configuration.

HSYNC, VSYNC, DE (HVD) Outputs and Counters

Overview

The GMSL2 CSI-2 deserializers can output the VS or DE signals from the video pipe data streams on MFP0 or MFP3 pins. Each pin can be selected to output from the video pipe.

GPIOs generally multiplex with multiple functions. Therefore, to output sync pulses, the default or higher-priority GPIO function(s) of the above pins must be disabled.

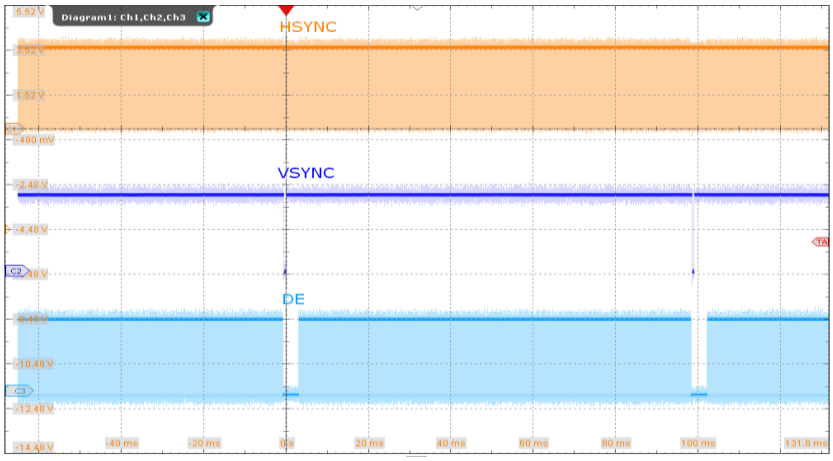

Figure 23. HS, VS and DE Outputs from GPIOs

Table 29. Sync Pulse Output Registers

Register	Bitfield Name	Bits	Default Value	Decode
0x540	VS_OUT1	7:5	0b000	0b001=Pipe Y 0b010=Pipe Z
0x541	VS_OUT2	7:5	0b000	0b001=Pipe Y 0b010=Pipe Z
0x542	HS_OUT1	7:5	0b000	0b001=Pipe Y 0b010=Pipe Z

Alternatively, sync pulse signals can be monitored through registers. Registers *0x575* contain status flags indicating the detection of DE/HS/VS signals for each video pipe. The polarity of the HS/VS signals is accessed in the same register. HVD counters may also be read to verify the pipe count values match the expected video input values. Additionally, comparators may be set up and enabled to flag dropped frames or lines as errors. See *Table 30* for additional information.

Table 30. Sync Signal Status Registers

Register	Bitfield Name	Bits	Default Value	Decode
0x575	DE_DET_Y, VS_DET_Y, HS_DET_Y	6:4	0b000	Bit 6: DE activity on Pipe Y 0b0=No activity present 0b1=Active DE Signal Bit 5: VS activity on Pipe Y 0b0=No activity present 0b1=Active VS Signal

				Bit 4: HS Activity on Pipe Y 0b0=No activity present 0b1=Active HS Signal
0x575	VS_POL_Y, HS_POL_Y	1:0	0b00	Bit 1: VS Polarity Status on Pipe Y 0b0=Active Low 0b1=Active High Bit 0: HS Polarity Status on Pipe Y 0b0=Active Low 0b1=Active High
0x576	DE_DET_Z, VS_DET_Z, HS_DET_Z	6:4	0b000	Bit 6: DE Activity on Pipe Z 0b0=No activity present 0b1=Active DE Signal Bit 5: VS Activity on Pipe Z 0b0=No activity present 0b1=Active VS Signal Bit 4: HS Activity on Pipe Z 0b0=No activity present 0b1=Active HS Signal
0x576	VS_POL_Z, HS_POL_Z	1:0		Bit 1: VS Polarity Status on Pipe Y 0b0=Active Low 0b1=Active High Bit 0: HS Polarity Status on Pipe Y 0b0=Active Low 0b1=Active High

Programming Example

```
#DES I2C Address=0x98
#Enable VS_OUT2 on MFP0 from Pipe Y
0x98,0x0540,0x20
Enable HS_OUT1 on MFP3 from Pipe Y
0x98,0x0542,0x80
```

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 occurred.

The error bar (ERRB) pin is an MFP pin that logically NORs many of the errors. So, it is a convenient way to check for errors. It is available at MFP4. Whether an error is included in the ERRB output depends on if its output-enable (OEN) is high. Most OENs are high by default.

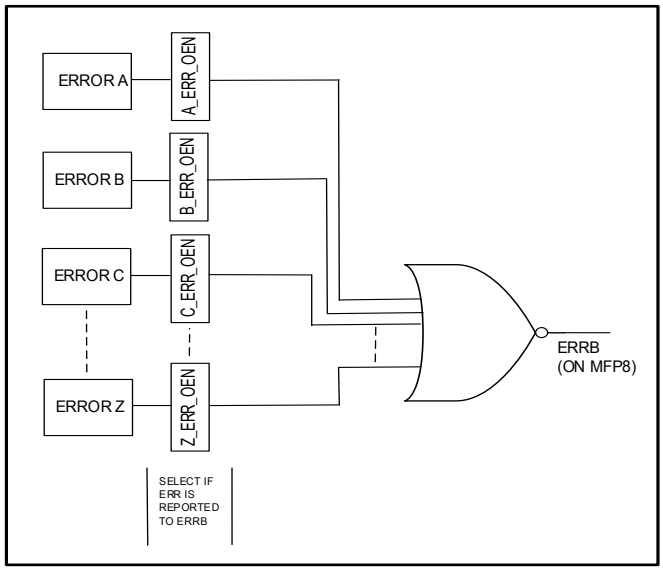

Figure 24. ERRB Reporting Flow

See *Table 31* for a description of each error flag in the MAX96716A/MAX96716F. Refer to the device data sheets for more details.

Table 31. Error Flags Table

Error Flag	Error Description
DEC_ERR_FLAG	Errors detected in GMSL packet on Link A
DEC_ERR_FLAG_B	Errors detected in GMSL packet on Link B
IDLE_ERR_FLAG	Idle-Word Error Flag on Link A
IDLE_ERR_FLAG_B	Idle-Word Error Flag on Link B
LFLT_INT	Line-Fault Interrupt Asserted when either one of line-fault monitors indicates a fault status
MEM_INT_ERR_FLAG	Memory Error Flag
REM_ERR_FLAG	Received remote side error status
PHY_INT_A	PHY interrupt on Link A
PHY_INT_B	PHY interrupt on Link B
WM_ERR_FLAG	Watermark Error Flag
PKT_CNT_FLAG	Packet Count Error Flag on Link A
PKT_CNT_FLAG_B	Packet Count Error Flag on Link B

RT_CNT_FLAG	ARQ Retransmission Event Flag for Link A
RT_CNT_FLAG_B	ARQ Retransmission Event Flag for Link B
MAX_RT_FLAG	ARQ Maximum Retransmission Event Flag for Link B
MAX_RT_FLAG_B	ARQ Maximum Retransmission Event Flag for Link B
FEC_RX_ERR_FLAG	FEC Receive Errors Flag on Link A
FEC_RX_ERR_FLAG_B	FEC Receive Errors Flag on Link B
EOM_ERR_FLAG_A	Eye opening monitor Error Flag on Link A
EOM_ERR_FLAG_B	Eye opening monitor Error Flag on Link B
VID_PXL_CRC_ERROR	GMSL Packet CRC Error Flag
VPRBS_ERR_FLAG	Video PRBS Error Flag
LCRC_ERR_FLAG	Line CRC Error Flag
FSYNC_ERR_FLAG	Frame Synchronization Error Flag
VDDBAD_INT_FLAG	Power Supply Error Flag: Internal LDO Under Voltage Error
VDDCMP_INT_FLAG	Power Supply Error Flag: V_{DD18}, V_{DDIO}, CAP_VDD, and V_{TERM} Undervoltage Error
VDD_OV_FLAG	Power Supply Error Flag: V_{DD} Overvoltage Flag
VDD18_OV_FLAG	Power Supply Error Flag: V_{DD18} Overvoltage Flag
MEM_ECC_ERR1_INT	2-Bit Uncorrectable Memory ECC Error Flag
MEM_ECC_ERR2_INT	1-Bit Uncorrectable Memory ECC Error Flag
EFUSE_CRC_ERR_FLAG	An issue with the device EFUSE occurred, the device should no longer be used.
RTTN_CRC_INT	Retention Memory Restore CRC Error Flag
I2C_UART_ CRC_ERR_INT	I^2C/UART CRC error flag. Asserted when CRC byte sent does not match calculated value.
I2C_UART_ MSGCNTR_ ERR_INT	I^2C /UART message counter error flag. Asserted when two message counter bytes sent do not match expected count value.
VIDEO_MEM_OVERFLOW	Backtop Memory Overflow Error Flag
LOSS_OF_LOCK_FLAG	Loss of Lock Detection Error Flag
VID_OVERFLOW_FLAG	Video Pipe Overflow Error Flag
TUN_ECC_CORR_ERR	Tunnel Mode: Correctable errors on DPHY ECC or CPHY header CRC Error Flag
TUN_ECC_UNCORR_ERR	Tunnel Mode: Uncorrectable errors on DPHY ECC or CPHY header CRC Error Flag
TUN_DATA_CRC_ERR	Tunnel Mode: DPHY/CPHY Data CRC Error Flag Error Flag

General-Purpose Input and Output (GPIO)

Overview

The MAX96716A/MAX96716F family has twelve multifunction pins (MFPs). Depending on the pin, they can be used as either full or partial general-purpose input and output (GPIO) pins or for other functionality (example, I^2C, LOCK, ERRB, etc.). This section explains the GPIO function of MFP pins. Refer to the data sheets for additional details on GPIO capabilities and default states after power-up.

The GPIO blocks of the MAX96716A/MAX96716F family 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.

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 MAX96716A/MAX96716F use 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 (that is, 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.

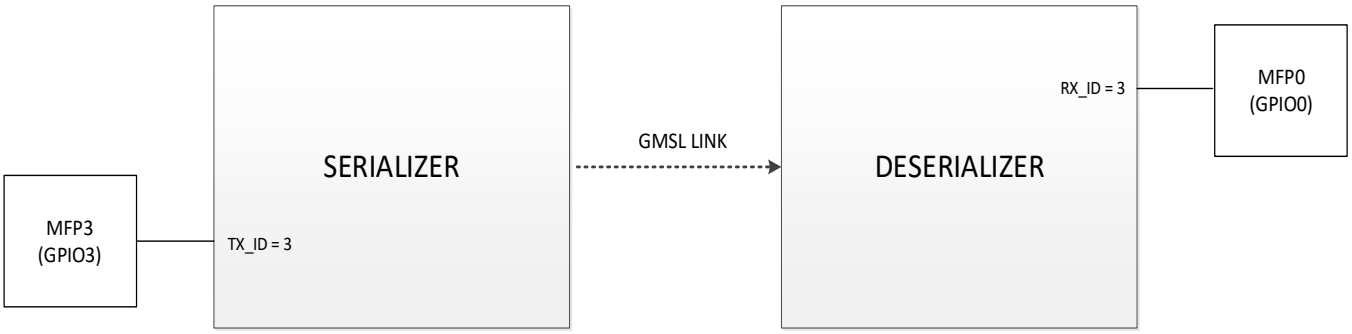

Figure 25. GPIO Forwarding Example with a Transition from MFP3 to MFP0

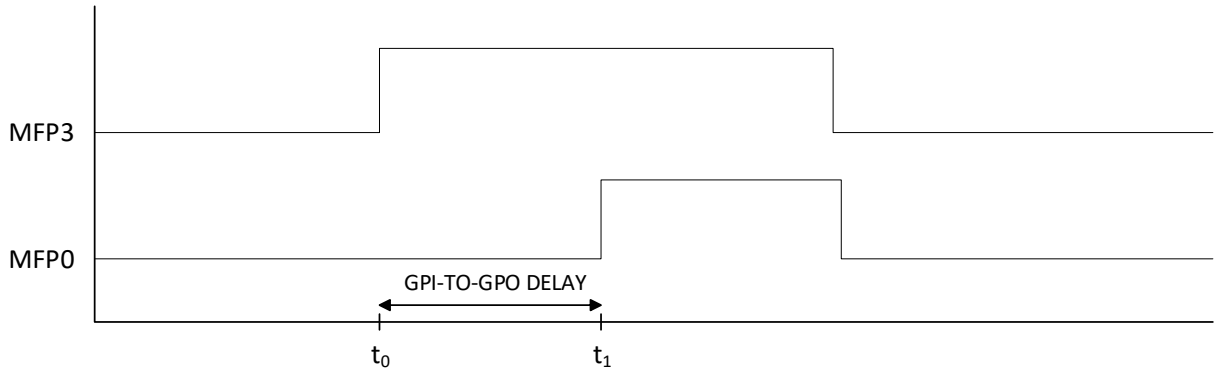

Figure 26. GPIO Forwarding Timing Diagram

MFP Capabilities: GPIO, GPI, GPO, and ODO

The MAX96716A/MAX96716F have twelve MFPs; five of these MFPs are general-purpose input or output (GPIO), two are general-purpose-output (GPO), four are general-purpose-input (GPI), and two are general-purpose-input, open-drain output (GPI_ODO), which are reserved for the primary control channel. *Table 32* shows the GPIO capabilities of each MFP.

Table 32. Multifunction Pin (MFP) Capabilities

Pin Name	Capability
MFP0	GPIO0
MFP1	GPIO1
MFP2	GPO2
MFP3	GPO3
MFP4	GPIO4
MFP5	GPIO5
MFP6	GPIO6
MFP7	GPI7
MFP8	GPI8
MFP9	GPI9
MFP10	GPI10
MFP11	GPI11_ODO11
MFP12	GPI12_ODO12

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 (that is, NMOS output driver enabled) by writing *OUT_TYPE* = 0 and for push-pull mode (that is, 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:

1. 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.
2. 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 27* is an example of this configuration.

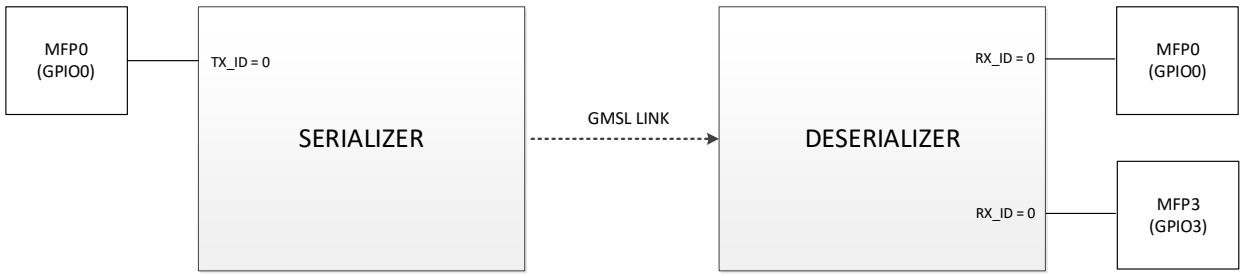

Figure 27. GPIO Broadcasting

GPIO Delay Compensation

In the non-delay-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 GMSL2 link. Delay compensation can be used to ensure that the timing delay between input transition and output transition is constant. *Table 33* shows the typical values and registers to set delay compensation.

Table 33. GPIO (with/without) Delay Compensation Values

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

Table 34. 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	GPIO_REV_CDLY	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 I^2C 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 GMSL2 link.
- Set *GPIO_OUT* to the desired value.

Table 35. GPIO Registers

Register	Bits	Default Value	Deode
0x2B0	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
0x2B1	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
0x2B2	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

0x2B3 to 0x2D6	…	…	Repeat Registers A, B, C for GPIO1 to GPIO12.

GPIO Programming Example

In this example, GPIO0 on a MAX96717 serializer is forwarded across the link to GPIO0 on a MAX96716A/MAX96716F 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.

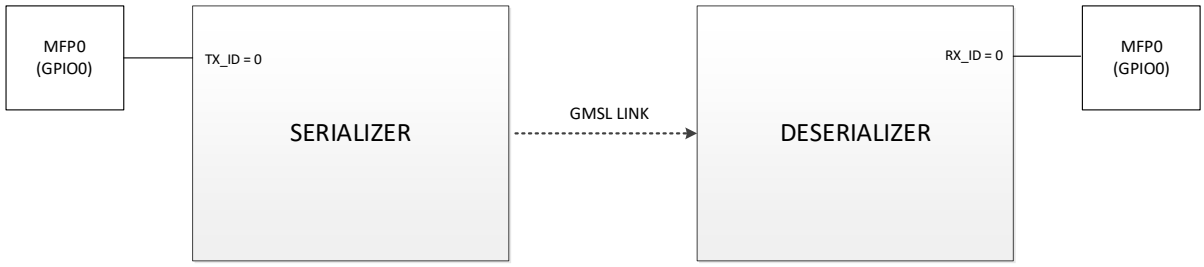

Figure 28. GPIO Forwarding Programming Example

```
#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 MAX96716A/MAX96716F 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 (example, 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.

Operation

The configurable slew rate applies to the various MFP functions differently:

1. **I^2C/UART**: MFP pins operating as an I^2C or UART function (that is, control channel or pass-through) are not affected by the MFP rise/fall setting. The I^2C/UART circuitry has a fixed falling-edge slew rate, and the rising-edge slew rate is determined by the external pull-up resistor.
2. **Dedicated Function**: The rise and fall times of MFP pins assigned dedicated functions (example, SPI, RCLKOUT, or LOCK and ERR) can be adjusted by the MFP slew rate control registers.
3. **GPIO**: MFP pins operating as GPI or GPO can be adjusted by the MFP rise/fall slew rate control register.

The V_{DDIO} supply voltage affects the I/O slew rate. The impact of the chosen V_{DDIO} voltage must be considered when programming MFP slew rates.

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. *Table 37* presents the typical rise and fall times for GMSL2 devices.

Table 36. MFP Slew Rate Registers

Register	Bitfield Name	Bits	Default Value	Description
0x570	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
	PIO03_SLEW	7:6	0x11	MFP3 Slew Rate 0b00=Fastest Slew Rate 0b11=Slowest Slew Rate

0x571	PIO04_SLEW	1:0	0x11	MFP4 Slew Rate 0b00=Fastest Slew Rate 0b11=Slowest Slew Rate
	PIO07_SLEW	7:6	0x10	MFP7 Slew Rate 0b00=Fastest Slew Rate 0b11=Slowest Slew Rate
0x572	PIO08_SLEW	1:0	0x10	MFP8 Slew Rate 0b00=Fastest Slew Rate 0b11=Slowest Slew Rate

Table 37. Typical MFP Rise/Fall Times

Register Value	Rise Time		Fall Time	
	V_{DDIO} = 1.8V	V_{DDIO} = 3.3V	V_{DDIO} = 1.8V	V_{DDIO} = 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

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 seen on the deserializer. The MIPI input to the serializer must be disabled before running this test.

Table 38. 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.

Table 39. Deserializer Video PRBS Generator and Checker Registers

Register	Bits	Default Value	Description	Decode
0x1FC, 0x21C	7	0x1	Pattern generator clock source for video PRBS7, PRBS9, PRBS24, checkerboard, and gradient patterns.	0b0: 150MHz 0b1: 375MHz (default)
	4	0b0	Enables video PRBS24 generator/checker	0b0: Disabled 0b1: Enabled
	3	0b0	Enables video PRBS7 generator/checker	0b0: Disabled 0b1: Enabled
	2	0b0	Enables video PRBS9 generator/checker	0b0: Disabled 0b1: Enabled
0x1FB, 0x21B	7:0	0x00	Video PRBS error counter, clears on read	0xXX: Number of video PRBS errors since last read
0x1D	2	0b1	Enables reporting of video PRBS errors (VPRBS_ERR_FLAG—0x2A) at ERRB pin.	0b0: Disable video PRBS error reporting 0b1: Enable video PRBS error reporting
0x1F	2	0b0	Video PRBS Error Flag. Asserted when VPRBS_ERR (0x1DB) > 0.	0b0: VPRBS_ERR = 0 0b1: VPRBS_ERR > 0

Programming Example

The following script configures a MAX96717 and a MAX96716A 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 1
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 (VPG)

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 deserializer has 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 input and uses the external 25 MHz crystal clock (that is, REFCLK input) to derive four different pixel clock (PCLK) options. Link lock is not required to use the VPG.

The VPG has its own register block settings for timing configurations. *Table 40* contains video pattern register addresses for the VPG.

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 port clock rate, which must be set to accommodate the VPG data stream. See *Table 41* for configuration details. *Table 42* contains reference information to select the VPG PCLK.

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

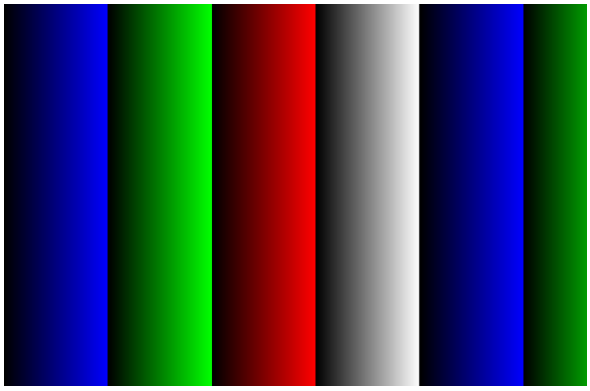

Figure 29. VPG (Gradient Pattern)

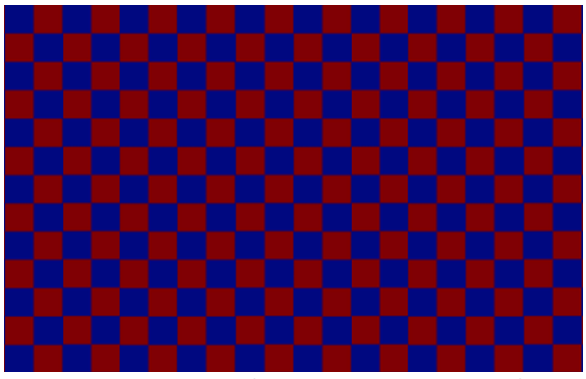

Figure 30. VPG (Checkerboard Pattern)

Table 40. Video Pattern Registers

Register	Bits	Default Value	Description
0x240	7:0	0x03	**Pattern Generator 0/1 Register:** Bit 7: Generate VS according to the timing setting. Bit 6: Generate HS according to the timing setting. Bit 5: Generate DE according to the timing setting. Bit 4: Invert VSYNC of Video Timing Generator. Bit 3: Invert HSYNC of Video Timing Generator. Bit 2: Invert DE of Video Timing Generator. Bits [1:0]: Video Interface Timing Generation Mode.
0x241	5:4	0x00	**Pattern Generator Mode Register:** Bits [5:4]: Pattern selection. 01 = Color Gradient 10 = Checkerboard
0x242	7:0	0x00	**VS Delay Register 2:** VS Delay in terms of PCLK cycles. (Bits [23:16])
0x243	7:0	0x00	**VS Delay Register 1:** VS Delay in terms of PCLK cycles. (Bits [15:8])
0x244	7:0	0x00	**VS Delay Register 0:** VS Delay in terms of PCLK cycles. (Bits [7:0])
0x245	7:0	0x00	**VS High Register 2:** VS High Period in terms of PCLK cycles. (Bits [23:16])
0x246	7:0	0x00	**VS High Register 1:** VS High Period in terms of PCLK cycles. (Bits [15:8])
0x247	7:0	0x00	**VS High Register 0:** VS High Period in terms of PCLK cycles. (Bits [7:0])
0x248	7:0	0x00	**VS Low Register 2:** VS Low Period in terms of PCLK cycles. (Bits [23:16])
0x249	7:0	0x00	**VS Low Register 1:** VS Low Period in terms of PCLK cycles. (Bits [15:8])
0x24A	7:0	0x00	**VS Low Register 0:** VS Low Period in terms of PCLK cycles. (Bits [7:0])
0x24B	7:0	0x00	**V2H Register 2:** VS edge to the rising edge of the first HS in terms of PCLK cycles. (Bits [23:16])
0x24C	7:0	0x00	**V2H Register 1:** VS edge to the rising edge of the first HS in terms of PCLK cycles. (Bits [15:8])
0x24D	7:0	0x00	**V2H Register 0:** VS edge to the rising edge of the first HS in terms of PCLK cycles. (Bits [7:0])
0x24E	7:0	0x00	**HS High Register 1:** HS High Period in terms of PCLK cycles. (Bits [15:8])
0x24F	7:0	0x00	**HS High Register 0:** HS High Period in terms of PCLK cycles. (Bits [7:0])
0x250	7:0	0x00	**HS Low Register 1:** HS Low Period in terms of PCLK cycles. (Bits [15:8])
0x251	7:0	0x00	**HS Low Register 0:** HS Low Period in terms of PCLK cycles. (Bits [7:0])
0x252	7:0	0x00	**HS Count Register 1:** HS pulses per frame. (Bits [15:8])

0x253	7:0	0x00	**HS Count Register 0:** HS pulses per frame. (Bits [7:0])
0x254	7:0	0x00	**V2D Register 2:** VS edge to the rising edge of the first DE in terms of PCLK cycles. (Bits [23:16])
0x255	7:0	0x00	**V2D Register 1:** VS edge to the rising edge of the first DE in terms of PCLK cycles. (Bits [15:8])
0x256	7:0	0x00	**V2D Register 0:** VS edge to the rising edge of the first DE in terms of PCLK cycles. (Bits [7:0])
0x257	7:0	0x00	**DE High Register 1:** DE High Period in terms of PCLK cycles. (Bits [15:8])
0x258	7:0	0x00	**DE High Register 0:** DE High Period in terms of PCLK cycles. (Bits [7:0])
0x259	7:0	0x00	**DE Low Register 1:** DE Low Period in terms of PCLK cycles. (Bits [15:8])
0x25A	7:0	0x00	**DE Low Register 0:** DE Low Period in terms of PCLK cycles. (Bits [7:0])
0x25C	7:0	0x00	**DE Count Register 0:** DE pulses per frame. (Bits [7:0])
0x25D	7:0	0x00	**Gradient Increment Register:** Set color gradient increment.
0x25E	7:0	0x00	**CHRK_COLOR_A_L Register:** Checkerboard mode color A low byte. RGB888 color Red (0-255).
0x25F	7:0	0x00	**CHRK_COLOR_A_M Register:** Checkerboard mode color A mid byte. RGB888 color Green (0-255).
0x260	7:0	0x00	**CHRK_COLOR_A_H Register:** Checkerboard mode color A high byte. RGB888 color Blue (0-255).
0x261	7:0	0x00	**CHRK_COLOR_B_L Register:** Checkerboard mode color B low byte. RGB888 color Red (0-255).
0x262	7:0	0x00	**CHRK_COLOR_B_M Register:** Checkerboard mode color B mid byte. RGB888 color Green (0-255).
0x263	7:0	0x00	**CHRK_COLOR_B_H Register:** Checkerboard mode color B high byte. RGB888 color Blue (0-255).
0x264	7:0	0x00	**CHRK_RPT_A Register:** Checkerboard mode color A repeat count.
0x265	7:0	0x00	**CHRK_RPT_B Register:** Checkerboard mode color B repeat count.
0x266	7:0	0x00	**CHRK_ALT Register:** Checkerboard mode alternate line count.
0x330	7	0x00	**MIPI PHY Mode Select Register:** Bit 7: Set to force all MIPI clocks running. Used with the MAX96716A/F internal pattern generator.

Table 41. PCLK Settings Registers

Register	Bits	Default Value	Description
0x38	1:0	0x00	**PIN_DRV_EN_0 Register:** Bit [1:0]: Set video pattern PCLK frequency. 00 = 25 MHz default 01 = 75 MHz 10 = PATGEN_CLK_SRC determines PCLK frequency 11 = PATGEN_CLK_SRC determines PCLK frequency
0x1FC	7	0x01	**PATGEN_CLK_SRC Register:** Select clock source for video pattern on pipe Y. 00 = 150 MHz 01 = 375 MHz Work together with Pattern CLK Freq register.
0x21C	7	0x01	**PATGEN_CLK_SRC Register:** Select clock source for video pattern on pipe Z. 00 = 150 MHz 01 = 375 MHz Work together with Pattern CLK Freq register.

Table 42 is a reference table for VPG PCLK selection. This is an alternative method of setting the pattern generator clock frequency with additional options for the PCLK frequency.

Table 42. Video Pattern PCLK Selection

Bitfield Name/Register		
PIN_DRV_EN_0 0x38, [1:0]	PATGEN_CLK_SRC Pipe Y: 0x1FC [7] Pipe Z: 0x21C [7]	PCLK Frequency
00	X	25MHz
01	X	75MHz
1x	0	150MHz
1x	1 (Default)	375MHz

Programming Example

```
# ----------------------
# Script that sets up pattern generator in MAX96716A out on MIPI Port A
# Settings: 8MP @8fps, RGB888
# ----------------------
#DES I2C Address=0x98
0x98,0x0240,0x13
0x98,0x0242,0x00
0x98,0x0243,0x00
0x98,0x0244,0x00
0x98,0x0245,0x00
0x98,0x0246,0x50
0x98,0x0247,0x78
0x98,0x0248,0x8A
0x98,0x0249,0x4E
0x98,0x024A,0x40
```

```
0x98,0x024B,0x00
0x98,0x024C,0x00
0x98,0x024D,0x00
0x98,0x024E,0x00
0x98,0x024F,0x2C
0x98,0x0250,0x0F
0x98,0x0251,0xEC
0x98,0x0252,0x08
0x98,0x0253,0x9D
0x98,0x0254,0x02
0x98,0x0255,0x94
0x98,0x0256,0x98
0x98,0x0257,0x0F
0x98,0x0258,0x00
0x98,0x0259,0x01
0x98,0x025A,0x18
0x98,0x025B,0x08
0x98,0x025C,0x70
0x98,0x0241,0x20
0x98,0x025D,0x04
0x98,0x01FC,0x00
0x98,0x0038,0x01
0x98,0x0240,0Xf3
```

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 are annotated to give a broad picture of the requirements users can expect to get a system working with the MAX96716A/MAX96716F deserializer and MAX96717 serializers.

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

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 is 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

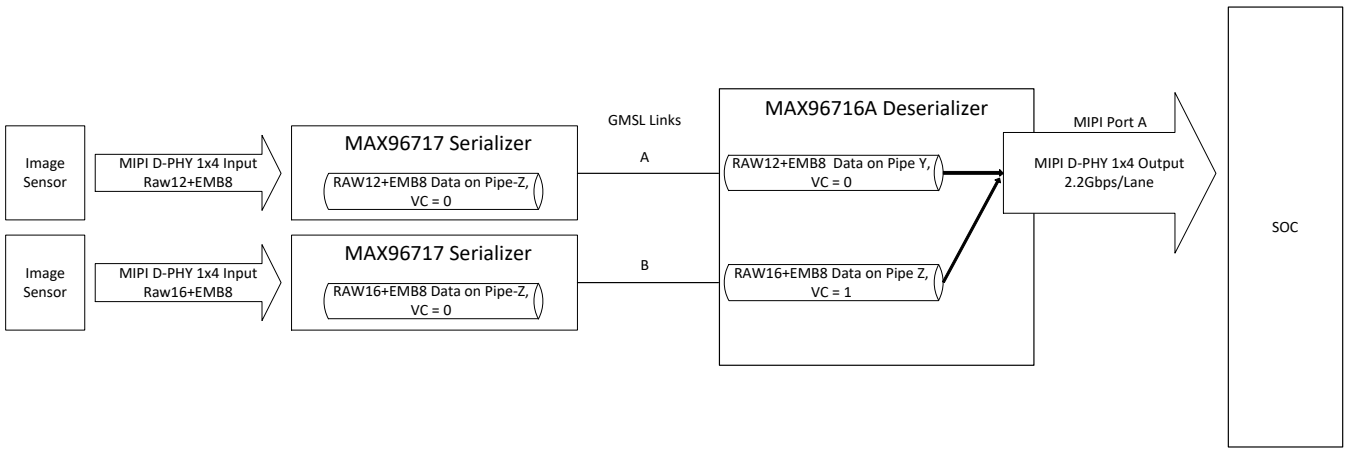

Figure 31. Use Case Example #1

Use Case Example #1 GUI Script

```
// GMSL-A / Serializer: MAX96717 (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: MAX96717 (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: MAX96716A / 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 MAX96717
//
// 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 MAX96717
//
// 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 MAX96716A

// 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

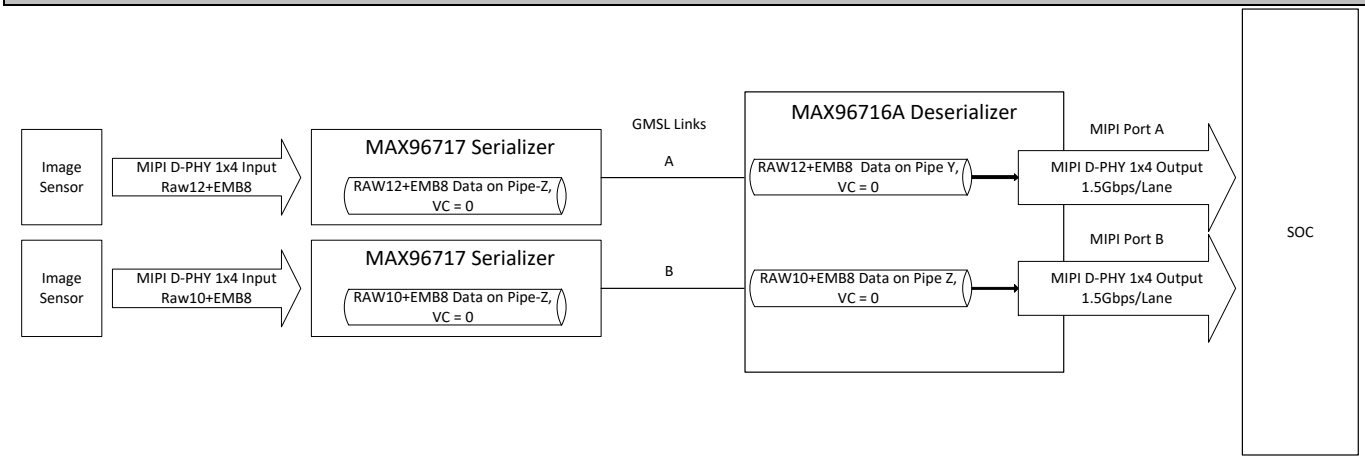

Figure 32. Use Case Example #2

Use Case Example #2 GUI Script

// GMSL-A / Serializer: MAX96717 (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: MAX96717 (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: MAX96716A / 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 MAX96717
//
// 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 MAX96717
//
// 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 MAX96716A

// 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
```

Appendix

List of Figures

List of Tables

Revision History

Revision	Changes	Date
0	Initial Revision	11/23