AHEAD OF WHAT'S POSSIBLE™

MAX9295D User Guide

GMSL SerDes Applications Team

Version 0

January, 2024

Table of Contents

Table of Contents 2
MAX9295D Serializer 3
Start-up and Programming Sequence 4
Configuration 6
Software Override 15
I^2C Control Channels 19
UART Control Channels 27
Serial Peripheral Interface (SPI) 30
Frame Synchronization (FSYNC) 36
Power Manager and Sleep Mode 39
Bandwidth Efficiency Optimization 44
Error Flags 46
General-Purpose Input and Output (GPIO) 48
Video Pattern Generator (VPG) 56
Pairing with GMSL1 Deserializers 60
Complete Use Case Programming Examples 67
Appendix 71
Revision History 73

MAX9295D Serializer

Device Overview

This user guide is intended for use in conjunction with other documents such as the MAX9295D 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. Within the examples, the serializer I^2C address can be assumed to be 0x80, unless otherwise noted.

Examples may be shown without errata writes necessary to ensure reliable operation in production. Be sure to contact the Analog Devices, Inc. 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 MAX9295D device for testing.

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. MAX9295D is capable of 3 Gbps or 6 Gbps forward link rate (selectable with resistors connected to the CFG pin or with register writes) and a 187.5 Mbps reverse direction rate.

Application Use Case

In a typical configuration, the MAX9295D is used to support cameras in the 1 MP to 8 MP range. Typically, each camera's image sensor feeds the video into the D-Phy CSI input port of the MAX9295D serializer. The serializer then takes that data, converts it to GMSL™, and sends it out over the link to a deserializer. In *Figure 1*, there is one MAX9295D operating and sending data from two cameras out to a deserializer. Coax or shielded twisted-pair (STP) cables can be used for the GMSL link.

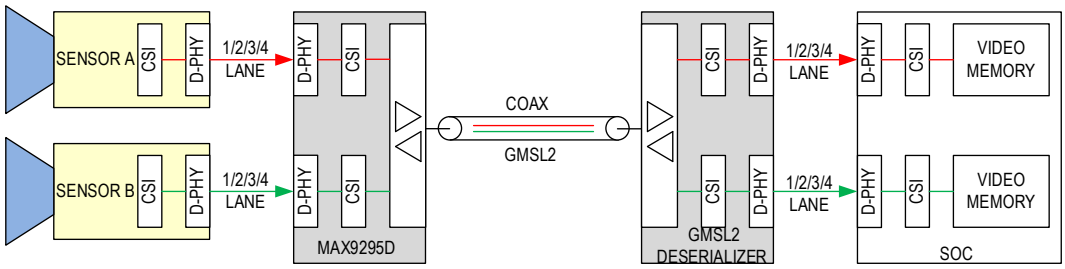

Figure 1. MAX9295D Two-Camera Application Example

Start-up and Programming Sequence

Overview

GMSL2 devices have many applications use cases and features that work in conjunction with each other. To avoid feature and system sequencing issues, *Table 1* 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 1. MAX9295D Start-Up Sequence

MAX9295D Startup 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 register map for correct registers to set up correct link operation mode. Refer to the deserializer user guide link initialization section for more information.
5	GMSL Link Lock is Established	If GMSL Link Lock is not established, verify the following: 1) Voltage rails are correct per DS specification. 2) Datarate, Coax/STP mode, and GMSL settings match between serializer and deserializer
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 (>1.5 Gbps/Lane)

		-MIPI Data Rate
11	DES GPIO and Other Feature Configurations	-GPIO Forwarding -FSYNC -I^2C/UART Pass-through Channels -Line Fault
12	DES Interrupt Handling (ERRB) and ASIL Configuration	Refer to 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 (>1.5 Gbps/Lane) -MIPI Continuous vs. Non-Continuous Mode
15	SER GPIO and Other Feature Configurations	-GPIO Forwarding -FSYNC - I^2C/UART Pass-through Channels -Reference Clock Out -ADC -Line Fault
16	SER Interrupt Handling (ERRB) and ASIL Configuration	Refer to the 'Error Flags' section and 'Safety Documents' of the serializers 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 relocks. 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. Refer to the deserializer user guide 'Register CRC' section for more information.
21	Enable Serializer Register CRC Safety Mechanism (If Necessary)	Review safety documents of the serializer for more information. Register CRC is not available with MAX9295D.
22	Start Video Source	

Notes:
1. Perform any configuration changes before enabling video source.
2. If changes are needed after video has started; stop the video, make changes, and restart video.
3. Dynamic configuration is not supported.

Configuration

Overview

The forward video path of the MAX9295D serializer is configured with the following programming:

- Link Initialization
- MIPI PHY Settings
- Video Pipes and Datatype (DT)/Virtual Channel (VC) Filtering

Only after the video path is configured should video be enabled; 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.

Link Initialization

Link initialization establishes the device link modes and link speeds. The MAX9295D is a GMSL2/GMSL1, dual-port serializer that can support coax or shielded-twisted pair (STP) cables. The MAX9295D can transmit at 3 Gbps or 6 Gbps in the forward direction over the GMSL link. Using the following registers, the GMSL link rate and COAX or STP cabling may be selected. 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 (*Table 2*). The selected configuration becomes the new default on power-up once the CFG pins are set and the part is power cycled.

Table 2. Basic Settings

CFG1 Value	Coax/ STP	HIM/Data Rate	GMSL1/GMSL2
0	COAX	6 Gbps	GMSL2
1	COAX	HIM Enabled	GMSL1
2	COAX	HIM Disabled	GMSL1
3	STP	6 Gbps	GMSL1
4	STP	3 Gbps	GMSL1
5	STP	HIM Enabled	GMSL1
6	STP	HIM Enabled	GMSL1
7	COAX	6 Gbps	GMSL2

Link Initialization Registers

Table 3. Link Initialization Registers

Register	Bitfield Name	Bits	Default Value	Decode
0x0001	TX_RATE[1:0]	3:2	0b10 (dependent on CFG pins)	01 = 3 Gbps 10 = 6 Gbps
0x0011	CXTP_A	0	0b1	0 = Shielded twisted pair drive 1 = Coax drive
0x0010	RESET_ALL	7	0b0	0 = no action 1 = activate chip reset (PWDNB)
0x0010	RESET_LINK	6	0b0	0 = release link reset 1 = activate link reset
0x0010	RESET_ONESHOT	5	0b0	0 = no action 1 = activate oneshot reset on link (bit self clears)

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.

Link Lock Check

If the device configuration is correct, the link automatically locks upon connection. Pin #22 (MFP9) is used as LOCK indication by default. Bit 3 in register 0x0013 asserts if the link is locked.

MIPI PHY Settings

The MIPI PHY settings must be programmed so that the serializer expects the appropriate number of lanes, clock type, desired lane swapping, lane polarities, etc. The MAX9295D is a dual MIPI ports device. Each port has one controller connected to two MIPI PHYs that can support up to 4-lane inputs (*Figure 2*).

In an example PHY configuration, 1x4 mode, the two MIPI PHYs (PHY1 and PHY2) are combined to establish the 4-lane configuration. By default, PHY 2 is the master PHY providing the MIPI clock for port B. So, if only two lanes are used, only PHY2 is sending data.

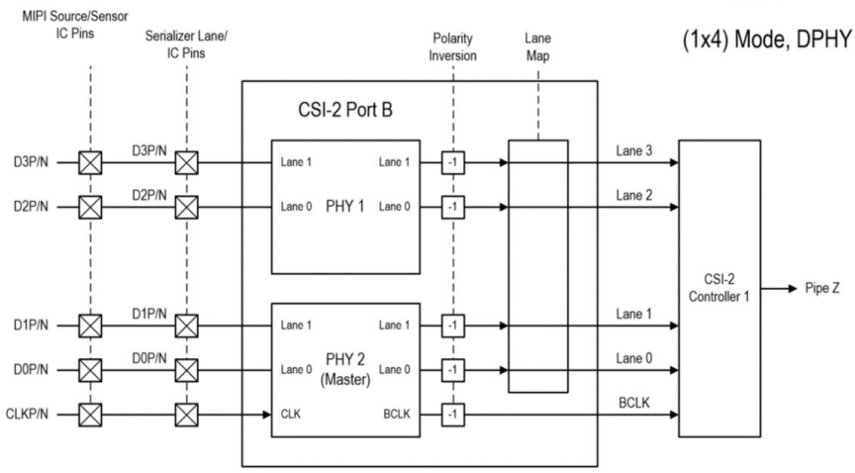

Figure 2. Inputs, PHY, and Controller Diagram (MIPI Port B Only)

VC reassignment is supported. VC assignments can be altered through the CTRL0_VC_MAPand CTRL1_VC_MAP registers.

MIPI Lane Swap

Lane swapping is available for pins on the same MIPI port. Any of the data pins can be interchanged among each other, not including the clock pins. If a customer's data pins are not aligned properly on their layout, they can just swap the pins through the phy{1,2}_lane_map registers instead of fabricating a new board. See *Table 4* for the register description.

Lane Swap Example

The data pins can be swapped within each port, but the clock location is fixed. For example, 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 support polarity inversion (phy{1,2}_pol_map). *Figure 3* shows an example where all four lanes are being swapped.

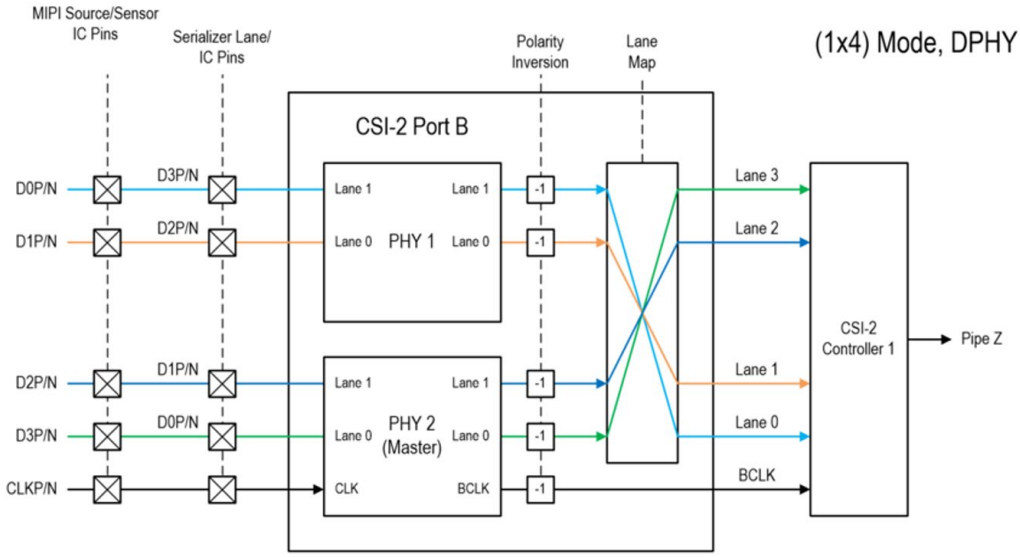

Figure 3. D-PHY Lane Swap Example (MIPI Port B Only)

Lane Swap Programming Example

This example programs the lane swapping as shown in *Figure 3*.

```
# Set lane mapping for all 4 lanes on ctrl 1. This is written to completely swap
the device pinout from default as shown in the figure.
0x80,0x0332,0x10 #D0 mapped to D3, D1 mapped to D2
0x80,0x0333,0x0B #D2 mapped to D1, D3 mapped to D0
```

MIPI Deskew

The MIPI interface can be configured to use interlane deskew using deskew patterns from the transmitter, but this is only recommended when the bit transmission rate is 1.5 Gbps/lane and above. Deskew is optional for data rates lower than 1.5 Gbps/lane. Deskew is initiated by the transmitter under CSI-PPI control. The

MAX9295D only has one bit for enabling the deskew calibration (ctrl1_deskewen), which can be found in REG 0x331 (see *Table 4*).

MIPI PHY Settings Registers

Table 4. MIPI PHY Settings Registers

Register	Bitfield Name	Bits	Default Value	Description
0x0330	ctrl1_vc_map_en	5	0b0	0 = disable virtual channel mapping 1 = enable virtual channel mapping
0x0330	mipi_rx_reset	3	0b0	0 = do not reset MIPI receiver 1 = reset MIPI receiver (This bit should be toggled HIGH and then LOW before any video is received – per errata).
0x0330	phy_config[2:0]	2:0	0b000	0 = 1x4 (only available option)
0x0331	ctrl1_vcx_en	7	0b0	0 = extended virtual channel disabled 1 = extended virtual channel enabled
0x0331	ctrl1_deskewen	6	0b0	0 = deskew calibration disabled 1 = deskew calibration enabled
0x033C, 0x033E	phy{1,2}_hs_err[7:6]	7:6	0b0000	Bit 7 represents lane 0, Bit 6 represents lane 1 0 = Deskew calibration pattern flag not received 1 = Deskew calibration pattern flag received
0x033C, 0x033E	phy{1,2}_hs_err[5:4]	5:4	0b0000	Bit 5 represents lane 0, Bit 4 represents lane 1 0 = Default 1 = Deskew calibration failure
0x0331	ctrl1_num_lanes[1:0]	5:4	0b11	00 = 1 data lane 01 = 2 data lanes 10 = 3 data lanes 11 = 4 data lanes
0x0332	phy1_lane_map[3:2]	7:6	0b11	00 = map lane0 to lane3 01 = map lane1 to lane3 10 = map lane2 to lane3 11 = map lane3 to lane3
0x0332	phy1_lane_map[1:0]	5:4	0b10	00 = map lane0 to lane2 01 = map lane1 to lane2 10 = map lane2 to lane2 11 = map lane3 to lane2
0x0333	phy2_lane_map[3:2]	3:2	0b01	00 = map lane0 to lane1 01 = map lane1 to lane1 10 = map lane2 to lane1 11 = map lane3 to lane1
0x0333	phy2_lane_map[1:0]	1:0	0b00	00 = map lane0 to lane0 01 = map lane1 to lane0 10 = map lane2 to lane0 11 = map lane3 to lane0
0x0334	phy0_pol_map[2]	2	0b0	0 = normal polarity for clock lane 1 = inverse polarity for clock lane
0x0334	phy0_pol_map[1]	1	0b0	0 = normal polarity for data lane 3 1 = inverse polarity for data lane 3
0x0334	phy0_pol_map[0]	0	0b0	0 = normal polarity for data lane 2 1 = inverse polarity for data lane 2

0x0334	phy1_pol_map[2]	6	0b0	0 = normal polarity for clock lane 1 = inverse polarity for clock lane
0x0334	phy1_pol_map[1]	5	0b0	0 = normal polarity for data lane 3 1 = inverse polarity for data lane 3
0x0334	phy1_pol_map[0]	4	0b0	0 = normal polarity for data lane 3 1 = inverse polarity for data lane 3
0x0335	phy2_pol_map[2]	2	0b0	0 = normal polarity for clock lane 1 = inverse polarity for clock lane
0x0335	phy2_pol_map[1]	1	0b0	0 = normal polarity for data lane 1 1 = inverse polarity for data lane 1
0x0335	phy2_pol_map[0]	0	0b0	0 = normal polarity for data lane 0 1 = inverse polarity for data lane 0
0x0335	phy3_pol_map[2]	6	0b0	0 = normal polarity for clock lane 1 = inverse polarity for clock lane
0x0335	phy3_pol_map[1]	5	0b0	0 = normal polarity for data lane 1 1 = inverse polarity for data lane 1
0x0335	phy3_pol_map[0]	4	0b0	0 = normal polarity for data lane 1 1 = inverse polarity for data lane 1
0x0345-0x0347, 0x036C-0x036F, 0x0377-0x037F	ctrl1_vc_map{0,15}	7:4	0b0000	Virtual channel reassignment registers. Description found in last paragraph above table.

Video Pipes and Datatype/Virtual Channel Filtering

The MAX9295D has dual CSI-2 input ports and four total video pipes (X, Y, Z, U). Register 0x0308, shown in *Table 5*, contains the bits for enabling/disabling MIPI ports.

Default Mapping = MIPI Port A (PHY0/1) → MIPI Controller 0 → Video Pipe X
Default Mapping = MIPI Port B (PHY2/3) → MIPI Controller 1 → Video Pipe Z

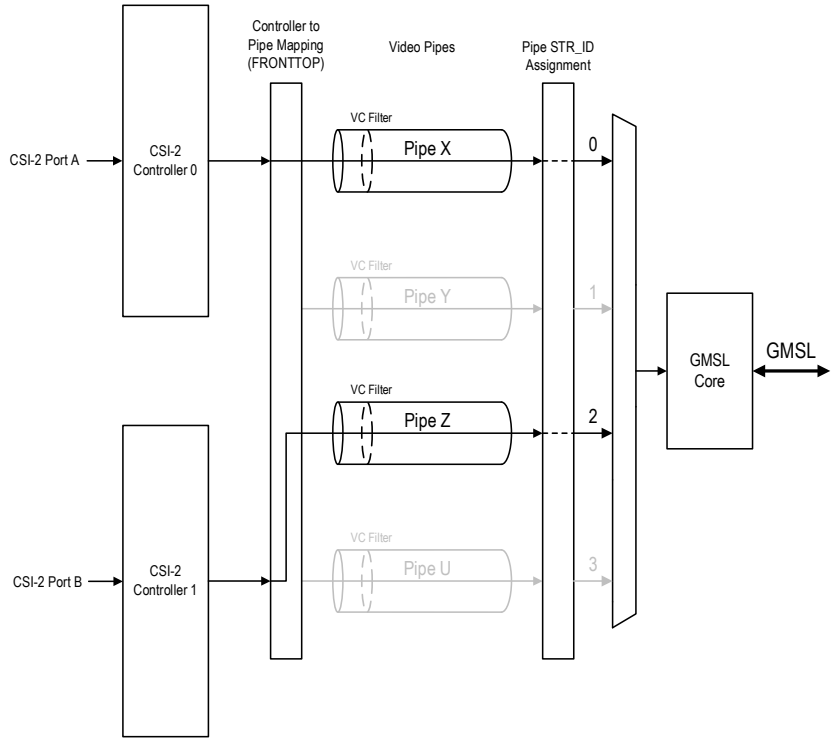

Figure 4. Default Controller to Pipe Mapping

Using the default video routing, any data received on the MIPI input port (A) is automatically routed onto video pipe X, any data received on the MIPI input port (B) is automatically routed onto video pipe Z, unless filtering through datatype (DT) or virtual channel (VC) is being used. Every CSI-2 packet includes a header that indicates the DT and VC. This information can be used to route the incoming data throughout the serial link system.

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

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

Stream IDs

After data is transmitted through the video pipe, a stream ID is assigned prior to transmission across the serial link. The stream ID is then used by the deserializer to determine video pipe routing. All video pipes have a dedicated stream ID set with the TX_STR_SEL[1:0] bitfield.

Note: Stream IDs must be set in the serializer to avoid data Rx conflicts in the deserializer when using 'Reverse Splitter' mode (that is, multiple serializers connected to a single deserializer).

Video Pipe and DT/VC Filtering Registers

Table 5. Video Pipe and Filtering Registers

Register	Bitfield Name	Bits	Default Value	Decode
0x0308	START_PORTA START_PORTB	4 5	0b1	0 = CSI on portdisabled 1 = CSI on portenabled
0x0308	CLK_SELX CLK_SELY CLK_SELZ CLK_SELU	0 1 2 3	0b1	0 = Port A selected for the pipe 1 = Port B selected for the pipe
0x0002	VID_TX_EN_X VID_TX_EN_Y VID_TX_EN_Z VID_TX_EN_U	4 5 6 7	0b1	0 = Video transmit on the pipe disabled 1 = Video transmit on the pipe enabled
0x005B	TX_STR_SEL[1:0]	1:0	0b10	00 = Stream ID for pipe is 0 01 = Stream ID for pipe is 1 10 = Stream ID for pipe is 2 11 = Stream ID for pipe is 3
0x0318, 0x0319, 0x03DC, 0x03DD	mem_dt{1,2,7,8}_selz[6]	6	0b0	0 = Datatype filtering disabled 1 = Datatype filtering enabled
0x0318, 0x0319, 0x03DC, 0x03DD	mem_dt{1,2,7,8}_selz[5:0]	5:0	0b000000	The value of bits 5:0 in this register should equal the data type ID of the data type to allow onto the video pipe (example, RAW12 = 0x2C).
0x03C8, 0x03C9	mem_dt{3,4}_selz[7:6]	7:6	0b00	These two bits select the two LSBs of the virtual channel to be filtered onto the video pipe.
0x03C8, 0x03C9	mem_dt{3,4}_selz[5:0]	5:0	0b000000	The value of bits 5:0 in this register should equal the data type ID of the data type to allow onto the video pipe.
0x03D1	mem_dt{3,4}_selz_en	1:0	0b00	0 = Disable filtering set in registers 0x3C8, 0x3C9 1 = Enable filtering set in registers 0x3C8, 0x3C9
0x0309 **0x030B** 0x030D 0x0310	VC_SELX_L VC_SELY_L VC_SELZ_L VC_SELU_L	7:0	0xFF	Bits 0 to 7 represent VC0 to VC7, respectively. If a bit is high, it means that VC is allowed onto the video pipe (example, if only bits 0 and 2 are HIGH, then only VC0 and VC2 are accepted).
0x030A 0x030C 0x030E 0x0310	VC_SELZ_H VC_SELZ_H VC_SELZ_H VC_SELZ_H	7:0	0xFF	This register works the same as register 0x30D, except bits 0 to 7 represent VC8 to VC15, respectively.

Video Pipe Filtering Programming Example

This example filters video pipe Z for RAW12 datatype and virtual channel 1

```
0x80, 0x0318, 0x6C, #enable DT filter for RAW12 (ID = 0x2C)
0x80, 0x030D, 0x02, #only VC1 allowed onto pipe
```

Limit Heartbeat Mode

By default, all GMSL2 serializers send heartbeat packets during blanking intervals. Heartbeat packets are GMSL packets that only contain low-frequency HVD signals (HS, VS, DE). Sending these packets during blanking ensures that the video clock regeneration in the receiver can properly track the number of pixel clocks that should be generated at the output.

The combination of video payload and heartbeat packets at the same time may exceed the GMSL maximum allowable payload, and heartbeat mode may need to be disabled. To ensure GMSL maximum payload is not exceeded, contact the Analog Devices applications team to verify the use case before disabling heartbeat.

Heartbeat mode settings must match between the serializer and deserializer. Heartbeat mode can be disabled for all video pipes (if needed) by setting LIM_HEART = 1 in the serializer.

Note: Heartbeat mode must be enabled when using a non-CSI-2 deserializer.

Table 6. Heartbeat Disable Register

Register Address (Video Pipe)	Bitfield Name	Bits	Default Value	Decode
0x102 (Pipe X) 0x10A **(Pipe Y)** **0x112 (Pipe Z)** **0x11A (Pipe U)**	LIM_HEART	2	0b0	0b0: Heartbeat enabled during blanking 0b1: Heartbeat disabled during blanking

Complete Configuration Examples

1. Configuration with DT and VC filtering:
    a. Only a 2-lane, RAW12 input assigned as VC1 is allowed onto the video pipe.
    b. Lane swap and polarity inversion used to remap the data pins.
    c. Limit heartbeat mode is disabled to save bandwidth.
    d. Data is transmitted over the GMSL link at 3 Gbps with a stream ID of 1.

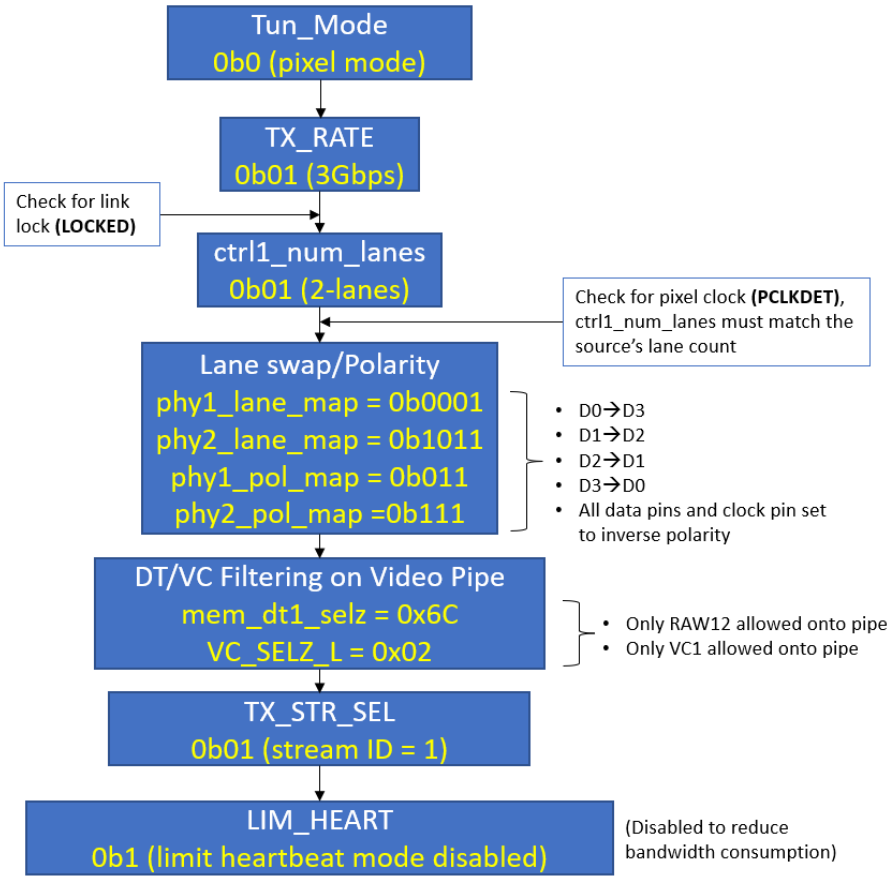

Figure 2. Complex Configuration Example

Software Override

Overview

The software override is used to manually override the video data type (DT) (that is, packet header), virtual channel number (VC), or bits per pixel (BPP). This operation affects the specification of the video data between the video pipe and MIPI controller. Overriding the DT and VC information is used for easier MIPI controller mapping on the deserializer side. See *Table 7* for a list of software override registers.

Note: VC can be changed individually. However, DT and BPP must be adjusted together to ensure settings compatibility. A specific DT could have a range of BPP values depending on if doubling or zero padding is used.

Here are some examples of software override settings for pipe Z:

- DT: soft_dtz[5:0]
  DT = 0x24 = 0b100100 for RGB888
- VC: soft_vcz[1:0]
  VC = 0x03 = 0b11 for VC3
- BPP: soft_bppz[4:0]
  BPP = 0xC = 0b01100 for RAW12

Software Override Registers

Table 1. Software Override Registers

Register	Bitfield Name	Bits	Default Value	Decode
0x031C (Pipe X) 0x031D (Pipe Y) 0x031E (Pipe Z) 0x031F (Pipe U)	soft_dtx_en soft_dty_en soft_dtz_en soft_dtu_en	7	0b0	0 = Data type software override disabled on the pipe 1 = Data type software override enabled on the pipe
0x031C (Pipe X) 0x031D (Pipe Y) 0x031E (Pipe Z) 0x031F (Pipe U)	soft_vcx_en soft_vcy_en soft_vcz_en soft_vcu_en	6	0b0	0 = Virtual channel software override disabled on the pipe 1 = Virtual channel software override enabled on the pipe
0x031C (Pipe X) 0x031D (Pipe Y) 0x031E (Pipe Z) 0x031F (Pipe U)	Soft_bppx_en soft_bppy_en soft_bppz_en soft_bppu_en	5	0b0	0 = BPP software override disabled on the pipe 1 = BPP software override enabled on the pipe

0x031C (Pipe X) **0x031D (Pipe Y)** **0x031E (Pipe Z)** 0x031F (Pipe U)	Soft_bppx[4:0] soft_bppy[4:0] soft_bppz[4:0] soft_bppu[4:0]	4:0	0b11000	These bits should be set to the smallest input BPP (before padding and after doubling).
0x0320	soft_vcx[1:0] soft_vcy[1:0] soft_vcz[1:0] soft_vcu[1:0]	1:0 3:2 5:4 7:6	0b00	00 = VC0 01 = VC1 10 = VC2 11 = VC3
0x0321 0x0322 0x0323 0x0324	soft_dtx[5:0] soft_dty[5:0] soft_dtz[5:0] soft_dtu[5:0]	5:0	0b110000	These bits should be set to the appropriate data type ID.

Input BPP Manipulation

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

Double Mode

Double mode is a data arrangement available for data types with BPP = 8, 10, or 12. With double mode enabled, two input pixels are concatenated and processed as a single pixel within the video pipe. This concatenation reduces the internal PCLK and increases the GMSL2 bandwidth efficiency. Double mode is enabled on a BPP basis.

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

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

Note: The connected deserializer must also be set appropriately to revert (undouble) the concatenated pixels to the original BPP.

Table 2. Double Mode Registers

Register	Bitfield Name	Bits	Default Value	Decode
0x312	bpp8dblx bpp8dbly bpp8dblz bpp8dblu	0 1 2 3	0b0	0: Send as 8-bit pixels 1: Send 8-bit pixels as 16-bit pixels
0x313	bpp10dblx bpp10dbly bpp10dblz bpp10dblu	0 1 2 3	0b0	0: Send as 10-bit pixels 1: Send 10-bit pixels as 20-bit pixels
0x313	bpp12dblx	4	0b0	0: Send as 12-bit pixels

	bpp12dbly bpp12dblz bpp12dblu	5 6 7		1: Send 12-bit pixels as 24-bit pixels
0x337	ctrl1_mode_UDT	5	0b0	0: Treat UDP as 8 bits 1: Treat UDP as 24 bits

Zero Padding

Pixel data being received by MAX9295D can be zero padded as it enters a video pipe up to a resulting BPP of 16. With zero padding, an input with multiple BPP rates can be routed through a video pipe if the following conditions are met:

1. 8 ≤ BPP ≤ 16 for all incoming BPP rates routed to the a video pipe.
    a. Zero padding occurs after doubling. The 8 ≤ BPP ≤ 16 requirement applies to the resulting BPP after doubling.
2. Bandwidth is lost proportionally to the amount of zero padding. Some amount of GMSL2/3 bandwidth is dedicated to sending zeros instead of the original CSI-2 data. Ensure system bandwidth requirements can be met using the calculations in the 'GMSL2 User Guide Bandwidth' section.
3. Video pipe's PCLK drift detection must be disabled.

Zero padding applies to all data being routed in the pipe. When enabled, the pipe PCLK is set to the fastest incoming PCLK (smallest BPP) and all data within the pipe is treated as having a pixel width set by the BPP bitfield. To enable zero-padding, set AUTO_BPP = 0, BPP = largest BPP in the pipe (≤16), soft_bpp = smallest BPP in the pipe, and soft_bpp_en = b1. PCLK drift detection must also be disabled using the pipe's DRIFT_DET_EN bit. Using this method, all incoming data types with a BPP < BPP (Register) are zero padded so that all BPP rates within the pipe are equal. See *Table 9* for zero padding registers.
The zero-padded data is automatically recovered correctly on the deserializer based on the DT information that is automatically transmitted to the deserializer. But any DT doubled in the serializer must be undoubled in the deserializer.

Table 9. Zero Padding Registers

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

Double Mode and Zero Padding Example

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

- AUTO_BPP = 0 – Do not set BPP based on CSI-2 header.
- BPP = 0x10 – Force the Pipe BPP to 16 by zero-padding.
- soft_bppz = 0x0C – Must be set to the smallest input BPP (before padding and after doubling).
- soft_bppz_en = 1 – This enables software override of BPP.

- bpp8dblz = 1 – This doubles all incoming BPP = 8 DT’s.
- DRIFT_DET_EN = 0 – PCLK frequency drift detection is disabled for the pipe.

I²C Control Channels

Overview

The MAX9295D features one main 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 port, a 10 µs delay from the write acknowledgement (ACK) to the next transaction is required.

Main I²C Control Channel

The main 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 serializer and deserializer are accessible from whichever side the controller microcontroller resides (for MAX9295D applications, the controller microcontroller typically resides on the deserializer side).

I²C Pass-Through Channel

There are two pass-through I²C channels to send I²C data across the GMSL link. These channels prevent multi-controller conflict. Thus, register access of the serializer/deserializer is not possible on the pass-through I²C channels.

Port Access and Routing

The MFPs in *Table 10* are used for the I²C control and pass-through channels.

Table 3. MFPs for I²C

MFP Pin	Main I²C Function	Other I²C Functions	Default Function	Notes
MFP11	SDA1		GPIO11	
MFP12	SCL21		GPIO12	
MFP15	SDA	SDA2	SDA/RX	SDA (I²C) or RX (UART) functionality determined by CFG0 status on power-up.
MFP16	SCL	SCL2	SCL/TX	SCL (I²C) or TX (UART) functionality determined by CFG0 status on power-up.

On power-up, the device should be set to the I²C mode through the CFG0 latch. The function names in *Table 10* and ensuing I²C sections assume the device is configured for I²C mode.

By default, the main I²C control channel lines are brought out on MFP15 and MFP16 for SDA and SCL, respectively. One can disable the main 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.

Note: A minimum 10 µs delay is required after enabling/disabling the I²C functionality through the DIS_LOCAL_CC and DIS_REM_CC fields in register 0x1.

The user can bring out the first pass-through I²C channel on MFP11 and MFP12 for SDA1 and SCL1, respectively. The second pass-through I²C channel overlaps with MFP pins for the main I²C control channel, as stated above, which is brought out again on MFP15 and MFP16 for SDA2 and SCL2, respectively. As mentioned above, it is possible to only use at most two channels at one time. Both pass-through channels are enabled by setting the fields IIC_1_EN and IIC_2_EN in register 0x1.

I²C Registers

Table 4. MAX9295D I²C Registers

Register	Bits	Default Value	Description
0x0001	7:6	0x08	**I²C Enable Register:** Bit [7]: Enable pass-through I²C Control Channel 2 (SDA2, SCL2) Bit [6]: Enable pass-through I²C Control Channel 1 (SDA1, SCL1)
0x0006	4	0x80	**I²C Selection Register:** Bit [4]: Enables I²C when set to a 1 or UART when set to a 0. Note: This bit is set according to the CFG0 pin value on power-up. Writing to this register is not recommended.

Control Channel Programming Example

This example enables pass-through I²C Channel 1, normally disabled by default.

```
# Enable pass-through I2C Control Channel 1
0x80,0x0001,0x48
```

I²C BroadcastingOverview

When transmitting to a multilink input deserializer, 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 10 µs delay from the write acknowledgement (ACK) to the next transaction is required.

An example of I^2C broadcasting is discussed in the ensuing section. Four equivalent camera modules, including an image sensor and GMSL2 serializer with the same respective addresses, are connected to a GMSL2 quad-deserializer. Each of the camera modules comprise a MAX9295D serializer at the default I^2C address 0x80 and an image sensor at address 0x20.

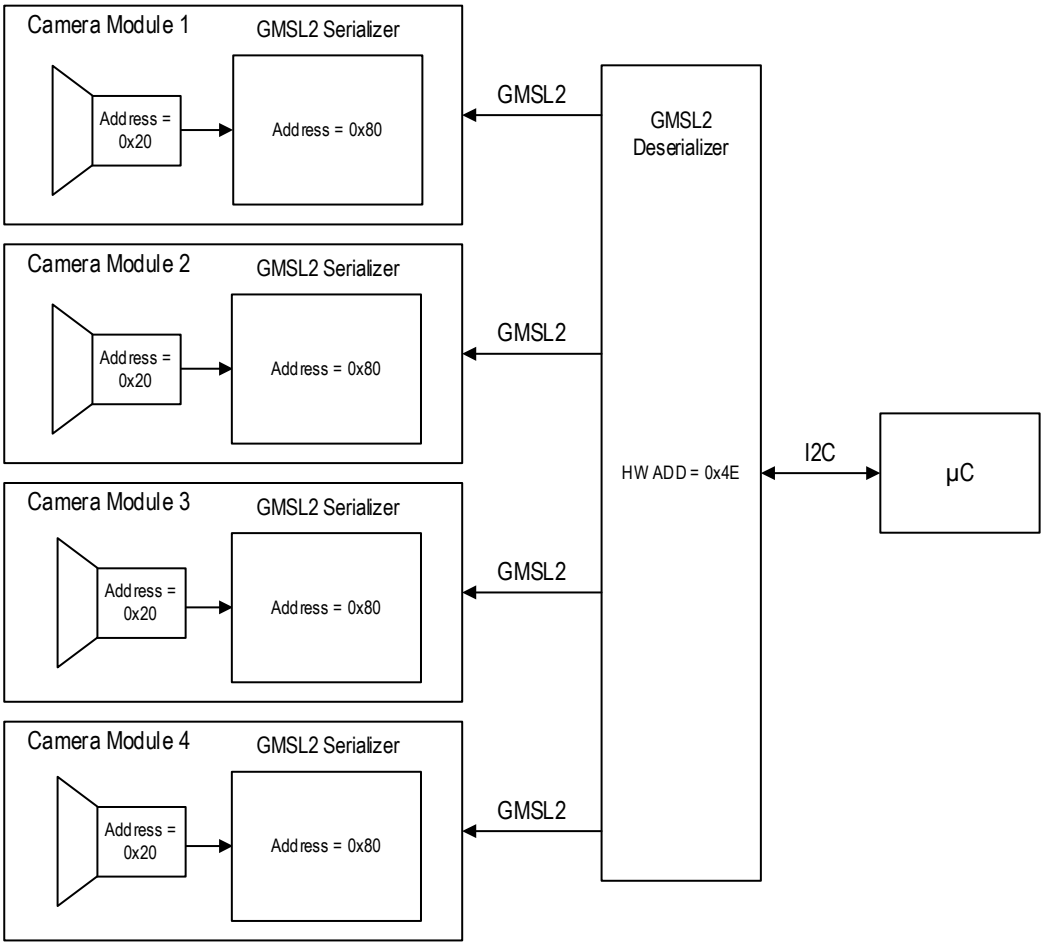

Figure 5. I^2C Interfaced Camera-Module System with Default Address Settings

I^2C Broadcasting Technique

The I^2C broadcasting technique allows to communicate with multiple camera-serializer modules with a single microcontroller, streamlining the transmission process.

The general procedure is to:

- Isolate a single camera/serializer module for remote I^2C access, meaning no other device with the same address should be connected to the I^2C data line.
- Change the serializer address to a unique address.
- Modify the first I^2C 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 I^2C 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 I^2C configuration, such as enabling or disabling an I^2C port, at least a 10µs delay from the write acknowledgement (ACK) to the next transaction is required.

I^2C Broadcasting GMSL2 Use Case Example

The procedure for the I^2C broadcasting example is described as follows.

1) Isolate camera module 1, by enabling deserializer Link A only for remote I^2C access.
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 enabling deserializer Link B only for remote I^2C access.
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) Isolate camera module 3, by enabling deserializer link C only for remote I^2C access.

10) Change the serializer device address in camera module 2 from 0x80 to 0x86. This is done with a register write to DEV_ADDR[6:0], located in REG0.
11) 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 0x86 in the destination register DST_A[6:0]. Thus, for the serializer in camera module 3, anything sent to address 0xC4 is sent to address 0x86 instead.
12) Modify the second translation register in this serializer to give a unique address to the image sensor. Program 0x26 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 3, anything sent to address 0x26 is sent to address 0x20 instead.
13) Isolate camera module 4, by enabling deserializer Link D only for remote I^2C access.
14) Change the serializer device address in camera module 4 from 0x80 to 0x88. This is done with a register write to DEV_ADDR[6:0], located in REG0.
15) 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 0x88 in the destination register DST_A[6:0]. Thus, for the serializer in camera module 4, anything sent to address 0xC4 is sent to address 0x88 instead.
16) Modify the second translation register in this serializer to give a unique address to the image sensor. Program 0x28 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 4, anything sent to address 0x28 is sent to address 0x20 instead.
17) Now enable all the links for remote main I^2C port access.
18) All devices should be present on the I^2C bus. Continue with any additional required system configuration.

Figure 6 shows the same camera module system with translated addresses. *Table 12* and *Table 13* summarize the changes.

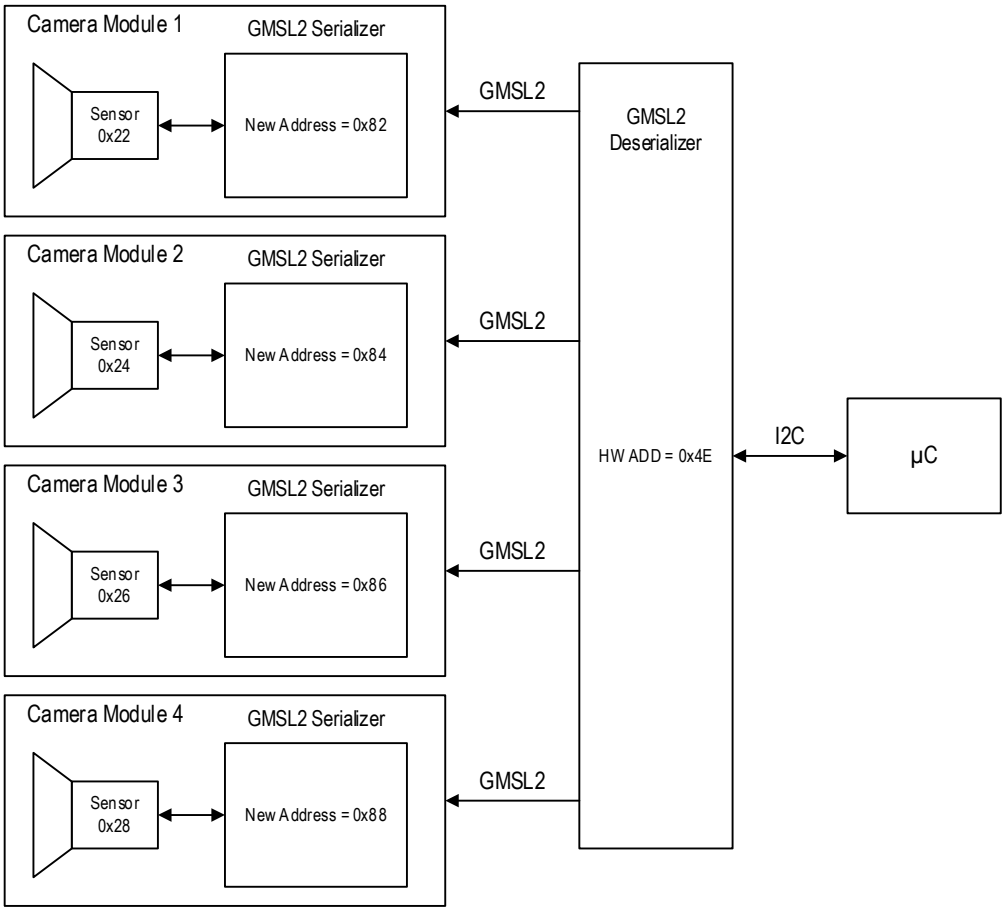

Figure 6. Camera-Module System with Translated Address Settings

The serializers are assigned a single device address to allow writes to all devices as a broadcast.

Table 5. I²C Broadcasting (Quad) Example (Serializer)

I²C Address	SRC_A	DST_A	Sink Device(s)
0x82	0xC4	0x82	Serializer in Camera Module 1
0x84	0xC4	0x84	Serializer in Camera Module 2
0x86	0xC4	0x86	Serializer in Camera Module 3
0x88	0xC4	0x88	Serializer in Camera Module 4

Each image sensor is assigned a unique device address.

Table 6. I²C Broadcasting (Quad) Example (Image Sensor)

I²C Address	SRC_B	DST_B	Sink Device(s)
0x20	0x22	0x20	Image Sensor in Camera Module 1
0x20	0x24	0x20	Image Sensor in Camera Module 2
0x20	0x26	0x20	Image Sensor in Camera Module 3
0x20	0x28	0x20	Image Sensor in Camera Module 4

I²C Broadcasting Programming Examples

This script sets up the I²C broadcasting (*Figure 6*).

```
# Enable Link A remote control channel only
0x4E,0x0003,0xFE
# 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 Link B remote control channel only
0x4E,0x0003,0xFB
# 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 Link C remote control channel only
```

```
0x4E,0x0003,0xEF
# Change I2C address for this Link C serializer
0x80,0x0000,0x86
# Set Ser source to 0xC4
0x86,0x0042,0xC4
# Set Ser destination to 0x86
0x86,0x0043,0x86
# Set Image sensor source to 0x26
0x86,0x0044,0x26
# Set Image sensor destination to 0x20
0x86,0x0045,0x20
# Enable Link D remote control channel only
0x4E,0x0003,0xBF
# Change I2C address for this Link D serializer
0x80,0x0000,0x88
# Set Ser source to 0xC4
0x88,0x0042,0xC4
# Set Ser destination to 0x88
0x88,0x0043,0x88
# Set Image sensor source to 0x28
0x88,0x0044,0x28
# Set Image sensor destination to 0x20
0x88,0x0045,0x20
# Enable All Links A-D remote control channel
0x4E,0x0003,0xAA
```

UART Control Channels

Overview

The MAX9295D features one main universal asynchronous receiver-transmitter (UART) control 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 a UART port, at least a 10 µs delay from the write acknowledgement (ACK) to the next transaction is required.

Main UART Control Channel

The main 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 the serializer and deserializer are accessible from whichever side the controller microcontroller resides (for MAX9295D applications, the controller microcontroller usually resides on the deserializer side).

Base Mode

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

Bypass Mode

In 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 own defined UART protocol. In this mode, the UART commands are still sent over the GMSL2 link. This mode serves to prevent inadvertent programming of the serializer/deserializer registers and can be switched in and out of during normal operation.

Pass-Through UART Channel

There are two pass-through UART channels used 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 main UART control channel in bypass mode (as described above).

Port Access and Routing

The MFPs in *Table 14* are used for the UART control and pass-through channels.

Table 7. MFPs for UART

MFP Pin	Main UART Function	Other UART Functions	Default Function	Notes
MFP10		MS		Mode select (MS) is used as a hard trigger to run main UART channel (TX/RX) in bypass mode.
MFP11		RX1		
MFP12		TX1		
MFP15	RX	RX2		
MFP16	TX	TX2		

On power-up, the device should be set to UART mode through the CFG0 latch. The function names in *Table 14* and ensuing UART sections assume the device is configured for UART mode.

By default, the main UART control channel lines are brought out on MFP15 and MFP16 for RX and TX, respectively. Disable the main control channel's line access by setting field DIS_LOCAL_CC in register 0x1. Also disable access to remote device control by setting field DIS_REM_CC in register 0x1.

Enabling UART Bypass Mode Through Register Setting (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, then 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.

Enabling UART Bypass Mode Through Pin Setting (Hard-Bypass)

UART bypass mode can also be enabled by the mode select (MS) pin, which the user can bring out on MFP10. 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 MFP10. To enable this setting, set the field LOC_MS_EN in register 0x48. This setting might be needed if MFP10 is needed for another use.

Enabling the UART Pass-Through Channels

The user can bring out the first pass-through UART channel on MFP11 and MFP12 for RX1 and TX1, respectively. The second pass-through UART channel overlaps with MFP pins for the main UART control channel, as stated above, which the user can bring out on MFP15 and MFP16 for RX2 and TX2, respectively. As mentioned above, it is possible to only use, at most, two channels at one time. Both pass-through channels are enabled by setting the fields UART_1_EN and UART_2_EN in register 0x3.

Table 8. MAX9295D UART Registers

Register	Bits	Default Value	Description
0x0001	5:4	0x08	**UART Control Channel Enable Register:** Bit [5]: Disable main UART Control Channel connection to TX/RX pins Bit [4]: Disable access to remote device control-channel over GMSL2 connection
0x0003	5:4	0x00	**UART Pass-Through Channel Enable Register:** Bit [5]: Enable pass-through UART Channel 2 Bit [4]: Enable pass-through UART Channel 1
0x0006	4	0x80	**UART Selection Register:** Bit [4]: Enables UART when set to a 0. **Note: This bit is set according to the CFG0 pin value on power-up. Writing to this register is not recommended.**

0x0048	5:0	0x42	**UART Bypass Mode Control Register:** Bit[5]: Enable UART bypass mode control by remote GPIO pin (Function **MS** on **MFP10**) Bit[4]: Enable UART bypass mode control by local GPIO pin (**GPIO2)** Bit[3]: Enable or disable parity bit in bypass mode Bit[2]: UART soft-bypass timeout duration Bit[1]: Enable UART soft-bypass mode
0x004F	7:6 3:2	0x00	**UART Pass-Through Channels Config Register:** Bit[7]: Use standard or custom bit rate Bit[6]: Enable parity bit Bit[3]: Use standard or custom bit rate Bit[2]: Enable parity bit

Enable Pass-Through UART Channel 1

This example enables pass-through UART Channel 1.

```
# Enable pass-through UART Channel 1
0x80,0x0003,0x08
```

Serial Peripheral Interface (SPI)

Overview

SPI is available on the MAX9295D. Unlike I^2C and UART, SPI is never able to modify any registers in either the serializer or deserializer. It is only used to transfer SPI data across the link from serializer to deserializer or vice versa. Typical SPI use cases are to send commands for other devices or to stream data other than video data (example, for sensors). With GMSL, SPI transmission at up to 25 MHz is possible.

SPI with GMSL devices requires more considerations than a normal SPI setup. Unlike a typical direct IC to IC SPI connection, the GMSL link introduces an inherent variable delay. SPI cannot appear immediately at the other end of the link because its transmission across the link must be scheduled with other types of data like video.

Figure 7 shows the GMSL SPI architecture. On each side of the link, the GMSL devices become part of a controller-target pair and have transmit and receive buffers inside.

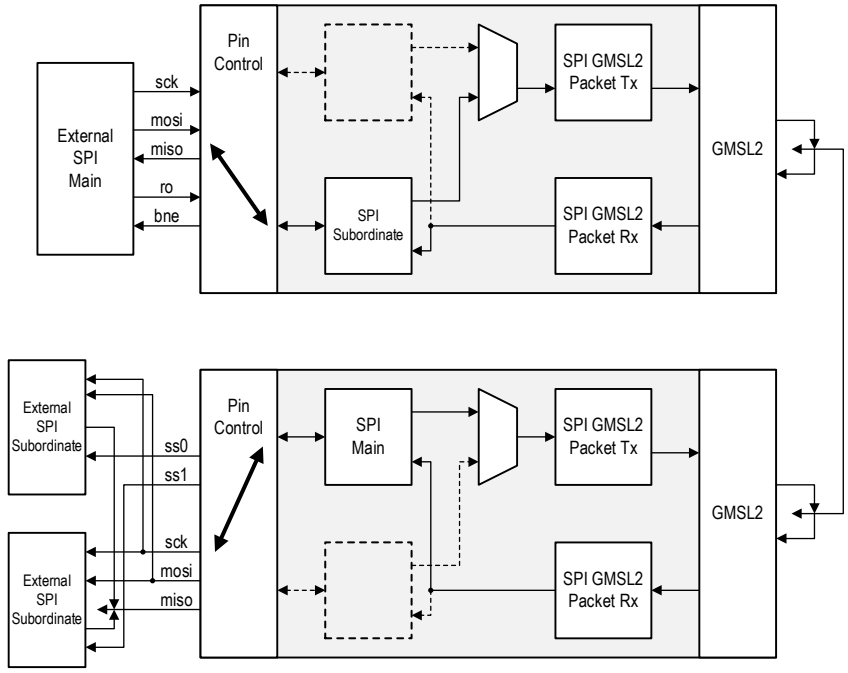

Figure 7. SPI Architecture

CFG Pin Setup

There are important setup considerations when using SPI with GMSL:

- Some MFP pins may have default alternate functions that must be disabled before enabling SPI. If this is not done, the SPI pin may not work correctly. The MAX9295D may be confused if multiple features are enabled at the same time on the MFP pin. Be sure to check the data sheet to verify that each pin has the proper features enabled and disabled. Consider using the MFP status tool in the GMSL GUI to check this.

- If any of the SPI pins are also used as CFG pins, do not let any SPI devices pull the CFG pins up or down until the device powers up and the CFG pins are latched. Power on the part with the SPI external SPI controller/generator not connected, or not pulling on the pins up or down. Otherwise, the part boots into an unwanted configuration.

SPI Setup Registers in MAX9295D

Table 16 shows the most critical setup registers. There are other settings within the 0x170 to 0x178 register range that may be important depending on the arrangement (which is the controller, and which is the target, clock timings, etc.). See the example script in the following section for examples with those registers.

Table 16. Important 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 slave 1 = SPI master
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 example setup script (0x80 is the serializer address, 0x98 is the deserializer address). Note: This example does not apply to the MAX96724, as it does not have SPI.

```
0x98,0x003,0x3  #enable info frames
0x98,0x162,0x0  #select SPI link
0x80,0x170,0x9  #enable SPI, default set to slave, and ignore the SPI header ID
0x80,0x171,0x1D #default, sets SPI packet size and GMSL link scheduler priority
0x80,0x172,0x0  #default, slave select is active low, SPI mode is 0, etc.
0x80,0x173,0x0  #default, delay between assertion of slave select and clock start
0x80,0x174,0x0  #default, SPI clock low time
0x80,0x175,0x0  #default, SPI clock  high time
0x80,0x176,0x3  #enable RO and BNE
0x80,0x178,0x0  #default, timeout delay
0x98,0x170,0xB  #Enable SPI channel, set as master
0x98,0x171,0x1D #GMSL link scheduler priority
0x98,0x172,0x0  #default, slave select is active low, SPI mode is 0, etc.
0x98,0x173,0x1E #default, delay between assertion of slave select and clock start
```

```
0x98,0x174,0x1E #SPI clock low time
0x98,0x175,0x1E #SPI clock high time
0x98,0x176,0xC  #Enable slave select 1 and 2, RO and BNE not enabled
0x98,0x178,0x0  #SPI timeout delay
```

SPI Example Using GMSL GUI and Evaluation Boards

- Make sure that the SPI output is at a slightly faster rate than the SPI input as a general best practice and to prevent buffer overflows. The output rate can be adjusted with register writes.
- Disconnect SPI external controller/generator and RO source from circuit.
- Power up EV boards (make sure V_{DDIO} on the EV board of the SPI external controller/generator side matches the SPI external controller/generator voltage).
- Start GMSL GUI.
- Load GMSL script (.csv).
- Reconnect SPI external controller/generator and RO source.
- Put RO high and write A0 A4. *Explanation: A0 to A3 are for SPI ID selection, mostly useful when using quad deserializers. A4 asserts SS1, A5 asserts SS2, and A6 deasserts both SS1 and SS2.*
- As a best practice, check BNE to ensure the buffer is empty. If BNE is high, there is data in the RX buffer ready to be read by the external SPI controller/generator. Set RO high and write FF until BNE = 0.
- Put RO low and write the normal SPI data. Data can now be streamed.

SPI With and Without Video Running

The MAX9295D's TX and RX buffers have a 32-byte 16-bytes capacity, respectively. *Figure 8* and *Figure 9* show the SPI oscilloscope probes on the SPI clock and data output (the receiving end at the deserializer). The data is flowing consistently without the video running. For 90% of the video, there are some intermittent pauses. The 32-byte buffer compensates for this scheduling delay and makes continuous streaming of the data possible.

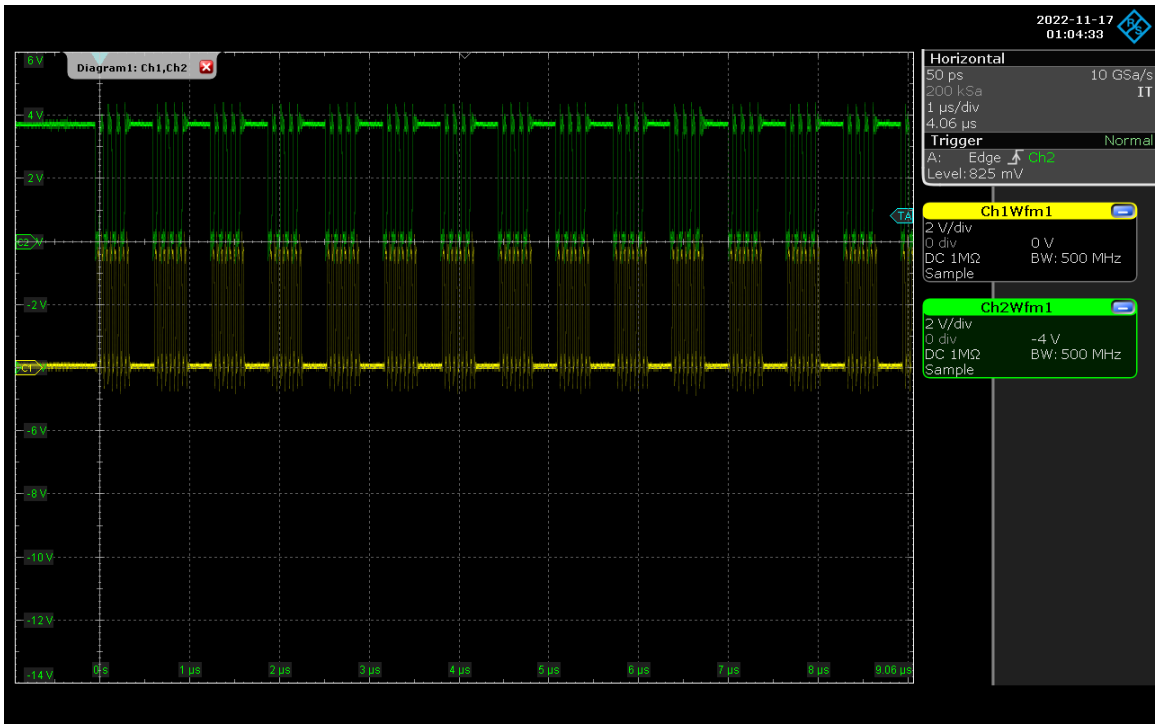

Figure 8. SPI Clock and Data at Final Output (at External SPI Target), No Video on GMSL Link

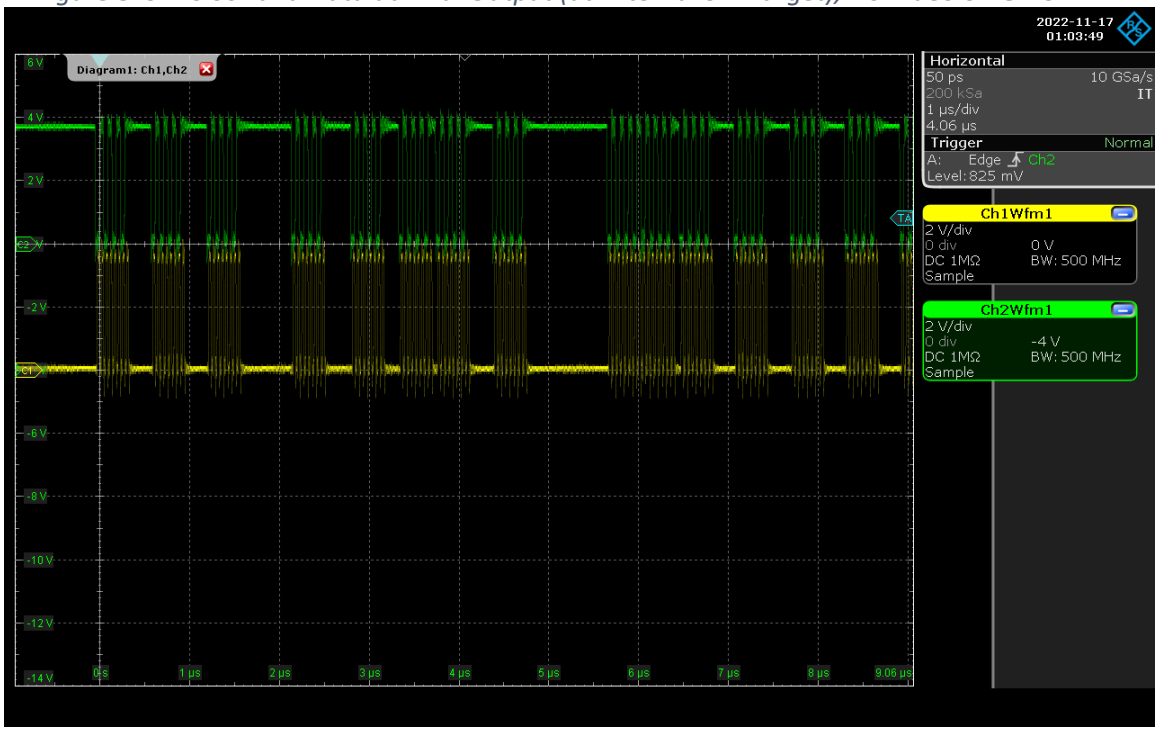

Figure 9. SPI Clock and Data at Final Output (at External SPI Target), 92% Video on GMSL Link

Table 17. Video Details for the Example

Parameter	Value
4-lane rate in Gbps	0.35
Pixels in line	2522
Number of lines	1388
BPP	12
FPS	28
Horizontal blanking (%)	10.0%
UI Period	2.86
Horizontal blanking (ns)	2161.71
Minimum horizontal blanking (ns)	1357
Bits per line	30264
Total line bits including blanking	33290.4
Data rate [Gbps]	1.4
Total line time [ns]	23778.86
% video	0.924142

Data Integrity and Avoiding Buffer Overflow

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

After a byte is sent across the link, the GMSL device on the other side sends the data out on the MFP pins. It also sends the data back across the link so that the SPI external controller/generator can read back the data.

State of read only (RO) pin dictates direction of data movement.

- RO = 0: Data transmitted between controller and target through MOSI.
- RO = 1: Data transmitted between controller and target 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 that it is an output pin; it is, in fact, an input pin toggled externally high or low depending on what operation is desired.

It is recommended to limit the amount of bytes in transit (bytes sent but not received) to 16. The SPI external controller/generator 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 at a time are sent, it is still possible (depending on timing) that all the data is sent properly. But it is not possible to easily be sure that the data is sent properly.

Figure 10 shows an example of sending data in a group of four bytes. This is the timeline of events:

1) RO is pulled high, and the A0 and A4 control commands are entered.
2) RO is pulled low, and data (0x80, 0x04, 0x01, 0x47) is sent from the external SPI controller/generator into the input side (serializer).
3) See bottom half of graph (slight overlap in time with step 2, RO is still low, and the bytes are observed coming out of the deserializer (output side)).

4) RO is high, and the bytes are read back by the external SPI controller/generator. Note BNE is high as the bytes are available (there is a time delay for them to come back).

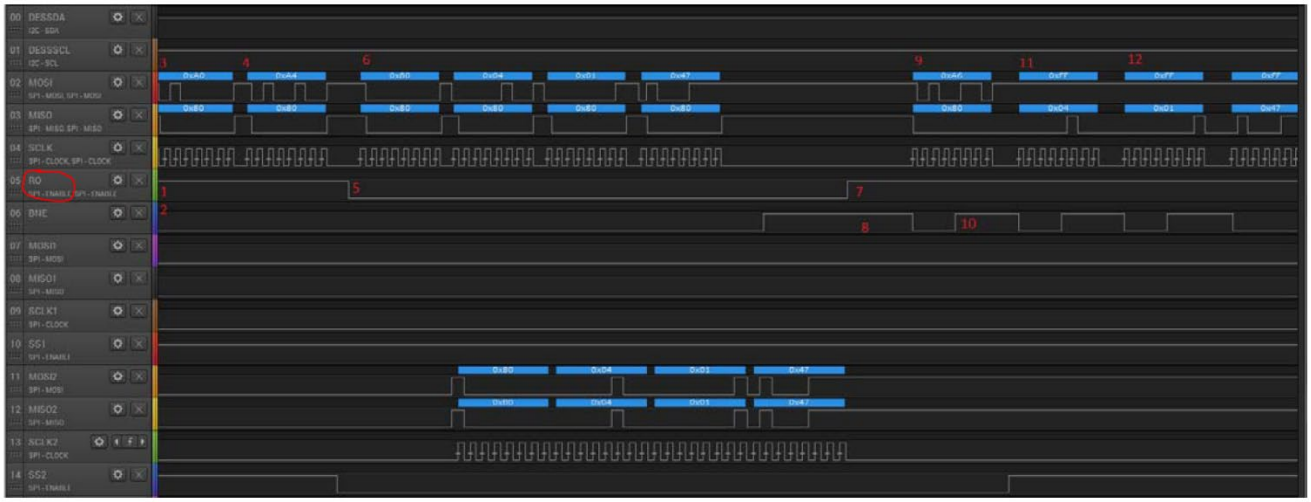

Figure 10. SPI Transmission Example

It is also possible to check overflow buffers. Each buffer has overflow detection logic with status bits that can be read at SPI_TX_OVRFLW and SPI_RX_OVRFLW. It is a good practice to have the external system components monitor these.

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).

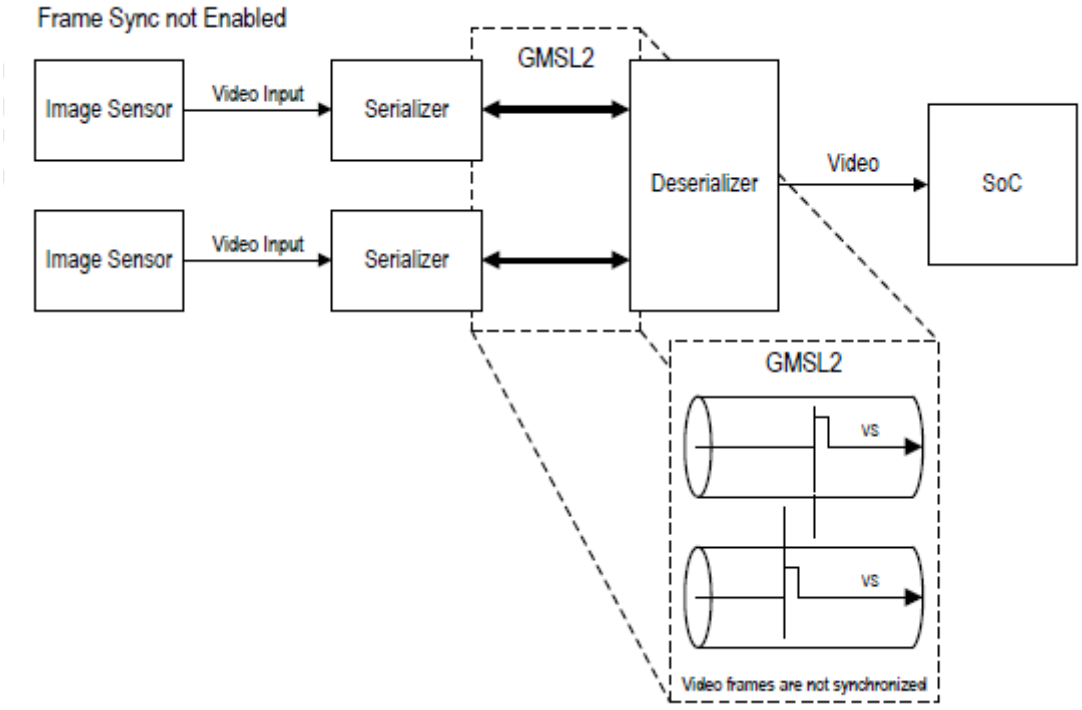

Figure 3. Frame Alignment (Without Frame Sync)

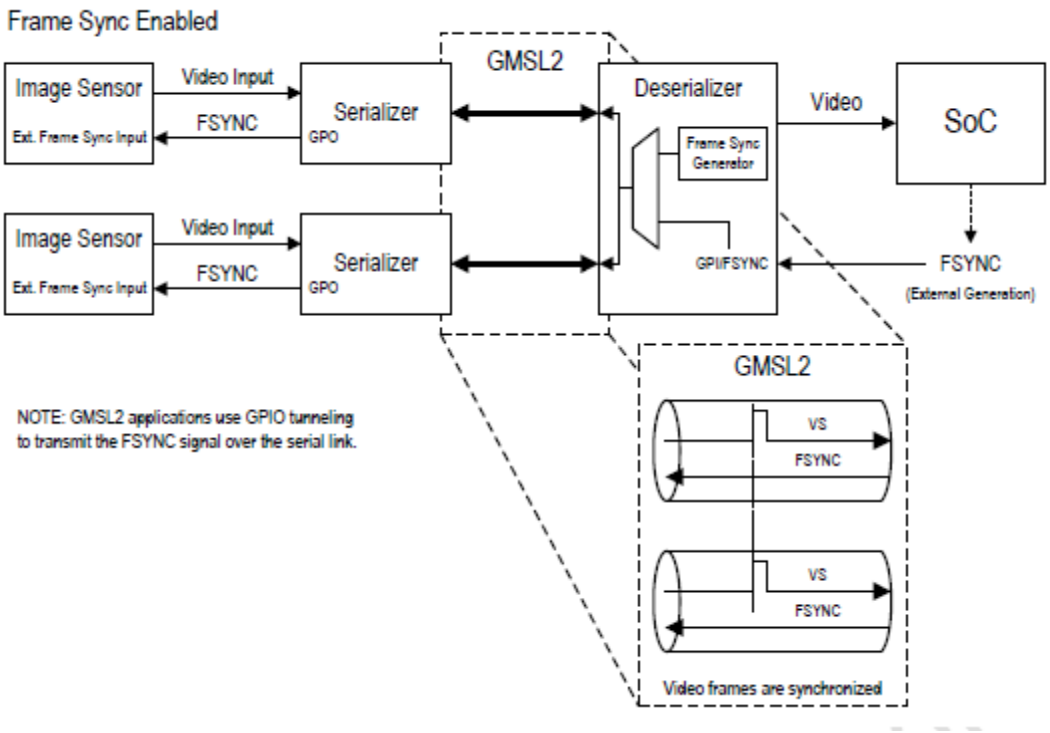

Figure 12. Frame Alignment (Frame Sync Enabled)

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 FSYNC (GMSL2)

The following script configures and enables external FSYNC in GMSL2 mode through GPIO tunneling.

```
# This is GMSL2 Ext. FSYNC example; MAX96724 MFP2/SER MFP1 are used for GPI/GPO.
# Update SER MFP1 RX ID = 1 to match MFP2 of MAX96724
0x80,0x02C3,0x01
# Config SER MFP1 to forward GPIO from the MAX96724. MFP1 is set to be an output.
0x80,0x02C1,0x84
# Config MAX96724 MFP2 to receive external FSYNC signal for each link
0x4E,0x0306,0x83
# Config MAX96724 MFP2 TX ID = 1 for links A-D
0x4E,0x0307,0xA1
0x4E,0x033D,0x21
0x4E,0x0374,0x21
0x4E,0x03AA,0x21
```

Use Case Example

```
# Example FSYNC application using externally generated 30Hz signal from an SOC.
This is then tunneled through the deserializer and two serializers out to the image
sensors.
```

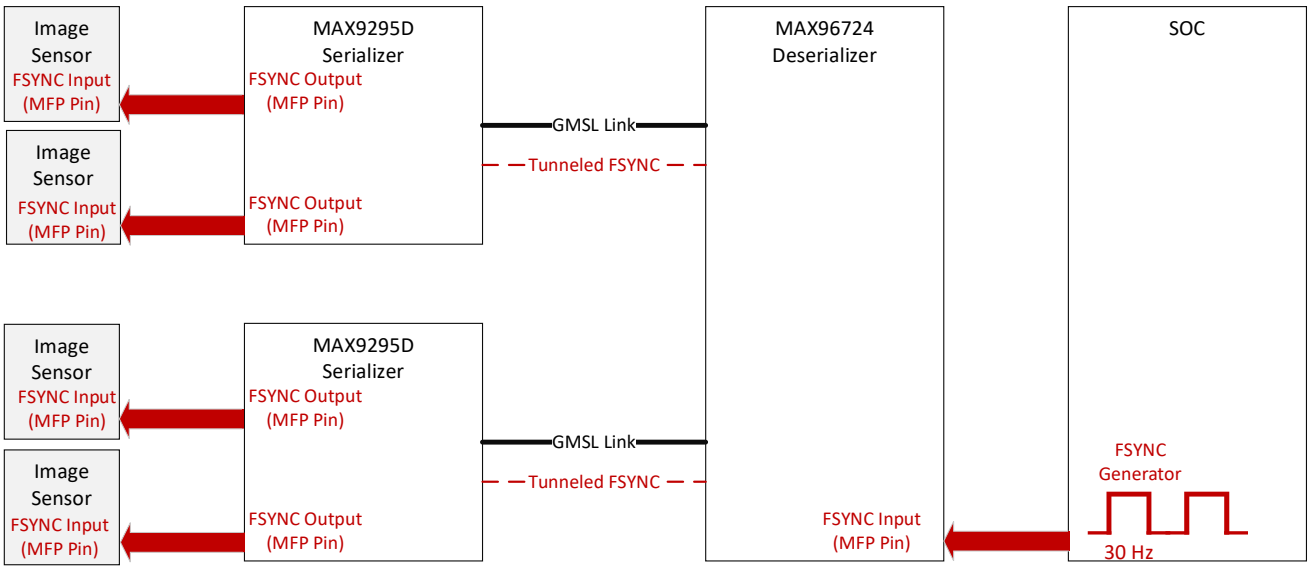

Figure 13. Frame Sync Application Diagram

Power Manager and Sleep Mode

Overview

MAX9295D 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 18. Power Manager and Sleep Mode Availability for MAX9295D Family

Part Number	Power Manager	Sleep Mode
MAX9295D	Supported	Supported

Device Power Operation

The part uses three common power rails (V_{DD}, V_{DD18}, 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 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. See the voltage monitoring section for additional details.

Power Supplies

The MAX9295D shares a common set of power supply voltages that powers universal functions such as the digital core, GMSL link circuitry, and GPIO. The following is a summary of the power supplies:

- **V_{DD}**: The input voltage to the VDD rail can be between 1.0 V and 1.2 V. An internal LDO regulates the voltage to 1.0 V.
- **V_{DD18}**: 1.8 V power rail.
- **V_{DDIO}**: 1.8 V or 3.3 V I/O power rail for I/O.
- **VDD_sw** (CAP_VDD pin): Internal 1 V power rail that powers the digital core logic.

External power is supplied directly to V_{DD}, V_{DD18}, and V_{DDIO}, but the VDD_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 a total of 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 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 14 shows the state diagram for the power manager.

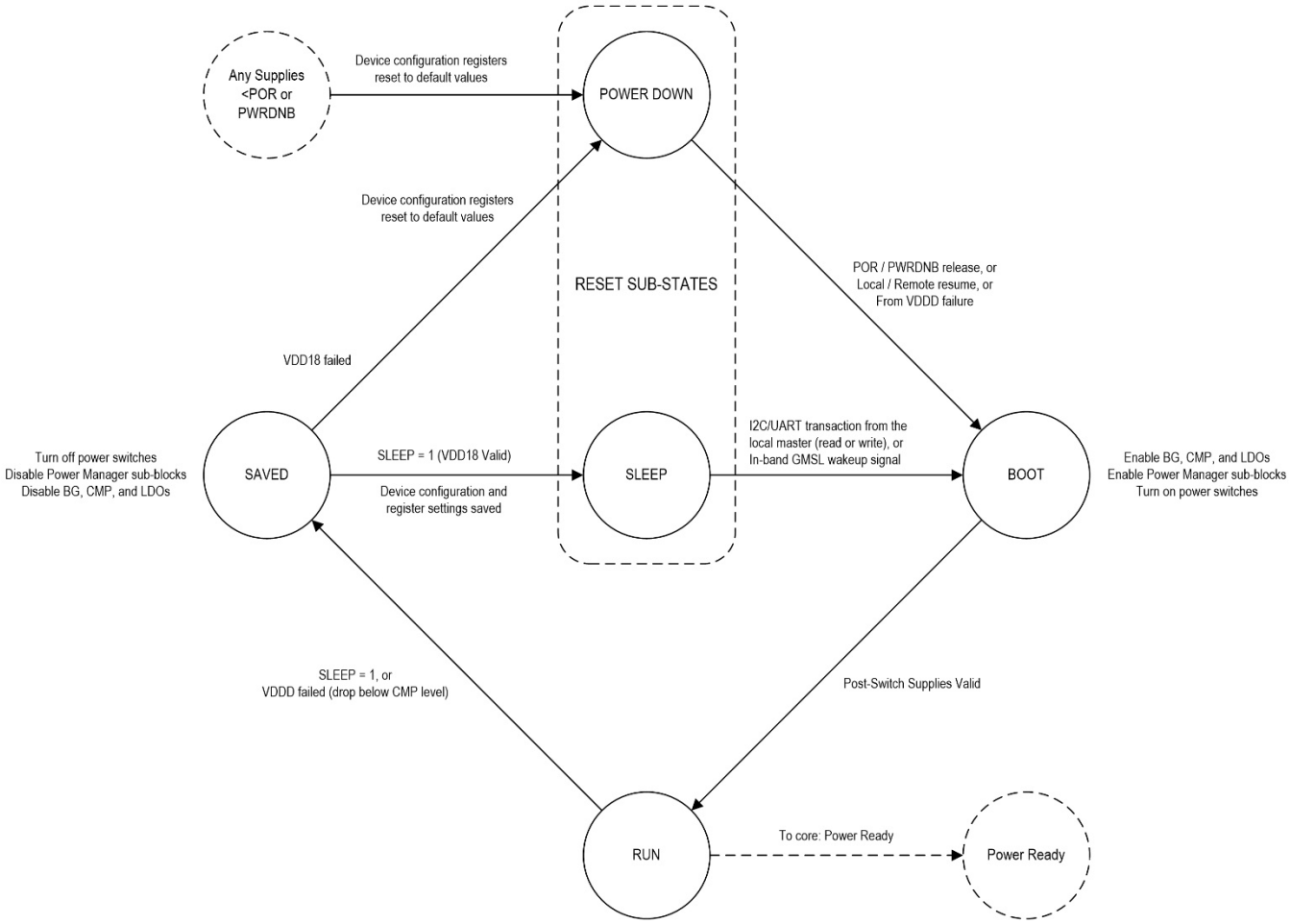

Figure 14. Power Manager State Diagram

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 MAX9295D supports sleep mode, which provides a low power state from which prior configuration information is automatically loaded upon wakeup. This enables fast recovery from low power sleep to full run operation by eliminating the need to reprogram configuration registers 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 8 ms 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

Note: 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 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 stored when the most recent SLEEP command is executed. As a result, the device resets to the state configured before entering sleep mode previously, rather than recovering in a clean power-up default state. The most severe implication of this is that the SLEEP = 1 state is saved in the retention memory. So, the device recovers from reset and immediately enters sleep mode.

Note: 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 presleep device state requires some repeated register configuration following resume from sleep. Registers that are stored in retention memory are marked with “*” in the register map in the data sheet.

Bandwidth Efficiency Optimization

Overview

Before implementing a new design, it is critical to do bandwidth (BW) calculations to verify the right devices and settings are used. If this is not done, it is possible that data is lost or corrupted. Although the MAX9295D family can transmit at 3 Gbps or 6 Gbps 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 19*.

Table 19. 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 the following equations.

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

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

Note: (1) H_{total} and V_{total} must include the horizontal and vertical blanking. (2) Use a BPP of 9 when calculating the BW for 8 BPP data types. 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.

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

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

The majority of GMSL2 serial link bandwidth comprises video transmission. The total link bandwidth consumed by 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.

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 links with and without utilizing double mode.

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

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

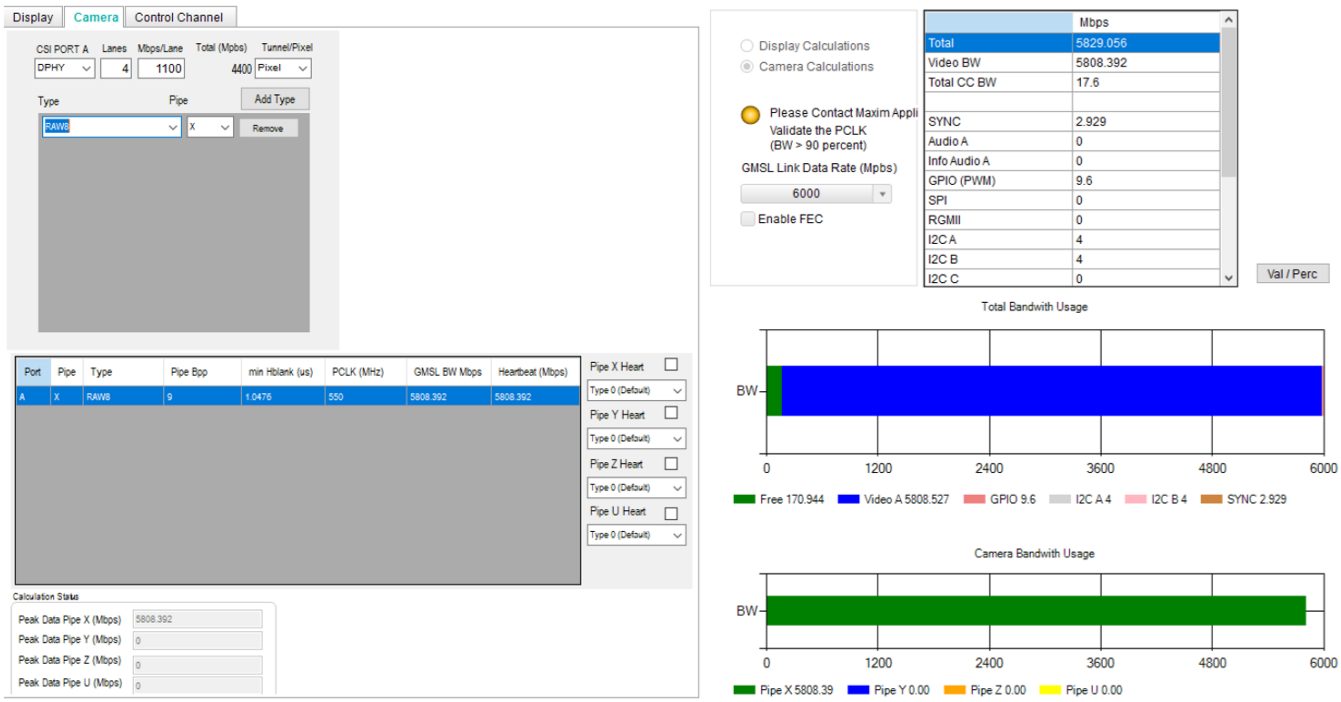

Figure 15. Bandwidth Calculations without Doubling

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

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

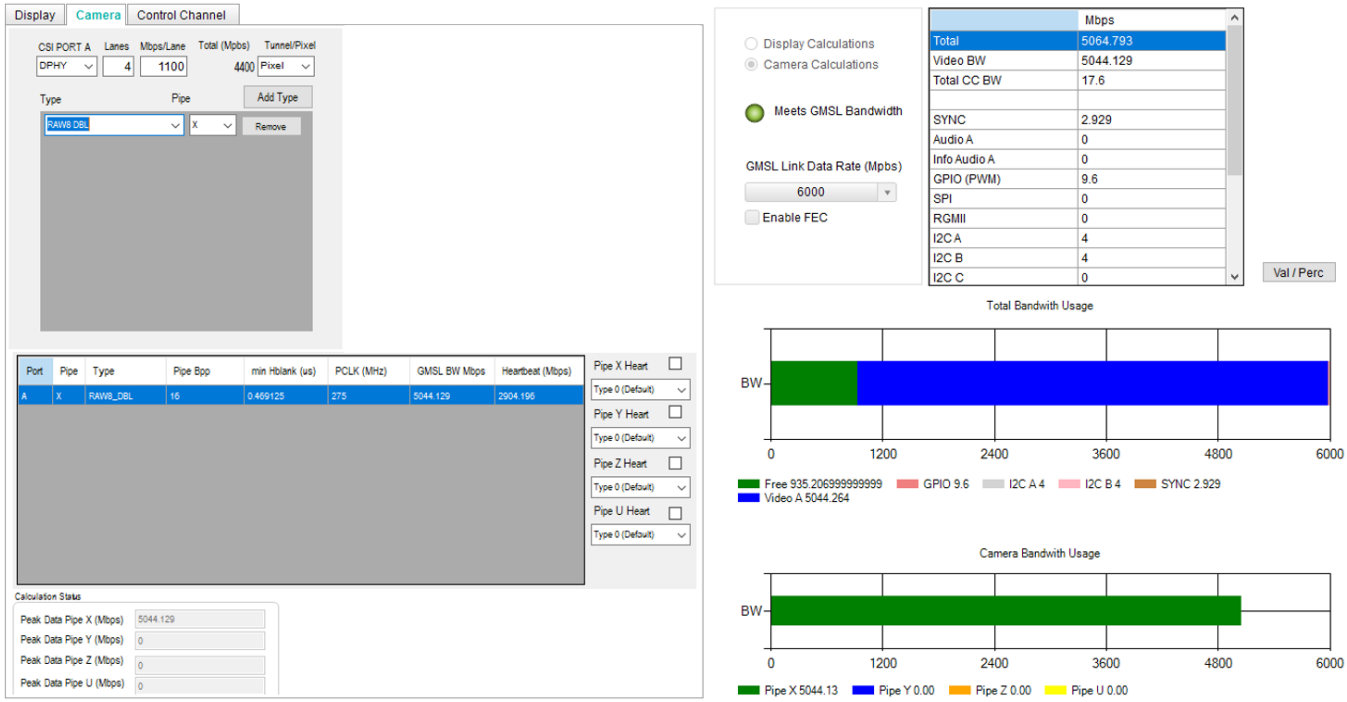

Figure 16. Bandwidth Calculations with Doubling

Error Flags

Overview

The device contains many internal error detection mechanisms to alert the system or user to 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 MFP3 or at MFP8 as an alternative. Whether an error is included in the ERRB output depends on if its output-enable (OEN) is high. Most OENs are high by default.

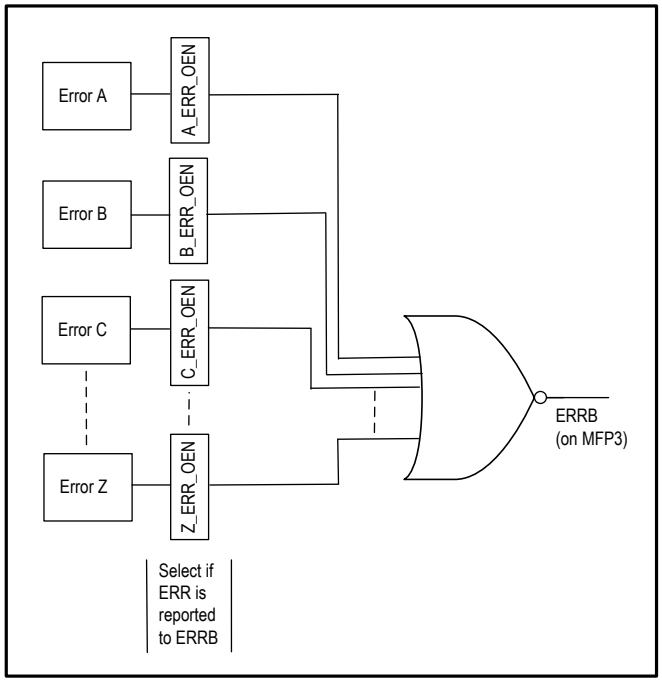

Figure 17. ERRB Reporting Flow

See *Table 20* for a description of each error flag in the MAX9295D.

Table 20. Error Flags Table for MAX9295D

Error Flag	Error Description
EOM_ERR_FLAG_A	Eye opening is below the configured threshold
VDD_OV_FLAG	V_{DD} overvoltage indication
MAX_RT_FLAG	Combined ARQ maximum retransmission limit error flag
RT_CNT_FLAG	Combined ARQ retransmission event flag
PKT_CNT_FLAG	Packet count flag
VDDCMP_INT_FLAG	Combined undervoltage comparator output. Asserted when GMP_SATATUS is asserted.
PORZ_INT_FLAG	PORZ interrupt flag. PORZ is monitoring of undervoltage levels of V_{DD18} and V_{DDIO}.
VDDBAD_INT_FLAG	Combined V_{DD} bad indicator. Asserted when either with VDDBAD_STATUS or CAP_VDD <0.82 V.
ADC_INT_FLAG	ADC interrupt
MIPI_ERR_FLAG	MIPI RX error flag
REM_ERR_FLAG	Received remote side error status

LFLT_INT	Line-fault interrupt asserted when either one of line-fault monitors indicates a fault status
IDLE_ERR_FLAG	Idle-word error flag
DEC_ERR_FLAG	Errors detected in GMSL packet
SPI_RX_OVRFLW	SPI Rx buffer overflow flag
SPI_TX_OVRFLW	SPI Tx buffer overflow flag
adc_overflow_ie	ADC digital correction overflow enabled
adc_lo_limit_ie	Enable ADC low limit monitor interrupt
adc_hi_limit_ie	Enable ADC high limit monitor interrupt
adc_ref_ready_ie	Enable ADC ready interrupt
adc_done_ie	Enable ADC conversion completed Interrupt
DRIFT_ERR	VID_TX_PCLK drift error detected
FIFO_WARN	Transmitted video (VID_TX_FIFO) is more than half full
OVERFLOW	VID_TX FIFO overflow occurred
CMP_STATUS	V_{DD18}, V_{DDIO}, and CAP_VDD supply voltage comparator status bits. Latched when the supply voltages are not in range.
phy*_lp_err	Unrecognized commands or Invalid line sequences are detected. *May be replaced with 1 or 2 for phy1/phy2.
phy*_hs_err	HS sync pattern with error detected on data lanes. *May be replaced with 1 or 2 for phy1/phy2.
ctrl1_csi_err_l	ECC or CRC errors detected in CSI-2 controller, low byte
ctrl1_csi_err_h	Packets terminated early or/and frame count error detected in CSI-2 controller

General-Purpose Input and Output (GPIO)

Overview

MAX9295D has seventeen 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 regarding GPIO capabilities and default states after power-up.

The GPIO blocks of MAX9295D 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 MAX9295D uses 5-bit pin IDs that can support mapping up to 32 GPIO pins. Note that the usable number of GPIOs is limited by the device-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.

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.

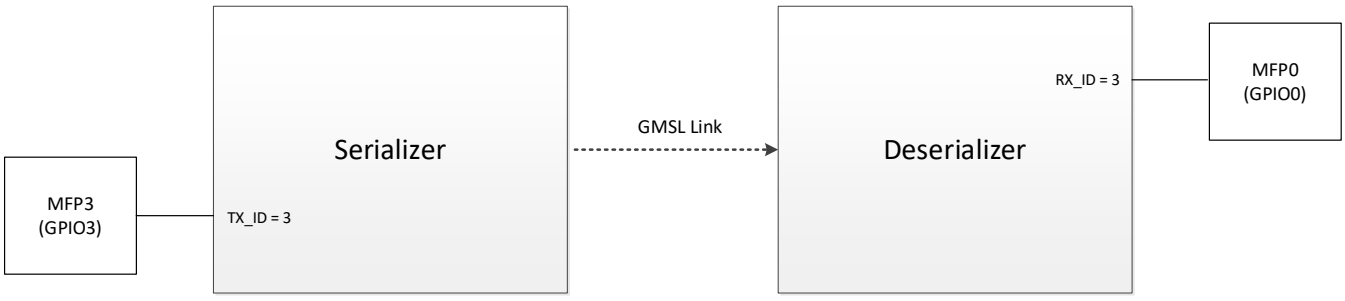

Figure 18. GPIO Forwarding Example with a Transition from MFP3 to MFP0

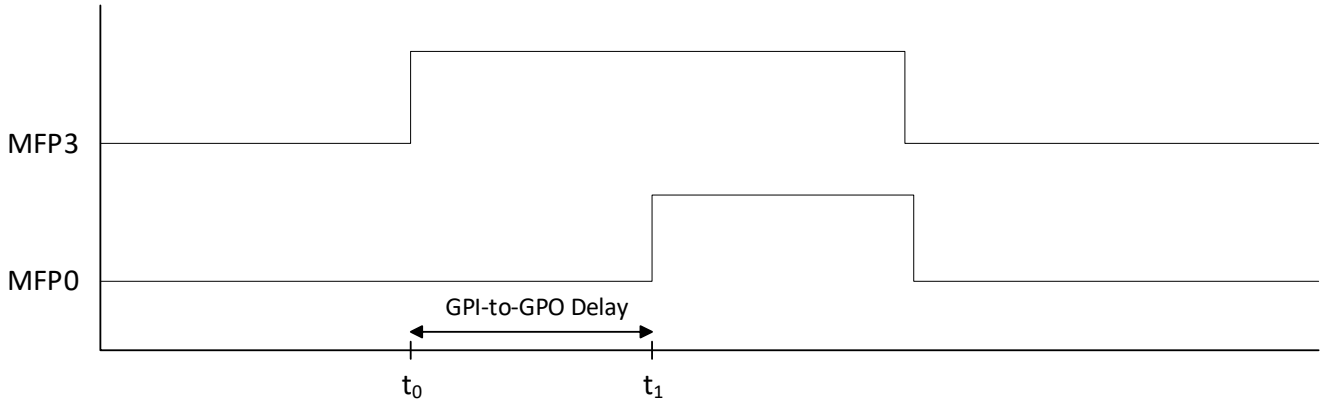

Figure 19. GPIO Forwarding Timing Diagram

MFP Capabilities: GPIO vs. GPI vs. GPO vs. ODO

The MAX9295D has seventeen MFPs; five of these MFPs are GPIO (general-purpose input or output), six are GPO (general-purpose output), and six are ODO_GPI (open-drain output or general-purpose input). *Table 21* shows the GPIO capabilities of each MFP:

Table 21. MFP Capabilities

	MAX9295D
MFP0	GPIO0
MFP1	GPO1
MFP2	GPO2
MFP3	GPIO3
MFP4	GPO4
MFP5	GPO5
MFP6	GPO6
MFP7	GPIO7
MFP8	GPIO8
MFP9	GPIO9
MFP10	GPIO10
MFP11	ODO11_GPI11
MFP12	ODO12_GPI12
MFP13	ODO13_GPI13
MFP14	ODO14_GPI14
MFP15	ODO15_GPI15
MFP16	ODO16_GPI16

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 40 kΩ or 1 MΩ.

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: 40kΩ
- 1: 1MΩ

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 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:

1. Set *GPIO_TX_ID* with a value from 0 to 31 to assign the GPIO pin ID.
2. 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 20* is an example diagram of this configuration.

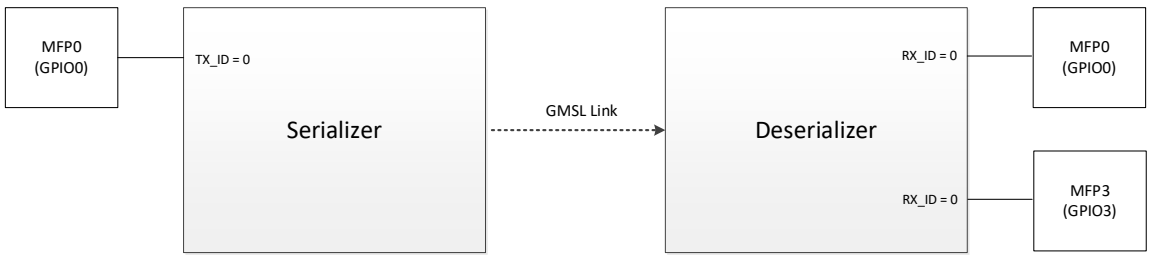

Figure 20. GPIO Broadcasting

Delay Compensation

In 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. The typical values and registers to set delay compensation are as following.

Table 22. Delay Values with and without Compensation

Direction	Delay Compensation	Delay
GPIO forwarding from serializer to deserializer	0	1 µs
	1	3.5 µs (default)
GPIO forwarding from deserializer to serializer	0	6 µs
	1	15 µs (default)

Table 23. Compensation Delay Registers

Register	Bits	Default Value	Description
0x2F	5:0	0b000001	**GPIOA:** Bit [5:0]: GPIO_FWD_CDLY 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 1.7 µs. Default delay is 3.4 µs.
0x31	5:0	0b001000	**GPIOB:** Bit [5:0]: GPIO_REV_CDLY Compensation delay multiplier for the forward 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 1.7 µs. Default delay is 3.4 µs.

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.

GPIO Registers

Table 24. GPIO Registers

Register	Bits	Default Value	Description
0x2BE	7:0	0x99	**GPIO0 A:** Bit 7: RES_CFG 0 = 40KΩ, 1 = 1MΩ Bit 4: GPIO_OUT 0 = Drive output to 0, 1 = Drive output to 1 Bit 3: GPIO_IN, GPIO input level 0 or 1 Bit 2: GPIO_RX_EN 0 = Disable receiving from the link. 1 = Enable receiving from the link Bit 1: GPIO_TX_EN 0 = Disable transmitting to the link. 1 = Enable transmitting to the link Bit 0: GPIO_OUT_DIS 0 = Output driver enabled 1 = Output driver disabled
0x2BF	7:0	0xA0	**GPIO0 B:** Bit [7:6]: Resistor configuration 00 = None 01 = Pull-up 10 = Pull-down Bit 5: OUT_TYPE 0 = Open-drain 1 = Push-pull Bit [4:0]: GPIO_TX_ID, Address = 0-31
0x2C0	6:0	0x40	**GPIO0 C:** Bit 6: GPIO_RECVED, Received GPIO Value 0 or 1 Bit [4:0]: GPIO_RX_ID, Address = 0-31
0x2C1 - 0x2DE	…	…	**Repeat Registers A, B, C for GPIO1-10**

GPIO Programming Example

In this example, GPIO0 on a MAX9295D serializer is forwarded across the link to GPIO0 on a MAX96724 deserializer. This example could 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.

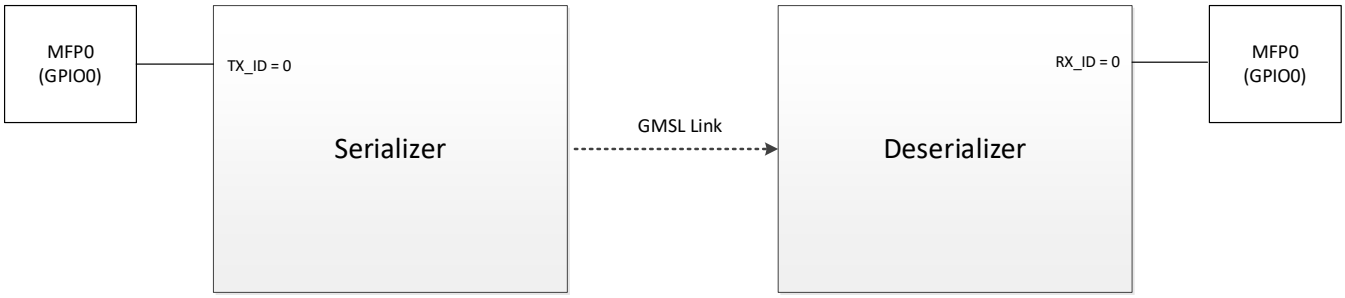

Figure 21. GPIO Forwarding Programming Example

Table 25. GPIO Programming Example

Step	Action	Device	Read/Write	Register Address	Value
1	Set up deserializer GPIO0 Rx enable, enable output driver, and set resistor to 1 MΩ.	DES	W	0x300	0x84
2	Set up deserializer GPIO output type to push-pull and configure resistor as pull-down.	DES	W	0x301	0xA0
3	Set up deserializer GPIO0 to receive ID GPIO_RX_ID to be 0.	DES	W	0x302	0x00
4	Set up serializer GPIO0 Tx enable.	SER	W	0x2BE	0x83
5	Set up serializer GPIO0 transmit ID GPIO_TX_ID to be 0.	SER	W	0x2BF	0xA0

MFP Slew Rate

The MAX9295D's multifunction pins (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. Care should be taken 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.

MFP Slew Rate Operation

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

1. **I²C/UART**: MFP pins operating as an I²C or UART function (that is, control channel or pass-through) are not affected by the MFP rise/fall setting. The I²C/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 & 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.

MFP Slew Rate Programming and 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.

The I²C/UART functions are not affected by the MFP slew rate adjustment. If an MFP is used as an I²C or UART pin, the slew rate is automatically adjusted to meet the applicable specification.

The device V_{DDIO} level determines the available range of the slew rate configuration options. For each V_{DDIO} level, the MFP speed groups have four available speed options configured by two speed control bits.

CMU4 is the MFP speed control register for the MAX9295D.
Refer to the corresponding device's data sheet "Control- and Side-Channel Typical Rise and Fall Times" section for V_{DDIO} timing details. Typical rise and fall times for GMSL2 devices are presented in the following table.

Table 26. Typical Rise/Fall Times for GMSL2 Devices

Register Value	Rise/Fall Time	
	VDDIO=1.8V	VDDIO=3.3V
0x0	2n	1n
0x1	4n	2n
0x2	8n	4n
0x3	16n	8n

MFP Slew Rate Example

Example 1. CMU4 Register Example Using the CSI-2 Serializer
The MAX9295D CMU4 (0x304) register to configure the MFP slew rate. This register has the following mapping:

Table 27. CMU4 Register

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SPI_MS_SLW		SPI_LS_SLW		REFOUT_SLW		GP_SLW	

Video Pattern Generator (VPG)

Overview

The VPG creates either a checkerboard or gradient pattern with programmable parameters. These patterns can be used to replace the incoming video to create an RGB888 video pattern when no video is present on the serializer input. Its ability to generate test images is useful for evaluation and debugging.

Video Pattern Generator Operation

The MAX9295D has an internal video pattern generator (VPG) that accommodates a wide range of resolutions and frame rates. The VPG requires an external PCLK source from the CSI input. Link lock is not required for the VPG to be used.

The VPG block generates and outputs video data based on specified timing parameters. The video timing is sourced from the input video stream. The VPG outputs either a color gradient pattern or a checkerboard pattern with user-definable color parameters and pattern details.

The VPG allows users to perform video tests without a video source. In *Figure 22* and *Figure 23*, patterns generated by the serializer are sent through the serial link, received by the deserializer, and output to test the serial link and downstream devices such as displays.

The GMSL GUI allows easy evaluation of the VPG. To enable VPG from GUI, go to the **Tools** tab and open **Video Timing and Pattern Generator**. A window pops up (*Figure 22*).

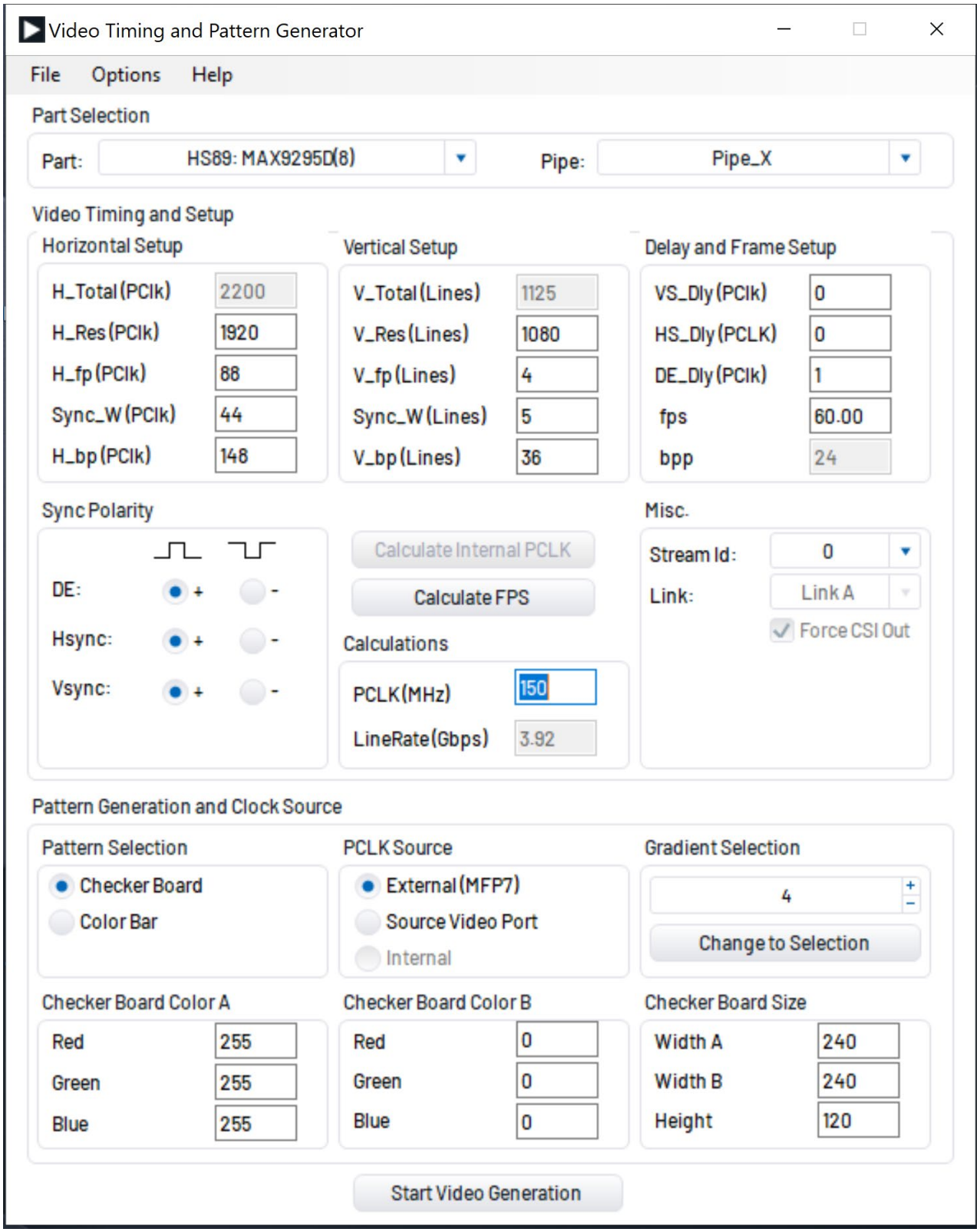

Figure 22. GMSL GUI Video Timing and Pattern Generator

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

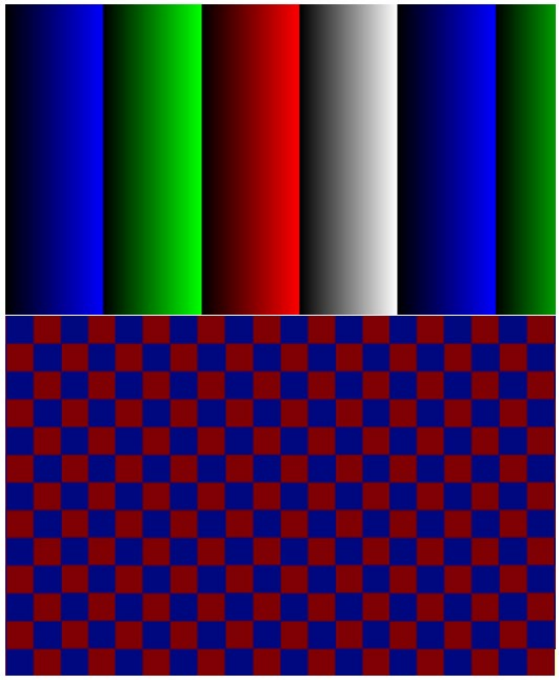

Figure 23. VPG Pattern Options, Gradient (Top), and Checkerboard (Bottom)

VPG Configuration

In gradient mode, the length of the gradient pattern and gradient direction are configurable. In checkerboard mode, two colors (color A and color B) are selected and the size of the pattern is user-defined. The VPG can generate in RGB888 format. The following table lists the configuration registers for the MAX9295D.

Table 28. VPG Configuration Registers for MAX9295D

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

In checkerboard mode: set the value of color A	VTX31 VTX32 VTX33	CHKR_A_L[7:0] CHKR_A_M[7:0] CHKR_A_H[7:0]	Sets the red component of color A Sets the green component of color A Sets the blue component of color A
In checkerboard mode: set the value of color B	VTX34 VTX35 VTX36	CHKR_B_L[7:0] CHKR_B_M[7:0] CHKR_B_H[7:0]	Sets the red component of color B Sets the green component of color B Sets the blue component of color B
In checkerboard mode: set the length of color A	VTX37	CHKR_RPT_A[7:0]	Sets the number of pixels of color A. The first line outputs color A first.
In checkerboard mode: set the length of color B	VTX38	CHKR_RPT_B[7:0]	Sets the number of pixels of color B. The first line outputs color B after CHKR_RPT_A pixels. Set equal to CHKR_RPT_A for a square checkerboard pattern.
In checkerboard mode: set the height of the checkerboard	VTX39	CHKR_ALT[7:0]	After CHKR_ALT lines, the pattern switches to output color B before color A. Set equal to CHKR_RPT_A and CHKR_RPT_B for a square checkerboard pattern.

Pairing with GMSL1 Deserializers

Overview

The MAX9295D is GMSL1 backwards compatible and may be paired with GMSL1 deserializers. Unlike GMSL2, the GMSL1 link does not automatically establish, and the forward video channel requires an external video pixel clock at the serializer input to be established.

The following procedure is used to establish a GMSL1 link:

- Build the reverse control channel (the CLINK).
- Program the serializer and image sensor through the reverse channel.
- Program the deserializer following the steps described in the *Configuration* section.
- Enable image sensor streaming and forward channel.

When using the GMSL1 mode, use only GMSL1 deserializers equipped with high immunity mode (HIM). HIM provides robust control channel electromagnetic compatibility (EMC) tolerance, which reduces the effects of bit errors and the potential to corrupt reverse channel communication. Devices operating without HIM can fail bulk current injection (BCI) tests and should not be used in applications requiring power-over-coax (PoC).

Other GMSL1 legacy mode deserializers without HIM support (example, MAX9272) can be used, but require additional design considerations. Building the CLINK without HIM enabled requires more configuration and is sensitive to hardware setup if PoC is used. This is not supported for new designs but may be supported for legacy systems.

A robust reverse communication channel is important when pairing the GMSL2 CSI-2 deserializer with GMSL1 devices. Enable packet-based control channel mode by setting the deserializer *PKTCC_EN* bits high for each link. If there is no I^2C main on the remote side of the link, set *NO_REM_MST* high to reduce unnecessary timeouts on the communication channels.

Pairing with a High Immunity Mode Capable GMSL1 Deserializer

A GMSL1 deserializer with HIM mode enabled upon power-up (example, MAX96706) is recommended for GMSL1 camera applications with the MAX9295D. This is achieved by adding a pull-up resistor at the HIM pin of the deserializer. Refer to the *MAX96706* data sheet for additional details.

There are two methods to build the CLINK. Power up the GMSL2 CSI-2 serializer with GMSL1 HIM mode enabled by setting the CFG pins or enable HIM mode with register write *HIGHIMM* (bit 7 in register *0x044D*).

If a MAX96706 is powered up without HIM enabled, the communication link, CLINK, must be built under non-HIM mode. After the CLINK is built, HIM mode can then be enabled through register writes.

The procedure is listed in *Figure 24* and *Table 29*. Note that in systems with multiple GMSL1 serializers, before HIM mode is established on both sides, it is recommended to individually enable and program each link to avoid potential I^2C command race conditions. This can be done by only having one serializer active at a time.

Programming Examples

This example demonstrates how to set up the communication channel for a GMSL 1 link. Note that this requires an I^2C interface capable of writing 8-bit or 16-bit register addresses.

```
# Turn on HIM on MAX96706 Link A
0x48,0x0B06,0xEF
# Turn on local I2C acknowledge as link is not established yet
0x48,0x0B0D,0x80
# Enable CLINK in MAX9295D that is connected on Link A
0x80,0x04,0x43
# Delay 5ms
# Turn off local I2C acknowledge
0x48,0x0B0D,0x00
```

This example shows how to set up a complete GMSL1 link and the required registers for the serializer. This example assumes that the deserializer is in HIM mode at power-up.

```
# Enable GMSL 1
0x48,0x0006,0x40
# Set deserializer link A: HIM Mode Enable
0x48,0x0B06,0xEF
# Turn on local I2C acknowledge as link is not established yet
0x48,0x0B0D,0x80
# At this point users should have access to the serializer I2C communication. If
using the GMSL GUI users need to hit identify devices under options tab.

# Turn on CLINK link and turn video link off.
0x80,0x04,0x43

# Serializer Enable double and HVEN
0x80,0x07,0x84
# Deserializer Enable double and HVEN
0x48,0x0B07,0x84
# Deserializer Disable Remote master
0x48,0x0B05,0x79
# Deserializer Enable SHIFT_VID_HVD
0x48,0x0BA7,0x45
# Disable I2C_LOC_ACK: This should be disabled once the forward channel is
established.
0x48,0x0B0D,0x00
# Deserializer, Set datatype to YUV422. This can be changed depending on the data
type used.
0x48,0x0B96,0x1B
# Deserializer, DE_EN = 0; This is dependent on the camera used.
0x98,0x0B0F,0x01
```

Configuration Flowchart and Detailed Steps

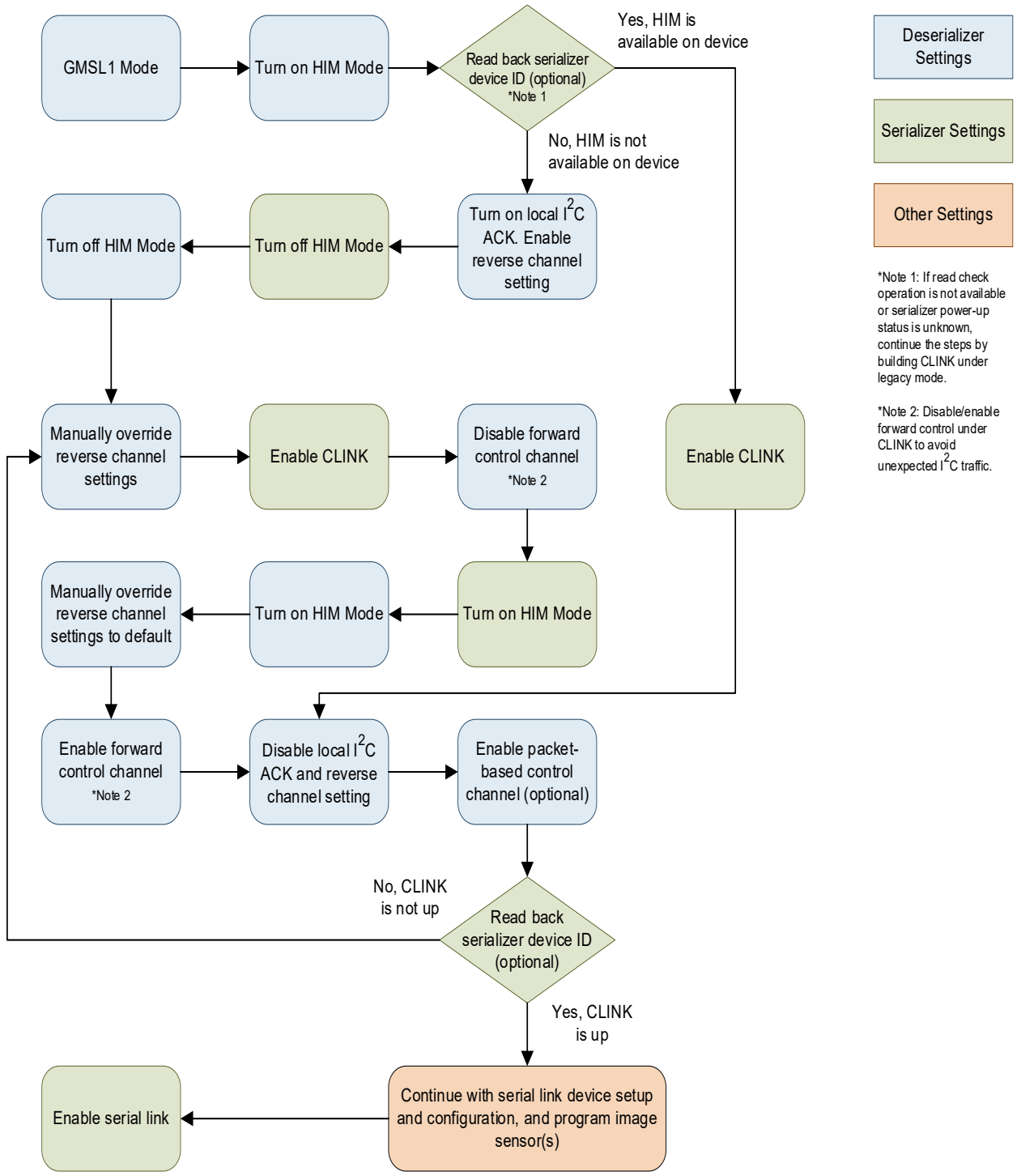

Figure 24. Building the CLINK on HIM-Capable GMSL1 Deserializer

Table 29. Configuration Steps for Pairing HIM-Capable GMSL1 Deserializers

STEP	DEVICE	SLAVE ID	REGISTER(S)	VALUE	DESCRIPTION
0	DES	0x48	0x0006	0x40	Enable Link A GMSL1 mode.
Delay 5 ms					
1	DES	0x48	0x0313	0x00	Turn off CSI output
2	DES	0x48	0x0B06	0xEF	Turn on HIM mode on Link A *Skip this step if the serializer is not in HIM mode.
Note					If Ser is powered up with HIM by default, CLINK is established with one more write on Ser (0x04 = 0x43). Proceed to step 17. Otherwise continue.
3	DES	0x48	0x0B0D	0x81	Enable reverse channel configuration on Link A. Turn on local I^2C acknowledge on Link A.
4	SER	0x80	0x44D	0x00	Turn off HIM on Ser.
Delay 10 ms					
5	DES	0x48	0x0B06	0x6F	Turn off HIM mode on Link A.
6	DES	0x48	0x14C5	0xAA	Manually override reverse channel pulse length for Link A.
7	Des	0x48	0x14C4	0x80	Manually override reverse channel rise/fall time for Link A.
8	Des	0x48	0x1495	0xC8	Manually override reverse channel Tx amplitude for Link A.
9	SER	0x80	0x0404	0x43	Enable control link only. Disable serial link.
Delay 5 ms					Control link should be built up successfully.
10	DES	0x48	0x0B04	0x02	Disable forward control channel transmitter for Link A.
11	SER	0x80	0x44D	0x80	Enable HIM mode on Ser (that is, remote side).
Delay 5 ms					
12	DES	0x48	0x0B06	0xEF	Turn on HIM mode on Link A.
CLINK should be established with HIM enabled.					
13	DES	0x48	0x14C5	0x40	Manually override reverse channel pulse length to default value for Link A.
14	DES	0x48	0x14C4	0x40	Manually override reverse channel rise/fall time to default value for Link A.
15	DES	0x48	0x1495	0x69	Manually override reverse channel Tx amplitude to default value for Link A.

16	DES	0x48	0x0B04	0x03	Enable forward control channel transmitter for Link A.
17	DES	0x48	0x0B0D	0x00	Disable local I^2C acknowledge. Disable reverse channel setting.
Optional	DES	0x48	0x0B08	0x25	Enable packet-based control channel mode for Link A.
Delay 10 ms					
Continue programming the deserializer configuration.					
Continue programming the serializer configuration and camera.					
Delay 10 ms					
19	SER	0x80	0x0404	0x83	Enable serial link.
(If HIBW)	SER	0x80	0x0407	0bX1XXXXXX	Enable HIBW mode
Delay 10 ms					
(If HIBW)	DES	0x48	0x0B07	0bXXXX1XXX	Enable HIBW mode
Delay 10 ms					
(If HIBW and PKTCC)	DES	0x48	0x0B19	Read	Clear out CC CRC errors due to HIBW switch
Last	DES	0x48	0x0313	0x02	Enable CSI output.

GPIO Forwarding with GPI-GPO and CNTL Pins

In the GMSL1 mode, GPIO forwarding is done using GPI-GPO for the reverse direction and CNTL pins in the forward direction.

Reverse Direction with GPI-GPO

GPI is located at MFP1 and is enabled by default. Disable GPI through register 0xB08. Do not use GPI-GPO forwarding when FSYNC is enabled.

GPI-GPO takes priority on the control channel. I^2C communication is paused through clock stretching, while UART communication gets overwritten and requires the microcontroller to retransmit.

Forward Direction with CNTL

GPIO tunneling in the forward direction is done through the CNTL outputs. The number of available CNTL signals depends on channel settings of the GMSL1 link and is detailed in *Table 30*. Note that while most CNTL outputs of the MAX96714 correspond with the same input on the GMSL1 part, CNTL4 corresponds with the GMSL1 audio bit.

Table 30. CNTL Availability and Bandwidth

MODE	CNTL0	CNTL1	CNTL2	CNTL3	CNTL4
BWS = 0, HIBW = 0	None	None	None	None	Full
BWS = 1	None	Full	Full	None	Full

BWS = 0 HIBW = 1	Blank*	Blank*	Blank*	Blank*	None

*In this mode, CNTL pins only change during the blanking period and are intended for slow signals only. Set the triggering modes in the serializer.

Note: CNTL operation requires an active video link, and certain register settings to work. The following bits must be set: DBL = 0, HVEN = 0, SEREN = 1, CLINKEN = 0, and PCLK is applied.

The CNTL signals can be reordered to change the output that sent a CNTL signal. By default, the signals are assigned in a straight through manner, that is, the CNTL0 signal outputs from CNTL0, CNTL1 outputs from CNTL1, etc., to change the output order, and change the CNTL OUT ORD bits. Example, when CNTL_OUT_ORD = 0b001, CNTL3 outputs from the CNTL4 pin, CNTL2 outputs from the CNTL3 pin, etc.

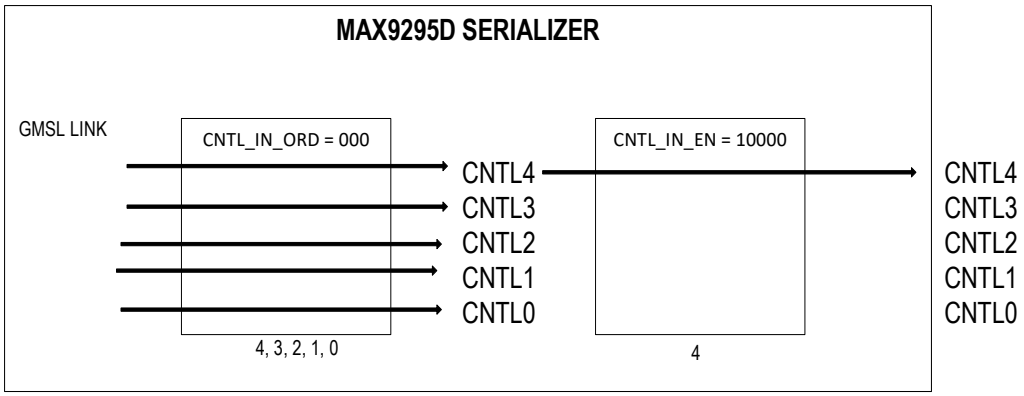

Figure 4. Default CNTL Order, CNTL4 Enabled

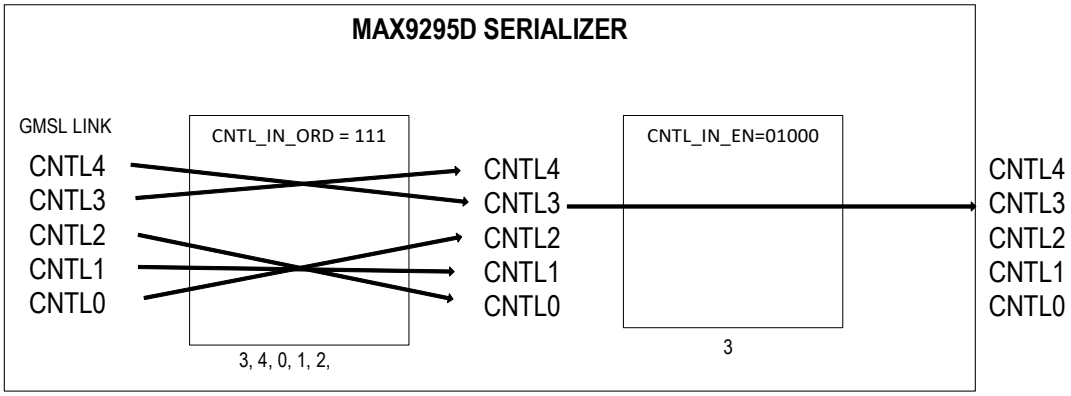

Figure 5. Outputting CNTL4 Signal from CNTL3 Pin

GMSL1 Delay Compensation

By default, the GMSL1 control channel (and thus GPI-GPO) is not a packet-based channel and does not need delay compensation.

If packet-based control channel is used, set GPI_COMP_EN = 1 (in the GMSL1 block) to turn on packet-based GPI-GPO delay compensation.

The CNTL pins are sent on the video path (not the control channel), and do not require delay compensation.

GMSL1 GPI-GPO and CNTL-Related Register Bits

Table 31. *Serializer Video PRBS Generator and Checker Registers*

REGISTER	BITS	DEFAULT VALUE	DESCRIPTION	DECODE
0xB06	4	0b0	GPI skew compensation enable (when using packet-based control channel).	0b0: GPI Skew compensation not enabled 0b1: GPI Skew compensation
0xBD1	7:5	0b000	Internal CNTL_OUT order to pins CNTL4 to CNTL0. By default, CNTL pins are assigned as shown in the pin description table.	**0b000 4, 3, 2, 1, 0 (Default)** 0b001 3, 2, 1, 0, 4 0b010 2, 1, 0, 4, 3 0b011 1, 0, 4, 3, 2 0b100 0, 1, 2, 3, 4 0b101 1, 2, 3, 4, 0 0b110 2, 3, 4, 0, 1 0b111 3, 4, 0, 1, 2
	4:0	0b00000	CNTL output enable for CNTL[4:0]	0b0XXXX: Disable CNTL4 0b1XXXX: Enable CNTL4 0bX0XXX: Disable CNTL3 0bX1XXX: Enable CNTL3 0bXX0XX: Disable CNTL2 0bXX1XX: Enable CNTL2 0bXXX0X: Disable CNTL1 0bXXX1X: Enable CNTL1 0bXXXX0: Disable CNTL0 0bXXXX1: Enable CNTL0

Complete Use Case Programming Examples

The following use case examples demonstrate how many of the features described throughout this document can be used together to program a SerDes system. These examples may need to be manipulated or completely changed for more specific use cases. The basic flow of programming and important steps is annotated to give a broad picture of the requirements users can expect to get a system working with the MAX96724 deserializer and MAX9295D serializer.

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

Table 32 Explanation of GUI Programming for .cpp files

Number of I2C transactions	Device Address	High Register Address	Low Register Address	Register value	Description
0x04	0x80	0x03	0x83	0x00	//Description

Use Case Example

This example has the following characteristics:

- Two image sensors connected to one MAX9295D and MAX96724 each.
- I^2C address of serializer is 0x80.
- I^2C address of deserializer is 0x4E.
- Link rate = 6 Gbps
- The serializer receives YUV8 from both sensors and assigns two video pipes and two VCs.
- The deserializer receives and outputs the data on Port A, 1.5 Gbps/lane.

Use Case Example Script

```
//**************
Setup details:
SensorA (0x20) ---> MIPI PHY A 4-lane of MAX9295D -------- GMSL2 6G Coax --------> MAX96724 Link A ---> VC0 aggregated at MIPI PHY A 1.5Gbps/lane @ 4-lane
SensorB (0x30) ---> MIPI PHY B 2-lane of MAX9295D -------- GMSL2 6G Coax --------> MAX96724 Link A ---> VC1 aggregated at MIPI PHY A 1.5Gbps/lane @ 4-lane

// ************* Serializer MAX9295D ********************
// Enable ERROR and Lock output
0x04,0x80,0x00,0x05,0xC0,
// Make sure that the SER is in 2x4 mode
```

```
0x04,0x80,0x03,0x30,0x06,
// Set 4 lane for PHY A and 2 lane for PHY B
0x04,0x80,0x03,0x31,0x13,
// Route RAW16 to VIDEO_X from PHY A SensorA
0x04,0x80,0x03,0x14,0x6E,
// Route EMB8 to VIDEO_Y from PHY A SensorA
0x04,0x80,0x03,0x16,0x4E,
// Route RAW12 to VIDEO_Z from PHY B SensorB
0x04,0x80,0x03,0x18,0x6C,
// Route EMB8 to VIDEO_U from PHY B SensorB
0x04,0x80,0x03,0x1A,0x4E,
// Enable pipe X/Y for SenorA and Z/U for SensorB and stream all
0x04,0x80,0x03,0x08,0x7C,
0x04,0x80,0x03,0x11,0xC3,
0x04,0x80,0x00,0x02,0xF3,

// Efficiency updates for all pipes
0x04,0x80,0x01,0x02,0x0E,
0x04,0x80,0x01,0x0A,0x0E,
0x04,0x80,0x01,0x12,0x0E,
0x04,0x80,0x01,0x1A,0x0E,

// ************* Deserializer MAX96724 ********************
// Efficiency updates for pipes 0-3
0x04,0x4E,0x01,0x00,0x23,
0x04,0x4E,0x01,0x12,0x23,
0x04,0x4E,0x01,0x24,0x23,
0x04,0x4E,0x01,0x36,0x23,

// Enable only one link
0x04,0x4E,0x00,0x06,0xF1,
//  Video Pipe Selection
0x04,0x4E,0x00,0xF0,0x20,   // Pipe Z from link A to pipe 1; Pipe X from link A to pipe 0
0x04,0x4E,0x00,0xF1,0x31,
0x04,0x4E,0x00,0xF4,0x0F,   // Turn on pipe 0/1

// MIPI PHY Setting
// Set Des in 2x4 mode
0x04,0x4E,0x08,0xA0,0x04,
// Set Lane Mapping for 4-lane port A
0x04,0x4E,0x08,0xA3,0xE4,
0x04,0x4E,0x08,0xA4,0xE4,
// C-PHY timing setting
//0x04,0x4E,0x08,0xAD,0x1F,
//0x04,0x4E,0x08,0xAE,0x5C,
// Set 4 lane D-PHY
0x04,0x4E,0x09,0x0A,0xC0,
0x04,0x4E,0x09,0x4A,0xC0,
```

```
0x04,0x4E,0x09,0x8A,0xC0,
0x04,0x4E,0x09,0xCA,0xC0,
// Turn on MIPI PHYs
0x04,0x4E,0x08,0xA2,0xF0,
// Set Data rate to be 1500Mbps
0x04,0x4E,0x04,0x15,0x2F,
0x04,0x4E,0x04,0x18,0x2F,
0x04,0x4E,0x04,0x1B,0x2F,
0x04,0x4E,0x04,0x1E,0x2F,

// Disable VS output on pipe 2 and 3
0x04,0x4E,0x04,0x36,0x0C,

// SensorA video pipeline 0
0x04,0x4E,0x09,0x0B,0x07,
0x04,0x4E,0x09,0x2D,0x15, // CSI2 controller 1
0x04,0x4E,0x09,0x0D,0x2E,
0x04,0x4E,0x09,0x0E,0x2E,
0x04,0x4E,0x09,0x0F,0x00,
0x04,0x4E,0x09,0x10,0x00,
0x04,0x4E,0x09,0x11,0x01,
0x04,0x4E,0x09,0x12,0x01,

// SensorB video pipeline 1
0x04,0x4E,0x09,0x4B,0x07,
0x04,0x4E,0x09,0x6D,0x15, // CSI2 controller 1
0x04,0x4E,0x09,0x4D,0x2C,
0x04,0x4E,0x09,0x4E,0x6C,
0x04,0x4E,0x09,0x4F,0x00,
0x04,0x4E,0x09,0x50,0x40,
0x04,0x4E,0x09,0x51,0x01,
0x04,0x4E,0x09,0x4E,0x41,

// EMB8, video pipeline 2
0x04,0x4E,0x09,0x8B,0x01,
0x04,0x4E,0x09,0xAD,0x01, // CSI2 controller 1
0x04,0x4E,0x09,0x8D,0x12,
0x04,0x4E,0x09,0x8E,0x12,
//0x04,0x4E,0x09,0x8F,0x00,
//0x04,0x4E,0x09,0x90,0x00,
//0x04,0x4E,0x09,0x91,0x01,
//0x04,0x4E,0x09,0x92,0x01,

// EMB8, video pipeline 3
0x04,0x4E,0x09,0xCB,0x01,
0x04,0x4E,0x09,0xED,0x01, // CSI2 controller 1
0x04,0x4E,0x09,0xCD,0x12,
0x04,0x4E,0x09,0xCE,0x4E,
```

```
//0x04,0x4E,0x09,0xCF,0x00,
//0x04,0x4E,0x09,0xD0,0x40,
//0x04,0x4E,0x09,0xD1,0x01,
//0x04,0x4E,0x09,0xD2,0x41,

// Disable CSI_OUTPUT
0x04,0x4E,0x04,0x0B,0x00,
0x00,0xFF,

// ************* Image Sensor Setup ******************
//Insert Image Sensor Config Script

// Enable CSI_OUTPUT
0x04,0x4E,0x04,0x0B,0x02,
// One shot link reset
0x04,0x4E,0x00,0x18,0x0F,
0x00,0x0F,
0x04,0x80,0x00,0x10,0x21,
```

Appendix

Figures

Figure 1. MAX9295D Two-Camera Application Example 3
Figure 2. Inputs, PHY, and Controller Diagram (MIPI Port B Only) 7
Figure 3. D-PHY Lane Swap Example (MIPI Port B Only) 8
Figure 4. Complex Configuration Example 14
Figure 5. I²C Interfaced Camera-Module System with Default Address Settings 21
Figure 6. Camera-Module System with Translated Address Settings 24
Figure 7. SPI Architecture 30
Figure 8. SPI Clock and Data at Final Output (at External SPI Target), No Video on GMSL Link 33
Figure 9. SPI Clock and Data at Final Output (at External SPI Target), 92% Video on GMSL Link 33
Figure 10. SPI Transmission Example 35
Figure 11. Frame Alignment (Without Frame Sync) 36
Figure 12. Frame Alignment (Frame Sync Enabled) 37
Figure 13. Frame Sync Application Diagram 38
Figure 14. Power Manager State Diagram 40
Figure 15. Bandwidth Calculations without Doubling 45
Figure 16. Bandwidth Calculations with Doubling 45
Figure 17. ERRB Reporting Flow 46
Figure 18. GPIO Forwarding Example with a Transition from MFP3 to MFP0 48
Figure 19. GPIO Forwarding Timing Diagram 49
Figure 20. GPIO Broadcasting 50
Figure 21. GPIO Forwarding Programming Example 53
Figure 22. GMSL GUI Video Timing and Pattern Generator 57
Figure 23. VPG Pattern Options, Gradient (Top), and Checkerboard (Bottom) 58
Figure 24. Building the CLINK on HIM-Capable GMSL1 Deserializer 62
Figure 25. Default CNTL Order, CNTL4 Enabled 65
Figure 26. Outputting CNTL4 Signal from CNTL3 Pin 65

Tables

Table 1. MAX9295D Start-Up Sequence 6
Table 2. Basic Settings 6
Table 3. Link Initialization Registers 7
Table 4. MIPI PHY Settings Registers 9
Table 5. Video Pipe and Filtering Registers 12
Table 6. Heartbeat Disable Register 13
Table 7. Software Override Registers 15
Table 8. Double Mode Registers 16
Table 9. Zero Padding Registers 17
Table 10. MFPs for I²C 19
Table 11. MAX9295D I²C Registers 20
Table 12. I²C Broadcasting (Quad) Example (Serializer) 25
Table 13. I²C Broadcasting (Quad) Example (Image Sensor) 25
Table 14. MFPs for UART 27
Table 15. MAX9295D UART Registers 28

TABLE 16. IMPORTANT SPI REGISTER SETTINGS 31
TABLE 17. VIDEO DETAILS FOR THE EXAMPLE 34
TABLE 18. POWER MANAGER AND SLEEP MODE AVAILABILITY FOR MAX9295D FAMILY 39
TABLE 19. MAXIMUM VIDEO PAYLOADS 44
TABLE 20. ERROR FLAGS TABLE FOR MAX9295D 46
TABLE 21. MFP CAPABILITIES 49
TABLE 22. DELAY VALUES WITH AND WITHOUT COMPENSATION 51
TABLE 23. COMPENSATION DELAY REGISTERS 51
TABLE 24. GPIO REGISTERS 52
TABLE 25. GPIO PROGRAMMING EXAMPLE 53
TABLE 26. TYPICAL RISE/FALL TIMES FOR GMSL2 DEVICES 54
TABLE 27. CMU4 REGISTER 55
TABLE 28. VPG CONFIGURATION REGISTERS FOR MAX9295D 58
TABLE 29. CONFIGURATION STEPS FOR PAIRING HIM-CAPABLE GMSL1 DESERIALIZERS 63
TABLE 30. CNTL AVAILABILITY AND BANDWIDTH 64
TABLE 31. SERIALIZER VIDEO PRBS GENERATOR AND CHECKER REGISTERS 66
TABLE 32 EXPLANATION OF GUI PROGRAMMING FOR .CPP FILES 67

Revision History

Revision Number	Revision Date	Description/Changes	Pages Changed
0	01/24	Initial Release	-

GMSL and GMSL2 are trademarks of Analog Devices, Inc., all rights reserved.