AHEAD OF WHAT'S POSSIBLE™

MAX96717/F/R User Guide

GMSL SerDes Applications Team

Version 0

May, 2023

Table of Contents

MAX96717/F/R Serializers

Device Overview

This user guide is intended to be used in conjunction with other documents such as the MAX96717 datasheets, errata documents, and other user and design guides. It provides explanations, examples, and instructions to help setup video configurations and use various features. Within the examples, the serializer I2C address can be assumed to be 0x80, unless otherwise noted.

Examples may be shown without errata writes that are necessary to ensure reliable operation in production. Be sure to contact your 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 MAX96717 device for testing.

The MAX96717 family of parts consists of the MAX96717 (plain version, with no suffix in the base part number), the MAX96717F, and the MAX96717R. The MAX96717 is the full-feature device. The F version omits the 6 Gbps mode, and several features, including the 6 Gbps mode, are not available in the R version. In some cases, the document refers to them all as only MAX96717, representing the family, unless it is necessary to differentiate between the three.

Table 1. Comparison of the MAX96717 Family

Part Number	Forward Link Rate	ASIL Rating	Sleep Mode	Package	Coax/STP	Virtual Channels	SPI	ADC	UART	Max MIPI Rate Per Lane	Number of MFP Pins	I/O Voltages (VDDIO)
MAX96717	3 or 6 Gbps	B	Yes	Wettable or Non-Wettable	Coax or STP	Up to 16	Yes	Yes	Yes	2500 Mbps	11	1.8 or 3.3 V
MAX96717F	3 Gbps	B	Yes	Wettable or Non-Wettable	Coax or STP	Up to 16	Yes	Yes	Yes	2500 Mbps	11	1.8 or 3.3 V
MAX96717R	3 Gbps	QM	No	Non-Wettable	Coax	Up to 4	No	No	No	600 Mbps	7	1.8 V

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

GMSL2™ serial links use packet-based, bidirectional architecture with forward and reverse channels. The forward channel transfers data from the serializer to the deserializer; the reverse channel transfers data from the deserializer to the serializer. The MAX96717 is capable of 3 or 6 Gbps forward link rate (selectable with resistors connected to the CFG pin or with register writes), while the -F and -R are capable of 3 Gbps. All variants have a 187.5 Mbps reverse direction rate.

Application Use Case

In a typical configuration the MAX96717 is used to support cameras in the 1 to 8 MP range. Typically, each camera's image sensor feeds the video into the D-Phy CSI input port of the MAX96717 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 are four MAX96717s operating independently and sending data out to a deserializer. Coax or shielded twisted-pair (STP) cables can be used for the GMSL link.

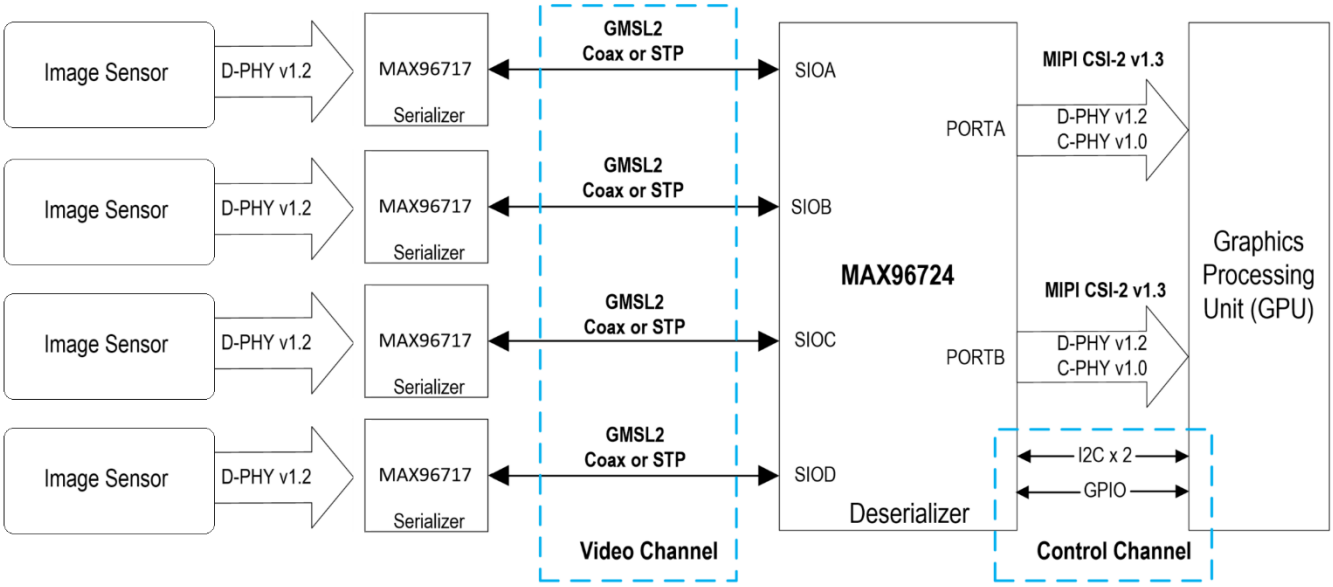


Figure 1. MAX96717 Four-Camera Application Example

Start up and Programming Sequence

Overview

The MAX96717 family has many applications use cases and features that work in conjunction with each other. To avoid feature and system sequencing issues, the preferred order is outlined in the table below. Features or configuration changes may not be required and may be skipped in the startup sequence, depending on system requirements and data configurations.

Recommended Startup Sequence

Table 2. MAX96717/F/R Startup Sequence (Note: Some features not available on MAX96717R)

MAX9617/F/R Startup and Programming Sequence		
General Device Settings		
Sequence Number	**Feature Enable Order**	**Notes**
*	Important note regarding configuration changes.	Any configuration should be done before video is running. Video must be stopped if a configuration change is required.
*	Voltage supply ramp order is independent.	No voltage sequencing required
1	Configuration Pins	Starting point (reboot if changed)
2	I2C wake time	From PWDNB pin
3	Main I2C / UART Config	
4	I2C / UART PT Config	
5	ADC	Enable temp/voltage monitoring as early as possible
6	GPIO Config	
7	GMSL PHY Config	
8	MIPI D-PHY Rx Config	
9	MIPI Lane Map/Swap and Polarity Inversion	
10	MIPI Deskew	Needed above 1.5 Gbps/lane
11	Video Pipe Config	
12	Line CRC Config	Enabled by default
13	Video Pixel CRC Config	Disabled by default
14	Pixel Doubling Config	
15	Virtual Channel/Datatype Override	
16	Line Fault	
17	Interrupt Handling (ERRB)	
18	Serializer is ready for video	
19	GMSL Link Lock	

Configuration

Overview

The forward video path of the MAX96717/F/R serializer is configured with the following programming:

- Pixel and Tunneling Mode
- 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.

Pixel and Tunneling Mode

The MAX96717/F/R supports both Pixel and Tunneling mode. Always ensure that the serializer and deserializer modes are matched.

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

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

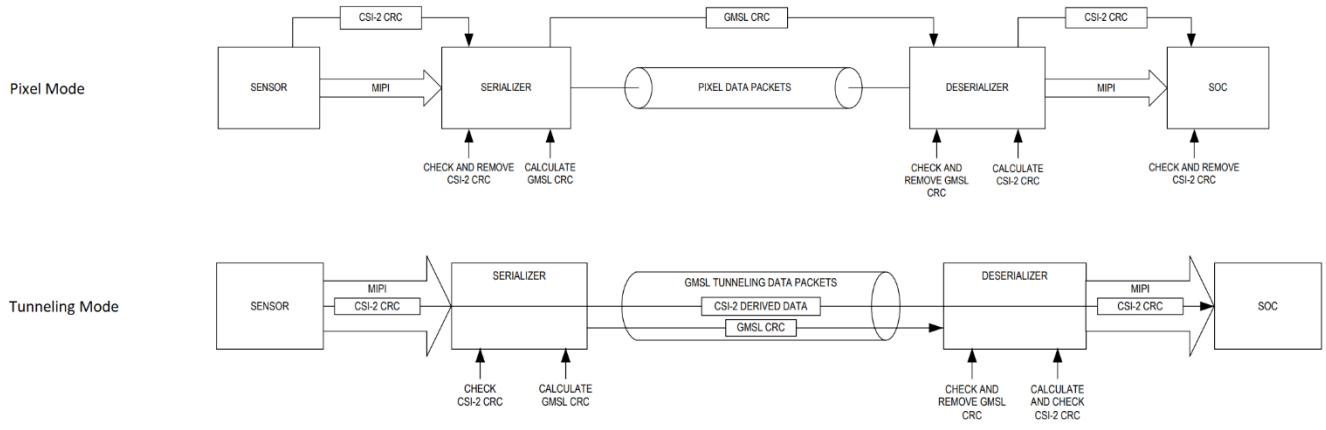


Figure 2. Tunneling and Pixel Mode

Table 3. Features Supported by Pixel vs Tunneling Mode

Feature	Pixel Mode	Tunneling Mode
Virtual Channel Reassignment*	Supported	Not Supported

End-to-End CRC coverage for video	Not Supported	Supported
Video Line CRC (LCRC)	Supported	Supported
GMSL Packet CRC (VID_PXL_CRC)	Supported	Supported
Line length >8k pixels at 24 BPP	Not Supported	Supported
16-Channel Virtual Channel Support	Supported	Supported

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

Link Initialization

Link initialization establishes the device link modes and link speeds. The MAX96717/F/R device family is a GMSL2 only, single port serializer that can support coax or shielded-twisted pair (STP) cables. The MAX96717 variant can transmit at 3Gbps or 6Gbps in the forward direction over the GMSL link while the F and R variants are only capable of 3Gbps. Using the registers found below, 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, see the table below. The selected configuration will become the new default on power up once the CFG pins have been set and the part has been power cycled.

Table 4. Basic Settings

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

Link Initialization Registers

Table 5. 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 will automatically lock upon connection. Pin #17 (MFP3) is used as LOCK indication by default. Bit 3 in register 0x0013 will assert 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 MAX96717/F/R has just one controller connected to two MIPI PHYs that can support up to 4-lane inputs.

In the only available 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, then only PHY2 will be sending data. The table below contains the MIPI PHY settings registers.

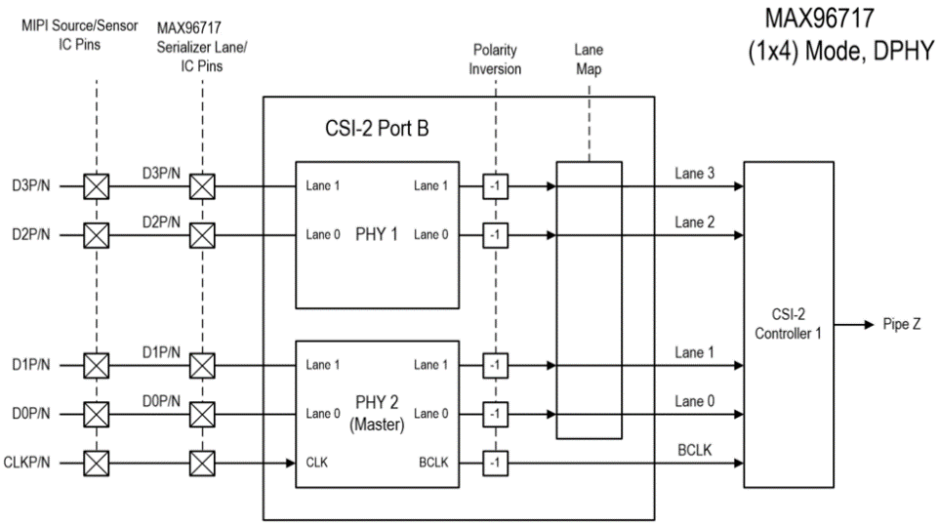


Figure 3. Inputs, PHY, and Controller Diagram

VC re-assignment is supported in Pixel mode. VC assignments can be altered via the CTRL1_VC_MAP{0:15} registers. For example, to change VC1 to VC7, write 0x07 in CTRL1_VC_MAP1[3:0]. VC re-assignment can be set with values between 0x0-0xF (0:15) because the MAX96717/F supports up to 16 VCs.

Note: The R variant (MAX96717R) only supports 4 virtual channels, not 16. Also note that the R variant only supports 600Mbps per lane on its input while the others support up to 2.5Gbps.

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 via the phy{1,2}_lane_map registers instead of fabricating a new board. Refer to Table 6. MIPI PHY Settings Registers 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.

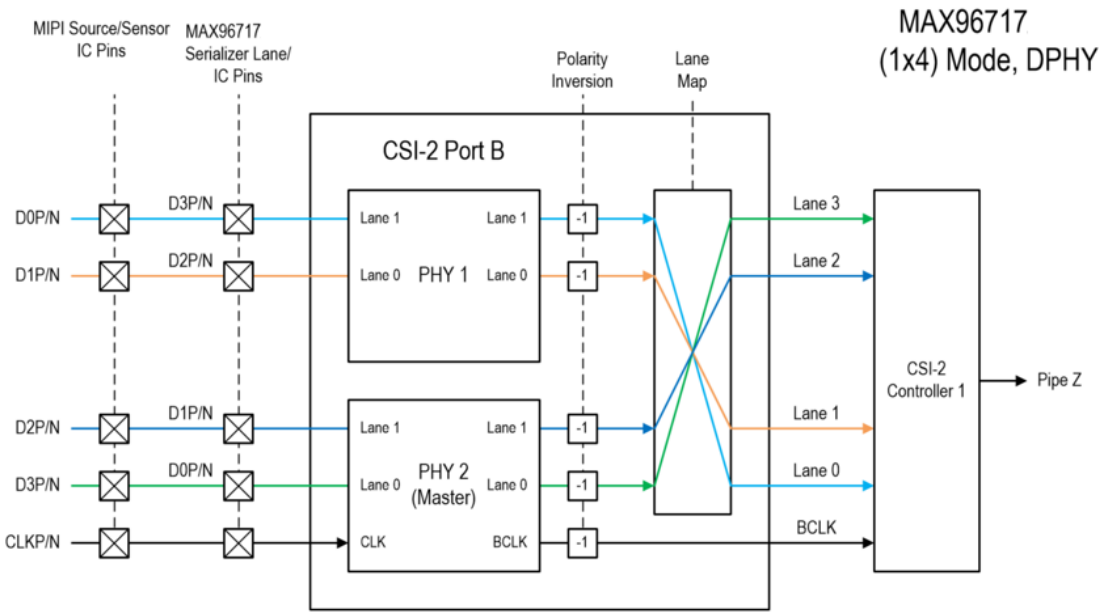


Figure 4. D-PHY Lane Swap Example

Lane Swap Programming Example

This example programs the lane swapping as shown in the figure.

```
# 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 1.5Gbps/lane and above. Deskew is optional for data rates lower than 1.5Gbps/lane. Deskew is initiated by the transmitter under CSI-PPI control. The MAX96717/F/R only has one bit for enabling the deskew calibration (ctrl1_deskewen), which can be found in REG 0x331 (see table).

MIPI PHY Settings Registers

Table 6. MIPI PHY Settings Registers

Register	Bitfield Name	Bits	Default Value	Description
0x0330	mipi_noncontclk_en	6	0b0	0 = enable MIPI continuous clock 1 = enable MIPI non-continuous clock
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	phy1_pol_map[1]	5	0b0	0 = normal polarity for data lane 3 1 = inverse polarity for data lane 3
0x0334	phy1_pol_map[0]	4	0b0	0 = normal polarity for data lane 2 1 = inverse polarity for data lane 2
0x0335	phy2_pol_map[2]	2	0b0	0 = normal polarity for clock lane 1 = inverse polarity for clock lane
0x0335	phy2_pol_map[1]	1	0b0	0 = normal polarity for data lane 1 1 = inverse polarity for data lane 1
0x0335	phy2_pol_map[0]	0	0b0	0 = normal polarity for data lane 0 1 = inverse polarity for data lane 0
0x0345-0x0347, 0x036C-0x036F, 0x0377-0x037F	ctrl1_vc_map{0,15}	7:4	0b0000	Virtual channel reassignment registers. Description found in last paragraph above table.

Pixel Clock Detect Check in Pixel Mode

CSI-2 can support multiple BPP rates on a single CSI-2 clock (i.e., variable PCLK). To accommodate this, the MAX96717/F/R will only generate an internal PCLK and assert PCLKDET (REG 0x112, bit 7) if both a valid CSI-2 clock and valid long-packets (pixel data) have been received. A PCLK will not be generated from a CSI-2 clock alone. PCLKDET will only assert if CTRL1_NUM_LANES (REG 0x331) matches the signal source lane count.

Pixel Clock Detect Check in Tunneling Mode

Internal PCLK is generated from only a valid CSI-2 clock. PCLKDET will assert once the serializer input detects a valid CSI-2 clock. Lane count does not contribute to PCLKDET in Tunneling mode.

Video Pipes and Datatype/Virtual Channel Filtering

The MAX96717/F/R has only one CSI-2 input port (B) and one video pipe (Z). This means the video routing is much simpler since there is only one path for video data to flow. Register 0x0308, shown in the Table 7. Video Pipe and Filtering Registers, contains the bits for enabling/disabling MIPI port B.

Default Mapping = MIPI Port B (PHY1/2) → MIPI Controller 1 → Video Pipe Z

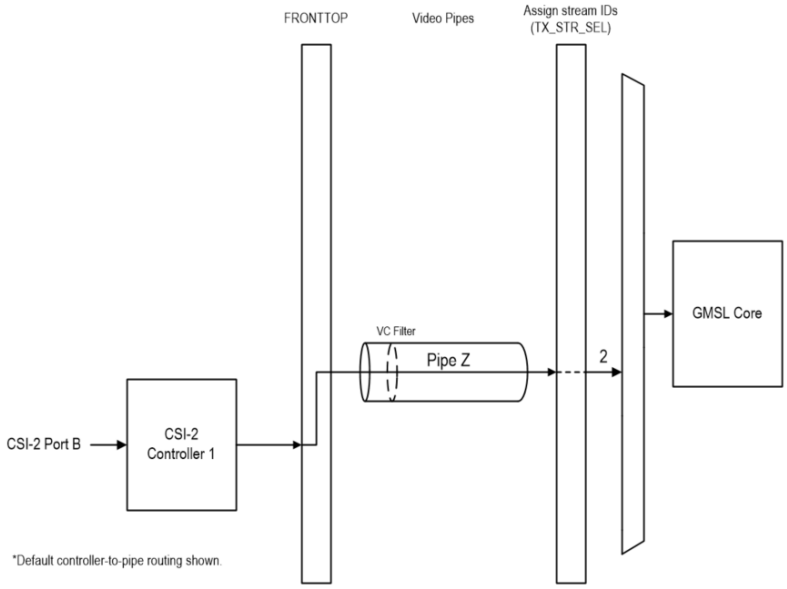


Figure 5. Default Controller to Pipe Mapping

-

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

The data within the serializer's controller can be filtered so that the user can control what data, if not all, gets serialized and sent across the GMSL link. The mem_dt_selz registers are used to filter the controller data by CSI-2 datatype code at the FRONTTOP before it reaches the video pipe. When using DT filtering, up to four data types can be routed from the controller to pipe Z. 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. In Pixel mode, when multiple data streams are transmitted over the same pipe with different virtual channels, the VC_SELZ_L and VC_SELZ_H bits must be set to represent the virtual channels present on that pipe. Each bit place represents a VC within these registers. For example, if VC_SELZ_L[0] = 1 and VC_SELZ_L[1] = 1, then pipe Z would expect to have VC 0 and 1 on the pipe. When a bit position is set to zero, that VC will be not be allowed to enter that pipe. The VC_SELZ_L/H registers on this part have a default value of 0xFF, meaning that all 16 VCs (0-15) will be 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. Video pipe Z has a dedicated stream ID that is 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 (i.e., multiple serializers connected to a single deserializer).

Video Pipe and DT/VC Filtering Registers

Table 7. Video Pipe and Filtering Registers

Register	Bitfield Name	Bits	Default Value	Decode
0x0308	START_PORTB	5	0b1	0 = CSI on port B disabled 1 = CSI on port B enabled
0x0308	CLK_SELZ	2	0b1	0 = Reserved (Port A does not exist) 1 = Port B selected for pipe Z
0x0002	VID_TX_EN_Z	6	0b1	0 = Video transmit on pipe Z disabled 1 = Video transmit on pipe Z enabled
0x005B	TX_STR_SEL[1:0]	1:0	0b10	00 = Stream ID for pipe Z is 0 01 = Stream ID for pipe Z is 1 10 = Stream ID for pipe Z is 2 11 = Stream ID for pipe Z is 3
0x0318, 0x0319, 0x03DC, 0x03DD	mem_dt{1,2,7,8}_selz[6]	6	0b0	0 = Datatype filtering disabled 1 = Datatype filtering enabled
0x0318, 0x0319, 0x03DC, 0x03DD	mem_dt{1,2,7,8}_selz[5:0]	5:0	0b000000	The value of bits 5:0 in this register should equal the data type ID of the data type you wish to allow onto the video pipe (e.g., RAW12 = 0x2C)
0x03C8, 0x03C9	mem_dt{3,4}_selz[7:6]	7:6	0b00	These two bits select the two LSBs of the virtual channel that is to be filtered onto the video pipe
0x03C8, 0x03C9	mem_dt{3,4}_selz[5:0]	5:0	0b000000	The value of bits 5:0 in this register should equal the data type ID of the data type you wish to allow onto the video pipe
0x03D1	mem_dt{3,4}_selz_en	1:0	0b00	0 = Disable filtering set in registers 0x3C8, 0x3C9

				1 = Enable filtering set in registers 0x3C8, 0x3C9
0x030D	VC_SELZ_L	7:0	0xFF	Bits 0-7 represent VC0-VC7, respectively. If a bit is high, it means that VC will be allowed onto the video pipe (e.g., if only bits 0 and 2 are HIGH, then only VC0 and VC2 are accepted).
0x030E	VC_SELZ_H	7:0	0xFF	This register works the same as register 0x30D except bits 0-7 represent VC8-VC15 respectively

Video Pipe 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 your use case before disabling heartbeat.

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

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

Table 8. Heartbeat Disable Register

Register Address (Video Pipe)	Bitfield Name	Bits	Default Value	Decode
0x112 (Pipe Z)	LIM_HEART	2	0b0	0b0: Heartbeat enabled during blanking 0b1: Heartbeat disabled during blanking

Complete Configuration Examples

1. Simplest configuration using Tunneling mode and default register settings:

a. Any DT or VC will be transmitted over the GMSL link at 6Gbps in Tunneling mode with a stream ID of 2.

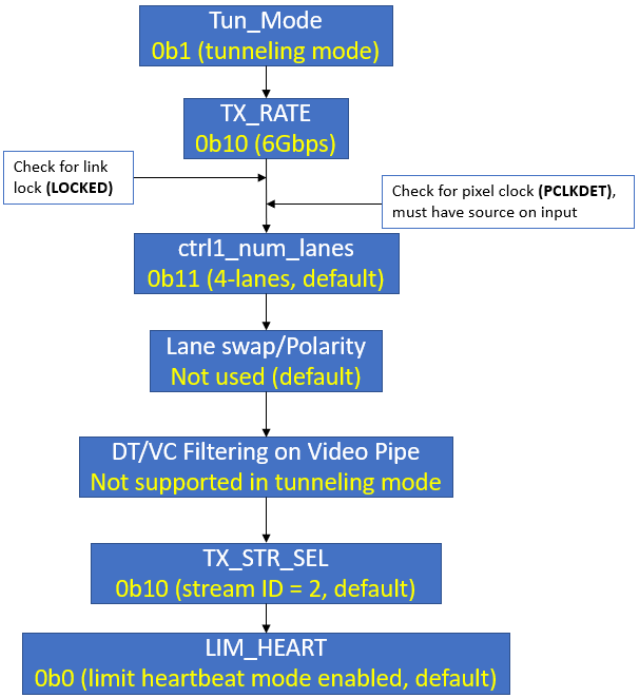


Figure 6. Simple Configuration Example

2. Complicated configuration using pixel mode with DT and VC filtering:
    a. Only a 2-lane, RAW12 input assigned as VC1 will be 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 will be transmitted over the GMSL link at 3Gbps with a stream ID of 1.

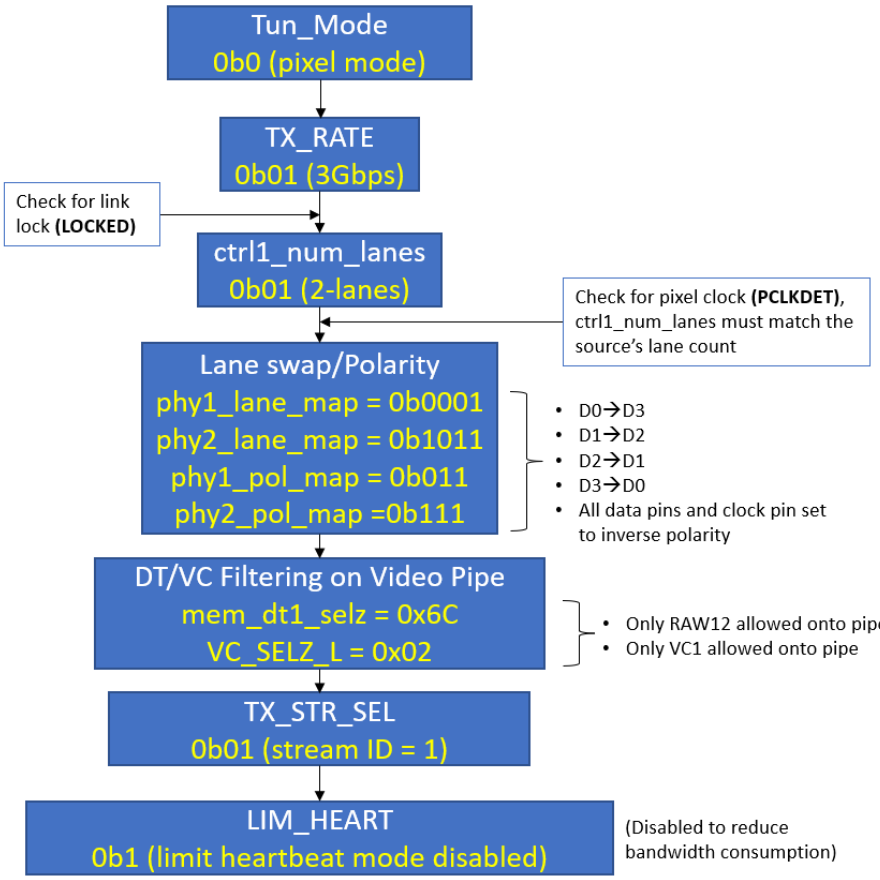


Figure 7. Complex Configuration Example

Extended Virtual Channels

Virtual channels (VCs) allow the serial link system to differentiate video inputs by the VC assigned to them. When extended VCs are enabled, the standard 2-bit VC selection is extended to 4-bit, increasing the number of available VCs to 16 for D-PHY applications. The increase in available VCs allows downstream systems to support more cameras. MAX96717 and MAX96717F can have a total of 16 VCs. MAX96717R does not have extended VCs so it can only have a total of 4 VCs.

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

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

Extended Virtual Channels Register Examples

The table below demonstrates the MSB and LSB of the extended virtual channel for the MAX96717 and how they may be remapped on the serializer. Each controller mapping 0-15 maps the input virtual channels 0-15 to the new selected virtual channel from bits [7:4] of the respected registers.

Table 9. New Virtual Channel Mapping with MAX96717 (Extended VC)

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

Extended Virtual Channel Registers

Table 10. Extended Virtual Channels Registers

Register	Bits	Default Value	Description
0x0330	7	0	**Ctrl1_vcx_en:** 0 = VC extension Disabled 1 = VC extension Enabled
0x0331	5	0	**Ctrl1_vc_map_en:** 0 = VC Remapping Disabled 1 = VC Remapping Enabled
0x0345	7:4	0x00	**Ctrl1_vc_map0:** Bits [7:4]: New Virtual Channel for VC0
0x0346	7:4	0x00	**Ctrl1_vc_map1:** Bits [7:4]: New Virtual Channel for VC1
0x0347	7:4	0x00	**Ctrl1_vc_map2:** Bits [7:4]: New Virtual Channel for VC2
0x036C	7:4	0x00	**Ctrl1_vc_map3:**

			Bits [7:4]: New Virtual Channel for VC3
0x036D	7:4	0x00	**Ctrl1_vc_map4:** Bits [7:4]: New Virtual Channel for VC4
0x036E	7:4	0x00	**Ctrl1_vc_map5:** Bits [7:4]: New Virtual Channel for VC5
0x036F	7:4	0x00	**Ctrl1_vc_map6:** Bits [7:4]: New Virtual Channel for VC6
0x0377	7:4	0x00	**Ctrl1_vc_map7:** Bits [7:4]: New Virtual Channel for VC7
0x0378	7:4	0x00	**Ctrl1_vc_map8:** Bits [7:4]: New Virtual Channel for VC8
0x0379	7:4	0x00	**Ctrl1_vc_map9:** Bits [7:4]: New Virtual Channel for VC9
0x037A	7:4	0x00	**Ctrl1_vc_map10:** Bits [7:4]: New Virtual Channel for VC10
0x037B	7:4	0x00	**Ctrl1_vc_map11:** Bits [7:4]: New Virtual Channel for VC11
0x037C	7:4	0x00	**Ctrl1_vc_map12:** Bits [7:4]: New Virtual Channel for VC12
0x037D	7:4	0x00	**Ctrl1_vc_map13:** Bits [7:4]: New Virtual Channel for VC13
0x037E	7:4	0x00	**Ctrl1_vc_map14:** Bits [7:4]: New Virtual Channel for VC14
0x037F	7:4	0x00	**Ctrl1_vc_map15:** Bits [7:4]: New Virtual Channel for VC15

Extended Virtual Channels Programming Example

This example enables the extended virtual channels on the MAX96717 and remaps virtual channels 0 and 1 to be 5 and 6.

```
# Turn on virtual channel extension
0x80, 0x331, 0xB0
# Enable VC remapping
0x80, 0x330, 0x20
# VC remap of VC0 to VC5
0x80, 0x345, 0x50
# VC remap of VC1 to VC6
0x80, 0x346, 0x60
```

Software Override

Overview

The software override is used to manually override the video data type (DT) (i.e., 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 the MIPI controller. Overriding the DT and VC information is used for easier MIPI controller mapping on the deserializer side. Refer to Table 11 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:

- 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 11. Software Override Registers

Register	Bitfield Name	Bits	Default Value	Decode
0x031E	soft_dtz_en	7	0b0	0 = Data type software override disabled on pipe Z 1 = Data type software override enabled on pipe Z
0x031E	soft_vcz_en	6	0b0	0 = Virtual channel software override disabled on pipe Z 1 = Virtual channel software override enabled on pipe Z
0x031E	soft_bppz_en	5	0b0	0 = BPP software override disabled on pipe Z 1 = BPP software override enabled on pipe Z
0x031E	soft_bppz[4:0]	4:0	0b11000	These bits should be set to the smallest input BPP (before padding and after doubling)
0x0320	soft_vcz[1:0]	5:4	0b00	00 = VC0 01 = VC1 10 = VC2 11 = VC3
0x0323	soft_dtz[5:0]	5:0	0b110000	These bits should be set to the appropriate data type ID

Input BPP Manipulation (Pixel Mode Only)

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

Double Mode

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

reduces the internal PCLK and increases the 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, 0x31–0x37, or 0x10–0x11, can alternatively be combined and transmitted by the serializer as 24-bit data. Set ctrl1_mode_UDT = 1 to treat these data types as 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 has a soft_bppz bitfield that must be set to the new BPP (e.g., 8->16, 8 ->24, 10->20, 12->24) and a soft_bpp_en bit.

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

Table 12. Double Mode Registers

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

Zero Padding

Pixel data being received by MAX96717/F/R can be zero padded as it enters video pipe Z up to a resulting BPP of 16. With zero padding, an input with multiple BPP rates can be routed through video pipe Z if the following conditions are met:

1. 8 ≤ BPP ≤ 16 for all incoming BPP rates that are routed to the video pipe Z
    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 will be dedicated to sending zeros instead of the original CSI-2 data. Ensure system bandwidth requirements can be met by using the calculations shown in the GMSL2 User Guide Bandwidth section.
3. Video pipe Z's PCLK Drift detection must be disabled.

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

Table 13. 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	Enable PCLK frequency drift detection, resets video pipeline upon error and reports it

Double Mode and Zero Padding Example

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

- AUTO_BPP = 0 – Do not set BPP based on CSI-2 header (Pipe Z).
- BPP = 0x10 – Force Pipe Z 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 Pipe Z.

I2C Control Channels

Overview

The MAX96717 and MAX96717F feature one main I2C channel and two pass-through I2C channels. Due to MFP limitations, it is possible to use at most two of the three channels in one application.

When making changes to any of the serializer or deserializer's I2C configuration, such as enabling or disabling an I2C port, a 10µs 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 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 master microcontroller resides (For MAX96717 applications, the master microcontroller typically resides on the deserializer side).

I2C Pass-Through Channel

Note: The MAX96717R variant features only the main I2C control channel and does not include I2C pass-through channel access.

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

Port Access and Routing

The MFPs shown in the table below are used for the I2C control and pass-through channels. This table does not apply to the MAX96717R variant.

Table 14. MFPs for I2C

MFP Pin	Main UART Function	Other UART Functions	Default Function	Notes
MFP7	SDA1		GPIO7	
MFP8	SCL1		GPIO8	
MFP9	SDA	SDA2	SDA_RX	SDA (I2C) or RX (UART) functionality determined by CFG0 status on power-up
MFP10	SCL	SCL2	SCL_TX	SDL (I2C) or TX (UART) functionality determined by CFG0 status on power-up

On power-up, the device should be set to I2C mode via the CFG0 latch. The function names in the above MFP table and ensuing I2C sections assume the device has been configured for I2C mode.

By default, the main I2C control channel lines are brought out on MFP9 and MFP10 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 I2C functionality via the DIS_LOCAL_CC and DIS_REM_CC fields in register 0x1.

The user can bring out the first pass-through I2C channel on MFP7 and MFP8 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 MFP9 and MFP10 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 for CRC + Message Counter

Note: This section does not apply to the MAX96717R variant

The MAX96717 provides two additional functional safety features by adding an optional CRC packet to either side of the link and an optional message counter for monitoring both write and read transactions. Both features are available only to the serializer's main I2C control channel for register read/write transactions. They are disabled by default and need to be enabled at register level.

CRC over I2C

The CRC feature is used to detect corrupt data written on the serializer's main I2C control channel. Each I2C transaction has a corresponding CRC packet associated with it. If a CRC error is detected, the command is not executed, and the transaction is reported as NACK. The CRC feature is enabled by setting fields CC_CRC_EN and CC_CRC_MSGCNTR_OVR in register 0x4.

Message Counter over I2C

The message counter feature is used to detect missing or repeated I2C transactions. For every transaction, the message counter is incremented accordingly while a copy of the counter is maintained on both ends of the I2C link. If a mismatch between copies is found, then the transaction is rejected. The message counter feature is enabled by setting fields CC_MSGCNTR_EN and CC_CRC_MSGCNTR_OVR in register 0x4.

I2C Registers

Table 15. MAX96717/MAX96717F I2C Registers (not applicable to MAX96717R)

Register	Bits	Default Value	Description
0x0001	7:4	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) Bit [5]: Disable main I2C Control Channel connection to SDA and SCL pins Bit [4]: Disable access to remote device control-channel over GMSL2 connection

0x0004	4:2	0x18	**Enable CRC and Message Counter Register:** Bit [4]: Enable I2C message counter. **Note: Only active when Bit [2] is also set to 1.** Bit [3]: Enable I2C CRC packeting. **Note: Only active when Bit [2] is also set to 1.** Bit [2]: Enable manual override of I2C CRC or message counter configuration. If set to a 0 then CRC and message counter features will be disabled.
0x0006	4	0x80	**I2C Selection Register:** Bit [4]: Enables I2C when set to a 1 or UART when set to a 0. Note: This bit is set according to the CFG0 pin value on power-up. Writing to this register is not recommended.
0x1D00	3	0x00	**Enable CRC Computation Register:** Bit [3]: Compute register CRC after every I2C register write
0x1D08	0	0x00	**Message Counter Reset Register:** Bit [0]: Reset Message Counter value to 0
0x1D09	1:0	0x00	**CRC Reset Register:** Bit [1]: Resets Message Counter error count to 0 Bit [0]: Resets CRC error count to 0.
0x1D0A	7:0	0x00	**Read CRC Value Register:** Bits [7:0]: CRC value for the last write transaction
0x1D0B	7:0	0x00	**Read Message Counter Low Bits Register:** Bits [7:0]: Low bits of current message counter value
0x1D0C	7:0	0x00	**Read Message Counter High Bits Register:** Bits [7:0]: High bits of current message counter value

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
```

Enable CRC and Message Counter on main I2C Control Channel

This example enables the CRC and Message Counter features on the main I2C Control Channel.

```
# Enables CRC and Message Counter features along with corresponding override enable
0x80,0x0004,0x1C
```

I2C Broadcasting

Overview

When transmitting to a multi-link input deserializer, each device on the serializer side will require 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 will ACK for all remote GMSL I2C ports.

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

An example of I2C broadcasting is discussed in the ensuing section. 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 of a MAX96717 serializer at the default I2C address 0x80 and an image sensor at address 0x20.

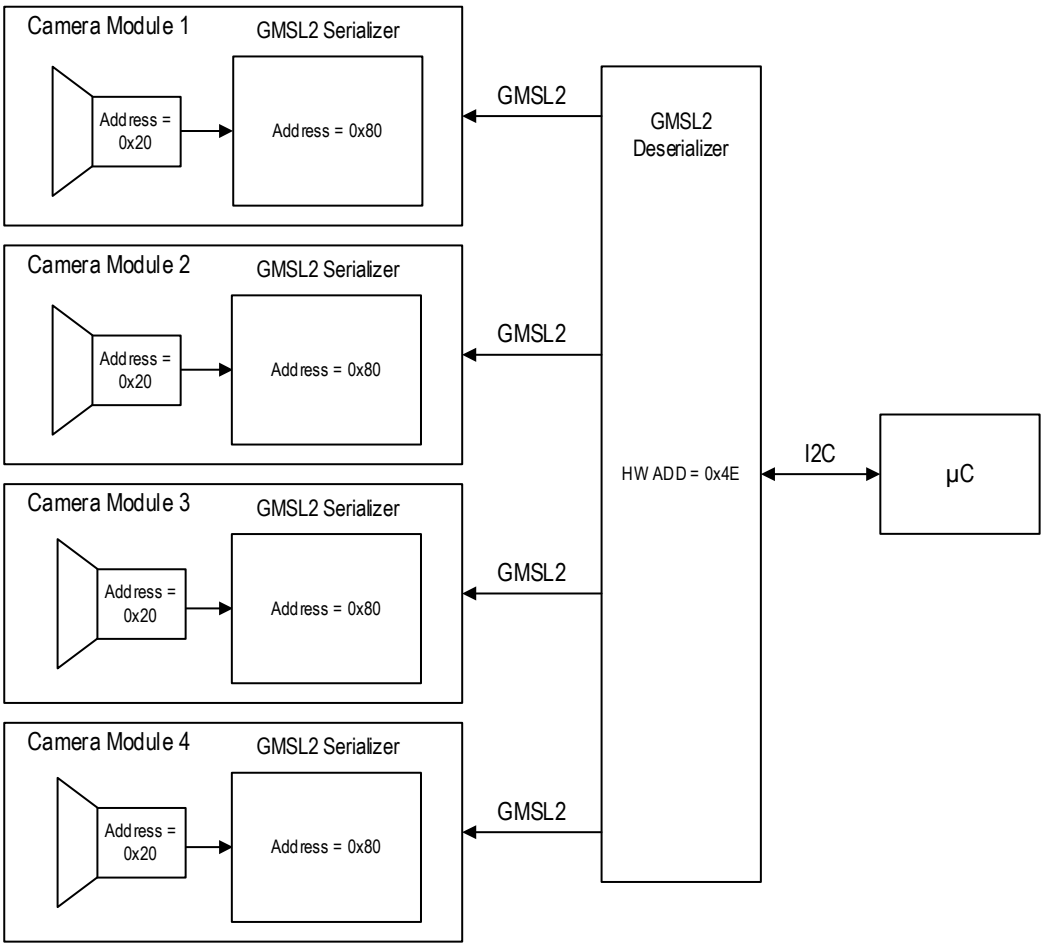


Figure 8. I2C Interfaced Camera-Module System with Default Address Settings

I2C Broadcasting Technique

The I2C broadcasting technique allows the user 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 I2C access, meaning no other device with the same address should be connected to the I2C data line.
- Change the serializer address to a unique address.
- Modify the first I2C address translation register with a common source address but the unique destination address. This is to streamline our interface with the serializer.
- Modify the second I2C address translation register with a unique source address but the default image sensor addresses for the destination address. This is to streamline our interface with the image sensor.
- Repeat this process for each camera serializer module.
- When making changes to any of the serializer or deserializer's I2C configuration, such as enabling or disabling an I2C port, at least a 10µs delay from the write acknowledgement (ACK) to the next transaction is required.

I2C Broadcasting GMSL2 Use Case Example

The procedure for the I2C broadcasting example is described below.

1) Isolate camera module 1, by enabling deserializer Link A only for remote I2C 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 will be 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 will be sent to address 0x20 instead.
5) Isolate camera module 2, by enabling deserializer Link B only for remote I2C 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 will be 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 will be sent to address 0x20 instead.
9) Isolate camera module 3, by enabling deserializer link C only for remote I2C 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 will be 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 will be sent to address 0x20 instead.
13) Isolate camera module 4, by enabling deserializer Link D only for remote I2C 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 will be 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 will be sent to address 0x20 instead.
17) Now enable all the links for remote main I2C port access.
18) All devices should be present on the I2C bus. Continue with any additional required system configuration.

The same camera module system is shown below with translated addresses. A summary of the changes made are also shown in the tables that follow.

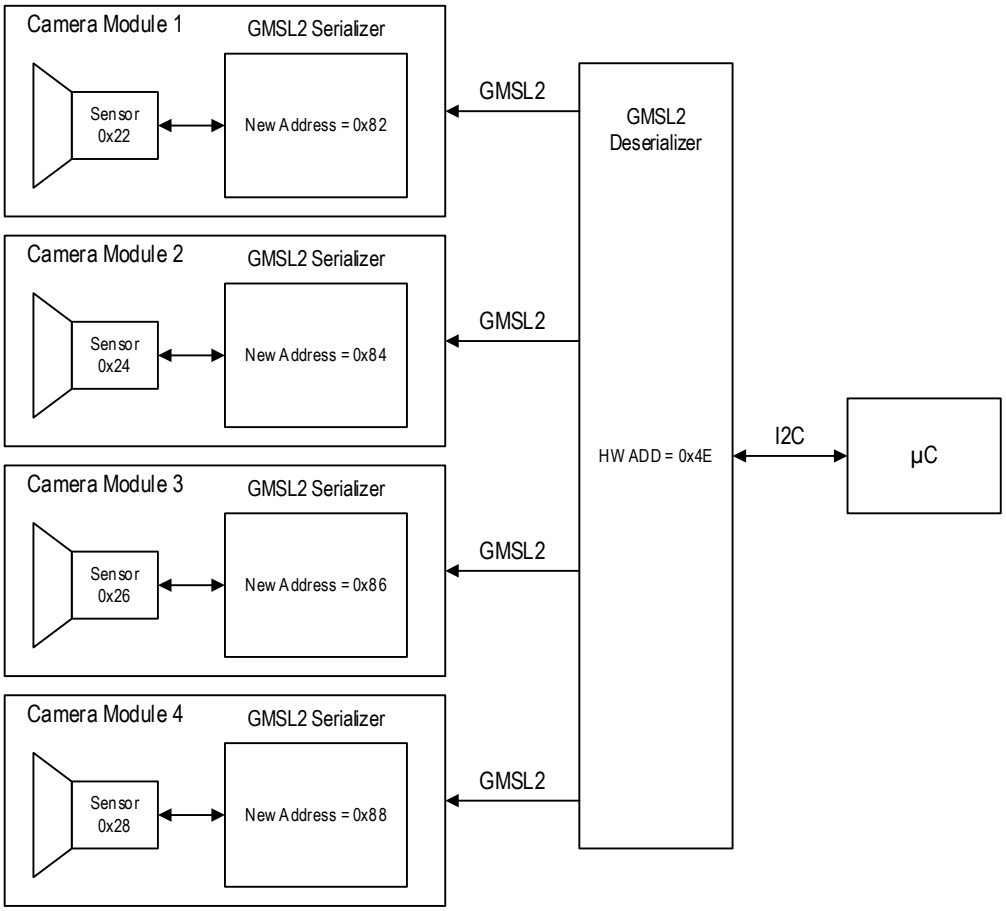


Figure 9. Camera-Module System with Translated Address Settings

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

Table 16. 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 17. 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 as shown in the figure.

```
# 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 MAX96717 and MAX96717F feature one main UART control channel and two pass-through UART channels. Due to MFP limitations, it is possible to use at most two of the three channels in one application.

Note: The MAX96717R variant does not feature UART capability and is not applicable to the UART Control Channels section.

When making changes to any of the serializer or deserializer's UART configuration, such as enabling or disabling a UART port, at least a 10µs 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 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 master microcontroller resides (for MAX96717 applications, the master 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 will 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 which are 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 shown in the table below are used for the UART control and pass-through channels.

Table 18. MFPs for UART

MFP Pin	Main UART Function	Other UART Functions	Default Function	Notes
MFP7	RX1		GPIO7	

MFP8	TX1	MS	GPIO8	Mode Select (MS) is used as a hard trigger to run main UART channel (TX/RX) in bypass mode
MFP9	RX	RX2	SDA_RX	SDA (I2C) or RX (UART) functionality determined by CFG0 status on power-up
MFP10	TX	TX2	SCL_TX	SCL (I2C) or TX (UART) functionality determined by CFG0 status on power-up

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

By default, the main UART control channel lines are brought out on MFP9 and MFP10 for RX and TX, respectively. The user 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.

Enabling UART Bypass Mode via Register Setting (Soft-Bypass)

UART Bypass mode can be enabled via register setting by first setting field BYPASS_EN in register 0x48. Next the user should 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 will remain in Bypass mode until the next power-cycle.

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

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

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

Enabling the UART Pass-Through Channels

The user can bring out the first pass-through UART channel on MFP7 and MFP8 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 MFP9 and MFP10 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.

UART for CRC + Message Counter

The MAX96717 provides two additional functional safety features by adding an optional CRC packet to either side of the link and an optional message counter for monitoring both write and read transactions. Both features are available only to the serializer's main UART control channel (not the pass-through channels). They are disabled by default and need to be enabled at register level.

CRC over UART

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

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

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

Message Counter Over UART

The message counter feature is used to detect missing or repeated UART transactions. For every transaction, the message counter is incremented accordingly while a copy of the counter is maintained on both ends of the UART transaction. If a mismatch between copies is found, then the transaction is rejected. The message counter feature is enabled by setting fields CC_MSGCNTR_EN and CC_CRC_MSGCNTR_OVR in register 0x4.

Table 19. MAX96717/MAX96717F UART Registers (not applicable to MAX96717R)

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
0x0004	4:2	0x18	**Enable CRC and Message Counter Register:** Bit [4]: Enable UART message counter. **Note: Only active when Bit [2] is also set to 1.** Bit [3]: Enable UART CRC packeting. **Note: Only active when Bit [2] is also set to 1.** Bit [2]: Enable manual override of UART CRC or message counter configuration. If set to a 0 then CRC and message counter features will be disabled.

0x0006	4	0x80	**UART Selection Register:** Bit [4]: Enables UART when set to a 0. **Note: This bit is set according to the CFG0 pin value on power-up. Writing to this register is not recommended.**
0x0048	5:0	0x42	**UART Bypass Mode Control Register:** Bit[5]: Enable UART bypass mode control by remote GPIO pin (Function **MS** on **MFP8**) 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
0x1D08	0	0x00	**Message Counter Reset Register:** Bit [0]: Reset Message Counter value to 0
0x1D09	1:0	0x00	**CRC Reset Register:** Bit [1]: Resets Message Counter error count to 0 Bit [0]: Resets CRC error count to 0.
0x1D0A	7:0	0x00	**Read CRC Value Register:** Bits [7:0]: CRC value for the last write transaction
0x1D0B	7:0	0x00	**Read Message Counter Low Bits Register:** Bits [7:0]: Low bits of current message counter value
0x1D0C	7:0	0x00	**Read Message Counter High Bits Register:** Bits [7:0]: High bits of current message counter value

Enable Pass-Through UART Channel 1

This example enables pass-through UART Channel 1.

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

Enable CRC and Message Counter on Main UART Control Channel

This example enables the CRC and Message Counter features on the main UART Control Channel.

```
# Enables CRC and Message Counter features along with corresponding override enable
0x80,0x0004,0x1C
```

Serial Peripheral Interface (SPI)

Overview

SPI is available on the MAX96717 and MAX96717F, but not on the MAX96717R. Unlike I2C 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 (e.g. 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.

The figure below shows the GMSL SPI architecture. On each side of the link the GMSL devices become part of a master-slave pair and have transmit and receive buffers inside.

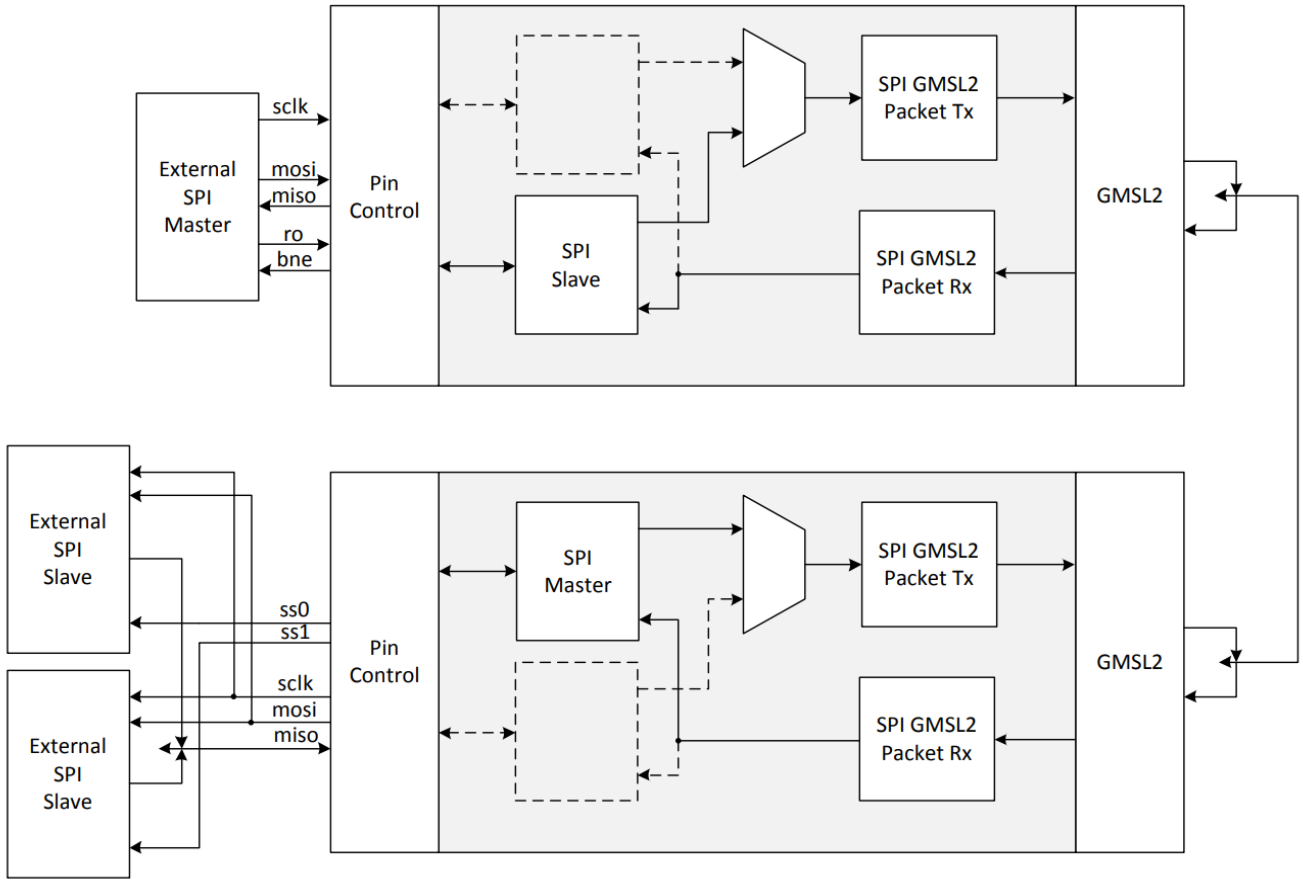


Figure 10. SPI Architecture

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, then the SPI pin may not work correctly. The MAX96717 may be confused if multiple features are enabled at the same time on the MFP pin. Be sure to check the datasheet 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 master/generator not connected, or not pulling on the pins up or down. Otherwise, the part will boot into a configuration that is unwanted.

SPI Setup Registers in MAX96717/F

In the table below, the most critical setup registers are shown. There are other settings within the 0x170-0x178 register range that may be important depending on the arrangement (which is the master, and which is the slave, clock timings, etc.). Refer to the example script in the section below for examples with those registers.

Table 20. 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: 37his example does not apply to the MAX96724, since 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
```

The below figures show MFP pins being used for SPI after running the script.

Serializer | Deserializer

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

Figure 11. MFP Pins for Serializer

Serializer | Deserializer

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

Figure 12. MFP Pins for Deserializer

SPI Example Using GMSL GUI and Evaluation Boards

- Make sure that the SPI output will be 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 master/generator and RO source from circuit
- Power up EV boards (make sure VDDIO on the EV board of the SPI external master/generator side matches the SPI external master/generator voltage)
- Start GMSL GUI
- Load GMSL script (.csv)
- Reconnect SPI external master/generator and RO source
- Put RO high and write A0 A4. *Explanation: A0-A3 are for SPI ID selection, mostly useful when using quad deserializers. A4 asserts SS1, A5 asserts SS2, and A6 de-asserts both SS1 and SS2.*
- As a best practice, check BNE to ensure that the buffer is empty. If BNE is high, there is data in the RX buffer that is ready to be read by the external SPI master/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 MAX96717's TX and RX buffers have a 32-byte 16-bytes capacity, respectively. Figure 13 and Figure 14 show the SPI oscilloscope probes on the SPI clock and data output (the receiving end at the deserializer). The data is flowing consistently without video running. In the case of 90% video, there are some intermittent pauses. The 32-byte buffer compensates for this scheduling delay and makes continuous streaming of the data possible.

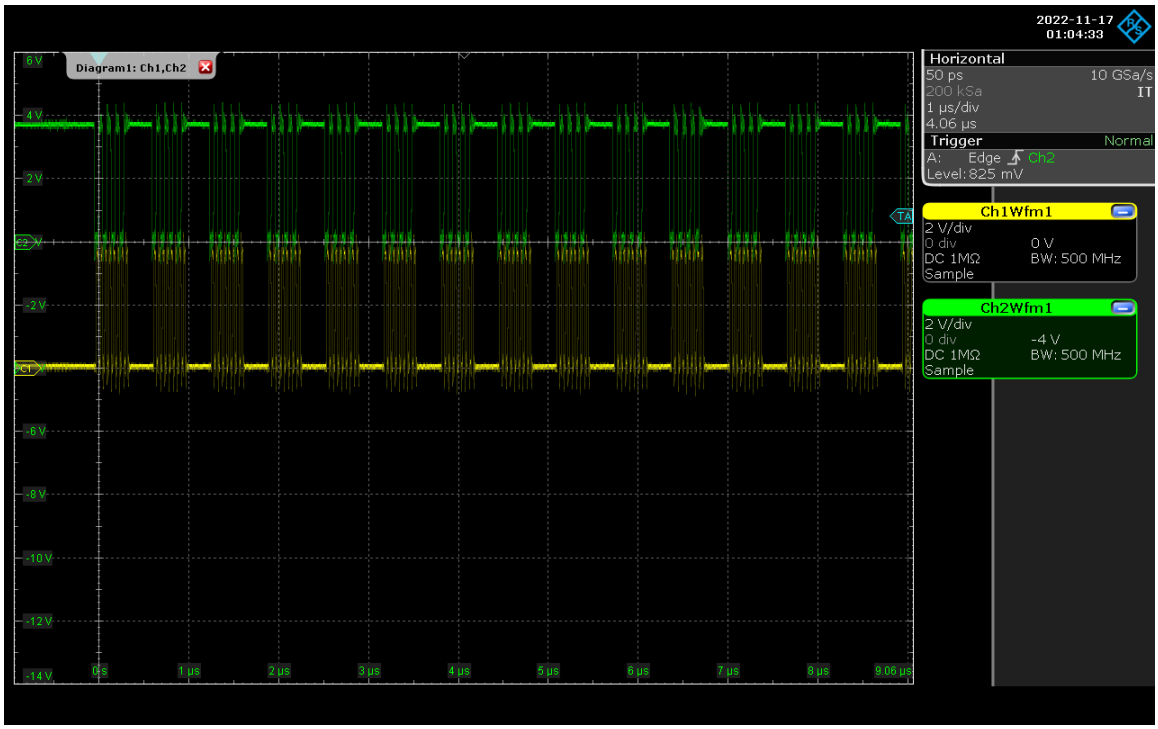


Figure 13. SPI Clock and Data at Final Output (at External SPI Slave), No Video on GMSL Link

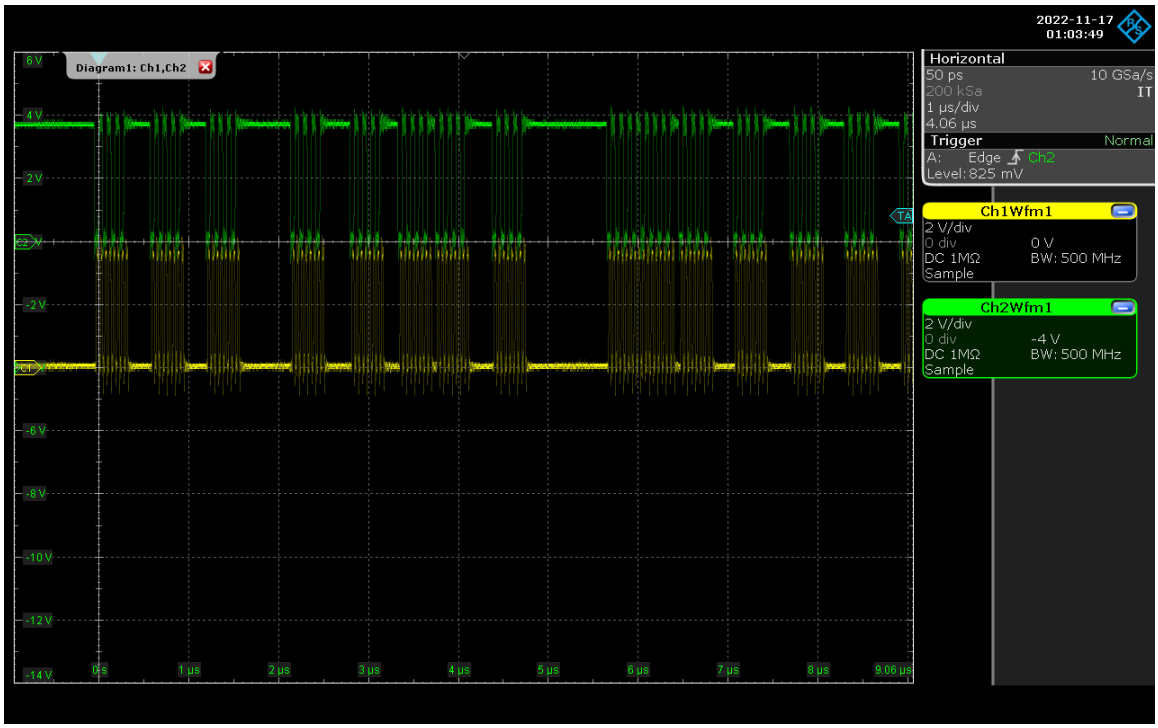


Figure 14. SPI Clock and Data at Final Output (at External SPI Slave), 92% Video on GMSL Link

Table 21. 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
Min 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 will stream continuously and without having the SPI external master/generator read back any of the values. However, the techniques in this section are additional steps that are 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 will send the data out on the MFP pins. It will also send the data back across the link so that the SPI external master/generator can read back the data.

State of RO (Read Only) pin dictates direction of data movement

- RO = 0: Data transmitted between master and slave via MOSI
- RO = 1: Data transmitted between slave and master via MISO, BNE will be high if there are bytes in this buffer to be read back.

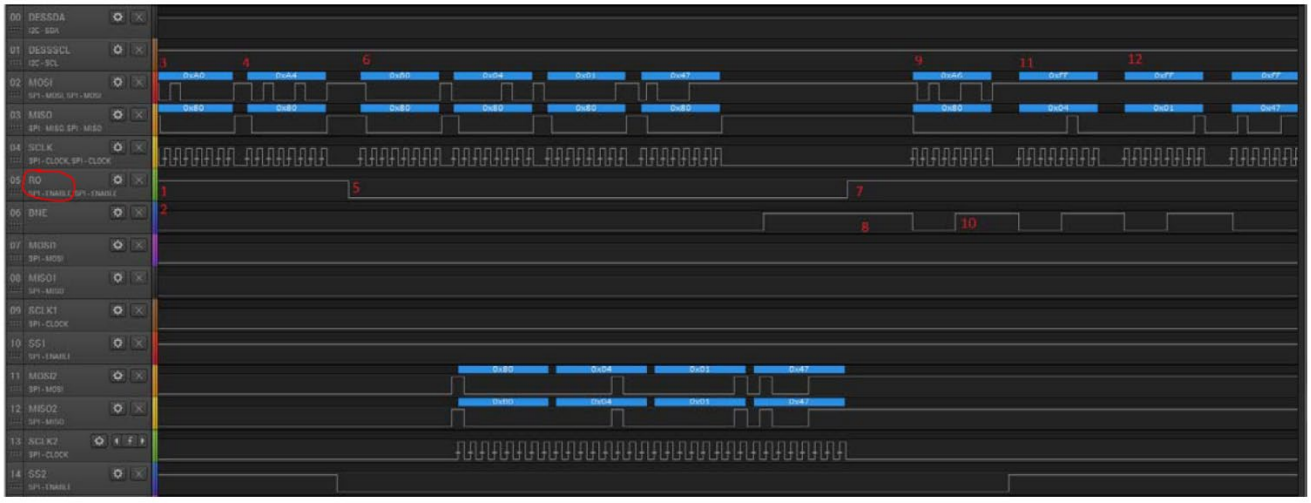
Note that the word “read” in the name of the RO pin does not mean that it is an output pin; it is, in fact, an input pin that is toggled externally high or low depending on what operation is desired.

It is recommended to limit the amount of bytes in transit (bytes that have been sent but not received back) to 16. The SPI external master/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 will be sent properly but it will not be possible to easily be sure that the data was sent properly.

An example of sending data in a group of four bytes can be seen in the figure below. 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 master/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 master/generator. Note BNE is high as the bytes are available (there is a time delay for them to come back).



GMSL2 SPI Implementation

Figure 15. 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).

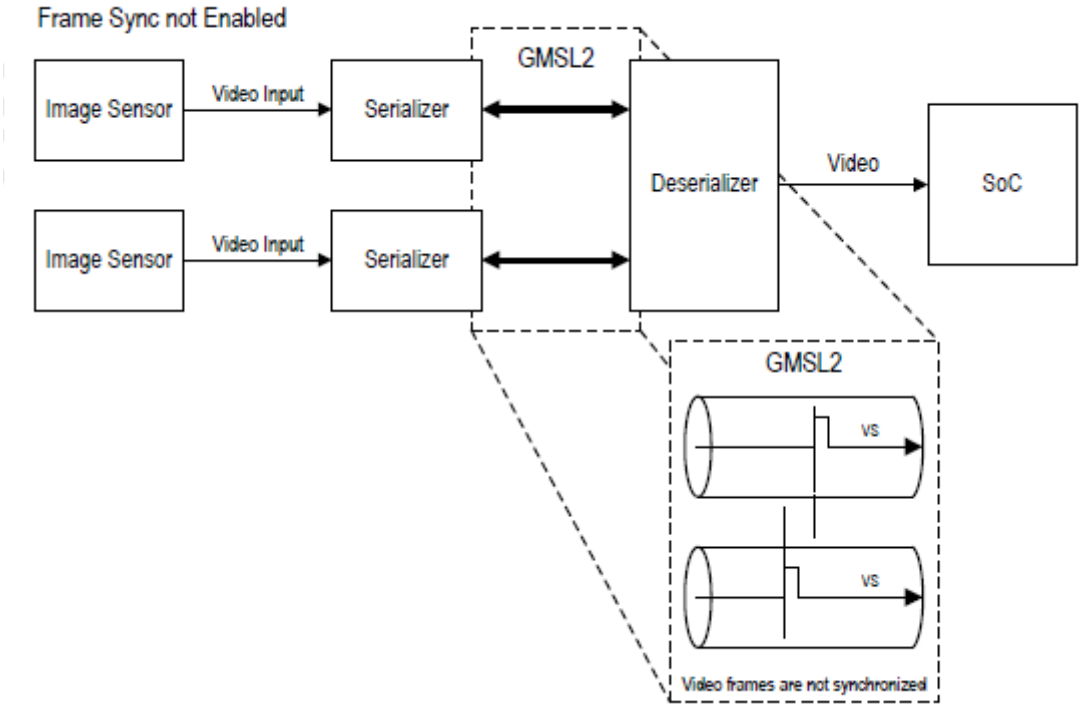


Figure 16. Frame Alignment (Without Frame Sync)

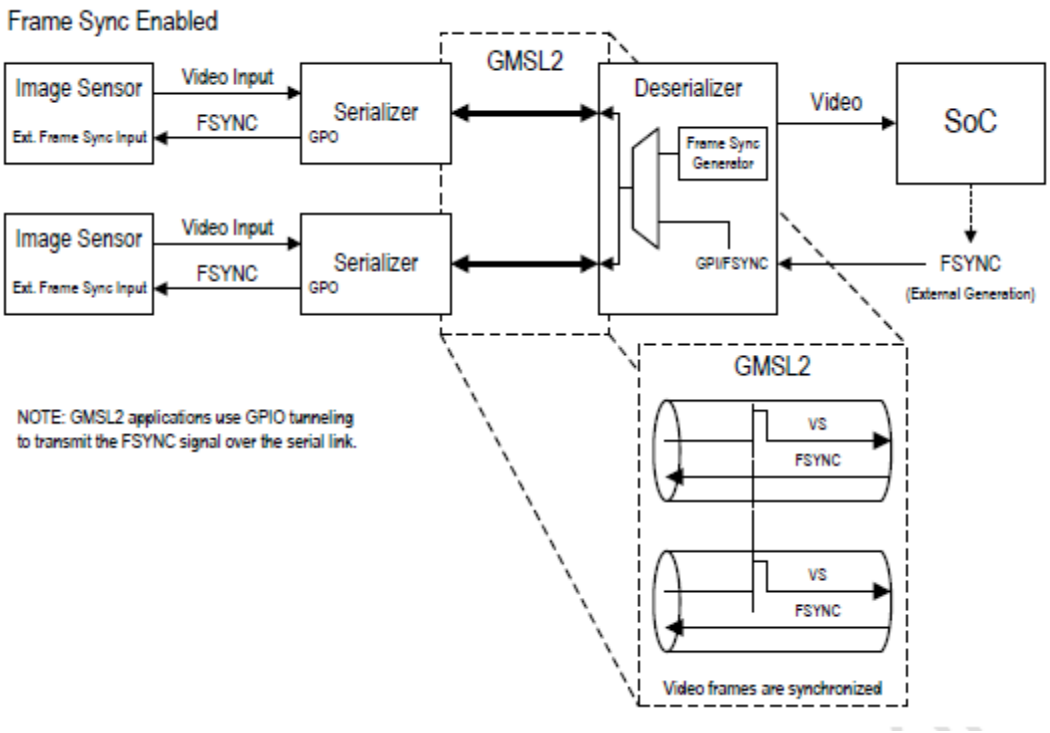


Figure 17. Frame Alignment (Frame Sync Enabled)

Configuration

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

Programming Examples

External 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.
```

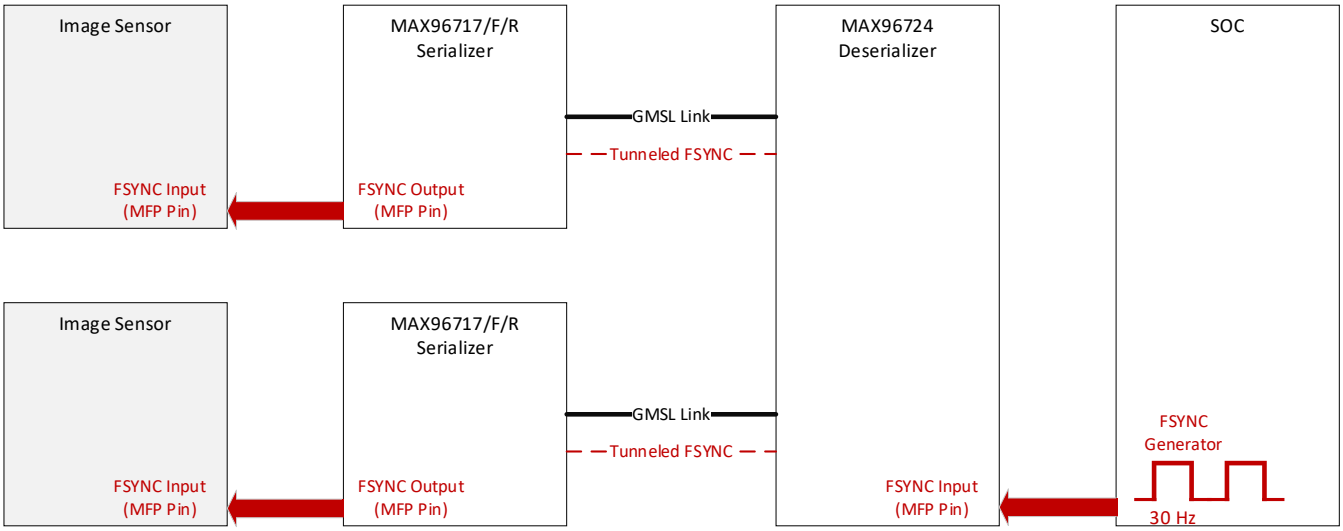


Figure 18. Frame Sync Application Diagram

Power Manager and Sleep Mode

Overview

MAX96717/F/R includes an integrated power manager that ensures the reliable and efficient operation of various power functions. The power manager controls the internal switched supply domains during the full sequence of power states so that the device powers up and down smoothly. During power-up, the power manager guards the device until the internal supplies have been validated and the digital core assumes normal operations. In all power modes, the power manger monitors power supplies for under- and over-voltage conditions. In Sleep mode, the power manager minimizes current consumption and can quickly restore device configurations after waking up. Please note that the MAX96717R has the power manager but does not have Sleep mode.

Table 22. Power Manager and Sleep Mode Availability for MAX96717 Family

Part Number	Power Manager	Sleep Mode
MAX96717	Supported	Supported
MAX96717F	Supported	Supported
MAX96717R	Supported	Not Supported

Device Power Operation

The part uses three common power rails (VDD, VDD18, and VDDIO) and an integrated internal VDD regulator.

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

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

Power Supplies

MAX96717/F/R share a common set of power supply voltages that power universal functions such as the digital core, GMSL link circuitry, and GPIO. The following is a summary of the power supplies:

- **VDD** – The input voltage to the VDD rail can be between 1.0 and 1.2 V. An internal LDO regulates the voltage to 1.0 V.
- **VDD18** – 1.8V power rail.
- **VDDIO** – 1.8 V or 3.3 V I/O power rail for I/O. 1.8 V only for MAX96717R.
- **VDD_sw** (CAP_VDD pin) – Internal 1V power rail that powers the digital core logic.

External power is supplied directly to VDD, VDD18, and VDDIO, 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” VDD18 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 VDD18 domain so that device configuration and register settings can be saved and restored.

The state diagram for the power manager is shown in the figure below.

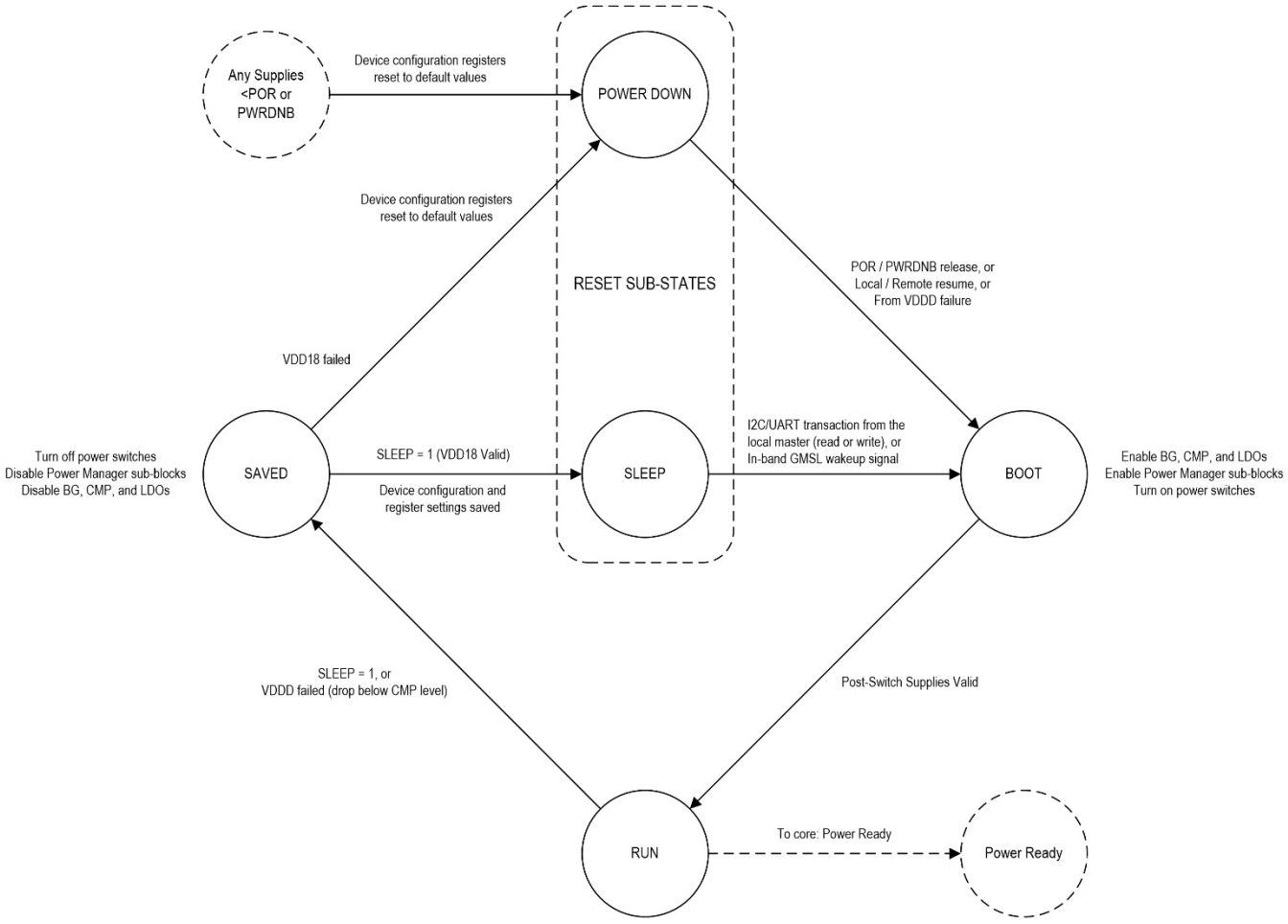


Figure 19. Power Manager State Diagram

Reset (Power Down/Sleep)

Power down and sleep are two sub-states of the reset state.

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

Sleep is a low-power consumption state that preserves the configurations and settings saved in the previous state and enables a much faster return to running operation than from power down. When the device is in the run state, the system (µC/SoC) can initiate sleep state with an I2C/UART command (SLEEP = 1). Sleep mode is entered automatically after the retention memory is loaded following the SLEEP=1 command. In the sleep state, the VDD18 supply must be continuously maintained to ensure that previous configurations and settings are preserved. It is recommended that all other supplies be maintained during sleep mode to simplify the sleep and 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 in-rush 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 I2C/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 MAX96717/F 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 for the user to reprogram configuration registers as is required after a full power cycle.

In Run mode (normal operation), writing SLEEP=1 starts the process of saving device configuration and register settings. The power manager shuts down all internal power supplies, the clocks are disabled, and the chip enters the very low power consumption sleep state. VDD18 must remain stable to provide continuous power to the data retention memory, and it is recommended that all supplies be maintained in their nominal operating range.

Sleep and Wake up Sequences

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

Enable Sleep Mode

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

Wake up (exit Sleep mode):

- **Remote device**
  - Write RESET_LINK = 0 to local device (or power-up/wake-up the local device)
  - Wait for LOCK = 1
  - Write SLEEP = 0 to remote device (Note: Perform within 8ms after LOCK = 1)
- **Local device**
  - Perform a dummy I2C/UART transaction (Note: This wakes up the device)
  - Wait 5ms
  - Write SLEEP = 0 to local device
  - Write RESET_LINK = 0 to local device to enable the link

If devices at both ends of a GMSL link are sleeping, the host processor must initiate the 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 that is intended to reset all sub-systems to their default configurations. However, if a device has previously gone through a sleep/wake cycle, issuing a RESET_ALL resets the device and erroneously loads the contents of the retention memory that had been stored when the most recent SLEEP command was executed. As a result, the device will reset to the state that had been configured prior to entering Sleep mode previously, rather than recovering in a clean power-up default state. The most severe implication of this is that the SLEEP=1 state is saved in the retention memory, so the device will recover from reset and immediately enter 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 that are not saved in retention after resume from sleep. In these cases, full recovery from Sleep mode to the pre-sleep device state requires some repeated register configuration following resume from sleep. Registers that are stored in retention memory are marked with "*" in the register map in the data sheet.

Reference over Reverse (RoR) and RCLKOUT

Crystal or RoR for Basic Function

The MAX96717 must receive an external clock for full function. This can come either from the traditional choice of a 25MHz crystal connected to the X1/OSC and X2 pins or via reference clock over reverse channel (RoR). Note that RoR is not available with the MAX96717R. In RoR, there is no need to connect a crystal next to the serializer, because the serializer's timing reference is extracted from the signal sent on the reverse channel of the GMSL link.

Using RoR instead of the crystal oscillator provides several advantages:

- reduced system cost
- increased reliability
- reduced board area
- simplified board layout

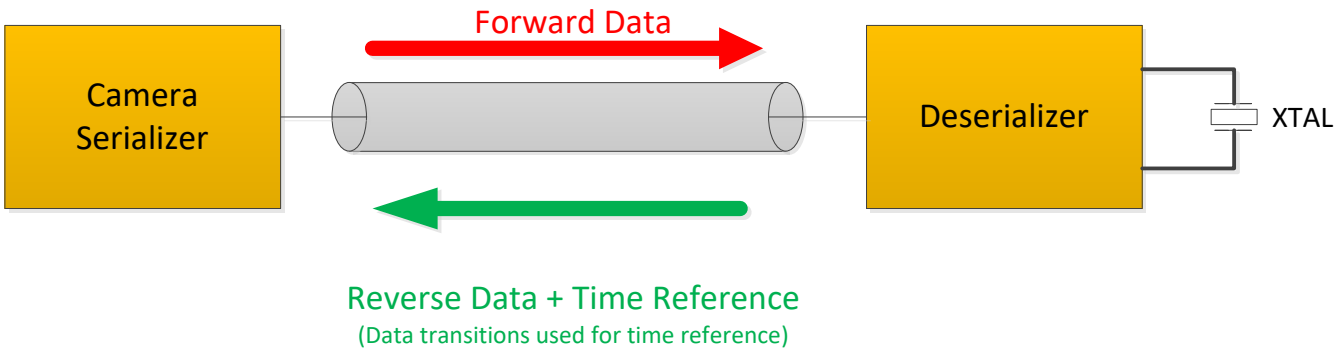


Figure 20: RoR Operation

RCLKOUT/DPLL_OUT

Instead of having separate crystal oscillators for the serializer and image sensor, it is recommended to just use one crystal or RoR and feed that clock signal from the serializer to the image sensor. RCLKOUT or DPLL_OUT is the name of a clock signal that gets sent from the serializer to the image sensor. RCLKOUT is 25 MHz and can be divided by 2 or 4. DPLL_OUT is programmable from 1 to 75 MHz. Note that DPLL_OUT is not available on the MAX96717R. RCLKOUT/DPLL_OUT is available from MFP4, or MFP2 as an alternate.

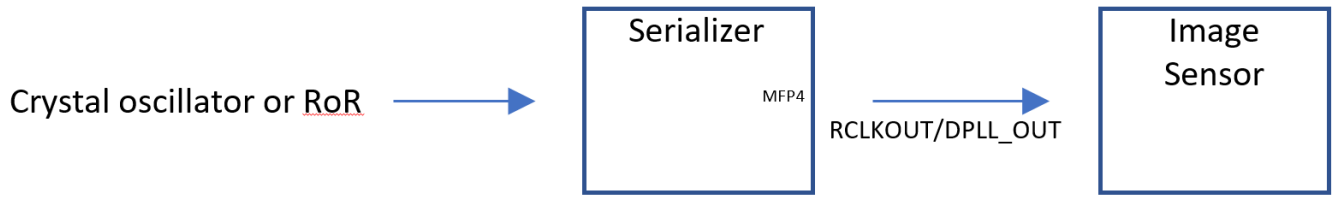


Figure 21. Simplified Clock Diagram

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

- **Xtal/OSC**: In Crystal mode, this block receives the 25MHz clock from a crystal via X1/OSC and X2 pins. The clock is then forwarded to the PLL/CMU block.
- **RoR CLOCK RX**: In RoR-mode, this block extracts the clock from the GMSL reverse channel. The clock is then forwarded to the PLL/CMU block.
- **DIVIDER/DPLL**: This block has two methods of generating an output clock signal: Divider mode and DPLL mode. In Divider mode, RCLKOUT/DPLL_OUT pin can be programmed to /1, /2 or /4 of the 25MHz input reference. In DPLL mode, RCLKOUT/DPLL_OUT pin can be programmed to any frequency from 1MHz to 75MHz. Serializer RCLKOUT/DPLL_OUT is disabled by default and can be enabled with a register write.

RCLKOUT/DPLL_OUT frequency is always relative to the input reference frequency. As long as the clock source's frequency stays within the limits, the RCLKOUT/DPLL_OUT frequency will stay within the limits. Refer to the specific device's data sheet for the limits.

The Path from Crystal/RoR to RCLKOUT/DPLL_OUT

In the Figure 22 a partial block diagram of the MAX96717 is shown. There are connections for the crystal oscillator, the clock signal output for the image sensor, and the GMSL link (SIOP & SION).

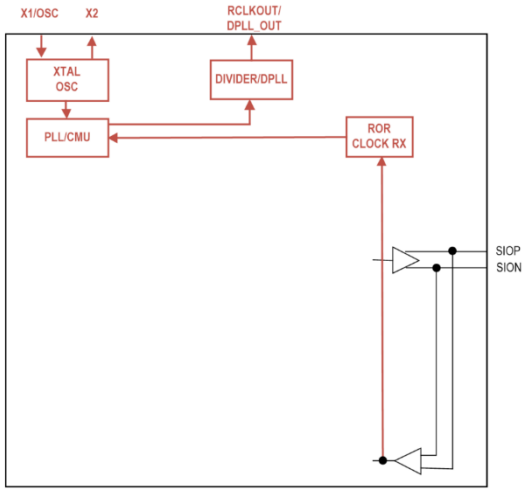


Figure 22. Functional Block Diagram of XTAL and RoR

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

1. The figure below shows the path of using a crystal as the clock source and divider to generate the reference clock.

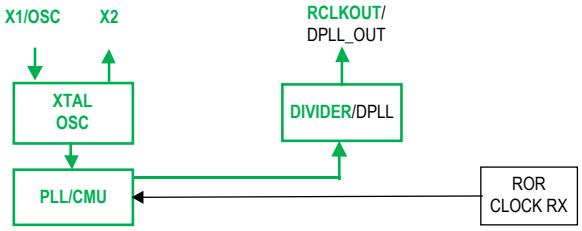


Figure 23. XTAL/OSC to RCLKOUT

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

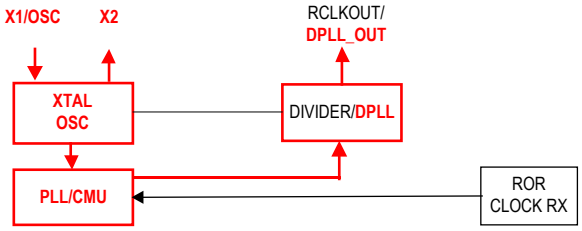


Figure 24. XTAL/OSC to DPLL_OUT

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

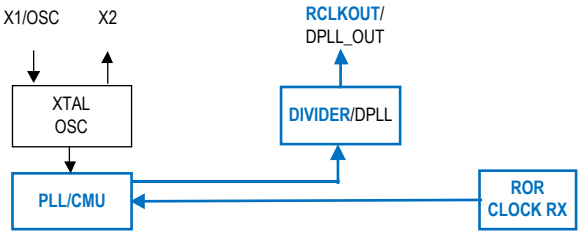


Figure 25. ROR to RCLKOUT

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

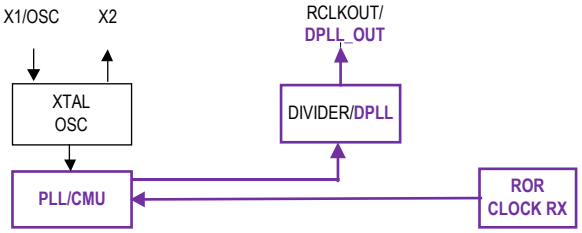


Figure 26. ROR to DPLL_OUT

Setting Serializer to Crystal or RoR

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

CFG0 INPUT VOLTAGE SPECIFICATION (% OF V_{DDIO}) (NOTES a, b)			SUGGESTED RESISTOR VALUES (1% TOLERANCE) (NOTE c)		MAPPED CONFIGURATION (NOTE c)		
MIN	TYP	MAX	R1 (Ω)	R2 (Ω)	I2CSEL	RoR/ Xtal	DEVICE ADDRESS
0.0%	0.0%	11.7%	OPEN	10k	I²C	RoR	0x80
16.9%	20.2%	23.6%	80.6k	20.5k			0x84
28.8%	32.1%	35.5%	68.1k	32.4k		Xtal	0x80
40.7%	44.0%	47.4%	56.2k	44.2k			0x84
52.6%	56.0%	59.3%	44.2k	56.2k	UART	RoR	0x84
64.5%	67.9%	71.2%	32.4k	68.1k			0x80
76.4%	79.8%	83.1%	20.5k	80.6k		Xtal	0x84
88.3%	100%	100%	10k	OPEN			0x80

Figure 27. CFG Settings Table Screenshot from the MAX96717 Datasheet

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

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

It is also possible to enable RoR mode using register writes.

Alternatively Enabling RoR with Register Writes

Alternatively, RoR mode can be enabled via register writes. This is useful for evaluating performance in RoR mode if the board is already setup with a crystal. The procedure is as follows and assumes I2C access on the deserializer side only.

To switch from Crystal mode to RoR mode:

1. Bring up the GMSL link in Crystal mode per normal use case.
2. Write serializer bit XTAL_PU=0 (bit 0 in register 0x4) to disable the serializer crystal driver. This will result in the GMSL link losing lock and automatically re-locking in RoR mode. It is not necessary to issue a reset link command.
3. Reading of serializer bit ROR_CLK_DET (bit 5 in register0x14AA) will now show a logic 1, indicating RoR mode is active.

To switch from RoR mode back to Crystal mode:

1. Write SER bit XTAL_PU=1 to enable the serializer crystal driver. This will result in the GMSL link losing lock and automatically re-locking in Crystal mode. It is not necessary to issue a reset link command.
2. Reading of serializer bit ROR_CLK_DET (bit 5 in register0x14AA) will now show a logic 0, indicating RoR mode is no longer active.

Hardware Considerations

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

For the MAX96717, there is no need to connect a capacitor to the Vref pin.

GMSL Link Lock Time

GMSL link lock time in RoR mode is similar to what it is in Crystal mode. Refer to the device datasheet for lock time specifications.

Jitter Considerations

Serializer RCLKOUT output jitter in RoR mode is increased over Crystal mode, but well within most system requirements. Most of the additional jitter is data dependent, resulting from inter-symbol interference (ISI) on the reverse channel. Refer to the serializer datasheet for output jitter specifications.

While the RoR jitter does affect the serializer transmit signal, the jitter bandwidth is limited and can be tracked by the clock recovery circuit in the deserializer.

Spread Spectrum Clocking (SSC)

Spread spectrum clocking (SSC) is an alternate configuration for the GMSL link that helps with EMI. SSC is supported in RoR mode. If it is used, it is enabled only in the deserializer. The serializer clock will follow the deserializer clock modulation and consequently the serializer will operate with a spread spectrum clock as well. The serializer RCLKOUT output reference will also be frequency modulated in the same manner.

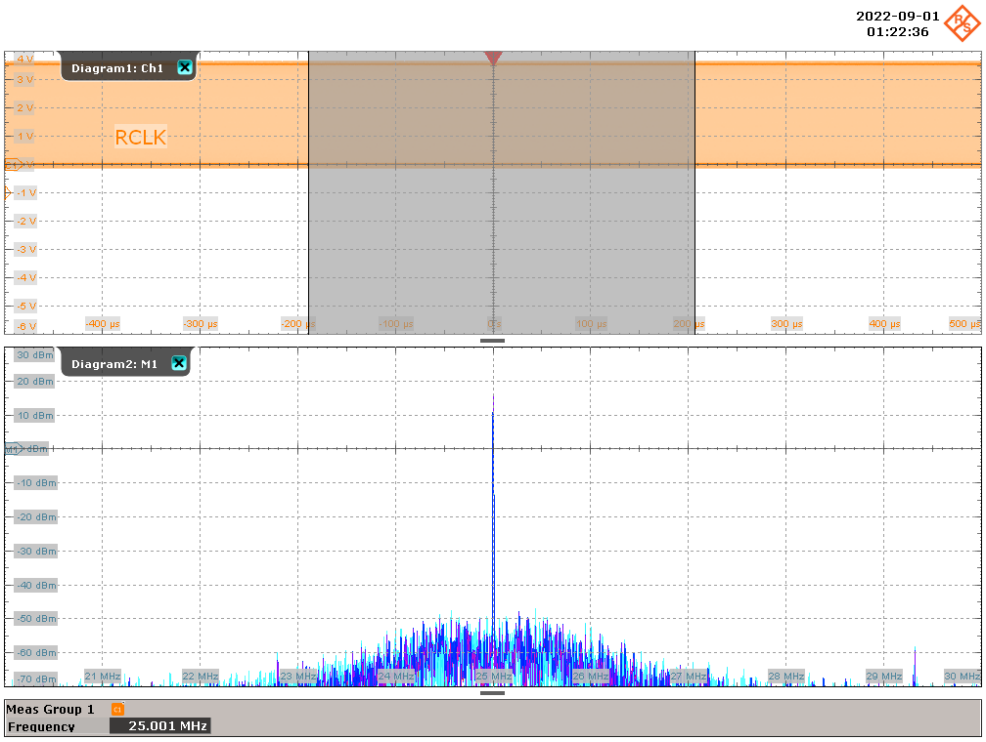


Figure 28. 25MHz RCLKOUT Signal using RoR Without SSC Feature Enabled

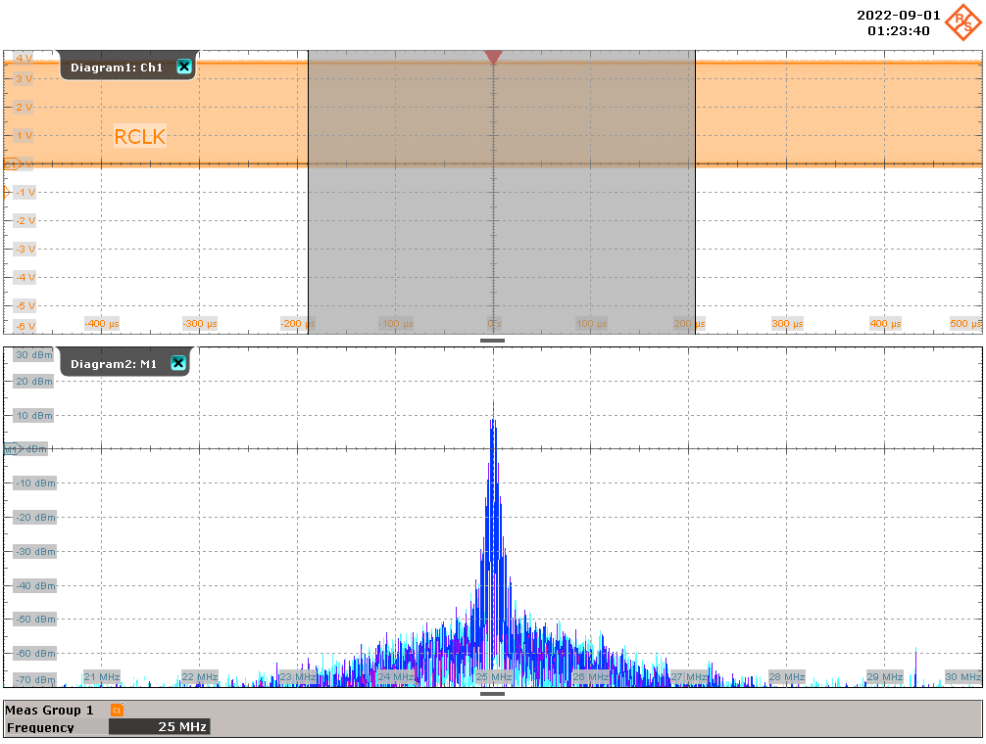


Figure 29. 25MHz RCLKOUT Signal using RoR with SSC Feature Enabled

Recovery After Loss of GMSL Link Lock

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

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

Upon loss of GMSL link lock, the RCLKOUT/DPLL_OUT frequency will initially increase approximately 16% for roughly 150µs, and then settle to the unlocked 10% reduced frequency of 22.5MHz. This behavior is shown in the figure below.

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

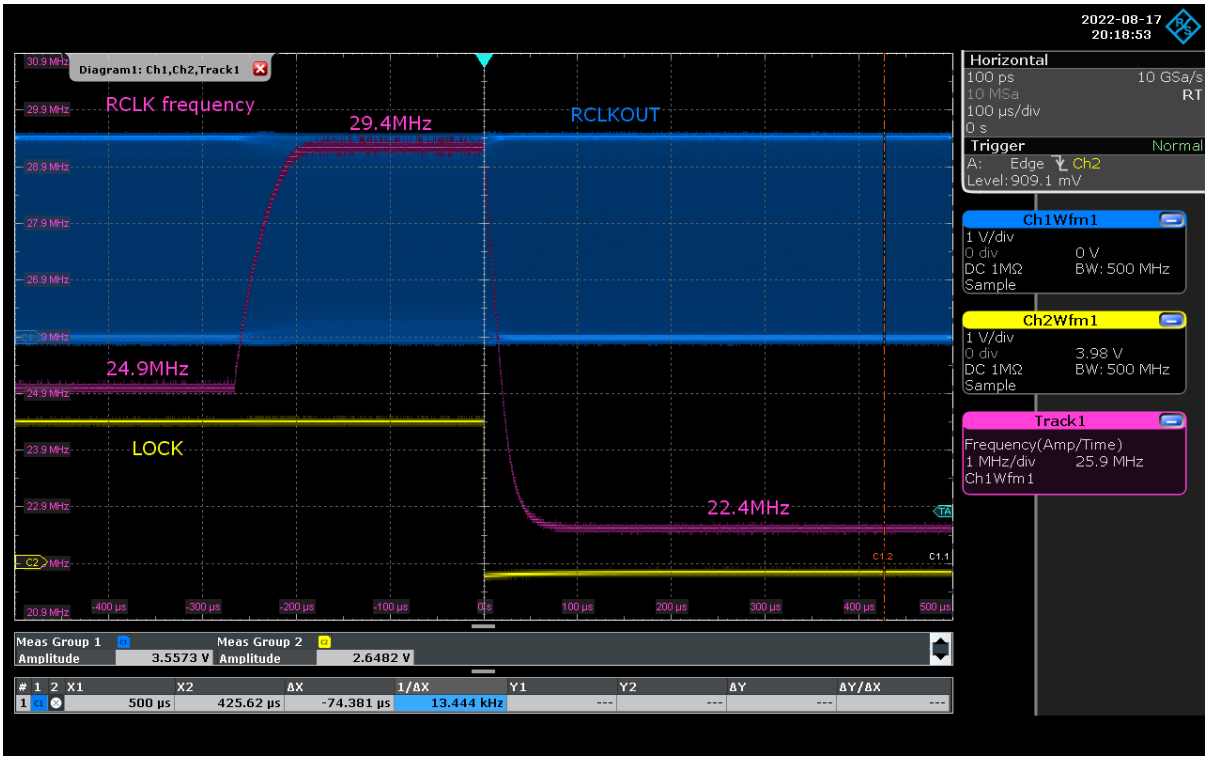


Figure 30. 25MHz RCLK to 16% Frequency Overshoot (29MHz) and then to 10% Reduced Frequency (22.5MHz)

Turning on RCLKOUT

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

Voltage Monitoring/ADC

High Level Description

The MAX96717 and MAX96717F feature a 10-bit analog-to-digital converter (ADC) with an analog input multiplexer (MAX96717R does not have these features). The multiplexer selects a single-ended input channel from external input lines (MFP3/5/6), internal power supply monitors, and a temperature monitor.

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

Introduction to Typical ADC Flow

The general steps of setting up the ADC for operation are listed below. Each portion will be expanded on in the "Details of Operation" section.

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

Details of Operation

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

ADC Setup

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

ADC Shutdown

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

Table 23. ADC Shutdown Flow

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

ADC Power Up

The following table shows the relevant registers necessary to apply power to the ADC and enable operation. Before the ADC can be utilized, the power manager for the MAX96717 must have VDD, VDD18 and VDDIO supplies available and the link between the serializer and de-serializer must be established.

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

Table 24. ADC Power Up Flow

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

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

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

On-Demand Conversions

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

Temperature Readings

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

Table 25. On-Demand Temperature Reading

Step	Action	Register Address	Bitfield Name	Bits	POR	Decode
1	Initiate temperature conversion	0x1D28	RUN_TMON_CAL	0	0b0	0b0: Do not run 0b1: Run, cleared on write
2	Wait for ADC done interrupt to assert	0x510	adc_done_if	0	0b0	0b0: Flag cleared 0b1: ADC conversion done
3	Wait for calibration done interrupt to assert	0x510	adc_calDone_if	7	0b0	0b0: Flag cleared 0b1: ADC accuracy calibration done
4	Read internal die temperature registers	0x1D3B 0x1D3C	T_EST_OUT_B0 T_EST_OUT_B1	7:0 7:6	0xFF 0b11	Bits 7:0 of TMON temp Bits 9:8 of TMON temp
5	Read alternate internal die temperature registers (optional)	0x1D3C 0x1D3D	ALT_T_EST_OUT_B1 ALT_T_EST_OUT_B0	1:0 7:0	0b11 0xFF	Bits 9:8 of alt TMON temp Bits 7:0 of alt TMON temp

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

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

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

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

Internal Voltage Reading

The MAX96717 provides the ability to measure the three internal voltage rails: VDDIO, VDD18 and CAP_VDD with the ADC. To take these measurements, the MUX must be set up to read the desired internal voltage prior to initiating the ADC conversion. After taking the ADC conversion, it is recommended to disable the input MUX to create a "break before make" functionality that avoids shorting two inputs momentarily.

Table 26. On-Demand Internal Voltage Reading

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

The following equation is used to convert the 10-bit value in the ADC.DATA0 and ADC.DATA1 registers to the corresponding voltage. It is important to use the correct voltage reference and divider value to get an accurate voltage value. Divider value should be 2 or 4 depending on which internal voltage is being monitored. If using the VDD18 voltage rail as the ADC reference use 900mV as the value for $V_{ref.}$ Use the following equation:

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

External Voltage Reading

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

Table 27. On-Demand External Voltage Reading

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

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

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

Round Robin Conversion

To use the round robin state machine, first shut down the ADC, then configure it to use the desired voltage reference. Next, HI/LO channel thresholds should be configured for the channels that are going to be monitored. These thresholds can be set for the 3 internal voltages, 3 external voltages, and the die temperature. After

configuring the HI/LO thresholds for the desired channels, follow the steps below to enable the round robin state machine.

Table 28. Round Robin Setup Flow

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

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

The following steps are the order in which the round robin state machine executes its conversions:

Table 29. Round Robin Timing

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

Channel HI/LO Limits

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

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

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

Internal/External Voltage Threshold Formula:

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

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

Internal Die Temperature Threshold Formula:

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

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

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

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

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

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

Next values of ADC_HOT and TRIM_TEMP will be calculated:

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

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

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

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

Writing 10-bit HI/LO Thresholds

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

Calculated LO threshold = 0x260Calculated HI threshold = 0x2E6

ADC_LIMIT_0 register = 0xXX

ADC_LIMIT_1 register = 0bXXXXXXXX

ADC_LIMIT_2 register = 0bXXXXXXX

Table 30. Channel HI/LO Limit Register Setup

Step	Action	Register Address	Bitfield Name	Bits	POR	Decode
1	Program the 10b LO threshold	ADC_LIMIT<***CH***>_0 ADC_LIMIT<***CH***>_1	chLoLimit_l<***CH***> chLoLimit_h<***CH***>	7:0 1:0	0x00 0b00	Bits 7:0 of 10b LO limit Bits 9:8 of 10b LO limit
2	Program the 10b HI threshold	ADC_LIMIT<***CH***>_1 ADC_LIMIT<***CH***>_2	chHiLimit_l<***CH***> chHiLimit_h<***CH***>	7:4 5:0	0x0 0x00	Bits 3:0 of 10b HI limit Bits 9:4 of 10b HI limit
3	Program the multiplexer input channel	ADC_LIMIT<***CH***>_3	ch_sel<***CH***>	3:0	0x00	0x0 – ADC0 (MFP3) 0x1 – ADC1 (MFP5) 0x2 – ADC2 (MFP6) 0x8 – VDDIO/4 0x9 – VDD18/2 0xA – CAP_VDD/2 0xB – Die Temperature
3a	Enable monitoring of ADC0, ADC1 or ADC2 (if monitoring MFP3/5/6)	0x53E	adc_pin_en	2:0	0x00	0bXX1 – Enable ADC0 0bX1X – Enable ADC1 0b1XX – Enable ADC2
4	Program the GPIO divider setting if needed	ADC_LIMIT<***CH***>_3	div_sel<***CH***>	4:5	0b00	0b00 – Divide by 1 0b01 – Divide by 2 0b10 – Divide by 3 0b11 – Divide by 4
5	Enable the channel HI interrupt	0x50D	ch<***CH***>_hi_limit_ie	7:0	0x00	0b1 – Enable Channel X interrupt in the [X] bit position
6	Enable the channel LO interrupt	0x50E	ch<***CH***>_lo_limit_ie	7:0	0x00	0b1 – Enable Channel X interrupt in the [X] bit position
7	Enable the global HI interrupt	0x50C	adc_hi_limit_ie	2	0b0	0b0 – ADC interrupt source disabled 0b1 – HI limit monitor asserts ADC interrupt
8	Enable the global LO interrupt	0x50C	adc_lo_limit_ie	3	0b0	0b0 – ADC interrupt source disabled 0b1 – LO limit monitor asserts ADC interrupt

Internal Testing

The MAX96717 has the capability to internally test various portions of the ADC. The checks include an ADC accuracy BIST and GPIO input MUX verification. The ADC accuracy is a set of internal tests which consist of manipulating the input voltage and divider settings to test the accuracy of the ADC. Each of these functions can be run as an on-demand (upon start-up or as required during operation), or the ADC accuracy BIST can be run as part of the round robin functionality.

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

ADC Accuracy BIST

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

Table 31. ADC Accuracy Tests

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

- REFLIM is a 5b register that controls the limits for accuracy of the reference comparisons (default: 17).
- REFLIMISCL1 is a 4b register that controls the limits for the accuracy of the second test (default: 5).
- REFLIMISCL2 is a 4b register that controls the limits for the accuracy of the third test (default: 5).
- REFLIMISCL3 is a 4b register that controls the limits for the accuracy of the fourth test (default: 5).

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

Table 32. ADC Accuracy BIST Setup

Step	Action	Register Address	Bitfield Name	Bits	POR	Decode
1	Set the desired REFLIM accuracy (limit between the two BG references)	0x1D31	REFLIM	7:0	0x0F	0xXX: LSBs of accuracy (Default: 17)
2	Set the reference scale /2 accuracy limit	0x1D32	REFLIMSCL1	7:0	0x0F	0xXX: LSBs of accuracy (Default: 5)
3	Set the ADC scale /2 accuracy limit	0x1D33	REFLIMSCL2	7:0	0x07	0xXX: LSBs of accuracy (Default: 5)
4	Set the ADC input /8 accuracy limit	0x1D34	REFLIMSCL3	7:0	0x07	0xXX: LSBs of accuracy (Default: 5)
5	Enable the ADC calibration done interrupt	0x50C	adc_calDone	7	0b0	0b0: Reporting disabled 0b1: Reporting enabled
6	Enable the interrupts for each of the REF limits	0x50F	reflim_ie reflimscl1_ie	6 5	0b0 0b0	0b0: Reporting disabled

			reflimscl2_ie reflimscl3_ie	4 3	0b0 0b0	0b1: Reporting enabled
7	Clear the interrupt status flags by reading registers	0x510 0x513	ADC_INTR0 ADC_INTR3	7:0 7:0	0x00 0x00	ADC Interrupt reporting, clear on read
8	Execute the ADC accuracy BIST	0x1D28	RUN_ACCURACY	2	0b0	0b0: Do not run 0b1: Run
9	Wait for calibration done and ADC done	0x510	adc_done_if adc_calDone_if	0 7	0b0 0b0	0b0: Flag cleared 0b1: Flag set
10	Read status interrupt registers	0x513	reflim_if reflimscl1_if reflimscl2_if reflimscl3_if	6 5 4 3	0b0 0b0 0b0 0b0	0b0: Flag cleared 0b1: Flag set

ADC GPIO Input Verification Test (Errata)

GPIO input MUX functionality is not tested using the built-in self-test of the ADC. This functionality can be manually verified via software. Verification is only required for the MFP channels that will be monitored using the ADC. If no MFP3/5/6 voltages are being monitored using the ADC, then this test is not necessary. Use the steps in Table 33 to complete this test. They are also described in the MAX96717 errata sheet.

ADC Power Up

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

Table 33. ADC GPIO Input Verification Power Up Flow

Step	Action	Register Address	Bitfield Name	Bits	POR	Decode
1	Reset the device	0x10	RESET_ALL	7	0b0	0b0: No action 0b1: Activate chip reset
2	Enable ADC clock	0x501	adc_clock_en	3	0b0	0b0: ADC clock disable 0b1: ADC clock enable
3	Enable ADC ready interrupt	0x50C	adc_ref_ready_ie	1	0b0	0b0: Reporting disabled 0b1: Reporting enabled
4	Enable ADC done interrupt	0x50C	adc_done_ie	0	0b0	0b0: Reporting disabled 0b1: Reporting enabled
5	Power up the ADC charge pump	0x500	adc_chgpump_pu	4	0b0	0b0: ADC charge pump off 0b1: ADC charge pump on
6	Power up the ADC	0x500	adc_pu	1	0b0	0b0: ADC powered off 0b1: ADC powered on
7	Power up the ADC reference buffer	0x500	adc_refbuf_pu	3	0b0	0b0: ADC reference buffer off 0b1: ADC reference buffer on
8	Power up the ADC internal buffer	0x500	buf_pu	2	0b0	0b0: ADC internal buffer off 0b1: ADC internal buffer on
9	Turn on multiplexer input enable	0x502	Inmux_en[0]	0	0b0	0b0: Input MUX is open

						0b1: MUX selected by adc_chsel field
10	Select ADC0 for input MUX	0x501	adc_chsel	7:4	0x0	0x0 – ADC0 (MFP3)
11	Initiate temperature conversion	0x1D28	RUN_TMON_CAL	0	0b0	0b0: Do not run 0b1: Run
12	Enable the MUX verification bit	0x1D28	MUXVER_EN	4	0b0	0b0: Disable 0b1: Enable

Input Verification Test

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

Table 34. ADC GPIO Input Verification Test Flow

Step	Action	Register Address	Bitfield Name	Bits	POR	Decode
1	Enable ADC0/1/2 Input	0x53E	adc_pin_en	2:0	0b000	0bXX1: Enable ADC0 0bX1X: Enable ADC1 0b1XX: Enable ADC2
2	Set channel for GPIO being tested to low and all other channels to high CH0 – ADC0 CH1 – ADC1 CH2 – ADC2	0x1D37	MUXV_CTRL	7:0	0x00	0bXXXXXXX0: CH0 driven low 0bXXXXXXX1: CH0 driven high …….. 0b0XXXXXXX: CH7 driven low 0b1XXXXXXX: CH7 driven high
3	Start an ADC conversion	0x500	cpu_adc_start	0	0b0	0b0: Conversion complete 0b1: Start ADC conversion
4	Read the ADC data registers and check that result is within 15LSB of 0x000	0x508 0x509	adc_data_l adc_data_h	7:0 1:0	0x00 0b00	Bits 7:0 of 10b ADC data Bits 9:8 of 10b ADC data
5	Set channel for GPIO being tested to high and all other channels to low CH0 – ADC0 CH1 – ADC1 CH2 – ADC2	0x1D37	MUXV_CTRL	7:0	0x00	0bXXXXXXX0: CH0 driven low 0bXXXXXXX1: CH0 driven high …….. 0b0XXXXXXX: CH7 driven low 0b1XXXXXXX: CH7 driven high
6	Start an ADC conversion	0x500	cpu_adc_start	0	0b0	0b0: Conversion complete 0b1: Start ADC conversion

7	Read the ADC data registers and check that result is within 35LSB of 0x3E6 (0x3C3 – 0x409)	0x508 0x509	adc_data_l adc_data_h	7:0 1:0	0x00 0b00	Bits 7:0 of 10b ADC data Bits 9:8 of 10b ADC data

Reset for Normal Operation

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

Table 35. Returning to Normal Operation

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

Examples of ADC Operation

ADC Shutdown Example

Table 36. ADC Shutdown

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

ADC Setup Example

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

Table 37. ADC Setup

Step	Action	Read/Write	Register Address	Value	Comments
1	Enable global ADC interrupt	W	0x1E	0xFF	
2	Enable ADC clock and select 1.25V internal reference	W	0x501	0x08	
3	Enable ADC ready interrupt, done interrupt and calibration done interrupt	W	0x50C	0x83	
4	Clear ADC interrupts	R	0x510 0x511 0x512 0x513	-	Read to clear

5	Select internal or external reference	W	0x502	0x00	
6	Power up the ADC, charge pump, internal buffer, and reference buffer	W	0x500	0x1E	
7	Wait for ready interrupt to be asserted	R	0x510	-	Waiting to read 0x02
8	Initialize a temperature conversion	W	0x1D28	0x01	
9	Wait for ADC done, and calibration done interrupt to assert	R	0x510	-	Waiting to read 0x81

On-Demand Read Examples:

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

Die Temperature

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

Table 38. On-Demand Die Temperature Reading

Step	Action	Read/Write	Register Address	Value	Comments
1	Initiate temperature conversion	W	0x1D28	0x01	
2	Wait for ADC done, and calibration done interrupt to assert	R	0x510	-	Waiting for 0x81
3	Read internal die temperature registers	R	0x1D3B 0x1D3C	-	
4	Read alternate internal die temperature registers (optional)	R	0x1D3C 0x1D3D	-	

Internal Voltage

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

Table 39. On-Demand Internal Voltage Reading

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

External Voltage (MFP3)

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

Table 40. On-Demand External Voltage Reading

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

HI/LO Channel Limit Example:

Voltage Threshold Calculations:

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

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

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

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

Temperature Threshold Calculations:

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

Read 0x1C2D = 0x67	0x1C2D [7:0] = 0x67 -> 103
Read 0x1C2E = 0x74	0x1C2E [6:0] = 0x74 -> 116
Read 0x1C26 = 0xEC	0x1C26 [7:0] = 0xEC -> 236
Read 0x1C27 = 0x4D	0x1C27 [6:0] = 0x4D -> 77
Read 0x1C2A = 0x06	0x1C2A [7:0] = 0x06 -> 6
Read 0x1C2E = 0x86	0x1C2E [3:0] = 0x06 -> 6

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

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

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

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

Next, ADC_HOT and TRIM_TEMP are calculated:

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

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

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

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

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

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

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

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

Round Robin Example:

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

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

CH1: +/-5% of VDD18

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

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

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

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

Table 41. ADC Round Robin Example

Step	Action	Read/Write	Register Address	Value	Comments
1	Program the 10b HI & LO thresholds for CH0 (VDDIO)	W	0x514 0x515 0x516	0x60 0x62 0x2E	LO threshold – 0x260 HI threshold – 0x2E6
2	Program the multiplexer input channel to VDDIO	W	0x517	0x08	
3	Program the 10b HI & LO thresholds for CH1 (VDD18)	W	0x518 0x519 0x51A	0xBC 0x52 0x30	LO threshold – 0x2BC HI threshold – 0x305
4	Program the multiplexer input channel to VDD18	W	0x51B	0x09	
5	Program the 10b HI & LO thresholds for CH2 (CAP_VDD)	W	0x51C 0x51D 0x51E	0x71 0x21 0x1C	LO threshold – 0x171 HI threshold – 0x1C2
6	Program the multiplexer channel 2 to CAP_VDD	W	0x51F	0x0A	
7	Read EFUSE registers used to calculate temperature thresholds	R	0x1C2D 0x1C2E 0x1C26 0x1C27 0x1C2A 0x1C2B	0x67 0x74 0xEC 0x4D 0x06 0x86	
8	Program the 10b HI & LO thresholds for CH3 (TMON)	W	0x520 0x521 0x522	0x50 0x51 0x21	LO threshold – 0x150 HI threshold – 0x215
9	Program the multiplexer input channel to TMON	W	0x523	0x0B	
10	Program the 10b HI & LO thresholds for CH4 (ADC0/MFP3)	W	0x524 0x525 0x526	0xE1 0x42 0x38	LO threshold – 0x2E1 HI threshold – 0x384
11	Enable ADC0 (MFP3)	W	0x53E	0x01	
12	Program the multiplexer input channel to ADC0 (MFP3) and set divider to Divide by 2	W	0x527	0x10	
13	Enable the channel HI interrupt	W	0x50D	0x1F	CH0-CH4 enabled
14	Enable the channel LO interrupt	W	0x50E	0x1F	CH0-CH4 enabled
15	Enable the global HI and LO interrupts	W	0x50C	0x8F	Keeps adc_done_ie, adc_ref_ready_ie and adc_calDone enabled
16	(Optional) Run ADC Accuracy test	W	0x1D28	0x80	0x80 will run ADC accuracy test in round robin state machine
17	Set number of conversion cycles between ADC conversion	W	0x536 0x537	0x6B 0x2D	5 second sleep time = 0x2D6B
18	Enable round robin state machine	W	0x534	0x01	

Internal Tests

ADC Accuracy BIST

Table 42. ADC Accuracy BIST Example

Step	Action	Read/Write	Register Address	Value	Comments
1	Set the desired REFLIM accuracy (limit between the two BG references)	W	0x1D31	0x11	Default 17 LSBs
2	Set the reference scale /2 accuracy limit	W	0x1D32	0x05	Default 5 LSBs
3	Set the ADC scale /2 accuracy limit	W	0x1D33	0x05	Default 5 LSBs
4	Set the ADC input /8 accuracy limit	W	0x1D34	0x05	Default 5 LSBs
5	Enable the ADC calibration done interrupt	W	0x50C	0x83	Preserves ADC done and ADC ref ready interrupt
6	Enable the interrupts for each of the REF limits	W	0x50F	0x78	Enables all interrupts for ADC accuracy tests
7	Clear the interrupt status flags by reading registers	R	0x510 0x513	-	Clear on read
8	Execute the ADC accuracy tests	W	0x1D28	0x04	
9	Wait for Calibration done and ADC done	R	0x510	-	Waiting for 0x81
10	Read status interrupt registers	R	0x513	-	0x00 for no failures

ADC GPIO Input Verification Test (Errata)

In this example, the GPIO input verification test is run for MFP3. Full scripts can also be found in the MAX96717 Errata Sheet.

Table 43. ADC GPIO Input Verification Test Example

Step	Action	Read/Write	Register Address	Value	Comments
1	Reset the device	W	0x10	0x80	Cleared on write
2	Enable ADC clock	W	0x501	0x08	
3	Enable ADC ready, ADC done interrupts	W	0x50C	0x03	
4	Power up the ADC, ADC charge pump, input buffer and internal reference	W	0x500	0x1E	
5	Turn on multiplexer input enable	W	0x502	0x01	
6	Select ADC0 for input MUX	W	0x501	0x08	Keep ADC clock enable on
7	Initiate temperature conversion	W	0x1D28	0x01	Cleared on write
8	Enable the MUX verification bit	W	0x1D28	0x10	
9	Enable ADC0/1/2 Input	W	0x53E	0x01	
10	Set channel for GPIO being tested to low and all other channels to high CH0 – ADC0 CH1 – ADC1	W	0x1D37	0x0FE	

	CH2 – ADC2				
11	Start an ADC conversion	W	0x500	0x1F	
12	Read the ADC data registers and check that result is within 15LSB of 0x000	R	0x508 0x509	-	
13	Set channel for GPIO being tested to high and all other channels to low CH0 – ADC0 CH1 – ADC1 CH2 – ADC2	W	0x1D37	0x01	
14	Start an ADC conversion	W	0x500	0x1F	
15	Read the ADC data registers and check that result is within 35LSB of 0x3E6	R	0x508 0x509	-	
16	Disable input MUX test bit for normal operation	W	0x1D28	0x00	

Bandwidth Efficiency Optimization

Overview

Before implementing a new design, it is critical to do bandwidth (BW) calculations to verify that the right devices and settings are used. If this is not done, it is possible that data will be lost or corrupted. Although the MAX96717 family can transmit at 3 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 this table.

Table 44. 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 equations below.

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

$$Video\ Payload = PCLK * bpp$$

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.

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

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

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 (e.g., GPIO, SPI) affect link bandwidth consumption and must be considered if enabled.

Optimizing Bandwidth

Data can be manipulated over the GMSL link to utilize the bandwidth more efficiently. The easiest way to optimize the bandwidth consumption is by using Tunneling mode. In Tunneling mode, all BPPs are forced to 24 regardless of data type and this allows for the PCLK and overall bandwidth to be reduced. Another common way of optimizing the bandwidth is by doubling or tripling the BPP. When transmitting an 8 BPP data type at 16 BPP or 24 BPP, less bandwidth is consumed for the same reason as when using Tunneling mode. With regards to

bandwidth, zero padding has the opposite effect that doubling or tripling has. Zero padding will always consume more bandwidth, so this method of data manipulation should only be used when necessary.

Bandwidth Optimization Example

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

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

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

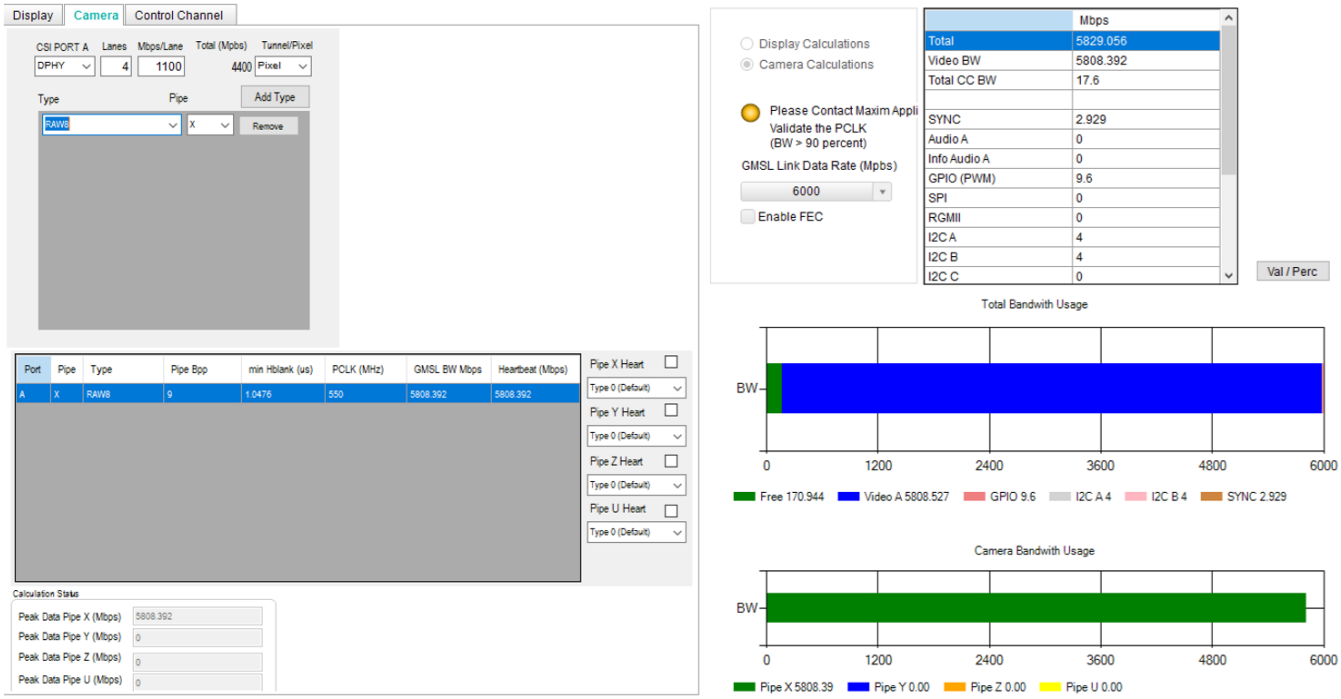


Figure 31. Bandwidth Calculations without Doubling

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

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

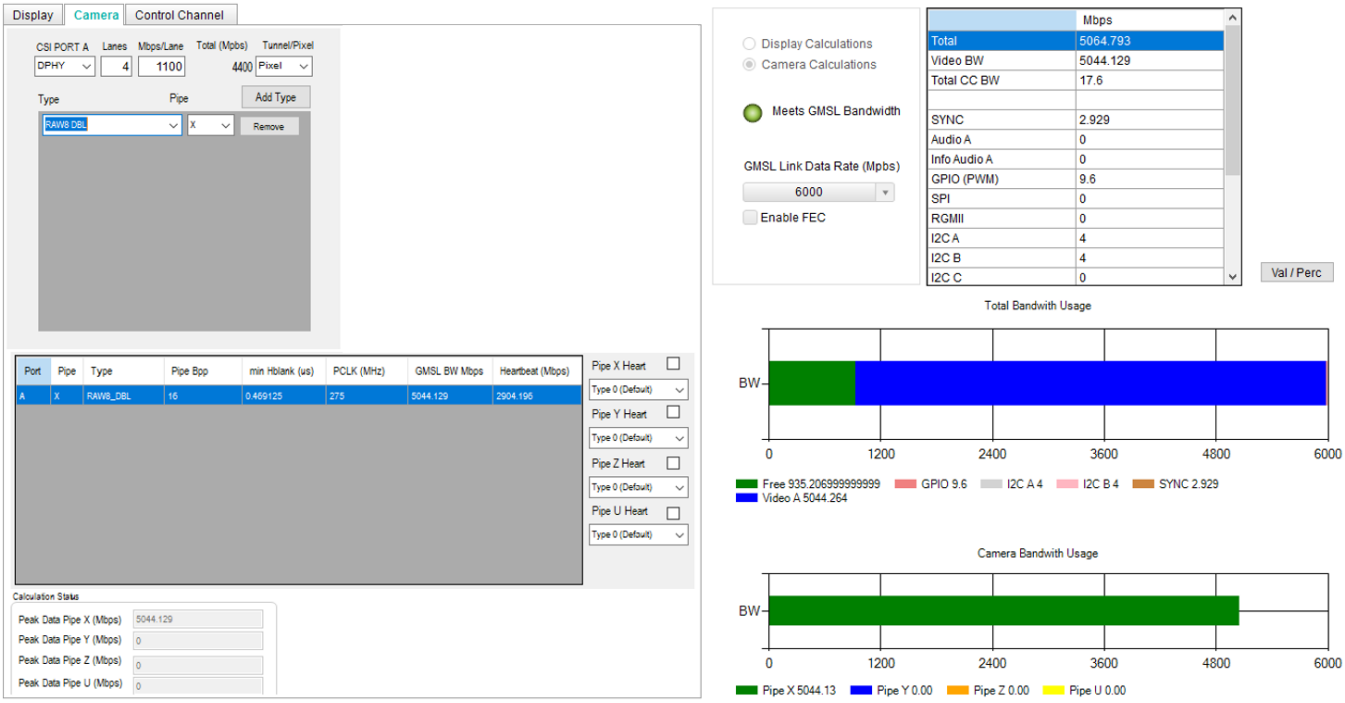


Figure 32. Bandwidth Calculations with Doubling

MIPI Packet Counters

Overview

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

MIPI Packet Counter Registers

Each GMSL link has a built-in, 8-bit received packet counter. Within the MIPI PHY/controller there are packet counters implemented that can be used to verify data is being transferred within the serializer. There are packet counters for MIPI PHY, MIPI controller, Tunneling mode packets and MIPI Phy clock.

The *csi1_pkt_cnt* registers report the packet count of the associated MIPI controller; the *phy1_pkt_cnt* registers report the packet count of the associated MIPI PHY. Sequential reads returning different values indicate that data is being transmitted by the associated MIPI controller or PHY. The register value will not change if data is not flowing.

Table 45. Counter Registers

Register	Bits	Default Value	Description	Decode
0x38D	7:0	0x00	Packet count of MIPI PHY1	**Phy1_pkt_cnt registers:** 0bXXXx: Toggling bits indicate MIPI data is active on the PHY
0x38E	7:0	0x00	Packet count of CSI-2 Controller	**csi1_pkt_cnt registers**: 0bXXXX: Toggling bits indicate MIPI data is active on the controller. (Packets Processed)
0x38F	7:0	0x00	Packet count of Tunnel Packet Processed	**tun_pkt_cnt register:** 0bXXXx: Toggling bits indicate MIPI data is active on the PHY
0x390	7:0	0x00	MIPI PHY Clock Counter	**phy_clk_cnt register:** 0bXXXx: Toggling bits indicate MIPI clock is detected

Programming Example

```
The following example demonstrates the use case of the MIPI Packet counters.
Setup:  MAX96717 and MAX96724 with SV4E Introspect Generator and Analyzer.

   1) Connect the SV4E Introspect generator to MAX96717.
   2) Connect Link B of MAX96724 to MAX96717.
   3) Generate a CSI_RGB888 Pattern on the Introspect Generator.
```

4) Check PCLKDET on the MAX96717 and video lock on the MAX96724. (Make sure to disable the other links on MAX96724.)
5) Now check register 0x38D on MAX96717 to check if the phy1_pkt_cnt bits are toggling.
6) Check reg 0x38E on MAX96717 to check if the csi1_pkt_cnt bits are toggling.
7) Check also reg 0x390 on MAX96717 to check if the phy_clk_cnt bits are toggling as well.
8) Perform similar register reads 0x8d0,0x8d2 for csi2_tx1_pkt_cnt, phy0_pkt_cnt and phy1_pkt_cnt bits toggling on MAX96724.
9) If the bits are not toggling, then check the part configuration.

Error Flags

Overview

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

The error bar (ERRB) pin is an MFP pin that logically NORs many of the errors, so it is a convenient way to check for errors. It is available at MFP3 or at MFP8 as an alternate. Whether an error is included in the ERRB output depends on if its output-enable (OEN) is high. Most OEN's are high by default.

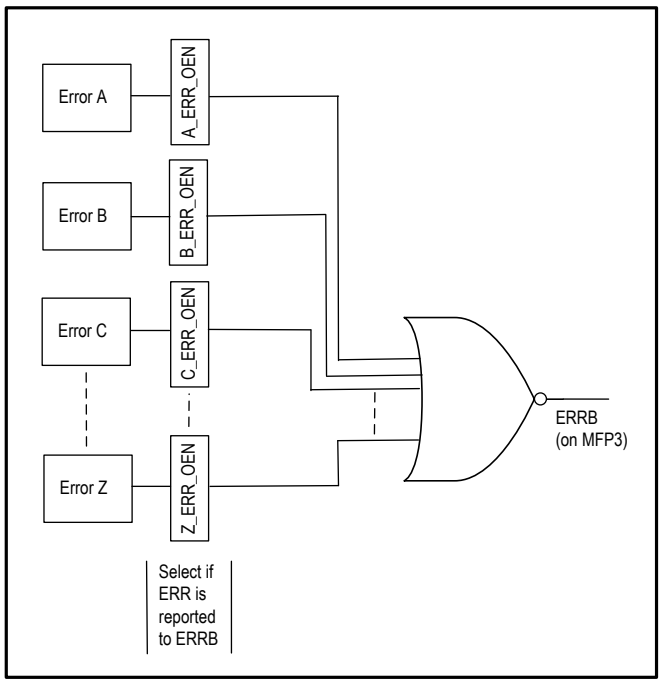


Figure 33. ERRB Reporting Flow

See the error flags table for a description of each error flag in the MAX96717 and MAX96717F.

Please note that some of these are not available in MAX96717R. See the device datasheets for full details.

Table 46. Error Flags Table for MAX96717 and MAX96717F

Error Flag	Error Description
VREG_OV_FLAG	CAP_VDD overvoltage indication
EOM_ERR_FLAG_A	Eye Opening is below the configured threshold
VDD_OV_FLAG	VDD overvoltage indication
VDD18_OV_FLAG	VDD18 overvoltage indication
MAX_RT_FLAG	Combined ARQ maximum retransmission limit error flag
RT_CNT_FL AG	Combined ARQ retransmission event flag
PKT_CNT_F LAG	Packet Count Flag
VDDCMP_INT_FLAG	Combined undervoltage comparator output. Asserted when GMP_SATATUS is asserted.
PORZ_INT_FLAG	PORZ interrupt flag. PORZ is Monitoring of undervoltage levels of VDD18 and VDDIO.

VDDBAD_INT_FLAG	Combined VDD bad indicator. Asserted when either with VDDBAD_STATUS or CAP_VDD <0.82V.
EFUSE_CRC_ERR	EFUSE CRC error indicator. An issue with the device EFUSE has occurred, the device should no longer be used.
RTTN_CRC_INT	Retention memory restore CRC error interrupt
ADC_INT_FLAG	ADC Interrupt
MIPI_ERR_FLAG	MIPI RX Error Flag
REFGEN_UNLOCKED	Reference DPLL generating RCLKOUT is not locked.
REM_ERR_FLAG	Received remote side error status
LFLT_INT	Line-Fault Interrupt Asserted when either one of line-fault monitors indicates a fault status
IDLE_ERR_FLAG	Idle-Word Error Flag
DEC_ERR_FLAG	Errors detected in GMSL packet
I2C_UART_ MSGCNTR_ ERR_INT	I2C/UART message counter error flag. Asserted when two message counter bytes sent do not match expected count value.
I2C_UART_ CRC_ERR_I NT	I2C/UART CRC error flag. Asserted when CRC byte sent does not match calculated value.
MEM_ECC_ ERR2_INT	Error flag for 2-bit uncorrectable memory ECC errors seen in any memories
MEM_ECC_ ERR1_INT	Error flag for 1-bit uncorrectable memory ECC errors seen in any memories
REG_CRC_ ERR_FLAG	Error occurred on the register CRC calculation.
SPI_RX_OVRFLW	SPI Rx Buffer Overflow Flag
SPI_TX_OVRFLW	SPI Tx buffer overflow flag
adc_overRange_ie	ADC Digital Correction Overrange enabled
adc_tmon_cal_ood_ie	Enable temperature sensor out-of-date interrupt
adc_lo_limit_ie	Enable ADC low limit monitor interrupt
adc_hi_limit_ie	Enable ADC high limit monitor interrupt
adc_ref_read y_ie	Enable ADC ready interrupt
adc_done_ie	Enable ADC conversion completed Interrupt
adc_calDone_ie	Signal that ADC accuracy/temperature sensor calibration is completed
DRIFT_ERR	VID_TX_PCLK drift error detected
FIFO_WARN	Transmitted video (VID_TX_FIFO) is more than half full
OVERFLOW	VID_TX FIFO overflow occurred
tun_fifo_overflow	Tunnel FIFO overflow occurred
CMP_STATUS	VDD18, VDDIO, and CAP_VDD supply voltage comparator status bits. Latched when the supply voltages are not in range.
phy*1_lp_err	Unrecognized commands or Invalid line sequences are detected. *May be replaced with phy1/phy2.
phy*1_hs_err	HS Synch pattern with error detected on data lanes. *May be replaced with phy1/phy2.
ctrl1_csi_err_ l	ECC or CRC errors detected in CSI-2 controller, low byte
ctrl1_csi_err_ h	Packets terminated early or/and Frame count error detected in CSI-2 controller

General-Purpose Input and Output (GPIO)

Overview

The MAX96717 and MAX96717F have 11 multi-function 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 (e.g., I2C, LOCK, ERRB, etc.). The MAX96717R has seven multi-function pins (MFPs) with either full or partial GPIO capability depending on the pin. This section explains the GPIO function of MFP pins. Refer to the datasheets for additional details regarding GPIO capabilities and default states after power-up.

The GPIO blocks of the MAX96717 family communicate and regenerate state changes of GPIO pins from one side of the serial link to the other. An input GPIO value on one side of the GMSL link may be sent to any of the GPIO outputs on the opposite side of the link.

Operation

GPIO pin mapping is coordinated across the serial link via 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 via registers.

The MAX96717 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 (i.e., the first three GPIO registers correspond to GPIO0, then next three to GPIO1, etc.). Additional details related to these registers can be found in the GPIO Registers section.

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 will be ignored by Rx since Rx is already high. All subsequent transitions will be correctly observed.

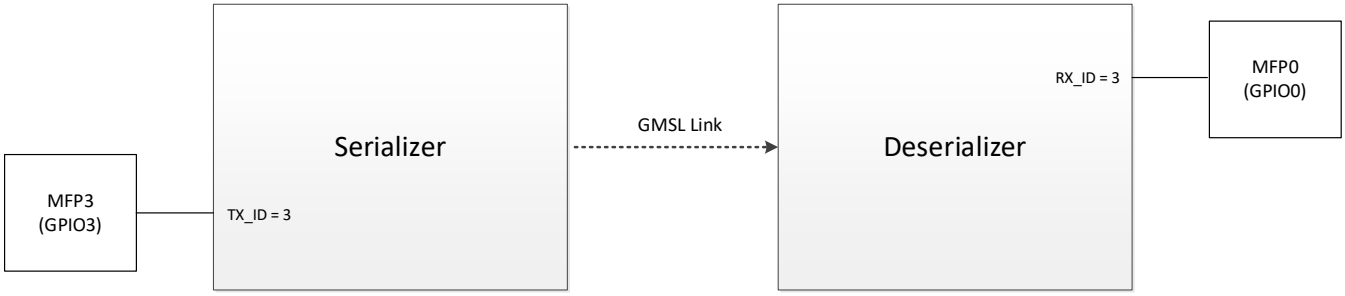


Figure 34. GPIO Forwarding Example with a Transition from MFP3 to MFP0

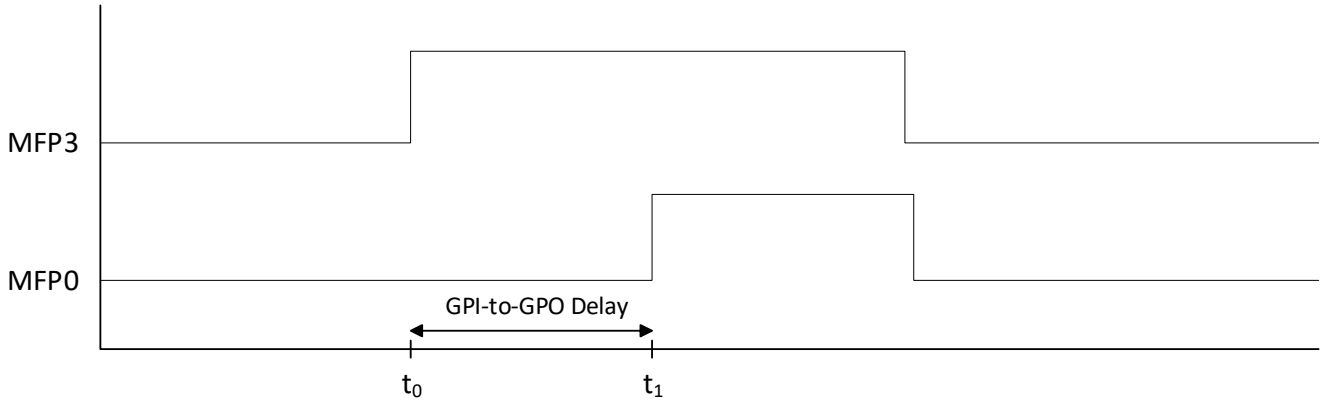


Figure 35. GPIO Forwarding Timing Diagram

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

The MAX96717 and MAX96717F each have 11 MFPs; five of those MFPs are GPIO (general-purpose input or output), two are GPO (general-purpose output), and four are ODO_GPI (open drain output or general-purpose input). The MAX96717R has seven MFPs; five of those MFPs are GPIO (general-purpose input or output) and the other two are ODO_GPI (open drain output or general-purpose input). The table below shows the GPIO capabilities of each MFP for the different devices:

Table 47. MFP Capabilities

	MAX96717/MAX96717F	MAX96717R
MFP0	GPIO0	GPIO0
MFP1	GPO1	N/A
MFP2	GPO2	N/A
MFP3	GPIO3	GPIO3
MFP4	GPIO4	GPIO4
MFP5	ODO5_GPI5	ODO5_GPI5
MFP6	ODO6_GPI6	ODO6_GPI6
MFP7	GPIO7	GPIO7
MFP8	GPIO8	GPIO8
MFP9	ODO9_GPI9	N/A
MFP10	ODO10_GPI10	N/A

GPIO Pull-Up and Pull-Down Resistor Setup

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

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

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

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

- 0: 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 (i.e., NMOS output driver enabled) by writing *OUT_TYPE* = 0 and for Push-Pull mode (i.e., both NMOS and PMOS output driver enabled) by writing *OUT_TYPE* = 1.

Configuring GPIO Forwarding

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

Configuring Input GPIO:

1. Set *GPIO_TX_ID* with a value from 0–31 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 0–31 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. An example diagram of this configuration is shown below:

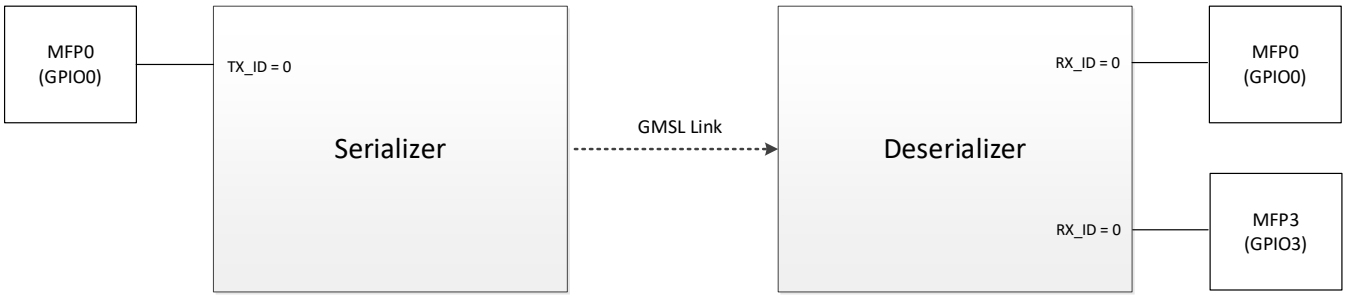


Figure 36. GPIO Broadcasting

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 shown below:

Table 48. Delay Values with and without Compensation

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

Table 49. Compensation Delay Registers

Register	Bits	Default Value	Description
0x30	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 1.7µs. Default delay is 3.4µs.
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 1.7µs. Default delay is 3.4µs.

Toggling GPIO Manually with Registers

GPIO pins can be manually controlled via 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 50. 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 MAX96717 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 prior to setting up the GPIO Tx side to prevent asynchronous states.

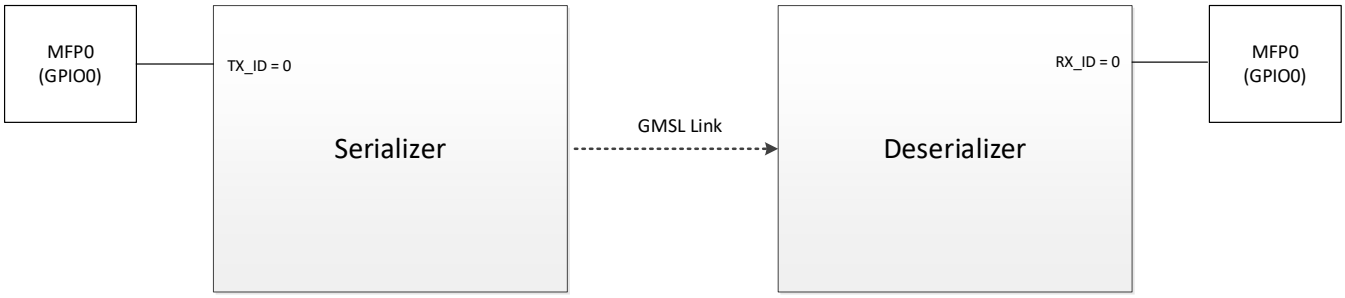


Figure 37. GPIO Forwarding Programming Example

Table 51. 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 1MΩ	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 MAX96717's multi-function 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 datasheet for the relevant register map and MFP speed grouping details.

The MFP edge transitions must be fast enough to meet the application's requirement; however, the high-speed I/O of the GMSL link and video protocols (e.g., MIPI) are sensitive to coupling and crosstalk from MFP transitions. 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 VDDIO supply values will 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 (i.e., 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 pullup resistor.
2. **Dedicated Function**: The rise and fall times of MFP pins assigned dedicated functions (e.g., 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 VDDIO supply voltage affects the I/O slew rate. The impact of the chosen VDDIO 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 will be 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 will be 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 will automatically be adjusted to meet the applicable specification.

The device VDDIO level determines the available range of the slew rate configuration options. For each VDDIO 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 MAX96717.
Refer to the corresponding device's data sheet "Control- and Side-Channel Typical Rise and Fall Times" section for VDDIO timing details. Typical rise and fall times for GMSL2 devices are presented in the table below.

Table 52. 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 MAX96717 CMU4 (0x304) register to configure the MFP slew rate. This register has the following mapping:

Table 53. 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	

Note: Register mappings vary by device family; refer to the device-specific data sheet for register mapping details.

Video PRBS Generator and Checker

Overview

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

In conjunction with the VPRBS generator/checker, the MAX96717/F/R's error generator feature can also be used to validate that the errors are seen on the deserializer. The MIPI input to the serializer must be disabled prior to running this test.

Serializer Video PRBS & Status Registers

Table 54. Serializer Video PRBS Generator and Checker Registers

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

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

MAX96724 Deserializer (Not Serializer) Video PRBS Registers

Table 55. Deserializer Video PRBS Generator and Checker Registers

Register	Bits	Default Value	Description	Decode
0x01DC 0x01FC 0x021C 0x023C	7	0x1	Pattern generator clock source for video PRBS7, PRBS9, PRBS24, checkerboard, and gradient patterns.	0b0: 150MHz 0b1: 375MHz (default)
	4	0b0	Enables video PRBS24 generator/checker	0b0: Disabled 0b1: Enabled
	3	0b0	Enables video PRBS7 generator/checker	0b0: Disabled 0b1: Enabled
	2	0b0	Enables video PRBS9 generator/checker	0b0: Disabled 0b1: Enabled
0x01DB 0x01FB 0x021B 0x023B	7:0	0x00	Video PRBS error counter, clears on read	0xXX: Number of video PRBS errors since last read

0x0029	2	0b1	Enables reporting of video PRBS errors (VPRBS_ERR_FLAG—0x2A) at ERRB pin.	0b0: Disable video PRBS error reporting 0b1: Enable video PRBS error reporting
0x002A	2	0b0	Video PRBS Error Flag. Asserted when VPRBS_ERR (0x1DB) > 0.	0b0: VPRBS_ERR = 0 0b1: VPRBS_ERR > 0

Video PRBS Programming Examples

Serializer Generated VPRBS

The following script will configure a single MAX96717 and a MAX96724 to conduct the standard VPRBS test. In this test, the serializer will generate a PRBS pattern, and the deserializer will check it. This test requires that the serializer has PCLK and that no video is running into the serializer.

```
# Connect Serializer GMSL link to Deserializer GMSL link B

#Enable Deserializer link B only
0x4E,0x0006,0xF2
#Disable auto bpp on serializer
0x80,0x0110,0x60
#Enable serializer internal PCLK generation, PCLK = 150MHz
0x80,0x024F,0x0D
#Enable serializer pattern generator
0x80,0x026B,0x01
#Delay for 3ms
#Enable PRBS(2^24) checker for deserializer pipe 1
0x4E,0x01FC,0x10
#Delay for 3ms
#Enable serializer VPRBS generator
0x80,0x026B,0x81

#Verify serializer has PCLKDET = 1
0x80,0x0112,0x8A
#Verify deserializer has VIDEO_LOCK = 1 for Pipe 1
0x4E,0x01FC,0x11
#Verify deserializer does not have PRBS errors VPBRB_ERR = 0x00
0x4E,0x01FB,0x00

#Optional Step: Enable Serializer errors to verify setup and PRBS error detection
#Note: Error generation also creates decode and idle errors, so these must be
cleared as well.
#Enable serializer error generator
0x80,0x0029,0x18
#Delay for a short time to accumulate errors.
#Disable serializer error generator
0x80,0x0029,0x08
#Verify deserializer has PRBS errors VPBRB_ERR > 0x00, 0xFF is used as the read
value in this example, but errors may vary. VPRBS errors should clear after reading
this register.
0x4E,0x01FB,0xFF
```

Video Timing Generator (VTG) and Video Pattern Generator (VPG)

Overview

The video timing generator (VTG) generates or adjusts video sync signals. It may be used to modify the timing details of incoming video streams, or it may be used with the video pattern eneratorg (VPG) to generate test images. The VTG and VPG are implemented within the video pipe of the serializer. The MAX96717, MAX96717F, and MAX96717R all have the same VTG and VPG functionality.

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

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

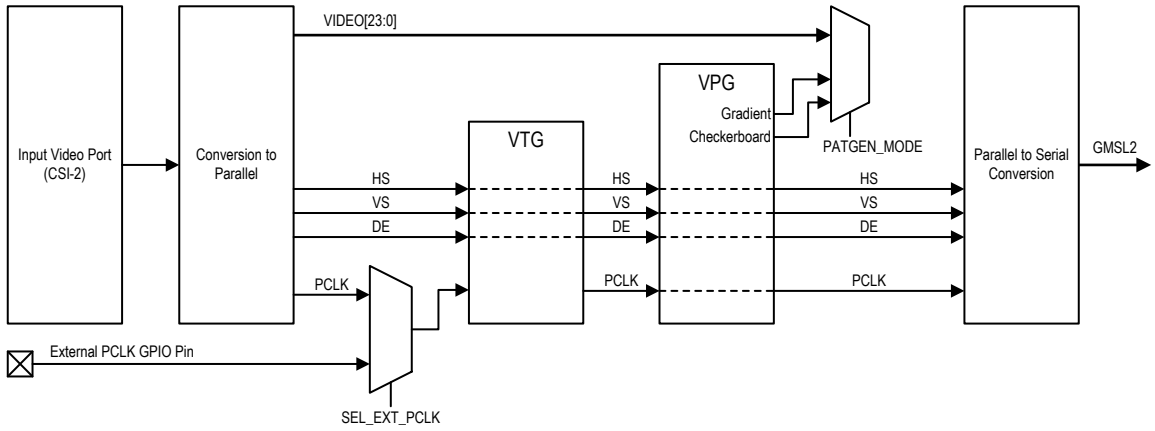


Figure 38. Video Path Through VTG and VPG

Video Timing Generator Operation

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

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

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

VTG Configuration

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

Table 56. VTG Operation Mode

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

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

VTG Trigger Modes

There are four different VTG Trigger modes:

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

VTG Timing Parameters

The sync pulse timing parameters for the VTG are shown in Figure 39. Refer to Table 57 for configuring the timing parameters for the VS, HS, and DE generation.

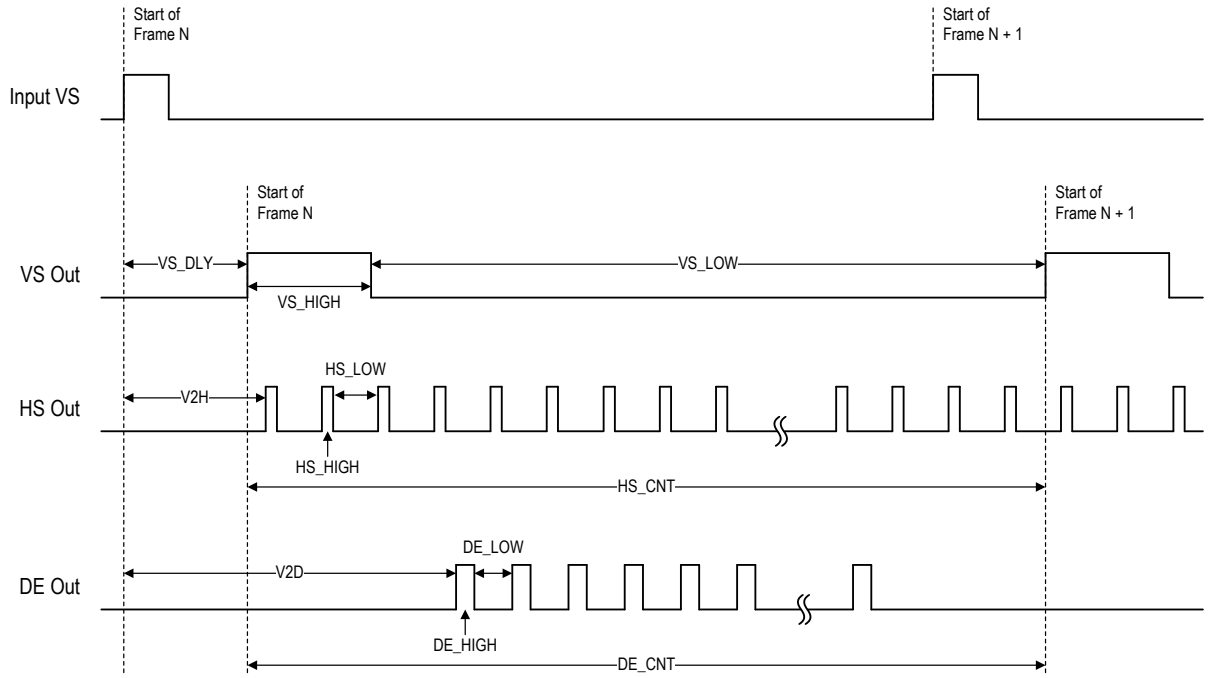


Figure 39. VTG Timing Parameters

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

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

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

Table 57. Timing Parameter Configuration Registers

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

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

Video Pattern Generator Operation

The MAX96717 family has an internal video pattern generator (VPG) that accommodates a wide range of resolutions and frame rates. The VPG does not require an external PCLK source from the CSI input, and uses the external 25MHz crystal clock (i.e., REFCLK input) to derive four different pixel clock (PCLK) options. 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 either the input video stream or user-configured timing values in the associated VTG block. 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 40 and Figure 41, 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 will pop-up as shown in the below figure.

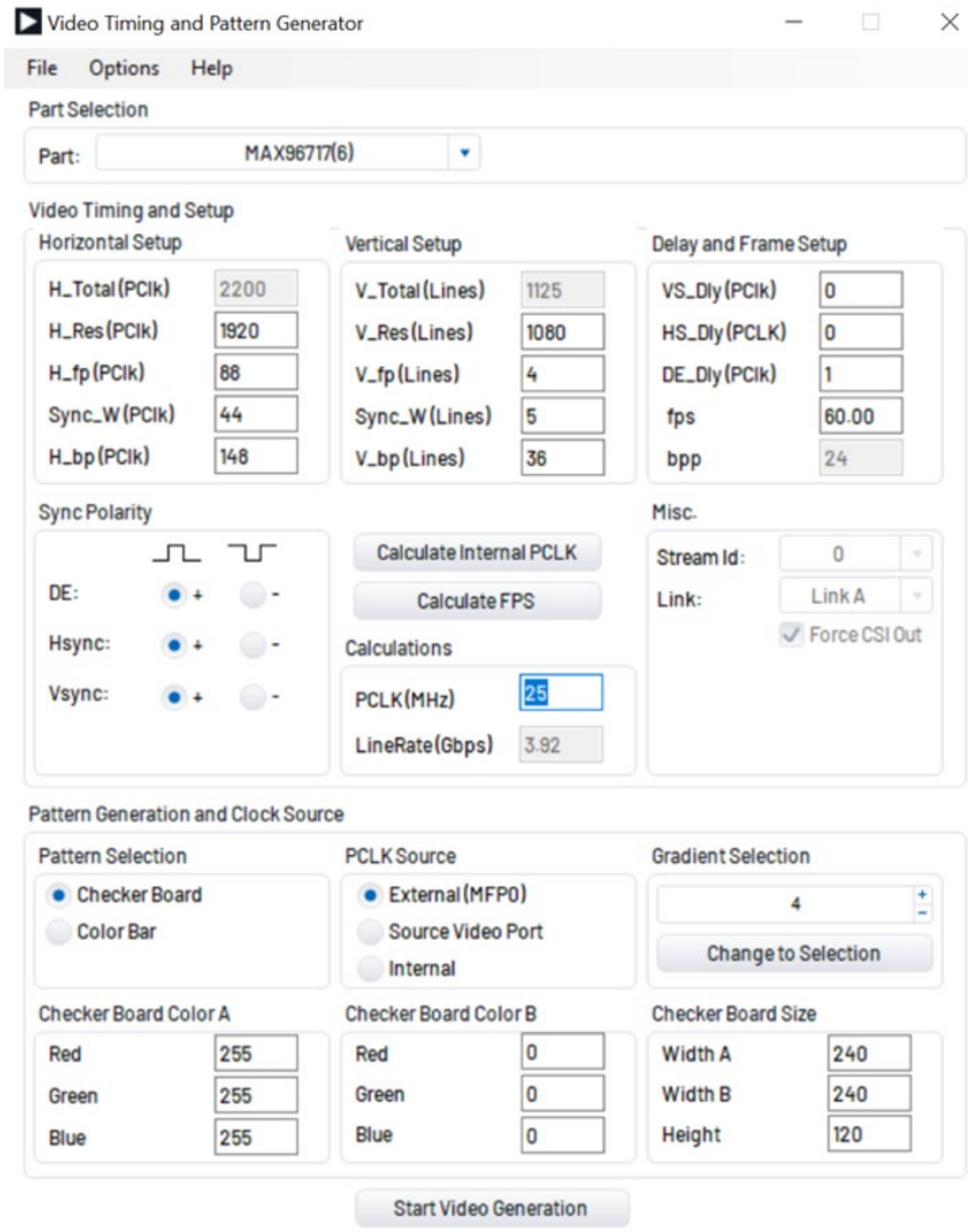


Figure 40. GMSL GUI Video Timing and Pattern Generator

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

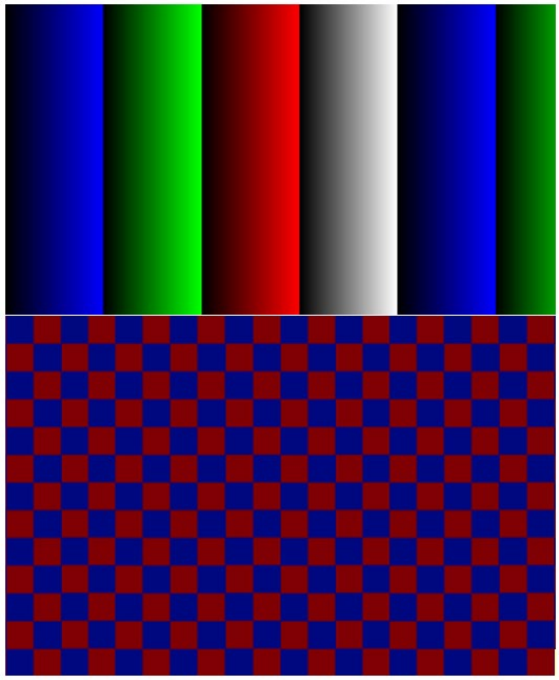


Figure 41. VPG Pattern Options, Gradient (Top) and Checkerboard (Bottom)

VPG Configuration

In Gradient mode, the length of the gradient pattern and gradient direction are configurable. In Checkerboard mode, two colors (i.e., color A and color B) are selected and the size of the pattern is user-defined. The VPG can generate in RGB888 format and Pixel mode only, not in Tunnel mode. The configuration registers for the MAX96717/F/R are listed in the table below.

Table 58. VPG Configuration Registers for MAX96717 Family

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.

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 MAX96717 serializers.

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

Table 59 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

RCLKOUT to send the clock signal to the image sensor is not included in the scripts. See the RoR and RCLKOUT section for information on how to turn enable it.

Use Case Example 1

This example has the following characteristics:

- Four MAX96717s connected to one MAX96724
- Pixel mode
- Initial I2C address (0x80) of serializers is changed to 0x80, 0x82, 0x84, and 0x86
- I2C address of deserializer is 0x4E
- Link rate = 6 Gbps
- Each serializer receives RAW12 and embedded 8 on its MIPI input port with VC = 0
- The deserializer reassigns the VCs (from 0 to 0-3) and outputs the data on Port A at 1300 Mbps

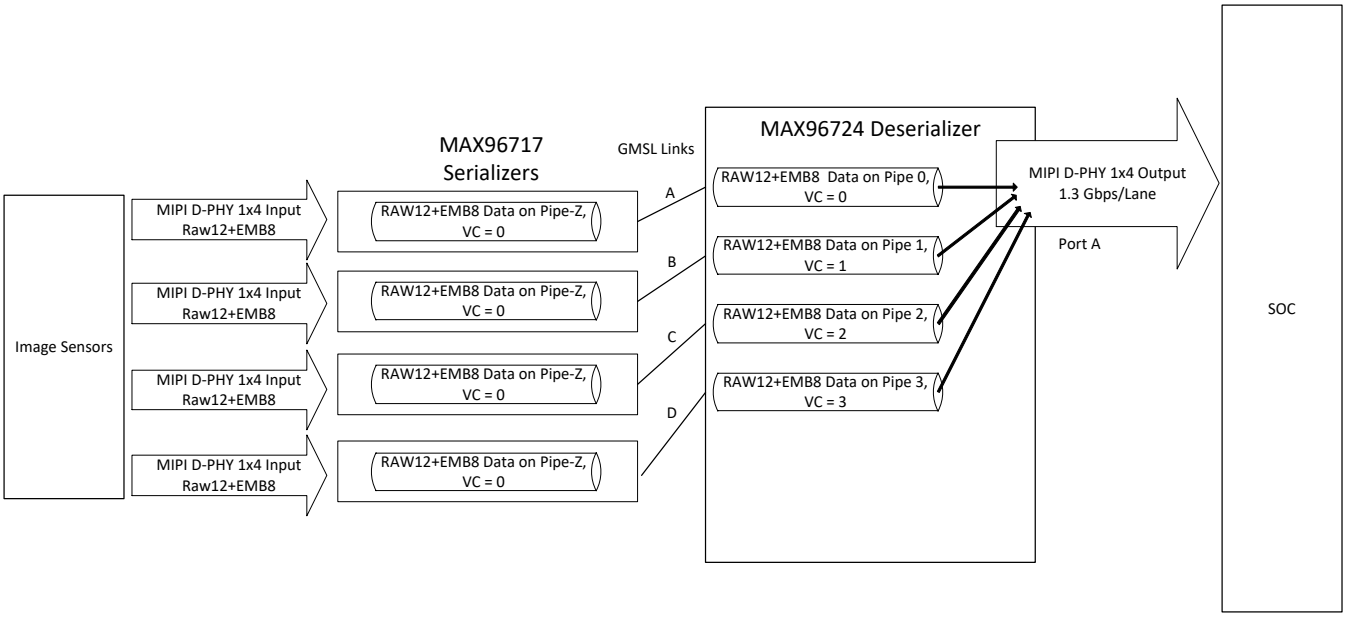


Figure 42. Use Case Example 1

Use Case Example 1 Script

```
// GMSL-A / Serializer: MAX96717 (Pixel Mode) / Mode: 1x4 / Device Address: 0x80 / Multiple-VC Case:
Single VC / Multiple-VC Pipe Sharing: N/A
// PipeZ:
// Input Stream: VC0 RAW12 PortB (D-PHY)
// Input Stream: VC0 EMB8 PortB (D-PHY)

// GMSL-B / Serializer: MAX96717 (Pixel Mode) / Mode: 1x4 / Device Address: 0x80 / Multiple-VC Case:
Single VC / Multiple-VC Pipe Sharing: N/A
// PipeZ:
// Input Stream: VC0 RAW12 PortB (D-PHY)
// Input Stream: VC0 EMB8 PortB (D-PHY)

// GMSL-C / Serializer: MAX96717 (Pixel Mode) / Mode: 1x4 / Device Address: 0x80 / Multiple-VC Case:
Single VC / Multiple-VC Pipe Sharing: N/A
// PipeZ:
// Input Stream: VC0 RAW12 PortB (D-PHY)
// Input Stream: VC0 EMB8 PortB (D-PHY)

// GMSL-D / Serializer: MAX96717 (Pixel Mode) / Mode: 1x4 / Device Address: 0x80 / Multiple-VC Case:
Single VC / Multiple-VC Pipe Sharing: N/A
// PipeZ:
// Input Stream: VC0 RAW12 PortB (D-PHY)
// Input Stream: VC0 EMB8 PortB (D-PHY)

// Deserializer: MAX96724 / Mode: 2 (1x4) / Device Address: 0x4E
// Pipe0:
// GMSL-A Input Stream: VC0 RAW12 PortB - Output Stream: VC0 RAW12 PortA (D-PHY)
// GMSL-A Input Stream: VC0 EMB8 PortB - Output Stream: VC0 EMB8 PortA (D-PHY)
// Pipe1:
// GMSL-B Input Stream: VC0 RAW12 PortB - Output Stream: VC1 RAW12 PortA (D-PHY)
// GMSL-B Input Stream: VC0 EMB8 PortB - Output Stream: VC1 EMB8 PortA (D-PHY)
// Pipe2:
// GMSL-C Input Stream: VC0 RAW12 PortB - Output Stream: VC2 RAW12 PortA (D-PHY)
// GMSL-C Input Stream: VC0 EMB8 PortB - Output Stream: VC2 EMB8 PortA (D-PHY)
// Pipe3:
// GMSL-D Input Stream: VC0 RAW12 PortB - Output Stream: VC3 RAW12 PortA (D-PHY)
// GMSL-D Input Stream: VC0 EMB8 PortB - Output Stream: VC3 EMB8 PortA (D-PHY)
```

```
0x04,0x4E,0x04,0x0B,0x00, // BACKTOP : BACKTOP12 | CSI_OUT_EN (CSI_OUT_EN): CSI output disabled
// Single Link Initialization Before Serializer Device Address Change
0x04,0x4E,0x00,0x06,0xF2, // DEV : REG6 | LINK_EN_A (LINK_EN_A): Disabled | (Default) LINK_EN_B
(LINK_EN_B): Enabled | LINK_EN_C (LINK_EN_C): Disabled | LINK_EN_D (LINK_EN_D): Disabled
0x00,0x78,
// GMSL-B Serializer Address Change from 0x80 to 0x82
0x04,0x80,0x00,0x00,0x82, // DEV : REG0 | DEV_ADDR (DEV_ADDR): 0x41
// Single Link Initialization Before Serializer Device Address Change
0x04,0x4E,0x00,0x06,0xF4, // DEV : REG6 | (Default) LINK_EN_A (LINK_EN_A): Disabled | LINK_EN_B
(LINK_EN_B): Disabled | LINK_EN_C (LINK_EN_C): Enabled | (Default) LINK_EN_D (LINK_EN_D): Disabled
0x00,0x78,
// GMSL-C Serializer Address Change from 0x80 to 0x84
0x04,0x80,0x00,0x00,0x84, // DEV : REG0 | DEV_ADDR (DEV_ADDR): 0x42
// Single Link Initialization Before Serializer Device Address Change
0x04,0x4E,0x00,0x06,0xF8, // DEV : REG6 | (Default) LINK_EN_A (LINK_EN_A): Disabled | (Default)
LINK_EN_B (LINK_EN_B): Disabled | LINK_EN_C (LINK_EN_C): Disabled | LINK_EN_D (LINK_EN_D): Enabled
0x00,0x78,
// GMSL-D Serializer Address Change from 0x80 to 0x86
0x04,0x80,0x00,0x00,0x86, // DEV : REG0 | DEV_ADDR (DEV_ADDR): 0x43
// Link Initialization for Deserializer
0x04,0x4E,0x00,0x06,0xFF, // DEV : REG6 | LINK_EN_A (LINK_EN_A): Enabled | LINK_EN_B (LINK_EN_B):
Enabled | LINK_EN_C (LINK_EN_C): Enabled | (Default) LINK_EN_D (LINK_EN_D): Enabled
0x00,0x78,
// Video Transmit Configuration for Serializer(s)
0x04,0x80,0x00,0x02,0x03, // DEV : REG2 | VID_TX_EN_Z (VID_TX_EN_Z): Disabled
0x04,0x82,0x00,0x02,0x03, // DEV : REG2 | VID_TX_EN_Z (VID_TX_EN_Z): Disabled
0x04,0x84,0x00,0x02,0x03, // DEV : REG2 | VID_TX_EN_Z (VID_TX_EN_Z): Disabled
0x04,0x86,0x00,0x02,0x03, // DEV : REG2 | VID_TX_EN_Z (VID_TX_EN_Z): Disabled
//
// INSTRUCTIONS FOR GMSL-A SERIALIZER MAX96717
//
// MIPI DPHY Configuration
0x04,0x80,0x03,0x30,0x00, // MIPI_RX : MIPI_RX0 | (Default) RSVD (Port Configuration): 1x4
0x04,0x80,0x03,0x83,0x00, // MIPI_RX_EXT : EXT11 | Tun_Mode (Tunnel Mode): Disabled
0x04,0x80,0x03,0x31,0x10, // MIPI_RX : MIPI_RX1 | ctrl1_num_lanes (Port B - Lane Count): 2
0x04,0x80,0x03,0x32,0xE0, // MIPI_RX : MIPI_RX2 | (Default) phy1_lane_map (Lane Map - PHY1 D0): Lane 2 |
(Default) phy1_lane_map (Lane Map - PHY1 D1): Lane 3
0x04,0x80,0x03,0x33,0x04, // MIPI_RX : MIPI_RX3 | (Default) phy2_lane_map (Lane Map - PHY2 D0): Lane 0 |
(Default) phy2_lane_map (Lane Map - PHY2 D1): Lane 1
0x04,0x80,0x03,0x34,0x00, // MIPI_RX : MIPI_RX4 | (Default) phy1_pol_map (Polarity - PHY1 Lane 0):
Normal | (Default) phy1_pol_map (Polarity - PHY1 Lane 1): Normal
0x04,0x80,0x03,0x35,0x00, // MIPI_RX : MIPI_RX5 | (Default) phy2_pol_map (Polarity - PHY2 Lane 0):
Normal | (Default) phy2_pol_map (Polarity - PHY2 Lane 1): Normal | (Default) phy2_pol_map (Polarity -
PHY2 Clock Lane): Normal
// Controller to Pipe Mapping Configuration
0x04,0x80,0x03,0x08,0x64, // FRONTTOP : FRONTTOP_0 | (Default) RSVD (CLK_SELZ): Port B | (Default)
START_PORTB (START_PORTB): Enabled
0x04,0x80,0x03,0x11,0x40, // FRONTTOP : FRONTTOP_9 | (Default) START_PORTBZ (START_PORTBZ): Start Video
0x04,0x80,0x03,0x18,0x6C, // FRONTTOP : FRONTTOP_16 | mem_dt1_selz (mem_dt1_selz): 0x6C
0x04,0x80,0x03,0x19,0x52, // FRONTTOP : FRONTTOP_17 | mem_dt2_selz (mem_dt2_selz): 0x52
0x04,0x80,0x03,0x15,0x80, //  (independent_vs_mode): Enabled
0x04,0x80,0x03,0x0D,0x01, // FRONTTOP : FRONTTOP_5 | VC_SELZ_L (VC_SELZ_L): 0x1
// Double Mode Configuration
0x04,0x80,0x01,0x10,0x60, // VID_TX__Z : VIDEO_TX0 | AUTO_BPP (AUTO_BPP Pipe Z): Use bpp from bpp
register
0x04,0x80,0x01,0x11,0x50, // VID_TX__Z : VIDEO_TX1 | BPP (BPP Pipe Z): 0x10
0x04,0x80,0x01,0x12,0x08, // VID_TX__Z : VIDEO_TX2 | RSVD (DRIFT_DET_EN Pipe Z): Disable PCLK frequency
drift detection
0x04,0x80,0x03,0x12,0x04, // FRONTTOP : FRONTTOP_10 | bpp8dblz (bpp8dblz): Send 8-bit pixels as 16-bit
0x04,0x80,0x03,0x1E,0x2C, // FRONTTOP : FRONTTOP_22 | soft_bppz (soft_bppz): 0xC | soft_bppz_en
(soft_bppz_en): Software override enabled
// Pipe Configuration
0x04,0x80,0x00,0x5B,0x00, // CFGV__VIDEO_Z : TX3 | TX_STR_SEL (TX_STR_SEL Pipe Z): 0x0
//
// INSTRUCTIONS FOR GMSL-B SERIALIZER MAX96717
//
// MIPI DPHY Configuration
0x04,0x82,0x03,0x30,0x00, // MIPI_RX : MIPI_RX0 | (Default) RSVD (Port Configuration): 1x4
0x04,0x82,0x03,0x83,0x00, // MIPI_RX_EXT : EXT11 | Tun_Mode (Tunnel Mode): Disabled
0x04,0x82,0x03,0x31,0x10, // MIPI_RX : MIPI_RX1 | ctrl1_num_lanes (Port B - Lane Count): 2
```

```
0x04,0x82,0x03,0x32,0xE0, // MIPI_RX : MIPI_RX2 | (Default) phy1_lane_map (Lane Map - PHY1 D0): Lane 2 |
(Default) phy1_lane_map (Lane Map - PHY1 D1): Lane 3
0x04,0x82,0x03,0x33,0x04, // MIPI_RX : MIPI_RX3 | (Default) phy2_lane_map (Lane Map - PHY2 D0): Lane 0 |
(Default) phy2_lane_map (Lane Map - PHY2 D1): Lane 1
0x04,0x82,0x03,0x34,0x00, // MIPI_RX : MIPI_RX4 | (Default) phy1_pol_map (Polarity - PHY1 Lane 0):
Normal | (Default) phy1_pol_map (Polarity - PHY1 Lane 1): Normal
0x04,0x82,0x03,0x35,0x00, // MIPI_RX : MIPI_RX5 | (Default) phy2_pol_map (Polarity - PHY2 Lane 0):
Normal | (Default) phy2_pol_map (Polarity - PHY2 Lane 1): Normal | (Default) phy2_pol_map (Polarity -
PHY2 Clock Lane): Normal
// Controller to Pipe Mapping Configuration
0x04,0x82,0x03,0x08,0x64, // FRONTTOP : FRONTTOP_0 | (Default) RSVD (CLK_SELZ): Port B | (Default)
START_PORTB (START_PORTB): Enabled
0x04,0x82,0x03,0x11,0x40, // FRONTTOP : FRONTTOP_9 | (Default) START_PORTBZ (START_PORTBZ): Start Video
0x04,0x82,0x03,0x18,0x6C, // FRONTTOP : FRONTTOP_16 | mem_dt1_selz (mem_dt1_selz): 0x6C
0x04,0x82,0x03,0x19,0x52, // FRONTTOP : FRONTTOP_17 | mem_dt2_selz (mem_dt2_selz): 0x52
0x04,0x82,0x03,0x15,0x80, //  (independent_vs_mode): Enabled
0x04,0x82,0x03,0x0D,0x01, // FRONTTOP : FRONTTOP_5 | VC_SELZ_L (VC_SELZ_L): 0x1
// Double Mode Configuration
0x04,0x82,0x01,0x10,0x60, // VID_TX__Z : VIDEO_TX0 | AUTO_BPP (AUTO_BPP Pipe Z): Use bpp from bpp
register
0x04,0x82,0x01,0x11,0x50, // VID_TX__Z : VIDEO_TX1 | BPP (BPP Pipe Z): 0x10
0x04,0x82,0x01,0x12,0x08, // VID_TX__Z : VIDEO_TX2 | RSVD (DRIFT_DET_EN Pipe Z): Disable PCLK frequency
drift detection
0x04,0x82,0x03,0x12,0x04, // FRONTTOP : FRONTTOP_10 | bpp8dblz (bpp8dblz): Send 8-bit pixels as 16-bit
0x04,0x82,0x03,0x1E,0x2C, // FRONTTOP : FRONTTOP_22 | soft_bppz (soft_bppz): 0xC | soft_bppz_en
(soft_bppz_en): Software override enabled
// Pipe Configuration
0x04,0x82,0x00,0x5B,0x00, // CFGV__VIDEO_Z : TX3 | TX_STR_SEL (TX_STR_SEL Pipe Z): 0x0
//
// INSTRUCTIONS FOR GMSL-C SERIALIZER MAX96717
//
// MIPI DPHY Configuration
0x04,0x84,0x03,0x30,0x00, // MIPI_RX : MIPI_RX0 | (Default) RSVD (Port Configuration): 1x4
0x04,0x84,0x03,0x83,0x00, // MIPI_RX_EXT : EXT11 | Tun_Mode (Tunnel Mode): Disabled
0x04,0x84,0x03,0x31,0x10, // MIPI_RX : MIPI_RX1 | ctrl1_num_lanes (Port B - Lane Count): 2
0x04,0x84,0x03,0x32,0xE0, // MIPI_RX : MIPI_RX2 | (Default) phy1_lane_map (Lane Map - PHY1 D0): Lane 2 |
(Default) phy1_lane_map (Lane Map - PHY1 D1): Lane 3
0x04,0x84,0x03,0x33,0x04, // MIPI_RX : MIPI_RX3 | (Default) phy2_lane_map (Lane Map - PHY2 D0): Lane 0 |
(Default) phy2_lane_map (Lane Map - PHY2 D1): Lane 1
0x04,0x84,0x03,0x34,0x00, // MIPI_RX : MIPI_RX4 | (Default) phy1_pol_map (Polarity - PHY1 Lane 0):
Normal | (Default) phy1_pol_map (Polarity - PHY1 Lane 1): Normal
0x04,0x84,0x03,0x35,0x00, // MIPI_RX : MIPI_RX5 | (Default) phy2_pol_map (Polarity - PHY2 Lane 0):
Normal | (Default) phy2_pol_map (Polarity - PHY2 Lane 1): Normal | (Default) phy2_pol_map (Polarity -
PHY2 Clock Lane): Normal
// Controller to Pipe Mapping Configuration
0x04,0x84,0x03,0x08,0x64, // FRONTTOP : FRONTTOP_0 | (Default) RSVD (CLK_SELZ): Port B | (Default)
START_PORTB (START_PORTB): Enabled
0x04,0x84,0x03,0x11,0x40, // FRONTTOP : FRONTTOP_9 | (Default) START_PORTBZ (START_PORTBZ): Start Video
0x04,0x84,0x03,0x18,0x6C, // FRONTTOP : FRONTTOP_16 | mem_dt1_selz (mem_dt1_selz): 0x6C
0x04,0x84,0x03,0x19,0x52, // FRONTTOP : FRONTTOP_17 | mem_dt2_selz (mem_dt2_selz): 0x52
0x04,0x84,0x03,0x15,0x80, //  (independent_vs_mode): Enabled
0x04,0x84,0x03,0x0D,0x01, // FRONTTOP : FRONTTOP_5 | VC_SELZ_L (VC_SELZ_L): 0x1
// Double Mode Configuration
0x04,0x84,0x01,0x10,0x60, // VID_TX__Z : VIDEO_TX0 | AUTO_BPP (AUTO_BPP Pipe Z): Use bpp from bpp
register
0x04,0x84,0x01,0x11,0x50, // VID_TX__Z : VIDEO_TX1 | BPP (BPP Pipe Z): 0x10
0x04,0x84,0x01,0x12,0x08, // VID_TX__Z : VIDEO_TX2 | RSVD (DRIFT_DET_EN Pipe Z): Disable PCLK frequency
drift detection
0x04,0x84,0x03,0x12,0x04, // FRONTTOP : FRONTTOP_10 | bpp8dblz (bpp8dblz): Send 8-bit pixels as 16-bit
0x04,0x84,0x03,0x1E,0x2C, // FRONTTOP : FRONTTOP_22 | soft_bppz (soft_bppz): 0xC | soft_bppz_en
(soft_bppz_en): Software override enabled
// Pipe Configuration
0x04,0x84,0x00,0x5B,0x00, // CFGV__VIDEO_Z : TX3 | TX_STR_SEL (TX_STR_SEL Pipe Z): 0x0
//
// INSTRUCTIONS FOR GMSL-D SERIALIZER MAX96717
//
// MIPI DPHY Configuration
0x04,0x86,0x03,0x30,0x00, // MIPI_RX : MIPI_RX0 | (Default) RSVD (Port Configuration): 1x4
0x04,0x86,0x03,0x83,0x00, // MIPI_RX_EXT : EXT11 | Tun_Mode (Tunnel Mode): Disabled
0x04,0x86,0x03,0x31,0x10, // MIPI_RX : MIPI_RX1 | ctrl1_num_lanes (Port B - Lane Count): 2
```

```
0x04,0x86,0x03,0x32,0xE0, // MIPI_RX : MIPI_RX2 | (Default) phy1_lane_map (Lane Map - PHY1 D0): Lane 2 |
(Default) phy1_lane_map (Lane Map - PHY1 D1): Lane 3
0x04,0x86,0x03,0x33,0x04, // MIPI_RX : MIPI_RX3 | (Default) phy2_lane_map (Lane Map - PHY2 D0): Lane 0 |
(Default) phy2_lane_map (Lane Map - PHY2 D1): Lane 1
0x04,0x86,0x03,0x34,0x00, // MIPI_RX : MIPI_RX4 | (Default) phy1_pol_map (Polarity - PHY1 Lane 0):
Normal | (Default) phy1_pol_map (Polarity - PHY1 Lane 1): Normal
0x04,0x86,0x03,0x35,0x00, // MIPI_RX : MIPI_RX5 | (Default) phy2_pol_map (Polarity - PHY2 Lane 0):
Normal | (Default) phy2_pol_map (Polarity - PHY2 Lane 1): Normal | (Default) phy2_pol_map (Polarity -
PHY2 Clock Lane): Normal
// Controller to Pipe Mapping Configuration
0x04,0x86,0x03,0x08,0x64, // FRONTTOP : FRONTTOP_0 | (Default) RSVD (CLK_SELZ): Port B | (Default)
START_PORTB (START_PORTB): Enabled
0x04,0x86,0x03,0x11,0x40, // FRONTTOP : FRONTTOP_9 | (Default) START_PORTBZ (START_PORTBZ): Start Video
0x04,0x86,0x03,0x18,0x6C, // FRONTTOP : FRONTTOP_16 | mem_dt1_selz (mem_dt1_selz): 0x6C
0x04,0x86,0x03,0x19,0x52, // FRONTTOP : FRONTTOP_17 | mem_dt2_selz (mem_dt2_selz): 0x52
0x04,0x86,0x03,0x15,0x80, //  (independent_vs_mode): Enabled
0x04,0x86,0x03,0x0D,0x01, // FRONTTOP : FRONTTOP_5 | VC_SELZ_L (VC_SELZ_L): 0x1
// Double Mode Configuration
0x04,0x86,0x01,0x10,0x60, // VID_TX__Z : VIDEO_TX0 | AUTO_BPP (AUTO_BPP Pipe Z): Use bpp from bpp
register
0x04,0x86,0x01,0x11,0x50, // VID_TX__Z : VIDEO_TX1 | BPP (BPP Pipe Z): 0x10
0x04,0x86,0x01,0x12,0x08, // VID_TX__Z : VIDEO_TX2 | RSVD (DRIFT_DET_EN Pipe Z): Disable PCLK frequency
drift detection
0x04,0x86,0x03,0x12,0x04, // FRONTTOP : FRONTTOP_10 | bpp8dblz (bpp8dblz): Send 8-bit pixels as 16-bit
0x04,0x86,0x03,0x1E,0x2C, // FRONTTOP : FRONTTOP_22 | soft_bppz (soft_bppz): 0xC | soft_bppz_en
(soft_bppz_en): Software override enabled
// Pipe Configuration
0x04,0x86,0x00,0x5B,0x00, // CFGV__VIDEO_Z : TX3 | TX_STR_SEL (TX_STR_SEL Pipe Z): 0x0

// INSTRUCTIONS FOR DESERIALIZER MAX96724

// Video Pipes And Routing Configuration
0x04,0x4E,0x00,0xF0,0x40, // VIDEO_PIPE_SEL : VIDEO_PIPE_SEL_0 | (Default) VIDEO_PIPE_SEL_0 (Pipe 0
GMSL2 PHY): A | VIDEO_PIPE_SEL_0 (Pipe 0 Input Pipe): X | (Default) VIDEO_PIPE_SEL_1 (Pipe 1 GMSL2 PHY):
B | VIDEO_PIPE_SEL_1 (Pipe 1 Input Pipe): X
0x04,0x4E,0x00,0xF1,0xC8, // VIDEO_PIPE_SEL : VIDEO_PIPE_SEL_1 | (Default) VIDEO_PIPE_SEL_2 (Pipe 2
GMSL2 PHY): C | VIDEO_PIPE_SEL_2 (Pipe 2 Input Pipe): X | (Default) VIDEO_PIPE_SEL_3 (Pipe 3 GMSL2 PHY):
D | VIDEO_PIPE_SEL_3 (Pipe 3 Input Pipe): X
0x04,0x4E,0x00,0xF4,0x0F, // VIDEO_PIPE_SEL : VIDEO_PIPE_EN | (Default) VIDEO_PIPE_EN (Video Pipe 0):
Enabled | (Default) VIDEO_PIPE_EN (Video Pipe 1): Enabled | (Default) VIDEO_PIPE_EN (Video Pipe 2):
Enabled | (Default) VIDEO_PIPE_EN (Video Pipe 3): Enabled | STREAM_SEL_ALL (Stream Select All): Disabled
// Pipe to Controller Mapping Configuration
0x04,0x4E,0x09,0x0B,0x0F, // MIPI_TX__0 : MIPI_TX11 | MAP_EN_L (MAP_EN_L Pipe 0): 0xF
0x04,0x4E,0x09,0x0C,0x00, // MIPI_TX__0 : MIPI_TX12 | (Default) MAP_EN_H (MAP_EN_H Pipe 0): 0x0
0x04,0x4E,0x09,0x0D,0x2C, // MIPI_TX__0 : MIPI_TX13 | MAP_SRC_0 (MAP_SRC_0 Pipe 0 DT): 0x2C | (Default)
MAP_SRC_0 (MAP_SRC_0 Pipe 0 VC): 0x0
0x04,0x4E,0x09,0x0E,0x2C, // MIPI_TX__0 : MIPI_TX14 | MAP_DST_0 (MAP_DST_0 Pipe 0 DT): 0x2C | (Default)
MAP_DST_0 (MAP_DST_0 Pipe 0 VC): 0x0
0x04,0x4E,0x09,0x0F,0x00, // MIPI_TX__0 : MIPI_TX15 | (Default) MAP_SRC_1 (MAP_SRC_1 Pipe 0 DT): 0x0 |
(Default) MAP_SRC_1 (MAP_SRC_1 Pipe 0 VC): 0x0
0x04,0x4E,0x09,0x10,0x00, // MIPI_TX__0 : MIPI_TX16 | (Default) MAP_DST_1 (MAP_DST_1 Pipe 0 DT): 0x0 |
(Default) MAP_DST_1 (MAP_DST_1 Pipe 0 VC): 0x0
0x04,0x4E,0x09,0x11,0x01, // MIPI_TX__0 : MIPI_TX17 | MAP_SRC_2 (MAP_SRC_2 Pipe 0 DT): 0x1 | (Default)
MAP_SRC_2 (MAP_SRC_2 Pipe 0 VC): 0x0
0x04,0x4E,0x09,0x12,0x01, // MIPI_TX__0 : MIPI_TX18 | MAP_DST_2 (MAP_DST_2 Pipe 0 DT): 0x1 | (Default)
MAP_DST_2 (MAP_DST_2 Pipe 0 VC): 0x0
0x04,0x4E,0x09,0x13,0x12, // MIPI_TX__0 : MIPI_TX19 | MAP_SRC_3 (MAP_SRC_3 Pipe 0 DT): 0x12 | (Default)
MAP_SRC_3 (MAP_SRC_3 Pipe 0 VC): 0x0
0x04,0x4E,0x09,0x14,0x12, // MIPI_TX__0 : MIPI_TX20 | MAP_DST_3 (MAP_DST_3 Pipe 0 DT): 0x12 | (Default)
MAP_DST_3 (MAP_DST_3 Pipe 0 VC): 0x0
0x04,0x4E,0x09,0x2D,0x55, // MIPI_TX__0 : MIPI_TX45 | MAP_DPHY_DEST_0 (MAP_DPHY_DST_0 Pipe 0): 0x1 |
MAP_DPHY_DEST_1 (MAP_DPHY_DST_1 Pipe 0): 0x1 | MAP_DPHY_DEST_2 (MAP_DPHY_DST_2 Pipe 0): 0x1 |
MAP_DPHY_DEST_3 (MAP_DPHY_DST_3 Pipe 0): 0x1
0x04,0x4E,0x09,0x4B,0x0F, // MIPI_TX__1 : MIPI_TX11 | MAP_EN_L (MAP_EN_L Pipe 1): 0xF
0x04,0x4E,0x09,0x4C,0x00, // MIPI_TX__1 : MIPI_TX12 | (Default) MAP_EN_H (MAP_EN_H Pipe 1): 0x0
0x04,0x4E,0x09,0x4D,0x2C, // MIPI_TX__1 : MIPI_TX13 | MAP_SRC_0 (MAP_SRC_0 Pipe 1 DT): 0x2C | (Default)
MAP_SRC_0 (MAP_SRC_0 Pipe 1 VC): 0x0
0x04,0x4E,0x09,0x4E,0x6C, // MIPI_TX__1 : MIPI_TX14 | MAP_DST_0 (MAP_DST_0 Pipe 1 DT): 0x2C | MAP_DST_0
(MAP_DST_0 Pipe 1 VC): 0x1
0x04,0x4E,0x09,0x4F,0x00, // MIPI_TX__1 : MIPI_TX15 | (Default) MAP_SRC_1 (MAP_SRC_1 Pipe 1 DT): 0x0 |
(Default) MAP_SRC_1 (MAP_SRC_1 Pipe 1 VC): 0x0
```

```
0x04,0x4E,0x09,0x50,0x40, // MIPI_TX__1 : MIPI_TX16 | (Default) MAP_DST_1 (MAP_DST_1 Pipe 1 DT): 0x0 |
MAP_DST_1 (MAP_DST_1 Pipe 1 VC): 0x1
0x04,0x4E,0x09,0x51,0x01, // MIPI_TX__1 : MIPI_TX17 | MAP_SRC_2 (MAP_SRC_2 Pipe 1 DT): 0x1 | (Default)
MAP_SRC_2 (MAP_SRC_2 Pipe 1 VC): 0x0
0x04,0x4E,0x09,0x52,0x41, // MIPI_TX__1 : MIPI_TX18 | MAP_DST_2 (MAP_DST_2 Pipe 1 DT): 0x1 | MAP_DST_2
(MAP_DST_2 Pipe 1 VC): 0x1
0x04,0x4E,0x09,0x53,0x12, // MIPI_TX__1 : MIPI_TX19 | MAP_SRC_3 (MAP_SRC_3 Pipe 1 DT): 0x12 | (Default)
MAP_SRC_3 (MAP_SRC_3 Pipe 1 VC): 0x0
0x04,0x4E,0x09,0x54,0x52, // MIPI_TX__1 : MIPI_TX20 | MAP_DST_3 (MAP_DST_3 Pipe 1 DT): 0x12 | MAP_DST_3
(MAP_DST_3 Pipe 1 VC): 0x1
0x04,0x4E,0x09,0x6D,0x55, // MIPI_TX__1 : MIPI_TX45 | MAP_DPHY_DEST_0 (MAP_DPHY_DST_0 Pipe 1): 0x1 |
MAP_DPHY_DEST_1 (MAP_DPHY_DST_1 Pipe 1): 0x1 | MAP_DPHY_DEST_2 (MAP_DPHY_DST_2 Pipe 1): 0x1 |
MAP_DPHY_DEST_3 (MAP_DPHY_DST_3 Pipe 1): 0x1
0x04,0x4E,0x09,0x8B,0x0F, // MIPI_TX__2 : MIPI_TX11 | MAP_EN_L (MAP_EN_L Pipe 2): 0xF
0x04,0x4E,0x09,0x8C,0x00, // MIPI_TX__2 : MIPI_TX12 | (Default) MAP_EN_H (MAP_EN_H Pipe 2): 0x0
0x04,0x4E,0x09,0x8D,0x2C, // MIPI_TX__2 : MIPI_TX13 | MAP_SRC_0 (MAP_SRC_0 Pipe 2 DT): 0x2C | (Default)
MAP_SRC_0 (MAP_SRC_0 Pipe 2 VC): 0x0
0x04,0x4E,0x09,0x8E,0xAC, // MIPI_TX__2 : MIPI_TX14 | MAP_DST_0 (MAP_DST_0 Pipe 2 DT): 0x2C | MAP_DST_0
(MAP_DST_0 Pipe 2 VC): 0x2
0x04,0x4E,0x09,0x8F,0x00, // MIPI_TX__2 : MIPI_TX15 | (Default) MAP_SRC_1 (MAP_SRC_1 Pipe 2 DT): 0x0 |
(Default) MAP_SRC_1 (MAP_SRC_1 Pipe 2 VC): 0x0
0x04,0x4E,0x09,0x90,0x80, // MIPI_TX__2 : MIPI_TX16 | (Default) MAP_DST_1 (MAP_DST_1 Pipe 2 DT): 0x0 |
MAP_DST_1 (MAP_DST_1 Pipe 2 VC): 0x2
0x04,0x4E,0x09,0x91,0x01, // MIPI_TX__2 : MIPI_TX17 | MAP_SRC_2 (MAP_SRC_2 Pipe 2 DT): 0x1 | (Default)
MAP_SRC_2 (MAP_SRC_2 Pipe 2 VC): 0x0
0x04,0x4E,0x09,0x92,0x81, // MIPI_TX__2 : MIPI_TX18 | MAP_DST_2 (MAP_DST_2 Pipe 2 DT): 0x1 | MAP_DST_2
(MAP_DST_2 Pipe 2 VC): 0x2
0x04,0x4E,0x09,0x93,0x12, // MIPI_TX__2 : MIPI_TX19 | MAP_SRC_3 (MAP_SRC_3 Pipe 2 DT): 0x12 | (Default)
MAP_SRC_3 (MAP_SRC_3 Pipe 2 VC): 0x0
0x04,0x4E,0x09,0x94,0x92, // MIPI_TX__2 : MIPI_TX20 | MAP_DST_3 (MAP_DST_3 Pipe 2 DT): 0x12 | MAP_DST_3
(MAP_DST_3 Pipe 2 VC): 0x2
0x04,0x4E,0x09,0xAD,0x55, // MIPI_TX__2 : MIPI_TX45 | MAP_DPHY_DEST_0 (MAP_DPHY_DST_0 Pipe 2): 0x1 |
MAP_DPHY_DEST_1 (MAP_DPHY_DST_1 Pipe 2): 0x1 | MAP_DPHY_DEST_2 (MAP_DPHY_DST_2 Pipe 2): 0x1 |
MAP_DPHY_DEST_3 (MAP_DPHY_DST_3 Pipe 2): 0x1
0x04,0x4E,0x09,0xCB,0x0F, // MIPI_TX__3 : MIPI_TX11 | MAP_EN_L (MAP_EN_L Pipe 3): 0xF
0x04,0x4E,0x09,0xCC,0x00, // MIPI_TX__3 : MIPI_TX12 | (Default) MAP_EN_H (MAP_EN_H Pipe 3): 0x0
0x04,0x4E,0x09,0xCD,0x2C, // MIPI_TX__3 : MIPI_TX13 | MAP_SRC_0 (MAP_SRC_0 Pipe 3 DT): 0x2C | (Default)
MAP_SRC_0 (MAP_SRC_0 Pipe 3 VC): 0x0
0x04,0x4E,0x09,0xCE,0xEC, // MIPI_TX__3 : MIPI_TX14 | MAP_DST_0 (MAP_DST_0 Pipe 3 DT): 0x2C | MAP_DST_0
(MAP_DST_0 Pipe 3 VC): 0x3
0x04,0x4E,0x09,0xCF,0x00, // MIPI_TX__3 : MIPI_TX15 | (Default) MAP_SRC_1 (MAP_SRC_1 Pipe 3 DT): 0x0 |
(Default) MAP_SRC_1 (MAP_SRC_1 Pipe 3 VC): 0x0
0x04,0x4E,0x09,0xD0,0xC0, // MIPI_TX__3 : MIPI_TX16 | (Default) MAP_DST_1 (MAP_DST_1 Pipe 3 DT): 0x0 |
MAP_DST_1 (MAP_DST_1 Pipe 3 VC): 0x3
0x04,0x4E,0x09,0xD1,0x01, // MIPI_TX__3 : MIPI_TX17 | MAP_SRC_2 (MAP_SRC_2 Pipe 3 DT): 0x1 | (Default)
MAP_SRC_2 (MAP_SRC_2 Pipe 3 VC): 0x0
0x04,0x4E,0x09,0xD2,0xC1, // MIPI_TX__3 : MIPI_TX18 | MAP_DST_2 (MAP_DST_2 Pipe 3 DT): 0x1 | MAP_DST_2
(MAP_DST_2 Pipe 3 VC): 0x3
0x04,0x4E,0x09,0xD3,0x12, // MIPI_TX__3 : MIPI_TX19 | MAP_SRC_3 (MAP_SRC_3 Pipe 3 DT): 0x12 | (Default)
MAP_SRC_3 (MAP_SRC_3 Pipe 3 VC): 0x0
0x04,0x4E,0x09,0xD4,0xD2, // MIPI_TX__3 : MIPI_TX20 | MAP_DST_3 (MAP_DST_3 Pipe 3 DT): 0x12 | MAP_DST_3
(MAP_DST_3 Pipe 3 VC): 0x3
0x04,0x4E,0x09,0xED,0x55, // MIPI_TX__3 : MIPI_TX45 | MAP_DPHY_DEST_0 (MAP_DPHY_DST_0 Pipe 3): 0x1 |
MAP_DPHY_DEST_1 (MAP_DPHY_DST_1 Pipe 3): 0x1 | MAP_DPHY_DEST_2 (MAP_DPHY_DST_2 Pipe 3): 0x1 |
MAP_DPHY_DEST_3 (MAP_DPHY_DST_3 Pipe 3): 0x1
// Double Mode Configuration
0x04,0x4E,0x09,0x73,0x10, // MIPI_TX__1 : MIPI_TX51 | ALT2_MEM_MAP8 (ALT2_MEM_MAP8 CTRL1): Alternate
memory map enabled
// MIPI DPHY Configuration
0x04,0x4E,0x08,0xA0,0x04, // MIPI_PHY : MIPI_PHY0 | (Default) phy_4x2 (Port Configuration): 2 (1x4)
0x04,0x4E,0x09,0x4A,0xD0, // MIPI_TX__1 : MIPI_TX10 | (Default) CSI2_LANE_CNT (Port A - Lane Count): 4
0x04,0x4E,0x08,0xA3,0xE4, // MIPI_PHY : MIPI_PHY3 | (Default) phy0_lane_map (Lane Map - PHY0 D0): Lane 0
| (Default) phy0_lane_map (Lane Map - PHY0 D1): Lane 1 | (Default) phy1_lane_map (Lane Map - PHY1 D0):
Lane 2 | (Default) phy1_lane_map (Lane Map - PHY1 D1): Lane 3
0x04,0x4E,0x08,0xA5,0x00, // MIPI_PHY : MIPI_PHY5 | (Default) phy0_pol_map (Polarity - PHY0 Lane 0):
Normal | (Default) phy0_pol_map (Polarity - PHY0 Lane 1): Normal | (Default) phy1_pol_map (Polarity -
PHY1 Lane 0): Normal | (Default) phy1_pol_map (Polarity - PHY1 Lane 1): Normal | (Default) phy1_pol_map
(Polarity - PHY1 Clock Lane): Normal
0x04,0x4E,0x1D,0x00,0xF4, //  (config_soft_rst_n - PHY1): 0x0
// This is to set predefined (coarse) CSI output frequency
// CSI Phy 1 is 1300 Mbps/lane.
```

```
0x04,0x4E,0x1D,0x00,0xF4, // (Default)
0x04,0x4E,0x04,0x18,0x2D,
0x04,0x4E,0x1D,0x00,0xF5, //  | (Default)  (config_soft_rst_n - PHY1): 0x1
0x04,0x4E,0x08,0xA2,0x34, // MIPI_PHY : MIPI_PHY2 | phy_Stdby_n (phy_Stdby_2): Put PHY2 in standby mode
| phy_Stdby_n (phy_Stdby_3): Put PHY3 in standby mode
0x04,0x4E,0x04,0x0B,0x02, // BACKTOP : BACKTOP12 | CSI_OUT_EN (CSI_OUT_EN): CSI output enabled
// Video Transmit Configuration for Serializer(s)
0x04,0x80,0x00,0x02,0x43, // DEV : REG2 | VID_TX_EN_Z (VID_TX_EN_Z): Enabled
0x04,0x82,0x00,0x02,0x43, // DEV : REG2 | VID_TX_EN_Z (VID_TX_EN_Z): Enabled
0x04,0x84,0x00,0x02,0x43, // DEV : REG2 | VID_TX_EN_Z (VID_TX_EN_Z): Enabled
0x04,0x86,0x00,0x02,0x43, // DEV : REG2 | VID_TX_EN_Z (VID_TX_EN_Z): Enabled
```

Use Case Example 2

This example has the following characteristics:

- Four MAX96717s connected to one MAX96724
- Tunnel mode
- Initial I2C address (0x80) of serializers is changed to 0x80, 0x82, 0x84, and 0x86
- I2C address of deserializer is 0x4E
- Link rate = 6 Gbps
- Each serializer receives RAW12 with VC = 0 and RAW16 VC = 1 on its MIPI input port
- The deserializer reassigns the VCs (from 0 to 0-3) and outputs the data on Port A at 1300 Mbps

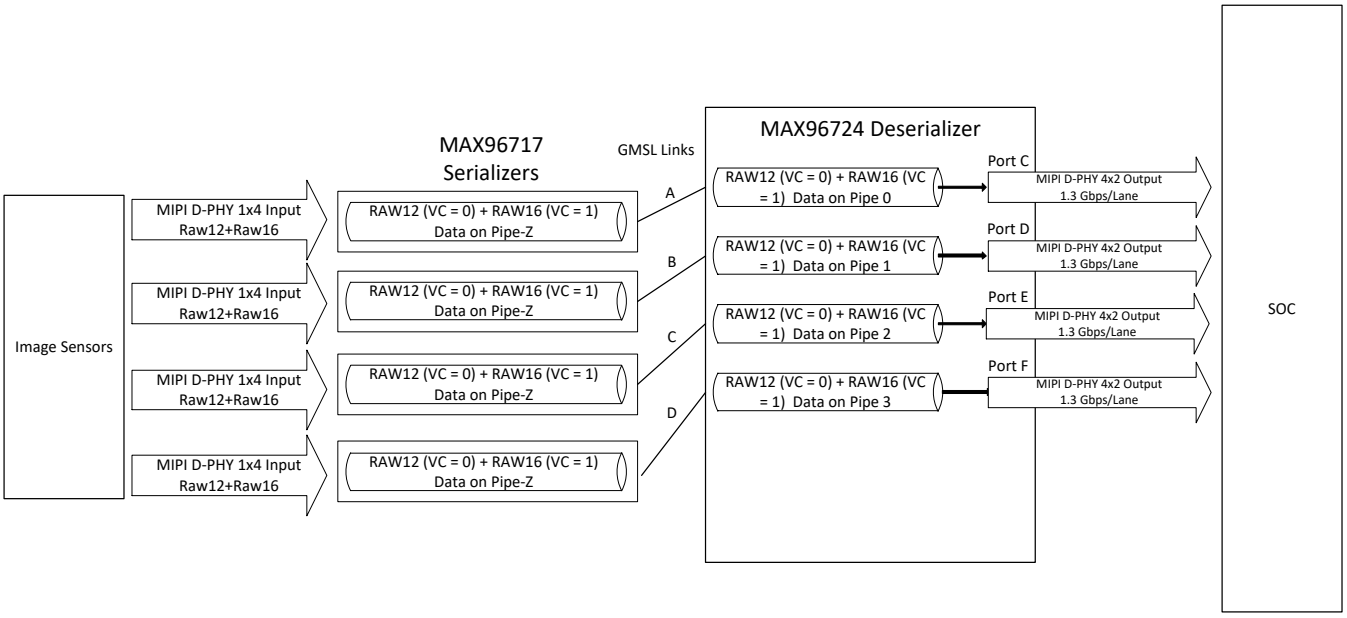


Figure 43. Use Case Example 2

Use Case 2 Example Script

```
// GMSL-A / Serializer: MAX96717 (Tunnel Mode) / Mode: 1x4 / Device Address: 0x80 / Multiple-VC Case:
Frame - Syncronized Line - Interleaved / Multiple-VC Pipe Sharing: Shared Pipe
// PipeZ:
// Input Stream: VC0 RAW12 PortB (D-PHY)
// Input Stream: VC1 RAW16 PortB (D-PHY)

// GMSL-B / Serializer: MAX96717 (Tunnel Mode) / Mode: 1x4 / Device Address: 0x80 / Multiple-VC Case:
Frame - Syncronized Line - Interleaved / Multiple-VC Pipe Sharing: Shared Pipe
```

```
// PipeZ:
// Input Stream: VC0 RAW12 PortB (D-PHY)
// Input Stream: VC1 RAW16 PortB (D-PHY)

// GMSL-C / Serializer: MAX96717 (Tunnel Mode) / Mode: 1x4 / Device Address: 0x80 / Multiple-VC Case:
Frame - Syncronized Line - Interleaved / Multiple-VC Pipe Sharing: Shared Pipe
// PipeZ:
// Input Stream: VC0 RAW12 PortB (D-PHY)
// Input Stream: VC1 RAW16 PortB (D-PHY)

// GMSL-D / Serializer: MAX96717 (Tunnel Mode) / Mode: 1x4 / Device Address: 0x80 / Multiple-VC Case:
Frame - Syncronized Line - Interleaved / Multiple-VC Pipe Sharing: Shared Pipe
// PipeZ:
// Input Stream: VC0 RAW12 PortB (D-PHY)
// Input Stream: VC1 RAW16 PortB (D-PHY)

// Deserializer: MAX96724 / Mode: 4 (1x2) / Device Address: 0x4E
// Pipe0:
// GMSL-A Input Stream: VC0 RAW12 PortB - Output Stream: VC0 RAW12 PortC (D-PHY)
// GMSL-A Input Stream: VC1 RAW16 PortB - Output Stream: VC1 RAW16 PortC (D-PHY)
// Pipe1:
// GMSL-B Input Stream: VC0 RAW12 PortB - Output Stream: VC0 RAW12 PortD (D-PHY)
// GMSL-B Input Stream: VC1 RAW16 PortB - Output Stream: VC1 RAW16 PortD (D-PHY)
// Pipe2:
// GMSL-C Input Stream: VC0 RAW12 PortB - Output Stream: VC0 RAW12 PortE (D-PHY)
// GMSL-C Input Stream: VC1 RAW16 PortB - Output Stream: VC1 RAW16 PortE (D-PHY)
// Pipe3:
// GMSL-D Input Stream: VC0 RAW12 PortB - Output Stream: VC0 RAW12 PortF (D-PHY)
// GMSL-D Input Stream: VC1 RAW16 PortB - Output Stream: VC1 RAW16 PortF (D-PHY)

0x04,0x4E,0x04,0x0B,0x00, // BACKTOP : BACKTOP12 | CSI_OUT_EN (CSI_OUT_EN): CSI output disabled
// Single Link Initialization Before Serializer Device Address Change
0x04,0x4E,0x00,0x06,0xF2, // DEV : REG6 | LINK_EN_A (LINK_EN_A): Disabled | (Default) LINK_EN_B
(LINK_EN_B): Enabled | LINK_EN_C (LINK_EN_C): Disabled | LINK_EN_D (LINK_EN_D): Disabled
0x00,0x78,
// GMSL-B Serializer Address Change from 0x80 to 0x82
0x04,0x80,0x00,0x00,0x82, // DEV : REG0 | DEV_ADDR (DEV_ADDR): 0x41
// Single Link Initialization Before Serializer Device Address Change
0x04,0x4E,0x00,0x06,0xF4, // DEV : REG6 | (Default) LINK_EN_A (LINK_EN_A): Disabled | LINK_EN_B
(LINK_EN_B): Disabled | LINK_EN_C (LINK_EN_C): Enabled | (Default) LINK_EN_D (LINK_EN_D): Disabled
0x00,0x78,
// GMSL-C Serializer Address Change from 0x80 to 0x84
0x04,0x80,0x00,0x00,0x84, // DEV : REG0 | DEV_ADDR (DEV_ADDR): 0x42
// Single Link Initialization Before Serializer Device Address Change
0x04,0x4E,0x00,0x06,0xF8, // DEV : REG6 | (Default) LINK_EN_A (LINK_EN_A): Disabled | (Default)
LINK_EN_B (LINK_EN_B): Disabled | LINK_EN_C (LINK_EN_C): Disabled | LINK_EN_D (LINK_EN_D): Enabled
0x00,0x78,
// GMSL-D Serializer Address Change from 0x80 to 0x86
0x04,0x80,0x00,0x00,0x86, // DEV : REG0 | DEV_ADDR (DEV_ADDR): 0x43
// Link Initialization for Deserializer
0x04,0x4E,0x00,0x06,0xFF, // DEV : REG6 | LINK_EN_A (LINK_EN_A): Enabled | LINK_EN_B (LINK_EN_B):
Enabled | LINK_EN_C (LINK_EN_C): Enabled | (Default) LINK_EN_D (LINK_EN_D): Enabled
0x00,0x78,
// Video Transmit Configuration for Serializer(s)
0x04,0x80,0x00,0x02,0x03, // DEV : REG2 | VID_TX_EN_Z (VID_TX_EN_Z): Disabled
0x04,0x82,0x00,0x02,0x03, // DEV : REG2 | VID_TX_EN_Z (VID_TX_EN_Z): Disabled
0x04,0x84,0x00,0x02,0x03, // DEV : REG2 | VID_TX_EN_Z (VID_TX_EN_Z): Disabled
0x04,0x86,0x00,0x02,0x03, // DEV : REG2 | VID_TX_EN_Z (VID_TX_EN_Z): Disabled
//
// INSTRUCTIONS FOR GMSL-A SERIALIZER MAX96717
//
// MIPI DPHY Configuration
0x04,0x80,0x03,0x30,0x00, // MIPI_RX : MIPI_RX0 | (Default) RSVD (Port Configuration): 1x4
0x04,0x80,0x03,0x83,0x80, // MIPI_RX_EXT : EXT11 | (Default) Tun_Mode (Tunnel Mode): Enabled
0x04,0x80,0x03,0x31,0x10, // MIPI_RX : MIPI_RX1 | ctrl1_num_lanes (Port B - Lane Count): 2
0x04,0x80,0x03,0x32,0xE0, // MIPI_RX : MIPI_RX2 | (Default) phy1_lane_map (Lane Map - PHY1 D0): Lane 2 |
(Default) phy1_lane_map (Lane Map - PHY1 D1): Lane 3
0x04,0x80,0x03,0x33,0x04, // MIPI_RX : MIPI_RX3 | (Default) phy2_lane_map (Lane Map - PHY2 D0): Lane 0 |
(Default) phy2_lane_map (Lane Map - PHY2 D1): Lane 1
0x04,0x80,0x03,0x34,0x00, // MIPI_RX : MIPI_RX4 | (Default) phy1_pol_map (Polarity - PHY1 Lane 0):
Normal | (Default) phy1_pol_map (Polarity - PHY1 Lane 1): Normal
```

```
0x04,0x80,0x03,0x35,0x00, // MIPI_RX : MIPI_RX5 | (Default) phy2_pol_map (Polarity - PHY2 Lane 0):
Normal | (Default) phy2_pol_map (Polarity - PHY2 Lane 1): Normal | (Default) phy2_pol_map (Polarity -
PHY2 Clock Lane): Normal
// Controller to Pipe Mapping Configuration
0x04,0x80,0x03,0x08,0x64, // FRONTTOP : FRONTTOP_0 | (Default) RSVD (CLK_SELZ): Port B | (Default)
START_PORTB (START_PORTB): Enabled
0x04,0x80,0x03,0x11,0x40, // FRONTTOP : FRONTTOP_9 | (Default) START_PORTBZ (START_PORTBZ): Start Video
0x04,0x80,0x03,0x15,0x00, // (Default)  (independent_vs_mode): Disabled
// Pipe Configuration
0x04,0x80,0x00,0x5B,0x00, // CFGV__VIDEO_Z : TX3 | TX_STR_SEL (TX_STR_SEL Pipe Z): 0x0
//
// INSTRUCTIONS FOR GMSL-B SERIALIZER MAX96717
//
// MIPI DPHY Configuration
0x04,0x82,0x03,0x30,0x00, // MIPI_RX : MIPI_RX0 | (Default) RSVD (Port Configuration): 1x4
0x04,0x82,0x03,0x83,0x80, // MIPI_RX_EXT : EXT11 | (Default) Tun_Mode (Tunnel Mode): Enabled
0x04,0x82,0x03,0x31,0x10, // MIPI_RX : MIPI_RX1 | ctrl1_num_lanes (Port B - Lane Count): 2
0x04,0x82,0x03,0x32,0xE0, // MIPI_RX : MIPI_RX2 | (Default) phy1_lane_map (Lane Map - PHY1 D0): Lane 2 |
(Default) phy1_lane_map (Lane Map - PHY1 D1): Lane 3
0x04,0x82,0x03,0x33,0x04, // MIPI_RX : MIPI_RX3 | (Default) phy2_lane_map (Lane Map - PHY2 D0): Lane 0 |
(Default) phy2_lane_map (Lane Map - PHY2 D1): Lane 1
0x04,0x82,0x03,0x34,0x00, // MIPI_RX : MIPI_RX4 | (Default) phy1_pol_map (Polarity - PHY1 Lane 0):
Normal | (Default) phy1_pol_map (Polarity - PHY1 Lane 1): Normal
0x04,0x82,0x03,0x35,0x00, // MIPI_RX : MIPI_RX5 | (Default) phy2_pol_map (Polarity - PHY2 Lane 0):
Normal | (Default) phy2_pol_map (Polarity - PHY2 Lane 1): Normal | (Default) phy2_pol_map (Polarity -
PHY2 Clock Lane): Normal
// Controller to Pipe Mapping Configuration
0x04,0x82,0x03,0x08,0x64, // FRONTTOP : FRONTTOP_0 | (Default) RSVD (CLK_SELZ): Port B | (Default)
START_PORTB (START_PORTB): Enabled
0x04,0x82,0x03,0x11,0x40, // FRONTTOP : FRONTTOP_9 | (Default) START_PORTBZ (START_PORTBZ): Start Video
0x04,0x82,0x03,0x15,0x00, // (Default)  (independent_vs_mode): Disabled
// Pipe Configuration
0x04,0x82,0x00,0x5B,0x00, // CFGV__VIDEO_Z : TX3 | TX_STR_SEL (TX_STR_SEL Pipe Z): 0x0
//
// INSTRUCTIONS FOR GMSL-C SERIALIZER MAX96717
//
// MIPI DPHY Configuration
0x04,0x84,0x03,0x30,0x00, // MIPI_RX : MIPI_RX0 | (Default) RSVD (Port Configuration): 1x4
0x04,0x84,0x03,0x83,0x80, // MIPI_RX_EXT : EXT11 | (Default) Tun_Mode (Tunnel Mode): Enabled
0x04,0x84,0x03,0x31,0x10, // MIPI_RX : MIPI_RX1 | ctrl1_num_lanes (Port B - Lane Count): 2
0x04,0x84,0x03,0x32,0xE0, // MIPI_RX : MIPI_RX2 | (Default) phy1_lane_map (Lane Map - PHY1 D0): Lane 2 |
(Default) phy1_lane_map (Lane Map - PHY1 D1): Lane 3
0x04,0x84,0x03,0x33,0x04, // MIPI_RX : MIPI_RX3 | (Default) phy2_lane_map (Lane Map - PHY2 D0): Lane 0 |
(Default) phy2_lane_map (Lane Map - PHY2 D1): Lane 1
0x04,0x84,0x03,0x34,0x00, // MIPI_RX : MIPI_RX4 | (Default) phy1_pol_map (Polarity - PHY1 Lane 0):
Normal | (Default) phy1_pol_map (Polarity - PHY1 Lane 1): Normal
0x04,0x84,0x03,0x35,0x00, // MIPI_RX : MIPI_RX5 | (Default) phy2_pol_map (Polarity - PHY2 Lane 0):
Normal | (Default) phy2_pol_map (Polarity - PHY2 Lane 1): Normal | (Default) phy2_pol_map (Polarity -
PHY2 Clock Lane): Normal
// Controller to Pipe Mapping Configuration
0x04,0x84,0x03,0x08,0x64, // FRONTTOP : FRONTTOP_0 | (Default) RSVD (CLK_SELZ): Port B | (Default)
START_PORTB (START_PORTB): Enabled
0x04,0x84,0x03,0x11,0x40, // FRONTTOP : FRONTTOP_9 | (Default) START_PORTBZ (START_PORTBZ): Start Video
0x04,0x84,0x03,0x15,0x00, // (Default)  (independent_vs_mode): Disabled
// Pipe Configuration
0x04,0x84,0x00,0x5B,0x00, // CFGV__VIDEO_Z : TX3 | TX_STR_SEL (TX_STR_SEL Pipe Z): 0x0
//
// INSTRUCTIONS FOR GMSL-D SERIALIZER MAX96717
//
// MIPI DPHY Configuration
0x04,0x86,0x03,0x30,0x00, // MIPI_RX : MIPI_RX0 | (Default) RSVD (Port Configuration): 1x4
0x04,0x86,0x03,0x83,0x80, // MIPI_RX_EXT : EXT11 | (Default) Tun_Mode (Tunnel Mode): Enabled
0x04,0x86,0x03,0x31,0x10, // MIPI_RX : MIPI_RX1 | ctrl1_num_lanes (Port B - Lane Count): 2
0x04,0x86,0x03,0x32,0xE0, // MIPI_RX : MIPI_RX2 | (Default) phy1_lane_map (Lane Map - PHY1 D0): Lane 2 |
(Default) phy1_lane_map (Lane Map - PHY1 D1): Lane 3
0x04,0x86,0x03,0x33,0x04, // MIPI_RX : MIPI_RX3 | (Default) phy2_lane_map (Lane Map - PHY2 D0): Lane 0 |
(Default) phy2_lane_map (Lane Map - PHY2 D1): Lane 1
0x04,0x86,0x03,0x34,0x00, // MIPI_RX : MIPI_RX4 | (Default) phy1_pol_map (Polarity - PHY1 Lane 0):
Normal | (Default) phy1_pol_map (Polarity - PHY1 Lane 1): Normal
```

```
0x04,0x86,0x03,0x35,0x00, // MIPI_RX : MIPI_RX5 | (Default) phy2_pol_map (Polarity - PHY2 Lane 0):
Normal | (Default) phy2_pol_map (Polarity - PHY2 Lane 1): Normal | (Default) phy2_pol_map (Polarity -
PHY2 Clock Lane): Normal
// Controller to Pipe Mapping Configuration
0x04,0x86,0x03,0x08,0x64, // FRONTTOP : FRONTTOP_0 | (Default) RSVD (CLK_SELZ): Port B | (Default)
START_PORTB (START_PORTB): Enabled
0x04,0x86,0x03,0x11,0x40, // FRONTTOP : FRONTTOP_9 | (Default) START_PORTBZ (START_PORTBZ): Start Video
0x04,0x86,0x03,0x15,0x00, // (Default)  (independent_vs_mode): Disabled
// Pipe Configuration
0x04,0x86,0x00,0x5B,0x00, // CFGV__VIDEO_Z : TX3 | TX_STR_SEL (TX_STR_SEL Pipe Z): 0x0

// INSTRUCTIONS FOR DESERIALIZER MAX96724

// Video Pipes And Routing Configuration
0x04,0x4E,0x00,0xF0,0x40, // VIDEO_PIPE_SEL : VIDEO_PIPE_SEL_0 | (Default) VIDEO_PIPE_SEL_0 (Pipe 0
GMSL2 PHY): A | VIDEO_PIPE_SEL_0 (Pipe 0 Input Pipe): X | (Default) VIDEO_PIPE_SEL_1 (Pipe 1 GMSL2 PHY):
B | VIDEO_PIPE_SEL_1 (Pipe 1 Input Pipe): X
0x04,0x4E,0x00,0xF1,0xC8, // VIDEO_PIPE_SEL : VIDEO_PIPE_SEL_1 | (Default) VIDEO_PIPE_SEL_2 (Pipe 2
GMSL2 PHY): C | VIDEO_PIPE_SEL_2 (Pipe 2 Input Pipe): X | (Default) VIDEO_PIPE_SEL_3 (Pipe 3 GMSL2 PHY):
D | VIDEO_PIPE_SEL_3 (Pipe 3 Input Pipe): X
0x04,0x4E,0x00,0xF4,0x0F, // VIDEO_PIPE_SEL : VIDEO_PIPE_EN | (Default) VIDEO_PIPE_EN (Video Pipe 0):
Enabled | (Default) VIDEO_PIPE_EN (Video Pipe 1): Enabled | (Default) VIDEO_PIPE_EN (Video Pipe 2):
Enabled | (Default) VIDEO_PIPE_EN (Video Pipe 3): Enabled | STREAM_SEL_ALL (Stream Select All): Disabled
// Double Mode Configuration
// Frame-Start/End Override Configuration
// MIPI DPHY Configuration
0x04,0x4E,0x08,0xA0,0x01, // MIPI_PHY : MIPI_PHY0 | phy_4x2 (Port Configuration): 4 (1x2)
0x04,0x4E,0x09,0x36,0x09, // MIPI_TX__0 : MIPI_TX54 | TUN_EN (Port C Tunnel Mode): Enabled
0x04,0x4E,0x09,0x0A,0x50, // MIPI_TX__0 : MIPI_TX10 | CSI2_LANE_CNT (Port C - Lane Count): 2
0x04,0x4E,0x08,0xA3,0xE4, // MIPI_PHY : MIPI_PHY3 | (Default) phy0_lane_map (Lane Map - PHY0 D0): Lane 0
| (Default) phy0_lane_map (Lane Map - PHY0 D1): Lane 1
0x04,0x4E,0x08,0xA5,0x00, // MIPI_PHY : MIPI_PHY5 | (Default) phy0_pol_map (Polarity - PHY0 Lane 0):
Normal | (Default) phy0_pol_map (Polarity - PHY0 Lane 1): Normal | (Default) phy0_pol_map (Polarity -
PHY0 Clock Lane): Normal
0x04,0x4E,0x1C,0x00,0xF4, //  (config_soft_rst_n - PHY0): 0x0
// This is to set predefined (coarse) CSI output frequency
// CSI Phy 0 is 1300 Mbps/lane.
0x04,0x4E,0x1C,0x00,0xF4, // (Default)
0x04,0x4E,0x04,0x15,0x2D,
0x04,0x4E,0x1C,0x00,0xF5, //  | (Default)  (config_soft_rst_n - PHY0): 0x1
0x04,0x4E,0x09,0x76,0x09, // MIPI_TX__1 : MIPI_TX54 | TUN_EN (Port D Tunnel Mode): Enabled
0x04,0x4E,0x09,0x4A,0x50, // MIPI_TX__1 : MIPI_TX10 | CSI2_LANE_CNT (Port D - Lane Count): 2
0x04,0x4E,0x08,0xA3,0x44, // MIPI_PHY : MIPI_PHY3 | phy1_lane_map (Lane Map - PHY1 D0): Lane 0 |
phy1_lane_map (Lane Map - PHY1 D1): Lane 1
0x04,0x4E,0x08,0xA5,0x00, // MIPI_PHY : MIPI_PHY5 | (Default) phy1_pol_map (Polarity - PHY1 Lane 0):
Normal | (Default) phy1_pol_map (Polarity - PHY1 Lane 1): Normal | (Default) phy1_pol_map (Polarity -
PHY1 Clock Lane): Normal
0x04,0x4E,0x1D,0x00,0xF4, //  (config_soft_rst_n - PHY1): 0x0
// This is to set predefined (coarse) CSI output frequency
// CSI Phy 1 is 1300 Mbps/lane.
0x04,0x4E,0x1D,0x00,0xF4, // (Default)
0x04,0x4E,0x04,0x18,0x2D,
0x04,0x4E,0x1D,0x00,0xF5, //  | (Default)  (config_soft_rst_n - PHY1): 0x1
0x04,0x4E,0x09,0xB6,0x09, // MIPI_TX__2 : MIPI_TX54 | TUN_EN (Port E Tunnel Mode): Enabled
0x04,0x4E,0x09,0x8A,0x50, // MIPI_TX__2 : MIPI_TX10 | CSI2_LANE_CNT (Port E - Lane Count): 2
0x04,0x4E,0x08,0xA4,0xE4, // MIPI_PHY : MIPI_PHY4 | (Default) phy2_lane_map (Lane Map - PHY2 D0): Lane 0
| (Default) phy2_lane_map (Lane Map - PHY2 D1): Lane 1
0x04,0x4E,0x08,0xA6,0x00, // MIPI_PHY : MIPI_PHY6 | (Default) phy2_pol_map (Polarity - PHY2 Lane 0):
Normal | (Default) phy2_pol_map (Polarity - PHY2 Lane 1): Normal | (Default) phy2_pol_map (Polarity -
PHY2 Clock Lane): Normal
0x04,0x4E,0x1E,0x00,0xF4, //  (config_soft_rst_n - PHY2): 0x0
// This is to set predefined (coarse) CSI output frequency
// CSI Phy 2 is 1300 Mbps/lane.
0x04,0x4E,0x1E,0x00,0xF4, // (Default)
0x04,0x4E,0x04,0x1B,0x2D,
0x04,0x4E,0x1E,0x00,0xF5, //  | (Default)  (config_soft_rst_n - PHY2): 0x1
0x04,0x4E,0x09,0xF6,0x09, // MIPI_TX__3 : MIPI_TX54 | TUN_EN (Port F Tunnel Mode): Enabled
0x04,0x4E,0x09,0xCA,0x50, // MIPI_TX__3 : MIPI_TX10 | CSI2_LANE_CNT (Port F - Lane Count): 2
0x04,0x4E,0x08,0xA4,0x44, // MIPI_PHY : MIPI_PHY4 | phy3_lane_map (Lane Map - PHY3 D0): Lane 0 |
phy3_lane_map (Lane Map - PHY3 D1): Lane 1
```

```
0x04,0x4E,0x08,0xA6,0x00, // MIPI_PHY : MIPI_PHY6 | (Default) phy3_pol_map (Polarity - PHY3 Lane 0):
Normal | (Default) phy3_pol_map (Polarity - PHY3 Lane 1): Normal | (Default) phy3_pol_map (Polarity -
PHY3 Clock Lane): Normal
0x04,0x4E,0x1F,0x00,0xF4, //  (config_soft_rst_n - PHY3): 0x0
// This is to set predefined (coarse) CSI output frequency
// CSI Phy 3 is 1300 Mbps/lane.
0x04,0x4E,0x1F,0x00,0xF4, // (Default)
0x04,0x4E,0x04,0x1E,0x2D,
0x04,0x4E,0x1F,0x00,0xF5, //  | (Default)  (config_soft_rst_n - PHY3): 0x1
// Tunnel Mode Configuration
0x04,0x4E,0x08,0xCA,0xE4, // MIPI_PHY : MIPI_CTRL_SEL | (Default) MIPI_CTRL_SEL_0 (MIPI Controller Pipe
0): 0x0 | (Default) MIPI_CTRL_SEL_1 (MIPI Controller Pipe 1): 0x1 | (Default) MIPI_CTRL_SEL_2 (MIPI
Controller Pipe 2): 0x2 | (Default) MIPI_CTRL_SEL_3 (MIPI Controller Pipe 3): 0x3
0x04,0x4E,0x09,0x39,0x00, // MIPI_TX__0 : MIPI_TX57 | TUN_DEST (Tunneling Destination Pipe 0): 0x0
0x04,0x4E,0x09,0x79,0x10, // MIPI_TX__1 : MIPI_TX57 | (Default) TUN_DEST (Tunneling Destination Pipe 1):
0x1
0x04,0x4E,0x09,0xB9,0x20, // MIPI_TX__2 : MIPI_TX57 | TUN_DEST (Tunneling Destination Pipe 2): 0x2
0x04,0x4E,0x09,0xF9,0x30, // MIPI_TX__3 : MIPI_TX57 | TUN_DEST (Tunneling Destination Pipe 3): 0x3
0x04,0x4E,0x04,0x0B,0x02, // BACKTOP : BACKTOP12 | CSI_OUT_EN (CSI_OUT_EN): CSI output enabled
// Video Transmit Configuration for Serializer(s)
0x04,0x80,0x00,0x02,0x43, // DEV : REG2 | VID_TX_EN_Z (VID_TX_EN_Z): Enabled
0x04,0x82,0x00,0x02,0x43, // DEV : REG2 | VID_TX_EN_Z (VID_TX_EN_Z): Enabled
0x04,0x84,0x00,0x02,0x43, // DEV : REG2 | VID_TX_EN_Z (VID_TX_EN_Z): Enabled
0x04,0x86,0x00,0x02,0x43, // DEV : REG2 | VID_TX_EN_Z (VID_TX_EN_Z): Enabled
```

Appendix

Figures

Tables

Revision History

Revision Number	Revision Date	Description/Changes	Pages Changed
0	05/23	Initial Release	-