



GMSL2 General User Guide

Rev 1; 12/23

User Guide Contents

GMSL2 General User Guide	1
0. User Guide Contents	2
Serial Link Features	4
1. GMSL2 Link Basics	5
2. Spread-Spectrum Clocking.....	17
3. Power Manager and Sleep Mode	19
4. Clocks and Frequency Reference	25
5. PRBS Testing	30
Video Transmission	33
6. Video Basics	34
7. Video Pipes	41
8. Dual-View	46
9. Forward Error Correction.....	57
10. Watermarking	62
11. Video Timing Generator (VTG) and Video Pattern Generator (VPG).....	70
12. Video Crossbar	83
13. Color Lookup Table (LUT).....	90
14. Frame Sync	93
15. Video Sync Pulse Outputs.....	102
16. Audio	106
Bidirectional Channels	125
17. I ² C/UART	126
18. General-Purpose Input and Output (GPIO).....	155
19. Serial Peripheral Interface	164
Bandwidth Calculations	176
20. GMSL2 Link System Bandwidth	177
Functional Safety	189
21. GMSL2 Error Reporting (ERRB Pin)	190
22. CRC Error Detection and ARQ Error Correction	207
23. Voltage Monitoring.....	214
24. Line Fault.....	218
25. Error Generator	221
Device Families	223
26. HDMI Serializer	225

27. MIPI D-PHY Deskew236

28. Dual oLDI.....243

Revision History255

Serial Link Features

0. GMSL2 Link Basics

0.1 GMSL2 Overview

This section provides a brief introduction to the proprietary physical interface (PHY) layer and link protocol used by GMSL2 serial links. Refer to the [GMSL2 Hardware Design Guide](#) for a more detailed discussion of the PHY and the implementation of the [GMSL2 Channel Specification](#).

The GMSL2 serial links use packet-based, full duplex 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 simultaneously. The programmable forward and reverse channel link rates are fixed and independent of the video pixel clock (PCLK). Typical GMSL2 devices have a forward serial bit rate of 3Gbps or 6Gbps and reverse channel serial bit rate of 187.5Mbps.

Note: *Transmit side* refers to the device in the serial link transmitting to the *receive side*. These arrangements are dependent upon application.

0.2 GMSL2 Link Configurations

The GMSL2 architecture can support the following link configurations.

0.2.1 Single-Link Mode

In single-link mode, a single serializer PHY connects to a single deserializer PHY ([Figure 1](#)). The available bandwidth is equal to the forward and reverse channel link rates that have been selected. In parts that only have a single GMSL2 PHY, this is the only link mode available.

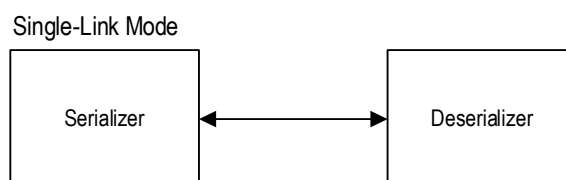


Figure 1. GMSL2 Single-Link Mode

0.2.2 Reverse Splitter Mode

In reverse splitter mode, two serializers connect to a single deserializer through two GMSL2 PHYs ([Figure 2](#)). The available bandwidth into the deserializer is equal to the sum of the forward channel link rates that have been selected in each serializer, allowing up to 12Gbps of bandwidth to be aggregated into the deserializer.

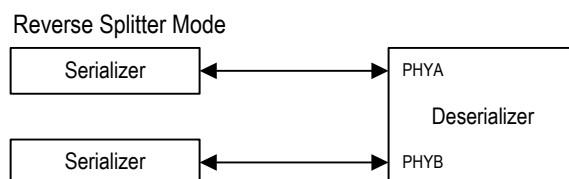


Figure 2. Reverse Splitter Mode

0.2.2.1 Coaxial Cables

Coaxial cables can be used for dual-link operation ([Figure 3](#)). The standard GMSL2 link requirements described in the GMSL2 Channel Specification must be adhered to when using coaxial cables in this application—with the addition of a cable-to-cable skew limit. When two coaxial cables are used for dual-link operation they must be relatively close in length to ensure that PHY-to-PHY skew does not exceed 10ns. This means there should be no more than 2m difference in length.

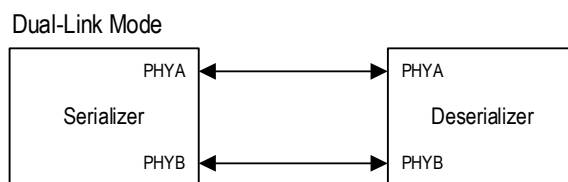


Figure 3. Dual-Link Mode

0.3 GMSL2 Link Rate and Configuration Programming

The link rate is configured with the `TX_RATE[1:0]` and `RX_RATE[1:0]` registers in each device. See [Table 1](#).

0.3.1 Programming Link Rate

Table 1. GMSL2 Link Rate Configuration

LINK RATE	SERIALIZER REGISTERS	DESERIALIZER REGISTERS
Forward Channel Link Rate		
6Gbps	<code>TX_RATE[1:0] = 10</code>	<code>RX_RATE[1:0] = 10</code>
3Gbps	<code>TX_RATE[1:0] = 01</code>	<code>RX_RATE[1:0] = 01</code>
Reverse Channel Link Rate		
187Mbps	<code>RX_RATE[1:0] = 00</code>	<code>TX_RATE[1:0] = 00</code>

0.3.2 Programming Link Configuration

Link configuration is controlled with the `AUTO_LINK` and `LINK_CFG[1:0]` registers. After link configuration settings are changed, reset the link for the changes to take effect. See the [Resets](#) section for more information.

0.3.2.1 Auto Link Mode

By default, the automatic link configuration mode is enabled (`AUTO_LINK = 1`). This means the device attempts to lock in single-link mode. It automatically detects if PHYA or PHYB is connected to the remote device and enables that PHY. If both PHYA and PHYB are connected to remote devices, it connects in single-link mode to whichever device is first detected.

Note: In auto link mode, the `LINK_CFG[1:0]` setting is ignored.

0.3.2.2 Manual Link Mode

Manual link mode should be used to manually select which PHY is used in single-link mode and to configure splitter, reverse splitter, or dual-link modes. To enable manual link mode, set `AUTO_LINK` = 0. See [Table 2](#) for configuration details of each mode. Manual link mode can impact the time to achieve GMSL link lock. For typical single-link applications, `AUTO_LINK` = 1 is recommended to simplify system level design.

Table 2. Manual Link Mode Configuration

LINK MODE	SERIALIZER SETTINGS	DESERIALIZER SETTINGS
Single-Link Mode (PHYA)	<code>AUTO_LINK</code> = 0 <code>LINK_CFG[1:0]</code> = 01	<code>AUTO_LINK</code> = 0 <code>LINK_CFG[1:0]</code> = 01
Single-Link Mode (PHYB)	<code>AUTO_LINK</code> = 0 <code>LINK_CFG[1:0]</code> = 10	<code>AUTO_LINK</code> = 0 <code>LINK_CFG[1:0]</code> = 10
Reverse Splitter Mode	In both serializers, either use auto link mode or manual mode and configure to single-link mode for the connected PHY (auto link mode recommended)	<code>AUTO_LINK</code> = 0 <code>LINK_CFG[1:0]</code> = 11

Note: Link rate and configuration changes require a link reset to take effect. Write `RESET_ONESHOT` to 1 in the serializer after making changes.

0.3.2.3 Programming Example

```
# Set both serializer and deserializer to 6G/187M rate
0x80,0x0001,0x48
0x90,0x0001,0x02
# Set both parts to dual-link mode
0x80,0x0010,0x00
0x90,0x0010,0x00
# Write one-shot reset in serializer
0x80,0x0010,0x20
```

0.3.2.4 Standard Splitter Mode: Switch from Splitter Mode to Single-Link Mode

An individual PHY in the serializer is programmed to single-link mode. Enable single-link mode by writing register `0x0010` with one of the following values:

- PHY A: `0x21`
- PHY B: `0x22`

Note: Example above is for serializer with two outputs.

This write switches the link to single-link mode and performs a “Oneshot Reset.” The following process takes <50ms to complete:

1. Reset the link into single-link mode.
2. Self-clear the reset bit.
3. Recalibrate the canceller.

4. Auto-adapt the equalizer.
5. Link up to the deserializer.
6. Pull the LOCK pin high.

Alternatively, the serializer can be programmed to “Auto Link Mode”. In auto link mode, the serializer automatically detects which PHY is connected to the deserializer and enables that PHY. Program auto link mode by writing the serializer register `0x0010` to `0x30`. The following process is automatically performed:

- Autodetect which of the two PHYs is connected to a deserializer.
- Reset the link to single-link mode to the detected deserializer device.
- Self-clear the reset bit.
- Recalibrate the canceller.
- Auto-adapt the equalizer.
- Link up to the deserializer.
- Pull the LOCK pin high.

0.3.2.5 Standard Splitter Mode: Switch from Single-Link Mode to Splitter Mode

If both deserializers connected to the serializers are powered-up again, the serializer remains linked to only one deserializer in single-link mode until splitter mode is re-enabled. To re-enable splitter mode, write serializer register `0x0010` to `0x23`. The following re-enabling process takes <50ms to complete:

- Reset the link to splitter mode to both deserializers.
- Self-clear the reset bit.
- Recalibrate the canceller.
- Auto-adapt the equalizer.
- Link up to both deserializers.
- Pull the LOCK pin high on all three devices.

0.4 Cables Supported

GMSL2 devices support both 50Ω coaxial and 100Ω differential pair cables (example, STP, SPP, STQ) that meet the GMSL2 Channel Specification. Refer to the [GMSL2 Hardware Design Guide](#) for more details on supported cables and connectors.

0.5 Device Power-Up

The GMSL2 device is in power-down mode when either the PWDNB pin is low or any power supply is below its respective power-on reset (POR) threshold. The device begins the power-up sequence after the PWDNB pin is driven high and all power supplies are above their respective POR thresholds. GMSL2 devices do not require power supply sequencing; however, it is recommended to enable all the power supplies followed by the PWDNB pin to precisely control when the device powers on. The initial power-up sequence applies to all GMSL2 devices; the [Link Start-Up Procedure](#) is different for devices operating in [GMSL2 Mode](#) and those operating in backward compatible GMSL1 Mode. The applicable procedures are indicated in the following steps.

0.5.1 Initial Power-Up Sequence

The following sequence is performed automatically by the device after the PWDNB pin is driven high and all power supplies are above their respective POR thresholds.

Note: There are differences depending on whether the device is in GMSL1 mode.

- The oscillator is powered up:
 - GMSL2 mode: The crystal oscillator is powered up.
- Internal termination resistance calibration using the 402Ω external resistor connected to the XRES pin is performed.
- Configuration (CFG) pins levels are sampled:
 - Two-level configuration pins: Pin levels are latched. Internal registers are set according to the latched pin levels.
 - Multi-level CFG pins: The CFG pin levels are sampled and latched. Internal registers are set according to the latched pin levels.
- GMSL2 Link Start-Up:
 - GMSL2 mode: Continue with the [GMSL2 Mode](#) Link Start-Up Procedure.

0.5.2 Link Start-Up Procedure

0.5.2.1 GMSL2 Mode

Following the [Initial Power-Up Sequence](#) section, the local-side control channel (I²C or UART) is functional, and the device registers are writable and readable. The power-up time to this state is given in each device's data sheet.

- GMSL2 devices (that is, serializer and deserializer) can power up in any order. After power-up, each serial link device sends a beacon signal out to available PHYs to recognize other devices present on the channel.
- A handshake procedure is initiated after successful recognition with the beacon signal.
- Transmitter (Tx) canceller calibration is automatically performed on serializer(s) and deserializer(s) for the forward and reverse channels.
- Equalizer autocalibration is performed on serializer(s) and deserializer(s) to optimize equalizer coefficients and maximize both horizontal and vertical eye openings for the forward and reverse channels. See the [Benefits of Adaptive Equalization](#) section for more details.
- Video and control channels are enabled. The LOCK pin and LOCK status register go high on both serializer and deserializer (that is, link lock is established).

Note: The GMSL2 [Link Start-Up Procedure](#) completes for any channel that meets [GMSL2 Channel Specification](#) in the maximum time provided in the device-specific data sheets (see the [t_{LOCK} parameter](#) section in the data sheet). LOCK time may vary due to channel, crystal stability, temperature, and manufacturing tolerance(s).

0.5.3 GMSL2 Link Lock

GMSL2 link lock is an automatic bidirectional lock that occurs when both a properly connected serializer and deserializer are both powered up (following the procedures in the [Initial Power-Up Sequence](#) section). In this state, the forward and reverse channels LOCK pins mirror each other (LOCK pins go high in all devices). The detection of sync words on the serial link determines the link lock.

In GMSL2 mode, the serial link uses the 25MHz crystal or external reference clock (see the [Clocks](#) section) as the clock source. A valid video input (pixel clock) is not necessary to establish GMSL2 link lock.

0.5.3.2 LOCK Pin

Both serializers and deserializers have an open-drain LOCK output pin. The LOCK pin indicates that the phase lock loops (PLLs) for the GMSL2 serial link are locked and that the GMSL2 data receive path (that is, forward channel in serializer, reverse channel in deserializer) has locked to the correct word boundary alignment. Control channels (that is, I²C/UART, SPI, and GPIOs) can be used immediately after LOCK is asserted.

Note: The LOCK pin or register MUST be monitored for successful GMSL lock status. Placing a hard-coded wait threshold is NOT recommended. The LOCK status should be obtained prior to continuing with the remaining system programming or actions. If a system level timeout is desired to ensure there is not a system issue, a timeout can be placed on the system to reset the devices after 1s.

0.5.3.3 Losing Link Lock

Link lock is lost (that is, LOCK pins go low) if the serial link cable is unplugged or there is some other interruption to the system connectivity. In single-link mode, the link lock is automatically re-established (following the [Link Start-Up Procedure](#)) when the serial link cable is reconnected (that is, “hot plugged”). Video and control channels are also automatically re-established.

Note: If the LOCK pin or bit is being monitored for such an interruption, allow 100ms of recovery time for a clean system relock. Only issue a reset to the link or device if it does not recover after this delay.

Similarly, the link is lost if either the serializer or deserializer is powered down. For single-link mode, the link lock is automatically re-established when the device is powered on again; however, video and control channel settings need to be reprogrammed. Follow the appropriate guidance for optimal GMSL link lock time based on the system.

0.5.4 Video Lock

Video lock indicates that the deserializer is receiving valid video data from the serializer. After the GMSL2 link has locked, the deserializer video lock sequence begins. Optionally, the LOCK pin behavior in the deserializer can be changed by a register setting (LOCK_CFG) so that the LOCK pin is asserted only when the deserializer is receiving video (asserted with VIDEO_LOCK).

0.5.5 Benefits of Adaptive Equalization

Adaptive equalization (AEQ) addresses critical aspects of serial link implementation and ensures highest possible link performance. Manual equalization is not recommended since it fails to dynamically respond to factors that affect channel quality.

1. AEQ automatically adapts to changes in channel characteristics. Optimal equalization requires dynamic response to variations in IC process(es), package(s), temperature, cable insertion and return loss, PCB layout, and power over coax (PoC) performance.
2. Equalization is nonlinear; nonoptimized equalizer settings have a nonlinear effect on eye opening and link margin. AEQ corrects for link margin to ensure optimum link quality.
3. Changes to equalization alters the signal-to-noise ratio on the bidirectional GMSL2 serial link nonlinearly across the frequency band. Fixed equalization results in a loss of correlation between link margin and channel quality.
4. Manual programming of the canceller registers is inexact. If the canceller is not optimized, signal may be misinterpreted as a noise source. The reduction of the signal-to-noise ratio would negatively impact link performance and link bit error ratio (BER).

0.5.6 Periodic Adaptation

The GMSL2 PHY for both forward and reverse channels periodically re-adapt the equalization to optimize receive-side gain and equalization settings for different environmental and use-case conditions. This allows the equalizer to compensate for changes in the channel due to temperature, aging cables, PoC loads, cable bending, and/or other external factors.

Channel characteristics are dynamic and environmentally dependent. The periodic adaptation (which default runs at a 1Hz rate) allows for the equalizer to continually track and adapt to these channel changes. This is illustrated in the eye diagrams ([Figure 4](#) and [Figure 5](#)).

0.5.6.1 Periodic Adaptation Performance Impact

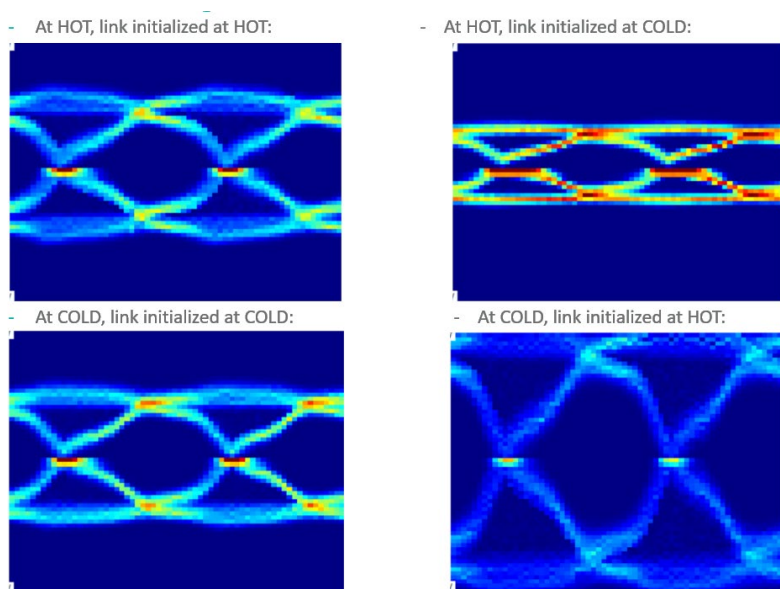


Figure 4. Periodic Adapt Not Enabled

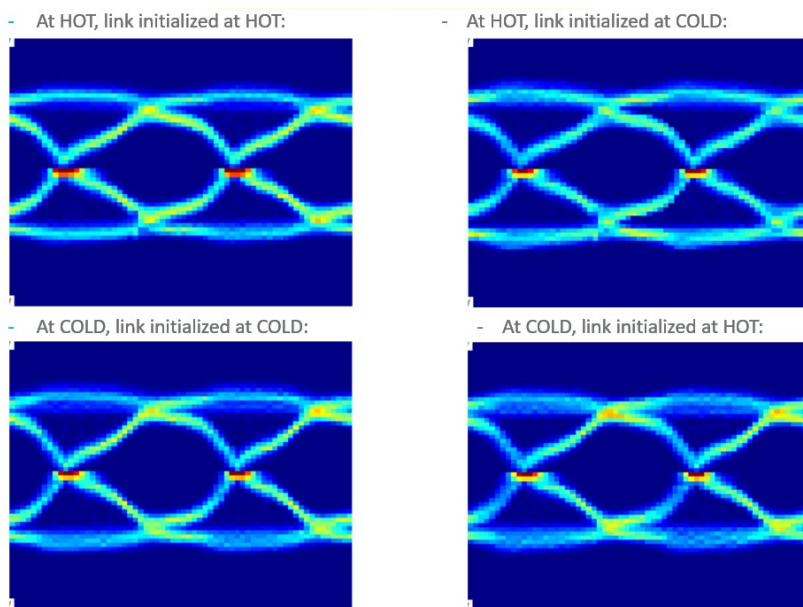


Figure 5. Periodic Adapt Enabled

With periodic adaptation enabled, the eye is continually optimized as the temperature changes. Without periodic adaptation enabled, variations in cable temperature from the initial link condition affect the eye-opening. A nonoptimized eye-opening leads to lower link BER.

0.6 Resets

Registers that affect GMSL2 PHY operation should be programmed when the link is held in reset.

A link reset must be used after programming:

- TX_RATE
- RX_RATE
- CXTP_A/B
- AUTO_LINK
- LINK_CFG
- GMSL2 mode

Note:

- 1) Resets in the above list may not be available in all parts.
- 2) There are other situations when a link reset may be needed which are described in their relevant sections.

0.6.1 Types of Resets

There are three different reset bits available for GMSL2 devices. These are as follows:

0.6.1.1 Reset All

Reset All is a full reset that resets the link, all registers, and all digital and analog blocks. Writing `RESET_ALL` = 1 (cleared when written) or toggling the PWDNB pin low then high performs a Reset All.

0.6.1.2 Oneshot Reset

The Oneshot Reset is a self-clearing bit that resets the link without resetting registers. This can only be performed by writing `RESET_ONESHOT = 1`. Use this reset after making a change that affects the GMSL2 serial link.

Note: `RESET_LINK`, not Oneshot Reset, is the preferred method of resetting the link for proper timing control, unless otherwise noted.

0.6.1.3 Reset Link

Reset Link is similar to Oneshot Reset, but it is not self-clearing. This bit should be used if the link is held in a reset state to suspend connect attempts to the remote side. After this bit is set, all local registers are still available, but the remote device is not accessible. This bit should not be set on the remote device: setting Reset Link on the remote device prevents write access on the link and the reset is unable to be cleared. Enable by writing `RESET_LINK = 1`; release the reset by writing `RESET_LINK = 0`.

0.7 GMSL2 Link Protocol

The GMSL2 is a programmable fixed rate packet-based protocol, flexibly sharing link bandwidth between video data and multiple bidirectional communications channels. Dynamic bandwidth allocation allows active channels to share full link bandwidth; inactive channels do not consume any link bandwidth. Additionally, maximum packet size is limited to prevent a single channel from overutilizing link bandwidth.

0.7.1.1 Encoding

The GMSL2 serial link uses a proprietary 9B/10B encoding method. This encoding method features the following:

- Less encoding overhead (11%) compared to traditional 8B/10B encoding (25%).
- DC-balance: The number of 1s and 0s in any given window is bounded to +/- 9.
- A maximum run length (consecutive number of 0s or 1s) limited to 7.
- Eight special symbols used for link synchronization, framing (packet start/stop indication), and control purposes. The Hamming distance between special symbols is at least 3, so these are robust to errors (that is, the special symbols cannot accidentally transform into another special symbol with 1- or 2-bit errors in the 10-bit symbol).
- Special sync words for word boundary locking contain an exclusive comma sequence that cannot exist in regular random encoded data stream starting from any bit position.

0.7.1.2 Scrambling

The GMSL2 serial link features **synchronous scrambling to improve clock and data recovery robustness**. This also improves electromagnetic interference (EMI) performance by reducing any pattern-dependent EMI emission. The scrambler in the transmitter and the descrambler in the receiver are synchronized with sync words.

0.7.1.3 Special Symbols

Special symbols (individually and in combination) are used as packet delimiters and headers. These are selected to keep GMSL2 link robust (that is, tolerant to bit errors) as much as possible. All special

9B/10B symbols have 3-bit Hamming distance between each other; it is impossible for a special symbol to transform into another special symbol with 1-bit or 2-bit errors.

0.7.1.4 Total Link Bandwidth

Total link bandwidth used by all different communication channels cannot exceed the fixed available link bandwidth. In typical use cases, available link bandwidth exceeds the bandwidth requirement. Idle packets are used to fill unused link bandwidth.

0.7.2 Packet Protocol

The GMSL2 serial links use a packet-based protocol comprising various packet types to transmit information across the link. The packet types are summarized in the following sections.

Note: The GMSL2 forward and reverse channels use the same protocol.

0.7.2.1 Video Packets

Video packets are used to transmit up to four concurrent video streams over the GMSL2 serial link. Each video packet consists of 36 pixels and a 1-bit sequence number that allows the receiver to detect dropped packets.

During video blanking time, pixel data values are not transmitted to save bandwidth. By default, Horizontal Sync (HS), Vertical Sync (VS), and Data Enable (DE) values are transmitted from the serializer so that the receiver can exactly reconstruct the same blanking time that the serializer receives on the video input. This is needed when driving a deserializer with open LVDS display interface (OLDI) output.

Optionally, the serializer can be programmed to “remove the heartbeat” (`LIM_HEART` = 1). In this mode, the serializer does not transmit HVD packets to minimize bandwidth usage. This mode can be used when driving a deserializer with CSI-2 or eDP/DP output, as those deserializers do not require HVD values. In this case, `SEQ_MISS_EN` = 0 and `DIS_PKT_DET` = 1 should be programmed in the deserializer so that it does not expect HVD packets during video blanking.

By default, video packets do not have embedded Cyclic Redundancy Check (CRC). When bandwidth is available, this can be enabled to attach a 16-bit CRC value to each video packet. This is checked by deserializer device and reported.

0.7.2.2 Control Channel Packets

Control packets are used for communications channels that use less than 100Mbps of bandwidth. This includes I²C, UART, SPI, GPIO, audio, and internal info frame packets. Info frame packets are internal control packets used to transmit information between serializer and deserializer (example, video PCLK PLL settings, audio PLL settings).

Control packet types are differentiated by header formats. Each control packet is tagged with a 4-bit sequence number and includes a 16-bit CRC value by default. These are used for error detection and correction. See the [CRC Error Detection and ARQ Error Correction](#) section for additional information.

0.7.2.3 Sync Words

Sync words are periodically transmitted packets used for word boundary locking, lock validation, and scrambler synchronization. Sync words are also used to align two GMSL2 lanes when the serial link is configured in dual-link mode.

0.7.2.4 Idle Packets

Idle packets are transmitted to fill the link bandwidth when there is no data to transmit. Both the header and payload data of idle packets are scrambled to minimize EMI. Idle packets are decoded and checked for errors, which are reported to the `IDLE_ERR_FLAG` register. See the [GMSL Idle Packet Errors](#) section for additional information.

0.7.3 Priority-Based Packet Scheduler

GMSL2 devices support multiple communication channels in addition to the forward channel video. Available bandwidth is flexibly shared between the various channels with dynamic bandwidth allocation.

GMSL2 devices use a bandwidth allocation scheduler that acts as an arbiter if multiple channels (that is, video or control channels) attempt to simultaneously send packets over the link. The scheduler uses a channel-based priority setting and a calculated running average of the bandwidth usage of each type of communication channel to schedule transmissions over the link and throttle channels if needed.

Link bandwidth sharing is based on predefined share ratios and the recent bandwidth usage of each channel. GMSL2 devices track averages of recent bandwidth usage for each channel compared to assigned bandwidth share ratios. When deciding which packet to transmit, the scheduler selects the channel with lowest usage ratio from the pending requests.

1. Spread-Spectrum Clocking

Spread-Spectrum Clocking (SSC) provides enhanced mitigation of EMI emitted from devices and interconnections. GMSL2 serial links offer exceptional EMI performance; however, it is recommended to use SSC when possible, to reduce emission peaks for additional margin. This option is a feature of all GMSL2 devices and is also available in GMSL1 mode.

Spread spectrum is available for the following interfaces:

- Forward serial link (GMSL2 mode)
- Reverse serial link
- Serializer camera clock reference output

Configuration is similar for each of these interfaces. Programming through the GMSL2 registers is described in the following sections.

1.1 SSC Operation

The SSC uses a constant frequency 6GHz Clock Multiplier Unit (CMU) clock and a phase interpolator. A digital block generates the phase commands to modulate the transmitter clock. Linear frequency modulation requires quadratic phase command generation. The digital logic supports five levels of SSC generation all at 25kHz. An example center frequency spread is illustrated in [Figure 6](#).

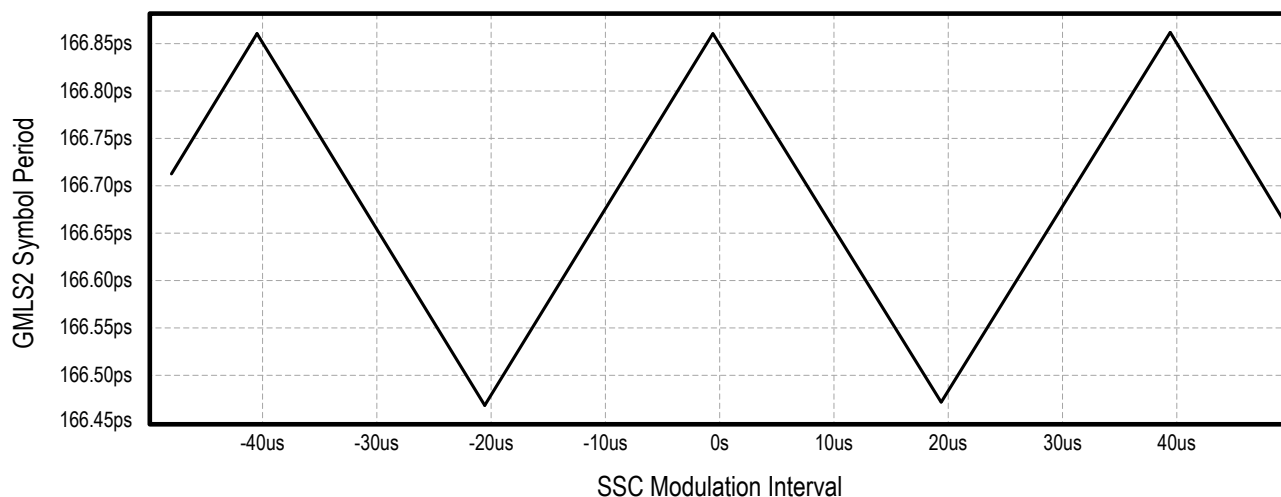


Figure 6. Example SSC Center Frequency Spread

1.3 SSC Configuration

Enabling spread spectrum for a GMSL2 serial link requires configuring registers controlling the phase command generation and maintenance of the 25kHz triangle waveform. The forward and reverse channel spread spectrum generator must be disabled before configuring its settings and then re-enabled once the settings are configured.

The SSC can be individually enabled and configured for various interfaces, allowing for user flexibility in their target application. See the register settings in [Table 3](#) to configure the spread spectrum transmitter on either side of the link. For GMSL2 devices, writing these values to the serializer registers implements spread spectrum on the forward serial channel while writing these values to the deserializer registers implements spread spectrum on the reverse serial channel.

The maximum spread for the GMSL2 serial link is 2530ppm.

1.3.1 Forward and Reverse Channel SSC

Care must be taken to prevent large phase transients on the transmit clock. The procedure to enable SSC on the GMSL2 forward/reverse serial link is as follows:

1. Disable SSC generator: [RLMS71\[0\]](#) = 0.
2. Configure SSC generator per the table below.
3. Enable SSC generator: [RLMS71\[0\]](#) = 1.

Table 3. GMSL2 Tx SSC Configuration for Fixed Ppm

SSC Ppm	SSC F_c (kHz)	RLMS 64	RLMS 70	RLMS 71	RLMS 72	RLMS 73	RLMS 74	RLMS 75	RLMS 76	RLMS 77
268	25	0x03	0x07	0x02	0xC9	0x02	0xF9	0x01	0x00	0x00
580	25	0x03	0x06	0x02	0xAB	0x00	0x63	0x07	0x00	0x00
970	25	0x03	0x03	0x02	0xAB	0x00	0x63	0x07	0x00	0x00
1750	25	0x03	0x01	0x02	0xF9	0x00	0x2C	0x05	0x00	0x00
2530	25	0x03	0x01	0x02	0xAB	0x00	0x63	0x07	0x00	0x00

After configuration is complete, enable the generator by writing “1” to [RLMS71\[0\]](#).

Note: Disabling the GMSL2 serial link spread spectrum may cause loss-of-lock. It is recommended to perform a link reset if link spread spectrum is disabled. See the [Resets](#) section for additional information.

2. Power Manager and Sleep Mode

2.1 Overview

GMSL2 devices include an integrated power manager that ensures the reliable and efficient operation of various power functions. The power manager controls the internal switched supply domains during the full sequence of power states so that the device powers up and down smoothly. During power-up, the power manager guards the device until the internal supplies are validated and the digital core assumes normal operations. In all power modes, the power manager monitors power supplies for undervoltage and overvoltage conditions. In sleep mode, the power manager minimizes current consumption and can quickly restore device configurations after waking up.

2.2 Operation

All GMSL2 devices include three common power supplies (V_{DD} , V_{DD18} , and V_{DDIO}), with select devices integrating an optional internal V_{DD} regulator. Some devices additionally include interface specific power supplies. All devices include sleep and power manager functions. Refer to device-specific data sheets for more information.

The power manager block is designed to minimize 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. Various device families have different undervoltage and overvoltage monitoring capabilities and associated indicators. Refer to the voltage monitoring section for additional details.

2.2.1 Power Supplies

Most GMSL2 devices share a common set of power supply voltages that power universal functions such as the digital core, GMSL link circuitry, and GPIO. These power supplies are summarized below. Additional power supplies associated with specific interfaces such as HDMI, CSI, and DP, etc., may be found in specific devices supporting these interfaces.

- **V_{DD}** – LDO input (1.2V) or power switch input (1.0V). This supply is named V_{REG} (1.8V) on the oLDI deserializers.
- **V_{DD18}** – Internally sensed 1.8V analog supply.
- **V_{DDIO}** – Internally sensed 3.3V - 1.8V I/O supply.
- **V_{DD_sw} (CAP_ V_{DD})** – Internally switched 1V supply that powers the digital core logic. This may connect through an internal switch, directly to a 1V power supply pin, or to the output of an internal V_{DD} regulator depending on device architecture. Note that this supply is not available on all parts.

2.2.2 Power Manager States

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

Figure 7 shows the state diagram for the power manager.

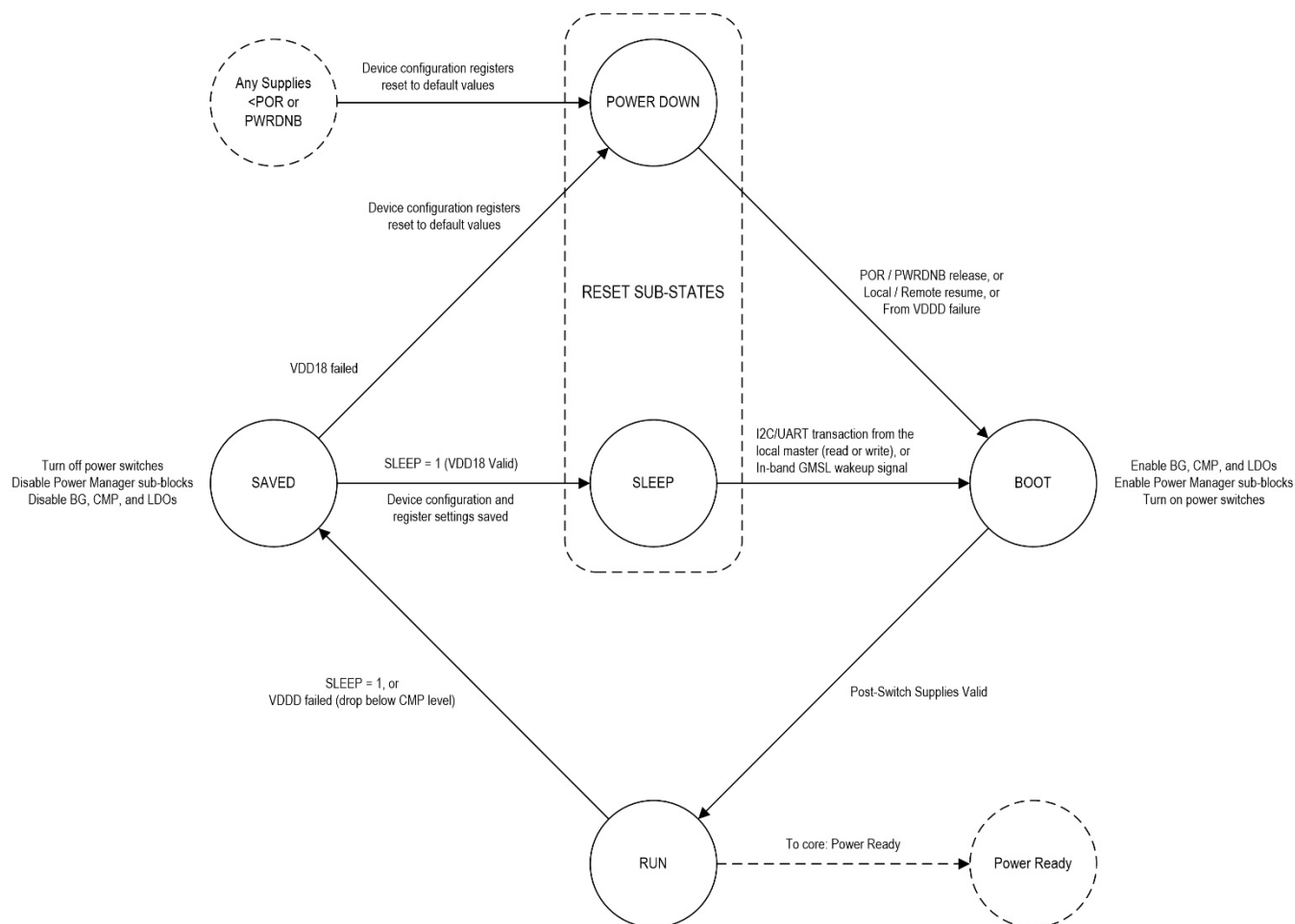


Figure 7. Power Manager State Diagram

2.2.2.1 RESET (POWER DOWN/SLEEP)

POWER DOWN and **SLEEP** are two sub-states of the **RESET** state.

The device enters the power down state if the PWRDNB pin is asserted (logic low), V_{DD_sw} falls below the internally set threshold (that is, power failure), or 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 one of the V_{DD18} PORs is tripped. Deasserting PWRDNB (logic 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 run, the system purposefully initiates the sleep state with an I²C/UART command (`SLEEP = 1`). Sleep is entered automatically after the retention memory is loaded following `SLEEP = 1` from the saved state when power switches are turned off. In the sleep state, the V_{DD18} supply must be continuously maintained to ensure that previous configurations and settings are preserved. It is recommended that all other supplies be maintained during sleep mode to simplify the sleep and wakeup sequences.

2.2.2.2 BOOT

The device can enter the boot state from reset after external supplies ramped up or the device 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.

Note: The power manager has an inrush current-control feature. In boot state, the core supply switches are turned on gradually.

2.2.2.3 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.

2.2.2.4 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.

2.3 Configuration

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.

2.3.1 Sleep Mode

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

There are two ways to wake up from sleep mode:

- **Local Wakeup:**
Local wakeup entails the local host processor initiating a dummy control channel transaction, which briefly wakes up the device. In I²C mode, the initial temporary wakeup requires four falling edges on SDA. Depending on the device address used, there may need to be multiple consecutive dummy transactions issued by the host to achieve the required four SDA falling edges. The dummy transaction(s) must immediately be followed by the host processor setting **SLEEP** = 0 to permanently exit sleep mode. The device automatically returns to sleep mode if **SLEEP** remains set to 1.
- **Remote Wakeup:**
All GMSL devices include wakeup detectors to observe GMSL link activity and briefly turn on the device when activity is detected. The link can then lock, providing an opportunity for a remote host to disable sleep mode (setting **SLEEP** = 0). The device automatically reverts to sleep mode if **SLEEP** remains set to 1.

In most cases, the GMSL link wakeup detectors associated with the host side of the link can be disabled because the local side GMSL device is only expected to wake up based on a request from the local host. The remote side wakeup detectors must not be disabled in most cases. A remote device with wakeup detectors disabled cannot wake up from sleep mode without a hard power cycle. Once a wakeup command is received, the power manager detects the command and transitions the device to run mode. After the digital core restores all saved device configuration and register values, the device enters run mode and normal operation is permitted to resume. The device automatically returns to sleep mode after a brief time unless the **SLEEP** bit is changed from 1 to 0 through local or remote register access.

If devices at both ends of a GMSL link are sleeping, the host processor must initiate the sequence by waking the local device first and waiting for link lock. The host then immediately disables 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.

2.3.1.1 Sleep and Wakeup Sequences

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).

Wake up 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`).

Enable SLEEP mode (local device):

- Write `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 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 lin.k

2.3.2 Sleep Mode Limitations

2.3.2.1 Sleep Mode Should Not be Used in Conjunction with RESET_ALL

When a device enters sleep mode, the content of key configuration registers is saved in a low power data retention memory. When the device exits sleep mode and resumes normal operation, the content of the retention memory is automatically loaded into the appropriate registers, enabling seamless wake up without any repeated configuration from the host.

The GMSL2 family includes a global soft reset function called `RESET_ALL`. This is a self-clearing reset command intended to reset all sub-systems to their default configurations. However, if a device has previously gone through a sleep/wake cycle, issuing a `RESET_ALL` resets the device and erroneously loads the contents of the retention memory that had been stored when the most recent `SLEEP` command was executed. As a result, the device “resets” to the state configured prior to entering sleep mode previously rather than recovering in a clean power-up default state. The most severe implication of this is that the `SLEEP = 1` state is saved in the retention memory. So, the device recovers from reset and immediately enters sleep mode.

Due to the above-described behavior, `RESET_ALL` and `SLEEP` should never be used together.

2.3.2.2 Not All Registers are Saved in Retention Memory

One major limitation of sleep mode is that not all registers are saved in retention memory. Only registers saved to the retention memory are automatically loaded with their pre-sleep values after resume from sleep. Most key registers corresponding to common device configuration are saved in retention memory. However, applications requiring extensive low-level configuration or infrequently used features may require writing to registers not saved in retention after resume from sleep. In these cases, full recovery from sleep mode to the pre-sleep device state requires some repeated register configuration following resume from sleep. The routing of register data to the retention memory is device dependent.

2.3.3 Registers

`DIS_LOCAL_WAKE` – By default, local μ C can wake the device through the primary control channel. Set bit high to disable the feature.

WAKE_EN_B/A – Enable wake-up by remote device connected to Link A or Link B. Set bit low to disable the feature.

SLEEP – Enable sleep mode by setting this bit to 1. Disable sleep mode by setting to 0.

Diagnostic indicators for various overvoltage and undervoltage conditions are detailed in the voltage monitoring section.

2.4 Debug Techniques

If the device is exhibiting unexpected behavior during the power-up sequence, poll status indicators for the power manager as detailed in the voltage monitoring section and probe supply pins to confirm that voltages are within nominal ranges (see device-specific data sheet). If status flags indicate normal operation and voltage readings are within range, troubleshoot other functional blocks of the device.

When trying to pinpoint power supply problems, make sure to observe the transient behavior of the supplies to capture any glitches or droop conditions.

Refer to the [Voltage Monitoring](#) section for further details of the available power supply monitoring diagnostic indicators.

In a GMSL system where one device is in sleep mode with wakeup detectors enabled and the other device is in active run mode, the link lock appears to oscillate between locked and unlocked states. This is because the sleeping device periodically wakes up when its wakeup detectors observe link activity. The power manager in this device then allows the link to briefly lock to provide the remote host with an opportunity to disable sleep mode. If sleep mode is not disabled, the power manager reverts the device to sleep mode. This process repeats periodically and creates a situation where the link transitions back and forth quickly between locked and unlocked states.

2.5 Feature Availability by Device

All devices support basic power manager functions and sleep mode. Refer to device specific data sheets for details of which supplies are available for a specific device. Refer to the [Voltage Monitoring](#) section of this document for details of power supply diagnostic function availability by device family.

3. Clocks and Frequency Reference

3.1 Overview

GMSL2 devices require a precise, low-jitter external frequency reference. This frequency reference is multiplied and used to establish the GMSL link clock, other internal clocks, and, in some cases, clocks associated with external video or peripheral interfaces.

Integrated oscillators are included in all GMSL2 devices. Most frequency reference applications require only the addition of an external crystal and associated passive components. Alternatively, the on-chip oscillator can be overdriven by an external oscillator or clock source. This enables multiple devices to share a common frequency reference and potentially reduces the number of crystals required in a complex system.

3.2 Architecture

The frequency reference for GMSL2 devices is typically realized using an external 25MHz crystal in conjunction with the on-chip oscillator to create an accurate, low-jitter reference. This reference is distributed throughout the GMSL2 device to establish the link and generate other required clocks. An external 25MHz clock source, such as another GMSL2 device, can alternatively be used in the place of the crystal. When using an external oscillator or clock source, the frequency precision and jitter must be within the bounds specified in the electrical characteristics of the device.

A simplified diagram of the frequency reference system is shown in [Figure 8](#). The diagram shows both on-chip and external circuit elements.

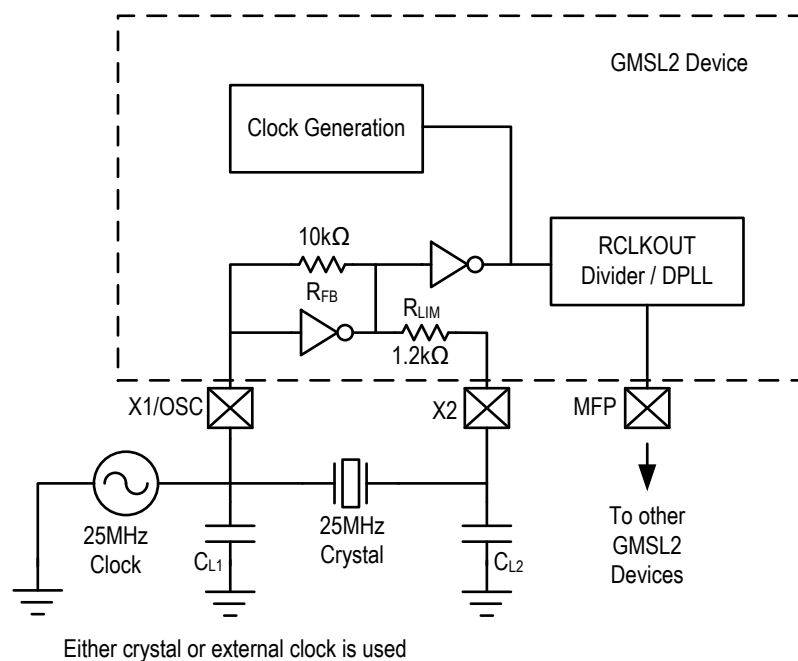


Figure 8. Simplified Diagram of Frequency Reference System

3.3 Operation

Depending on the architecture of a given system, either a crystal or external precision clock source must be used to provide a frequency reference to the GMSL2 device. The following information details some of the key considerations involved in successfully implementing either a crystal- or external clock-based frequency reference. Note that no device programming is required to accommodate one operating mode versus the other.

3.3.1 Crystal Mode

When deriving the frequency reference using a crystal, a device with the required frequency and tolerance (that is, 25MHz \pm 200ppm) as well as environment characteristics must be selected to ensure compatibility and reliable performance.

The implementation of the crystal circuit is illustrated in [Figure 8](#). The values of C_{L1} and C_{L2} must be selected based on the specified load capacitance of the crystal and the estimated parasitic capacitances of the PCB and GMSL2 device. An example of the typical parasitic capacitances of the GMSL2 X1/OSC and X2 pins is given in [Table 4](#). Refer to device-specific data sheets for the capacitance of these pins, as they vary by device. Note that the feedback resistor and limit resistor are included on-chip. The selected crystal is connected between the X1/OSC and X2 pins.

Table 4. Example DC Characteristics of X1/OSC and X2 Pins

REFERENCE CLOCK INPUT REQUIREMENTS (CRYSTAL) (X1/OSC, X2)		
PARAMETER	SYMBOL	VALUE (TYP)
X1/OSC Input Capacitance	C_{IN_X1}	3pF
X2 Input Capacitance	C_{IN_X2}	1pF
Internal X2 Limit Resistor	R_{LIM}	1.2k Ω
Internal Feedback Resistor	R_{FB}	10k Ω
Transconductance	g_M	28mA/V

Proper layout of the crystal circuitry is important to achieving the best possible performance. [Figure 9](#) provides an example layout of the crystal and associated components. The crystal and associated external load capacitors should be placed near the X1/OSC and X2 pins of the GMSL2 device to reduce interconnect capacitance and susceptibility to noise coupling. If preferred, the crystal can be placed on the opposite side of the board from the GMSL2 device. There should be a solid ground plane in the region of both the crystal and GMSL2 device, and the crystal's ground terminals should be connected to the ground plane with minimal inductance. Finally, noisy components and interconnects should not be arranged near the crystal or associated interconnects. Length matching and impedance of the X1/OSC and X2 interconnects is not critical.

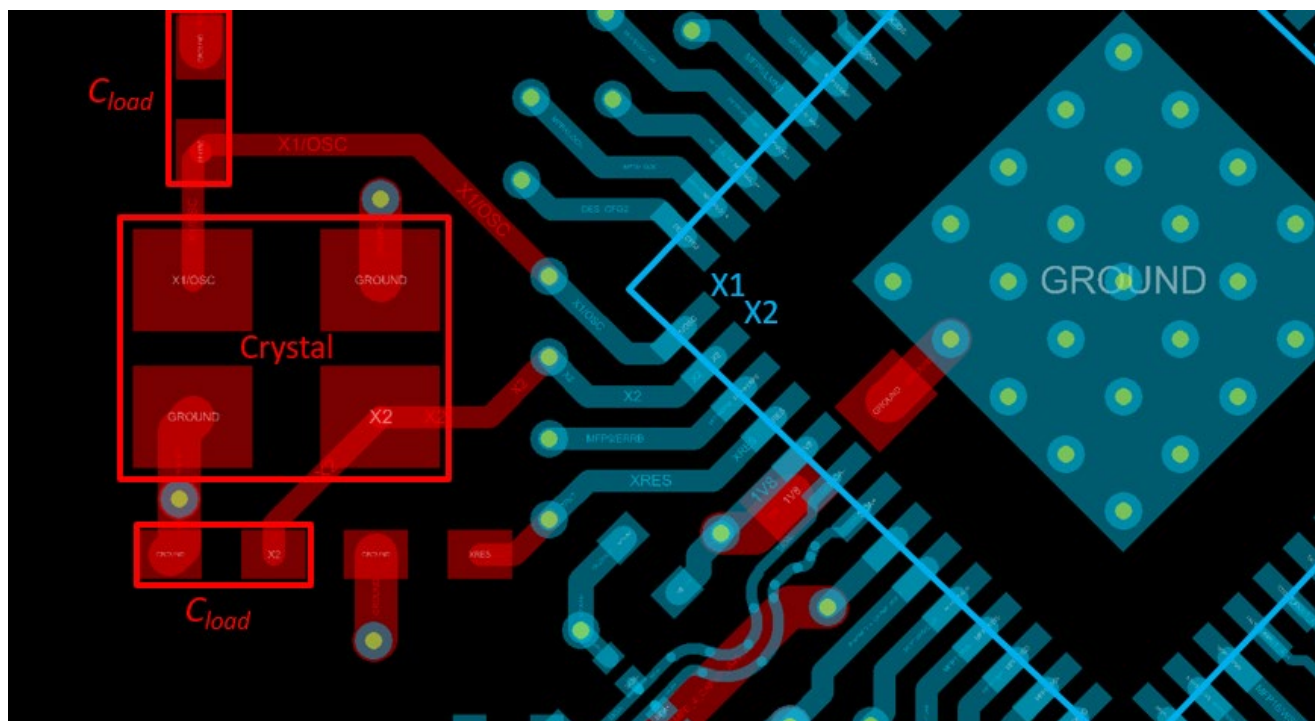


Figure 9. Example Crystal Layout

3.3.2 External Reference Mode

The external load capacitors and crystal can be omitted if the frequency reference is supplied by an external clock source such as a TCXO or the RCLKOUT function of a GMSL2 device. The external clock source must be compliant with the specifications detailed in [Table 5](#) and [Table 6](#). Note that the maximum signal level is constrained by V_{DDIO} . Assuming that the clock source uses a CMOS driver and is powered by the same V_{DDIO} as the device that is being driven, the clock signal can be DC coupled to the X1/OSC pin of the GMSL2 device. For example, a GMSL2 device's RCLKOUT function can be directly connected to another GMSL2 device's X1/OSC pin assuming that the two devices use a common V_{DDIO} . The required frequency accuracy and tolerance are identical to the crystal specifications (that is, 25MHz \pm 200ppm). Note that the jitter, duty cycle, and fall time must all be considered when using an external clock source.

Table 5. External Clock DC Characteristics

REFERENCE CLOCK REQUIREMENTS (EXTERNAL INPUT ON X1/OSC, X2 UNCONNECTED)			
PARAMETER	MIN	TYP	MAX
High-Level Input Voltage	0.9V		V_{DDIO}
Low-Level Input Voltage			0.4V
Input Impedance		10k Ω	
X1/OSC Input Capacitance		3pF	

Table 6. External Clock AC Characteristics

REFERENCE CLOCK REQUIREMENTS (EXTERNAL CLOCK INPUT ON X1/OSC, X2 UNCONNECTED)			
PARAMETER	MIN	TYP	MAX
Frequency		25MHz	
Frequency Stability + Frequency Tolerance			±200ppm
Input Jitter		600ps (p-p)	
Input Duty Cycle	40%		60%
Input Fall Time			4ns

Table 7 includes an example of the typical RCLKOUT characteristics of a GMSL2 device. The jitter and RCLKOUT fall time are within the specified bounds, confirming that this device's RCLKOUT meets the requirements to be used as the frequency reference source for another GMSL2 device. Note that in the case of GMSL2 devices, the drive strength of the RCLKOUT MFP pin must be specified appropriately to achieve the desired fall time depending on V_{DDIO} . A similar set of specifications should be reviewed while evaluating any external clock source.

Table 7. Example RCLKOUT Characteristics

REFERENCE CLOCK REQUIREMENTS (EXTERNAL CLOCK INPUT ON X1/OSC, X2 UNCONNECTED)				
PARAMETER	CONDITIONS	MIN	TYP	MAX
Frequency	Crystal or reference clock input		25MHz	
Rise Time	20% to 80%, $C_L = 10\text{pF}$			4ns
Fall Time	80% to 20%, $C_L = 10\text{pF}$			4ns
Jitter	$C_L = 10\text{pF}$, rising or falling edge $f_{\text{REFOUT}} = 12.5\text{MHz}$		210ps (p-p)	

In contrast to crystal mode, the frequency reference is not guaranteed to automatically be available at power-up in external reference mode. For example, if the RCLKOUT function of another GMSL2 device is used to provide the frequency reference, that device's RCLKOUT function must be enabled prior to the frequency reference being operational.

The layout of a design that uses an external frequency reference must be carefully considered. The clock should be routed so that it is not susceptible to picking up noise or corrupting other noise-sensitive functions. The interconnect should be as short as possible, and a solid ground plane should be used in the vicinity of the clock source and along the length of the interconnect. A provision should be included for a source termination resistor.

3.3.3 Frequency Reference Debugging

Proper operation of the frequency reference is required for GMSL2 link functionality. If a system is not achieving the anticipated operation and power supplies have been checked, verify that the frequency reference is operational.

Test the frequency reference by probing the X2 pin with a low-capacitance, high-impedance probe. If the frequency reference is functioning correctly, X2 should toggle at 25MHz. If not, inspect the associated solder joints, clean the PCB, and consider replacing the crystal and load capacitors. Finally, confirm that the equivalent series resistance (ESR) of the crystal is within the bounds supported by the GMSL2 device and verify that the load capacitance values employed are consistent with the requirements of the crystal.

3.4 Applications and Examples

3.4.1 BER Testing Using the GMSL2 Idle Link

The GMSL2 serial link itself can also be used as a pseudo-PRBS test. The packet-based transmission protocol of the GMSL2 link sends data at a fixed rate while it is active. If there is no video, control, or other data to send, the link transmits 'idle packets.' These idle packets are filled with random data and undergo the same serial link protocol encoding, scrambling, etc. processes of other data types. Analyzing the number of errors per unit of time while sending at a fixed rate provides a BER, which can be used to determine the quality of the link.

See [Table 8](#) for an example of the classification of the BER with respect to time without an observed error.

Table 8. Observation Time Required for 95% Confidence Interval of Different BER (@ 6Gbps)

BER	TIME INTERVAL BETWEEN ERRORS
10⁻⁶	~500 microseconds
10⁻⁹	~500 milliseconds
10⁻¹²	~8.3 minutes
10⁻¹⁵	~5.8 days
10⁻¹⁸	~16 years

4. PRBS Testing

4.1 Overview

GMSL2 devices are equipped with internal **pseudo-random binary sequence (PRBS)** tests. PRBS tests are a type of built-in self-test (BIST) that enable the device to test and debug a system. Further, GMSL2 systems can use PRBS testing to determine the bit error rate (BER) of video/audio packets and of the GMSL link itself.

The forward channel PRBS uses a pattern generator in the serializer and a corresponding pattern checker in the deserializer. The serializer populates video or audio packets with PRBS data, which is then checked by the deserializer. Some devices also support reverse channel audio PRBS. Here, the deserializer generates audio PRBS data to be decoded and checked by the serializer.

4.2 Operation

GMSL2 devices may offer up to three PRBS tests to evaluate supported channels. Refer to [Feature Availability by Device Family](#).

- Forward Channel Video PRBS
- Forward Channel Audio PRBS
- Reverse Channel Audio PRBS

For GMSL2 camera parts that do not support audio, only the forward video PRBS test is available.

Note: GMSL2 devices have an error generator to verify the PRBS is operational and recording errors properly. To verify the PRBS test, enable the ERRG during the PRBS test. Ensure that the PRBS checker is identifying expected errors. Refer to the [Error Generator \(ERRG\)](#) section for additional information.

The video and audio PRBS generator/checker blocks in GMSL2 serializers and deserializers are illustrated in [Figure 10](#) and [Figure 11](#), respectively.

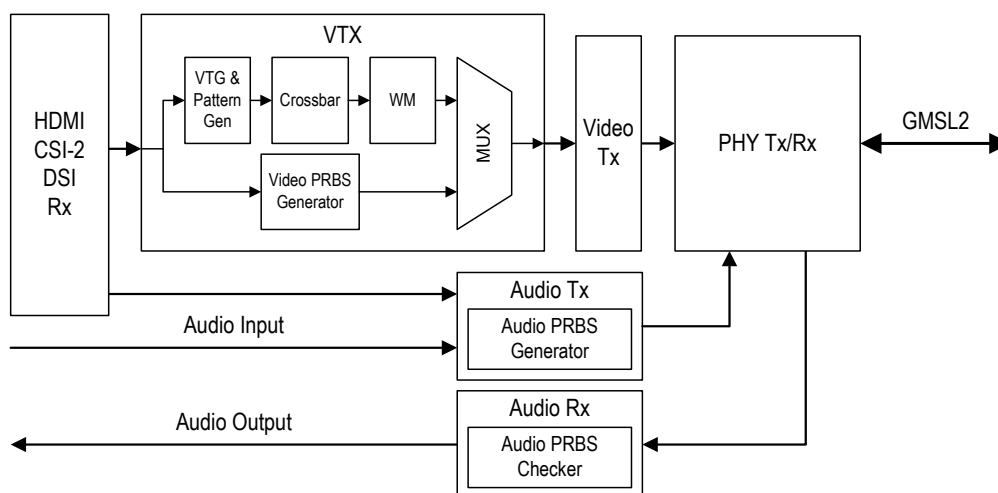


Figure 10. GMSL2 Serializer Video and Audio PRBS Generator/Checker Blocks

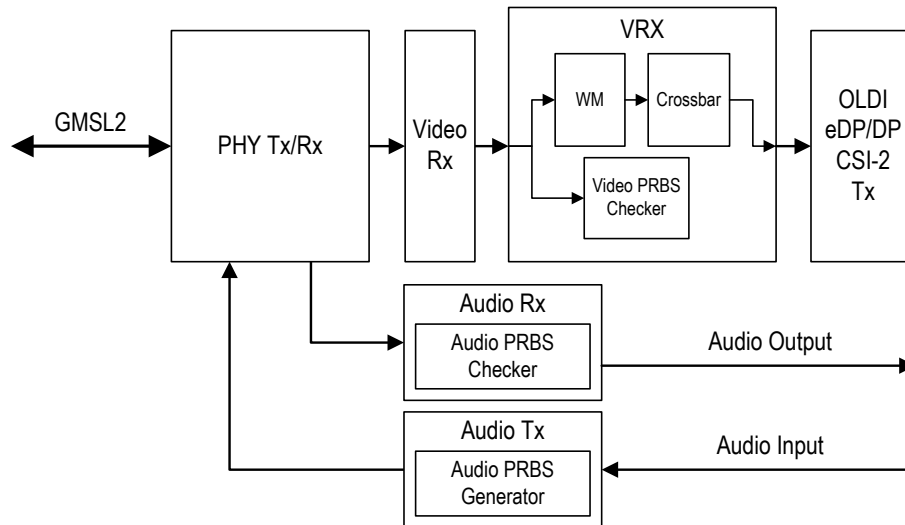


Figure 11. GMSL2 Deserializer Video and Audio PRBS Generator/Checker Blocks

4.2.1 Forward Channel Video PRBS

The forward channel video PRBS evaluates the video channels by testing the BER of video packets. This test is limited to the video channel(s), and the serial link is not affected. To enable the video PRBS test, the serializer must receive a valid clock from the video interface (that is, HDMI)

The **PRBS_TYPE** register determines which GMSL1 PRBS test is run. GMSL1 devices may have limited PRBS testing capabilities; refer to device data sheets for information regarding available tests. The **PRBS_TYPE** setting must be matched on both devices before enabling PRBS.

The “legacy” PRBS test (**PRBS_TYPE** = 0) is similar to the GMSL2 video PRBS, while the **PRBS_TYPE** = 1 tests the serial link.

Note: Do not use **PRBS_TYPE** = 0 with **DBL** = 1 or **PRBS_TYPE** = 1 with **HIBW** = 1.

The procedure to enable GMSL1 PRBS tests is similar to the procedure for GMSL2 devices.

- Test procedure for **PRBS_TYPE** = 0
 - Set **PRBSEN** = 1 in the serializer first and then in the deserializer. Communication with the remote side is possible in this mode. Errors are generated in the serializer through the error generator and read in the deserializer through the **PRBS_ERR** registers. The **PRBS_ERR** register is cleared upon read. **PRBS_OK** = 0 in this mode.
- Test procedure for **PRBS_TYPE** = 1
 - Set **PRBSEN** = 1 in the deserializer first and then in the serializer. Communication through the serial link is not possible for the duration of the PRBS test. The length of the PRBS test pattern is set with the **PRBS_LEN** register. Set the PRBS length and enable the error generator in the serializer before enabling PRBS. After the test finishes, read **PRBS_OK** = 1 and the **PRBS_ERR** register in the deserializer.

4.2.2 BER Testing using the GMSL2 Idle Link

The GMSL2 serial link itself can also be used as a pseudo-PRBS test. The packet-based transmission protocol of the GMSL2 link sends data at a fixed rate while it is active. If there is no video, control, or other data to send, the link transmits 'idle packets.' These idle packets are filled with random data and undergo the same serial link protocol encoding, scrambling, etc., processes of other data types. Analyzing the number of errors per unit of time while sending at a fixed rate provides a bit error ratio (BER), which can be used to determine the quality of the link.

See [Table 9](#) for an example of the classification of the BER with respect to time without an observed error.

Table 9. Observation Time Required for 95% Confidence Interval of Different BER (at 6Gbps)

BER	Time Interval Between Errors
10^{-6}	~500 microseconds
10^{-9}	~500 milliseconds
10^{-12}	~8.3 minutes
10^{-15}	~5.8 days
10^{-18}	~16 years

4.3 Feature Availability by Device Family

All GMSL2 devices support video PRBS. Select GMSL2 devices support audio PRBS. See [Table 10](#) for feature availability by device family.

Table 10. PRBS Support by Device Family

Device Family	Video PRBS	Audio PRBS
HDMI Serializers	X	X
CSI-2 Serializers	X	
Advanced CSI-2 Serializers	X	
CSI-2 Camera Deserializers	X	
oLDI Deserializers	X	X

Video Transmission

5. Video Basics

GMSL devices transmit digital video. Digital video transmissions consist of a series of specifically formatted video frames. The architecture of these frames, which borrows heavily from analog video, must be understood to ensure proper performance of the GMSL device(s) and the system as a whole. This section defines and reviews the core concepts of digital video, the terminology used throughout the GMSL2 User Guide, and the basic equations used to calculate video bandwidth.

5.1 Video Frame Architecture

The different regions of a video image are identified in [Figure 12](#). Note that there are variations in how these parameters are defined across the industry and that their usage may differ from analog video. The definitions are used throughout this document and are applicable to all GMSL products. When designing systems, be aware that concepts and parameters presented here may not align with other industry definitions.

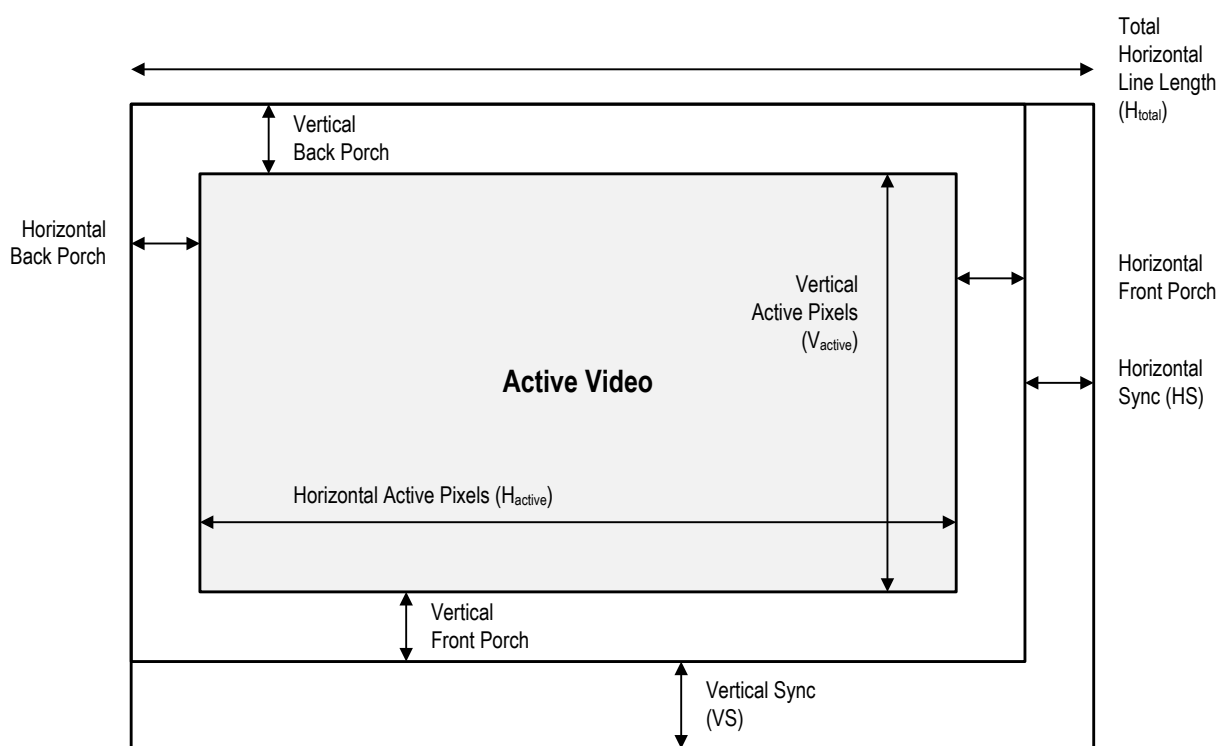


Figure 12. Video Frame Regions

The diagram includes both the active video and blanking periods. The active video frame consists of horizontal lines scanned from left to right. The first pixel transmission begins in the top left corner of the frame. After the horizontal line has completed scanning, a new line is started directly below it, and the left-to-right scan pattern repeats. This process continues until the last line of the frame is completed, and the process restarts for a new frame. Note that the front porch and back porch locations are not intuitive. This naming convention comes from the parameters' location in time relative to the sync pulse (see [Figure 13](#)).

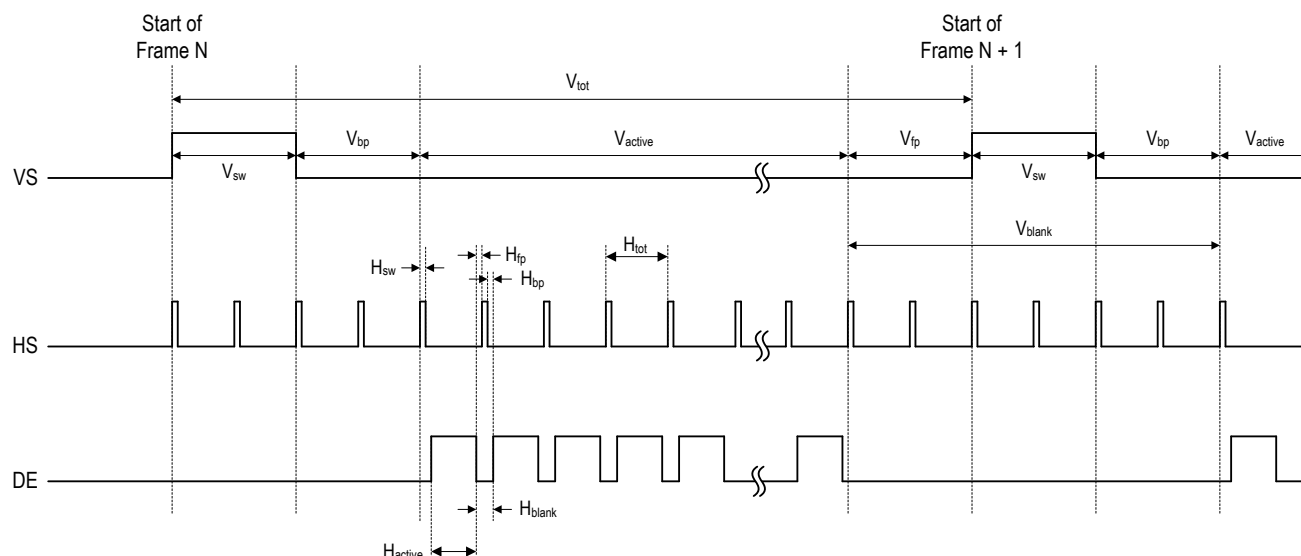


Figure 13. Sync Pulses: Data Enable and the Vertical and Horizontal Sync Signals

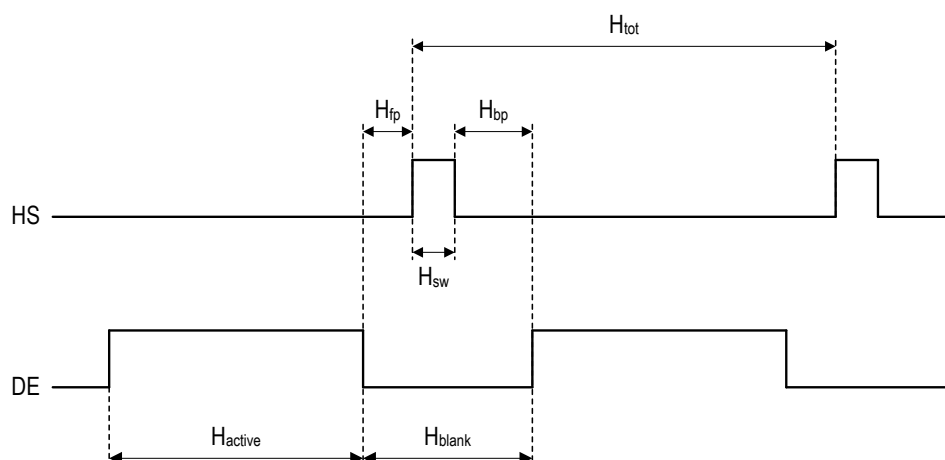


Figure 14. Horizontal Sync and Data Enable Signals

The diagram in [Figure 13](#) shows the relationships between the Vertical Sync, Horizontal Sync, and Data Enable signals. [Figure 14](#) exaggerates the horizontal blanking period to illustrate the relationship between the HS and DE signals as well as the video frame parameters.

5.2 Equations

This section contains video timing and blanking equations and demonstrates how to calculate video bandwidth. Note that total link bandwidth available for video data is less than the link transfer speed due to link encoding overhead and side channel usage (example, primary control channel, SPI, etc.). Also, the pixel clock and GMSL bandwidth must consider the total resolution of a video (that is, video data plus blanking). See the [GMSL2 Link System Bandwidth](#) section for further details.

5.2.1 Relationships Between Video Signals

Vertical Sync

- V_{SW} – Vertical Sync Width
- V_{FP} – Vertical Front Porch
- V_{BP} – Vertical Back Porch
- V_{active} – Vertical Active
- V_{tot} – Total Vertical Line Length
- V_{blank} – Vertical Blanking

Horizontal Sync

- H_{SW} – Horizontal Sync Width
- H_{FP} – Horizontal Front Porch
- H_{BP} – Horizontal Back Porch
- H_{active} – Horizontal Active
- H_{tot} – Total Horizontal Line Length
- H_{blank} – Horizontal Blanking

The video is active (that is, the display time in which the video contains visible pixel data) during the horizontal active period. The horizontal blanking period separates the active video of each visible horizontal line. The horizontal sync (during the blanking period) indicates the start of a new horizontal line. The vertical sync signals have analogous functions to the horizontal counterparts; however, the vertical signals are used to indicate frame boundaries and are sometimes used to trigger other events (example, buffer resets) between frames. The vertical sync signal occurs during the vertical blanking period.

These concepts are illustrated in the diagram ([Figure 12](#)). The active video (that is, display time) is equivalent to the video screen that actively shows video data. The pixel data advances horizontally one pixel with each successive pixel clock period until the entire horizontal line is formed. The end of the horizontal line is designated by a horizontal sync pulse (HS) and signifies a shift to begin a new horizontal line directly below. This process continues until the entire frame is filled. When the entire frame is filled with horizontal lines, a vertical sync pulse (VS) indicates the end of the frame. The frame then refreshes and the process repeats. Note that cameras and SoCs often express resolution in terms of “active video” (that is, viewable resolution). This designation ignores blanking. The pixel clock and GMSL bandwidth is dependent on the total resolution, however, and includes blanking.

5.2.2 Total Blanking Interval

$$\text{Horizontal Blanking} = (H.\text{Front Porch}) + (H.\text{Sync Width}) + (H.\text{Back Porch})$$

$$\text{Vertical Blanking} = (V.\text{Front Porch}) + (V.\text{Sync Width}) + (V.\text{Back Porch})$$

5.2.3 Total Line and Frame Period

$$H_{total} = \text{Horizontal Active Pixels} + \text{Horizontal Blanking}$$

$$V_{total} = \text{Vertical Active Lines} + \text{Vertical Blanking}$$

5.2.4 Pixel Clock

Calculating the pixel clock (PCLK) is necessary to determine if video transmissions are compatible with interface and GMSL2 link bandwidth availability.

$$PCLK = H_{total} * V_{total} * \text{Framerate}$$

5.2.4.1 Pixel Clock (Alternative Calculation)

In some situations, the specific blanking timings are not known. In these cases, an approximation of the blanking timing (called “Blanking Ratio”) should be used to calculate an approximate PCLK value. Blanking ratios are typically in the range of 1.1–1.25; those values should be used for approximate calculations.

$$PCLK = (\text{Active Width}) * (\text{Active Height}) * (\text{Blanking Ratio}) * (\text{Frame Rate})$$

5.2.4.2 Sample Calculation

$$1920 \text{ h.pixels} * 1080 \text{ v.pixels} * 1.2 \text{ blanking ratio} * 60 \text{ Hz} = 149\text{MHz pclk}$$

5.2.5 Calculating Total Video Bandwidth

$$\text{Video BW} = PCLK * \text{bpp}$$

Where bpp = bits per pixel. For RGB888 video this is 24 bits per pixel (8 bits per color). Note that the bits per pixel value varies with other video formats (example, RAW).

For example, a 1920x1080 60Hz 24-bit RGB display has a video bandwidth of ~3.56 Gbps.

The GMSL2 link has additional overhead that shares bandwidth with video data. Therefore, a 6Gbps video stream cannot be transferred on a 6Gbps GMSL2 link. See the [GMSL2 Link System Bandwidth](#) section for further details.

5.3 Definitions

This section contains definitions of basic video terminology.

5.3.1.1 Aspect Ratio

The ratio of the visible picture width to the height. Traditional televisions and monitors have an aspect ratio of 4:3 (1.33). The standard aspect ratio of modern computer monitors and automotive infotainment displays is 16:9 (1.78). Automotive instrument cluster displays are typically 8:3 (2.66), but custom aspect ratios are also common.

5.3.1.2 Back Porch

The area of a composite video signal defined as the time between the end of the sync signal and the start of the of active video.

5.3.1.3 Blanking Interval

There are horizontal and vertical blanking intervals. The horizontal blanking interval is the time period from the last active pixel in a line to the first active pixel in the following line. The vertical blanking interval is the time period from the last active pixel in a field or frame to the first active pixel in the following field or frame. The synchronizing signals occupy a portion of the blanking interval.

5.3.1.4 Color Bars

A standard video waveform used to test the calibration of a video system. It consists of a sequence of seven colored bars of a standard amplitude and size. The standard active-low color sequence is white, yellow, cyan, green, magenta, red, and blue. There are several amplitude standards, the most common being 75% amplitude (brightness) with 100% saturation (intensity of the color).

5.3.1.5 Data Enable

The signal defines the valid video data. When the Data Enable signal is off, the video data is ignored. Note that cameras typically do not use DE signals, and instead use the HS and VS signals to indicate valid video data. Some display applications use only DE signals, while the HS and VS sync signals are ignored.

5.3.1.6 Fields and Frames

A frame is one complete scan of a picture. In interlaced scanning systems, a field is half of a frame; thus, two fields make a frame.

5.3.1.7 Frame Rate

See Vertical Frame Rate.

5.3.1.8 Front Porch

The area of a composite video waveform between the end of the active video and the leading edge of sync.

5.3.1.9 GMSL

Gigabit Multimedia Serial Link. GMSL is a proprietary serial link protocol used to transmit video, bidirectional audio, and bidirectional communication channel data over an automotive-grade physical interface.

5.3.1.10 Horizontal Blanking

The horizontal blanking interval is the time period from the last active pixel in a line to the first active pixel in the following line. Synchronizing signals occupy a portion of the blanking interval.

5.3.1.11 Horizontal Line Frequency

The inverse of the period of the total horizontal line time.

5.3.1.12 Horizontal Sync

The beginning of a horizontal line is indicated by a HS “pulse” during the horizontal blanking period. Sometimes, the Horizontal Sync is replaced by Data Enable.

5.3.1.13 Interlaced Scan

The process whereby each frame of a picture is created by first scanning half of the lines and then scanning the second set of lines. The second set of lines is interleaved between the first set to complete the picture. Each half is referred to as a field. Two fields make a frame. See also Progressive Scan.

5.3.1.14 Pixel

Picture element. A pixel is the smallest piece of display detail, and each has a unique brightness and color. In a digital image, a pixel is an individual point in the image, represented by a certain number of bits to indicate the brightness.

5.3.1.15 Pixel Clock

The clock rate of pixels in a video stream. Pixel data advances at every clock edge.

5.3.1.16 Progressive Scan

The process whereby a picture is created by scanning all the lines of a frame in successive passes. Contrast with Interlaced Scan. The process of converting from interlaced to progressive scan is called "line doubling."

5.3.1.17 Refresh Rate

See Vertical Frame Rate.

5.3.1.18 Sync Signals/Pulses

Sync signals, also known as sync pulses, are timing pulses in video signals that are used by video processing or display devices to synchronize the horizontal and vertical portions of the display. They include *Horizontal Sync* and *Vertical Sync*. These signals typically occur during the blanking period(s).

5.3.1.19 Vertical Blanking

The vertical blanking interval is the time period from the last active pixel in a field or frame to the first active pixel in the following field or frame. Synchronizing signals occupy a portion of the blanking interval.

5.3.1.20 Vertical Field Frequency

The inverse of the time (or period) to produce one field of video (half of a frame). In NTSC, it is 59.94Hz.

5.3.1.21 Vertical Frame Rate

The inverse of the time (or period) to produce one frame of video. Also called "refresh rate" or "vertical refresh rate."

5.3.1.22 Vertical Sync

The beginning of a frame is indicated by a VS "pulse" during the vertical blanking period in which data enable is also turned off.

5.4 Configuration

The video Tx/Rx blocks have the following programmable modes of operation:

- Heartbeat Mode On: Enables video clock regeneration
- Heartbeat Mode Off: Disables video clock regeneration (CSI-2)

5.4.1 Heartbeat Mode On

Default setting. No additional configuration is required. 

5.4.2 Heartbeat Mode Off

For applications where PCLK regeneration is not required, it is suggested to completely turn off data transmission during blanking periods. This eliminates video bandwidth consumption during blanking.

Note: Heartbeat mode should only be disabled when the deserializer device has CSI-2.

Serializer:

- Disable heartbeat packet transmission: `LIM_HEART = 1'b1`

Deserializer:

- Disable heartbeat packet transmission: `LIM_HEART = 1'b1` (if available)

Otherwise

- Disable the packet sequence number checker: `SEQ_MISS_EN = 1'b0`
- Disable the packet detector: `DIS_PKT_DET = 1'b1`

Note: If the GMSL2 device has multiple video pipes, these registers are set individually for each video pipe.

6. Video Pipes

6.1 Overview

All GMSL2 devices use a common set of video data processing blocks to transfer video data across the serial link. This section describes the video input and output blocks and explains how data flows through the “video pipe” structures. Video pipes are used to route video data through the GMSL2 devices and contain several important video data processing features, including:

- Test features: VPG, VTG, and PRBS
- Color mapping tools: Video crossbar and color lookup tables (LUT)
- Authentication features: Watermark
- HS and VS monitoring functions

6.2 Operation

The GMSL2 video channel is designed to transmit 8-bit to 24-bit video data. Video data is received by the serializer at the input video interface (example, HDMI), converted to parallel video data, and transmitted through the serializer video pipes to the GMSL packetizer. The data is then packetized and transmitted by the GMSL PHY across the serial link. The connected deserializer receives the data on the GMSL PHY, and the data is depacketized into parallel video data. The parallel video data is transmitted through the video pipes to the output video interface (example, oLDI). Within the GMSL parts, the parallel video data consists of the pixel data and HS, VS, and DE sync signals. [Figure 15](#) shows a functional block diagram of the video processing blocks within a GMSL system.

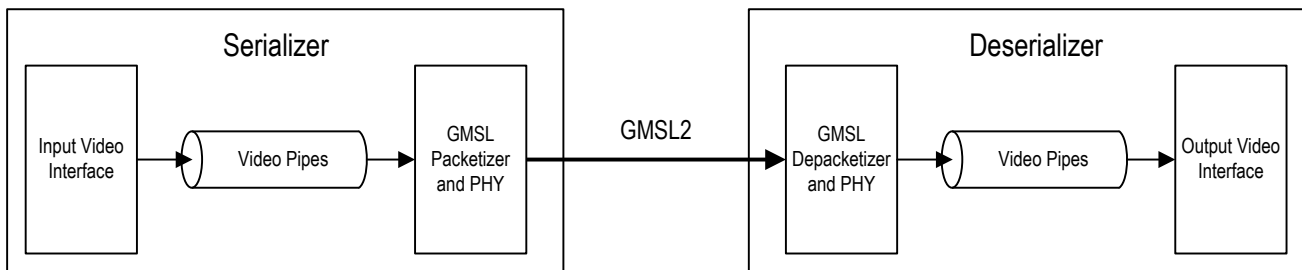


Figure 15. Video Pipes in a GMSL2 System

6.2.1 Video Packetizer (Serializer)

The video packetizer receives video data from the crossbar output when video transmission is enabled, and the link is locked. Upon receiving video data, the packetizer prepares the video data to be transmitted throughout the serial link system.

The packetizer detects the presence of the PCLK and measures the frequency once it has stabilized. The measured PCLK value is transmitted to the deserializer through info frame packets for clock regeneration. The PCLK is monitored for maximum threshold and flag violations that result in video blanking. PCLK drift detection (that is, $\pm 1.25\%$ drift without spread tracking and $\pm 0.25\%$ drift when tracking $\pm 0.5\%$ input spread at PCLK) reports errors to the `DRIFT_ERR` register and restarts the sub-system.

Note: The maximum PCLK value varies by part number. Refer to device-specific data sheet for more information.

The video is packed according to the selected mode and bits per pixel (bpp) setting to create video packets. These packets fill the transmit FIFO and a transmission request is sent to the scheduler. The video data is then transmitted to the serial link with optional video line CRC (enabled by default) or video pixel CRC (disabled by default). See the [CRC Error Detection and ARQ Error Correction](#) section for additional information and configuration details.

Note: The transmit FIFO has a level warning, auto-priority increment, and overflow detection.

6.2.2 Video Depacketizer (Deserializer)

The transmitted video packets are received by the GMSL PHY and sent to the depacketizer to be decoded. The data is then written to the video clock regeneration FIFO in a burst according to the selected encoding mode, bpp setting, and packet type. The clock regeneration module receives the decoded video from the link clock domain and streams out the video using the PCLK regenerated with the DPLL and clock control loop.

Note: The number of video Rx blocks per deserializer varies by part number. Refer to device-specific data sheet for more information.

6.2.2.1 Video Packet Detector

The video packet detector observes the output of the packet receiver on the deserializer side and measures the density of video packets within the total link bandwidth to determine if video is running on the link. Using the 150MHz oscillator, the total valid video packets inside the link are measured; packet detection is set if the measured PCLK is larger than 5.3MHz. As there are 36 pixels in each packet, packet throughput of ~146 kilo packets per second is set as threshold.

6.3 Block Descriptions

The following sections briefly describe the blocks within the video pipes. More information can be found in the dedicated sections.

6.3.1 GMSL2 Serializers

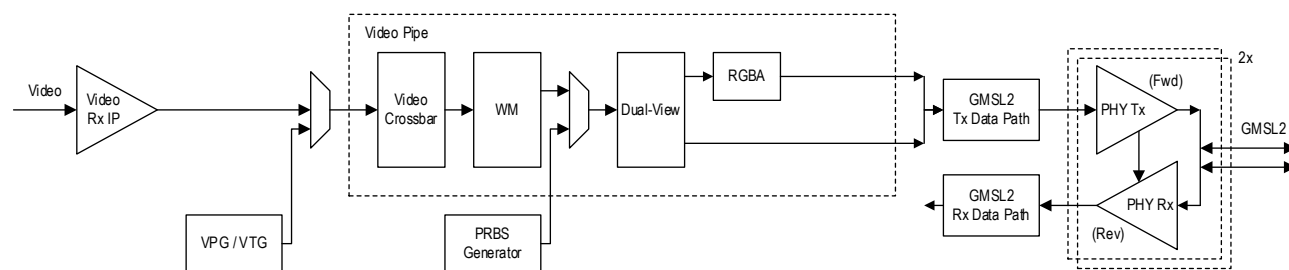


Figure 16. GMSL2 Serializer Video Pipe Structure

6.3.2 GMSL2 Deserializers

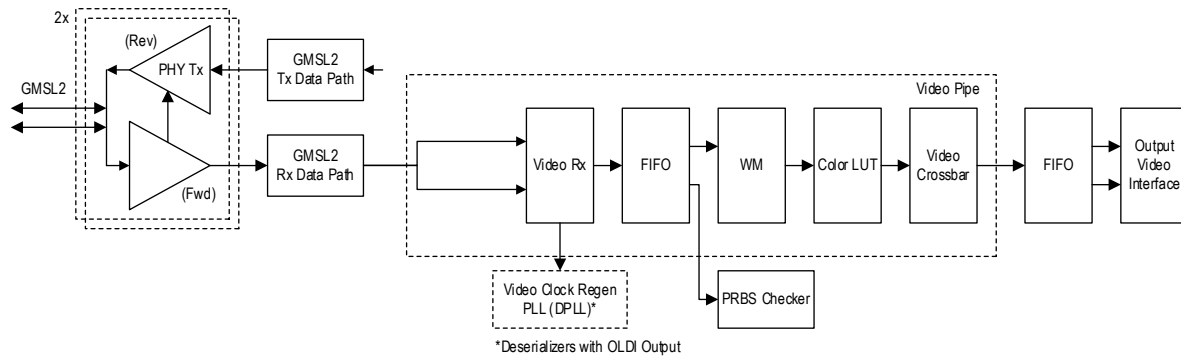


Figure 17. GMSL2 Deserializer Video Pipe Structure

6.3.3 Video Timing Generator (VTG)

The *Video Timing Generator (VTG)* enables the creation or modification of the HS, VS, and DE signals.

6.3.4 Video Pattern Generator (VPG)

The *Video Pattern Generator (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 a video pattern when no video is present on the serializer input.

6.3.5 Watermark (WM)

Serializers contain a watermark generator and detector block in video pipe X. Deserializers contain a watermark generator and detector block in video pipe X. Reference the *Watermarking* section for additional information.

6.3.6 Video PRBS Tx/Rx

Serializer video pipes contain a video PRBS generator that sends a PRBS pattern within the video packets sent across the link; the PRBS pattern is checked by the video PRBS checker in the deserializer video pipe. See the *PRBS Testing* section for additional information.

6.3.7 Dual-View Control

Dual-View enables incoming video to be split into two video streams.

6.3.8 Video Crossbar

The *Video Crossbar* is a mux that allows any video input bit to be mapped to any output bit using the configuration registers. This is used to remap/reorder pixel components.

6.3.9 Color Lookup Table (LUT)

The *Color Lookup Table (LUT)* allows any color in RGB domain to be mapped to any other RGB color. This can be used for generating color filters or gamma correction maps.

6.3.10 HS and VS Monitoring

For debug purposes, HS and VS can be measured through dedicated GPIO pins. This function must be enabled for each signal individually. Refer to device-specific data sheets for details regarding the GPIO pin used for each monitor function and the location of the associated enable bit(s).

6.4 Configuration

The video Tx/Rx blocks have the following programmable modes of operation:

- *Heartbeat Mode On*: Enables video clock regeneration (required for oLDI output).
- *Heartbeat Mode Off*: Disables video clock regeneration (CSI-2 output).
- *GMSL2 Stream ID Select*
- *GMSL2 PHY-Level Distribution of Video Streams*

See the interface-specific sections for complete configuration details.

6.4.1 Heartbeat Mode On

Default setting. No additional configuration is required.

6.4.2 Heartbeat Mode Off

For applications where PCLK regeneration is not required, it is suggested to completely turn off data transmission during blanking periods. This eliminates video bandwidth consumption during blanking.

Note: Heartbeat mode should only be disabled when the deserializer device has CSI-2 or eDP/DP output.

Serializer:

- Disable heartbeat packet transmission: `LIM_HEART = 1'b1`

Deserializer:

- Disable the packet sequence number checker: `SEQ_MISS_EN = 1'b0`
- Disable the packet detector: `DIS_PKT_DET = 1'b1`

Note: If the GMSL2 device has multiple video pipes, the registers above are set individually for each video pipe.

6.4.3 GMSL2 Stream ID Select

Each active video pipe in the serializer transmits data with a unique `STREAM_ID` assigned with the `TX_STR_SEL` register. On the deserializer side, each video pipe has a configurable `STR_SEL` register that selects the `STREAM_ID` to be received from the serializer. Video streams can be duplicated by programming multiple deserializer video pipes to the same `STR_SEL` setting.

Note: The GMSL2 video `STREAM_ID` is only related to the GMSL2 protocol. It is distinct from the virtual channels of MIPI CSI-2 and the `STREAM_ID` of the HDMI

6.4.4 GMSL2 PHY-Level Distribution of Video Streams

Select GMSL2 serializer devices have two GMSL PHYs (that is, PHY A and PHY B). Video pipe output can be configured to be sent from PHY A, PHY B, or both using the GMSL2 packet protocol scheduler. This is set with the `TX_SPLT_MASK_A_VIDEO_X/Y` and `TX_SPLT_MASK_B_VIDEO_X/Y` registers. For each serializer video pipe, these registers define the destination deserializer device(s).

6.5 Status and Error Bits

The following register bits provide status and error information related to video pipe operation in serial link systems.

6.5.1 Deserializer Status and Error Bits

`VID_LOCK` (register `VIDEO_RX8`): Indicates that a valid video stream is received for the selected channel ID.

Note: For oLDI deserializers, `VID_LOCK` also indicates that the valid video stream is properly recovered by the LVDS to drive the display. For all other deserializers, `VID_LOCK` = 1 when the video is received regardless of the status of the output interface.

`VID_PKT_DET` (register `VIDEO_RX8`): Indicates that a stream of video packet for the selected stream ID is detected by the deserializer.

6.5.2 Serializer Status and Error Bits

`PCLKDET` (register `VIDEO_TX2`): Status registers for video pipe X, Y, Z, and U. Normal operation reads 1 if the pipe is utilized.

`OVERFLOW` (register `VIDEO_TX2`): Reports if video bandwidth is exceeded.

`DRIFT_ERR` (register `VIDEO_TX2`): Reports a drift error caused by the instability of the video PCLK frequency.

6.6 Feature Availability by Device Family

All GMSL2 devices have video pipes. Device-specific feature availability is presented in [Table 11](#).

Table 11. Video Pipe Feature Support by Device Family

Device Family	Video Pipes	DSC	RGBA	Heartbeat Modes	Video Crossbar	HDCP	GMSL1 Mode
HDMI Serializers	2			X	X	X	
CSI-2 Serializers	4			X	X		X
Advanced CSI-2 Serializers	4			X	X		X
CSI-2 Camera Deserializers	4	X	X	X	X		X
oLDI Deserializers	1			X	X	X	
CSI-2 Quad Deserializers	8			X	X		X

7. Dual-View

7.1 Overview

Dual-view is a configurable feature included with HDMI GMSL2 serializers that enables a single serializer to support two different video streams. A serializer with dual-view enabled can receive a video stream comprising two videos combined into a single frame. This combined frame is called a superframe. The dual-view block takes the incoming superframe video stream, splits it into two symmetric or asymmetric video streams according to the enabled mode, and outputs the video onto two independent GMSL stream IDs. The output video streams can then be routed to one or more deserializers depending on application.

In the example below, a superframe of 3840 x 1200 at 60Hz is created in the SoC by combining two videos side-by-side into a single frame. The superframe is output from the SoC and received by the serializer. The serializer, with symmetric dual-view enabled, splits the superframe into two separate video streams marked as 'Video A' and 'Video B' ([Figure 18](#)). Each of these streams is 1920 x 1200 at 60Hz.

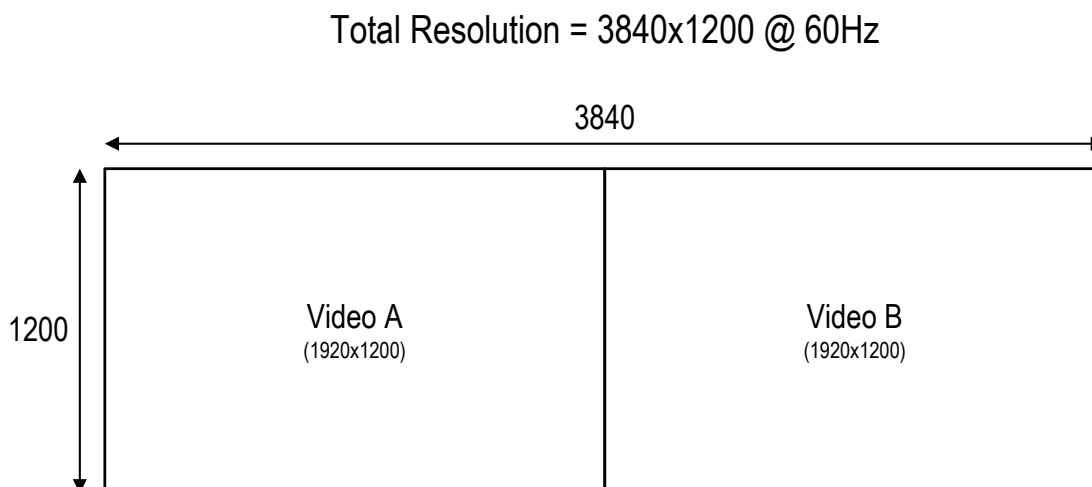


Figure 18. Example of Side-by-Side Symmetric Dual-View

7.2 Modes of Operation

The HDMI serializer supports a dual-view mode of operation categorized as symmetric.

7.2.1 Symmetric Dual-View Modes

Symmetric dual-view modes require that constituent videos (that is, the sub-frames) of the incoming superframe have identical timing (that is, active resolution, blanking interval, and frame rate). The incoming superframe video on the serializer input (from the SoC) is split into two separate sub-frames by the dual-view block and output by the serializer. Three types of symmetric dual-view superframes are supported:

- Pixel-interleaved dual-view mode

- Side-by-side dual-view mode
- Line-interleaved dual-view mode

The choice of dual-view mode is dependent on the superframe timing as output by the SoC and received by the serializer. In each mode, the serializer should be configured to match the superframe created by the SoC. When configured, the superframe is split into two symmetrical video frames by the dual-view block of the serializer and output onto two independent GMSL stream IDs. [Figure 19](#) shows the timings for each sub-frame, Video A and Video B.

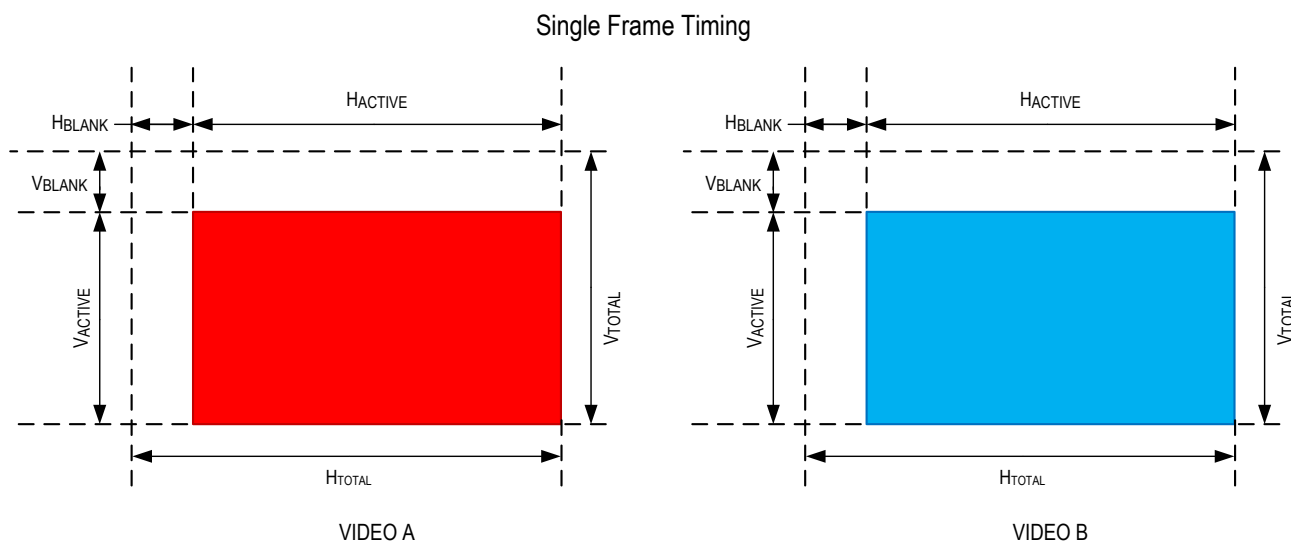


Figure 19. Single Frame Timing

7.2.1.1 Pixel-Interleaved Dual-View Mode

Pixel-interleaved mode (also known as column-interleaved mode) separates the video stream by assigning every odd pixel to one stream and every even pixel to another stream ([Figure 20](#)). This includes the horizontal blanking, which is split in half and symmetrically distributed to the two video streams. Vertical blanking timing is maintained from superframe to each sub-frame video stream. The

PCLK is divided by two for each sub-frame.

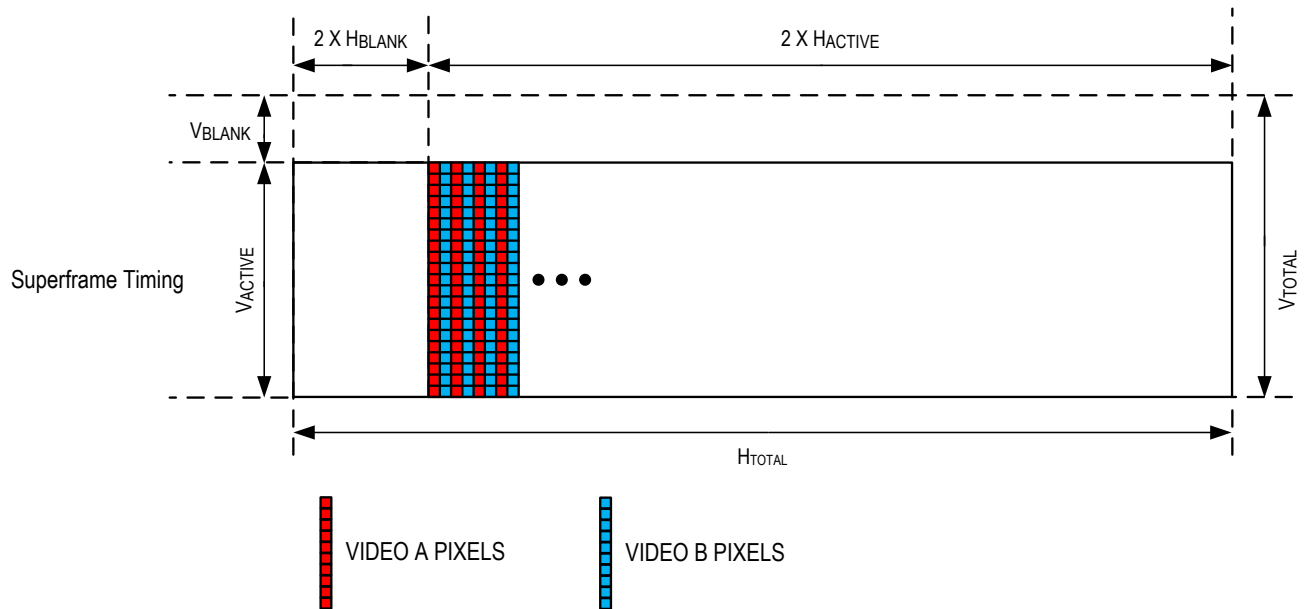


Figure 20. Pixel-Interleaved Dual-View Mode

7.2.1.2 Side-by-Side Dual-View Mode

Side-by-side mode is used to split the superframe in half by creating sub-frames from the left and right halves of the superframe, as shown in [Figure 21](#). Side-by-side dual-view mode works by first converting the superframe data into pixel-interleaved data. Then, the pixel-interleaved data is separated by assigning every odd pixel to one stream and every even pixel to another stream. When side-by-side dual-view mode is enabled, the dual-view block measures the length of the superframe line that contains the side-by-side image format. Assuming that the pixel-length of the line is $2N$, the pixels are reordered in the following order:

0,N,1,N+1,.....N-2,2N-2,N-1,2N-1

Here, the odd pixels are sent to one video stream and even pixels are sent to another stream.

Horizontal blanking is split symmetrically between streams, and vertical timing is maintained from superframe to the two sub-frame video streams. The PCLK is divided by two for each sub-frame.

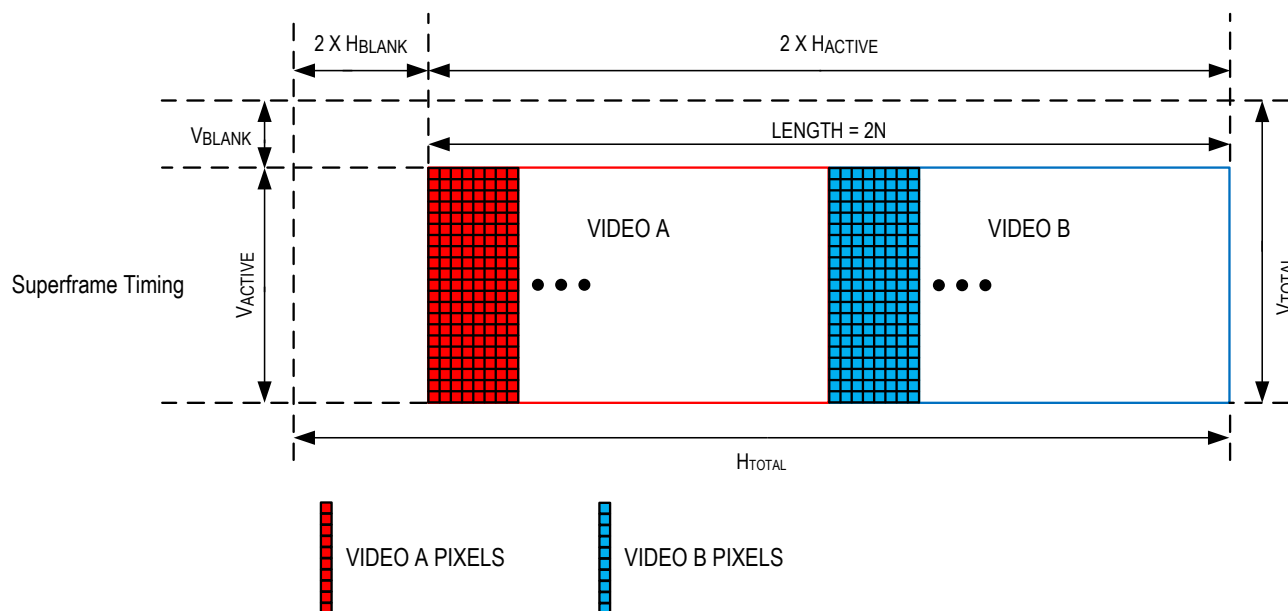


Figure 21. Side-by-Side Dual-View Mode

7.2.1.3 Line-Interleaved Dual-View Mode

In line-interleaved mode, the incoming data is split horizontally line-by-line (Figure 22). Odd lines (beginning with the first active line of a frame) contain Video A; even lines contain Video B. When line-interleaved mode is enabled, odd lines are split into one video stream and even lines into the other. Horizontal blanking timing is maintained from superframe to sub-frame, but vertical timing is symmetrically split between streams. The PCLK is divided by two for the sub-frames.

Note: Not all devices support line-interleaved dual-view mode; see [Feature Availability by Device Family](#) for more information.

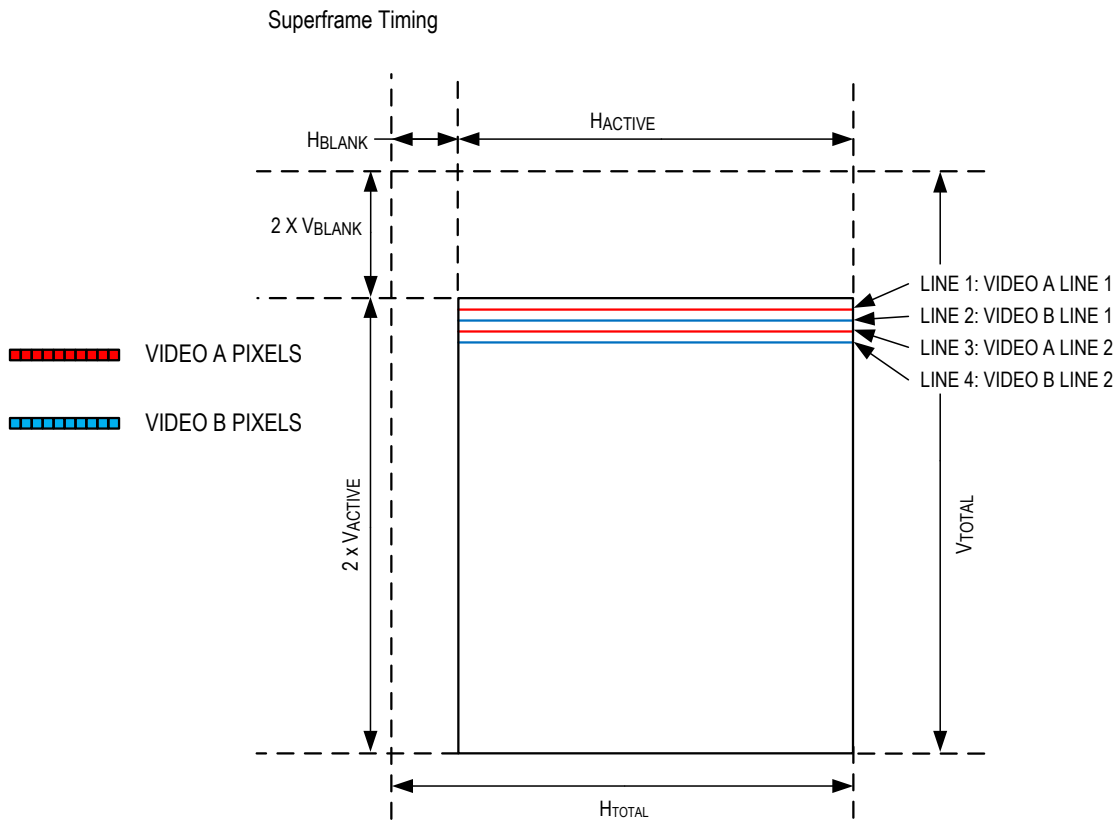


Figure 22. Line-Interleaved Dual-View Mode

7.3 Configuration

Note: Dual-view configuration registers must be written in the order presented in the tables.

7.3.1 Pixel-Interleaved Dual-View

Enable dual-view mode by setting `DV_EN = 1` in register `DV0`. Set `DV_SPL = 1` to split the received superframe into sub-frames that are sent to video pipe X and video pipe Y. Odd pixels (starting with the first pixel of a line) output to pipe X; even pixels output to pipe Y. In register `DV2`, enable video pipe X by setting `VID_EN_X = 1` and enable video pipe Y by setting `VID_EN_Y = 1`.

The stream ID of the pipe X video stream is set with `TX_STR_SEL` in register `VIDEO_X TX3`. If the device is in splitter mode (that is, connected to two deserializers), prevent the video in pipe X from being sent to one or both SIOx serial link outputs by enabling `TX_SPLT_MASK_A` and/or `TX_SPLT_MASK_B`. Repeat these steps for pipe Y. This configuration option is typically used to save link bandwidth. If dual-view is being used in combination with splitter mode, these registers are used to ensure that correct signal routing is established (example, one video is sent to the deserializer on SIOA and the other video is only sent to the deserializer on SIOB).

The configuration process for enabling pixel-interleaved dual-view mode is presented in [Table 12](#).

Table 12. Pixel-Interleaved Dual-View Configuration Registers

Step	Description	Register Name	Register Write
1	Enable dual-view block.	DV0	DV_EN = 1
2	Split the video into two streams, outputting on pipe X and pipe Y.	DV0	DV_SPL = 1
3	Select which pipe receives even/odd pixels.	DV0	DV_SWP_AB = 0 sends odd pixels to pipe X and even pixels to pipe Y. DV_SWP_AB = 1 sends even pixels to pipe X and odd pixels to pipe Y.
4	Enable video stream on pipe X (default).	DV2	VID_EN_X = 1
5	Enable video stream on pipe Y.	DV2	VID_EN_Y = 1
6	Assign pipe X stream ID and select which PHY to send video.	VIDEO_X TX3	Set TX_STR_SEL, TX_SPLT_MASK_A, and TX_SPLT_MASK_B.
7	Assign pipe Y stream ID and select which PHY to send video.	VIDEO_Y TX3	Set TX_STR_SEL, TX_SPLT_MASK_A, and TX_SPLT_MASK_B.

7.3.2 Side-by-Side Dual-View

Enable dual-view mode by setting **DV_EN** = 1 in register **DV0** and set **DV_CONV** = 1 to enable side-by-side to pixel-interleaved conversion. Set **DV_SPL** = 1 to split the received superframe into sub-frames that are sent to video pipe X and video pipe Y. Left-side pixels output to pipe X; right-side pixels output to pipe Y. To switch the output orientation, set **DV_SWP_AB** = 1 to output right-side pixels to pipe X and left-side pixels to pipe Y. In register **DV2**, enable video pipe X by setting **VID_EN_X** = 1 and enable video pipe Y by setting **VID_EN_Y** = 1.

The stream ID of the pipe X video stream is set with **TX_STR_SEL** in register **VIDEO_X TX3**. If the device is in splitter mode (that is, connected to two deserializers), prevent the video in pipe X from being sent to one or both SIOx serial link outputs by enabling **TX_SPLT_MASK_A** and/or **TX_SPLT_MASK_B**. Repeat these steps for pipe Y. This configuration option is typically used to save link bandwidth. If dual-view is being used in combination with splitter mode, these registers are used to ensure that correct signal routing is established (example, one video is sent to the deserializer on SIOA and the other video is only sent to the deserializer on SIOB).

The configuration process for enabling side-by-side dual-view mode is presented in [Table 13](#).

Table 13. Side-by-Side Dual-View Configuration Registers

Step	Description	Register Name	Register Write
1	Enable dual-view block.	DV0	DV_EN = 1
2	Convert side-by-side to pixel interleaved video stream.	DV0	DV_CONV = 1
3	Split the video into two streams, outputting on pipe X and pipe Y.	DV0	DV_SPL = 1

4	Select which pipe receives even/odd pixels.	DV0	DV_SWP_AB = 0 sends odd pixels to pipe X and even pixels to pipe Y. DV_SWP_AB = 1 sends even pixels to pipe X and odd pixels to pipe Y.
5	Enable video stream on pipe X (default).	DV2	VID_EN_X = 1
6	Enable video stream on pipe Y.	DV2	VID_EN_Y = 1
7	Assign pipe X stream ID and select which PHY to send video.	VIDEO_X TX3	Set TX_STR_SEL, TX_SPLT_MASK_A, and TX_SPLT_MASK_B.
8	Assign pipe Y stream ID and select which PHY to send video.	VIDEO_Y TX3	Set TX_STR_SEL, TX_SPLT_MASK_A, and TX_SPLT_MASK_B.

7.3.3 Line-Interleaved Dual-View

Configuring line-interleaved dual-view requires configuring the dual-view block as well as programming the *Video Timing Generator (VTG)*. The line-interleaved dual-view block divides the horizontal and vertical timings by two. So, the VTG is required to double the horizontal timings before the dual-view block to preserve the original horizontal timing of the input. This is shown in *Figure 23*.

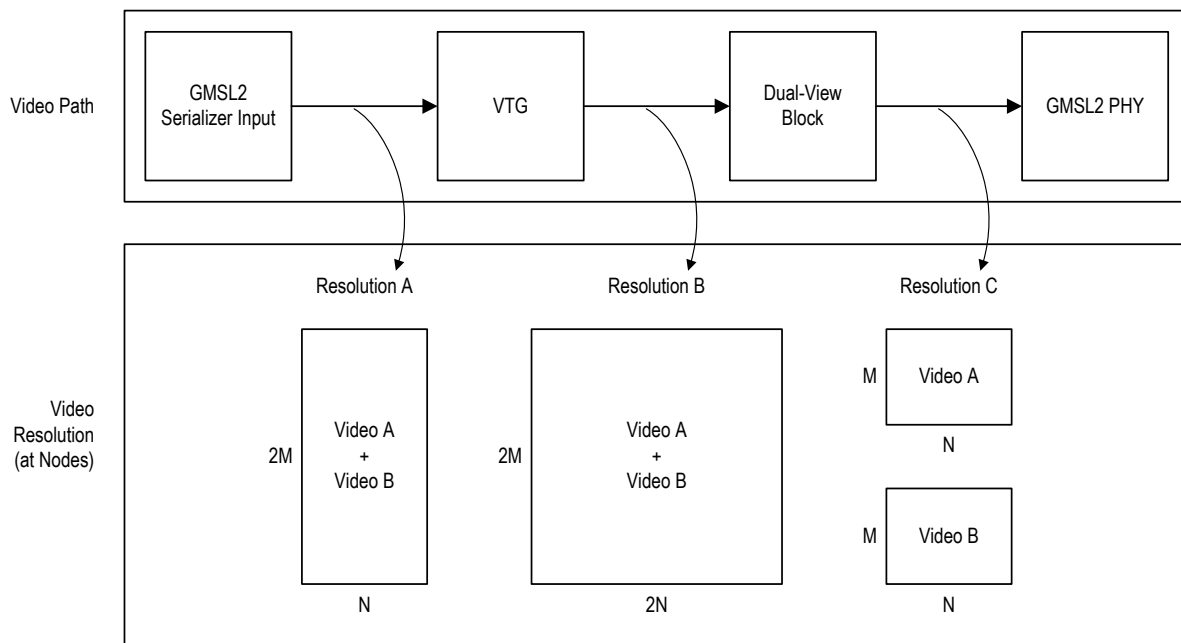


Figure 23. Video Timing Along the Video Path Through the Serializer

Enable line-interleaved dual-view mode by setting **LINE_ALT** = 1 in register **DV0**. Set **DV_SPL** = 1 to split the received superframe into sub-frames that are sent to video pipe X and video pipe Y. Odd lines (starting with the first line) output to pipe X; even lines output to pipe Y. To swap the between video pipe X and video pipe Y, set **DV_SWP_AB** = 1. In register **DV2**, enable video pipe X by setting **VID_EN_X** = 1 and enable video pipe Y by setting **VID_EN_Y** = 1.

The stream ID of the pipe X video stream is set with **TX_STR_SEL** in register **VIDEO_X TX3**. If the device is in splitter mode (that is, connected to two deserializers), prevent the video in pipe X from being sent to one or both SIOx serial link outputs by enabling **TX_SPLT_MASK_A** and/or **TX_SPLT_MASK_B**. Repeat these steps for pipe Y. This configuration option is typically used to save link bandwidth. If dual-view is being used in combination with splitter mode, these registers are used to ensure that correct signal routing is established (example, one video is sent to the deserializer on SIOA and the other video is only sent to the deserializer on SIOB).

As noted previously, when using line interleaved mode, the serializer's video timing generator (VTG) must be used to reconstruct the sync timings of the incoming video stream before it is sent to the dual-view block. A tool is available to create the correct VTG settings for a given superframe timing. These settings are also defined in [Table 14](#). For additional VTG operation and programming details, see [Video Timing Generator \(VTG\)](#).

Finally, in serializer register **DV0**, set **DV_EN** = 1 to enable dual-view.

Note: The system is designed for positive VS polarity. If the VS polarity is negative, see the configuration instructions in the [Line-Interleaved Dual-View with Negative Polarity](#) section.

The following register writes are defined for a superframe with video timing:

- Horizontal total pixels = Htot
- Horizontal active pixels = Ha
- Horizontal front porch = Hfp
- Horizontal sync width = Hsw
- Horizontal back porch = Hbp
- Vertical total pixels = Vtot
- Vertical active pixels = Va
- Vertical front porch = Vfp
- Vertical sync width = Vsw
- Vertical back porch = Vbp

The configuration process for enabling line-interleaved dual-view mode is presented in [Table 14](#).

Table 14. Line-Interleaved Dual-View Configuration Registers

Step	Description	Register Name	Register Write
1	Enable Line-interleaved dual-view mode.	DV0	LINE_ALT = 1
2	Split the video into 2 streams, outputting on pipe X and pipe Y.	DV0	DV_SPL = 1
3	Select which pipe receives even/odd pixels.	DV0	DV_SWP_AB = 0 sends odd lines to pipe X and even pixels to pipe Y. DV_SWP_AB = 1 sends even lines to pipe X and odd pixels to pipe Y.
4	Enable video stream on pipe X (default).	DV2	VID_EN_X = 1
5	Enable video stream on pipe Y.	DV2	VID_EN_Y = 1

6	Assign pipe X stream ID and select which PHY to send video.	VIDEO_X TX3	Set TX_STR_SEL, TX_SPLT_MASK_A, and TX_SPLT_MASK_B.
7	Assign pipe Y stream ID and select which PHY to send video.	VIDEO_Y TX3	Set TX_STR_SEL, TX_SPLT_MASK_A, and TX_SPLT_MASK_B.
8	VTG : Enable VTG, set to VS one-trigger mode, and generate DE, HS, and VS signals.	VTX0	VTG_MODE[1:0] = 01 GEN_DE = 1 GEN_HS = 1 GEN_VS = 1
9	VTG : Trigger VS on rising edge.	VTX1	VS_TRIG = 1
10	VTG : Time between VS edge of superframe and VTX VS generation. Used as a buffer for making sure first line of superframe is written to the memory before starting to read. (1 line + 40 pixels)	VTX2 - VTX4	VS_DLY = Htot + 40
11	VTG : VS high time of the superframe in terms of superframe pixels.	VTX5 - VTX7	VS_HIGH = Vsw x Htot
12	VTG : VS low time of the superframe in terms of superframe pixels.	VTX8 - VTX10	VS_LOW = (Vtot - Vsw) x Htot
13	VTG : Time between VS edge of superframe and first VTX HS generation. Must be equal to VS_DLY to keep VS and HS aligned.	VTX11 - VTX13	V2H = Htot + 40
14	VTG : Double HS high time in terms of superframe pixels.	VTX14 - VTX15	HS_HIGH = 2 x Hsw
15	VTG : Double HS low time in terms of superframe pixels.	VTX16 - VTX17	HS_LOW = 2 x (Htot - Hsw)
16	VTG : Half the number of HS pulses in the superframe.	VTX18 - VTX19	HS_CNT = Vtot/2
17	VTG : Time between VS rising edge of superframe and first VTG DE generation. This value is equal to VS_DLY + vertical sync width plus back porch time + two horizontal sync width plus horizontal back porch time.	VTX20 - VTX22	V2D = (Htot + 40) + Htot x (Vsw + Vbp) + 2 x (Hsw + Hbp)
18	VTG : Two times the DE high time in a superframe line in terms of superframe pixels.	VTX23 - VTX24	DE_HIGH = 2 x Ha
19	VTG : Two times the DE low time in a superframe line in terms of superframe pixels.	VTX25 - VTX26	DE_LOW = 2 x (Hfp + Hsw + Hbp)
20	VTG : Half the number of active lines in the superframe.	VTX27 - VTX28	DE_CNT = Vactive/2
21	Enable dual-view block.	DV0	DV_EN = 1

7.3.3.1 Line-Interleaved Dual-View with Negative Polarity

The system is designed for positive VS polarity. If the VS polarity is negative, invert VS using the **CROSS_25** register. Then, perform the following sequence to start the system properly:

```
#Method
#program the configuration
#turn off video tx to link
0x80,0x01A2,0x8A

#start HDMI video
#wait for HDMI_SCDT,CKDT,PCLKDET to be 1

#selextpclk=1
0x80,0x01C9,0xF1

#force vs=0, de=0
0x80,0x01F2,0x3A
0x80,0x01F1,0x39
0x80,0x2A0C,0x02

#turn off dual-view
0x80,0x01A0,0x26

#selextpclk=0
0x80,0x01C9,0xE1

#release force vs,de
0x80,0x01F1,0x19
0x80,0x01F2,0x1A
0x80,0x2A0C,0x00

#turn on dual-view
0x80,0x01A0,0x27

#turn on video tx to the link
0x80,0x01A2,0xEA
```

7.4 Status and Debug Registers

If the system does not perform as expected, analyze the video source output using video analyzer equipment to ensure that it satisfies the exact timing requirements.

7.4.1.1 Symmetric Dual-View

The key status registers associated with symmetric dual-view mode are:

- **DV_LOCK** (register **DV0**) shows whether the dual-view engine is locked or not.
- **PCLKDET** (register **VIDEO_TX2**) registers for pipe X and pipe Y should both be 1 if the dual-view engine is running and sending video to the two corresponding pipes.
- **VID_LOCK** (register **VIDEO_RX8**) register of the deserializer expecting the video should be 1 after the serializer starts sending the video streams through the GMSL2 link.

If the above registers are not in the expected states, check the `TX_SPLT_MASK_A` and `TX_SPLT_MASK_B` bits of register `CFGV_VIDEO_TX3` to determine whether the routing to the GMSL links are correct or not. Check that both video pipes used are enabled (`VID_EN_X = VID_EN_Y = 1`). Ensure that the `TX_STR_SEL` bits in the serializer register `CFGV_VIDEO_TX3` and the `STR_SEL` bits in the deserializer register `CFGH_VIDEO_RX0` indicate that the serializer and deserializer stream IDs match.

7.5 Feature Availability by Device Family

Dual-view is supported by GMSL2 HDMI and DSI serializers. Mode support varies by device family ([Table 15](#)).

Table 15. Dual-View Support by Device Family

Device Family	Pixel-Interleaved Dual-View	Side-by-Side Dual-View	Line-Interleaved Dual-View
HDMI Serializers	X	X	X
CSI-2 Serializers			
Advanced CSI-2 Serializers			
CSI-2 Camera Deserializers			
oLDI Deserializers			
CSI-2 Quad Deserializers			

8. Forward Error Correction

8.1 Overview

Forward Error Correction (FEC) is used in GMSL2 devices to detect and correct bit errors occurring during the transmission of compressed video on the serial link. The FEC implementation in GMSL2 devices corrects both single and burst errors and results in an improved bit error rate (BER). An additional CRC mechanism is used to indicate the presence of uncorrectable bit errors. FEC consumes an additional fixed 6.7% bandwidth overhead when enabled; however, it provides error correction without requiring reverse channel communication or incurring video stream delays.

FEC is used to increase the robustness of data transmitted over the forward (6Gbps) path on a GMSL3 link. The use of FEC and other design elements of GMSL2, ensure reliable communication.

8.2 Operation

8.2.1 Architecture

In the serializer, data is grouped into codeword blocks of 2560 bits and converted to GMSL data. After conversion, the GMSL data is encoded by the FEC block then processed by the 9b10b encoder/scrambler. The GMSL data is transmitted from the serializer's GMSL PHY over the serial link to the deserializer's GMSL PHY. In the deserializer, the GMSL data is first processed by the 9b10b decoder and de-scrambler. This block detects decode and idle errors. Bit errors are detected and corrected when GMSL data passes through the FEC decoder. The count of errors corrected by the FEC decoder block are reported to `fec_bit_errors_corrected`. The GMSL data then moves to the FEC CRC block. An 18-bit CRC is calculated per codeword block. If the received CRC matches the calculated CRC, the block is deemed correct; if the CRC values do not match, it is deemed an uncorrectable codeword block and reported to `fec_uncorrectable_blks`. The block diagram (Figure 24) illustrates the data path through the serial link system and highlights error detection, correction, and reporting behavior.

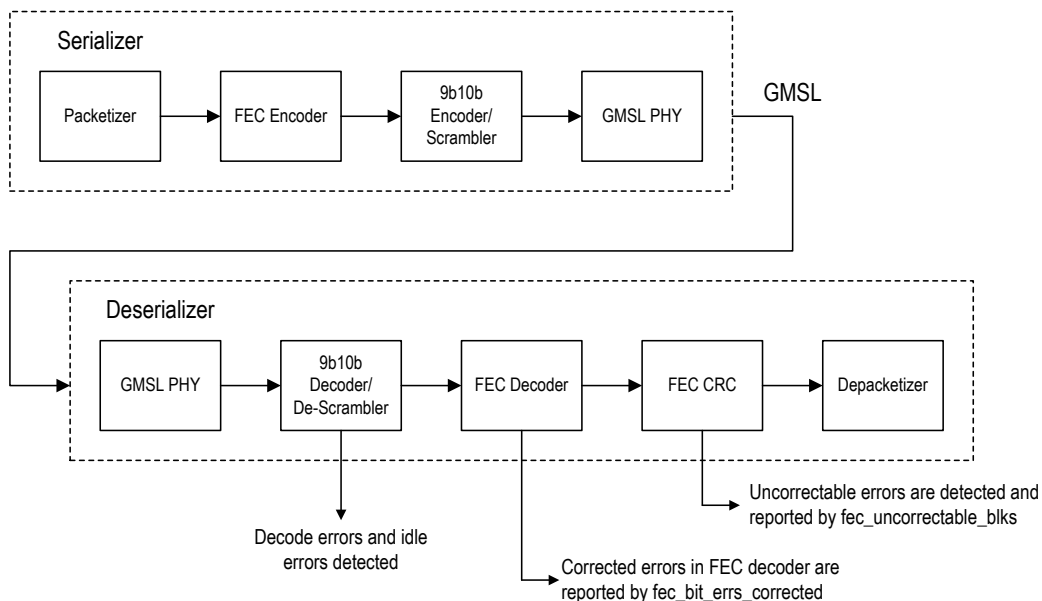


Figure 24. Architecture of FEC Implementation in GMSL Devices

8.2.2 BER Reduction Performance

FEC is designed to lower the BER on GMSL links containing compressed video streams. The FEC block operates with an error correction scheme based on Reed-Solomon error-correction codes. This coding scheme improves the BER in high-error systems (example, wireless transmission), and it is used to substantially improve the BER in GMSL systems and effectively mitigate the incidence of uncorrectable bit errors on compressed video streams. Without FEC enabled, compliant GMSL2 links operate with a BER range of approximately 10^{-12} to 10^{-16} ; with FEC enabled, the GMSL2 links operate with an improved BER range of approximately 10^{-40} to 10^{-60} . To illustrate, a link operating with a BER of 10^{-15} is improved to a BER of 10^{-55} with FEC enabled, which corresponds to a single uncorrectable bit error every 1.6×10^{38} years.

8.2.3 Bandwidth Overhead and System Impacts

The overall FEC bandwidth overhead is the combination of the Reed-Solomon correction overhead and the 18-bit CRC correction overhead. The code rate of Reed-Solomon $[N/K/S] = [127/121/7]$ plus an additional symbol to pack parity bits and the CRC results in a total bandwidth overhead of $128/120 = 6.677\%$.

The link overhead added when FEC is enabled must be considered when calculating GMSL bandwidth consumption. Additionally, FEC adds a small increase in the link latency of the video and control channel data that is proportional to the forward channel serial link rate. For 6Gbps link rate, the additional latency due to FEC encoding and decoding is 720ns; for 3Gbps link rate, the latency is 1440ns. I²C clock stretching, GPIO delay compensation, etc. can be used to compensate for this latency. See the [I²C/UART](#) and [Interface-Specific Bandwidth Calculations](#) sections for details.

8.2.4 Resynchronization

If a receiver loses and subsequently regains synchronization, FEC returns to normal operation without freezing. Therefore, FEC auto-recovers if the GMSL link lock is lost and recovered.

8.2.5 Power-Up Configuration

The FEC block is disabled by default on all GMSL2 devices. It is enabled using the procedure in the Configuration section.

8.3 Configuration

FEC only requires a single register write per device to be enabled; additional configuration may be needed to change the ERFB reporting depending on the requirements of the application.

Ensure that FEC is either enabled or disabled on both serializer and deserializer. The deserializer FEC should be enabled before or simultaneous to the serializer. Note that valid operation cannot be guaranteed until both devices have been properly configured using the procedures presented in the following sections.

Once both devices have been configured, FEC auto-recovers if the link is lost and re-established or either device is reset (and re-initialized).

8.3.1 Enabling FEC in a Single Microcontroller System

This procedure applies if all configuration of the serializer and deserializer is performed on one side of the link.

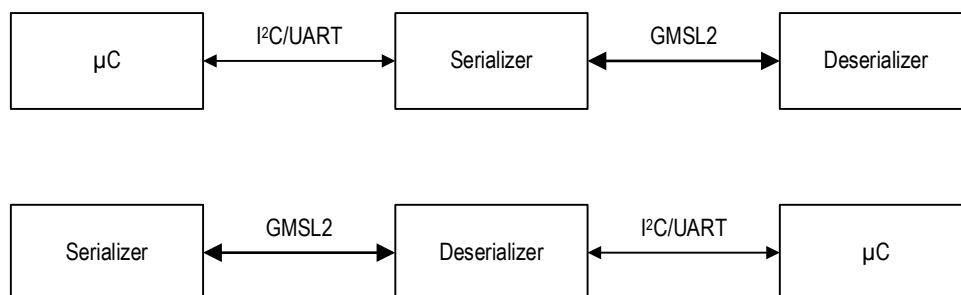


Figure 25. Block Diagram of a Single Microcontroller System

In a single microcontroller system, the microcontroller can be on either side of the link (Figure 25). The remote device is configured over the serial link through the control channel.

8.3.1.1 Single Microcontroller System Configuration Procedure

1. Wait for link LOCK (ensure bit `LOCKED` = 1 on serializer or deserializer)
2. Enable FEC in the deserializer (set `fec_en` = 1)
3. Enable FEC in the serializer (set `TX_FEC_EN` = 1)

8.3.2 Enabling FEC in a Dual Microcontroller System

This procedure describes how to enable FEC in a system that has a microcontroller on both sides of the link and has the control channel disabled (Figure 26). The devices can be programmed in any order provided that the link is held in reset and link lock is not established before configuration is completed.



*There is no control channel on the GMSL2 link

Figure 26. Block Diagram of a Dual Microcontroller System

8.3.2.1 Deserializer Programming Procedure

1. Write `RESET_LINK` = 1 (this prevents premature link lock).
2. Write other initialization registers (configuration of video, control channels, etc.).
3. Enable FEC (set `fec_en` = 1).
4. Clear all ERRs.
5. Write `RESET_LINK` = 0 (this allows link to lock when both devices are ready).

8.3.2.2 Serializer Programming Procedure

1. Write `RESET_LINK` = 1 (this prevents premature link lock).
2. Write other initialization registers (configuration of video, control channels, etc.).
3. Enable FEC (set `TX_FEC_EN` = 1).
4. Write `RESET_LINK` = 0 (this allows link to lock when both devices are ready).

8.4 Status and Debug Registers

FEC enables reporting of uncorrected link errors to the ERRB pin, as well as providing status registers to determine the exact link BER rate and post-correction BER rate. See the [GMSL2 Error Reporting \(ERRB Pin\)](#) section for additional information.

8.4.1 Error Reporting to the ERRB Pin

When the number of uncorrectable FEC errors is greater than `FEC_ERR_THR` (set to 1 by default), the `FEC_RX_ERR_FLAG` is set to 1. If `FEC_RX_ERR_OEN` = 1, the `FEC_RX_ERR_FLAG` error sets the ERRB pin high.

When `FEC_RX_ERR_FLAG` is read, it clears to 0. To reset error reporting so that additional FEC errors are detected, the FEC statistics counters must also be reset by setting `fec_clr_stats` = 1 (this bit self-clears after writing).

8.4.1.1 Example System Pseudocode

```

if Deserializer ERRB pin goes low and triggers hardware interrupt:
    # Read error registers to determine what error is active
    Read registers 0x001B, 0x001D, and 0x001F
    if FEC_RX_ERR_FLAG = 1
        react to error as defined by system engineer
        write fec_clr_stats = 1 to clear error counters
        read FEC_RX_ERR_FLAG to clear error
    end
end
  
```

8.4.2 Statistics Registers

The FEC block has status registers that indicate the number of uncorrected errors, number of corrected errors, and total number of codeword blocks processed. These status registers can be cleared at any time by writing `fec_clr_stats` = 1.

1. `fec_blks_processed` [31:0] – Blocks processed: number of codeword blocks received divided by 32768. For example, a value of 10 indicates that 327,680 blocks have been received. The scaling down by 32768 is implemented inside chip to avoid saturating the counter too soon. Write to clear or clear by writing `fec_clr_stats` = 1.
2. `fec_uncorrectable_blks` [31:0] – Uncorrectable errors: Number of codeword blocks that were uncorrectable by FEC (as detected by CRC). Write to clear or cleared by writing `fec_clr_stats` = 1.
3. `fec_bit_errs_corrected` [31:0] – Corrected bit errors: Number of bit errors corrected in the FEC block. Note that this does not map directly to the number of bit errors on the serial link (see [Link BER](#)). Write to clear or cleared by writing `fec_clr_stats` = 1.

8.4.3 BER Calculations

For any system, the measured BER is given by the equation:

$$BER = \frac{\text{Bit Errors}}{\text{Total Bits}}$$

However, since the observation period for bit errors can be prohibitively long in systems with a low BER, an estimated BER can be determined for a given observation period of no bit errors using a Poisson distribution. In this case, the confidence level that a system is operating at or better than a given BER after observing no bit errors over a period of N_{bits} is given by:

$$Confidence\ Level\ (CL) = 1 - e^{BER * N_{bits}}$$

Rearranging and solving for a 95% confidence level:

$$BER_{estimated} = \frac{-\ln(0.05)}{N_{bits}}$$

These equations can be combined with the status registers in the FEC block to get the measured and estimated BER for a system.

8.4.3.1 Link BER

The Link BER (pre-FEC bit corrections) can be a good indicator of the health of the link, but it is not the BER seen by the system. To calculate the link BER, the number of bit errors and total bits in an observation period must be known.

Due to GMSL2 9b10b encoding, bit errors on the serial link do not map directly to bit errors seen by the FEC decoder block. An error on the link has a probability to cause between one and nine errors within the FEC decoder block. On average, a single bit error corrupts approximately 4.1 bits in the FEC block, so a scaling factor of 4.1 is applied to the BER approximation equations.

The approximate link BER is given by:

$$BER = \frac{fec_bit_errs_corrected}{4.1 * 2560 * 32768 * blks_processed}$$

If no bit errors are observed ($fec_bit_errors_corrected = 0$), using the Poisson distribution equation:

$$BER_{estimated} = \frac{-\ln(0.05)}{4.1 * 2560 * 32768 * blks_processed}$$

The approximate BER after the FEC block corrects errors is given by:

$$BER_{FEC\ OUTPUT} = \frac{fec_uncorrectable_blks}{4.1 * 2560 * 32768 * blks_processed}$$

There is a tool available in the GUI which provides the BER based on these equations.

9. Watermarking

9.1 Overview

The watermarking (WM) feature, available in some GMSL2 devices, is designed to detect frozen frames caused by SoCs in safety-critical applications. GMSL2 parts can both watermark safety-relevant video streams with a time-varying watermark or detect a watermarked video stream. It is implemented in a system by enabling the watermark in the first GMSL2 device in the video chain and detecting the watermark in the final device in the system. A frozen frame failure is indicated if the watermark detector fails to see all the generated watermarks between the generator and detector in a frame-based processing system (note that frozen frames that occur before the watermark generator or after the watermark detector are not detected). In this error condition, the watermark detector can generate an interrupt and/or blank the output video and restore the display to a safe state in less than 500ms.

The watermark feature incorporates the following capabilities:

- A highly redundant design that is robust to image processing.
- Use of four unique watermarks ensures frozen frame failures are detected in a video processing path with up to a three-frame video buffer.
- The watermark is embedded in the LSBs of the video images and is removed by the detector; frame rate control is used to restore the image quality with visually undetectable processing.
- Video streams without watermarks are not modified by the detector.
- The watermark control registers are keyed to prevent inadvertent disabling in an active system.
- The watermark processing supports RGB888 datatype only.
- End-to-end test capability is enabled with error generation; the error condition is emulated by only generating two of four watermarks.
- The watermark detect signal can be output to a GPIO on oLDI deserializers to provide an interrupt when watermark detection is lost.

Note: The watermark block is only on pipe X (except for CSI-2 quad deserializers). See [Feature Availability by Device Family](#).

The watermark feature is immune to the following image processing:

- Image resizing (50% to 200%)
- Image rotation (± 5 degrees)
- Obstructions of up to 25% of the image area
- Contrast adjustment
- Color compensation
- Image horizontal and vertical flipping

9.2 Watermark Operation

An example application is illustrated in [Figure 27](#). Here, a surround view system generates a human-viewable video stream routed through a non-ASIL processor to a console display. The serializer on the surround view system embeds the time-varying watermarks in the outbound video stream. The deserializer on the display interface continuously monitors for watermarked video streams. If a watermarked video stream is detected, the video stream is considered safety-relevant, and the detector confirms that the expected time-varying watermarks are present. If a watermarked video stream does not contain all the expected watermarks, the watermark detector signals an error condition (example, the head unit is generating a static image) and the detector asserts an error interrupt and/or video stream blanking.

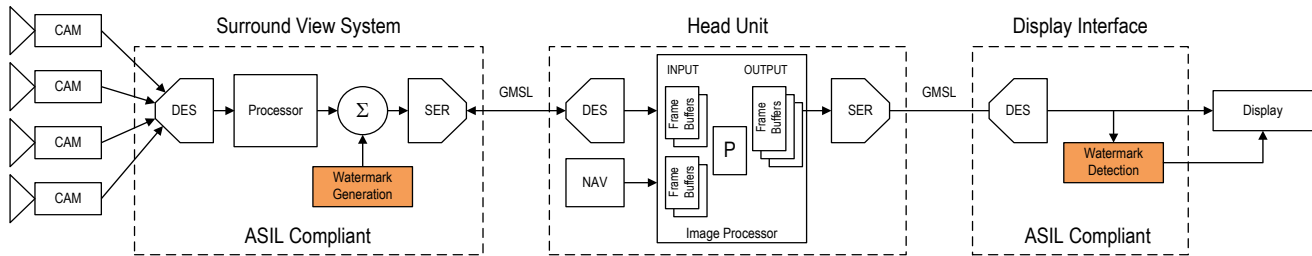


Figure 27. Surround View System Watermark Application

9.2.1 Watermark on Links with DSC Compression

A constant color (homogeneous) region is required for robust watermark detection on links with DSC compression enabled. The homogeneous area must be larger than 32 watermarks, calculated by:

$$\text{rounddown}\left(\frac{\text{width}}{320}\right) \times \text{height} \geq 32 \text{ watermarks}$$

Note: Watermark is not generated in top line of image.

9.2.2 Watermark Register Configuration

GMSL2 serializer and deserializers can generate or detect watermarks. Register configurations are based on the watermark generator and watermark detector located in the following register blocks:

1. WM_0 – WM_7
2. WM0_0 – WM0_7
3. WM1_0 – WM1_7
4. WM_WREN_0 – WM_WREN_1

9.2.3 Modes of Operation

- Watermark Generator
- Watermark Generator (with Error Generation)
- Watermark Detector
- Watermark Detector with Watermark LOCK Output to GPIO2
- Watermark Detector with Watermark ERR Output to ERRB
- Watermark Detector with Automatic Blank on WM ERR
- Watermark Detector with WM DET Filtering

9.2.3.1 Watermark Generator

Unlock registers: WM_WREN_L = 0xBA
 WM_WREN_H = 0xDC
 Enable the generator: WM_EN = 1'b1
 Lock registers: WM_WREN_L = 0x00
 WM_WREN_H = 0x00

9.2.3.2 Watermark Generator (with Error Generation)

Unlock registers: WM_WREN_L = 0xBA
 WM_WREN_H = 0xDC
 Enable the generator: WM_EN = 1'b1
 Enable ERR mode: WM_MODE[2:0] = 3'b001
 Lock registers: WM_WREN_L = 0x00
 WM_WREN_H = 0x00

9.2.3.3 Watermark Detector

Unlock registers: WM_WREN_L = 0xBA
 WM_WREN_H = 0xDC
 Configure as Detector: WM_DET[1:0] = 2'b01
 Enable the Detector: WM_EN = 1'b1
 Set the threshold correctly: WM_TH[6:0] = 0x0E
 Lock registers: WM_WREN_L = 0x00
 WM_WREN_H = 0x00

9.2.3.4 Watermark Detector with Watermark LOCK Output to GPIO2

9.2.3.4.1 oLDI Deserializers

Unlock registers: WM_WREN_L = 0xBA
 WM_WREN_H = 0xDC
 Configure as Detector: WM_DET[1:0] = 2'b01
 Enable the Detector: WM_EN = 1'b1
 Set the threshold correctly: WM_TH[6:0] = 0x0E
 Enable output to GPIO: WME_EN = 1'b1
 Lock registers: WM_WREN_L = 0x00
 WM_WREN_H = 0x00

9.2.3.5 Watermark Detector with Watermark ERR Output to ERRB

Unlock registers: WM_WREN_L = 0xBA
 WM_WREN_H = 0xDC
 Configure as Detector: WM_DET[1:0] = 2'b01
 Enable the Detector: WM_EN = 1'b1
 Set the threshold correctly: WM_TH[6:0] = 0x0E
 Enable Interrupt to ERRB: WM_ERR_OEN = 1'b1
 Lock registers: WM_WREN_L = 0x00
 WM_WREN_H = 0x00

9.2.3.6 Watermark Detector with Automatic Blank on WM ERR

Unlock registers: $WM_WREN_L = 0xBA$
 $WM_WREN_H = 0xDC$
 Configure as Detector: $WM_DET[1:0] = 2'b01$
 Enable the Detector: $WM_EN = 1'b1$
 Set the threshold correctly: $WM_TH[6:0] = 0x0E$
 Enable Video blanking: $WM_MASKMODE[1:0] = 2'b01$
 Lock registers: $WM_WREN_L = 0x00$
 $WM_WREN_H = 0x00$

9.2.3.7 Watermark Detector with WM DET Filtering

Unlock registers: $WM_WREN_L = 0xBA$
 $WM_WREN_H = 0xDC$
 Configure as Detector: $WM_DET[1:0] = 2'b01$
 Enable the Detector: $WM_EN = 1'b1$
 Set the threshold correctly: $WM_TH[6:0] = 0x0E$
 Enable WM timer: $WMtimer[7:0]$ – set to desired timer length. See [Filter the Watermark Error Condition](#).
 Lock registers: $WM_WREN_L = 0x00$
 $WM_WREN_H = 0x00$

9.2.4 System Implementation

It is recommended that the watermark generator and detector configuration be completed during initial systems configuration before the video stream is enabled. It should not be dynamically enabled/disabled during use.

9.2.5 Watermark Status Registers and Outputs

[Figure 28](#) illustrates the WM status output options. In the WM detection circuit, there are three live status registers: WM_ERROR , WM_ERR_FLAG , and WM_DETOUT .

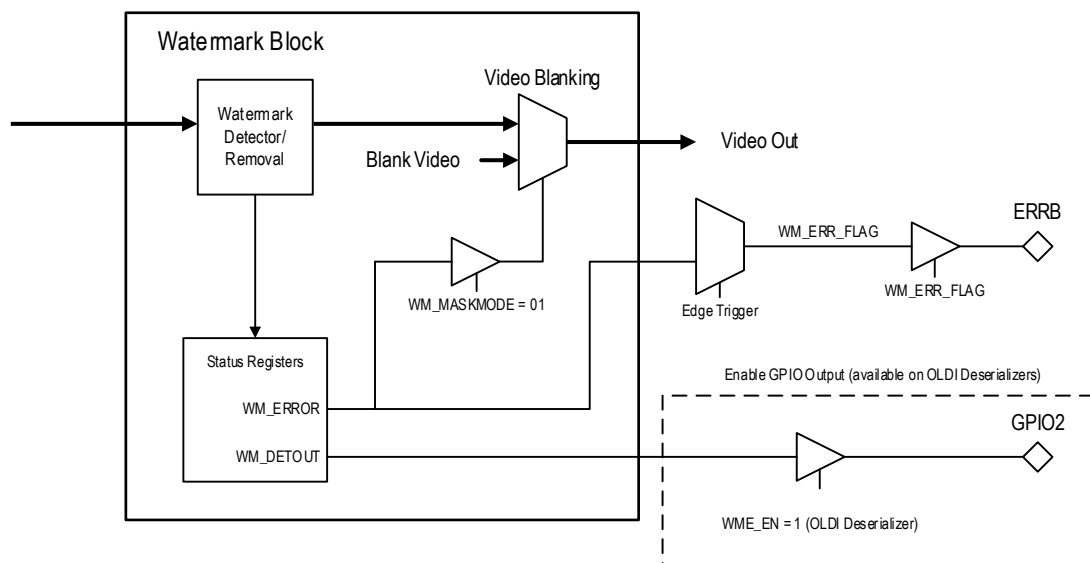


Figure 28. Watermark Status Outputs

9.3 Status and Debug Registers

The watermarking status can be monitored through the following status registers in the watermark detector.

- **WM_DETOUT**: This bit indicates if a watermark is detected. It is high if a watermark is detected in each frame and is low if the watermark is not detected. This stays high if there is a frozen frame condition because the watermark is still in the image. This bit status can be mapped out to a GPIO, if desired (eDP/DP and oLDI deserializers only). Watermarks can only be detected if valid video is flowing.
- **WM_ERROR**: This bit is a real-time flag indicating if a watermark error occurs. A watermark error is defined as a state in which all four unique watermarks are not detected in the previous eight frames. This differs from a frozen frame condition wherein the same watermark is repeated in every frame. The **WM_ERROR** register goes high if there is a watermark error and immediately returns low once the error condition no longer exists.
- **WM_ERR_FLAG**: This error flag is triggered on the initial occurrence (edge) of the **WM_ERROR** bit. The status of this error flag is output to the ERRB pin (if **WM_ERR_OEN** = 1). This flag is cleared on read even if the WM error condition still exists. If **WM_ERROR** returns low and then is triggered on a second WM error event, this flag is re-triggered.

9.3.1 ERRB Error Handling

The following is a suggested procedure for ERRB error handling:

- During nominal operation, use the ERRB output as an interrupt.
- When ERRB is active, check error registers (**INTR3**, **INTR5**, **INTR7**) to determine what error occurred.
- If **WM_ERROR** occurs, react appropriately for frozen frame error handling. An option to automatically blank video to black is available (see [Watermark Detector with Automatic Blank on WM ERR](#)).
- Monitor the **WM_ERROR** register until the frozen frame condition is fixed. The **WM_ERROR** register bit returns low when the frozen frame condition resolves.

Note: See the [GMSL2 Error Reporting \(ERRB Pin\)](#) section for additional information.

9.3.2 Status Bits Under Various Conditions

The state of registers for common conditions are listed below.

Default state

1. **WM_DETOUT** = 0
2. **WM_ERROR** = 0
3. **WM_ERR_FLAG** = 0
4. **ERRB** = 1

WM is enabled with no errors.

- **WM_DETOUT** = 1
- **WM_ERROR** = 0
- **WM_ERR_FLAG** = 0
- **ERRB** = 1

WM is disabled with no errors.

- WM_DETOUT = 0
- WM_ERROR = 0
- WM_ERR_FLAG = 0
- ERRB = 1

WM is enabled with lost video signal.

- WM_DETOUT = 0
- WM_ERROR = 0
- WM_ERR_FLAG = 0
- ERRB = 1

WM is enabled with WM error (frozen frame).

- WM_DETOUT = 1
- WM_ERROR = 1
- WM_ERR_FLAG = 1
- ERRB = 0

Error condition occurs (INTR5 is read).

- WM_DETOUT = 1
- WM_ERROR = 1
- WM_ERR_FLAG = 0
- ERRB = 1

Note: When an error condition occurs and register INTR5 is read, WM_ERR_FLAG is cleared on read, but WM_ERROR remains high.

Error condition resolves (INTR5 is read).

- WM_DETOUT = 1
- WM_ERROR = 0
- WM_ERR_FLAG = 0
- ERRB = 1

Note: When an error condition goes away, WM_ERR_FLAG is cleared on read and WM_ERROR returns low.

Error condition resolves then returns.

- WM_DETOUT = 1
- WM_ERROR = 1
- WM_ERR_FLAG = 1
- ERRB = 0

Note: When an error condition goes away and returns, WM_ERR_FLAG is re-triggered high until it is read again.

9.3.3 Watermark Generation/Detection Test

The following procedure is recommended to test watermark operation.

- Check normal operation.

- Configure the watermark generator and detectors, as desired, and check the **WM_DETOUT** and **WM_ERROR** status registers, as well as check that GPIO2 reflects the **WM_DETOUT** status as expected (if using **WM_DETOUT** output to GPIO2 mode).
- Check error handling.
 - Configure the watermark generator as described in [Watermark Generator \(with Error Generation\)](#) to replicate a frozen frame condition by only generation two watermark and ensure the system error handling is correct. The **WM_ERROR** register should return a value of 1, and depending on configuration of the ERRB pin, it should become active and/or the video output should be blanked.
- Check error recovery.
 - Configure the watermark generator back into normal.

9.3.4 Filter the Watermark Error Condition

The watermark detector can filter a watermark error condition. **WMtimer[7:0]** defines the time a watermark error condition must exist before an error is asserted and is configured in 2ms steps that the frozen frame condition must exist before an error is triggered. A value of 0 (default) disables this filtering.

9.3.5 Watermark Operation Troubleshooting

If watermark generation/detection is not functioning properly:

- Verify that video data is being routed properly. Ensure that **PCLKDET** = 1 in pipe X of the serializer and **VIDEO_LOCK** = 1 in pipe X of the deserializer.
- Confirm that HS is available to embed watermark. If there is no horizontal sync (MIPI serializer), the [Video Timing Generator \(VTG\)](#) must be configured to generate a horizontal sync signal.
- Ensure that sync signal polarity is correct. See the following section for how to determine proper sync polarity.

9.3.6 Sync Polarity in Watermarking Blocks

The watermark block uses the vertical and horizontal sync signals to determine row and frame boundaries. The watermark block requires that both the HS and VS polarities be positive to operate correctly. A polarity control can be used to invert the sync signal internal to the watermark generator and detector while preserving the original sync polarity outside the watermark blocks.

1. **VsyncPol**:
 - a. 0 (default): Retains sync signal polarity (use for positive sync polarity: 1'b1 during vertical blanking interval).
 - b. 1: Inverts sync signal polarity (use for negative sync polarity: 0 during vertical blanking interval).
2. **HsyncPol**:
 - a. 0 (default): Retains sync signal polarity (use for positive sync polarity: 1'b1 during vertical blanking interval).
 - b. 1: Inverts sync signal polarity (use for negative sync polarity: 0 during vertical blanking interval).

Note: Sync signal polarities are application-specific (example, if the VESA guidelines are followed or a custom scheme is used). Some customers define the video timing and sync polarity throughout the telematic system. The ANSI/CTA-861-F standard is a suggested reference resource.

9.3.6.1 Searching for the Correct Watermark Polarity

If the correct sync polarities are unknown, the watermark generator/detector can be used to determine the correct sync signal polarities. Configure the generator to output an error condition and the detector to blank on watermark error, then cycle through the 16 possible polarity configurations (in generator and detector). When the screen blanks, note the active configuration as this is the correct polarity settings.

Generator Settings

Unlock registers: `WM_WREN_L` = 0xBA
`WM_WREN_H` = 0xDC
 Enable the generator: `WM_EN` = 1'b1
 Enable ERR mode: `WM_MODE[2:0]` = 3'b001
 Lock registers: `WM_WREN_L` = 0x00
`WM_WREN_H` = 0x00

Detector Settings

Unlock registers: `WM_WREN_L` = 0xBA
`WM_WREN_H` = 0xDC
 Configure as Detector: `WM_DET[1:0]` = 2'b01
 Enable the Detector: `WM_EN` = 1'b1
 Set the threshold correctly: `WM_TH[6:0]` = 0x0E
 Enable Video blanking: `WM_MASKMODE[1:0]` = 2'b01
 Lock registers: `WM_WREN_L` = 0x00
`WM_WREN_H` = 0x00

Loop through sync signal polarity combinations. A blank screen indicates that the sync signal polarity combination is achieved.

9.4 Feature Availability by Device Family

Watermarking is available to all GMSL2 devices (except advanced CSI-2 serializers). See [Table 16](#) for feature availability and watermarking pipe.

Table 16. Watermarking Support by Device Family

Device Family	WM Generator/Detector	WM PIPE*
HDMI Serializers	X	Pipe X
CSI-2 Serializers	X	Pipe X
Advanced CSI-2 Serializers		
CSI-2 Camera Deserializers	X	Pipe X
oLDI Deserializers	X	Pipe X
CSI-2 Quad Deserializers	X (2)	Pipe 0 and Pipe 4

Note: See the [Video Pipes](#) section for additional details.

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

10.1 Overview

The video timing generator (VTG) feature allows users to manipulate video timing parameters. The VTG generates or adjusts video sync signals to modify the timing details of incoming video streams or create video timing to be used with the video pattern generator (VPG) function to generate test images. All GMSL2 serializers and the quad deserializers are equipped with these two features (see the [Feature Availability by Device Family](#) section). The VTG and VPG are implemented within the video pipes on both the serializers and deserializers.

The VTG provides user-configurable options for the video sync signals: vertical sync (VS), horizontal sync (HS), and data enable (DE). When 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 can generate user-defined RGB888 color patterns using the incoming PCLK and timing parameters sourced from either the input video or the VTG. These generated patterns can be used to test the GMSL link and downstream devices (example, displays) without a video source connected to the serializer. GMSL2 serializers output these patterns to deserializers as standard video over the GMSL link ([Figure 29](#)); the GMSL2 deserializers that support VPG can output test patterns in MIPI CSI-2 format directly. Two types of patterns (that is, color gradient and checkerboard) with configurable color parameters are available.

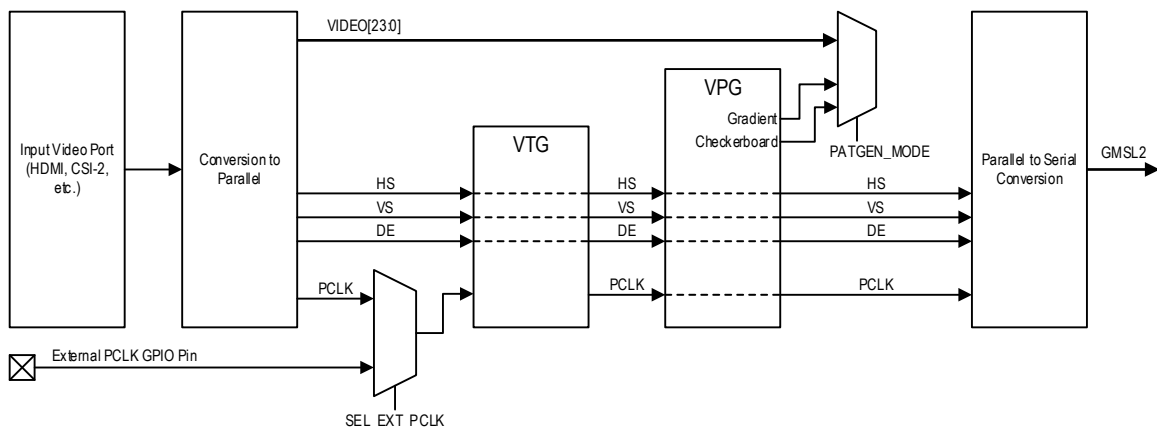


Figure 29. Video Path Through the VTG and VPG in GMSL2 Serializers

10.2 Video Timing Generator (VTG)

10.2.1 VTG Operation

The core function of the VTG block is to generate VS, HS, and DE signals based off a trigger. The VTG can be configured to generate these signals internally or use the VS/HS/DE signals at the input of the

VTG to drive the output signals (when disabled). This selection is made individually for each sync signal through the **GEN_VS**, **GEN_HS**, and **GEN_DE** register bits.

The VTG can be triggered by either a VS input transition (that is, the external trigger) or an internally generated VS trigger (that is, the tracking VS signal). In the case of the external trigger, the VS transition trigger is selected to be either the rising edge or 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 VTG consists of three pulse generators to generate the VS, HS, and DE signals. The polarity, start timing (that is, 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.

10.2.2 VTG Configuration

Configuring the VTG consists of two steps: selecting the VTG operation mode ([Table 17](#)) and configuring the timing parameters for the VS, HS, and DE generation ([Table 20](#)).

Table 17. VTG Operation Mode Configuration Registers

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	See VTG Trigger Modes section for information on following modes: 00: VS tracking mode 01: VS one-trigger mode 10: Auto-repeat mode 11: Free-running mode (default)
Select PCLK source	VTX1	SEL_EXT_PCLK	Selects the PCLK source for the VTG block. See PCLK Selection section for details. 0: Use the PCLK received from the video interface input 1: Use an external PCLK applied to a GPIO Note: If using external PCLK mode with CSI-2 or DSI serializers, see the PCLK Selection section for additional configuration information.
VS trigger polarity	VTX1	VS_TRIG	0: Falling edge 1: Rising edge (default)

Sync signal polarity	VTX0	VS_INV	<p>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.</p> <p>0: Do not invert VS signal 1: Invert VS signal</p> <p>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.</p>
Sync signal polarity	VTX0	HS_INV	<p>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.</p> <p>0: Do not invert HS signal 1: Invert HS signal</p> <p>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.</p>
Sync signal polarity	VTX0	DE_INV	<p>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.</p> <p>0: Do not invert DE signal 1: Invert DE signal</p> <p>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.</p>

10.2.2.1 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 (example, 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 is considered correct. The previous VS/HS/DE signals are cut or extended at the time point of the rising edge of the newly generated VS, HS, and DE signals.
- 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.

10.2.2.2 PCLK Selection

There are three options to provide a PCLK to the VTG block: using the clock from the video input, supplying an external PCLK through a GPIO, or using an internally generated PCLK (GMSL2 quad deserializers only).

The correct PCLK value is important for achieving the desired pattern resolutions and frame rates. See the [Video Basics](#) section for more information pertaining to calculating the PCLK value. Note that supplying an external PCLK through GPIO offers the most precise control of the PCLK value.

10.2.2.2.1 Clock from the Input Video Source

The PCLK is calculated differently depending on the video interface:

- **HDMI serializers in HDMI 1.4 mode**

$$PCLK = \text{HDMI Clock}$$

- **HDMI serializers in HDMI 2.0 mode**

$$PCLK = \frac{\text{HDMI Clock}}{4}$$

- **CSI-2 Serializers**

$$PCLK = \frac{\text{Data rate} * \text{Lanes}}{\text{bpp}}$$

Where data rate is double the MIPI clock rate, lanes is the number of active MIPI lanes, and bpp is bits per pixel (which is determined by the CSI-2 datatype).

10.2.2.2.2 External Clock through GPIO

HDMI serializers and CSI-2 serializers have a GPIO pin that can be used as a PCLK input pin. An external clock can be injected directly on this GPIO to trigger the VTG. This GPIO varies by device family, as given in the [Feature Availability by Device Family](#) section.

To use the GPIO input with a CSI-2 serializer, the device must be programmed into parallel mode so that the GPIO 0 pin becomes the PCLK input pin. With this configuration, only the pattern generator in video pipe X can be used. Enable parallel mode by setting bit 0 of register `PAR_VID_EN` in `REG7`.

In most cases, a camera serializer can output a reference clock from another GPIO to eliminate the need of an external clock source. The relevant registers for configuration are presented in [Table 18](#).

Table 18. Additional CSI-2 Serializer PCLK Register Settings

Register	Bits	Default Value	Description
REG7 (0x0007)	7:0	0xF6	Bit [7:4]: Parallel Data Input Pins: Parts dependent; See DS/Reg map for details. Bit 2: Parallel VS Input Pin: 0: Parallel VS Input disabled 1: Parallel VS Input enabled Bit 1: Parallel HS Input Pin: 0: Parallel HS Input disabled 1: Parallel HS Input enabled Bit 0: Parallel Video Input: 0: Parallel video disabled 1: Parallel video enabled
REF_VTG0 (0x03F0)	1	0	Bit 0: Reference PLL enable: 0: Disable reference generation PLL 1: Enable reference generation PLL
REF_VTG0 (0x03F0)	5:4	01	Bit [5:4]: Reference PLL frequency setting: 00: 19.2MHz 01: 27MHz 10: 37.125MHz 11: 74.25MHz
REF_VTG1 (0x03F1)	0	0	Bit 0: PCLKEN: 0: PCLK output disabled 1: PCLK output enabled on selected PCLK_GPIO
REF_VTG1 (0x03F1)	5:1	00000	Bit [5:1]: PCLK_GPIO: Select the reference clock output GPIO

10.2.2.2.3 Using an Internally Generated PCLK

GMSL2 quad deserializers have two clock configuration registers for setting the PCLK value for the video pipes. The video pattern PCLK frequency can optionally be set to 25MHz or 75MHz for all eight pipes or 150MHz or 375MHz on a per pipe basis. See video pattern PCLK settings registers ([Table 19](#)) for configuration details.

Table 19. Video Pattern PCLK Selection

Register Name (Reg #)		PCLK Frequency
DEBUG_EXTRA (0x09 [1:0])	PATGEN_CLK_SRC (bit 7)	
00	X	25MHz
01	X	75MHz
1x	0	150MHz
1x	1	375MHz

10.2.4 VTG Timing Parameters

The sync pulse timing parameters for the VTG are shown in [Figure 30](#) and described in [Table 20](#).

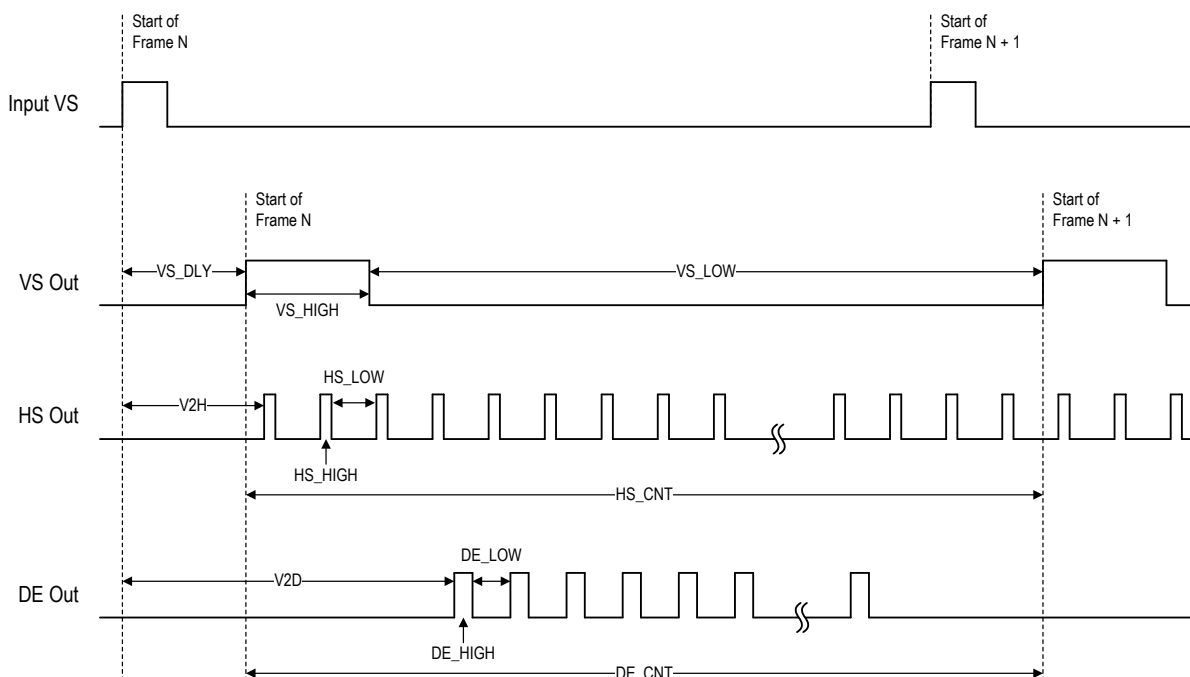


Figure 30. VTG Timing parameters

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 results in lost video data. Example, if DE is delayed four PCLKs, the first four pixels from the incoming data are lost and the last four are converted to zeros (that is, padded).

Table 20. 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 delays 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.

Note: For all VTG parameters that span multiple registers, the MSBs are programmed into the first register.

10.2.4.1 Programming for Standard Video Timings

The VTG should be configured to match the timing specification of the target application. Though, in many cases, it is preferred to use standard video timings. The video timings for standard resolutions are defined by the CTA-861 specification. The specification also defines the relationship of the sync signals and stipulates, in particular, that the active edge of VS and HS be synchronous. This definition is used in the remainder of this section; however, it is important to note that some displays and applications may define these timings differently.

For a standard video with timings, as defined in [Figure 31](#), the recommended values to program into the VTG are listed in [Table 21](#).

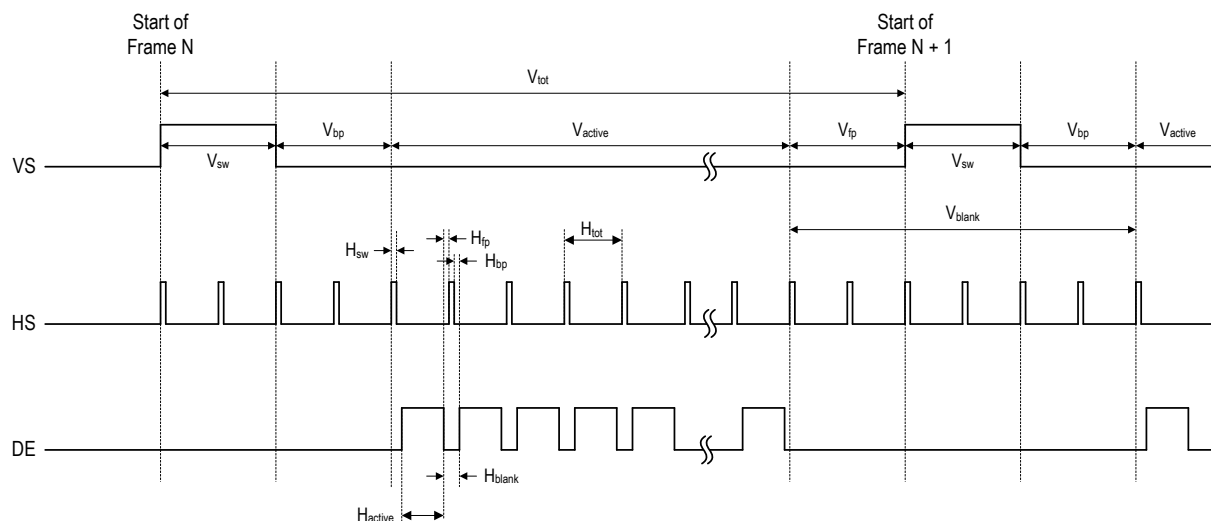


Figure 31. Video Timing Definition

[Figure 31](#) shows a vertical front porch, sync width, and back porch all equal to two lines. Note that the HS and VS rising edges are synchronous. [Figure 32](#) illustrates the relationship between the different components of the HS and DE signals.

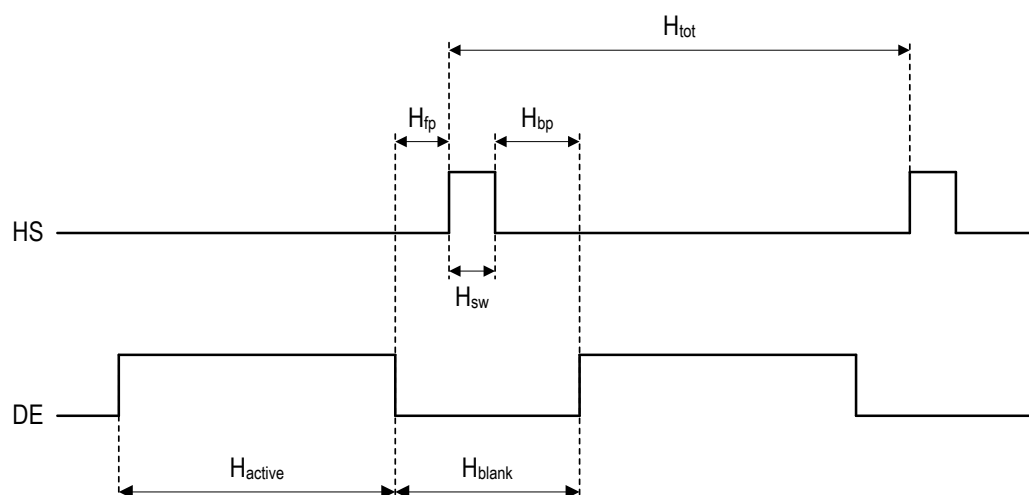


Figure 32. HS and DE Signal Components

Table 21. Recommended Register Settings for VTG Parameters (Standard Video Timings)

Parameter	Recommended Value	Recommended Value Description
VS_DLY	0	Setting VS delay causes the output frame to not be delayed with respect to the input trigger. If a delay is desired, program this register with the delay time terms of PCLK periods. A delay offsets the output sync pulse frame with respect to the input frame.
VS_HIGH	$V_{sw} \times H_{tot}$	Program to V_{sw} (multiplied by H_{tot} to convert from lines to PCLK cycles).
VS_LOW	$(V_{active} + V_{fp} + V_{bp}) \times H_{tot}$	Program to the remainder of the vertical period: $V_{tot} - V_{sw}$. This is equivalent to $V_{active} + V_{fp} + V_{bp}$ (multiplied by H_{tot} to convert from lines to PCLK cycles).
V2H	VS_DLY	This should be set equal to VS_DLY to align the rising edges of the generated VS and HS pulses (as specified in the CTA-861 specification). If VS_DLY is set to 0, V2H should also be set to 0.
HS_HIGH	H_{sw}	Program to H_{sw} .
HS_LOW	$H_{active} + H_{fp} + H_{bp}$	Program to the remainder of the horizontal period: $H_{tot} - H_{sw}$. This is equivalent to $H_{active} + H_{fp} + H_{bp}$.
HS_CNT	V_{tot}	The number of HS pulses needed is equal to the number of total lines (active plus blanking since HS are present through the vertical blanking period).
V2D	$VS_DLY + H_{tot} \times (V_{sw} + V_{bp}) + (H_{sw} + H_{bp})$	V2D defines the delay from the active edge of the VS trigger to the rising edge of the first DE signal in a frame. To calculate, begin by aligning with the start of the generated frame by adding the VS_DLY value. Then, add the vertical sync width and the vertical back porch time in terms of PCLK cycles ($V_{sw} + V_{bp}$ multiplied by H_{tot}). Since the first active line begins with a HS signal, add the horizontal sync width and the horizontal back porch. This sum is the delay to the first DE rising edge.
DE_HIGH	H_{active}	Program with H_{active} so that DE is the active line length.
DE_LOW	$H_{fp} + H_{sw} + H_{bp}$	Program with the horizontal blanking period (H_{blank}), which is equal to $H_{fp} + H_{sw} + H_{bp}$.
DE_CNT	V_{active}	The number of DE pulses needed in a frame is equal to the number of active lines (there are no DE pulses in the vertical blanking period).

10.2.4.2 VTG Programming Example

The following example script is for HDMI serializers and video with VESA-standard 1080p resolution with timing.

PCLK = 148.5MHz

$[H_{tot}, H_{active}, H_{fp}, H_{sw}, H_{bp}] = [2200, 1920, 88, 44, 148]$

$[V_{tot}, V_{active}, V_{fp}, V_{sw}, V_{bp}] = [1125, 1080, 4, 5, 36]$

```
#Enable the VTG for 1920x1080p video and do not modify the timing
0x80,0x01C8,0xE1
0x80,0x01C9,0xE1
0x80,0x01CA,0x00
0x80,0x01CB,0x00
0x80,0x01CC,0x00
0x80,0x01CD,0x00
0x80,0x01CE,0x2A
0x80,0x01CF,0xF8
0x80,0x01D0,0x25
0x80,0x01D1,0x99
0x80,0x01D2,0x00
0x80,0x01D3,0x00
0x80,0x01D4,0x00
0x80,0x01D5,0x00
0x80,0x01D6,0x00
0x80,0x01D7,0x2C
0x80,0x01D8,0x08
0x80,0x01D9,0x6C
0x80,0x01DA,0x08
0x80,0x01DB,0x98
0x80,0x01DC,0x01
0x80,0x01DD,0x61
0x80,0x01DE,0x18
0x80,0x01DF,0x07
0x80,0x01E0,0x80
0x80,0x01E1,0x01
0x80,0x01E2,0x18
0x80,0x01E3,0x04
0x80,0x01E4,0x38
```

10.3 Video Pattern Generator (VPG)

10.3.1 VPG Operation

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. Here, 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. Patterns generated by the quad CSI-2 deserializer can output MIPI CSI-2 data directly without any connection to the serial link.

Note: There is a GUI tool that can be used to configure the VTG and VPG.

10.3.3 VPG Modes of Operation

The VPG has two available patterns with configurable options: color gradient and checkerboard. The color gradient pattern ramps across the full 8-bit range (0x00–0xFF) for blue, green, red, and white (all colors simultaneously), as depicted in [Figure 33](#). The checkerboard pattern consists of two alternating colors ([Figure 34](#) shows blue and red); the colors and size of the pattern are configurable. Note that, since there is no frame memory, other patterns cannot be loaded into the GMSL devices.



Figure 33. VPG (Gradient Pattern)

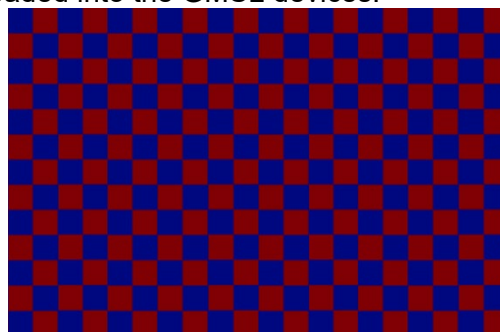


Figure 34. VPG (Checkerboard Pattern)

10.3.4 VPG Configuration

In gradient mode, the length of the gradient pattern and gradient direction are configurable. In checkerboard mode, two colors (that is, color A and color B) are selected and the size of the pattern is user-defined. The first video line starts with color A for the programmed number of pixels, then switches to color B for its programmed length. After color B, the pattern repeats. After a set number of lines, the pattern swaps, and outputs color B before color A. This process repeats for the height of the frame. The configuration registers for serializers are listed in [Table 22](#) and for quad deserializers in [Table 23](#). Operation is the same between devices; however, there are small differences in the bitfield names.

Table 22. VPG Configuration Registers for Serializers

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

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

Table 23. VPG Configuration Registers for Quad Deserializers

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_INCR[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_COLOR_A_L[7:0] CHKR_COLOR_A_M[7:0] CHKR_COLOR_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_COLOR_B_L[7:0] CHKR_COLOR_B_M[7:0] CHKR_COLOR_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.
--	-------	---------------	--

10.3.4.1 Application Setup of VPG

The following is a typical procedure to enable the VPG for serializers:

- Connect the deserializer and program it for the desired output format(s).
- Program the VTG timing details (that is, resolution, blanking, and frame rate) and calculate required PCLK frequency.
- Program the VPG pattern details (that is, pattern type and colors).
- Feed the PCLK and enable VTG/VPG through registers.

For additional information on configuring VPG in the quad deserializers, see the [Video Pattern Generator \(VPG\)](#) section in the quad deserializer section.

10.4 Feature Availability by Device Family

VTP and VPG support by device family is presented in [Table 24](#).

Table 24. VTG/VPG Support by Device Family

Device Family	VTG Available	External PCLK GPIO Pin	VPG Available
HDMI Serializers	Pipe X	GPIO 4	Pipe X
CSI-2 Serializers	All four video pipes	GPIO 0	All four video pipes
Advanced CSI-2 Serializers	All four video pipes	No	All four video pipes
CSI-2 Camera Deserializers	No	No	No
oLDI Deserializers	No	No	No
CSI-2 Quad Deserializers	All eight video pipes	No	Two available

Note: See the [CSI-2 Quad Deserializers](#) section for information regarding VTG and VPG implementation.

11. Video Crossbar

11.1 Overview

GMSL2 devices integrate video crossbar switches to reorder color and sync signals. This enables the serial link to be configured as a bridge between different input and output video interfaces and ensures wide format compatibility. Video bits can be selected to be inverted or forced to either 1 or 0 states. Video crossbar is used between standardized input and output video interfaces (example, CSI-2 and HDMI) that accept only one possible bitmapping format. It is also used for interfaces with non-standard bitmapping or interfaces with multiple bitmaps (example, parallel or LVDS). Select GMSL2 devices contain video crossbar switches to ensure compatibility even if the companion device does not (example, a GMSL1 device). Refer to the device data sheet for more details.

11.2 Operation

The GMSL2 video crossbar switches allow any input video bit to be mapped to any output video bit through register settings. It is also possible to configure one video crossbar input to multiple outputs. Each video crossbar output has dedicated bitmap configuration registers ([Table 25](#)).

Table 25. Video Crossbar Output Bitmapping Registers

Bitfield	Bit Position	Default Value	Description
CROSSx_I	6	0	Enable Crossbar Bit Inversion: Inverts this video bit when set to 1. If this bit and the force bit is enabled at the same time, the video bit is forced high.
CROSSx_F	5	0	Enable Crossbar Bit Forcing: Forces this video bit low when set to 1. If this bit and the invert bit is enabled at the same time, the video bit is forced high.
CROSSx_	[4:0]	See description	Crossbar Input Selection: Select the input bit to map to the output bit. By default, CROSS0 is mapped from input bit 0, CROSS1 is mapped from input bit 1, etc.

[Figure 35](#) illustrates the video crossbar block diagram for a single output bit.

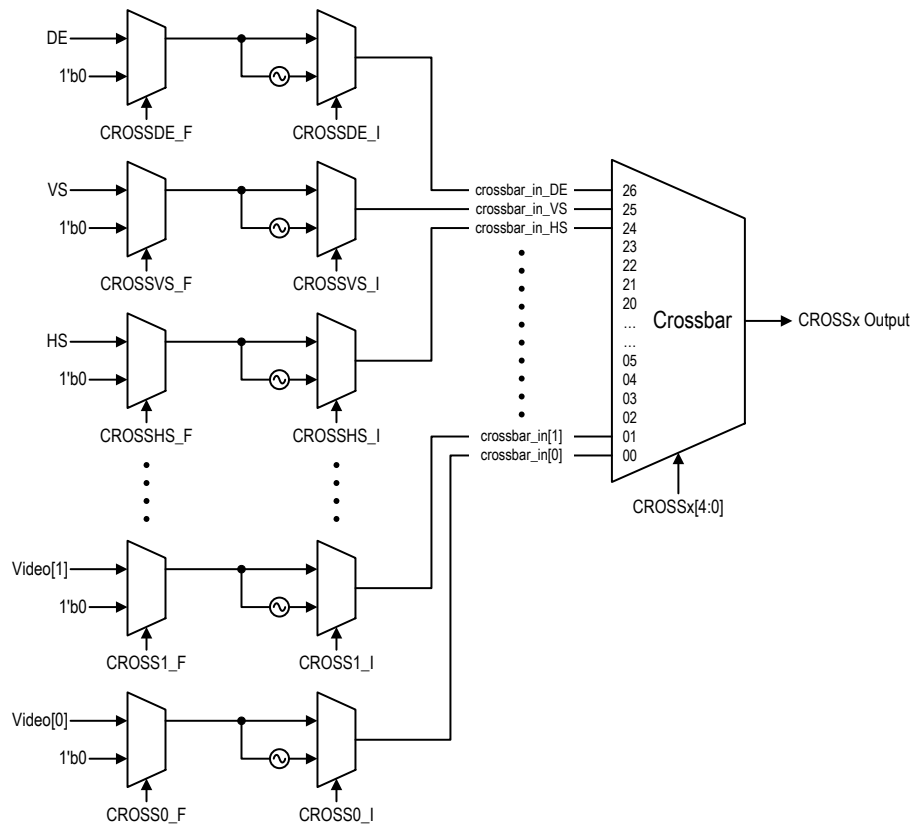


Figure 35. Video Crossbar Block Diagram (Single Output Bit)

11.3 Configuration

The video crossbar switches should be configured before initializing video on the serial link system. Ensure that the video source hardware, serial link, and display hardware are properly connected. Then, configure the video crossbar registers according to the colormaps of the input and output video formats of the application. In the video crossbar registers, confirm that the color bits are reordered and/or rerouted as needed and adhere to the relevant colormap tables. Start the video transmission, wait for video lock, and verify the functionality of the video crossbar. See the [Status and Debug Registers](#) section if undesired behavior is exhibited.

11.3.1 Video Crossbar Registers for Different Interfaces

Each video pipe has dedicated video crossbar switches with independent configuration registers. In cases with multiple video pipes, the video crossbar register names are appended with extensions (that is, X, Y, Z, and U) that correspond with the assigned video pipe.

The `CROSS_[23:0]` registers contain video data crossbar settings and are common to all video interfaces.

Interface-specific configuration information is provided in the following sections.

11.3.1.1 HDMI Input

The `VTX[42:40]` registers configure to the DE, VS, and HS signals, respectively. GMSL2 uses an active-high VS; for active-low resolutions (example, STMPE resolutions), invert the VS signal in the video crossbar switch.

11.3.1.2 CSI and Parallel Input

The [VTX\[42:40\]](#) registers configure the DE, VS, and HS signals, respectively.

11.3.1.2.1 CSI Mode

The start of a long packet is mapped to a DE rising edge, and the end of a long packet is mapped to a DE falling edge. The frame start packet is mapped to a VS rising edge, and the frame end packet is mapped to a VS falling edge. CSI-2 ignores and does not map HS signals. In DSI mode, however, the line start packet is mapped to a HS rising edge, and the line end packet is mapped to a HS falling edge.

11.3.1.2.2 Parallel Mode

Video sync signals typically consist of HS/VS/DE; however, some systems only send HS/VS (the 'Data Enable' signal is labelled as "HS" and the horizontal sync signal does not exist). In this case, set the DE crossbar switch to use the HS input. Configure the invert bit as needed to ensure that DE is fed an active-high signal. GMSL2 uses an active-high VS; if VS is active-low (example, STMPE resolutions), VS should be inverted in the crossbar switch.

11.3.1.3 oLDI Output

The [CROSS_\[27:24\]](#) registers configure the LVDS RES, DE, VS, and HS signals, respectively.

11.3.1.4 CSI-2 Output

The [CROSS_HS](#), [CROSS_VS](#), and [CROSS_DE](#) registers configure the respective video sync signals. The start of a long packet is mapped to a DE rising edge, and the end of a long packet is mapped to a DE falling edge. The frame start packet is mapped to a VS rising edge, and the frame end packet is mapped to a VS falling edge. CSI-2 ignores and does not map HS signals. The [CROSS_\[29:27\]](#) registers are only used in GMSL1 backward compatible mode.

11.3.2 Colormaps for Video Formats

Table 26. RGB Color Map

Input Bits	RGB888	RGB666	RGB565
DIN[0]	R[0]	R[0]	R[0]
DIN[1]	R[1]	R[1]	R[1]
DIN[2]	R[2]	R[2]	R[2]
DIN[3]	R[3]	R[3]	R[3]
DIN[4]	R[4]	R[4]	R[4]
DIN[5]	R[5]	R[5]	G[0]
DIN[6]	R[6]	G[0]	G[1]
DIN[7]	R[7]	G[1]	G[2]
DIN[8]	G[0]	G[2]	G[3]
DIN[9]	G[1]	G[3]	G[4]
DIN[10]	G[2]	G[4]	G[5]
DIN[11]	G[3]	G[5]	B[0]
DIN[12]	G[4]	B[0]	B[1]
DIN[13]	G[5]	B[1]	B[2]
DIN[14]	G[6]	B[2]	B[3]
DIN[15]	G[7]	B[3]	B[4]
DIN[16]	B[0]	B[4]	—
DIN[17]	B[1]	B[5]	—

DIN[18]	B[2]	—	—
DIN[19]	B[3]	—	—
DIN[20]	B[4]	—	—
DIN[21]	B[5]	—	—
DIN[22]	B[6]	—	—
DIN[23]	B[7]	—	—

Table 27. RAW Color Map

Input Bits	RAW (8bit)	RAW (8bit) DBL	RAW (10bit)	RAW (10bit) DBL	RAW (12bit)	RAW (12bit) DBL	RAW (14bit)	RAW (16bit)	RAW (20bit)
DIN[0]	P[0]	P[0]	P[0]	P[0]	P[0]	P[0]	P[0]	P[0]	P[0]
DIN[1]	P[1]	P[1]	P[1]	P[1]	P[1]	P[1]	P[1]	P[1]	P[1]
DIN[2]	P[2]	P[2]	P[2]	P[2]	P[2]	P[2]	P[2]	P[2]	P[2]
DIN[3]	P[3]	P[3]	P[3]	P[3]	P[3]	P[3]	P[3]	P[3]	P[3]
DIN[4]	P[4]	P[4]	P[4]	P[4]	P[4]	P[4]	P[4]	P[4]	P[4]
DIN[5]	P[5]	P[5]	P[5]	P[5]	P[5]	P[5]	P[5]	P[5]	P[5]
DIN[6]	P[6]	P[6]	P[6]	P[6]	P[6]	P[6]	P[6]	P[6]	P[6]
DIN[7]	P[7]	P[7]	P[7]	P[7]	P[7]	P[7]	P[7]	P[7]	P[7]
DIN[8]	—	P[0]	P[8]	P[8]	P[8]	P[8]	P[8]	P[8]	P[8]
DIN[9]	—	P[1]	P[9]	P[9]	P[9]	P[9]	P[9]	P[9]	P[9]
DIN[10]	—	P[2]	—	P[0]	P[10]	P[10]	P[10]	P[10]	P[10]
DIN[11]	—	P[3]	—	P[1]	P[11]	P[11]	P[11]	P[11]	P[11]
DIN[12]	—	P[4]	—	P[2]	—	P[0]	P[12]	P[12]	P[12]
DIN[13]	—	P[5]	—	P[3]	—	P[1]	P[13]	P[13]	P[13]
DIN[14]	—	P[6]	—	P[4]	—	P[2]	—	P[14]	P[14]
DIN[15]	—	P[7]	—	P[5]	—	P[3]	—	P[15]	P[15]
DIN[16]	—	—	—	P[6]	—	P[4]	—	—	P[16]
DIN[17]	—	—	—	P[7]	—	P[5]	—	—	P[17]
DIN[18]	—	—	—	P[8]	—	P[6]	—	—	P[18]
DIN[19]	—	—	—	P[9]	—	P[7]	—	—	P[19]
DIN[20]	—	—	—	—	—	P[8]	—	—	—
DIN[21]	—	—	—	—	—	P[9]	—	—	—
DIN[22]	—	—	—	—	—	P[10]	—	—	—
DIN[23]	—	—	—	—	—	P[11]	—	—	—

Time 0

Time 1

Table 28. YUV/YCrCB Color Map

Input Bits	YUV422 (8bit)	YUV422 (8bit) Mux=1	YUV422 (10bit)	YUV422 (10bit) Mux=1	YUV444 (8bit)	YUV422 (12bit)
DIN[0]	Cb/Cr [0]	YCb/YCr [0]	Cb/Cr [0]	YCb/YCr [0]	Cr [0]	Cb/Cr [0]
DIN[1]	Cb/Cr [1]	YCb/YCr [1]	Cb/Cr [1]	YCb/YCr [1]	Cr [1]	Cb/Cr [1]
DIN[2]	Cb/Cr [2]	YCb/YCr [2]	Cb/Cr [2]	YCb/YCr [2]	Cr [2]	Cb/Cr [2]
DIN[3]	Cb/Cr [3]	YCb/YCr [3]	Cb/Cr [3]	YCb/YCr [3]	Cr [3]	Cb/Cr [3]
DIN[4]	Cb/Cr [4]	YCb/YCr [4]	Cb/Cr [4]	YCb/YCr [4]	Cr [4]	Cb/Cr [4]
DIN[5]	Cb/Cr [5]	YCb/YCr [5]	Cb/Cr [5]	YCb/YCr [5]	Cr [5]	Cb/Cr [5]
DIN[6]	Cb/Cr [6]	YCb/YCr [6]	Cb/Cr [6]	YCb/YCr [6]	Cr [6]	Cb/Cr [6]
DIN[7]	Cb/Cr [7]	YCb/YCr [7]	Cb/Cr [7]	YCb/YCr [7]	Cr [7]	Cb/Cr [7]
DIN[8]	Y[0]	—	Cb/Cr [8]	YCb/YCr [8]	Y[0]	Cb/Cr [8]
DIN[9]	Y[1]	—	Cb/Cr [9]	YCb/YCr [9]	Y[1]	Cb/Cr [9]
DIN[10]	Y[2]	—	Y[0]	—	Y[2]	Cb/Cr [10]
DIN[11]	Y[3]	—	Y[1]	—	Y[3]	Cb/Cr [11]
DIN[12]	Y[4]	—	Y[2]	—	Y[4]	Y[0]
DIN[13]	Y[5]	—	Y[3]	—	Y[5]	Y[1]
DIN[14]	Y[6]	—	Y[4]	—	Y[6]	Y[2]
DIN[15]	Y[7]	—	Y[5]	—	Y[7]	Y[3]
DIN[16]	—	—	Y[6]	—	Cb [0]	Y[4]
DIN[17]	—	—	Y[7]	—	Cb [1]	Y[5]
DIN[18]	—	—	Y[8]	—	Cb [2]	Y[6]
DIN[19]	—	—	Y[9]	—	Cb [3]	Y[7]
DIN[20]	—	—	—	—	Cb [4]	Y[8]
DIN[21]	—	—	—	—	Cb [5]	Y[9]
DIN[22]	—	—	—	—	Cb [6]	Y[10]
DIN[23]	—	—	—	—	Cb [7]	Y[11]

Table 29. UDP8/Generic8 Color Map

Input Bits	UDP8 (8bit)	UDP8 (8bit)	UDP8 (8bit) DBL	Generic8 (8bit)	Generic8 (8bit)	Generic8 (8bit) DBL
DIN[0]	P[0]	P[0]	P[0]	P[0]	P[0]	P[0]
DIN[1]	P[1]	P[1]	P[1]	P[1]	P[1]	P[1]
DIN[2]	P[2]	P[2]	P[2]	P[2]	P[2]	P[2]
DIN[3]	P[3]	P[3]	P[3]	P[3]	P[3]	P[3]
DIN[4]	P[4]	P[4]	P[4]	P[4]	P[4]	P[4]
DIN[5]	P[5]	P[5]	P[5]	P[5]	P[5]	P[5]
DIN[6]	P[6]	P[6]	P[6]	P[6]	P[6]	P[6]
DIN[7]	P[7]	P[7]	P[7]	P[7]	P[7]	P[7]
DIN[8]	P[0]	—	P[0]	P[0]	—	P[0]
DIN[9]	P[1]	—	P[1]	P[1]	—	P[1]
DIN[10]	P[2]	—	P[2]	P[2]	—	P[2]
DIN[11]	P[3]	—	P[3]	P[3]	—	P[3]
DIN[12]	P[4]	—	P[4]	P[4]	—	P[4]
DIN[13]	P[5]	—	P[5]	P[5]	—	P[5]
DIN[14]	P[6]	—	P[6]	P[6]	—	P[6]
DIN[15]	P[7]	—	P[7]	P[7]	—	P[7]
DIN[16]	P[0]	—	—	P[0]	—	—
DIN[17]	P[1]	—	—	P[1]	—	—
DIN[18]	P[2]	—	—	P[2]	—	—
DIN[19]	P[3]	—	—	P[3]	—	—
DIN[20]	P[4]	—	—	P[4]	—	—
DIN[21]	P[5]	—	—	P[5]	—	—
DIN[22]	P[6]	—	—	P[6]	—	—
DIN[23]	P[7]	—	—	P[7]	—	—

Time 0
Time 1
Time 2

11.4 Status and Debug Registers

If the video crossbar is not functioning properly, perform the following troubleshooting process:

- Verify the physical connections in the system and check that the voltage levels on the input pins are within the expected range.
- Confirm that the video input is started and that the video stream is being transmitted.
- Verify that the GMSL link is locked and that the control channel (I²C/UART) is accessible.
- Ensure that the video stream is routed to the correct GMSL PHY using the `TX_SPLT_MASK_A/B` registers of the relevant video pipe.
- Verify that the video is locked on the deserializer side of the link.
- Ensure that the video input format is stable.

- Confirm that the video crossbar is correctly configured according to the desired output format.

11.5 Feature Availability by Device Family

Video crossbar is available in all GMSL2 devices except eDP/DP deserializers ([Table 30](#)).

Table 30. Video Crossbar Support by Device Family

Device Family	Video Crossbar
HDMI Serializers	X
CSI-2 Serializers	X
Advanced CSI-2 Serializers	X
CSI-2 Camera Deserializers	X
oLDI Deserializers	X
CSI-2 Quad Deserializers	X

12. Color Lookup Table (LUT)

12.1 Overview

All GMSL2 oLDI deserializers contain color lookup tables (LUT) that support automatic 1:1 translation of all RGB888 pixel color values to any desired RGB888 pixel color values. The LUTs are generally used for color filtering or gamma correction.

The color LUT enables 1:1 translation of 8-bit color input data to any 8-bit color output values. The color LUT comprises three discrete color channels that accept 8-bit wide input for each color (that is, red, green, and blue) with 256-entry depth. Each LUT RGB color sub-system has a separate configuration register block that can be programmed to automatically translate incoming pixel data. If color translation is needed, all three color LUTs must be configured and enabled.

No configuration of the color LUTs is required for a straight 1:1 mapping of the input data to output data. By default, the color LUTs are disabled and bypassed such that pixel values are not translated to other values. If pixel value translation is needed, the LUT registers should be programmed before enabling the color LUT block.

All oLDI deserializers have one video pipe and one LUT.

Figure 36 shows the LUT and LUT RGB sub-system architecture with incoming pixel data, RGB LUTs, and output pixel data. Note that each color can be user-programmed independently.

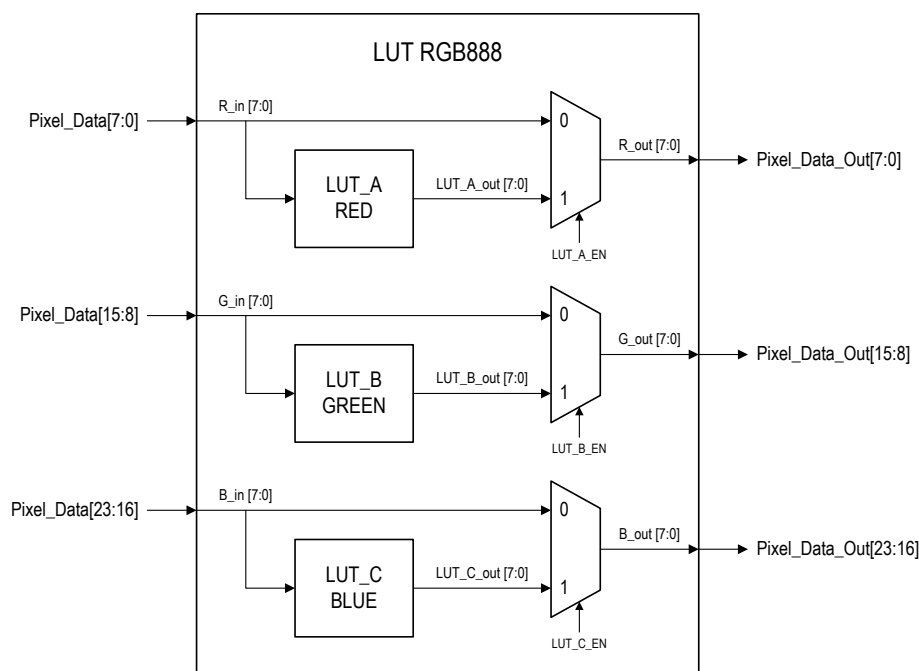


Figure 36. RGB Pixel Data Translation using Color Lookup Tables (LUT)

12.2 Configuration

At power-up, the LUTs are assigned random values but are not enabled. The LUTs should first be configured through the appropriate register space(s) before setting the register enable bit.

The beginning of the register address space for each LUT corresponds to the LSB of the respective color.

The LUT can be reverted to a 1:1 straight color mapping after the random value assignment at power-up. For example, use the following programming pattern to revert the red color LUT of the oLDI deserializer to 1:1 mapping values:

- 0x1000 = 0x00
- 0x1001 = 0x01
- 0x1002 = 0x02
- ...
- 0x10FF = 0xFF

Enable the LUT using the [LUT_x_EN](#) register bit after programming the LUT registers.

The following tables contain the LUT register information. These include the address range for each LUT and their corresponding enable bits. After configuring, the incoming pixel data automatically translates to the new output LUT values after enabling the LUT enable bit.

Note: LUTs cannot be partially enabled. If color translation is needed, all three LUTs must be enabled. See [Enabling LUTs](#) for more information.

12.2.1 oLDI Deserializers

Table 31. oLDI Deserializer LUT Registers

Field/Register Name	Address Range/Register Name	Bit Position	Default Value	Description
LUT_A (RED) register space	0x1000 – 0x10FF		Random	Address Range for RED LUT Data
LUT_B (GREEN) register space	0x1100 – 0x11FF		Random	Address Range for GREEN LUT Data
LUT_C (BLUE) register space	0x1200 – 0x12FF		Random	Address Range for BLUE LUT Data
LUT_A_EN	0x01CD / OLDI0	0	0: Disabled	Enables LUT_A (RED) translation
LUT_B_EN	0x01CD / OLDI0	1	0: Disabled	Enables LUT_B (GREEN) translation
LUT_C_EN	0x01CD / OLDI0	2	0: Disabled	Enables LUT_C (BLUE) translation

12.2.2 Readback Restriction

The following sequence is common in typical use cases:

- Program the LUT translation values.
- Readback the LUT translation values (to ensure proper programming).
- Enable the LUT.

LUT translation values do not readback correctly (example, 0xFF) in GMSL2 devices unless valid video data is received by the deserializer. The internal block is powered down until valid video data is received.

However, writes to the LUT are always valid (even when valid video data does not exist) if the I²C/UART ACK is received to confirm the register write. To confirm the LUT values by reading back through I²C/UART, valid video data is required.

12.2.3 Enabling LUTs

Although **LUT_A**, **LUT_B**, and **LUT_C** have separate enable registers in GMSL2 devices, the LUTs cannot be enabled partially. All LUTs (that is, **LUT_A**, **LUT_B**, and **LUT_C**) in a single video pipe must be enabled/disabled together for proper operation.

If the configuration of a single color LUT is required, but the others are not, configure the single LUT with the desired translation values and the remaining LUTs with 1:1 mapping values. For example, if only red color translation is needed, populate **LUT_A** with the desired translation values and fill green (**LUT_B**) and blue (**LUT_C**) with the linear 1:1 translation values.

12.3 Feature Availability by Device Family

Color lookup tables (LUT) are available on all GMSL2 oLDI deserializers ([Table 32](#)).

Table 32. Color Lookup Table Support by Device Family

Device Family	Color Lookup Tables (LUT)
HDMI Serializers	
CSI-2 Serializers	
Advanced CSI-2 Serializers	
CSI-2 Camera Deserializers	
oLDI Deserializers	X
CSI-2 Quad Deserializers	

13. Frame Sync

13.1 Overview

Certain GMSL applications may require the aggregation of multiple video sources/inputs into a single MIPI output. The packet-based MIPI protocol supports video aggregation through merging separate streams on a first come, first serve (FCFS) basis. The SoC can process the merged stream by allocating the constituent streams to different virtual channels. If the application requires that the video streams be time-synchronized for advanced processing, frame sync signals (FSYNC) are used to enable video stream synchronization and alignment. Here, the SoC sends FSYNC signals to synchronize video frames at the video source(s). FSYNC signals allow video frames to be processed simultaneously and provide the option to concatenate multiple video streams into a single synchronized video frame.

13.2 Operation

Compatible GMSL2 CSI-2 deserializers (see [Feature Availability by Device Family](#)) feature an internal frame sync (FSYNC) generator block and/or pin that accepts external FSYNC generation (example, from the SoC) with low-latency GPI-to-GPO tunneling. These devices interface with image sensors with an external FSYNC input.

The FSYNC signal is generated periodically or pseudo-periodically. The frame length is defined by the number of pixels in each frame or by referencing the 25MHz crystal clock (eliminating dependence on the image sensor PCLK). After generation, the FSYNC signal is transmitted through the reverse channel on the serial link to connected serializers and, subsequently, image sensors to synchronize the captured video frames.

Video frame synchronization occurs by synchronizing the vertical sync (VS) signals of the various video streams at the image sensors ([Figure 37](#) and [Figure 38](#)). This initial synchronization prepares the video streams for processing by the SoC when output from the serial link system.

Frame synchronization permits the deserializer to concatenate the received video streams. With concatenation, the deserializer can output video frames in line-concatenated (4WxH) or line-interleaved (Wx4H) formats.

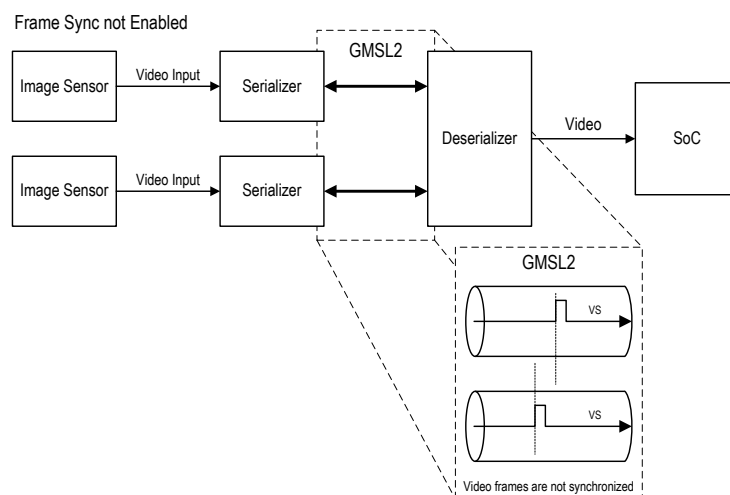


Figure 37. VS Signal Path (Frame Sync not Enabled)

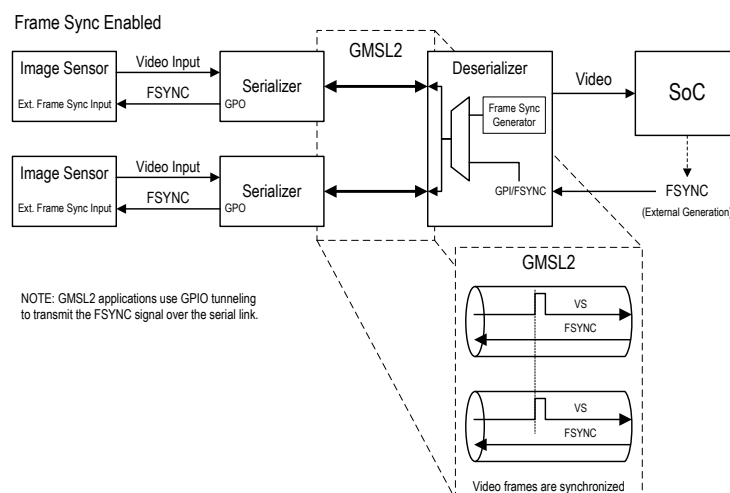


Figure 38. VS Signal Path (Frame Sync Enabled)

Frame sync is typically used in serial link systems featuring a single deserializer connected to multiple serializers; in extended systems with multiple image sensor arrays, two or more deserializers may be required. Here, the frame sync signal must be coordinated between multiple deserializers and all connected image sensors. One deserializer is configured as a main for frame sync generation and the other(s) are configured as frame sync subordinate(s). After the main deserializer generates the frame sync signal, it is routed to the subordinate deserializer(s); all deserializers send the frame sync signal to the connected serializers and image sensors (as described above).

Note that the frame sync signal pattern differs in GMSL1 mode. In GMSL2 mode, (subordinate) deserializers transmit the frame sync signal as a standard GPIO signal through GPI-GPO tunnel(s). In GMSL1 mode, there is only one shared GPI-GPO tunnel used exclusively by the primary 'Control Channel' or for GPI-GPO tunneling (or frame sync) at a time. When a frame sync signal is transmitted in GMSL1 mode, I²C is temporarily paused by clock stretching. In GMSL2 mode, there are no restrictions on frame sync transmissions nor any adverse effects on other channels.

13.3 Configuration

Frame sync applications require that all devices in the serial link system be compatible with frame sync. For compatible systems, the GMSL2 deserializer must be configured before frame sync can be enabled.

1. Determine the *Frame Sync Mode* required for the application and program the **FSYNC_MODE** register ([Table 33](#)).
2. Configure the *Frame Sync Method* (**FSYNC_METH**) to set the method FSYNC signal generation ([Table 66](#)).
3. GMSL2 devices: Determine and configure the GPIO ID in both the deserializer and serializer.

The connected image sensor(s) must be able to accept external FSYNC signals. Configure the image sensor before proceeding with GMSL2 device configuration.

13.3.1 Frame Sync Mode

The topology of the serial link system determines the frame sync mode of operation that should be programmed. There are four possible modes selected by the **FSYNC_MODE** register in the deserializer:

Table 33. Frame Sync Mode (FSYNC_MODE)

Value	Decode
00	Frame sync generation is on (internal frame sync mode). GPIO is not used as FSYNC input or output.
01	Frame sync generation is on (internal frame sync mode). GPIO is used as FSYNC output and drives a subordinate device.
10	Frame sync generation is off. In GMSL1 mode, GPIO is used as FSYNC input driven by a main device.
11	Frame sync generation is off. In GMSL1 mode, GPIO is not used as FSYNC input or output.

A typical frame sync application is in a serial link system comprising multiple serializers connected to one deserializer. In this scenario, the deserializer can either generate the FSYNC signal (enabled by programming register **FSYNC_MODE** = 00) or receive the FSYNC signal from an external source (**FSYNC_MODE** = 10). Any of the FSYNC methods may be used, depending on application (see *Frame Sync Method* for more information).

If multiple deserializers are used in the system, one can be configured as a frame sync main for the other connected subordinate deserializers. Program register **FSYNC_MODE** = 01 in the main deserializer to output the FSYNC signal from an MFP pin; then, program register **FSYNC_MODE** = 10 in the subordinate deserializer(s) to receive the FSYNC signal on a GPI.

If frame sync is not required, set **FSYNC_MODE** = 11 in the device.

The frame sync applications are similar for both GMSL2 and GMSL1 modes. However, there are differences. For details, see [Table 34](#) for GMSL2 applications and [Table 35](#) for GMSL1 applications. The **FSYNC_METH** register is explained in the *Frame Sync Method* section.

Note: Refer to device data sheets for specific register details.

Table 34. Frame Sync Modes and Methods (GMSL2)

FSYNC_MODE	FSYNC_METH	GMSL2 Mode Explanation	GMSL2 Usage
00	Manual; Auto; Semi-Auto	Frame sync generation is ON. The deserializer GPIO is NOT used as an output for frame sync.	1. Internal frame sync 2. No sync signal output
01	Manual; Auto; Semi-Auto	Frame sync generation is ON. The deserializer GPIO used as an output for frame sync.	1. Internal frame sync 2. Sync signal output at FRSYNC_OUT
10	N/A	Frame sync generation is off. GPIO is not used as FSYNC output.	1. No internal frame sync 2. May use GPIO forwarding for external frame sync
11	N/A	Frame sync generation is off. GPIO is not used as FSYNC output.	

Note: Quad deserializer registers may differ than those in [Table 34](#). See the quad deserializer frame synchronization section for additional details.

Table 35. Frame Sync Modes and Methods (GMSL1)

FSYNC_MODE	FSYNC_METH	GMSL1 Mode Explanation	GMSL1 Usage
00	Manual; Auto; Semi-Auto	Frame sync generation is ON. The frame sync pin is in HIGH-Z mode.	1. Internal frame sync 2. No sync signal output
01	Manual; Auto; Semi-Auto	Frame sync generation is ON. The deserializer GPIO is NOT used as an output for frame sync.	1. Internal frame sync 2. Sync signal output at FRSYNC_OUT (MFP0 for GMSL2 deserializer)
10	N/A	Frame sync generation is off. FSYNC pin is driven by a main device.	1. No internal frame sync 2. Device is configured as a subordinate device to receive frame sync signal from external main device. 3. The frame sync signal input from FRSYNC_IN, which is controlled by FSYNC_OUT_PIN.
11	N/A	Frame sync generation is off. GPIO is not used as FSYNC output.	1. No internal frame sync 2. GPI-to-GPO tunnel can be used to transmit external frame sync signal to serializer. 3. Use GPI_SEL to choose GPI_x; GPI_COMP_EN needs to be turned on if PKTCC_EN is on.

In GMSL1 mode, **FSYNC_MODE** = 10 may be used to configure the deserializer in subordinate mode to receive the frame sync signal from another deserializer (configured as a main: **FSYNC_MODE** = 01).

13.3.2 Frame Sync Method

When the frame sync generator is enabled, the `FSYNC_METH` register is used to set the method of frame sync generation. The frame sync generator has three methods of operation: manual, auto, and semi-auto. When the deserializer receives an external FSYNC signal, the `FSYNC_METH` register setting has no effect.

Table 36. FSYNC Methods of Operation (`FSYNC_METH`)

<code>FSYNC_METH</code>	Decode
0b00	Manual: User-defined clock periods for FSYNC generation.
0b01	Semi-Auto: FSYNC period is automatically set; use this method when only the row counter is reset.
0b10	Auto: FSYNC period is automatically set; use this method when both the row and column counters are reset.
0b11	Reserved

13.3.2.1 Overlap Window

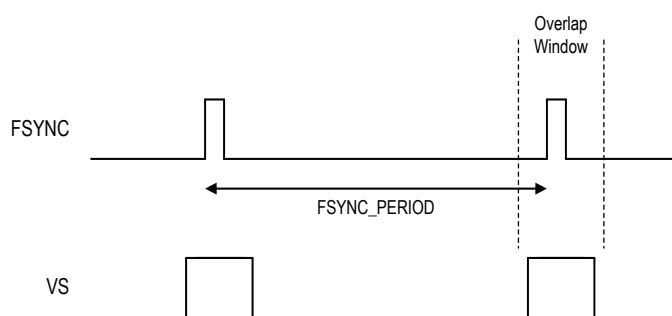


Figure 39. Overlap Window (`FSYNC` and `VS` signals)

The overlap window defines a threshold that starts from the `VS` rising edge of the fastest image sensor and ends when a `FSYNC` signal is received (Figure 39). It is measured in terms of `PCLK` cycles with the `OVLW_WINDOW_L` and `OVLW_WINDOW_H` bitfields. The window is used to check that the `FSYNC` signal is successfully sent from the deserializer to the image sensor and the corresponding `VS` signal is successfully sent from the image sensor to the deserializer. The overlap window should be programmed according to calculations considering the maximum frame synchronization skew and the maximum frequency difference (ppm) of all camera links.

Note: The overlap window may be disabled if there is an image signal processor (ISP) between the image sensor and serializer. The ISP may delay frame generation and cause a failure in the overlap protection.

13.3.2.2 Manual Frame Synchronization

Manual mode is the simplest (and most common) frame sync method. This method uses fixed clock periods defined by `FSYNC_PERIOD` register for frame sync signal generation. The `FS_USE_XTAL` register is used to specify whether the `PCLK` or the crystal oscillator (GMSL2 mode only) is used as the clock source. See the [Crystal Clock Option](#) section for additional information.

13.3.2.2.1 Crystal Clock Option

GMSL2 devices have an integrated crystal clock source that can be used as an alternative to the PCLK for frame sync generation. In manual mode, a 25MHz clock from the crystal oscillator can be used to generate the frame sync signal. Set `FS_USE_XTAL=1` to periodically generate the frame sync signal using the 25MHz clock as the base period. Set `FS_USE_XTAL=0` if using the PCLK.

13.3.2.3 Automatic Frame Synchronization

Automatic mode is recommended for use in systems with image sensors that run in continuous mode and reset both row and column counters (in the image sensor) upon receipt of the frame sync signal (example, OV10635). The `FSYNC_PERIOD` register is not user-configured in this mode.

When automatic frame synchronization is enabled, the deserializer detects and analyzes free-running VS signals on the active GMSL links. After determining the slowest VS pulse and allowing for timing margin adjustments, the deserializer sends the FSYNC signal on the reverse channel. Frame synchronization is locked when all received VS signals are synchronized for two consecutive cycles.

1. Automatic frame synchronization requires that the initial VS signals of each link be free running. The main link frame length (FL) is counted in terms of PCLK cycles until the deserializer detects VS signals from all connected and enabled GMSL links.
 - Note: The main link is either the first link that locks or defined by the user.
2. The first FSYNC signal is sent on the reverse channel. The row and column counters of each connected image sensor are reset to align the start of each link's subsequent frame.
3. The aligned VS signals are received by the deserializer. These signals arrive at slightly different times due to differences on each link (that is, clock frequency offset and transmission timing skew). The difference between the main link and slowest link is calculated in terms of *master pixel clock* – t_{DIFF} .
4. The deserializer dynamically adjusts each link's FSYNC signal generation to compensate for link differences and ensure that the frames arrive at the deserializer synchronized.
 - Loss of lock occurs if the difference between the arrival of the earlier and later signal exceeds the threshold.

13.3.2.4 Semi-Automatic Frame Synchronization

Semi-automatic mode is recommended for use in systems with image sensors that reset only the row counter upon receipt of the frame sync signal (example, OV10640).

This mode is similar to [Automatic Frame Synchronization](#). However, in semi-automatic mode, the deserializer does not check the skew between the different GMSL links (t_{DIFF}); adjustments of timing margins are instead measured from the main link. The FSYNC signal resets only the row counter in the image sensors.

1. Semi-automatic frame synchronization requires that the initial VS signals of each link be free running. The main link frame length (FL) is counted in terms of PCLK cycles until the deserializer detects VS signals from all connected and enabled GMSL links.
 - a. The main link is either the first link that locks or defined by the user.
2. The first FSYNC signal is sent on the reverse channel to align the VS signals of connected image sensors. This reset causes the subsequent frame starts of the connected image sensors to be within two rows of each other.
3. The aligned VS signals are received by the deserializer. Note that these signals arrive at slightly different times due to differences on each link (that is, clock frequency offset and transmission timing skew) and that only the rows (and not the columns) are reset.
4. The deserializer dynamically adjusts FSYNC signal generation based on timing measurements of the main link to ensure that the frames arrive at the deserializer synchronized.
 - a. Loss of lock occurs if the difference between the arrival of the earlier and later signal exceeds the threshold.

13.3.3 Status and Error Bits

The frame sync locking mechanism and associated registers are the same for all three frame synchronization methods. The frame sync lock bit ([FSYNC_LOCKED](#)) indicates that frame sync has successfully locked. [FSYNC_LOSS_OF_LOCK](#) goes high if frame sync lock is lost. Other important registers for the locking mechanism include the [FSYNC_LOSS_OF_LOCK](#), [FRM_DIFF_ERR_THR](#), [FSYNC_ERR_CNT](#), and [FSYNC_ERR_THR](#) register bits.

1. [CALC_FRM_LEN](#): The calculated VS period (number of PCLKs) of main link in automatic or semi-automatic synchronization mode.
2. [FRM_DIFF](#): The difference between the fastest and slowest frame in terms of main PCLK cycles
3. [FRM_DIFF_ERR_THR](#): Error threshold for difference between the earliest and latest VS signals in terms of PCLK cycles. The default is 40μs for a 96MHz PCLK. The function is disabled when all 13 bits are 0.
4. [FSYNC_ERR_CNT](#): Frame sync error counter. Incremented when error condition is detected and flagged. Resets to 0 when read or when [FSYNC_LOCKED](#) goes high.
5. [FSYNC_ERR_THR](#): Frame sync error reporting threshold. [FSYNC_ERR_FLAG](#) is asserted when [FSYNC_ERR_CNT](#) is greater than [FSYNC_ERR_THR](#).

The frame sync error status ([FSYNC_ERR_FLAG](#)) can be configured to drive the ERRB pin ([FSYNC_ERR_OEN](#) = 1).

The deserializer checks that enabled links have valid VS signals and that the VS signals occur within the overlap window. The overlap window is programmed through the [OVLW_WINDOW](#) bits. The overlap window may be disabled if there is an image signal processor (ISP) between the image sensor and serializer. The ISP may delay frame generation and cause a failure in the overlap protection.

13.3.4 Configuration Process

13.3.4.1 GMSL2 Deserializer in GMSL2 Mode

The following process is used to enable frame sync for a GMSL2 deserializer in the GMSL2 mode:

- Write `AUTO_FS_LINKS` high (default) to enable FSYNC on all enabled links. Alternatively, the bits `FS_EN_X/Y/Z/U` can be used to enable FSYNC on video pipes individually.
- Disable all unused video controllers in the deserializer. Unused video controllers can be identified by the absence of `VIDEO_LOCK`. For example, if `VIDEO_LOCK` is only indicated for controllers “X” and “Y”, then controllers “Z” and “U” must be disabled. Video controller configuration registers are in `REG2`.
- Choose the proper main link for the frame sync generation through `FSYNC_2`. This can be any video controller with `VIDEO_LOCK`.
 - Note: If a main link is not designated, then the frame sync block defaults the main link designation to the first link to achieve link lock.
- Configure the GPIO TX ID for frame sync so that the FSYNC pulse is output to the correct GPIO pin of the serializer. This is configured in the `FSYNC_17` register.
- Set up the GPIO RX ID for the GPIO pin on the serializer that is to receive the FSYNC signal. Ensure that the deserializer output pin and serializer input pin are correctly mapped. See the [General-Purpose Input and Output \(GPIO\)](#) section for more information.
- Set the *Frame Sync Mode* and *Frame Sync Method*. Relevant registers are in `FSYNC_0`.
- If necessary, set the overlap window (`OVLP_WINDOW`).
 - `OVLP_WINDOW`: Defines a threshold that starts from the VS rising edge of fastest image sensor and ends when a frame sync signal is seen. Measured in pixel clocks cycles.
 - `OVLP_WINDOW_L` in register `FSYNC_10`
 - `OVLP_WINDOW_H` in register `FSYNC_11`
 - Note that `OVLP_WINDOW` may be disabled if there is an ISP between the image sensor and serializer. The ISP may delay the frame generation and overlap protection fails.

13.3.4.2 GMSL2 Deserializer in GMSL1 Mode

GMSL2 deserializers configured for backward compatible GMSL1 mode can use frame sync. Operation in GMSL1 mode is similar to operation in GMSL2 mode (see [Table 65](#)). Note:

1. Link A is in Pipe X and Link B is in Pipe Y.
2. For GMSL2 deserializer, Link A and Link B have discrete GMSL1 blocks containing separate GMSL1 registers for each.
3. The `EN_FSYNC_TX` register must be set to use the frame sync signal.
4. Register `GPI_EN` is used to enable GPI-to-GPO transmission (default is on). Ensure that the MFP is available for GPI usage.
5. `GPI_COMP_EN` must be enabled if `PCKTCC_EN` is on and GPI-to-GPO is used.

13.4 Feature Availability by Device Family

Frame sync is available on select GMSL2 deserializers. See [Table 37](#) for part numbers with frame sync capabilities.

Note: All GMSL1 and GMSL2 camera serializers are compatible with FSYNC serial link applications.

Table 37. Frame Sync Support by Device Family

Device Family	Frame Sync
HDMI Serializers	
CSI-2 Serializers	
Advanced CSI-2 Serializers	
CSI-2 Dual Deserializers	X
oLDI Deserializers	
CSI-2 Quad Deserializers	X

14. Video Sync Pulse Outputs

14.1 Overview

GMSL2 serializers and deserializers can output video sync signals (that is, VS, HS, and DE) to aid in system-level debugging. The polarity of these signals can be flipped to accommodate requirements of video formatting and connected devices. The sync signal timing can also be modified with the video timing generator (VTG). See the [Video Timing Generator \(VTG\)](#) and [Video Pattern Generator \(VPG\)](#) section for additional information.

14.2 Operation

Videos may use either active-high or active-low VS and HS signals. The polarity of HS, VS, and DE signals are individually configurable with GMSL2 devices. This provides flexibility in system design and ensures compatibility with different cameras and displays.

In serializers, the sync signal polarity can be inverted using the following bits in the **VTX** register block:

- **VS_INV** – Inverts the VS pulse.
- **HS_INV** – Inverts the HS pulse.
- **DE_INV** – Inverts the DE pulse.

In deserializers, the sync signal polarity can be inverted using the following bits in the **VRX** register block:

- **CROSS_VS_I** – Inverts the VS pulse.
- **CROSS_HS_I** – Inverts the HS pulse.
- **CROSS_DE_I** – Inverts the DE pulse.

14.3 Configuration

HS, VS, or DE signals can be routed to any available MFP/GPIO pin. To configure sync signal MFP/GPIO output, first ensure that the alternate functions of MFP(s) are disabled and GPIO output is enabled. For a particular MFP/GPIO, set the following:

- **GPIO_RX_EN** = 0
- **GPIO_TX_EN** = 0
- **GPIO_OUT_DIS** = 0
- **PULL_UPDN_SEL** = 00
- **OUT_TYPE** = 1

The pins and register configuration used for sync signal output vary by device family. See the following tables ([Table 38](#) to [Table 43](#)) for details for each device family.

Table 38. HDMI Serializers

Sync Signal	Sync Output Available	Sync Output Pin	Sync Output Configuration
VS	Yes	GPIO17	Register: VTX1 Bit: VS_OUT_EN
HS	Yes	GPIO16	Register: VTX1 Bit: HS_OUT_EN
DE	No	N/A	N/A

Table 39. Advanced CSI-2 Serializers

Sync Signal	Sync Output Available	Sync Output Pin	Sync Output Configuration
VS	No	N/A	N/A
HS	No	N/A	N/A
DE	No	N/A	N/A

Table 40. CSI-2 Serializers

Sync Signal	Sync Output Available	Sync Output Pin	Sync Output Configuration
VS, HS, DE from CSI-2 Input	No	N/A	N/A
VS from REF_VTG block	Yes	Selectable	Use REF_VTG block to set the sync high and low period. Set REFGEN_EN=1 Set GEN_VS=1 Select which GPIO to output with VS_GPIO[4:0] Set VSEN = 1 to enable the output
HS from REF_VTG block	Yes	Selectable	Use REF_VTG block to set the sync high and low period. Set REFGEN_EN=1 Set GEN_HS=1 Select which GPIO to output with HS_GPIO[4:0] Set HSEN = 1 to enable the output
PCLK from REF_VTG block	Yes	Selectable	Set REFGEN_EN=1 Select predefined frequency in PLL (REFGEN_PREDEF_FREQ [1:0]) or RCLK frequency (RCLKEN_Y) = 1 for desired PCLK frequency. Select which GPIO to output with PCLK_GPIO[4:0] Set PCLKEN = 1 to enable the output

Note: If PCLK, HS, and VS signals are mapped to the same pin, the priority from high to low is HS, VS, and PCLK. See the [CSI-2 Serializer Sync Pulse Configuration](#) section for setting high and low periods for sync signals.

Table 41. CSI-2 Camera Deserializers

Sync Signal	Sync Output Available	Sync Output Pin	Sync Output Configuration
VS	Yes	MFP3	VS_OUT1[2] enables VS output to MFP3 VS_OUT1[1:0] selects which video pipe to output
VS	Yes	MFP0	VS_OUT2[2] enables VS output to MFP0 VS_OUT2[1:0] selects which video pipe to output
HS	No	No	N/A
DE	Yes	MFP0/MFP3	HS_OUT1[2] enables DE output to MFP0 and MFP3 HS_OUT1[1:0] selects which video pipe to output

Note: If DE and VS signals are mapped to the same pin, VS signal has higher priority.

Table 42. oLDI Deserializers

Sync Signal	Sync Output Available	Sync Output Pin	Sync Output Configuration
VS	Yes	GPIO1	Register: OLDI2 Bit: VS_OUT_EN
HS	Yes	GPIO1	Can be output to GPIO1 by swapping HS and VS with the video crossbar. Not available at the same time as VS output. Set CROSS25[4:0] = 0x18 and CROSS24[4:0] = 0x19
DE	No	N/A	Can be output to GPIO1 by swapping DE and VS with the video crossbar. Not available at the same time as VS output. Set CROSS25[4:0] = 0x1A and CROSS26[4:0] = 0x19

Note: The register HS_OUT_EN is reserved.

Table 43. CSI-2 Quad Deserializers

Sync Signal	Sync Output Available	Sync Output Pin	Sync Output Configuration
VS	Yes	MFP4	Register: HVD_GPIO_CTRL Bit: VS_EN enables VS output to MFP4 Bit: HVD_SEL[2:0] selects which video pipe to output
HS	Yes	MFP13	Register: HVD_GPIO_CTRL Bit: HS_EN enables HS output to MFP13 Bit: HVD_SEL[2:0] selects which video pipe to output
DE	Yes	MFP14	Register: HVD_GPIO_CTRL Bit: DE_EN enables DE output to MFP14 Bit: HVD_SEL[2:0] selects which video pipe to output

Note: Set register the DIS_LOC_CC_P2 = 1 to output DE and VS. HS output functionality requires either that the connected serializer supports HS (example, parallel input) or that the HS signal is internally generated in the serializer (example, pattern generation).

14.3.1 CSI-2 Serializer Sync Pulse Configuration

CSI-2 serializers can use the REF_VTG block to output sync signals from MFP pins. The REF_VTG block uses the user-programmable DPLL for PCLK generation.

There are two options for PCLK generation:

1. Predefined frequencies:
REFGEN_PREDEF_EN = 1

REFGEN_PREDEF_FREQ[1:0]	PREDEF_FREQ_ALT = 0	PREDEF_FREQ_ALT = 1
00	19.2MHz	13.5MHz

01	27MHz	24MHz
10	37.125MHz	27MHz
11	74.25MHz	27MHz

2. RCLK frequency by setting **RCLKEN_Y** = 1 for user-defined frequency generation (see [RCLKOUT Configuration](#) sub-section for additional information).

The PCLK, HS, and VS signals generated by the **REF_VTG** block are synchronous; these can be output on MFP pins and routed to connected image sensor(s).

14.3.2 Key VTG Timing Parameters

Note: See the [VTG Configuration](#) section for comprehensive configuration details.

1. **GEN_VS** – Enable VS signal generation.
2. **VS_INV** – Invert the VS signal (default = positive sync polarity).
3. **GEN_HS** – Enable HS signal generation.
4. **HS_INV** – Invert the HS signal (default = positive sync polarity).
5. **VS_DLY** – Delay from the VS trigger to the generated VS signal expressed in PCLK cycles (default = 0).
6. **VS_HIGH** – VS high period in terms of PCLK cycles.
7. **VS_LOW** – VS low period in terms of PCLK cycles.
8. **V2H** – The delay from the VS trigger to the rising edge of the generated HS signal in terms of PCLK cycles.
9. **HS_HIGH** – HS high period in terms of PCLK cycles.
10. **HS_LOW** – HS low period in terms of PCLK cycles.
11. **HS_CNT** – HS counts/pulses per frame.
12. **REFGEN_RST** – Reset the REFGEN_PLL; used when changing the frequency.

14.4 Feature Availability by Device Family

Sync pulse outputs are available on select GMSL2 devices ([Table 44](#)).

Table 44. Sync Pulse Outputs Support by Device Family

Device Family	Sync Pulse Outputs
HDMI Serializers	X
CSI-2 Serializers	X
Advanced CSI-2 Serializers	
CSI-2 Camera Deserializers	X
oLDI Deserializers	X
CSI-2 Quad Deserializers	X

15. Audio

15.1 Overview

GMSL2 devices support bidirectional transmission of audio (that is, I²S and TDM) over the forward and reverse channels of the serial link. Audio signals received on the local side are tunneled over the serial link and reconstructed on the remote side. For system flexibility, GMSL2 devices can act as either the audio subordinate or the main at either end of the link. See [Feature Availability by Device Family](#) for audio capabilities of each GMSL2 device.

Traditional audio systems often comprise a mix of analog and digital transmission protocols. In these systems, analog interfaces at the input and output are bridged and processed digitally. This is illustrated in [Figure 40](#). I²S and TDM formats are typically used for inter-IC data communications and transfers on the same PC board, whereas off-board data connections may require other, long-distance protocols.

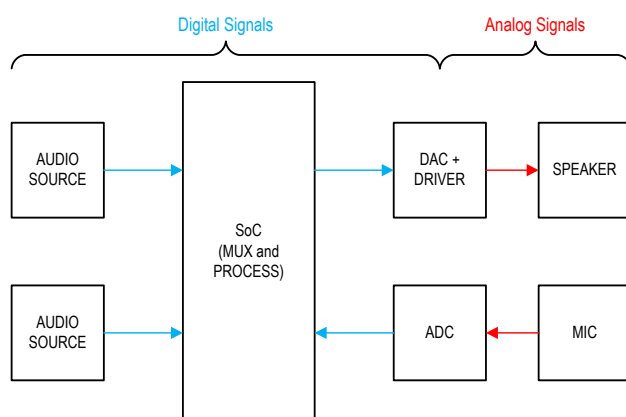


Figure 40. Typical Audio System Block Diagram

GMSL2 devices and the serial link serve as an integrated transmission interface for multiple data formats (example, video, audio, and control channel) within a system. Tunneling I²S/TDM audio data over the serial link eliminates the need for long-distance audio protocols (example, S/PDIF, optical, etc.) and simplifies the design of larger, interconnected systems ([Figure 41](#)).

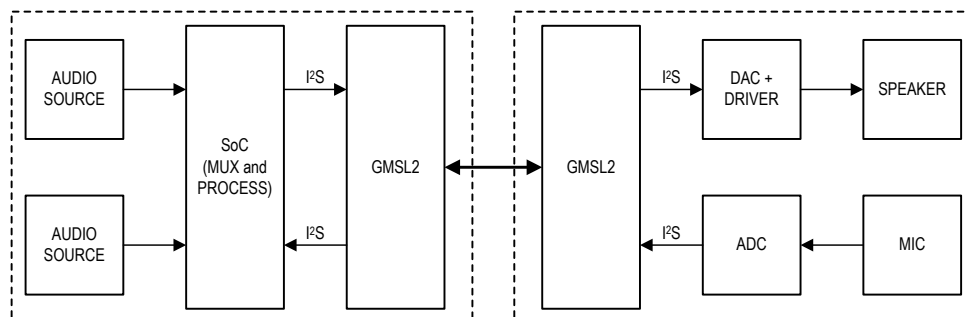


Figure 41. Typical GMSL2 Audio System Block Diagram

15.2 Operation

15.2.1 Hardware Connection

An I²S interface consists of three pins: clock signal (SCK), data signal (SD), and word select (WS).

[Table 45](#) contains the I²S pin naming convention for forward and reverse audio.

Table 45. GMSL2 Device Audio Pinouts

GMSL2 Device	AUDIO Input Pins	AUDIO Output Pins
Serializer	SD/SCK/WS SD2/SCK2/WS2	SDOR/SCKOR/WSOR SDOR2/SCKOR2/WSOR2
Deserializer	SDIR/SCKIR/WSIR	SD/SCK/WS

Note: TDM audio uses the same pins as I²S audio.

15.2.2 Initialization

Configure and enable the audio block before initializing audio transmissions. When the configured audio block is enabled, the following operations are performed automatically upon device power-up.

Audio Main:

Upon detecting the presence of an audio signal on the input pins:

- The number of SD bits per WS period are measured (when the reference clock stabilizes).
- The SCK clock frequency is measured.
- The SCK clock frequency and WS length are sent to the remote device using an info frame packet.
- Audio packets are sent over the link with the configured stream ID (there are four possible IDs). One audio packet is sent for each WS period.

Audio Subordinate:

Upon receiving audio packets over the GMSL link:

1. The audio receiver detects the presence of audio packets for the configured audio stream.
2. The info frame transmitted by the audio main (containing the SCK frequency and WS length) is received, and the audio output PLL initializes with the received SCK frequency.
3. The audio output SCK frequency is fine-tuned to match the detected input clock frequency using a FIFO control loop.
4. Audio is output from the enabled MFP pins.
5. If CRC/ARQ is enabled (default):
 - a. **No CRC error:** The packet is sent to the audio receiver and an acknowledge is sent to the audio transmitter.
 - b. **CRC error:** The packet is discarded and the retransmitted packet from the audio transmitter side is expected (see [CRC Error Detection and ARQ Error Correction](#) section for more information).

15.2.3 Input Audio Properties

15.2.3.1 Data Format

GMSL2 devices support both I²S stereo and up to eight channels of audio in TDM mode. Data is tunneled from the local to the remote side of the serial link. Any data format that meets the length and frequency requirements can be used.

15.2.3.2 Length/Frequency

Sample rates (that is, WS frequency) of 8kHz to 192kHz and sample depths of 8-bits to 32-bits are supported. For each WS cycle, there must be between 16 and 256 even-numbered bits. The SCK frequency range supported is 0.5MHz to 49.152MHz.

15.2.3.3 I²S vs TDM

Inter-IC sound (I²S) and time-division multiplexing (TDM) are digital audio formats used to transmit audio data. Both formats use three signals (that is, sync, clock, and data). I²S audio supports two-channel stereo ([Figure 42](#)), and TDM mode supports up to eight channels of audio ([Figure 43](#)). GMSL2 devices handle I²S and TDM formats similarly (except the WS polarity (see the [Signal Polarity](#) section)). For system flexibility, GMSL2 devices (both serializers and deserializers) can act as either the audio subordinate or audio main at either end of the serial link.

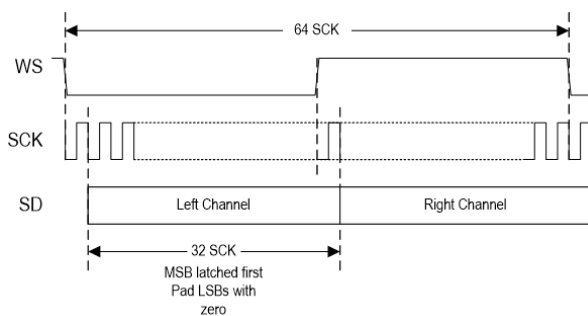


Figure 42 8-Channel TDM

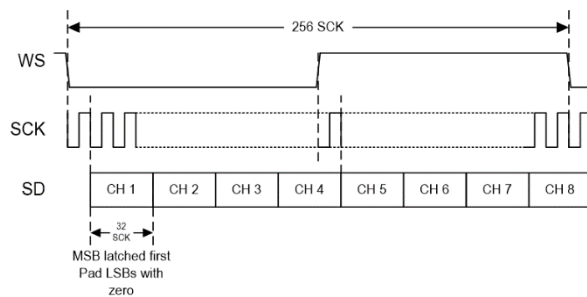


Figure 43 Two-Channel Stereo

Note: The WS period is defined by the audio source and is not limited to the 50% duty cycle, as shown in [Figure 42](#) and [Figure 43](#). However, the audio output duty cycle from the remote-side GMSL device is always 50%. Therefore, for WS inputs with a non-50% duty cycle, the audio output is not a direct reproduction of the input.

In I²S format, two channels of audio are transmitted during each WS period; the audio data is transmitted in alternating channels starting with the MSB of the left channel. TDM format is similar to I²S format but includes up to eight channels of audio data.

Note: For I²S, the left and right channel designation are not limited to what is shown in [Figure 42](#). For TDM mode, applications are not exclusively limited to 8-channel TDM, as shown in [Figure 43](#). The GMSL2 audio interface can apply TDM mode in a range of possible channel configurations up to 8 channels.

GMSL2 devices do not interpret the transmitted I²S data; the audio data is replicated on the opposing serializer or deserializer. The replicated data is interpreted by the processor in the receiver device on the opposing link. Ensure the GMSL2 devices are configured properly to ensure the transmitted audio data meets the requirements for the application.

15.2.3.4 Signal Polarity

GMSL2 systems allow for polarity configuration of the audio signal. By default, the SD and WS signals are sampled at the positive edge of the SCK ([Figure 44](#)). This behavior can be changed with the `INV_SCK` register in the `AUDIO_TX` block. The input polarity defaults to sampling audio frames at the

falling edge of the WS signal. The `INV_WS` register in the `AUDIO_TX` block changes the tracking edge for the WS frame.

If a different polarity is used on the input, set the `INV_SCK_TX` or `INV_WS_TX` bits = 1 accordingly to use the reverse input polarity of the SCK or WS signal.

The WS and SCK input, and output polarities can all be set independently. This allows for the device(s) to work with all polarity combinations on the input and output.

Note: TDM uses a rising edge of WS to indicate a start of frame. Bit `INV_WS_TX` must be set when using TDM audio as an input.

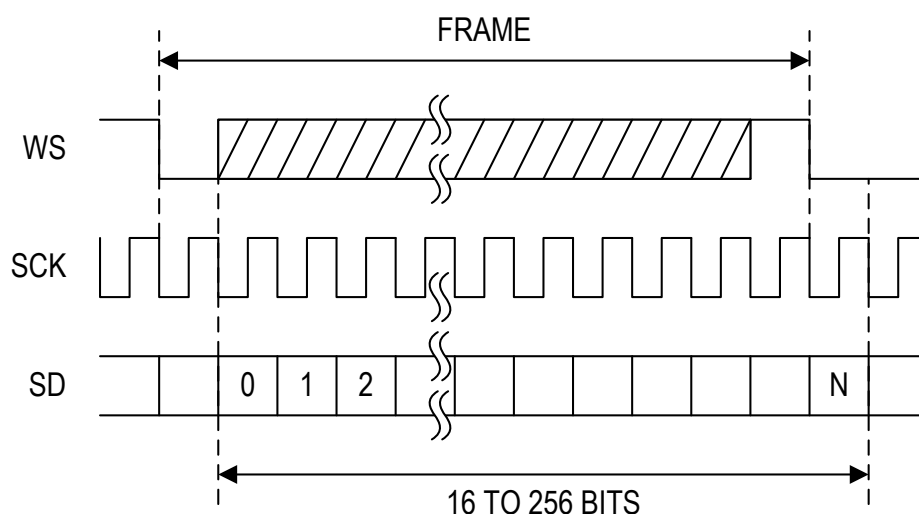


Figure 44. Default Input and Output Audio Polarity

15.2.4 Output Audio Properties

15.2.4.1 Signal Format

The output polarity defaults to sampling audio frames at the falling edge of the WS signal with data valid on the SCK rising edge. The `INV_SCK_RX` or `INV_WS_RX` registers in the `AUDIO_RX` block are used to independently change the SCK and WS output polarities. If the `INV_WS` register in the `AUDIO_RX` block is set, the frame data stays the same, but the frame boundary for audio samples is tracked at the rising edge of the WS signal.

The output WS always has a 50% duty cycle regardless of input duty cycle.

Note: TDM uses a rising edge of WS to indicate a start of frame. Bit `INV_WS_RX` must be set when using TDM audio as an output.

15.2.4.2 MCLK

The main I²S clock (MCLK) is not transmitted through the GMSL2 link. If needed by the audio sync device, use either an externally generated source for the MCLK or configure the reference clock output

(if available) to be used as the MCLK. See the [Clocks and Frequency Reference](#) section for configuration details.

15.2.5 Audio Calculations

15.2.5.1 Audio Bandwidth Calculations

The GMSL2 link bandwidth consumption by the audio channel (either I²S or TDM) is given by the following formulas.

$$\text{Audio Forward BW (Mbps)} = \left(\frac{\text{sample_rate}}{1000} \right) \times 20 \times \left(\text{roundup} \left(\text{channels} \times \frac{\text{sample_depth}}{18} + 0.5 \right) + 1 + \text{CRC} \right)$$

$$\text{Audio Reverse BW (Mbps)} = \left(\frac{\text{sample_rate}}{1000} \right) \times 6 \times 20 \times \text{CRC}$$

where:

- sample_rate = audio sample rate (kbps).
- sample_depth = audio sample depth (bits per sample).
- channels = number of channels.
- roundup() is a function defined as rounding up the contained number to the next integer value.
- CRC = 1 if CRC is enabled; CRC = 0 if CRC is not enabled.

15.2.5.2 Audio Clock Frequency Calculations

The audio clock frequencies for forward and reverse channel audio are calculated with the following formulas.

Note: Convert all values to decimal for the equations.

Audio Clock Frequency (MHz) (Forward Channel Audio)

Calculate the forward channel audio clock frequency with deserializer status register values.

$$\text{Audio Clock (MHz)} = \left(\text{INFO_AUD_FB_DIV} + \frac{\text{INFO_AUD_FB_DIV_FRAC}}{64} \right) \times \left(\frac{75}{2 \times \text{INFO_AUD_ODIV} \times 2^{\text{INFO_AUD_ODIV_EXP}}} \right)$$

Audio Clock Frequency (MHz) (Reverse Channel Audio)

Calculate the reverse channel audio clock frequency with serializer status register values.

$$\text{Audio Clock (MHz)} = \left(\text{INFO_AUD_FB_DIV} + \frac{\text{INFO_AUD_FB_DIV_FRAC}}{64} \right) \times \left(\frac{75}{\text{INFO_AUD_ODIV} \times 2^{\text{INFO_AUD_ODIV_EXP}}} \right)$$

Note: See the [GMSL2 Link Bandwidth Consumption from Side Channels \(Audio\)](#) section for more information.

15.3 Configuration

This section contains descriptions of the various operating modes and register configurations required for programming. In each operating mode, the audio input is tunneled through the GMSL link and reconstructed on the audio output (except for any configured polarity inversion).

Modes of Operation

The following modes of operation and tests are supported by various GMSL2 devices. See [Feature Availability by Device Family](#) or the part data sheet to confirm modes supported by device.

I²S/TDM Audio

1. [Forward Channel Audio](#)
2. [Reverse Channel Audio](#)
3. [Forward and Reverse Channel Audio](#)
4. [Two Forward Channel Audio in Splitter Mode](#)
5. [Two Reverse Channel Audio in Splitter Mode](#)
6. [Four-Pin Bidirectional Audio I2S](#)

HDMI Audio

7. [HDMI Audio](#) (HDMI serializers only)

PRBS Test Modes

8. Forward Channel Audio PRBS Test – Internal Oscillator ([Table 86](#))
9. Forward Channel Audio PRBS Test – External Audio Clock ([Table 87](#))
10. Reverse Channel Audio PRBS Test – Internal Oscillator ([Table 88](#))
11. Reverse Channel Audio PRBS Test – External Audio Clock ([Table 89](#))

Note: See the [PRBS Testing](#) section for additional details.

15.3.1 I²S/TDM Audio Configuration

15.3.1.1 Forward Channel Audio

Forward channel audio is serializer-to-deserializer audio ([Figure 45](#)). The serializer receives the audio on the I²S input pins and transmits the audio data over the serial link. The deserializer receives the audio data from the serial link, reconstructs the audio, and outputs the data to the I²S pins. Each serializer has up to two forward audio channels (that is, channel X and channel Y). Configuration is detailed in [Table 46](#) and [Table 47](#).

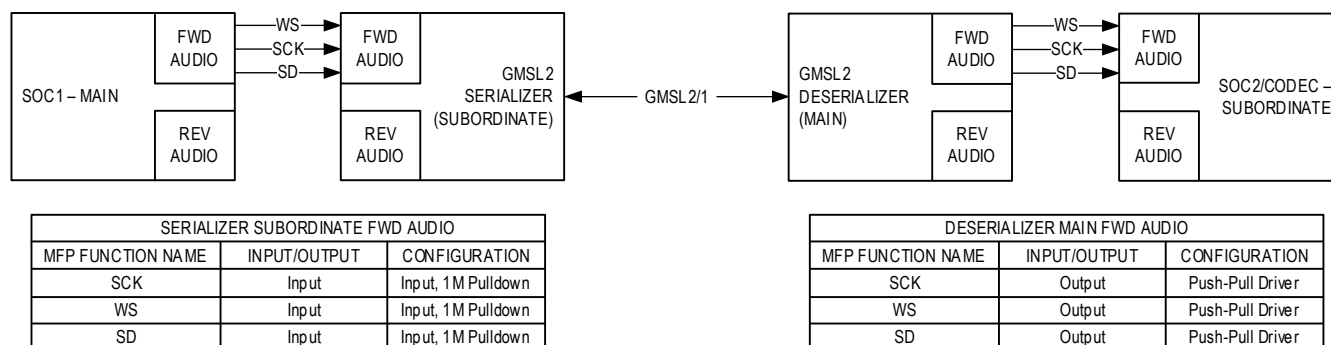


Figure 45. Forward Audio Channel I²S Block Diagram

Table 46. Configuration of Forward Channel Audio on Channel X

Description	Serializer Configuration	Deserializer Configuration
Enable audio transmission	Register: REG2 Bit: AUD_TX_EN_X = 1	-
Select Audio Stream ID. Program serializer and deserializer to the same value.	Register: Audio X: AUDIO_TX1 Bit: AUD_STR_TX[1:0]	Register: AUDIO_RX7 Bit: AUD_STRM[1:0]
If TDM mode is used, invert WS.	Register: AUDIO_TX0 Bit: INV_WS	-
Enable audio receiver	-	Register: AUDIO_RX1 Bit: AUD_EN_RX = 1

Table 47. Configuration of Forward Channel Audio on Channel Y

Description	Serializer Configuration	Deserializer Configuration
Enable audio transmission	Register: REG2 Bit: AUD_TX_EN_Y = 1	-
Select Audio Stream ID. Program serializer and deserializer to the same value.	Register: Audio Y: AUDIO_TX1 Bit: AUD_STR_TX[1:0]	Register: AUDIO_RX7 Bit: AUD_STRM[1:0]
If TDM mode is used, invert WS.	Register: AUDIO_TX0 Bit: INV_WS	-
Enable audio receiver	-	Register: AUDIO_RX1 Bit: AUD_EN_RX = 1

15.3.1.2 Reverse Channel Audio

Reverse channel audio is deserializer-to-serializer audio ([Figure 46](#)). The deserializer receives the audio on the I²S input pins and transmits the audio data over the serial link. The serializer receives the audio data from the serial link, reconstructs the audio, and outputs the data to the I²S pins. Forward and reverse audio channels are functionally the same, however, total available bandwidth is different. See [Serial Link Rates by Part Number](#) in [GMSL2 Link Basics](#) and/or device-specific data sheets for more information. Configuration is detailed in [Table 48](#).

Note: The deserializer input I²S bus pin names are different than the serializer output I²S bus pin names. The deserializer uses the additional “I” to denote *Input*, whereas the serializer uses “O” to denote *Output* and “R” to indicate it is the *Reverse Channel*. See [Figure 46](#).

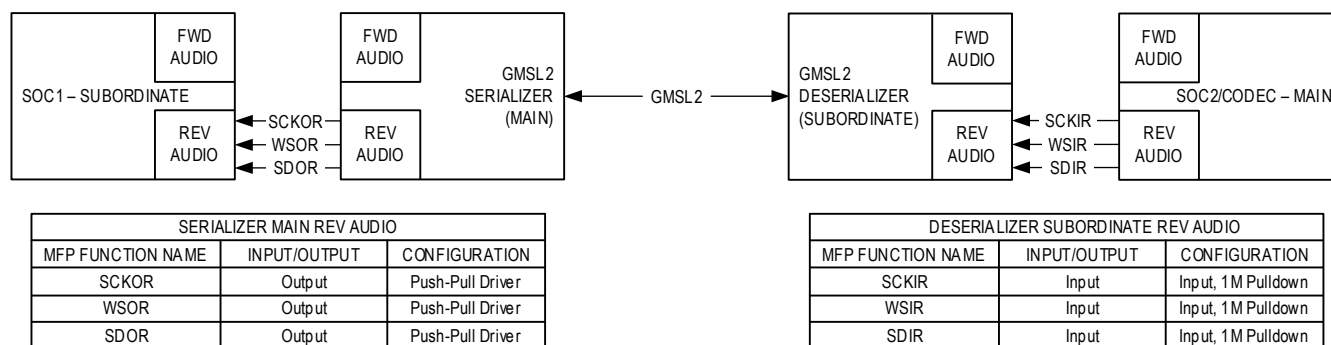


Figure 46. Reverse Channel I²S Block Diagram

Table 48. Configuration of Reverse Channel Audio

Description	Serializer Configuration	Deserializer Configuration
Enable audio transmission	-	Register: REG2 Bit: AUD_TX_EN = 1
Select Audio Stream ID. Program serializer and deserializer to the same value.	Register: AUDIO_RX7 Bit: AUD_STRM[1:0]	Register: AUDIO_TX1 Bit: AUD_STR_TX[1:0]
If TDM mode is used, invert WS.	-	Register: AUDIO_TX0 Bit: INV_WS
Enable audio receiver	Register: AUDIO_RX1 Bit: AUD_EN_RX = 1	-

15.3.1.3 Forward and Reverse Channel Audio

Forward and reverse channel audio is the simultaneous transmission of both serializer-to-deserializer and deserializer-to-serializer audio (Figure 47). On the forward channel, the serializer receives the audio on the I²S input pins and transmits the audio data over the serial link. The deserializer receives the audio data from the serial link, reconstructs the audio, and outputs the data to the I²S pins. On the reverse channel, the deserializer receives the audio on the I²S input pins and transmits the audio data over the serial link. The serializer receives the audio data from the serial link, reconstructs the audio, and outputs the data to the I²S pins. The configuration procedure to enable both forward and reverse channel audio at the same time is presented in Table 49.

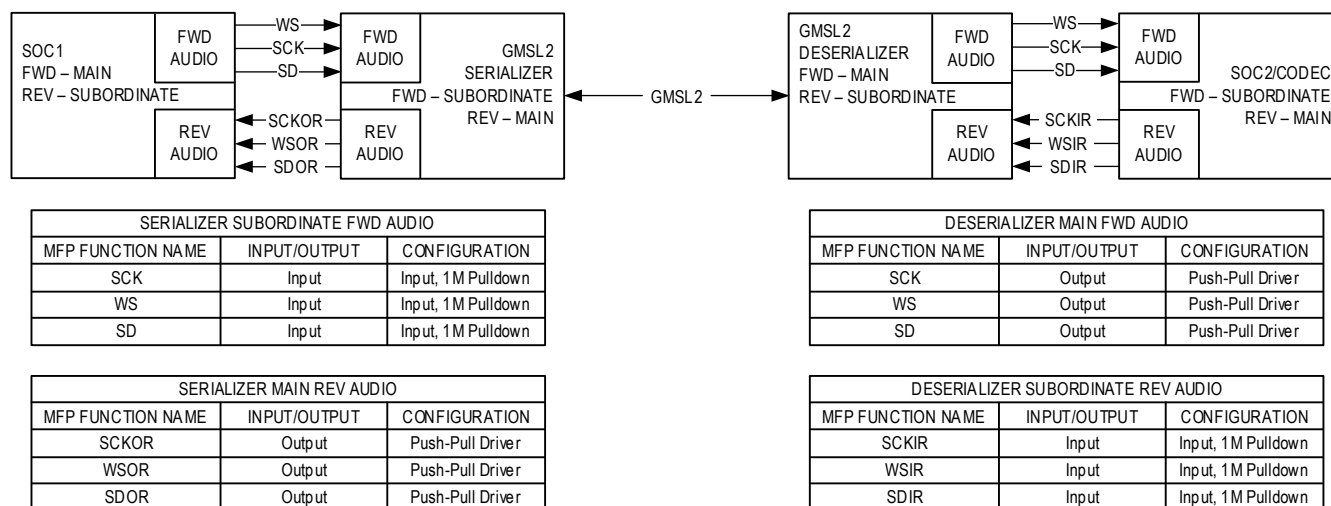


Figure 47. Forward and Reverse Channel I²S Block Diagram

Table 49. Forward and Reverse Channel Audio

Description	Serializer Configuration	Deserializer Configuration
Enable forward channel audio transmitter	Register: REG2 Bit: AUD_TX_EN_X = 1	-
Enable forward channel audio receiver	-	Register: AUDIO_RX1 Bit: AUD_EN_RX = 1
Enable reverse channel audio transmitter	-	Register: REG2 Bit: AUD_TX_EN = 1
Enable reverse channel audio receiver	Register: Audio X: AUDIO_RX1 Bit: AUD_EN_RX = 1	
Select Audio Stream ID for Forward Channel Audio. Program serializer and deserializer to the same value.	Register: Audio X: AUDIO_TX1 Bit: AUD_STR_TX[1:0]	Register: AUDIO_RX7 Bit: AUD_STRM[1:0]
Select Audio Stream ID for Reverse Channel Audio. Program serializer and deserializer to the same value.	Register: AUDIO_RX7 Bit: AUD_STRM[1:0]	Register: AUDIO_TX1 Bit: AUD_STR_TX[1:0]
Only for DSI Serializers: Select which PHY to receive reverse channel audio from.	Register: REG3 Bit: AUD_RX_A_SRC = 0 for PHY A Bit: AUD_RX_A_SRC = 1 for PHY B	-
Only for DSI Serializers: Select which audio input pins to use for forward channel audio input.	Register: REG4 Bit: AUD_TX_SRC_X = 0 for SCK,SD,WS Bit: AUD_TX_SRC_X = 1 for SCK2,SD2,WS2	-

15.3.1.4 Two Forward Channel Audio in Splitter Mode

Two forward channel audio in splitter mode is serializer-to-deserializer audio ([Figure 48](#)). A single serializer in splitter mode (through two GMSL2 PHYs) connects to two deserializers. The serializer receives the audio on the I²S input pins and transmits the audio data over the serial link. Both deserializers receive the audio data from the serial link, independently reconstruct the audio, and output the data to their respective I²S pins. The available bandwidth to each deserializer is equal to the forward channel link rate that has been configured. Configuration is detailed in [Table 50](#).

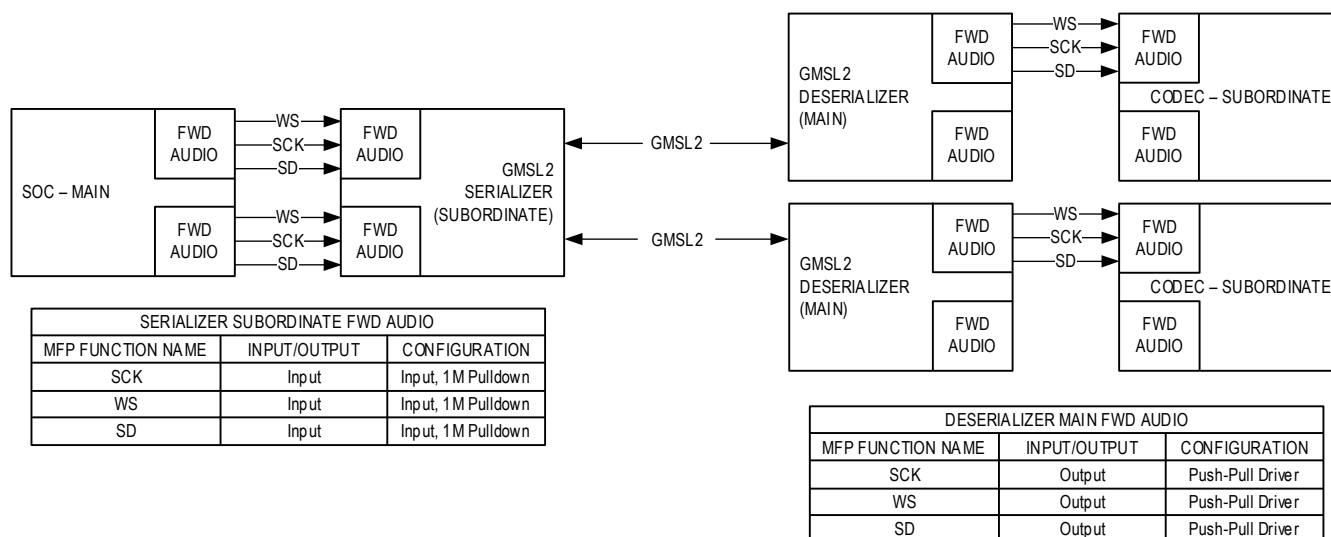


Figure 48. Two Forward Channel I²S in Splitter Mode Block Diagram

Table 50. Two Forward Channel Audio in Splitter Mode

Description	Serializer Configuration	Deserializer 1 Config	Deserializer 2 Config
Send Audio X packets to GMSL PHY A	Register: AUDIO_X: TR3 Bit: TX_SPLT_MASK_A = 1	-	-
Do not send Audio X packets to GMSL PHY B	Register: AUDIO_X: TR3 Bit: TX_SPLT_MASK_B = 0	-	-
Do not send Audio Y packets to GMSL PHY B	Register: AUDIO_Y: TR3 Bit: TX_SPLT_MASK_A = 0	-	-
Send Audio Y packets to GMSL PHY B	Register: AUDIO_Y: TR3 Bit: TX_SPLT_MASK_B = 1	-	-
Audio channel X: Set ARQ settings to manual mode	Register: AUDIO_X: ARQ0 Bit: ARQ_AUTO_CFG = 0	-	-
Audio channel X: Use TX_SRC_ID bitfield instead of SRC_ID of received data packet.	Register: AUDIO_X: ARQ0 Bit: ACK_SRC_ID = 1	-	-
Audio channel X: Ack packet is accepted if SRC_ID of the received Ack packet matches TX_SRC_ID register.	Register: AUDIO_X: ARQ0 Bit: MATCH_SRC_ID = 1	-	-

Audio channel X: Wait for one ACK instead of two	Register: AUDIO_Y: ARQ0 Bit: ACK_CNT = 0	-	-
Audio channel Y: Set ARQ settings to manual mode	Register: AUDIO_Y: ARQ0 Bit: ARQ_AUTO_CFG = 0	-	-
Audio channel Y: Use TX_SRC_ID bitfield instead of SRC_ID of received data packet.	Register: AUDIO_Y: ARQ0 Bit: ACK_SRC_ID = 1	-	-
Audio channel Y: Ack packet is accepted if SRC_ID of the received Ack packet matches TX_SRC_ID register.	Register: AUDIO_Y: ARQ0 Bit: MATCH_SRC_ID = 1	-	-
Audio channel Y: Wait for one ACK instead of two	Register: AUDIO_Y: ARQ0 Bit: ACK_CNT = 0	-	-
Audio channel Y: Enable ARQ (not enabled by default ton Pipe Y)	Register: AUDIO_Y: ARQ0 Bit: ARQ_EN = 1	-	-
Deserializers: Set ARQ settings to manual mode	-	Register: AUDIO: ARQ0 Bit: ARQ_AUTO_CFG = 0	Register: AUDIO: ARQ0 Bit: ARQ_AUTO_CFG = 0
Deserializers: Use SRC_ID of the received data packet	-	Register: AUDIO: ARQ0 Bit: ACK_SRC_ID = 0	Register: AUDIO: ARQ0 Bit: ACK_SRC_ID = 0
Deserializers: All received Ack packets are accepted	-	Register: AUDIO: ARQ0 Bit: MATCH_SRC_ID = 0	Register: AUDIO: ARQ0 Bit: MATCH_SRC_ID = 0
Deserializers: Wait for one ACK instead of two	-	Register: AUDIO: ARQ0 Bit: ACK_CNT = 0	Register: AUDIO: ARQ0 Bit: ACK_CNT = 0
Audio Channel X: Transmit audio packets on stream id 0	Register: Audio X: AUDIO_TX1 Bit: AUD_STR_TX[1:0] = 00	-	-
Audio Channel Y: Transmit audio packets on stream id 1	Register: Audio Y: AUDIO_TX1 Bit: AUD_STR_TX[1:0] = 01	-	-
Set deserializers to Receive on audio Stream ID 0, and 1 respectively	-	Register: AUDIO_RX7 Bit: AUD_STRM[1:0] = 00	Register: AUDIO_RX7 Bit: AUD_STRM[1:0] = 01
Transmit Audio with STREAM ID 0 from Channel X	Register: Audio X: AUDIO_TX1 Bit: AUD_STR_TX[1:0] = 00	-	-
Transmit Audio with STREAM ID 1 from Channel Y	Register: Audio Y: AUDIO_TX1 Bit: AUD_STR_TX[1:0] = 01	-	-
Select audio input pins for audio channel X. Set AUD_TX_SRC_X	Register: REG4 Bit: AUD_TX_SRC_X	-	-

depending on which pins are desired			
Select audio input pins for audio channel Y. Set AUD_TX_SRC_Y depending on which pins are desired.	Register: REG4 Bit: AUD_TX_SRC_Y	-	-
Disable reverse channel audio on deserializers	-	Register: REG2 Bit: AUD_TX_EN = 0	Register: REG2 Bit: AUD_TX_EN = 0
Disable reverse channel audio channel X	Register: Audio X: AUDIO_RX1 Bit: AUD_EN_RX = 0	-	-
Disable reverse channel audio channel Y	Register: Audio Y: AUDIO_RX1 Bit: AUD_EN_RX = 0	-	--
Forward channel audio enable in deserializers	-	Register: AUDIO_RX1 Bit: AUD_EN_RX = 1	Register: AUDIO_RX1 Bit: AUD_EN_RX = 1
If TDM mode is used on channel X, invert WS	Register: AUD_TX_AX: AUDIO_TX0 Bit: INV_WS	-	-
If TDM mode is used on channel Y, invert WS	Register: AUD_TX_AY: AUDIO_TX0 Bit: INV_WS	-	-
Audio transmitter X enable	Register: REG2 Bit: AUD_TX_EN_X = 1	-	-
Audio transmitter Y enable	Register: REG2 Bit: AUD_TX_EN_Y = 1	-	-
Reset link	Register: CTRL0 Bit: RESET_ONESHOT = 1	-	-

15.3.1.5 Two Reverse Channel Audio in Splitter Mode

Two reverse channel audio in splitter mode is deserializer-to-serializer audio ([Figure 49](#)). Two deserializers connect to a single serializer in splitter mode (through two GMSL2 PHYs). The deserializers receive the audio on the respective I²S input pins and independently transmit the audio data over the serial link. The serializer receives the audio data from the serial link, reconstructs the audio, and outputs the data to the I²S pins. The available bandwidth from each deserializer is equal to the reverse channel link rate that has been configured. Configuration is detailed in [Table 51](#).

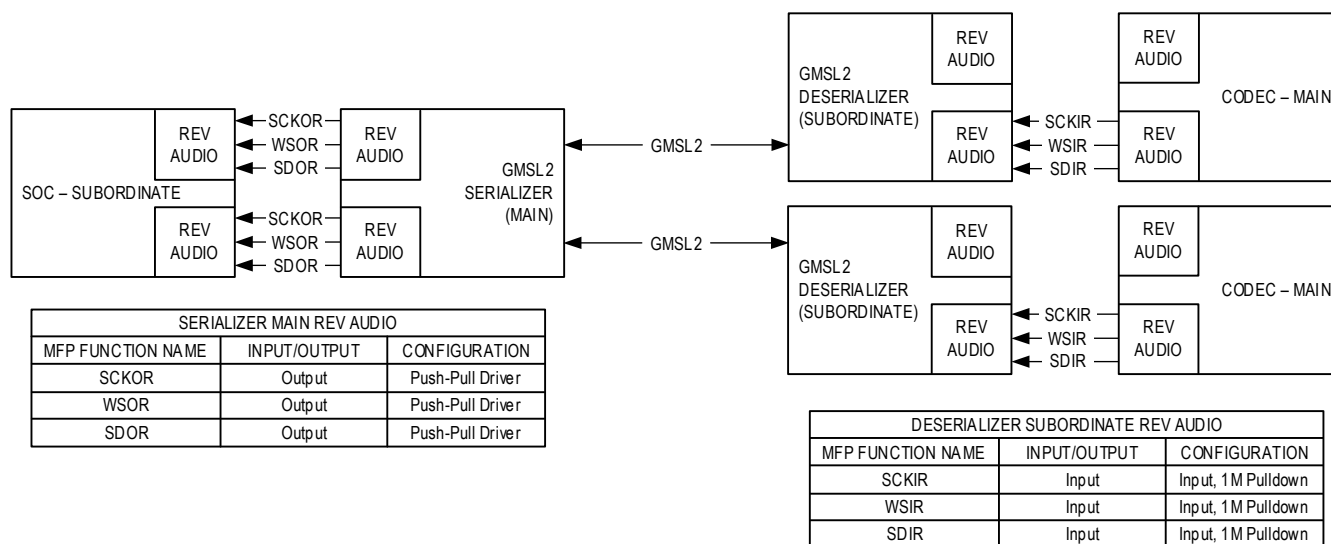


Figure 49. Two Reverse Channel I²S in Splitter Mode Block Diagram

Table 51. Reverse Channel Audio

Description	Serializer Configuration	Deserializer 1 Config	Deserializer 2 Config
Packets NOT transmitted over GMSLA in splitter	Register: AUDIO_Y: TR3 Bit: TX_SPLT_MASK_A = 0	-	-
Packets ARE transmitted over GMSLB in splitter	Register: AUDIO_Y: TR3 Bit: TX_SPLT_MASK_B = 1	-	-
Packets are transmitted over GMSLA in splitter	Register: AUDIO_X: TR3 Bit: TX_SPLT_MASK_A = 1	-	-
Packets NOT transmitted over GMSLB in splitter	Register: AUDIO_X: TR3 Bit: TX_SPLT_MASK_B = 0	-	-
Set Audio Tx stream id 0 and 1 respectively	-	Register: AUDIO_TX1 Bit: AUD_STR_TX[1:0] = 00	Register: AUDIO_TX1 Bit: AUD_STR_TX[1:0] = 01
Audio transmitter X disable	Register: REG2 Bit: AUD_TX_EN_X = 0	-	-
Audio transmitter Y disable	Register: REG2 Bit: AUD_TX_EN_Y = 0	-	-
Audio Receiver X enable	Register: Audio X: AUDIO_RX1 Bit: AUD_EN_RX = 1	-	-
Audio Receiver Y enable	Register: Audio Y: AUDIO_RX1 Bit: AUD_EN_RX = 1	-	-
If TDM mode is used, invert WS	-	Register: AUDIO_TX0 Bit: INV_WS	Register: AUDIO_TX0 Bit: INV_WS
Select SRC ID to receive on X	Register: AUDIO_X: TR4 Bit: RX_SRDC_SEL[7:0] = All zeroes except the bit with index equal to DESB's TX_SRC_ID_AUDIO	-	-

Select SRC ID to receive on X	Register: AUDIO_Y : TR4 Bit: RX_SRDC_SEL[7:0] = All zeroes except the bit with index equal to DESB's TX_SRC_ID_AUDIO	-	-
ARQ settings for X are based on selections in ACK_SRC_ID and MATCH_SRC_ID bitfields.	Register: AUDIO_X: ARQ0 Bit: ARQ_AUTO_CFG = 0	-	-
Wait for 1 ACK instead of 2 on X	Register: AUDIO_X: ARQ0 Bit: ACK_CNT_AUDIO_X = 0	-	-
All received Ack packets are accepted	Register: AUDIO_X: ARQ0 Bit: MATCH_SRC_ID = 0	-	-
Use TX_SRC_ID bitfield instead of SRC_ID for X	Register: AUDIO_X: ARQ0 Bit: ACK_SRC_ID = 1	-	-
ARQ settings for Y are based on selections in ACK_SRC_ID and MATCH_SRC_ID bitfields.	Register: AUDIO_Y: ARQ0 Bit: ARQ_AUTO_CFG = 0	-	-
Wait for 1 ACK instead of 2 on Y	Register: AUDIO_Y: ARQ0 Bit: ACK_CNT_AUDIO_X = 0	-	-
All received Ack packets are accepted	Register: AUDIO_Y: ARQ0 Bit: MATCH_SRC_ID = 0	-	-
Use TX_SRC_ID bitfield instead of SRC_ID for Y	Register: AUDIO_Y: ARQ0 Bit: ACK_SRC_ID = 1	-	-
Select Audio channel to receive audio from PHYA	Register: REG3 Bit: AUD_RX_A_SRC = 0 (Toggle to swap pins on SER output)	-	-
Select Audio channel to receive audio from PHYB	Register: REG3 Bit: AUD_RX_B_SRC = 0 (Toggle to swap pins on SER output)	-	-
Receive audio stream ID 0 on A	Register: AUDIO_RX7 Bit: AUD_STRM[1:0] = 00	-	-
Receive audio stream ID 1 on B	Register: AUDIO_RX7 Bit: AUD_STRM[1:0] = 01	-	-
Audio Receiver disable	-	Register: AUDIO_RX1 Bit: AUD_EN_RX = 0	Register: AUDIO_RX1 Bit: AUD_EN_RX = 0
Audio transmitter enable	-	Register: REG2 Bit: AUD_TX_EN = 1	Register: REG2 Bit: AUD_TX_EN = 1
Reset link	Register: CTRL0 Bit: RESET_ONESHOT = 1	-	-

15.3.1.6 Four-Pin Bidirectional Audio I²S

When the serial link audio configuration is the same for both audio directions, it is possible to use only the SD pin (and eliminate the WS and SCK pins) for one of the directions to reduce the system pin count from six to four. Here, the SD pin is clocked off the SCK of the opposite channel.

On the forward channel, the serializer receives the audio on the I²S input pins and transmits the audio data over the serial link. The deserializer receives the audio data from the serial link, reconstructs the audio, and outputs the data to the I²S pins. On the reverse channel, the deserializer receives the audio on I²S input pins and transmits the audio data over the serial link. The serializer receives the audio data from the serial link, reconstructs the audio, and outputs the data to the I²S pins.

Figure 50 diagrams illustrate the system when the forward channel or reverse channel (Figure 37) are designated as the main for the bidirectional 4-pin audio system. Configuration for the forward channel is detailed in Table 52 and for the reverse channel in Table 53.

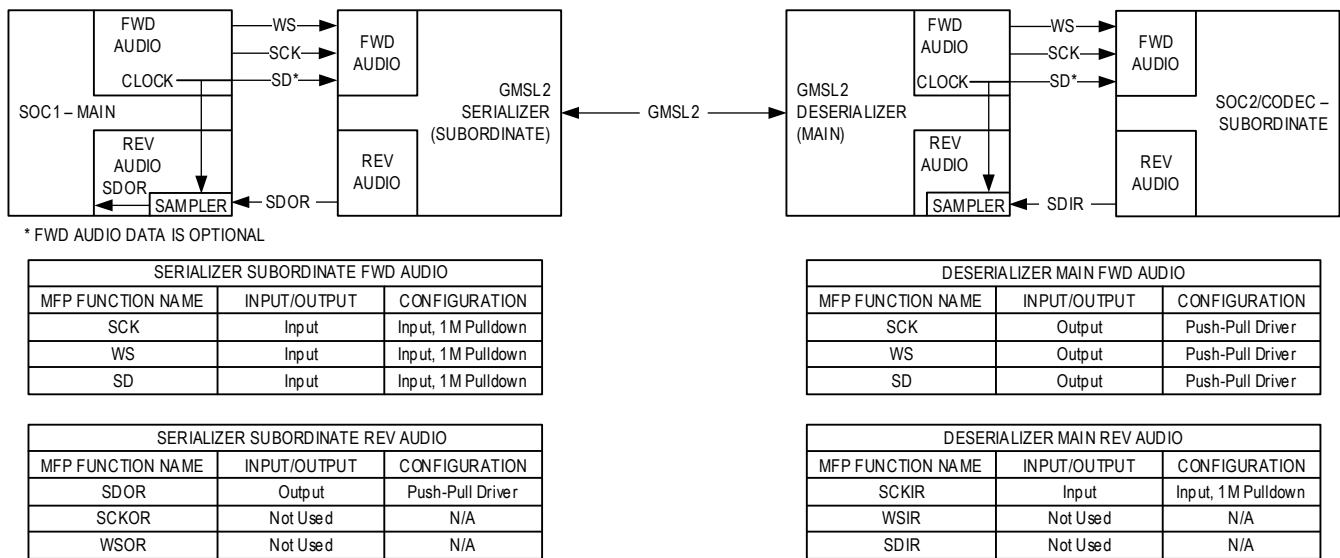


Figure 50. Forward Audio Main Set: AUD_SINK_SRC (Deserializer) and AUD_RX_SINK_SRC (Serializer)

Table 52. Configuration for Four-Pin Bidirectional Audio (Forward Audio Main)

Description	Serializer Configuration	Deserializer Configuration
Enable forward channel audio transmission	Register: REG2 Bit: AUD_TX_EN = 1	Register: AUDIO_RX1 Bit: AUD_EN_RX = 1
If TDM mode is used, invert WS	Register: AUDIO_TX0 Bit: INV_WS	Register: AUDIO_TX0 Bit: INV_WS
Select Audio Stream ID. Program serializer and deserializer to the same value.	Register: Audio X: AUDIO_TX1 Bit: AUD_STR_TX[1:0]	Register: AUDIO_RX7 Bit: AUD_STRM[1:0]
Enable reverse channel audio transmission	Register: AUDIO_RX1 Bit: AUD_EN_RX = 1	Register: REG2 Bit: AUD_TX_EN = 1
Enable bidirectional sync sourced mode for the reverse channel audio	Register: AUDIO_RX1 Bit: AUD_RX_SINK_SRC = 1	Register: AUDIO_TX0 Bit: AUD_SINK_SRC = 1

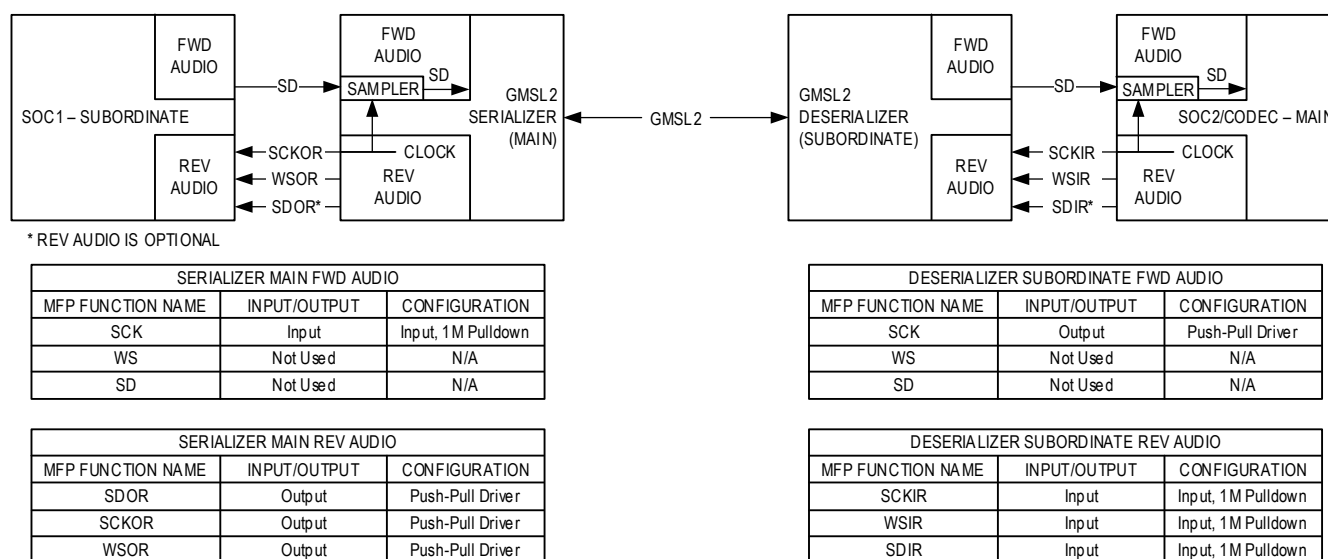


Figure 51. Reverse Audio Main Set: *AUD_SINK_SRC* (Serializer) and *AUD_RX_SINK_SRC* (Deserializer)

Table 53. Configuration for Four-Pin Bidirectional Audio (Reverse Audio Main)

Description	Serializer Configuration	Deserializer Configuration
Enable forward channel audio transmission	Register: REG2 Bit: AUD_TX_EN = 1	Register: AUDIO_RX1 Bit: AUD_EN_RX = 1
If TDM mode is used, invert WS	Register: AUDIO_TX0 Bit: INV_WS	Register: AUDIO_TX0 Bit: INV_WS
Select Audio Stream ID. Program serializer and deserializer to the same value.	Register: Audio X: AUDIO_TX1 Bit: AUD_STR_TX[1:0]	Register: AUDIO_RX7 Bit: AUD_STRM[1:0]
Enable reverse channel audio transmission	Register: AUDIO_RX1 Bit: AUD_EN_RX = 1	Register: REG2 Bit: AUD_TX_EN = 1
Enable bidirectional sync sourced mode for the forward channel audio	Register: AUDIO_TX0 Bit: AUD_SINK_SRC = 1	Register: AUDIO_RX1 Bit: AUD_RX_SINK_SRC = 1

15.3.2 HDMI Audio Configuration

GMSL2 HDMI serializers use HDMI audio for the serial link audio channel if audio is present on the HDMI input. For consumer port applications, HDMI audio capability must be added to the EDID table, and the source device settings must have 'HDMI display' selected as the audio output device. For embedded applications, the SoC must enable its HDMI audio output. See the [HDMI Serializers](#) section for additional information.

15.3.2.1 HDMI Audio

To set the HDMI as the audio source, toggle the `reg_i2s_mode[0]` bit in register `0x2129` to '1' along with the basic I²S forward audio setup.

15.3.2.1.1 HDMI Source – Forward Audio Example

Serializer Setup:

1. 0x80,0x2129,0x1C # `reg_i2s_mode[0]` = 1
2. 0x80,0x0002,0x47 # `AUD_TX_EN_X[0]` = 1
3. 0x80,0x0120,0x00 # `AUD_I2S_CFG[0-1]` = 0

Deserializer Setup:

4. 0x54,0x0140,0x21 # `AUD_EN_RX[0]` = 1
5. 0x54,0x01CE,0x5E # `OLDI Display Setup` = 1
- 0x80,0x0001,0x88 # `SER_HDMI_AUTOS[0]` = 1

15.3.3 Audio PRBS Tests

GMSL2 devices can perform various audio pseudo-random binary sequence (PRBS) tests that can be used to test and debug the audio system. These tests can also be performed to determine the bit error rate (BER) of audio packets. See the [PRBS Testing](#) section for additional information (including other PRBS tests available).

Table 54. Forward Channel Audio PRBS Test (Internal Oscillator)

Description	Serializer Configuration	Deserializer Configuration
Enable audio receiver	-	<code>AUD_EN_RX</code> = 1'b1
Enable audio transmission (Channel X)	<code>AUD_TX_EN_X</code> = 1'b1	-
Use the internal oscillator clock (150MHz) for PRBS test	<code>PRBS_SEL</code> = 1'b1	-
Enable audio PRBS test	<code>PRBSEN_AUD</code> = 1'b1	-
Enable audio PRBS checker	-	<code>APRBS_CHK_EN</code> = 1'b1

Table 55. Forward Channel Audio PRBS Test (External Audio Clock)

Description	Serializer Configuration	Deserializer Configuration
Enable audio receiver	-	<code>AUD_EN_RX</code> = 1'b1
Enable audio transmission (Channel X)	<code>AUD_TX_EN_X</code> = 1'b1	-
Use SCK for PRBS test	<code>PRBS_SEL</code> = 1'b0	-
Enable audio PRBS test	<code>PRBSEN_AUD</code> = 1'b1	-
Enable audio PRBS checker	-	<code>APRBS_CHK_EN</code> = 1'b1

Table 56. Reverse Channel Audio PRBS Test (Internal Oscillator)

Description	Serializer Configuration	Deserializer Configuration
Enable audio receiver (Channel X)	<code>AUD_EN_RX_X</code> = 1'b1	-
Enable audio transmission	-	<code>AUD_TX_EN</code> = 1'b1
Use the internal oscillator clock (150MHz) for PRBS test	-	<code>PRBS_SEL</code> = 1'b1
Enable audio PRBS test	-	<code>PRBSEN_AUD</code> = 1'b1
Enable audio PRBS checker	<code>APRBS_CHK_EN</code> = 1'b1	-

Table 57. Reverse Channel Audio PRBS Test (External Audio Clock)

Description	Serializer Configuration	Deserializer Configuration
Enable audio receiver (Channel X)	AUD_EN_RX_X = 1'b1	-
Enable audio transmission	-	AUD_TX_EN = 1'b1
Use SCK for PRBS test	-	PRBS_SEL = 1'b0 to use SCK
Enable audio PRBS test	-	PRBSEN_AUD = 1'b1
Enable audio PRBS checker	APRBS_CHK_EN = 1'b1	-

15.4 Audio Status and Debug

This section explains how to check for audio errors and provides debugging procedures for unexpected behavior.

15.4.1 Status Registers

The following status registers are used to ensure the proper implementation of audio in a system.

- **ACLKDET** (register **AUDIO_TX5**): Indicates if an audio input clock is detected. If this is 0, check that the audio system is enabled with the **AUD_TX_EN** registers, the audio input pins are set up and multiplexed to the **AUDIO_TX** block, and valid audio input is applied. Ensure that there is adequate bandwidth for the audio transformer by confirming that the **AUD_OVERFLOW** register is 0.
- **AUD_LOCK** (register **AUDIO_RX9**): Real-time flag indicating if the audio receiver output is operational. Check that the **AUD_EN_RX** is enabled and that the proper audio stream is selected from **AUD_STRM** to ensure that the audio receiver sub-system is enabled.
- **AUD_PKT_DET** (register **AUDIO_RX9**): Real-time flag indicating if the audio receiver is receiving the desired audio packets properly. Check that the **AUD_EN_RX** is enabled and that the proper audio stream is selected from **AUD_STRM** to verify that the audio receiver sub-system is enabled.

Registers **ACLKDET[0]**, **AUD_LOCK[0]**, and **AUD_PKT_DET[0]** aid during the I²S audio debugging process. If there is no audio on the receiver side, check if the register **ACLKDET[0]** is high. If **ACLKDET[0]** is low, then there is no audio clock detected at the receiver side. Ensure that the I²S receiver side has the **AUD_EN_RX[0]** bit set and the transmitter side has the **AUD_TX_EN[0]** bit set.

15.4.2 Debug Registers

- **AUD_DRIFT_ERR**: Audio clock drift error. The audio transmitter automatically restarts the audio transmission and reports an error if the audio clock drifts excessively.
- **ACLKDET**: Audio clock stop. If audio clock stops even for a short duration, this event is detected and flagged by the transmitter side clock detector and audio sub-system restarts.
- **AUD_BLK_LEN_ERR**: Audio receiver block length error. The receiver side audio receiver automatically restarts if the audio clock frequency output finetuning FIFO overflows or underflows due to errors. This can happen due to errors on the link especially when ARQ is turned off.

The following device audio error registers can be used for error diagnostics:

- **APRBS_ERR[0-7]** Number of audio PRBS errors detected; clears on read.
- **AUD_FIFO_WARN[0]** Audio FIFO is more than half full if high.

- [AUD_OVERFLOW\[0\]](#) Audio FIFO buffer overflow detected if high.
- [AUD_DRIFT_ERR\[0\]](#) Audio Clock frequency drift detected if high.

15.4.3 Debug Procedure

Perform the following procedure to debug the audio system:

- Check that the audio input is connected to the correct input pins, is exhibiting the expected IOVDD voltage level, and has been started.
- Ensure that the audio input format is stable and compliant with I²S specifications and the data sheet EC table.
- Verify that both the audio receiver and transmitter are enabled.
- Confirm that both ends of the link are using the same audio stream ID.
- Check that there are no errors or retransmission events on the link through the [RT_CNT](#) and [DEC_ERR](#) registers. This can indicate poor link quality causing audio packet corruption.
- Ensure that the audio stream is routed to the correct GMSL PHY with the [TX_SPLT_MASK_A/B_AUD](#) registers.
- Verify that the debug/status/error registers (above) are at their expected values.
- Toggle ARQ to check if a configuration problem in ARQ sub-system is preventing audio from functioning properly.
- Verify that there is sufficient link bandwidth available for the audio stream.

15.5 Feature Availability by Device Family

All GMSL2 devices have audio capabilities except some CSI-2 parts. The physical connections and capabilities may vary by device family. See [Table 58](#) for details.

Table 58. Audio Support by Device Family

Device Family	I ² S Input Pin Sets	I ² S Output Pin Sets	Audio Tx Blocks	Audio Rx Blocks	HDMI Audio
HDMI Serializers	1	1	2	1	Yes
CSI-2 Serializers	0	0	0	0	No
Advanced CSI-2 Serializers	0	0	0	0	No
CSI-2 Camera Deserializers	0	0	0	0	No
oLDI Deserializers	1	1	1	1	No
CSI-2 Quad Deserializers	0	0	0	0	No

Bidirectional Channels

16. I²C/UART

16.1 Overview

I²C and UART are serial communication protocols supported by all GMSL2 devices. The control channel (CC) and pass-through (tunneling) channels utilize these protocols to transmit control information from the host controller to the GMSL2 devices and connected peripherals from either end of the serial link.

The CC provides access to both the internal registers of GMSL2 devices and connected peripheral devices; the pass-through I²C/UART channels provide a direct connection to remote I²C/UART peripherals but do not provide access to internal GMSL2 device registers.

16.2 Primary Control Channel (I²C/UART)

16.2.1 Overview

The I²C/UART control channel (CC) provides a main microcontroller (μC) with access to GMSL2 device registers and connected peripheral devices from either end of the serial link. Typical applications use one main μC ([Figure 52](#)). I²C multi-main applications are also supported, provided that a software arbitration method is used to prevent collisions ([Figure 53](#)). Note that the serial link allows only one main μC to communicate at any given time. The CC also enables the main μC to configure peripherals connected at the other end of the serial link (that is, remote side). This application requires that both GMSL2 devices are configured for the same CC protocol (that is, both I²C or both UART); GMSL2 devices do not support I²C-to-UART or UART-to-I²C conversion.

Note: In CC applications, the local GMSL2 device is connected to the μC; the remote GMSL2 device is connected to the remote peripheral I²C/UART device (see [Figure 52](#)).

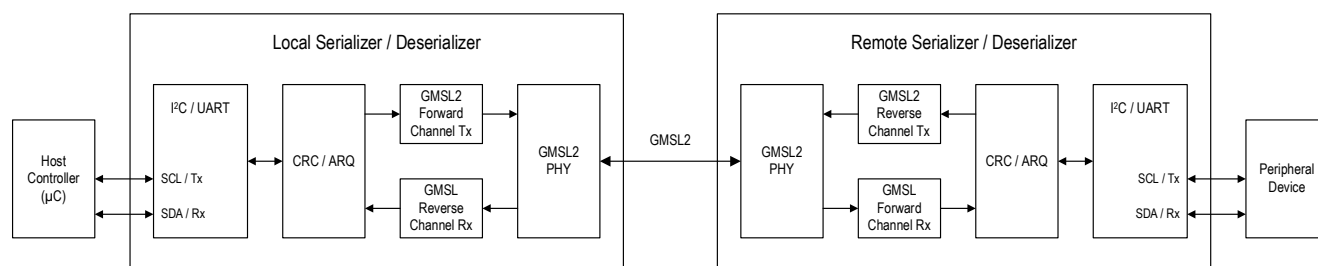


Figure 52. Typical GMSL2 Serial Link System with the Control Channel

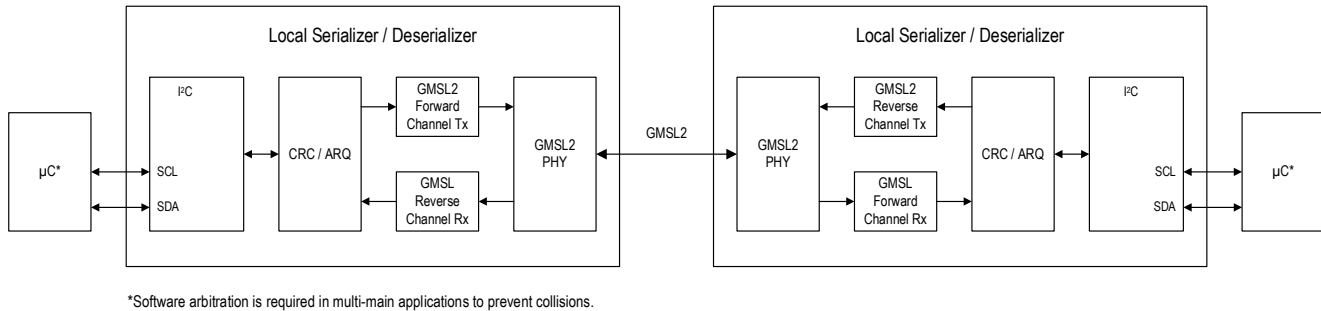


Figure 53. GMSL2 System with Multi-Main Control Channel (I²C)

Both CC protocols use the GMSL2 device's SDA_RX and SCL_TX pins. Selection of I²C (SDA and SCL) or UART (Rx and Tx) is determined at power-up with the I²CSEL pin state or the configuration pin (CFG). Refer to the device-specific data sheet for the protocol selection settings; note that some parts may feature pins with shared functionality. The I²C/UART outputs are open-drain and require external pullup resistors to ensure proper operation. The value of the external pullup resistors is application-specific and is determined by the characteristics of the connected µC or other peripherals and the value of VDDIO.

16.2.2 Operation

All GMSL2 device registers are accessible with an external µC through the control channel. GMSL2 devices use 16-bit register addressing. Protocol details are provided (see [I²C Data Transfer Format](#) and [UART Frame Format](#)).

Note: GMSL1 devices use 8-bit register addressing; GMSL2 devices use 16-bit register addressing.

The CC protocol is selected by setting the I²CSEL pin or CFG pins to the required value with a resistor network. The pin level is latched at power-up and selects either I²C or UART operation. Refer to the device-specific data sheet for configuration details.

The GMSL2 device directly connected to the µC that programs the serial link system and accesses remote peripherals through the CC is called the local device. The GMSL2 device on the opposite side of the serial link of the local device is called the remote device. See [Figure 52](#).

Remote-side devices are connected to the CC of local GMSL2 device when the link is locked. To configure remote devices over the serial link, both the serializer and deserializer must be configured to the same protocol (that is, I²CSEL or CFG settings must match for both devices).

Each device on the I²C/UART CC must have a unique device address that is used as the I²C/UART subordinate address. The voltage level of the ADD or CFG pins (depending on availability) at power-up sets the initial GMSL2 device address and the default value of the TX_SRC_ID registers for bidirectional channels. After power-up, the GMSL2 device address can be changed by writing to the DEV_ADDR register. If a device address is changed, all subsequent transactions with the device must use the new address.

In general, only one μC (I^2C or UART main) should be connected to either the serializer or deserializer CC pins (SDA/RX, SCL/TX) at any one time within a serial link system. If both the serializer and deserializer within a single system are connected to μCs and the remote CC is not disabled, the two μCs must take turns using the CC. Without arbitrated access, concurrent transfers from each μC result in a NACK (I^2C) or a collision (UART).

Note: Multi-main UART applications have many design risks and are not recommended.

Remote-side control channel access is blocked by setting `DIS_REM_CC` = 1 in both devices. When remote access is disabled, GMSL2 device access is limited to locally connected μCs , and CC access to remote-side device(s) is disabled.

A GMSL2 device can be set to read-only on the CC by setting the `CFG_BLOCK` register to 1. After setting a device to read-only, it cannot be configured through the CC until the device powers down (`PWRDNB` = 0 or V_{DD} = 0). This bit can be used to freeze device configuration.

Applications may require remote access to GMSL2 device registers while blocking I^2C /UART transmissions from being repeated at the remote device's CC pins. Remote device CC pins are disabled by setting `DIS_LOCAL_CC` = 1 in the remote device. When disabled, these pins can be used for another function (example, GPIO).

Note: Some devices with reduced pin counts share CC and pass-through functionality on the same set of pins. Extreme care must be taken when enabling and disabling the shared pins with respect to the channel being used. These devices must have `DIS_LOCAL_CC` = 1 set to use the pins for pass-through I^2C /UART. Refer to device-specific data sheets for pinout information.

By default, I^2C /UART CC transmission on the serial link are protected by automatic repeat request (ARQ) for error correction. I^2C /UART CC ARQ and packet CRC can only be disabled by a local register write in both devices while the link is not locked. See the [CRC Error Detection and ARQ Error Correction](#) section for details.

Note: Disabling ARQ and packet CRC for the I^2C /UART CC is discouraged for field applications and should only be used for debugging or testing purposes.

16.2.3 I^2C Control Channel

The I^2C is a bidirectional communication protocol that connects multiple devices through a single two-wire bus. Microcontrollers can be programmed to generate I^2C data transfers that GMSL2 devices can process. The following sections contain configuration information.

16.2.3.1 I^2C Mode Configuration

The following tables and sections contain registers associated with the I^2C control channel (CC). Note that the `I2CSEL`, `DIS_REM_CC`, `DIS_LOCAL_CC`, and `CFG_BLOCK` bitfields are common to the I^2C and UART control channel protocols.

Note: Registers listed in [Table 59](#), [Table 60](#), and [Table 61](#) may not exist in all parts. Refer to device-specific data sheets and register documents for the most accurate part information.

Table 59. I²C/UART Selection Status Register

BITFIELD	DESCRIPTION	DECODE
I ² CSEL[0]	This bit is set according to the latched I ² CSEL/CFG pin value at power-up.	0b0: UART 0b1: I ² C

Note: I²CSEL[0] should only be used as a status bit, and writing to this bitfield is discouraged. It is not recommended to change the CC from I²C to UART or vice versa by writing to this register after power-up.

Table 60. Remote/Local Control Channel and Device Configuration Registers

BITFIELD	DESCRIPTION	DECODE
DIS_REM_CC[0]	Disables the remote-control channel over the GMSL2 link.	0b0: Remote control-channel enabled 0b1: Remote control-channel disabled
DIS_LOCAL_CC[0]	Disables control-channel connection to RX/SDA and TX/SCL pins.	0b0: RX/SDA and TX/XCL connected to control channel 0b1: RX/SDA and TX/SCL disconnected from control
CFG_BLOCK[0]	Configuration block. When set, all registers become nonwritable (read-only). This bit can be used to freeze the chip configuration.	0b0: Not Blocked 0b1: Blocked

Table 61. Number of I²C-to-I²C Links Configuration Registers

BITFIELD	DESCRIPTION	DECODE
I ² C_AUTO_CFG[0]	When set to 1, I ² C-to-I ² C number of links is automatically determined based on splitter mode. In splitter mode, response from two I ² C channels are expected, otherwise response from one I ² C channel is expected.	0b0: Number of I ² C-to-I ² C links set by I ² C_SRC_CNT[2:0] bits 0b1: Splitter mode automatically determines the number of I ² C-to-I ² C links
I ² C_SRC_CNT[2:0]	I ² C-to-I ² C number of links (valid when I ² C_AUTO_SRC = 0). Set this field to N - 1 when expecting I ² C response from N remote I ² C transmitters (usually the same as the number of remote devices connected to this device).	0b000: 1 serializer/deserializer connected 0b001: 2 serializers/deserializers connected 0b010: Reserved 0b011: Reserved 0b100: Reserved 0b101: Reserved 0b110: Reserved 0b111: Reserved

16.2.3.1.1 I²C Internal Register Access

Each GMSL2 device has an internal I²C subordinate for register access. The internal registers can be written and read according to the I²C protocol using the data transfer formats. Register addresses are 16-bits wide. Single or multiple data bytes can be written or read (by address auto-increment). I²C data transfers can be monitored with the read-only registers for I²C acknowledge bits and time-out status (Table 62).

Note: Devices have auto-increment limits. Refer to device-specific data sheets for information.

16.2.3.1.1.1 I²C Data Transfer Formats

Note: In Figure 54 and Figure 55, unshaded blocks indicate data transfers from the main to the subordinate and shaded blocks indicate data transfers from the subordinate to the main.

I²C Write:

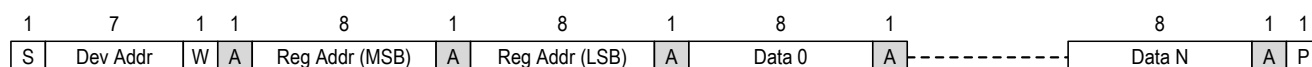


Figure 54. I²C Write Data Transfer Format

I²C Read:

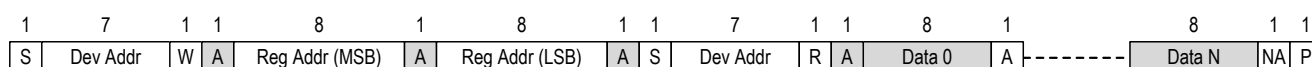


Figure 55. I²C Read Data Transfer Format

Table 62. I²C Acknowledge Bit and Time-Out Status (Read-Only Registers)

BITFIELD	DESCRIPTION	DECODE
REM_ACK_ACKED[0]	Inverse of the I ² C acknowledge bit received from remote side.	0b0: I ² C acknowledge bit received as 1 0b1: I ² C acknowledge bit received as 0
REM_ACK_RECVD[0]	I ² C acknowledge bit for any I ² C byte is received from the remote side for the previous GMSL packet with I ² C data.	0b0: I ² C acknowledge bit not received 0b1: I ² C acknowledge bit received
I ² C_TIMED_OUT[0]	Internal I ² C-to-I ² C subordinate or main timed out while receiving data transfer from remote device.	0b0: Time-out has not occurred 0b1: Time-out has occurred

16.2.3.1.2 I²C over the GMSL2 Link (Remote Device Access)

The I²C channel connects the serializer and deserializer I²C interfaces (that is, SDA_RX and SCL_TX) over the serial link.

In addition to the internal I²C subordinate used for register access, GMSL2 devices have I²C link mains and link subordinates used for remote device access through I²C transmissions over the GMSL2 link. When an external I²C main (example, μ C) performs an I²C transaction on the I²C bus on one side of the link (that is, local side), the I²C events (example, Start, Stop, Data Bit 0, and Data Bit 1) are forwarded to the other side of the link (that is, remote side) by the local-side device's I²C link subordinate. The I²C events are received by the remote-side device's I²C link main and generated on the remote-side I²C bus. The remote-side I²C link main sends back across the link any I²C events that are expected to be driven by the remote-side I²C subordinate(s) (example, Ack bits during writes and read data bits during reads) to the local-side I²C link subordinate.

I²C over the GMSL2 link operates such that the remote-side internal I²C main mimics the actions of the local-side external I²C main (example, μ C) and the local-side internal I²C link subordinate mimics the actions of the external I²C subordinate(s) on the remote side. This method logically connects the two separate I²C buses: an entire I²C transaction looks like as if it has been performed on the same physical I²C bus (except for incurred timing differences).

I²C data transfers require an immediate acknowledgement from the receiver following each byte. To account for timing differences between the main and subordinate and to allow time for data to be forwarded and received across the serial link, the I²C protocol uses clock stretching (that is, holding SCL low) to temporarily pause communication as the acknowledge propagates through the I²C control channel. In GMSL2 systems, all I²C devices on the local side (that is, external main μ C and any attached peripherals) must support clock stretching; the remote-side I²C peripherals are not required to support clock stretching ([Figure 56](#)).

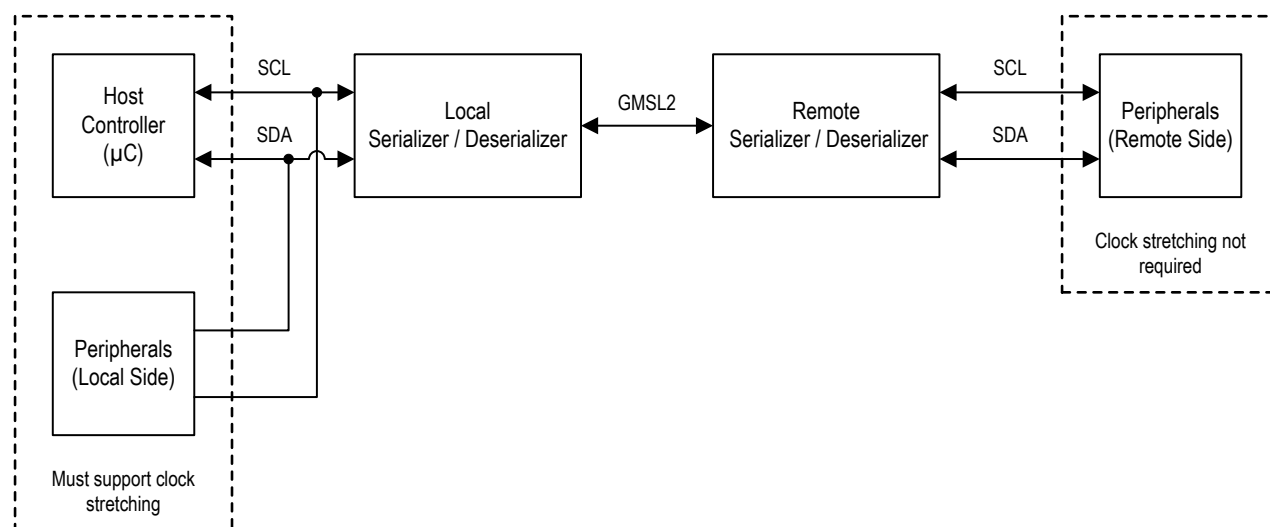


Figure 56. I²C Clock Stretching

The remote-side link I²C main operates according to timing settings configured with the remote-side **MST_BT** bitfield. It is strongly recommended that users program the **MST_BT** bitfield as close as possible to the bit rate used by the external main (example, μ C). The local-side I²C subordinate timing is configured with the **SLV_SH** register. The **SLV_TO** and **MST_TO** bitfields select the time-out durations that release the local or remote side I²C bus in case an expected response from remote side is not received within the selected time-out duration. Ensure that all timing registers ([Table 63](#) and [Table 64](#)) are programmed according to the desired I²C bit rate and in compliance with the official I²C timing parameters.

Table 63. Internal I²C Subordinate Configuration Registers

BITFIELD	DESCRIPTION	DECODE
SLV_SH[1:0]	I ² C-to- I ² C subordinate setup and hold time setting (setup, hold). Configures the interval between SDA and SCL transitions when driven by the internal I ² C subordinate.	0b00: Set for I ² C Fast-mode Plus speed 0b01: Set for I ² C Fast-mode speed 0b10: Set for I ² C Standard-mode speed 0b11: Reserved
SLV_TO[2:0]	I ² C-to- I ² C subordinate time-out setting. Internal GMSL2 I ² C subordinate times out after the configured duration if it does not receive any response while waiting for a packet from the remote device.	0b000: 16us 0b001: 1ms 0b010: 2ms 0b011: 4ms 0b100: 8ms 0b101: 16ms 0b110: 32ms 0b111: Disabled

Table 64. Internal I²C Main Configuration Registers

BITFIELD	DESCRIPTION	DECODE
MST_BT[2:0]	I ² C-to- I ² C main bit rate setting. Configures the I ² C bit rate used by the internal I ² C main (in the device on the remote side from the external I ² C main). Set this according to the I ² C speed mode.	0b000: 9.92Kbps - Set for I ² C Standard- mode speed 0b001: 33.2Kbps - Set for I ² C Standard-mode speed 0b010: 99.2Kbps - Set for I ² C Standard- or Fast-mode speed 0b011: 123Kbps - Set for I ² C Fast-mode speed 0b100: 203Kbps - Set for I ² C Fast-mode speed 0b101: 397Kbps - Set for I ² C Fast- or Fast-mode Plus speed 0b110: 625Kbps - Set for I ² C Fast-mode Plus speed 0b111: 980Kbps - Set for I ² C Fast-mode Plus speed
MST_TO[2:0]	I ² C-to- I ² C main time-out setting. Internal GMSL2 I ² C main times out after the configured duration if it does not receive any response while waiting for a packet from remote device.	0b000: 16us 0b001: 1ms 0b010: 2ms 0b011: 4ms 0b100: 8ms 0b101: 16ms 0b110: 32ms 0b111: Disabled

The external I²C main (example, μ C) can be located on the serializer-side (typically for display applications) or the deserializer-side (typically for camera applications). Multi-microcontroller operations are supported provided that a software arbitration method is used to prevent collisions (see [I²C Multi-Main Options](#)). The serial link assumes that only one microcontroller is transmitting at any given time.

16.2.3.1.3 I²C Splitter Mode

The I²C channel can be used in GMSL2 I²C splitter mode applications (that is, a single serializer connects to two deserializers). In GMSL2 splitter mode:

- The μ C connected to the local device in splitter mode can communicate with each connected remote device and the attached peripherals.
- The μ C connected to a remote device can communicate with the local device in splitter mode and its attached peripheral. It cannot communicate with the other remote device(s).
- If more than one μ C is connected to the serial link system, only one μ C can use the control channel at a time. The serial link assumes that only one μ C is using the CC at any one time. Multiple μ Cs must take turns using the control channel to avoid error conditions. If the CC is in use by a μ C, attempts from another μ C to access the CC result in a NACK. If software arbitration is not implemented, the remote-side controller should be blocked to avoid collisions.

Note: The I²C channel can also operate in GMSL2 [Reverse Splitter Mode](#). The operation is the same as splitter mode but reversed.

I²C address configuration options are presented in [Table 65](#). Each function is described in more detail in following sections. Note that these functions may be used in combination.

Table 65. I²C Address Configuration

FEATURE	DESCRIPTION	PURPOSE	NOTES
I²C Address Reassignment	Allows users to configure a GMSL2 device's I ² C address from the default address at power-up (determined by the CFG pin).	Used to assign unique I ² C addresses to GMSL2 devices when a GMSL2 system comprises multiple devices with the same default address.	In splitter and reverse splitter applications, I ² C address reassignment is used to avoid address conflicts and data collisions.
I²C Address Translation	Configurable mapping of one I ² C address to another (virtual) I ² C address. Address translation occurs within the GMSL2 device.	Map a virtual I ² C address to a peripheral I ² C device on the remote side of the GMSL2 link. This allows a host μ C to separately access different devices and peripherals with the same device address through a unique virtual I ² C address.	The mapped virtual I ² C address must be unique within the system.

I²C Broadcasting	Allows multiple GMSL2 devices to be addressed simultaneously at a single I ² C address.	Simplifies configuration by allowing a single write to simultaneously configure multiple devices as a group to identical settings.	It is not possible to determine if all devices provide an acknowledge following an I ² C broadcast write. Each device must be individually checked to verify correct configuration.
------------------------------------	--	--	--

16.2.3.1.4 I²C Address Reassignment

Note: The methodology discussed in this section also applies to UART.

In splitter and reverse splitter mode applications, it is recommended to use a unique I²C device address for each connected serializer/deserializer. I²C address reassignment is required if identical device addresses are selected at power-up.

Note: The follow configuration procedures do not apply to GMSL2 quad deserializer devices.

16.2.3.1.4.1 Camera Setup – Two Serializers to One Deserializer

In systems with two camera modules connected to one deserializer, address reassignment should be used to avoid address conflicts and data collisions.

This procedure applies to a system comprising two identical camera modules connected to a dual CSI-2 camera deserializer, with the microcontroller on the deserializer side.

1. Isolate one camera module by configuring the deserializer into single-link mode. For example, enable link A only by setting `LINK_CFG[1:0]` to 0b01 and `AUTO_LINK[0]` low on the deserializer.
2. Perform a link reset by writing the self-clearing bit `RESET_ONESHOT` high.
3. Poll the LOCKED pin until it goes high.
4. Modify the serializer I²C device address connected to link A with a register write to `DEV_ADDR[6:0]` located in `REG0`.
5. Modify each of the source identifiers (`TX_SRC_ID`) for each of the GMSL protocol packets to a value unique relative to the other serializer in the system.
6. Configure the deserializer to reverse splitter mode by writing `LINK_CFG[1:0]` to 0b11.
7. Perform a link reset by writing the self-clearing bit `RESET_ONESHOT` high.
8. Poll the LOCKED pin until it goes high.
9. All devices should be present on the I²C bus. Continue with any additional system configuration.

16.2.3.1.5 I²C Address Translation

Address translation is a function in I²C mode that enables mapping one device address to a virtual device address. In GMSL2 serial link systems, two separate device addresses can be translated to another two separate device address. This function is used when two identical modules with the same device address are used within the same system. Address translation allows the host μ C to separately access different devices and peripherals with the same device address through software configuration. Configuration details are presented in [Table 66](#).

Note: There may be more than two devices/modules with the same address used with GMSL2 Quad Deserializer systems.

Table 66. I²C Address Translator Configuration Registers

BITFIELD	DESCRIPTION	DECODE
SRC_A[6:0]	I ² C address translator source A. When I ² C device address matches I ² C SRC_A, internal I ² C main (on remote side) replaces the device address by I ² C DST_A.	0bXXXXXXXX: Value of I ² C SRC_A
DST_A[6:0]	I ² C address translator destination A. See the description of I ² C SRC_A.	0bXXXXXXXX: Value of I ² C DST_A
SRC_B[6:0]	I ² C address translator source B. When I ² C device address matches I ² C_SRC_B, internal I ² C main (on remote side) replaces the device address by I ² C_DST_B.	0bXXXXXXXX: Value of I ² C SRC_B
DST_B[6:0]	I ² C address translator destination B. See the description of I ² C SRC_B.	0bXXXXXXXX: Value of I ² C DST_B

Address translation can be used in addition to address reassignment (see [Camera Setup – Two Serializers to One Deserializer](#)) for systems with two identical camera modules connected to one deserializer. Address translation allows two camera modules with image sensors at the same I²C address to be independently addressed at user-defined virtual addresses to avoid address conflicts. GMSL2 devices allow for up to two address translations (permitting two devices to have their addresses translated). In this scenario, I²C address translation is configured for the serializer device. I²C commands are sent from the deserializer over the serial link to the serializer. In the serializer, address [SRC_A\[6:0\]](#) is then translated into address [DST_A\[6:0\]](#) by the remote link main ([Table 66](#)). Both the serializer and the peripheral(s) see this translated address (that is, the [DST_A](#) address).

Note: An unused address translation can be used for I²C broadcasting. See the [I²C Broadcasting](#) section for details.

This procedure provides the configuration steps for two identical camera modules connected to a CSI-2 deserializer. The μ C is located on the deserializer side ([Figure 28](#)).

- Use address reassignment to modify one serializer's I²C device address so that each serializer has a unique address. See address reassignment procedure given (see [Camera Setup – Two Serializers to One Deserializer](#)). Ensure that the deserializer is in reverse splitter mode and that all devices are present on the I²C bus before proceeding with the following steps.
- Program the address translation registers in the (link A) serializer by setting the desired image sensor address in the source register [SRC_A\[6:0\]](#) and the original image sensor address in the destination register [DST_A\[6:0\]](#).
- Program the address translation registers in the (link B) serializer by setting the desired image sensor address in the source register [SRC_A\[6:0\]](#) and the original image sensor address in the destination register [DST_A\[6:0\]](#).

16.2.3.1.5.1 I²C Address Translation Example

An I²C address reassignment and translation example is shown in the block diagram (Figure 57). A GMSL2 CSI-2 deserializer is connected to two identical camera modules. Each camera module comprises a GMSL2 serializer at I²C address 0x80 and an image sensor at address 0x6C.

I²C address reassignment is used to change the link A serializer's device address from 0x80 to 0x84. Then, I²C address translation is performed on each serializer to allow individual access of each image sensor. The I²C address for the link A serializer is translated from 0x20 to 0x6C; the I²C address for the link B serializer is translated from 0x22 to 0x6C.

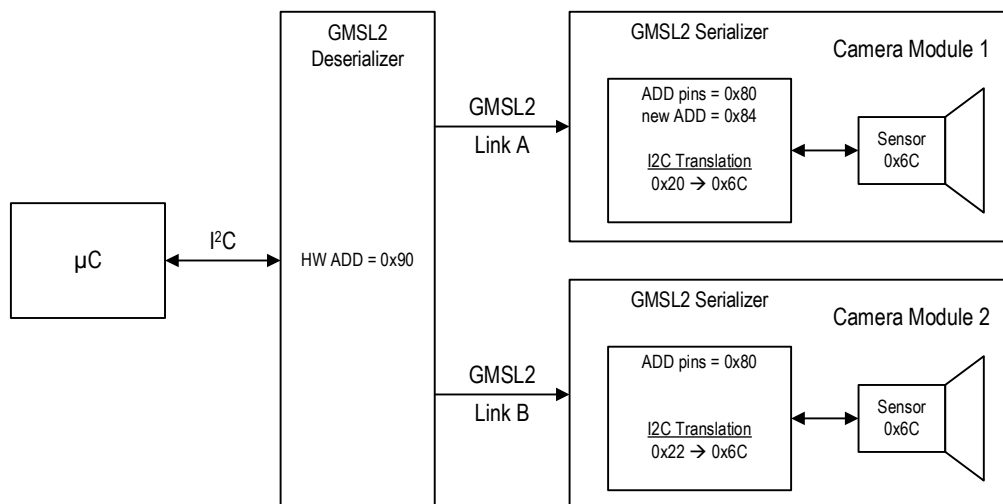


Figure 57. I²C Address Translation

16.2.3.1.6 I²C Broadcasting

I²C broadcasting can be used to simplify programming in GMSL2 systems consisting of a deserializer connected to two identical camera modules. I²C broadcasting enables the deserializer to configure the connected camera modules as a group to identical settings. The deserializer uses the I²C address of the image sensors assigned at power-up and the I²C address of the serializers assigned through I²C address translation as the broadcasting address.

Additional configuration may be necessary depending on the system application. If individual configuration of the image sensors is also required, each must be given a unique address with [I2C Address Translation](#). This allows the deserializer to access the image sensors both as a group through I²C broadcasting and individually with the unique addresses.

Note: To assign unique addresses to each image sensor, an I²C address translation must be performed on each device.

To avoid address conflicts and data collisions, use [I2C Address Reassignment](#) to assign a unique I²C address to each serializer (see [Camera Setup – Two Serializers to One Deserializer](#)). If serializers with unique addresses share many configuration settings, address translation can be used to assign a broadcasting address so that the serializers can be configured as a group to identical settings by the

deserializer. Note that programming a broadcasting address for both serializers requires one address translation per serializer.

Note: Analog Devices, Inc., cannot guarantee the programming of all devices. Each device must be individually checked to verify correct configuration, as Analog Devices cannot guarantee or distinguish which device provides the acknowledge. If one subordinate pulls the SDA line low, it can mask the state of the other subordinates. Analog Devices cannot guarantee that all subordinates pulled SDA low. Therefore, the user must implement a mechanism to verify that each subordinate is properly configured.

This procedure provides the configuration steps for two identical camera modules connected to a CSI-2 deserializer. The μ C is located on the deserializer side ([Figure 58](#)).

1. Use address reassignment to modify one serializer's I²C device address to have a unique address for each serializer. See the [I²C Address Reassignment](#) procedure given (see [Camera Setup – Two Serializers to One Deserializer](#)). Ensure that the deserializer is in reverse splitter mode and that all devices are present on the I²C bus before proceeding with the following steps.
2. Modify the first address translation register in the (link A) serializer to give a broadcast address to the serializer. Set the desired broadcasting address in the source register `SRC_A[6:0]` and the modified serializer address at Step 1 in the destination register `DST_A[6:0]`.
3. Modify the second translation register in the (link A) serializer to give a unique address to the image sensor. Set the desired image sensor address in the second source register `SRC_B[6:0]` and the original image sensor address in the second destination register `DST_B[6:0]`.
4. Modify the first address translation register in the (link B) serializer to give a broadcast address to the serializer. Set the desired broadcasting address in the source register `SRC_A[6:0]` and the original serializer address in the destination register `DST_A[6:0]`.
5. Modify the second translation register in the (link B) serializer to give a unique address to the image sensor. Set the desired image sensor address in the second source register `SRC_B[6:0]` and the original image sensor address in the second destination register `DST_B[6:0]`.

16.2.3.1.6.1 I²C Broadcasting Example

An example of I²C address reassignment, translation, and broadcasting is shown in the block diagram ([Figure 58](#)). A GMSL2 CSI-2 deserializer is connected to two identical camera modules. Each camera module comprises a GMSL2 serializer at I²C address 0x80 and an image sensor at address 0x6C.

I²C address reassignment is used to change the link A serializer's device address from 0x80 to 0x84. Then, I²C address translation is performed twice on each serializer to allow individual access of each image sensor and configure an I²C broadcasting address for the serializers. For individual access of the image sensors, the I²C address for the link A serializer is translated from 0x20 to 0x6C, and the I²C address for the link B serializer is translated from 0x22 to 0x6C. The serializers have an additional I²C translation programmed to establish the I²C broadcasting address. Here, the broadcasting address is set as the source register, and the serializer device address is set as the destination register. In the link A serializer, 0xC4 is translated to 0x84 (that is, the modified device address configured with I²C address reassignment); in the link B serializer, 0xC4 is translated to 0x80.

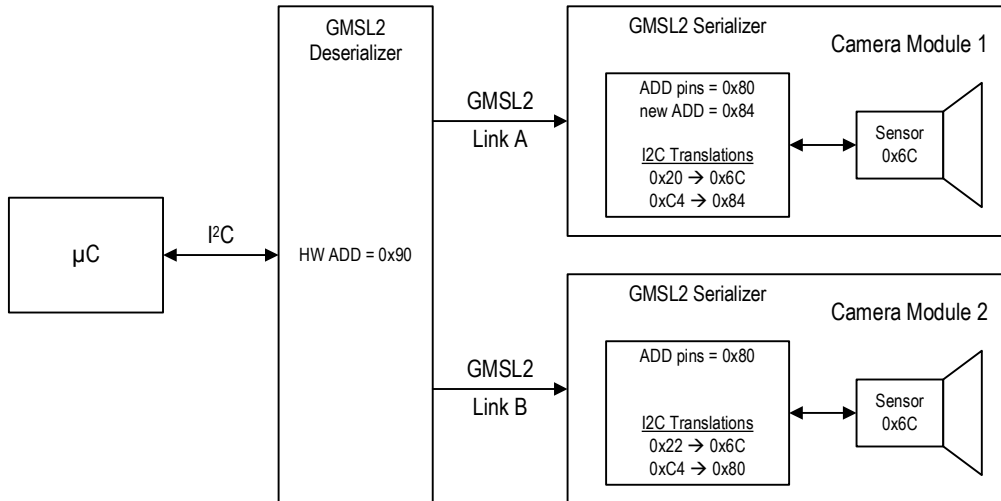


Figure 58. I²C Broadcasting

16.2.3.2 I²C Multi-Main Options

GMSL2 systems can support multi-main I²C applications. However, measures to avoid conflicts/collisions must be implemented to ensure proper operation. There is no officially supported measure; options are provided as follows:

- **Islanding:** The remote CC can be disabled to separate the two µCs. See [Disabling Remote Control Channel on Power-Up](#) for details.
- **Pass-through Channel:** Depending on the application and its requirements, one of the pass-through I²C channels can be used to separate the two I²C mains.
- **Single Main:** One of the µCs is designated as the main. Only the main sends I²C commands through the GMSL2 link. Alternatively, software arbitration can permit a second main to send I²C commands (after a delay) when the first main is not active.
- **Token Pass:** Similar to single main, except that the main also sets up the GPIOs. The subordinate can request control through one of the GPIOs, and the main can notify the subordinate that control is allowed through a different GPIO. Note that most GPIOs are low by default, and polarity must be carefully chosen to ensure that the subordinate µC waits if the GPIOs are not yet programmed.

If one of the µCs does not need to program GMSL2 devices, the pass-through I²C channel should be used instead of the I²C CC.

Note: Multi-main UART applications have many design risks and are not recommended.

16.2.3.4 I²C Channel GMSL2 Bandwidth Utilization

See the [GMSL2 Link Bandwidth Consumption from Side Channels](#) I²C section for details.

16.2.3.5 I²C Debug Techniques

The I²C protocol is used by most μ Cs. Ensure that μ Cs are programmed to generate I²C data transfers that GMSL2 devices can process and respond to I²C data transfers from GMSL2 devices.

If I²C issues are exhibited, inspect the I²C port(s) using a logic analyzer or digital oscilloscope. Verify that the waveforms observed on the SDA and SCL pins are as expected.

The `REM_ACK_RECVD` register is used to check if an I²C Ack bit for any I²C byte has been received from the remote side for the previous I²C data transfer. Alternatively, a logic analyzer or digital oscilloscope can be used to monitor if I²C Ack bits have been received from the remote side for any transmitted I²C byte.

The `REM_ACK_ACKED` register can be read back to see if the received Ack is 1 or 0.

The `I2C_TIMED_OUT` register can be read back to check if the internal I²C–I²C link subordinate or link main has timed-out while receiving data from the remote device.

16.2.4 Main UART Control Channel

UART is typically a point-to-point communication protocol; however, in GMSL2 systems, it is possible to have microcontroller(s), serializer(s), deserializer(s), and peripheral subordinate(s) physically connected to the same bus. This expanded application of UART involves several important restrictions with ramifications on system design.

16.2.4.1 UART Mode Configuration

The following tables and sections contain registers that are associated with the main UART control channel. Note that the I²C `SEL`, `DIS_REM_CC`, `DIS_LOCAL_CC`, and `CFG_BLOCK` registers are common to the main I²C and UART control channel protocols. See the [I²C Mode Configuration](#) section for information regarding the shared registers ([Table 59](#) and [Table 60](#)).

Devices with multiple independent GMSL2 links require that UART programming be performed on a link-by-link basis because UART communication is not broadcast across all links. Program the `UART_0_LINK_SELECT`, `UART_1_LINK_SELECT` and `UART_2_LINK_SELECT` registers accordingly. This contrasts with I²C, which can be used to communicate over any number of available links simultaneously.

Note: Registers listed in the following sections and tables may not exist in all parts. Refer to device-specific data sheets and register documents for the most accurate part information.

16.2.4.1.1 UART Base/Bypass Mode

GMSL2 devices provide two main UART CC modes of operation: UART Base Mode and UART Bypass Mode. Base Mode is used to program and configure the serializer and deserializer. The μ C can

communicate with both GMSL2 devices and attached (compatible) peripherals on the remote side. Bypass Mode is used to bypass GMSL2 devices and provides direct point-to-point access to a peripheral device. The UART CC can be programmed to transition between these modes, allowing flexible use of the UART CC.

UART Bypass Mode is distinct from Pass-through UART: Pass-through UART only provides access to remote peripherals (internal GMSL2 device registers are inaccessible), while UART Bypass Mode is a control channel mode. If a GMSL2 system includes a peripheral connected to the remote device that is not compatible with the GMSL2 UART protocol, UART Bypass Mode allows a μ C to communicate with this peripheral without requiring the use of the UART Pass-through connection. After the communication is complete, the μ C can access GMSL2 devices in UART Base Mode. Pass-through UART does not have access to GMSL2 devices.

16.2.4.1.1.1 UART Base Mode

UART Base Mode is enabled by default at power-up. In base mode, μ Cs are the host and use the GMSL2 [UART Frame Format](#) to write and read the internal registers of GMSL2 devices from either side of the serial link. The μ C can also communicate with attached remote peripherals compatible with the GMSL2 UART protocol. UART data transmitted by the μ C is output from the remote-side GMSL2 device Tx pin by default. To disable the UART Tx and Rx pins on remote-side GMSL2 device, set `DIS_LOCAL_CC` = 1 in the remote GMSL2 device.

Single or multiple bytes can be written or read using the GMSL2 UART protocol. Between each UART transmission, the μ C must wait up to 48-bits time or 200 μ s, whichever is longer, for the expected response from the GMSL2 device. This logically enforces half-duplex control channel operation. After completely receiving the response to a transmission, the μ C must wait and keep the line high for at least 48-bits times before sending the next transmission. The high-time duration sums up to 49 bits with the stop bit of previous data transfer included. Note that response to a read or write request (including Ack frame) is also part of the data transfer, and the 48-bits time wait starts after the stop bit of the last UART frame of the response.

The first received UART byte of a transmission (that is, sync or acknowledge frame) can be configured to be delayed by 0-bit, 1-bit, 4-bits, or 8-bits time with the `OUT_DELAY` register () in UART base mode. This programming can be used to ensure that UART frames of the same data transfer are output one after the other on the remote side.

Table 67. UART Initial Output Delay Configuration Register

BITFIELD	DESCRIPTION	DECODE
OUT_DELAY[1:0]	UART initial output delay. In base mode, the first received UART byte of a packet (sync or acknowledge frame) is delayed by the configured number of bit times to output the UART frames of the same packet back-to-back on remote side.	0b00: 0 bits 0b01: 4 bits 0b10: 8 bits 0b11: 1 bit

16.2.4.1.1.2 UART Bypass Mode

In UART bypass mode, UART commands are not interpreted by the GMSL2 devices, and data is passed directly from the μ C to connected peripherals (and from peripherals to the μ C). The μ C cannot access GMSL2 device registers. There are two methods to enable UART bypass mode: with the **BYPASS_EN** register bit or with the MS pin (shared with GPIO). Configure the MS pin method of UART bypass mode enable with **REM_MS_EN** and **LOC_MS_EN** ().

Note: Ensure that the GPIO pins with MS pin assignments are available for use before configuring the system for MS pin control of UART bypass mode. Refer to device-specific data sheet for pinout information.

Note: UART bypass mode should ONLY be enabled when GMSL2 link lock is preset.

UART bypass mode is implemented with software by setting the **BYPASS_EN** register in the remote device first, then in the local device. This programming can be temporary or permanent. In temporary programming, bypass mode is automatically exited and the **BYPASS_EN** register is reset to 0 when the UART line stays high beyond the time-out duration defined with the **BYPASS_TO** register. **BYPASS_TO** has four settings: 2ms, 8ms, 32ms, and “Disabled”. Permanent programming is configured by setting **BYPASS_TO** to “Disabled”: **BYPASS_EN** is never cleared, and the device stays in bypass mode until power is cycled. UART bypass mode configuration registers are presented in.

Note: Programming **BYPASS_TO** to “Disabled” is discouraged because this setting blocks control channel access to the GMSL2 devices.

Table 68. UART Bypass Mode Configuration Registers

BITFIELD	DESCRIPTION	DECODE
REM_MS_EN[0]	Enables UART bypass mode control by remote GPIO pin. When set, remote chip's GPIO is used as MS pin (UART Mode Select). Refer to the specific device's data sheet to see which GPIO has MS functionality. When MS is high, chip is in bypass mode, otherwise chip is in base mode.	0b0: UART bypass mode not controlled by remote MS pin 0b1: UART bypass mode controlled by remote MS pin
LOC_MS_EN[0]	Enables UART bypass mode control by local GPIO pin. Set to use relevant GPIO pin as MS pin (UART Mode Select). Refer to device data sheet or <i>UART Base/Bypass Mode Operation</i> section to see which GPIO has MS functionality. When MS is high, chip is in bypass mode, otherwise chip is in base mode.	0b0: UART bypass mode not controlled by local MS pin 0b1: UART bypass mode controlled by local MS pin
BYPASS_DIS_PAR[0]	Selects whether to receive and send parity bit in bypass mode.	0b0: Receive and send parity bit in bypass mode 0b1: Do not receive and send parity bit in bypass mode
BYPASS_TO[1:0]	UART soft bypass time-out duration. When set to 0b11, BYPASS_EN is never cleared, so the device stays in bypass mode until next power down.	0b00: 2ms 0b01: 8ms 0b10: 32ms 0b11: Disabled

BYPASS_EN[0]	Enables UART soft bypass mode. Bypass mode remains active if there is UART activity. When there is no UART activity for selected duration configured by BYPASS_TO register, device exits bypass mode, and the bit is automatically cleared.	0b0: UART soft bypass mode disabled 0b1: UART soft bypass mode enabled
---------------------	---	---

This procedure provides the steps required to control UART bypass mode with the MS pin.

Note: Refer to device-specific data sheet. In the procedure, the GPIO number corresponding to the MS pin is represented by *.

- Drive the MS pin low.
- If the remote device is an OLDI deserializer or an eDP/DP deserializer:
 - Set `GPIO_TX_ID_*` of the local device to 2.
- Else (remote device is not an OLDI deserializer or an eDP/DP deserializer):
 - Set `GPIO_TX_ID_*` of the local device to 29.
- Set `GPIO_TX_EN_*` = 1 in the local device.
- Set `REM_MS_EN` = 1 in the remote device.
- Set `LOC_MS_EN` = 1 in the local device.
- Repeat the following as needed:
 - Drive MS pin high from local side to switch to bypass mode.
 - Perform UART communication with the peripheral UART subordinate.
 - Drive MS pin low from local side to switch to base mode.
 - Perform UART communication with serializer or deserializer.

UART is in bypass mode when MS is high; UART is in base mode when MS is low.

16.2.4.1.2 UART Splitter Mode

The UART channel can be used in GMSL2 UART splitter mode applications (that is, a single serializer connected to two deserializers). In GMSL2 splitter mode:

- The μ C connected to the local device in splitter mode can communicate with each connected remote device and the attached peripheral(s).
- The μ C connected to a remote device can communicate with the local device in splitter mode and its attached peripheral(s). It cannot communicate with the other remote device(s) or attached peripheral(s).
- If a serial link system in splitter mode has multiple remote devices connected to μ Cs, the first μ C to communicate with the local device in splitter mode gets dedicated access to the UART link (the UART link from the other μ C is blocked at the splitter device). The UART link remains dedicated to that μ C if it maintains communication. When the μ C stops communication for the UART arbitration time-out duration (default = 2ms, programmable by `ARB_TO_LEN`), the link becomes available again and the process repeats ([Table 69](#)). Any μ C can attempt to communicate on the UART link until it is made available and is able to gain dedicated access.

Note: The UART channel can also operate in GMSL2 [Reverse Splitter Mode](#). The operation is the same as splitter mode but reversed.

See the [I²C Splitter Mode](#) section for shared configuration details.

Table 69. UART Rx Source Arbitration Time-Out Configuration Register

BITFIELD	DESCRIPTION	DECODE
ARB_TO_LEN[1:0]	UART RX source arbitration time-out duration. UART RX processes packets from a single UART source at any time. When UART RX does not receive any UART packets for this duration, it selects the next UART source according to the source ID of the next following received packet.	0b00: 1ms 0b01: 2ms 0b10: 8ms 0b11: 32ms

16.2.4.2 UART Frame Format

A regular UART frame with an even parity bit is used to carry one byte of data (*Figure 59*). UART frames consist of a low start bit, 8 data bits, a parity bit, and a high stop bit. The parity bit is high if the number of ones in 8-bit data is odd, otherwise it is low. There must be at least one high stop bit. If the next frame is in the same transmission, there can be at most four high bits from the end of the stop bit to the beginning of the next start bit. If there is a parity bit error, the transmission is discarded starting from the frame with the error.

The phase of the internal UART bit clock is adjusted using the start bit of each frame. The UART receiver asynchronously samples the incoming data at a higher rate than the actual data rate and establishes the period of the data such that it can correctly identify each bit (that is, the incoming data is quantized).

Note: UART rates up to 5Mbps are given sufficient bit-error protection from the internal clock recovery running at 150MHz (30x oversampling).

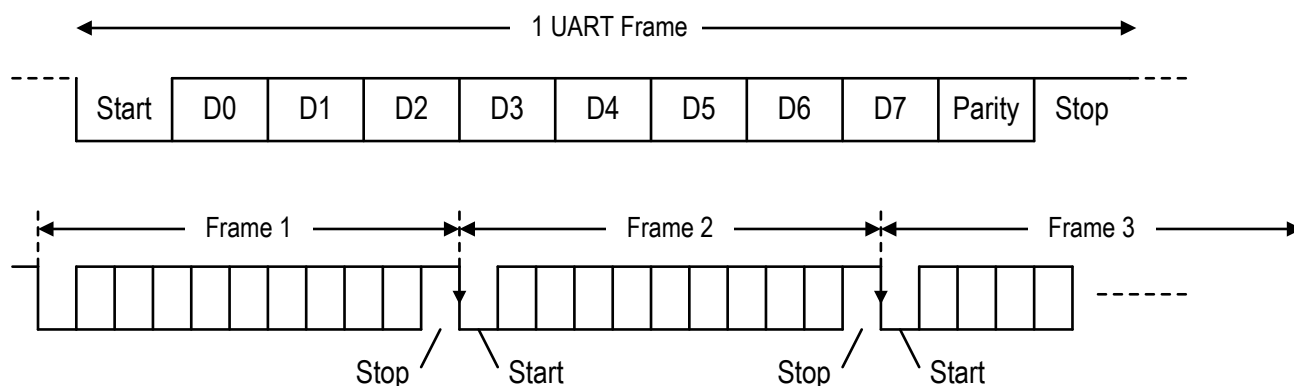


Figure 59. UART Frame Format

In bypass mode, the parity bit is enabled by default, but GMSL2 devices do not check frames for correct parity. Even or odd parity can be used. The parity bit is passed through the serial link along with the UART data so that the receiver can check for errors. The parity bit can be disabled in bypass mode by setting `BYPASS_DIS_PAR` to 1 before entering bypass mode.

The bit rate (baud rate) in bypass mode must be same as the last bit rate used in base mode.

16.2.4.2.1 UART Synchronization Frame

The UART bit rate is unknown to GMSL2 devices; the internal bit-length counters must be calibrated to correctly recover UART frames. GMSL2 devices use UART sync frames to calibrate the bit length in terms of the internal 150MHz clock. Each UART transmission begins with a sync frame (Figure 60). Sync frames are regular UART frames with a value of 0x79. They must be successfully detected to ensure that the remaining frames of the transmission are received correctly, as the data pattern of the sync frame is used to set the UART data bit rate for the rest of the data transfer. No transactions are allowed (the line must stay high) for a minimum of 48-bits time between UART transmissions.

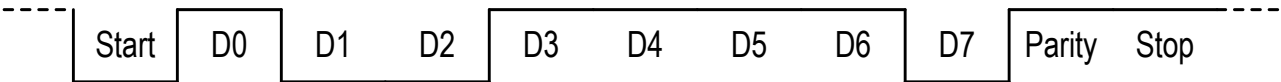


Figure 60. UART Synchronization Frame

16.2.4.2.2 UART Write Protocol

The UART write protocol consists of a 5-byte header followed by one or more data bytes (Figure 61). The LSB of the device address frame is 0. The addressed device responds with an Ack frame if no errors are detected, and the transmission is valid. The byte count indicates the number of data bytes to be written (N). This must be a non-zero number.

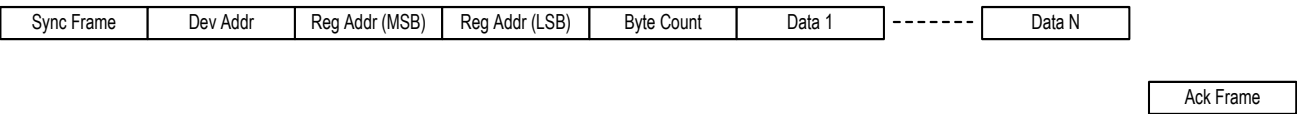


Figure 61. UART Write Protocol Format

16.2.4.2.3 UART Read Protocol

The UART read protocol consists of 5 bytes (Figure 62). The LSB of the device address frame is 1. The addressed device responds with an Ack frame followed by one or more data bytes if no errors are detected, and the transmission is valid. The byte count indicates the number of data bytes to be read (N). This must be a non-zero number.

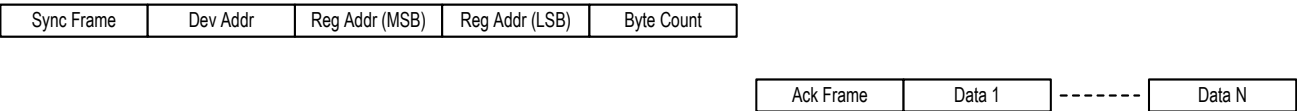


Figure 62. UART Read Protocol Format

16.2.4.2.4 UART Acknowledge Frame

The Ack frame is a regular UART frame with a value of 0xC3 ([Figure 63](#)). When a transmission is successfully received and recognized, the addressed device responds with an acknowledge (Ack) frame to inform the μ C that the transmitted data was received, no errors were detected, and it was recognized as valid. The addressed device responds with an Ack frame after the last bit of the transmission is received.



Figure 63. UART Acknowledge Frame Format

16.2.4.2.5 UART Bit Rate

In base mode, GMSL2 devices automatically detect the UART bit rate using the sync frame at the start of each transmission. The UART bit rate can be any value from 9.6Kbps to 1Mbps and can be changed (by the μ C) after a transaction is completed (that is, when the μ C receives an Ack for a write or Ack and data for a read). When changing to a lower bit rate, the ratio of high and low bit rates must not exceed a factor of 3.5. The μ C can begin transmissions with the new bit rate after waiting 48-bits time (measured by the slower of the old and new bit rates).

In bypass mode, the UART bit rate cannot be changed. The last bit rate used in base mode before entering bypass mode remains the bit rate for bypass mode.

16.2.4.3 UART Channel GMSL2 Bandwidth Utilization

See the [GMSL2 Link Bandwidth Consumption from Side Channels](#) – UART section for details.

16.2.4.4 UART Debug Techniques

If UART issues are exhibited, inspect the UART port(s) using a logic analyzer or digital oscilloscope. Verify that the waveforms observed on the UART Tx and Rx pins are as expected.

BITLEN_LSB and **BITLEN_MSB** are read-only bits that contain the UART bit rate detected by the device ([Table 70](#)). The value in these status bits is the ratio of 150MHz to the detected UART bit rate (example, it is ~300 when UART bit rate is 500Kbps).

Table 70. UART Detected Bit Length (Read-Only Registers)

BITLEN_LSB	BITLEN_MSB	DECODE
BITFIELD	DESCRIPTION	DECODE
BITLEN_LSB[7:0]	UART detected bit length in terms of internal 150MHz clock, low 8 bits.	0xFFFFFFFF: UART detected bit length, low 8 bits
BITLEN_MSB[5:0]	UART detected bit length in terms of internal 150MHz clock, high 6 bits.	0bXXXXXX: UART detected bit length, high 6 bits

UART_RX_OVERFLOW and **UART_TX_OVERFLOW** are read-only bits that latch if an overflow condition has occurred with UART CC communications across the serial link ([Table 71](#)).

Note: In rare circumstances, a link reset during a UART communication or insufficient serial link bandwidth may result in overflow conditions.

Table 71. UART Rx/Tx FIFO Overflow Status (Read-Only Registers)

BITFIELD	DESCRIPTION	DECODE
UART_RX_OVERFLOW[0]	UART RX FIFO overflow. Set to 1 following an overflow condition. Clears upon read.	0b0: No overflow occurred 0b1: Overflow occurred
UART_TX_OVERFLOW[0]	UART TX FIFO overflow. Set to 1 following an overflow condition. Clears upon read.	0b0: No overflow occurred 0b1: Overflow occurred

16.2.5 Disabling Remote Control Channel on Power-Up

The following method describes the process to power up a GMSL2 link with the control channel disabled. This is important when a link is to be established without any I²C/UART communication visible on the remote device.

1. Keep **PWDNB** low initially at local device (or keep VDD off).
2. Set **PWDNB** high to power up the GMSL2 device.
3. Wait ~2ms for power-up to complete.
4. Write **RESET_LINK** high within 10ms (before the **LOCKED** status bit goes high) to prevent the serializer and deserializer from locking.
5. Set **DIS_REM_CC** high.
6. Write **RESET_LINK** low.
7. Poll the **LOCKED** pin until it goes high.
8. The devices are now locked with the control channel disabled.

16.3 Pass-Through Channels (I²C/UART)

16.3.1 Overview

Most GMSL2 devices have two pass-through I²C/UART channels available for local or remote peripheral control. The pass-through I²C/UART channels do not have access to serializer and deserializer registers. The pass-through channels typically have the following naming convention:

- **Channel 1:**
 - SDA1_RX1
 - SCL1_TX1
- **Channel 2:**
 - SDA2_RX1
 - SCL2_TX2

Pass-through I²C and UART modes require pullup resistors (see the [Primary Control Channel – I²C/UART](#) section).

Note: The GMSL2 CSI-2 Quad Deserializers have three independent I²C/UART ports. Two of the ports are pass-through by default. These pass-through ports always provide access to local registers (a unique feature among GMSL2 devices).

Note: Some devices with reduced pin counts share primary CC and pass-through CC functionality on the same set of pins. Extreme care must be taken when enabling and disabling the shared pins with respect to the channel being used. Refer to device-specific data sheets for pinout information. Incorrect system behavior may result if ports sharing the same pins are enabled simultaneously.

16.3.2 Operation

The pass-through I²C/UART channels are independent of the primary I²C/UART control channel. The pass-through channels provide a direct connection to remote I²C/UART peripherals but do not provide any access to internal GMSL2 device registers (except GMSL2 CSI-2 Quad Deserializers).

The pass-through I²C channels provide a connection to a remote I²C port without the GMSL2 devices' internal I²C register subordinates hanging off the bus. This provides a mechanism to separate I²C channels and avoid multi-main conflicts.

The pass-through UART channels provide a direct point-to-point connection to the remote UART peripheral without necessitating UART Bypass Mode (as required by the primary UART CC).

The pass-through channels are protected by the ARQ (Automatic Repeat Request) to improve the robustness of the channel. This is the same as the primary CC. See the [CRC Error Detection and ARQ Error Correction](#) section for additional information.

Pass-through I²C channels support the basic Single Main I²C protocol with 7-bit subordinate address. Multi-main busing is not supported. The remote I²C port supports multiple subordinates. The μ C connected to the local I²C port must support clock stretching to accommodate possible link latency with remote peripheral access(es) (*Figure 21*). A time-out controller prevents locking up the I²C bus (example, in the case that the far side of the link fails to respond).

Pass-through UART channel data must conform to the UART Frame Format. Additionally, pass-through UART data must have a specified bit length. This requirement ensures that the remote device can create the proper UART signaling to the remote UART target peripheral device. Most devices provide two methods for specifying the bit length: manual and control channel inheritance. Note that control channel inheritance only applies if UART data has been written or read to a GMSL2 device using the properly formatted UART synchronization frame and UART accesses on the pass-through channel use the same data rate. If the UART control channel is not being used or a different data rate is desired on the pass-through UART channel(s), the pass-through channel bit length must be manually specified. GMSL2 devices support UART bit rates between 9.6Kbps and 1Mbps.

A simplified system-level block diagram for the pass-through I²C/UART feature is illustrated in [Figure 64](#). The local serializer/deserializer device is connected to the μ C; the remote deserializer/serializer is connected to the remote peripheral I²C/UART device.

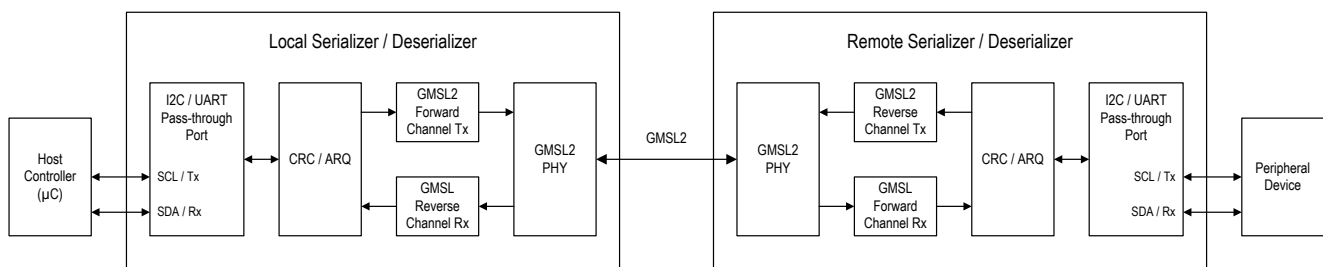


Figure 64. Pass-Through I²C/UART

16.3.3 Pass-Through I²C

16.3.3.1 I²C Mode Configuration

The configuration of pass-through I²C channels is independent from primary I²C/UART control channel configuration. The following tables contain registers associated with the pass-through I²C channels.

Note: Registers listed in the following tables may not exist in all parts. Refer to device-specific data sheets and register documents for the most accurate part information.

Pass-through I²C channels are enabled on a per channel basis by setting the **IIC_1_EN** (Channel 1) and **IIC_2_EN** (Channel 2) in both the serializer and deserializer ([Table 72](#)).

Table 72. Pass-through I²C Channels Enable Registers

BITFIELD	DESCRIPTION	DECODE
IIC_1_EN[0]	Enables pass-through I ² C Channel 1 (SDA1/RX1, SCL1/TX1).	0b0: I ² C pass-through Channel 1 disabled 0b1: I ² C pass-through Channel 1 enabled
IIC_2_EN[0]	Enables pass-through I ² C Channel 2 (SDA1/RX1, SCL1/TX1)	0b0: I ² C pass-through Channel 2 disabled 0b1: I ² C pass-through Channel 2 enabled

The following tables contain the pass-through I²C internal subordinate ([Table 73](#)) and main ([Table 74](#)) configuration registers. The I²C pass-through bit rate is set with the **SLV_SH_PT** (local side) and **MST_BT_PT** (remote side) bitfields. These should be programmed according to the desired I²C pass-through bit rate to satisfy official I²C timing parameters. Set **MST_DBL_PT** high in the remote-side device to double the I²C-to-I²C main bit rate. The **SLV_TO_PT** and **MST_TO_PT** bits select the time-out durations which release the local- or remote-side I²C bus in case an expected response from remote side is not received within the selected time-out duration.

Table 73. Pass-Through I²C Internal Subordinate Configuration Registers

BITFIELD	DESCRIPTION	DECODE
SLV_SH_PT[1:0]	Pass-through I ² C-to-I ² C subordinate setup and hold time setting (setup, hold). Configures the interval between SDA and SCL transitions when driven by the internal I ² C subordinate. Set this according to the I ² C speed mode.	0b00: Set for I ² C Fast-mode Plus speed 0b01: Set for I ² C Fast-mode speed 0b10: Set for I ² C Standard-mode speed 0b11: Reserved
SLV_TO_PT[2:0]	Pass-through I ² C-to-I ² C subordinate time-out setting. Internal GMSL2 I ² C subordinate times out after the configured duration if it does not receive any response while waiting for a packet from the remote device.	0b000: 16us 0b001: 1ms 0b010: 2ms 0b011: 4ms 0b100: 8ms 0b101: 16ms 0b110: 32ms 0b111: Disabled

Table 74. Pass-Through I²C Internal Main Configuration Registers

BITFIELD	DESCRIPTION	DECODE
MST_BT_PT[2:0]	Pass-through I ² C-to-I ² C main bit-rate setting. Configures the I ² C bit rate used by the internal I ² C main (in the device on remote side from the external I ² C main).	0b000: 9.92Kbps – Set for I ² C Standard- mode speed 0b001: 33.2Kbps – Set for I ² C Standard-mode speed 0b010: 99.2Kbps – Set for I ² C Standard- or Fast-mode speed 0b011: 123Kbps – Set for I ² C Fast-mode speed 0b100: 203Kbps – Set for I ² C Fast-mode speed 0b101: 397Kbps – Set for I ² C Fast- or Fast-mode Plus speed 0b110: 625Kbps – Set for I ² C Fast-mode Plus speed 0b111: 980Kbps – Set for I ² C Fast-mode Plus speed
MST_TO_PT[2:0]	Pass-through I ² C-to-I ² C main timeout setting. Internal GMSL2 I ² C main times out after the configured duration if it does not receive any response while waiting for a packet from the remote device.	0b000: 16us 0b001: 1ms 0b010: 2ms 0b011: 4ms 0b100: 8ms 0b101: 16ms 0b110: 32ms 0b111: Disabled
MST_DBL_PT[0]	Doubles the pass-through I ² C-to-I ² C main bit rate	0b0: Do not double the pass-through I ² C-to-I ² C main bit rate 0b1: Double the pass-through I ² C-to-I ² C main bit rate

Table 75 contains read-only I²C acknowledge and time-out status registers. Note that the numbers (that is, 1 and 2) used in the bitfield names indicate whether the bit is associated with either pass-through I²C Channel 1 or Channel 2.

Table 75. Pass-Through I²C Acknowledge and Time-Out Status (Read-Only Registers)

BITFIELD	DESCRIPTION	DECODE
REM_ACK_ACKED_1[0]	In pass-through I ² C Channel 1, inverse of the I ² C acknowledge bit received from remote side.	0b0: I ² C acknowledge bit received as 1 0b1: I ² C acknowledge bit received as 0
REM_ACK_RECVD_1[0]	In pass-through I ² C Channel 1, I ² C acknowledge bit for any I ² C byte is received from remote side for the previous I ² C packet.	0b0: I ² C acknowledge bit not received 0b1: I ² C acknowledge bit received
I²C_TIMED_OUT_1[0]	In pass-through I ² C Channel 1, internal I ² C-to-I ² C subordinate or main has timed out while receiving packet from remote device.	0b0: Time-out has not occurred 0b1: Time-out has occurred
REM_ACK_ACKED_2[0]	In pass-through I ² C Channel 2, inverse of the I ² C acknowledge bit received from remote side.	0b0: I ² C acknowledge bit received as 1 0b1: I ² C acknowledge bit received as 0
REM_ACK_RECVD_2[0]	In pass-through I ² C Channel 2, I ² C acknowledge bit for any I ² C byte is received from remote side for the previous I ² C packet.	0b0: I ² C acknowledge bit not received 0b1: I ² C acknowledge bit received

I²C_TIMED_OUT_2[0]	In pass-through I ² C Channel 2, internal I ² C-to-I ² C subordinate or main has timed out while receiving packet from remote device.	0b0: Time-out has not occurred 0b1: Time-out has occurred
--------------------------------------	--	--

The pass-through I²C/UART pin assignments can be swapped on select devices with the **PT_SWAP** bitfield. Refer to device-specific data sheets and register documents for device support. This allows the pass-through I²C/UART pins for Channel 1 (SDA1 and SCL1) to be used as pass-through I²C/UART pins for Channel 2 (SDA1 and SCL2) or vice versa. Pass-through I²C/UART channels can be connected to the primary CC on the remote side on a per channel basis by configuring the **XOVER_EN_1** and **XOVER_EN_2** bitfields.

See Table 76 for I²C/UART pin assignments and channel connections configuration registers.

Table 76. Pass-Through I²C/UART Device Pin Assignments and Channel Connections Configuration Registers

BITFIELD	DESCRIPTION	DECODE
PT_SWAP[0]	Swaps I ² C/UART pass-through device pin assignments	0b0: Do not swap pin assignments 0b1: Swap pin assignments
XOVER_EN_1[0]	Connects pass-through I ² C/UART Channel 1 to primary control channel on remote side.	0b0: Do not connect 0b1: Connect
XOVER_EN_2[0]	Connects pass-through I ² C/UART Channel 2 to primary control channel on remote side.	0b0: Do not connect 0b1: Connect

16.3.3.1.1 Pass-Through I²C Address Translation

Pass-through I²C channels allow address translation for the remote-side GMSL2 device (Table 77). In the remote serializer/deserializer, the GMSL2 I²C link main recreates the I²C signaling to the remote peripheral. Additionally, with address translation, the remote I²C link main can translate a given device address into another device address with the following bitfields:

- **Channel 1:**
 - **SRC_A_1** and **DST_A_1**
 - **SRC_B_1** and **DST_B_1**
- **Channel 2:**
 - **SRC_A_2** and **DST_A_2**
 - **SRC_B_2** and **DST_B_2**

Table 77. Pass-Through I²C Address Translator Configuration Registers

BITFIELD	DESCRIPTION	DECODE
SRC_A_1[6:0]	I ² C address translator source A. When I ² C device address matches SRC_A_1, internal I ² C main replaces the device address by DST_A_1.	0bXXXXXXXX: I ² C address translator source A
DST_A_1[6:0]	I ² C Address Translator Destination A. See the description of SRC_A_1	0bXXXXXXXX: I ² C address translator destination A
SRC_B_1[6:0]	I ² C Address Translator Source B. When I ² C device address matches SRC_B_1, internal I ² C main replaces the device address by DST_B_1.	0bXXXXXXXX: I ² C address translator source B
DST_B_1[6:0]	I ² C Address Translator Destination B. See the description of SRC_B_1	0bXXXXXXXX: I ² C address translator destination B

SRC_A_2[6:0]	I ² C address translator source A. When I ² C device address matches SRC_A_2, internal I ² C main replaces the device address by DST_A_2.	0bXXXXXXX: I ² C address translator source A
DST_A_2[6:0]	I ² C Address Translator Destination A. See the description of SRC_A_2.	0bXXXXXXX: I ² C address translator destination A
SRC_B_2[6:0]	I ² C Address Translator Source B. When I ² C device address matches SRC_B_2, internal I ² C main replaces the device address by DST_B_2.	0bXXXXXXX: I ² C address translator source B
DST_B_2[6:0]	I ² C Address Translator Destination B. See the description of SRC_B_2.	0bXXXXXXX: I ² C address translator destination B

16.3.4 Pass-Through UART

16.3.4.1 UART Mode Configuration

The following tables contain bitfields associated with the pass-through UART channels. Note that the **PT_SWAP**, **XOVER_EN_1**, and **XOVER_EN_2** bitfields (only available on select devices) are common to the pass-through I²C and UART channels. See Table 76 for configuration information.

Note: Bitfields listed in the following tables may not exist in all parts. Refer to device-specific data sheets and register documents for the most accurate part information.

Pass-through UART channels are enabled on a per channel basis by setting the **UART_1_EN** (Channel 1) and **UART_2_EN** (Channel 2) in both the serializer and deserializer (Table 78).

Table 78. Pass-Through UART Channels Enable Registers

BITFIELD	DESCRIPTION	DECODE
UART_1_EN[0]	Enables pass-through UART Channel 1 (SDA1/RX1, SCL1/TX1).	0b0: Pass-through UART Channel 1 disabled 0b1: Pass-through UART Channel 1 enabled
UART_2_EN[0]	Enables pass-through UART Channel 2 (SDA2/RX2, SCL2/TX2).	0b0: Pass-through UART Channel 2 disabled 0b1: Pass-through UART Channel 2 enabled

The UART parity bit can be enabled or disabled in pass-through operation on a per channel basis (Table 79).

Table 79 - Pass-Through UART Parity Check Configuration Registers

BITFIELD	DESCRIPTION	DECODE
DIS_PAR_1[0]	Disables parity bit in pass-through UART (Channel 1)	0b0: Parity bit enabled 0b1: Parity bit disabled
DIS_PAR_2[0]	Disables parity bit in pass-through UART (Channel 2)	0b0: Parity bit enabled 0b1: Parity bit disabled

If the primary UART CC is being used and UART data has been written or read to a GMSL2 device (serializer or deserializer) using the properly formatted UART synchronization frame in base mode, pass-through UART channels use the same data rate automatically detected by the GMSL2 device in base mode.

If the primary UART CC is not being used, or if a different data rate is required on the pass-through UART channels, the pass-through UART channels bit length must be manually programmed. Bit length is manually configured by first setting **BITLEN_MAN_CFG_1/2** for the appropriate channel(s). Then, the 14-bit bitlength is specified in terms of the internal 150MHz clock with the concatenation of bitfields **BITLEN_PT_1_H[5:0] / BITLEN_PT_1_L[7:0]** for Channel 1 and **BITLEN_PT_2_H[5:0] / BITLEN_PT_2_L[7:0]** for Channel 2 (Table 80).

Table 80. Pass-Through UART Bit Rate Configuration Registers

BITFIELD	DESCRIPTION	DECODE
BITLEN_MAN_CFG_1[0]	Uses the custom UART bit rate (selected by the BITLEN_PT_1_L and BITLEN_PT_1_H bitfields) in pass-through UART Channel 1.	0b0: Use standard bit rate 0b1: Use custom bit rate
BITLEN_PT_1_L[7:0]	Low byte of custom UART bit length for pass-through UART Channel 1. Set this register to the UART bit length divided by 6.666ns (LSB 8 bits). Set BITLEN_MAN_CFG_1 to 1 to use this value.	0xFFFFFFFF: Low byte of custom UART bit length for pass-through UART Channel 1
BITLEN_PT_1_H[5:0]	High byte of custom UART bit length for pass-through UART Channel 1. Set this register to the UART bit length divided by 6.666ns (MSB 6 bits). Set BITLEN_MAN_CFG_1 to 1 to use this value.	0xFFFFFFFF: High byte of custom UART bit length for pass-through UART Channel 1
BITLEN_MAN_CFG_2[0]	Uses the custom UART bit rate (selected by the BITLEN_PT_2_L and BITLEN_PT_2_H bitfields) in pass-through UART Channel 2.	0b0: Use standard bit rate 0b1: Use custom bit rate
BITLEN_PT_2_L[0]	Low byte of custom UART bit length for pass-through UART Channel 2. Set this register to the UART bit length divided by 6.666ns (LSB 8 bits). Set BITLEN_MAN_CFG_2 to 1 to use this value.	0xFFFFFFFF: Low byte of custom UART bit length for pass-through UART Channel 2
BITLEN_PT_2_H[0]	High byte of custom UART bit length for pass-through UART Channel 2. Set this register to the UART bit length divided by 6.666ns (MSB 6 bits). Set BITLEN_MAN_CFG_2 to 1 to use this value.	0xFFFFFFFF: High byte of custom UART bit length for pass-through UART Channel 2

Overflow conditions are monitored for each UART pass-through channel. **UART_RX_OVERFLOW_1/2** and **UART_TX_OVERFLOW_1/2** are read-only bitfields that latch if an overflow condition has occurred with UART pass-through communications across the serial link (Table 81).

Note: In rare circumstances, a link reset during a UART communication or insufficient serial link bandwidth may result in overflow conditions.

Table 81. Pass-Through UART Rx/Tx FIFO Overflow Status (Read-Only Registers)

BITFIELD	DESCRIPTION	DECODE
UART_RX_OVERFLOW_1[0]	Pass-through UART RX FIFO overflow. Set to 1 following an overflow condition. Clears upon read.	0b0: No overflow occurred 0b1: Overflow occurred
UART_TX_OVERFLOW_1[0]	Pass-through UART TX FIFO overflow.	0b0: No overflow occurred 0b1: Overflow occurred

	Set to 1 following an overflow condition. Clears upon read.	
UART_RX_OVERFLOW_2[0]	Pass-through UART RX FIFO overflow. Set to 1 following an overflow condition. Clears upon read.	0b0: No overflow occurred 0b1: Overflow occurred
UART_TX_OVERFLOW_2[0]	Pass-through UART TX FIFO overflow. Set to 1 following an overflow condition. Clears upon read.	0b0: No overflow occurred 0b1: Overflow occurred

16.3.5 Pass-Through I²C/UART in Splitter Mode

The pass-through I²C and UART channels operate point-to-point. This differs from the bus operation of the primary CC. For systems with a GMSL2 link in Splitter Mode or Reverse Splitter Mode, a separate pass-through I²C/UART channel is required for each link to provide a connection to remote I²C/UART peripherals. For example, two deserializers, each with connected peripherals, are connected to a serializer in splitter mode. If pass-through I²C/UART Channel 1 is enabled on the deserializer connected to Link A, pass-through I²C/UART Channel 2 must be enabled on the deserializer connected to Link B.

16.3.6 Pass-Through Channels Debug Techniques

The pass-through I²C/UART channels can be inspected with a logic analyzer or digital oscilloscope. If there are issues with the pass-through channels, verify that the waveforms observed on the SDA1_RX1 and SCL1_TX1 pins (Channel 1) and SDA2_RX1 and SCL2_TX2 pins (Channel 2) are as expected. Refer to the I²C timing diagram in the device data sheet for additional information.

Ensure that the pass-through I²C/UART channel is enabled in both the serializer and deserializer. For example, for pass-through I²C Channel 1, the **IIC_1_EN** register must be set to 1 in both the serializer and deserializer.

For pass-through I²C channels, there are several methods to verify that I²C Ack bits are received from the remote side for any transmitted I²C byte. The Ack bit after each I²C byte can be observed with a logic analyzer or digital oscilloscope. Alternatively, the **REM_ACK_RECVD_1** and **REM_ACK_RECVD_2** bitfields read 1 if an I²C Ack bit has been received from the remote side for the previous I²C byte, and **REM_ACK_ACKED_1** and **REM_ACK_ACKED_2** can be read back to see the value of the received Ack bit.

The **I²C_TIMED_OUT_1** and **I²C_TIMED_OUT_2** registers can be read back to check if the internal I²C-I²C main or subordinate has timed-out while receiving packets from the remote device on pass-through I²C Channel 1 or Channel 2.

UART overflow conditions are flagged independently for each channel with the following bitfields: **UART_RX_OVERFLOW_1** / **UART_RX_OVERFLOW_2** and **UART_TX_OVERFLOW_1** / **UART_TX_OVERFLOW_2**.

17. General-Purpose Input and Output (GPIO)

17.1 Overview

All GMSL2 devices have multifunction pins (MFP) that can be used as general-purpose input and output (GPIO) pins or for other functionality (example, I²C, I²S, SPI, etc.). This section explains the GPIO function of MFP pins. Note that the *I/O Matrix (MFP)* is device-specific; refer to device data sheets for individual device support.

The GPIO blocks of GMSL2 devices communicate and regenerate state changes of GPIO pins from one side of the serial link to the other. GMSL2 serializers and deserializers each contain one transmitter block (GPIO Tx) and one receiver block (GPIO Rx). The transmit block collects the changes from GPIO pins and transmits them to other side of the serial link; the receive block at the other side of the link regenerates the observed changes at the corresponding pins.

17.2 Operation

GPIO pin mapping is coordinated across the serial link through GPIO “pin ID” assignments. Each GPIO input is assigned a pin ID that is included in the packet sent across the serial link and corresponds with a GPIO output. The pin ID of each GPIO transition packet sent from the input is compared with the mapping on the receive-side device, and the transition is reflected on the corresponding output GPIO pin. By default, the GPIO mapping is GPIO0–GPIO0, GPIO1–GPIO1, GPIO2–GPIO2, etc. The GPIO mappings can be changed through registers.

GMSL2 devices use 5-bit pin IDs that can support mapping up to 32 GPIO pins. Note that the usable number of GPIOs is limited by the device-specific GPIO pinout. GPIOs are matched with the pins in the I/O matrix for each part number. Refer to data sheets for individual device GPIO support.

Delay compensation can be configured for GPIO packet transmissions. With delay compensation enabled, GPIO packets are transmitted over the serial link and a configurable fixed latency is applied before the changes are observed on the receive-side GPIO pins. This fixed latency mitigates variable delay that is observed as jitter. In standard mode (without delay compensation), GPIO packets are sent as soon as possible based on available link bandwidth, and changes observed as they are received by the device on the other side of the link. Standard mode offers lower latency than delay compensated mode. However, transmissions may incur variable delay (jitter).

Additionally, the GPIO transmit priority over the link can be adjusted through the priority-based scheduler settings.

17.2.1 Initialization

1. **On link lock:** The GPIO Tx block sends the initial values of enabled GPIO Tx pins upon the establishment of link lock regardless of the pin states.
2. **In sleep mode and on wake up from sleep mode:** The GPIO Rx block drives enabled GPIO Rx pin(s) with the last value received before it entered sleep mode.

17.2.2 GPIO Transmit Control

The GPIO transmit control block is responsible for collecting, scheduling, and transmitting the requests from GPIO pins. The requests are scheduled in the order in which they are received; the queue operates on a first-come-first-served basis.

17.2.3 GPIO Delay Compensation

GPIO delay compensation mode is used to minimize the jitter of the transmitted signal by adding a fixed delay to every GPIO transition on the receive side. This is particularly useful for pulse generation where timing is critical.

Delay compensation allows input transitions to be observed at the output with a fixed latency (with up to ± 10 ns variation). This latency is configurable through registers. However, the default latency value should be used for typical use cases.

In delay compensation mode, a buffer is used to store the transitions before they are reflected on the output pin. This buffer can store up to 15 transitions. The maximum delay compensation value is limited by this buffer size. For instance, if the data rate of the input signal is 4Mbps, the maximum delay compensation value is 3.5 μ s.

17.2.4 Bidirectional Operation

GMSL2 devices support bidirectional GPIO operation. This capability allows control channel and event-driven activity of peripheral devices on each side of the link to be transmitted across the serial link through the GPIO pins. For bidirectional operation, the GPIO Rx block disables the GPIO Tx block for a short duration to prevent false transmissions while the received value is driven to the GPIO pin.

17.3 Configuration

Each GPIO is controlled by three registers: `GPIO_A`, `GPIO_B`, and `GPIO_C`. In the register documentation, the GPIO mapping is sequential (that is, the first three GPIO registers correspond to GPIO0, then next three to GPIO1, etc.) The GPIOs are mapped to different MFP pins in the package depending on the part number.

Note: It is possible that some MFP pins of GMSL2 devices do not function as GPIO by default. Disable specific pins to free up the pins for GPIO use. Refer to the pin description in the data sheet or check the 'MFP Status' tool in the GUI to identify the default function of each MFP pin after power-up.

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

17.3.1 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 40k Ω or 1M Ω .

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

Note: Register `RES_CFG` is ignored if the resistor is disabled (`PULL_UPDN_SEL[1:0] = 00`).

17.3.2 GPIO Output Driver Setup

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

17.3.3 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 ensure reliable forwarding, the local and remote side GPIOs must be properly configured. Each GPIO has register-configurable `GPIO_TX_ID` and `GPIO_RX_ID` used for mapping GPIO pins across the serial link. Note that this configuration applies to both serializer-to-deserializer and deserializer-to-serializer communications.

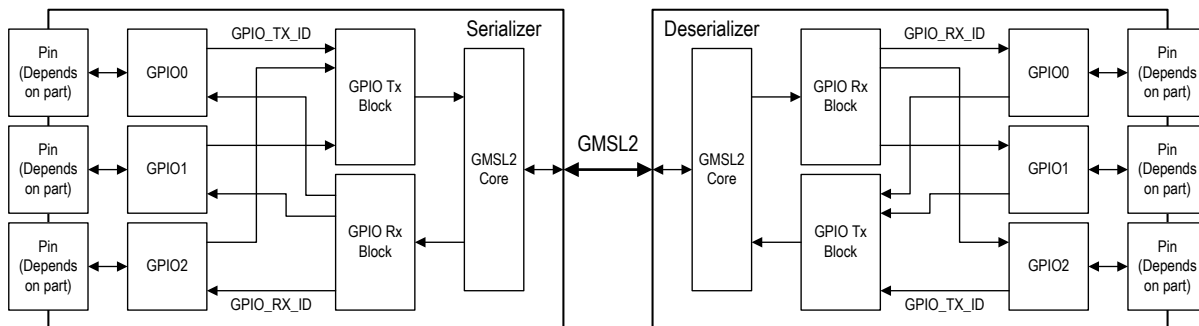


Figure 65. Block Diagram of GPIO Tx and Rx Blocks Mapping

17.3.3.1 Individual GPIO Pin Configuration

Configuring Input GPIO:

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

Configuring Output GPIO:

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

By default, the `GPIO_TX_ID` and `GPIO_RX_ID` are the same value as the GPIO number. For example, the default `GPIO_TX_ID` and `GPIO_RX_ID` values for GPIO1 is 1; accordingly, GPIO1 is mapped to GPIO1 on the opposite side of the serial link by default. Transitions on a single GPIO input can also be mapped to multiple GPIO outputs (broadcasting). This is configured by setting multiple GPIOs to the same `GPIO_RX_ID`.

17.3.4 Delay Compensated Mode

Delay compensation mode is used to minimize the jitter of the transmitted signal by adding a fixed delay to every GPIO transition. Delay compensated mode is enabled by setting `TX_COMP_EN = 1` on the input GPIO. `GPIO_FWD_CDLY[5:0]` is used to configure the forward compensation delay and `GPIO_REV_CDLY[5:0]` is used to configure the reverse compensation delay. Typically, the default forward-channel and reverse-channel compensation delay values are recommended and should only be increased if required by the application.

The total forward channel compensation delay has a default value of $3.4\mu\text{s}$ and is calculated with the following equation:

$$(\text{GPIO_FWD_CDLY}[5:0] + 1) \times 1.7\mu\text{s}$$

The total reverse channel compensation delay has a default value $15.3\mu\text{s}$ and is calculated with the following equation:

$$(\text{GPIO_REV_CDLY}[5:0] + 1) \times 1.7\mu\text{s}$$

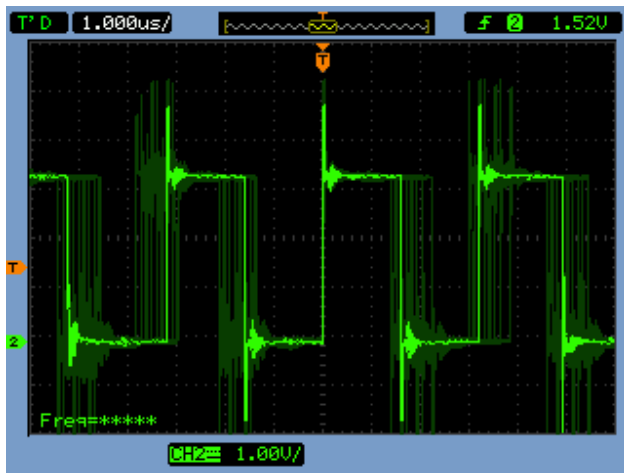


Figure 66. GPIO Forwarding (Standard Mode)

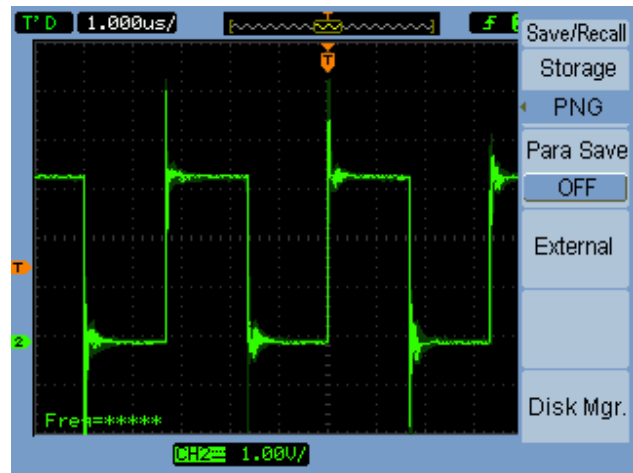


Figure 67. GPIO Forwarding (Delay Compensated Mode)

Figure 66 and Figure 67 show a 300kHz square wave transmitted from a GMSL2 deserializer to a GMSL2 serializer. Figure 66 shows the serializer GPIO output when configured in standard mode; Figure 67 shows the same output with delay compensation enabled. Note the substantial reduction in jitter in Figure 67 compared to Figure 66.

Note: The delay compensated mode delay value may incur a $1.7\mu\text{s}$ offset (that is, one step value) following a link reset event.

17.3.5 Bidirectional Operation Configuration

Bidirectional operation allows GPIO inputs on each side of the link to be transmitted across the serial link. To configure the link for bidirectional function, enable the GPIO output driver and set it into open-drain mode, enable both Tx and Rx across the serial link, and set the `TX_ID` and `RX_ID` on both sides to be identical.

The following procedure configures an active-low, bidirectional GPIO scheme between a GPIO on the serializer and a GPIO on the deserializer.

1. Write `GPIO_OUT_DIS = 0` to enable the output driver.
2. Set open-drain mode by writing `OUT_TYPE = 0`.

3. Write `GPIO_TX_EN` = 1 to enable the GPIO Tx.
4. Write `GPIO_RX_EN` = 1 to enable the GPIO Rx.
5. Set both the Tx and Rx IDs to the same value (`GPIO_TX_ID` = `GPIO_RX_ID`).

The input pull-up/pull-down resistor configuration and delay compensation can be configured as needed.

17.4 Application Examples

17.4.1 Toggling GPIOs Through Register Settings

GPIO pins can be manually controlled through 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 and configure output driver by setting `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.

17.4.2 GPIO Forwarding

This is the primary mode of operation for GPIOs within GMSL2 systems. [Figure 68](#) shows an example setup. In this example, GPIO0 and GPIO1 of the serializer are mapped to GPIO0 and GPIO2 of the deserializer, respectively, and GPIO1 of the deserializer is mapped to GPIO2 of the serializer.

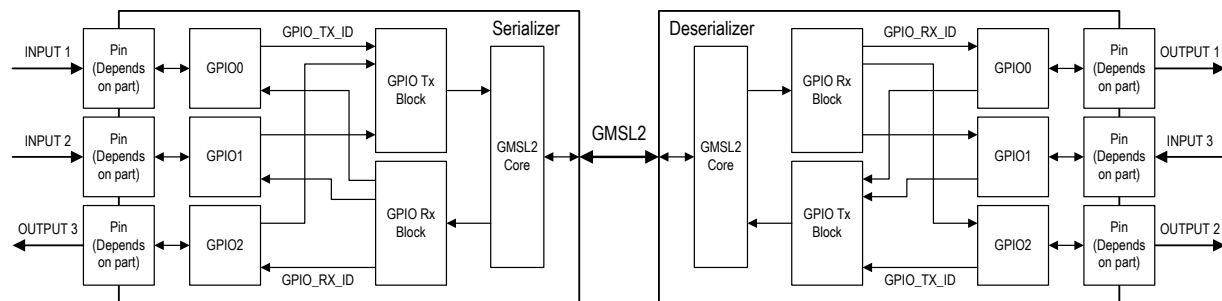


Figure 68. Example of GPIO Forwarding with Three GPIOs

17.4.2.1 GPIO Forwarding Example

Serializer Programming Procedure:

GPIO0: The serializer sets logic level.

1. `GPIO_OUT_DIS` = 0
2. `GPIO_TX_EN` = 1
3. `GPIO_RX_EN` = 0
4. `GPIO_TX_ID` [4:0] = 1

GPIO1: The logic level is applied externally.

1. `GPIO_OUT_DIS` = 1
2. `GPIO_TX_EN` = 1
3. `GPIO_RX_EN` = 0
4. `GPIO_TX_ID` [4:0] = 2

GPIO2: Tracks deserializer GPIO2.

1. `GPIO_OUT_DIS` = 0
2. `GPIO_TX_EN` = 0

3. `GPIO_RX_EN = 1`
4. `GPIO_RX_ID [4:0] = 3`
5. `OUT_TYPE = 1`

Deserializer Programming Procedure:

GPIO0: Tracks serializer GPIO0.

1. `GPIO_OUT_DIS = 0`
2. `GPIO_TX_EN = 0`
3. `GPIO_RX_EN = 1`
4. `GPIO_RX_ID [4:0] = 1`

GPIO1: Tracks serializer GPIO2.

1. `GPIO_OUT_DIS = 1`
2. `GPIO_TX_EN = 1`
3. `GPIO_RX_EN = 0`
4. `GPIO_TX_ID [4:0] = 3`

GPIO2: Deserializer sets logic level.

1. `GPIO_OUT_DIS = 0`
2. `GPIO_TX_EN = 0`
3. `GPIO_RX_EN = 1`
4. `GPIO_RX_ID [4:0] = 2`

Write `RESET_ONESHOT = 1` (on either serializer or deserializer) to apply reset the link and ensure GPIOs are synchronized.

The value of `GPIO_IN` (`GPIO_A` bit 3) reflects the value detected on the input GPIO and the value of `GPIO_RECVD` (`GPIO_C` bit 6) reflects the value received on the output GPIO.

17.4.3 GPIO Broadcasting

Transitions on a single GPIO input can be mapped to multiple GPIO outputs. This can be configured by setting multiple GPIOs to the same `RX_ID`. The following example ([Figure 69](#)) shows a single input routed to two output GPIO pins.

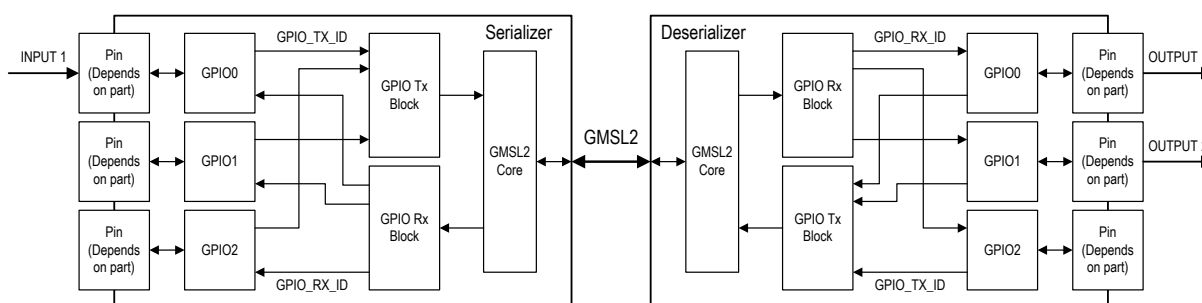


Figure 69. Example of a GPIO Input Routed to Multiple Outputs

17.4.3.1 One GPIO Input Driving Two GPIO Outputs Example

Serializer Programming Procedure:

GPIO0: The logic level is applied externally.

1. `GPIO_OUT_DIS = 1`
2. `GPIO_TX_EN = 1`
3. `GPIO_RX_EN = 0`

4. `GPIO_TX_ID [4:0] = 1`

Deserializer Programming Procedure:

GPIO0: Tracks serializer GPIO0.

1. `GPIO_OUT_DIS = 0`
2. `GPIO_TX_EN = 0`
3. `GPIO_RX_EN = 1`
4. `GPIO_RX_ID [4:0] = 1`

GPIO1: Tracks serializer GPIO1.

1. `GPIO_OUT_DIS = 0`
2. `GPIO_TX_EN = 0`
3. `GPIO_RX_EN = 1`
4. `GPIO_RX_ID [4:0] = 1`

17.5 Feature Availability by Device Family

All GMSL2 devices support GPIO.

17.6 MFP Slew Rate

GMSL2 devices' multifunction pins (MFPs) have configurable rise and fall times (slew rate). This parameter may be referred to as the I/O "speed (control)," "slew (rate)," or "edge rate" in register control bit names. Note that the MFP slew rate cannot be adjusted independently on a per-pin basis. MFPs are divided into separate speed groups; the slew rate adjustment register contains a bitfield for each group that configures the rise and fall time to all pins in the group. Refer to the device-specific data sheet for the relevant register map and MFP speed grouping details.

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

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

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

17.6.1 MFP Slew Rate Operation

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

1. **I²C/UART:** MFP pins operating as an I²C or UART function (that is, control channel or pass-through) are not affected by the MFP rise/fall setting. The I²C/UART circuitry has a fixed falling-edge slew rate, and the rising-edge slew rate is determined by the external pullup resistor.
2. **Dedicated Function:** The rise and fall times of MFP pins assigned dedicated functions (example, SPI, I²S, RCLKOUT, or LOCK & ERR) can be adjusted by the MFP slew rate control registers.

3. **GPIO:** MFP pins operating as GPI or GPO can be adjusted by the MFP rise/fall slew rate control register.

The V_{DDIO} supply voltage affects the I/O slew rate. The impact of the chosen V_{DDIO} voltage must be considered when programming MFP slew rates. MFP slew rate programming and configuration GMSL2 device MFPs are divided into speed groups by digital function. The slew rate adjustment register configures the rise and fall times for each MFP in the group simultaneously. The MFP slew rate can be adjusted at any time, and the changes are applied immediately.

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

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

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

The location of MFP slew rate speed control bits varies depending on the device and available functions. [Table 82](#) contains the register location(s) of the slew rate controls and the total number of MFP speed groups for each GMSL2 device family.

Table 82. MFP Speed Control Registers

Device Family	MFP Speed Control Register	Number of MFP Speed Groups
HDMI Serializers	CMU4	3
CSI-2 Serializers	CMU4	3
Advanced CSI-2 Serializers	CMU4	3
CSI-2 Camera Deserializers	CMU4	4
CSI-2 Display Deserializers	CMU4	3
oLDI Deserializers	CMU4	2
CSI-2 Quad Deserializers	CMU4	3

Refer to the corresponding device's data sheet "Control- and Side-Channel Typical Rise and Fall Times" section for V_{DDIO} timing details. Typical rise and fall times for GMSL2 devices are presented in [Table 83](#).

Table 83. Typical Rise/Fall Times for GMSL2 Devices

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

17.6.2 MFP Slew Rate Applications and Examples

17.6.2.1 Example 2. CMU4 Register Example Using the CSI-2 Serializer

The GMSL2 CSI-2 serializer is an example of a GMSL2 device that uses the CMU4 register to configure the MFP slew rate. This register has the following mapping:

CMU4 (0x304)

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.

18. Serial Peripheral Interface

18.1 Overview

The serial link bridges several interfaces, including SPI, between GMSL2 devices. SPI is a serial communication interface that provides a simple connection between ICs and requires less I/Os than parallel buses. Data is synchronized with the clock signal and transmitted through two unidirectional data lines. A separate control line determines when the interface is active. This simplicity allows for implementation without dedicated hardware or complex software code. This section provides setup and initialization details for multiple SPI link use cases.

Key Features

- Four-wire main (connects to remote peripheral) or four-wire subordinate (connects to μ C/SoC).
- Remote-side SPI bus supports SPI modes 0 or 3; local-side SPI bus supports SPI mode 0.
- Device filtering on (multiple SPI interfaces with different SPI IDs) or off (point-to-point SPI interface).
- Subordinate Select active low or high.
- 600kHz to 25MHz or 50MHz SPI clock (depending on device).
- MSB first (for control commands).
- Pin or I²C control of RO and BNE input/output.

18.1.1 GMSL2 SPI Architecture

GMSL2 enables an SPI main on one side of the link to control an SPI peripheral on the opposite side. Functionally, the GMSL2 link does not act as a transparent bridge. On the local side, an internal SPI subordinate receives data from an external SPI main and transmits it across the serial link. On the remote side, the device receives the data and uses an internal SPI main to transmit the data to the external SPI subordinate devices. Within the serial link, this has the effect of appearing as a four-wire SPI main controlled by a four-wire SPI subordinate. [Figure 70](#) shows a block diagram of the GMSL2 SPI interface.

Data on one side is forwarded from the transmit FIFO through the GMSL2 link and into the remote receive FIFO. This data transfer incurs a slight transmit delay. Due to this delay, the local side (connected to the μ C) operates with transmit data sent first. Once data is received, the μ C then reads back the received data.

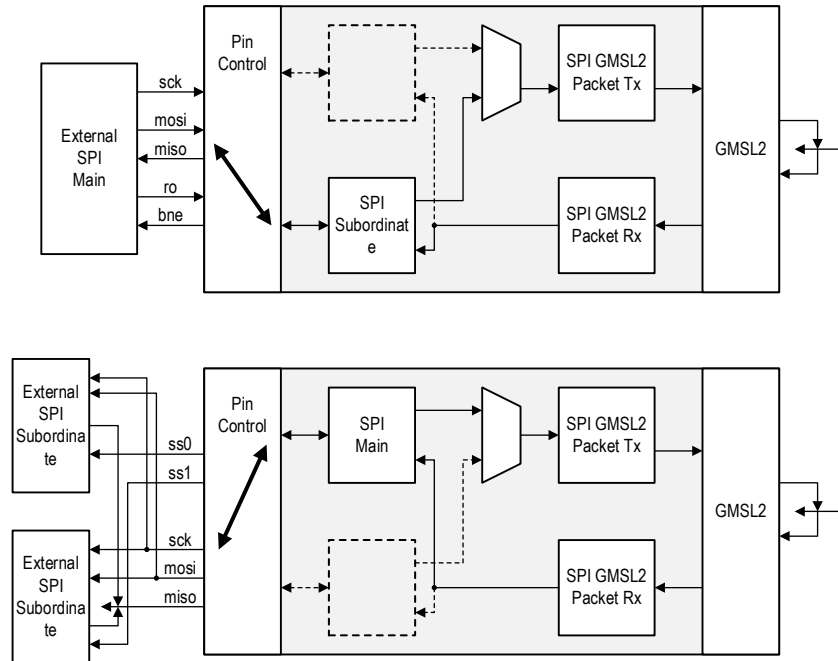


Figure 70. GMSL2 SPI Interface

18.2 Operation

18.2.1 SPI Bridge

The SPI bridge has a Tx buffer used to store bytes prior to transmission on the GMSL2 link and an Rx buffer used to store bytes received from the GMSL2 link prior to being read through a SPI transaction. All SPI modes provide this buffering, which is done in both directions. Each buffer has overflow detection logic with status bits that can be read at [SPI_TX_OVRFLW](#) and [SPI_RX_OVRFLW](#). The status bits are latching and are set with any overflow event. Status bits clear automatically upon read.

18.2.2 SPI Subordinate Control Bits/Pins

The SPI subordinate uses two bits/pins for control: Read Only (RO) and Buffer Not Empty (BNE).

Note: Refer to the latest device data sheets for SPI MFP pins. Some MFP pins may have default alternate functions that must be disabled before enabling SPI. If the SPI pins are also used as CFG pins, do not pull CFG pins until the device powers up and the CFG pins are latched.

18.2.2.1 Read Only

Read Only (RO) is an input bit that determines if the SPI subordinate is in read/command or write mode.

Write mode is enabled if RO is set low. In this mode, any data clocked into the SPI port exits the remote SPI port. This mode is used to write data to an external peripheral (that is, SPI subordinate device).

Read/command mode is enabled if RO is set high. Any data sent during this time is interpreted as control commands for SPI. Control commands are used to select an SPI port, set/clear CS/SS pins, or clock out data from the receive FIFO.

18.2.2.2 Buffer Not Empty

Buffer Not Empty (BNE) is an output bit that shows the receive FIFO state. BNE is low when the buffer is empty; BNE is high when there is data in the buffer. This bit is used to determine the status of the buffer for data transfers and avoiding buffer overflow. This signal de-asserts during a read operation and re-asserts if the buffer remains not empty after the read has completed.

Ensure that the buffer is empty before starting an SPI data transfer. If it is not empty, clock out the excess data until the buffer is empty (BNE = 0). BNE is also indicates the availability of a read byte on the local device. This helps in avoiding buffer overflow. See the [Read Data](#) and [Configuration](#) sections for more details.

18.2.3 SPI Control Commands

Several control commands are used when RO is high and read/command mode is enabled.

18.2.3.1 Device Select

These commands select which GMSL device responds based on the programmable SPI ID. These are only used in multipoint GMSL topologies and are not used in point-to-point applications (example, with GMSL2 quad deserializers). See the [Multiple SPI IDs](#) section.

- 0xA0: Select SPI ID '00'
- 0xA1: Select SPI ID '01'
- 0xA2: Select SPI ID '10'
- 0xA3: Select SPI ID '11'

18.2.3.2 Subordinate Select

These commands control the remote subordinate select (SS) outputs. Note that actual output voltage depends on the programmed SS polarity.

- 0xA4: Assert SS1 output
- 0xA5: Assert SS2 output
- 0xA6: De-assert both SS outputs

18.2.3.3 Read Data

Sending the control byte (0xA7) during a normal buffer read allows the user to request the read of another byte from the remote side without having to toggle RO. This allows multiple data bytes to be read from the SPI subordinate without toggling the RO bit and requires only half as many local-side SPI byte accesses (that is, all reads instead of alternating reads and writes).

Once there is at least a single byte to read in the local buffer (BNE = 1), the byte can be read (RO = 1), and an additional byte read request from the remote-side SPI subordinate can be sent (by sending 0xA7 into the MOSI). The next SPI subordinate read byte is ready to be read from the local buffer when BNE returns high.

When BNE returns high, the next SPI subordinate read byte is ready to read from the local buffer.

18.2.4 SPI Clock

The SPI clock (SCK) on local-side SPI subordinate is determined by the external SPI main.

The specified SPI main timing on the remote-side SPI bus is slow due to long I/O path delays. Depending on the SPI subordinate Clk→Q delay, operating this bus at 50MHz can be challenging. To improve performance reliability, the GMSL2 SPI interface provides a mode to use the full SCK clock period for off-chip, Clk→Q, and on-chip timing for MISO reading.

The `FULL_SCK_SETUP` register bit sets whether MISO is sampled after a half or full SCK period. Normal SPI timing has the subordinate transition on the falling clock edge and the main transition on the rising clock edge (half SCK period, `FULL_SCK_SETUP` = 0). If required, the GMSL2 main device can read the subordinate MISO data on the falling edge (full SCK period, `FULL_SCK_SETUP` = 1). Note that the `FULL_SCK_SETUP` bit has no effect on the GMSL2 subordinate device.

18.2.4.1 SPI SCK

The SCK on the remote-side SPI main can be set between 600kHz and 50MHz using register settings. Register `SPI_4` is used for changing SCK low time and `SPI_5` is used for changing SCK high time in numbers of 300MHz clocks.

Example 1: SCK = 1MHz

Write 0x96 (dec 150) to registers `SPI_4` and `SPI_5`.

$$\text{SCK} = 300\text{MHz} / (150 \times 2) = 1\text{MHz}$$

Example 2: SCK = 5MHz

Write 0x1E (dec 30) to registers `SPI_4` and `SPI_5`.

$$\text{SCK} = 300\text{MHz} / (30 \times 2) = 5\text{MHz}$$

Example 3: SCK = 42.85MHz:

Write 0x03 (dec 03) to register `SPI_4` and write 0x04 (dec 04) to register `SPI_5`.

$$\text{SCK} = 300\text{MHz} / (3 + 4) = 42.85\text{MHz}$$

For higher SCK values, some GMSL2 device SPI MFP pins may need to be adjusted to have faster speed group settings (that is, the rise and fall transition times for each MFP pin).

Note: GMSL2 HDMI Serializers do not have speed controls for MFP pins and no changes are required for different SCK speeds. For other GMSL2 devices, MFP speed group settings can be changed using register settings. Refer to the latest device-specific data sheet for the recommended SPI latching edge and speed group details.

Note: The maximum SPI bandwidth is SCK/8, however, there may be overhead due to BNE sampling, initiating, SPI traffic from the SoC, and/or programming overhead from the SPI subordinate. Actual throughput depends on the implementation and may be less than the calculated maximum bandwidth.

18.2.4.2 Minimum Timing Requirements

Use register [SPI_3](#) to guarantee that minimum timing requirements are met between the assertion of SS and the start of SCK clocks, the end of SCK clocks and the de-assertion of SS, or the time between de-assertion of SS and re-assertion of SS.

All three timing events use the same 8-bit field which defines the minimum number of 300MHz clock cycles allowed (Table 84).

Table 84. SPI Minimum Timing Requirements

BIT	LABEL	R/W	DESCRIPTION	DEFAULT
7:0	SPIM_SS_DLY_CLKS	R/W	Number of 300MHz clocks to delay between: <ol style="list-style-type: none"> 1. Assertion of SS and Start of SCK pulses 2. End of SCK pulses and De-assertion of SS 3. De-assertion of SS and Re-assertion of SS (if necessary). 4. 0xXX: Number of clock delays 	00000000

For all modes of operation, control registers are available to limit the minimum and maximum GMSL2 SPI packet payload as well as set the transmit request priority. These are:

- [SPI_LOC_N](#): This control limits the maximum packet size generated for GMSL2 transmission to (2N+1) bytes. The default value is 6'd7, which limits the packet size to 15 bytes. If this value is programmed to be larger than 7 bytes, the SPI ARQ function must be disabled to avoid ARQ buffer overflows. See the [CRC Error Detection and ARQ Error Correction](#) section for additional information.
- [REQ_HOLD_OFF](#): This controls defines the minimum number of extra bytes required in the Tx buffer prior to requesting the GMSL2 link. The default value of 3'd0 requires no extra bytes in the Tx buffer, and the link request is granted as soon as there is data to transmit. **Note:** All available bytes (as limited by [SPI_LOC_N](#)) are sent when the link request has been granted.
- [REQ_HOLD_OFF_TO](#): The control defines the timeout duration for the [REQ_HOLD_OFF](#) logic in terms of 100ns units. With a timeout, a GMSL2 request is issued regardless of how many extra bytes are required per the [REQ_HOLD_OFF](#) control field. The default value of 8'd0 disables the timeout function.
- [SPI_BASE_PRIORITY](#): This field defines the SPI bridge request priority for the GMSL2 link. The priority levels increase lowest to highest from 0 to 3. The priority automatically increases by one, if possible, when the Tx buffer is over half-full. The default value is 2'd1.

18.3 Configuration

18.3.1 Initialization

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

1. Configure SPI mode 0 or 3 on the serializer and deserializer and set SS output polarity (remote side).
2. Set the clock delay and high/low times (in number of 300MHz clocks).
3. Program the IO pin enables (BNE/RO/SS1/SS2).
4. Configure Main/Subordinate mode, SPI ID (if needed), and enable SPI.

18.3.2 Sending a Four-Wire SPI Command (Up to 15 Bytes)

Perform the following to send a full duplex command:

1. Set RO high to put the device into command mode.
2. Check BNE to ensure that the buffer is empty. If not, clock out the excess data until the buffer is empty.
3. (Optional) send 0xA0 – A3 to select which SPI Device to talk to.
4. Send 0xA4 or 0xA5 to assert SS1 or SS2 on the remote main.
5. Set RO low to put the device into write mode.
6. Write 4 data bytes on MOSI.
7. Wait until BNE is high (indicating that data has been received from the remote device), then set RO high to put the device into command mode.
8. BNE is high for the number of bytes available in the FIFO.
9. Send 0xA6 to de-assert SS1 and SS2. Read one data byte on MISO.
10. Check if BNE is still high.
11. If BNE is still high, send 0xFF to read more bytes on MISO.
12. Set RO low to put the device into write mode.

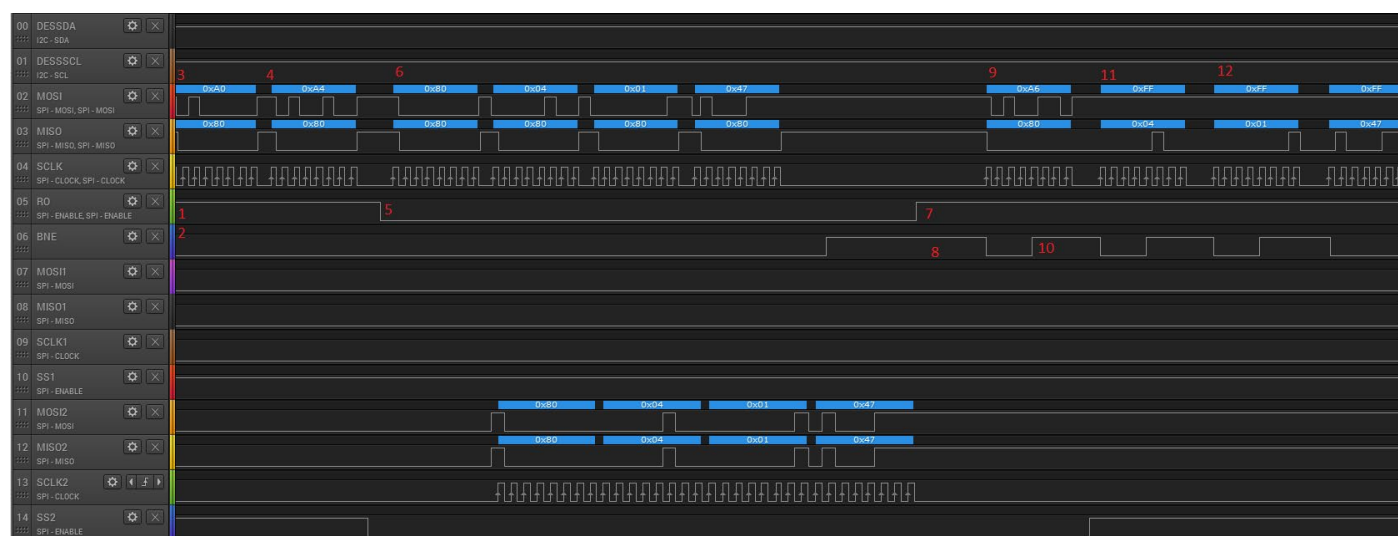


Figure 71. GMSL2 SPI Implementation

18.3.3 SPI Burst Read/Write

The SPI Tx FIFO is 16 bytes and Rx FIFO is 32 bytes. It is possible to read/write burst data (more than 16 bytes) across the GMSL link. A burst write access generates a sequence of write bytes, while a burst read access reads multiple bytes from the read buffer. The number of bytes available to read in the buffer is available in the status field `SPIS_BYTE_CNT`.

Buffer overflows can be avoided by tracking the number of bytes that are moving through the SPI bridge. This measurement is referred to as Bytes in Transit (BIT). The potential for buffer overflows is avoided by maintaining a maximum BIT less than the maximum buffer size (16 bytes).

At the beginning of the burst, initiate a string of write transactions until the maximum BIT has been met. Maintain the maximum BIT during the burst by initiating a new write transaction for every data byte that is read through a read transaction. At the end of the burst, perform only read transactions until the BIT is zero.

18.3.3.1 SPI Burst Write

1. Set RO.
2. Send 0xA0 (Set SPI Target = 0, optional if only one device).
3. Send 0xA4/A5 (Assert SS1/SS2).
4. Clear RO.
5. Send Cmd Byte (Read/Write and Address MS bit).
6. Send Adrs Byte.
7. Send Write Byte.
8. Set RO.
9. Wait for BNE = 1.
10. Send 0x00/Read Byte (Discard).
11. Clear RO.
12. Repeat 7-11 until all data is written.
13. Set RO.
14. Wait for BNE = 1.
15. Send 0xA6/Read Byte (Discard) (Clear SS).
16. Send 0xA6/Read Byte (Discard) (Clear SS).

18.3.3.2 SPI Burst Read

1. Set RO.
2. Send 0xA0 (Set SPI Target = 0, optional if only one device).
3. Send 0xA4/A5 (Assert SS1/SS2).
4. Clear RO.
5. Send Cmd Byte (Read/Write and Address MS bit).
6. Send Adrs Byte.
7. Set RO.
8. Wait for BNE = 1.
9. Send 0xA7/Read Byte (Read Data) (Discard first two reads, remaining are valid).
10. Repeat Steps 8–9 until all but two bytes are read.
11. Wait for BNE = 1.
12. Send 0xA6/Read Byte (Valid Data) (Clear SS).
13. Send 0xA6/Read Byte (Last Valid Data) (Clear SS).

18.3.4 Multiple SPI IDs

In point-to-point SPI applications, the GMSL devices can be set to ignore the packet IDs and accept all packets (`SPI_IGNORE_ID = 1`). For multipoint GMSL topology (example, one serializer linked to two deserializers or vice-versa), each remote-side device can be assigned a different SPI ID using register writes to filter SPI packets. Four SPI IDs (that is, A0 – A3) are available for this application.

Example 1: One serializer → Two deserializers

The configuration below is used to create a SPI network using one GMSL2 serializer as an internal subordinate and two GMSL2 deserializer devices as internal mains to be able to control up to four subordinates using Splitter Mode ([Figure 72](#)).

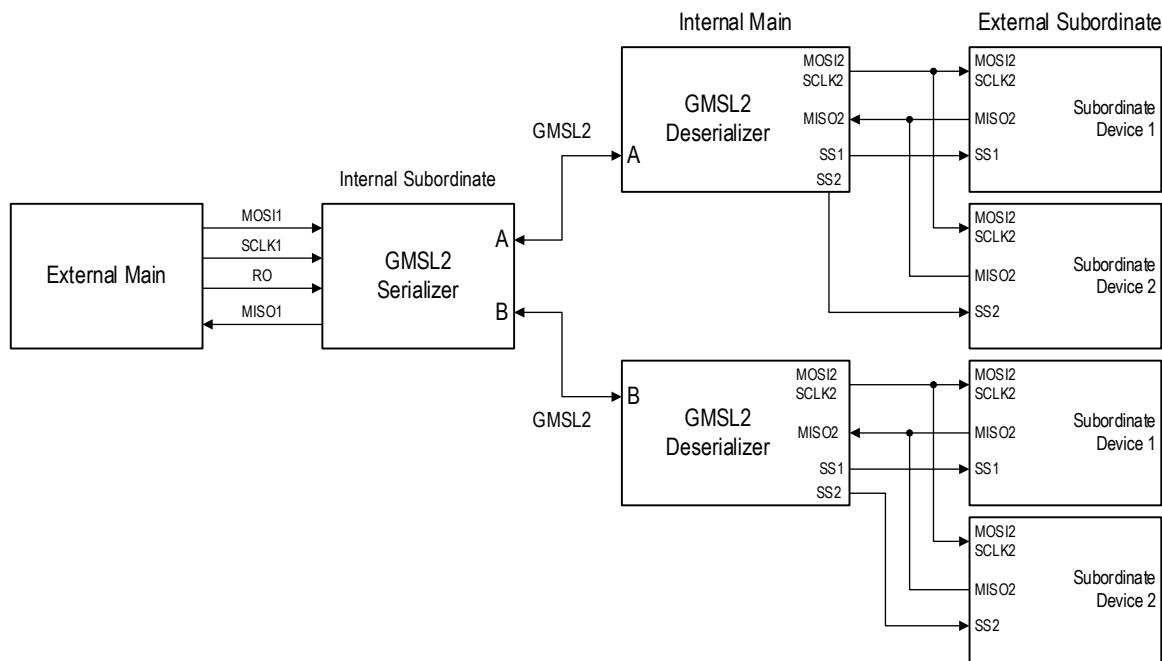


Figure 72. SPI Network: Two Deserializers and One Serializer

In this SPI network, splitter mode is enabled, and the SPI registers are configured to acknowledge the header ID to determine packet acceptance. The two deserializers are linked to one serializer and communicate with each external subordinate by sending command bytes of 0xA0 (link A) or 0xA1 (link B) to select the desired link and 0xA4 or 0xA5 to select the desired Subordinate Select (SS).

Script:

```
#SPI 1 to 2 communication
#enable splitter mode

#Ser  SUBORDINATE
#SER
#####SPLITTER MODE SETUP#####
#set splitter mode, auto link, and reset link
0x80,0x10,0x53
#turn off reset link
0x80,0x10,0x13
#####SPLITTER MODE SETUP#####

#####SPI SETUP#####
#SPI0
0x80,0x170,0x9
#SPI1
0x80,0x171,0x0
#SPI2
0x80,0x172,0x0
#SPI3
0x80,0x173,0x0
#SPI4
0x80,0x174,0x2C
#SPI5
0x80,0x175,0x2C
#SPI6
0x80,0x176,0x03
#SPI7
0x80,0x177,0x98

#Des  MAIN
#DES dev90
#SPI mode 0 default
#SPI0
0x90,0x160,0x3
#SPI1
0x90,0x161,0x2
#SPI2
0x90,0x162,0x4
#SPI3
0x90,0x163,0x20
#SPI4
0x90,0x164,0x80
#SPI5
0x90,0x165,0x80
#SPI6
0x90,0x166,0xC
#SPI7
0x90,0x167,0x0

#Des  MAIN2
#DES dev94
#SPI mode 0 default
#SPI0
0x94,0x160,0x43
```

```
#SPI1
0x94,0x161,0x2
#SPI2
0x94,0x162,0x4
#SPI3
0x94,0x163,0x20
#SPI4
0x94,0x164,0x80
#SPI5
0x94,0x165,0x80
#SPI6
0x94,0x166,0xC
#SPI7
0x94,0x167,0x0
```

Example 2: Two serializers → One deserializer

Similarly, an SPI network using two GMSL2 serializer devices as internal mains and one GMSL2 deserializer device as an internal subordinate can be configured to control up to four external subordinates using Reverse Splitter Mode ([Figure 73](#)).

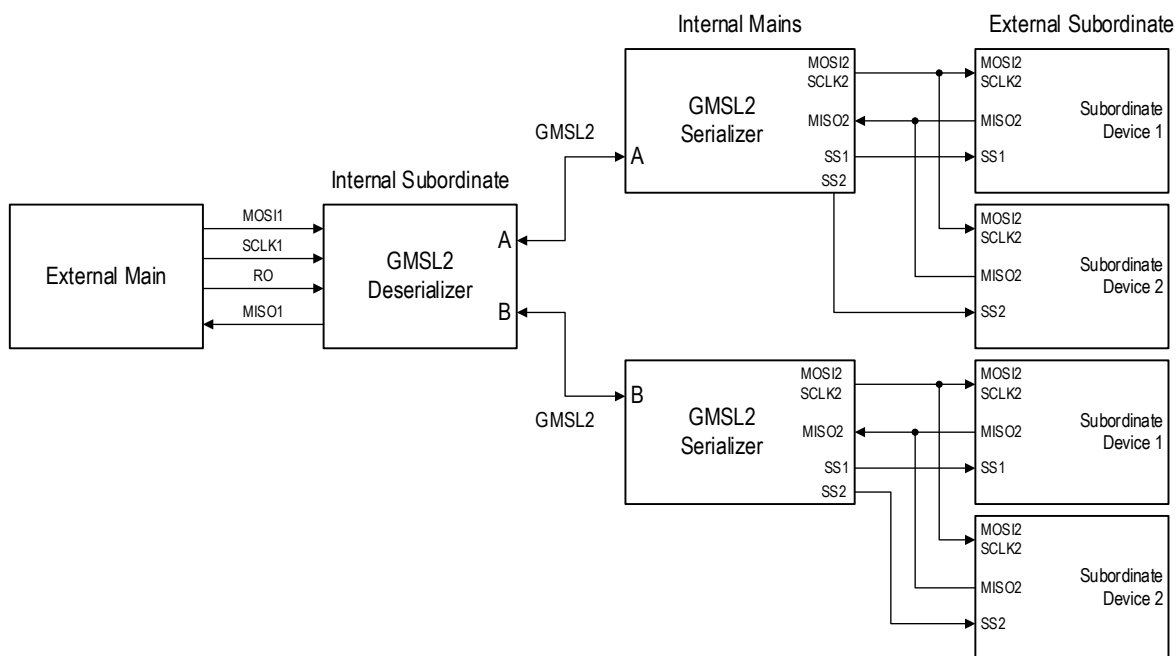


Figure 73. SPI Network: Two Serializers and One Deserializer

Script:

```
#Script for two Ser and one Des SPI

#5MHz SCK output
#
#Reset Parts - Remove if not required
#0x90,0x0010,0x91
#0x80,0x0010,0x91
#disable UART pass-through on Des
0x90,0x0003,0x00
#
#0x90 deserializer is configured as µC local side
#0x80 and 0x84 serializer are configured as µC remote side
#
#enable reverse splitter mode
0x90,0x0010,0x23
0x90,0x0170,0x09
0x90,0x0176,0x03
#0x90,0x0162,0x08
#
0x80,0x0173,0x1E
0x80,0x0174,0x1E
0x80,0x0175,0x1E
0x80,0x0176,0x0C
0x80,0x0172,0x00
#To ignore the SPI ID
#0x80,0x0170,0x0B
#SPI ID = 0x01, use 0xA1 command
0x80,0x0170,0x53
#
0x84,0x0173,0x1E
0x84,0x0174,0x1E
0x84,0x0175,0x1E
0x84,0x0176,0x0C
0x84,0x0172,0x00
#To ignore the SPI ID
#0x84,0x0170,0x0B
#SPI ID = 0x00, use 0xA0 command
0x84,0x0170,0x03
```

18.3.5 Typical Application

In this example script, it is assumed that external SPI main (µC) is connected to a GMSL2 deserializer, and the peripheral SPI devices are connected to a GMSL2 serializer. See [Figure 74](#) for the block diagram of this application.

Note: If the µC is connected to the GMSL2 serializer, interchange the register settings for the serializer and the deserializer as indicated in the script.

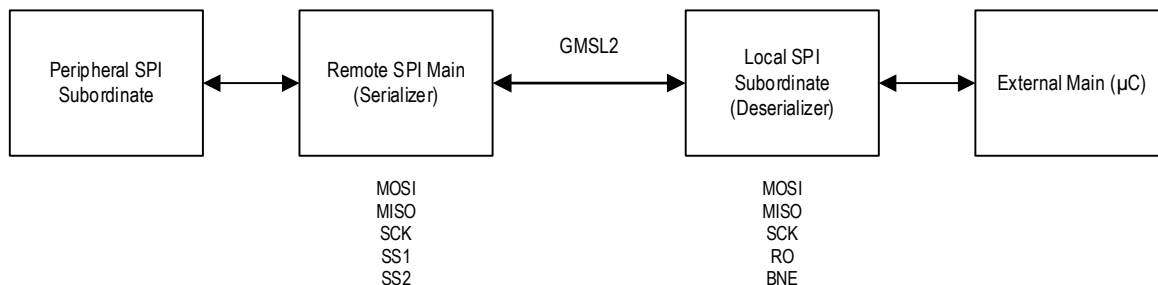


Figure 74. Typical SPI Application

Script:

```

#disable UART pass-through on Des
0x90,0x0003,0x03
#
#0x90 deserializer is configured as µC local side
#0x80 serializer is configured as µC remote side
#swap only below registers 0x80 and 0x90 if µC is on Ser side
#
#SCK = 5MHz
0x80,0x0173,0x1E
0x80,0x0174,0x1E
0x80,0x0175,0x1E
#SS1 and SS2 enable
0x80,0x0176,0x0C
#enable RO and BNE
0x90,0x0176,0x03
#SS is active low
0x80,0x0172,0x00
#enable SPI, ignore SPI ID and internal SPI subordinate
0x90,0x0170,0x09
#enable SPI, ignore SPI ID and internal SPI main
0x80,0x0170,0x0B

```

Bandwidth Calculations

19. GMSL2 Link System Bandwidth

19.1 Overview

GMSL2 systems provide robust serial link connections between camera and display interfaces. Each interface in these interconnected systems (that is, input interface, GMSL2 link, and output interface) has its own bandwidth requirements. Ensuring proper operation of the system requires that all bandwidth consumption be compatible and within specified limits. The overall system bandwidth is limited by the most restrictive interface: this can be either the serializer input interface, deserializer output interface, or the serial link itself.

The input and output interfaces have protocol-specific bandwidth requirements. In order to ensure compatibility with GMSL2 systems, these requirements and figures must be translated into values used by GMSL2 systems. The most important factor here is the pixel clock (PCLK). The PCLK of video received at the input interface is used to determine the video bandwidth consumed on the serial link. The GMSL video bandwidth figure is then converted into the protocol of the output interface to ensure compatibility with remote devices.

GMSL2 link bandwidth is shared by the forward channel video and other sideband channels, and both must be considered to ensure that the combined bandwidth consumption remains within protocol limits. This consideration requires evaluating two related and interactive concepts: payload size and bandwidth consumption. Payload size is the discrete size of a channel's data and is a product of the data received at the input interface; bandwidth is the total consumption by a channel and includes the payload size with the addition of encoding and other protocol overhead. Proper operation of a GMSL2 system requires that the cumulative bandwidth usage of all channels does not exceed GMSL2 protocol limits.

The following image ([Figure 75](#)) and table ([Table 85](#)) show an example of the integrated calculations necessary for determining bandwidth compliance of a serial link system.

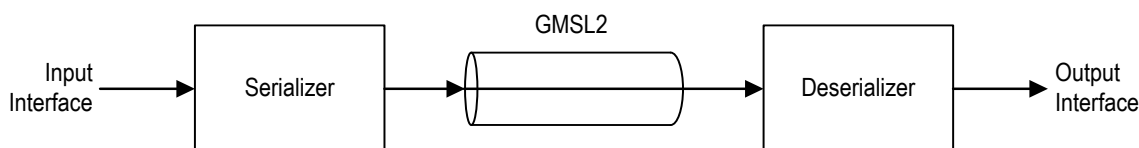


Figure 75. GMSL2 System Block Diagram

[Figure 75](#) is a visual representation of the serial link system stages referenced in the [Table 85](#) example calculations used to demonstrate the concepts of payload and bandwidth throughout a serial link system. In this example, a GMSL2 system consisting of a serializer and a deserializer is transmitting a 1080p RGB888 video with I²C control channel information and a 1Mbps switching GPIO.

Table 85. Example Calculations for 1080p, RGB888 Video for a Serializer and a Deserializer

	INPUT INTERFACE	GMSL LINK	OUTPUT INTERFACE
PCLK	148.5MHz	148.5MHz	148.5MHz
Video Payload (PCLK * bpp)	3.56Gbps	3.56Gbps	3.56Gbps
Video Bandwidth Consumed (video payload + interface overhead)	4.45Gbps	4.04Gbps	4.45Gbps
Total Bandwidth Consumed (video bandwidth + additional interface bandwidth)	4.45Gbps	4.084Gbps*	4.45Gbps

* GMSL Link Total Bandwidth Consumed Calculation:

$$4.04\text{Gbps} + I2C + GPIO = \text{Total GMSL Link Bandwidth}$$

$$4.04\text{Gbps} + 4\text{Mbps} + 40\text{Mbps} = 4.084\text{Gbps}$$

See [GMSL2 Link Bandwidth Consumption from Side Channels](#) for additional information.

19.2 MIPI Video Bandwidth

The video bandwidth is calculated using the video resolution, blanking times, and frame rate. When configuring the DSI video source, use the slowest lane rate possible which supports the video bandwidth (with sufficient margin). This results in:

- reduced MIPI bandwidth results in lower MIPI frequency and improved signal integrity, and
- increased accuracy of the timing relationship of input PCLK to output MIPI clock.

The number of MIPI lanes used contributes to the overall MIPI video payload. The following equations can be used to calculate the MIPI lane rate(s):

$$PCLK = (Total\ Horizontal) \times (Total\ Vertical) \times (Frames\ Per\ Second)$$

$$Video\ Payload = PCLK \times 24\ (bpp), \text{ must not exceed } 5.2\text{Gbps due to link overhead}$$

$$MIPI\ Lane\ Rate = \frac{PCLK \times bpp}{\# of Lanes}, \text{ four possible lanes (see Table 86 for maximum lane rates)}$$

The MIPI video payload can be derived by the video throughput and number of lanes used. For a 6Gbps serial link, the maximum video throughput is 5.2Gbps (due to 9b10b encoding overhead and guard band protection). If using four lanes, $5.4\text{Gbps}/4 = 1.35\text{Gbps}$ per lane; this limit should be further rounded down to 1.3Gbps per lane to provide margin for other data channels. Each MIPI lane supports up to 2.5Gbps. See [Table 86](#) for maximum MIPI lane rates which fit within a 6Gbps GMSL link.

See the [GMSL2 Link System Bandwidth](#) section for information regarding other data channels and overall serial link bandwidth.

Table 86. Maximum MIPI Lane Rates for 6Gbps Link

NUMBER OF MIPI LANES	MAXIMUM MIPI LANE RATE(S) FOR A 6GBPS LINK
4	1.3Gbps
3	1.7Gbps
2	2.5Gbps
1	2.5Gbps

19.3 GMSL2 Link Bandwidth Consumption from Video

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 optional features (example, forward error correction and display stream compression) affect link bandwidth consumption and must be considered if enabled. For details on PCLK calculation, see the [Video Basics Equations](#) section.

Note: Video transmission consumes the largest proportion of serial link bandwidth; however, all data transmitted on the serial link must be considered when calculating link bandwidth to ensure optimum serial link performance.

The equation used to calculate the forward channel video payload is:

$$\text{Video Payload} = \text{PCLK} * \text{bpp}$$

The equation used to calculate the total forward channel video bandwidth is shown as follows:

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

$$\text{Video BW} = \text{PCLK} * \left[(\text{bpp}) + \left(\frac{1}{2}\right) + \left(\frac{1}{2}\right) \right] * \left(\frac{10}{9}\right) * \left(\frac{2048}{2047}\right) * \left[\left(\frac{128}{120}\right) * \left(\frac{1}{3}\right)\right]$$

$$\text{Video BW} = \text{PCLK} * \left[(\text{bpp}) + \frac{1}{2} + \frac{1}{2} \right] * \left(\frac{10}{9}\right) * \left(\frac{2048}{2047}\right) * \left[\left(\frac{128}{120}\right) * \left(\frac{1}{3}\right)\right]$$

Note: Video Pixel CRC, FEC, and DSC are optional features; see the [Forward Error Correction](#) section for additional information and refer to device-specific data sheets for availability details. Video **pixel** CRC is disabled by default as video **line** CRC is enabled by default and is typically preferred. See the [Video Data CRC](#) section for additional details.

Note: BPP = 9 is the minimum allowed bpp value for GMSL2 link BW calculation. For video datatypes with a bpp value less than 9 (example, RAW8 and EMB8), a bpp value of 9 must be used when calculating GMSL2 link bandwidth consumption.

This forward channel video bandwidth calculation is applicable to all input interfaces.

Forward channel rates vary depending on GMSL2 mode. [Table](#) lists the maximum video payload for each mode. The maximum rates factor in sufficient margin to accommodate sideband channels.

Table 87. Maximum Video Payload for GMSL2 Modes

GMSL2 MODE	MAXIMUM VIDEO PAYLOAD
3Gbps Mode	2.6Gbps (2600Mbps)
6Gbps Mode	5.2Gbps (5200Mbps)

19.4 Interface-Specific Bandwidth Calculations

19.4.1 CSI-2 Bandwidth Calculations

The pixel clock for MIPI input interfaces is determined by the input MIPI speed (that is, the MIPI clock multiplied by the number of MIPI D-PHY or C-PHY data lanes). For a given video format, higher MIPI speeds allow margin for the image sensor to transmit the required video stream; however, this results in a higher PCLK value and increased consumption of GMSL2 link bandwidth. This balance means that careful consideration is required, especially for marginal system design.

19.4.1.1 CSI-2 Serializers – D-PHY Input

This section explains how the received MIPI video stream relates to the resulting PCLK frequency and bpp that feeds into the GMSL2 transmitter block. Here, the input data stream rates and packet formats are converted to GMSL formatting.

The maximum allowed PCLK is 600MHz. The MIPI D-PHY can use up to four available lanes; each lane has a maximum bandwidth of 2.5Gbps.

The PCLK is calculated with the following equation:

$$PCLK = \frac{LANE_CNT * LANE_RATE}{bpp}$$

The total MIPI data rate is the product of the video PCLK and bpp:

$$Total\ MIPI\ data\ rate = LANE_CNT * LANE_RATE = PCLK * bpp$$

The maximum total MIPI data rate supported on GMSL devices is:

$$Total\ MIPI\ data\ rate = LANE_CNT * LANE_RATE = 4 * 2.5Gbps = 10Gbps$$

Note: A higher MIPI lane rate results in a higher PCLK, which consumes more bandwidth on the GMSL link. When configuring the MIPI lane rate in the video source, use the slowest lane rate possible which supports the video bandwidth (with sufficient margin).

The bpp is the bits per pixel of the chosen CSI-2 or DSI datatype. Example bpp values for various datatypes are given below. Note that, for purposes of bandwidth calculations, DSI can be considered equal to the RGB888 CSI-2 data stream (bpp = 24).

- RGB888: 24 bpp
- RAW12: 12 bpp
- RAW8: 8 bpp
- EMB8: 8 bpp

19.4.1.1.1 Managing Multiple PCLK Values in a MIPI Input Stream (Time Multiplexed)

In the following modes, the video stream is time multiplexed: the data is sequential as it would typically be transmitted from a single camera source.

19.4.1.1.1.1 Constant BPP Video Pipe Mode

This is the default mode for processing streams containing multiple PCLK values. In this basic mode, every video pipe is arranged to have a single bpp value. The maximum allowed PCLK is 600MHz regardless of the GMSL2 bandwidth. The resulting maximum allowed MIPI rates for the given D-PHY input is provided in [Table 88](#).

$$PCLK = \frac{\text{Total MIPI Data Rate}}{\text{bpp}}$$

Table 88. Maximum CSI-2 Lane Rates (Mbps) in Constant BPP Mode

MAX CSI-2 LANE RATE		NUMBER OF LANES			
		1	2	3	4
bpp	8	2500.00	2400.00	1600.00	1200.00
	10	2500.00	2500.00	2000.00	1500.00
	12	2500.00	2500.00	2400.00	1800.00
	14	2500.00	2500.00	2500.00	2100.00
	16	2500.00	2500.00	2500.00	2400.00
	18	2500.00	2500.00	2500.00	2500.00
	20	2500.00	2500.00	2500.00	2500.00
	24	2500.00	2500.00	2500.00	2500.00

19.4.1.1.1.2 Double Loading Mode

Double loading mode provides increased bandwidth efficiency and expanded use case possibilities. This data arrangement mode is available for low-bpp video datatypes with a bpp value of 8, 10, or 12. Double mode concatenates two input pixels so that they are processed as a single pixel in a video pipe. This concatenation enables video streams with heterogenous PCLK values to be combined into a single video pipe. Double loading mode also provides serial link bandwidth efficiency gains for single video streams.

Note: Double loading mode is one method of combining two video streams in a single video pipe. This allows multiple datatypes differing by a factor of two to be sent through a single video pipe without loss in bandwidth efficiency.

Example: RAW12 and RGB888 Double Loading

An image sensor transmits RAW12 (bpp = 12) and RGB888 (bpp = 24) video packets over two lanes, each operating at 1Gbps. The RAW12 packets are double loaded so that the bpp of the RAW12 packets matches the bpp of the RGB888 packets. The two datatypes are transmitted through the same video pipe.

$$PCLK = \frac{LANE_CNT * LANE_RATE}{bpp}$$

$$PCLK = \frac{2 * 1Gbps}{24\ bpp} = 83.33MHz \text{ (note: } PCLK_{max} \leq 600MHz \text{)}$$

The link bandwidth calculation uses the highest bpp value of the transmitted datatypes. Here, the RGB888 bpp value of 24 is used.

$$BW = 83.33MHz * (24\ bpp + 0.5) * \left(\frac{10}{9}\right) * \left(\frac{2048}{2047}\right) = 2.27Gbps$$

The GMSL2 link bandwidth consumption of RAW12 and RGB888 datatypes is 2.27Gbps.

Example: RAW12 Double Loading

An image sensor transmits RAW12 (bpp = 12) video packets over one lane at 1.5Gbps.

$$PCLK = \frac{LANE_CNT * LANE_RATE}{bpp}$$

$$PCLK = \frac{1 * 1.5Gbps}{12\ bpp} = 125MHz \text{ (note: } PCLK_{max} \leq 600MHz \text{)}$$

GMSL2 bandwidth consumption is calculated with the following equation:

$$BW = 125MHz * (12\ bpp + 0.5) * \left(\frac{10}{9}\right) * \left(\frac{2048}{2047}\right) = 1.737Gbps$$

However, the RAW12 packets can be double loaded to reduce the link bandwidth consumption (by reducing protocol overhead).

$$PCLK = \frac{LANE_CNT * LANE_RATE}{bpp}$$

$$PCLK = \frac{1 * 1.5Gbps}{24\ bpp} = 62.5MHz \text{ (note: } PCLK_{max} \leq 600MHz \text{)}$$

GMSL2 bandwidth consumption is calculated with the following equation:

$$BW = 62.5MHz * (24\ bpp + 0.5) * \left(\frac{10}{9}\right) * \left(\frac{2048}{2047}\right) = 1.702Gbps$$

In this example, double loading the RAW12 packets provides a bandwidth reduction of 35Mbps.

19.4.1.1.1.3 Zero-Padding Mode

Zero-padding allows multiple datatypes that do not have the same bpp value to be transmitted on a video pipe together. This option is an alternative to double loading mode and does not require the bpp values to differ by a factor of two. With zero-padding, the datatype with the lower bpp value is zero-padded to be the same length as the bpp of the larger datatype.

The PCLK is derived from the datatype with the lowest bpp value. Prior to zero-padding one of the datatypes, there are two PCLKs due to the two different bpp values. The slower video stream (that is, the datatype with the larger bpp value) can only be processed with the clock from the faster video stream (that is, the datatype with the smaller bpp value).

The link bandwidth calculation uses the bpp value of the video pipe. This is equal to the highest bpp value of the transmitted datatypes.

The video streams are time multiplexed. The multiple datatypes are not transmitted simultaneously; individual frames of each datatype are transmitted sequentially.

Note: Zero-padding mode is available for all datatypes with $\text{bpp} \leq 16$.

Example: EMB8 and RAW12 Zero-Padding

An image sensor transmits EMB8 ($\text{bpp} = 8$) and RAW12 ($\text{bpp} = 12$) video packets over two lanes, each operating at 1Gbps. The EMB8 packets are zero-padded with four zeros per pixel, matching the bpp of RAW12. The two datatypes are transmitted through the same video pipe.

EMB8 has a $\text{bpp} = 8$ and RAW12 has a $\text{bpp} = 12$. Since the PCLK is calculated using the datatype with the lowest bpp value, the EMB8 value is used in the equation.

$$PCLK = \frac{LANE_CNT * LANE_RATE}{bpp}$$

$$PCLK = \frac{2 * 1Gbps}{8 \text{ bpp}} = 250MHz \text{ (note: } PCLK_{max} \leq 600MHz \text{)}$$

The 250MHz value is confirmed to be smaller than 600MHz (the maximum PCLK value).

The link bandwidth calculation uses the highest bpp value of the transmitted datatypes. Here, the RAW12 bpp value of 12 is used.

$$BW = 250MHz * (12 \text{ bpp} + 0.5) * \left(\frac{10}{9}\right) * \left(\frac{2048}{2047}\right) = 3.47Gbps$$

The GMSL2 link bandwidth consumption of EMD8 and RAW12 datatypes is 3.47Gbps.

19.4.1.1.4 Double Loading Mode in Combination with Zero-Padding Mode

Double loading mode and zero-padding mode can be used together to accommodate a wide range of use cases involving many different datatypes.

Example: An image sensor transmits RAW12 and EMB8 video packets in a two-lane CSI-2 stream, each lane operating at 1Gbps. In the serializer, The EMB8 and RAW12 packets are transmitted together in a video pipe. Here, double loading mode is used to pack two EMB8 pixels together into a 16 bpp pixel, and zero-padding is used to increase the bpp of the RAW12 packet from 12 to 16. With the effective bpp of each datatype equal to 16, both datatypes can be sent together through the same video pipe.

To calculate the PCLK, the bpp from the datatype with the lowest bpp value (that is, higher PCLK value) is used. In this example, the RAW12 bpp value is the lowest (bpp = 12) and is used for the PCLK calculation.

$$PCLK = \frac{LANE_CNT * LANE_RATE}{bpp}$$

$$PCLK = \frac{2 * 1Gbps}{12 \text{ bpp}} = 166.67MHz$$

The link bandwidth calculation uses the highest bpp value of the transmitted datatypes. Here, the double loaded EMB8 bpp value of 16 is used.

$$BW = 166.67MHz * (16 \text{ bpp} + 0.5) * \left(\frac{10}{9}\right) * \left(\frac{2048}{2047}\right) = 3.057Gbps$$

The GMSL2 link bandwidth consumption of EMB8 and RAW12 datatypes is 3.057Gbps.

19.4.1.1.2 Managing Video that is not Time Multiplexed

These calculations assume that the video stream is time multiplexed. If the video streams are not time multiplexed, the MIPI long packets may overlap in time. This may occur if multiple independent MIPI inputs are received by the serializer (Figure 76). To calculate the peak bandwidth consumed on the serial link, the peak bandwidth demands of each video stream are calculated separately then summed together. This figure must be less than the maximum video bandwidth limit to ensure the proper operation of the serial link.

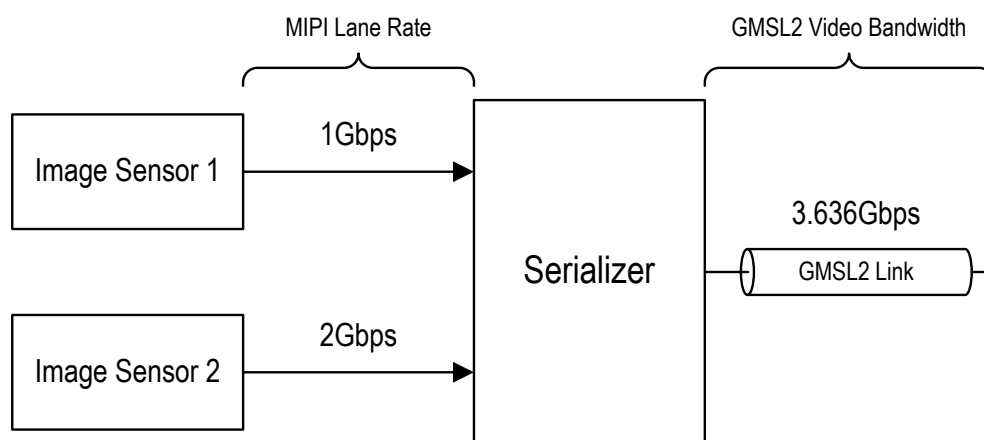


Figure 76. A GMSL2 Serializer with Two Independent MIPI Inputs

Example: Two image sensors separately transmit EMB8 (bpp = 8) and RAW12 (bpp = 12) to a single CSI-2 serializer. The EMB8 video stream has a lane rate of 1Gbps; the RAW12 video stream has a lane rate of 2Gbps.

EMB8 video stream:

$$PCLK = \frac{LANE_CNT * LANE_RATE}{bpp}$$

$$PCLK = \frac{1 * 1Gbps}{8 \text{ bpp}} = 125MHz$$

$$BW = 125MHz * (9 \text{ bpp} + 0.5) * \left(\frac{10}{9}\right) * \left(\frac{2048}{2047}\right) = 1.320Gbps$$

Note: BPP = 9 is the minimum allowed bpp value for GMSL2 link BW calculation.

RAW12 video stream:

$$PCLK = \frac{LANE_CNT * LANE_RATE}{bpp}$$

$$PCLK = \frac{1 * 2Gbps}{12 \text{ bpp}} = 166.67MHz$$

$$BW = 166.67MHz * (12 \text{ bpp} + 0.5) * \left(\frac{10}{9}\right) * \left(\frac{2048}{2047}\right) = 2.316Gbps$$

Combined GMSL2 bandwidth consumption:

$$BW_{peak} = 1.320Gbps + 2.316Gbps = 3.636Gbps$$

19.4.1.2 CSI-2 Deserializers – D-PHY Output

The MIPI CSI-2 output clock frequency is independently configured. The `phy_csi_tx_dppll_predef_freq` registers allow adjustments with 100Mbps step resolution. The `CSI2_LANE_CNT` register programs the `LANE_CNT` definition (one to four lanes). Further configuration of the output frequency is covered in the section.

Relevant equations:

- $Total \text{ MIPI data rate} = LANE_RATE * LANE_CNT$
- $Total \text{ MIPI data rate} \geq PCLK * (BPP_{single \text{ video receiver pipe}})$

If there are multiple video pipes aggregated to a single CSI-2 output, the calculation must consider if the packets are time domain multiplexed between the video pipes.

- If multiple video pipes have time-overlapping data:
 $(PCLK1 * BPP1) + (PCLK2 * BPP2) + \dots \leq Total \text{ MIPI data rate}$
- If multiple video pipes have time-separated data:
 $MAX\{(PCLK1 * BPP1), (PCLK2 * BPP2), \dots\} \leq Total \text{ MIPI data rate}$

The maximum allowed PCLK is 600MHz. In standard operation, the maximum MIPI Lane Rate is 2500Mbps. GMSL2 dual-link mode increases the serial link bandwidth between the serializer and the deserializer to 12Gbps. This increased bandwidth can transmit CSI-2 data to the deserializer at 10.4Gbps. Enabling DSC can further increase the CSI-2 data available to the deserializer.

19.4.1.3 CSI-2 Deserializers – C-PHY Output

CSI-2 C-PHY output supports higher output bandwidth than D-PHY. C-PHY uses three-phase symbol encoding that delivers 2.28 bits per symbol over three-wire trios, each with an embedded clock (**Note:** ‘trios’ are also referred to as ‘lanes’ or ‘pins’). Each trio operates at 2.5GSym/s, resulting in a bandwidth of 5.7Gbps per trio. Lanes can be combined to increase available bandwidth ([Table 89](#)). There is no line coding overhead (example, 8b10b). Video pipes Y and Z permit 1x3 and 1x4 configurations, while video pipes X and U only allow 1x1 and 1x2 lane configurations ([Figure 77](#)).

Table 89. MIPI D-PHY and C-PHY Lane Configurations and Output Bandwidth

Width (CLKxDATA)	D-PHY		C-PHY	
	Pins	Rate (Gbps)	Pins	Rate (Gbps)
1x1	4	2.5	3	5.7
1x2	6	5.0	6	11.4
1x3	8	7.5	9	17.1
1x4	10	10.0	12	22.8

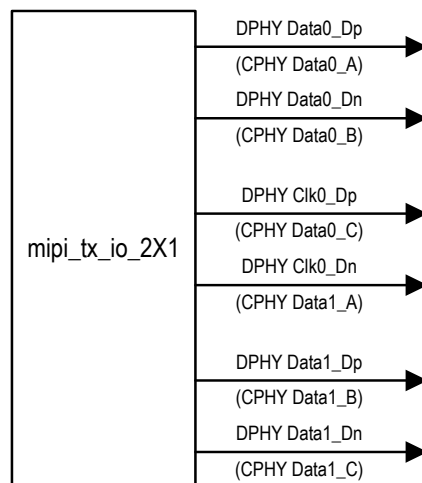


Figure 77. MIPI Lane Configurations

19.5 GMSL2 Link Bandwidth Consumption from Side Channels

Side channels occupy channel bandwidth when enabled. The total link bandwidth usage is the sum of the side channel consumption, the utilization from video, and protocol overhead. This calculation must be below the specified bandwidth for the serial link to avoid overflow.

Note: Video signals only occupy the forward channel (serializer to deserializer) of the GMSL2 link, while side channels may operate on both the forward and reverse channels. Typically, the side channels consume bandwidth in the direction the data is sent—referred to here as “forward bandwidth” and also in the opposite direction—referred to here as “reverse bandwidth” if CRC is enabled.

The calculations are agnostic of GMSL link direction. For example, if audio is sent from the deserializer to the serializer, the audio forward bandwidth is consumed on the GMSL reverse channel, and the audio reverse bandwidth is consumed on the GMSL forward channel.

Note: CRC increases bandwidth consumption for all side channels (in both directions) and is enabled by default for all channels. Disabling CRC is not recommended for most applications. See the [CRC Error Detection and ARQ Error Correction](#) section for additional details.

19.5.1 GPIO

GPIO bandwidth consumption is a function of how many transitions per second occur. This consumption is increased if delay-compensation mode is enabled. Bandwidth is consumed in the direction of the transmitted GPIO transmission; however, if CRC/ARQ is enabled, the ARQ acknowledge packets also consume reverse channel bandwidth.

$$\text{GPIO Forward BW (Mbps)} = (T * (2 + COMP + CRC) * 20) / 1000000$$

$$\text{GPIO Reverse BW (Mbps)} = (T * CRC * 2 * 20) / 1000000$$

Where:

- T = transitions per second.
- COMP = 1 if delay compensated mode is enabled; COMP = 0 if delay compensated mode is disabled.
- CRC = 1 if CRC is enabled; CRC = 0 if CRC is not enabled.

For example, if a GPIO is used to transmit a touch interrupt signal, the maximum transitions per second would be approximately five, resulting in a bandwidth consumption of 0.3kbps on the forward channel.

Note: For serializers the Forward BW is consumed from serializer to deserializer and Reverse BW is consumed from deserializer to serializer. For deserializers the opposite is true, the Forward BW is consumed from deserializer to serializer and Reverse BW is consumed from serializer to deserializer.

19.5.2 SPI

SPI bandwidth consumption is a function of the SPI clock and byte length. Bandwidth is consumed in the direction of the SPI transaction as well as the reverse direction for the acknowledge packets.

$$\text{SPI Forward BW (Mbps)} = (f_{SCK} / (8 * L)) * (2 + CRC + \text{roundup}((L - 1) / 2)) * 20$$

$$\text{SPI Reverse BW (Mbps)} = (f_{SCK} / (8 * L)) * 2 * 20 * CRC$$

Where:

- fSCK = SPI clock (SCK) frequency in Mbps.
- L = SPI read/write length in bytes.
- roundup() is a function defined as rounding up the contained number to the next integer value.
- CRC = 1 if CRC is enabled; CRC = 0 if CRC is not enabled.

Note: For serializers the Forward BW is consumed from serializer to deserializer and Reverse BW is consumed from deserializer to serializer. For deserializers the opposite is true, the Forward BW is consumed from deserializer to serializer and Reverse BW is consumed from serializer to deserializer.

19.5.3 I²C

The I²C bandwidth consumption is a function of the SCL frequency. The I²C is generated simultaneously in both directions and includes ARQ ACK traffic; bandwidth is consumed in both the forward and reverse directions.

$$\text{I}^2\text{C Forward BW (Mbps)} = (f_{\text{SCL}}/4500) * (2 + \text{CRC}) * 20$$

$$\text{I}^2\text{C Reverse BW (Mbps)} = 40 * (f_{\text{SCL}}/1000) * \text{CRC}$$

Where:

- f_{SCL} is the I²C clock rate (kbps).
- CRC = 1 if CRC is enabled; CRC = 0 if CRC is not enabled.

Note: For serializers the Forward BW is consumed from serializer to deserializer and Reverse BW is consumed from deserializer to serializer. For deserializers the opposite is true, the Forward BW is consumed from deserializer to serializer and Reverse BW is consumed from serializer to deserializer.

19.5.4 UART

UART bandwidth consumption is a function of the UART baud rate. UART consumes bandwidth in in the direction of the UART transaction as well as the reverse direction for the acknowledge packets.

$$\text{UART Forward BW (Mbps)} = (f_{\text{UART}}/10000) * (2 + \text{CRC}) * 20$$

$$\text{UART Reverse BW (Mbps)} = (f_{\text{UART}}/10000) * 2 * 20$$

Where:

- f_{UART} is the UART baud rate.
- CRC = 1 if CRC is enabled; CRC = 0 if CRC is not enabled.

Note: For serializers the Forward BW is consumed from serializer to deserializer and Reverse BW is consumed from deserializer to serializer. For deserializers the opposite is true, the Forward BW is consumed from deserializer to serializer and Reverse BW is consumed from serializer to deserializer.

Functional Safety

20. GMSL2 Error Reporting (ERRB Pin)

20.1 Overview

All GMSL2 devices have a multi-purpose open-drain error reporting and interrupt status output (ERRB). The configurable ERRB pin can output detected error status(es) and/or certain internal device events relevant to the host processor (that is, interrupts). The ERRB function is assigned to an MFP (see device-specific data sheets for pinout information).

The ERRB pin operates in open-drain output mode with a 40k Ω internal pull-up resistor. It is an active-low signal: 1 indicates “no error and no interrupt event,” 0 indicates the occurrence of an “error and/or interrupt event.”

Note: The ERRB pin reflects the status of the **ERROR** bit (register 0x0013) but with inverted logic (that is, **ERROR**=1 and ERRB pin low indicates an error condition).

Multiple errors and event sources can be simultaneously configured to drive the ERRB pin. Each error and event able to drive the ERRB pin has a status flag register. When several errors and/or events are configured to drive the ERRB pin, these status registers can be read to determine the source of the ERRB assertion. Each ERRB source can be individually enabled and disabled.

Functions and uses include:

- Very fast routing of on-chip diagnostics to device pin used as an interrupt for the SoC.
- ASIL-related functions route to the internal ERRB signal to be used as an interrupt.
- Remote-side forwarding of diagnostics to local ERRB signal permits all GMSL2 system devices to centralize ERRB reporting to a single pin.

20.2 Operation

Most GMSL2 device errors/statuses can be output to the ERRB pin. Independent of ERRB reporting, the individual device error statuses are reported in the local registers of the feature. Many of these errors have attendant counters that track the cumulative number of errors that have occurred. Refer to device-specific data sheets for more information.

Errors are individually configured to be output to the ERRB pin. Each error eligible for ERRB pin routing has an output enable (OEN) bit that is used to send the error status to the ERRB OR gate. The OR gate controlling the ERRB status is illustrated in [Figure 78](#). Note that concept is universal to GMSL2 devices; available errors and event statuses may vary by device.

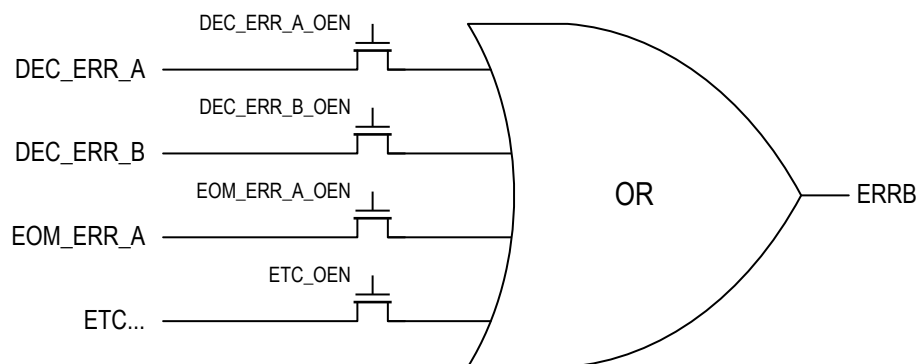


Figure 78. ERRB Logic

The ERRB pin can be used to monitor important error or status flags. If the ERRB pin is driven low and multiple errors/statuses are configured to output to ERRB, the local registers for the enabled errors/statuses should be read to determine the cause. [Figure 79](#) shows the internal error reporting mechanism.

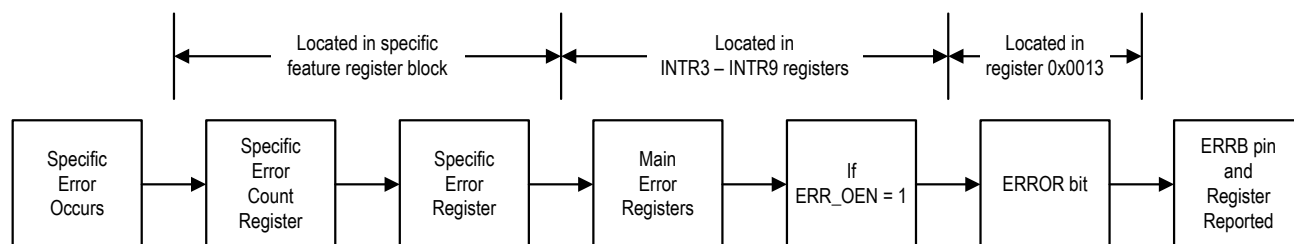


Figure 79. Internal Error Reporting Mechanism

If an error occurs and its **ERR_OEN** bit is enabled, the ERRB pin is driven low (that is, active) and the **ERROR** bit (register 0x0013) reports a 1. This alerts the user to a nonspecific error state among the enabled error bits. To determine what error occurred and whether the error is continuous or a one-time event, a sequence of register reads of the ERRB errors/statuses must be executed. GMSL2 errors are latching: once an error occurs, an error flag is set and is not cleared until it or the associated error count register is read/cleared.

[Figure 80](#) contains a suggested process for identifying errors resulting in an ERRB transition.

20.2.1 Example ERRB System Reaction

This example is presented to illustrate the function of errors within a GMSL2 system and describe the process used to determine the source of the error condition ([Figure 80](#)).

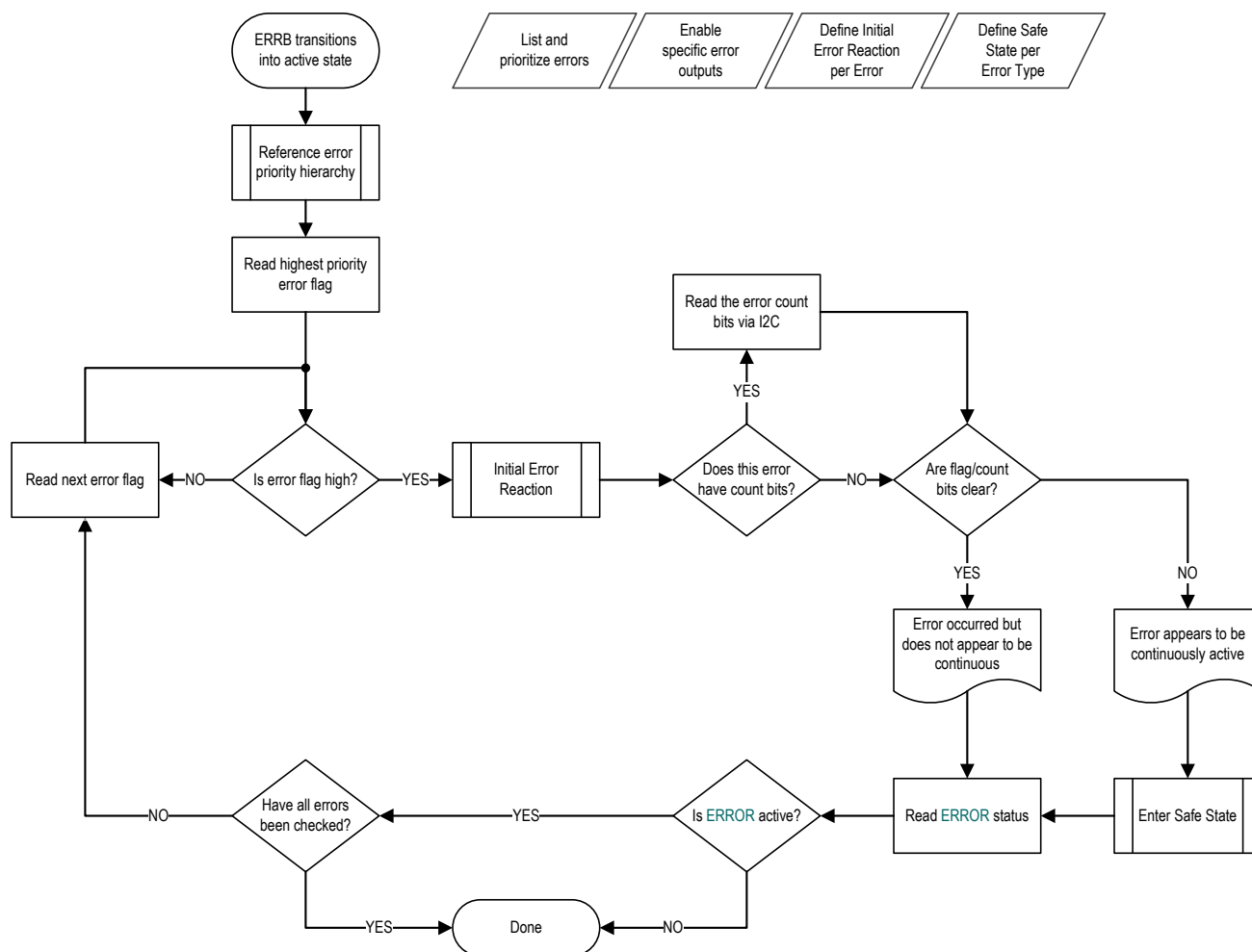


Figure 80. ERRB Transition System Reaction Process

Four monitored errors are assigned the following priority:

1. Line fault error
2. GMSL decoding error
3. EOM error
4. Watermark error

During initial system configuration, enable ERRB output for the relevant errors:

- Line fault error: `LFLT_INT_OEN = 1`
- GMSL decoding error: `DEC_ERR_OEN = 1`
- EOM error: `EOM_ERR_OEN = 1`
- Watermark error: `WM_ERR_OEN = 1`
- All other error output enable (OEN) bits are set to 0.

A link quality fault event occurs and ERRB transitions to the active state. The system reaction process (defined in [Figure 80](#)) is used to determine the cause(s) of the error state.

- ERRB transitions to active state (`ERROR = 1`).
- The highest priority error is line fault error.
 - `LFLT_ERR_FLAG` is read, returns `LFLT_ERR_FLAG = 0`.
- The next highest priority error is decode error.
 - `DEC_ERR_FLAG` is read, returns `DEC_ERR_FLAG = 1`.
 - System reacts as defined by system engineer for decode error fault condition.
 - The decode error counter register `DEC_ERR_A[7:0]` is read and returns a non-zero value.
 - The decode error counter register `DEC_ERR_A[7:0]` is read again and returns zero, indicating this was a noncontinuous error condition.
 - ERRB pin status is checked again (either by ERRB pin state or reading `ERROR` register value). The ERRB status is still active (`ERROR = 1`).
- The next highest priority error is EOM error.
 - `EOM_ERR_FLAG` is read, returns `EOM_ERR_FLAG = 1`.
 - System reacts as defined by system engineer for EOM error fault condition.
 - The EOM error does not have count bits.
 - `EOM_ERR_FLAG` is read again and now returns `EOM_ERR_FLAG = 0`, indicating this was a noncontinuous error condition.
 - ERRB status is checked again and is no longer active (`ERROR = 0`).
- The remaining lower priority errors do not need to be checked as there is no longer an error condition present (that is, watermark errors do not need to be checked).
- System returns to nominal operating state.

20.2.2 Disabling the ERRB Function

The ERRB function can be disabled if, for example, error/interrupt reporting is not required, or the MFP pin used by ERRB is needed for another function. To disable the ERRB function of the default MFP, set `ERRB_EN = 0`. If ERRB is disabled, the MFP pin defaults to GPIO functionality. Note that the pin can be further reassigned if different functionality is required. Select GMSL2 devices allow for the assignment of an alternate MFP to be used for ERRB output. For compatible devices, use the `ALT_ERRB_EN` register to enable the alternative MFP for ERRB.

20.2.3 Remote Error Reporting

After the serial link is locked, GMSL2 devices can report the ERRB status of remote device(s) through a GPIO with programmable ID. To enable remote error reporting, set `ERR_TX_EN = 1`, and assign the GPIO ID by programming the `ERR_TX_ID` register. The local device receives the transmitted error over the GPIO tunnel.

20.2.3.1 Output Error Status with Local GPIO

Configure the remote device to output the error status through GPIO tunneling. Then, program the local device with the following register writes to connect the local device to the GPIO tunnel and output the error status on an available MFP:

- Write `GPIO_RX_EN` = 1 to enable the GPIO receive block for the GPIO pin.
- Write `GPIO_TX_EN` = 0 to disable the GPIO transmit block for the GPIO pin.
- Set `GPIO_OUT_DIS` = 0 to enables the GPIO output driver.
- Write `GPIO_TX_ID` = "Remote `ERR_TX_ID` value" to assign the GPIO pin ID with the same GPIO pin ID as the remote-side ID transmitting the error status.

20.2.3.2 Combine Remote and Local Error Statuses for ERRB Output

The remote and local error statuses can be combined for consolidated ERRB output on the local device. Set `ERR_TX_EN` = 1 in the remote device. If necessary, program the `ERR_TX_ID` register to an unused value. In the local device, set `ERR_RX_ID` = "Remote `ERR_TX_ID` value", `ERR_RX_EN` = 1, and `REM_ERR_OEM` = 1.

The remote ERRB status is transmitted to the local device and drives the local ERRB pin active (unless already active due to a previous local error or interrupt condition). The host controller can read the `REM_ERR_FLAG` register to determine whether the source of the error originated in the remote or local device.

GMSL2 devices that support connections to multiple remote devices (that is, devices with two links and serializers that support daisy-chaining) can consolidate all remote ERRB status signals to the local ERRB pin. For daisy-chain applications, multiple ERRB signals can be combined and output from a separate MFP (that is, combining multiple GPIO tunnel outputs).

20.3 Configuration

All possible errors and events capable of driving the ERRB pin are listed with the configuration details. "Flag" indicates the name of the read-only register that is asserted to 1 in the case of an error condition; "OEN" indicates the name of the register used to enable reporting of the flag status to the ERRB pin.

Note: Although ERRB is supported by all GMSL2 devices, not all errors/interrupts addressed in this section are available in all devices. Refer to device-specific data sheets for available errors and interrupts. If the register controlling error/interrupt output to the ERRB pin is not listed, that error/interrupt reporting function is not available in that device.

20.3.1 GMSL Decoding Errors

Flag:

- `DEC_ERR_FLAG_A`
- `DEC_ERR_FLAG_B`
- `DEC_ERR_FLAG_C`
- `DEC_ERR_FLAG_D`
- `DEC_ERR_FLAG_SIO`
- `DEC_ERR_FLAG_DCIO`

OEN (Output Enable):

- `DEC_ERR_OEN_A`
- `DEC_ERR_OEN_B`
- `DEC_ERR_OEN_C`
- `DEC_ERR_OEN_D`
- `DEC_ERR_OEN_SIO`
- `DEC_ERR_OEN_DCIO`

Description: Each GMSL link has a built-in 8-bit counter that tracks detected 9b10b disparity errors. Disparity errors are detected whenever the 9b10b running disparity exceeds the minimum or maximum limit.

Note: Isolated single-bit errors are detected with a high probability. Burst errors and single-bit errors occurring in close proximity may not be detected or may be counted as a single error. Therefore, decode errors cannot be used to calculate bit error rate (BER) statistics. See the [BER Testing Using the GMSL2 Idle Link](#) section) for additional information.

Decoding error counter registers:

- [DEC_ERR_A](#)
- [DEC_ERR_B](#)
- [DEC_ERR_C](#)
- [DEC_ERR_D](#)
- [DEC_ERR_SIO](#)
- [DEC_ERR_DCIO](#)

Each decoding error flag is asserted when the corresponding decoding error counter value is greater than or equal to the error counter threshold value defined by [DEC_ERR_THR](#) (default is 1).

Related registers: [DEC_ERR_THR](#), [AUTO_ERR_RST](#)

Clearing: Decoding error counters are reset to 0 when read, reset by the Auto Error Reset function, or after establishing link lock after loss of lock. After a decoding error counter is reset, its error flag is also reset (the counter value of 0 is always less than the error threshold value).

20.3.2 GMSL Idle Packet Errors

Flag:

- [IDLE_ERR_FLAG](#)
- [IDLE_ERR_FLAG_A](#)
- [IDLE_ERR_FLAG_B](#)
- [IDLE_ERR_FLAG_C](#)
- [IDLE_ERR_FLAG_D](#)
- [IDLE_ERR_FLAG_SIO](#)
- [IDLE_ERR_FLAG_DCIO](#)

OEN (Output Enable):

- [IDLE_ERR_OEN](#)
- [IDLE_ERR_OEN_A](#)
- [IDLE_ERR_OEN_B](#)
- [IDLE_ERR_OEN_C](#)
- [IDLE_ERR_OEN_D](#)
- [IDLE_ERR_OEN_SIO](#)
- [IDLE_ERR_OEN_DCIO](#)

Description: Each GMSL link has a built-in 8-bit idle packet error counter that counts detected idle packet errors. Idle packets have a data payload consisting of all zeros (before scrambling) and are transmitted to fill unused link bandwidth. The idle error counter increments when an 18-bit word in an idle packet is not equal to 0 after decoding and descrambling.

When the GMSL link is primarily idle, idle packets comprise most of the transmitted packets. If the link is run idle for an amount of time, checking the detected idle errors can provide information about the Bit Error Rate (BER) of the link during that period. See the [BER Testing using the GMSL2 Idle Link](#) in the *PRBS Testing* section for additional information.

Note: An error is not detected by idle error detection if the error affects only the header of an idle packet.

Idle packet error counter registers:

- [IDLE_ERR](#)
- [IDLE_ERR_A](#)

- IDLE_ERR_B
- IDLE_ERR_C
- IDLE_ERR_D
- IDLE_ERR_SIO
- IDLE_ERR_DCIO

Each idle error flag is asserted when the corresponding idle error counter value is greater than or equal to the error counter threshold value defined by `DEC_ERR_THR` (default is 1).

Related registers: `DEC_ERR_THR`, `AUTO_ERR_RST`

Clearing: Idle error counters are reset to 0 when read, reset by the Auto Error Reset function, or after establishing link lock after loss of lock. After an idle error counter is reset, its error flag is also reset (the counter value of 0 is always less than the error threshold value).

20.3.3 GMSL Packet Count Interrupt

Note: the GMSL Packet Count Interrupt is intended for tests and diagnostics.

Flag:

- `PKT_CNT_FLAG`
- `PKT_CNT_FLAG_A`
- `PKT_CNT_FLAG_B`
- `PKT_CNT_FLAG_C`
- `PKT_CNT_FLAG_D`

OEN (Output Enable):

- `PKT_CNT_OEN`
- `PKT_CNT_OEN_A`
- `PKT_CNT_OEN_B`
- `PKT_CNT_OEN_C`
- `PKT_CNT_OEN_D`

Description: Each GMSL link has a built-in, user-configurable 8-bit received packet counter. The `PKT_CNT_SEL` and `PKT_CNT_LBW` registers are used to define the packet types that are counted. The counter is exponentially scaled. The user-configurable unit of count is defined as 2^N , where N is a value from 0 to 15 programmed with the `PKT_CNT_EXP` register.

Packet counter registers:

- `PKT_CNT`
- `PKT_CNT_A`
- `PKT_CNT_B`
- `PKT_CNT_C`
- `PKT_CNT_D`

The packet count flag is asserted when the packet counter value is greater than or equal to the packet counter threshold value defined by `PKT_CNT_THR` (default is 1).

Related registers: `PKT_CNT_THR`, `PKT_CNT_EXP`, `PKT_CNT_SEL`, `PKT_CNT_LBW`, `AUTO_CNT_RST`

Clearing: The packet counter is reset to 0 when read or reset by the Auto Packet Count Reset function. When a packet counter is reset, its error flag is also reset (the counter value of 0 is always less than the error threshold value).

20.3.4 Line Fault Error

Flag:

- LFLT_INT

OEN (Output Enable):

- LFLT_INT_OEN

Description: Each GMSL device has a built-in Line Fault detector for link diagnostics. An error is reported when the line is shorted to power, shorted to ground, or the wires of the differential pair are shorted. An error is also reported when line is open. See the [Line Fault](#) section for additional information.

Related registers: LF_0, LF_1, LF_2, LF_3, PU_LF_0, PU_LF_1, PU_LF_2, PU_LF_3

Clearing: Resolve the line error condition.

20.3.5 Eye Opening Monitor Error

Flag:

- EOM_ERR_FLAG_A
- EOM_ERR_FLAG_B
- EOM_ERR_FLAG_C
- EOM_ERR_FLAG_D
- EOM_ERR_FLAG_SIO
- EOM_ERR_FLAG_DCIO

OEN (Output Enable):

- EOM_ERR_OEN_A
- EOM_ERR_OEN_B
- EOM_ERR_OEN_C
- EOM_ERR_OEN_D
- EOM_ERR_OEN_SIO
- EOM_ERR_OEN_DCIO

Description: GMSL devices have a built-in Eye Opening Monitor that periodically measures the received serial data eye after equalization. An eye opening error is reported if the measured eye opening falls below the programmed threshold (EOM_MIN_THR).

Related registers: EOM, EOM_DONE, EOM_EN, EOM_PER_MODE, EOM_CHK_THR, EOM_CHK_AMOUNT, EOM_MIN_THR

Clearing: The eye-opening monitor is a latching bit. The error is set once a single eye-opening measurement is below the programmed threshold and is cleared only when the flag is read. Note that if the last eye-opening measurement is below the error threshold when the flag is read, the flag is not cleared.

20.3.7 Lock Status Interrupt

Flag:

- LOCK
- LOCK_A
- LOCK_B

OEN (Output Enable):

- LOCK_OEN
- LOCK_A_OEN
- LOCK_B_OEN

Description: Flag is asserted when the GMSL link loses lock. This is useful if the LOCK pin cannot be monitored separately by the host controller.

Related registers: None

Clearing: When LOCK output to the ERRB pin is enabled, loss of lock causes ERRB to go low. ERRB goes high when the link locks if there are not any other active error drivers.

20.3.8 Loss of Lock Status Interrupt

Flag:

- LOSS_OF_LOCK_FLAG

OEN (Output Enable):

- LOSS_OF_LOCK_OEN

Description: Flag is asserted when the GMSL link loses lock. This flag is latching. Note that this interrupt is different than the live Lock Status Interrupt and is not cleared when the link locks again. This is useful if the LOCK pin cannot be monitored separately by the host controller.

Related registers: None

Clearing: Flag is cleared when read.

Note: Only some GMSL2 devices have this interrupt—check product-specific data sheet for availability.

20.3.9 Retransmission Count Interrupt

Flag:

- RT_CNT_FLAG
- RT_CNT_FLAG_A
- RT_CNT_FLAG_B
- RT_CNT_FLAG_C
- RT_CNT_FLAG_D

OEN (Output Enable):

- RT_CNT_OEN
- RT_CNT_OEN_A
- RT_CNT_OEN_B
- RT_CNT_OEN_C
- RT_CNT_OEN_D

Description: Low-speed GMSL2 channels (that is, control channels, PT_X, PT_Y, SPI, GPIO, Audio, Auto-HDCP) have a built-in Automatic Repeat Request / Automatic Retransmission (ARQ). When enabled, packets that are not acknowledged by the receiver are automatically retransmitted by the ARQ. Each low-speed channel has a 7-bit counter that tracks the number of ARQ retransmissions (that is, RT_CNT register in the register block of each sub-channel). When a channel has at least one retransmission event and the retransmission count reporting is enabled (RT_CNT_OEN register in the register block of each sub-channel, disabled by default), then the combined RT_CNT_FLAG is asserted.

Related registers: The RT_CNT and RT_CNT_OEN registers of each sub-channel.

Clearing: The retransmission counter (**RT_CNT**) of each sub-channel is reset when it is read.

20.3.10 Maximum Retransmission Interrupt

Flag:

- **MAX_RT_FLAG**
- **MAX_RT_FLAG_A**
- **MAX_RT_FLAG_B**
- **MAX_RT_FLAG_C**
- **MAX_RT_FLAG_D**

OEN (Output Enable):

- **MAX_RT_OEN**
- **MAX_RT_OEN_A**
- **MAX_RT_OEN_B**
- **MAX_RT_OEN_C**
- **MAX_RT_OEN_D**

Description: Low-speed GMSL2 channels (that is, control channels, PT_X, PT_Y, SPI, GPIO, Audio, Auto-HDCP) have a built-in Automatic Repeat Request / Automatic Retransmission (ARQ). When enabled, packets that are not acknowledged by the receiver are automatically retransmitted by the ARQ. If the same packet is retransmitted for a maximum number of times (set by **MAX_RT** register in the register block of each sub-channel) and the expected acknowledge packet is not received, then the maximum retransmission error of that sub-channel is set (**MAX_RT_ERR** register in the register block of each sub-channel). The **MAX_RT_ERR** flag of all sub-channels are combined to generate the combined maximum retransmission flag (**MAX_RT_FLAG**).

Related registers: **MAX_RT** and **MAX_RT_ERR** registers of each sub-channel.

Clearing: The maximum retransmission error (**MAX_RT_ERR**) of each sub-channel is reset when it is read.

20.3.11 Remote Error

Flag:

- **REM_ERR_FLAG**
- **REM_ERR_FLAG_A**
- **REM_ERR_FLAG_B**

OEN (Output Enable):

- **REM_ERR_OEN**
- **REM_ERR_OEN_A**
- **REM_ERR_OEN_B**

Description: Each GMSL2 device can report the ERRB status of the remote-side serial link device if the GMSL link is locked. See detailed description in the Remote Error Reporting section.

Related registers:

ERR_TX_EN, ERR_TX_ID
ERR_TX_EN_A, ERR_TX_ID_A
ERR_TX_EN_B, ERR_TX_ID_B
REM_ERR_OEN, REM_ERR_FLAG
REM_ERR_OEN_A, REM_ERR_FLAG_A
REM_ERR_OEN_B, REM_ERR_FLAG_B
ERR_RX_EN, ERR_RX_ID, ERR_RX_RECVD
ERR_RX_EN_A, ERR_RX_ID_A, ERR_RX_RECVD_A
ERR_RX_EN_B, ERR_RX_ID_B, ERR_RX_RECVD_B
ERR_RX_EN_C, ERR_RX_ID_C, ERR_RX_RECVD_C
ERR_RX_EN_D, ERR_RX_ID_D, ERR_RX_RECVD_D
ERR_RX_EN2_A, ERR_RX_ID2_A, ERR_RX_RECVD2_A

ERR_RX_EN3_A, ERR_RX_ID3_A, ERR_RX_RECVD3_A
 ERR_RX_EN4_A, ERR_RX_ID4_A, ERR_RX_RECVD4_A
 ERR_RX_EN2_B, ERR_RX_ID2_B, ERR_RX_RECVD2_B
 ERR_RX_EN3_B, ERR_RX_ID3_B, ERR_RX_RECVD3_B
 ERR_RX_EN4_B, ERR_RX_ID4_B, ERR_RX_RECVD4_B

Refer to the device data sheet register descriptions for more detailed information.

Clearing: Resolve the error status on the remote device.

20.3.12 Video Line CRC Error

Flag:

- LCRC_ERR_FLAG

OEN (Output Enable):

- LCRC_ERR_OEN

Description: The video line CRC checker detects pixel corruption within video line packets with very high probability. Flag is asserted when a video line CRC error is detected. By default, the GMSL link has video line CRC checking enabled. Video line CRC is a 32-bit code appended to each video line and is transferred to the receiver through info frames. See the [CRC Error Detection and ARQ Error Correction](#) section for additional information.

Related registers: LCRC_ERR, LINE_CRC_EN, LINE_CRC_SEL

Clearing: The video line CRC flag is a latching bit that resets to 0 when read or the link establishes lock after loss of lock.

20.3.13 Video Pixel CRC Error

Flag:

- VID_PXL_CRC_ERR_FLAG

OEN (Output Enable):

- VID_PXL_CRC_ERR_OEN

Description: The video pixel CRC checker detects pixel corruption within video packets with very high probability. Flag is asserted when at least one video pixel CRC error is detected (VID_PXL_CRC_ERR > 0).

GMSL links have video pixel CRC checking disabled by default; video line CRC (enabled by default) provides sufficient coverage. Video pixel CRC is enabled by setting TX_CRC_EN = 1 in the video transmit register block for each video pipe in the GMSL2 serializer and RX_CRC_EN = 1 in the video receive register block for each video pipe in the GMSL2 deserializer. When enabled, each video packet carrying 36 pixels is protected by a 16-bit CRC value. See the [CRC Error Detection and ARQ Error Correction](#) section for additional information.

Related registers: TX_CRC_EN_VIDEO, RX_CRC_EN_VIDEO

Clearing: Video pixel CRC error counter is reset to 0 when it is read or due to Auto Error Reset function or due to link gaining lock after loss of lock. When video pixel CRC error counter is reset, its error flag is also reset.

20.3.14 Video Sequence Error

Flag:

- `VID_SEQ_ERR`

OEN (Output Enable):

- `VID_SEQ_ERR_OEN`

Description: Flag is asserted when the deserializer detects nonsequential video sequence numbers. This indicates errors in the video transmission, dropped video packets, and/or incorrect sequencing of packets combined from the two PHYs in dual-link mode. This flag is latching.

Related registers: None

Clearing: Flag is cleared when read.

20.3.15 Video Block Length Error

Flag:

- `VID_BLK_LEN_ERR`

OEN (Output Enable):

- `VID_BLK_LEN_ERR_OEN`

Description: Flag is asserted when the deserializer detects incorrect video packet block length. This flag is latching.

Related registers: None

Clearing: Flag is cleared when read.

20.3.16 Video Mask Error

Flag:

- `VIDEO_MASKED_0_FLAG`
- `VIDEO_MASKED_1_FLAG`
- `VIDEO_MASKED_2_FLAG`
- `VIDEO_MASKED_3_FLAG`

OEN (Output Enable):

- `VIDEO_MASKED_0_OEN`
- `VIDEO_MASKED_1_OEN`
- `VIDEO_MASKED_2_OEN`
- `VIDEO_MASKED_3_OEN`

Description: Flag is asserted when video masking is active. Video masking means that the specific video pipe is outputting blank video lines (data = 0's, black pixels). Up to three video pipes can lose `VIDEO_LOCK` and data continues to output.

Note: This error is only available on GMSL2 CSI-2 Quad Deserializers operating in synchronous aggregation modes (4WxH or Wx4H) and on video pipes 0-3.

Related registers: `VIDEO_MASKED_OEN`, `VIDEO_MASKED_FLAG`, `MIPI_PHY_15`, `MIPI_PHY_16`, `MIPI_TX52`

Clearing: The `video_masked` registers (0x934, 0x974, 0x9B4, 0x9F4) indicate whether a video pipe / MIPI output is currently masked or was previously masked. `Video_Mask_Latch_Reset` (0x8C7) resets the `video_masked_latched` registers (0x934, 0x974, 0x9B4, 0x9F4). Video pipe are individually configured to output status to the ERRB pin with `VIDEO_MASKED_x_OEN` ERRB output enable (0x49) and the associated flags (0x4A).

20.3.17 Video PRBS Error

Flag:

- VPRBS_ERR_FLAG

OEN (Output Enable):

- VPRBS_ERR_OEN

Description: Flag is asserted when the Video PRBS checker detects at least one error (VPRBS_ERR > 0) while the Video PRBS test is running. See the *PRBS Testing* section for additional information.

Related registers: VPRBS_CHK_EN, VPRBS_ERR

Clearing: The video PRBS error counter is reset to 0 and the flag is cleared when read.

20.3.18 Frame Sync Error

Flag:

- FSYNC_ERR_FLAG

OEN (Output Enable):

- FSYNC_ERR_OEN

Description: Flag is asserted when the Frame Sync error counter (FSYNC_ERR_CNT) exceeds the Frame Sync error threshold (FSYNC_ERR_THR). The Frame Sync block can continuously monitor the received VS signals and assert an error when all VS pulses are not within a certain window.

Related registers: FSYNC_ERR_CNT, FSYNC_ERR_THR, FSYNC_MODE, FRM_DIFF_ERR_THR, OVLW_WINDOW, EN_FSIN_LAST

Clearing: The Frame Sync error counter is reset to 0 when it is read. When the error counter is reset, its error flag is also reset (the counter value of 0 is always less than the error threshold value).

20.3.19 FEC Corrected and Uncorrectable Errors

Flag:

- FEX_RX_ERR_FLAG

OEN (Output Enable):

- FEC_RX_ERR_OEN

Description: Flag is asserted when the GMSL *Forward Error Correction* (FEC) decoder's uncorrected error counter (UNCORRECTED_BLOCKS) exceeds uncorrected error threshold (UNCORRECTED_ERROR_THRESHOLD) or the corrected error counter (BIT_ERRS_CORRECTED) exceeds corrected error threshold (BIT_ERRS_CORRECTED_THRESHOLD).

Related registers: UNCORRECTED_BLOCKS, UNCORRECTED_ERROR_THRESHOLD, BIT_ERRS_CORRECTED, BIT_ERRS_CORRECTED_THRESHOLD

Clearing: Write 1 to CLEAR_STATS, CLEAR_BLOCKS_UNCORRECTABLE or CLEAR_BITS_CORRECTED.

20.3.20 ADC Interrupt

Flag:

- ADC_INT_FLAG

OEN (Output Enable):

- ADC_INT_OEN

Description: ADC interrupt. Asserted whenever any monitored ADC interrupt is asserted.

Related registers: ADC_INTR0→ADC_INTR3, ADC_LIMITX_0→ADC_LIMITX_3 (where X can be 0→7).

Clearing: The flag is cleared when the respective latched error source register is read.

20.3.21 VDD Comparator Error

Flag:

- VDDCMP_INT_FLAG

OEN (Output Enable):

- VDDCMP_INT_OEN

Description: Flag is asserted when the internal voltage comparator detects that selected internal VDD node is below a threshold. See CMP_STATUS description. This flag is latching.

Related registers: CMP_STATUS, CMP_STATUS_MASK, VDDCMP_MASK

Clearing: Flag is cleared when read.

20.3.22 VDD Brown Out Error

Flag:

- VDDBAD_INT_FLAG

OEN (Output Enable):

- VDDBAD_INT_OEN

Description: Flag is asserted when the internal voltage comparator detects either VDD or VDDA has dropped below 0.82V. This flag is latching.

Related registers: VDDBAD_STATUS

Clearing: Flag is cleared when read.

20.3.23 PORZ VDD Status Interrupt

Flag:

- PORZ_INT_FLAG

OEN (Output Enable):

- PORZ_INT_OEN

Description: Flag is asserted when the internal voltage comparator detects that VDD18 is below 1.516V during power-up. This flag is latching.

Related registers: PORZ_STATUS

Clearing: Flag is cleared when read.

20.3.24 VDD Overvoltage Interrupt

Flag:

- VDD_OV_FLAG

OEN (Output Enable):

- VDD_OV_OEN

Description: Flag is asserted when the internal voltage comparator detects that internal VDD, VDD1P8, or VREG voltage is above the threshold set by OV_LEVEL, OV1P8_LEVEL, or OVREG_LEVEL registers and the video channel is active (locked).

Related registers:

VDD_OV_LIVE, OV_LEVEL, OV_HYSTERESIS, VDD1P8_OV_LIVE, OV1P8_LEVEL, OV1P8_HYSTERESIS, VREG_OV_LIVE, OVREG_LEVEL, OVREG_HYSTERESIS

Clearing: Flag is cleared when read.

20.3.25 MIPI Rx Error

Flag:

- MIPI_ERR_FLAG

OEN (Output Enable):

- MIPI_ERR_OEN

Description: Flag is asserted when MIPI Rx controller detects an error.

Related registers: MIPI_RX12→MIPI_RX20

Clearing: The flag is cleared when the respective latched error source register is read.

20.3.26 Internal Memory Error

Flag:

- MEM_INT_ERR_FLAG

OEN (Output Enable):

- MEM_INT_ERR_OEN

Description: Some GMSL2 devices can continuously monitor internal memories for errors and assert this flag when an error is detected. The flag is latching. When a memory error is detected, the host SoC can decide to discard the transmitted video frame.

Related registers: DV_MEM_CRC_ERR, DV_MEM_CRC_ERR_A, DV_MEM_CRC_ERR_B, BACKTOP_MEM_CRC_ERR

Clearing: The flag is cleared when MEM_INT_ERR_FLAG is read.

20.3.27 Retention Memory CRC Error

Flag:

- RTTN_CRC_INT

OEN (Output Enable):

- RTTN_CRC_ERR_OEN

Description: Flag is asserted when a CRC error is detected after coming out of sleep mode (after reading retention memory).

Related registers: INJECT_RTTN_CRC_ERR

Clearing: This flag is cleared when read.

20.3.28 eFuse CRC Error

Flag:

- EFUSE_CRC_INT

OEN (Output Enable):

- EFUSE_CRC_ERR_OEN

Description: When the EFUSE is programmed during the device's production testing, a CRC signature is added to the last two locations of the EFUSE array. This signature is generated using the programming data for the remainder of the EFUSE array. On power-up, the contents of the EFUSE are

read by the device, and its contents are used to initialize and configure device-specific registers. As part of this initial read, the device recomputes the CRC based on the read values and checks it against the signature programmed into the array. If this signature is incorrect, the contents of the EFUSE have changed since the initial programming and the device asserts the ERRB flag to signal an issue.

Related registers: [INTR6](#), [INTR7](#)

Clearing: This flag is cleared when read.

20.4 Debug Techniques

The error generator function ([ERRG_EN](#)) can be used to evaluate a device's error detection and reporting capabilities. The error generator injects bit or burst errors to the outgoing link (that is, the forward channel in serializer and the reverse channel in deserializers). This function can be used to evaluate the decoding error ([DEC_ERR](#)), idle error ([IDLE_ERR](#)), retransmission count ([RT_CNT](#)), maximum retransmission error ([MAX_RT](#)), Audio PRBS, and Video PRBS error detection and reporting.

21. CRC Error Detection and ARQ Error Correction

21.1 Overview

GMSL2 serial links incorporate 16-bit Cyclic Redundancy Check protection for error detection of control channel (including I²C, UART, SPI, and GPIO), video, and audio data. The 16-bit CRC can also include Automatic Repeat Request (ARQ) error correction on both the forward and reverse channels (not available for video data). Video data is protected with 32-bit CRC error detection by default.

Note: Forward Error Correction is used in GMSL2 devices to detect and correct bit errors occurring during the transmission of compressed video on the serial link. See the [Forward Error Correction](#) for additional details.

CRC ensures link errors caused by EMI or other noise events do not corrupt control channel data.

Note: Video data can be protected with either 16-bit or 32-bit CRC error detection. See [Video Data CRC](#) for more information.

21.2 CRC Operation

21.2.1 16-Bit CRC

Every video, audio, and control channel packet (excluding idle and acknowledge packets) can be protected with 16-bit CRC. For enabled channels, the CRC block generates a 16-bit code (calculated with the polynomial as shown in the equation) that is appended to each packet. All low-bandwidth control channels have packet CRC enabled by default; each packet type can be individually configured to enable or disable CRC protection. The primary and pass-through control channels are protected in both I²C and UART modes.

The 16-bit packet CRC generator calculates following polynomial:

$$\text{CRC16} = x^{16} + x^{15} + x^2 + 1$$

Note: This is the same polynomial used in the USB protocol for data packets.

21.2.2 32-Bit CRC

Video data can be protected with a 32-bit CRC calculated per video line. The CRC block generates a 32-bit code that is appended to each DE (default) or HS (selected by register) pulse. This code is transferred to the receiver side of the serial link through info frames. The receiver-side CRC checker generates the same code and checks if the CRC codes match. An error is asserted if the codes do not match. The CRC check is processed at every falling edge of DE (or HS) on the receiver side even if the info frame is not received.

The 32-bit packet CRC generator calculates the following polynomial:

$$\text{CRC32} = x^{32} + x^{31} + x^{28} + x^{25} + x^{24} + x^{23} + x^{21} + x^{18} + x^{11} + x^8 + x^7 + x^6 + x^5 + x^3 + x^1 + 1$$

21.2.3 Video Data CRC

Video data can be protected with either 16-bit or 32-bit CRC. These are two different error detection schemes: Video Pixel CRC (16-bit) and Video Line CRC (32-bit). Video Pixel CRC detects pixel corruption in each packet of video data (36 pixels) with a 16-bit CRC value. Video Line CRC detects pixel corruption in each line of video data with a 32-bit CRC value. By default, Video Line CRC is enabled, and Video Pixel CRC is disabled.

Video Line CRC (default) provides robust video data protection with minimal bandwidth overhead. However, there are use cases where Video Pixel CRC may be the preferred video data CRC scheme:

- Applications that require errors to be detected as soon as possible.
- Applications where the rough location of the error in the line would like to be known.

21.3 Automatic Repeat Request/Automatic Retransmission

Automatic Repeat Request (ARQ) is an automatic packet retransmission method used to ensure data integrity on communications channels with low-bandwidth control data. ARQ works in conjunction with 16-bit packet CRC to detect whether packets are received without error or not. With a successful data transmission, the ARQ mechanism on the transmit side receives confirmation of error-free transmission from the receiver side. In the case of a transmission error (example, corrupted or dropped packet), the ARQ on the transmit side does not receive confirmation of an error-free transmission and automatically retransmits the packet.

21.3.1 ARQ Operation

Each control packet is appended by a 4-bit sequence number that continuously increments and rolls over. The transmitter saves the last 15 packets transmitted on each communication channel. When the receiver receives a control packet, it checks that the CRC and sequence number are correct. If both are validated, the receiver sends an acknowledgement packet back to notify that the packet with a certain sequence number has been correctly received. If the transmitter does not get an acknowledge packet from the receiver for a data packet with a certain sequence number, it automatically retransmits the data packet after reading it from the internal packet memory.

The 4-bit sequence number allows the transmitter to transmit up to 15 packets without receiving acknowledgment from the receiver. This allows pipelined operation which minimizes the effects of round-trip latency with acknowledge packets and increases the continuous available bandwidth.

The acknowledge packet uses the same header field as low-bandwidth packets, but it begins with a different special symbol to distinguish it from regular data packets. This simplified format keeps retransmission exchanges independent from the communication channel. Note that this smaller packet format contains no data, obviating the need for full 16-bit CRC. Instead, acknowledge packets are protected by a 5-bit CRC (polynomial: $x^5 + x^2 + 1$). The acknowledge packets include the same 4-bit sequence number of the correctly received data packet.

21.3.1.1 Go-Back-N

The GMSL2 ARQ mechanism operates using the 'Go-Back-N' principle, where N = 15 for a 4-bit sequence number. This 15-packet sliding window improves bandwidth usage and availability as 15 packets can be transmitted without receiving an acknowledge.

Figure 81 demonstrates 'Go-Back-N' where $N = 3$.

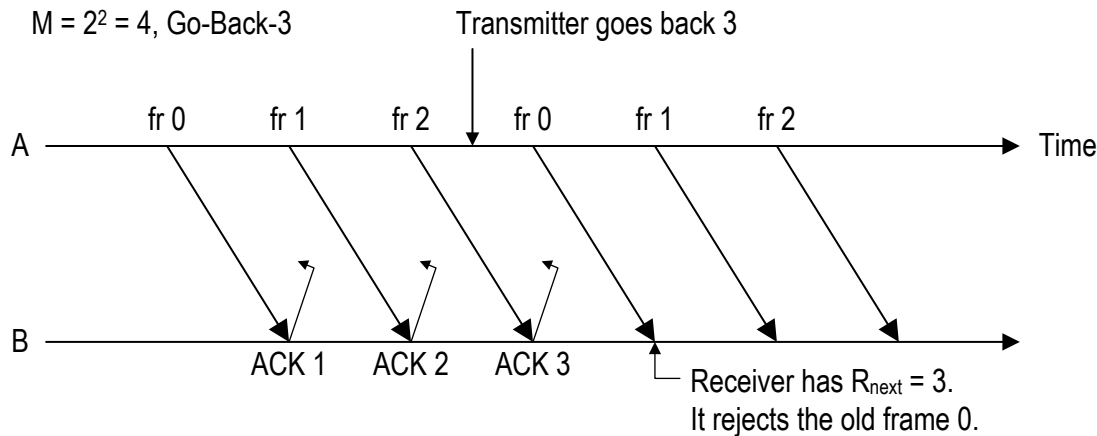


Figure 81. Operation of Go-Back-N ARQ with Go-Back-3

In this example, the transmitter sends up to three packets. The transmitter begins waiting for the acknowledge after the first packet is sent; if more than one packet is sent, waiting and transmission of the additional packets is handled in parallel. If the expected acknowledge does not arrive, the transmitter goes back three and retransmits the packet. Any retransmission attempt is counted and stored. This retransmission process repeats until the package is either acknowledged or the maximum number of retransmissions is reached. Upon reaching the maximum, the packet is assumed to be acknowledged, it is dropped from the transmission queue, and the send window moves forward to transmit subsequent packets. In cases where the maximum number of retransmissions is met, a register error is flagged and the ERRB pin changes state to signal 'Error Output.'

Note: In the actual GMSL2 ARQ system, $N = 15$.

A 3-bit register is allocated to set the maximum number of retransmissions. A packet can be retransmitted up to seven times depending on the register value set in `MAX_RT` for each communication channel.

The receiver side has a receiver window of size one and only waits for one packet with a match to the expected sequence number. The receiver passes a packet to the corresponding adapter if the packet is received completely without error and the sequence number matches the expected value. Here, the receiver also sends an acknowledge packet back to the transmitter. In the case of an error or unmatched sequence number, the receiver drops the received packet.

21.3.1.2 ARQ Path

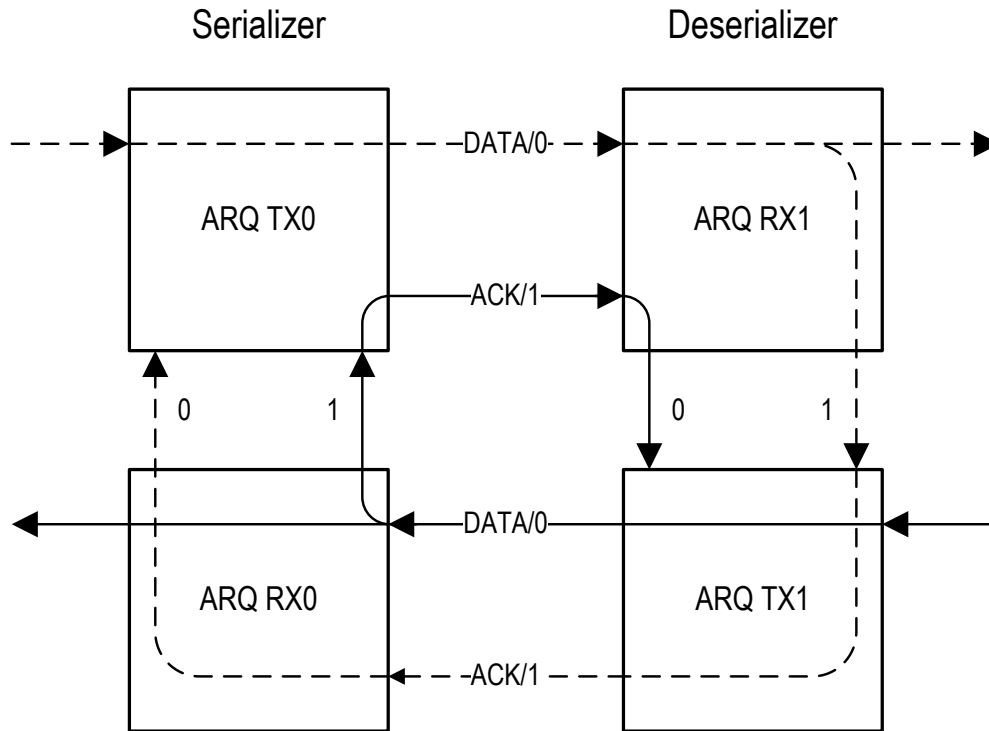


Figure 82. ARQ Path

The dashed line in the [Figure 82](#) shows the data flow from the serializer to the deserializer and the subsequent acknowledge from the deserializer to the serializer. The solid line depicts the data flow from the deserializer to the serializer and the subsequent acknowledge from the serializer to the deserializer. The ARQ blocks of both the serializer and the deserializer can operate simultaneously and support bidirectional data transfer(s). For example, ARQ TX0 (in the serializer) can transmit either data or an acknowledge of data received from the deserializer. The same function is performed by ARQ TX1 in the deserializer. Here, ARQ TX1 can transmit either data or an acknowledge of data received from the serializer. The ARQ RX blocks operate in a similar manner with the corresponding ARQ TX blocks. This coordination facilitates bidirectional operation.

21.4 CRC/ARQ Configuration

The following sections describe how to enable/disable and configure CRC and ARQ in GMSL2 devices. CRC is enabled by default for all packet types; ARQ is enabled by default for all supported packet types. ARQ does not support video.

21.4.1 32-Bit CRC Configuration

To enable Video Line CRC, set the bit `LINE_CRC_EN` to 1 in the serializer for each video pipe that needs CRC protection. Additionally, set `LINE_CRC_EN` to 1 for the deserializer video pipe that is receiving the video. This is enabled in all serializers and deserializers by default.

The CRC code can be appended to either the DE or HS pulse. This is configured with the `LINE_CRC_SEL` bit. Write 0 to use the DE pulse for video line CRC and write 1 to use the HS pulse.

21.4.1.1 32-Bit Video CRC Errors

Video Line CRC errors are reported to the **LCRC_ERR_FLAG** bit in the deserializer register **INTR7**. To enable/disable the output of this error flag to the **ERRB** pin, set the **LCRC_ERR_OEN** bit in the deserializer register **INTR6**.

21.4.2 16-Bit CRC Configuration

To enable packet CRC for the audio or control channels, set the bit **TX_CRC_EN** to 1 in the serializer for each packet type that needs CRC protection. Additionally, set **RX_CRC_EN** to 1 for the corresponding deserializer packet types. This is enabled for all audio and control channel packet types by default.

Table 90 lists the packet types that can enable/disable 16-bit CRC with their corresponding registers.

Table 90. 16-Bit CRC Registers

16-BIT CRC PACKET TYPE	REGISTER BLOCK*	REGISTER NAME
Video X [†]	VIDEO_X	TX0
Video Y [†]	VIDEO_Y	TX0
Audio X	AUDIO_X	TR0
Audio Y	AUDIO_Y	TR0
SPI	SPI	TR0
GPIOs	GPIO	TR0
HDCP Control Packets	AHDCP	TR0
Main Primary Control Channel [‡]	CC	TR0
Pass- through Channel 1	IIC_X	TR0
Pass- through Channel 2	IIC_Y	TR0

* Register block name may vary by device family. Refer to the device data sheet for specific register block name. For example, devices with only one audio block, the audio registers are named **AUDIO** without “_X” or “_Y” suffix.

[†] Not enabled by default. Video Line CRC should be disabled if Video Pixel CRC is enabled.

[‡] CRC and ARQ should not be disabled on the primary control channel packets.

Note: CRC configuration for pass-through Channel 1 (in I²C or UART mode) is controlled by register **IIC_X_TR0** and CRC configuration for pass-through Channel 2 (in I²C or UART mode) is controlled by register **IIC_Y_TR0**.

21.4.2.1 16-Bit Video CRC Errors

Video Pixel CRC errors are reported to the `VID_PXL_CRC_ERR_FLAG` bit in the deserializer register `INTR7`. To enable/disable the output of this error flag to the `ERRB` pin, set the `VID_PXL_CRC_ERR_OEN` bit in the deserializer register `INTR6`.

21.4.3 16-Bit ARQ Configuration

The ARQ configuration registers for each channel are listed in [Table 91](#). By default, ARQ is enabled for all channels (except Video Pixel), and error reporting to the `ERRB` pin is enabled.

Note: ARQ is not available for 16-bit Video Pixel CRC.

Table 91. 16-Bit ARQ Registers

CONTROL CHANNEL PACKET TYPE	REGISTER BLOCK*	REGISTER NAMES
Video X [†]	Not available	Not available
Video Y [†]	Not available	Not available
Audio X	AUDIO_X	ARQ0 – ARQ2
Audio Y	AUDIO_Y	ARQ0 – ARQ2
SPI	SPI	ARQ0 – ARQ2
GPIOs	GPIO	ARQ0 – ARQ2
HDCP Control Packets	AHDCP	ARQ0 – ARQ2
Main Primary Control Channel [‡]	CC	ARQ0 – ARQ2
Pass-through Channel 1	IIC_X	ARQ0 – ARQ2
Pass-through Channel 2	IIC_Y	ARQ0 – ARQ2

* Register block name may vary by device family. Refer to the device data sheet for specific register block name.

[†] ARQ is not available.

[‡] CRC and ARQ should not be disabled on the primary control channel packets.

Note: ARQ configuration for pass-through Channel 1 (in I²C or UART mode) is controlled by register `IIC_X_ARQ0 – ARQ2` and ARQ configuration for pass-through Channel 2 (in I²C or UART mode) is controlled by register `IIC_Y_ARQ0 – ARQ2`.

If there is a CRC error, the corrupted packet is retransmitted as described in the [ARQ Operation](#) section.

21.4.3.1 Reporting of CRC Errors/ARQ retries

ARQ can only be enabled on channels with CRC enabled. If any of the CRC-enabled control channels reports an error, the corrupted/dropped packet is retransmitted with the ARQ scheme. If the bit `RT_CNT_OEN` (register `xx_ARQ1`) is set high for a given channel, the flag bit `RT_CNT_FLAG` (register `INTR5`) is set high; if the main `RT_CNT_OEN` (in register `INTR4`) is enabled, the error is reported to the `ERRB` pin.

The number of ARQ retries is reported to `RT_CNT[6:0]` (register `xx_ARQ2`) for each channel.

21.4.3.2 Reporting of Maximum ARQ retries

If the number of retries for a channel exceeds a threshold (set by `MAX_RT[2:0]`), it is reported to `MAX_RT_FLAG`. If the `MAX_RT_ERR_OEN` (register `xx_ARQ1`) is set high, the error is reported to the main `MAX_RT_FLAG` (register `INTR5`) error bit. If `MAX_RT_OEN` (register `INTR4`) is set high, the error is reported to the `ERRB` pin.

22. Voltage Monitoring

22.1 Overview

GMSL2 devices monitor various onboard supply voltages and provide alerts for overvoltage or undervoltage conditions. Dedicated status register flags are driven by internal comparators that measure each supply voltage relative to an internal voltage reference. These registers are latching status flags. Supply voltages with latching status flags retain alerts and can be used to capture temporary conditions and be read back later (at which time the previously latched state is cleared). Many status flags have user-configurable parameters including custom voltage thresholds and the option to report status registers to the ERRB pin (which asserts low when an errant condition is sensed). See the [GMSL2 Error Reporting \(ERRB Pin\)](#) section for additional information.

Note: Each GMSL2 device has a unique combination of voltage monitoring and status/error reporting mechanisms. A small subset of the devices provides additional power supply monitoring capabilities through an integrated ADC.

22.1.1 Architecture

Most GMSL2 devices include a common set of power supplies that provide power to the digital core (VDD_SW), GMSL link circuitry (VDD18), and general GPIO pins (VDDIO). All devices include undervoltage monitoring on the common set of power supplies with some providing extended monitoring capabilities. Each GMSL2 product family has unique, family-specific power supplies with dedicated functionality. Examples here include HDMI power (VDD33) and MIPI output power (VTERM). Except for VTERM, these family-specific power supplies are not typically monitored.

The following table details the power supplies for which voltage monitoring is available ([Table 92](#)).

Table 92. Power Supply Monitoring Functions Available

SUPPLY NAME	DESCRIPTION	MONITOR FUNCTIONS AVAILABLE
VDD_SW	Internal 1V digital core supply	Undervoltage (all devices) Overvoltage (some devices)
VDD18	GMSL 1.8V supply	Undervoltage (all devices)
VDDIO	1.8V to 3.3V I/O supply	Undervoltage (all devices)
VTERM	1.2V MIPI CSI-2 Output Supply	Undervoltage (CSI-2 deserializers only)
VDD/VREG	1V LDO input	Not monitored
VDD33, VDDA, VDDIORG	Misc. special function supplies	Not monitored

The following block diagram ([Figure 83](#)) details the connectivity between the voltage monitor and internal power supplies. The naming convention detailed here is representative of most GMSL2 devices. In some cases, there may be dedicated analog VDD pins and/or explicit VDD regulator input pins (example, VREG). Note that VDD/VREG pins are not typically monitored.

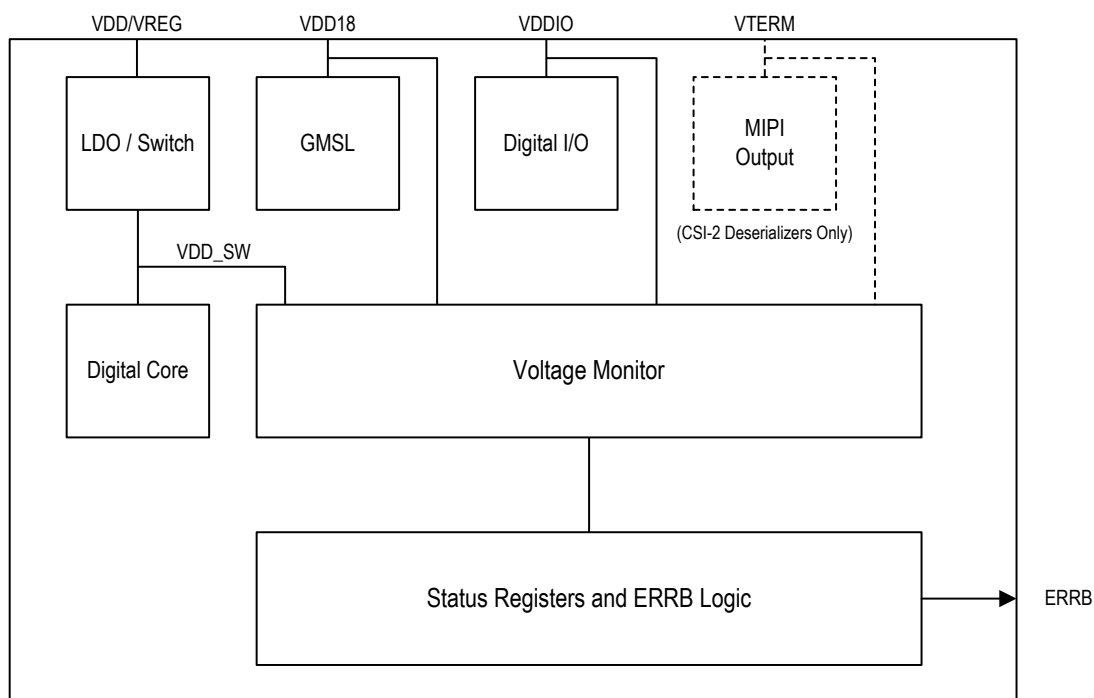


Figure 83. Voltage Monitor and Internal Power Pins

22.2 Operation

22.2.1 VDD_SW Monitoring Details

Undervoltage (UV) and overvoltage (OV) monitoring of VDD_SW (the primary 1V core supply) is included on nearly all GMSL2 devices (VDD_SW OV monitoring is not available on HDMI and advanced HDMI serializers). VDD_SW is typically derived from VDD/VREG either through an internal LDO or series switches.

22.2.1.1 Undervoltage Monitoring of VDD_SW

An undervoltage condition on VDD_SW nominally occurs when $VDD_SW < 0.82V$. Note that the precise threshold varies some between devices. Refer to the data sheet specific to the part number to verify the specified VDD_SW undervoltage threshold for a given device.

In the case of an undervoltage condition, the power manager triggers a reset of the digital core and resets all registers powered by VDD_SW. When power has recovered, the reset is released. The reset of the digital core ensures that a brownout does not result in corruption of the registers.

The presence of an undervoltage event is recorded through two status flags:

- **VDDBAD_STATUS[1]** and **VDDBAD_STATUS[0]** – latched high following undervoltage event.
- **CMP_STATUS[2]** – latched low following undervoltage.

The memory that maintains the **VDDBAD_STATUS** and **CMP_STATUS[2]** bits is powered by the 1.8V power supply; as a result, they are persistent following a reset triggered by a VDD_SW undervoltage event. This enables a VDD_SW undervoltage event to be reported following the recovery of the VDD

power supply, and the associated interrupt flags, `VDDBAD_INT_FLAG` and `VDDCMP_INT_FLAG`, can drive `ERRB` low to alert the system of a brownout.

To clear `VDDBAD_INT_FLAG`, the associated `VDDBAD_STATUS` bits must be read first to clear. After the `VDDBAD_STATUS` bits are cleared, `VDDBAD_INT_FLAG` can then be read, at which point the flag is cleared. The process to clear `VDDCMP_INT_FLAG` is similar (that is, the associated `CMP_STATUS` bit must be cleared first through reading).

22.2.1.2 Overvoltage Monitoring of `VDD_SW`

An overvoltage condition on `VDD_SW` is reported if the observed voltage exceeds a user-selectable threshold specified by the associated `OV_LEVEL` bit field. Refer to the data sheet specific to the part number for details regarding the specific threshold levels supported by a given device. If `VDD_SW` is greater than the specified threshold, the `VDD_OV_FLAG` is set. The `VDD_OV_FLAG` is a latching bit that is set when an overvoltage condition occurs and does not clear until read. `VDD_OV_FLAG` can be configured to drive the `ERRB` pin to alert the system of a fault condition. Note that operation of the `VDD_OV_FLAG` requires that the video path be active.

Note: The overvoltage flag for eDP/DP deserializers is `VDD_OV_INT_FLAG`.

22.2.2 VDD18 Monitoring Details

All GMSL2 devices include undervoltage monitoring of `VDD18`.

22.2.2.1 Undervoltage Monitoring of `VDD18`

An undervoltage condition on `VDD18` is nominally flagged when $VDD18 < 1.625V$. Note that the precise threshold varies some between devices. Refer to the data sheet specific to the part number to verify the specified `VDD18` undervoltage threshold for a given device.

In the case of an undervoltage condition, the status register bit `CMP_STATUS[0]` is latched low. The error can be flagged using the `VDDCMP_INT_FLAG`, which can be configured to drive the fault condition to the `ERRB` pin. The error status is cleared by first reading `CMP_STATUS[0]` and then reading `VDDCMP_INT_FLAG`.

22.2.3 VDDIO Monitoring Details

`VDDIO` includes undervoltage monitoring only; overvoltage monitoring is not available. This monitoring is available to all GMSL2 devices. An undervoltage condition on `VDDIO` is nominally flagged when $VDDIO < 1.625V$. Note that the precise threshold varies some between devices. Refer to the data sheet specific to the part number to verify the specified `VDDIO` undervoltage threshold for a given device.

In the case of an undervoltage condition, the status register bit `CMP_STATUS[1]` is latched low. The error can be flagged using the `VDDCMP_INT_FLAG`, which can be configured to drive the fault condition to the `ERRB` pin. The error status is cleared by first reading `CMP_STATUS[1]` and then reading `VDDCMP_INT_FLAG`.

22.2.4 VTERM Monitoring Details

`VTERM` includes undervoltage monitoring only; overvoltage monitoring is not available. An undervoltage condition on `VTERM` is nominally flagged when $VTERM < 1.0V$. Note that `VTERM` is only available in MIPI CSI-2 deserializers.

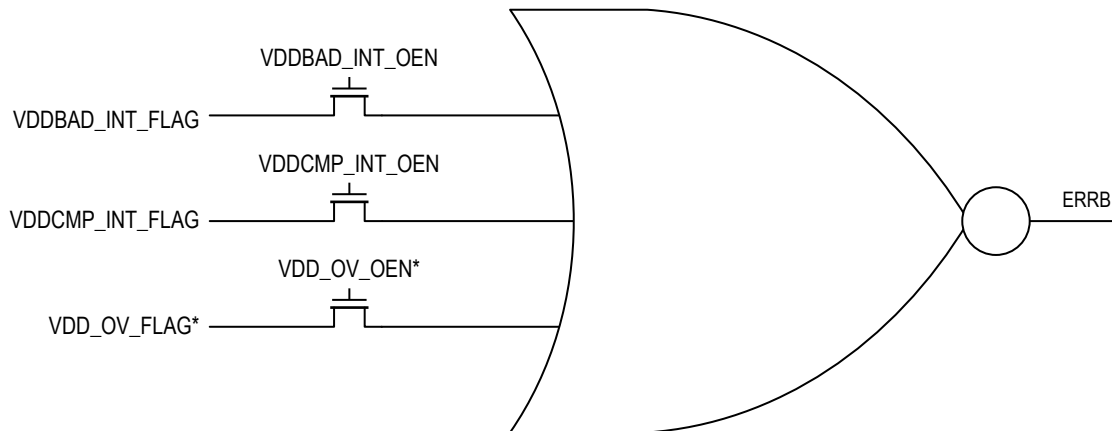
In the case of an undervoltage condition, the status register bit **CMP_VTERM_STATUS** is latched low. The error can be flagged using the **VDDCMP_INT_FLAG**, which can be configured to drive the fault condition to the ERRB pin. The error status is cleared by first reading the **PWR0** register and then reading **VDDCMP_INT_FLAG**.

22.3 Error Reporting and Status

22.3.1 ERRB Configuration

The ERRB pin can be used to notify the system of undervoltage and overvoltage conditions. Each of the available monitor functions can be separately routed to the ERRB pin as described in the above sections detailing the various voltage monitors. The user-configurable register fields are listed below. These register fields, when set to 1, enable the reporting of the associated undervoltage/overvoltage condition to the ERRB pins (that is, the error condition asserts ERRB low). [Figure 84](#) contains a block diagram of the relationship between the voltage monitoring interrupt enable registers and ERRB pin. See the [GMSL2 Error Reporting \(ERRB Pin\)](#) section for additional information.

- **VDDBAD_INT_OEN** – asserts ERRB low when **VDDBAD_INT_FLAG** = 1 (VDD_SW UV).
- **VDDCMP_INT_OEN** – asserts ERRB low when **VDDCMP_INT_FLAG** = 1 (VDD18, VDDIO, VTERM, and/or VDD_SW UV).
- **VDD_OV_OEN** – asserts ERRB low when **VDD_OV_FLAG** = 1 (VDD_SW OV).



*Note: The overvoltage flag for eDP/DP deserializers is **VDD_OV_INT_FLAG** and the ERRB configuration register is **VDD_OV_INT_OEN**

Figure 84. Voltage Monitoring ERRB Configuration

22.3.1.1 Latching Status Bits and Clearing Errors

The status flags that drive ERRB are latching. They are set in the event of an error condition; the error indication is persistent following recovery of the error. The flags and other associated bits are cleared automatically when read. Note that some of the flags are driven by status bits that are also latched and must be cleared prior to clearing the flag. Flags associated with overvoltage conditions only become active when the video path is active. If the video path is not active, the flag bits are not set in the event of an overvoltage.

23. Line Fault

23.1 Operation

Line-fault detection can be added to GMSL2 systems with the addition of two external resistors at each end of the serial link. Line-fault detection requires that both ends of the cable shield are tied to ground. The external resistor (R_{EXT}) is connected to the LMNx pins (that is, where the fault condition is to be detected). The external resistor (R_{PD}) is required on the nondetecting side of the link. This scheme detects various application fault conditions, including:

- Short-to-battery
- Short-to-ground
- Open-circuit
- Line-to-line short

The line-fault detection configuration options and status are accessible through registers. If unmasked, a line-fault condition asserts **ERRB**. See the [GMSL2 Error Reporting \(ERRB Pin\)](#) section for additional information.

Note: The external LMN resistor limits the current flowing into the LMN pin to less than 1mA in the event of a short to battery or ground.

23.2 Hardware Requirements

23.2.1 Coax Mode (Single-Ended)

The local side performs the line fault detection function. The local-side device requires a single 48.7k Ω resistor connected directly from an LMNx pin to the serial link. The remote side of the serial link requires a 49.9k Ω resistor connected to GND, making the remote side hardware-compatible with existing designs using GMSL1 devices. Any of the line-monitor pins may be used when the serial link is single-ended (coax). The line-fault signal assignment is shown in [Table 93](#).

Table 93. Line-Fault Signal Assignment to SIO, and Resistors in Coax Mode

SIGNAL	SIOP (IF SIOP IS THE ACTIVE PHY)	SION (IF SION IS THE ACTIVE PHY)
Line Fault Pin	LMN0–3 (any may be used) 48.7k Ω to serial link	LMN0–3 (any may be used) 48.7k Ω to serial link

Note: Line-fault detection can be implemented in either the serializer or deserializer; the orientation is dependent on where the microcontroller is located on the serial link system.

The two configurations for line-fault detection are shown in [Figure 85](#) and [Figure 86](#). Configuration 1 is typically used for display links and Configuration 2 is typically used for camera links; however, either configuration can be used in any serial link system.

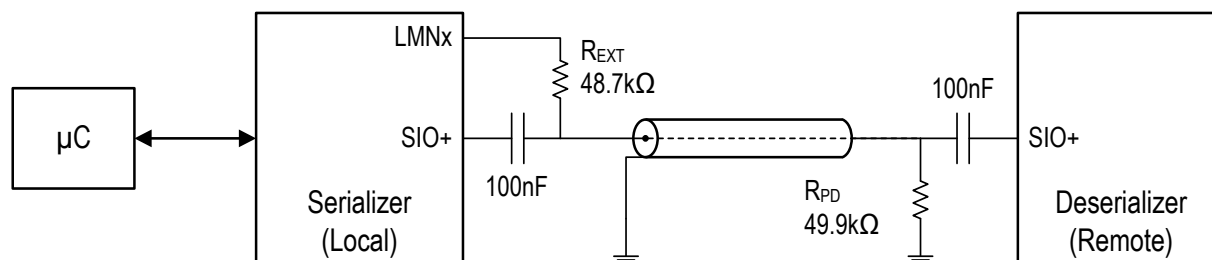


Figure 85. Line-Fault Configuration 1: Local-Side Serializer (Coax Mode)

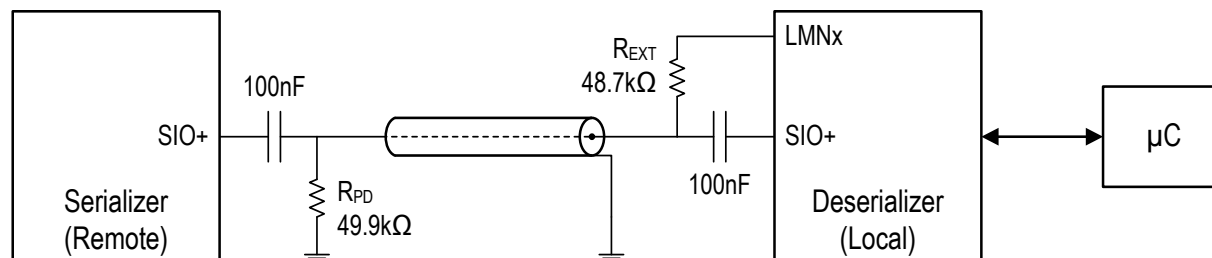


Figure 86. Line-Fault Configuration 2: Local-Side Deserializer (Coax Mode)

The LMNx pins (LMN0–LMN3) are typically mapped to different multifunctional pins on each unique part and package options. Some parts may have up to four line-fault detectors depending on the package options and pin availability. Refer to device-specific data sheets for more information.

23.3 Configuration

GMSL2 device register control allows access to enable the line-fault detectors, read the line-fault status codes, and program the line-fault interrupts. The line-fault registers are outlined in [Table 94](#).

Table 94. Register Mapping and Descriptions for the Line-Fault Registers

REGISTER NAME	BIT(S)	BIT NAME	BIT DESCRIPTION
REG5	3	PU_LF3	Power up Line-Fault Detector 3 (LMN3) if applicable
REG5	2	PU_LF2	Power up Line-Fault Detector 2 (LMN2) if applicable
REG5	1	PU_LF1	Power up Line-Fault Detector 1 (LMN1) if applicable
REG5	0	PU_LF0	Power up Line-Fault Detector 0 (LMN0) if applicable
REG27	6:4	LF_3[2:0]	LMN3 status (see Table 95 Decodes)
REG27	3:0	LF_2[2:0]	LMN2 status
REG26	6:4	LF_1[2:0]	LMN1 status
REG26	3:0	LF_0[2:0]	LMN0 status
INTR2	3	LFLT_INT_OEN	Sends line-fault interrupt to ERRB pin
INTR3	3	LFLT_INT	Line-fault interrupt asserted when any of the four enabled detectors indicates a fault condition

Note: The behavior of the line-fault interrupt can be selected to be either latching or live with the [LFLT_STKY_INT](#) bit (interrupt is live by default). Feature availability varies by device; refer to device-specific data sheet for availability details.

The line-fault detection status registers are encoded to represent various conditions. The code translations are presented in [Table 95](#). Unused detectors ($PU_LFx = 0$) return “010” (that is, the status code for ‘Normal Operation’).

Table 95. Line-Fault Detection Decode Table

LINE-FAULT CONDITION	LF_0,1,2,3 [2:0]
Short-to-battery	000
Short-to-GND	001
Normal Operation (no fault)	010
Open-line	011
Line-to-line short	1xx

23.3.1 Line-Fault Detection Application

To use line-fault detection, power up the line-fault detector (REG5) that corresponds with the pin used on the device that reports the error condition (local device). For example, if using pins LMN0 and LMN1, set the local-side line-fault REG5 register bits PU_LF0 and PU_LF1 to 1. Do not enable the line-fault detector on the remote side.

In normal operation, the status registers LF_0 [2:0] and LF_1 [2:0] each return “010” when read. If a fault is detected, the status registers change according to error condition (see [Table 95](#)). Unused line-fault detectors default to the “010” status code.

For example, assume LMN0 is connected to SIO+ through R_{EXT} as shown in [Figure 85](#). If the serial link (SIO+) is pulled to GND, LF_0 [2:0] returns a code of “001”, and the interrupt bit $LFLT_INT$ in the $INTR3$ register is set to 1. If $LFLT_INT_OEN$ in the $INTR2$ register is enabled, the $ERRB$ pin transitions low to reflect the line-fault interrupt condition.

23.3.1.1 Operation During Line-Fault Detection Events

The SIO+/SIO- pins are protected by AC coupling capacitors and are not exposed to high voltage caused by a line-fault event. In the event of a short-to-battery, the current into the LMN pin is limited to less than 1mA by the external LMN resistor. Following a line-fault event, no immediate action is required, however, system designers should take the appropriate steps to verify the system condition. $ERRB$ remains low until the line-fault event is reversed.

Note: If $ERRB$ is low, follow the procedure described in the [Example \$ERRB\$ System Reaction](#) section to determine the source of the error condition.

23.3.1.2 Line Fault with PoC Recommendation

Line-fault monitoring cannot be enabled while using PoC. Analog Devices recommends a supervisory power supply device for line fault detection and monitoring PoC voltage and current. The MAX20086–MAX20089 devices are designed specifically to work with all GMSL devices in automotive camera applications and are ASIL-compliant for safety-critical functions (ASIL-B and ASIL-D compliant versions are available). See the ASIL section for more information.

24. Error Generator

All GMSL2 devices have an error generator (ERRG) located after the packet scheduler to simulate the effect(s) of bit errors on the link. This is primarily used to test system-level reactions to serial link bit errors, including ASIL error handling in safety-relevant systems. The ERRG can also be used to validate internal self-tests (example, PRBS Testing).

24.1 Operation

The error generator is located within each PHY after the scheduler and packetizer. When enabled in the serializer, bit errors are added to the forward channel; when enabled in the deserializer, bit errors are added to the reverse channel. Bit errors are generated by flipping bits in the bitstream after data packetization and encoding, so there is no control over what channels (example, video, RMII) are affected.

24.2 Configuration

To enable the ERRG, first set the ERRG parameters in register **TX2** ([Table 96](#)). Then, enable the error generator in register **TX1** ([Table 97](#)) for all devices except CSI-2 Quad Deserializers, which use register **GMSL_x:TX1** ([Table 98](#)).

Note: To clear ERRB status after disabling ERRG mode (that is, **ERRG_EN_A** or **ERRG_EN_B** = 0), clear as described in the [GMSL2 Error Reporting \(ERRB Pin\)](#) section or perform a one-shot reset.

24.2.1 ERRG Parameters

The following parameters must be configured in register **TX2** before enabling the error generator:

- **ERRG_RATE[1:0]** – The error generation rate controls how often error events are triggered (that is, the generated BER). The recommended setting is **ERRG_RATE[1:0]** = 11. This sets a BER of 4.77×10^{-8} , which is four orders of magnitude worse than the expected worst-case BER for a compliant GMSL2 link.
- **ERRG_CNT[1:0]** – The error count determines how long to run the ERRG. In continuous mode, the ERRG produces errors as long as **ERRG_EN_x** is set high. For the other values of **ERRG_CNT**, the ERRG generates a specified number of error events at the rate specified by **ERRG_RATE[1:0]**. Setting **ERRG_EN_x** low resets the error count.
- **ERRG_PER[0]** – Sets the error generation mode. In periodic mode, the ERRG generates a bit flip at exactly the rate specified by **ERRG_RATE[1:0]**. In pseudorandom mode, the bit flip is generated at a random point by an internal pseudorandom number generator at a rate defined by **ERRG_RATE[1:0]** so that the average rate of error generation is equal to the rate specified.
- **ERRG_BURST[2:0]** – Error burst defines how many errors are generated per error event. By default, this is set to '000' and generates a single error per error event. If set to a value larger than 1 bit, a burst of sequential errors is generated: the first and last bits are flipped, and the other bits have a 50% probability of being flipped. For example, if a value of 8 bits is selected, the first and eighth bits get flipped, and bits 2–7 each have a 50% chance of being flipped.

Table 96. ERRG Parameters (Register TX2)

PARAMETER	BITFIELD	DECODE
Select the error generation mode	ERRG_PER[0]	0: Pseudorandom 1: Periodic
Set the burst length of errors to be generated at every error event	ERRG_BURST[2:0]	000: 1 bit 001: 2 bits 010: 3 bits 011: 4 bits 100: 8 bits 101: 12 bits 110: 16 bits 111: 20 bits
Set error event generation rate	ERRG_RATE[1:0]	00: 1 error in 5120 bits 01: 1 error in 81,920 bits 10: 1 error in 1,310,720 bits 11: 1 error in 20,971,520 bits
Set the number of error events to be generated after ERRG_EN is set high.	ERRG_CNT[1:0]	00: Continuous 01: 16 10: 128 11: 1024

24.2.1.1 ERRG Enable: All Device Families Except CSI-2 Quad Deserializers

Table 97. ERRG Enable (Register TX1)

PARAMETER	BITFIELD	DECODE
Enable the error generator for the desired GMSL PHY	ERRG_EN_A ERRG_EN_B	0: Disabled 1: Enabled

24.2.1.2 ERRG Enable: CSI-2 Quad Deserializers

Table 98. ERRG Enable (Register GMSL__x:TX1)

PARAMETER	BITFIELD	DECODE
Enable the error generator for the desired GMSL PHY	ERRG_EN	0: Disabled 1: Enabled

Note: There is a GMSL__x:TX1 register associated with each of the four GMSL PHYs.

Device Families

24.2.2 DSI vs. CSI-2

- DSI is Display Serial Interface and is focused on display applications; CSI-2 is Camera Serial Interface and is focused on camera applications.
- GMSL2 DSI serializers accept only 24-bit RGB888 color.
- The DSI and CSI-2 packet-level protocols are different and not interchangeable. A DSI transmitter must be matched to a DSI receiver; a CSI-2 transmitter must be matched to a CSI-2 receiver.
- The physical layer (PHY) is the same for most cases, but packet processing does not work for mismatched MIPI interfaces.
- All MIPI devices operate in the continuous clock mode only.

The MIPI Rx DSI interface receives MIPI data and converts it to parallel data to be sent into the video pipelines. The data is then serialized and sent across the link. The video pipes only contain generalized video pixel data and do not retain MIPI data.

25. HDMI Serializer

25.1 Overview

All GMSL2 HDMI serializers convert audio and video from HDMI 1.4 or 2.0 input to single or dual GMSL2 output.

GMSL2 HDMI serializers support a maximum HDMI video pixel clock rate of 430MHz.

See [Feature Availability by Device](#) for a list of GMSL2 HDMI devices.

A functional block diagram ([Figure 87](#)) shows the video path from the HDMI input pins to the video pipes.

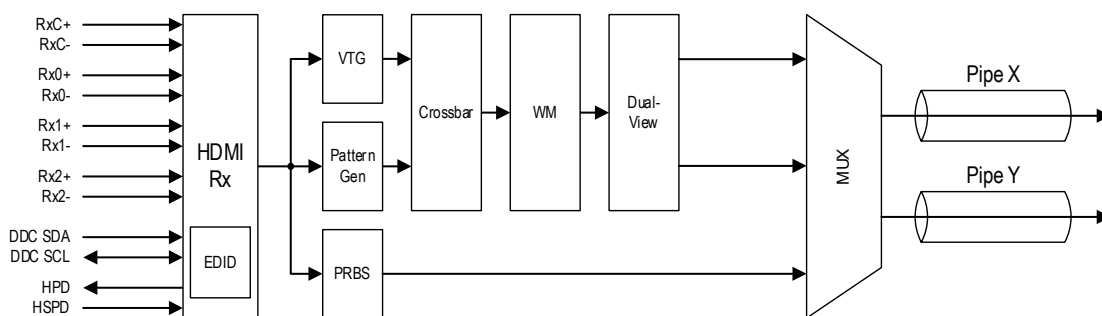


Figure 87. Block Diagram of the Video Path Through GMSL2 HDMI Serializers

25.2 Operation

The HDMI receiver in the GMSL2 HDMI serializers supports RGB888 format video input. The HDMI receiver includes the following functions/features:

- HDMI 1.4 and 2.0 input.
- HDMI audio receiver and transmission over the forward channel.

25.2.1 Use Cases: Embedded and Consumer Port Applications

25.2.1.1 Embedded Applications

GMSL2 HDMI serializers are most used in embedded applications, where the video source (typically an SoC) is installed on the same PCB as the GMSL2 serializer ([Figure 88](#)). In these implementations, a PCB trace provides the HDMI connection from the video source to the serializer; the sub-system is contained within the PCB, and there are no external off-board connections. This use case requires careful consideration of PCB design and layout to ensure good signal quality on the HDMI connection. HDMI compliance testing is not required for this application.

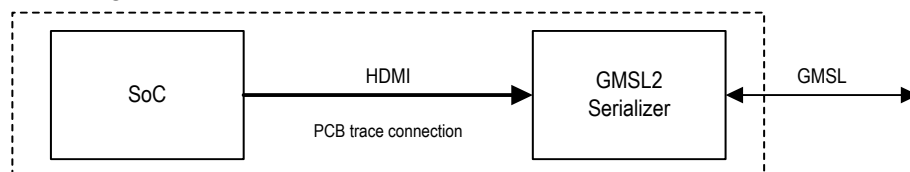


Figure 88. Embedded HDMI Application

25.2.1.2 Consumer Port Applications

HDMI consumer port applications integrate an off-board HDMI connection ([Figure 89](#)) that allows end users to connect an external video source. Rear seat infotainment systems with an external HDMI connector are a common example of consumer port applications. This use case typically requires HDMI compliance certifications. To be compliant, the PCB layout must meet HDMI hardware requirements, and the serializer must be configured according to the register optimization process (see [HDMI Receiver Optimization Configuration](#)).

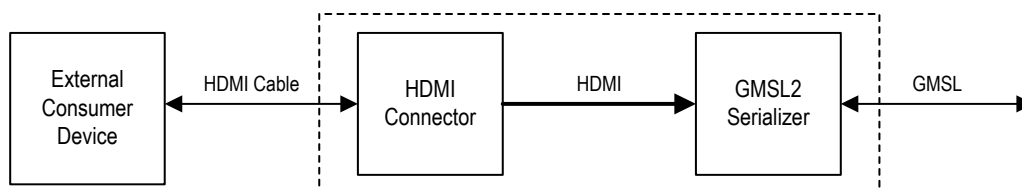


Figure 89. Consumer Port Application

25.3 Configuration

The following configuration steps are used to enable video in the HDMI serializers.

- Configure the EDID table (if required). See [HDMI EDID Tables](#) for details.
- Set the HDMI mode. See the [HDMI Mode \(HDMI 1.4 or HDMI 2.0\)](#) section for configuration details.
- For consumer port applications, write HDMI receiver optimization registers ([HDMI Receiver Optimization Configuration](#)).
- Set up dual-view registers (registers [DV0](#) and [DV2](#)). These are disabled by default. See the [Dual-View](#) section for additional details.
- Set the stream ID for video. The [TX_STR_SEL](#) bits set the stream ID mapping for each video pipe.
 - a. [TX3 \(0x0053\)](#): Controls pipe X (Default stream ID is 0). Selects which stream ID to transmit the main HDMI video stream.
 - b. [TX3 \(0x0057\)](#): Controls pipe Y (used when dual-view is enabled). See the [Video Pipes](#) section for additional information.
- Enable pixel repetition mode (if required). Pixel repetition mode must be used for PCLK values under 25MHz. See the [HDMI Pixel Repetition Mode](#) section for additional information.
- Configure HDMI audio (if required).
- Enable HDMI input block by setting [HDMI_AUTOS](#) = 1 (register [REG1](#)). This is required to receive HDMI video.

25.3.1 HDMI EDID Tables

Extended display identification data (EDID) is a data structure used to describe the capabilities of a display (that is, sync device) to a video source with the VESA defined standard format. An EDID table loaded in an HDMI sync device communicates the sync device's supported resolutions, timing information, manufacturer ID, serial number, etc., to the HDMI source device(s). GMSL2 serializers can store EDID data and share it with the source device ([Figure 90](#)).

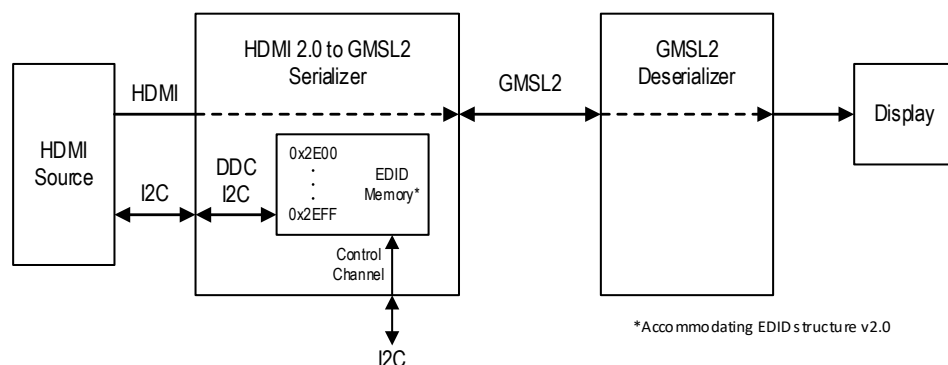


Figure 90. EDID Memory in the Serializer HDMI Rx Block

A typical application of EDID tables is to ensure functional compatibility of computer monitors and graphic cards. The EDID table in the computer monitor is read by the computer's graphics card so that the necessary video characteristics are generated and output on the HDMI port.

25.3.1.1 EDID Table Usage Requirement

An EDID table may be required depending on the HDMI source used in the system. For consumer port applications or when using a computer as a video source, an EDID table must be loaded to ensure proper operation. For embedded applications, however, an EDID table is typically not required as the video resolution is usually hard coded in the SoC and EDID tables are ignored.

Note: For consumer port applications or when using a computer as a video source, the HDMI source device must first see a valid EDID table in the HDMI sync device (that is, the HDMI serializer) for it to recognize the presence of an HDMI sync device before it outputs video.

25.3.1.2 Sync Device EDID Memory

The GMSL2 HDMI serializer acts as the sync device and functionally represents the display to the HDMI source. The GMSL2 HDMI serializer provides 256 registers for EDID storage (accommodating EDID structure v2.0). The EDID data is written to volatile memory space in the serializer through the device's primary I²C or UART interface. The EDID register block is 0x2E00–0x2EFF.

Note: EDID data must be written to the serializer whenever the device is power cycled.

25.3.1.3 Source EDID Access

The HDMI source must access the EDID data through the display data channel (DDC) at the HDMI interface. The DDC is an I²C interface and is separate from the serializer's I²C bus.

25.3.1.4 EDID Registers

The EDID table memory space is located in registers 0x2E00–0x2EFF. When the HDMI source device in the HDMI link detects hot plug detect (HPD) high, it reads the EDID table through the DDC_SDA and DDC_SCL lines of the HDMI interface.

25.3.1.5 EDID Configuration

If an EDID table is needed, the typical configuration steps are:

1. Write register to drive HPD pin low (register 0x20F5 set to 0x00).
2. Write EDID table (registers 0x2E00–0x2EFF).
3. Write register to drive HPD pin high (register 0x20F5 set to 0x01).

By toggling the HPD pin, the source recognizes that there is a new EDID table to read. If HPD is not cycled low then high, the source device may not re-read the new EDID table.

Note: HDMI source devices do not use a universal method of polling the status of the HPD pin. Some HDMI source devices detect an edge change on the pin and immediately react; some devices poll the pin level at a set time interval. For certain devices, this polling interval can be up to multiple seconds. Ensure the delay between steps 1 and 3 (above) is sufficient for the target source device to recognize the HPD level change.

25.3.2 HDMI Mode (HDMI 1.4 or HDMI 2.0)

HDMI serializers receive video input from an HDMI source through the transition-minimized differential signaling (TMDS) interface. TMDS is an 8b/10b encoding scheme with an interface consisting of a differential set of clock signals and three sets of differential data channel signals.

In HDMI 1.4 mode, the three sets of data channels operate at 10x of the pixel clock frequency (up to 3.4Gbps). The TMDS clock signal pair has the same frequency as the pixel clock. There is no phase requirement between the clock signals and data channels. In HDMI 2.0 mode, the frequency of the TMDS clock signal pair is divided by four. The data channels' rate is 40x the TMDS clock rate. Data scrambling is required to reduce electromagnetic interference/radio frequency interference (EMI/RFI).

- HDMI 1.4 mode (default) is used for PCLK values under 340MHz.
- HDMI 2.0 mode is used for PCLK values between 340MHz and 600MHz.

25.3.2.1 HDMI 1.4 Mode Configuration

By default, the HDMI Rx block is configured for HDMI 1.4; this mode **MUST** be used if the source PCLK is 340MHz or below. No configuration is needed to enable the HDMI input block in HDMI 1.4 mode.

25.3.2.2 HDMI 2.0 Mode Configuration

HDMI source devices utilize HDMI 2.0 to transmit videos with a PCLK value above 340MHz. HDMI serializers **MUST** be configured to HDMI 2.0 mode to receive this input. Enable HDMI 2.0 mode using the following register writes:

```
# Enable HDMI 2.0 Mode
# Hold HDMI receiver digital logic in reset
0x80,0x2005,0x01
# Enable all 3 TMDS data channels
0x80,0x3D0C,0x20
0x80,0x3E8D,0x40
# HDMI2 overwrite, scdt on
0x80,0x3E50,0x23
# 5MHz OSC for Zone detect reset
0x80,0x3C04,0x04
# Release reset of 5MHz OSC
```

```

0x80,0x3C04,0x00
0x80,0x230E,0x24
0x80,0x230F,0x00
0x80,0x233B,0x80
0x80,0x2301,0x00
# Scramble-on overwrite, HDMI2-on overwrite
0x80,0x2040,0x33
0x80,0x2313,0x20
# Release HDMI receiver digital reset
0x80,0x2005,0x00

```

25.3.2.3 Status and Control Data Channel (SCDC) for TMDS Configuration

In accordance with the HDMI 2.0 specification, the serializer contains the SCDC register for TMDS configuration. For HDMI 2.0 consumer port applications, the EDID programmed into the serializer should contain the HDMI-forum vendor-specific data block (HF-VSDB) **SCDC_Present** bit set high to indicate that it implemented the SCDC.

Note: The EDID must be correctly programmed for HDMI 2.0 operation. The HF-VSDB block in the EDID indicates that the sync device supports HDMI 2.0 features (HDMI 2.0 specification §10.3.2). If an HDMI source does not find this block in the HDMI sync device, the source assumes it is connected to an HDMI 1.4 device.

25.3.3 HDMI Receiver Optimization Configuration

For consumer port applications, the following configuration should be used to optimize the HDMI receiver equalizer for jitter tolerance performance needed to pass HDMI compliance tests. The receiver optimization configuration is needed for HDMI 1.4 mode; no optimization is required for HDMI 2.0 mode.

25.3.3.1 HDMI 1.4 Receiver Optimization

```

# Script to set up HDMI 1.4 RX EQ Optimizations
0x80,0x3D0A,0x1F
0x80,0x3E5C,0x0D
# CTS EQ Lookup table - PEQ_VAL0-7
0x80,0x3E08,0x18
0x80,0x3E09,0x38
0x80,0x3E0A,0x58
0x80,0x3E0B,0x78
0x80,0x3E0C,0x98
0x80,0x3E0D,0xB8
0x80,0x3E0E,0xD8
0x80,0x3E0F,0x78
# cdr bypass
0x80,0x3D0E,0x82
# bp_fix
0x80,0x3E15,0x40
# auto mode and auto engine decision enable
0x80,0x3E00,0x20

```

Note: This configuration is not compatible with HDMI 2.0 operation.

25.3.3.2 HDMI 2.0 Receiver Operation

For consumer port applications in HDMI 2.0 mode, no optimization is needed. For applications that switch between HDMI 1.4 mode and 2.0 mode, the following script should be used to disable the HDMI 1.4 receiver optimizations.

```
# Restore 2.0 EQ Optimizations with default values
0x80,0x3D0A,0x18
0x80,0x3E5C,0x1D
# Restore cdr_ctl1 default value
0x80,0x3D0E,0x02
# Revert bp_fix to default value
0x80,0x3E15,0x60
# Restore auto engine decision enable (reg_cfg_reg_edon) to default
0x80,0x3E00,0x00
```

25.3.3.3 Dual Mode Applications (HDMI 1.4 and HDMI 2.0 Modes)

For consumer port applications that support multiple video resolutions, additional steps are required to detect the HDMI mode sent by the video source. In compliance with the HDMI specification, the HDMI source uses the DDC channel to set registers in the HDMI receiver in the serializer. These registers are located in the serializer register block **DDC_SCDC_REG** (address **0xA801** to **0xA8FF**). For HDMI2.0 operation, the source should program **reg_scrbl_enable** (scramble enable) high and **reg_tmnds_bclk_ratio** (TMDS bit clock ratio) high in register **DDC_SCDC_REG** **TDMS_Config** (register **0xA820**). A change in value of the bit **reg_scbi_enable** can be configured to trigger an interrupt to the ERRB pin. The interrupt event can then be used to inform the application controller that the HDMI source has changed modes and can reconfigure the system (including the HDMI serializer) into the correct mode.

This interrupt sequence requires that the register **reg_intr7_stat6_aon** (register **0x2082**) be enabled and interrupt **reg_intr7_mask6_aon** (register **0x2092**) be enabled. In addition, the **HDMI_INT_OEN** (register **0x001A**) and **reg_rxdig_intr_mask** (**0x3C11**) registers must be enabled. See the [HDMI Video Status and Interrupt](#) section for additional details.

25.3.4 HDMI Pixel Repetition Mode

HDMI video sources use a pixel repetition (replication) scheme when the selected video format produces a pixel clock slower than 25MHz. This scheme duplicates pixels and increases the pixel clock frequency. GMSL2 HDMI serializers can be programmed to remove the replicated pixels in multiples of two, four, or eight.

25.3.4.1 Pixel Repetition Removal

GMSL2 HDMI serializers have a pixel repetition mode that can return the received video to the original video format and rate if the HDMI source device uses the pixel repetition scheme. See [Table 99](#) for pixel repetition mode details.

Table 99. Pixel Repetition Removal

Register Address	Bitfield	Description	Decode
0x2A15	Pixel_Rate[1:0]	Remove replicated pixels	0x0: Pixel Replication 1x 0x1: Pixel Replication 2x 0x2: Pixel Replication 4x

25.3.4.2 Pixel Repetition Removal Test

Pixel repetition removal can be tested by enabling an internal test mode and observing the pixel clock. To enable the internal test mode and verify pixel repetition mode, program the device according to the following procedure:

- Write the following sequence using low-level commands:


```
0x80, 0x003F, 0xA0
0x80, 0x003F, 0xC0
0x80, 0x003F, 0xCF
0x80, 0x003F, 0xA0
0x80, 0x003F, 0x06
0x80, 0x003E, 0xC0
```
- Observe the pixel clock on GPIO13 (SDOR/ADD0).
- Set the signal source to 1360 x 768 (60). The observed pixel clock at GPIO13 should be 85MHz.
 - If preferred, an alternate timing can be used. The expected clock from the specified timing should be observed at GPIO13.
- Change the signal source to 480i (PCLK = 13.5MHz). The measured pixel clock should be 27MHz due to pixel repetition at the source.
- Write register **0x2A15** to value 0x1 for pixel repetition of 2x. The measured pixel clock changes to 13.5MHz (the original PCLK rate) if pixel repetition mode is operating correctly.

25.4 HDMI Rx Errors and Debug

This section explains how to detect if a GMSL2 HDMI serializer is receiving video and how to read out the detected pixel clock and video timing parameters (that is, resolution and blanking).

25.4.1 HDMI Serializer Video Detection

GMSL2 HDMI serializers contain debug registers that can be used to detect if the device is receiving a valid input video signal ([Table 100](#)).

Table 100. GMSL2 HDMI Serializer Debug Registers

Register	Register Address	Bit name (bit)	Description
VIDEO_TX2	0x0102	PCLKDET (bit 7)	High if a stable PCLK is detected on HDMI input (Serializer PLL locked to PCLK)
VIDEO_TX2	0x0102	DRIFT_ERR (bit 6)	High if the PCLK has drifted significantly
VIDEO_TX2	0x0102	OVERFLOW (bit 5)	High if video transmit buffer in serializer has overflowed
VIDEO_TX2	0x0102	FIFO_WARN (bit 4)	High if the video FIFO is more than half full
VTX1	0x01C9	HDMI_CKDT (bit 6)	High if a HDMI clock is detected
VTX1	0x01C9	HDMI_SCDT (bit 7)	High if a HDMI sync is detected

25.4.2 Received Video Timing

GMSL2 HDMI serializers have a register block that is used to determine the received PCLK and video timing parameters. Each parameter is reported as a hexadecimal number output across two registers, LSBs in the first and MSBs in the second register. Values are reported less than two seconds after `HDMI_AUTOS = 1`.

The registers that report the video timing parameters are listed as follows.

- Horizontal active resolution = $\text{value}(0x2A8C) + (\text{value}(0x2A8D) \times 0x100)$
- Vertical active resolution = $\text{value}(0x2A8E) + (\text{value}(0x2A8F) \times 0x100)$
- HS low period = $\text{value}(0x2A90) + (\text{value}(0x2A91) \times 0x100)$
- HS high period = $\text{value}(0x2A92) + (\text{value}(0x2A93) \times 0x100)$
- Horizontal front porch = $\text{value}(0x2A94) + (\text{value}(0x2A95) \times 0x100)$
- Horizontal back porch = $\text{value}(0x2A96) + (\text{value}(0x2A97) \times 0x100)$
- VS low period = $\text{value}(0x2A98) + (\text{value}(0x2A99) \times 0x100)$
- VS high period = $\text{value}(0x2A9A) + (\text{value}(0x2A9B) \times 0x100)$
- Vertical front porch = $\text{value}(0x2A9C) + (\text{value}(0x2A9D) \times 0x100)$
- Vertical back porch = $\text{value}(0x2A9E) + (\text{value}(0x2A9F) \times 0x100)$

Note: 'value(0xXXXX)' refers to a read to address 0xXXXX in the HDMI serializer.

25.4.3 Pixel Clock Parameters

These parameters are reported by the PCLK PLL. This reported PCLK value is an estimate with the following accuracies:

- $\pm 10\%$ accurate between 50MHz to 300MHz.
- $\pm 5\%$ accurate above 300MHz.

Values reported in hexadecimal:

- ODIV = $\text{VID_ODIV_L}[7:0] + (\text{VID_ODIV_H} \times 0x100)$
- FBDIV = $\text{FB_DIV}[5:0]$
- FBDIVFRAC = $\text{FB_DIV_FRAC}[5:0]$

To calculate PCLK, convert hex values to decimal:

- $\text{PCLK (MHz)} = \frac{75}{\text{ODIV}} \times \left(\text{FBDIV} + \frac{\text{FBDIVFRAC}}{64} \right)$
- $\text{Total Pixels} = (\text{HS low period} + \text{HS high period}) \times (\text{VS low period} + \text{VS high period})$
- $\text{Frame rate} = \frac{\text{PCLK} \times 10^6}{\text{Total Pixels}}$

Note: The received PCLK and video timing parameters can also be shown in the GUI. With the HDMI serializer connected to the GUI, the relevant dashboard is accessed in 'Tools -> Configure Display Ser Des -> HDMI Timing'.

25.4.4 HDMI Video Status and Interrupts

HDMI event status registers and interrupts are used for a variety of purposes depending on device implementation and system application. This section describes and details the configuration of the status registers, interrupt registers, and masks.

25.4.4.1 Video Status Registers

Video status registers provide information about the presence of the pixel clock, HDMI clock, and sync character. These three detect signals are present in the **VTX1 (0x1C9)** register. These status bits are read-only and do not drive the ERRB pin. Since HDMI data is spread across the three HDMI *Transition-Minimized Differential Signaling* (TMDS) signals, valid sync character detection gives confidence in the validity of the incoming video stream.

Register **VTX1 (0x01C9)**:

- Bit 5: **PCLKDET** – PCLK Detect
- Bit 6: **HDMI_CKDT** – HDMI Clock Detect
- Bit 7: **HDMI_SCDT** – HDMI Sync Character Detect

25.4.4.2 HDMI Interrupts

HDMI serializers can detect certain HDMI events and create attendant interrupt signals. These interrupt signals, readable in status registers, can be further configured to output to the ERRB pin to create a hardware interrupt, as shown in *Figure 141*.

The following interrupts are available:

- **HDMI Sync Detect:** Indicates sync characters are stable and detected across all three HDMI TMDS lanes.
- **HDMI Clock Detect:** Indicates HDMI clock is detected.
- **HDMI Audio Sample Frequency Change:** Indicates change in audio sample frequency (Fs).
- **HDMI Cable Unplug:** Indicates HDMI HSPD pin has been disconnected from HDMI source.
- **HDMI *scramble_en* Change:** Indicates HDMI source change from HDMI 1.4 to HDMI 2.0 mode.
- **HDMI Cable Plug-in:** Indicates HDMI HSPD pin has been pulled high from HDMI source.

25.4.4.3 Interrupt Status

Each interrupt has a latched status register (*Table 101*). Write 0x1 to clear the bit.

Table 101. HDMI Rx Interrupt Status Registers

Register Address	Register Name	Bitfield	Decode
0x2080	RX_INTR2_AON	reg_intr2_stat3_aon	Bit 3: HDMI Sync Detect change Interrupt

0x2080	RX_INTR2_AON	reg_intr2_stat4_aon	Bit 4: TMDS Clock Detect change Interrupt
0x2344	RX_INTR5	reg_intr5_stat0	Bit 0: Audio sample frequency change event interrupt
0x2081	RX_INTR6_AON	reg_intr6_stat0_aon	Bit 0: HDMI Cable Unplug Interrupt
0x2082	RX_INTR7_AON	reg_intr7_stat6_aon	Bit 6: HDMI TMDS register scramble_en change event interrupt (indicates a change between HDMI 1.4 and HDMI 2.0 mode)
0x2083	RX_INTR8_AON	reg_intr8_stat1_aon	Bit 1: HDMI Cable Plug-in Interrupt

25.4.4.4 Interrupt Configuration

To enable an interrupt to be output to the ERRB pin:

- Set the interrupt enable bit given in [Table 170](#).
- Enable HDMI receiver "Rx digital interrupt" by setting `reg_rxdig_intr_mask` to 1 (0x3C11).
- Set `HDMI_INT_OEN` (0x001A) to 1.

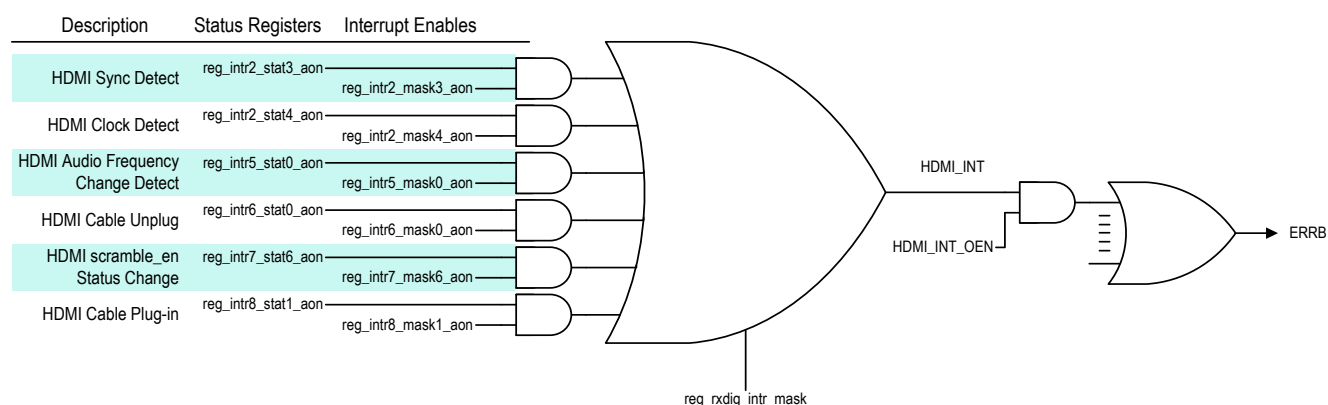


Figure 91. HDMI Fault/Error Register Architecture

Table 102. GMSL2 HDMI Rx Interrupt Enable

Register Address	Register Name	Bitfield	Decode
0x2090	RX_INTR2_MASK_AON	reg_intr2_mask3_aon	Bit 3: HDMI Sync Detect change Interrupt Enable
0x2090	RX_INTR2_MASK_AON	reg_intr2_mask4_aon	Bit 4: TMDS Clock Detect change Interrupt Enable
0x2354	RX_INTR5_MASK	reg_intr5_mask0	Bit 0: Enable audio sample frequency change event interrupt
0x2091	RX_INTR6_MASK_AON	reg_intr6_mask0_aon	Bit 0: HDMI Cable Unplug Interrupt Enable
0x2092	TX_INTR7_MASK_AON	reg_intr7_mask6_aon	Bit 6: Enable Interrupt for scramble_en change event (indicates a change between HDMI 1.4 and HDMI 2.0 mode)

0x2093	RX_INTR8_MASK_AON	reg_intr8_mask1_aon	Bit 1: HDMI Cable Plug-in Interrupt Enable
--------	-------------------	---------------------	--

The GMSL2 HDMI serializer's receiver error bit, which is one of several sources of the ERRB interrupt (see the [GMSL2 Error Reporting \(ERRB Pin\)](#) section), is status bit **HDMI_INT** (register **INTR3**, address **0x001B**). To clear the **HDMI_INT** status, clear the interrupts enabled in registers **0x2080**, **0x2081**, **0x2083** by writing each status bitfield high ([Table 169](#)). See [Table 103](#) for the HDMI serializer interrupt enable registers.

Table 103. GMSL2 HDMI Serializer Interrupt Enable Registers

Register Address	Register Name	Bitfield	Decode
0x3C11	INTR_STAT0	reg_rxdig_intr_mask	Bit 7: HDMI Error Enable (direct HDMI Interrupts to serializer error handling)
0x001B	INTR3	HDMI_INT	Bit 4: HDMI_INT status
0x001A	INTR2	HDMI_INT_OEN	Bit 4: Enables reporting of HDMI interrupt (HMDI_INT) to the ERRB pin

26. MIPI D-PHY Deskew

26.1 Overview

CSI-2 devices integrate MIPI D-PHY v1.2 ports conformant with the specifications published by the MIPI Alliance. For these devices, serializers have D-PHY input ports, and deserializers have D-PHY output ports. Refer to the product-specific data sheet for more information.

MIPI D-PHY v1.2 supports individual lanes speeds up to 2.5Gbps per lane. To provide the most robust and reliable MIPI connection, deskew calibration is available to minimize the Tx-to-Rx MIPI clock-to-data skew. In a GMSL2 system, this can occur in two different locations: from the MIPI source device to the GMSL2 serializer and/or from the GMSL2 deserializer to the MIPI sink device. Links available for MIPI D-PHY Deskew are indicated in [Figure 92](#). (that is, 'MIPI Link 1' and 'MIPI Link 2').

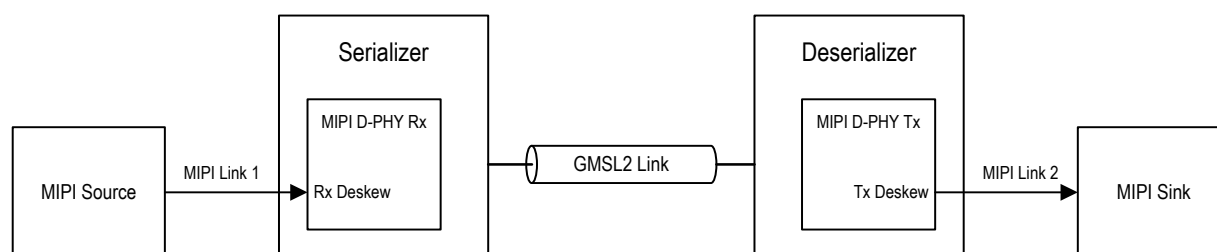


Figure 92. MIPI D-PHY Deskew Within a GMSL2 System

Note: The combination of MIPI Tx pins to the MIPI Rx pins must be within the MIPI deskew guidelines for the deskew procedure to be valid. GMSL2 D-PHY devices only support continuous clock mode.

Deskew calibration is intended for D-PHY lane speeds in excess of 1.5Gbps.

The MIPI D-PHY receiver in the serializers supports the deskew calibration. When it is enabled, the D-PHY receiver detects a special deskew burst from the MIPI transmitter. The D-PHY receiver then uses a deskew pattern to internally align the clock and data lanes. This alignment process increases clock-to-data skew tolerance and reduces data errors. The MIPI D-PHY transmitter in the deserializers sends out the special deskew packets if deskew calibration is required.

GMSL2 MIPI D-PHY devices, in conformance with the D-PHY v1.2 specification, support both the mandatory deskew calibration upon initialization and the optional periodic deskew calibration. The transmission of the deskew calibration sequence is not required for D-PHY lane speeds below 1.5Gbps; periodic deskew is optional for all lane speeds.

Note: GMSL2 MIPI D-PHY v1.2 input and output ports are conformant with published MIPI Alliance specifications. Refer to the appropriate documentation for more information regarding the D-PHY v1.2 specification.

26.2 Operation

GMSL D-PHY serializers support the Rx portion of the deskew calibration(s); D-PHY deserializers support the Tx portion. Deskew calibration is disabled by default. When operating the MIPI D-PHY above 1.5Gbps, deskew calibration must be enabled through the enable register bit.

Both the initial and periodic deskew calibrations are enabled through register writes. Note that these deskew procedures differ in length of the calibration sequence.

In the serializer, Rx deskew is enabled after configuring the data rate, lane selection, and D-PHY lane map but prior to receiving video from the MIPI source. Once enabled, the initialization calibration sequence is automatically generated when the MIPI D-PHY receiver in the serializer detects the sync pattern from the source. The deskew calibration function has successfully aligned the clock and data lanes when the clock-to-data skew is less than $\pm 0.4UI$.

Note: The first packet received by the MIPI D-PHY receiver must be the deskew packet. Once the receiver detects the deskew sync pattern, the calibration sequence begins.

The Tx deskew registers must be set prior to configuring the CSI-2 PLL. After CSI-2 PLL lock and video lock are established, the MIPI Tx clock lane starts and is automatically followed by the initial deskew pattern. The Tx deskew burst uses a sync pattern comprising a series of ones for a duration of 16 UI. After the sync pattern is sent, a clock pattern payload is transmitted. This pattern comprises alternating zeroes and ones (that is, 0101010101...) for a minimum duration of 2^{15} UI for the initial deskew calibration and 2^{10} UI for the periodic calibration(s).

Figure 93 depicts a high-speed MIPI data transmission in normal mode. *Figure 94* shows the deskew calibration process (see *Figure 95* for a detailed view).

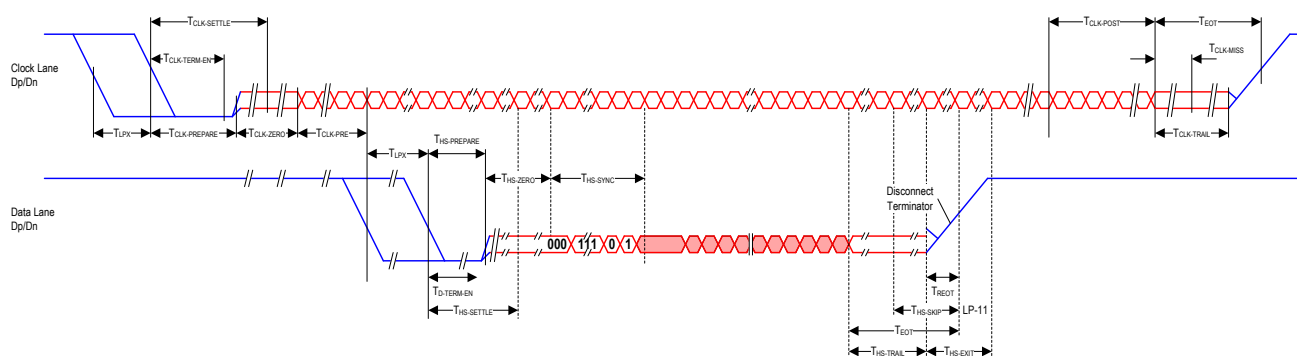


Figure 93. High-Speed Data Transmission in Normal Mode

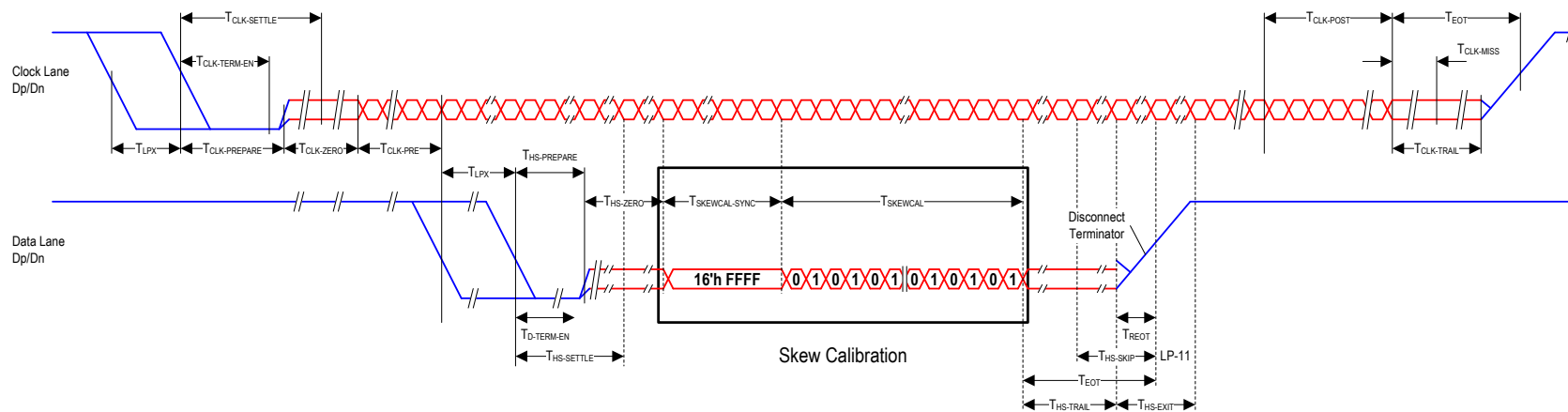


Figure 94. High-Speed Deskew Calibration

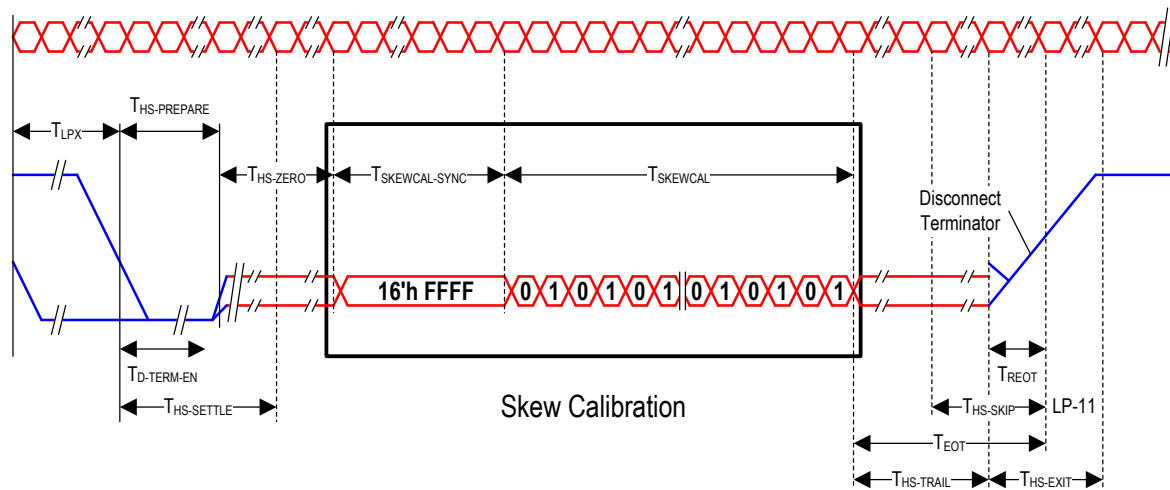


Figure 95. Skew Calibration (Detailed View)

26.2.1 Modes of Operation

GMSL2 MIPI D-PHY deserializers have two modes of MIPI deskew packet generation: Initial Deskew and Periodic Deskew.

Initial Deskew

- **Auto Mode:** Initial deskew is automatically generated upon MIPI power-up when enabled.
- **Manual Mode:** Initial deskew is inserted once between MIPI packets through register control.

Periodic Deskew

- **Auto Mode:** The periodic deskew is automatically generated. The occurrence and length of the deskew are programmable.

The deskew modes differ by length of training sequence. The initial deskew training sequence is longer than that of the periodic deskew. The initial deskew sequence is longer because the receive needs sufficient time to calibrate the MIPI clock frequency. Periodic deskew is used to fine-tune the skew to compensate for operational variations in supply and temperature.

Note: GMSL2 MIPI D-PHY serializers are compatible with both initial and periodic deskew.

In the serializer, the Rx deskew is enabled after configuring the data rate, lane selection, and D-PHY lane map but prior to receiving video from the MIPI source. If the deskew is enabled, the first packet received by the MIPI receiver must be the initial deskew packet. In the deserializer, the Tx deskew registers must be set prior to configuring the CSI-2 PLL.

Note: The Rx deskew is calibrated to the current MIPI clock frequency. Therefore, the clock frequency must be stable before the deskew packet is sent out. The MIPI clock frequency must remain stable after the calibration.

26.3 Configuration

PHY and controller availability is package-dependent; refer to the device-specific data sheet(s) for more information.

26.3.1 D-PHY Serializers

Deskew calibration for GMSL2 serializers with D-PHY is enabled through the [MIPI_RX1](#) register. Each controller has a dedicated enable bit and is used for both initial and periodic deskew calibration:

- Controller 0: [MIPI_RX1\[2\]](#)
- Controller 1: [MIPI_RX1\[6\]](#)

The initial Rx deskew procedure must be completed before receiving MIPI packets. For periodic deskew, the Rx deskew enable be configured before the start of MIPI operations (including clock and data lanes). Deskew enable cannot be dynamically changed and must remain active after the initialization deskew procedure.

Each MIPI PHY has a dedicated deskew calibration status register. There are two status bits for each lane: one bit indicates whether the calibration pattern was received, and the other bit indicating if the calibration was successful. MIPI PHY lane configuration must be considered when monitoring the status registers.

Deskew calibration status registers:

- PHY0: [MIPI_RX10](#)
- PHY1: [MIPI_RX12](#)
- PHY2: [MIPI_RX14](#)
- PHY3: [MIPI_RX16](#)

Table 105. D-PHY Serializers Deskew Calibration Configuration Registers

REGISTER	BITFIELD	POR	DESCRIPTION
MIPI_RX1	ctrl0_deskewen ctrl1_deskewen	0 0	Deskew calibration settings: Bit 6: ctrl1_deskewen for port B <ul style="list-style-type: none"> • 0: Deskew calibration disabled • 1: Deskew calibration enabled Bit 2: ctrl0_deskewen for port A <ul style="list-style-type: none"> • 0: Deskew calibration disabled • 1: Deskew calibration enabled
MIPI_RX10 MIPI_RX12 MIPI_RX14 MIPI_RX16	phy0_hs_err[4:7] phy1_hs_err[4:7] phy2_hs_err[4:7] phy3_hs_err[4:7]	0x00	Deskew calibration status: Bit 7: Deskew calibration pattern flag on data lane 0 <ul style="list-style-type: none"> • 0: Not received • 1: Received Bit 6: Deskew calibration pattern flag on data lane 1 <ul style="list-style-type: none"> • 0: Not received • 1: Received Bit 5: Deskew calibration failure flag on data lane 0 <ul style="list-style-type: none"> • 0: Default • 1: Failed Bit 4: Deskew calibration failure flag on data lane 1 <ul style="list-style-type: none"> • 0: Default • 1: Failed

26.3.2 D-PHY Deserializers

The initial deskew pattern for GMSL2 D-PHY deserializers is set with the [DESKEW_INIT](#) registers. Note that each controller is independently configured. These registers must be programmed before configuring the CSI-2 PLL settings (that is, the MIPI clock and data rates).

Automatic initial deskew is enabled by writing [DESKEW_INIT\[7\]](#). The MIPI Tx clock lane starts after video lock and CSI-2 PLL lock are established, followed by the transmission of an automatic initial deskew pattern. At any point after the clock lane is established, a one-time initial deskew pattern can be inserted. Manual initial deskew generation is configured with the [DESKEW_INIT\[5\]](#) register and is enabled with [DESKEW_INIT\[4\]](#).

The following lists the register addresses for each controller:

- Controller 0: [0x403](#) and [0x404](#)
- Controller 1: [0x443](#) and [0x444](#)
- Controller 2: [0x483](#) and [0x484](#)
- Controller 3: [0x4C3](#) and [0x4C4](#)

Periodic deskew is configured with the [DESKEW_PER](#) registers. The period of the Tx deskew calibration can be defined within the range of 1 to 128 frames. Periodic deskew can be enabled on either the rising or falling edge of VSYNC.

DESKEW_INIT[2:0] determines initial deskew width, that is, Tskewcal in the initial skew-calibration mode. We recommend set DESKEW_INIT[2:0] = 1 to guarantee it greater than minimum allowed value. DESKEW_PER[2:0] determines periodic deskew width, that is, Tskewcal in the periodic skew-calibration mode. We recommend set DESKEW_PER[2:0] = 1 to guarantee it greater than the minimum allowed value.

Note: Periodic deskew can be independently configured for any virtual channel in an enabled controller using the SKEW_PER_SEL[7:0] bitfield(s).

Table 106. D-PHY Deserializers Deskew Calibration Configuration Registers

REGISTER	BITFIELD	BITS	DESCRIPTION
MIPI_TX3	DESKEW_INIT[7:0]	7:0	Initial deskew pattern settings Bit 7: Auto initial deskew on/off Bit 6: Reserved Bit 5: when bit 4 = 1, any change of this bit triggers one-time immediate initial skew. Bit 4: Manual initial on/off Bit 3: Reserved Bits [2:0]: Select initial deskew width: 1, 2, 3, ... 8 * (32K) UI
MIPI_TX4	DESKEW_PER[7:0]	7:0	Periodic deskew pattern settings Bit 7: Period deskew calibration on/off Bit 6: Select generation on rising or falling edge of VS Bit [5:3]: Select periodic interval at every: 1, 2, 4, 8, ... 128 frames Bit [2:0]: Select periodic deskew width: 1, 2, 3, ... 8 * (1K) UI
MIPI_TX50	SKEW_PER_SEL[7:0]	7:0	Periodic deskew select register Bit 7: Select periodic deskew calibration for one or all virtual channels. <ul style="list-style-type: none"> 0: Generate periodic deskew on all VC. 1: on selected VC by bit 4:0 Bits [4:0] Virtual channel to generate periodic deskew calibration when Bit[7]=1

26.4 Debug Techniques

Note that external MIPI receivers use the calibration patterns generated by the GMSL2 D-PHY deserializers for calibration purposes. MIPI deskew debugging must consider all parts of the MIPI system.

26.4.1 GMSL2 D-PHY Serializers

If deskew status registers indicate that the deskew pattern is not received, ensure that the initial deskew pattern is sent before any data packets and that the deskew pattern transmitted by the MIPI source adheres to the MIPI D-PHY v1.4 specifications (that is, deskew sync pattern and timing requirements).

If the MIPI receive indicates that the deskew calibration has failed, measure the skew between the clock and data lanes. The calibration circuit cannot guarantee skew compensation if the skew is greater than $\pm 0.4\text{UI}$. Check the MIPI rate to ensure that it is within the operating range of the deskew function

(between 1.5Gbps and 2.5Gbps). Verify that the MIPI clock is stable and that any variation meets the MIPI specification. Verify that the initial deskew configuration procedure was followed and that an initial deskew pattern is transmitted before MIPI packets are sent.

26.4.2 GMSL2 D-PHY Deserializers

Repeat the process described above to continue debugging the MIPI system. Consult relevant documentation for the external MIPI receiver for further system debugging.

27. Dual oLDI

27.1 Overview

OpenLDI (oLDI) is a high-bandwidth digital-video interface that uses low-voltage differential signaling (LVDS) to transmit digital video without digital-to-analog conversion or complex decoding protocols. GMSL2 oLDI deserializers convert single-link or dual-link GMSL2 input to single or dual oLDI video output with flexible output configurations, a color lookup table (LUT) for gamma correction, and programmable spread-spectrum clocking (SSC). See [Feature Availability by Device Family](#).

oLDI output configurations:

- Single port (4 lanes or 8 lanes) or dual port (2 x 4 lanes)
 - 1 x 4, 2 x 4, or 1 x 8 oLDI output lane configurations
- Each port supports pixel clock rates up to 150MHz (300MHz in dual port mode). See the [Interface-Specific Bandwidth Calculations](#) sub-section for additional information.

27.2 Operation

GMSL2 oLDI deserializers receive GMSL data from the serial link and convert it to video data. The video data is then transmitted by the video pipes and mapped to ports A and/or B of the output interface. The output video interface has user-configurable port arrangements and can transmit single or dual oLDI video in either oLDI or VESA formats (see [LVDS Data Mapping](#)).

[Table 107](#) contains the oLDI output video interface pin numbers, names, and descriptions.

Table 107. oLDI Interface Pin Descriptions

Pin Number	Pin Name	Function
30	TxOUT_A3+	LVDS Data Output Port A
31	TxOUT_A3-	LVDS Data Output Port A
32	TxOUT_A2+	LVDS Data Output Port A
33	TxOUT_A2-	LVDS Data Output Port A
36	TxOUT_A1+	LVDS Data Output Port A
37	TxOUT_A1-	LVDS Data Output Port A
38	TxOUT_A0+	LVDS Data Output Port A
39	TxOUT_A0-	LVDS Data Output Port A
34	TxCLK_OUTA+	LVDS Clock Output Port A
35	TxCLK_OUTA-	LVDS Clock Output Port A
46	TxOUT_B3+	LVDS Data Output Port B
47	TxOUT_B3-	LVDS Data Output Port B
48	TxOUT_B2+	LVDS Data Output Port B
49	TxOUT_B2-	LVDS Data Output Port B
52	TxOUT_B1+	LVDS Data Output Port B
53	TxOUT_B1-	LVDS Data Output Port B
54	TxOUT_B0+	LVDS Data Output Port B
55	TxOUT_B0-	LVDS Data Output Port B
50	TxCLK_OUTB+	LVDS Clock Output Port B
51	TxCLK_OUTB-	LVDS Clock Output Port B

The GMSL2 oLDI deserializer generates the oLDI clock output from the tracked and measured clock frequency of the video input at the connected serializer. [Figure 96](#) shows the LVDS pulse positions with the oLDI clock.

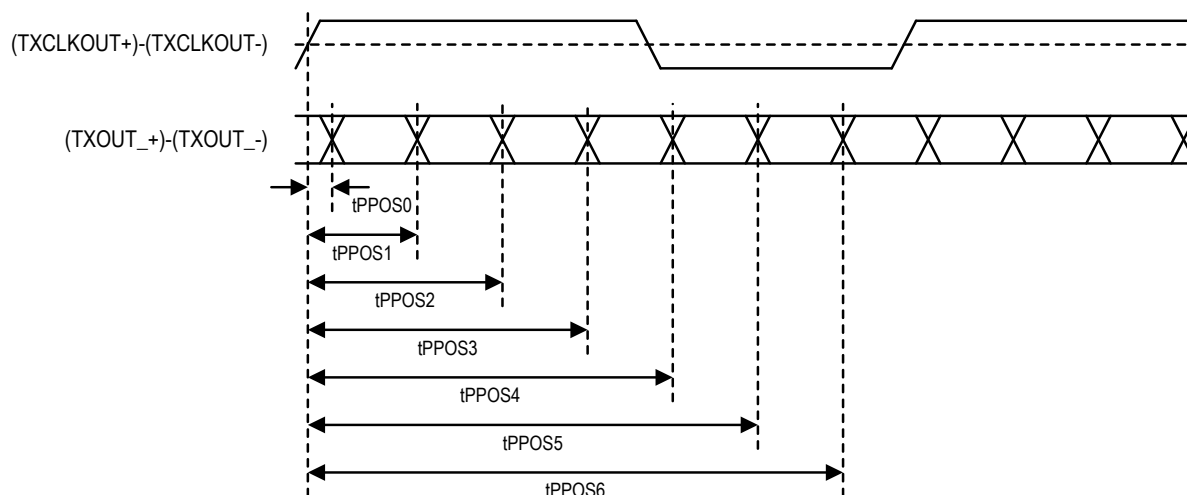


Figure 96. LVDS Pulse Positions

GMSL2 oLDI deserializers can be configured as single port (4 lanes or 8 lanes) or dual port (2 x 4 lanes) to accommodate a range of display applications. Each port supports pixel clock (PCLK) rates up to 150MHz and a combined PCLK of 300MHz in dual-port mode. Video data can be mapped to oLDI ports A, B, or both. Additionally, video lanes can be inverted and/or swapped to assist in compatibility with display interfaces. Both oLDI and VESA output formats are supported.

27.2.1 LVDS Data Mapping

The following sections provide the LVDS data mapping for single-port and dual-port oLDI output.

27.2.1.1 Single Pixel per Clock Output

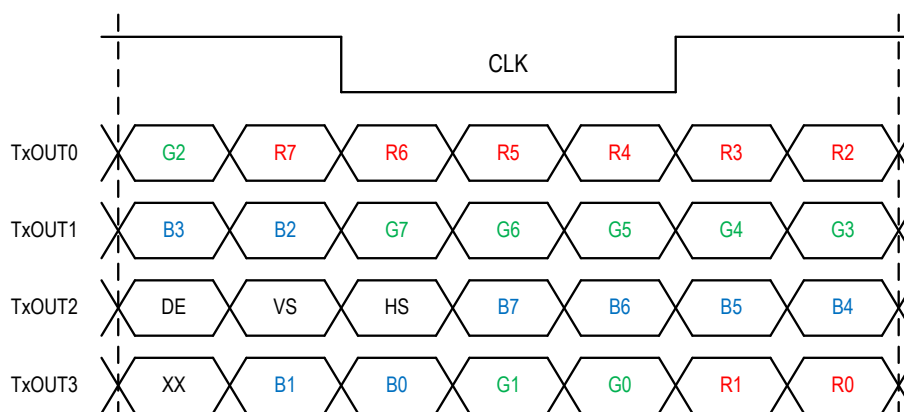


Figure 97. Single oLDI Port Mapping (OLDI/Format 1)

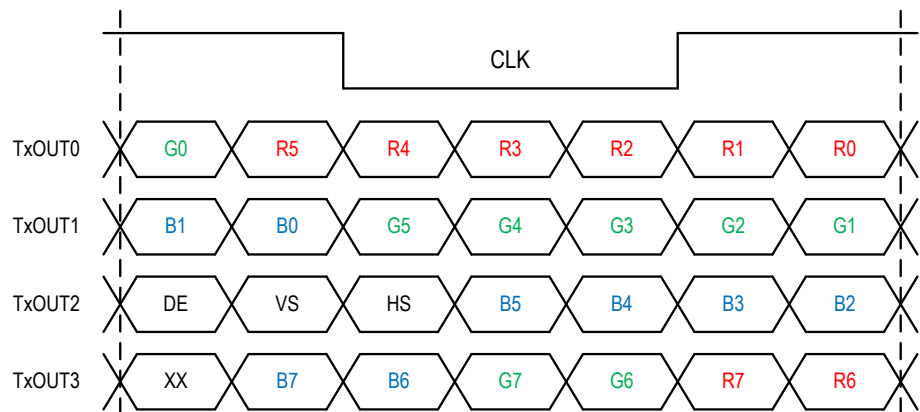


Figure 98. Single oLDI Port Mapping (VESA/Format 2)

27.2.1.2 Dual Pixel per Clock Output

The first pixel is mapped to TxOUTA.

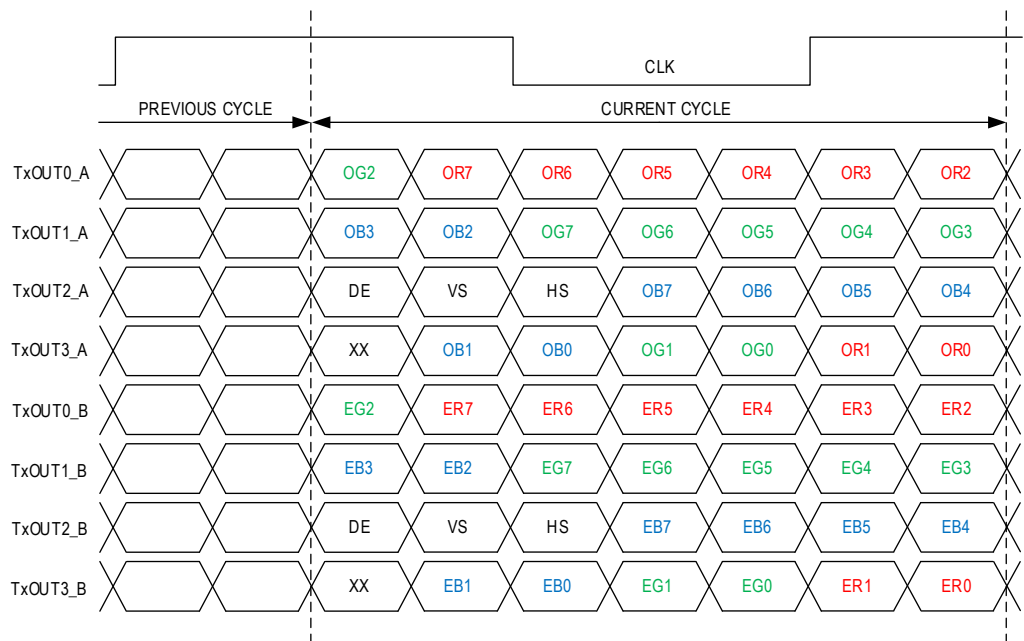


Figure 99. Dual oLDI Port Mapping (oLDI/Format 1)

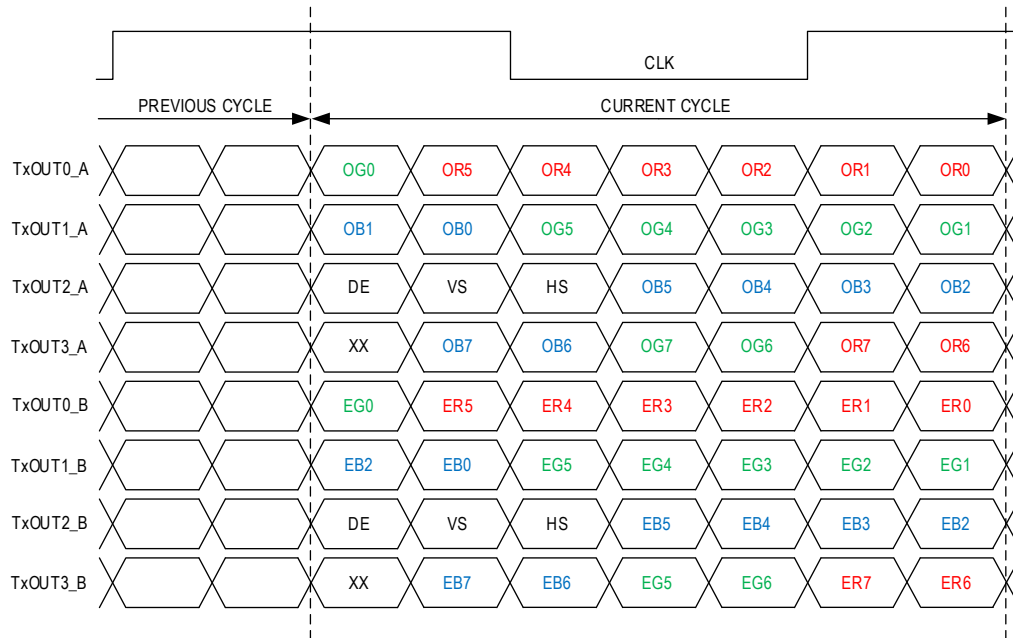


Figure 100. Dual oLDI Port Mapping (VESA/Format 2)

27.2.2 Output Programming

- Output ports are configurable as 1 x 4, 1 x 8, or 2 x 4 lanes. See the [Configuration](#) section for programming details related to this block.
- oLDI ports can be configured as 4-lane (for RGB888) or 3-lane (for RGB666) with the [OLDI_4TH_LANE](#) register bit.
- The oLDI output lanes can be inverted and/or swapped. The [LANE_SEL_*](#) registers select the output pin mapping for the port and lane designated by the register name; the [LANE_INV_*](#) registers invert the polarity of the port and lane designated by the register name.
- Vertical sync (VS) can be viewed on GPIO01. An optional single-cycle glitch filter is enabled with the [VS_OUT_EN](#) register bit.

27.2.3 Color Lookup Table (LUT)

The LUT enables 1:1 translation of 8-bit RGB color input data to any 8-bit RGB color output values. The feature is primarily intended for color filtering and gamma correction. The LUT consists of three discrete color channels that accept 8-bit wide entry for each color (that is, red, green, and blue) with 256-entry depth. See the [Color Lookup Table \(LUT\)](#) section for additional information and configuration details.

27.2.4 Spread-Spectrum Clocking

The oLDI output port can be programmed to have 0.5% center spread-spectrum clocking (SSC) with a 20kHz to 40kHz triangular modulation frequency. It is possible to program the spread-spectrum percentage up to 4% with the [config_spread_bit_ratio](#) bitfield (register [DPLL_3](#)).

Note: oLDI SSC mode programming is independent of the GMSL high-speed clock SSC mode. See the [Spread-Spectrum Clocking \(SSC\)](#) section for GMSL high-speed clock SSC information.

27.2.5 Odd/Even Pixel Interleaving in Dual oLDI Mode

The oLDI output processor can split the output of the two oLDI ports at half the PCLK frequency by using the active-high or active-low versions of the DE or HS signals for even/odd pixel alignment or no alignment.

In single-port mode, the maximum oLDI PCLK of 150MHz supports up to FHD (1920 x 1080 24-bit/60Hz) or equivalent. In dual-port mode, the interleaved even and odd pixels of the video stream are split and routed to the A and B output ports and transmitted to a single display. This process divides the video PCLK by a factor of two at the oLDI ports. For FHD video, the clocking on each port is approximately 75MHz and is below the FM radio band.

In dual-port mode, the GMSL2 oLDI deserializers support up to a 300MHz PCLK if both GMSL2 link inputs are used. This enables display of interleaved WQXGA (2560 x 1600 24-bit/60Hz) or equivalent video. If only one GMSL link is used, the maximum PCLK is limited to approximately 217MHz by the 6Gbps link bandwidth. This enables display of 2560 x 1080 24-bit/60Hz video or equivalent.

27.2.6 Video Replication and Dual-View Splitter Mode

oLDI deserializer replication enables identical FHD (or equivalent) video to be sent to two displays. Dual-view splitter mode supports column-interleaved FHD displays with different content split from a single video stream. Ports A and B can be swapped as needed. Note that side-by-side to column-interleaved conversion is processed on the serializer side; see the [Dual-View](#) section for additional information.

27.3 Application Examples

GMSL2 oLDI deserializers have a single video pipeline and support single video streams, video replication, and dual-view splitting.

The following application examples illustrated show a GMSL2 system with an HDMI source and HDMI serializer. These configurations are also possible with MIPI DSI and CSI-2 sources. See the appropriate serializer section for configuration details. GMSL2 oLDI deserializer programming is detailed in the [Configuration](#) section.

In [Figure 101](#), the serializer transmits a single video stream to the oLDI deserializer. The oLDI deserializer outputs to one connected display in either single or dual oLDI mode.

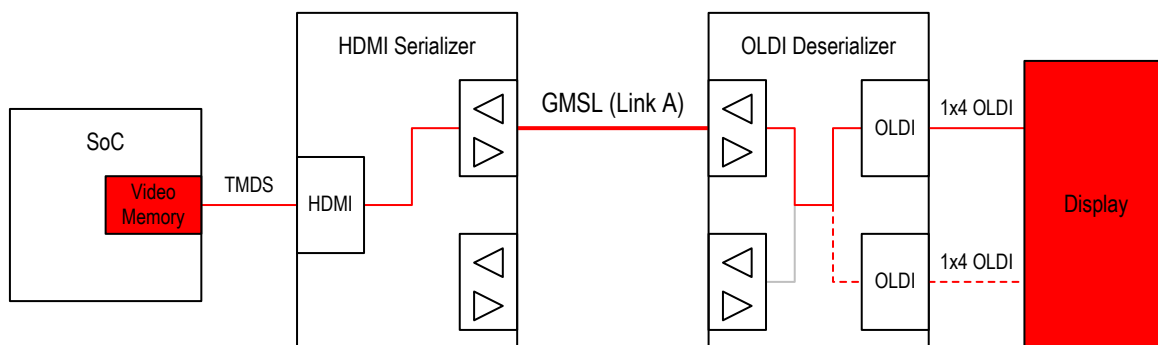


Figure 101. Serializer Single Link Mode with oLDI Deserializer Single or Dual Mode

In [Figure 102](#), the serializer in pixel-interleaved dual-view mode receives a superframe from the video source. The dual-view block splits the superframe and transmits the pixel-interleaved data on the serial link. The oLDI deserializer operates in splitter mode to split pixel-interleaved data into two video streams. The streams are transmitted from separate oLDI ports to two connected displays.

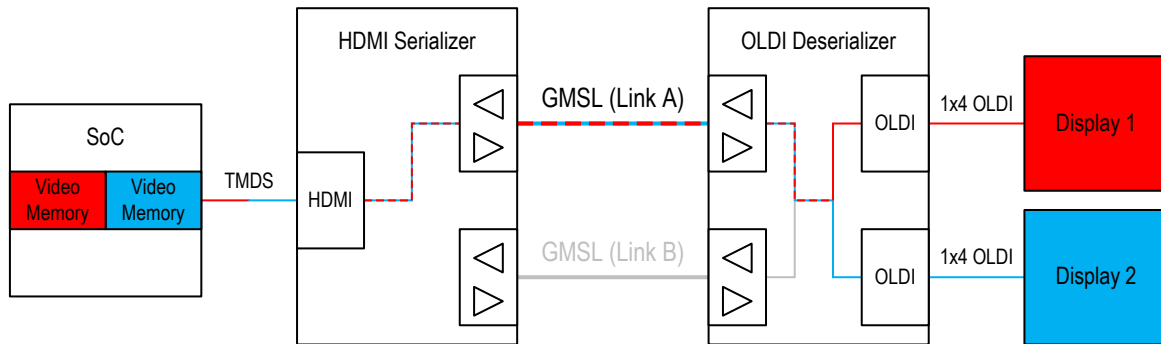


Figure 102. Serializer Dual-View Mode with oLDI Deserializer Splitter Mode

The GMSL2 system in [Figure 103](#) is configured for GMSL2 dual link mode. Video is transmitted from the serializer on dual GMSL2 links. The oLDI deserializer outputs to one connected display in either single or dual oLDI mode.

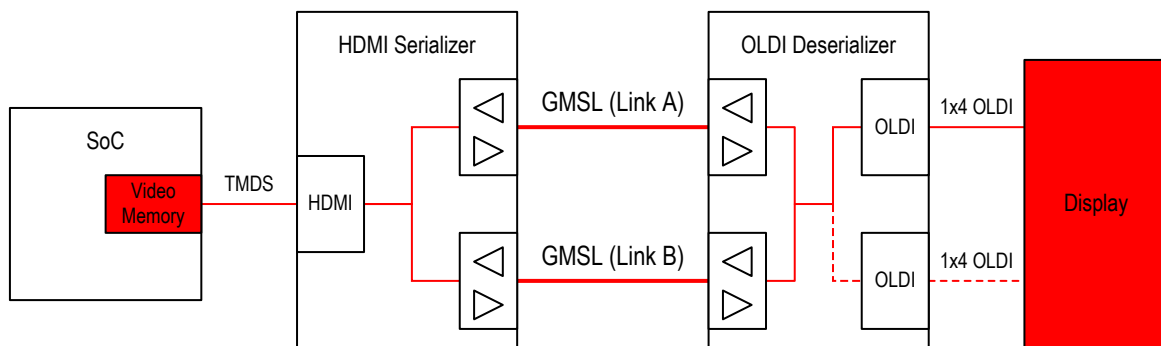


Figure 103. Single Video Stream and GMSL2 Dual Link Mode

The serializer in [Figure 104](#) transmits a replicated video stream in GMSL2 dual link mode to two connected oLDI deserializers. Each oLDI deserializer outputs to one connected display in either single or dual oLDI mode.

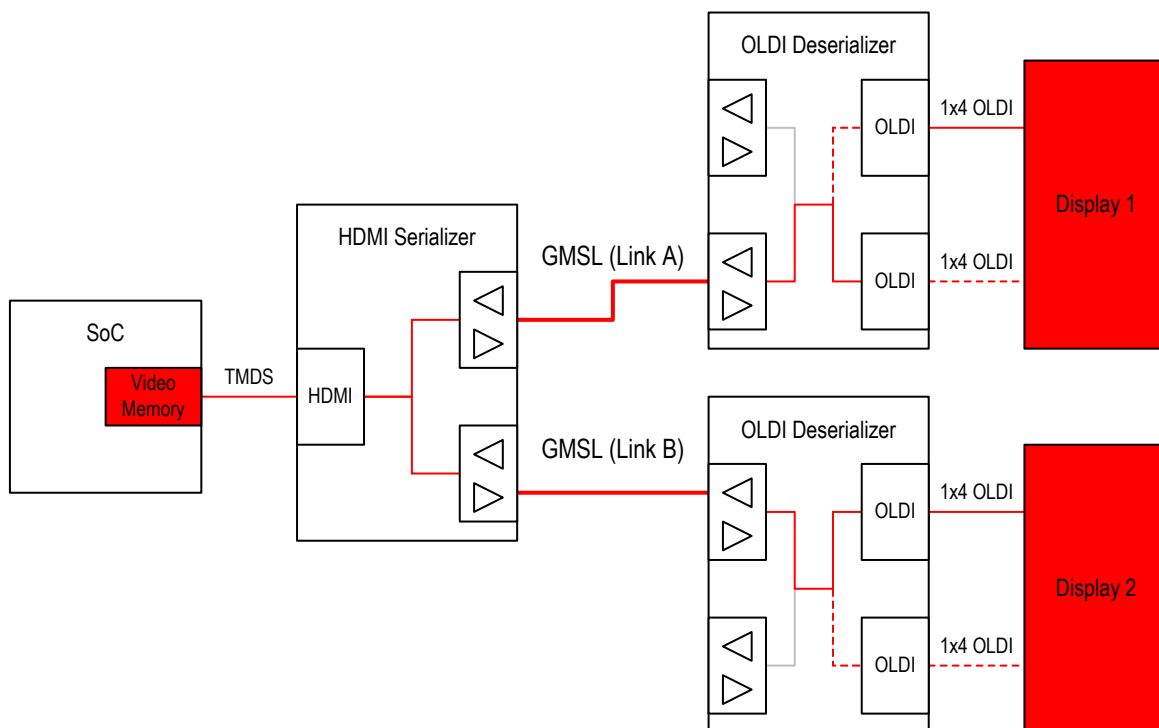


Figure 104. Single Video Stream with Serializer Replication

In [Figure 105](#), the serializer transmits one video stream to the oLDI deserializer. The deserializer replicates the video by sending identical streams to both oLDI ports, which are connected to separate displays.

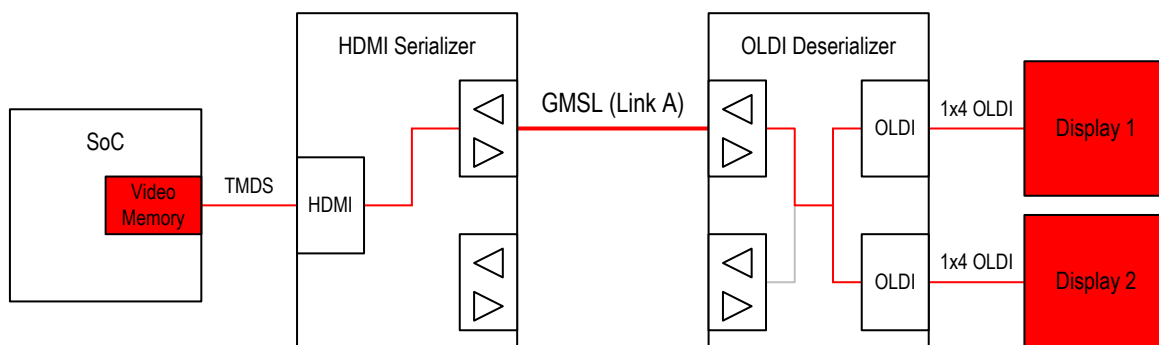


Figure 105. Single Video Stream with oLDI Deserializer Replication

27.4 Configuration

This section contains the registers relevant to oLDI output. oLDI port configuration registers are presented in [Table 108](#). See the [Programming Examples](#) section for use-case applications. Note that applications may require serializer configuration (see appropriate sections in this document for details).

For proper operation in oLDI splitter mode, the display timing parameters must have an even number of PCLK cycles before the oLDI split. If this is not applied, the synchronization between line starts of the two LVDS ports may become skewed.

Program the output format of the LVDS port with [OLDI_FORMAT](#) with the format of the connected display to ensure proper color mapping.

Note: All registers in [Table 184](#) (except [OLDI_SSEN](#)) must only be set when [VID_LOCK](#) = 0. The [OLDI_SPL_POL](#) bit in register [OLDI1 \(0x1CE\)](#) must always be set to 0 for dual oLDI applications.

Table 108. oLDI Port Configuration Registers

Register Address	Bitfield	Description	Decode
0x1CE	OLDI_OUTSEL	Selects which oLDI port to use for this data output.	0b0: Port A 0b1: Port B
0x1CE	OLDI_FORMAT	Output format of LVDS port.	0b0: oLDI 0b1: VESA
0x1CE	OLDI_4TH_LANE	oLDI lane count.	0: 4 lanes 1: 3 lanes
0x1CE	OLDI_SWAP_AB	Swaps oLDI ports A and B.	0b0: Ports A and B not swapped 0b1: Ports A and B swapped
0x1CE	OLDI_SPL_EN	oLDI Splitter enable.	0b0: Splitter disabled 0b1: Splitter enabled
0x1CE	OLDI_SPL_MODE	oLDI Splitter Mode.	0b00: Random 0b01: Split with HS 0b10: Split with VS 0b11: Split with DE
0x1CE	OLDI_SPL_POL	Set to 0 for dual oLDI applications.	0b0: Falling edge 0b1: Rising edge
0x1CF	PD_LVDS_B	Selects power-up/power-down status for LVDS output driver B.	0b0: LVDS output driver B powered up 0b1: LVDS output driver B powered down
0x1CF	PD_LVDS_A	Selects power-up/power-down status for LVDS output driver A.	0b0: LVDS output driver A powered up 0b1: LVDS output driver A powered down
0x1CF	OLDI_DUP	Selects whether or not to duplicate oLDI at both ports	0b0: oLDI not duplicated 0b1: oLDI duplicated at both ports
0x1CF	SSEN	Enables spread spectrum oLDI output clock. Spread percentage is adjusted from OLDIPLL registers (see the	0b0: Spread spectrum disabled 0b1: Spread spectrum enabled

		<i>Spread-Spectrum Clocking (SSC)</i> section). Default percentage is 0.5%.	
--	--	---	--

Table 109 contains registers used to output polarity inversion and output pin mapping. Polarity is inverted on a lane-by-lane basis. The pin mapping registers enable programming the pin assignments for the OLDI output ports and lanes.

Table 109. oLDI Output Polarity Inversion and Pin Mapping Registers

Register Address	Bitfield	Description	Decode
0x1D0	LANE_INV_B0	Lane B0 polarity inversion	0b0: Lane polarity not inverted 0b1: Lane polarity inverted
0x1D0	LANE_INV_A0	Lane A0 polarity inversion	0b0: Lane polarity not inverted 0b1: Lane polarity inverted
0x1D1	LANE_INV_B1	Lane B1 polarity inversion	0b0: Lane polarity not inverted 0b1: Lane polarity inverted
0x1D1	LANE_INV_A1	Lane A1 polarity inversion	0b0: Lane polarity not inverted 0b1: Lane polarity inverted
0x1D0	LANE_SEL_B0[2:0]	Selects oLDI Port B Lane 0 output pins	0b000: Output from B0 0b001: Output from B1 0b010: Output from B2 0b011: Output from B3 0b100: Output from BCLK 0b101: Reserved 0b110: Reserved 0b111: Reserved
0x1D0	LANE_SEL_A0[2:0]	Selects oLDI Port A Lane 0 output pins	0b000: Output from A0 0b001: Output from A1 0b010: Output from A2 0b011: Output from A3 0b100: Output from ACLK 0b101: Reserved 0b110: Reserved 0b111: Reserved
0x1D1	LANE_SEL_B1[2:0]	Selects oLDI Port B Lane 1 output pins	0b000: Output from B0 0b001: Output from B1 0b010: Output from B2 0b011: Output from B3 0b100: Output from BCLK 0b101: Reserved 0b110: Reserved 0b111: Reserved
0x1D1	LANE_SEL_A1[2:0]	Selects oLDI Port A Lane 1 output pins	0b000: Output from A0 0b001: Output from A1 0b010: Output from A2

			0b011: Output from A3 0b100: Output from ACLK 0b101: Reserved 0b110: Reserved 0b111: Reserved
--	--	--	---

PRBS registers are presented in [Table 110](#); see the [PRBS Testing](#) section for additional information.

Table 110. PRBS Registers

Register Address	Bitfield	Description	Decode
0x1CD	VPRBS_FAIL	Video PRSB pass/fail	0b0: Video PRBS check passed 0b1: Video PRBS check failed
0x1CD	VPRBS_CHK_EN	Enables video PRBS checker	0b0: Video PRBS checker disabled 0b1: Video PRBS checker enabled

[Table 111](#) contains registers used to enable the LUT; see the [Color Lookup Table \(LUT\)](#) section for additional information.

Table 111. LUT Registers

Register Address	Bitfield	Description	Decode
0x1CD	LUT_C_EN	Enables Color C lookup table (LUT) [23:16]	0b0: Color C LUT disabled 0b1: Color C LUT enabled
0x1CD	LUT_B_EN	Enables Color B lookup table (LUT) [15:8]	0b0: Color B LUT disabled 0b1: Color B LUT enabled
0x1CD	LUT_EN_A	Enables Color A lookup table (LUT) [7:0]	0b0: Color A LUT disabled 0b1: Color A LUT enabled

Sync signals can be viewed on GPIO01 if enabled with the registers in [Table 112](#). See the device-specific data sheet for pin-mapping details.

Table 112. GPIO Sync Signal Output

Register Address	Bitfield	Description	Decode
0x1CD	HS_OUT_EN	Output HSYNC from GPIO.	0b0: HSYNC not output 0b1: Output HSYNC from GPIO
0x1CF	VS_OUT_EN	Output VSYNC from GPIO.	0b0: VSYNC not output 0b1: Output VSYNC from GPIO

27.4.1 Programming Examples

27.4.1.1 Stream at Single oLDI Output

Perform the following writes in the oLDI deserializer:

1. `OLDI_OUTSEL` = Select LVDS output port (A or B) to put the LVDS video data.

2. **OLDI_FORMAT** = Select the mapping format (oLDI or VESA) for the LVDS port.
3. **OLDI_4TH_LANE** = Select 24-bit or 18-bit output (that is, 4-lane or 3-lane output port).
4. **OLDI_DUP** = Select if the same video should be duplicated on both LVDS ports.

27.4.1.2 Stream at Dual oLDI Output

Perform the following writes in the oLDI deserializer:

1. **OLDI_FORMAT** = Select the mapping format (oLDI or VESA) for the LVDS port.
2. **OLDI_4TH_LANE** = Select 24-bit or 18-bit output (that is, 4-lane or 3-lane output port).
3. **OLDI_SPL_MODE** = 3
4. **OLDI_SPL_EN** = 1
5. **OLDI_SWAP_AB** = Swap routing of even or odd pixels to ports A and B.
6. **OLDI_SPL_POL** = 0

27.4.1.3 Swap Port Lane Mappings and Invert Signal

Swap port A lanes 1 and 2, and invert lane 2 polarity. Perform the following writes in the oLDI deserializer:

1. **LANE_SEL_A1** = 2 (lane 1 pins output internal lane 2)
2. **LANE_SEL_A2** = 1 (lane 2 pins output internal lane 1)
3. **LANE_INV_A2** = 1 (invert polarity of lane 2 pins)

27.5 Status and Debug Registers

27.5.1 Status and Error Bits

The following register bits provide status and error information related to LVDS output.

- **VID_LOCK** (register **VIDEO_RX8**): Indicates that a valid video stream is received for the selected channel ID, the video pipeline is locked, and that the video stream is properly recovered by the LVDS to drive the display.
- **VID_PKT_DET** (register **VIDEO_RX8**): Indicates that a stream of video packet for the selected stream ID is detected by the deserializer.
- **DUAL_OLDI_AUTO_RST_ALIGNED** (register **VRX46**): This bitfield reports if the oLDI output has reset itself after detecting a non-alignment on the oLDI outputs. Only applicable if **DUAL_OLDI_AUTO_RST_ALIGN** = 1.
- **OUT_STUCK_ERR_FLAG** (register **INTR3**): Flag is asserted when the oLDI output is stuck either low or high. Enable with the **OUT_STUCK_ERR_OEN** bit. The flag is cleared when read if the output stuck event is resolved.
- **OUT_OPEN_ERR_FLAG** (register **INTR3**): Flag is asserted when the oLDI output is open. Enable with the **OUT_OPEN_ERR_OEN** bit. The flag is cleared when read if the output open even is resolved.

See the [GMSL2 Error Reporting \(ERRB Pin\)](#) section for additional information.

The following video receiver status registers and interrupts may also be used:

- Line CRC interrupt (**LCRC_ERR_FLAG**)
- Video block length error interrupt (**VID_BLK_LEN_ERR**)
- Video sequence number error (**VID_SEQ_ERR**)

27.5.2 Debug Procedure

If the oLDI deserializer is not performing as expected, perform the following sequence:

1. Ensure that minimum and maximum frequency limits are not exceeded.
2. Verify that a valid video stream is detected by the deserializer ([VID_PKT_DET](#)).
3. Verify that a valid video stream is received and recovered for the selected channel ID, and that it is properly recovered by the LVDS ([VID_LOCK](#)).
4. Ensure that the oLDI split control signal is properly mapped to the falling edge of the DE signal ([OLDI_SPL_MODE](#) and [OLDI_SPL_POL](#)).
5. Ensure that oLDI ports A and B are properly mapped ([OLDI_SWAP_AB](#)).
6. Set [DUAL_OLDI_AUTO_RST_ALIGN](#) = 1 if pixels are being swapped between oLDI Tx ports in Dual oLDI mode applications.
7. Check that the fourth LVDS lane is not disabled for 24-bit video ([OLDI_4TH_LANE](#)).

Revision History

REVISION NUMBER	REVISION DATE	DESCRIPTION
0	7/23	Initial release
1	12/23	<p>Added Sections:</p> <ul style="list-style-type: none">• Power Manager and Sleep Mode• PRBS Testing• Video Pipes• Dual-View• Watermarking• Video Timing Generator and Video Pattern Generator• Video Crossbar• Color Lookup Table (LUT)• Frame Sync• Video Sync Pulse Outputs• Audio• General-Purpose Input and Output (GPIO)• HDMI Serializers• Dual OLDI <p>Updated:</p> <ul style="list-style-type: none">• I²C/UART: Changed all mentions of Main CC/I²C/UART to Primary CC/I²C/UART to distinguish between Main/Subordinate terminology.