GMSL2 General User Guide

Rev 0; 7/23

GMSL2 General User Guide

List of Figures

List of Tables

Serial Link Features

1. GMSL2 Link Basics

1.1 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, bidirectional architecture with forward and reverse channels. The forward channel transfers data from the serializer to the deserializer; the *reverse channel* transfers data from the deserializer to the serializer. The 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.

1.2 GMSL2 Link Configurations

The GMSL2 architecture can support the following link configurations. R

1.2.1Single-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.

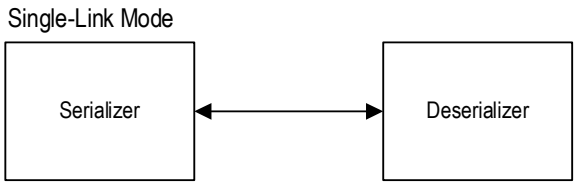


Figure 1. GMSL2 Single-Link Mode

1.2.2Reverse 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.

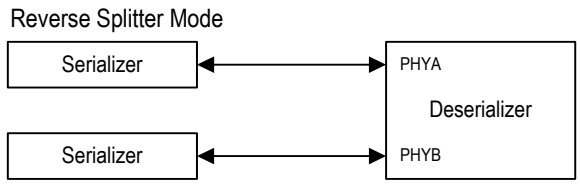


Figure 2. Reverse Splitter Mode

1.2.2.1 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.

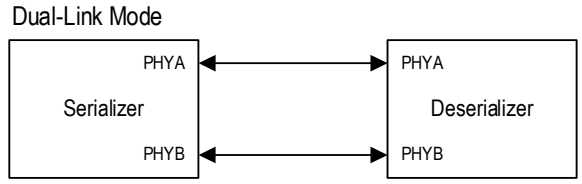


Figure 3. Dual-Link Mode

1.3 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*.

1.3.1Programming Link Rate

Table 1. GMSL2 Link Rate Configuration

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

1.3.2Programming Link Configuration

Link configuration is controlled with the AUTO_LINK and LINK_CFG[1:0] registers. After link configuration settings are changed, a RESET_ONESHOT command must be sent to either the serializer or deserializer for the link change to take effect. See the *Resets* section for more information.

1.3.2.1Auto 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.

1.3.2.2Manual 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)	AUTO_LINK = 0 LINK_CFG[1:0] = 01	AUTO_LINK = 0 LINK_CFG[1:0] = 01
Single-Link Mode (PHYB)	AUTO_LINK = 0 LINK_CFG[1:0] = 10	AUTO_LINK = 0 LINK_CFG[1:0] = 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)	AUTO_LINK = 0 LINK_CFG[1:0] = 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.

1.3.2.3 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
```

1.3.2.4 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

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.

1.3.2.5Standard 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.

1.3.2.6Cables Supported

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

1.4 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.

1.4.1Initial 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 that there are differences depending on whether the device is in GMSL2 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.

1.4.2Link Start-Up Procedure

1.4.2.1GMSL2 Mode

Following the *Initial Power-Up Sequence* section, the local-side control channel (I^2C 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 (i.e., 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 (i.e., 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).

1.4.3GMSL2 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.

1.4.3.1LOCK 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 (i.e., forward channel in serializer, reverse channel in deserializer) has locked to the correct word boundary alignment. Control channels (i.e., I^2C/UART, SPI, GPIO, and RGMII) 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.

1.4.3.2Losing Link Lock

Link lock is lost (i.e., 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 (i.e., “hotplugged”). 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.

1.4.3.3Video 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).

1.4.4Benefits 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).

1.4.5Periodic 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*).

1.4.5.1 Periodic Adaptation Performance Impact

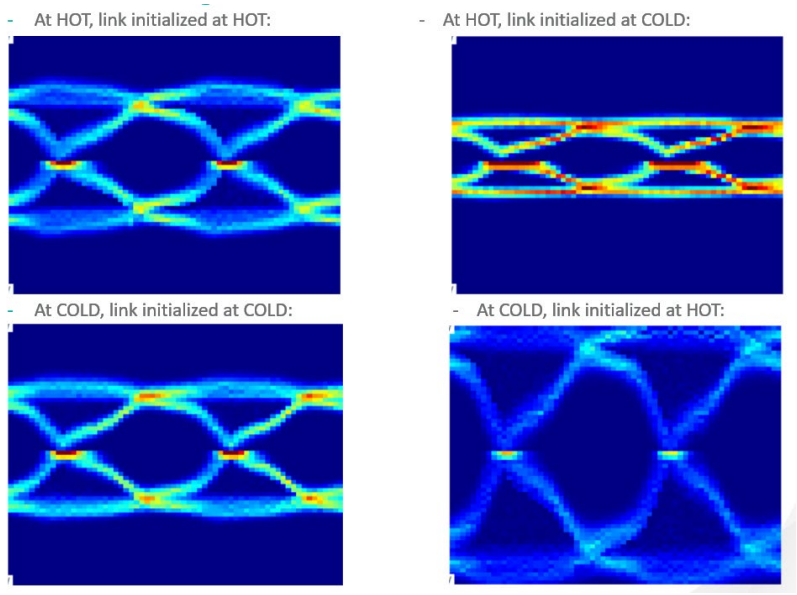


Figure 4. Periodic Adapt Not Enabled

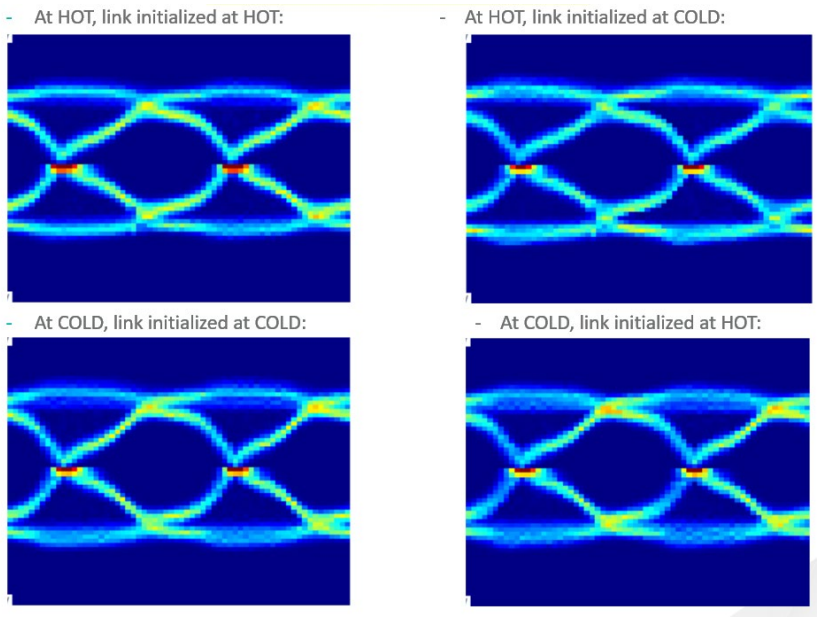


Figure 5. Periodic Adapt Enabled

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.

1.5 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
- GMSLGMSL2 mode

Note: There are other situations when a link reset may be needed which are described in their relevant sections.

1.5.1Types of Resets

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

1.5.1.1Reset 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.

1.5.1.2Oneshot 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, such as changing the link rate.

1.5.1.3Reset 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.

1.6 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.

1.6.1.1 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 (i.e., 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.

1.6.1.2 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.

1.6.1.3 Special Symbols

Special symbols (individually and in combination) are used as packet delimiters and headers. These are selected to keep GMSL2 link robust (i.e., 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- or 2-bit errors.

1.6.1.4 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.

1.6.2 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.

1.6.2.1 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 (HVD)] 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.

1.6.2.2 Control Channel Packets

Control packets are used for communications channels that use less than 100Mbps of bandwidth. This includes I^2C, UART, SPI, GPIO, audio, and internal info frame packets. Info frame packets are internal control packets used to transmit information between serializer and deserializer (e.g., 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.

1.6.2.3 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.

1.6.2.4 RGMII/RMII Packets

The RGMII/RMII packets are used to tunnel RGMII and RMII data. CRC is disabled by default for this packet type, as Ethernet has error-detection capability at higher protocol layer. However, CRC can optionally be enabled if bandwidth is available.

1.6.2.5 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.

1.6.3Priority-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 (i.e., 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.

2 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.

2.1 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*.

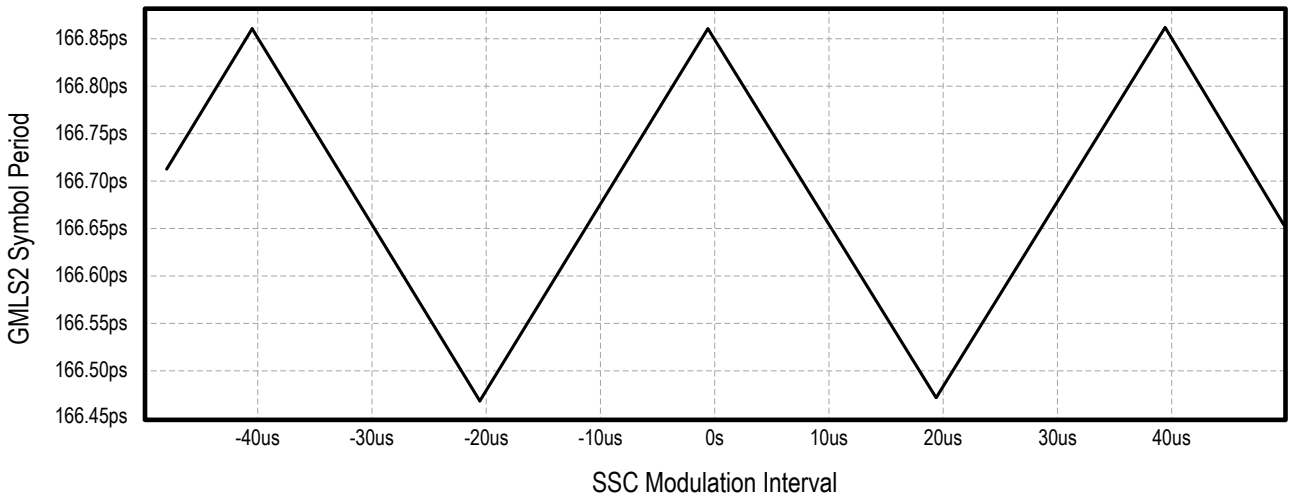


Figure 6. Example SSC Center Frequency Spread

2.2 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.

2.2.1 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.

3 Clocks

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 7*. The diagram shows both on-chip and external circuit elements.

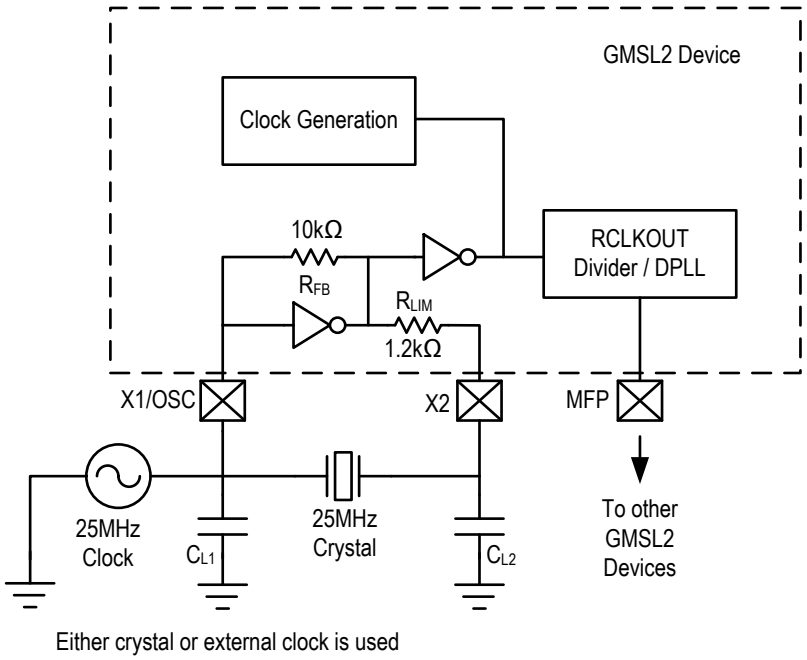


Figure 7. Simplified Diagram of Frequency Reference System

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.

3.3.1Crystal Mode

When deriving the frequency reference using a crystal, a device with the required frequency and tolerance (i.e., 25MHz ±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 7*. 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}	1.2kΩ
Internal Feedback Resistor	R_{FB}	10kΩ
Transconductance	g_M	28mA/V

Proper layout of the crystal circuitry is important to achieving the best possible performance. *Figure 8* 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.

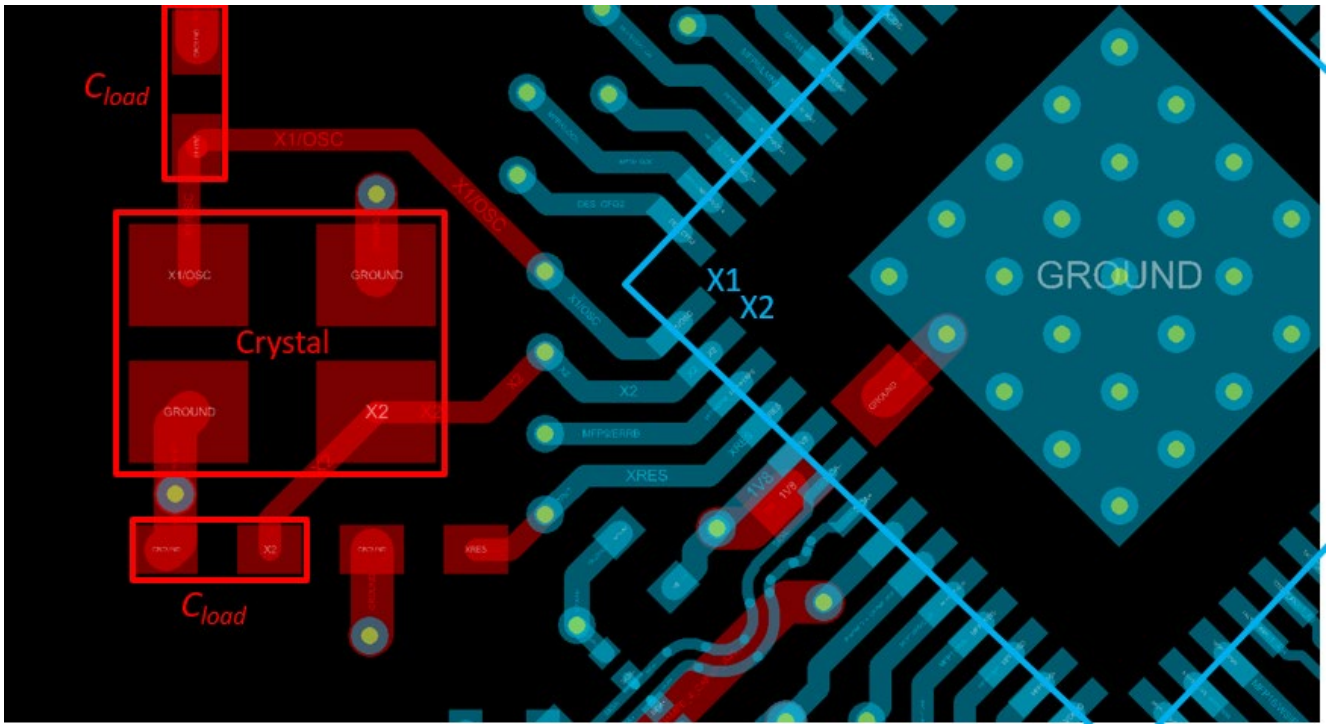


Figure 8. Example Crystal Layout

3.3.2External 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 (i.e., 25MHz ±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		10kΩ	
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 = 10pF			4ns
Fall Time	80% to 20%, C_L = 10pF			4ns
Jitter	C_L = 10pF, rising or falling edge f_{REFOUT} = 12.5MHz		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.

3.3.3Frequency 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

3.4.1BER 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

Video Transmission

4 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.

4.1 Video Frame Architecture

The different regions of a video image are identified in *Figure 9*. 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.

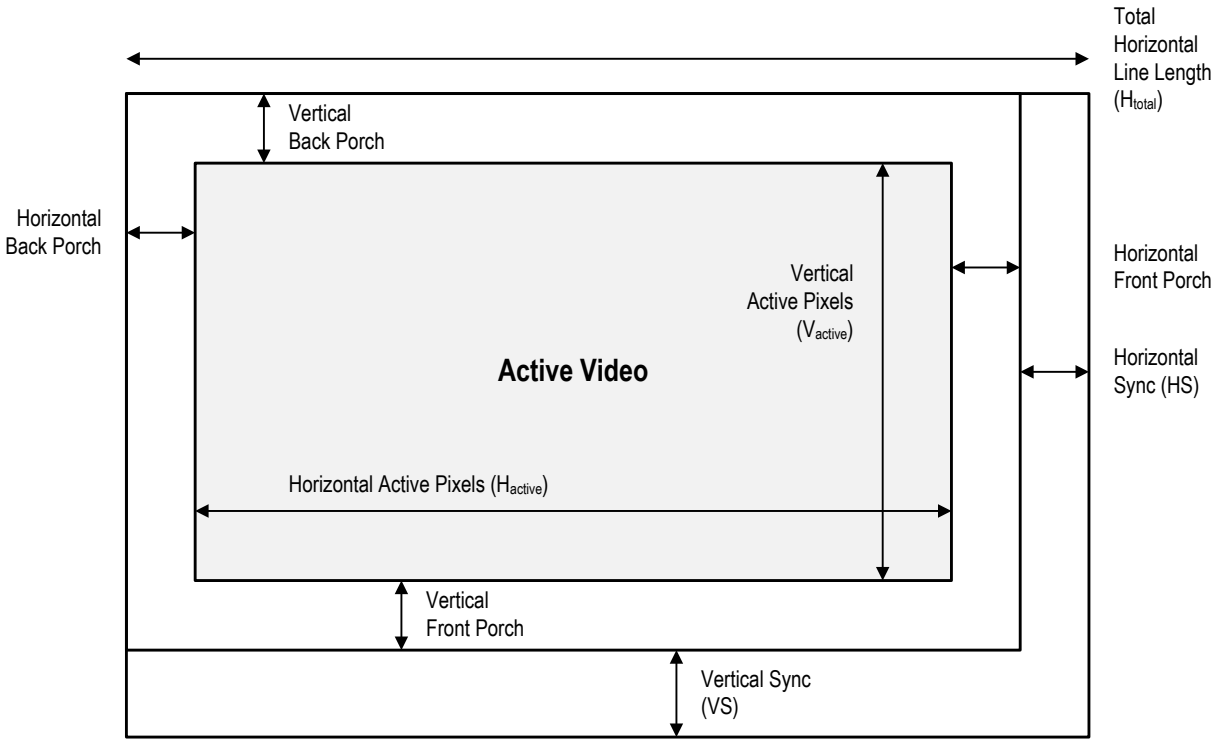


Figure 9. Video Frame Regions

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 10*).

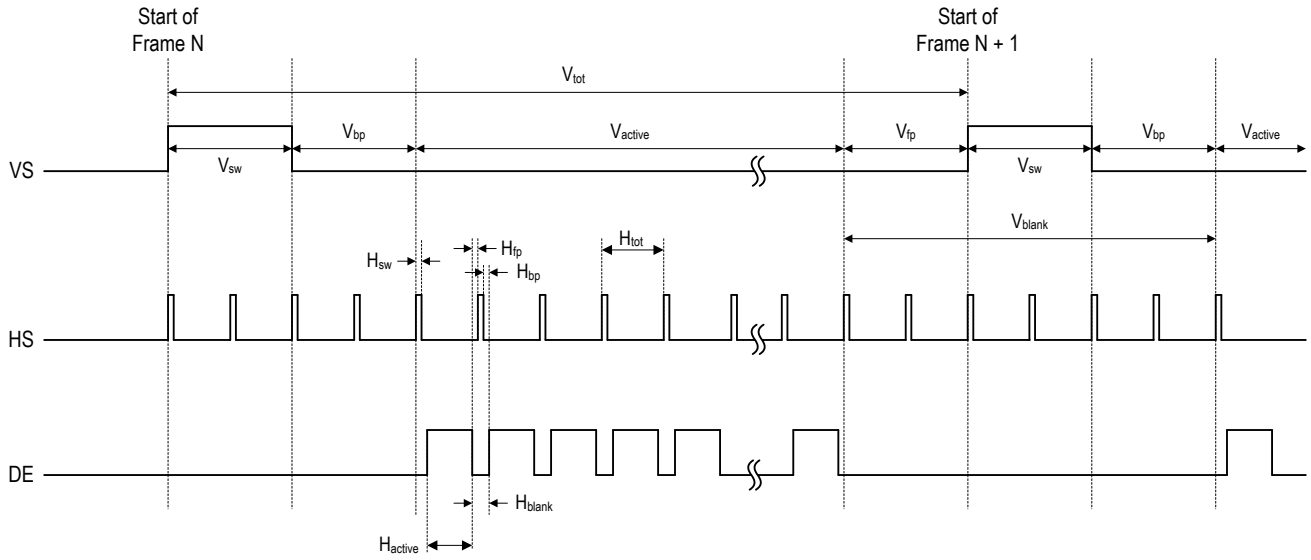


Figure 10. Sync Pulses: Data Enable and the Vertical and Horizontal Sync Signals

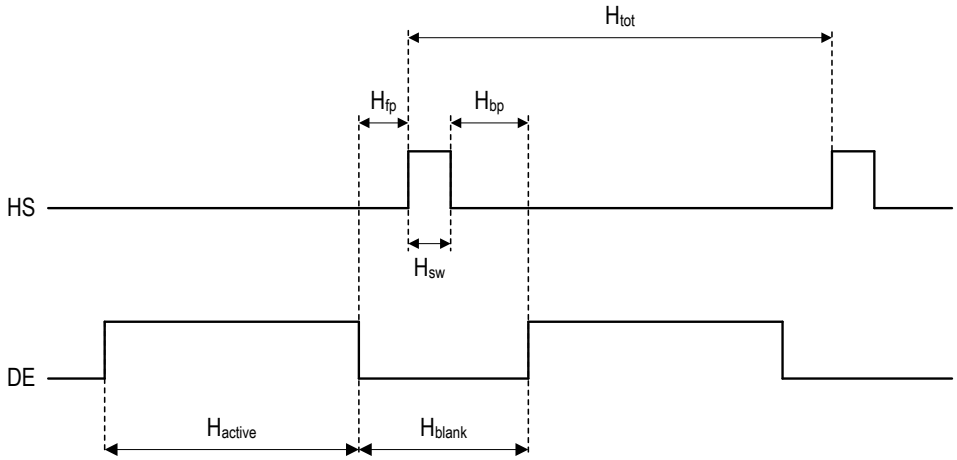


Figure 11. Horizontal Sync and Data Enable Signals

The diagram in *Figure 10* shows the relationships between the Vertical Sync, Horizontal Sync, and Data Enable signals. *Figure 11* exaggerates the horizontal blanking period to illustrate the relationship between the HS and DE signals as well as the video frame parameters.

4.2 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 overheard and side channel usage (e.g., main control channel, SPI, etc.). Also, the pixel clock and GMSL bandwidth must consider the total resolution of a video (i.e., video data plus blanking). See the *GMSL2 Link System Bandwidth* section for further details.

4.2.1 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 (i.e., 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 (e.g., buffer resets) between frames. The vertical sync signal occurs during the vertical blanking period.

These concepts are illustrated in the diagram (*Figure 9*). The active video (i.e., 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” (i.e., viewable resolution). This designation ignores blanking. The pixel clock and GMSL bandwidth is dependent on the total resolution, however, and includes blanking.

4.2.2Total Blanking Interval

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

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

4.2.3Total Line and Frame Period

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

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

4.2.4Pixel Clock

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

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

4.2.4.1Pixel 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.

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

4.2.4.2Sample Calculation

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

4.2.5Calculating Total Video Bandwidth

$$Video\ BW = PCLK * bpp$$

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 (e.g., 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.

4.3 Definitions

This section contains definitions of basic video terminology.

4.3.1.1 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.

4.3.1.2 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.

4.3.1.3 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.

4.3.1.4 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).

4.3.1.5 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.

4.3.1.6 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.

4.3.1.7 Frame Rate

See Vertical Frame Rate.

4.3.1.8 Front Porch

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

4.3.1.9 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.

4.3.1.10 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.

4.3.1.11 Horizontal Line Frequency

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

4.3.1.12 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.

4.3.1.13 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.

4.3.1.14 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.

4.3.1.15 Pixel Clock

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

4.3.1.16 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."

4.3.1.17 Refresh Rate

See Vertical Frame Rate.

4.3.1.18 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).

4.3.1.19 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.

4.3.1.20 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.

4.3.1.21 Vertical Frame Rate

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

4.3.1.22 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.

4.4 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)

4.4.1Heartbeat Mode On

Default setting. No additional configuration is required.

4.4.2Heartbeat 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 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.

5 Forward Error Correction

5.1 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 primarily used to enable robust transmission of compressed video over a GMSL2 link. On an uncompressed video stream, a single bit error on a video packet corrupts only one pixel and is visually undetectable (*Figure 12*). However, on a compressed video stream, a single bit error can possibly corrupt the entirety of the DSC macroblock, typically comprising thousands of pixels, and can usually be detected visually (*Figure 13*).

To test the visual impact of bit errors on a compressed video stream, 1000 compressed 4K frames, each with a single bit error, were transmitted over a GMSL2 link with FEC disabled. The average observed macroblock error was 484x48 pixels (0.28% of the image) and was visibly detectable approximately 60% of the time.

Since a single bit error can cause visual corruption of a compressed video stream, FEC is required for all GMSL links with compressed video to minimize the BER so that the probability of bit errors becomes negligible.

Figure 12. Uncompressed Video Frame with 91,480 Random Bit Errors

Figure 13. Compressed Video Frame with 1 Bit Error

The images contrast the visual impact of bit errors on uncompressed and compressed video frames. In *Figure 12*, the 91,480 random bit errors are undetectable. However, in *Figure 13*, a single bit error on a video packet has corrupted an entire DSC macroblock and resulted in visually detectable damage to the video frame.

5.2 Operation

5.2.1Architecture

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 14*) illustrates the data path through the serial link system and highlights error detection, correction, and reporting behavior.

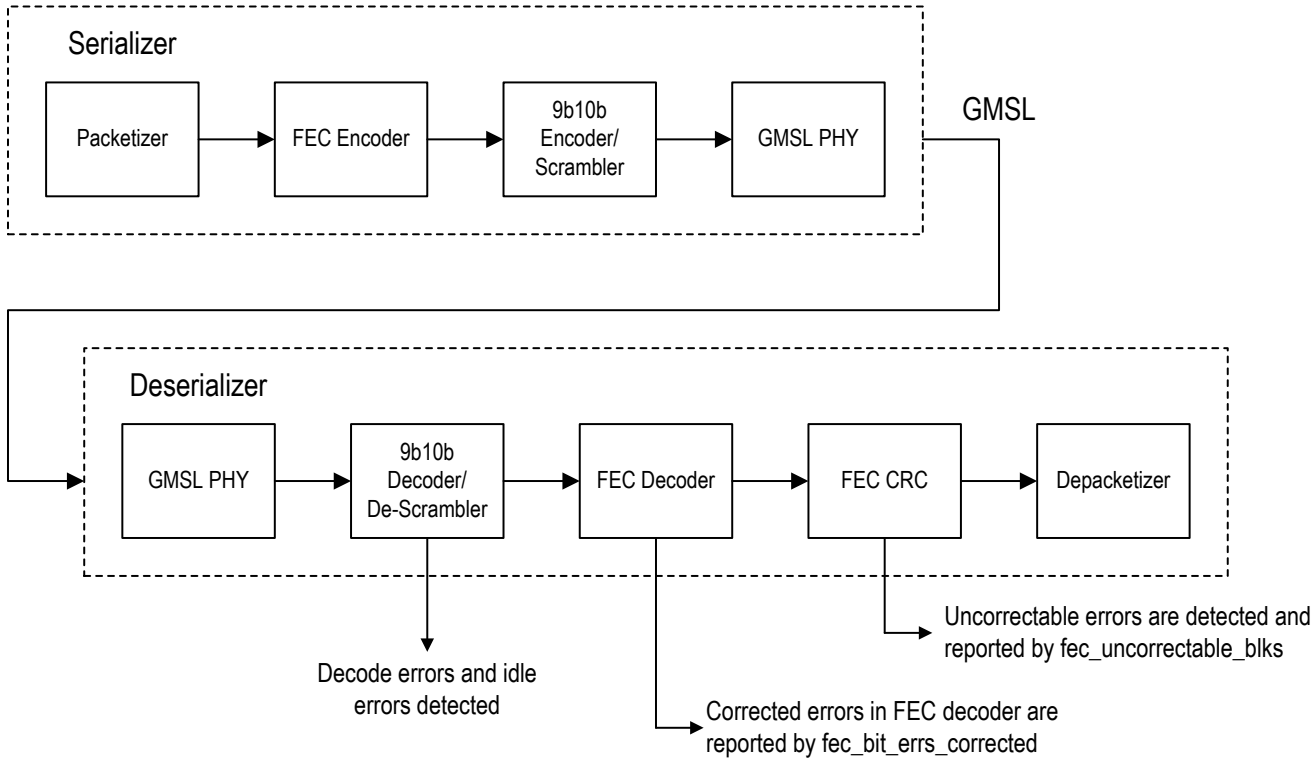


Figure 14. Architecture of FEC Implementation in GMSL Devices

5.2.2BER 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 (e.g., 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 x 10^{38} years.

5.2.3Bandwidth 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. I^2C clock stretching, GPIO delay compensation, etc. can be used to compensate for this latency. See the *I^2C/UART* and *Interface-Specific Bandwidth Calculations* sections for details.

5.2.4Resynchronization

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.

5.2.5Power-Up Configuration

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

5.3 Configuration

FEC only requires a single register write per device to be enabled; additional configuration may be needed to change the ERRB 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 autorecovers if the link is lost and re-established or either device is reset (and re-initialized).

5.3.1Enabling 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.

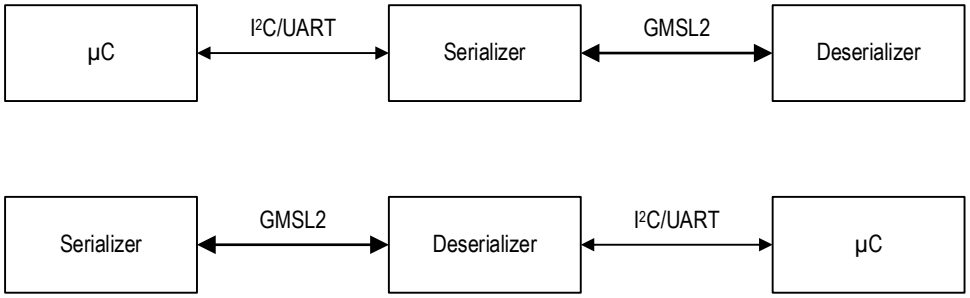


Figure 15. Block Diagram of a Single Microcontroller System

In a single microcontroller system, the microcontroller can be on either side of the link (*Figure 15*). The remote device is configured over the serial link through the control channel.

5.3.1.1 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)

5.3.2Enabling 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 16*). 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.

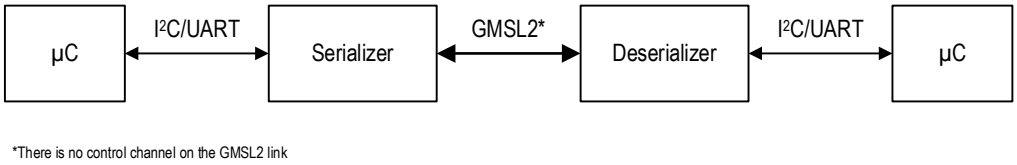


Figure 16. Block Diagram of a Dual Microcontroller System

5.3.2.1Deserializer 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).

5.3.2.2Serializer 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).

5.4 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.

5.4.1 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).

5.4.1.1 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
```

5.4.2 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_clk_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_clk_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.

5.4.3BER Calculations

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

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

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.

5.4.3.1Link 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.

Bidirectional Channels

6 I²C/UART

6.1 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.

6.2 Main Control Channel – I²C/UART

6.2.1Overview

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 17*). I²C multi-main applications are also supported, provided that a software arbitration method is used to prevent collisions (*Figure 18*). 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 (i.e., remote side). This application requires that both GMSL2 devices are configured for the same CC protocol (i.e., 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 17*).

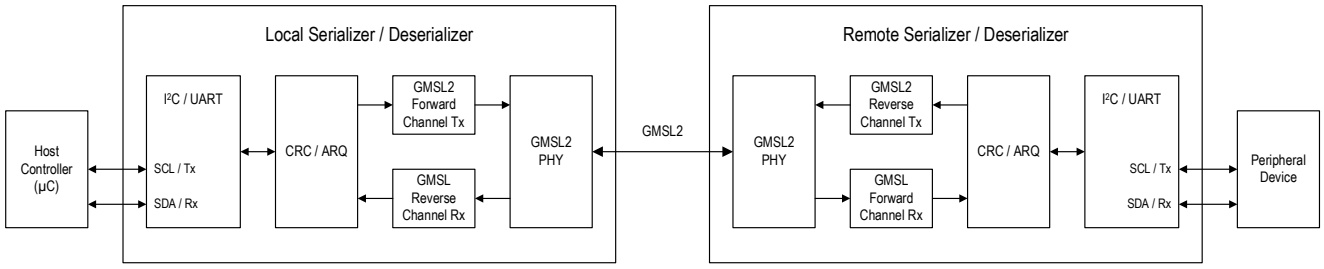


Figure 17. Typical GMSL2 Serial Link System with the Control Channel

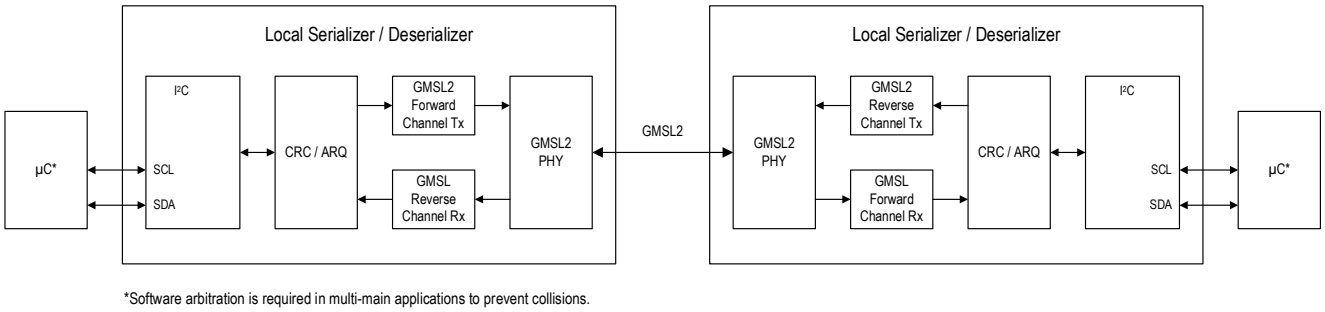


Figure 18. GMSL2 System with Multi-Main Control Channel (I^2C)

Both CC protocols use the GMSL2 device's SDA_RX and SCL_TX pins. Selection of I^2C (SDA and SCL) or UART (Rx and Tx) is determined at power-up with the I^2CSEL pin state or the Configuration Pin (CFG). Refer to device-specific data sheet for the protocol selection settings; note that some parts may feature pins with shared functionality. The I^2C/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 V_{DDIO}.

6.2.2Operation

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 I^2C *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 I^2CSEL pin or CFG pins to the required value with a resistor network. The pin level is latched at power-up and selects either I^2C or UART operation. Refer to 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 17*.

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 (i.e., I^2CSEL or CFG settings must match for both devices).

Each device on the I^2C/UART CC must have a unique device address that is used as the I^2C/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 (PWNDNB = 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 (e.g., 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 ARQ (Automatic Repeat Request) 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.

6.2.3 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.

6.2.3.1 I^2C Mode Configuration

The following tables and sections contain registers associated with the I^2C control channel (CC). Note that the I^2CSEL, 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 9, Table 10, and Table 11* may not exist in all parts. Refer to device-specific data sheets and register documents for the most accurate part information.

Table 9. I²C/UART Selection Status Register

BITFIELD	DESCRIPTION	DECODE
I²CSEL[0]	This bit is set according to the latched I²CSEL/CFG pin value at power-up.	0b0: UART 0b1: I²C

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 10. 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 11. Number of I^2C-to-I^2C Links Configuration Registers

BITFIELD	DESCRIPTION	DECODE
I^2C_AUTO_CFG[0]	When set to 1, I^2C-to-I^2C number of links is automatically determined based on splitter mode. In splitter mode, response from two I^2C channels are expected, otherwise response from one I^2C channel is expected.	0b0: Number of I^2C-to-I^2C links set by I^2C_SRC_CNT[2:0] bits 0b1: Splitter mode automatically determines the number of I^2C-to-I^2C links
I^2C_SRC_CNT[2:0]	I^2C-to-I^2C number of links (valid when I^2C_AUTO_SRC = 0). Set this field to N - 1 when expecting I^2C response from N remote I^2C 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

6.2.3.1.1 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 12*).

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

6.2.3.1.1.1 I²C Data Transfer Formats

Note: In *Figure 19* and *Figure 20*, 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**:

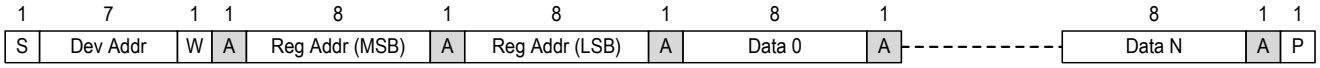


Figure 19. I²C Write Data Transfer Format

I²C **Read**:

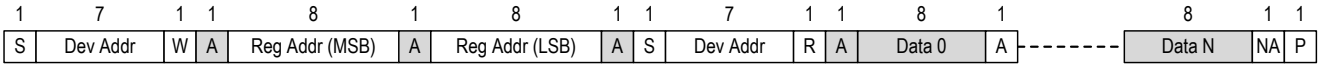


Figure 20. I²C Read Data Transfer Format

Table 12. I²C Acknowledge Bit and Time-Out Status (Read-Only Registers)

BITFIELD	DESCRIPTION	DECODE
REM_ACK_ACKED[0]	Inverse of the I²C acknowledge bit received from remote side.	0b0: I²C acknowledge bit received as 1 0b1: I²C acknowledge bit received as 0
REM_ACK_RECVED[0]	I²C acknowledge bit for any I²C byte has been received from the remote side for the previous GMSL packet with I²C data.	0b0: I²C acknowledge bit not received 0b1: I²C acknowledge bit received
I²C_TIMED_OUT[0]	Internal I²C-to-I²C subordinate or main has timed out while receiving data transfer from remote device.	0b0: Time-out has not occurred 0b1: Time-out has occurred

6.2.3.1.2 I²C over the GMSL2 Link (Remote Device Access)

The I²C channel connects the serializer and deserializer I²C interfaces (i.e., 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 throughI²C transmissions over the GMSL2 link. When an external I²C main (e.g., µC) performs an I²C transaction on the I²C bus on one side of the link (i.e., local side), the I²C events (e.g., Start, Stop, Data Bit 0, and Data Bit 1) are forwarded to the other side of the link (i.e., 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) (e.g., 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 (e.g., µC) 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 (i.e., 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 (i.e., external main µC and any attached peripherals) must support clock stretching; the remote-side I²C peripherals are not required to support clock stretching (*Figure 21*).

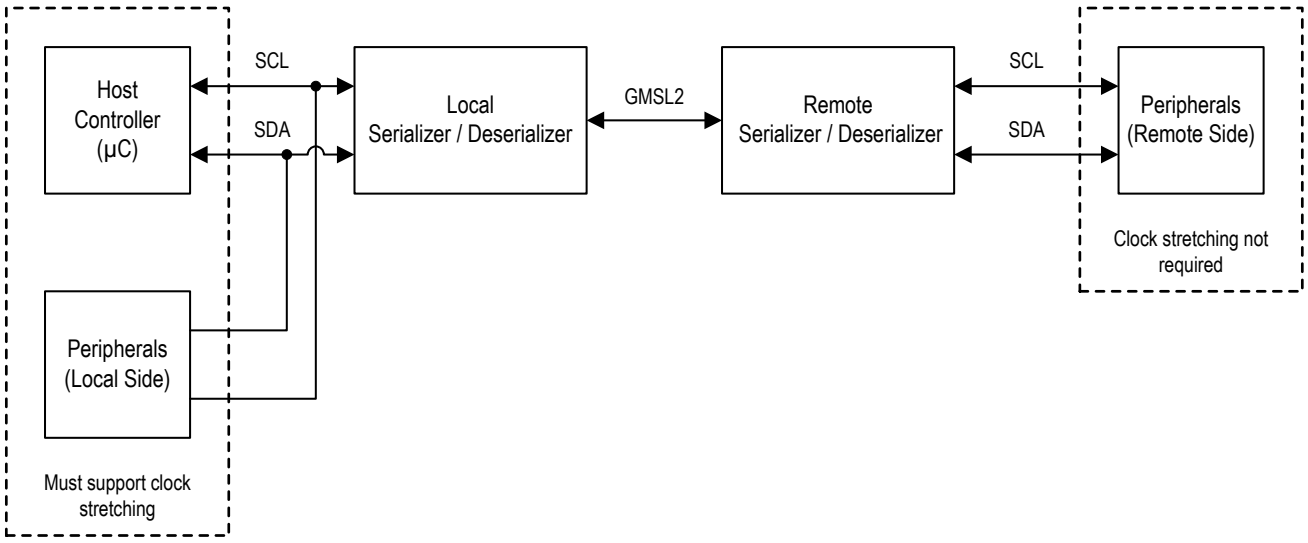


Figure 21. I²C Clock Stretching

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 (e.g., µC). 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 13* and *Table 14*) are programmed according to the desired I²C bit rate and in compliance with the official I²C timing parameters.

Table 13. Internal I²C Subordinate Configuration Registers

BITFIELD	DESCRIPTION	DECODE
SLV_SH[1:0]	I²C-to-I²C subordinate setup and hold time setting (setup, hold). Configures the interval between SDA and SCL transitions when driven by the internal I²C subordinate.	0b00: Set for I²C Fast-mode Plus speed 0b01: Set for I²C Fast-mode speed 0b10: Set for I²C Standard-mode speed 0b11: Reserved
SLV_TO[2:0]	I²C-to-I²C subordinate time-out setting. Internal GMSL2 I²C 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 14. Internal I²C Main Configuration Registers

BITFIELD	DESCRIPTION	DECODE
MST_BT[2:0]	I²C-to-I²C main bit rate setting. Configures the I²C bit rate used by the internal I²C main (in the device on the remote side from the external I²C main). Set this according to the I²C speed mode.	0b000: 9.92Kbps - Set for I²C Standard- mode speed 0b001: 33.2Kbps - Set for I²C Standard-mode speed 0b010: 99.2Kbps - Set for I²C Standard- or Fast-mode speed 0b011: 123Kbps - Set for I²C Fast-mode speed 0b100: 203Kbps - Set for I²C Fast-mode speed 0b101: 397Kbps - Set for I²C Fast- or Fast-mode Plus speed 0b110: 625Kbps - Set for I²C Fast-mode Plus speed 0b111: 980Kbps - Set for I²C Fast-mode Plus speed
MST_TO[2:0]	I²C-to-I²C main time-out setting. Internal GMSL2 I²C 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 (e.g., µC) 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.

6.2.3.1.3 I²C Splitter Mode

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

- The µC connected to the local device in splitter mode can communicate with each connected remote device and the attached peripherals.
- The µC 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 µC is connected to the serial link system, only one µC can use the control channel at a time. The serial link assumes that only one µC is using the CC at any one time. Multiple µCs must take turns using the control channel to avoid error conditions. If the CC is in use by a µC, attempts from another µC 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 15*. Each function is described in more detail in following sections. Note that these functions may be used in combination.

Table 15. I²C Address Configuration

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

6.2.3.1.4 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.

6.2.3.1.4.1 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.

6.2.3.1.5 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 µC to separately access different devices and peripherals with the same device address throughsoftware configuration. Configuration details are presented in *Table 16*.

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

Table 16. I²C Address Translator Configuration Registers

BITFIELD	DESCRIPTION	DECODE
SRC_A[6:0]	I²C address translator source A.	0bXXXXXXX: Value of I²C SRC_A

	When I²C device address matches I²C SRC_A, internal I²C main (on remote side) replaces the device address by I²C DST_A.	
DST_A[6:0]	I²C address translator destination A. See the description of I²C SRC_A.	0bXXXXXXX: Value of I²C DST_A
SRC_B[6:0]	I²C address translator source B. When I²C device address matches I²C_SRC_B, internal I²C main (on remote side) replaces the device address by I²C_DST_B.	0bXXXXXXX: Value of I²C SRC_B
DST_B[6:0]	I²C address translator destination B. See the description of I²C SRC_B.	0bXXXXXXX: Value of I²C 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 16*). Both the serializer and the peripheral(s) see this translated address (i.e., 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 µC 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].

6.2.3.1.5.1 I²C Address Translation Example

An I²C address reassignment and translation example is shown in the block diagram (*Figure 22*). 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.

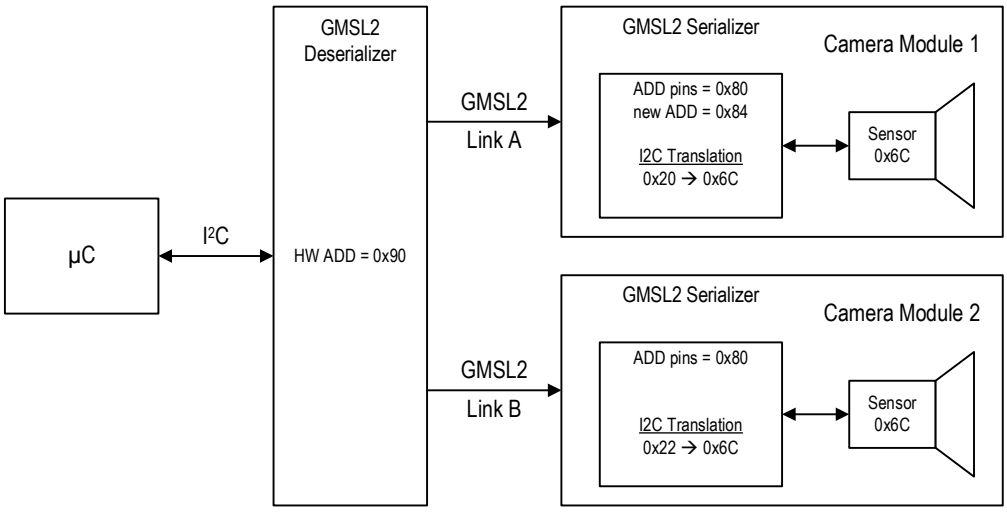


Figure 22. I²C Address Translation

6.2.3.1.6 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 throughI²C 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 throughI²C 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 µC is located on the deserializer side (*Figure 23*).

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].

6.2.3.1.6.1 I²C Broadcasting Example

An example of I²C address reassignment, translation, and broadcasting is shown in the block diagram (*Figure 29*). A GMSL2 CSI-2 deserializer is connected to two identical camera modules. Each camera module comprises a GMSL2 serializerat 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 (i.e., the modified device address configured with I²C address reassignment); in the link B serializer, 0xC4 is translated to 0x80.

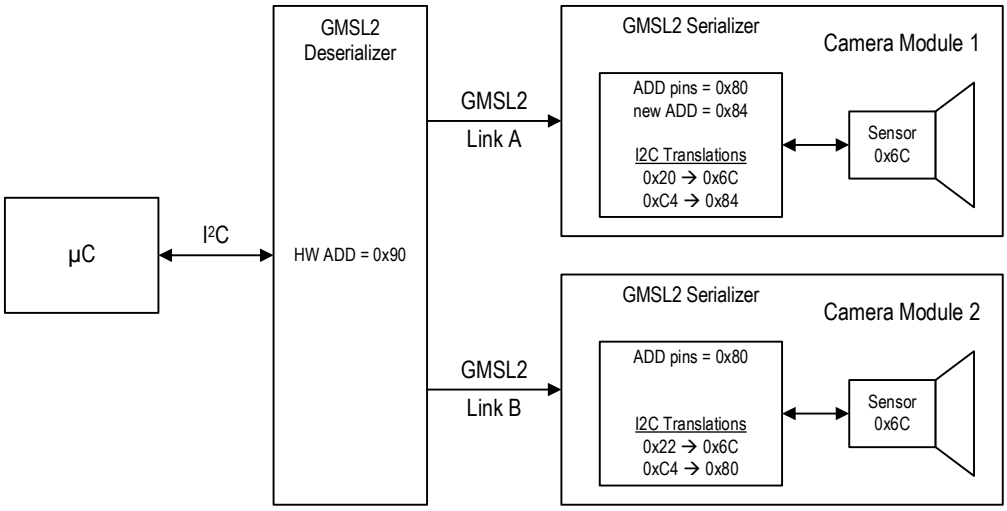


Figure 23. I²C Broadcasting

6.2.3.2 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.

6.2.3.3 I²C Channel GMSL2 Bandwidth Utilization

See the *GMSL2 Link Bandwidth Consumption from Side Channels* – I²C section for details.

6.2.3.4 I²C Debug Techniques

The I²C protocol is used by most µCs. Ensure that µCs 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_RECVED 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.

6.2.4Main 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.

6.2.4.1UART Mode Configuration

The following tables and sections contain registers that are associated with the main UART control channel. Note that the I^2CSEL, DIS_REM_CC, DIS_LOCAL_CC, and CFG_BLOCK registers are common to the main I^2C and UART control channel protocols. See the *I2C Mode Configuration* section for information regarding the shared registers (*Table 9* and *Table 10*).

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 I^2C, 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.

6.2.4.1.1 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 µC 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 µC to communicate with this peripheral without requiring the use of the UART Pass-through connection. After the communication is complete, the µC can access GMSL2 devices in UART Base Mode. Pass-through UART does not have access to GMSL2 devices.

6.2.4.1.1.1 UART Base Mode

UART Base Mode is enabled by default at power-up. In base mode, µCs 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 µC can also communicate with attached remote peripherals compatible with the GMSL2 UART protocol. UART data transmitted by the µC 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 µC must wait up to 48-bits time or 200µs, 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 µC 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 (i.e., sync or acknowledge frame) can be configured to be delayed by 0-, 1-, 4-, or 8-bits time with the OUT_DELAY register (*Table 17*) 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 17. 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 in order 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

6.2.4.1.1.2UART Bypass Mode

In UART bypass mode, UART commands are not interpreted by the GMSL2 devices, and data is passed directly from the µC to connected peripherals (and from peripherals to the µC). The µC 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 (*Table 18)*.

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_T O 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 *Table 18*.

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

Table 18. 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 *UART Base/Bypass Mode Operation* 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.

6.2.4.1.2 UART Splitter Mode

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

- The µC connected to the local device in splitter mode can communicate with each connected remote device and the attached peripheral(s).
- The µC 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 µCs, the first µC to communicate with the local device in splitter mode gets dedicated access to the UART link (the UART link from the other µC is blocked at the splitter device). The UART link remains dedicated to that µC if it maintains communication. When the µC 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 19*). Any µC 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 *I^2C Splitter Mode* section for shared configuration details.

Table 19. 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

6.2.4.2UART Frame Format

A regular UART frame with an even parity bit is used to carry one byte of data (*Figure 24*). 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 (i.e., 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).

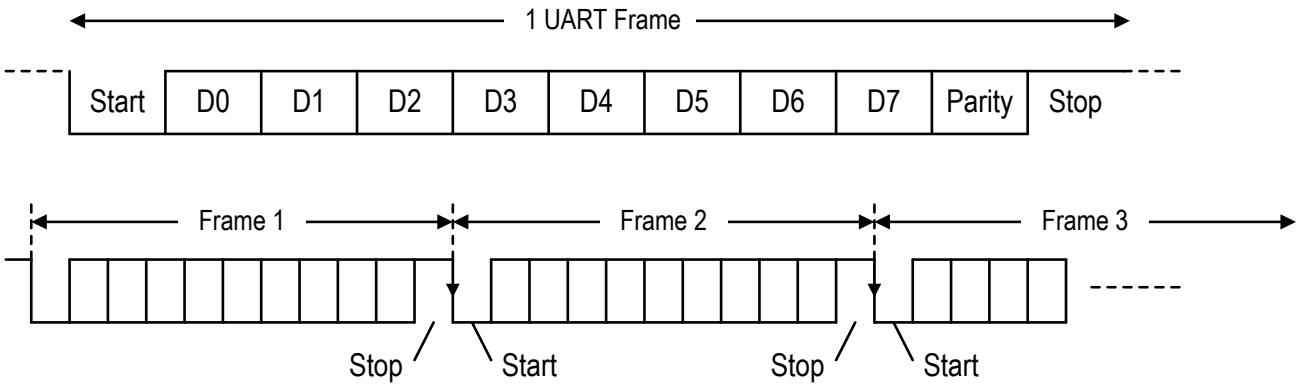


Figure 24. UART Frame Format

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.

6.2.4.2.1 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 25*). 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.

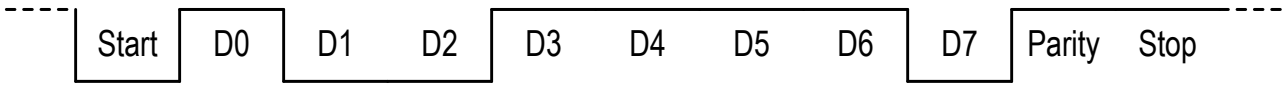


Figure 25. UART Synchronization Frame

6.2.4.2.2 UART Write Protocol

The UART write protocol consists of a 5-byte header followed by one or more data bytes (*Figure 26*). 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.

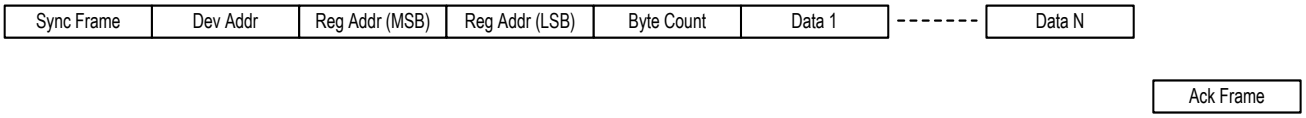


Figure 26. UART Write Protocol Format

6.2.4.2.3 UART Read Protocol

The UART read protocol consists of 5 bytes (*Figure 27*). 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.

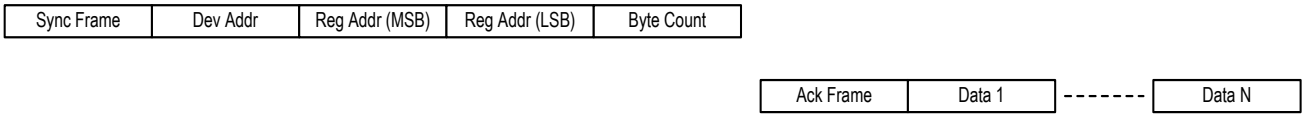


Figure 27. UART Read Protocol Format

6.2.4.2.4 UART Acknowledge Frame

The Ack frame is a regular UART frame with a value of 0xC3 (*Figure 28*). When a transmission is successfully received and recognized, the addressed device responds with an acknowledge (Ack) frame to inform the µC 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.

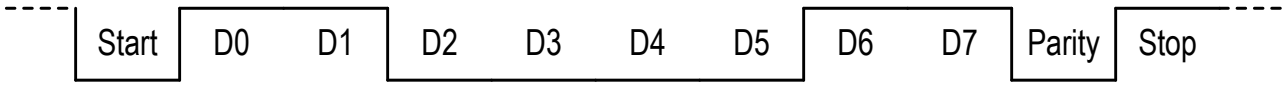


Figure 28. UART Acknowledge Frame Format

6.2.4.2.5 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 µC) after a transaction is completed (i.e., when the µC 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 µC 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.

6.2.4.3UART Channel GMSL2 Bandwidth Utilization

See the *GMSL2 Link Bandwidth Consumption from Side Channels* – UART section for details.

6.2.4.4UART 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 20*). The value in these status bits is the ratio of 150MHz to the detected UART bit rate (e.g., it is ~300 when UART bit rate is 500Kbps).

Table 20. UART Detected Bit Length (Read-Only Registers)

BITFIELD	DESCRIPTION	DECODE
BITLEN_LSB[7:0]	UART detected bit length in terms of internal 150MHz clock, low 8 bits.	0xXXXXXXXX: 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 21*).

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

Table 21. 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

6.2.5Disabling 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 I^2C/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.

6.3 Pass-Through Channels – I²C/UART

6.3.1 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 *Main 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 main 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.

6.3.2Operation

The pass-through I²C/UART channels are independent of the main 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 main 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 main 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 µC 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 (e.g., 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 29*. The local serializer/deserializer device is connected to the µC; the remote deserializer/serializer is connected to the remote peripheral I²C/UART device.

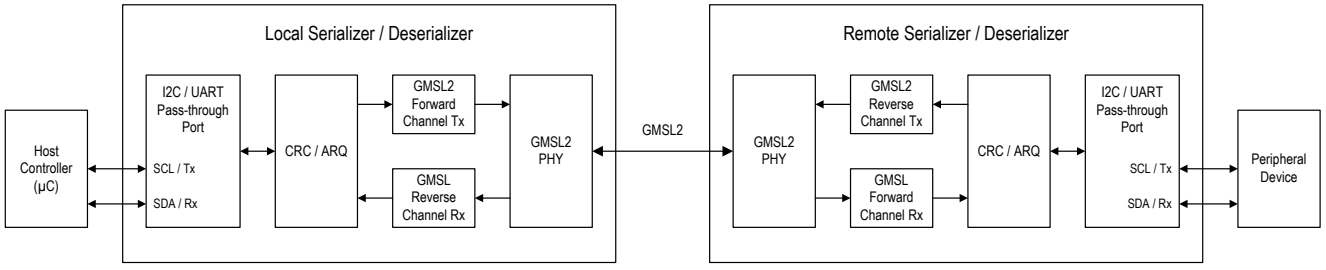


Figure 29. Pass-Through I²C/UART

6.3.3Pass-Through I²C

6.3.3.1I²C Mode Configuration

The configuration of pass-through I²C channels is independent from main 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 22*).

Table 22. Pass-through I²C Channels Enable Registers

BITFIELD	DESCRIPTION	DECODE
IIC_1_EN[0]	Enables pass-through I²C Channel 1 (SDA1/RX1, SCL1/TX1).	0b0: I²C pass-through Channel 1 disabled 0b1: I²C pass-through Channel 1 enabled
IIC_2_EN[0]	Enables pass-through I²C Channel 2 (SDA1/RX1, SCL1/TX1)	0b0: I²C pass-through Channel 2 disabled 0b1: I²C pass-through Channel 2 enabled

The following tables contain the pass-through I²C internal subordinate (*Table 23*) and main (*Table 24*) 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 23. Pass-Through I²C Internal Subordinate Configuration Registers

BITFIELD	DESCRIPTION	DECODE
SLV_SH_PT[1:0]	Pass-through I²C-to-I²C subordinate setup and hold time setting (setup, hold). Configures the interval between SDA and SCL transitions when driven by the internal I²C subordinate. Set this according to the I²C speed mode.	0b00: Set for I²C Fast-mode Plus speed 0b01: Set for I²C Fast-mode speed 0b10: Set for I²C Standard-mode speed 0b11: Reserved
SLV_TO_PT[2:0]	Pass-through I²C-to-I²C subordinate time-out setting. Internal GMSL2 I²C 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 24. Pass-Through I²C Internal Main Configuration Registers

BITFIELD	DESCRIPTION	DECODE
MST_BT_PT[2:0]	Pass-through I²C-to-I²C main bit-rate setting. Configures the I²C bit rate used by the internal I²C main (in the device on remote side from the external I²C main).	0b000: 9.92Kbps – Set for I²C Standard- mode speed 0b001: 33.2Kbps – Set for I²C Standard-mode speed 0b010: 99.2Kbps – Set for I²C Standard- or Fast-mode speed 0b011: 123Kbps – Set for I²C Fast-mode speed 0b100: 203Kbps – Set for I²C Fast-mode speed 0b101: 397Kbps – Set for I²C Fast- or Fast-mode Plus speed 0b110: 625Kbps – Set for I²C Fast-mode Plus speed 0b111: 980Kbps – Set for I²C Fast-mode Plus speed
MST_TO_PT[2:0]	Pass-through I²C-to-I²C main timeout setting. Internal GMSL2 I²C 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 I²C-to-I²C main bit rate	0b0: Do not double the pass-through I²C-to-I²C main bit rate 0b1: Double the pass-through I²C-to-I²C main bit rate

Table 25 contains read-only I²C acknowledge and time-out status registers. Note that the numbers (i.e., 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 25. Pass-Through I²C Acknowledge and Time-Out Status (Read-Only Registers)

BITFIELD	DESCRIPTION	DECODE
REM_ACK_ACKED_1[0]	In pass-through I²C Channel 1, inverse of the I²C acknowledge bit received from remote side.	0b0: I²C acknowledge bit received as 1 0b1: I²C acknowledge bit received as 0
REM_ACK_RECVED_1[0]	In pass-through I²C Channel 1, I²C acknowledge bit for any I²C byte is received from remote side for the previous I²C packet.	0b0: I²C acknowledge bit not received 0b1: I²C acknowledge bit received
I²C_TIMED_OUT_1[0]	In pass-through I²C Channel 1, internal I²C-to-I²C 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 I²C Channel 2, inverse of the I²C acknowledge bit received from remote side.	0b0: I²C acknowledge bit received as 1 0b1: I²C acknowledge bit received as 0
REM_ACK_RECVED_2[0]	In pass-through I²C Channel 2, I²C acknowledge bit for any I²C byte is received from remote side for the previous I²C packet.	0b0: I²C acknowledge bit not received 0b1: I²C acknowledge bit received

I²C_TIMED_OUT_2[0]	In pass-through I²C Channel 2, internal I²C-to-I²C 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 main CC on the remote side on a per channel basis by configuring the XOVER_EN_1 and XOVER_EN_2 bitfields.

See *Table 26* for I²C/UART pin assignments and channel connections configuration registers.

Table 26. Pass-Through I²C/UART Device Pin Assignments and Channel Connections Configuration Registers

BITFIELD	DESCRIPTION	DECODE
PT_SWAP[0]	Swaps I²C/UART pass-through device pin assignments	0b0: Do not swap pin assignments 0b1: Swap pin assignments
XOVER_EN_1[0]	Connects pass-through I²C/UART Channel 1 to main control channel on remote side.	0b0: Do not connect 0b1: Connect
XOVER_EN_2[0]	Connects pass-through I²C/UART Channel 2 to main control channel on remote side.	0b0: Do not connect 0b1: Connect

6.3.3.1.1 Pass-Through I²C Address Translation

Pass-through I²C channels allow address translation for the remote-side GMSL2 device (*Table 27*). 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 27. Pass-Through I²C Address Translator Configuration Registers

BITFIELD	DESCRIPTION	DECODE
SRC_A_1[6:0]	I²C address translator source A. When I²C device address matches SRC_A_1, internal I²C main replaces the device address by DST_A_1.	0bXXXXXXX: I²C address translator source A
DST_A_1[6:0]	I²C Address Translator Destination A. See the description of SRC_A_1	0bXXXXXXX: I²C address translator destination A
SRC_B_1[6:0]	I²C Address Translator Source B. When I²C device address matches SRC_B_1, internal I²C main replaces the device address by DST_B_1.	0bXXXXXXX: I²C address translator source B
DST_B_1[6:0]	I²C Address Translator Destination B. See the description of SRC_B_1	0bXXXXXXX: I²C address translator destination B
SRC_A_2[6:0]	I²C address translator source A. When I²C device address matches SRC_A_2, internal I²C main replaces the device address by DST_A_2.	0bXXXXXXX: I²C address translator source A
DST_A_2[6:0]	I²C Address Translator Destination A. See the description of SRC_A_2.	0bXXXXXXX: I²C address translator destination A
SRC_B_2[6:0]	I²C Address Translator Source B. When I²C device address matches SRC_B_2, internal I²C main replaces the device address by DST_B_2.	0bXXXXXXX: I²C address translator source B
DST_B_2[6:0]	I²C Address Translator Destination B. See the description of SRC_B_2.	0bXXXXXXX: I²C address translator destination B

6.3.4Pass-Through UART

6.3.4.1UART 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 I^2C and UART channels. See *Table 26* 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 28*).

Table 28. 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 29*).

Table 29. 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 main 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 main 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 30*).

Table 29. 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.	0xXXXXXXXX: 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.	0xXXXXXX: 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.	0xXXXXXXXX: 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.	0xXXXXXX: 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 31*).

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

Table 30. 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. Set to 1 following an overflow condition. Clears upon read.	0b0: No overflow occurred 0b1: Overflow occurred
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

6.3.5Pass-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 main 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.

6.3.6Pass-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_RECVED_1 and REM_ACK_RECVED_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.

7 Serial Peripheral Interface

7.1 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 IOs than parallel buses. Data is synchronized with the clock signal and transmitted throughtwo 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 µC/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 I^2C control of RO and BNE input/output.

7.1.1GMSL2 SPI Architecture

GMSL2 enables an SPI main on one the 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 30* 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 µC) operates with transmit data sent first. Once data is received, the µC then reads back the received data.

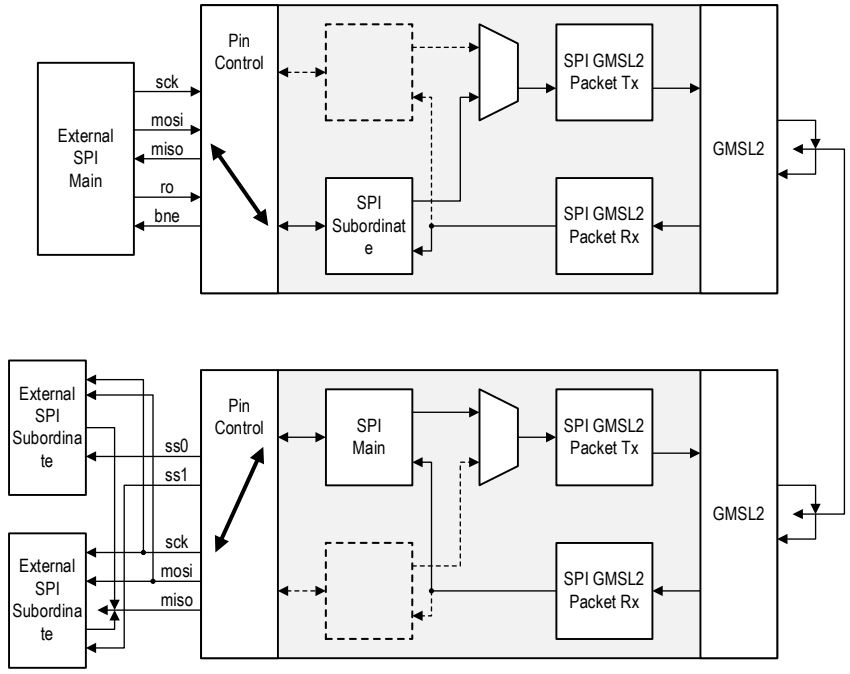


Figure 30. GMSL2 SPI Interface

7.2 Operation

7.2.1SPI 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 throughan 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.

7.2.2SPI 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.

7.2.2.1Read 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 (i.e., 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.

7.2.2.2Buffer 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.

7.2.3SPI Control Commands

Several control commands are used when RO is high and read/command mode is enabled.

7.2.3.1Device 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 (e.g., 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'

7.2.3.2Subordinate 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

7.2.3.3Read 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 (i.e., 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.

7.2.4SPI 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.

7.2.4.1SPI 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.
SCK = 300MHz / (150 x 2) = 1MHz

Example 2: SCK = 5MHz
Write 0x1E (dec 30) to registers SPI_4 and SPI_5.
SCK = 300MHz / (30 x 2) = 5MHz

Example 3: SCK = 42.85MHz:
Write 0x03 (dec 03) to register SPI_4 and write 0x04 (dec 04) to register SPI_5.
SCK = 300MHz / (3 + 4) = 42.85MHz

For higher SCK values, some GMSL2 device SPI MFP pins may need to be adjusted to have faster speed group settings (i.e., 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.

7.2.4.2Minimum 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 32*).

Table 31. 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: 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.

7.3 Configuration

7.3.1 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.

7.3.2 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.

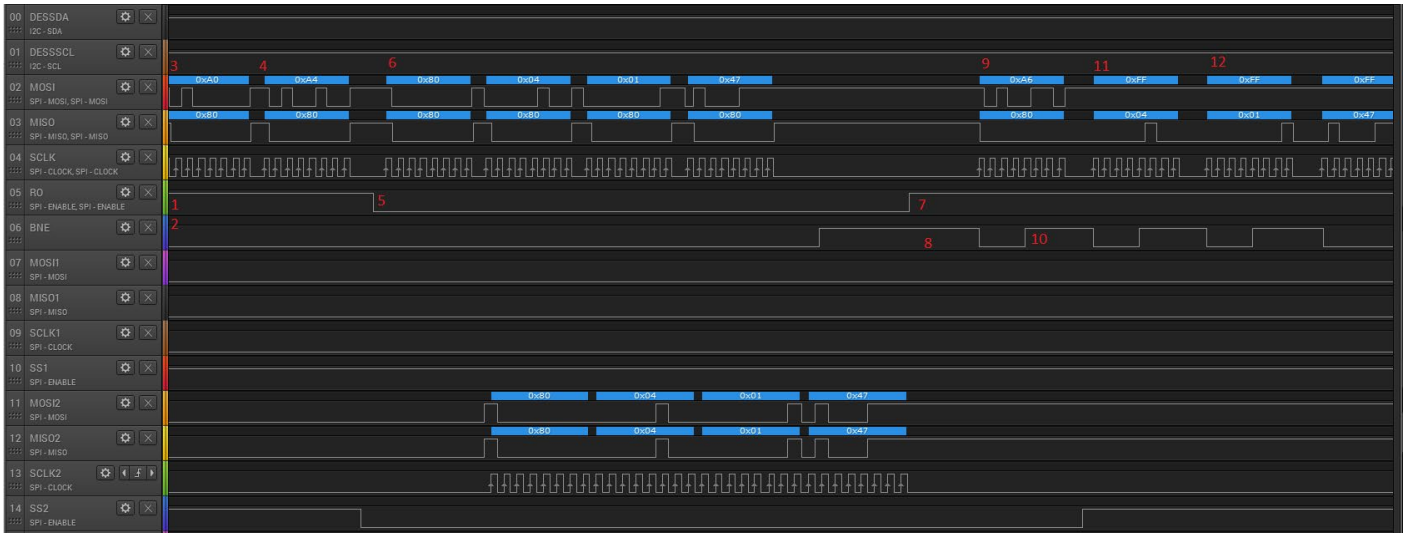


Figure 31. GMSL2 SPI Implementation

7.3.3SPI 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 througha read transaction. At the end of the burst, perform only read transactions until the BIT is zero.

7.3.3.1SPI 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).

7.3.3.2SPI 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).

7.3.4Multiple SPI IDs

In point-to-point SPI applications, the GMSL devices can be set to ignore the packet IDs and accept all packets (SPI_IGNR_ID = 1). For multipoint GMSL topology (e.g., 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 (i.e., 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 32*).

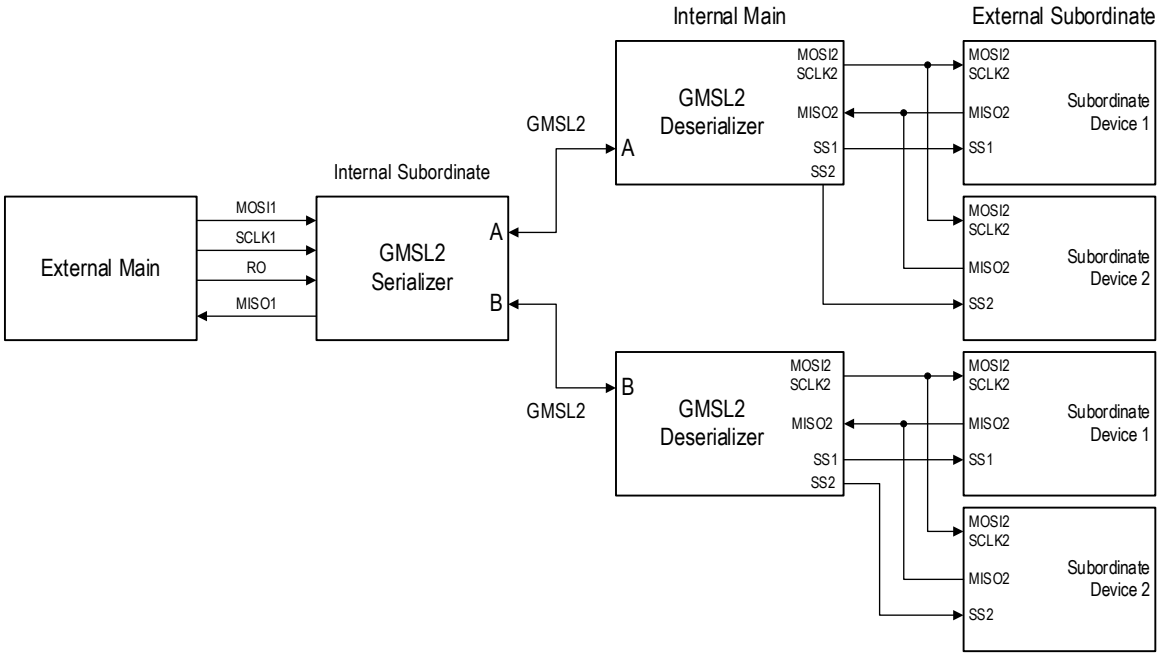


Figure 32. SPI Network: Two Deserializers and One Serializer

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 33*).

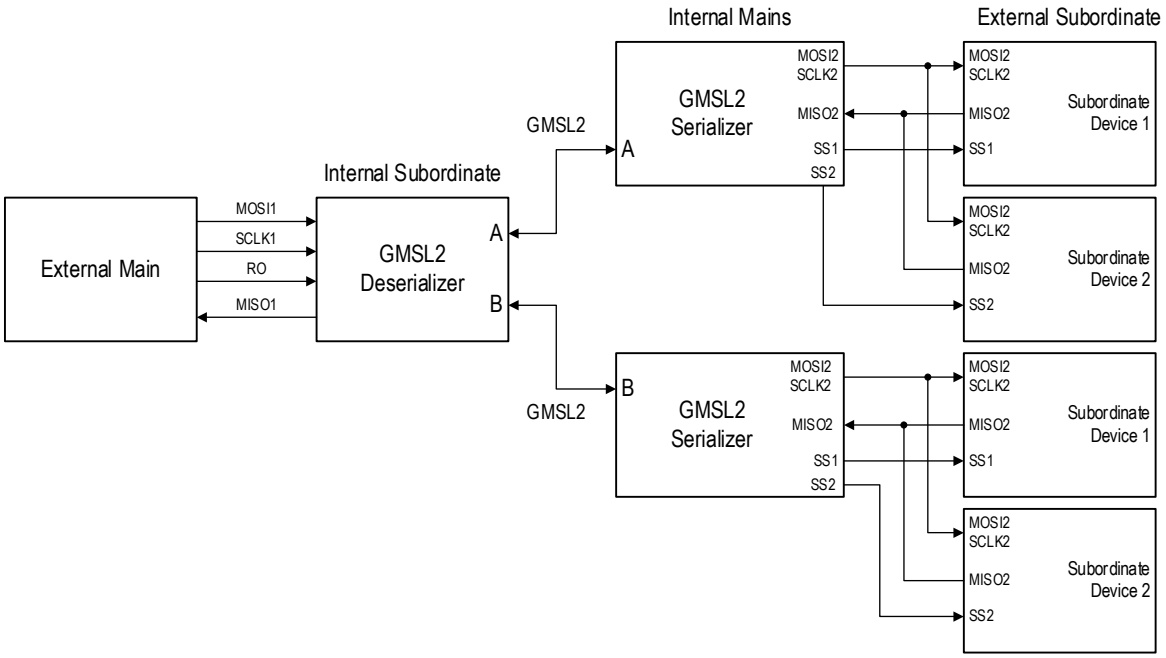


Figure 33. SPI Network: Two Serializers and One Deserializer

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
```

7.3.5Typical 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 34* 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.

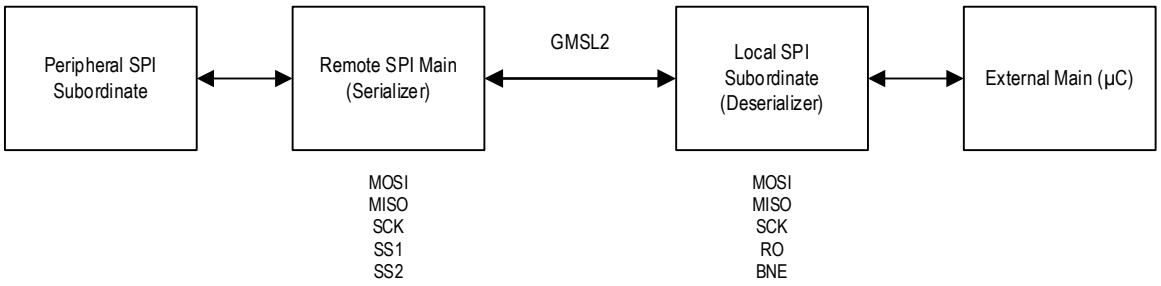


Figure 34. Typical SPI Application

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

8 GMSL2 Link System Bandwidth

8.1 Overview

GMSL2 systems provide robust serial link connections between camera and display interfaces. Each interface in these interconnected systems (i.e., 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 35*) and table (*Table 33*) show an example of the integrated calculations necessary for determining bandwidth compliance of a serial link system.

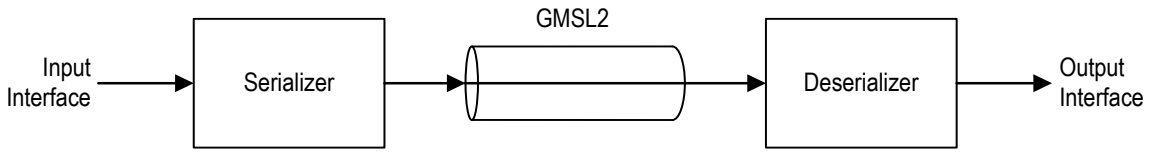


Figure 35. GMSL2 System Block Diagram

Figure 35 is a visual representation of the serial link system stages referenced in the *Table 33* 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 I^2C control channel information and a 1Mbps switching GPIO.

Table 32. 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:

$$4.04Gbps + I2C + GPIO = Total\ GMSL\ Link\ Bandwidth$$

$$4.04Gbps + 4Mbps + 40Mbps = 4.084Gbps$$

See *GMSL2 Link Bandwidth Consumption from Side Channels* for additional information.

8.2 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 (e.g., 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:

$$Video\ Payload = PCLK * bpp$$

The equation used to calculate the total forward channel video bandwidth is shown as follows:

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

$$Video\ BW = \text{PCLK} * \left[(bpp) + \left(\frac{1}{2}\right) + \left(\frac{1}{2}\right)\right] * \left(\frac{10}{9}\right) * \left(\frac{2048}{2047}\right) * \left[\left(\frac{128}{120}\right) * \left(\frac{1}{3}\right)\right]$$

$$Video\ BW = \text{PCLK} * \left[(bpp) + \frac{1}{2} + \frac{1}{2}\right] * \left(\frac{10}{9}\right) * \left(\frac{2048}{2047}\right) * \left[\left(\frac{128}{120}\right) * \left(\frac{1}{3}\right)\right]$$

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 (e.g., 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 34* lists the maximum video payload for each mode. The maximum rates factor in sufficient margin to accommodate sideband channels.

Table 33. Maximum Video Payload for GMSL2 Modes

GMSL2 MODE	MAXIMUM VIDEO PAYLOAD
3Gbps Mode	2.6Gbps (2600Mbps)
6Gbps Mode	5.2Gbps (5200Mbps)

8.3 Interface-Specific Bandwidth Calculations

8.3.1CSI-2 Bandwidth Calculations

The pixel clock for MIPI input interfaces is determined by the input MIPI speed (i.e., 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.

8.3.1.1CSI-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

8.3.1.1.1 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.

8.3.1.1.1.1 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 35*.

$$PCLK = \frac{Total\ MIPI\ Data\ Rate}{bpp}$$

Table 34. 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

8.3.1.1.1.2Double 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.

$$PCLK = \frac{LANE\_CNT * LANE\_RATE}{bpp}$$

$$PCLK = \frac{2 * 1Gbps}{24\ bpp} = 83.33MHz\ (note: PCLK_{max} \leq 600MHz)$$

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.

$$PCLK = \frac{LANE\_CNT * LANE\_RATE}{bpp}$$

$$PCLK = \frac{1 * 1.5Gbps}{12\ bpp} = 125MHz\ (note: PCLK_{max} \leq 600MHz)$$

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).

$$PCLK = \frac{LANE\_CNT * LANE\_RATE}{bpp}$$

$$PCLK = \frac{1 * 1.5Gbps}{24\ bpp} = 62.5MHz\ (note: PCLK_{max} \leq 600MHz)$$

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.

8.3.1.1.1.3Zero-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 (i.e., the datatype with the larger bpp value) can only be processed with the clock from the faster video stream (i.e., 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 bpp ≤16.

Example: EMB8 and RAW12 Zero-Padding

An image sensor transmits EMB8 (bpp = 8) and RAW12 (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 bpp = 8 and RAW12 has a bpp = 12. Since the PCLK is calculated using the datatype with the lowest bpp value, the EMB8 value is used in the equation.

$$PCLK = \frac{LANE\_CNT * LANE\_RATE}{bpp}$$

$$PCLK = \frac{2 * 1Gbps}{8\ bpp} = 250MHz\ (note: PCLK_{max} \leq 600MHz)$$

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.

$$BW = 250MHz * (12\ bpp + 0.5) * \left(\frac{10}{9}\right) * \left(\frac{2048}{2047}\right) = 3.47Gbps$$

The GMSL2 link bandwidth consumption of EMD8 and RAW12 datatypes is 3.47Gbps.

8.3.1.1.1.4Double 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 throughthe same video pipe.

To calculate the PCLK, the bpp from the datatype with the lowest bpp value (i.e., higher PCLK value) is used. In this example, the RAW12 bpp value is the lowest (bpp = 12) and is used for the PCLK calculation.

$$PCLK = \frac{LANE\_CNT * LANE\_RATE}{bpp}$$

$$PCLK = \frac{2\ * 1Gbps}{12\ bpp} = 166.67MHz$$

The link bandwidth calculation uses the highest bpp value of the transmitted datatypes. Here, the double loaded EMB8 bpp value of 16 is used.

$$BW = 166.67MHz * (16\ bpp + 0.5) * \left(\frac{10}{9}\right) * \left(\frac{2048}{2047}\right) = 3.057Gbps$$

The GMSL2 link bandwidth consumption of EMB8 and RAW12 datatypes is 3.057Gbps.

8.3.1.1.2 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 36*). 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.

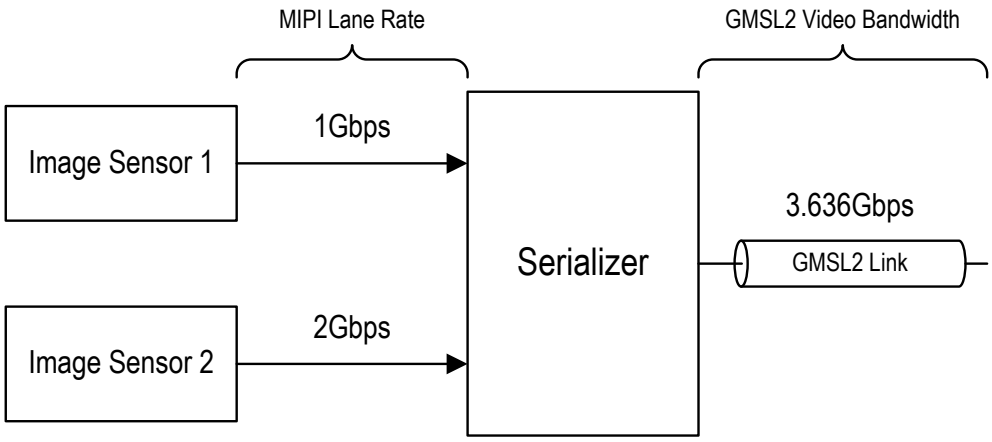


Figure 36. A GMSL2 Serializer with Two Independent MIPI Inputs

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:

$$PCLK = \frac{LANE\_CNT * LANE\_RATE}{bpp}$$

$$PCLK = \frac{1 * 1Gbps}{8\ bpp} = 125MHz$$

$$BW = 125MHz * (9\ bpp + 0.5) * \left(\frac{10}{9}\right) * \left(\frac{2048}{2047}\right) = 1.320Gbps$$

Note: BPP = 9 is the minimum allowed bpp value for GMSL2 link BW calculation.

RAW12 video stream:

$$PCLK = \frac{LANE\_CNT * LANE\_RATE}{bpp}$$

$$PCLK = \frac{1 * 2Gbps}{12\ bpp} = 166.67MHz$$

$$BW = 166.67MHz * (12\ bpp + 0.5) * \left(\frac{10}{9}\right) * \left(\frac{2048}{2047}\right) = 2.316Gbps$$

Combined GMSL2 bandwidth consumption:

$$BW_{peak} = 1.320Gbps + 2.316Gbps = 3.636Gbps$$

8.3.1.2CSI-2 Deserializers – D-PHY Output

The MIPI CSI-2 output clock frequency is independently configured. The phy_csi_tx_dpll_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:

- $Total\ MIPI\ data\ rate = LANE\_RATE * LANE\_CNT$
- $Total\ MIPI\ data\ rate \geq PCLK * (BPP_{single\ video\ receiver\ pipe})$

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:

$$(PCLK1 * BPP1) + (PCLK2 * BPP2) + \cdots \leq Total\ MIPI\ data\ rate$$

- If multiple video pipes have time-separated data:

$$MAX\{(PCLK1 * BPP1), (PCLK2 * BPP2), \ldots\} \leq Total\ MIPI\ data\ rate$$

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.

8.3.1.3CSI-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 36*). There is no line coding overhead (e.g., 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 37*).

Table 35. MIPI D-PHY and C-PHY Lane Configurations and Output Bandwidth

	D-PHY		C-PHY	
Width (CLKxDATA)	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

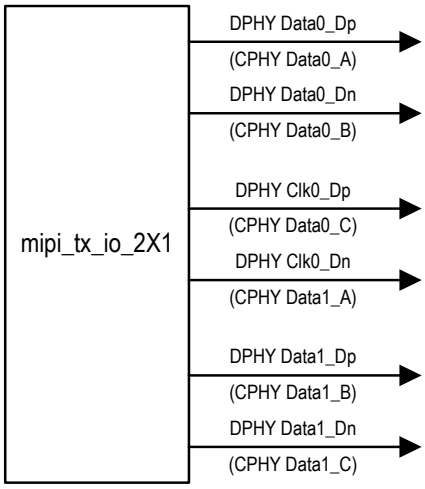


Figure 37. MIPI Lane Configurations

8.4 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 besides RGMII. Disabling CRC is not recommended for most applications. See the *CRC Error Detection and ARQ Error Correction* section for additional details.

8.4.1GPIO

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.

GPIO Forward BW (Mbps) = $(T * (2 + COMP + CRC) * 20)/1000000$

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.

8.4.2SPI

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.

SPI Forward BW (Mbps) = $(fSCK/(8 * L)) * (2 + CRC + roundup((L - 1)/2)) * 20$

SPI Reverse BW (Mbps) = $(fSCK/(8 * L)) * 2 * 20 * CRC$

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.

8.4.3 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.

I²C **Forward BW (Mbps)** = $(fSCL/4500) * (2 + CRC) * 20$

I²C **Reverse BW (Mbps)** = $40 * (fSCL/1000) * CRC$

Where:

- fSCL 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.

8.4.4UART

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.

UART Forward BW (Mbps) = $(fUART/10000) * (2 + CRC) * 20$

UART Reverse BW (Mbps) = $(fUART/10000) * 2 * 20$

Where:

- fUART 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.

9 GMSL2 Error Reporting (ERRB Pin)

9.1 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 (i.e., 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 40kΩ 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 (i.e., 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.

9.2 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 38*. Note that concept is universal to GMSL2 devices; available errors and event statuses may vary by device.

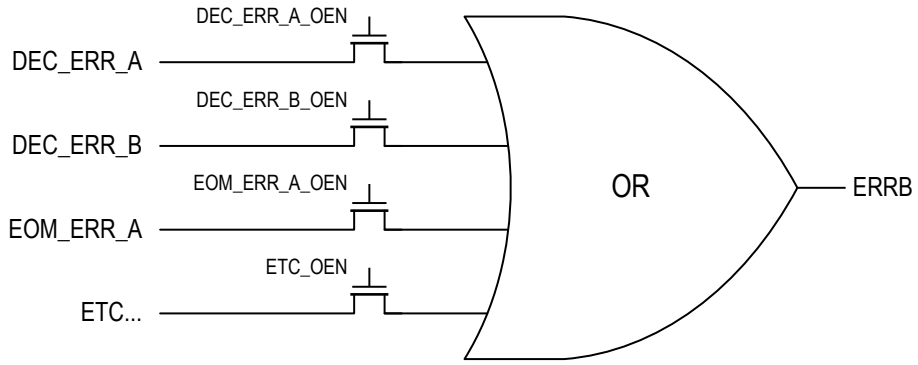


Figure 38. ERRB Logic

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 39* shows the internal error reporting mechanism.

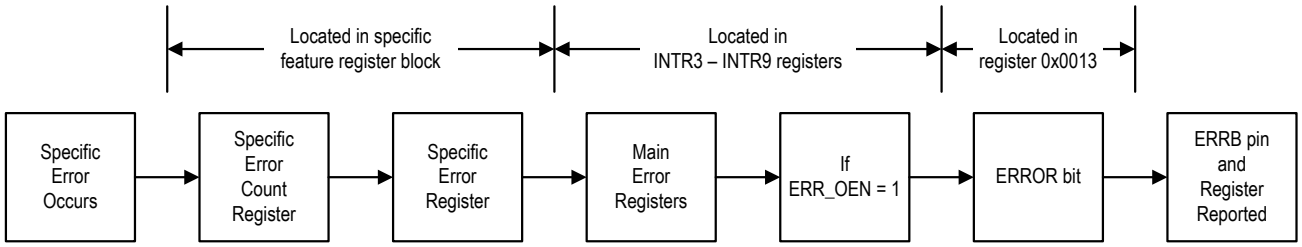


Figure 39. Internal Error Reporting Mechanism

If an error occurs and its ERR_OEN bit is enabled, the ERRB pin is driven low (i.e., 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 40 contains a suggested process for identifying errors resulting in an ERRB transition.

9.2.1Example 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 46*).

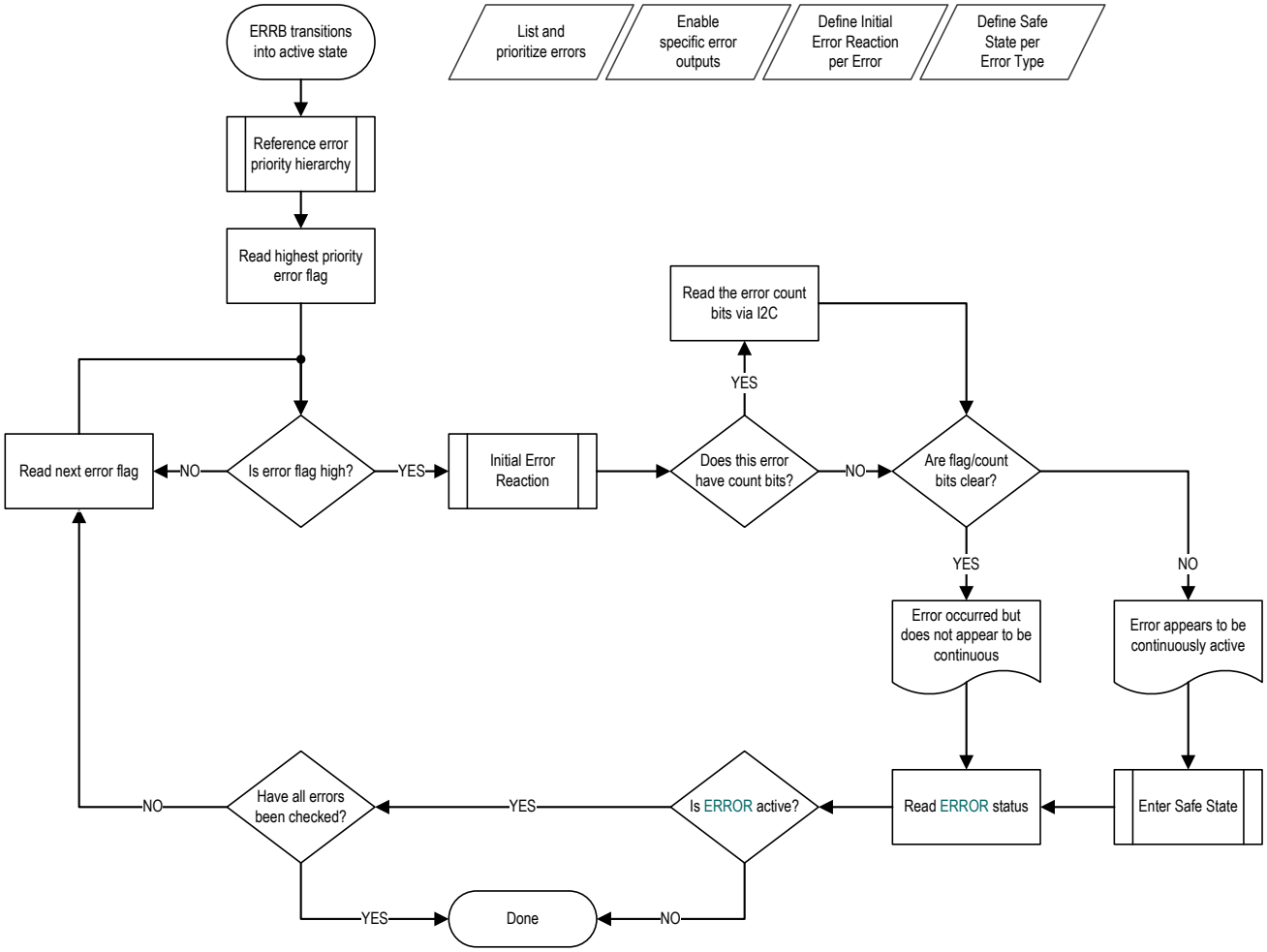


Figure 40. ERRB Transition System Reaction Process

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 40*) 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 (i.e., watermark errors do not need to be checked).
- System returns to nominal operating state.

9.2.2Disabling 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.

9.2.3Remote 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.

9.2.3.1Output Error Status with Local GPIO

Configure the remote device to output the error status throughGPIO 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.

9.2.3.2Combine 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 (i.e., 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 (i.e., combining multiple GPIO tunnel outputs).

9.3 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.

9.3.1 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).

9.3.2GMSL 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).

9.3.3GMSL 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).

9.3.4Line 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.

9.3.5Eye 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.

9.3.6Lock 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.

9.3.7Loss 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.

9.3.8Retransmission 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 (i.e., 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 (i.e., 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.

9.3.9Maximum 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 (i.e., 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.

9.3.10 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_RECVED
ERR_RX_EN_A, ERR_RX_ID_A, ERR_RX_RECVED_A
ERR_RX_EN_B, ERR_RX_ID_B, ERR_RX_RECVED_B
ERR_RX_EN_C, ERR_RX_ID_C, ERR_RX_RECVED_C
ERR_RX_EN_D, ERR_RX_ID_D, ERR_RX_RECVED_D
ERR_RX_EN2_A, ERR_RX_ID2_A, ERR_RX_RECVED2_A
ERR_RX_EN3_A, ERR_RX_ID3_A, ERR_RX_RECVED3_A
ERR_RX_EN4_A, ERR_RX_ID4_A, ERR_RX_RECVED4_A
ERR_RX_EN2_B, ERR_RX_ID2_B, ERR_RX_RECVED2_B
ERR_RX_EN3_B, ERR_RX_ID3_B, ERR_RX_RECVED3_B
ERR_RX_EN4_B, ERR_RX_ID4_B, ERR_RX_RECVED4_B

Refer to the device data sheet register descriptions for more detailed information.

Clearing: Resolve the error status on the remote device.

9.3.11 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 throughinfo 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.

9.3.12 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.

9.3.13 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.

9.3.14 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.

9.3.15 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).

9.3.16 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.

9.3.17 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, OVLP_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).

9.3.18 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.

9.3.19 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.

9.3.20 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.

9.3.21 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.

9.3.22 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.

9.3.23 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.

9.3.24 MIPI Rx Error

Flag:
- MIPI_ERR_FLAG

- MIPI_ERR_OEN

OEN (Output Enable):
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.

9.3.25 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.

9.3.26 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.

9.3.27 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.

9.4 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 (i.e., 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.

10 CRC Error Detection and ARQ Error Correction

10.1 Overview

GMSL2 serial links incorporate 16-bit Cyclic Redundancy Check protection for error detection of control channel (including I^2C, 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 RGMII or 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.

10.2 CRC Operation

10.2.1 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 main and pass-through control channels are protected in both I^2C and UART modes.

The 16-bit packet CRC generator calculates following polynomial:

$$CRC16 = x^{16} + x^{15} + x^2 + 1$$

Note: This is the same polynomial used in the USB protocol for data packets.

10.2.2 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 throughinfo 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:

$$CRC32 = x^{32} + x^{31} + x^{28} + x^{25} + x^{24} + x^{23} + x^{21} + x^{18} + x^{11} + x^{8} + x^{7} + x^{6} + x^{5} + x^{3} + x^{1} + 1$$

10.2.3 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.

10.3 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 (e.g., corrupted or dropped packet), the ARQ on the transmit side does not receive confirmation of an error-free transmission and automatically retransmits the packet.

10.3.1 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.

10.3.1.1 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 41 demonstrates 'Go-Back-N' where N = 3.

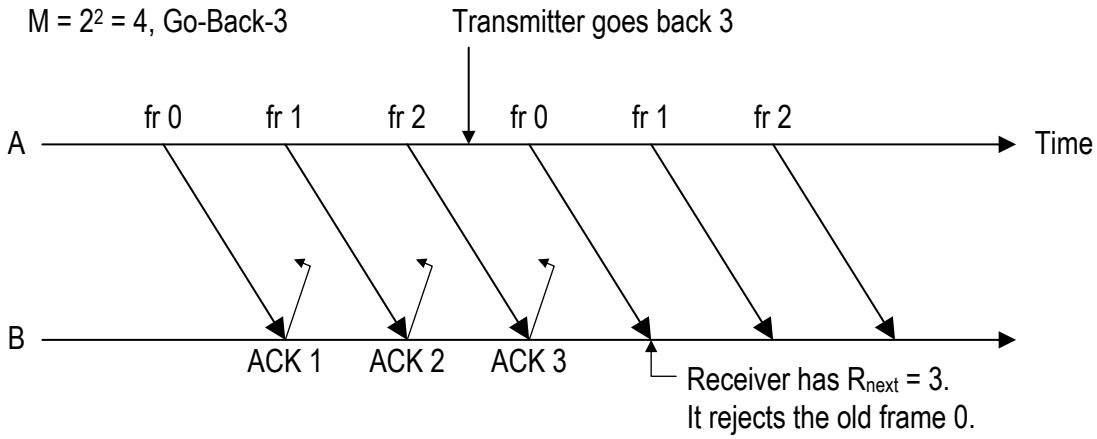


Figure 41. Operation of Go-Back-N ARQ with Go-Back-3

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.

10.3.1.2 ARQ Path

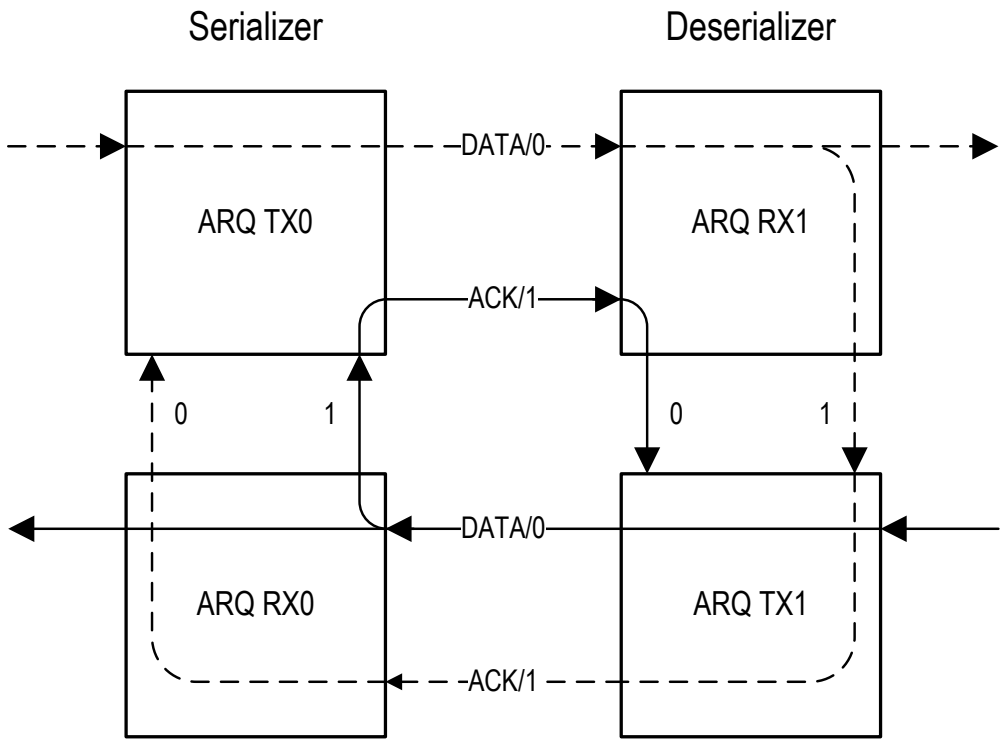


Figure 42. ARQ Path

The dashed line in the *Figure 42* 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.

10.4 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 except RGMII; ARQ is enabled by default for all supported packet types. ARQ does not support RGMII or video.

10.4.1 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.

10.4.1.1 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.

10.4.2 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, except RGMII, by default.

Table 37 lists the packet types that can enable/disable 16-bit CRC with their corresponding registers.

Table 36. 16-Bit CRC Registers

16-BIT CRC PACKET TYPE	REGISTER BLOCK*	REGISTER NAME
RGMII†	RGMII	TR0
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 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. Use of CRC on RGMII packets is discouraged as RGMII has error detection and retransmission mechanisms at a higher protocol level. Video Line CRC should be disabled if Video Pixel CRC is enabled.
‡ CRC and ARQ should not be disabled on the main 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.

10.4.2.1 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.

10.4.3 16-Bit ARQ Configuration

The ARQ configuration registers for each channel are listed in *Table 38*. By default, ARQ is enabled for all channels (except RGMII and Video Pixel), and error reporting to the ERRB pin is enabled.

Note: ARQ is not available for 16-bit Video Pixel CRC or RGMII.

Table 37. 16-Bit ARQ Registers

CONTROL CHANNEL PACKET TYPE	REGISTER BLOCK*	REGISTER NAMES
RGMII†	Not available	Not available
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 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 main 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.

10.4.3.1 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.

10.4.3.2 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.

11 Voltage Monitoring

11.1 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.

11.1.1 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), RGMII power (VDDIORG), 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 39*).

Table 39. 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 43*) 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 (e.g., VREG). Note that VDD/VREG pins are not typically monitored.

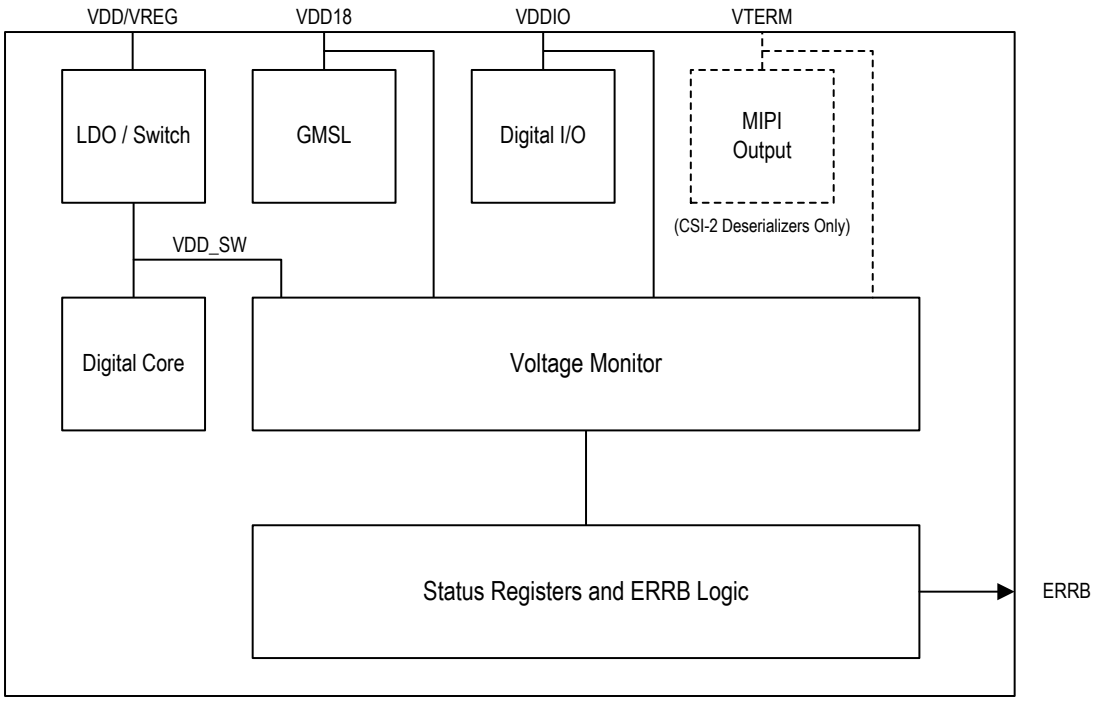


Figure 43. Voltage Monitor and Internal Power Pins

11.2 Operation

11.2.1 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 throughan internal LDO or series switches.

11.2.1.1 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 throughtwo 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 (i.e., the associated CMP_STATUS bit must be cleared first throughreading).

11.2.1.2 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.

11.2.2 VDD18 Monitoring Details

All GMSL2 devices include undervoltage monitoring of VDD18.

11.2.2.1 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.

11.2.3 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.

11.2.4 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.

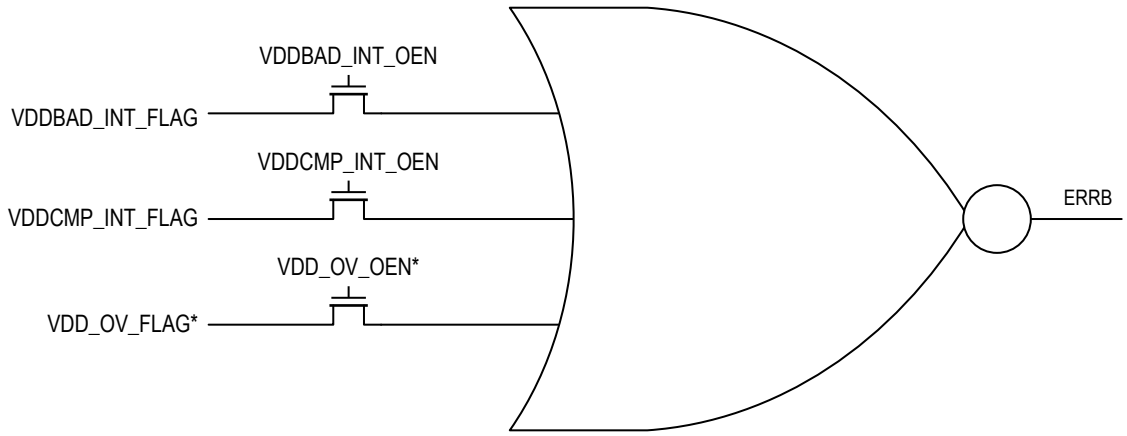
In the case of an undervoltage condition, the status register bit CMP_VTERM_STATUS is latched low. The error can be flagged using the VDDCMP_INT_FLAG, which can be configured to drive the fault condition to the ERRB pin. The error status is cleared by first reading the PWR0 register and then reading VDDCMP_INT_FLAG.

11.3 Error Reporting and Status

11.3.1 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 (i.e., the error condition asserts ERRB low). *Figure 44* 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).
    - For eDP/DP deserializers: VDD_OV_INT_OEN – asserts ERRB low when VDD_OV_INT_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 44. Voltage Monitoring ERRB Configuration

11.3.1.1 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.

12 Line Fault

12.1 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 (i.e., 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 throughregisters. 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.

12.2 Hardware Requirements

12.2.1 Coax Mode (Single-Ended)

The local side performs the line fault detection function. The local-side device requires a single 48.7kΩ resistor connected directly from an LMNx pin to the serial link. The remote side of the serial link requires a 49.9kΩ 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 40*.

Table 38. 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) 48.7kΩ to serial link	LMN0–3 (any may be used) 48.7kΩ 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 45* and *Figure 46*. 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.

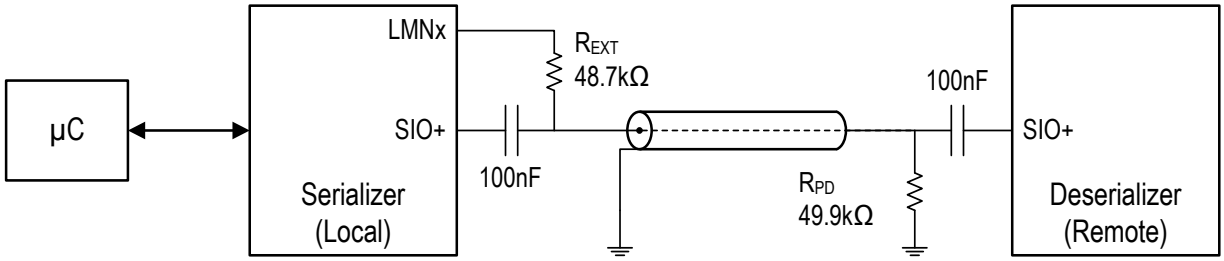


Figure 45. Line-Fault Configuration 1: Local-Side Serializer (Coax Mode)

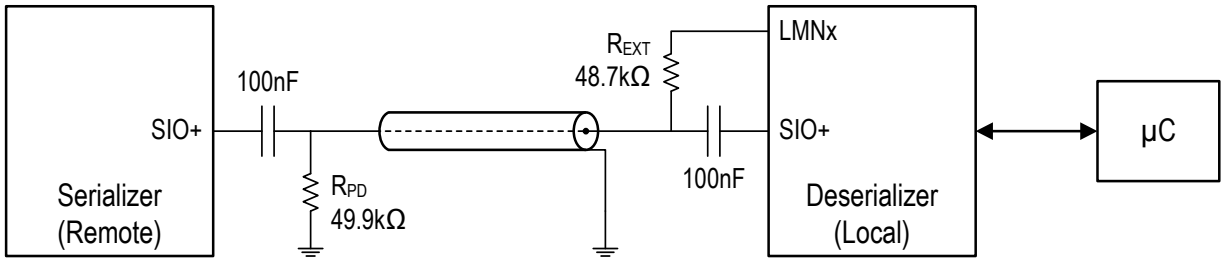


Figure 46. Line-Fault Configuration 2: Local-Side Deserializer (Coax Mode)

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.

12.3 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 41*.

Table 39. 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 42* 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 42.* Unused detectors (PU_LFx = 0) return “010” (i.e., the status code for ‘Normal Operation’).

Table 40. 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

12.3.1 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 42*). 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 45*. 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.

12.3.1.1 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.

12.3.1.2 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.

13 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 (e.g., PRBS Testing).

13.1 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 (e.g., video, RMII) are affected.

13.2 Configuration

To enable the ERRG, first set the ERRG parameters in register TX2 (*Table 43*). Then, enable the error generator in register TX1 (*Table 44*) for all devices except CSI-2 Quad Deserializers, which use register GMSL__x:TX1 (*Table 45*).

Note: To clear ERRB status after disabling ERRG mode (i.e., 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.

13.2.1 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 (i.e., 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 41. 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

13.2.1.1 ERRG Enable: All Device Families Except CSI-2 Quad Deserializers

Table 42. 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

13.2.1.2 ERRG Enable: CSI-2 Quad Deserializers

Table 43. 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

13.2.2 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):

$$PCLK = (Total\ Horizontal) \times (Total\ Vertical) \times (Frames\ Per\ Second)$$

$$Video\ Payload = PCLK \times 24\ (bpp)$$, must not exceed 5.2Gbps due to link overhead

$$MIPI\ Lane\ Rate = \frac{PCLK \times bpp}{\#\ of\ Lanes}$$, four possible lanes (see *Table 46* 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, 5.4Gbps/4 = 1.35Gbps 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 46* 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 44. 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

13.2.2.1 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.

14 MIPI D-PHY Deskew

14.1 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 47* (i.e., 'MIPI Link 1' and 'MIPI Link 2').

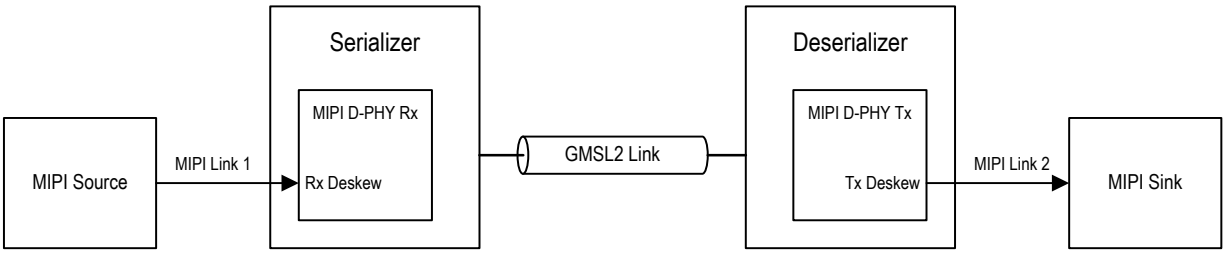


Figure 47. MIPI D-PHY Deskew Within a GMSL2 System

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.

14.2 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 throughthe 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 ±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 (i.e., 010101010101…) for a minimum duration of 2^{15} UI for the initial deskew calibration and 2^{10} UI for the periodic calibration(s).

Figure 48 depicts a high-speed MIPI data transmission in normal mode. *Figure 49* shows the deskew calibration process (see *Figure 50* for a detailed view).

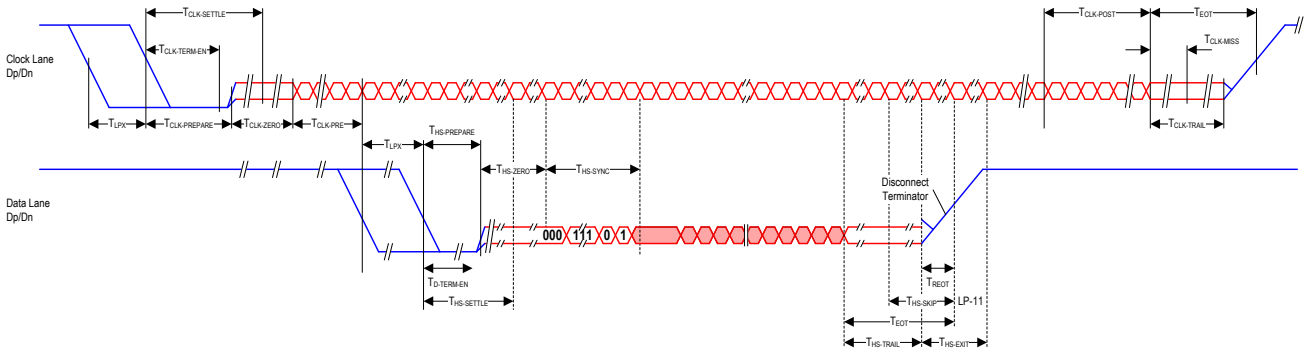


Figure 48. High-Speed Data Transmission in Normal Mode

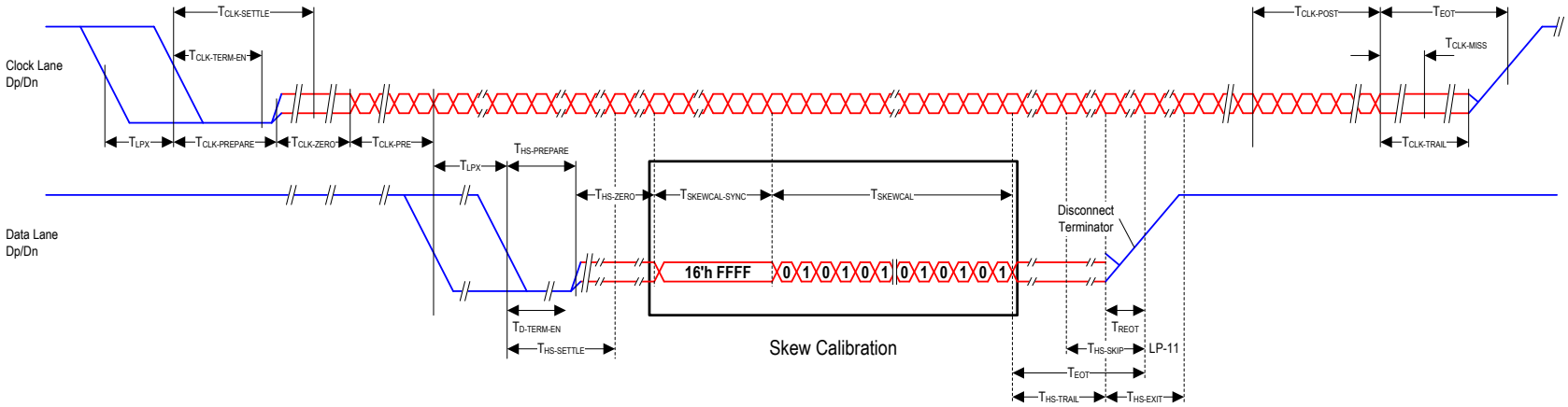


Figure 49. High-Speed Deskew Calibration

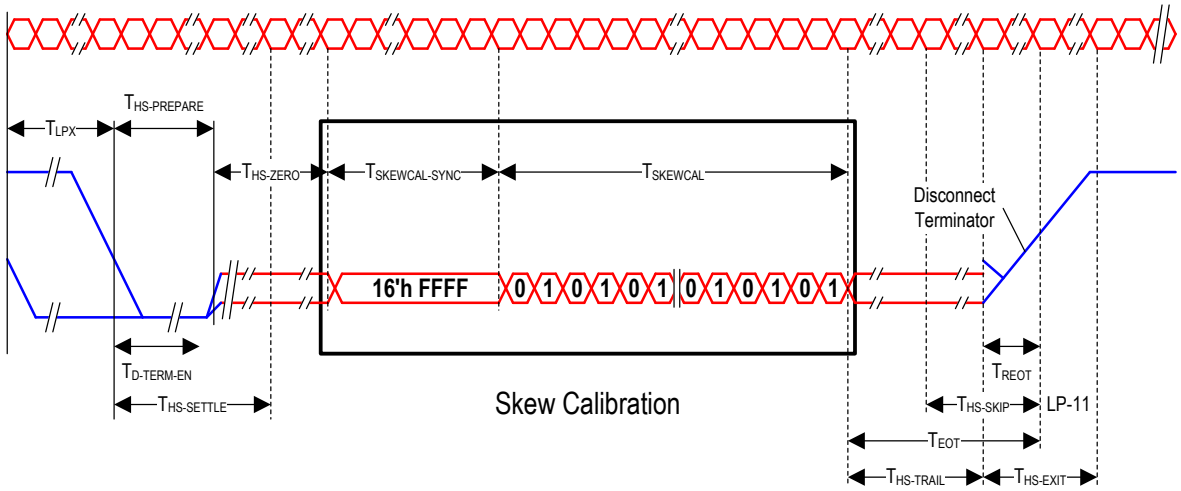


Figure 50. Skew Calibration (Detailed View)

14.2.1 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 throughregister 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.

14.3 Configuration

PHY and controller availability is package-dependent; refer to the device-specific data sheet(s) for more information.

14.3.1 D-PHY Serializers

Deskew calibration for GMSL2 serializers with D-PHY is enabled throughthe 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 45. 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 • 0: Deskew calibration disabled • 1: Deskew calibration enabled Bit 2: ctrl0_deskewen for port A • 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 • 0: Not received • 1: Received Bit 6: Deskew calibration pattern flag on data lane 1 • 0: Not received • 1: Received Bit 5: Deskew calibration failure flag on data lane 0 • 0: Default • 1: Failed Bit 4: Deskew calibration failure flag on data lane 1 • 0: Default • 1: Failed

14.3.2 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 (i.e., 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, i.e., 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, i.e., 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 46. 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. • 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

14.4 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.

14.4.1 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 (i.e., 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 ±0.4UI. 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.

14.4.2 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.

Revision History

REVISION NUMBER	REVISION DATE	DESCRIPTION
0	7/23	Initial release