Antmicro

SDI to MIPI CSI-2 video platform

2023-08-07

CONTENTS

CHAPTER ONE

INTRODUCTION

This documentation describes Antmicro's open source SDI to MIPI CSI-2 Bridge which is a Lattice Crosslink FPGA-based converter board between SDI, a popular standard used in integrated video cameras for e.g. broadcasting, and MIPI CSI-2, a mobile/embedded camera standard supported directly by a variery of embedded SoCs.

The SDI bridge is a completely open source device (including KiCad PCB design files, software and FPGA designs), part of Antmicro's broad open source hardware portfolio.

1.1 Structure

The documentation includes all the information necessary to use the SDI bridge in a practical scenario, with the following chapters:

- *Hardware setup* - description of the hardware, physical connections and configurations,
- *Building the FPGA design* - how to generate FPGA bitstreams for all available SDI signal configurations,
- *Preparing the software* - how to build and use a Linux BSP supporting the SDI bridge for several example embedded boards,
- *Running the video stream* - how to upload a bitstream and display the resulting video stream.

1.2 Custom engineering services

The SDI bridge is a development board and prototyping platform for building customized devices - Antmicro offers custom hardware, software and FPGA engineering services for this and other FPGA-based video processing use cases, including:

- integrating the SDI bridge with other embedded plaforms
- custom, integrated embedded boards with SDI input directly on the same PCB
- BSPs and drivers for embedded video systems
- advanced FPGA-based processing platforms
- FPGA video processing pipelines
- AI processing and pipelines

If you are interested in learning more, please reach out at contact@antmicro.com and let us know about your requirements.

CHAPTER

TWO

HARDWARE SETUP

A complete setup using the SDI-MIPI Bridge would typically consist of at least 3 devices:

- *SDI Bridge* - Antmicro's SDI to MIPI CSI-2 Bridge responsible for deserializing SDI data and packeting it with the MIPI CSI-2 protocol,
- *Transmitter* - a device transmitting SDI video data, connected to the SDI input port of the bridge,
- *Receiver* - a device receiving MIPI CSI-2 video data, connected to the MIPI CSI-2 output port of the bridge.

2.1 SDI Bridge

The core part of the setup is the SDI to MIPI CSI-2 Bridge that you can obtain from Antmicro's partner Capable Robot Components. You can also design and manufacture custom variants and integrated hardware through Antmicro's engineering services - please refer to the *Custom engineering services* section to learn more.

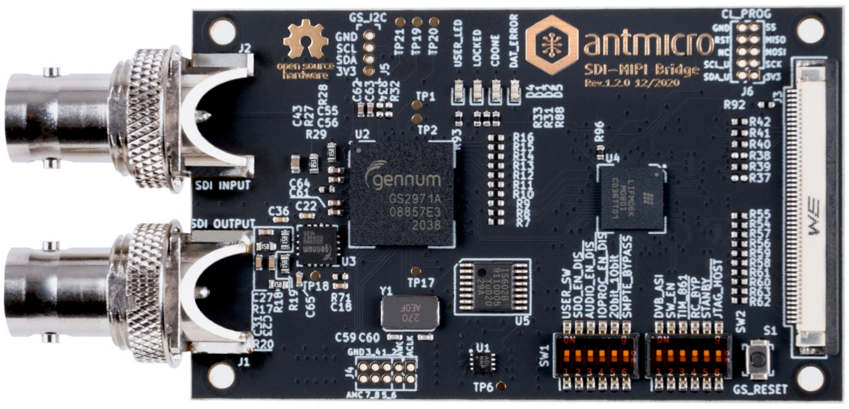

Figure 2.1: SDI to MIPI CSI-2 Bridge, rev. 1.2.0

2.1.1 Key features

- Operates at 2.97Gb/s, 2.97/1.001Gb/s, 1.485Gb/s, 1.485/1.001Gb/s and 270Mb/s
- Supports SMPTE ST 425 (Level A and Level B), SMPTE ST 424, SMPTE ST 292, SMPTE ST 259-C and DVB-ASI
- Integrated SDI adaptive cable equalizer and output loopback connector
- I^{2}S de-embedded audio output for up to 8 channels at 48kHz exposed on a 10-pin header
- Two 4-lane MIPI D-PHY transceivers at 6 Gbps per PHY exposed on a 50 pin FFC connector
- I^{2}C programming and communication interface for the CrossLink FPGA and Semtech SDI deserializer
- SPI interface for programming the CrossLink FPGA
- User button
- 2 LED indicators for user purposes

2.1.2 Architecture and operation

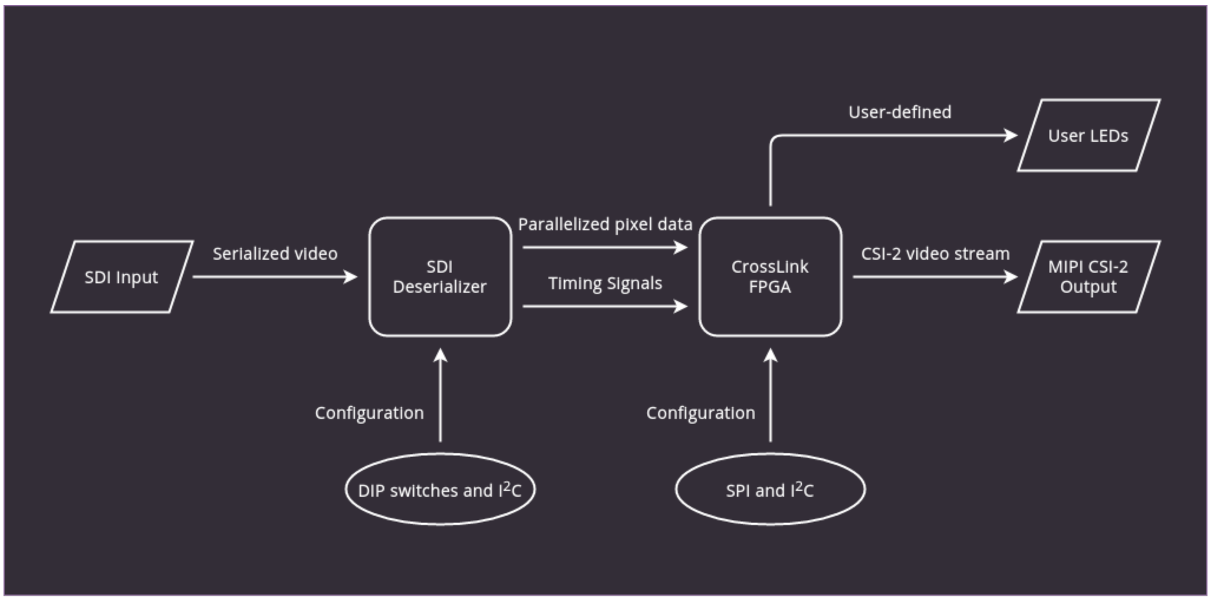

Figure 2.2: SDI to MIPI CSI-2 Bridge architecture

For video data, the board has an SDI input BNC connector and a MIPI CSI-2 output 50-pin connector, as well as an additional SDI loopback BNC connector.

SDI signal conversion is implemented with the Semtech GS2971A deserializer which passes the parallel 10-bit data to the Lattice CrossLink FPGA. The CrossLink FPGA family targets video bridging and processing applications, with hard MIPI D-PHY interfaces. In this application, it is used to convert parallel data to the MIPI CSI-2 format.

There is also a 10-pin connector on the board which provides audio data extracted from the SDI stream in I^{2}S format. Additionally, the board exposes SPI and I^{2}C interfaces.

SDI deserializer configuration

Due to the board's wide range of capabilities and possible support for various different configurations there are 12 resitors for configuring the SDI deserializer. A detailed description of their functions is available in the Semtech GS2971A documentation.

Below you can find a short description of each signal with its default setting which should be set in order to make the board work with the software/bitstream setup provided by Antmicro:

Name	Default	Notes
USER_SW	OFF	Connected to the PB6D input of the Crosslink FPGA.
SDO_EN	ON	Enables/disables the SDI loopback output of GS2971A, which is buffered and exposed on the 'SDI Output' BNC.
AUDIO_EN	ON	Enables/disables audio extraction fuctionality of GS2971A.
IOPROC_EN	ON	Enables/disables signal processing features of GS2971A like error correction and level conversion.
20bit_10bit	OFF	Used to select the output bus width. Must be set to low for proper operation on this board.
SMPTE_BYPASS	ON	When ON, GS2971A carries out SMPTE scrambling and I/O processing. When OFF, GS2971A operates in data pass-through mode.
DVB_ASI	OFF	Enables/disables the DVB-ASI mode of GS2971A.
SW_EN	OFF	When OFF, the default state of GS2971A's SW_EN pin is low. A rising edge (via switch or FPGA GPIO) will cause GS2971A to re-lock on the input video stream. Generally not needed unless the video source has been externally switched between two sources
TIM_861	OFF	When ON, GS2971A outputs CEA 861 timing signals (HSYNC/VSYNC/DE) instead of H:V:F digital timing signals.
RC_BYP	OFF	When ON, the serial digital output is the re-timed version of the serial input. When OFF, the serial digital output is simply the buffered version of the serial input, bypassing the GS2971A's internal reclocker.
STANDBY	OFF	When ON, GS2971A is placed in a power-saving mode. No data processing occurs, and the digital I/Os are powered down.
JTAG_HOST	OFF	When ON, the host interface port of GS2971A is configured for JTAG test. When OFF, GS2971A operates normally.

2.1.3 Board dimensions

2.2 Transmitter

The transmitter is a device (typically a camera) outputting data via the SDI interface. It must be compatible with one of the following available formats:

- 720p25Hz YUV422,
- 720p30Hz YUV422,
- 720p50Hz YUV422,

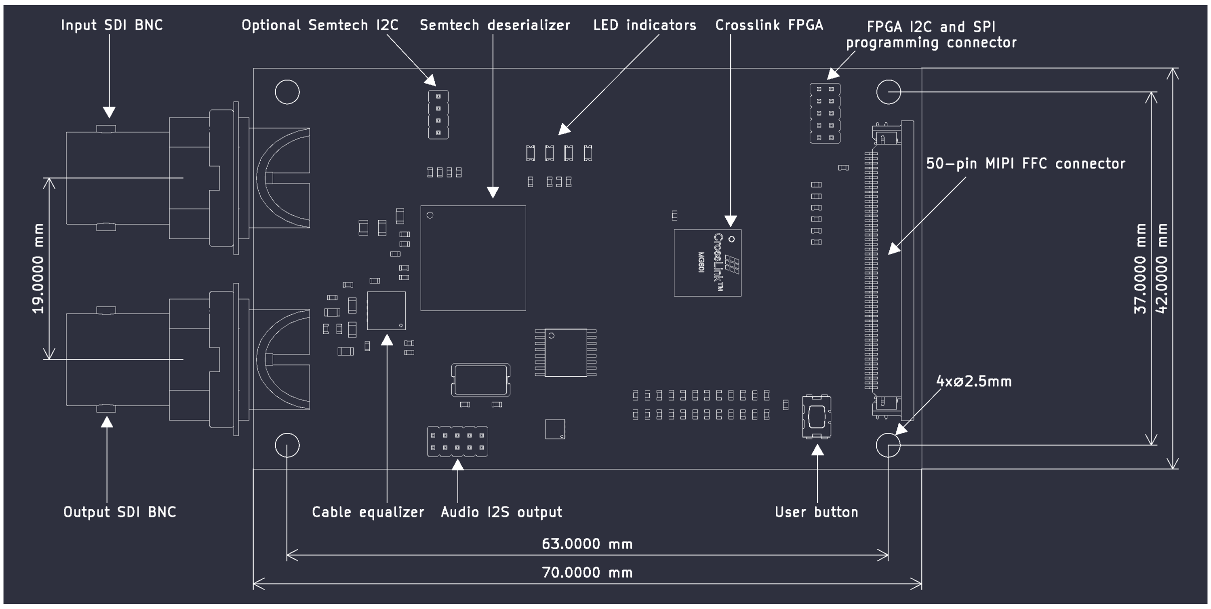

Figure 2.3: SDI Bridge dimensions - rev.1.3.1

- 720p60Hz YUV422,
- 1080p25Hz YUV422,
- 1080p30Hz YUV422,
- 1080p50Hz YUV422,
- 1080p60Hz YUV422.

Several input devices were tested with the SDI to MIPI Bridge, e.g.:

- Atomos Shogun Flame,
- Blackmagic HDMI to SDI Converter,
- Decimator HDMI/SDI 4K Cross Converter.

2.3 Receiver

The receiver is a device capable of receiving data through the MIPI CSI-2 interface. The *Software section* of this documentation covers support for 3 boards:

- Jetson Xavier NX
- Jetson TX2
- Raspberry Pi CM4

Antmicro provides sources for a Linux distribution configured for SDI Bridge support for these boards. If you need help with making the board work with other edge AI platforms or integrating the SDI interface into your own dedicated device, Antmicro offers *Custom engineering services*.

CHAPTER

THREE

BUILDING THE FPGA DESIGN

The CrossLink FPGA is responsible for converting data deserialized from SDI by the Semtech chip to MIPI CSI-2. The FPGA design consists of 2 main parts:

- *Top module* - module written in Python with Migen to instantiate and connect required FPGA components such as Oscillator and CMOS2DPHY converter.
- Lattice CMOS to D-PHY IP - converts standard parallel video data into CSI-2 byte packets.

3.1 CMOS to MIPI D-PHY

The Lattice CMOS to D-PHY IP core provides a bridging solution for converting parallelized pixel data from the deserializer into a MIPI CSI-2 video stream. The configuration of the IP core depends on the selected resolution and framerate. Therefore a different bistream needs to be used for some of the supported video formats.

3.1.1 Multiple variant support

Since the Lattice CMOS to D-PHY IP Core can not be dynamically reconfigured for different resolutions, the video parameters need to be specified at build time. The table below summarizes the parameters used for the supported resolutions:

Parameter	720p HD	1080p HD	1080p 3G	720p HD	1080p HD	1080p 3G
TXInterface	CSI-2	CSI-2	CSI-2	CSI-2	CSI-2	CSI-2
Num-berofTXLanes	2	2	2	4	4	4
TXLineRate [MHz]	594	594	1188	297	297	594
DataType	YUV422_8	YUV422_8	YUV422_8	YUV422_8	YUV422_8	YUV422_8
VirtualChannel	0	0	0	0	0	0
WordCount [Bytes]	2560	3840	3840	2560	3840	3840
PixelClock [MHz]	74.25	74.25	74.25	37.125	37.125	74.25

3.2 Setting up the environment

The design in the FPGA design repository requires the Lattice Diamond tool for generating the bitstream. For instructions on installing and using Diamond, please refer to the Lattice Diamond 3.12 Installation Guide for Linux.

Additionally, make sure that you have the sources of the CMOS to D-PHY 1.3 IP-Core installed using Diamond Clarity Designer. For more information, check the *Accomplishing Tasks with Clarity Designer* section from Clarity Designer User Manual (p. 16).

Once you have Diamond set up, install the Python prerequisites:

```
pip3 install -r requirements.txt
```

3.3 Building the bitstream

After you've prepared your environment, you can build the project with a `make`-based build flow by running:

```
make <video-format>-<lanes>
```

The output files will be generated in the `build/<video-format>-<lanes>` directory. For example, to produce a bitstream for the 1280x720 resolution with a 2-lane data bus, execute:

```
make 720p_hd-2lanes
```

The build files will be located in the `build/720p_hd-2lanes` directory.

There are 3 bitstream variants for different video formats, each of them can be built for either two or four MIPI CSI-2 lanes:

- `720p_hd` supports `720p25`, `720p30`, `720p50` and `720p60`,
- `1080p_hd` supports `1080p25` and `1080p30`,
- `1080p_3g` supports `1080p50` and `1080p60`.

CHAPTER

FOUR

PREPARING THE SOFTWARE

In order to be able to receive video from a SDI-MIPI Bridge, the receiver must include drivers for both the Semtech GS2971A deserializer and the FPGA manager for loading the bitstream. Antmicro provides a preconfigured Linux kernel for the following devices:

- Jetson Xavier NX
- Jetson TX2
- Raspberry Pi CM4

4.1 Building the BSP

To enable the SDI-MIPI Bridge support, you need to build the kernel with the required drivers and the device tree for the selected board. Choose a tab that matches your board and follow the instructions below.

Jetson Xavier NX

The steps to fetch, build and apply it to a stock L4T 32.4.4 package are listed below.

1. Obtain and extract the cross-compilation toolchain:

```
wget http://releases.linaro.org/components/toolchain/binaries/7.3-2018.05/aarch64-
↪linux-gnu/gcc-linaro-7.3.1-2018.05-x86_64_aarch64-linux-gnu.tar.xz
tar xf gcc-linaro-7.3.1-2018.05-x86_64_aarch64-linux-gnu.tar.xz
export PATH=$(pwd)/gcc-linaro-7.3.1-2018.05-x86_64_aarch64-linux-gnu/bin:$PATH
```

2. Obtain and set up the L4T-based host software:

```
wget https://developer.nvidia.com/embedded/L4T/r32_Release_v4.4/r32_Release_v4.4-
↪GMC3/T186/Tegra186_Linux_R32.4.4_aarch64.tbz2
tar xf Tegra186_Linux_R32.4.4_aarch64.tbz2
wget https://developer.nvidia.com/embedded/L4T/r32_Release_v4.4/r32_Release_v4.4-
↪GMC3/T186/Tegra_Linux_Sample-Root-Filesystem_R32.4.4_aarch64.tbz2
sudo tar xf Tegra_Linux_Sample-Root-Filesystem_R32.4.4_aarch64.tbz2 -C Linux_for_
↪Tegra/rootfs/
pushd Linux_for_Tegra
sudo ./apply_binaries.sh
sudo chown -R $USER rootfs/lib/modules
```

(continues on next page)

(continued from previous page)

```
sudo chown -R $USER rootfs/lib/firmware
popd
```

3. Obtain the kernel sources and choose the branch that matches your hardware:

```
git clone https://github.com/antmicro/sdi-mipi-bridge-linux
pushd sdi-mipi-bridge-linux
git checkout tags/v1.5          # JNB rev. >=1.5, data on two MIPI CSI-2 lanes
# git checkout tags/v1.4.x      # JNB rev. <1.5, data on two MIPI CSI-2 lanes
```

4. Build the kernel:

```
export ARCH=arm64
export CROSS_COMPILE=aarch64-linux-gnu-
make tegra_defconfig
make -j$(nproc)
```

5. Install the kernel image, modules and device tree blob:

```
cp ./arch/arm64/boot/Image ../Linux_for_Tegra/kernel/
cp ./arch/arm64/boot/dts/tegra194-p3668-all-p3509-0000.dtb ../Linux_for_Tegra/
↪kernel/dtb/
INSTALL_MOD_PATH=../Linux_for_Tegra/rootfs/ make modules_install
sudo chown -R root ../Linux_for_Tegra/rootfs/lib/modules
sudo chown -R root ../Linux_for_Tegra/rootfs/lib/firmware
popd
```

6. Copy the helper scripts from this repository to the root filesystem:

```
git clone https://github.com/antmicro/sdi-mipi-bridge
pushd sdi-mipi-bridge
cp -r scripts/* ../Linux_for_Tegra/rootfs/usr/local/bin/
popd
```

Jetson TX2

The steps to fetch, build and apply it to a stock L4T 32.4.4 package are listed below.

1. Obtain and extract the cross-compilation toolchain:

```
wget http://releases.linaro.org/components/toolchain/binaries/7.3-2018.05/aarch64-
↪linux-gnu/gcc-linaro-7.3.1-2018.05-x86_64_aarch64-linux-gnu.tar.xz
tar xf gcc-linaro-7.3.1-2018.05-x86_64_aarch64-linux-gnu.tar.xz
export PATH=$(pwd)/gcc-linaro-7.3.1-2018.05-x86_64_aarch64-linux-gnu/bin:$PATH
```

2. Obtain and set up the L4T-based host software:

```
wget https://developer.nvidia.com/embedded/L4T/r32_Release_v4.4/r32_Release_v4.4-
↪GMC3/T186/Tegra186_Linux_R32.4.4_aarch64.tbz2
tar xf Tegra186_Linux_R32.4.4_aarch64.tbz2
```

(continues on next page)

(continued from previous page)

```
wget https://developer.nvidia.com/embedded/L4T/r32_Release_v4.4/r32_Release_v4.4-
↪GMC3/T186/Tegra_Linux_Sample-Root-Filesystem_R32.4.4_aarch64.tbz2
sudo tar xf Tegra_Linux_Sample-Root-Filesystem_R32.4.4_aarch64.tbz2 -C Linux_for_
↪Tegra/rootfs/
pushd Linux_for_Tegra
sudo ./apply_binaries.sh
sudo chown -R $USER rootfs/lib/modules
sudo chown -R $USER rootfs/lib/firmware
popd
```

3. Obtain the kernel sources and choose the branch that matches your hardware:

```
git clone https://github.com/antmicro/sdi-mipi-bridge-linux
pushd sdi-mipi-bridge-linux
git checkout tags/v1.5          # JNB rev. >=1.5, data on two MIPI CSI-2 lanes
# git checkout tags/v1.4.x      # JNB rev. <1.5, data on two MIPI CSI-2 lanes
```

4. Build the kernel:

```
pushd sdi-mipi-bridge-linux
export ARCH=arm64
export CROSS_COMPILE=aarch64-linux-gnu-
make tegra_defconfig
make -j$(nproc)
```

5. Install the kernel image, modules and device tree blob:

```
cp ./arch/arm64/boot/Image ../Linux_for_Tegra/kernel/
cp ./arch/arm64/boot/dts/tegra186-quill-p3310-1000-a00-00-base.dtb ../Linux_for_
↪Tegra/kernel/dtb/
INSTALL_MOD_PATH=../Linux_for_Tegra/rootfs/ make modules_install
sudo chown -R root ../Linux_for_Tegra/rootfs/lib/modules
sudo chown -R root ../Linux_for_Tegra/rootfs/lib/firmware
popd
```

6. Copy the helper scripts from this repository to the root filesystem:

```
git clone https://github.com/antmicro/sdi-mipi-bridge
pushd sdi-mipi-bridge
cp -r scripts/* ../Linux_for_Tegra/rootfs/usr/local/bin/
popd
```

Raspberry Pi CM4

The Raspberry BSP is based on the Raspberry Pi distro, which is available on Raspberry Pi site. The kernel used on the device is based on the `rpi-5.15.y` branch of the Raspberry Pi fork of the Linux kernel.

1. Download Raspberry Pi OS (64-bit) with desktop from the Raspberry Pi archives download site:
2. Unpack the archive:

```
xz -d 2022-04-04-raspios-bullseye-arm64.img.xz
```

3. Write the SD card with the `.img` file:

```
dd if=2022-04-04-raspios-bullseye-arm64.img of=/dev/sdX bs=512 # where X is a␣
↪letter of a block device representing SD-Card
```

4. Download the ARM64 toolchain from Linaro release page and unpack it to your home directory.
5. Obtain Linux kernel sources:

```
git clone https://github.com/antmicro/sdi-mipi-bridge-linux-rpi
```

6. Build the Linux kernel:

```
cd sdi-mipi-bridge-linux-rpi
export PATH="${HOME}/gcc-linaro-7.3.1-2018.05-x86_64_aarch64-linux-gnu/bin":$
↪{PATH}
mkdir ../modules
export INSTALL_MOD_PATH=../modules/
export ARCH=arm64
export CROSS_COMPILE=aarch64-linux-gnu-
KERNEL=kernel8
make bcm2711_defconfig
make -j8
make -j8 modules_install
```

4.2 Uploading the BSP

Jetson Xavier NX

To flash the software on the device, put it in recovery mode, connect to the host PC with a USB cable and use the following command:

```
pushd Linux_for_Tegra
sudo ./flash.sh jetson-xavier-nx-devkit-emmc mmcblk0p1
popd
```

Jetson TX2

To flash the software on the device, put it in recovery mode, connect to the host PC with a USB cable and use the following command:

```
pushd Linux_for_Tegra
sudo ./flash.sh jetson-tx2 mmcblk0p1
popd
```

Raspberry Pi CM4

1. Copy the binaries to the SD card:

```
cd sdi-mipi-bridge-linux-rpi
sudo cp arch/arm64/boot/dts/broadcom/*.dtb <sd_card>/boot/
sudo cp arch/arm64/boot/dts/overlays/*.dtb* <sd_card>/boot/overlays/
sudo cp arch/arm64/boot/dts/overlays/README <sd_card>/boot/overlays/
sudo cp arch/arm64/boot/Image <sd_card>/boot/
sudo cp -r ../modules/lib/modules/* <sd_card>/lib/modules/
```

2. Additionally, the `<sd_card>/boot/config.txt` file on the SD Card needs to include the following lines:

```
kernel=Image
dtoverlay=dwc2,dr_mode=host
dtoverlay=disable-bt
dtoverlay=sdi-mipi-bridge-j5-cam1-4lane
```

3. Insert the SD card into your board and power it on.

CHAPTER

FIVE

RUNNING THE VIDEO STREAM

Streaming from the SDI-MIPI bridge to the chosen board is done through the MIPI CSI interface in a 2 or 4 lane configuration. The SDI to MIPI bridge supports only the YUV422 format in 1080p (30 or 60 FPS) or 720p (60 FPS) resolution.

5.1 Loading the bitstream

Different resolutions and framerates require different bitstreams. Therefore the FPGA manager subsystem in the Linux kernel is used to reprogram the FPGA as needed. In order to load the bitstream, first place it in the `/lib/firmware/sdi_bridge` directory as this is the path where the FPGA Manager looks for files to load. Once that is done, pass the bitstream's filename to the `load` node and wait until the process is finished.

```
echo "sdi_bridge/<bitstream_name.bit>" | sudo tee /sys/class/fpga_manager/fpga0/
↪load
```

When the bitstream is loaded, the `CDONE` LED should be blinking and the `USER_LED` should be turned on. The `USER_LED` is wired to the `tInit` output of the CMOS to D-PHY IP core which activates when the core has been initialized.

Note: The `DAT_ERROR` LED is turned on if there are no errors and turns off if data errors occur.

By default there are bitstreams for each supported video format located in the `/lib/firmware/sdi_bridge` directory. Default bitstreams are named with the following pattern:

```
<video_format>-<lanes>.bit
```

For example, to load the 1080p60 2-lanes variant:

```
echo "sdi_bridge/1080p60-2lanes.bit" | sudo tee /sys/class/fpga_manager/fpga0/load
```

Important: The lane count of the uploaded bitstream variant must match the device tree used in your Linux kernel.

5.2 Setting up the stream

If valid SDI signal is present on the SDI-MIPI bridge input, the `LOCKED` LED turns on. It indicates that the deserializer was able to acquire lock to the input signal. To be able to get the proper video data on the device, the video source has to stream in YUV422 format.

5.3 Testing the video stream

Note: If you see incorrect colors or there is no picture, try pressing the reset button on the SDI Bridge. If that doesn't help, reload the bitstream.

Jetson Xavier NX & TX2

To test the video stream, you can launch e.g. `gstreamer` as follows:

```
gst-launch-1.0 v4l2src device=/dev/video0 ! 'video/x-raw,width=<width>,height=
↪<height>' ! xvimagesink
```

The `<width>` and `<height>` parameters should match the currently used bitstream and SDI video resolution, either `1920x1080` or `1280x720`.

Raspberry Pi CM4

To test the video stream, you can use e.g. `qv4l2` app. There, you should set the expected pixel format and resolution according to the video source and loaded bitstream.

Note: Raspberry Pi currently supports only 4-lanes variants.

Also, the `v4l2-ctl` tool can be used to grab frames:

```
v4l2-ctl --stream-mmap --set-fmt-video=width=<width>,height=<height>,
↪pixelformat=VYUY --stream-count=0 --stream-to=/dev/null
```

The `<width>` and `<height>` parameters should match the currently used bitstream and SDI video format, either `1920x1080` or `1280x720`. To write captured frames to a file, set your target destination with `--stream-to=`. To get the exact number of frames grabbed, set `--stream-count=`.