AN13430

i.MX 8MP Dual ISI channels for single CSI

Rev. 0 — 25 October 2021

Application Note

Contents

1 Introduction......................................1
2 ISI...1
3 Application reference...................... 6
4 Revision history..............................7

1 Introduction

The i.MX 8M Plus family is a set of NXP products focused on machine learning applications. It combines state-of-art multimedia features with high-performance processing optimized for low-power consumption.

The ISI (Image Sensor Interface) is a simple camera interface that supports image processing and transfer via a bus commander interface for up to 2 cameras. As more users are intended to use the ISI module to process images, there are many use cases of the ISI module. This document describes how to connect dual ISI channels to one CSI, and output different format data separately.

The target audiences of the document are those who want:

1. Different format data through one camera sensor.
2. Customization of the ISI input source and output target.

NOTE

This experiment is implemented based on i.MX 8M plus, L5.10.35.

2 ISI

This chapter provides detailed information about how to change the ISI driver.

2.1 ISI overview

Image Sensor Interface (ISI) module interfaces up to 2 pixel link sources to obtain the image data for processing in its pipeline channels. Each pipeline processes the image line from a configured source and performs one or more functions that are configured by software, such as down scaling, color space conversion, de-interlacing, alpha insertion, cropping, and rotation (horizontal and vertical). The processed image is stored in programmable memory locations.

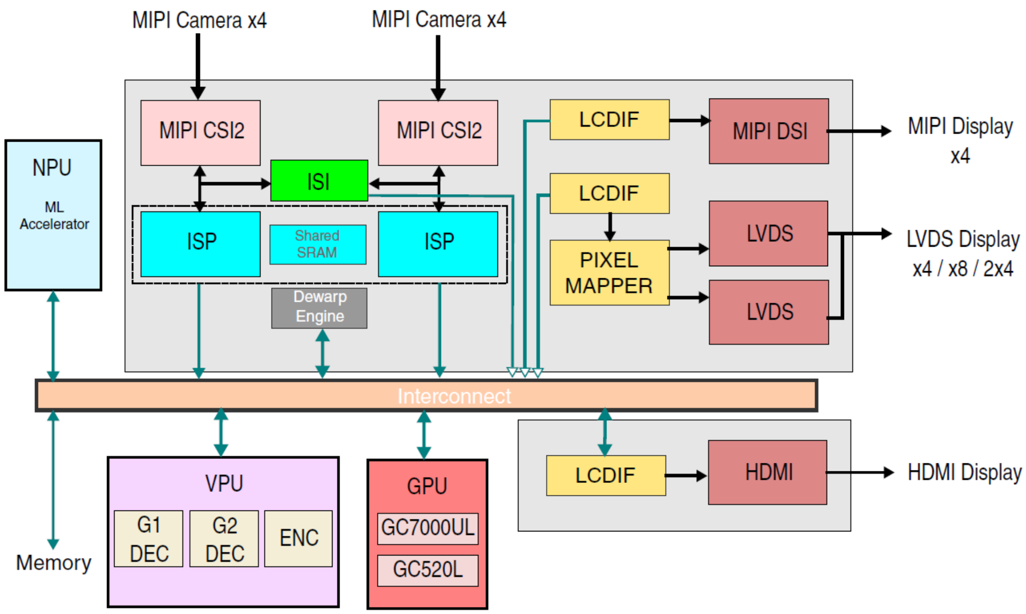

Figure 1. i.MX 8MP display, imaging, camera I/F diagram

2.2 Dual ISI channels solution

There is one ISI instance with 2 channels in i.MX 8M plus. They include the following features:

- Each processing pipeline or channel can be assigned to the same or different pixel input source.
- Several supported Pixel Formats when storing image into memory:
  - RAW8, RAW10, RAW12, RAW16
  - RGB888, BGR888, RGB565, RGB 10-bit, BGR 10-bit
  - YUV444, YUV422, YUV420 (8-bit, 10-bit, 12-bit) in planar or semi-planar formats
  - More formats are listed in the description of the FORMAT field in the `IMG_CTRL` register of the channel.
- Downscaling of the input image via Decimation and Bilinear filtering
- Decimation by 2, 4, or 8
- Further downscaling of the bilinear filter by 1.0 to 2.0 (fractional downscaling)
- Color Space Conversion (CSC)
- RGB, YUV, YCbCr
- User-defined color space matrix-based conversion

In this case, `isi_0` means ISI channel 0 and `isi_1` means ISI channel 1. `Isi_0` and `isi_1` are both connected to `csi0`. The application receives different data from `isi_0` and `isi_1` separately:

Table 1. Data received from **`isi_0`** and **`isi_1`**

	size	format
isi_0	1920*1080	NV12
isi_1	640*480	RGB

The following diagram illustrates the process:

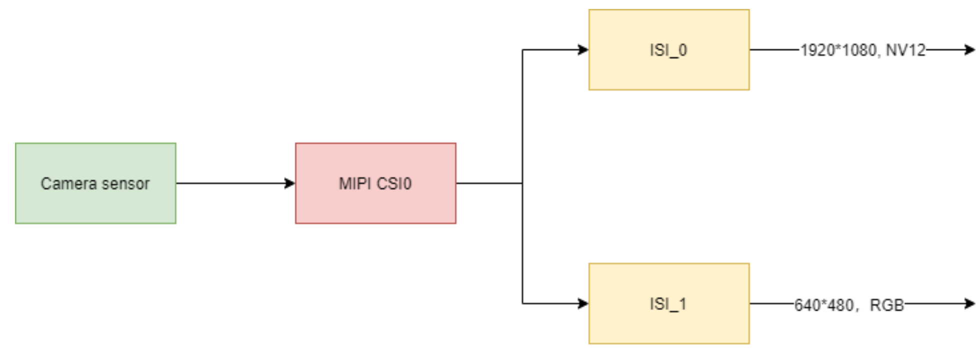

Figure 2. Architecture diagram

The application does not need to care about the place of connection of ISI. It only must control two device nodes separately, and receive the image from the nodes separately.

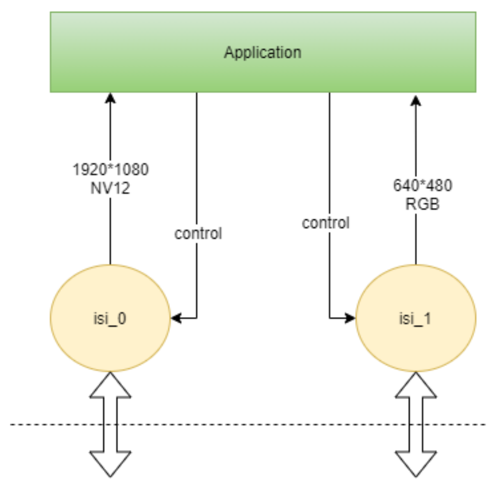

Figure 3. Application control diagram

The steps of implementing this solution are given in the following chapter.

2.3 ISI input source

First, the `isi_0` and `isi_1` input source should be changed to `csi0`. The ISI `interface = <Input VCx Output>` node defined in the device tree shows the mapping configurations, the meaning of each value is as follows:

Table 2. Meanings of the input, VCx, and output values

	0	1	2	3	4	5
Input	DC0	DC1	CSI0	CSI1	HDMI	MEM
VCx	VC0	VC1	VC2	VC3		
Output	DC0	DC1	MEM			

According to the table above, change the node to `interface = <2 0 2>` in the label `isi_0` and `isi_1` in `imx8mp.dtsi`. Open `isi_1` as it is default disabled in `imx8mp-evk.dts`.

After those changes in `dts`, there are two more device nodes in `dev`:

- `/dev/video3: isi_0` video device node.
- `/dev/video4: isi_1` video device node.

As a regular V4L2 device, it supports the V4L2 command.

```
root@imx8mpevk: # v4l2-ctl  -d /dev/video3 -D
Driver Info:
      Driver name      : mxc-isi-cap
      Card type        : mxc-isi-cap
      Bus info         : platform:32e00000.isi:cap_devic
      Driver version   : 5.10.35
```

2.4 Camera sensor control

Processed images are output from the pipeline and stored into the memory location specified by the software. There is source input format and destination output format in ISI. According to the two formats, ISI decides how to process the image (down scaling, color space conversion, and so on).

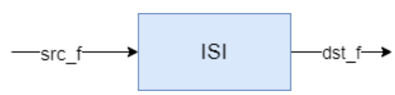

Figure 4. ISI input and output format

The destination format can be set by the application through the `ioctl` command of `VIDIOC_S_FMT`. The format, which has been passed down by the application, can be printed in the function `mxc_isi_cap_s_fmt_mplane` of the ISI driver. The format mainly contains image width/height and pixel format.

The source format comes from `VIDIOC_G_FMT` that is specified by the camera sensor. The code section is in the function `mxc_isi_source_fmt_init` of the ISI driver:

```
    memset(&src_fmt, 0, sizeof(src_fmt));
    src_fmt.pad = source_pad->index;
    src_fmt.which = V4L2_SUBDEV_FORMAT_ACTIVE;
    ret = v4l2_subdev_call(src_sd, pad, get_fmt, NULL, &src_fmt);
    if (ret < 0 && ret != -ENOIOCTLCMD) {
           v4l2_err(&isi_cap->sd, "get remote fmt fail!\n");
           return ret;
    }
```

In the common use case, `isi_0` and `isi_1` are connected to different CSI and request different data size, which can be output by a different camera sensor.

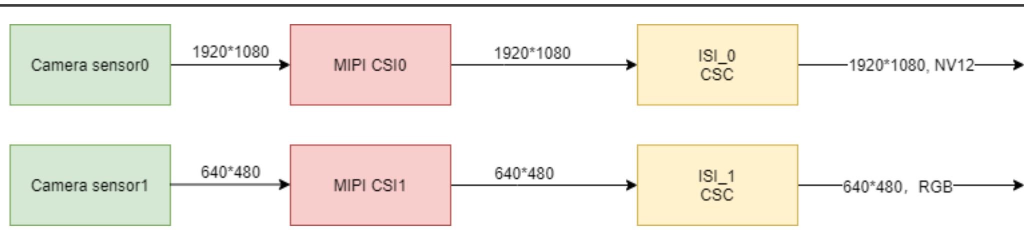

Figure 5. Common use case

But in this case, dual ISI channels are connected to one camera sensor and output different data size. Color Space Conversion in `isi_0` changes the data format to output, `isi_1` needs one more step to do the image scale.

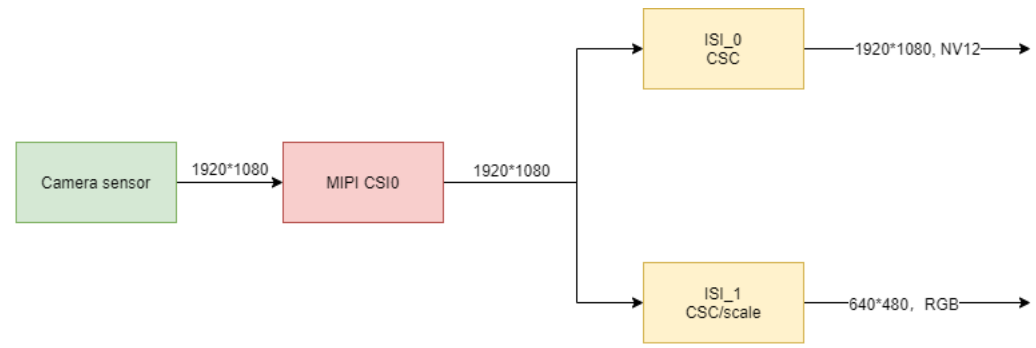

Figure 6. Dual ISI channel use case

The camera sensor output is 1920*1080, do the following steps to implement it:

1. The application opens the device node `/dev/video3` and streams on at the resolution of 1920*1080, the input and output resolution is the same (1920*1080). Record this resolution in the camera sensor driver.
2. The application opens the second device node `/dev/video4` and wants to stream on at the resolution of 640*480 (`VIDIOC_S_FMT`). `isi_1` gets the `src_fmt` through the `V4L2` command of `VIDIOC_G_FMT` in the function `mxc_isi_source_fmt_init`. This command passes to the camera sensor driver, and `src_fmt` is set in the camera sensor driver. If you leave it unchanged, the source resolution is the same as the destination resolution (640*480). When the `ioctl` command for `VIDIOC_G_FMT` is received, the camera sensor returns the previous resolution of 1920*1080 as the camera sensor is already streaming on at 1920*1080. The source resolution is 1920*1080, and the destination resolution is 640*480. ISI downscales the input image according to the destination resolution. Select the scale configurastions in the function `mxc_isi_channel_set_scaling`.

The static variables `pre_width` and `pre_height` in the camera sensor driver are used to record the previous resolution. They are initialized when the camera sensor is opened for the first time.

```
@@ -2282,17 +2286,32 @@ static int ov5640_get_fmt(struct v4l2_subdev *sd,
      if (format->which == V4L2_SUBDEV_FORMAT_TRY)
             fmt = v4l2_subdev_get_try_format(&sensor->sd, cfg,
                                             format->pad);
    else
+   {
            fmt = &sensor->fmt;
+           if(pre_width == 0)
+                   pre_width = fmt->width;
+           else
+                   fmt->width = pre_width;
+           if(pre_height == 0)
+                   pre_height = fmt->height;
+           else
+                   fmt->height = pre_height;
+     }
```

2.5 Stream control

If you want to control `isi_0` and `isi_1` separately, but they are both connected to one camera and one CSI, make the following changes:

1. When the `isi_0` stream is already enabled, the stream on `isi_1` fails. The camera sensor driver prevents `set_fmt` when it detects this sensor is already enabled. Opening one camera sensor twice is not under the considered scope, unlock this limitation.

```
 @@ -2367,12 +2367,13 @@ static int ov5640_set_fmt(struct v4l2_subdev *sd,
               return -EINVAL;
        mutex_lock(&sensor->lock);
+       /*
        if (sensor->streaming) {
               ret = -EBUSY;
               goto out;
        }
+      */
```

2. When you close one `/dev/videoX`, the other also stops, because the camera sensor and CSI stopped working. To avoid this situation, add the close counter mechanism, stop camera sensor, and then close CSI.

To prevent the camera sensor from early stop, add the code section to the camera sensor driver:

```
 @@ -3028,7 +3030,16 @@ static int ov5640_s_stream(struct v4l2_subdev *sd, int enable)
        int ret = 0;
        mutex_lock(&sensor->lock);
+       if (!enable)
+       {
+              open_cnt--;
+              if(open_cnt > 0)
+                     goto out;
+       }
        if (sensor->streaming == !enable) {
@@ -3062,6 +3073,12 @@ static int ov5640_s_stream(struct v4l2_subdev *sd, int enable)
             if (!ret)
                    sensor->streaming = enable;
      }
+     if (enable)
+            open_cnt++;
```

If any ISI is opened, CSI must not stop. To prevent early stop, add the similar code section to `mipi_csis_s_stream` of the CSI driver in `imx8-mipi-csi2-sam.c` .

3 Application reference

This chapter gives information about application settings and buffers alignment.

3.1 Application setting

There are two video device nodes in `dev`. The application should control those nodes respectively.

The destination format is set by the application through `VIDIOC_S_FMT`:

1920*1080 @ NV12

```
      format.type = V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE;
      format.fmt.pix_mp.width = 1920;
      format.fmt.pix_mp.height = 1080;
      format.fmt.pix_mp.pixelformat = V4L2_PIX_FMT_NV12; //2 planes
      format.fmt.pix_mp.field = V4L2_FIELD_NONE;
```

640*480 @ RGB24

```
format.type = V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE;
format.fmt.pix_mp.width = 640;
format.fmt.pix_mp.height = 480;
format.fmt.pix_mp.pixelformat = V4L2_PIX_FMT_RGB24;
format.fmt.pix_mp.field = V4L2_FIELD_NONE;
```

3.2 Two ISI buffers synchronization

As the application controls ISI respectively, it makes the application receive buffer unsynced. There are two flags to help align buffers:

1. Use `buffer.index` from `v4l2_buffer` to align buffers.
2. Use `buffer.timestamp` from `v4l2_buffer` to align buffers.

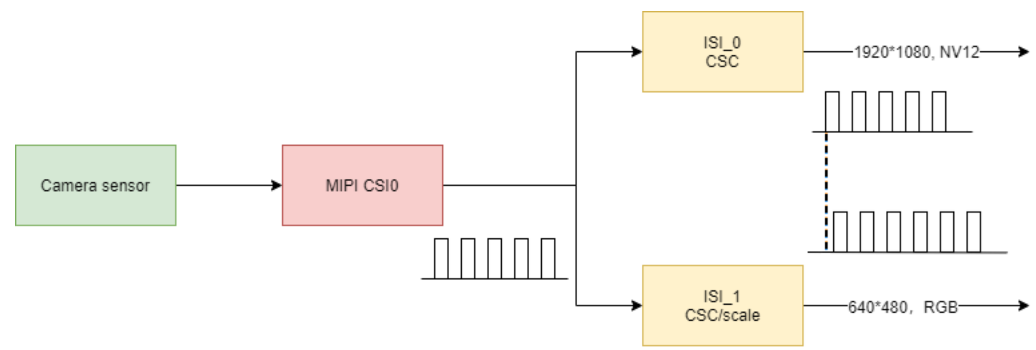

Figure 7. Buffer sequence

The application can dump output to check whether the changes work.

After the application received buffer sequence:

1. Image in the nv12 format, which has two planes, can be saved in the YUV format.
2. Image in the RGB format can be saved to the BMP format after adding the BMP format header.

4 Revision history

Table 3. Revision history

Revision number	Date	Substantive changes
0	25 October 2021	Initial release

Date of release: 25 October 2021

Document identifier: AN13430