i.MX6 IPU TVIN Application Note

This document intends to describe the i.MX6 IPU TVIN feature.

Contents

1 Overview

This document introduced the iMX6 IPU based TVIN solution and some common issues.

Below is the basic TVIN system block diagram, the interface between TVIN chip and iMX6 can be parallel CSI or MIPI-CSI2:

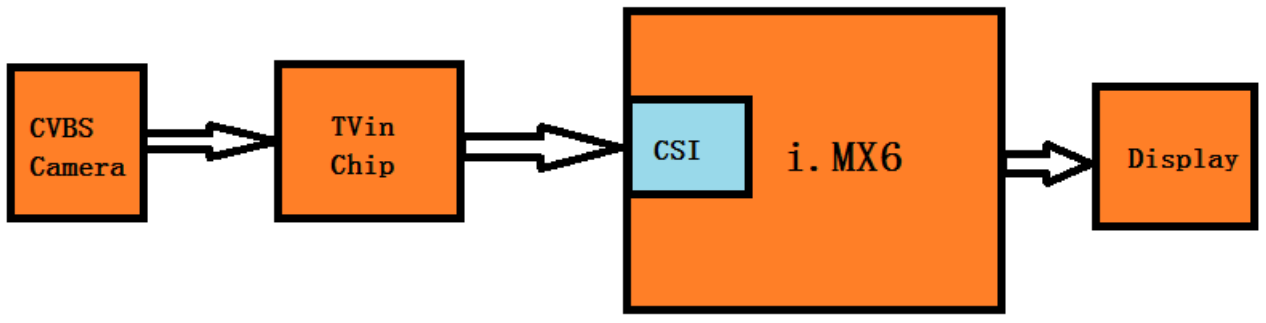

Figure 1-1. Parallel CSI TVIN Block Diagram

2 Parallel CSI TVIN

For parallel CSI interface, based on different interface width, the TVIN can support two kinds of protocol: BT.656 and BT.1120.

2.1 BT.656

For BT.656 interface, although the protocol can support up to 10 bits interface, we suggest to use 8 bits interface only. Because in IPU internal, each Y/U/V will be stored as 8 bits only, so for 10 bits YUV data, IPU can't be used to post-process them.

Hardware interface (1 clock + 8 data lines):

IPU_CSI_PIXCLK

IPU_CSI_DATA19 (MSB)

IPU_CSI_DATA18

IPU_CSI_DATA17

IPU_CSI_DATA16

IPU_CSI_DATA15

IPU_CSI_DATA14

IPU_CSI_DATA13

IPU_CSI_DATA12 (LSB)

Note: If the camera output is 10 bits BT.656, please just connect the 8 bits MSB to IPU CSI port, DATA19 ~ DATA12.

2.1.1 BT.656 Interlace Setting

NTSC (720*480i) and PAL (720*576i) format are used as the example. Customized format can also be supported with correct EAV/SAV and resolution setting.

CSI register setting:

- IPU_CSI_SENS_CONF

  CSI_EXT_VSYNC = 0x0, internal VSYNC mode.

  CSI_DATA_WIDTH = 0x1, 8 bits per color.

  CSI_SENS_DATA_FORMAT = 0x2, UYVY format.

  CSI_SENS_PRTCL = 0x3, BT.656 interlaced mode.

- IPU_CSI_SENS_FRM_SIZE, it is the whole frame size, include blank data.

  For PAL, it is 720x625, so the register value = 0x027002CF.

  For NTSC, it is 720x525, so the register value = 0x020C02CF.

- IPU_CSI_ACT_FRM_SIZE, it is the active video frame size.

  For PAL, it is 720x576, so the register value = 0x023F02CF.

  For NTSC, it is 720*480, so the register value = 0x01DF02CF.

- IPU_CSI_OUT_FRM_CTRL, this register can be used to skip some pixel in each line or skip some lines in each frame.

  For PAL, the register is set to 0.

  For NTSC, it depends on TVIN chip, for BT.656-4 and above version TVIN chip, CSI_VSC is 3; for BT.656-3 TVIN chip, CSI_VSC is 13. Details, please reference to 2.6 The CSI_VSC setting for NTSC.

- IPU_CSI_CCIR_CODE_1, it is for field 0 EAV/SAV setting:

  For PAL, 0x40596

  For NTSC, 0xD07DF

- IPU_CSI_CCIR_CODE_2, it is for field 1 EAV/SAV setting:

  For PAL, 0xD07DF

  For NTSC, 0x40596

- IPU_CSI_CCIR_CODE_3, CCIR pre-command code:

  0xFF0000

Note: For NTSC, in Linux BSP CCIR_CODE_1 and CCIR_CODE_2 setting had been switched. Please reference to 2.5 CCIR Register and EAV/SAV.

IDMAC Setting: scan order is set to interlaced, and stride is set to two lines; reference to Figure 2-1, it will start to capture data line 1 of field 0 (Based on IPU_CSI_CCIR_CODE_1 register setting), and save it to memory buffer's line 1, after finished one line, the buffer point will jump two lines and point to line 3 of the memory buffer. After one field data is captured, the buffer point will go to line 2 of the memory buffer to store field 1 data (Based on IPU_CSI_CCIR_CODE_2 register setting).

So after data stores into memory, it will be filled as followed:

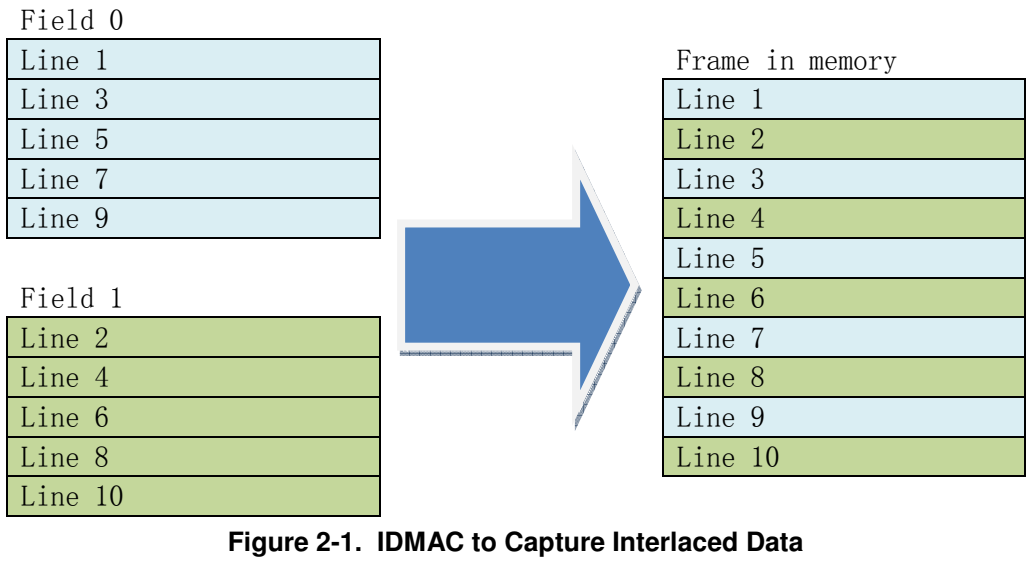

Figure 2-1. IDMAC to Capture Interlaced Data

2.1.2 BT.656 Progressive Setting

480P and 576P format are used as the example. Customized format can also be supported with correct EAV/SAV and resolution setting.

CSI register setting:

- IPU_CSI_SENS_CONF

  CSI_EXT_VSYNC = 0x0, internal VSYNC mode.

  CSI_DATA_WIDTH = 0x1, 8 bits per color.

  CSI_SENS_DATA_FORMAT = 0x2, UYVY format.

  CSI_SENS_PRTCL = 0x2, BT.656 progressive mode.

- IPU_CSI_SENS_FRM_SIZE, it is the whole frame size, include blank data.

  For 576P, it is 720x625, so the register value = 0x027002CF.

  For 480P, it is 720x525, so the register value = 0x020C02CF.

- IPU_CSI_ACT_FRM_SIZE, it is the active video frame size.

  For 576P, it is 720x576, so the register value = 0x023F02CF.

  For 480P, it is 720*480, so the register value = 0x01DF02CF.

- IPU_CSI_OUT_FRM_CTRL, this register can be used to skip some pixel in each line or skip some lines in each frame.

  The register is set to 0.

- IPU_CSI_CCIR_CODE_1, it is for frame EAV/SAV setting:

  0x40010

- IPU_CSI_CCIR_CODE_2, not used for progressive mode:

  0x0

- IPU_CSI_CCIR_CODE_3, CCIR pre-command code:

  0xFF0000

Note: in Linux BSP, IPU_CSI_CCIR_CODE_1 register is set to 0x40030, it can also work, details reference to 2.5.2 CCIR Register Setting for Progressive Signal.

IDMAC Setting: scan order is set to progressive, and stride is set to one line; so the capture memory buffer is filled line by line.

2.2 BT.1120

For BT.1120 interface, although the protocol can support up to 20 bits interface, we suggest to use 16 bits interface, each Y/U/V is 8 bits. Because in IPU internal, each Y/U/V will be stored as 8 bits only, so for 10 bits YUV data, IPU can't be used to post-process them.

Hardware interface (1 clock + 16 data lines):

IPU_CSI_PIXCLK	
IPU_CSI_DATA19	(MSB)
IPU_CSI_DATA18	
IPU_CSI_DATA17	
IPU_CSI_DATA16	
IPU_CSI_DATA15	
IPU_CSI_DATA14	
IPU_CSI_DATA13	
IPU_CSI_DATA12	(LSB)

IPU_CSI_DATA09 (MSB)

IPU_CSI_DATA08

IPU_CSI_DATA07

IPU_CSI_DATA06

IPU_CSI_DATA05

IPU_CSI_DATA04

IPU_CSI_DATA03

IPU_CSI_DATA02 (LSB)

Note: If the camera output is 20 bits BT.1120, please just connect two 8 bits Y and UV MSB to IPU CSI port, DATA19 ~ DATA12 and DATA9 ~ DATA2.

2.2.1 BT.1120 Interlace Setting

NTSC (720*480i) and PAL (720*576i) format are used as the example. Customized format can also be supported with correct EAV/SAV and resolution setting.

CSI register setting:

- IPU_CSI_SENS_CONF

  CSI_EXT_VSYNC = 0x0, internal VSYNC mode.

  CSI_DATA_WIDTH = 0x1, 8 bits per color.

  CSI_SENS_DATA_FORMAT = 0x2, UYVY format.

  CSI_SENS_PRTCL = 0x7, BT.1120 SDR interlaced mode. Or CSI_SENS_PRTCL = 0x6, BT.1120 DDR interlaced mode.

- IPU_CSI_SENS_FRM_SIZE, it is the whole frame size, include blank data.

  For PAL, it is 720x625, so the register value = 0x027002CF.

  For NTSC, it is 720x525, so the register value = 0x020C02CF.

- IPU_CSI_ACT_FRM_SIZE, it is the active video frame size.

  For PAL, it is 720x576, so the register value = 0x023F02CF.

  For NTSC, it is 720*480, so the register value = 0x01DF02CF.

- IPU_CSI_OUT_FRM_CTRL, this register can be used to skip some pixel in each line or skip some lines in each frame.

  For PAL, the register is set to 0.

  For NTSC, it depends on TVIN chip, for BT.656-4 and above version TVIN chip, CSI_VSC is 3; for BT.656-3 TVIN chip, CSI_VSC is 13. Details, please reference to 2.6 The CSI_VSC setting for NTSC.

- IPU_CSI_CCIR_CODE_1, it is for field 0 EAV/SAV setting:

  For PAL, 0x40596

  For NTSC, 0xD07DF

- IPU_CSI_CCIR_CODE_2, it is for field 1 EAV/SAV setting:

  For PAL, 0xD07DF

  For NTSC, 0x40596

- IPU_CSI_CCIR_CODE_3, CCIR pre-command code:

  0xFF0000

Note 1: For NTSC, in Linux BSP CCIR_CODE_1 and CCIR_CODE_2 setting had been switched. Please reference to 2.5 CCIR Register and EAV/SAV.

Note 2: For BT1120 mode, the CSI_DATA_WIDTH is the bit width of each Y/U/V, so it is 8, not 16.

IDMAC Setting: same as BT.656 interlaced mode.

2.2.2 BT.1120 Progressive Setting

480P and 576P format are used as the example. Customized format can also be supported with correct EAV/SAV and resolution setting.

CSI register setting:

- IPU_CSI_SENS_CONF

  CSI_EXT_VSYNC = 0x0, internal VSYNC mode.

  CSI_DATA_WIDTH = 0x1, 8 bits per color.

  CSI_SENS_DATA_FORMAT = 0x2, UYVY format.

  CSI_SENS_PRTCL = 0x5, BT.1120 SDR progressive mode; CSI_SENS_PRTCL = 0x4, BT.1120 DDR progressive mode.

- IPU_CSI_SENS_FRM_SIZE, it is the whole frame size, include blank data.

  For 576P, it is 720x625, so the register value = 0x027002CF.

  For 480P, it is 720x525, so the register value = 0x020C02CF.

- IPU_CSI_ACT_FRM_SIZE, it is the active video frame size.

  For 576P, it is 720x576, so the register value = 0x023F02CF.

  For 480P, it is 720*480, so the register value = 0x01DF02CF.

- IPU_CSI_OUT_FRM_CTRL, this register can be used to skip some pixel in each line or skip some lines in each frame.

  The register is set to 0.

- IPU_CSI_CCIR_CODE_1, it is for frame EAV/SAV setting:

  0x40010

- IPU_CSI_CCIR_CODE_2, not used for progressive mode:

  0x0

- IPU_CSI_CCIR_CODE_3, CCIR pre-command code:

  0xFF0000

Note 1: in Linux BSP, IPU_CSI_CCIR_CODE_1 register is set to 0x40030, it can also work, details reference to 2.5.2 CCIR Register Setting for Progressive Signal.

Note 2: For BT1120 mode, the CSI_DATA_WIDTH is for the bit width of each Y/U/V, so it is 8, not 16.

IDMAC Setting: same as BT.656 progressive mode.

2.3 The CSI_DATA_EN pin

Although the DATA_EN pin is not needed for BT.656 and BT.1120 mode, but if software had set the CSI_DATA_EN function for that pin, and it is internal pulled down or external pulled down. Then with the default CSI_DATA_EN_POL setting (0: IPP_IND_SENSB_DATA_EN is directly applied to internal circuitry) in IPU_CSI_SENS_CONF register, the CSI_DATA_EN will be always invalid, in this case IPU CSI can't receive video data.

So from software side, there should be no code to enable the CSI_DATA_EN pin function in BT.656 and BT.1120 mode.

2.4 Internal and External VSYNC Mode

CSI_EXT_VSYNC in IPU_CSI_SENS_CONF register is used to identify the VSYNC mode, for BT.656 and BT.1120 mode, it should always be 0, internal VSYNC mode, because they don't need the external VSYNC pin, the timing information is embedded on data line: EAV/SAV.

But for iMX6 SabreAuto board Linux BSP, with the adv7180 TVIN chip, both internal and external VSYNC mode can work fine. Because on that board, it had connected CSI_VSYNC pin.

For internal VSYNC mode, the CSI captures the timing signal from data line with EAV/SAV. In the driver, such as adv7180.c, function ioctl_g_ifparm():

"p->u.bt656.bt_sync_correct = 0;"

The above code tells v4l2 capture driver to use internal VSYNC mode.

For external VSYNC mode, the CSI captures the timing signal from CSI_VSYNC pin. In the driver, such as adv7180.c, function ioctl_g_ifparm():

"p->u.bt656.bt_sync_correct = 1;"

The above code tells v4l2 capture driver to use external VSYNC mode.

2.5 CCIR Register and EAV/SAV

From BT.656 and BT.1120 protocol, the EAV/SAV format is followed, it defined 10 bits, but only MSB 8 bits are used now.

The EAV/SAV signal are only transferred on 8 bits MSB of 16 bits BT.1120 interface.

Data bit number	First word (FF)	Second word (00)	Third word (00)	Fourth word (XY)
9 (MSB)	1	0	0	1
8	1	0	0	F
7	1	0	0	V
6	1	0	0	H
5	1	0	0	P3
4	1	0	0	P2
3	1	0	0	P1
2	1	0	0	P0
1	1	0	0	0
0	1	0	0	0

Table 2-1. Video Timing Reference Code

F = 0 during field 0; F = 1 during field 1.

V = 1 during field blanking; V = 0 elsewhere.

H = 0 in SAV; H = 1 in EAV.

P0, P1, P2, P3: protection bits.

2.5.1 CCIR Register Setting for Interlaced Signal

The BT.656-4 and above version NTSC frame format is as followed: there are total 487 lines of active video, field 0 has 244 lines and field 1 has 243 lines.

If it is BT.656-3 NTSC frame, then there are total 507 lines of active video, field 0 has 254 lines and field 1 has 253 lines.

So field 1 has one more line than field 0, to iMX6 CSI, to avoid the scroll issue, it needs receive field 1 first. That's why the software need switch the CCIR_CODE_1 and CCIR_CODE_2 register setting.

For PAL frame, there is no such difference, both of the two fields have 288 lines of active video.

Table 2-2 is the BT.656-4 NTSC frame format, and Table 2-3 is the PAL frame format:

Field 0 - First Vertical Blanking(Top) - Repeat for 16 lines															
EAV Code				Blanking Video				SAV Code				Active Video			
FF	0	0	B6	80	10	80	10	FF	0	0	AB	80	10	80	10
Repeat 1 time				Repeat 67 times				Repeat 1 time				Repeat 360 times			
Field 0 - Active Video - Repeat for 244 lines															
EAV Code				Blanking Video				SAV Code				Active Video			
FF	0	0	9D	80	10	80	10	FF	0	0	80	xx	xx	xx	xx
Repeat 1 time				Repeat 67 times				Repeat 1 time				Repeat 360 times			
Field 0 - Second Vertical Blanking(Bottom) - Repeat for 2 lines															
EAV Code				Blanking Video				SAV Code				Active Video			
FF	0	0	B6	80	10	80	10	FF	0	0	AB	80	10	80	10
Repeat 1 time				Repeat 67 times				Repeat 1 time				Repeat 360 times			
Field 1 - First Vertical Blanking(Top) - Repeat for 17 lines															
EAV Code				Blanking Video				SAV Code				Active Video			
FF	0	0	F1	80	10	80	10	FF	0	0	EC	80	10	80	10
Repeat 1 time				Repeat 67 times				Repeat 1 time				Repeat 360 times			
Field 1 - Active Video - Repeat for 243 lines															
EAV Code				Blanking Video				SAV Code				Active Video			
FF	0	0	DA	80	10	80	10	FF	0	0	C7	xx	xx	xx	xx
Repeat 1 time				Repeat 67 times				Repeat 1 time				Repeat 360 times			
Field 1 - Second Vertical Blanking(Bottom) - Repeat for 3 lines															
EAV Code				Blanking Video				SAV Code				Active Video			
FF	0	0	F1	80	10	80	10	FF	0	0	EC	80	10	80	10
Repeat 1 time				Repeat 67 times				Repeat 1 time				Repeat 360 times			

Table 2-2. NTSC Frame

Field 0 - First Vertical Blanking(Top) - Repeat for 22 lines															
EAV Code				Blanking Video				SAV Code				Active Video			
FF	0	0	B6	80	10	80	10	FF	0	0	AB	80	10	80	10
Repeat 1 time				Repeat 70 times				Repeat 1 time				Repeat 360 times			
Field 0 - Active Video - Repeat for 288 lines															
EAV Code				Blanking Video				SAV Code				Active Video			
FF	0	0	9D	80	10	80	10	FF	0	0	80	xx	xx	xx	xx
Repeat 1 time				Repeat 70 times				Repeat 1 time				Repeat 360 times			
Field 0 - Second Vertical Blanking(Bottom) - Repeat for 2 lines															
EAV Code				Blanking Video				SAV Code				Active Video			
FF	0	0	B6	80	10	80	10	FF	0	0	AB	80	10	80	10
Repeat 1 time				Repeat 70 times				Repeat 1 time				Repeat 360 times			
Field 1 - First Vertical Blanking(Top) - Repeat for 23 lines															
EAV Code				Blanking Video				SAV Code				Active Video			
FF	0	0	F1	80	10	80	10	FF	0	0	EC	80	10	80	10
Repeat 1 time				Repeat 70 times				Repeat 1 time				Repeat 360 times			
Field 1 - Active Video - Repeat for 288 lines															
EAV Code				Blanking Video				SAV Code				Active Video			
FF	0	0	DA	80	10	80	10	FF	0	0	C7	xx	xx	xx	xx
Repeat 1 time				Repeat 70 times				Repeat 1 time				Repeat 360 times			
Field 1 - Second Vertical Blanking(Bottom) - Repeat for 2 lines															
EAV Code				Blanking Video				SAV Code				Active Video			
FF	0	0	F1	80	10	80	10	FF	0	0	EC	80	10	80	10
Repeat 1 time				Repeat 70 times				Repeat 1 time				Repeat 360 times			

Table 2-3. PAL Frame

IPU_CSI_CCIR_CODE_3 register is the setting for EAV/SAV first three words, FF 00 00. So this register is set to 0xFF0000 for all BT.656 and BT.1120 format.

For interlaced input, the IPU_CSI_CCIR_CODE_1 register is for the fourth word of field 0's EAV/SAV, the three bits register value for each region (MSB to LSB) is H, V, F data. And IPU_CSI_CCIR_CODE_2 register is for the fourth word of field 1's EAV/SAV setting.

The detail register setting (Reference to Table 2-4):

IPU_CSI_CCIR_CODE_1 = CSI_END_FLD0_BLNK_1ST +

(CSI_STRT_FLD0_BLNK_1ST<<3) +
(CSI_END_FLD0_BLNK_2ND<<6) +
(CSI_STRT_FLD0_BLNK_2ND<<9) +
(CSI_END_FLD0_ACTV<<16) +
(CSI_STRT_FLD0_ACTV<<19)
= 0x40596;

IPU_CSI_CCIR_CODE_2 = CSI_END_FLD1_BLNK_1ST +
(CSI_STRT_FLD1_BLNK_1ST<<3) +
(CSI_END_FLD1_BLNK_2ND<<6) +
(CSI_STRT_FLD1_BLNK_2ND<<9) +
(CSI_END_FLD1_ACTV<<16) +
(CSI_STRT_FLD1_ACTV<<19)
= 0xD07DF.

Field 0 Blanking(Top) EAV = 0xB6	F = 0	V = 1	H = 1	CSI_END_FLD0_BLNK_1ST HVF = 0b110 = 0x6
Field 0 Blanking(Top) SAV = 0xAB	F = 0	V = 1	H = 0	CSI_STRT_FLD0_BLNK_1ST HVF = 0b010 = 0x2
Field 0 Active Video EAV = 0x9D	F = 0	V = 0	H = 1	CSI_END_FLD0_ACTV HVF = 0b100 = 0x4
Field 0 Active Video SAV = 0x80	F = 0	V = 0	H = 0	CSI_STRT_FLD0_ACTV HVF = 0b000 = 0x0
Field 0 Blanking(Bottom) EAV = 0xB6	F = 0	V = 1	H = 1	CSI_END_FLD0_BLNK_2ND HVF = 0b110 = 0x6
Field 0 Blanking(Bottom) SAV = 0xAB	F = 0	V = 1	H = 0	CSI_STRT_FLD0_BLNK_2ND HVF = 0b010 = 0x2
Field 1 Blanking(Top) EAV = 0xF1	F = 1	V = 1	H = 1	CSI_END_FLD1_BLNK_1ST HVF = 0b111 = 0x7
Field 1 Blanking(Top) SAV = 0xEC	F = 1	V = 1	H = 0	CSI_STRT_FLD1_BLNK_1ST HVF = 0b011 = 0x3
Field 1 Active Video EAV = 0xDA	F = 1	V = 0	H = 1	CSI_END_FLD1_ACTV HVF = 0b101 = 0x5
Field 1 Active Video SAV = 0xC7	F = 1	V = 0	H = 0	CSI_STRT_FLD1_ACTV HVF = 0b001 = 0x1
Field 1 Blanking(Bottom) EAV = 0xF1	F = 1	V = 1	H = 1	CSI_END_FLD1_BLNK_2ND HVF = 0b111 = 0x7

Field 1 Blanking(Bottom) SAV = 0xEC	F = 1	V = 1	H = 0	CSI_STRT_FLD1_BLNK_2ND HVF = 0b011 = 0x3

Table 2-4. EAV/SAV to CCIR CODE register for Interlaced Signal

2.5.2 CCIR Register Setting for Progressive Signal

The BT.656 720*480 and 720*576 progressive frame format is as followed:

Vertical Blanking - Repeat for 45 lines															
EAV Code				Blanking Video				SAV Code				Active Video			
FF	0	0	B6	80	10	80	10	FF	0	0	AB	80	10	80	10
Repeat 1 time				Repeat 67 times				Repeat 1 time				Repeat 360 times			
Active Video - Repeat for 480 lines															
EAV Code				Blanking Video				SAV Code				Active Video			
FF	0	0	9D	80	10	80	10	FF	0	0	80	xx	xx	xx	xx
Repeat 1 time				Repeat 67 times				Repeat 1 time				Repeat 360 times			

Table 2-5. 480P Frame

Vertical Blanking - Repeat for 49 lines															
EAV Code				Blanking Video				SAV Code				Active Video			
FF	0	0	B6	80	10	80	10	FF	0	0	AB	80	10	80	10
Repeat 1 time				Repeat 70 times				Repeat 1 time				Repeat 360 times			
Active Video - Repeat for 576 lines															
EAV Code				Blanking Video				SAV Code				Active Video			
FF	0	0	9D	80	10	80	10	FF	0	0	80	xx	xx	xx	xx
Repeat 1 time				Repeat 70 times				Repeat 1 time				Repeat 360 times			

Table 2-6. 576P Frame

IPU_CSI_CCIR_CODE_3 register is setting for EAV/SAV first three word, FF 00 00. So this register is set to 0xFF0000 for all BT.656 and BT.1120 format.

For progressive input, the IPU_CSI_CCIR_CODE_2 register is not needed, and in register IPU_CSI_CCIR_CODE_1, only CSI_STRT_FLD0_BLNK_1ST, CSI_STRT_FLD0_ACTV and CSI_END_FLD0_ACTV are used, the three bits register value for each region (MSB to LSB) is H, V, F data, and F is always 0.

The detail register setting (Reference to Table 2-7):

IPU_CSI_CCIR_CODE_1 = (CSI_STRT_FLD0_BLNK_1ST<<3) +

(CSI_END_FLD0_ACTV<<16) +

(CSI_STRT_FLD0_ACTV<<19)

= 0x40010.

Note: CSI_STRT_FLD0_BLNK_1ST is filled with blanking SAV data, and CSI_END_FLD0_BLNK_1ST is not used for progressive input. In Linux BSP, CSI_STRT_FLD0_BLNK_1ST is filled with blanking EAV data (0b110, 0x6), it can also work, because when CSI find the setting code in CSI_STRT_FLD0_BLNK_1ST, those lines will be dropped as blanking lines.

Blanking SAV = 0xAB	F = 0	V = 1	H = 0	CSI_START_FLD0_BLNK_1ST HVF = 0b010 = 0x2
Active Video EAV = 0x9D	F = 0	V = 0	H = 1	CSI_END_FLD0_ACTV HVF = 0b100 = 0x4
Active Video SAV = 0x80	F = 0	V = 0	H = 0	CSI_STRT_FLD0_ACTV HVF = 0b000 = 0x0

Table 2-7. EAV/SAV to CCIR CODE register for Progressive Signal

2.6 CSI_VSC Setting for NTSC

From BT.656-4 and above version protocol, the active video lines in each frame is 487, not 480. But for CVBS camera, the real video are 480 lines, other 7 lines are padded by TVIN chip, so these 7 lines should be dropped by software. The CSI_VSC is used for such purpose.

For BT.656-3, the NTSC padded lines are 27 lines.

And for PAL, there is no such padded lines.

To interlaced signal, the CSI_VSC will skip same lines for both fields. And for the BT.656-4 NTSC signal, its first field has 244 lines of active video and the second field has 243 lines of active video. The 4 lines of field 0 and 3 lines of field 1 are not valid video, they should be skipped.

After switched CCIR_CODE_1 and CCIR_CODE_2 register setting, and set CSI_VSC to 3 for NTSC, the IDMAC will capture field 1 data first, then field 0.

First skip 3 lines and capture 240 lines of field 1, then skip 3 lines and capture 240 lines of field 0, IDMAC will generate EOF interrupt, and software will prepare the capture buffer and start to capture the next frame, between two frames of capture, the hardware needs at least 1 blank line time, so the last line of field 0 will be dropped in this way.

For PAL signal, the video lines in each field are all 288, so there is no such issue.

In Linux BSP, the CSI_VSC setting is in data struct video_fmt_t, active_top of file “drivers\media\platform\mxc\capture\mxc_v4l2_capture.c”. The NTSC setting “active_top = 13” is used for BT.656-3 mode, it is the default mode of adv7180. In BT.656-3 NTSC frame, there are total 507 lines of active video, field 0 has 254 lines and field 1 has 253 lines:

```
static video_fmt_t video_fmts[] = {
{                               /*! NTSC */
 .v4l2_id = V4L2_STD_NTSC,
 .name = "NTSC",
 .raw_width = 720,              /* SENS_FRM_WIDTH */
 .raw_height = 525,             /* SENS_FRM_HEIGHT */
 .active_width = 720,           /* ACT_FRM_WIDTH */
 .active_height = 480,          /* ACT_FRM_HEIGHT */
 .active_top = 13,
 .active_left = 0,
 },
```

2.7 Clock Mode Setting in Driver

The default BSP can only support two kind of clock mode in driver mxc_v4l2_capture.c:

```
if (ifparm.u.bt656.clock_curr == 0)
        csi_param.clk_mode = IPU_CSI_CLK_MODE_CCIR656_INTERLACED;
else
        csi_param.clk_mode = IPU_CSI_CLK_MODE_GATED_CLK;
```

So in TVIN driver, such as adv7180.c, the function ioctl_g_ifparm() is used to transfer these information to mxc_v4l2_capture driver, it used “p->u.bt656.clock_curr”.

To support more clock mode, such as BT.656 progressive, BT.1120, please reference to followed community link:

https://community.nxp.com/thread/295157

That reference code used “clock_curr = 0” for BT.656/1120 interlaced clock mode; “clock_curr = 1” for BT.656/1120 progressive clock mode; “clock_curr = others” for gated clock mode.

3 MIPI-CSI2 TVIN

MIPI CSI2 data transferred from MIPI CSI to IPU Gasket to IPU CSI, it is always in non-gated clock mode, although software can set the clock mode to gated clock mode in register

IPU_CSI_SENS_CONF, the hardware still works in non-gated clock mode. Linux BSP had set it to gated clock mode for MIPI CSI2 cameras in mxc_v4l2_capture driver, due to this reason, there is no problem.

And for the IPU_CSI_SENS_CONF register VSYNC mode setting, it is always external VSYNC mode in this register. SENS_DATA_FORMAT and DATA_WIDTH setting in the same register are ignored, the data format is set from CSI_MIPI_DI and the interface to IPU CSI is fixed at 16 bits.

If the input is progressive signal, it can be handled as normal camera signal. But for interlaced signal, such as NTSC or PAL, there is some difference.

There are two different kinds of MIPI CSI2 TVIN chips to send interlaced signal to IMX6.

- No frame end and start MIPI CSI2 short packages inserted between two fields. It can be captured with interlaced scan order IDMAC, and two fields can be combined into one frame in memory. It is called frame mode in TVIN chip.
- Frame end and start MIPI CSI2 short packages inserted between two fields. It can be handled as field data only. If it is captured with interlaced scan order IDMAC and frame buffer, there will be “New Frame before EOF” error reported from IPU_INT_STAT register. It is called field mode in TVIN chip.

CSI register setting example for BT.656-3 NTSC input on MIPI CSI2 interface:

- IPU_CSI_SENS_CONF

  CSI_EXT_VSYNC = 0x1, external VSYNC mode.

  CSI_DATA_WIDTH, it is ignored.

  CSI_SENS_DATA_FORMAT, it is ignored. It is set in register IPU_CSI_DI, CSI_MIPI_DI0 = 0x1E, UYVY format.

  CSI_SENS_PRTCL = 0x1, non-gated clock mode (In BSP, it is set to gated clock mode, but the hardware will always work in non-gated clock mode).

- IPU_CSI_SENS_FRM_SIZE, MIPI CSI2 has no blank data input to IPU CSI, so it is the active video frame size.

  For BT.656-3 NTSC, it is 720x507, so the register value = 0x01FA02CF.

- IPU_CSI_ACT_FRM_SIZE, it is the active video frame size, just capture 480 lines.

  For NTSC, it is 720*480, so the register value = 0x01DF02CF.

- IPU_CSI_OUT_FRM_CTRL, this register can be used to skip some pixel in each line or skip some lines in each frame.

  For NTSC, it depends on TVIN chip, for BT.656-4 and above version TVIN chip, CSI_VSC is 3; for BT.656-3 TVIN chip, CSI_VSC is 13. Details, please reference to 2.6 The CSI_VSC setting for NTSC.

- IPU_CSI_CCIR_CODE_1:

  Not used for MIPI CSI2 case, 0x0.

- IPU_CSI_CCIR_CODE_2:

  Not used for MIPI CSI2 case, 0x0.

- IPU_CSI_CCIR_CODE_3:

  Not used for MIPI CSI2 case, 0x0.

IDMAC Setting for frame mode: scan order is set to interlaced, and stride is set to two lines.

IDMAC Setting for field mode: scan order is set to progressive, and stride is set to one line, the capture buffer side is set to field data size, so it is captured in 60 fps of field data for NTSC or 50 fps field data for PAL.

3.1 How to identify ODD and EVEN field

Since to IPU CSI, the clock mode is always non-gated clock mode for MIPI CSI2 input, it can only receive video data, no EAV/SAV data can be received. So how to identify the field order is a problem.

A possible solution: the camera or TVIN chip can generate a special line at the top or bottom of one field, then software can check this special line to identify the field order in captured memory buffer. The field order is important for VDI de-interlace.

3.2 MIPI CSI2 Virtual Channel Issue

For some MIPI CSI2 TVIN chips, when software sets their virtual channel to not 0, they will only send MIPI long packages (video data) to other virtual channel but keep MIPI short packages (timing data such as frame start, frame end) on virtual channel 0. Such kind of TVIN chip can't be supported by iMX6 with not 0 virtual channel. They must work at virtual channel 0.

With such TVIN chip, to hardware design, if parallel CSI0 is also used for another camera sensor, then the two cameras will fail to work together.

4 VDI De-interlace

iMX6 IPU VDI can support three kind of motion mode to do the de-interlace, VDI_MOT_SEL (0,1,2) in register IPU_VDI_C, motion mode 0 and 1 need 3 field data to processing; and motion mode 2 needs only one field data.

4.1 The TVIN Unit Test Application

The unit test application “mxc_v4l2_tvin.c” is the BSP released sample code to preview the TVIN input with VDI de-interlace support.

Test command: /unit_tests/mxc_v4l2_tvin.out –ow 800 –oh 480 –m 1

The details for the parameters in this application:

-ow: output width.

-oh: output height.

-m: the motion mode, 0, 1 or 2, VDI_MOT_SEL in register IPU_VDI_C.

-tb: top field first or bottom field first, VDIC top filed (manual) in register IPU_VDI_C. If software switched the field order in CCIR_CODE_1 and CCIR_CODE_2, the test application should change the “-tb” to set the correct field order in VDI.

Some improvement for the test application:

- Remove the memcpy.

  The default mxc_v4l2_tvin application used memcpy to copy the data from capture buffer to v4l2 ouput buffer, this memcpy can be removed and software can use display framebuffer as the capture buffer directly, then call FBIOPAN_DISPLAY to render.

- Using G2D render.

  Except for v4l2 output render, software can also use the G2D to render the video. It is easy to support multi camera render in this way.

The enhanced TVIN test application: https://community.nxp.com/docs/DOC-328548

4.2 Memory to Memory De-interlace

The mxc_v4l2_tvin test application used the CSI->MEM path to capture, then it used memory to memory de-interlace method in V4l2 output driver. The code for CSI->MEM path is “g_input = 1” with VIDIOC_S_INPUT.

IPU task can also be used for memory to memory de-interlace: https://community.nxp.com/docs/DOC-158531

4.3 On The Fly De-interlace

Except for the memory to memory de-interlace, the IPU can also support on the fly de-interlace with CSI->VDI->MEM (g_input = 3 with VIDIOC_S_INPUT) and CSI->VDI->IC->MEM (g_input = 2 with VIDIOC_S_INPUT) capture path: https://community.nxp.com/docs/DOC-330441

Limitations:

1. Since the IC can only output resolution up to 1024*1024, so this is the limitation on output for CSI->VDI->IC->MEM path.
2. Since the VDI can only output resolution up to 968*1024, so CSI->VDI->MEM can't work for 1080i capture.
3. Only VDI motion mode 2 was supported. To avoid dithering, the driver should set one field data for CSI->VDI->xxx path, set VDI_SKIP and VDI_MAX_RATIO_SKIP to 1 in register IPU_SKIP.

Note: The reason to set VDI_SKIP to skip one field data, ODD field starts from line 1 and EVEN field starts from line2, so they are one line shift, if the VDI using both field data, the output video will dithering between switching ODD and EVEN field.

4.4 Double FPS De-interlace

In default Linux BSP, there are 3 kinds of de-interlace mode, motion = 0, 1 and 2 mode, motion mode 0 and 1 use three fields for de-interlace, and motion mode 2 uses one field for de-interlace, when received one frame of data, the driver will de-interlace and render one frame, so the whole de-interlace fps is 30 for NTSC input and 25 for PAL input.

In this mode, for motion mode 0 and 1, field 1, 2 and 3 are used for first VDI output frame of display; and field 3, 4 and 5 are used for second VDI output frame of display; field 5, 6 and 7 are used for third VDI output frame of display. One field data (such as 2, 4, and 6) was used only once, so the weight for one field data is less than another field, this had impact the video quality.

After improvement, the VDI de-interlace output can be 60fps for NTSC input and 50 fps for PAL: for motion mode 0 and 1, field 0, 1 and 2 are used for first VDI output frame of display; field 1, 2 and 3 are used for second VDI output frame of display; field 2, 3 and 4 are used for third VDI output frame of display. So all field data will be used twice, two field data are balanced, the VDI quality is improved.

The BSP patch for this solution:

https://community.nxp.com/docs/DOC-173003

5 Common issues

5.1 Screen Scroll issue

Sometimes there will be incomplete frame on BT.656 and BT.1120 interface, they will cause screen scroll issue for TVIN preview. For such case, the software should re-start the v4l2 capture task to make it re-synced.

Although the IPU CSI port can find the correct sync signal at once after BT.656 signal is stable again, the IDMAC can't drop the captured dirty data and re-start to fill the buffer from its base address. So software needs detect the failure and do the whole recovery.

Some methods can be used to do the recovery, the key point is to detect such signal error issue.

5.1.1 Detect NFB4EOF Error or TVIN Signal Lock Lost

Checking the signal lock lost event from TVIN chip's status registers; checking the CSI NFB4EOF error from iMX6 IPU status registers. If any of the above case happens, recover the failure with software workaround.

The reference kernel patch based on L3.0.35_GA4.1.0 release:

```
diff --git a/drivers/media/video/mxc/capture/adv7180.c b/drivers/media/video/mxc/capture/adv7180.c
index 3c12fdc..187123d 100644
--- a/drivers/media/video/mxc/capture/adv7180.c
+++ b/drivers/media/video/mxc/capture/adv7180.c
@@ -68,6 +68,7 @@ static struct i2c_driver adv7180_i2c_driver = {
 struct sensor {
         struct sensor_data sen;
         v4l2_std_id std_id;
+        int lost_lock_status;
 } adv7180_data;


@@ -234,22 +235,29 @@ static void adv7180_get_std(v4l2_std_id *std)
         dev_dbg(&adv7180_data.sen.i2c_client->dev, "In adv7180_get_std\n");

         /* Read the AD_RESULT to get the detect output video standard */
-        tmp = adv7180_read(ADV7180_STATUS_1) & 0x70;
```

```
+       tmp = adv7180_read(ADV7180_STATUS_1);
+       adv7180_data.lost_lock_status |= (tmp & 0x02);

        mutex_lock(&mutex);
-       if (tmp == 0x40) {
-               /* PAL */
-               *std = V4L2_STD_PAL;
-               idx = ADV7180_PAL;
-       } else if (tmp == 0) {
-               /*NTSC*/
-               *std = V4L2_STD_NTSC;
-               idx = ADV7180_NTSC;
-       } else {
-               *std = V4L2_STD_ALL;
+       if ((tmp & 0x01) == 0) {
+               /* signal not locked */
+               *std = V4L2_STD_UNKNOWN;
                idx = ADV7180_NOT_LOCKED;
-               dev_dbg(&adv7180_data.sen.i2c_client->dev,
-                       "Got invalid video standard!\n");
+       } else {
+               if ((tmp & 0x70) == 0x40) {
+                       /* PAL */
+                       *std = V4L2_STD_PAL;
+                       idx = ADV7180_PAL;
+               } else if ((tmp & 0x70) == 0) {
+                       /*NTSC*/
+                       *std = V4L2_STD_NTSC;
+                       idx = ADV7180_NTSC;
+               } else {
+                       *std = V4L2_STD_ALL;
+                       idx = ADV7180_NOT_LOCKED;
+                       dev_dbg(&adv7180_data.sen.i2c_client->dev,
+                               "Got invalid video standard!\n");
```

```
+                       }
                }
                mutex_unlock(&mutex);

@@ -702,6 +710,17 @@ static int ioctl_g_chip_ident(struct v4l2_int_device *s, int *id)
                return 0;
 }

+static int ioctl_g_lock_status(struct v4l2_int_device *s)
+{
+       int status;
+
+       status = adv7180_read(ADV7180_STATUS_1);
+       status |= adv7180_data.lost_lock_status;
+       adv7180_data.lost_lock_status = 0;
+
+       return status;
+}
+
 /*!
  * ioctl_init - V4L2 sensor interface handler for VIDIOC_INT_INIT
  * @s: pointer to standard V4L2 device structure
@@ -776,6 +795,7 @@ static struct v4l2_int_ioctl_desc adv7180_ioctl_desc[] = {
                                        (v4l2_int_ioctl_func *) ioctl_enum_framesizes},
        {vidioc_int_g_chip_ident_num,
                                        (v4l2_int_ioctl_func *)ioctl_g_chip_ident},
+       {vidioc_int_g_lock_status_num, (v4l2_int_ioctl_func*)ioctl_g_lock_status},
 };

 static struct v4l2_int_slave adv7180_slave = {
diff --git a/drivers/media/video/mxc/capture/mxc_v4l2_capture.c b/drivers/media/video/mxc/capture/mxc_v4l2_capture.c
index 9130388..a070b8a 100644
--- a/drivers/media/video/mxc/capture/mxc_v4l2_capture.c
```

```
+++ b/drivers/media/video/mxc/capture/mxc_v4l2_capture.c
@@ -413,6 +413,8 @@ static int mxc_streamon(cam_data *cam)
        if (cam->overlay_on == true)
                stop_preview(cam);

+       ipu_reset_csi_frame_error(cam->ipu);
+
        if (cam->enc_enable) {
                err = cam->enc_enable(cam);
                if (err != 0) {
@@ -500,6 +502,39 @@ static int mxc_streamoff(cam_data *cam)
        return err;
 }

+static inline void append_list(struct list_head *dst, struct list_head* src)
+{
+       src->prev->next = dst->next;
+       dst->next->prev = src->prev;
+       dst->next = src->next;
+       src->next->prev = dst;
+}
+
+static void mxc_reset_stream(cam_data *cam)
+{
+       struct list_head tmp_queue;
+       struct mxc_v4l_frame *frame;
+
+       INIT_LIST_HEAD(&tmp_queue);
+
+       /* collect buffers from all queues and store them in tmp_queue */
+       append_list(&tmp_queue, &cam->working_q);
+       append_list(&tmp_queue, &cam->ready_q);
+       append_list(&tmp_queue, &cam->done_q);
+
```

```
+	/* stop capturing */
+	mxc_streamoff(cam);
+
+	/* enqueue buffers */
+	list_for_each_entry(frame, &tmp_queue, queue) {
+		frame->buffer.flags |= V4L2_BUF_FLAG_QUEUED;
+	}
+	append_list(&cam->ready_q, &tmp_queue);
+
+	/* start capturing */
+	mxc_streamon(cam);
+}
+
 /*!
  * Valid and adjust the overlay window size, position
  *
@@ -1490,9 +1525,19 @@ static int mxc_v4l_dqueue(cam_data *cam, struct v4l2_buffer *buf)
 	int retval = 0;
 	struct mxc_v4l_frame *frame;
 	unsigned long lock_flags;
+	struct ipu_soc *ipu = cam->ipu;

 	pr_debug("In MVC:mxc_v4l_dqueue\n");

+	if (cam->device_type == 1) {
+		int status;
+
+		status = vidioc_int_g_lock_status(cam->sensor);
+		if (ipu_get_csi_frame_error(ipu) || ((status & 0x03)==0x03)) {
+			mxc_reset_stream(cam);
+		}
+	}
+
 	if (!wait_event_interruptible_timeout(cam->enc_queue,
```

```
                                                cam->enc_counter != 0, 10 * HZ)) {
                        pr_err("ERROR: v4l2 capture: mxc_v4l_dqueue timeout "
diff --git a/drivers/mxc/ipu3/ipu_common.c b/drivers/mxc/ipu3/ipu_common.c
index c2b5717..6acf6379 100644
--- a/drivers/mxc/ipu3/ipu_common.c
+++ b/drivers/mxc/ipu3/ipu_common.c
@@ -2456,10 +2456,10 @@ static irqreturn_t ipu_err_irq_handler(int irq, void *desc)
                                dev_warn(ipu->dev,
                                        "IPU Warning - IPU_INT_STAT_%d = 0x%08X\n",
                                        err_reg[i], int_stat);
-                               /* Disable interrupts so we only get error once */
-                               int_stat = ipu_cm_read(ipu, IPU_INT_CTRL(err_reg[i])) &
-                                       ~int_stat;
-                               ipu_cm_write(ipu, int_stat, IPU_INT_CTRL(err_reg[i]));
+                               if (err_reg[i]==5 && (int_stat & 0xf)) {
+                                       /* new frame before end of frame error on CSI_MEM channels 0-3 */
+                                       ipu->csi_nfb4eof_error |= int_stat;
+                               }
                        }
                }

@@ -2594,6 +2594,37 @@ bool ipu_get_irq_status(struct ipu_soc *ipu, uint32_t irq)
 EXPORT_SYMBOL(ipu_get_irq_status);

 /*!
+ * This function returns the current CSI frame error status.
+ *
+ * @param   ipu     ipu handler
+ *
+ * @return      Returns the frame error status.
+ */
+int ipu_get_csi_frame_error(struct ipu_soc *ipu)
+{
+       int err;
```

```
+
+	mutex_lock(&ipu->mutex_lock);
+	err = ipu->csi_nfb4eof_error;
+	mutex_unlock(&ipu->mutex_lock);
+	return err;
+}
+EXPORT_SYMBOL(ipu_get_csi_frame_error);
+
+/*!
+ * This function resets the current CSI frame error status.
+ *
+ * @param   ipu     ipu handler
+ */
+void ipu_reset_csi_frame_error(struct ipu_soc *ipu)
+{
+	mutex_lock(&ipu->mutex_lock);
+	ipu->csi_nfb4eof_error = 0;
+	mutex_unlock(&ipu->mutex_lock);
+}
+EXPORT_SYMBOL(ipu_reset_csi_frame_error);
+
+/*!
  * This function registers an interrupt handler function for the specified
  * interrupt line. The interrupt lines are defined in \b ipu_irq_line enum.
  *
diff --git a/drivers/mxc/ipu3/ipu_prv.h b/drivers/mxc/ipu3/ipu_prv.h
index a0d71dd..97cfeae 100644
--- a/drivers/mxc/ipu3/ipu_prv.h
+++ b/drivers/mxc/ipu3/ipu_prv.h
@@ -125,6 +125,8 @@ struct ipu_soc {
	int		vdoa_en;
	struct task_struct *thread[2];

+	/* CSI frame errors */
```

```
+           int csi_nfb4eof_error;
 };

 struct ipu_channel {
diff --git a/include/media/v4l2-int-device.h b/include/media/v4l2-int-device.h
index fbf5855..bd4ad19 100644
--- a/include/media/v4l2-int-device.h
+++ b/include/media/v4l2-int-device.h
@@ -222,6 +222,8 @@ enum v4l2_int_ioctl_num {
             /* VIDIOC_DBG_G_CHIP_IDENT */
             vidioc_int_g_chip_ident_num,

+           vidioc_int_g_lock_status_num,
+
             /*
              *
              * Start of private ioctls.
@@ -304,4 +306,6 @@ V4L2_INT_WRAPPER_0(reset);
 V4L2_INT_WRAPPER_0(init);
 V4L2_INT_WRAPPER_1(g_chip_ident, int, *);

+V4L2_INT_WRAPPER_0(g_lock_status);
+
 #endif
```

There are two difference cases for IPU status register to generate the NFB4EOF error or not.

Case 1: NFB4EOF is generated

- Data stream are missing lines and/or missing end of frame indication
- This reaction is useful when input signal to the TVIN chip is interrupted and restored – data stream starts from the beginning. New frame before end of the frame (NFB4EOF) is asserted:
  - First half of the frame is correctly transferred
  - Noise occur – stream is interrupted or restarted
  - Data stream continues with new frame

– IDMAC recognizes NF before IDMAC buffer EOF
– NFB4EOF is generated
– NFB4EOF will be generated in every frame until stream is restarted or re-synchronized

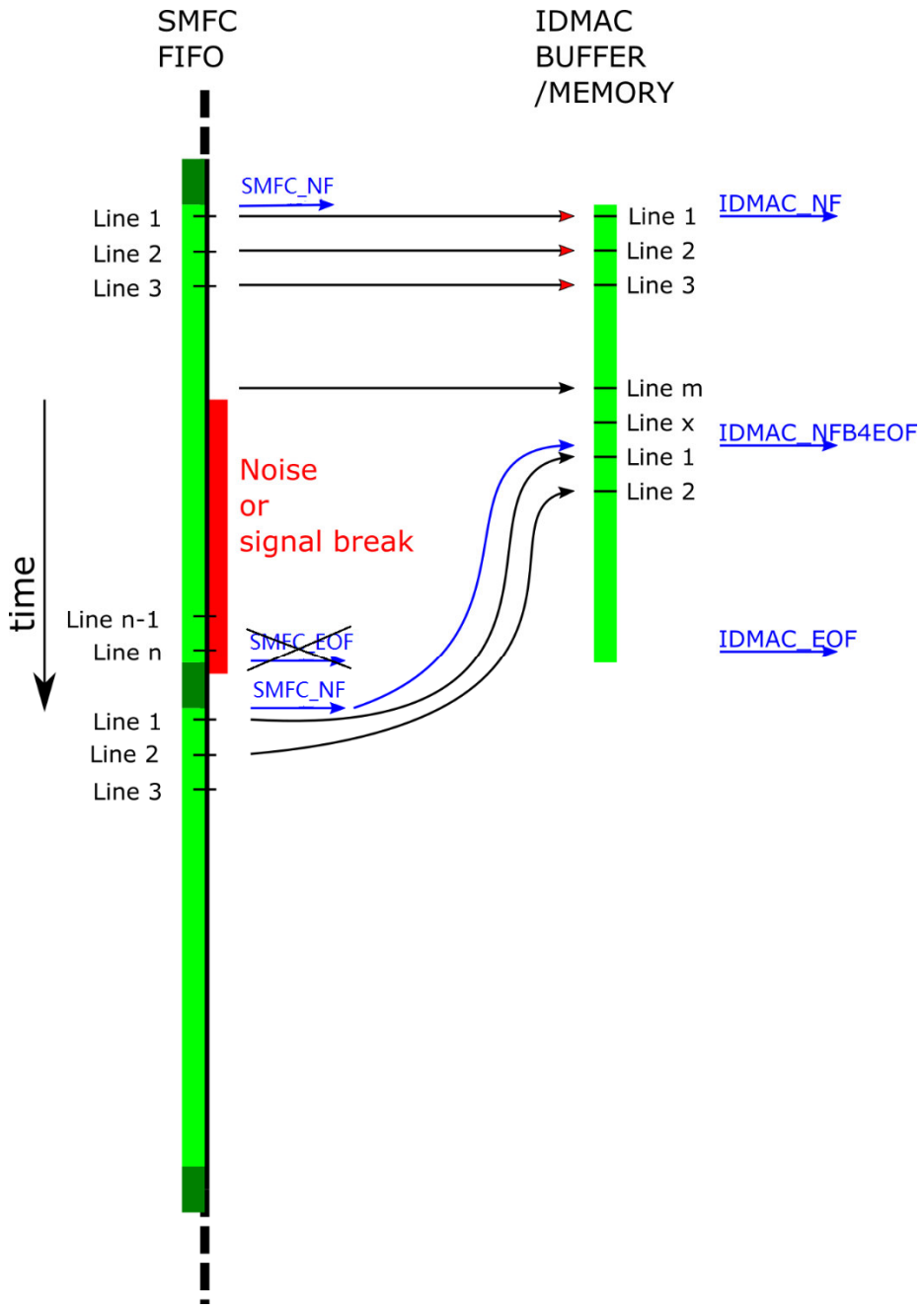

Figure 5-1. NFB4EOF generated

Case 2: NFB4EOF is not generated.

- Data stream are missing lines and/or missing start of frame indication

- New frame before end of the frame (NFB4EOF) cannot be asserted:
  - Missing SOF and lines at beginning
  - IDMAC start fill buffer from line 3
  - SMFC sends EOF before IDMAC_EOF – IDMAC state machine is restarted and perform reads: line n, line x, line 1
  - NF is not before IDMAC_EOF -> NFB4EOF is not generated
  - Next frame is shifted one line
  - Repeat till SMFC_EOF fits with IDMAC_EOF
  - Data stream is synchronized again

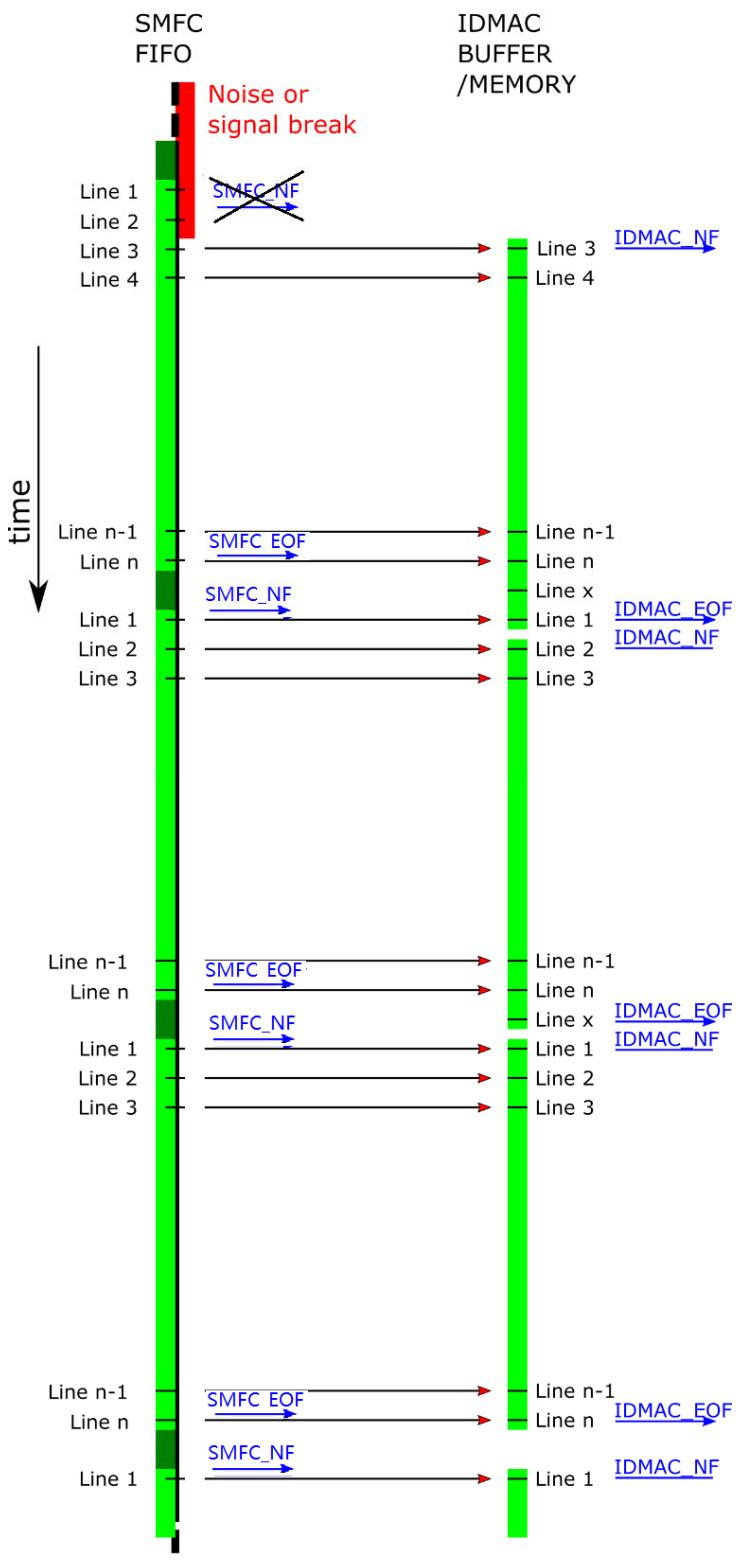

Figure 5-2. NFB4EOF not generated

5.1.2 Measure the Period between Each Two Frames

Measuring time between consecutive frames and restart the data stream if time between 2 consecutive frames is too long or too small.

In mxc_v4l2_tvin.c, the application will get return from VIDIOC_DQBUF when one frame data was captured in driver, so user can call gettimeofday() each time to know the period between two VIDIOC_DQBUF, if there is signal lost issue, the period will increase, then software can know it is issue time.

5.1.3 Use Special Line to Identify the Issue

Generate special data line at the top or bottom of the video frame, after captured each frame, software needs check the position of this special line to find whether the scroll issue happens or not.

If the special line is not at top or bottom, which means the signal lost issue happens, software should re-start the capture task.

5.2 No video issue

Mxc_v4l2_tvin application reports timeout and exit, the error messages like "ERROR: v4l2 capture: mxc_v4l_dqueue timeout enc_counter0 VIDIOC_DQBUF failed". This means the v4l2 capture driver hasn't captured valid video frame.

The followed things can be checked:

- Is CSI_DATA_EN pin been enabled by software but pulled down?
- Is there valid pixel clock?
- Is there EAV/SAV embedded on data line?
- Adjust the drive strength of the BT.656 pins.
- Is the frame size set in IPU_CSI_SENS_FRM_SIZE register bigger than the real frame size?
- Is the frame size set in IPU_CSI_ACT_FRM_SIZE register bigger than the real frame size?
- Is the IOMUX_GPR1 for iMX6DQ or IOMUX_GPIO13 for iMX6SDL been correctly set?
- To avoid hardware issue, the customer can set the data and clock pins as GPIO and output 0/1, then measure if each PIN can work correctly.
- If it is MIPI CSI2 interface, virtual channel and MIPI PHY clock setting in "mipi_csi2_reset()" should be checked.

5.3 How to Support Surround View Chips

There are some surround view chips which can combine multiple CVBS cameras into one BT.656 or MIPI CSI2 stream. For iMX6, it can only support some of them.

On BT.656 interface, if the combining method is pixel interleaved, then it is not the standard BT.656 signal, and iMX6 CSI can't capture them. For example, after combined two cameras, the SAV/EAV for pixel interleave will be "FF FF 00 00 00 00 00 00 00 00 XX XX", iMX6 IPU CSI can only support SAV/EAV in format "FF 00 00 XX", 4 bytes mode.

So on BT.656 interface, only line interleave mode can be supported, to IPU CSI, it just increases the line number for each frame, and each data line still aligns with BT.656 specification. If four cameras are combined, then IPU CSI can capture them as width*height*4 frame size, after captured into memory, software needs split them into four video for four cameras. In such line interleave mode, the camera id information should be captured into memory, if such information is in EAV/SAV, they can't be got by software.

For MIPI-CSI2 interface, if all cameras are combined into one video, it doesn't care the EAV/SAV setting, but software also needs be used to split them.

If each camera can be mapped to each MIPI-CSI2 virtual channel, then iMX6S/DL can support update to two cameras and iMX6D/Q can support update to 4 cameras. In this case, the combined camera video is split by virtual channel, so software is not needed to do the split.

6 Revision History

Table6-1 provides a revision history for this application note. Note that this revision history table reflects the changes to this template, but it can also be used for your document's revision history.

Table 6-1. Revision History

Rev. Number	Date	Substantive Change
0	08/2016	Initial release

How to Reach Us:

Home Page:
www.nxp.com

Web Support:
http://www.nxp.com/support

Document Number:
Rev.0
07/2016