Freescale Semiconductor
Application Note

Document Number: AN3285
Rev. 0, 08/2006

Image Capture Using PrP, CSI, and I^{2}C

MC9328MX21

by: Teri Cheng

1 Abstract

This document is a complement to the *Image Capture with MC9328MX21 Application Note* (AN2676) and provides a more step-by-step description of how to use the enhanced Multimedia Accelerator's (eMMA), Pre-Processor block (PrP), CMOS Sensor Interface (CSI), and I^{2}C modules of the i.MX21 to do image capture. This document also includes considerations for using and choosing different camera sensors and example code for different cameras.

This document applies to Freescale's MC9328MX21 applications processors, referred in this document as the i.MX21.

All documentation for the i.MX21, including application notes and the reference manual mentioned are located at the i.MX21 product page at www.freescale.com/imx (referred to as *online* in this document). See also, Section 7, "Reference Documents."

Contents

2 Introduction

This document will detail the use of the PrP, CSI, and I^{2}C and provide example code for each. The document will also describe considerations for choosing a camera for the i.MX21 and for writing code for cameras with differing capabilities.

As mentioned in application note AN2676, the image data is sent from the camera sensor to the i.MX21 via the CMOS Sensor Interface (CSI) module. This data is directly passed via a dedicated link to the Pre-Processor (PrP) module of the eMMA of the i.MX21. The data can then be sent to main memory for display via channel 1 of the PrP or to main memory for further processing (such as encoding/decoding or compressing) via channel II of the PrP.

Cameras called *smart* sensors are capable of doing their own pre-processing, in which case the data may be sent directly to the display buffer (allocated by software) without having to use the PrP. However, if further processing on the image data is required, such as encoding/decoding, data compression, color space conversion (CSC), or further resizing, the user must send it to main memory.

The Post-processor (PP) block of the eMMA module takes decoded data from memory and does de-blocking, de-ringing, image re-sizing, and CSC before sending it to display. CSC is the process of converting image data from YUV into RGB or vice versa, and it is done by either programming the CSC block or by software. Note that this document does not describe how to use the PP module or do any post-processing or data compression because it mainly focuses on image capture and display. Refer to the enhanced Multimedia Accelerator chapter of the i.MX21 Reference Manual for more information.

2.1 Preliminary Camera Information

Please note that the two camera sensors mentioned in this document (IM8012 VGA camera by iMagic, which uses the MT9V111 processor, and OV9640 1.3MPixel camera by OmniVision) might be discontinued by the time this application note is published (make sure to check with the manufacturers). However, this does not change the fundamentals discussed within the document. If further information is required about these cameras, please go to the following sites: for the VGA camera, www.imagictek.com and www.micron.com, and for the 1.3MP camera, www.ovt.com.

2.2 Register Information

Information about most registers referenced in the text or in sample code can be found in the i.MX21 Reference Manual. The registers are spelled in the example code as they are spelled in the reference manual unless otherwise specified.

2.3 Code Information

This document refers to code from two software application demos, VGA_imgCapture and 1.3MP_imgCapture, which can be found online at the Freescale product page. They are referred to in this document as *demo programs* or *demos*. They were developed using the two cameras mentioned in Section 2.1, "Preliminary Camera Information" and are setup to run from the i.MX21 ADS board without an operating system.

3 Initialization

3.1 System

First, the user must initialize the i.MX21 board for communication. This is accomplished by an initialization file used with the ARM® AXD Debugger or a codelet file used for Nucleus® EDGE. AXD is the debugger currently used in conjunction with the CodeWarrior for ARM IDE, and Nucleus EDGE is a similar tool with its own debugger embedded. A codelet file, which is very similar to the AXD initialization file, establishes communication to the ADS board and is used specifically with Nucleus EDGE. This file should initialize such registers as those for the peripheral clock control, chip select, GPIO registers, memory (like SDRAM) control, clocks, and so on. See examples of an AXD init file in either of the demo projects (OmniVision_1.3MP_imgCapture or iMagic_VGA_imgCapture) and the codelet file (i.MX21EVB_AXD_Init_v0.0.7_(For_Tape_Out_2_ONLY).txt) at the i.MX21 product page online. Also see the white paper called *WhitePaper_usingEDGE.doc* online for more information about using EDGE.

NOTE

The Nucleus EDGE development tool and debugger (from Advanced Technology) was under development at the time of this document's publication. Any files referenced to Nucleus are for example purposes only and do not imply eventual Nucleus availability.

Other initialization protocols include setting up the program memory, program counter, start of application code, and so on. See the init.s, heap.s, and Vector.s files in the demos. The user must also setup the memory management unit and caches as in the SysInit(void) function in the file, *SysInit.c*, which is called in main( ) in the demos.

3.2 Modules

The modules used require GPIO and clock enablement. The enable bits specified in Table 1 enable the clocks for the corresponding module. The ports specified in Table 2 are dedicated GPIO pins for the corresponding module which need to be configured using the GIUS register for the appropriate port (A, B, or D). Examples of how to set these registers are included in the following sections. Also, see the chapters on the Phase-Locked Loop, Clock and Reset Controller and the General Purpose I/O (GPIO) of the i.MX21 Reference Manual for more information on the clocking system for the i.MX21, specifically the ipg_clk, PERCLK's and other system clocks, and the GPIO In Use (GIUS) Register.

Table 1. Clock Trees for the CSI, I^{2}C, DMA, PrP, and LCDC Module (from AN2676)

Module	Peripheral Clock Control Register	Enable Bit
CSI	PCCR0	31
I^{2}C	PCCR0	12
DMA	PCCR0	30,23
PRP	PCCR0	27, 15
LCDC	PCCR0	26

Table 2. Pin Out for CSI, I^{2}C, and LCDC Modules

Module	Port
CSI	PB [21:10]
I^{2}C	PD [18:17]
LCDC	PA[31:5]

3.2.1 PrP in eMMA

As mentioned, the PrP receives data from the camera through the CSI across a dedicated link and performs processing to do resizing and CSC before sending the data to system memory for either display or further processing.

3.2.1.1 PrP Clock Enable and Reset

The HCLK and the peripheral ipg clock (PERCLK) into the eMMA module need to be enabled using the Peripheral Clock Control Register 0 (PCCR0). The frequency of the peripheral clock is defined by the Peripheral Clock Divider Register (PCDR1). This register designates the divisor for PERCLK1, 2, 3, and 4.

In the example in Section 3.2.1.2, “PrP Example Init, the AHB-Lite to IP bus Interface (AIPI) module is set to allow user access to all registers on the AIPI bus. AIPI1_PAR and AIPI2_PAR are the Peripheral Access Registers (PAR) for the AIPI1 and AIPI2 interfaces. A 1 for each bit in these registers indicates the corresponding peripheral as a supervisor-access only peripheral; a 0 indicates that the decision is up to the peripheral. For more information on the PERCLK and AIPI, see the chapter on the *PLL, Clock, and Reset Controller* and the chapter on the AIPI respectively in the i.MX21 Reference Manual.

Before configuring the rest of the PrP settings, the PrP must be reset by writing a 1 to the SWRST bit of the PRP_CNTL register. This resets all the PrP registers to their default values.

3.2.1.2 PrP Example Init

Example 1 is sample code to setup the PrP to receive VGA data in YUV422 format and output it in QVGA resolution, RGB565 format to channel 1 and YUV420 format to channel 2. The code also demonstrates how to program the CSC block to do color space conversion from YUV to RGB and resizing from VGA to QVGA (640 × 480 to 320 × 240 pixels). For more information on how to do resizing, cascading, and CSC, see the chapter in the i.MX21 Reference Manual on the eMMA and Application Note AN2886, *Operating Principle of Resize in the eMMA Pre-Processor.*

Example 1. PrP to Receive VGA Data

```
//        PRP Init Case 33
//
//        CSI Input   : VGA YUV422
//        Ch1 Output : QVGA RGB565
//        Ch2 Output : QVGA YUV420
//
//        resize = 2:1
//        stride = 320
//        loop mode
//        cascade mode disable
//
void PRP_init33(int ch1out, int ch2y, int ch2u, int ch2v)
{
          unsigned short in_csc[10];
          int width = 640;
          int height = 480;
          int ch1stride = 320;

//PRP_module_init
//HCLK enable for eMMA
          *(uint32_t *)PCCR0 |= 0x8008000;          //bit 27 & bit 15
//set all registers on AIPI bus to be user-accessible
          *(uint32_t *)AIPI1_PAR = 0x0;             //PAR reg for AIPI1
          *(uint32_t *)AIPI2_PAR = 0x0;             //PAR reg for AIPI2

//PRP_reset
          *(uint32_t *)PRP_CNTL = 0x1000;

//common control
          *(uint32_t *)PRP_CNTL = 0x0000862C;       //loop mode
                                                    //cascade mode disable
                                                    //input = YUV422
                                                    //ch1 output = RGB565
                                                    //ch2 output = YUV420
                                                    //csi enable
//        *(uint32_t *)PRP_INTRCNTL = 0x000001AF;   //enable all interrupts

//source control
          *(uint32_t *)PRP_SRC_PIXEL_FORMAT_CNTL = 0x20100888;      //input = YVYU
          *(uint32_t *)PRP_SOURCE_FRAME_SIZE = width << 16 | height;//source

//ch1 (RGB) dest control
          *(uint32_t *)PRP_DEST_RGB1_PTR = ch1out;              //output buf1
          *(uint32_t *)PRP_DEST_RGB2_PTR = ch1out;              //output buf2, same as buf1
          *(uint32_t *)PRP_CH1_OUT_IMAGE_SIZE = width << 16 | height;//ch1 size
          *(uint32_t *)PRP_CH1_PIXEL_FORMAT_CNTL = 0x2ca00565; //rgb565
          *(uint32_t *)PRP_CH1_LINE_STRIDE = ch1stride * 2;    //dest stride

//ch2 (YUV) dest control
          *(uint32_t *)PRP_DEST_Y_PTR = ch2y;                   //output buf1
          *(uint32_t *)PRP_DEST_CB_PTR = ch2u;
          *(uint32_t *)PRP_DEST_CR_PTR = ch2v;
          *(uint32_t *)PRP_SOURCE_Y_PTR = ch2y;                 //output buf2, resuse buf1
          *(uint32_t *)PRP_SOURCE_CB_PTR = ch2u;
          *(uint32_t *)PRP_SOURCE_CR_PTR = ch2v;

          *(uint32_t *)PRP_CH2_OUT_IMAGE_SIZE = width << 16 | height;//ch2 size

//csc matrix coeff
          in_csc[0] = 0;    //index  0..3 : A.1 A.0 B.1 B.0
          in_csc[1] = 0;    //direction0 : YUV2RGB  1 : RGB2YUV

          csc_tbl(in_csc);
          *(uint32_t *)PRP_CSC_COEF_012 = (in_csc[0] << 21) |
                             (in_csc[1] << 11) | in_csc[2];
```

```
        *(uint32_t *)PRP_CSC_COEF_345= (in_csc[3] << 21) |
                          (in_csc[4] << 11) | in_csc[5];
        *(uint32_t *)PRP_CSC_COEF_678= (in_csc[6] << 21) |
                          (in_csc[7] << 11) | in_csc[8] |
                                   (in_csc[9] << 31);

//resize control
//averaging, resize = 2:1
        *(uint32_t *)PRP_CH2_RZ_HORI_COEF1 = (0x4 << 3) | 0x4;
        *(uint32_t *)PRP_CH2_RZ_HORI_COEF2 = 0x0;
        *(uint32_t *)PRP_CH2_RZ_HORI_VALID = 0x02000002;
        *(uint32_t *)PRP_CH2_RZ_VERT_COEF1 = (0x4 << 3) | 0x4;
        *(uint32_t *)PRP_CH2_RZ_VERT_COEF2 = 0x0;
        *(uint32_t *)PRP_CH2_RZ_VERT_VALID = 0x02000002;
        *(uint32_t *)PRP_CH1_RZ_HORI_COEF1 = (0x4 << 3) | 0x4;
        *(uint32_t *)PRP_CH1_RZ_HORI_COEF2 = 0x0;
        *(uint32_t *)PRP_CH1_RZ_HORI_VALID = 0x02000002;
        *(uint32_t *)PRP_CH1_RZ_VERT_COEF1 = (0x4 << 3) | 0x4;
        *(uint32_t *)PRP_CH1_RZ_VERT_COEF2 = 0x0;
        *(uint32_t *)PRP_CH1_RZ_VERT_VALID = 0x02000002;

        return;
}
//csc coefficient generator
static void  csc_tbl(unsigned short csc[10])
{
//      csc     in      csc[0]=index      0..3 : A.1 A.0 B.1 B.0
//                      csc[1]=direction0 : YUV2RGB  1 : RGB2YUV
//              out     csc[0..4] are coefficients c[9] is offset
//                      csc[0..8] are coefficients c[9] is offset

        static const unsigned short     _r2y[][9] =
        {
                {0x4D, 0x4B, 0x3A, 0x57, 0x55, 0x40, 0x40, 0x6B, 0x29 },
                {0x42, 0x41, 0x32, 0x4C, 0x4A, 0x38, 0x38, 0x5E, 0x24 },
                {0x36, 0x5C, 0x25, 0x3B, 0x63, 0x40, 0x40, 0x74, 0x18 },
                {0x2F, 0x4F, 0x20, 0x34, 0x57, 0x38, 0x38, 0x66, 0x15 },
        };
        static const unsigned short     _y2r[][5] =
        {
                {0x80, 0xb4, 0x2c, 0x5b, 0x0e4},
                {0x95, 0xcc, 0x32, 0x68, 0x104},
                {0x80, 0xca, 0x18, 0x3c, 0x0ec},
                {0x95, 0xe5, 0x1b, 0x44, 0x1e0},
        };
        unsigned short*_csc;
        int               _csclen;

        csc[9] = csc[0] & 1;
        _csclen = csc[0] & 3;

        if (csc[1])
        {
                _csc = (unsigned short *)_r2y[_csclen];
                _csclen = sizeof(_r2y[0]);
        }
        else
        {
                _csc = (unsigned short *)_y2r[_csclen];
                _csclen = sizeof(_y2r[0]);
                memset(csc + 5, 0, sizeof(short) * 4);
        }
        memcpy(csc, _csc, _csclen);
}
```

The parameter, “ch1out” is the address at which the start of the frame for channel 1 output data is located, and “ch2y, ch2u, ch2v” are likewise the addresses where the start of frame for the channel 2 YUV data is located. The addresses for these memory buffers must be loaded into the destination registers for each channel, such as register PRP_DEST_RGB1_PTR for buffer 1 of channel 1.

3.2.1.3 Loop Mode vs. Single Frame Mode

The PrP can operate in either single or continuous frame (loop) mode, which is selected by the PRP_CNTL register. Each of the two channels of the PrP outputs data into two memory buffers. In loop mode, the data in the buffers are continuously updated in a ping pong fashion. This means data is written into buffer 1, then buffer 2, then buffer 1, then 2, and so on. so that new data is continuously received by the next module by reading from the last buffer that was updated.

In single frame mode, once a frame is captured, poll the PRP_INTRSTATUS register before enabling the PrP to capture the next frame from the CSI. This enablement is accomplished by setting the enable bit in the PRP_CNTL register for the desired channel(s). Both channels can be simultaneously enabled. Note that the enable bit for any channel that is set in single frame mode is self clearing on success or error, while in loop mode, it does not self clear.

The user may want to use single frame mode if he or she wants to modify the image before it is sent to the display module. Otherwise, if the data is immediately ready for display or transfer to memory, loop mode is desirable to maximize the refresh rate.

3.2.2 CMOS Sensor Interface (CSI)

The CSI enters data from the camera into the i.MX21. The main setup for the CSI-to-PrP interface and interrupt generation is done in CSI Control Register 1 (CSICR1). CSICR2 controls Bayer data from the camera (if Bayer data is being collected), and CSICR3 is an extension to CSICR1 adding additional control features. The GPIO and clock enables must also be set. Code Example 2 is for the 1.3MP camera. Notice that in the example, CSI_CSICR1 refers to the CSICR1 Register of the CSI module.

Example 2. 1.3MP Camera Interface to CSI

```
void CSI_init_PRP_YUV(void)
{
//CSI_module_init
//disable GPIO PB[21..10]
        * (uint32_t *)PTB_GIUS &= ~0x3FFC00;
//HCLK clock enable
        * (uint32_t *)PCCR0 |= 0x80000000;
//PERCLK4 enable
        * (uint32_t *)PCCR0 |= 0x00400000;
```

```
         *(uint32_t *) CSI_CSICR1 = 0x0;              //register clear
         *(uint32_t *) CSI_CSICR1 |= 0x1200;          //MCLK = HCLK / 4

//CCIR control
//       *(uint32_t *) CSI_CSICR1 |= 0x400;           //CCIR mode enable
//       *(uint32_t *) CSI_CSICR1 |= 0x8000000;       //CCIR interlace mode
//       *(uint32_t *) CSI_CSICR1 |= 0x40000000;      //external VSYNC
//timing control
         *(uint32_t *) CSI_CSICR1 |= 0x20000;        //SOF rising edge
         *(uint32_t *) CSI_CSICR1 |= 0x10000;        //SOF INT enable
         *(uint32_t *) CSI_CSICR1 |= 0x2;            //latch on rising edge
         *(uint32_t *) CSI_CSICR1 |= 0x10;           //gated clock mode
         *(uint32_t *) CSI_CSICR1 |= 0x800;          //hsync active high
//FIFO control
         *(uint32_t *) CSI_CSICR1 |= 0x100;          //sync FIFO clear
         *(uint32_t *) CSI_CSICR1 |= 0x40000;        //RXFF INT enable
         *(uint32_t *) CSI_CSICR1 |= 0x100000;       //RXFF level = 16
         *(uint32_t *) CSI_CSICR1 |= 0x1000000;      //RXFF overflow int
//data manipulation
         //*(uint32_t *) CSI_CSICR1 |= 0x80000000;   //swap16 enable
         *(uint32_t *) CSI_CSICR1 |= 0x80;           //big endian //Teri

//PRP i/f control
         *(uint32_t *) CSI_CSICR1 |= 0x10000000;     //PRP i/f enable

         return;
}
```

If the PrP is not being used, as with certain smart camera sensors, the PrP i/f enable bit of the CSICR1 register does not need to be set, and data received by the CSI can be accessed directly via the CSI RxFIFO register (CSIRFIFO). See the VGA_imgCapture demo for sample code to do this.

In some cases the camera sensor will have its own master clock and will not require a clock from the i.MX21 (PERCLK4 designated for the CSI). The I^{2}C module, discussed in the next section, controls the settings for this clock since it is on the camera.

3.2.3 I^{2}C

To enable the I^{2}C, the user must enable the GPIO ports assigned to it in hardware and enable the peripheral clock for the I^{2}C by doing the following:

```
         * (uint32_t *)PTD_GIUS &= ~0x60000;   //port select
         * (uint32_t *)PCCR0 |= 0x1000;        //clock enable
```

The serial bit clock frequency must be configured by setting the I^{2}C Frequency Divider Register (IFDR). See the chart in the chapter on the I^{2}C module of the i.MX21 Reference Manual. Note that the registers, I2C_IFDR and I2C_I2CR, refer to register names IFDR and I2CR respectively in the i.MX21 Reference Manual.

```
         * (uint32_t *)I2C_IFDR  = (uint32_t) I2C_CLKDIV;//clock select* (uint32_t
*)I2C_I2CR |= (uint32_t) 0x80;//I2C enable
         * (uint32_t *)I2C_I2CR |= (uint32_t) 0x08;//ack disable
```

Next, the user may choose the variety of options provided by the camera via a series of I^{2}C writes and/or reads of 8-bit data to and from the sensor registers. If the sensor registers are 16 bits long, each byte can be written separately, MSB first. See the i2c.c files in each of the demos for example procedures of I^{2}C reads and writes.

The following is an example of how to setup the IM8012 camera using the I^{2}C. For information on the IM8012 VGA camera's processor (MT9V111), visit www.micron.com.

Example 3. Setting Up the IM8012 Camera Using the I^{2}C

```
void IM8012_QVGA_RGB_init(void)
{
         uint32_t IC_rev, IFP_rev;
         uint32_t rdata;

printf("QVGA RGB\n");

//hard init
         SensorStandbyOff();   //enter active mode
         SensorReset();        //hard reset
         SensorRSlevel(0);     //set '0' for >= 15fps

//---------------------------
// Image Core (IC) Control
//---------------------------

         //===========
         //       Reset IC
         //===========
         I2C_write(0x01, 0x0004); //select IC register space
         I2C_write(0x0D, 0x0001); //put MT9V111 into reset mode
         I2C_write(0x0D, 0x0000); //resume operation
         I2C_read(0x36, &IC_rev); //check silicon ID
//       printf("IC rev = 0x%04X\n", IC_rev);


         //====================
         //       Pixel Control
         //====================
         I2C_read(0x20, &rdata);
//       printf("%02x\n", rdata);
         rdata |= 0x8080;//read mode ctrl: reverse row order, read out 1 row later
         rdata |= 0x4020;//read mode ctrl: reverse col order, read out 1 column later
         I2C_write(0x20, rdata);

//-------------------------------------
// Image Flow Processor (IFP) Control
//-------------------------------------

         //============
         //       Reset IFP
         //============
         I2C_write(0x01, 0x0001);    //select IFP register space
         I2C_write(0x07, 0x0001);    //reset by setting bit 0 to "1" and then "0"
         I2C_write(0x07, 0x0000);
         I2C_read(0x49, &IFP_rev);   //check silicon ID
//       printf("IFP rev = 0x%04X\n", IFP_rev);

         //========================
        //                          Output Format Control
        //========================
        I2C_write(0xA5, 0x8000);       //freeze update of horizontal decimation params
        I2C_write(0xA6, 0x8000 + 640); //freeze + original horizontal size
        I2C_write(0xA7, 0x8000 + 320); //freeze + horizontal output size
        I2C_write(0xA8, 0x8000);       //freeze update of vertical decimation parameters
        I2C_write(0xA9, 0x8000 + 480); //freeze + original vertical size
        I2C_write(0xAA, 0x8000 + 240); //freeze + vertical output size
        I2C_write(0xA5, 0x0000);       //enable resize (synchronous load of settings)

        I2C_write(0x08, 0xD800);       //RGB565 output
//      I2C_write(0x08, 0xCD00);       //YUV422 output

        return;
}
```

In the Output Format Control section of code Example 3, an input resolution of 640 × 480 and output of 320 × 240 is configured for the camera. This camera resizes without using the i.MX21's PrP. This protocol will differ from one camera to another, but the protocol for I^{2}C reading and writing will be very similar.

3.2.4 LCD Interface

The Liquid Crystal Display Controller (LCDC), module on the i.MX21 controls an LCD via the LCD interface on the i.MX21 EVB ADS Base Board. Please see Application Note AN2868 on *Using the i.MX21 to Create a VGA Digital Photo Album* and the *DigitalPhotoAlbum* application software online for a detailed overview about how to use the LCD controller. Please also refer to the i.MX21 Reference Manual on the chapter about the LCDC and the demo programs for more details and examples.

4 Choosing Cameras for i.MX21

4.1 I/O Power Supply

It is important that the I/O supply for the i.MX21 and the camera sensor are compatible to avoid damage to either of the devices. The camera must drive signals within the input range of the i.MX21 and vice versa. See the MC9328MX21 Data Sheet for recommended operating ranges for the I/O supply voltages.

4.1.1 Examples of How to Resolve I/O Supply Issues

All power supplies have a tolerance within which they can vary and are specified as 2V ± 10% (which is 1.8V to 2.2V), for instance. Since the i.MX21 I/O can be powered only as high as 3.3V, the camera's supply must not exceed 3.3V. This is because the inputs of the MX21 cannot be driven above the positive supply. For example, a camera power supply of 3.3V ± 1% will exceed this 3.3V limit, but 3V ± 10% is within this limit. Note that a CMOS camera's output logic high voltage (VOH) is typically equal to its supply voltage.

If the camera module connected to the CSI module requires a power supply of 3.0V to 3.6V and the CSI module requires that of 1.7V to 3.3V, then the design must do the following:

- Keep the camera from driving signals to the i.MX21 CSI module above 3.3V.

- Ensure that the i.MX21 drives the camera's inputs to at least the camera's VIH logic level (with some margin).

The next two sections provide two examples of how to do this. Note that the logic output low voltage (VIL) for either the i.MX21 or the camera is not an issue because they typically output close to 0V for logic low signals.

4.1.1.1 Logic Level Converter

Logic-level converters (LLC's) can be placed directly on the I/O lines between the camera module and the i.MX21. Various types of LLC's are available from several semiconductor suppliers. They are used to either reduce or increase logic voltage levels. Figure 1 illustrates this example.

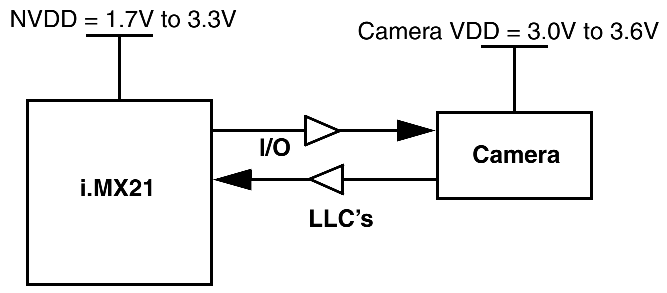

Figure 1. LLC's Inserted on I/O Lines

In general, the downside to this method is that each I/O line will require a connection to an LLC, which adds to component count on a circuit board. Additionally, the device introduces propagation delays that are undesirable for high frequency I/O transfers, such as fast memory accesses. For the camera interface, the propagation delay needs to be considered when selecting the LLC such that the delay does not cause violation of setup and hold times.

4.1.1.2 Tightened Supply Tolerance

Another approach is to use one power supply and tighten the supply tolerance. Depending on how tight the tolerance needs to be, the user may want to investigate this option. In the example discussed in Section 4.1.1, "Examples of How to Resolve I/O Supply Issues," the CSI and the camera had a common range of 3.0V to 3.3V. The supply lines can be restricted to this range by regulating the voltage (which might come from the i.MX21 ADS board) to the midpoint of this common range and with the highest tolerance to meet this range. See the following equations and Figure 2.

3.3V - 3.0V = 0.3V (total allowable supply variation)

0.3V ÷ 2 = 0.15V

3.0V + 0.15V = 3.15V (supply midpoint)

0.15V ÷ 3.15V = 0.04762 → *rounding down* to 4.7% (to avoid exceeding 3.3V limit)

Therefore, the new I/O supply lines must be regulated to 3.15V ± 4.7%.

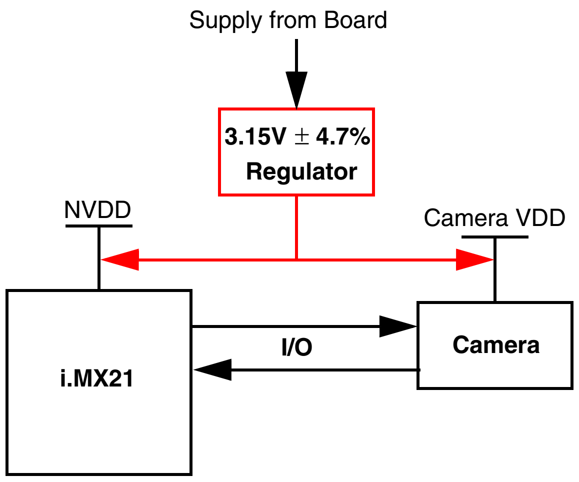

Figure 2. Using Direct Interface with Tighter Tolerance on Supply

This solution can result in a more expensive regulator, but eliminates the need for logic-level converters. The user must weigh his or her options when deciding how to resolve this issue.

4.2 Data Format

An appropriate CMOS camera sensor for the i.MX21 must be able to output a data format that the i.MX21 (specifically the eMMA modules) can process. The CSI will accept data of various formats, for example: YCC, YUV, Bayer, and RGB. The LCDC can accept only RGB data and a maximum resolution of 800 x 600. The PrP of the eMMA module can accept the following data formats: arbitrarily formatted RGB pixels (16 or 32 bits), YUV 4:2:2 (Pixel interleaved), YUV 4:2:0 (IYUV, YV12). The PrP can only accept frame sizes from 32 × 32 to 2044 × 2044 pixels (and a max of 2040 × 2040 pixels for YUV 4:2:0). Table 3 provides the data formats accepted by the PrP from the CSI (data from camera sensor) and from system memory.

Table 3. Input Data Formats for the Pre-Processor

Source	Format	Resolution
CSI	RGB	16bpp
	RGB	32 bpp (unpacked RGB888)
	YUV 4:2:2	Pixel interleaved
	YUV 4:4:4	32 bpp–pixel interleaved
Memory	RGB	16bpp
	RGB	32 bpp (Unpacked RGB888)
	YUV 4:2:2	Pixel interleaved
	YUV 4:2:0	Band interleaved (IYUV and YV12)
	YUV 4:4:4	32 bpp–pixel interleaved

See the chapters on the LCDC, CSI, and eMMA of the i.MX21 Reference Manual for more information.

The CSI supports generic sensor interface timing as well as CCIR656 video interface timing.

4.3 Camera Format

In choosing a CMOS camera sensor for the i.MX21, the user must verify that the camera is able to use i.MX21's CSI interface timing parameters, and be able to use control interfaces such as I^2C and Serial Peripheral Interface (SPI) modules. Other timing and data specific parameters related to the eMMA, SDRAM, and so on. must be checked to make sure there is a match between the customer's application and the i.MX21 processor interface.

5 Other Considerations

5.1 Camera Clock

As mentioned before, the modules need peripheral clocks for register accesses and system clocks for memory access. Some cameras have their own internal clock(s). The 1.3MPixel camera can provide its own master clock as an option, from which the pixel clock derives. In this case, the peripheral clock for the CSI, PERCLK4, does not need to be enabled.

5.2 Refresh Rate

To meet refresh rate requirements, the user must consider different data rates:

1. The rate (frames per second or fps) at which data is being sent to the system.
2. The refresh rate of the LCD, which is set by the LCDC.
3. Modules clocks (CSI and PrP) for register accesses, controlled by peripheral clock control and module control registers.
4. CPU processing (application software).

Although the refresh rate of the LCD may be set to 60 fps, for example, the video may only refresh at 15 fps due to the slower refresh rate of the camera. The 1.3MP camera has a maximum refresh rate of 15 fps when its resolution is set to SXGA (1280×960), but when it is set to VGA (640×320), the refresh rate can increase to an output of 30 fps from the camera. This camera setting is adjusted using the I^2C. The relationship between the resolution and the refresh rate depends on the capability of the camera.

Software routines to synchronize the data processing and do further processing, such as image rotation, can also reduce the final frame rate.

5.3 Image Orientation

The orientation of an image received by the camera may be different from the way the LCD displays the data. The cameras and LCD's used in the demos both required rotation software to convert the image from a portrait view to landscape. To rotate an image, the frames must be captured one at a time, rotated, and sent to the final module for display, thus it is better to rotate the image prior to display and after all other processing is complete.

5.3.1 Rotation of Data from PrP (Using “Dumb” Sensor)

If the PrP is being used, the user must verify that the destination line stride number for channel 1 matches the output dimensions of the image data. The PrP Destination Channel-1 Line Stride Register (PRP_DEST_CH1_LINE_STRIDE) determines the distance in bytes between the start addresses of adjacent lines in the Channel-1 output. For example, if the data is in the 2-bytes-per-pixel format and the final image (after rotated) has 320 pixels from left to right (horizontally), then the user must program the number 640 (= 320 × 2) into this register.

The PrP must be set to single frame mode instead of loop mode by the PRP_CNTL Register, and a frame complete bit(s) needs to be polled within the PRP_INTRSTATUS Register before the next frame is captured. Section 3.2.1.3, “Loop Mode vs. Single Frame Mode” provides more discussion about single frame and loop mode. Figure 3 is an example algorithm.

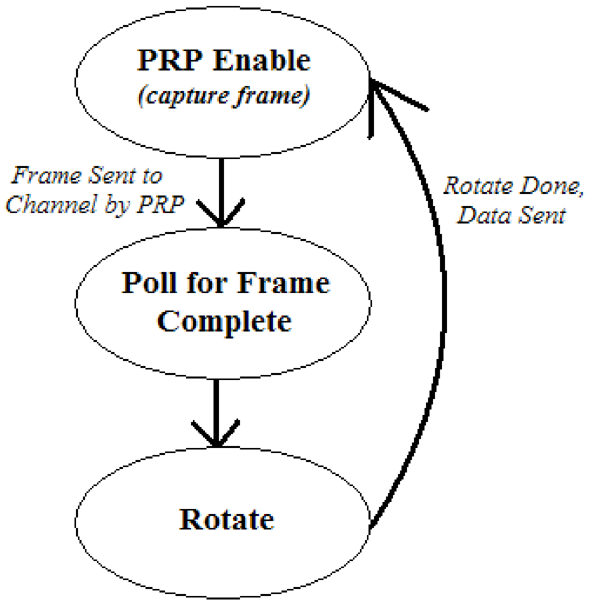

Figure 3. Example Algorithm for Rotating Data from PrP Channel

5.3.2 Rotation of Data from CSI (Using “Smart” Sensor)

Because the VGA camera is a *smart* camera that does not need the PrP to do additional data processing, the data can be retrieved directly from the CSI by (1) reading the CSI RxFIFO Register (CSIRFIFO) and (2) updating memory (or display buffer for display) each time this FIFO is full. The FIFO full status is checked by polling the CSI Status Register (CSISR). As soon as the entire frame is transferred, the LCDC can output the data in a display buffer, which can be allocated by software in memory.

The VGA demo software collects the image data from the CSI Rx FIFO Register, rotates the images, and sends them to the LCDC display buffer for display before capturing the next frame as described in the preceding paragraph. See the VGA demo for example code.

5.3.3 Rotation Example

Example 4 provides a software rotation routine for the clockwise rotation of 16bpp image data.

Example 4. Software Rotation Routine

```
void RGB565Rotate(uint32_t BufIn, uint32_t BufOut, uint32_t imgH, uint32_t imgV, uint32_t dir)
{
        int32_t v, h, k;
        uint16_t * _dataIn = (uint16_t *)BufIn;
        uint16_t * _dataOut = (uint16_t *)BufOut;

        switch(dir)
        {
                case 90:
                {
                        k = 0;
                        for(h = 0; h < imgH; h ++)
                        {
                                for(v = imgV - 1; v >= 0; v --)
                                {
                                        _dataOut[k] = _dataIn[v * imgH + h];
                                        k ++;
                                }
                        }
                        break;
                }
                case 270:
                {
                        k = 0;
                        for(h = imgH - 1; h >= 0; h --)
                        {
                                for(v = 0; v < imgV; v ++)
                                {
                                        _dataOut[k] = _dataIn[v * imgH + h];
                                        k ++;
                                }
                        }
                        break;
                }
        }
        return;
}
```

The pixels are simply rearranged and placed in a new buffer, BufOut, as shown in Figure 4. Note that each number represents one pixel in memory. Therefore, the data in memory converts from the order shown as 1, 2, 3, 4, 5, 6, … to 10, 7, 4, 1, 11, 8, 5, 2, … .

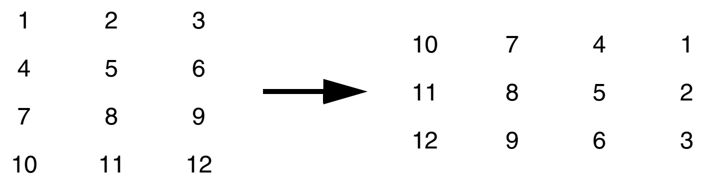

Figure 4. Clockwise 90° Degree Rotation

5.4 Tearing Effect

5.4.1 Why It Occurs

The image displayed may distort in a way that causes part of the image to change while the other part stays the same for a fraction of a second on several frames. This is most likely the cause of the asynchronous refresh rates for the different modules involved in the image capture and the way the data buffers are setup. If the LCDC retrieves data from the same buffer that is being updated by another module, such as the CSI or PrP, the image data for one particular frame may be refreshed while that frame is in mid-transfer to the LCD.

5.4.2 How to Fix It

There are several ways to resolve this issue. One way is to create a mid-point buffer that will be updated only when the LCD is ready to receive a new frame as illustrated Figure 5.

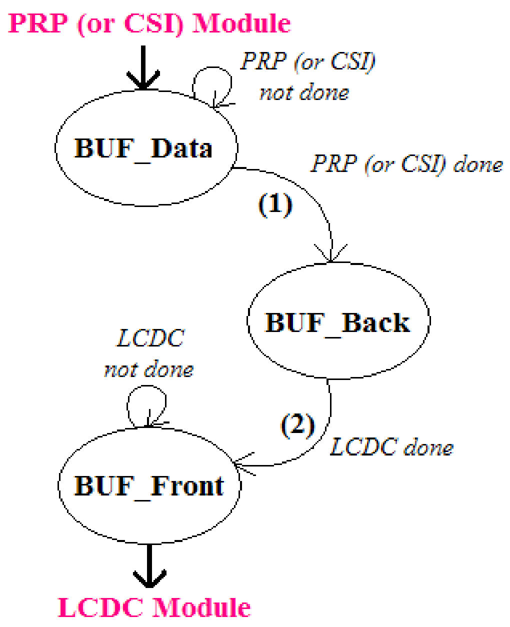

Figure 5. Data Flow for Code to Fix Tearing

In step (1), the data for one frame is transferred from buffer BUF_Data to BUF_Back when the PrP (or CSI) is finished updating BUF_Data with the entire frame. Otherwise, BUF_Back is not updated. Likewise in step (2), the LCDC waits until it is finished displaying the frame data that is in BUF_Front before updating BUF_Front with the frame data in BUF_Back. This method is illustrated in the demo programs.

5.5 Data Format

5.5.1 Endianess

The user must check the endianess of the data by referring to the camera sensor documentation. The CSI will receive the data in whichever format the user chooses in control register CSICR1. If the incorrect endianess is selected, the image will likely distort.

5.5.2 Output Format

The I^{2}C is used to select various camera settings, including the output format of the image data. Depending on the format received by the CSI, the PrP may need to be programmed to convert the data to either RGB for display and/or YUV420 for encoding or post-processing. Application Note AN2676 and the i.MX21 Reference Manual provides additional information.

6 Conclusion

The main purpose of this document is to elaborate on how to do image capture, including details on the PrP, CSI, and I^{2}C modules and considerations for selecting and using camera sensors of varying capabilities. Depending on the camera sensor used, the user may need to consider image rotation mechanisms, camera resolution settings, and whether the PrP is needed. The PrP is needed any time the data outputted to the CSI by the camera module is not in the proper format or size required for its next purpose, such as display, encoding, or post-processing. The user must also consider the effects of *tearing* and camera frame rates.

7 Reference Documents

The following i.MX technical documents may be found at the Freescale Semiconductor Inc. World Wide Web site at http://www.freescale.com/imx. These documents may be downloaded directly from the World Wide Web site.

MC9328MX21 Reference Manual (order number MC9328MX21RM)

MC9328MX21 Data Sheet (order number MC9328MX21)

Image Capture with MC9328MX21 Application Note (order number AN2627)

Operating Principle of Resize in the eMMA Pre-Processor Application Note (order number AN2886)

Using the MC9328MX21 (i.MX21) to Create a VGA Digital Photo Album Application Note (order number AN2868)

8 Revision History

Rev. 0 is the initial release of this document.

Document Number: AN3285
Rev. 0
08/2006

freescale™
semiconductor