MT9D131 Developer Guide

1/3.2-Inch, 2Mp SOC CMOS Digital Image Sensor

About this Guide

The MT9D131 is a complete system-on-a-chip solution. It incorporates sophisticated, on-chip camera functions and is programmable through a simple two-wire serial interface. The developer guide is a thorough reference for engineers who wish to develop applications for the MT9D131. The guide provides detailed information on working with chip registers and variables and explains how to use Micron's developer software—DevWare. The MT9D131 data sheet should be used along with this guide as a referenece for specific register and programing information.

Table of Contents

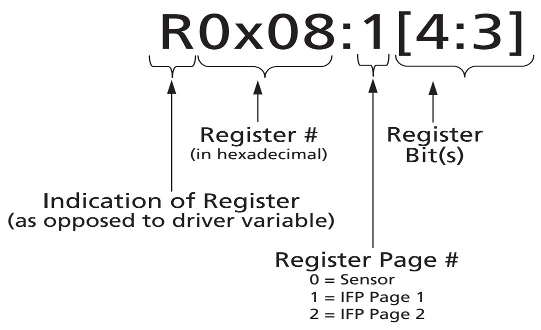
List of Figures

List of Tables

Introduction to Registers

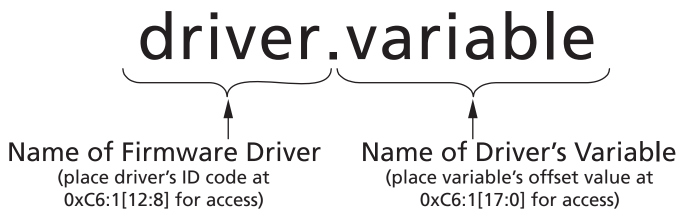
This developer guide refers to various memory locations and registers that the user reads from or writes to for altering the MT9D131 operation. Hardware registers appear as follows and may be read or written by sending the address and data information over the two-wire serial interface.

Figure 1: Register Legend



Other memory locations are within the microcontroller block and may be accessed by using hardware registers from 0xC6:1 through 0xD1:1 (see the MT9D131 data sheet for further details on how to use these registers). These are denoted below:

Figure 2: Firmware Variable Legend



The MT9D131 was designed to facilitate customization to optimize image quality processing. As the image data travel through the various stages of image processing, the user can adjust the parameters in these stages to affect the images' appearances. This section describes most of these available adjustments.

Two-Wire Serial Interface

Overview

The only external control interface to the MT9D131 is a two-wire serial interface. This chapter shows how to access the MT9D131 registers. For the complete specifications, refer to the MT9D131 data sheet.

The MT9D131 contains three pages of registers as well as the firmware-driver variables. Each page has 256 address locations, and each location is 16 bits wide. Not all locations and bits are accessible (refer to the register table for detailed information on each register). Included in these three pages is the indirect access for MCU (drivers) and JPEG memory.

The bus address of the two-wire serial interface is selectable between two sets of values. Changing the state of the hardware pin, SADDR, or of R0x0D:0[10] selects between them as follows:

Table 1: Selecting Values in the Bus Address (Read/Write)

Register	SADDR Set HIGH	SADDR Set LOW
R0x0D:0[10] = 0 (default)	0xBB/0xBA	0x91/0x90
R0x0D:0[10] = 1	0x91/0x90	0xBB/0xBA

Table 2: Register Page Description

Page	Description
Page 0: Sensor	Sensor and PLL control.
Page 1: SOC 1	SOC Image processing, and MCU register, MCU memory indirect access.
Page 2: SOC 2	JPEG control and Soc control. JPEG indirect memory access.

Page Selection Register: R0xF0:0

Register 0xF0:0 is a unique register; it is used to select which of the three pages are active when reading or writing. Physically there is only one register—regardless of which page is selected, it accesses the same content.

Reg 0xF0:0 = 0x0000 = > Page 0, Sensor

Reg 0xF0:0 = 0x0001 = > Page 1, SOC 1

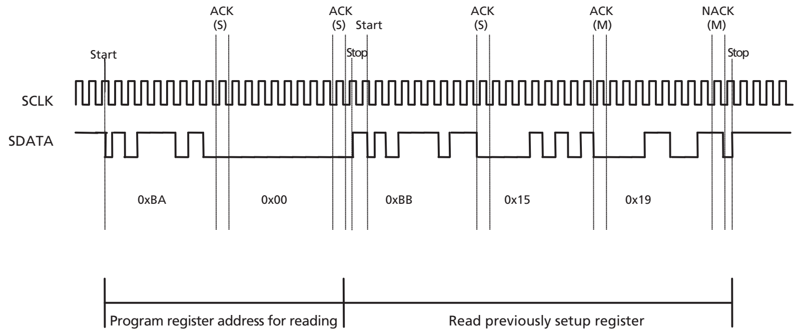
Reg 0xF0:0 = 0x0002 = > Page 2, SOC 2

Example: 16-Bit Register Read

This is an example of a 16-bit register read from Chip Version register (R0x00:0), expected value = 0x1519

1. Send Start
2. Send Device Address
   a. 0xBA
3. Wait for ack
4. Send register address (8-bit)
   a. 0x00
5. Wait for ack
6. Send Stop
7. Send Start
8. Send device address for read
   a. 0xBB
9. Wait for ack
10. Slave device sends 8-bit data (MSB byte)
    a. 0x15
11. Master send ack
12. Slave device sends another 8-bit data (LSB byte)
    a. 0x19
13. Master send nack
14. Send Stop

Figure 3: Example of 16-Bit Register Read from Chip Version Register (R0x00:0), Value = 0x1519

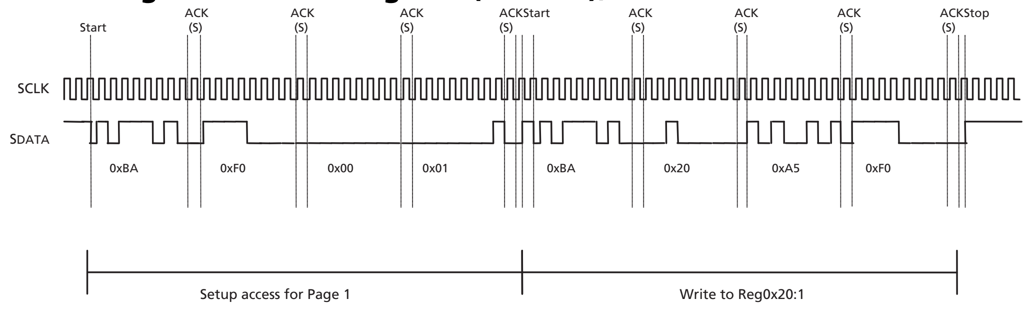


Example: 16-Bit Register Write

This is an example of a 16-bit register write to register (R0x20:1), value = 0xA5F0

1. Send Start
2. Send Device Address
   a. 0xBA
3. Wait for ack
4. Send register address (8-bit)
   a. 0xF0
5. Wait for ack
6. Send 8-bit data (MSB byte)
   a. 0x00
7. Wait for ack
8. Send another 8-bit data (LSB byte)
   a. 0x01
9. Wait for ack
10. Send Stop
11. Send Start
12. Send Device Address
    a. 0xBA
13. Wait for ack
14. Send register address (8-bit)
    a. 0x20
15. Wait for ack
16. Send 8 bit data (MSB byte)
    a. 0xA5
17. Wait for ack
18. Send another 8-bit data (LSB byte)
    a. 0xF0
19. Wait for ack
20. Send Stop

Figure 4: Example of 16-Bit Register Write to Register (R0x20:1), Value = 0xA5F0



Accessing the Firmware Drivers' Variables

One register (R0xC6:1) is used for the memory address and another (R0xC8:1) is used for data in the address.

Write Access

A write to the indirect access data register triggers a write to the targeted memory after the two-wire serial interface has completed the WRITE cycle.

Read Access

Data is pre-fetched once the indirect access address register is updated; therefore, when the user reads from the indirect access data register, the data is available.

MCU Memory

R0xC6:1[15:0], Indirect Access Address Register

R0xC8:1[15:0], Indirect Access Data Register

Table 3: R0xC6:1[15:0], Indirect Access Address Register

Bit	Description
7:0	Bits 7:0 of address for physical access; driver variable offset for logical access.
12:8	Bits 12:8 of address for physical access; driver ID for logical access.
14:13	Bits 14:13 of address for physical access; R0xC6:1[14:13] = 01 select logical access.
15	1 = 8-bit access; 0 = 16-bit access

JPEG Memory

R0x1E:2[15:0], Indirect Access Address Register

R0x1F:2[15:0], Indirect Access Data Register

Table 4: R0x1E:2[15:0], JPEG Indirect Access Control Register

Bit	Description
10:0	Indirect access address register: This 11-bit register contains the address of the register or memory to be accessed indirectly.
12:11	Unused.
13	Enable two-wire serial interface burst: When this bit is set, the two-wire serial interface decoder operates in burst mode for the indirect data register (READ burst and WRITE burst). The longest burst supported is 16 (128 READ or WRITE cycles).
14	Enable indirect writing: When set, data from the indirect data register is written to the Indirect address location specified by [10:0] of this register except when auto-increment is set. Reading the same address location when this bit is reset to "0.”
15	Address auto-increment: When this bit is set, the value in the indirect access address register is automatically incremented after every read or write, to the JPEG indirect access data register. This feature is used to emulate a burst access to memory or registers being accessed indirectly.

Initializing the MT9D131

Power-up Sequence

There are no specific requirements to the order in which different supplies are turned on. Once the last supply is stable within the valid ranges specified below, follow the hard reset sequence to complete the power-up sequence.

Analog Voltage: 2.8V for best image performance

Digital Voltage: 1.8V ±0.1V (1.7V–1.9V)

I/O Voltage: 1.7V–3.1V

Hard Reset Sequence

After power-up, a hard reset is required. Assuming all supplies are stable, the assertion of RESET_BAR (active LOW) will set the device in reset mode. The clock is required to be active when RESET_BAR is released. Hence, leaving the input clock running during the reset duration is recommended. After 24 clock cycles (EXTCLK), the two-wire serial interface is ready to accept commands on the two-wire serial interface.

Note: Reset should not be activated while STANDBY is asserted.

To activate a hard reset sequence to the camera:

1. Wait for all supplies to be stable.
2. Assert RESET_BAR (active LOW) for at least 1µs.
3. De-assert RESET_BAR (input clock must be running).
4. Wait 24 clock cycles before using the two-wire serial interface.

Soft Reset Sequence

To activate a soft reset to the camera:

1. Bypass the PLL, R0x65:0 = 0xA000, if it is currently used.
2. Perform MCU reset by setting R0xC3:1 = 0x0501.
3. Enable soft reset by setting R0x0D:0 = 0x0021. Bit 0 is used for the sensor core reset while bit 5 refers to SOC reset.
4. Disable soft reset by setting R0x0D:0 = 0x0000.
5. Wait 24 clock cycles before using the two-wire serial interface.

Note: No access to the MT9D131 registers—both page 1 and page 2—is possible during soft reset.

Standby Sequence

Standby mode can be activated by two methods:

1. The first method is to assert STANDBY, which places the chip into hard standby. Turning off the input clock (EXTCLK) reduces the standby power consumption to the maximum specification of 100µA at 55°C. There is no serial interface access for hard standby.
2. The second method is activated through the serial interface by setting R0x0D:0[2] = 1 known as the soft standby. As long as the input clock remains on, the chip will allow access through the serial interface in soft standby.

Standby should only be activated from the preview mode (context A), and not the capture mode (context B). In addition, the PLL state (off/bypassed/activated) is recorded at the time of firmware standby (seq.cmd = 3) and restored once the camera is out of firmware standby. In both hard and soft standby scenarios, internal clocks are turned off and the analog circuitry is put into a low power state. Exit from standby must go through the same interface as entry to standby. If the input clock is turned off, the clock must be restarted before leaving standby.

To Enter Standby

1. To prepare for standby:
   a. Issue the STANDBY command to the firmware by setting *seq.cmd = 3*
   b. Poll *seq.state* until the current state is in standby (*seq.state = 9*)
   c. Bypass the PLL if used by setting R0x65:0[15] = 1
2. To prevent additional leakage current during standby:
   a. Set R0x0A:1[7] = 1 to prevent elevated standby current. It will control the bidirectional pads DOUT, LINE_VALID, FRAME_VALID, PIXCLK.
   b. If the outputs are allowed to be left in an unknown state while in standby, the current can increase. Therefore, either have the receiver hold the camera outputs HIGH or LOW, or allow the camera to drive its outputs to a known state by setting R0x0D:0[6] = 1. R0x0D:0[4] needs to remain at the default value of “0.” In this case, some pads will be HIGH while some will be LOW. For dual camera systems, at least one camera has to be driving the bus at any time so that the outputs will not be left floating.
   c. Configure internal reserved I/O as outputs and drive LOW by the setting the respective bit to “0” in the reserved I/O variables 0x1078, 0x1079, 0x1070, and 0x1071 (accessed through R0xC6:1 and R0xC8:1). The following settings should be used:
      i. R0xC6:1 = 0x9078
      ii. R0xC8:1 = 0x0000
      iii. R0xC6:1 = 0x9079
      iv. R0xC8:1 = 0x0000
      v. R0xC6:1 = 0x9070
      vi. R0xC8:1 = 0x0000
      vii. R0xC6:1 = 0x9071
      viii. R0xC8:1 = 0x0000
3. To put the camera in standby:
   Assert STANDBY = 1.
   Optionally, stop the EXTCLK clock to minimize the standby current specified in the MT9D131 data sheet.
   For soft standby, program standby R0x0D:0[2] = 1 instead.

To Exit Standby

1. De-assert standby:
   a. Provide EXTCLK clock, if it was disabled when using STANDBY.
   b. De-assert STANDBY = 0 if hard standby was used.
      Or program *R0x0D:0[2] = 0* if soft standby was used.
2. Reconfiguring output pads:
   a. Go to preview.
   b. Issue a GO_PREVIEW command to the firmware by setting *seq.cmd = 1*.
   c. Poll *seq.state* until the current state is preview (*seq.state = 3*).

The following timing requirements should be met to turn off EXTCLK during hard standby:

1. After the asserting standby, wait 10 clock cycles before stopping the clock.
2. Restart the clock 24 clock cycles before de-asserting standby.

PLL Setup

The PLL output frequency is determined by three constants (M, N, and P) and the input clock frequency. These three values are set in:

- R0x66:0 // [15:8] for M; [5:0] for N
- R0x67:0 // [6:0] for P

Their relations can be shown by the following equation:

$$f_{OUT} = f_{IN} \text{ x } M \text{ / } [2 \text{ x } (N+1) \text{ x } (P+1)] \qquad \text{(EQ 1)}$$

However, since the following requirements must be satisfied, then not all combinations of M/N/P are valid:

- M must be 16 or higher
- f_{PFD}, f_{VCO}, f_{OUT} ranges are satisfied

Table 5: Frequency Parameters

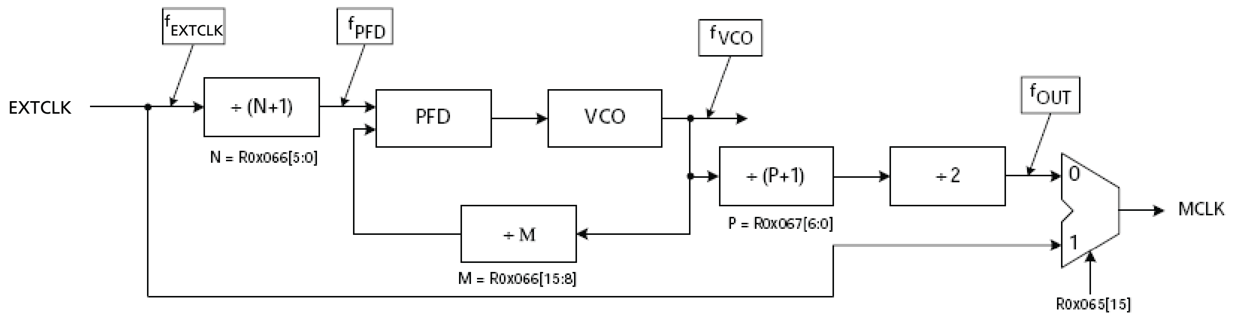
Frequency	Equation	Min (MHz)	Max (MHz)
f_{PFD}	f_{IN}/ (N+1)	2	13
f_{VCO}	f_{PFD} x M	110	240
f_{OUT}	f_{VCO} / [2 x (P+1)]	6	80
f_{IN}	—	6	64

After determining the proper M, N, and P values and setting them in R0x066:0/R0x067:0, the PLL can be enabled by the following sequence:

1. R0x65:0[14] = 0 // powers on PLL
2. R0x65:0[15] = 0 // disable PLL bypass (enabling PLL)

Note: If PLL is used, bypass the PLL (R0x65:0[15] = 1) before going into hard standby. It can be enabled again (R0x65:0[15] = 0) once the sensor is out of standby.

Figure 5: PLL Block Diagram



Identifying Chip Version

The sensor version as well as the firmware version can be determined by reading its respective register/variable.

R0x00:0 = 0x1519 // sensor chip version #

mon.ver (ID = 0, Offset = 12, 8-bit variable) // firmware version #

Context Switching and Output Configuration

Context Switch and Setup

There are two contexts (or modes) available, A and B. Context A is known as the preview mode and has a default resolution of 800 x 600, while context B is called the capture mode with a default resolution of 1600 x 1200.

To switch from preview to the capture state, set the following variables:
- seq.captureParams.mode[1] = 1// ID = 1, Offset = 0x20
- seq.cmd = 2// ID = 1, Offset = 0x03

To switch from capture back into preview mode, use the following settings:
- seq.captureParams.mode[1] = 0// ID = 1, Offset = 0x20
- seq.cmd = 1// ID = 1, Offset = 0x03

The current context mode can also be read through the serial interface from the variable mode.context (DriverID = 7, Offset = 0x02). If mode.context is read back as logic "1", the capture mode is active. When the bit is cleared, the current mode is preview.

For each context, there is a set of variables that enables the user to configure its properties. These settings are automatically put into effect by the firmware during context switching. Examples of configurable options are: output resolution, crop sizes, data format, output FIFO, spoof mode, slew rate, special effects, and gamma table. For more information, see the related description in this chapter or “Mode Driver–Setting Up Preview (A) and Capture (B) Modes” on page 62."

Changing the Output Resolution

To change the output size of a context, the associated context image height and width values can be updated before switching to the targeted context. If the size is changed for the current (active) context, then a refresh command (*seq.cmd = 5*) needs to be executed additionally.

For context A, use the variable *mode.Output Width A* (ID = 7, offset = 3) and *mode.Output Height A* (ID = 7, offset = 5).

For context B, use the variable *mode.Output Width B* (ID = 7, offset = 7) and *mode.Output Height B* (ID = 7, offset = 9). These settings only change the output image size and do not have any effect on the frame rate or field of view.

Selecting Output Data Formats

The MT9D131 can output several different formats. They are YCbCr, 565RGB, 555RGB, 444RGB, JPEG, and raw data.

To select between YUV and RGB, bit 5 of variables *mode.output_format_A* (ID = 7, Offset = 0x7D) and *mode.output_format_B* (ID = 7, Offset = 0x7E) should be used, depending on which context mode is of interest. Within RGB, 565/555/444x/x444 modes can be chosen by bits 6–7 of the same variable:

Table 6: Changing Output Format Variables

Bits[7:6]	RGB Mode
00	16-bit 565RGB
01	15-bit 555RGB

Table 6: Changing Output Format Variables

Bits[7:6]	RGB Mode
10	12-bit 444xRGB
11	12-bit x444RGB

A refresh is needed (*seq.cmd* = *5*) before the new settings are effective. As for JPEG images, only context B has support for it. See "Enabling and Capturing JPEG" on page 28 for more details.

Other bits for mode.output_format_A (ID = 7, Offset = 0x7D) and mode.output_format_B (ID = 7, Offset = 0x7E) are used for:

Table 7: Output Format Option Configuration Settings

Bit	Option / Configuration
0	In YUV output mode, setting this bit high would swap Cb and Cr channels. In RGB mode, it will swap the R and B channel. This bit is subject to synchronous update.
1	Setting this bit high would swap the chrominance byte with luminance byte in YUV output. In RGB mode, it will swap odd and even bytes. This bit is subject to synchronous update.
2	Progressive Bayer.
3	Monochrome output.

To select the 10-bit raw data from the sensor core, see "Raw Bayer Data Output."

Raw Bayer Data Output

There are two ways to obtain raw Bayer data. In both cases, the data from the sensor core bypasses the color pipeline. Hence, it does not go through the any of its image processing units (however, AE, AWB, and so on may still be active).

The first option is to output all 8 most significant bits in parallel (Bayer 8). In this case, DOUT0–DOUT7 represent sensor core data D[9:2].

1. In order to put the sensor in this mode, the MCU (microcontroller unit) should first be disabled by setting: R0x03:1 = 1// disable MCU
2. Next, enable the bypass mode with: R0x09:1 = 0// enable sensor core bypass

The pad slew while in bypass can be set using R0x0A:1[2:0]. With the color pipeline and MCU disabled, the sensor core parameters (integration time, gains, image size, power mode, etc.) can be programmed manually.

The second option is to output all 10 bits by enabling "8+2 bypass" mode. In this mode (Bayer 8+2), the data bits are sent out in two bytes: D9–D2 in the first byte, and D1–D0 (with 0s padded in the more significant bit positions) for the second byte.

To enable the Bayer 8+2 bypass mode:

1. Program register R0x09:1[2:0] = 1 to set the proper data flow.
2. Turn on "8+2" with R0x98:1[6] = 1.

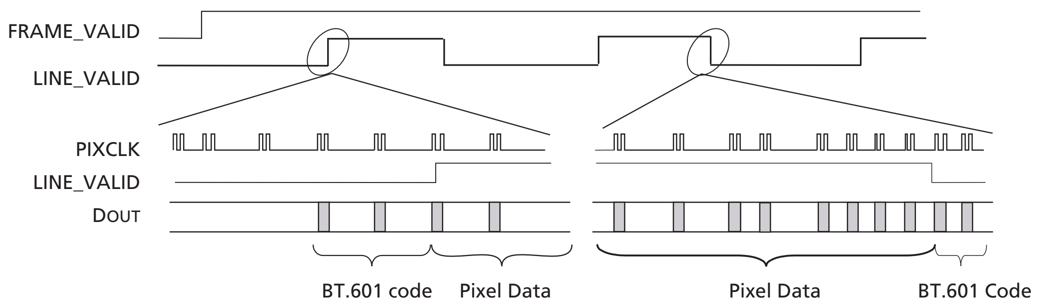
Table 8: Bayer 8+2 Mode Output Data Order

Mode	Byte	DOUT7	DOUT6	DOUT5	DOUT4	DOUT3	DOUT2	DOUT1	DOUT0
Bayer 8+2	First	B9	B8	B7	B6	B5	B4	B3	B2
	Second	0	0	0	0	0	0	B1	B0

Output Format and Timing

Uncompressed YUV or RGB data can be output either directly from the output formatting block or through a FIFO buffer with a capacity of 1,600 bytes, enough to hold one half uncompressed line at full resolution. Buffering of data is a way to equalize the data output rate when image decimation is used. Decimation produces an intermittent data stream consisting of short high-rate bursts separated by idle periods. Figure 6 depicts such a stream. High pixel clock frequency during bursts may be undesirable due to EMI concerns.

Figure 6: Timing of Decimated Uncompressed Output Bypassing the FIFO



Note: PIXCLK default inverted.

Figure 7 depicts the output timing of uncompressed YUV/RGB when a decimated data stream is equalized by buffering or when no decimation takes place. The pixel clock frequency remains constant during each LINE_VALID high period. Decimated data are output at a lower frequency than full size frames, which helps to reduce EMI.

Figure 7: Timing of Uncompressed Full Frame or Decimated Output Passing Through the FIFO

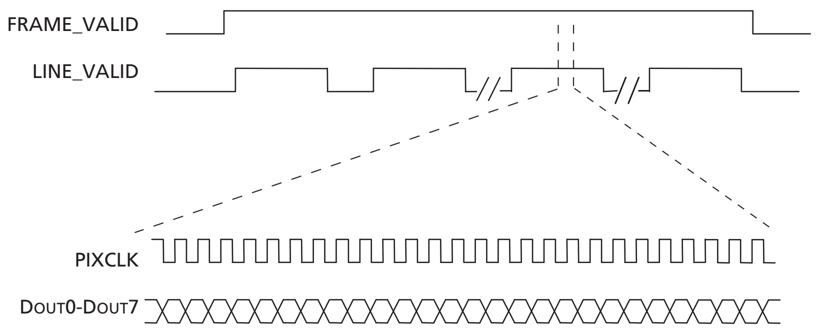
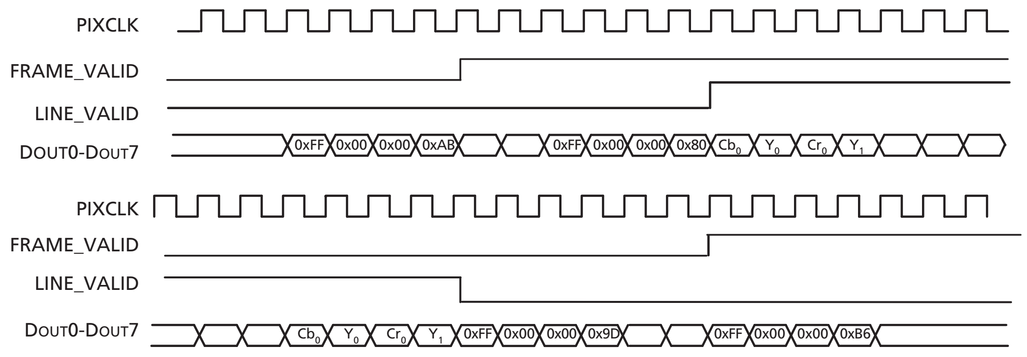


Figure 8: Example of Timing for Non-Decimated Uncompressed Output Bypassing Output FIFO



The MT9D131 supports swapping YCrCb mode as illustrated in Table 9, “YCrCb Output Data Ordering,” on page 20.

Table 9: YCrCb Output Data Ordering

Mode				
Default (no swap)	Cb_i	Y_i	Cr_i	Y_{i+1}
Swapped CrCb	Cr_i	Y_i	Cb_i	Y_{i+1}
Swapped YC	Y_i	Cb_i	Y_{i+1}	Cr_i
Swapped CrCb, YC	Y_i	Cr_i	Y_{i+1}	Cb_i

The RGB output data ordering in default mode is shown in Table 10. The odd and even bytes are swapped when luma/chroma swap is enabled. R and B channels are bitwise swapped when chroma swap is enabled.

Table 10: RGB Ordering in Default Mode

Mode (Swap Disabled)	Byte	$D_7D_6D_5D_4D_3D_2D_1D_0$
565RGB	Odd	$R_7R_6R_5R_4R_3G_7G_6G_5$
	Even	$G_4G_3G_2B_7B_6B_5B_4B_3$
555RGB	Odd	0 $R_7R_6R5_5R_4R_3G_7G_6$
	Even	$G_4G_3G_2B_7B_6B_5B_4B_3$
444xRGB	Odd	$R_7R_6R_5R_4G_7G_6G_5G_4$
	Even	$B_7B_6B_5B_4$ 0 0 0 0
x444RGB	Odd	0 0 0 0 $R_7R6_6R_5R_4$
	Even	$G_7G_6G_5G_4B_7B_6B_5B_4$

JPEG compression of IFP output produces a data stream structure that differs from that of an uncompressed YUV/RGB stream. Frames are no longer sequences of lines of constant length. This difference is reflected in the timing of the LINE_VALID signal. When JPEG compression is enabled, logical HIGHs on LINE_VALID do not correspond to image lines, but to variably sized packets of valid data. In other words, the LINE_VALID signal is in fact a DATA_VALID signal. It is a good analogy to compare the JPEG output of the MT9D131 to an 8-bit parallel data port wherein the LINE_VALID signal indicates valid data and the FRAME_VALID signal indicates frame timing.

The JPEG compressed data can be output either continuously or in blocks simulating image lines. The latter output scheme is intended to spoof a standard video pixel port connected to the MT9D131 and for that purpose treats JPEG entropy-coded segments as if they were standard video pixels. In the continuous output mode, JPEG output clock can be free-running or gated. In all, three timing modes are available and are depicted in Figure 7 on page 18, Figure 8 on page 19, and Figure 9 on page 21. These timing diagrams are merely three typical examples of many variations of JPEG output. The “continuous” and spoof JPEG output modes differ primarily in how the LINE_VALID output is asserted. In the continuous mode, LINE_VALID is asserted only during output clock cycles containing valid JPEG data. The resulting LINE_VALID signal pattern is nonuniform and highly image dependent, reflecting the inherent nature of JPEG data stream. In the spoof mode, LINE_VALID is asserted and de-asserted in a more uniform pattern emulating uncompressed video output with horizontal blanking intervals. When LINE_VALID is de-asserted, available JPEG data are not output, but instead remain in the FIFO until LINE_VALID is asserted again. During the time when LINE_VALID is asserted, the output clock is gated off whenever there is no valid JPEG data in the FIFO.

Note: As a result, spoof "lines" containing the same number of valid data bytes may be output within different time intervals depending on constantly varying JPEG data rate.

The host processor configures the spoof pattern by programming the total number of LINE_VALID assertion intervals, as well as the number of output clock periods during and between LINE_VALID assertions. In other words, the host processor can define a temporal "frame" for JPEG output, preferably with "size" tailored to the expected JPEG file size. If this frame is too large for the total number of JPEG bytes actually produced, the MT9D131 either de-asserts FRAME_VALID or continues to pad unused "lines" with zeros until the end of the frame. If the frame is too small, the MT9D131 either continues to output the excess JPEG bytes until the entire JPEG compressed image is output or discards the excess JPEG bytes and sets an error flag in a status register accessible to the host processor.

In the continuous mode, the JPEG output clock can be configured to be either gated off or running while LINE_VALID is de-asserted. To save extra power, the JPEG output clock can also be gated off between frames (when FRAME_VALID is de-asserted) in both continuous and spool output mode. In the continuous output mode, there is an option to insert JPEG SOI (0xFFD8) and EOI (0xFFD9) markers respectively before and after valid JPEG data. SOI and EOI can be inserted either inside or outside the FRAME_VALID assertion period, but always outside LINE_VALID assertions.

The output order of even and odd bytes of JPEG data can be swapped in the spoof output mode. This option is not supported in the continuous mode.

Output clock speed can optionally be made to vary according to the fullness of the FIFO, to reduce the likelihood of FIFO overflow. When this option is enabled, the output clock switches at three fixed levels of FIFO fullness (25 percent, 50 percent and 75 percent) to a higher or lower frequency, depending on the direction of fullness change. The set of possible output clock frequencies is restricted by the fact that its period must be an integer multiple of the master clock period. The frequencies to be used are chosen by programming three output clock frequency divisors in the mode driver FIFO variables (mode.fifo_conf1_A/B and mode.fifo_conf2_A/B). Divisor N1 is used if the FIFO is less than 50 percent full and last fullness threshold crossed has been 25 percent. When the FIFO reaches 50 percent and 75 percent fullness, the output clock switches to divisor N2 and N3, respectively. When the FIFO fullness level drops to 50 percent and 25 percent, the output clock is switched back to divisor N2 and N1, respectively.

The host processor can read registers containing JPEG status flags and JPEG data length (total byte count of valid JPEG data) via a two-wire serial interface. In addition, the JPEG data length and JPEG status byte are always appended at the end of JPEG spoof frame. JPEG status byte can be optionally appended at the end of JPEG continuous frame. JPEG data stream sent to the host does not have a header.

Figure 9: Timing of JPEG Compressed Output in Free-Running Clock Mode

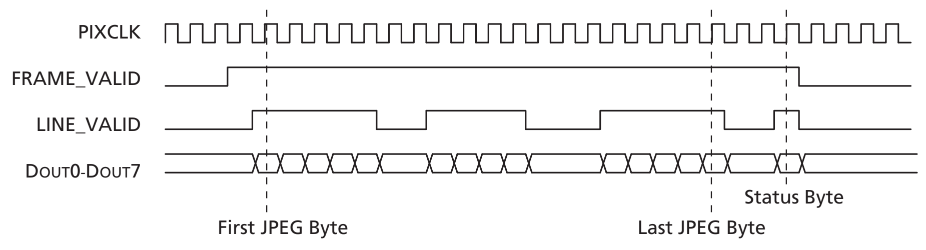
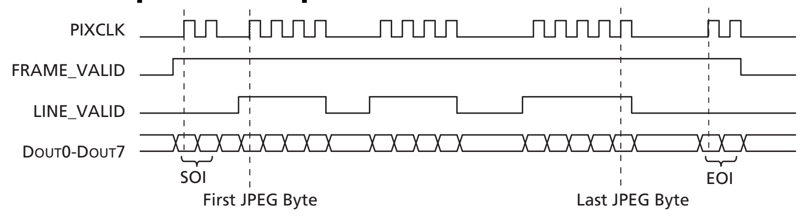
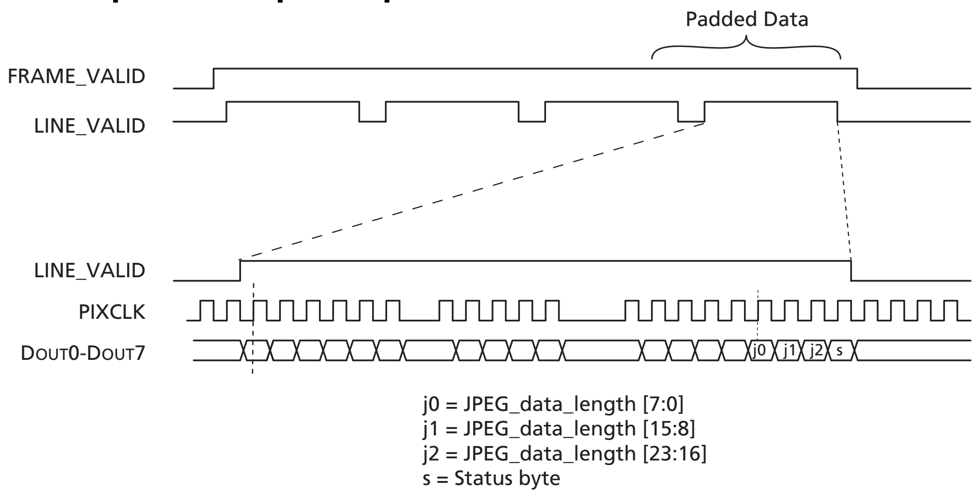


Figure 10: Timing of JPEG Compressed Output in Gated Clock Mode



Note: PIXCLK default inverted.

Figure 11: Timing of JPEG Compressed Output in Spoof Mode


Padded Data
FRAME_VALID
LINE_VALID
LINE_VALID
PIXCLK
DOUT0-DOUT7
j0
j1
j2
s
j0 = JPEG_data_length [7:0]
j1 = JPEG_data_length [15:8]
j2 = JPEG_data_length [23:16]
s = Status byte

Note: PIXCLK default inverted.

In the spoof mode, the timing of FRAME_VALID and LINE_VALID mimics an uncompressed output. The timing of PIXCLK and DOUT within each LINE_VALID assertion period is variable and therefore unlike that of uncompressed data. Valid JPEG data are padded with dummy data to the size of the original uncompressed frame.

Decimation, Zoom, and Pan

Decimation

Registers linking to decimation can be automatically adjusted by setting the related firmware variables. In preview mode, these variables need to be adjusted to a resolution of 800 x 600 or lower:

- *mode.Output Width A// ID = 7, Offset = 0x03*
- *mode.Output Height A// ID = 7, Offset = 0x05*

In capture mode, these variables need to be adjusted to a resolution of 1600 x 1200 or lower:

- *mode.Output Width B// ID = 7, Offset = 0x07*
- *mode.Output Height B// ID = 7, Offset = 0x09*

Keep in mind that at all times, the output sizes above need to be equal or smaller than the input size to the decimator (crop sizes).

If the modification in decimation occurs for the current (active) context, a REFRESH command (*seq.cmd* = 6) is needed to reflect the new values. Otherwise, a context switch to the targeted context automatically refreshes in these values.

Zoom

The zoom feature can be enabled only if the output size is smaller than the cropped size for each context mode. Once the output size is reduced, the crop size can be reduced also to provide a "zooming" effect. The variables involved are:

For context A (preview):

mode.Output Width A// ID = 7, Offset = 0x03

mode.Output Height A// ID = 7, Offset = 0x05

mode.crop_X0_A// ID = 7, Offset = 0x27

mode.crop_X1_A// ID = 7, Offset = 0x29

mode.crop_Y0_A// ID = 7, Offset = 0x2B

mode.crop_Y1_A// ID = 7, Offset = 0x2D

seq.cmd = 6// to refresh in the new settings

For context B (capture):

mode.Output Width B// ID = 7, Offset = 0x07

mode.Output Height B// ID = 7, Offset = 0x09

mode.crop_X0_B// ID = 7, Offset = 0x35

mode.crop_X1_B// ID = 7, Offset = 0x37

mode.crop_Y0_B// ID = 7, Offset = 0x39

mode.crop_Y1_B// ID = 7, Offset = 0x3B

seq.cmd = 6// to refresh in the new settings

Note: MT9D131 does not have a upscale feature.

Pan

The Pan feature can be used once the image is zoomed in as describe above. To move (or "Pan") the image view horizontally or vertically, an offset is applied to the corresponding crop variables:

In preview mode, apply the same offset to *mode.crop_X0_A* and *mode.crop_X1_A* to pan horizontally. Apply the same offset to *mode.crop_Y0_A* and *mode.crop_Y1_A* to pan vertically.

For capture mode, apply the same offset to *mode.crop_X0_B* and *mode.crop_X1_B* to pan horizontally. Apply the same offset to *mode.crop_Y0_B* and *mode.crop_Y1_B* to pan vertically.

Reducing FOV

For applications like electronic Pan-Tilt-Zoom (ePTZ), it may be beneficial to also reduce the field of view (FOV) as this will lend to an increase in frame rate where simple decimation does not affect the frame rate.

Reductions to the FOV, or original size, are made by adjusting the row height and column width variables. Reductions in FOV are only possible in capture mode (context B).

In capture mode, program the smaller row height to *mode.s_row_height_B* and program the smaller column width to *mode.s_col_width_B*.

If decimation or zoom is also to be used, it is mandatory to set these registers first. Otherwise, the image sensor will enter an error state.

To pan around the full 2Mp FOV using this smaller FOV, program the starting horizontal coordinate (relative to the original 2Mp FOV) to *mode.s_row_start_B* and program the starting vertical coordinate to *mode.s_col_start_B*.

To make FOV changes take effect, the sequencer must be toggled between preview and capture modes. *seq.cmd = 1* followed by *seq.cmd = 2*.

Sample ePTZ Code

First, set the output widow size. Here the output window size is being set to 800 x 600.

```
VAR = 7, 9, 0x258  // output height_B = 600
VAR = 7, 7, 0x320  // output width_B = 800
```

Second, set the crop window size. The X0 and Y0 values are used in the pan equation and the X1 and Y1 values are used in the zoom equation. Here, the crop window size is being set equal to the output window size of 800 x 600, which represents a no zoom (1X) and no pan condition.

```
VAR = 7, 53, 0x0    // crop_x0_B = 0
VAR = 7, 55, 0x320  // crop_x1_B = 800
VAR = 7, 57, 0x0    // crop_y0_B = 0
VAR = 7, 59, 0x258  // crop_y1_B = 600
```

Third, set the original (sensor core) window size which reduces the FOV. Here, an original window size of 800 x 600 is chosen. Because the number of rows that need to be read out is reduced, an increase in the maximum allowed frame rate is expected. For proper operation, the original window defined here must be larger than the crop window defined in the second step.

```
VAR = 7, 27, 0x01C  // s_row_start_B = 28
VAR = 7, 29, 0x03C  // s_col_start_B = 60
VAR = 7, 31, 0x258  // s_row_height_B = 600
VAR = 7, 33, 0x320  // s_col_width_B = 800
```

Enabling Special Effects

Special effect (monochrome, sepia, negative, solarization) formats can be selected by variables *mode.spec_effects_A* (ID = 7, Offset = 0x7F) in context A or *mode.spec_effects_B* (ID = 7, Offset = 0x81) in context B. Their settings are shown in Table 11.

Table 11: Enabling Special Effects

Value of Bits[2:0]	Effect
0	Disabled (no special effects)
1	Monochrome
2	Sepia
3	Negative
4	Solarization with unmodified UV
5	Solarization with UV

Note: *Seq.cmd = 5* is also necessary to refresh the new settings for special effects.

Mirroring the Image

An image can be mirrored horizontally or vertically or both in the current context (A or B). The registers associated with these functions are:

- R0x20:0[0]// set to mirror rows
- R0x20:0[1]// set to mirror columns

Column and Row Skip

Column and row skipping are available in both context A and B. When skipping is enabled, the image is subsampled, which results in a smaller dimension. This is one method used to reduce image size and have less constraint on the row time requirements. In all cases, the row and column sequencing ensures that the Bayer pattern is preserved. Skipping can be enabled and configured by:

Context A

R0x21:0[3:2]   // Row skip level (00 = 2x, 01 = 4x, 10 = 8x, 11 = 16x)

R0x21:0[4]   // Row skip enable

R0x21:0[6:5]   // Column Skip level (00 = 2x, 01 = 4x, 10 = 8x, 11 = 16x)

R0x21:0[7]   // Column skip enable

Context B

R0x20:0[3:2]   // Row skip level (00 = 2x, 01 = 4x, 10 = 8x, 11 = 16x)

R0x20:0[4]   // Row skip enable

R0x20:0[6:5]   // Column Skip level (00 = 2x, 01 = 4x, 10 = 8x, 11 = 16x)

R0x20:0[7]   // Column skip enable

Note: For skip level 4x or higher, the MCU must be disabled by setting *R0xC3:1[0] = 1*. In addition, the proper image output and crop sizes must be updated beforehand. As an example, the following values are set to enable 4x skipping in context A.

mode.Output Width A = 400   // ID = 7, Offset = 3

mode.Output Height A = 300   // ID = 7, Offset = 5 *mode.crop_X1_A = 400* *// ID = 7, Offset = 41*

mode.crop_Y1_A = 300 *// ID = 7, Offset = 45*

seq.cmd = 5 *// ID = 1, Offset = 3*

R0xC3:1[0] = 1 *// disable MCU*

R0x21:0 = 1204 *// 4x skipping, no binning*

Binning

The MT9D131 supports a 2 x 2 binning mode, which is used primarily instead of 2x skip to decimate a picture without losing information. The effect of aliasing in preview mode is eliminated when binning is used instead of just skipping rows and columns. To enable binning in preview mode, set R0x21:0[15] high; to enable it in capture mode, set R0x20:0[15] high.

For proper binning mode operation:

- Start address must be divisible by four (row and column)
- Window size must be divisible by four in both directions, after dividing by zoom factor and skip factor (because they both reduce the effective window size from the sensor's point of view)

Example

Default row size = 1200. 8x zoom means the actual window on the sensor is divided by 8, so 8x zoom and binning is not allowed with default window size, because 1200/8 = 150, which is not divisible by 4.

- Since binning can be seen as an extra level of skip, the combination binning/16x skip is not possible.

Configuring Pad Slew

During normal operation (no bypass), the slew rate for the DOUT0-DOUT7, PIXCLK, FRAME_VALID, and LINE_VALID are set by mode variables. In context A, they are defined by:

- *mode.fifo_conf1_A[7:5]* *// PCLK1 slew rate (ID = 7, Offset = 109)*
- *mode.fifo_conf1_A[15:13]* *// PCLK2 slew rate (ID = 7, Offset = 109)*
- *mode.fifo_conf2_A[7:5]* *// PCLK3 slew rate (ID = 7, Offset = 111)*
- *seq.cmd = 5* *// refresh for new settings to be effective*

For context B, they are set by:

- *mode.fifo_conf1_B[7:5]* *// PCLK1 slew rate (ID = 7, Offset = 116)*
- *mode.fifo_conf1_B[15:13]// PCLK2 slew rate (ID = 7, Offset = 116)*
- *mode.fifo_conf2_B[7:5]* *// PCLK3 slew rate (ID = 7, Offset = 118)*
- *seq.cmd = 5// refresh for new settings to be effective*

A value of 7 is designated as the fastest slew while a value 0 is defined as the slowest slew.

Note: The actual slew depends on load, temperature, and I/O voltage. Hence the proper slew rate should be tested and determined for each system. The default slew value will not work for all setups.

For bypass mode—such as sensor or SOC bypass, as set by R0x09:1[2:0]—the slew rate for DOUT0-DOUT7, PIXCLK, FRAME_VALID, and LINE_VALID is set by R0x0A:1[2:0].

Capturing Still Pictures

To capture still images:

1. First clear the capture video mode bit:
   seq.captureParams.mode[1] = 0
2. Next, specify the output image size for context B by using the variables *mode.Output Width_B* and *mode.Output Height_B.*
3. If the image is cropped from the original size, the following variables should also be set appropriately:

• *mode.crop_X0_B*	*// ID = 7, Offset = 0x35*
• *mode.crop_X1_B*	*// ID = 7, Offset = 0x37*
• *mode.crop_Y0_B*	*// ID = 7, Offset = 0x39*
• *mode.crop_Y1_B*	*// ID = 7, Offset = 0x3B*

For example, if the user wants to capture a full resolution image (no cropping), these values should be set:

• *mode.Output width_B = 1600*	*// ID = 7, Offset = 0x07*
• *mode.Output height_B = 1200*	*// ID = 7, Offset = 0x09*
• *mode.crop_X0_B = 0*	*// ID = 7, Offset = 0x35*
• *mode.crop_X1_B = 1600*	*// ID = 7, Offset = 0x37*
• *mode.crop_Y0_B = 0*	*// ID = 7, Offset = 0x39*
• *mode.crop_Y1_B = 1200*	*// ID = 7, Offset = 0x3B*

For an 800 x 600 capture with no cropping:

- *mode.Output width_B = 800*
- *mode.Output height_B = 600*
- *mode.crop_X0_B = 0*
- *mode.crop_X1_B = 1600*
- *mode.crop_Y0_B = 0*
- *mode.crop_Y1_B = 1200*

For a VGA capture with cropping of 800 x 600:

- *mode.Output Width B = 640*
- *mode.Output Height B = 480*
- *mode.crop_X0_B = 0*
- *mode.crop_X1_B = 800*
- *mode.crop_Y0_B = 0*
- *mode.crop_Y1_B = 600*

If the image size is less than or equal to 800 x 600, binning mode with 1ADC must be enabled by:

- *R0x20:0[10] = 1 and R0x20:0[15] = 1*

In addition, set the horizontal blanking for context B (r0x05:0) such that the integration time is the same as preview mode.

Next, set how many frames to be issued in capture mode before sequencer goes back to preview mode:

- *seq.captureParams.numFrames = 2   // 2 for this example (ID = 1, Offset = 0x21)*

Lastly, call the "CAPTURE" command:

- *seq.cmd = 2 // ID = 1, Offset = 0x03*

Capturing Videos

To capture videos:

1. First, set the capture video mode bit by:
   seq.captureParams.mode[1] = 1   // ID = 1, Offset = 0x20
2. Next, specify the output size with *mode.Output Width B* and *mode.Output Height B*. Similar to the "Capturing Still Pictures" on page 27, the appropriate variables (ID = 7, Offset = 0x35/37/39/3B) also need to be adjusted if cropping is used.
3. If the image size is less than or equal to 800 x 600 turn on power (1ADC) and binning mode for context_B:
   R0x20:0[10] = 1, R0x20:0[15] = 1
4. Set H_Blanking for context_B (R0x05:0) to keep the same integration time as in a preview mode.
5. Set mode.sensor_x_delay_B (ID = 7, Offset = 0x23) to adjust frame timing accurately.
6. Set V_Blanking for context_B (R6:0) to obtain 30 fps or another target frame rate.
7. Lastly, call the "CAPTURE" command:

- *seq.cmd = 2   // ID = 1, Offset = 0x03*

To stop video capture and return back to preview, call "PREVIEW" command:
seq.cmd = 1

Enabling and Capturing JPEG

To enable JPEG output, confirm the following value:

mode.mode_config[5] = 0   // ID = 7, offset = 0x0B

JPEG is supported in context B, not A. If the JPEG size needs to be changed, the context B image height and width need to be updated. For example, to change the JPEG image to 800x600 requires the following:

- *VAR = 7, 0x07, 0x0320   // MODE_OUTPUT_WIDTH_B*
- *VAR = 7, 0x09, 0x0258   // MODE_OUTPUT_HEIGHT_B*

After enabling JPEG output, a still JPEG image can be captured by following the procedure in the "Capturing Still Pictures" on page 27. For JPEG video, enable JPEG output, then follow "Capturing Videos" on page 28.

Switching Between JPEG 4:2:2, 4:2:0, and Monochrome

Before going into the capture mode for JPEG images, the desired resolution and format should be selected. To set the output size, these variables need to have the target values:

- *mode.Output Width B   // output width for capture (ID = 7, Offset = 0x07)*
- *mode.Output Height B   // output height for capture (ID = 7, Offset = 0x09)*

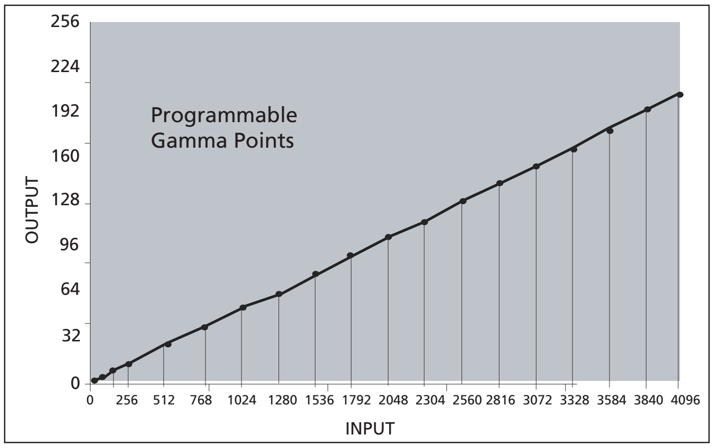
As for the JPEG format, the user can select between 4:2:2, 4:2:0 or monochrome by setting the variable:

- *jpeg.format   // 0 = 4:2:2, 1 = 4:2:0, 2 = monochrome (ID = 9, Offset = 0x06)*

Note: For the 4:2:0 format, the maximum width is 384 pixels (no special restriction on height). Next, switch to capture mode (see "Context Switch and Setup" on page 16) to output the new JPEG settings.

Gamma and Contrast

Figure 12: Gamma Correction Curve



Gamma

Because most video systems have been designed to compensate for the nonlinear light-intensity response of the CRT, the camera should output a gamma-corrected image. This means that an exponential gamma curve should be applied to the image before it leaves the camera system so that when it is viewed on a CRT of other modern imaging system, it does not appear too dark.

The MT9D131 image processing chain contains a gamma correction stage. All pixel values (in RGB color coordinates) are remapped to new values based on a piecewise linear extrapolation from a 19 point lookup table. The image pixel data coming in to the gamma correction stage is 12 bits (0–4095) and is remapped to 8-bit values (0–255). The 19 input points of the gamma table correspond to the following values: 0, 64, 128, 256, 512, 768, 1024, 1280, 1536, 1792, 2048, 2304, 2560, 2816, 3072, 3328, 3584, 3840, and 4095. (There are more points at the low pixel values where gamma curves usually change the most.)

Each input point is mapped to an output point (0–255). Pixel values that fall between two points are calculated based on a linear extrapolation between the nearest two points. Each of the three colors (R, G, and B) uses the same input gamma table. Either the mode driver can calculate the values that make up this lookup table, or the user can upload a custom table.

The mode driver remembers the identity of two separate gamma tables:

- one gamma table for preview (A) mode
- one gamma table for capture (B) mode

Upon each mode change, these tables are calculated and uploaded to the appropriate hardware registers at the end of the current frame. For changes to be immediately apparent in the image during development, the user must issue a refresh command (5) to the sequencer.cmd variable so that the new values are recalculated an uploaded without changing modes.

For each mode (A or B), the user can select from three predefined gamma tables, or upload and select a custom table. The predefined tables can be selected by changing the mode driver variables *gam_cont_A*, bits[2:0], or *gam_cont_B*, bits[2:0], depending on the intended mode (see mode driver variable description). Setting this value to "3" causes the mode driver to use the values stored in the mode driver variables *gamma_table_A/B_0* through *gamma_table_A/B_18* to construct the gamma table.

Contrast

Contrast enhancement can be mathematically applied to the gamma curve to give the image the appearance of wider dynamic range, but at the cost of slight errors in the color hues. The mode driver allows for predefined levels of contrast enhancement to be applied to each mode's gamma setting (or table) before updating the gamma-correction image processor. The contrast level may be set using the mode driver variables *gam_cont_A, bits[6:4]*, or *gam_cont_B, bits[6:4]*, depending on the intended mode (see mode driver variable description).

For changes to be immediately apparent in the image during development, the user must issue a refresh command (5) to the sequencer.cmd variable so that the new values are recalculated and uploaded without changing modes. Otherwise, the new settings take effect upon the next sequencer state change. The mode driver applies an s-curve function (see "S-Curve" section in documentation for more details) to the selected gamma table values and this result is uploaded to the gamma-correction image processor.

Lens Shading and Correction

Introduction

This section outlines the lens shading basics featured in the MT9D131 and shows how to adjust the lens shading settings automatically. Miniature camera modules have signal degradation on sensor periphery due to optical and geometrical factors. Lens shading correction compensates the signal degradation by digitally gaining pixels on the image periphery.

Lens Shading Approach

The digital gain to correct signal degradation can be expressed as the following:

$$Gain(x, y, color) = 1 - G + F_{\text{horizontal}}(x, color) + F_{\text{vertical}}(y, color) + k * F_{\text{horizontal}}(x, color) * F_{\text{vertical}}(y, color) \qquad \text{(EQ 2)}$$

Where:

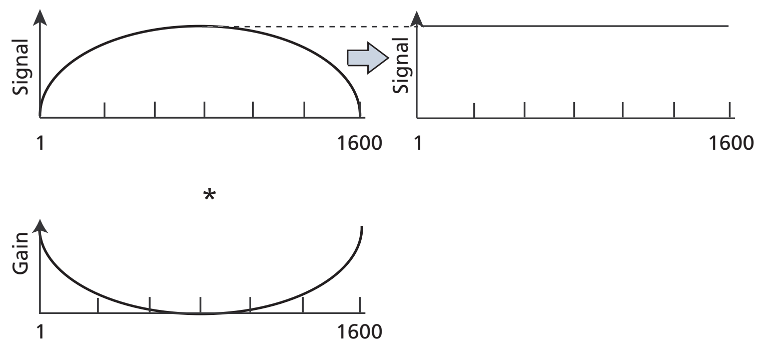
color = R, G, or B
K is the corner parameter, defined by register 0xAE.
G is a global constant, define by register 0xAD, which offsets the maximum gain of 1 (G is set to 0 for lens shading Auto-Adjust function).

The signal of each pixel is gained as follows:

$$Signalafter\_lc = Signalbefore\_lc * Gain(x,y,color) \qquad \text{(EQ 3)}$$

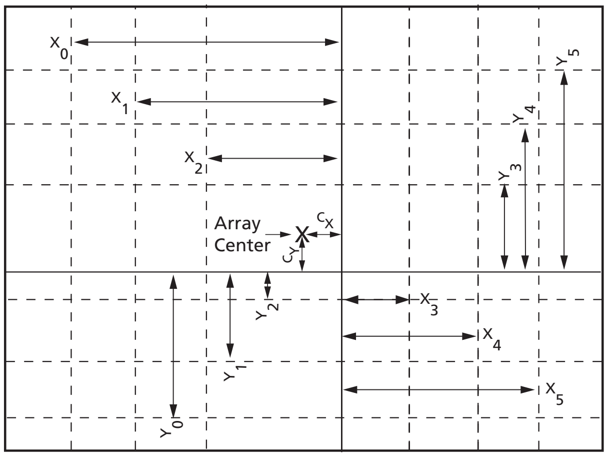
The relationship of the signal before and after is shown in Figure 13.

Figure 13: Signal Modification for Lens Shading



In the MT9D131, $F_{\text{horizontal}}$(x, color) and F_{vertical}(y, color) are piecewise quadratic (PWQ). The sensor is divided into eight zones in the x and y directions (see Figure 14). Therefore, each function for each color is represented by 10 numbers—8 numbers for the eight zone values in the corresponding direction and two initial conditions used to iteratively calculate function values across the entire pixel array.

Figure 14: Lens Correction Zones

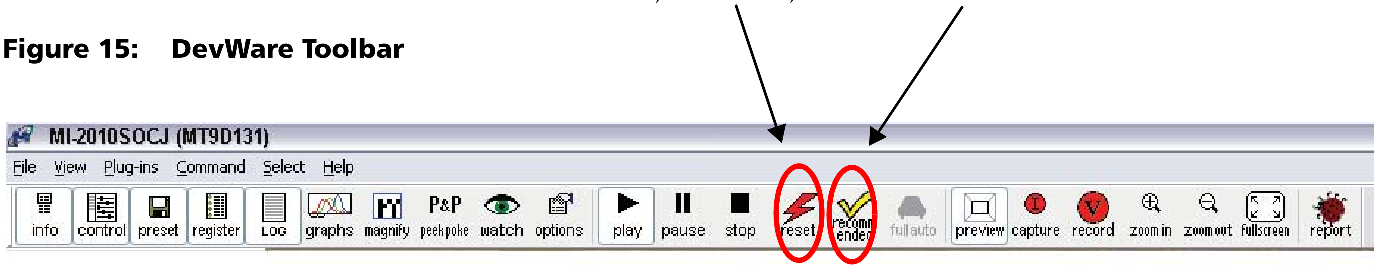


Note: The array center parameters, C_x and C_y, are defined by Register 0x87 (bit 0~7 for X coordinate, 8~15 for Y coordinate).

Setup

1. Start up DevWare version 2.6 Beta 5 (or above) and camera in the default state.
2. Point camera at a flat and uniformly illuminated calibration target.
   The variation in light intensity should be no more than 5 percent; target under Daylight (6500K) produces the best result. For best results, shield the light source and sensor demo system from external light sources.
3. When DevWare is loaded, click **reset**, then click **recommended**.

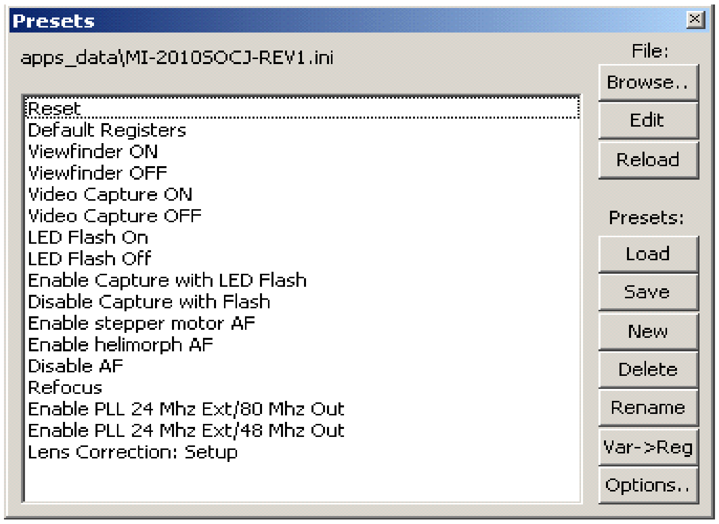
Figure 15: DevWare Toolbar



4. Reset—Loads all the default registers and variables.
5. Recommended—Loads all recommended registers and variables.
6. The automatic lens correction algorithm operates in either 800 x 600 viewfinder mode (preview) or 1600 x 1200 full resolution mode (capture). The user can simply leave at the default preset mode, or change to full auto mode and click on zoom out once to see the full image on the screen.

Preset and Load

Figure 16: Presets Dialog Box



Presets

apps_data\MI-2010SOCJ-REV1.ini

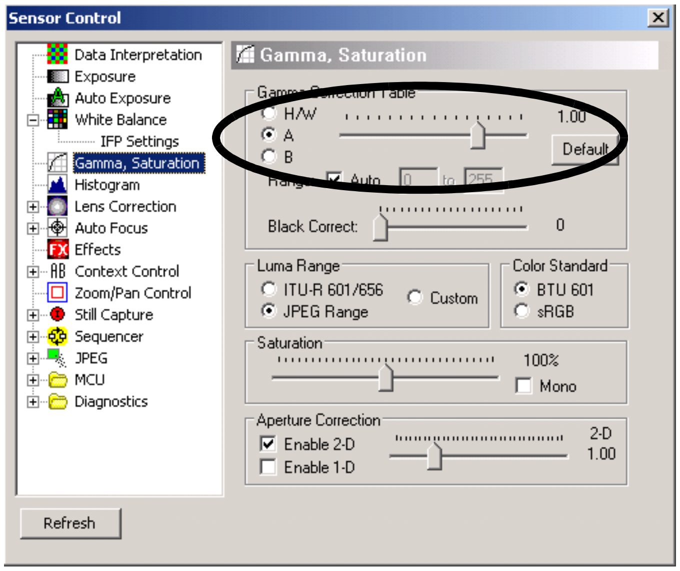
- Reset
- Default Registers
- Viewfinder ON
- Viewfinder OFF
- Video Capture ON
- Video Capture OFF
- LED Flash On
- LED Flash Off
- Enable Capture with LED Flash
- Disable Capture with Flash
- Enable stepper motor AF
- Enable helimorph AF
- Disable AF
- Refocus
- Enable PLL 24 Mhz Ext/80 Mhz Out
- Enable PLL 24 Mhz Ext/48 Mhz Out
- Lens Correction: Setup

Select **Lens Correction: Setup** in the **Presets** window and click **Load.** This loads all relevant register conditions that are needed to perform lens correction. The user can also choose not to load this setup setting and manually set all initial conditions listed in the following section.

Setup Conditions

1. Select Sensor Control.
2. Select Gamma, Saturation and set the gamma to 1.0.

Figure 17: Setting Gamma to 1.0



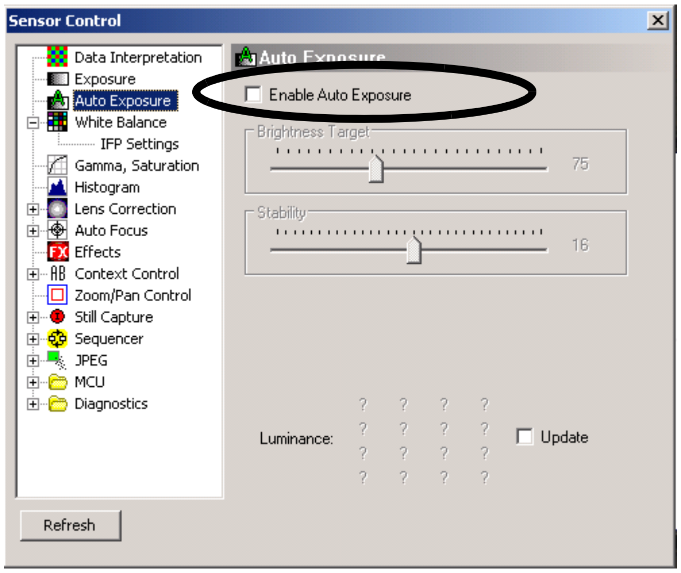
Sensor Control

Gamma, Saturation

Gamma Correction Table: H/W [✓ A] B — 1.00
Range: ☑ Auto 0 to 255
Black Correct: 0
Luma Range: ITU-R 601/656 [✓ JPEG Range] Custom
Color Standard: [✓ BTU 601] sRGB
Saturation: 100% ☐ Mono
Aperture Correction: ☑ Enable 2-D ☐ Enable 1-D — 2-D 1.00

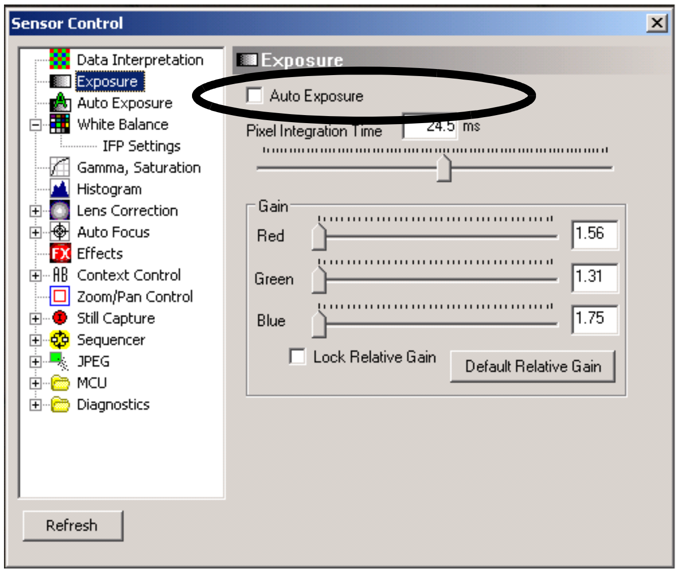
3. Go to **Sensor Control>Auto Exposure**.

Figure 18: Enable Auto Exposure



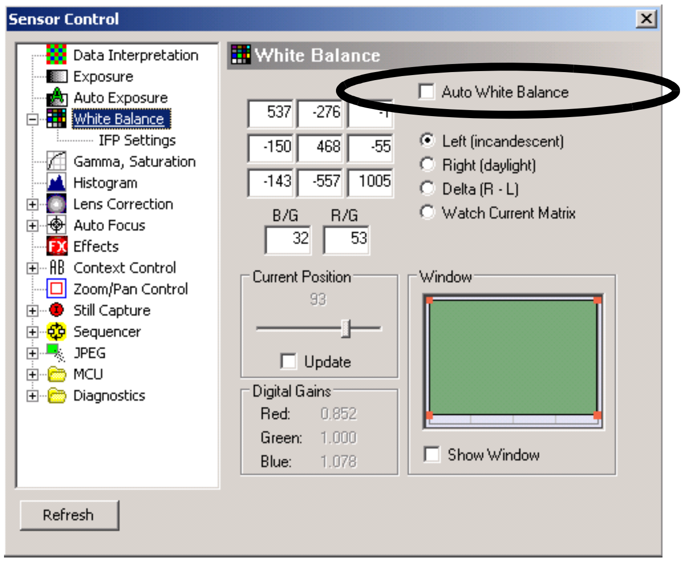
4. Open Graphs from the Devware toolbar to look at the intensity graph before calibration.
5. Adjust Brightness Target so peak brightness is about 75 percent of the maximum intensity scale on the Analysis Graph (this correspond to about 180), then turn off Auto Exposure.

Figure 19: Enable Auto Exposure



6. Go to **Sensor Control>White Balance** and turn off **Auto White Balance**.

Figure 20: Disable AWB



Auto White Balance ☐

537	-276	-1
-150	468	-55
-143	-557	1005

B/G 32 R/G 53

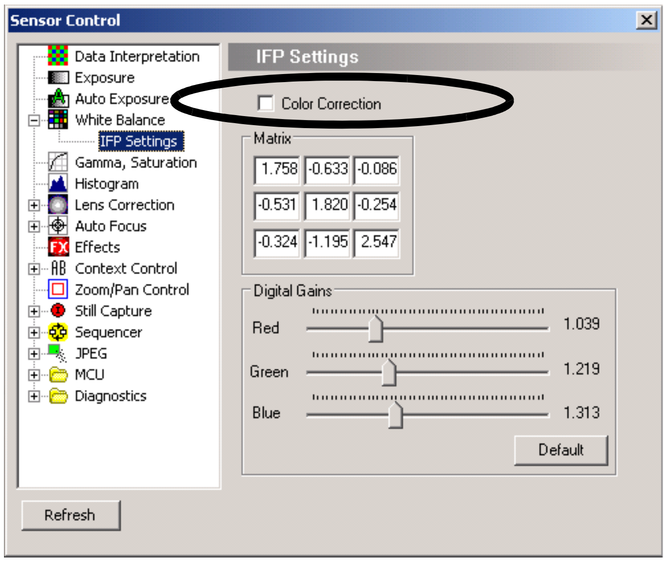
(●) Left (incandescent)
( ) Right (daylight)
( ) Delta (R - L)
( ) Watch Current Matrix

Current Position: 93
Update ☐

Digital Gains
Red: 0.852
Green: 1.000
Blue: 1.078

7. Go to Sensor **Control>White Balance>IF Settings** and turn off **Color Correction.**

Figure 21: Disable Color Correction



Color Correction ☐

Matrix

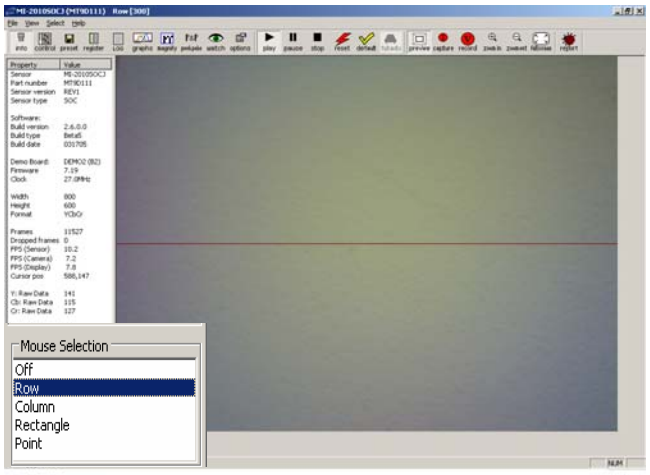
1.758	-0.633	-0.086
-0.531	1.820	-0.254
-0.324	-1.195	2.547

Digital Gains
Red 1.039
Green 1.219
Blue 1.313

Calibration

1. Set the row and column line at the middle row/column of the image. (In preview mode set row line at 300, set column line at 400. In full auto mode with 0.5x zoom, set row line at 600, column line at 800.)
2. Open Analysis Graph and view the Intensity plot of the middle row/column of the image. This will produce an analysis graph.
3. Go to **Sensor Control>Lens Correction** and follow the examples in Figure 23 on page 38.
4. Observe the progress on the Analysis Graph as in Figure 24 on page 38. When the graph has settled, unchecked Auto-Adjust.
5. Observe the progress on the Analysis Graph as in Figure 24. When the graph has settled, uncheck **Auto-Adjust**.
6. Slide the horizontal scroll bar all the way to the right so the "k factor" slider becomes visible. Adjust k factor for the best appearance of the corners.
7. Make other final adjustments as you judge necessary. The user has the option to choose not to completely calibrate the lens to obtain flat intensity curves by selecting different percentage for curvature. 100 percent corresponds to a completely flat curve. See Figure 26 on page 39 for correlation between percentage and curvature.
8. Click **Save As** to save the lens correction as an *.ini* file.
9. Load the new settings in DevWare and check the image quality.

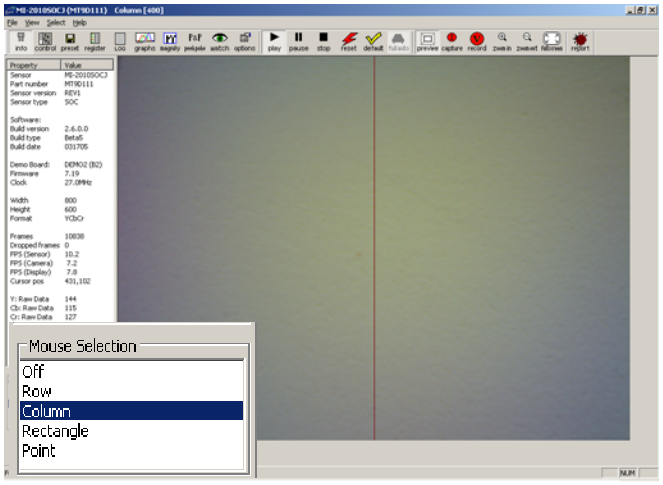
Figure 22: Setting the Row and Column Line



MI-2010SOCJ (MT9D111) Row [300]

Property	Value
Sensor	MI-2010SOCJ
Part number	MT9D111
Sensor version	REV1
Sensor type	SOC
Software:	
Build version	2.6.0.0
Build type	Beta5
Build date	031705
Demo Board:	DEMO2 (B2)
Firmware	7.19
Clock	27.0MHz
Width	800
Height	600
Format	YCbCr
Frames	11527
Dropped frames	0
FPS (Sensor)	10.2
FPS (Camera)	7.2
FPS (Display)	7.8
Cursor pos	586,147
Y: Raw Data	141
Cb: Raw Data	115
Cr: Raw Data	127

Mouse Selection: Off, [✓ Row], Column, Rectangle, Point



MI-2010SOCJ (MT9D111) Column [400]

Property	Value
Sensor	MI-2010SOCJ
Part number	MT9D111
Sensor version	REV1
Sensor type	SOC
Software:	
Build version	2.6.0.0
Build type	Beta5
Build date	031705
Demo Board:	DEMO2 (B2)
Firmware	7.19
Clock	27.0MHz
Width	800
Height	600
Format	YCbCr
Frames	10838
Dropped frames	0
FPS (Sensor)	10.2
FPS (Camera)	7.2
FPS (Display)	7.8
Cursor pos	431,102
Y: Raw Data	144
Cb: Raw Data	115
Cr: Raw Data	127

Mouse Selection: Off, Row, [✓ Column], Rectangle, Point

Figure 23: Sensor Control Dialog Box Showing Lens Correction Settings

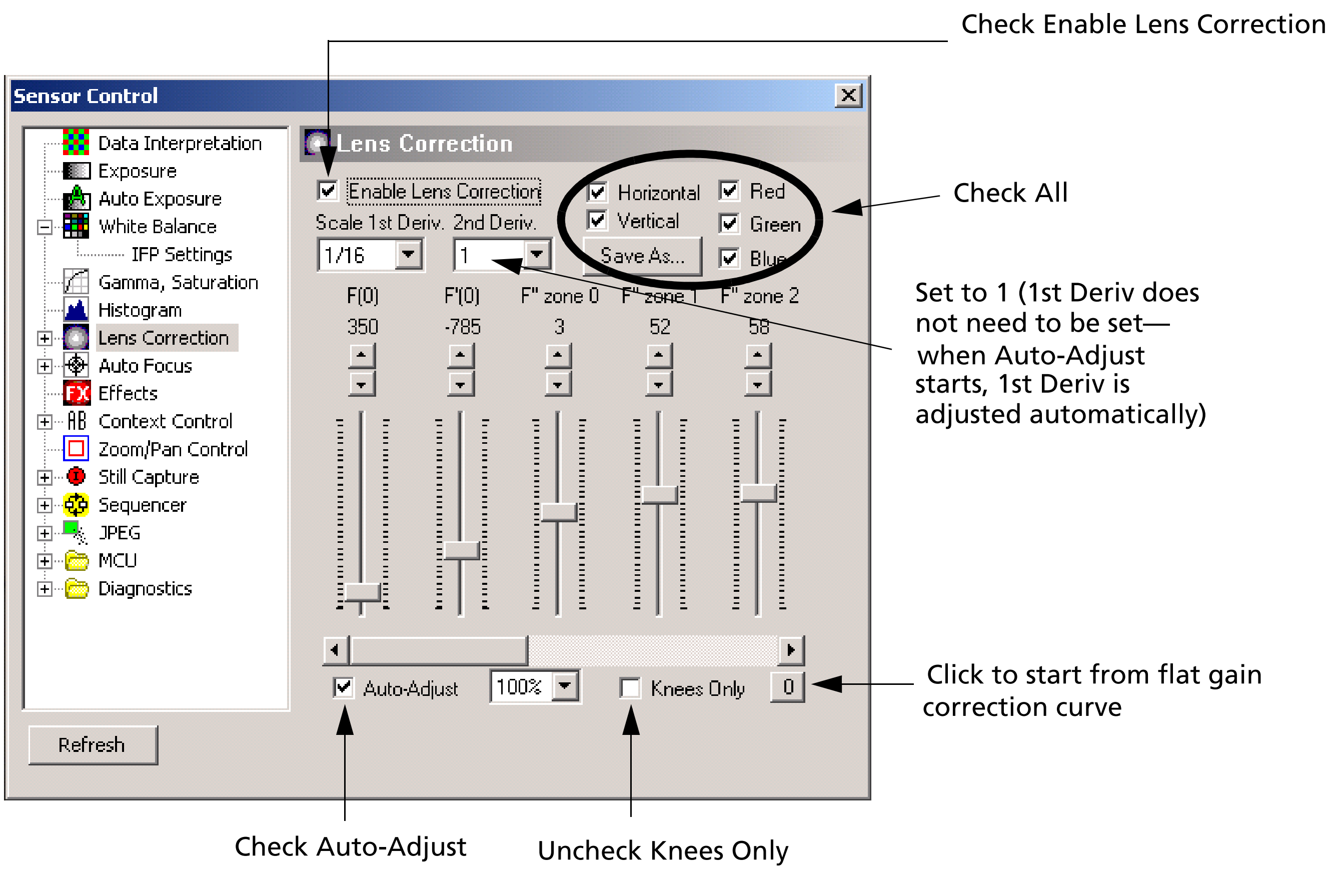


Figure 24: Settled Analysis Graph

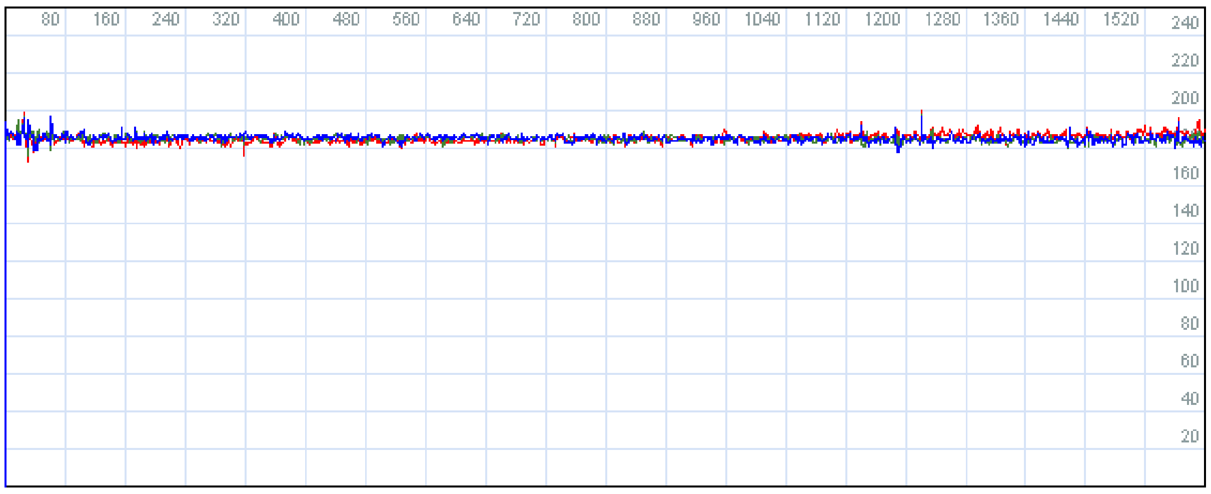


Figure 25: Adjusting K Factor

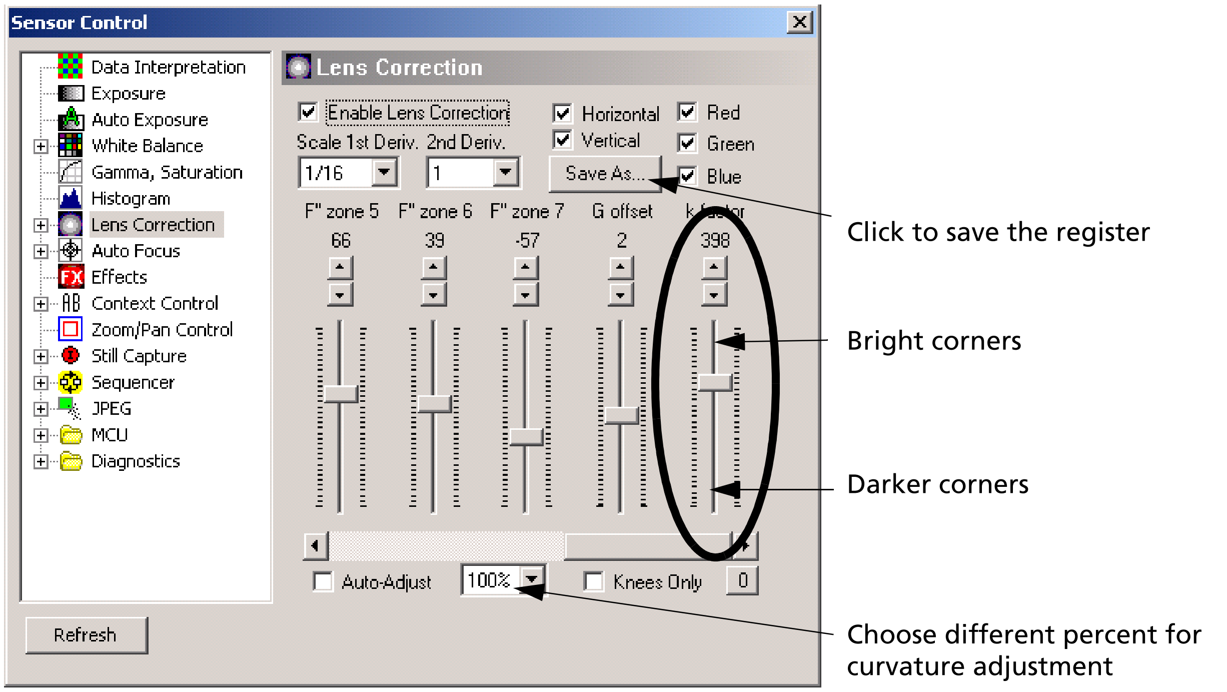
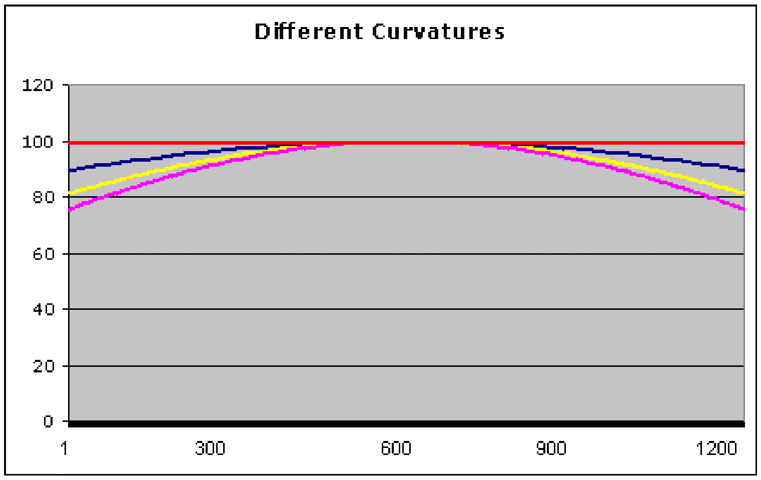


Figure 26: Correlation Between Percentage and Curvature



Result

Figure 27 shows a set of data and images BEFORE (on the left) and AFTER (on the right) the lens correction calibration procedure.

Figure 27: Lens Correction Result, Before and After

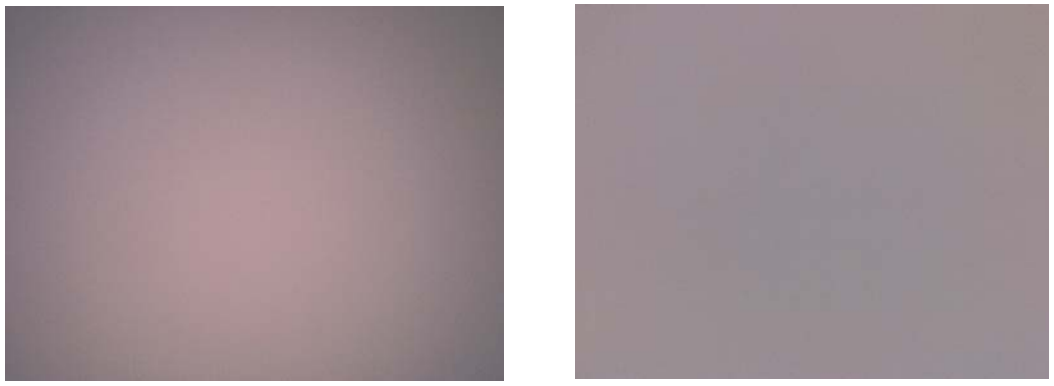


Figure 28: Intensity Graph (horizontal) Before and After

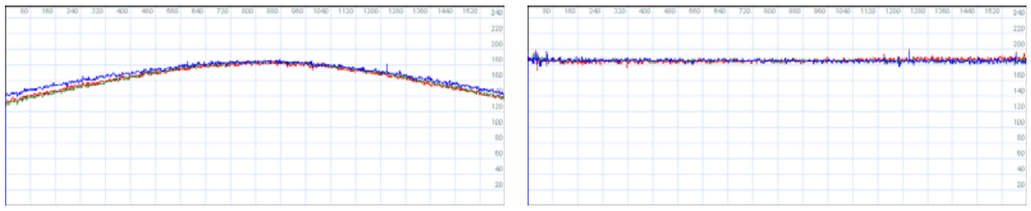
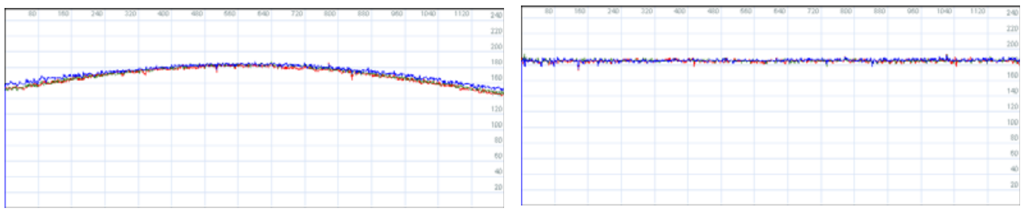


Figure 29: Intensity Graph (vertical) Before and After

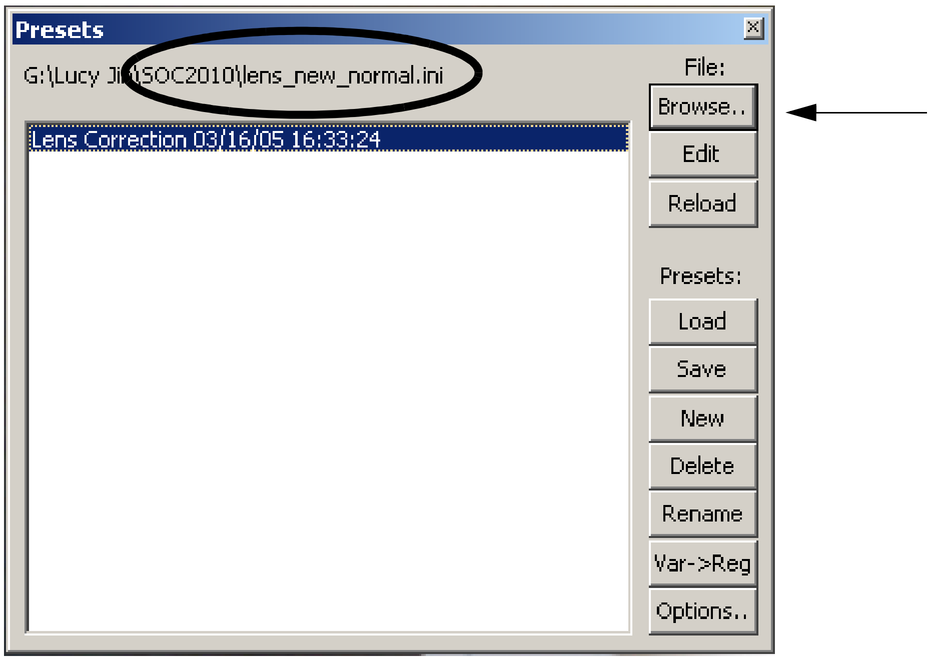


Verification

For a thorough verification of lens corrected image results, the following steps are recommended:

1. Place the MT9D131 camera module in front of flat and uniformly illuminated targets for inspection. The following scenarios are suggested: Daylight (6500K), Incandescent (2850K), Cool White Florescent (5300K).
2. Load DevWare and reset all registers (this enables AWB, Color correction, AE, Gamma Correction).
3. Open Preset window, click on **Browse** and locate the lens correction.ini file.
4. For each of the scenario, look at the picture for visual inspection and the intensity graph to see if a flat curve for each color (R,G,B) is obtained.
5. Point the camera module at a well-lit scenery and inspect the picture as a last check on image quality.
6. If needed, repeat the lens correction procedure to obtain improved register setting values and repeat the verification process to check image quality.

Figure 30: Locating the Lens Correction.ini File


G:\Lucy Ji\SOC2010\lens_new_normal.ini

Lens Correction 03/16/05 16:33:24

Related Register List

REG = 2, 0x80 //LENS_CORRECTION_CONTROL

REG = 2, 0x81 //ZONE_BOUNDS_X1_X2

REG = 2, 0x82 //ZONE_BOUNDS_X0_X3

REG = 2, 0x83 //ZONE_BOUNDS_X4_X5

REG = 2, 0x84 //ZONE_BOUNDS_Y1_Y2

REG = 2, 0x85 //ZONE_BOUNDS_Y0_Y3

REG = 2, 0x86 //ZONE_BOUNDS_Y4_Y5

REG = 2, 0x87 //CENTER_OFFSET

REG = 2, 0x88 //FX_RED

REG = 2, 0x89 //FX_GREEN

REG = 2, 0x8A //FX_BLUE

REG = 2, 0x8B //FY_RED

REG = 2, 0x8C //FY_GREEN

REG = 2, 0x8D //FY_BLUE

REG = 2, 0x8E //DF_DX_RED

REG = 2, 0x8F //DF_DX_GREEN

REG = 2, 0x90 //DF_DX_BLUE

REG = 2, 0x91 //DF_DY_RED

REG = 2, 0x92 //DF_DY_GREEN

REG = 2, 0x93 //DF_DY_BLUE

REG = 2, 0x94 //SECOND_DERIV_ZONE_0_RED

REG = 2, 0x95 //SECOND_DERIV_ZONE_0_GREEN

REG = 2, 0x96 //SECOND_DERIV_ZONE_0_BLUE

REG = 2, 0x97 //SECOND_DERIV_ZONE_1_RED

REG = 2, 0x98 //SECOND_DERIV_ZONE_1_GREEN

REG = 2, 0x99 //SECOND_DERIV_ZONE_1_BLUE

REG = 2, 0x9A //SECOND_DERIV_ZONE_2_RED

REG = 2, 0x9B //SECOND_DERIV_ZONE_2_GREEN

REG = 2, 0x9C //SECOND_DERIV_ZONE_2_BLUE

REG = 2, 0x9D //SECOND_DERIV_ZONE_3_RED

REG = 2, 0x9E //SECOND_DERIV_ZONE_3_GREEN

REG = 2, 0x9F //SECOND_DERIV_ZONE_3_BLUE

REG = 2, 0xA0 //SECOND_DERIV_ZONE_4_RED

REG = 2, 0xA1 //SECOND_DERIV_ZONE_4_GREEN

```
REG = 2, 0xA2 //SECOND_DERIV_ZONE_4_BLUE
REG = 2, 0xA3 //SECOND_DERIV_ZONE_5_RED
REG = 2, 0xA4 //SECOND_DERIV_ZONE_5_GREEN
REG = 2, 0xA5 //SECOND_DERIV_ZONE_5_BLUE
REG = 2, 0xA6 //SECOND_DERIV_ZONE_6_RED
REG = 2, 0xA7 //SECOND_DERIV_ZONE_6_GREEN
REG = 2, 0xA8 //SECOND_DERIV_ZONE_6_BLUE
REG = 2, 0xA9 //SECOND_DERIV_ZONE_7_RED
REG = 2, 0xAA //SECOND_DERIV_ZONE_7_GREEN
REG = 2, 0xAB //SECOND_DERIV_ZONE_7_BLUE
REG = 2, 0xAC //X2_FACTORS
REG = 2, 0xAD //GLOBAL_OFFSET_FXY_FUNCTION
REG = 2, 0xAE //K_FACTOR_IN_K_FX_FY
STATE = Lens Correction Center X, 800
STATE = Lens Correction Center Y, 600
BITFIELD = 1, 0x08 //LENS_CORRECTION
```

Auto Exposure

Overview

The auto exposure (AE) algorithm performs automatic adjustments of the image brightness by controlling the exposure time and analog gains of the sensor core as well as the digital gains applied to the image.

Two auto exposure algorithm modes are available:
- preview
- scene evaluative

Auto exposure is implemented by means of a firmware driver that analyzes image statistics collected by exposure measurement engine, decides the best exposure and gain settings, and programs the sensor core and color pipeline accordingly. The measurement engine subdivides the image into 16 windows organized as 16 programmable equal-size rectangular windows forming a 4 x 4 grid.

Preview Mode

In preview mode, 16 windows are combined in two segments: central and peripheral. The central segment includes the four central windows. All remaining windows belong to the peripheral segment.

Scene brightness is calculated as average luma in each segment taken with certain weights. The variable *ae.weights*[3:0] specifies the central zone weight, and *ae.weights*[7:4] specifies the peripheral zone weight.

The preview exposure mode is activated during preview or video capture. It relies on the AE measurement engine that tracks the speed and amplitude of the change of the overall luminance in the selected windows of the image.

The backlight compensation is achieved by weighting the luminance in the center of the image higher than the luminance on the periphery.

Other algorithm features include the rejection of fast fluctuations in illumination (time-averaging), control of the response speed, and control of the sensitivity to the small changes. While the default settings are adequate in most situations, the user can program the target brightness, the measurement window, and other parameters described above.

Scene-Evaluative Mode

A scene-evaluative AE algorithm is available for use in “snapshot” mode. The algorithm performs scene analysis and classification with respect to its brightness, contrast and composure, and then decides to increase, decrease or keep original exposure target. It makes the most difference for backlight and bright outdoor conditions.

AE Sport Mode

Digital sensor “sport mode” is essentially reducing motion-blur by limiting the exposure time to a small amount while allowing increased gain to compensate. The MT9D131 AE algorithm allows for advanced control over its parameters.

Based on the “Gain vs. Exposure” in the specification sheet, the host can upload register values to limit the exposure time. These limits may be uploaded to *ae.IndexTH23* or *ae.maxIndex*. Using *ae.IndexTH23* allows for slower shutter speeds if the exposure/gain limits are insufficient to capture the scene. Using *ae.maxIndex* to limit the exposure time sets the absolute AE limit, and thus low-light scenes will appear dark.

The value of either the *ae.maxIndex* or *ae.IndexTH23* is the integer multiple of the 50 or 60Hz period respectively, (depending on current flicker frequency) in the units of # of sensor lines. For example, if the required sport shutter speed is less than 1/100 seconds then, taking the longest of the two flicker periods 50Hz (rectified = 100Hz), the maximum AE index should be set to 1. Issue a REFRESH Sequencer command after uploading new driver parameters for them to take effect.

Calibrating the AE Exposure Value (EV) Reference

The AE driver translates the luminance data in to EV for some decisions (for example, *seq.sharedParams.FlashTH*). This translation must be calibrated to ensure accurate results. This calibration should occur after the lens module has been calibrated for roll-off.

To calibrate:

1. Place an 18 percent reflectance grey card in full sunlight.
2. Point the camera at the card so that the entire frame is filled with the grey card image.
3. Allow the AE algorithm to settle on a new exposure time.
4. Change the value of *ae.mmShiftEV* until the *ae.mmMeanEV* variable shows a value of 15 (EV).

Modifying the Image Brightness

The appearance of the final image can be adjusted by a variety of controls each with a different specific effect on brightness.

When using the auto exposure driver, the target luminance of the MT9D131 can easily be changed by modifying the variable *ae.Target*. This causes the AE to select a different exposure time/gain combination to capture a brighter or dimmer image histogram.

To attain a relative exposure value (EV) number from the *ae.Target* value, use the relation: *relative EV = LOG(ae.Target/reference ae.Target)/LOG(2)* where each log is base-10. Reference *ae.Target* is the normal (EV = 0) operating target, and *ae.Target* is the new position. For example, if the normal *ae.Target* is 75, then to get an EV value of +1, the *ae.Target* must be set to 150.

Modifying the gamma curve characteristics is another method to affect the final image's brightness. These controls are found in the mode driver's *gam_cont_A/B* variables (see the MT9D131 data sheet or "Gamma and Contrast" on page 29). A preloaded gamma table can be selected or a user-defined gamma table may be uploaded. Using the gamma table to affect brightness allows brightening some shades in the image while leaving others as is. This maintains the image's dynamic range while boosting the overall brightness.

The mode-driver variables *y_rgb_offset_A/B* (these overwrite hardware register 0xBF:1) allow for an offset to be applied to the image data. Changing these values alters the apparent brightness whether the system is outputting RGB data or YUV data. This adjustment only shifts the image histogram data and is not a good option for increasing the image quality.

Speeding Up and Slowing Down AE Adjustments

Three variables in the AE driver influence the AE speed:

- *ae.SkipFrames*
  Specifies how many frames AE driver has to skip between AE register calculations. Larger numbers slow the AE.
- *ae.JumpDivisor*
  Specifies how much luminance that the AE jumps in one calculation.
  1: full distance
  2: half the distance
  3: 1/3 of the distance, and so on.
  The smaller the number, the faster the AE (1 is minimal number). The bigger the number, the smoother the AE. (The expected brightness after one AE step is Y*new* = *ae.CurrentY + (ae.Target – ae.CurrentY) / ae.JumpDivisor*)
- *ae.lumaBufferSpeed*
  Speed of luma buffering (32 = fastest, 1 = slowest). To avoid unwanted reactions of the AE on small fluctuations of scene brightness or of momentary scene changes, the AE driver uses a temporal filter for the luminance.

Setting the values *ae.lumaBufferSpeed = 32*, *ae.SkipFrames = 0*, and *ae.JumpDivisor = 1* specifies the maximum AE speed, but the image may appear to oscillate in certain dynamic lighting conditions.

The Sequencer controls AE speed in every state by the variables:

- *seq.sharedParams.aeContBuff*
- *seq.sharedParams.aeContStep*
- *seq.haredParams.aeFastBuff*

- *seq.sharedParams.aeFastStep*

For example, to achieve the fastest AE during the "Leave Preview" state, set:

- *seq.sharedParams.aeFastBuff* = *32*
- *seq.sharedParams.aeFastStep* = *1*

and select the fast AE mode for the "Leave Preview" state:

seq.previewParLeave.ae = *1*

The AE requires more time to achieve a luminance target if the scene suddenly becomes bright as compared to if the scene suddenly becomes dark. To quicken the light to dark AE adjustment time, set the variable *ae.status[7]* to 1.

Maintaining Specific Frame Rates

Specific frame rates are attained by establishing a set of image timing variables (that is, hblank, vblank, PLL timing, extra delay, and so on). However, the integration time (also known as the shutter width) may be longer than a frame time for low light scenes. This causes the frame rate to be dependent on the lighting conditions. To avoid drop in the frame rate, the AE driver provides several optional controls.

To reject flicker, integration time is adjusted in increments of *ae.R0x09_step*. *ae.R0x09_step* specifies the duration in row-times equal to one flicker period. The AE driver sets integration time as an integer multiple of *ae.R0x09_step* (integration time, in *rows* = *ae.R0x09_step* * *ae.Index*) and these multiples are labeled "index." Adjusting the parameters *ae.IndexTH23* and *ae.maxIndex* controls the relationships between integration time and gain used to achieve the luminance targets. Refer to the specification sheet for a more detailed diagram of the relationships between shutter width and gain. (If the AE index control variables are changed, the sequencer command "Do Refresh" (*seq.cmd* = *5*) must be called to activate the new settings.)

Using Manual Exposure and Manual Gain

If the AE driver is active (*seq.mode[0]* = *1*), then the user cannot change some of the registers related to auto exposure (for example, R0x09:0–shutter width, R0x0C:0–shutter delay, R0x2B–R0x2F:0–analog gains, R0x4E:1, R0x0A6–0x6E:1–digital gains, and so on.) because the AE driver continuously overwrites them. The user can manually set the AE parameters in two configurations:

1. Disable the microcontroller (see "Using the Test Patterns" on page 73). Once disabled, the sensor and image processing registers may be set without being overwritten by the firmware.
2. Disable the AE driver (set to manual mode in the sequencer states), and change the AE driver's variables to affect the exposure time and the gain (see *ae.R0x09*, *ae.VirtGain*, *ae.DGainAE1*, *ae.DGainAE2*).

Flicker Avoidance

Background

Most low-power fluorescent lights flicker quickly on and off to reduce power consumption. While this fast flickering is marginally detectable by the human eye, it is very noticeable in digital images. This is because the flicker period of the light source is very close to the range of digital images' exposure times.

Many CMOS sensors use a "rolling shutter" readout mechanism that greatly improves sensor data readout times. This allows pixel data to be read out much sooner than other methods that wait until the entire exposure is complete before reading out the first pixel data. The rolling shutter mechanism exposes a range of pixel rows at a time. This range of exposed pixels starts at the top of the image and then "rolls" down to the bottom during the exposure period of the frame. As each pixel row completes its exposure, it is ready to be read out. If the light source oscillates (flickers) during this rolling shutter exposure period, the image appears to have alternating light and dark horizontal bands.

If the sensor uses the traditional snapshot readout mechanism in which all pixels are exposed at the same time and then the pixel data is read out, then the image may appear overexposed or underexposed due to light fluctuations from the flickering light source.

The rate of light flicker is most often either 100Hz or 120Hz; the value is determined by the power specifications adopted by different nations around the world.

To avoid this flicker effect, the exposure times must be multiples of the light source flicker periods. For example, in a scene lit by 120Hz lighting, the available exposure times are 8.3ms, 16.67ms, 25ms, 33.33ms, and so on. (The need for an exposure time less than 8.3ms under artificial light is extremely rare.)

The camera designer must first detect whether there is a flickering light source in the scene, and if so, determine its flickering frequency. In this case, the auto exposure must limit the integration time to an integer multiple of the light's flicker period.

By default, the MT9D131 does all of this automatically, ensuring that all exposure times avoid any noticeable light flicker in the scene. The MT9D131 auto exposure algorithm is always setting exposure times to be integer multipliers of either 100Hz or 120Hz. The flicker detection microcontroller driver keeps monitoring the incoming frames to detect whether the scene's lighting has changed to the other of the two light sources.

Using the Flicker Detection Driver

The flicker detection hardware registers are located between 0x7A:1 and 0x7D:1, but these are merely statistical controls used by the flicker detection driver (ID = 4) in the microcontroller code and their values should not be changed.

The intelligence and adjustability of the flicker detection algorithm resides in the micro-controller code (ID = 4). The flicker detection driver's actions may be controlled by the sequencer driver (ID = 1). For each of the sequencer's programmable states (preview enter, preview, preview leave, capture; all variables' names end in ".fd"), the flicker detection driver can be set to off (0), continuous (1), or manual (2).

Upon setting these values, the sequencer must be commanded to refresh the state parameters by issuing a "refresh mode" command (*sequencer.cmd = 6*) for the settings to be reflected in the image stream. Changing the value of the flicker detection's mode variable is not recommended because the sequencer driver, upon the next state change, will likely overwrite this value.

Off (0) mode completely disables the flicker detection and the auto exposure driver uses the current value stored in the R0x09_step as the exposure step size.

Continuous (1) mode constantly analyzes the image statistics to detect whether the lighting source has changed from the current frequency to the other frequency. If it detects the change, it uploads the new flicker period step size to the auto exposure driver (R0x09_step) so that all auto exposure times are integer multipliers of this period.

Manual (2) mode disables the automatic analysis of the light source, but allows the user to alter the flicker detection driver variable "mode" bit 6 to select the light source manually. Once changed, the new value, at either R0x09_step50 or R0x09step60, is uploaded to the auto exposure driver's R0x09_step variable.

To set up the flicker detection driver:

1. Know the time required for each sensor line to read out.
2. Calculate how many lines are in a 100Hz flicker period and a 120Hz flicker period. For example, if the line time is 75µsec, then the 120Hz (for 60Hz lighting sources) flicker period is 111 lines (1/(120 * 75µsec)).
3. Upload the resulting values for 50Hz and 60Hz flicker period lines to the flicker detection driver's R0x09_step50 and R0x09_step60 variables, respectively.
4. Calculate the search window for the flicker detection window for each lighting source. This is the flicker period search range that the flicker detection driver uses (in units of lines/5) to identify whether flickering is occurring in the scene. For most applications, the values are simply the flicker period (calculated above) divided by 5, plus or minus 1.
   For example, if the flicker period lines were calculated to be 111, then the flicker search period is from 21 ("f1") to 23 ("f2") using the formula: lines/5 +/- 1.
5. Upload these search period values to the variables *search_f1_50*, *search_f2_50*, *search_f1_60*, and *search_f2_60*.

These driver variable settings should be sufficient for reliable flicker detection. However, if there is a further need to fine-tune the algorithm, some additional (advanced) variables are available for adjustment.

The variables *stat_min* and *stat_max* help control the sensitivity of the algorithm to flickering light in a scene. If the algorithm detects *stat_min* instances of flickering in *stat_max* measurements, then the flicker detection driver responds by changing the flickering frequency (if different than the existing detected frequency). Altering the ratio between the two values affects the sensitivity of the algorithm and the time required to decide whether to change the detected light flicker period.

Increasing *ae.SkipFrame* increases the number of frames that are skipped between each measurement. However, because the algorithm depends on detecting differences in light intensity given a constant scene, setting this value too high may allow the camera user to change the scene framing between flicker detection frames, making this algorithm less effective overall.

Color Correction

Auto White Balance

To achieve good color rendition and color saturation, interpolated colors of all pixels are subjected to color correction. The color correction is a linear transformation of the image with a 3 x 3 color correction matrix. The optimal values of the correction matrix elements depend on the spectrum of light incident on the sensor. They can be either programmed by the user or automatically selected by the auto white balance (AWB) algorithm.

The AWB algorithm is designed to compensate for the effects of the changing spectra of the scene illumination on the quality of the color rendition. This sophisticated algorithm consists of two major parts:

- a measurement engine performing statistical analysis of the image (R0x30–R0x32:1)
- a firmware driver performing the selection of the optimal color correction matrix, digital, and analog gains

The color correction matrix consists of nine values, each of which represents a digital gain factor on the corresponding color channel with the diagonal elements representing the gain factors on each color channel and the off diagonal terms representing the gain factors to compensate for color crosstalk. The matrix is normalized so that the sum of each row is 1.

Table 12: Color Corection Matrix Structure

Color	Gain R	Gain G	Gain B
Saturation Red	CCM[0]	CCM[1]	CCM[2]
Saturation Green	CCM[3]	CCM[4]	CCM[5]
Saturation Blue	CCM[6]	CCM[7]	CCM[8]

Variable bits *awb.ccmL[0]* through *awb.ccmL[8]* keep the matrix values for the left (incandescent) matrix. Variable bits *awb.ccmL[9]* and *awb.ccmL[10]* contain the gain ratios between Red/Green and Blue/Green used for digital gain for this matrix. Variables *awb.ccmRL[0]* through *awb.ccmRL[10]* keep the difference between the right (daylight) and the left (incandescent) matrices. The relationship between the current CCM and the calibrated left and right CCM and current *awb.CCMposition* is shown in Equation 4:

$$awb.ccm[i] = awb.ccm[i] + awb.ccmRL[i]\ x \left(\frac{awb.CCMposition}{127}\right) \qquad \text{(EQ 4)}$$

where:

awb.CCMPosition ranges from 0 to 127. Normally, *awb.CCMPosition* = 0 corresponds to incandescent light at 2850K and *awb.CCMPosition* = 127 corresponds to daylight at 6500K if the matrices where calibrated at the color temperatures.

$$Total\ WB\ gain = \frac{Red\ Gain(R0x2D:0)}{Global\ Gain(R0x2F:0)} \times awb.GainR \qquad \text{(EQ 5)}$$

Changing the Color Saturation

The variable *awb.saturation* defines the color saturation:

- *awb.saturation = 128* corresponds to 100 percent of saturation when *awb.ccm* is calculated in Equation 4 on page 50.
- *awb.saturation = 0* corresponds to the unity matrix.
- Any other value of *awb.saturation* corresponds to a matrix which is a linear interpolation between 100 percent saturation and the unit CCM.

To set a 30-percent saturated matrix:

- Enable AWB *seq.mode[2] = 1*
- Set awb.*saturation = 30 * (128 / 100) = 38*

Speeding Up or Slowing Down AWB

Two variables are responsible for the AWB speed:

- *awb.GainBufferSpeed*
  Speed of the AWB digital gains buffering (32 = fastest, 1 = slowest)
- *awb.JumpDivisor*
  Specifies how much the AWB parameters (gains, CCM, and so on) are traversed in one jump (1 = all the way, 2 = halfway, 3 = 1/3 of the way, and so on)

The smaller the number, the faster AWB (1 is the minimal number). The sequencer controls the AWB speed in every state by these variables:

- *seq.sharedParams.awbContBuff*
- *seq.sharedParams.awbContStep*
- *seq.sharedParams.awbFastBuff*
- *seq.sharedParams.awbFastStep*

For example, to make the AWB as fast as possible on "Leave Preview," set the fastest parameters for fast AWB mode:

- *seq.sharedParams.awbFastBuff = 32*
- *seq.sharedParams.awbFastStep = 1*

Then select the fast AWB mode for the "Leave Preview" state.
seq.previewParLeave.awb = 1

Using a Static CCM

There are two ways to use a static CCM:

1. Use default matrices:
   a. Turn AWB on.
      seq.mode[2] = 1
   b. Set digital WB gain to 1.
      awb.mode[5] = 1
   c. Set WB position to select the desired CCM.
      awb.CCMposition = 0 corresponds to incandescent CCM
      awb.CCMposition = 127 corresponds to daylight CCM
   d. CCMPosition is decided from B/G
      If *awb.GainB* is outside the range of *SteadyBGainOutMin* and *SteadyBGainOutMax*, the *CCMPosition* would be changed, and analog gain is adjusted. *CCMPosition* keeps adjusting until *awb.GainB* is in range of *SteadyBGainInMin* and *SteadyBGainInMax*. If CCM Position reaches one end, but AWB is not satisfied,

digital gains would be increased or decreased until it reaches *GainMax/GainMin*.

2. Download user-defined CCM:
   a. Turn AWB on.
      seq.mode[2] = 1
   b. Set digital WB gain to 1
      awb.mode[5] = 1
   c. Write coefficients of new CCM into variables *awb.ccmL[0]* through *awb.ccmL[10]* (256 corresponds to 1.0)
      *awb.ccmL[0] = CCM11*256*
      *awb.ccmL[1] = CCM12*256*
      . . .
      *awb.ccmL[8] = CCM33*256*
      *awb.ccmL[9] = RedAnalogGain/GreenAnalogGain *32*
      *awb.ccmL[10] = BlueAnalogGain/GreenAnalogGain *32*
   d. Set variables *awb.ccmRL[0]* through *awb.ccmRL[10]* to 0
      awb.ccmRL[0] = 0
      . . .
      awb.ccmRL[10] = 0

Performing Color Calibration

The user should calibrate the lens before calibrating color. For the procedure on lens calibration, see "Lens Shading and Correction" on page 31.

Setup

To set up hardware:

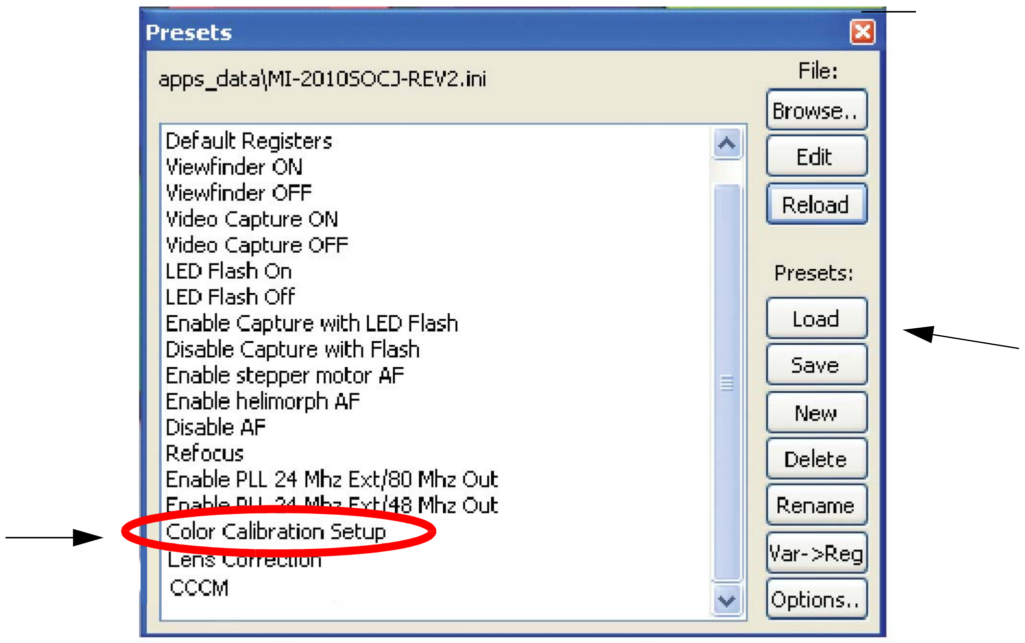
1. Make sure the lens and sensor module are completely shielded from external light entering the module from the side. If needed, shield the lens module from external light using black tape.
2. Place the color rendition chart in the middle of the image screen to avoid corner color shading effect to the chart. Make sure the entire chart is in the image screen.
3. Turn on one of the light sources (Blue Daylight at 6500 Kelvin or red incandescent light at 2850 Kelvin). For the procedure described below, we show the calibration for blue daylight first, then ask the user to repeat the process for red light.

To set up software:

1. Start up DevWare version 2.6 Beta 5 (or above) and the camera module.

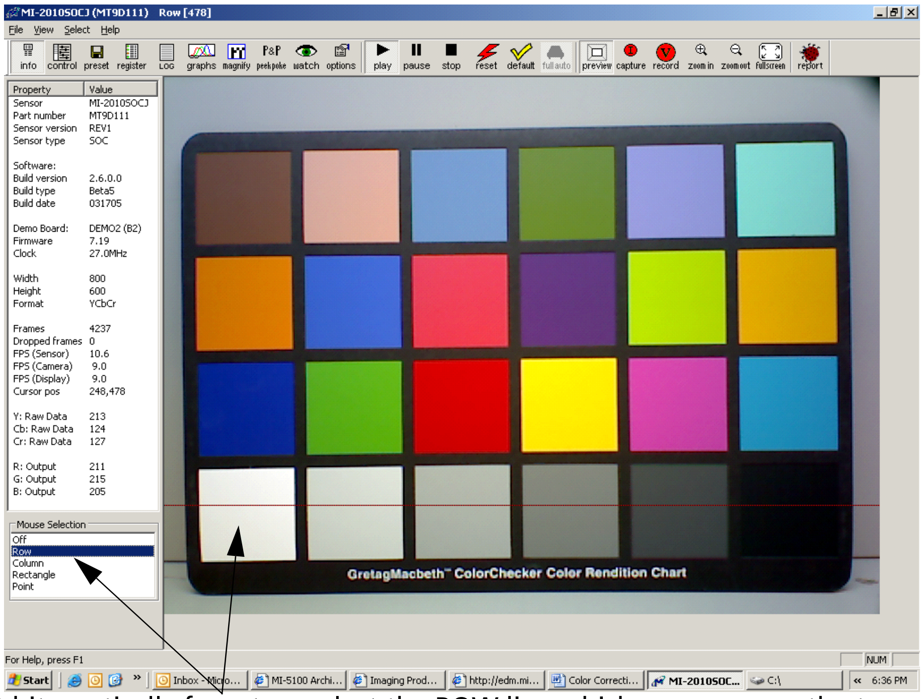
Note: Do not load Register Default settings upon starting the software.

2. Preset and Load
   a. Click the preset button to open the **Presets** window.
   b. In the **Presets** window, select **Color Calibration Setup** and click on **Load**—this loads all relevant register setting that are needed to perform the tuning in DevWare.



   c. Select **Lens Correction** to load the settings for lens calibration.

3. Calibration in DevWare

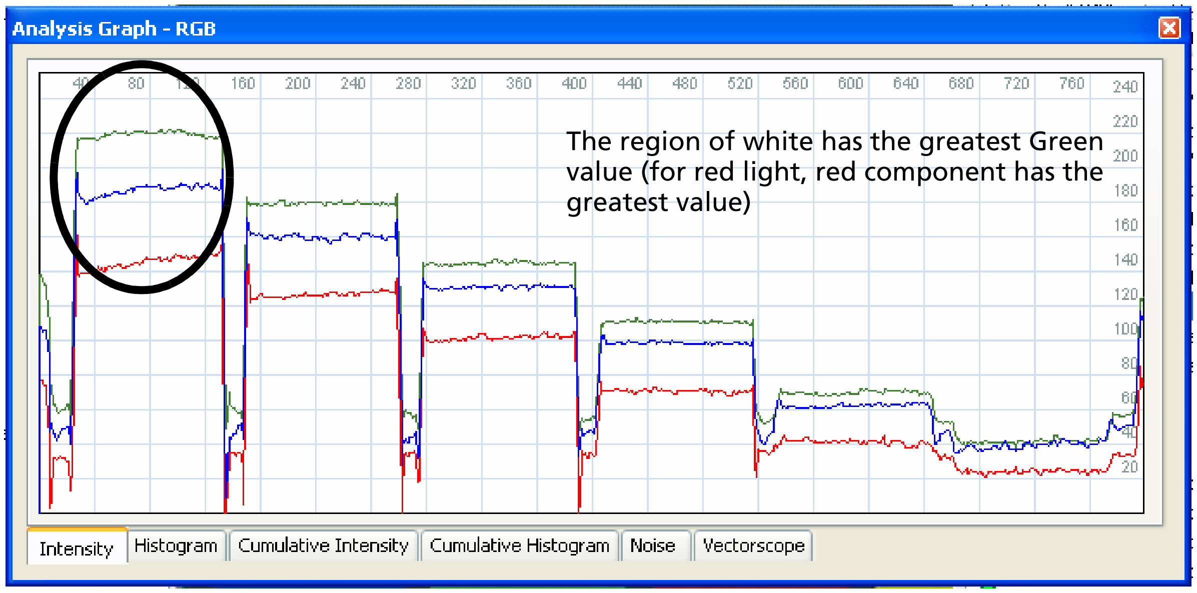

MI-2010SOCJ (MT9D111) Row [478]
File View Select Help
info control preset register Log graphs magnify peek/poke watch options play pause stop reset default full auto preview capture record zoom in zoom out fullscreen report
GretagMacbeth™ ColorChecker Color Rendition Chart
For Help, press F1
NUM
Start
6:36 PM

Property	Value
Sensor	MI-2010SOCJ
Part number	MT9D111
Sensor version	REV1
Sensor type	SOC
Software:	
Build version	2.6.0.0
Build type	Beta5
Build date	031705
Demo Board:	DEMO2 (B2)
Firmware	7.19
Clock	27.0MHz
Width	800
Height	600
Format	YCbCr
Frames	4237
Dropped frames	0
FPS (Sensor)	10.6
FPS (Camera)	9.0
FPS (Display)	9.0
Cursor pos	248,478
Y: Raw Data	213
Cb: Raw Data	124
Cr: Raw Data	127
R: Output	211
G: Output	215
B: Output	205

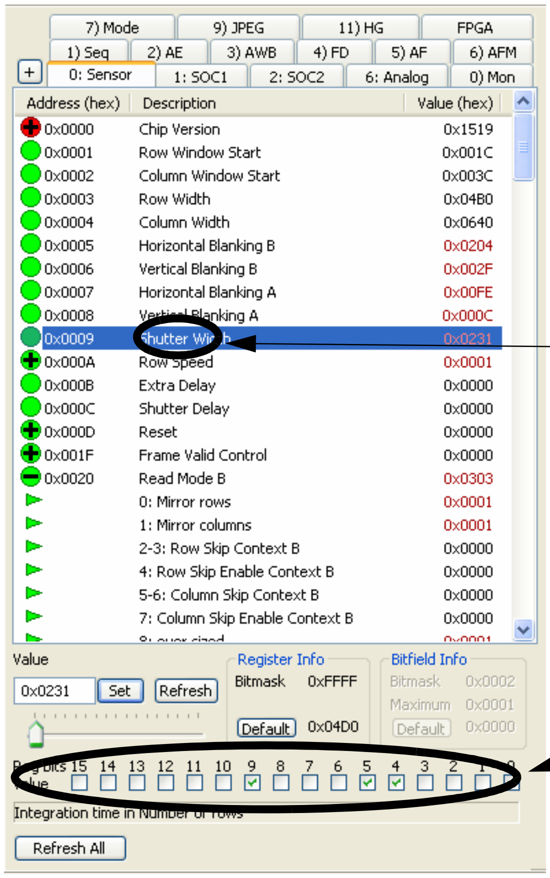
Mouse Selection
- Off
- [✓ Row]
- Column
- Rectangle
- Point

Since the light hits vertically from top, select the ROW line which comes across the top of the white color square. Thus, the maximum Green needed for white balance to be obtained.

a. Open the Cumulative Intensity Graph and adjust the shutter width so the Green in white patch is at 220



The region of white has the greatest Green value (for red light, red component has the greatest value)



Address (hex)	Description	Value (hex)
0x0000	Chip Version	0x1519
0x0001	Row Window Start	0x001C
0x0002	Column Window Start	0x003C
0x0003	Row Width	0x04B0
0x0004	Column Width	0x0640
0x0005	Horizontal Blanking B	0x0204
0x0006	Vertical Blanking B	0x002F
0x0007	Horizontal Blanking A	0x00FE
0x0008	Vertical Blanking A	0x000C
0x0009	Shutter Width	0x0231
0x000A	Row Speed	0x0001
0x000B	Extra Delay	0x0000
0x000C	Shutter Delay	0x0000
0x000D	Reset	0x0000
0x001F	Frame Valid Control	0x0000
0x0020	Read Mode B	0x0303
	0: Mirror rows	0x0001
	1: Mirror columns	0x0001
	2-3: Row Skip Context B	0x0000
	4: Row Skip Enable Context B	0x0000
	5-6: Column Skip Context B	0x0000
	7: Column Skip Enable Context B	0x0000
	8: over sized	0x0001

Value: 0x0231 Set Refresh

Register Info — Bitmask 0xFFFF; Default 0x04D0

Bitfield Info — Bitmask 0x0002; Maximum 0x0001; Default 0x0000

Reg bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value							✓				✓	✓				

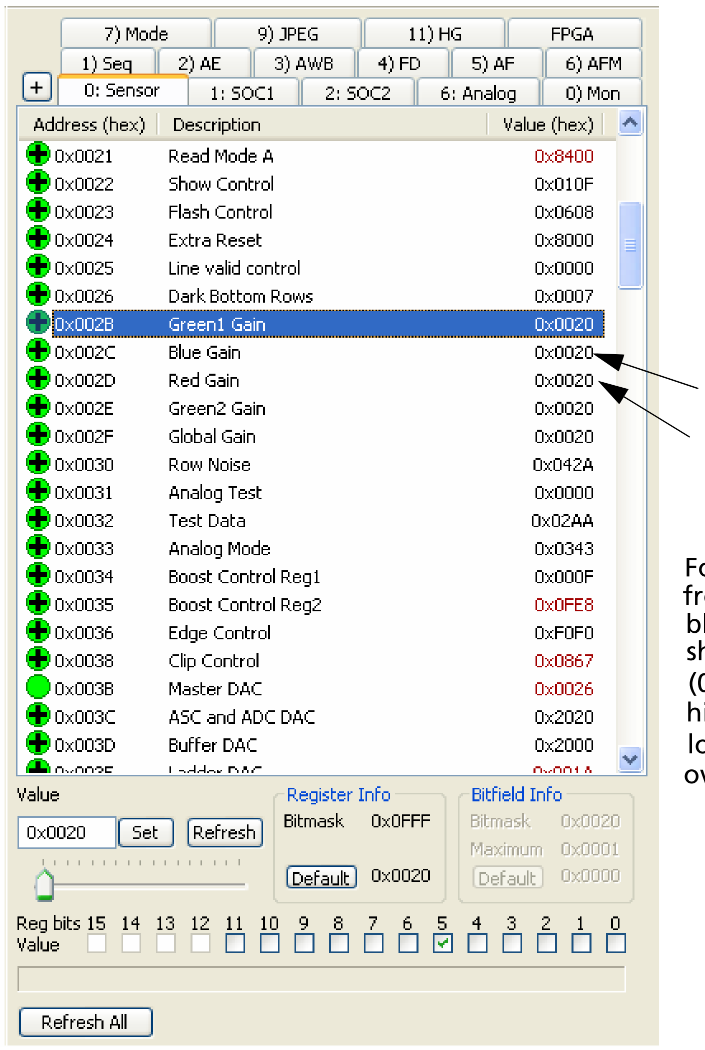
Integration time in Number of rows

Refresh All

c. Adjust "Shutter Width" register values so the maximum Green in the intensity graph reaches "220," as shown in the graph above.

d. Adjust the "Shutter Width" register by changing the binary bits of gain. Start from left to right to approximate values.

b. Adjust the Red and Blue Gain, so that the curves overlaps each other approximately (start from Red gain and then Blue gain)



Address (hex)	Description	Value (hex)
0x0021	Read Mode A	0x8400
0x0022	Show Control	0x010F
0x0023	Flash Control	0x0608
0x0024	Extra Reset	0x8000
0x0025	Line valid control	0x0000
0x0026	Dark Bottom Rows	0x0007
0x002B	Green1 Gain	0x0020
0x002C	Blue Gain	0x0020
0x002D	Red Gain	0x0020
0x002E	Green2 Gain	0x0020
0x002F	Global Gain	0x0020
0x0030	Row Noise	0x042A
0x0031	Analog Test	0x0000
0x0032	Test Data	0x02AA
0x0033	Analog Mode	0x0343
0x0034	Boost Control Reg1	0x000F
0x0035	Boost Control Reg2	0x0FE8
0x0036	Edge Control	0xF0F0
0x0038	Clip Control	0x0867
0x003B	Master DAC	0x0026
0x003C	ASC and ADC DAC	0x2020
0x003D	Buffer DAC	0x2000

Value: 0x0020

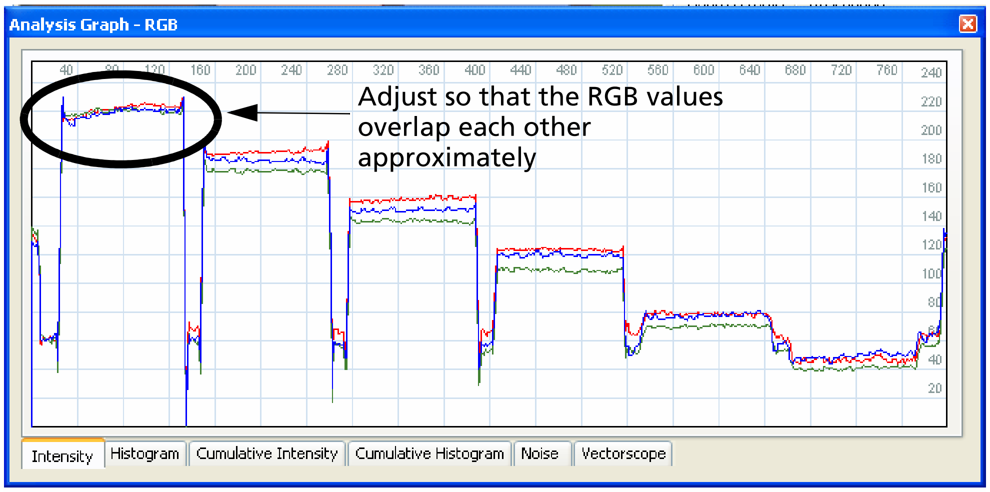
Register Info — Bitmask 0x0FFF; Default 0x0020

Bitfield Info — Bitmask 0x0020; Maximum 0x0001; Default 0x0000

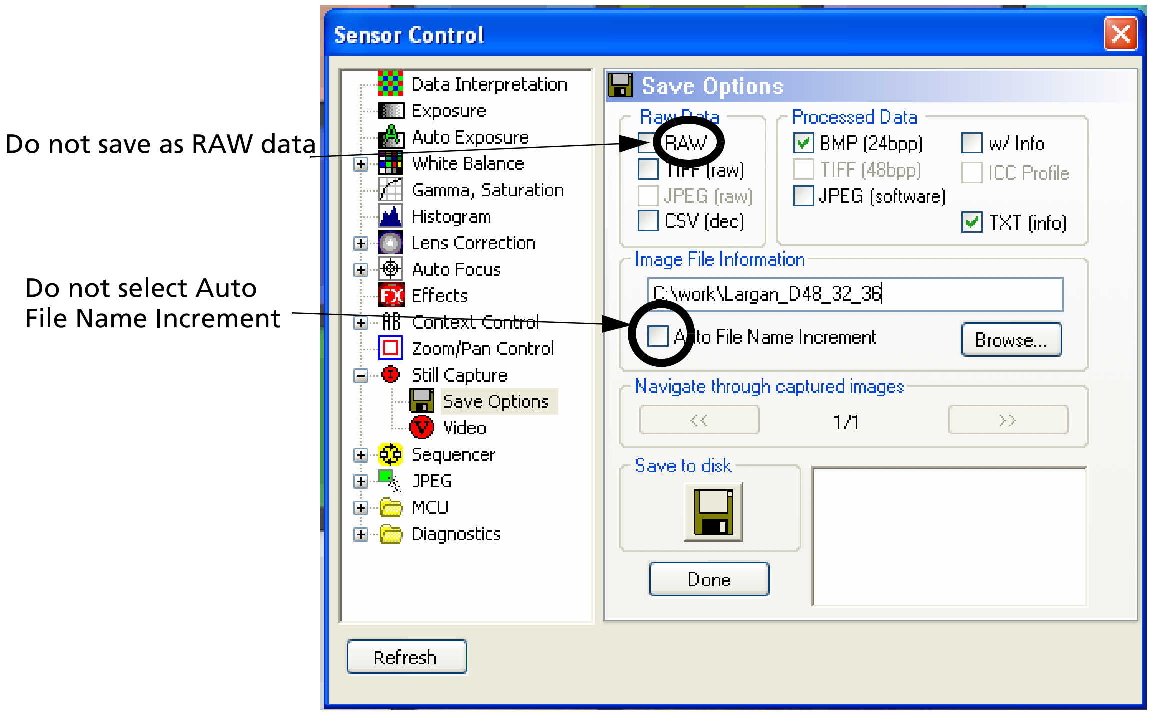
Reg bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value					☐	☐	☐	☐	☐	☐	✓	☐	☐	☐	☐	☐

Start from adjusting Red gain and move to Blue gain. (Green gain should not be adjusted it should be the default 0x0020).

For red light, start from adjusting blue gain. Green gain should not be adjusted (0x0020). If Red is higher than Green, then lower Red gain so it overlaps with Green.



c. Capture the picture and save it. The naming convention is important here. The image should be named: *Module ID* _ *Light Condition* and *Red Gain* _ *Green Gain* _ *Blue Gain*. For the light condition field, the user should enter "D" for D65 light and "A" for A28 light. For example, in the GUI below, we show capturing an image taken by the Largan module with D65 and Red Gain = 48, Green Gain = 32, and Blue Gain = 36. This example is explained throughout this document.



Raw Data: ☐ RAW, ☐ TIFF (raw), ☐ JPEG (raw), ☐ CSV (dec)

Processed Data: ☑ BMP (24bpp), ☐ TIFF (48bpp), ☐ JPEG (software), ☐ w/ Info, ☐ ICC Profile, ☑ TXT (info)

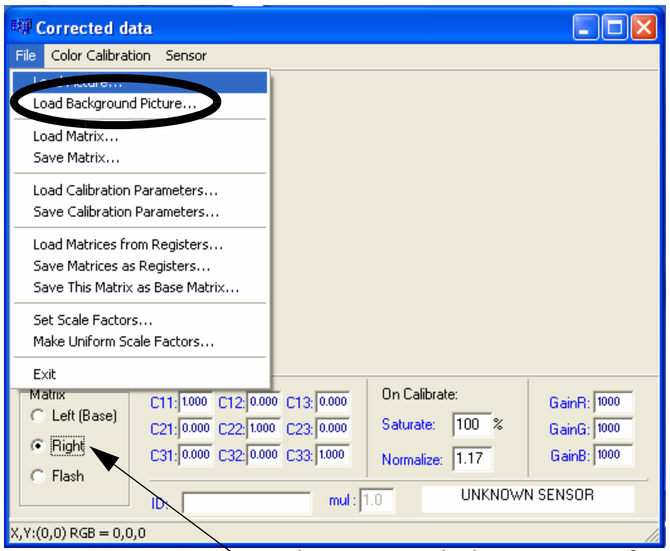
Image File Information: C:\work\Largan_D48_32_36

☐ Auto File Name Increment

d. With all settings staying the same, take away the Color Rendition Chart from the light box and take an image of the background. Capture this image and save it as "Largan_D48_32_36_bk". Note the "_bk" stands for background.

e. Repeat Step 3 (Calibration in DevWare) for incandescent light and save the color chart picture and the background picture as a different file name .

4. Calibration of Color Correction Matrix


Corrected data
File
Color Calibration
Sensor
Load Background Picture...
Load Matrix...
Save Matrix...
Load Calibration Parameters...
Save Calibration Parameters...
Load Matrices from Registers...
Save Matrices as Registers...
Save This Matrix as Base Matrix...
Set Scale Factors...
Make Uniform Scale Factors...
Exit
Matrix
Left (Base)
Right
Flash
On Calibrate:
Saturate:
Normalize:
ID:
mul :
UNKNOWN SENSOR

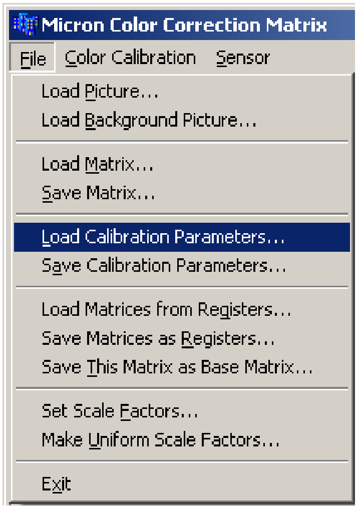
C11:	1.000	C12:	0.000	C13:	0.000
C21:	0.000	C22:	1.000	C23:	0.000
C31:	0.000	C32:	0.000	C33:	1.000

Saturate: 100 %
Normalize: 1.17
mul : 1.0
GainR: 1000
GainG: 1000
GainB: 1000
X,Y:(0,0) RGB = 0,0,0

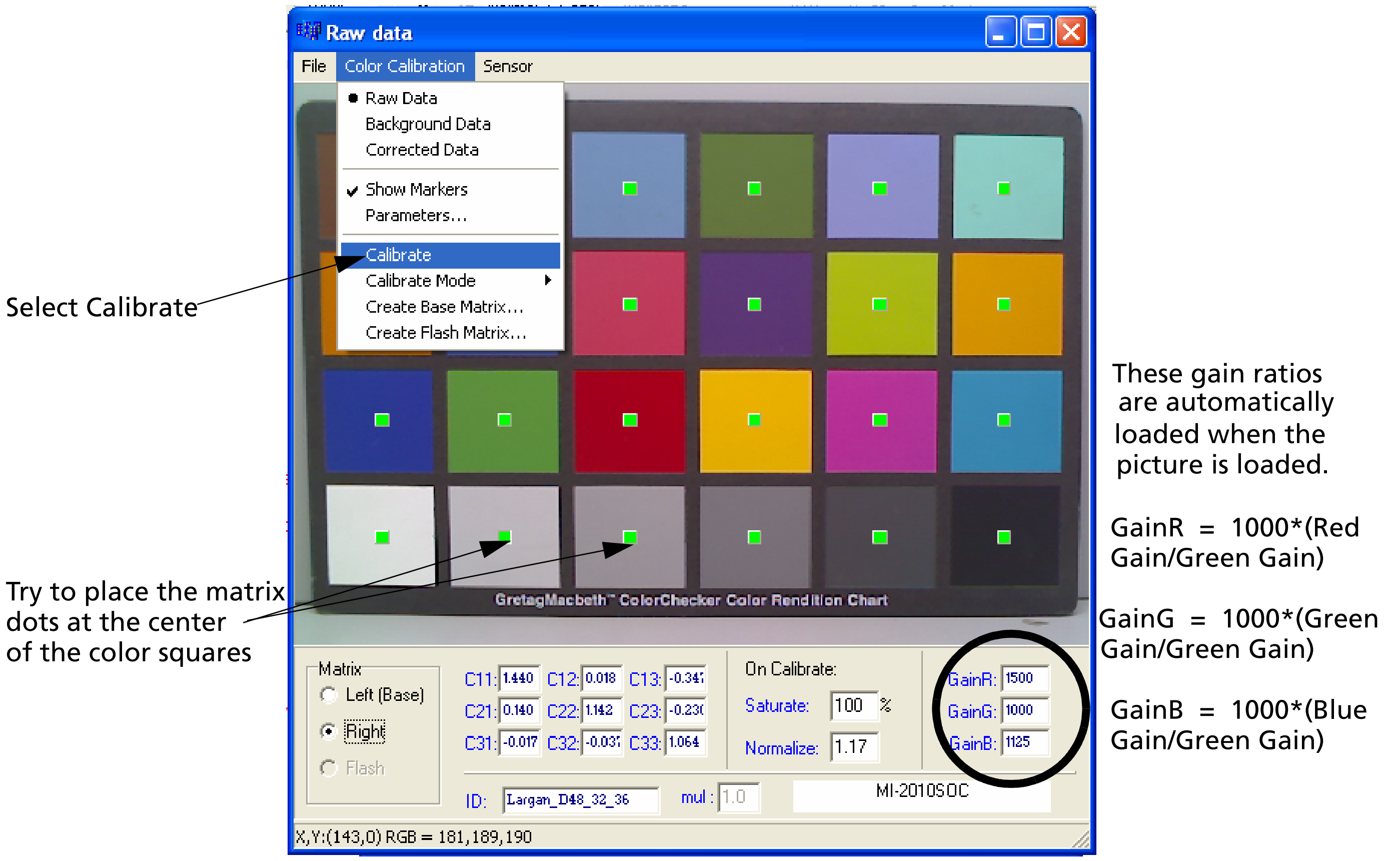
Make sure Daylight picture is for the Right matrix

a. Load the D_48_32_36.bmp as the “Right” picture. Note that D_48_32_36_bk.bmp is loaded automatically as the background picture

b. Load the calibration target (it is located in C:\Program Files\Micron Imaging\cccm). The target is called *DAY.cal* for D65 lighting and *A.cal* for A28 lighting.


Micron Color Correction Matrix
File
Color Calibration
Sensor
Load Picture...
Load Background Picture...
Load Matrix...
Save Matrix...
Load Calibration Parameters...
Save Calibration Parameters...
Load Matrices from Registers...
Save Matrices as Registers...
Save This Matrix as Base Matrix...
Set Scale Factors...
Make Uniform Scale Factors...
Exit

c. Manipulate and adjust so that the Matrix dots are approximately located at the middle of each color square. If the GainR is less than 1,000 for incandescent light, change the number to 1000; then go to the **Color Calibration** and select **Calibrate**.



Matrix: Left (Base) [✓ Right] Flash

C11:	1.440	C12:	0.018	C13:	-0.347
C21:	0.140	C22:	1.142	C23:	-0.230
C31:	-0.017	C32:	-0.037	C33:	1.064

On Calibrate:
Saturate: 100 %
Normalize: 1.17

GainR: 1500
GainG: 1000
GainB: 1125

ID: Largan_D48_32_36 mul : 1.0 MI-2010SOC

X,Y:(143,0) RGB = 181,189,190

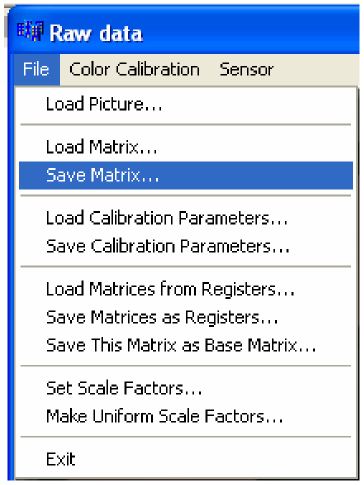
These gain ratios are automatically loaded when the picture is loaded.

GainR = 1000*(Red Gain/Green Gain)

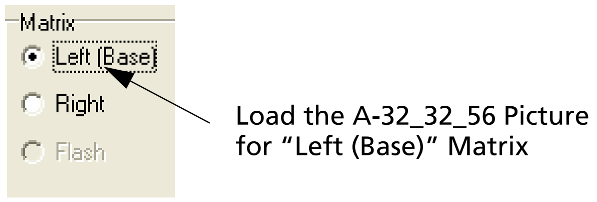
GainG = 1000*(Green Gain/Green Gain)

GainB = 1000*(Blue Gain/Green Gain)

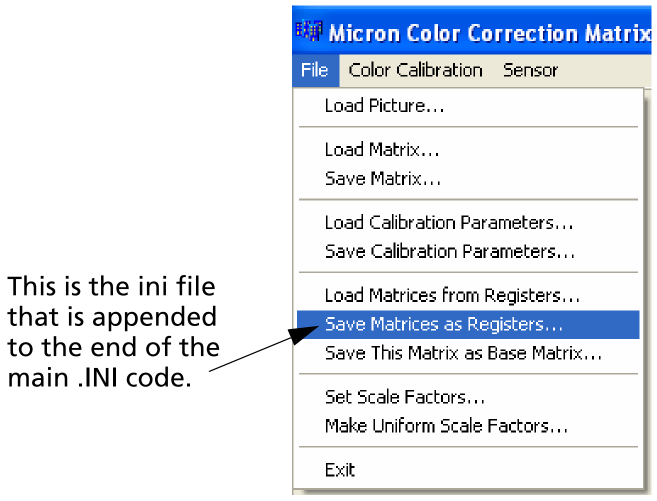
d. Save the calibrated Daylight picture as a Matrix data file.


Raw data
File
Color Calibration
Sensor
Load Picture...
Load Matrix...
Save Matrix...
Load Calibration Parameters...
Save Calibration Parameters...
Load Matrices from Registers...
Save Matrices as Registers...
Save This Matrix as Base Matrix...
Set Scale Factors...
Make Uniform Scale Factors...
Exit

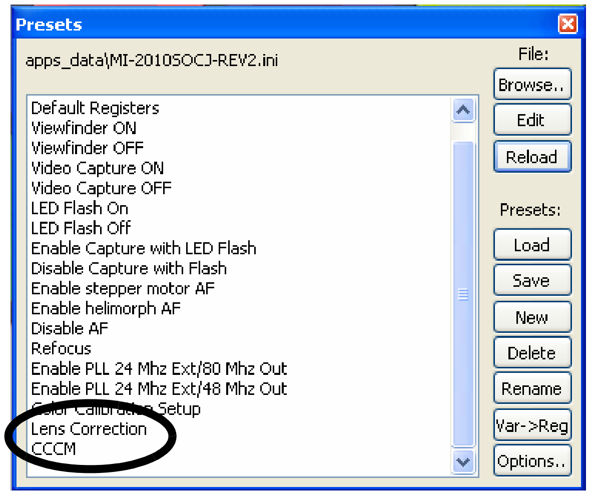
e. After calibrating the picture, repeat steps [a]–[d]. loading the Incandescent light picture as the "Left (Base)" Matrix



f. After both pictures have been calibrated and the corresponding matrix data file has been saved, save a combined *.ini* file

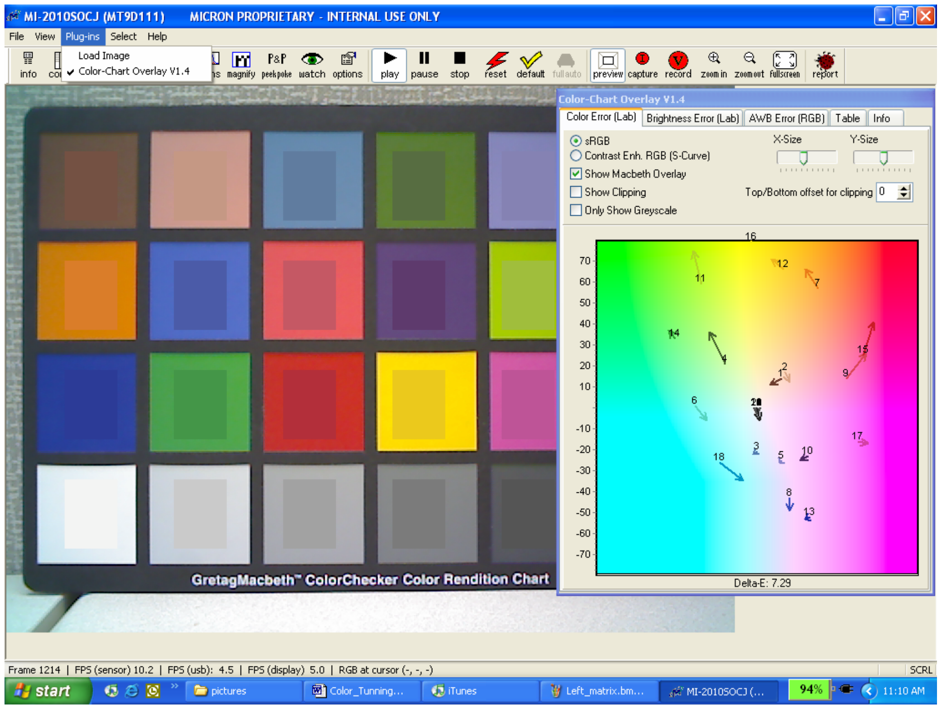

Micron Color Correction Matrix
File
Color Calibration
Sensor
Load Picture...
Load Matrix...
Save Matrix...
Load Calibration Parameters...
Save Calibration Parameters...
Load Matrices from Registers...
Save Matrices as Registers...
Save This Matrix as Base Matrix...
Set Scale Factors...
Make Uniform Scale Factors...
Exit
This is the ini file that is appended to the end of the main .INI code.

5. Verification
   a. Hardware setup is the same as described above.
   b. Load DevWare and reset all registers (this enables AWB, Color Correction, AE, and Gamma Correction).
   c. Load **Lens Correction**, then load **CCCM** in the **Presets** window.



d. Go to Plug-ins>Color-Chart Overlay v1.4. Make sure the patches are right on top of the color bars in the image. If not, adjust X-Size and Y-Size to have the patches positioned

e. The Delta-E parameter on **Color-Chart Overlay V1.4** should be no more than 8.0



(●) sRGB
( ) Contrast Enh. RGB (S-Curve)
[✓] Show Macbeth Overlay
[ ] Show Clipping
[ ] Only Show Greyscale
X-Size
Y-Size
Top/Bottom offset for clipping 0

Delta-E: 7.29

Frame 1214 | FPS (sensor) 10.2 | FPS (usb): 4.5 | FPS (display) 5.0 | RGB at cursor (-, -, -)

f. If needed, repeat the color correction procedure to obtain improved register setting values and repeat the verification process to check image quality.

Related Register List

REG = 1, 0x60 //COLOR_CORR_MATRIX_SCALE_14

REG = 1, 0x61 //COLOR_CORR_MATRIX_SCALE_11

REG = 1, 0x62 //COLOR_CORR_MATRIX_1_2

REG = 1, 0x63 //COLOR_CORR_MATRIX_3_4

REG = 1, 0x64 //COLOR_CORR_MATRIX_5_6

REG = 1, 0x65 //COLOR_CORR_MATRIX_7_8

REG = 1, 0x66 //COLOR_CORR_MATRIX_9

REG = 1, 0x6A //DIGITAL_GAIN_1_RED

REG = 1, 0x6B //DIGITAL_GAIN_1_GREEN1

REG = 1, 0x6C //DIGITAL_GAIN_1_GREEN2

REG = 1, 0x6D //DIGITAL_GAIN_1_BLUE

Mode Driver–Setting Up Preview (A) and Capture (B) Modes

The mode driver (ID = 7) serves two major functions:

1. Storing local copies of image sensor and pipeline registers for both preview and capture modes so that they may be uploaded at the appropriate time between frames (to avoid midframe changes)
2. Generating a gamma correction table that may be selected to include a predefined level of contrast enhancement, thus adding contrast control to this part

The mode driver contains shadow registers for context A (preview) and context B (capture). These shadow registers upload their values to the corresponding image sensor and pipeline control hardware registers at the proper time so as not to introduce midframe changes that would cause frame corruption. These shadow registers have been chosen to provide all foreseeable changes that a user would want to make between preview and captures modes (see data sheet for affected registers for each mode driver variable). There are additional preview-specific and capture-specific registers in the sensor core (R0x05:0, R0x06:0, R0x07:0, R0x08:0, R0x20:0, R0x21:0), that are not included in the mode driver's values.

Upon power-up, the user should upload all nondefault register values desired, including the mode driver values. The user does not need to upload to any hardware registers that correspond to the mode driver values because the registers are overwritten upon initialization, a context change (preview to capture or vise versa), or a sequencer REFRESH or REFRESH_MODE command.

The user may also change these mode driver variables at any time, but their changes are not reflected in the images until either: (a) the user issues a REFRESH or REFRESH_MODE command to the sequencer, (b) the user changes the sequencer state from preview to capture or vice versa, or (c) the user issues a STANDBY sequencer command and then returns. This allows the user to change values without affecting the immediate image processing, avoiding image corruption.

“Gamma and Contrast” on page 29 describes the mode driver’s gamma and contrast functionality.

MT9D131 Register Wizard

The MT9D131 Register Wizard tool is a software program that allows a user to generate the proper settings for timing, PLL, state parameters, and gamma/contrast parameters for the sensor.

After specifying the desired operating frequency, frame rate, resolution, and other parameters, the user can save the resulting register/variable values in an INI file format that can easily be loaded in Micron's DevWare demo software.

The tool is in its prerelease form and has not yet been extensively tested. Additional features may be added in the future. Any bug reports should be reported to your local Micron Field Applications Engineer (FAE).

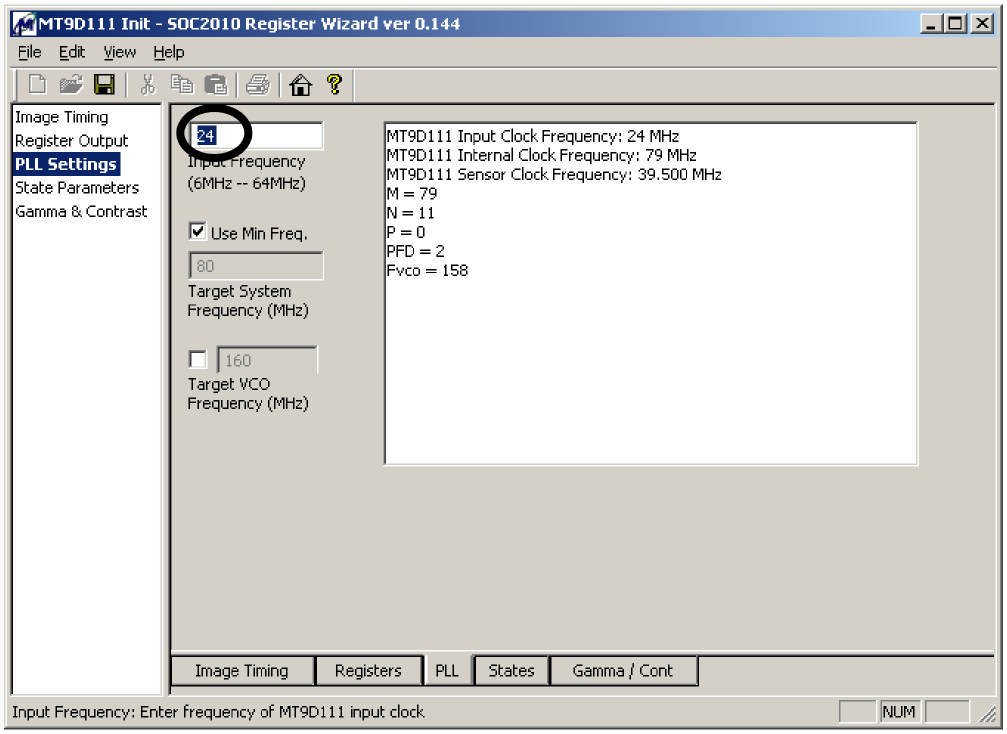
Procedure

After opening the MT9D131 Register Wizard tool, the user should first go to the PLL settings section to specify the input clock and PLL output frequencies (see Figure 31.)

In the first text box, enter the input clock frequency to the sensor (EXTCLK). Next, enter the targeted output frequency of the PLL in the second text box. If the text box is disabled, uncheck the **Use Min Freq** checkbox. Alternatively, you can leave the **Use Min Freq** box checked and let the tool select the minimum PLL output frequency needed based on the **Image Timing** section.

The window on the right will then show the required PLL settings—set by the M, N, and P values—to achieve the named configuration.

Figure 31: Input Clock and PLL Output Frequencies



```
MT9D111 Input Clock Frequency: 24 MHz
MT9D111 Internal Clock Frequency: 79 MHz
MT9D111 Sensor Clock Frequency: 39.500 MHz
M = 79
N = 11
P = 0
PFD = 2
Fvco = 158
```

Input Frequency: Enter frequency of MT9D111 input clock

The **Target VCO Frequency** (MHz) field allows the user to specify a target VCO frequency—the frequency of one of the stages of the PLL, which has a valid range of 110 to 240 MHz. The tool limits its calculations based on this range, as well as other requirements—see Table 13 for more details.

Table 13: PLL Specifications

Specification	Equation	Min	Max
M	–	16	–
f_{PFD}	$f_{IN}/(N+1)$	2 MHz	13 MHz
f_{VCO}	$f_{PFD}*M$	110 MHz	240 MHz
f_{OUT}	$f_{VCO}/(2*(P+1))$	6 MHz	80 MHz
f_{IN}	–	6 MHz	64 MHz

The tool assumes the use of PLL. However, if the sensor is operating directly from the EXTCLK frequency (no PLL), follow the steps below:

1. Enter the same frequency in the **Input Frequency** and **Target System Frequency** text boxes. The **Use Min Freq** checkbox should be unchecked.
2. After all the parameters are set from the other sections (Image Timing, State Parameters, and Gamma and Contrast), save the settings as an INI file.
3. Use a text editor to remove the following lines from the INI file.

```
REG  =  0, 0x66, 0x****        //PLL Control 1  =  ****
REG  =  0, 0x67, 0x****        //PLL Control 2  =  ****
REG  =  0, 0x65, 0xA000        //Clock CNTRL: PLL ON  =  40960
REG  =  0, 0x65, 0x2000        //Clock CNTRL: USE PLL  =  8192
```

Figure 32: Image Timing Section

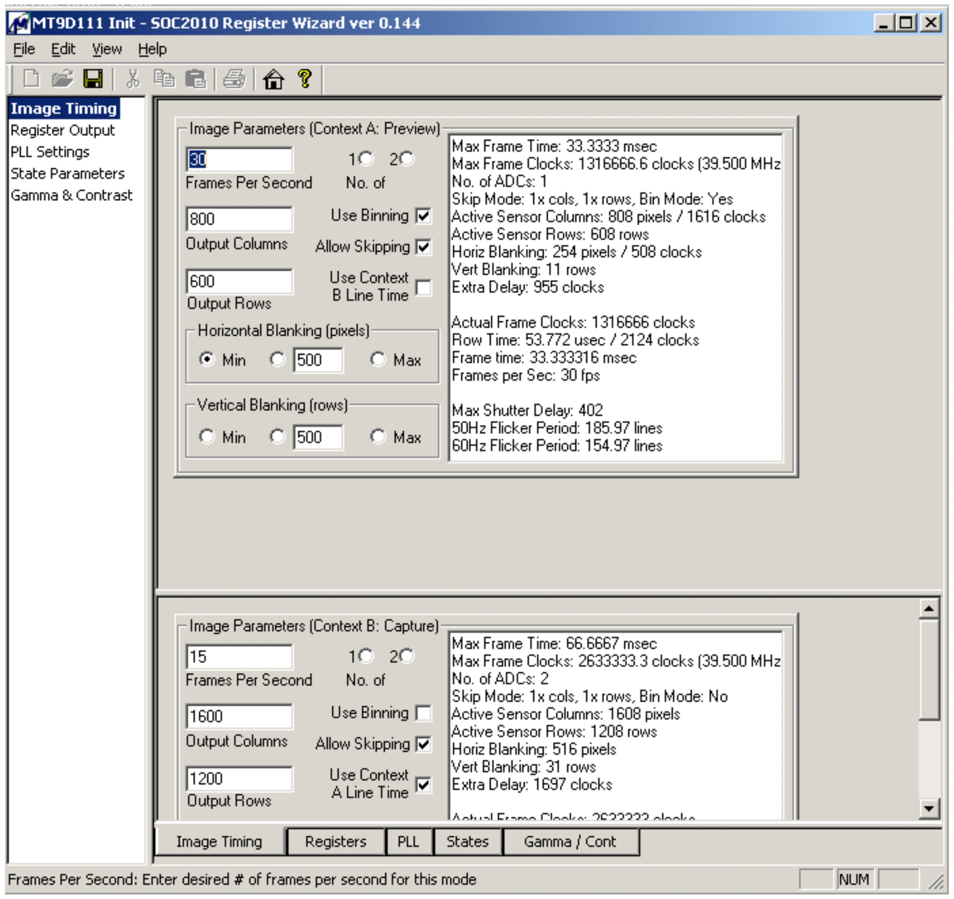


Image Parameters (Context A: Preview)

30 Frames Per Second
No. of ADCs: 1 ○ 2 ○
800 Output Columns
Use Binning ✓
Allow Skipping ✓
600 Output Rows
Use Context B Line Time ☐
Horizontal Blanking (pixels): ● Min ○ 500 ○ Max
Vertical Blanking (rows): ○ Min ○ 500 ○ Max

Max Frame Time: 33.3333 msec
Max Frame Clocks: 1316666.6 clocks (39.500 MHz
No. of ADCs: 1
Skip Mode: 1x cols, 1x rows, Bin Mode: Yes
Active Sensor Columns: 808 pixels / 1616 clocks
Active Sensor Rows: 608 rows
Horiz Blanking: 254 pixels / 508 clocks
Vert Blanking: 11 rows
Extra Delay: 955 clocks

Actual Frame Clocks: 1316666 clocks
Row Time: 53.772 usec / 2124 clocks
Frame time: 33.333316 msec
Frames per Sec: 30 fps

Max Shutter Delay: 402
50Hz Flicker Period: 185.97 lines
60Hz Flicker Period: 154.97 lines

Image Parameters (Context B: Capture)

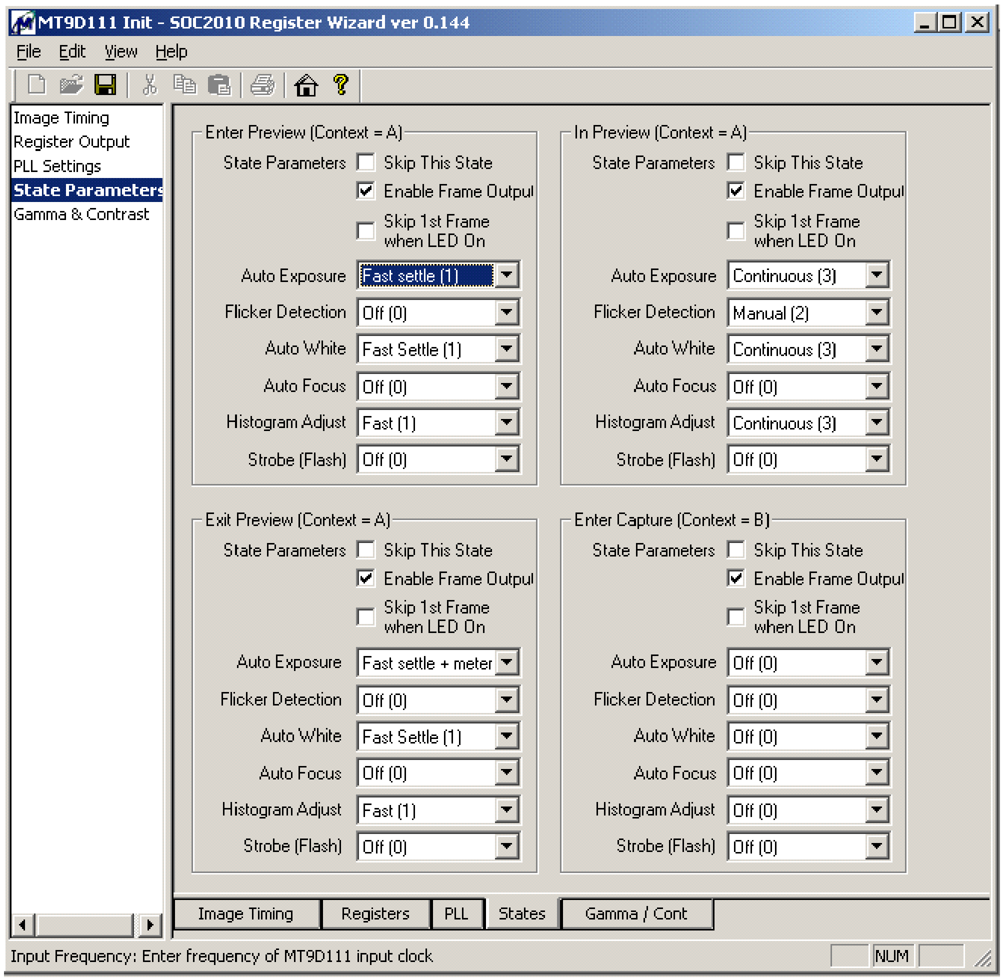
15 Frames Per Second
No. of ADCs: 1 ○ 2 ○
1600 Output Columns
Use Binning ☐
Allow Skipping ✓
1200 Output Rows
Use Context A Line Time ✓

Max Frame Time: 66.6667 msec
Max Frame Clocks: 2633333.3 clocks (39.500 MHz
No. of ADCs: 2
Skip Mode: 1x cols, 1x rows, Bin Mode: No
Active Sensor Columns: 1608 pixels
Active Sensor Rows: 1208 rows
Horiz Blanking: 516 pixels
Vert Blanking: 31 rows
Extra Delay: 1697 clocks
Actual Frame Clocks: 2633333 clocks

Frames Per Second: Enter desired # of frames per second for this mode

The Image Timing section allows the user to configure the frame rate, resolution, ADC mode, binning option, skipping option, and blanking option for each context. If the input values are beyond specifications, the corresponding text box will be highlighted in yellow. A warning message will also appear in the window on the right side. For example, if a user enters a frame rate that is too high, the warning message will specify the maximum frame rate achievable based on the current operating frequency. By default, the **Use Context A Line Time** checkbox is selected. For the firmware drivers (such as auto exposure) to function optimally, it is necessary to match the line (row) time for both contexts.

Figure 33: State Parameters Tab



Enter Preview (Context = A)
State Parameters: ☐ Skip This State; ☑ Enable Frame Output; ☐ Skip 1st Frame when LED On
Auto Exposure: Fast settle (1)
Flicker Detection: Off (0)
Auto White: Fast Settle (1)
Auto Focus: Off (0)
Histogram Adjust: Fast (1)
Strobe (Flash): Off (0)

In Preview (Context = A)
State Parameters: ☐ Skip This State; ☑ Enable Frame Output; ☐ Skip 1st Frame when LED On
Auto Exposure: Continuous (3)
Flicker Detection: Manual (2)
Auto White: Continuous (3)
Auto Focus: Off (0)
Histogram Adjust: Continuous (3)
Strobe (Flash): Off (0)

Exit Preview (Context = A)
State Parameters: ☐ Skip This State; ☑ Enable Frame Output; ☐ Skip 1st Frame when LED On
Auto Exposure: Fast settle + meter
Flicker Detection: Off (0)
Auto White: Fast Settle (1)
Auto Focus: Off (0)
Histogram Adjust: Fast (1)
Strobe (Flash): Off (0)

Enter Capture (Context = B)
State Parameters: ☐ Skip This State; ☑ Enable Frame Output; ☐ Skip 1st Frame when LED On
Auto Exposure: Off (0)
Flicker Detection: Off (0)
Auto White: Off (0)
Auto Focus: Off (0)
Histogram Adjust: Off (0)
Strobe (Flash): Off (0)

Input Frequency: Enter frequency of MT9D111 input clock

In the **State Parameters** section, the user can configure the mode for each driver in the following four states: Enter Preview, Preview, Leave Preview, and Enter Capture. The configurable drivers are: Auto Exposure, Flicker Detection, Auto White Balance, Auto Focus, Histogram, and Flash. There is also an option to skip a particular state by using the associated checkbox.

Figure 34: State Diagram and Transitions of the MT9D131

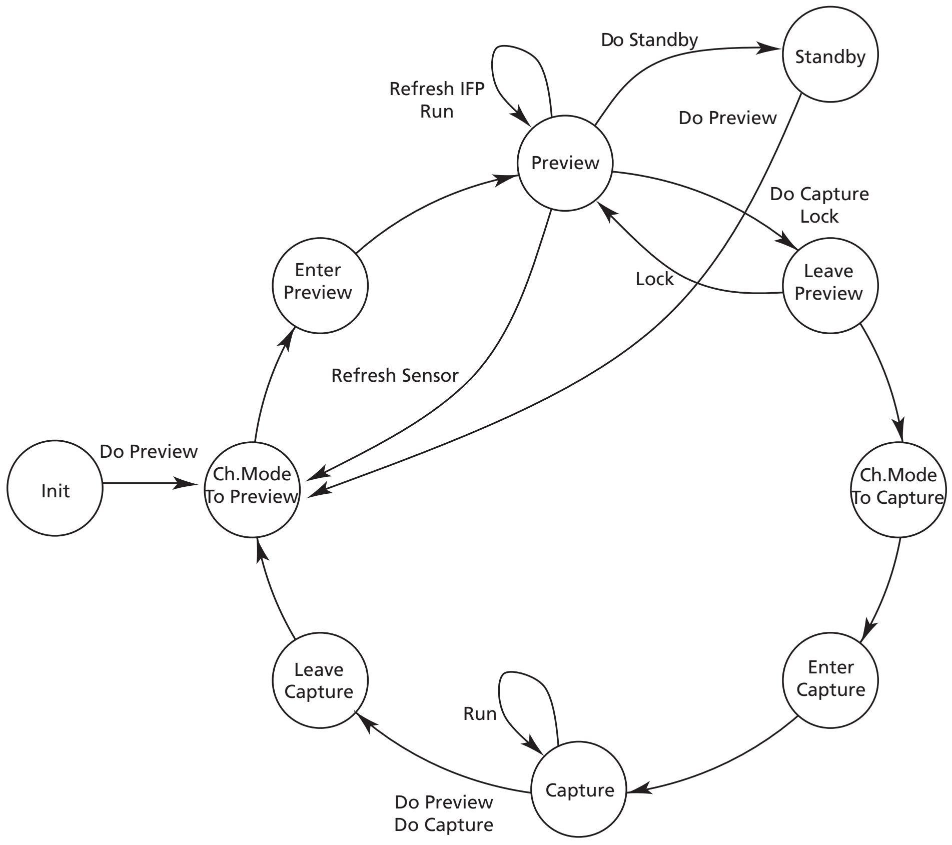
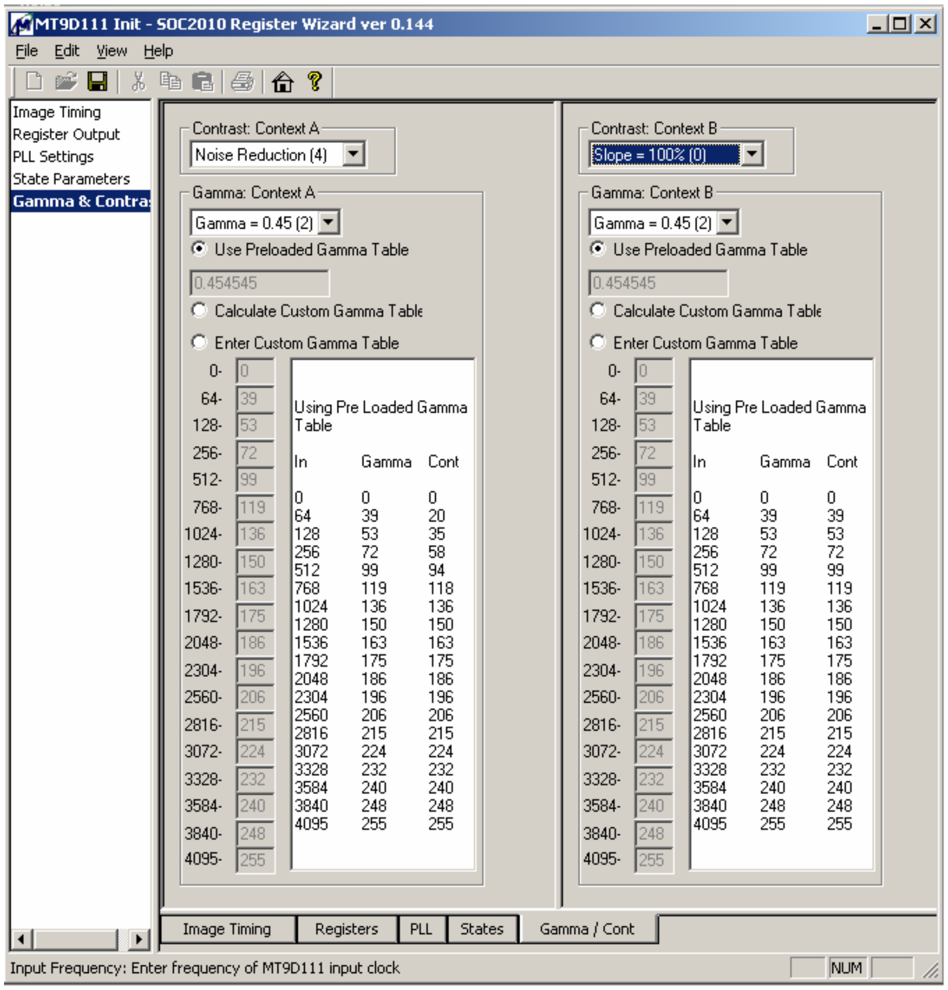


Figure 35: Gamma and Contrast Tab


MT9D111 Init - SOC2010 Register Wizard ver 0.144
File Edit View Help
Image Timing
Register Output
PLL Settings
State Parameters
Gamma & Contra
Image Timing | Registers | PLL | States | Gamma / Cont
NUM

Contrast: Context A: Noise Reduction (4)

Gamma: Context A: Gamma = 0.45 (2)

[✓ Use Preloaded Gamma Table] 0.454545

Calculate Custom Gamma Table

Enter Custom Gamma Table

Contrast: Context B: Slope = 100% (0)

Gamma: Context B: Gamma = 0.45 (2)

[✓ Use Preloaded Gamma Table] 0.454545

Calculate Custom Gamma Table

Enter Custom Gamma Table

	Context A	Context B
0-	0	0
64-	39	39
128-	53	53
256-	72	72
512-	99	99
768-	119	119
1024-	136	136
1280-	150	150
1536-	163	163
1792-	175	175
2048-	186	186
2304-	196	196
2560-	206	206
2816-	215	215
3072-	224	224
3328-	232	232
3584-	240	240
3840-	248	248
4095-	255	255

Using Pre Loaded Gamma Table (Context A)

In	Gamma	Cont
0	0	0
64	39	20
128	53	35
256	72	58
512	99	94
768	119	118
1024	136	136
1280	150	150
1536	163	163
1792	175	175
2048	186	186
2304	196	196
2560	206	206
2816	215	215
3072	224	224
3328	232	232
3584	240	240
3840	248	248
4095	255	255

Using Pre Loaded Gamma Table (Context B)

In	Gamma	Cont
0	0	0
64	39	39
128	53	53
256	72	72
512	99	99
768	119	119
1024	136	136
1280	150	150
1536	163	163
1792	175	175
2048	186	186
2304	196	196
2560	206	206
2816	215	215
3072	224	224
3328	232	232
3584	240	240
3840	248	248
4095	255	255

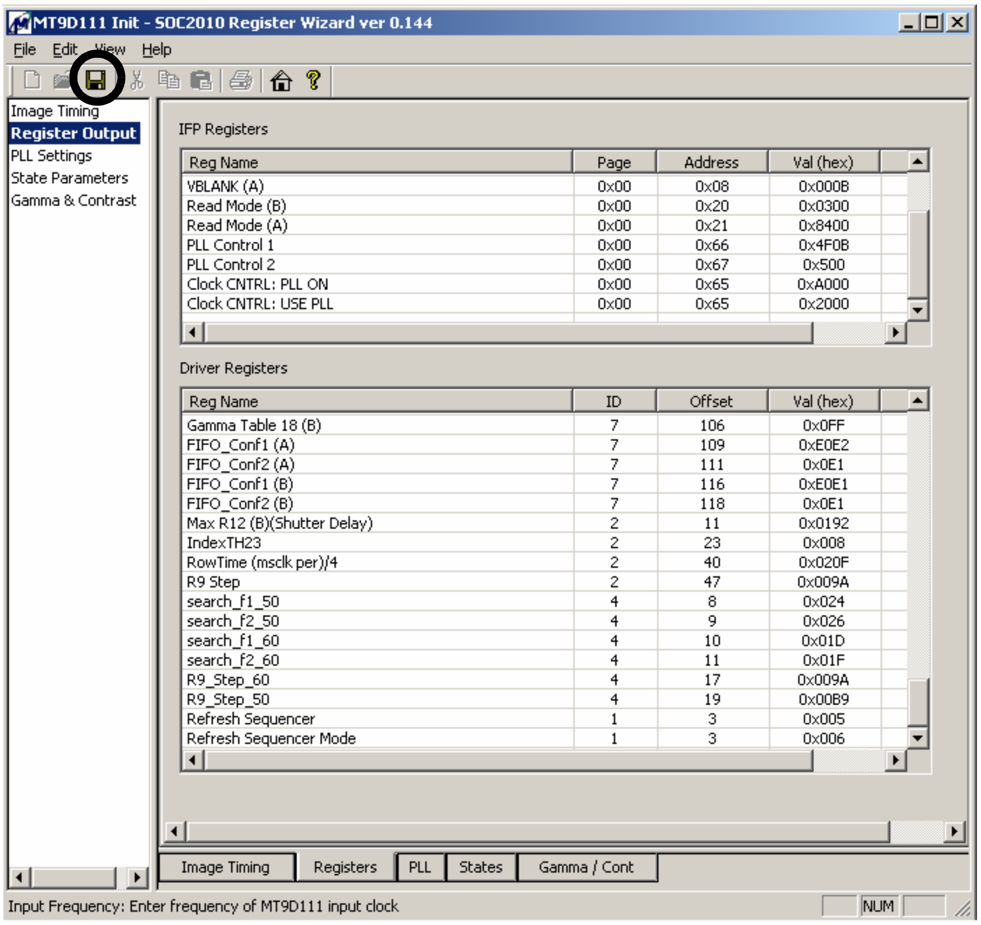
Input Frequency: Enter frequency of MT9D111 input clock

In the final section—Gamma & Contrast—the user can select predefined gamma and contrast settings. The user also has the option to manually program the gamma/contrast table.

The available gamma options are: 1.0, 0.56, 0.45, or user-defined. Contrast options are: 100 percent (no contrast increase), 125 percent, 150 percent, 175 percent, and noise-reduction.

For additional gamma or contrast settings, the user can modify individual data points (see Figure 35). Data point values may range from 0 to 255.

Figure 36: Register Output Tab



IFP Registers

Reg Name	Page	Address	Val (hex)
VBLANK (A)	0x00	0x08	0x000B
Read Mode (B)	0x00	0x20	0x0300
Read Mode (A)	0x00	0x21	0x8400
PLL Control 1	0x00	0x66	0x4F0B
PLL Control 2	0x00	0x67	0x500
Clock CNTRL: PLL ON	0x00	0x65	0xA000
Clock CNTRL: USE PLL	0x00	0x65	0x2000

Driver Registers

Reg Name	ID	Offset	Val (hex)
Gamma Table 18 (B)	7	106	0x0FF
FIFO_Conf1 (A)	7	109	0xE0E2
FIFO_Conf2 (A)	7	111	0x0E1
FIFO_Conf1 (B)	7	116	0xE0E1
FIFO_Conf2 (B)	7	118	0x0E1
Max R12 (B)(Shutter Delay)	2	11	0x0192
IndexTH23	2	23	0x008
RowTime (msclk per)/4	2	40	0x020F
R9 Step	2	47	0x009A
search_f1_50	4	8	0x024
search_f2_50	4	9	0x026
search_f1_60	4	10	0x01D
search_f2_60	4	11	0x01F
R9_Step_60	4	17	0x009A
R9_Step_50	4	19	0x00B9
Refresh Sequencer	1	3	0x005
Refresh Sequencer Mode	1	3	0x006

Input Frequency: Enter frequency of MT9D111 input clock

Once all the parameters are selected, the user can review all the associated register/variable values in the Register Output section of the tool, as shown in Figure 36.

To save these settings, click the diskette icon in the tool bar. Once the INI file is generated, it may be loaded into DevWare or converted to a different system format.

Note: DevWare/demo2 may not operate in the frequency specified by the user in the PLL Settings section. In this case, supply an external oscillator to the demo2 system with the same frequency that was entered in the tool.

The Register Wizard tool currently outputs register/variable values even if they have default values, so not all settings in the INI file are necessary to program the sensor. If an option is not modified, the user can remove the associated default register/variable setting in the INI file.

Note: This tool is under development. The information in this manual and the features in the Register Wizard tool are subject to change without notice. Report any typos, errors, or bugs to your local FAE.

Histogram Driver

Seting Up the Histogram Driver Variable

The histogram driver works to reduce image flare by continually analyzing the input image histogram and dynamically adjusting the black level, R0x59:1. When flare is present, the image histogram does not contain dark tones, causing the driver to subtract a higher black level, thus regaining the lost contrast (at the expense of dynamic range). In certain situations, the scene may contain no dark tones without flare. The histogram driver cannot distinguish this condition and alters the black level just the same, causing the image to have more contrast, which looks acceptable in many situations.

The variable *hg.maxDLevel* sets the maximum level that can be subtracted from the input data (set this value to match the lens flare percentage). For example, if a lens typically has a five percent flare, set this value to *0.05*1024 = 51*. To disable flare subtraction in all modes, set this value to 0. The maximum allowed value is 128. Read variable *hg.DLevel* to see the current subtracted value. The variable *hg.percent* indicates the percentage of histogram dark tones that needs to be clipped. The recommended value is 0. The variable *hg.DLevelBufferSpeed* controls the speed of adjustment, and has a range of 32 (fastest) to 1 (slowest).

The histogram driver operates with two sets of bins:

- The first set of bins is programmed to calculate low signal distribution in the image and is used to estimate black level, which should be subtracted to reduce image flare as described previously. Variables *hg.lowerLimit1* and *hg.binSize1* define the first set of bin. Variable *hg.lowerLimit1 = 0* sets offset for bin0 (divided by 4 on a 10-bit scale). Variable *hg.binSize1* sets bin width (0–4LSB, 1–8LSB, 2–16LSB, 7–512LSB on a 10-bit scale), so the first set of bins covers input signals 0–64.
- The second set of bins is programmed to calculate high signal distribution in the image and is used to estimate the percent of oversaturated pixels. Variables *hg.lowerLimit2 = 192* and *hg.binSize2 = 4* define the signal range from 768 to 1024, covered by the second histogram.

Variable *hg.scaleGFactor* sets the precision of histogram. If *hg.scaleGFactor = 1*, then R0xD8:2[15:18] has a maximal value = 255, if all pixels have the same value and hit into bin1. This variable defines minimal bin resolution as 1/256 of the total number of pixels in histogram window. If *hg.scaleGFactor = 2*, bin resolution is 1/512, and so on.

If the user wants to change one from the variables *hg.scaleGFactor*, *hg.lowerLimit1*, *hg.binSize1*, *hg.lowerLimit2*, or *hg.binSize2*, command DO_REFRESH (*seq.cmd = 5*) has to be called for new values have effect.

The histogram driver uses the same window as the AE driver.

Flash Strobe, Mechanical Shutter, and Global Reset

Still Capture Using Xenon/LED Flash with User-defined Image Quality Settings

The MT9D131 supports flash mode with user-defined settings for auto exposure and auto white balance for both xenon and LED flashes. To capture images using predefined settings for xenon or LED flash, the user (host) must:

1. Program the sequencer to select the appropriate flash type.
2. Turn the flash on.
3. Select the "load user defined settings" modes for the CaptureEnter state.
4. Disable all auto functionality for the PreviewLeave, CaptureEnter, and Capture states.
5. Transfer the sequencer into step-by-step mode (*sequencer.stepMode[0] = 1*) and call the DO_CAPTURE command (*sequencer.cmd*).
6. Wait until the sequencer comes to the PreviewLeave state and load the white balance and exposure settings to the corresponding AWB, AE, and histogram (HG) driver variables.
7. Release the step-by-step sequencer mode (*sequencer.stepMode[0] = 0*) so that the sequencer automatically passes all states from PreviewLeave back to the Preview state.

While in the CaptureModeChange state, the sequencer changes the imaging frame size from preview to capture. In the CaptureEnter state, the settings loaded to the driver variables in the PreviewLeave state are automatically loaded into SOC registers and the flash is engaged.

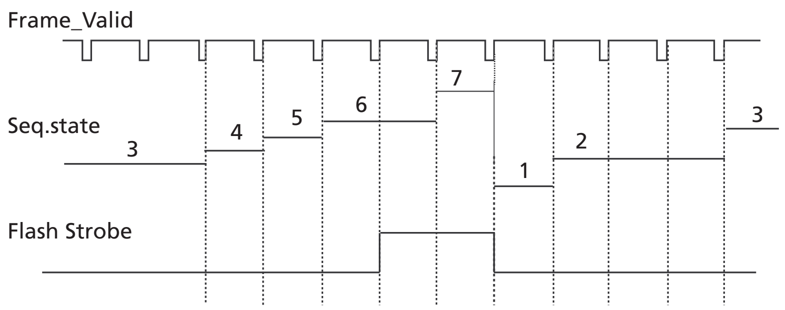
After this, the sequencer comes back to the Capture state. During N frames (see variable *sequencer.captureParams.numFrames*) when the sequencer stays in Capture state, the user must grab a frame. After N frames in the Capture state or the DO_CAPTURE command (whichever comes first), the sequencer turns off the flash pin (if flash was enabled for every frame), and goes back to the Preview state.

Command Sequence

- seq.sharedParams.flashType = 1 (LED) or 2(xenon)
- seq.capParEnter.flash = 129 // enable flash and loading of the user defined settings for "CaptureEnter" state.
- seq.capParEnter.skipframe = 32 // skip one frame after LED flash is enabled (optional, for LED flash only)
- seq.previewParLeave.ae = 0
- seq.previewParLeave.fd = 0
- seq.previewParLeave.awb = 0
- seq.previewParLeave.hg = 0
- seq.previewParLeave.flash = 0 // disable all auto functionality for PreviewLeave states
- seq.capParEnter.ae = 0
- seq.capParEnter.fd = 0
- seq.capParEnter.awb = 0
- seq.capParEnter.hg = 0 // disable all auto functionality for "CaptureEnter" states
- seq.captureParams.mode = 0 // disable all auto functionality for Capture states
- seq.captureParams.numFrames = 1 // specify how many frames sequencer should stay in Capture state

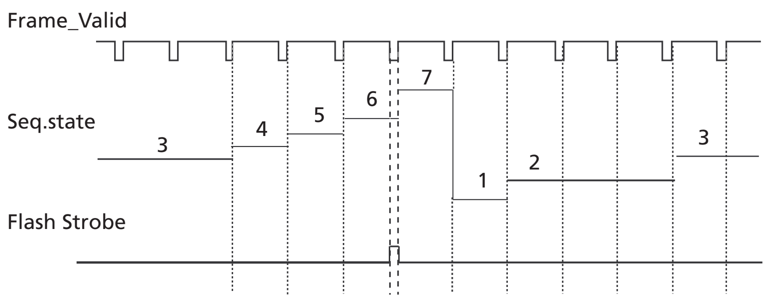
- Do the following:
  a. seq.stepMode = 3 // set step-by-step sequencer mode and let to do next step
  b. seq.cmd = 2 // call DO_CAPTURE command
  c. wait until seq.mode = 4 // wait until sequencer comes to PreviewLeave state.
- for (i = 0; i < 11; i++)
  awb.ccm[i] = ud_ccm[i]; // load user-defined (ud) color correction matrix
- awb.GainR = ud_GainR;
- awb.GainG = ud_GainG;
- awb.GainB = ud_GainB; // load ud digital WB gains
- ae.VirtGain = ud_VirtGain; // load ud virtual analog gain
- ae.DGainAE1 = DGainAE1; // AE digital gain1
- ae.DGainAE2 = DGainAE2; // AE digital gain2
- ae.R0x09 = ud_IT; // integration time
- ae.R65 = ud_R65; // ADC reference
- hg.DLevel = ud_DLevel; // dark level
- Do the following:
  a. seq.stepMode = 2 // release step-by-step sequencer mode
  b. wait until seq.mode = 7 and capture frame

Figure 37: LED Flash Timing Diagram



Note: Parameters: integration time = one frame. Skip one frame after LED is ON (seq.state 6). One frame in Capture state (seq.state 7). Same frame size for Preview and Capture modes.

Figure 38: Xenon Flash Timing Diagram



Note: Parameters: integration time = one frame. One frame in Capture state. Same frame size for Preview and Capture modes.

Still Capture Using LED Flash with Automatic White Balance and Exposure Control

The MT9D131 supports LED flash mode with automatic white balance and exposure control. The sequencer can be programmed to make fast AE and AWB calculations for a scene illuminated by an LED flash in the PreviewLeave and CaptureEnter states. (Calculating AE and AWB in PreviewLeave state rather than CaptureEnter state is recommended because the frame rate is usually faster in preview mode.)

Use the following steps to capture an image using AWB and AE:

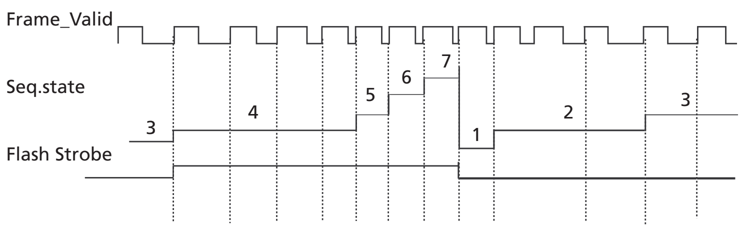
1. Program the sequencer to select the LED flash type.
2. Turn on the flash in the PreviewLeave and CaptureEnter states.
3. Enable the desired automatic functions in the PreviewLeave state, and disable them in the CaptureEnter and Capture states.
4. Specify how many frames the sequencer should stay in the Capture state. Call the DO_CAPTURE sequencer command, wait until sequencer comes to the Capture state, and grab the appropriate frame.

After receiving the DO_CAPTURE command, the sequencer changes to the PreviewLeave state (step 4), enables the LED flash, and calculates the AE and AWB values. These calculations take one to seven frames.

Command Sequence

- seq.sharedParams.flashType = 1 (LED)
- seq.previewParLeave.ae = 1
- seq.previewParLeave.fd = 0
- seq.previewParLeave.awb = 1
- seq.previewParLeave.hg = 1 // enable all auto functionality for PreviewLeave states
- seq.previewParLeave.flash = 1 // enable flash for PreviewLeave state.
- seq.previewParLeave.skipframe = 32 // skip one frame after LED flash is enabled (optional, for LED flash only)
- seq. capParEnter.ae = 0
- seq. capParEnter.fd = 0
- seq. capParEnter.awb = 0
- seq. capParEnter.hg = 0 // disable all auto functionality for CaptureEnter states
- seq.capParEnter.flash = 1 // enable flash for CaptureEnter state.
- seq.captureParams.mode = 0 // disable all auto functionality for Capture states
- seq.captureParams.numFrames = 1 // specify how many frames sequencer should stay in Capture state
- Do the following:
  a. seq.cmd = 2 // call DO_CAPTURE command
  b. wait until seq.mode = 7 // and capture frame

Figure 39: LED Flash Timing Diagram with Automatic Exposure and White Balance



Note: Same frame size for Preview and Capture modes.

Using the Test Patterns

The MT9D131 allows predefined test images to be loaded into the beginning of the image processor, replacing the live pixel readout of the imager. This provides a static image for testing the various image processing algorithms and function blocks independent of the scene data.

The test patterns are enabled by working with registers R0x48:1 through R0x4B:1. Manipulating R0x48:1[0:2] allows for the selection of the following test patterns:

- 001—flat field; RGB values are specified in R0x49–R0x4B:1
- 010—vertical monochrome ramp
- 011—vertical color bars
- 100—vertical monochrome bars; set bar intensity in R0x49:1 and R0x4A:1
- 101—pseudo-random test pattern
- 110—horizontal monochrome bars; set bar intensity in R0x49:1 and R0x4A:1
- 111—white background

Disabling All Firmware Drivers

Although the firmware drivers may be disabled in the sequencer, their values may still automatically upload to the image processing hardware registers at the end of each frame, thus overwriting attempts to manually control the image processing hardware or the sensor's timing controls (shutter time, for example).

To disable all firmware drivers from changing the hardware registers, the microcontroller must be commanded to stop executing any firmware:

1. Set R0xC3:1[3] = 1 (puts microcontroller in "safe mode").
2. Set R0xC3:1[0] = 1 (resets current execution of firmware).

Use the following procedure to resume firmware execution (reboot):

1. Set R0xC3:1[3] = 0 (puts microcontroller in "normal mode").
2. Set R0xC3:1[0] = 0 (reboots execution of firmware).

JPEG Functionality

Enabling or Disabling the JPEG Output

JPEG output is only available in capture mode (Context B).

To enable JPEG output, set *mode.mode_config[5] = 0*.

To disable JPEG output, set *mode.mode_config[5] = 1*.

Enabling or disabling JPEG takes effect only when the MT9D131 is switched to capture mode.

Due to the instantaneous change nature of JPEG data rate, we recommend enabling the variable output clock rate when capturing JPEG compressed data by setting *mode.fifo_conf0_B[6] = 1*. This makes the output clock run at a lower frequency when output FIFO occupancy is low and at a higher frequency when output FIFO occupancy is high, avoiding FIFO overflow.

However, when capturing uncompressed data, we recommend disabling the variable output clock rate (*mode.fifo_conf0_B[6] = 0*) and setting PCLK divisor N1 (*mode.fifo_conf1_B[3:0]*) to less than or equal to the image width decimating ratio, due to the constant data rate. For example, if output image width is decimated from 1600 to 800, set *mode.fifo_conf1_B[3:0] = 2*.

Setting the JPEG Color Format

The MT9D131 supports YCbCr 4:2:2, YCbCr 4:2:0, and monochrome for JPEG. The color format is specified by *jpeg.format* (0 = YCbCr 4:2:2, 1 = YCbCr 4:2:0, 2 = monochrome).

The minimum image resolution for all color formats is 8 x 8. The maximum image width for YCbCr 4:2:0 is 384. There is no other limitation on image resolution.

Setting the Restart Marker Interval

The MT9D131 can insert restart markers into the JPEG data stream. The restart marker interval is specified by *jpeg.restartInt*. Setting this variable to 0 disables restart marker insertion.

Getting the JPEG Status

There are two ways to get JPEG status:

1. In spoof mode, the JPEG status byte is always appended to the end of the JPEG data stream. In continuous mode, the appending of JPEG status byte can be optionally enabled by setting *mode.fifo_conf0_B[9] = 1*.
2. Read R0x02:2[7:0]

Note: Care should be taken when reading R0x02:2. The JPEG status in R0x02:2 is cleared by the JPEG driver shortly after JPEG data transfer from output FIFO is completed if the next frame is going to be JPEG encoded. However, the user can set *jpeg.config[1] = 1* to keep the JPEG driver from clearing the status of an unsuccessful JPEG frame until *jpeg.config[3]* is set by the user.

Bit 7:6 of status indicates which set of quantization tables is used for the current frame JPEG encoding. Bit 7:6 = 0 means first set, 1 means second set, and 2 means third set.

Getting the JPEG Data Length

There are two ways to get JPEG data length:

1. In spoof mode, the three JPEG data length bytes (in the order of LSB to MSB) are output right before JPEG status byte at the end of the JPEG data stream.
2. Read *jpeg.dataLengthMSB* and *jpeg.dataLengLSBs*; they are bit 23:16 and bit 15:0 of the JPEG data length, respectively.

Handling JPEG Errors

If any of the JPEG error flags (bit 3:1 of JPEG status) are set, the received JPEG frame should be discarded. In single frame still JPEG capture, the MT9D131 can be configured to encode the next frame when an error occurs by setting *jpeg.config[2] = 1*. In multiple-frame still JPEG capture or video JPEG capture, it continually encodes subsequent frames until the specified number of still JPEG frames are successfully captured or video JPEG capture is terminated by a user command.

If *jpeg.config[5] = 1*, the JPEG driver automatically selects another set of quantization tables for next frame JPEG encoding when an error occurs by rolling through the three sets of quantization tables in the order of first, second, third, first, and so on. The first set of quantization tables is always used for the first frame of each JPEG capture command.

If *jpeg.config[5] = 0*, the user is responsible to specify the quantization table by setting *jpeg.config[7:6]* for every frame.

If desired, the user can enable handshaking at every erroneous JPEG frame by setting *jpeg.config[1] = 1*, which halts JPEG encoding after an erroneous JPEG frame until *jpeg.config[3]* is set to 1. This provides time for the user to handle the erroneous JPEG frame and react to the error condition, such as setting new quantization table scaling factor, loading new customized quantization tables, changing spoof frame size, and so on.

Caution This could also cause frame loss if the user does not set *jpeg.config[3]* quickly enough, or terminate JPEG capture altogether if *jpeg.config[3]* is not set within the number of frames specified in *jpeg.timeoutFrames*.

Reading or Writing to the JPEG Quantization and Huffman Table Memories

JPEG quantization and Huffman table memories can be accessed through the JPEG Indirect Access Control register R0x1E:2 and the JPEG Indirect Access Data register R0x1F:2.

Example: read quantization table 0 memory starting from address 0x80

- Set R0x1E:2 = 0x8080 (bit 15 = 1 indicates address auto-increment)
- Read R0x1F:2 repeatedly returns contents of memory address 0x80, 0x81, 0x82, …

Example: write quantization table 0 memory starting from address 0x80

- Set R0x1E:2 = 0xC080
- Write R0x1F:2 repeatedly writes to memory address 0x80, 0x81, 0x82, …

Programming the Quantization Table

The MT9D131 can store up to three sets of quantization tables; each set consists of one table for luma and one for chroma. There are two ways to program quantization tables:

- scaled version of standard quantization tables in JPEG specification
- customized quantization tables

Scaled Standard Quantization Table

To have the JPEG driver program scaled version of standard quantization tables, write the desired scaling factor to bit 6:0 of the JPEG driver variables *jpeg.qscale1*, *jpeg.qscale2*, *jpeg.qscale3*, and 1 to bit 7 of the *qscale* variable. Quantization table scaling factor can be changed any time, even during JPEG capture (the new value takes effect after the current frame JPEG encoding is finished).

Example: set scaling factor of the first set of quantization tables to 8

- Set jpeg.config[4] = 1
- Set jpeg.qscale1 = 0x88

The calculation of the scaled version of standard quantization tables is as follows:

$$scaled\_Q = (scaling\_factor * sandard\_Q + 16) >> 5 \qquad (EQ\ 6)$$

The value of scaled_Q is clipped to [1, 255]. The standard quantization tables are tables K.1 and K.2 of the ISO/IEC 10918-1 Specification (duplicated in Table 14 and Table 15 for convenience).

Table 14: Luminance Quantization

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Table 15: Chrominance Quantization

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

Customized Quantization Table

Customized quantization tables can be loaded into quantization memory through JPEG Indirect Access Control register R0x1E:2 and JPEG Indirect Access Data register R0x1F:2. The address of quantization memory is mapped to indirect registers 0x080 through 0x1FF.

Table 16: Quantization Address Map

Indirect Register Address	Quantization Table
0x080 – 0x0BF	1st set luminance quantization table (table 0)
0x0C0 – 0x0FF	1st set chrominance quantization table (table 1)
0x100 – 0x13F	2nd set luminance quantization table (table 2)
0x140 – 0x17F	2nd set chrominance quantization table (table 3)
0x180 – 0x1BF	3rd set luminance quantization table (table 4)
0x1C0 – 0x1FF	3rd set chrominance quantization table (table 5)

The 14-bit floating-point reciprocal of the 8-bit quantization table value should be loaded into quantization memory in zigzag order. The reciprocal value can be found from the following lookup table.

```
const unsigned int Quant_Reciprocal_Lookup[256] =

{

0x0000, 0x0001, 0x0400, 0x02ab, 0x0200, 0x0b33, 0x0155, 0x0a49,

0x0100, 0x09c7, 0x00cd, 0x12e9, 0x0955, 0x093b, 0x1249, 0x0911,

0x0080, 0x08f1, 0x11c7, 0x11af, 0x08cd, 0x08c3, 0x08ba, 0x08b2,

0x1155, 0x251f, 0x113b, 0x1a5f, 0x1a49, 0x1a35, 0x1111, 0x2421,

0x0880, 0x23e1, 0x10f1, 0x10ea, 0x19c7, 0x19bb, 0x19af, 0x10d2,

0x10cd, 0x231f, 0x10c3, 0x197d, 0x10ba, 0x10b6, 0x10b2, 0x22b9,

0x1955, 0x229d, 0x10a4, 0x2d05, 0x193b, 0x1935, 0x225f, 0x1095,

0x2249, 0x223f, 0x2235, 0x2c57, 0x1911, 0x2219, 0x1084, 0x1082,

0x1080, 0x18fc, 0x18f8, 0x21e9, 0x18f1, 0x21db, 0x18ea, 0x2b9b,

0x21c7, 0x21c1, 0x21bb, 0x21b5, 0x21af, 0x2b53, 0x18d2, 0x367b,

0x18cd, 0x2b29, 0x18c8, 0x218b, 0x18c3, 0x2b03, 0x217d, 0x2af1,

0x18ba, 0x18b8, 0x18b6, 0x18b4, 0x18b2, 0x2ac1, 0x2ab9, 0x2159,

0x2155, 0x3547, 0x2a9d, 0x214b, 0x18a4, 0x2a89, 0x2141, 0x189f,

0x213b, 0x189c, 0x2135, 0x34c9, 0x2a5f, 0x2a59, 0x1895, 0x349d,

0x2a49, 0x1891, 0x2a3f, 0x211d, 0x2a35, 0x188c, 0x2a2b, 0x344d,

0x2111, 0x343b, 0x2a19, 0x2a15, 0x1884, 0x3419, 0x1882, 0x1881,

0x1880, 0x20fe, 0x20fc, 0x33e9, 0x20f8, 0x3fb3, 0x29e9, 0x33cb,

0x20f1, 0x33bd, 0x29db, 0x33af, 0x20ea, 0x29d1, 0x20e7, 0x3395,

0x29c7, 0x20e2, 0x29c1, 0x20df, 0x29bb, 0x20dc, 0x29b5, 0x20d9,

0x29af, 0x3359, 0x3353, 0x29a7, 0x20d2, 0x3343, 0x299f, 0x20ce,
```

0x20cd, 0x2997, 0x3329, 0x20c9, 0x20c8, 0x298d, 0x298b, 0x3311,

0x20c3, 0x3e0f, 0x3303, 0x3dfd, 0x297d, 0x297b, 0x2979, 0x32ed,

0x20ba, 0x3dc9, 0x20b8, 0x20b7, 0x20b6, 0x20b5, 0x20b4, 0x20b3,

0x20b2, 0x3d89, 0x20b0, 0x32bd, 0x295d, 0x3d6b, 0x2959, 0x2957,

0x2955, 0x32a7, 0x20a9, 0x20a8, 0x20a7, 0x3299, 0x294b, 0x3293,

0x20a4, 0x20a3, 0x3289, 0x2943, 0x2941, 0x3cff, 0x209f, 0x3279,

0x293b, 0x3273, 0x209c, 0x326d, 0x2935, 0x3ccf, 0x2099, 0x3cc3,

0x325f, 0x2097, 0x3259, 0x3cad, 0x2095, 0x3251, 0x2927, 0x2093,

0x3249, 0x3c8d, 0x2091, 0x3c83, 0x323f, 0x3c79, 0x291d, 0x3c6f,

0x3235, 0x3c65, 0x208c, 0x2917, 0x208b, 0x3229, 0x3227, 0x3c49,

0x2911, 0x2088, 0x290f, 0x3c37, 0x3219, 0x3217, 0x3215, 0x3c25,

0x2084, 0x3c1d, 0x2083, 0x2905, 0x2082, 0x2903, 0x2081, 0x2901

};

Translating Between Qscale and Quality Factor

Quantization tables are computed as follows:

```
Qtable  =  (standard_qtable * qscale + 16) / 32;
```

In the ubiquitous JPEG quality factor case:

```
if (quality < 50)
    quality  =  5000/quality;
else
    quality  =  200 - 2*quality;
Qtable  =  (standard_qtable * quality + 50) / 100;
```

So from there, we can derive:

```
if (quality < 50)
    qscale  =  1600 / quality;
else
    qscale  =  16 * (100 - quality) / 25;
```

Since qscale is 7-bit, the maximum value is 127, which implies the minimum quality is 13 (very poor quality—such a low setting should never be necessary).

The quality factor for the default qscale is as follows:

- qscale1 = 6 --> quality = 90.625 (or 91)
- qscale2 = 9 --> quality = 85.937 (or 86)
- qscale3 = 12 --> quality = 81.25 (or 81)

Programming the Customized Huffman Table

The MT9D131 can store two sets of Huffman DC and AC tables (one each for luma and chroma). The standard Huffman tables K.3 and K.4 in the ISO/IEC 10918-1 Specification are programmed into Huffman memory by the JPEG driver. These standard Huffman tables can be overwritten by customized Huffman tables through the JPEG Indirect Access Control Register R0x1E:2 and the JPEG Indirect Access Data Register R0x1F:2.

Each AC table contains 176 12-bit words; each DC table contains 16 12-bit words. The memory map of the Huffman memory is shown in Table 17.

Table 17: Huffman Memory Map

First Address	Last Address	Table
0	175	AC Huffman Table 0
176	351	AC Huffman Table 1
352	367	DC Huffman Table 0
368	383	DC Huffman Table 1

Each Huffman code is stored as a record containing the actual code and its length, as shown in the Table 18.

Table 18: Structure of Huffman Code in Huffman Memory

11	10	9	8	7	6	5	4	3	2	1	0
HLEN	HCODE										

HLEN is the number of bits in the Huffman code—HCODE—minus 1.

HCODE are the 8 least-significant bits of the Huffman code. If the Huffman code is less than 8 bits long, the bits not used must be 0.

Although Huffman codes used in the JPEG encoding can be up to 16 bits long, when the code is more than 8 bits long, the most significant bits are always 1. Thus it is unnecessary to specify more than 8 bits for any code, as the most significant bits are generated internally within the MT9D131.

In the case of the JPEG baseline algorithm, 162 Huffman codes are required for the encoding of the AC run-length codes and 12 Huffman codes are required for the encoding of the DC coefficients. The location of the Huffman codes for the 162 run length codes in an AC table is shown in Table 19.

Table 19: Location of AC Huffman Codes in Huffman Memory

Address	Value
0 – 9	Huffman code of run lengths 0/1 to 0/A
10 – 19	Huffman code of run lengths 1/1 to 1/A
20 – 29	Huffman code of run lengths 2/1 to 2/A
30 – 39	Huffman code of run lengths 3/1 to 3/A
40 – 49	Huffman code of run lengths 4/1 to 4/A
50 – 59	Huffman code of run lengths 5/1 to 5/A
60 – 69	Huffman code of run lengths 6/1 to 6/A
70 – 79	Huffman code of run lengths 7/1 to 7/A

Table 19: Location of AC Huffman Codes in Huffman Memory (continued)

Address	Value
80 – 89	Huffman code of run lengths 8/1 to 8/A
90 – 99	Huffman code of run lengths 9/1 to 9/A
100 – 109	Huffman code of run lengths A/1 to A/A
110 – 119	Huffman code of run lengths B/1 to B/A
120 – 129	Huffman code of run lengths C/1 to C/A
130 – 139	Huffman code of run lengths D/1 to D/A
140 – 149	Huffman code of run lengths E/1 to E/A
150 – 159	Huffman code of run lengths F/1 to F/A
160	Huffman code of EOB
161	Huffman code of ZRL
162 – 167	0xFFF
168 – 175	0xFD0 – 0xFD7

The location of the Huffman codes for the 12 codes in each DC table is shown in Table 20.

Table 20: Location of DC Huffman Codes in Huffman Memory

Address	Value
0 – 11	Huffman code of DC codes 0 to A
12 – 15	Not used

The address of Huffman memory 0 through 383 is mapped to indirect registers 0x200 through 0x37F.

Appending the JPEG Header

The MT9D131 outputs the entropy-coded data segments of JPEG data without a header. The user is responsible for adding all JPEG markers except the restart marker. To generate a complete JPEG file in JFIF format, the following JPEG header markers must be added to the beginning of JPEG data:

- Start of image
- JFIF APP0
- Define quantization table
- Start of frame
- Define Huffman table
- Define restart interval
- Start of scan

For the details of JPEG header markers, see the following section on Sample C Code. At the end of JPEG data, an end of image marker (0xFFD9) should be appended.

Sample C Code

The following sample C code is Micron Proprietary and Confidential.

```
DO NOT DISTRIBUTE. All or part of the code subject to change
©2005, Micron Technology, Inc. All rights reserved.


#define FORMAT_YCBCR4220
#define FORMAT_YCBCR4201
#define FORMAT_MONOCHROME2

// JEPG tables
unsigned char JPEG_StdQuantTblY_ZZ[64]  =
{
  16,  11,  12,  14,  12,  10,  16,  14,
  13,  14,  18,  17,  16,  19,  24,  40,
  26,  24,  22,  22,  24,  49,  35,  37,
  29,  40,  58,  51,  61,  60,  57,  51,
  56,  55,  64,  72,  92,  78,  64,  68,
  87,  69,  55,  56,  80, 109,  81,  87,
  95,  98, 103, 104, 103,  62,  77, 113,
   121, 112, 100, 120,  92, 101, 103,  99
};

unsigned char JPEG_StdQuantTblC_ZZ[64]  =
{
  17,  18,  18,  24,  21,  24,  47,  26,
  26,  47,  99,  66,  56,  66,  99,  99,
  99,  99,  99,  99,  99,  99,  99,  99,
  99,  99,  99,  99,  99,  99,  99,  99,
  99,  99,  99,  99,  99,  99,  99,  99,
  99,  99,  99,  99,  99,  99,  99,  99,
  99,  99,  99,  99,  99,  99,  99,  99,
  99,  99,  99,  99,  99,  99,  99,  99
};

unsigned int JPEG_StdHuffmanTbl[384]  =
{
  0x100, 0x101, 0x204, 0x30b, 0x41a, 0x678, 0x7f8, 0x9f6,
  0xf82, 0xf83, 0x30c, 0x41b, 0x679, 0x8f6, 0xaf6, 0xf84,
  0xf85, 0xf86, 0xf87, 0xf88, 0x41c, 0x7f9, 0x9f7, 0xbf4,
  0xf89, 0xf8a, 0xf8b, 0xf8c, 0xf8d, 0xf8e, 0x53a, 0x8f7,
  0xbf5, 0xf8f, 0xf90, 0xf91, 0xf92, 0xf93, 0xf94, 0xf95,
    0x53b, 0x9f8, 0xf96, 0xf97, 0xf98, 0xf99, 0xf9a, 0xf9b,
    0xf9c, 0xf9d, 0x67a, 0xaf7, 0xf9e, 0xf9f, 0xfa0, 0xfa1,
    0xfa2, 0xfa3, 0xfa4, 0xfa5, 0x67b, 0xbf6, 0xfa6, 0xfa7,
    0xfa8, 0xfa9, 0xfaa, 0xfab, 0xfac, 0xfad, 0x7fa, 0xbf7,
    0xfae, 0xfaf, 0xfb0, 0xfb1, 0xfb2, 0xfb3, 0xfb4, 0xfb5,
    0x8f8, 0xec0, 0xfb6, 0xfb7, 0xfb8, 0xfb9, 0xfba, 0xfbb,
    0xfbc, 0xfbd, 0x8f9, 0xfbe, 0xfbf, 0xfc0, 0xfc1, 0xfc2,
    0xfc3, 0xfc4, 0xfc5, 0xfc6, 0x8fa, 0xfc7, 0xfc8, 0xfc9,
    0xfca, 0xfcb, 0xfcc, 0xfcd, 0xfce, 0xfcf, 0x9f9, 0xfd0,
    0xfd1, 0xfd2, 0xfd3, 0xfd4, 0xfd5, 0xfd6, 0xfd7, 0xfd8,
    0x9fa, 0xfd9, 0xfda, 0xfdb, 0xfdc, 0xfdd, 0xfde, 0xfdf,
    0xfe0, 0xfe1, 0xaf8, 0xfe2, 0xfe3, 0xfe4, 0xfe5, 0xfe6,
    0xfe7, 0xfe8, 0xfe9, 0xfea, 0xfeb, 0xfec, 0xfed, 0xfee,
    0xfef, 0xff0, 0xff1, 0xff2, 0xff3, 0xff4, 0xff5, 0xff6,
    0xff7, 0xff8, 0xff9, 0xffa, 0xffb, 0xffc, 0xffd, 0xffe,
    0x30a, 0xaf9, 0xfff, 0xfff, 0xfff, 0xfff, 0xfff, 0xfff,
    0xfd0, 0xfd1, 0xfd2, 0xfd3, 0xfd4, 0xfd5, 0xfd6, 0xfd7,
    0x101, 0x204, 0x30a, 0x418, 0x419, 0x538, 0x678, 0x8f4,
    0x9f6, 0xbf4, 0x30b, 0x539, 0x7f6, 0x8f5, 0xaf6, 0xbf5,
    0xf88, 0xf89, 0xf8a, 0xf8b, 0x41a, 0x7f7, 0x9f7, 0xbf6,
    0xec2, 0xf8c, 0xf8d, 0xf8e, 0xf8f, 0xf90, 0x41b, 0x7f8,
    0x9f8, 0xbf7, 0xf91, 0xf92, 0xf93, 0xf94, 0xf95, 0xf96,
    0x53a, 0x8f6, 0xf97, 0xf98, 0xf99, 0xf9a, 0xf9b, 0xf9c,
    0xf9d, 0xf9e, 0x53b, 0x9f9, 0xf9f, 0xfa0, 0xfa1, 0xfa2,
    0xfa3, 0xfa4, 0xfa5, 0xfa6, 0x679, 0xaf7, 0xfa7, 0xfa8,
    0xfa9, 0xfaa, 0xfab, 0xfac, 0xfad, 0xfae, 0x67a, 0xaf8,
    0xfaf, 0xfb0, 0xfb1, 0xfb2, 0xfb3, 0xfb4, 0xfb5, 0xfb6,
    0x7f9, 0xfb7, 0xfb8, 0xfb9, 0xfba, 0xfbb, 0xfbc, 0xfbd,
    0xfbe, 0xfbf, 0x8f7, 0xfc0, 0xfc1, 0xfc2, 0xfc3, 0xfc4,
    0xfc5, 0xfc6, 0xfc7, 0xfc8, 0x8f8, 0xfc9, 0xfca, 0xfcb,
    0xfcc, 0xfcd, 0xfce, 0xfcf, 0xfd0, 0xfd1, 0x8f9, 0xfd2,
    0xfd3, 0xfd4, 0xfd5, 0xfd6, 0xfd7, 0xfd8, 0xfd9, 0xfda,
    0x8fa, 0xfdb, 0xfdc, 0xfdd, 0xfde, 0xfdf, 0xfe0, 0xfe1,
    0xfe2, 0xfe3, 0xaf9, 0xfe4, 0xfe5, 0xfe6, 0xfe7, 0xfe8,
    0xfe9, 0xfea, 0xfeb, 0xfec, 0xde0, 0xfed, 0xfee, 0xfef,
    0xff0, 0xff1, 0xff2, 0xff3, 0xff4, 0xff5, 0xec3, 0xff6,
    0xff7, 0xff8, 0xff9, 0xffa, 0xffb, 0xffc, 0xffd, 0xffe,
    0x100, 0x9fa, 0xfff, 0xfff, 0xfff, 0xfff, 0xfff, 0xfff,
    0xfd0, 0xfd1, 0xfd2, 0xfd3, 0xfd4, 0xfd5, 0xfd6, 0xfd7,
    0x100, 0x202, 0x203, 0x204, 0x205, 0x206, 0x30e, 0x41e,
    0x53e, 0x67e, 0x7fe, 0x8fe, 0xfff, 0xfff, 0xfff, 0xfff,
    0x100, 0x101, 0x102, 0x206, 0x30e, 0x41e, 0x53e, 0x67e,
    0x7fe, 0x8fe, 0x9fe, 0xafe, 0xfff, 0xfff, 0xfff, 0xfff
};

int JfifApp0Marker(char *pbuf)
{
    *pbuf++  =  0xFF;// APP0 marker
    *pbuf++  =  0xE0;
    *pbuf++  =  0x00;// length
    *pbuf++  =  0x10;
    *pbuf++  =  0x4A;// JFIF identifier
    *pbuf++  =  0x46;
    *pbuf++  =  0x49;
    *pbuf++  =  0x46;
    *pbuf++  =  0x00;
    *pbuf++  =  0x01;// version
    *pbuf++  =  0x02;
    *pbuf++  =  0x00;// units
    *pbuf++  =  0x00;// X density
    *pbuf++  =  0x01;
    *pbuf++  =  0x00;// Y density
    *pbuf++  =  0x01;
    *pbuf++  =  0x00;// X thumbnail
    *pbuf++  =  0x00;// Y thumbnail

    return 18;
  }

  int FrameHeaderMarker(char *pbuf, int width, int height, int format)
  {
    int length;

    if (format  =  =  FORMAT_MONOCHROME)
      length  =  11;
    else
      length  =  17;

    *pbuf++  =  0xFF;// start of frame: baseline DCT
    *pbuf++  =  0xC0;
    *pbuf++  =  length>>8;// length field
    *pbuf++  =  length&0xFF;
    *pbuf++  =  0x08;// sample precision
    *pbuf++  =  height>>8;// number of lines
    *pbuf++  =  height&0xFF;
    *pbuf++  =  width>>8;// number of samples per line
    *pbuf++  =  width&0xFF;

    if (format  =  =  FORMAT_MONOCHROME)// monochrome
    {
      *pbuf++  =  0x01;// number of image components in frame
      *pbuf++  =  0x00;// component identifier: Y
      *pbuf++  =  0x11;// horizontal | vertical sampling factor: Y
      *pbuf++  =  0x00;// quantization table selector: Y
    }
    else if (format  =  =  FORMAT_YCBCR422) // YCbCr422
    {
      *pbuf++  =  0x03;// number of image components in frame
      *pbuf++  =  0x00;// component identifier: Y
      *pbuf++  =  0x21;// horizontal | vertical sampling factor: Y
      *pbuf++  =  0x00;// quantization table selector: Y
      *pbuf++  =  0x01;// component identifier: Cb
      *pbuf++  =  0x11;// horizontal | vertical sampling factor: Cb
      *pbuf++  =  0x01;// quantization table selector: Cb
      *pbuf++  =  0x02;// component identifier: Cr
      *pbuf++  =  0x11;// horizontal | vertical sampling factor: Cr
      *pbuf++  =  0x01;// quantization table selector: Cr
    }
    else// YCbCr420
    {
      *pbuf++  =  0x03;// number of image components in frame
      *pbuf++  =  0x00;// component identifier: Y
      *pbuf++  =  0x22;// horizontal | vertical sampling factor: Y
      *pbuf++  =  0x00;// quantization table selector: Y
      *pbuf++  =  0x01;// component identifier: Cb
      *pbuf++  =  0x11;// horizontal | vertical sampling factor: Cb
      *pbuf++  =  0x01;// quantization table selector: Cb
      *pbuf++  =  0x02;// component identifier: Cr
      *pbuf++  =  0x11;// horizontal | vertical sampling factor: Cr
      *pbuf++  =  0x01;// quantization table selector: Cr
    }

    return (length+2);
  }

  int ScanHeaderMarker(char *pbuf, int format)
```

```
{
  int length;

  if (format  =  =  FORMAT_MONOCHROME)
    length  =  8;
  else
    length  =  12;

  *pbuf++  =  0xFF;// start of scan
  *pbuf++  =  0xDA;
  *pbuf++  =  length>>8;// length field
  *pbuf++  =  length&0xFF;

  if (format  =  =  FORMAT_MONOCHROME)// monochrome
  {
    *pbuf++  =  0x01;// number of image components in scan
    *pbuf++  =  0x00;// scan component selector: Y
    *pbuf++  =  0x00;// DC | AC huffman table selector: Y
  }
  else// YCbCr
  {
    *pbuf++  =  0x03;// number of image components in scan
    *pbuf++  =  0x00;// scan component selector: Y
    *pbuf++  =  0x00;// DC | AC huffman table selector: Y
    *pbuf++  =  0x01;// scan component selector: Cb
    *pbuf++  =  0x11;// DC | AC huffman table selector: Cb
    *pbuf++  =  0x02;// scan component selector: Cr
    *pbuf++  =  0x11;// DC | AC huffman table selector: Cr
  }
  *pbuf++  =  0x00;// Ss: start of predictor selector
  *pbuf++  =  0x3F;// Se: end of spectral selector
  *pbuf++  =  0x00;// Ah | Al: successive approximation bit position

  return (length+2);
}

int DefineQuantizationTableMarker(char *pbuf, int qscale, int format)
{
  int i, q, length;

  if (format  =  =  FORMAT_MONOCHROME)// monochrome
    length  =  67;
  else
    length  =  132;

  *pbuf++  =  0xFF;// define quantization table marker
  *pbuf++  =  0xDB;
  *pbuf++  =  length>>8;// length field
  *pbuf++  =  length&0xFF;

  *pbuf++  =  0;// quantization table precision | identifier
  for (i  =  0; i < 64; i++)
  {
    q  =  (JPEG_StdQuantTblY_ZZ[i] * qscale + 16) >> 5;
    *pbuf++  =  q;// quantization table element
  }

  if (format ! =  FORMAT_MONOCHROME)
  {
    *pbuf++  =  1;// quantization table precision | identifier
    for (i  =  0; i < 64; i++)
    {
      q  =  (JPEG_StdQuantTblC_ZZ[i] * qscale + 16) >> 5;
```

```
        *pbuf++  =  q;// quantization table element
      }
    }

    return (length+2);
  }

  int DefineHuffmanTableMarkerDC(char *pbuf, unsigned int *htable, int class_id)
  {
    int i, l, count;
    char *plength;
    int length;

    *pbuf++  =  0xFF;// define huffman table marker
    *pbuf++  =  0xC4;
    plength  =  pbuf;// place holder for length field
    *pbuf++;
    *pbuf++;
    *pbuf++  =  class_id;// huffman table class | identifier
    for (l  =  0; l < 16; l++)
    {
      count  =  0;
      for (i  =  0; i < 12; i++)
      {
        if ((htable[i] >> 8)  =  =  l)
          count++;
      }
      *pbuf++  =  count;// number of huffman codes of length l+1
    }
    length  =  19;
    for (l  =  0; l < 16; l++)
    {
      for (i  =  0; i < 12; i++)
      {
        if ((htable[i] >> 8)  =  =  l)
        {
          *pbuf++  =  i;// HUFFVAL with huffman codes of length l+1
          length++;
        }
      }
    }
    *plength++  =  length>>8;// length field
    *plength  =  length&0xFF;

    return (length + 2);
  }

  int DefineHuffmanTableMarkerAC(char *pbuf, unsigned int *htable, int class_id)
  {
    int i, l, a, b, count;
    char *plength;
    int length;
    int eob  =  1;
    int zrl  =  1;

    *pbuf++  =  0xFF;// define huffman table marker
    *pbuf++  =  0xC4;
    plength  =  pbuf;// place holder for length field
    *pbuf++;
    *pbuf++;
    *pbuf++  =  class_id;// huffman table class | identifier
    for (l  =  0; l < 16; l++)
    {
      count  =  0;
      for (i  =  0; i < 162; i++)
      {
        if ((htable[i] >> 8)  =  =  l)
          count++;
      }
      *pbuf++  =  count;// number of huffman codes of length l+1
    }
    length  =  19;
    for (l  =  0; l < 16; l++)
    {
      // check EOB: 0|0
      if ((htable[160] >> 8)  =  =  l)
      {
        *pbuf++  =  0;// HUFFVAL with huffman codes of length l+1
        length++;
      }
      // check HUFFVAL: 0|1 to E|A
      for (i  =  0; i < 150; i++)
      {
        if ((htable[i] >> 8)  =  =  l)
        {
          a  =  i/10;
          b  =  i%10;
          *pbuf++  =  (a<<4)|(b+1);// HUFFVAL with huffman codes of length l+1
          length++;
        }
      }
      // check ZRL: F|0
      if ((htable[161] >> 8)  =  =  l)
      {
        *pbuf++  =  0xF0;// HUFFVAL with huffman codes of length l+1
        length++;
      }
      // check HUFFVAL: F|1 to F|A
      for (i  =  150; i < 160; i++)
      {
        if ((htable[i] >> 8)  =  =  l)
        {
          a  =  i/10;
          b  =  i%10;
          *pbuf++  =  (a<<4)|(b+1);// HUFFVAL with huffman codes of length l+1
          length++;
        }
      }
    }
    *plength++  =  length>>8;// length field
    *plength  =  length&0xFF;

    return (length + 2);
  }

  int DefineRestartIntervalMarker(char *pbuf, int ri)
  {
    *pbuf++  =  0xFF;// define restart interval marker
    *pbuf++  =  0xDD;
    *pbuf++  =  0x00;// length
    *pbuf++  =  0x04;
    *pbuf++  =  ri >> 8;// restart interval
    *pbuf++  =  ri & 0xFF;

    return 6;
  }
```

```
/*
  Purpose: Create JPEG header in JFIF format
  Parameters:header - pointer to JPEG header buffer
         width - image width
         height - image height
         format - color format (0  =  YCbCr422, 1  =  YCbCr420, 2  =  monochrome)
         restart_int - restart marker interval
         qscale - quantization table scaling factor
  Return: length of JPEG header (bytes)
*/
int CreateJpegHeader(char *header, int width, int height, int format, int restart_int, int
qscale)
{
  char *pbuf  =  header;
  int length;

  // SOI
  *pbuf++  =  0xFF;
  *pbuf++  =  0xD8;
  length  =  2;

  // JFIF APP0
  length + =  JfifApp0Marker(pbuf);

  // Quantization Tables
  pbuf  =  header + length;
  length + =  DefineQuantizationTableMarker(pbuf, qscale, format);

  // Frame Header
  pbuf  =  header + length;
  length + =  FrameHeaderMarker(pbuf, width, height, format);

  // Huffman Table DC 0 for Luma
  pbuf  =  header + length;
  length + =  DefineHuffmanTableMarkerDC(pbuf, &JPEG_StdHuffmanTbl[352], 0x00);

  // Huffman Table AC 0 for Luma
  pbuf  =  header + length;
  length + =  DefineHuffmanTableMarkerAC(pbuf, &JPEG_StdHuffmanTbl[0], 0x10);

  if (format ! =  FORMAT_MONOCHROME)// YCbCr
  {
    // Huffman Table DC 1 for Chroma
    pbuf  =  header + length;
    length + =  DefineHuffmanTableMarkerDC(pbuf, &JPEG_StdHuffmanTbl[368], 0x01);

    // Huffman Table AC 1 for Chroma
    pbuf  =  header + length;
    length + =  DefineHuffmanTableMarkerAC(pbuf, &JPEG_StdHuffmanTbl[176], 0x11);
  }

  // Restart Interval
  if (restart_int > 0)
  {
    pbuf  =  header + length;
    length + =  DefineRestartIntervalMarker(pbuf, restart_int);
  }

  // Scan Header
  pbuf  =  header + length;
  length + =  ScanHeaderMarker(pbuf, format);
  return length;
}
```

JPEG Power Saving

Turning off JPEG clock Reg 11:1[4] during preview mode will reduce about 25 percent of current draw from VDD (50mA -> 38mA) and power on VDD is about 50 percent of the total power consumption for the MT9D131. Manually change this bit between preview/capture context switches. Turn on the clock just before DO_CAPTURE, then turn off the clock once the sequencer state is back to Preview or PreviewEnter.

Appendix A—Frequently Asked Questions

Initializing FAQs

1. Is there any other way to make entering Standby Mode faster?
2. If the PLL settings must change during operation, which registers and which order should we set? Could you show me sequence flow charts of context switch with PLL setting changes?

Context Switching and Output Configuration

3. What are the recommended settings if we want to share 1 ADC in contexts A and B because brightness changes when mode change?
4. Is it possible to have the same pclk in Context A and Context B?
5. Is it possible to use only 1 ADC in context B? How do we disable the default JPEG setting to use YUV4:2:2?
6. Is it required to write seq.cmd = 1(Do preview) but only seq.cmd = 5(Refresh) when the camera starts up?
7. Is there seq.cmd writing timing limitation? In our evaluation when I wrote seq.cmd = 5 after seq.cmd = 1, the setting of seq.cmd = 5 was not reflected.

Gamma and Contrast

8. How are brightness, contrast, saturation, sharpening, and color tones adjusted?
9. Can contrast be adjusted by the predefined S-curves?
10. How are color tones set?
11. Does aperture correction mean sharpening? Is it programmable, so that the desired amount of sharpening can be done?
12. Is it possible increase or decrease the color saturation?
13. The sensor output does not correspond to normal Bayer output when binning is used. Do you use different Bayer interpolation algorithm for binned and full output modes?

Lens Shading and Correction

14. Where are the lens shading correction control registers located?
15. Can the lens shading calibration be used for AF calibration?

Auto Exposure

16. To implement EV shift, the target exposure must be changed, is there a register to set for over or under exposure?
17. Can we use single AE measurement window, whose size could be freely programmed?
18. Manual exposure time & ISO setting. Is it possible to override the auto exposure and set the integration time and analog gain manually?
19. Can we lock integration time and analog gain to their current values? Can we read the values of the integration time and analog gain from the sensor and set them manually afterwards?

Flicker Avoidance

20. What are the instructions to enable flicker detection (and presumably suppression) for the rev 1 MT9D131 samples?

Color Correction

21. How can I load a new color correction matrix?

Mode Driver Preview and Driver

22. What is the maximum frame rate that can be achieved if I use 65 MHz clock?

Flash Strobe, Mechanical Shutter, and Global Reset

23. For flash LED applications, what is the proper sequence for the flash light to be turned on before a frame is captured?
24. We are considering using the CMOS sensor to detect the ambient light for the purpose of auto-flash application. Can you tell me which register contains the luminance value from the sensor? How do we convert the luminance to lux and what is the accuracy?

JPEG Functionality

25. Do I have to capture JPEGs in 422 Mode for a full-resolution image?
26. MT9D131 spec does not provide information about the status byte that is at the end of each LINE_VALID stream in SPOOF mode. Please provide a description of the field.
27. From MT9D131 Spec: “The timing of PIXCLK and Dout within each LINE_VALID assertion period is variable and therefore unlike that of uncompressed data”. Why is the timing different? Spoof mode is supposed to emulate the uncompressed data, which will require no changes on PIXCLK or Dout behavior, right? I was expecting that filler bytes will be placed after JPEG bytes.
28. What are the specific recommendations that map the most popular SNAPSHOT resolutions with a preferred programming for the Spoof Frame Width and Spoof Frame Height (IFP Register's Page2 Reg 0x10 and 0x11)? Does Micron recommend a FIXED Spoof Frame Width/Height regardless of resolution?
29. Does the JPEG data length include the 4 bytes used for J0, J1, J2, and status bytes at the end of each LINE_VALID sync?
30. Does the JPEG compression algorithm support compressed 565 output?
31. How does the use of the JPEG engine affect the frame rate? Use of JPEG should reduce the amount of data transmitted, but does it affect the frame rate?

Miscellaneous FAQs

32. Fixed Pattern Noise (FPN) appears in Preview mode in low light conditions when 2 ADC are used in Context A. How do we reduce the FPN?
33. When should REFRESH be used? When not? What does REFRESH do? When does REFRESH start to affect the system, after how many frames, etc.? What is the difference between REFRESH and REFRESH_MODE?
34. When can the registers be written? When not?
35. Is there a need for a delay after the REFRESH command?
36. Does the MT9D131 contain flash memory? Can the user access the MCU on the chip for general computing?
37. What does “PRNU” stand for?
38. What is ^{t}EXTCLK? Is this the period of master clock?
39. Is it possible to have an ISO setting in the MT9D131?
40. How do I change the PLL setting after the firmware is running?
41. Why does random noise appear in low light conditions?
42. How do you adjust several functions (AE target, AWB parameter) after power-on and initialization? I think that it is not possible to set the parameter while the firmware is running...
43. Why does random noise appear in low light conditions?
44. Do I have to wait after a REFRESH command?
45. Can VDDPLL be left unconnected if PLL is not used (by default, we knew the PLL is bypassed and powered down)?
46. Are the registers REG0x61:0, REG0x62:0, REG0x63:0, and REG0x64:0 related to a row noise? When row noise occurs, these register values seem to change a lot.

47. According to the specifications sheet, I have to issue the REFRESH command (seq.cmd = 5) when changing the image resolution, effect, gamma and so on. In this timing, it seems that hardware registers were changed by drivers. See the following example for an explanation: First, I changed the slew rate control register in R0x0E:2 and R0x0F:2 (hardware registers), but in this timing, I do not change the slew rate control variables in the mode driver(ID = 7). Next, I issued the REFRESH command to change the image resolution. After issuing the command, the hardware registers were changed by the drivers. I think that there are many registers like this. I would like to know which registers were changed by drivers when I issued the seq.cmd = 5. I cannot understand how to set the register correctly without this information.
48. How should we treat the STANDBY terminal? Should it be connected to GND or open?

Initializing FAQs

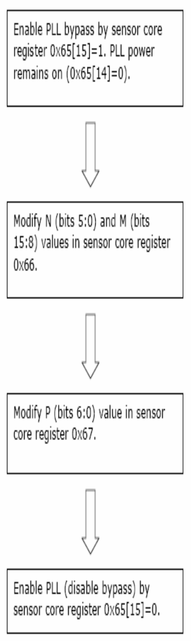
1. Is there any other way to make entering Standby Mode faster?

We recommend that you follow the standby sequence we have outlined in the document because the MCU needs to prepare for standby (seq.cmd = 3), then the sensor acknowledges standby (seq.state = 9) and then asserts STANDBY. To make the camera enter standby mode quicker, you can set the PLL to run at maximum frequency (80 MHz).

2. If the PLL settings must change during operation, which registers and which order should we set? Could you show me sequence flow charts of context switch with PLL setting changes?

Use sensor core registers 0x65, 0x66, 0x67 to change PLL settings. For the order of settings, see the flow chart in Figure 40.

Figure 40: PLL Setting Change Flow Chart


Enable PLL bypass by sensor core register 0x65[15]=1. PLL power remains on (0x65[14]=0).
Modify N (bits 5:0) and M (bits 15:8) values in sensor core register 0x66.
Modify P (bits 6:0) value in sensor core register 0x67.
Enable PLL (disable bypass) by sensor core register 0x65[15]=0.

Context Switching and Output Configuration

3. What are the recommended settings if we want to share 1 ADC in contexts A and B because brightness changes when mode change?

When context A and context B share 1 ADC, it is best to enable the AE driver in the captureEnter state. One may be trying to minimize the state transition time by skipping previewLeave and captureEnter states. However, there is a trade-off here between quality and time. Therefore, we recommend having the following settings:

- *VAR8 = 1, 0x37, 0x0001 // SEQ_PREVIEW_3_AE (fast AE in captureEnter state)*
- *VAR8 = 1, 0x3D, 0x0000 // SEQ_PREVIEW_3_SKIPFRAME (no skipping captureEnter state)*

The previewLeave state may be skipped, but in order to have same brightness in context A and B, captureEnter state should not be skipped.

4. Is it possible to have the same pclk in Context A and Context B?

Yes, it is possible to have the same PCLK frequency in context A and context B. If this setup is used, both contexts will need to have the same number of ADC. Horizontal blanking will also need adjustments otherwise column FPN might appear.

5. Is it possible to use only 1 ADC in context B? How do we disable the default JPEG setting to use YUV4:2:2?

It is possible to use only 1 ADC in context B. Rev2 and Rev3 have JPEG as the default setting for context B. To disable JPEG and use YUV422, set the following settings:

- *Mode.mode_config: ID = 7, Offset = 11, bit 5 = 1 // bypass JPEG*
- *Mode.fifo_conf1_B: ID = 7, Offset = 116, bit[3:0] = 1 // N1*

6. Is it required to write seq.cmd = 1(Do preview) but only *seq.cmd = 5(Refresh)* when the camera starts up?

At startup, the MT9D131 will automatically go into preview mode after initialization. Hence, no *seq.cmd = 1 (go to preview)* or *seq.cmd = 5 (refresh)* is needed. However, if settings for context A is changed (such as image size), then the following command is needed:

- if firmware settings for context A is changed while the current context is A, then seq.cmd = 5 (refresh is needed).
- if firmware settings for context A is changed while the current context is B, then seq.cmd = 1 (go to preview) will go to context A and automatically refresh (no seq.cmd = 5 is needed).

For example, to set an image size of 160 x 120 (with RGB format) while in context A, the following sequence is needed:

1. VAR = 7, 0x03, 0x00A0 // MODE_OUTPUT_WIDTH_A
2. VAR = 7, 0x05, 0x0078 // MODE_OUTPUT_HEIGHT_A
3. VAR8 = 7, 0x7D, 0x0020 // MODE_OUTPUT_FORMAT_A
4. VAR8 = 1, 0x03, 0x0005 // SEQ_CMD (refresh in the new settings above)

Crop settings are used for zooming and panning purposes. They are not needed if you simply want to resize the original image.

7. Is there seq.cmd writing timing limitation? In our evaluation when I wrote seq.cmd = 5 after seq.cmd = 1, the setting of seq.cmd = 5 was not reflected.

Setting seq.cmd = 1 automatically refreshes for the new settings, hence seq.cmd = 5 is not needed in this case. The context switch will automatically check for new settings. In terms of timing, seq.state variable (which is a status variable located in ID = 1, offset = 4) should be monitored (polled) for the status of the firmware.

For example, if the user wants to switch from context A to B, and back to A immediately, then one should follow the timing:

(while in preview: seq.state = 3)

issue seq.cmd = 2 (go to context B)

poll seq.state, make sure it is equal to 7 before issuing seq.cmd = 1 (go back to context A)

The above is just a sample, in most cases, the user will not switch back and forth from A to B immediately. While the firmware switches from context A to B, the seq.state value will travel from 3 to 7. If a command to "go back to context A" is called before seq.state = 7, it will not work because the current state is not context B yet. Note that the time required to switch from A to B is small (several frames) and depends on the clock frequency.

Gamma and Contrast

8. How are brightness, contrast, saturation, sharpening, and color tones adjusted?

There are different methods of changing brightness of the image—a luminance offset can be applied at the output or the exposure target can be changed from the default value.

LCD display brightness should be changed by programming luma offset. Snapshot/picture brightness should be changed using ae.target. Changing gamma to change brightness is not recommended—just set the gamma to match your display device.

9. Can contrast be adjusted by the predefined S-curves?

Yes.

10. How are color tones set?

Color tones such as sepia, black and white, and negative are programmable using the variables *mode.spec.effects_A* and *mode.spec.effects_B*.

11. Does aperture correction mean sharpening? Is it programmable, so that the desired amount of sharpening can be done?

Yes, aperture correction sharpen edges and the desired amount of sharpening is programmable.

12. Is it possible increase or decrease the color saturation?

Yes, by interpolation CCM between 100 percent saturated and unit matrix.

13. The sensor output does not correspond to normal Bayer output when binning is used. Do you use different Bayer interpolation algorithm for binned and full output modes?

Sensor is outputting the data in the same bayer pattern configuration during binning and therefore no other interpolation algorithm is necessary. The sensor output is the same for both modes.

Lens Shading and Correction

14. Where are the lens shading correction control registers located?

The lens shading correction control register is located in R0x80:2. The lens shading feature can be turned on/off by R0x08:1[2]. Other lens correction settings can be found in R0x81:2 to R0xAE:2. For more details on the registers, refer to the latest MT9D131 data sheet.

15. Can the lens shading calibration be used for AF calibration?

Our lens shading calibrations are for only one position as there is only one set of values. This may introduce some minor deviation from ideal correction values if the focus movement causes the field of view to change, but it is likely undetectable.

Auto Exposure

16. To implement EV shift, the target exposure must be changed, is there a register to set for over or under exposure?

There is a variable named ae.target under the AE driver (ID = 2) for the customer to modify. Here is a description from the data sheet that explains how the MCU variables are accessed:

Microcontroller variables are similar to two-wire serial interface registers, except that they are located in MCU memory. Variables are accessed by specifying their address in R0xC6:1 and reading/writing the value to R0xC8:1. Variables can be accessed as 8-bit (byte) and 16-bit (word) at a time. Variable address can be specified as physical or logical. Use logical address to configure driver variables. A logical address consists of a driver ID (0 = monitor, 1 = sequencer, etc.) and an offset into the driver data structure.

17. Can we use single AE measurement window, whose size could be freely programmed?

Yes, you can use one window (read register which specifies average luma in this window and use it as current luma. Or average all 16 windows and consider it as one big window). This is the way how AE works in preview mode. You program size and position for Top/Left window, another window positions are calculated automatically making grid 4 x 4.

18. Manual exposure time and ISO setting. Is it possible to override the auto exposure and set the integration time and analog gain manually?

By disabling AE and AWB, integration time and analog gains will be frozen, then they can be modified.

19. Can we lock integration time and analog gain to their current values? Can we read the values of the integration time and analog gain from the sensor and set them manually afterwards?

By disabling the AE and AWB the integration time and analog gains will be frozen and can then be read and written to manually in the sensor core.

Flicker Avoidance

20. What are the instructions to enable flicker detection (and presumably suppression) for the rev 1 MT9D131 samples?

The flicker detection feature is enabled by default in context A (preview). It is capable of detecting 50Hz and 60Hz frequencies. To turn it on or off, use the following settings:

- ID = 1, Offset = 2: seq.mode[1] = 1 for flicker detection enable; = 0 to disable

If the above is turned off, upon the REFRESH command , the flicker detection is turned back on, unless the feature is specifically turned off in each of these states:

- *ID = 1, Offset = 35 : seq.previewParEnter.fd* (FD mode in the PreviewEnter state)
- *ID = 1, Offset = 42 : seq.previewPar.fd* (FD mode in the Preview state)
- *ID = 1, Offset = 49 : seq.previewParLeave.fd* (FD mode in the PreviewLeave state)
- *ID = 1, Offset = 56 : seq.captureParEnter.fd* (FD mode in the CaptureEnter state)

Color Correction

21. How can I load a new color correction matrix?

The CCM can be loaded using the AWB driver (ID = 3). The offsets for the left (A) and right (daylight) CCMs and gain ratios are as follows:

- ID = 3, Offset = 6-26 (ccmL) contains the CCM values and gain ratios for the red-light (left color correction matrix)
- ID = 3, Offset = 28-48 (ccmRL) contains the detla values between the left and right matrices

The AWB driver will automatically adjust the current CCM values (ID = 3, Offset = 50–70) to the proper values based on the two extreme cases above.

Mode Driver Preview and Driver

22. What is the maximum frame rate that can be achieved if I use 65 MHz clock?

For 65 MHz, you can do 25 fps for QVGA (without skipping) and 12.8 fps for UXGA capture. (The register wizard will show this information.)

Flash Strobe, Mechanical Shutter, and Global Reset

23. For flash LED applications, what is the proper sequence for the flash light to be turned on before a frame is captured?

Assuming that the customer is going from preview to capture mode for the flash capture, the proper sequence should be:

1. In Preview (seq.state = 3).
2. Turn on flash.
3. Send command to switch to context B.
4. Turn off flash when seq.state = 8 or 1, 2, 3.

24. We are considering using the CMOS sensor to detect the ambient light for the purpose of auto-flash application. Can you tell me which register contains the luminance value from the sensor? How do we convert the luminance to lux and what is the accuracy?

There is a firmware variable (ae.mmMeanEV, ID = 2 Offset = 78, 8-bit variable) that provides an "EV" score. It is the mean of five EV zones. There is no straight conversion from this score to lux. Determining when the host should turn on the flash (that is, which EV values require flash) calls for some experimentation and optimization on the customer's side.

JPEG Functionality

25. Do I have to capture JPEGs in 422 Mode for a full-resolution image?

Yes, for full-resolution JPEG, you will need 4:2:2 format rather than 4:2:0. For full resolution in JPEG, you can capture in 4:2:2 and monochrome format.

26. MT9D131 spec does not provide information about the status byte that is at the end of each LINE_VALID stream in SPOOF mode. Please provide a description of the field.

In spoof mode, the JPEG status byte is always appended to the end of the JPEG data stream. In continuous mode, the appending of the JPEG status byte can be optionally enabled by setting mode.FIFO_config0_B[9] = 1. The status byte is the value of R0x02:2[7:0].

27. From MT9D131 Spec: "The timing of PIXCLK and DOUT within each LINE_VALID assertion period is variable and therefore unlike that of uncompressed data". Why is the timing different? Spoof mode is supposed to emulate the uncompressed data, which will require no changes on PIXCLK or DOUT behavior, right? I was expecting that filler bytes will be placed after JPEG bytes.

Spoof mode is used such that the LINE_VALID (which in this case is DATA_VALID) are in regular periods, instead of being non-uniform and dependent on when the compressed data is available. The timing of PIXCLK and DOUT are based on when the valid data is available.

28. What are the specific recommendations that map the most popular SNAPSHOT resolutions with a preferred programming for the Spoof Frame Width and Spoof Frame Height (IFP Register's Page2 Reg 0x10 and 0x11)? Does Micron recommend a FIXED Spoof Frame Width/Height regardless of resolution?

Spoof frame width and height should be set according to the expected JPEG frame size. We recommend enabling ignore spoof frame height to avoid spoof frame error when the spoof frame size is set too small, and to avoid excessive padding of dummy data when the spoof frame size is set too large. Spoof frame width should be much larger than the horizontal blanking—spoof LINE_VALID lead plus spoof LINE_VALID trail—to prevent FIFO overflow. The user should set the desired spoof frame width and height to mode.spoof_width_B and mode.spoof_height_B.

29. Does the JPEG data length include the 4 bytes used for J0, J1, J2, and status bytes at the end of each LINE_VALID sync?

The JPEG data length does not include J0, J1, or J2. Status bytes are appended only at the end of the last LINE_VALID of the JPEG spoof frame, *not* every LINE_VALID.

30. Does the JPEG compression algorithm support compressed 565 output?

No. We can only output RGB565 format in uncompressed mode.

31. How does the use of the JPEG engine affect the frame rate? Use of JPEG should reduce the amount of data transmitted, but does it affect the frame rate?

No, the use of JPEG engine would not affect the frame rate. The limiting factor for frame rate is the AE algorithm and image size, blanking, system clock, and so on. Regardless of the compression ratio, AE needs to collect enough luminance information from the image and change exposure time and gain to achieve the new target luminance, and therefore limit the frame rate.

Miscellaneous FAQs

32. Fixed Pattern Noise (FPN) appears in Preview mode in low light conditions when 2 ADC are used in Context A. How do we reduce the FPN?

The AE settings show that it was in the highest zone number, which means that all the gains are at maximum values due to the dark lighting conditions. In such a scenario (almost complete darkness), we do expect some column FPN. However, the column FPN can be reduced by:

- Limiting all the maximum gain settings in the AE driver (ID = 2)
- Use 1 ADC. Since the frame rate will be reduced in dark conditions, 1 ADC is enough for the slower FPS.

33. When should REFRESH be used? When not? What does REFRESH do? When does REFRESH start to affect the system, after how many frames, etc.? What is the difference between REFRESH and REFRESH_MODE?

REFRESH and REFRESH_MODE both upload image processing parameters (from drivers to IP hardware) for the current context. However, only REFRESH_MODE uploads sensor size and timing parameters and only REFRESH_MODE updates the drivers' windows (IP statistics). REFRESH does not upload the sensor size and timing parameters, but REFRESH does update/reset all of the drivers' operating values (for example, AE limits, Low-light parameters, etc.). For crop/pan changes (like digital zoom), use REFRESH_MODE. These changes are applied at the end of the current frame in which the REFRESH/REFRESH_MODE command is issued.

34. When can the registers be written? When not?

Registers may be written any time the MT9D131 is awake and running via the two-wire serial interface. However, to avoid mid-frame changes in an image's appearance, avoid changing Sensor Core registers (page 0) or SOC registers (page 1, 2) during the FRAME_VALID time (these may be changed at any time if appearance is not a factor, for example, upon startup). Most changes to image parameters may be written to the driver variables at any time. Driver variables are internally programmed to update the IP hardware at the correct time between frames.

35. Is there a need for a delay after the REFRESH command?

REFRESH should only be issued once per frame. REFRESH_MODE may be issued as often as possible.

36. Does the MT9D131 contain flash memory? Can the user access the MCU on the chip for general computing?

This part does not contain flash memory. Also, we have not attempted to use the MT9D131 to drive an external flash part. The MCU contained in the MT9D131 is not designed to be used for general computing.

37. What does "PRNU" stand for?

PRNU stands for "photoresponse nonuniformity."

38. What is ^{t}EXTCLK? Is this the period of master clock?

^{t}EXTCLK is the period of master clock cycle (if PLL is not used) or PLL cycle (if PLL is enabled).

39. Is it possible to have an ISO setting in the MT9D131?

The MT9D131 does not have registers that map directly to an ISO-equivalent setting. Since our sensors employ an electronic rolling shutter (ERS), the meaning and application of ISO is different (and less beneficial) than it is for a traditional camera with mechanical shutters. ISO is essentially a control of sensitivity to light. For the MT9D131, it means to have different exposure time and gains. Our parts have automatic gain control (hence auto ISO speed). In order to have fixed sensitivity, you would have to either:

1. Turn off the firmware drivers (AE, AWB, and so on) to manually control gain and exposure time. However, this might lead to incorrect exposure and flicker detection.
2. Fix the max/min gains and max/min indexes ("zones") settings in the AE driver. By locking the camera into a certain zone and gains, you will have fixed frame rate and gains.

Again, the MT9D131 does not provide direct ISO settings. You may control the gain/exposure time to control the sensitively (to simulate ISO settings), but doing so is not likely to work as well as the auto mode.

40. How do I change the PLL setting after the firmware is running?

You may change the PLL settings (M/N/P) during operation by bypassing the PLL first. For example:

```
REG = 0, 0x65, 0xA000 // bypass PLL
...change PLL settings here...
REG = 0, 0x65, 0x2000 // disable PLL bypass
```

41. How do you adjust several functions (AE target, AWB parameter) after power-on and initialization? I think that it is not possible to set the parameter while the firmware is running.

We are considering the following sequence:

1. **Preview mode set**
2. **Firmware stop**
3. **Parameter setting (ex. AE target...)**
4. **REFRESH command (#Invoke Sequencer Refresh[B103 0005])**

Firmware parameters (AE, AWB, and so on) may be adjusted after initialization. You should note that some firmware variables are updated immediately, while others require a REFRESH/REFRESH_MODE command after the change in values. For example, if ae.target (ID = 2, Offset = 6) is changed from the default value of 60 to 10, the image brightness changes suddenly (the image becomes very dark).

42. Why does random noise appear in low light conditions?

At under 5 lux, we do expect some noise in the image. However, there are several ways to improve the image quality in low-light conditions, including the “night mode” preset below:

```
[Increased Gain/Lower Saturation (Night Mode)]
VAR8 = 2, 0x10, 0x0080 // AE_MAX_VIRTGAIN
VAR8 = 2, 0x18, 0x0080 // AE_MAXGAIN23
VAR8 = 2, 0x0E, 0x0018 // AE_MAX_INDEX
VAR8 = 2, 0x16, 0x0060 // AE_MAX_DGAIN_AE2
VAR8 = 7, 0x43, 0x0042 // MODE_GAM_CONT_A
VAR8 = 7, 0x44, 0x0042 // MODE_GAM_CONT_B
VAR8 = 1, 0x18, 0x0040 // SEQ_LLSAT1
VAR8 = 1, 0x03, 0x0005 // SEQ_CMD
```

If the above settings do not provide improvement, you can also configure the low-light (“LL”) variables in the sequencer driver. For example, you may use:

- seq.sharedParams.LLmode (ID = 1, Offset = 21)
- Bit 0-change interpolation threshold
- Bit 1-reduce color saturation
- Bit 2-reduce aperture correction
- Bit 3-increase aperture correction threshold
- Bit 4-enable Y filter

For details related to low-light variables, refer to the “Driver Variable-Sequencer Driver (ID = 1)” table in the MT9D131 data sheet.

The above settings are only an example and can be tuned to your needs. The maximum gain was increased for low-light conditions, which will introduce more noise. For experimental purposes, you can reduce the maximum allowed gain by the following variables in the AE driver (ID = 2):

- ae.maxVirtGain (offset = 16) –maximum allowed virtual gain
- ae.maxDGainAE1 (offset = 20) –maximum digital gain pre-LC (lens correction)
- ae.maxDGainAE2 (offset = 22) –maximum digital gain post-LC (lens correction)
- ae.mazGain23 (offset = 24) –maximum gain to increase in low-light before dropping frame rate

43. Do I have to wait after a REFRESH command?

Changes to the firmware variables would be effective starting the next frame after the REFRESH command is called, except for a few cases related to changes to integration time (which takes two frames instead of one). Hence there is no significant delay. If you are using DevWare, you will notice longer delays, but these delays ares due to the software, not the chip itself.

44. Can VDDPLL be left unconnected if PLL is not used (by default, we knew the PLL is bypassed and powered down)?

For ESD reasons, we do not recommend leaving VDDPLL floating even if they are not used. If they are not used, they will not consume much power.

45. Are the registers REG0x61:0, REG0x62:0, REG0x63:0, and REG0x64:0 related to a row noise? When row noise occurs, these register values seem to change a lot.

An offset can be provided to the ADC for green1, blue, red, and green2 (determined by 0x61-64:0). When 0x60:0[0] = 0 (default), the values in R0x61-64:0 are read-only; they show the offset values determined by the black-level calibration algorithm. The purpose of this is to fully utilize the ADC input dynamic range.

46. According to the specifications sheet, I have to issue the REFRESH command (seq.cmd = 5) when changing the image resolution, effect, gamma and so on. In this timing, it seems that hardware registers were changed by drivers. See the following example for an explanation: First, I changed the slew rate control register in R0x0E:2 and R0x0F:2 (hardware registers), but in this timing, I do not change the slew rate control variables in the mode driver(ID = 7). Next, I issued the REFRESH command to change the image resolution. After issuing the command, the hardware registers were changed by the drivers. I think that there are many registers like this. I would like to know which registers were changed by drivers when I issued the seq.cmd = 5. I cannot understand how to set the register correctly without this information.

The REFRESH command (seq.cmd = 5) is only needed when variable (not register) settings are changed. As you have noted, some variables are mapped to registers, so the variable value overwrites the associated register. Since the MT9D131 has many features and registers, the firmware is designed such that the user can make programming easier (because the firmware calculates and programs multiple registers automatically).

Therefore, we recommend programming via variables instead of registers whenever possible. For example, slew rate values should be programmed from the firmware (mode driver) instead of the SOC page2 register. Most frequently used settings related to the color pipeline can be configured using firmware variables.

47. How should we treat the STANDBY terminal? Should it be connected to GND or open?

STANDBY should not be left open. If it is not used, it should be tied to ground.

Appendix B—Glossary of Terms

Table 21: Glossary of Terms

Term	Definition
AWB	Auto white balance
Bayer color filter array	Color space jointly developed by Microsoft and Hewlett Packard as a color standard. Refer to http://www.w3.org/Graphics/Color/sRGB.
Bi Level	An image with only two colors
CCD	Charge-coupled device—one of the two main types of image sensors used in digital cameras
CCM	Color correction module or color correction matrix
Chrominance	Comprises the two components of a television signal that encode color information. Chrominance defines the difference between a color and a chosen reference color of the same luminous intensity.
CMOS	Complementary metal-oxide semiconductor
Gamma	Different platforms such as PC, Mac, and UNIX interpret color values slightly differently. Any images that look dark on a PC might look bright on a Mac. The gamma correction is a way to explain how an image should be displayed.
Gamma characteristic	A gamma characteristic is a power-law relationship that approximates the relationship between the encoded luminance in a television system and the actual desired image brightness. With this nonlinear relationship, equal steps in encoded luminance correspond to subjectively approximately equal steps in brightness.
IFP	Image flow processor—performs color recovery and correction, sharpening, gamma correction, lens shading correction, and on-the-fly defect correction
Interlaced	Graphic data is split (usually into two parts), and displayed alternately line by line.
Interlacing	Also known as progressive display—GIF, JPG and TIFF have supported this feature since the early 1990s.
JPEG	Joint photographic experts group—a file format.
LC	Lens shading correction
LED	Light emitting diode
LSB	Least significant bit
Luma	Intensity or brightness component of pixel information
Luminance	The quality of being luminous—emitting or reflecting light. Luminosity is measured relative to that of our sun.
MSB	Most significant bit.
MTF	Modulation transfer function—the sharpness of a photographic imaging system or of a component of the system (lens, film, scanner, enlarging lens, etc.) is characterized by MTF.
NTSC	National Television Standards Committee—the North American standard (525-line interlaced raster-scanned video) for the generation, transmission, and reception of television signals.
Output Resolutions	Includes, but is not limited to, VGA, QVGA, CIF, and QCIF.
PAL	Phase Alternating Line, Phase Alternation by Line or Phase Alternation Line—mainly a European standard of displaying analog television signals. It consists of 625 horizontal lines of resolution at 50Hz. Also see NTSC.

Table 21: Glossary of Terms (continued)

Term	Definition
Pixels	A short name for picture element. The smallest part that can be displayed on a monitor.
Progressive Scan	An image sensor that gathers its data and processes each scan line one after another in sequence. Compare to Interlaced.
Resolution	A measure on how much information is stored in an image.
RGB	Red, green and blue. A way to represent color on a monitor. Also see CMYK.
SADDR	Sensor address
SCLK	Serial clock
V_{REF}	Voltage reference

Revision History

Rev. B 2/11/2008

- Added Table 8, "Bayer 8+2 Mode Output Data Order," on page 17
- Updated Figure 13: "Signal Modification for Lens Shading," on page 31
- Updated Figure 15: "DevWare Toolbar," on page 32
- Added Table 12, "Color Corection Matrix Structure," on page 50
- Updated "Auto White Balance" on page 50
- Updated "Disabling All Firmware Drivers" on page 73
- Updated CLKIN to EXTCLK throughout the document
- Updated RESET# to RESET_BAR throughout the document

Rev A. 6/28/2007

- Initial release